# Developing IoT Projects with ESP32

Automate your home or business with inexpensive Wi-Fi devices

Vedat Ozan Oner

# Developing IoT Projects with ESP32




Automate your home or business with inexpensive Wi-Fi devices

**Vedat Ozan Oner**

Packt

BIRMINGHAM—MUMBAI

# Developing IoT Projects with ESP32



**Group Product Manager**: Wilson D'souza
**Publishing Product Manager**: Preet Ahuja
**Senior Editor**: Sangeeta Purkayastha
**Content Development Editor**: Nihar Kapadia
**Technical Editor**: Sarvesh Jaywant
**Copy Editor**: Safis Editing
**Project Coordinator**: Neil Dmello
**Proofreader**: Safis Editing
**Indexer**: Rekha Nair
**Production Designer**: Shankar Kalbhor

First published: August 2021

Production reference: 1080721

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-83864-116-0

www.packt.com

*For my daughters, Melis and Selin, and my wife, Ferah; how lucky I am to have you!*

# Contributors

## About the author

**Vedat Ozan Oner** is an IoT product developer and software architect with a good blend of technical knowledge and experience. During his career, he participated in many IoT projects in different roles, which allowed him to see all aspects of developing successful IoT products in highly competitive markets. He holds a bachelor's degree in METU/computer engineering and holds several industry-recognized credentials and qualifications, including PMP®, ITIL®, and AWS Certified Developer.

Vedat started his limited company, Mevoo Ltd (`https://mevoo.co.uk`), in London in 2018 to provide consultancy services to his clients as well as develop his own IoT products. He still lives there with his family.

You can reach Vedat at `https://www.linkedin.com/in/vedatozanoner/`.

*Special thanks to the Packt team for asking me to write this book. It was a great opportunity for me to share and to learn. I believe I did a good job together with the Packt team and the reviewers, Carlos and Tarik, to present the subjects to you in the most effective way possible. Their valuable feedback and efforts made this book worth publishing.*

# About the reviewers

**Carlos Bugs** is an electrical engineer with more than 15 years' experience of working with technology. He has experience in all the main stages of IoT product development, including preparation for large-scale production. He has undertaken projects in a variety of areas, including automotive, energy, instrumentation, medicine, and agriculture. Currently, he is the CTO of a tech company that integrates hardware projects (sensors, nodes, and gateways) with data science. He did the setup for large-scale production in China and thereafter in Brazil, where more than 70,000 sensors have been manufactured for big clients.

He has also undertaken many research projects and has won awards in Brazil and the United States. He can be reached on LinkedIn: `https://www.linkedin.com/in/carlos-bugs-6a272458/`

*I would like to thank God for allowing me to contribute to this amazing book. I also wish to thank my family (my wife and son) for being my inspiration and for being patient during the review process. Finally, I would like to thank the author, Vedat, Neil, and the entire Packt team, who worked so hard on this book.*

**Tarik Ceber** is currently employed as a hardware development engineer at TechSat GmbH in Germany. He started his career as a self-learner in 2005 as a C++ developer and worked on various unmanned aerial vehicle projects. Because of his passion for embedded systems and avionics, he has also developed electronic printed circuit boards, including flight control computers, inertial navigation systems, battery management systems, and avionics test equipment for a vast arsenal of aerial platforms (small fixed-wing UAVs, tactical UAVs, quadcopters, and eVTOLs). Aside from the aviation business, he has also participated in IoT projects and designed printed circuit boards for smart home appliances, including BLE-enabled smart meters and RGB color bulbs.

*I would like to thank the author for allowing me to be a technical reviewer of this informative and well-organized book, which truly fills a practice-oriented information source gap in the fast-growing IoT world.*

# Table of Contents

# 3
# Impressive Outputs with Displays



# 4
# A Deep Dive into the Advanced Features

## 5

## Practice – Multisensor for Your Room



# Section 2: Local Network Communication

## 6

## A Good Old Friend – Wi-Fi



## 7

## Security First!

# 8
## I Can Speak BLE



# 9
## Practice – Making Your Home Smart



# Section 3: Cloud Communication

# 10
## No Cloud, No IoT – Cloud Platforms and Services

# 11

## Connectivity Is Never Enough – Third-Party Integrations



# 12

## Practice – A Voice-Controlled Smart Fan



## Other Books You May Enjoy

## Index

# Preface

**Internet of Things** (**IoT**) technology has been in our lives for more than a decade now. When I first came across a single-board computer at an expo 20 years ago, I was fascinated, as a young computer engineer, by the possibilities that this device introduced. It was the gateway, in my eyes, to a smart home where you could know what was going on at home when you were miles away!

Since then, I have participated in many IoT projects in different positions, so I have had many chances to see IoT products from different perspectives. As developers, we mostly tend to forget about what the use of technology is while trying to solve a technical problem. However, as we develop an IoT product, the first question that we should ask is what is the value of this product, what benefit will people get from it? It doesn't matter if it is a consumer product or an industrial IoT solution; it should help people to solve real problems. In the last chapter of each part of this book, you will find a complete project that can help people in their daily lives.

There is no single driving force behind IoT, but we can count several strong enablers, such as the emergence of inexpensive silicon chips available in large quantities, mobile technologies, and cloud computing, to name just a few. I think ESP32 has contributed to this on its own terms. When it was launched in 2016 by Espressif Systems, I was working for a smart home company as a technical product manager. We had seen the opportunity immediately – that this chip could cut the cost of our home gateway to a quarter of the one that we had! There was no other Wi-Fi **system on a chip** (**SoC**) on the market as a complete computing solution with that price tag. I know it is not possible to discuss everything that we can do with ESP32, but I believe you will find this book quite useful before starting your next IoT project with ESP32.

Besides as a profession, it is my hobby to build new IoT devices together with my daughter at home. I always love sharing my knowledge and experience with her and with people around me, as well as learning from them. I hope you enjoy reading this book and developing the projects together with me.

# Who this book is for

This book targets embedded software developers, IoT software architects/developers, and technologists who want to learn how to employ ESP32 effectively in their IoT projects. Hobbyists will also find the examples in this book useful when they need a powerful Wi-Fi SoC with professional features.

# What this book covers

*Chapter 1*, *Getting Started with ESP32*, introduces you to IoT technology in general, the ESP32 hardware, and development environment options.

*Chapter 2*, *Talking to the Earth – Sensors and Actuators*, discusses different types of sensors and actuators and how to interface them with ESP32.

*Chapter 3*, *Impressive Outputs with Displays*, explains how to select and use different display types in ESP32 projects. FreeRTOS is also discussed in detail.

*Chapter 4*, A *Deep Dive into the Advanced Features*, covers audio/video applications with ESP32 and the power management subsystem for low-power requirements.

*Chapter 5*, *Practice – Multisensor for Your Room*, is the first reference project in the book, where several sensors are integrated as a single ESP32 device.

*Chapter 6*, *Good Old Friend – Wi-Fi*, shows how to use ESP32 in the station and access point modes of Wi-Fi. Some TCP/IP protocols are discussed in the context of ESP32.

*Chapter 7*, *Security First!*, explores the security features of ESP32 and provides examples of secure firmware updates and secure communication techniques.

*Chapter 8*, *I Can Speak BLE*, introduces the BLE basics and shows how to develop a BLE beacon, a GATT server, and BLE mesh nodes.

*Chapter 9*, *Practice – Making Your Home Smart*, builds a full-fledged smart home solution with a gateway, a light sensor, and a relay switch in a BLE mesh network.

*Chapter 10*, *No Cloud, No IoT – Cloud Platforms and Services*, discusses the important IoT protocols and introduces the IoT platforms from different providers with an example of AWS IoT integration.

*Chapter 11*, *Connectivity Is Never Enough – Third-Party Integrations*, focuses on integration with popular services such as voice assistants and IFTTT.

*Chapter 12*, *Practice – A Voice-Controlled Smart Fan*, converts an ordinary fan into an Alexa-enabled smart device as the last project of the book.

# To get the most out of this book

IoT technology requires many different disciplines and skills to develop an IoT product. Fundamentally, you are expected to read Fritzing diagrams to set up the hardware prototypes in the examples, in addition to having programming capabilities with C and Python. It is also assumed that you have a familiarity with TCP/IP protocols and cryptography basics to follow the examples easily. In some chapters, there are reference books suggested for reading if you don't feel comfortable with the basics of the subject.

The necessary hardware components are listed before each example. However, you should have a breadboard, jumper wires, and a multimeter ready to be able to build the circuits. It is also advisable to have soldering equipment since many of the new modules need headers to be soldered in order to connect them to the breadboard.

As the development environment, you should have VS Code installed on your PC. The examples in this book are developed and tested on a Linux machine, but all should work regardless of the OS platform. The alternatives of the external tools are suggested for different platforms where necessary.

There are several mobile applications required for testing purposes. Therefore, you will need a mobile device while working on the examples. These mobile applications are available for both Android and iOS platforms.

To gain proficiency in a new subject requires a lot of practice. Each reference project at the end of the parts is for that purpose. After completing them, it would be highly beneficial to continue to work on the projects as suggested at the end of the chapters. Some more ideas are given for how to improve them further.

| **Software/hardware covered in the book** | **OS requirements** |
|---|---|
| ESP32 and different additional hardware components | Windows, macOS, or Linux (any) |
| ESP-IDF and several external libraries | |

**If you are using the digital version of this book, we advise you to type the code yourself or access the code via the GitHub repository (link available in the next section). Doing so will help you avoid any potential errors related to the copying and pasting of code.**

# Download the example code files

You can download the example code files for this book from GitHub at `https://github.com/PacktPublishing/Internet-of-Things-with-ESP32`. If there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

# Code in Action

Code in Action videos for this book can be viewed at `https://bit.ly/2T0ynws`.

# Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `http://www.packtpub.com/sites/default/files/downloads/9781838641160_ColorImages.pdf`.

# Conventions used

There are a number of text conventions used throughout this book.

`Code in text`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "After installation, it integrates itself to the VSCode UI where you can find all the functionality that `idf.py` of ESP-IDF provides."

A block of code is set as follows:

```
#define GPIO_LED 2
#define GPIO_LED_PIN_SEL (1ULL << GPIO_LED)
#define GPIO_BUTTON 5
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
static void button_handler(void *arg);

static void init_hw(void)
```

Any command-line input or output is written as follows:

```
$ ls -R
```

**Bold**: Indicates a new term, an important word, or words that you see on screen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "Click on the **New Project** button on the PIO home."

> **Tips or important notes**
> Appear like this.

# Get in touch

Feedback from our readers is always welcome.

**General feedback**: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at `customercare@packtpub.com`.

**Errata**: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/support/errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

**Piracy**: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

**If you are interested in becoming an author**: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

# Share Your Thoughts

Once you've read Developing IoT Projects with ESP32, we'd love to hear your thoughts! Please click here to go straight to the Amazon review page for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

# Section 1: Using ESP32

In this section, you will learn about the development platform/framework options for starting with ESP32 and how to use ESP32 in a project by interfacing with different sensors and actuators.

This part of the book comprises the following chapters:

- *Chapter 1*, *Getting Started with ESP32*
- *Chapter 2*, *Talking to the Earth – Sensors and Actuators*
- *Chapter 3*, *Impressive Outputs with Displays*
- *Chapter 4*, A *Deep Dive into the Advanced Features*
- *Chapter 5*, *Practice – Multisensor for Your Room*

# 1
# Getting Started with ESP32

Espressif ESP32 is a powerful tool in the toolbox of a developer for many types of **Internet of Things** (**IoT**) projects. We are all developers, and we all know how important it is to select the right tool for a given problem in a domain. To solve the problem, we need to understand the domain, and we need to know the available tools and their features that are important for that specific problem in order to find the right one (or perhaps several combined). After selecting the tool, we eventually need to figure out how to use it in the most efficient and effective way possible so as to maximize the added value for end users.

In this chapter, I will discuss the technology, IoT, in general, what an IoT solution looks like in terms of basic architecture, and how ESP32 fits into those solutions as a tool. If you are new to IoT technology, or are thinking of using ESP32 in your next project, this chapter helps you to understand the big picture from the technology perspective by describing what ESP32 provides, its capabilities, and its limitations.

The main topics covered in this chapter are as follows:

- IoT as an emerging technology and its application areas with some examples
- The basic structure of IoT solutions, including security considerations
- An introduction to the ESP32 platform and modules
- Available development platforms and frameworks
- **Real-Time Operating System** (**RTOS**) options for ESP32

# Technical requirements

In this book, we are going to have many practical examples where we can learn how to use ESP32 effectively in real-world scenarios. Although links to the examples are provided within each chapter, you can take a sneak peek at the online repository here: `https://github.com/PacktPublishing/Internet-of-Things-with-ESP32`. The examples are placed in their relative directories of the chapters for easy browsing. There is also a common source code directory that contains the shared libraries across the chapters.

We will use different software tools and hardware components throughout the book. Each chapter shows its own list of these tools and components.

# IoT as an emerging technology

When I started my career 20 years ago, my first project involved collecting data regarding radio and TV stations by measuring some **Radio Frequency** (**RF**) parameters of broadcasted channels. The task was to design and develop a system in order to understand whether the stations comply with the existing regulations in the country. As a solution for this engineering problem, the technical leaders in the team designed a van with various equipment, including the following:

- A spectrum analyzer
- A TV demodulator
- Different types of antennas to measure those parameters
- An industrial PC to run the application software
- A radio transmitter to upload the measurements and some basic analysis to a data center

I was lucky that I participated in such a project in my very first job and saw how a complete data acquisition system was designed and developed to solve a real-world problem. This project was in the year just after Kevin Ashton introduced the term *Internet of Things* to technology literature in 1999.

When I first heard this term and was trying to understand what it actually means, I quickly noticed the similarities between an IoT solution and our monitoring van. We collected data from the environment by using some sensing devices, we had a processing unit, and we also transferred information to a central data storage and processing center. This last part was to access more processing and spot correlation between data coming from multiple vans. So, why not call it an IoT product? Well, not exactly. From that perspective, you could easily call any SCADA or PLC product an IoT system as well, so IoT would only then constitute a rebranding of existing technologies.

## What is IoT?

Although the definition of IoT might change slightly from different viewpoints, there are some key concepts in the IoT world that differentiate it from other types of technologies:

- **Connectivity**: An IoT device is connected, either to the internet or to a local network. An old-style thermostat on the wall waiting for manual operation with basic programming features doesn't count as an IoT device.
- **Identification**: An IoT device is uniquely identified in the network so that data has a context identified by that device. In addition, the device itself is available for remote update, remote management, and diagnostics.
- **Autonomous operation**: IoT systems are designed for minimal or no human intervention. Each device collects data from the environment where it is installed, and it can then communicate the data with other devices to detect the current status of the system and respond as configured. This response can be in the form of an action, a log, or an alert if required.
- **Interoperability**: Devices in an IoT solution talk to one another, but they don't necessarily belong to a single vendor. When devices designed by different vendors share a common application-level protocol, adding a new device to that heterogeneous network is as easy as clicking on a few buttons on the device or on the management software.
- **Scalability**: IoT systems are capable of horizontal scalability to respond to an increasing workload. A new device is added when necessary to increase capacity instead of replacing the existing one with a superior device (vertical scalability).

- **Security**: I wish I could say that every IoT solution implements at least the minimal set of mandatory security measures, but unfortunately, this is not the case, despite a number of bad experiences, including the infamous Mirai botnet attack. On a positive note, I can say that IoT devices mostly have secure boot, secure update, and secure communication features to ensure confidentiality, integrity, and availability the (CIA triad).

Gartner added IoT in the 2011 hype cycle, with the expectation of more than 10 years to mainstream adoption. However, many related technologies, such as RFID, mesh networking, and Bluetooth, were already on the list many years before 2011, along with enablers such as mobile and cloud technologies. Since then, Gartner has added several other IoT technologies and applications to its list, including the following:

- IoT platform
- Connected home
- Smart dust
- Edge computing
- Low-cost, single-board computers at the edge

5G and embedded AI are other revolutionary technologies on the Gartner list that support IoT and expand its area of application.

## Where do we apply IoT?

The application areas are vast, but conceptually speaking, we can group them into two basic categories:

- In the **consumer IoT** category, we can see mainly smart home and security systems, personal healthcare products, wearable technologies, and asset tracking applications.
- The **industrial IoT** category has more application areas, as you might expect. Every year, IoT Analytics publishes a top-10 trend list for industrial applications by reviewing thousands of new projects and the 2020 list contains manufacturing, transportation, energy, retail, cities, healthcare, supply chain, agriculture, and building applications in that order (`https://iot-analytics.com/top-10-iot-applications-in-2020`).

Since we have limited space in this book, I don't want to waste pages talking about each of these application areas. Instead, I'd like to share more interesting cases to show how the IoT technology can provide powerful solutions when incorporated with other cutting-edge technologies.

## AI/ML on the edge

AI has been around for a long time and there are many successful examples of machine vision, **Natural Language Processing** (**NLP**), speech recognition, and ML projects. However, they all require energy-hungry powerful hardware to be able to cope with CPU and memory-intensive calculations, which is not possible with humble sensor devices that have much less memory and processing power. TensorFlow Lite addresses this problem. Its converter can output a model, a set of rules to make predictions by running data through them, with a size as low as 14 KB to fit into any modern microcontroller, such as an ARM Cortex-M3 device with a very low power consumption, which enables you to have battery-operated sensor devices with ML capabilities. One interesting project comes from Benjamin Cabé (on Twitter: `@kartben`). In his project, he managed to train a model to discern different types of spirits with an accuracy of 92%. He used a Wio Terminal from SeeedStudio as the computing board, which has an ARM Cortex-M4F core running at 120 MHz.

Implications are enormous. Instead of a dummy sensor device, now we have the capability of developing a *real* smart device such that it can add meaning to data it collects and can react based not only on data, but also the meaning. Benjamin employed a simple gas sensor to detect various gases, such as carbon monoxide (CO), nitrogen dioxide ($NO_2$), ethyl alcohol ($C_2H_5CH$), and some other types. But the device itself can understand what it actually *smells*, thanks to the ML model it uses in its firmware. Without such a capability, the device would have to send its data to another more powerful machine or a cloud to make this analysis and then wait for a reply to decide what to do next. Moreover, if it loses its network connectivity somehow, nothing could be done more until connectivity is restored.

This subject definitely deserves another book, but if you want to do some experiments, ESP32 is also on the list of supported platforms on the TensorFlow Lite website.

> **Important note**
>
> You can have a look at the supported platforms for TensorFlow Lite at the following link: `https://www.tensorflow.org/lite/microcontrollers`.

## Energy harvesting

A vital discussion and research subject for **Wireless Sensor Networks** (**WSNs**) has always been the energy consumption of sensor nodes. Obviously, less is better. If you have some experience with the development of battery-operated wireless devices, you know the concept of *run to sleep*, which means do the job and go into sleep mode as soon as possible to preserve the most valuable resource, energy. Nonetheless, whatever you do, sensor nodes must consume energy and the user will have to replace the batteries after a while. An interesting technology comes to your aid at this point – energy harvesting, which has been around since the days of Nikola Tesla. The energy can be harvested from various ambient sources, including light, vibration, and wireless energy sources. To do that, a harvesting solution first needs to access that ambient energy by means of various components, depending on the energy type.

It is an RF antenna if the energy comes from an RF source, or a photovoltaic cell if light is the source. Then, this *raw* electrical energy has to be converted with the help of an integrated circuit in order to store it in a capacitor or a battery. But you know that this is easier said than done. Although there are several **Power Management Integrated Circuits** (**PMICs**) from different silicon vendors on the market, it is hard to say whether they solve this problem efficiently. The major challenges are very low levels of energy to harvest, the need to boost the very low voltage to higher logic levels, the need for multiple external components to operate, and a large chip footprint on the PCB. Therefore, these challenges have prevented vendors from producing high-performance energy harvesting chips. One product does sound promising, though.

Nowi Energy promotes its NH2D0245 PMIC as the most efficient and the smallest footprint power management IC compared to other semiconductor giants on the market. To prove their arguments, they launched a hybrid smartwatch module together with the module company MMT, such that a watch with that module requires no charge to operate during its lifetime. Energy harvesting is a hot topic, so there are, of course, competitors, such as e-peas semiconductors from Belgium. You might want to try one of those PMICs in your next WSN project.

## Nanorobotics

Before we move on, we should look at one last project, a research project from Cornell University. The result of this research has been published in *Nature Journal* in August 2020 as an article named *Electronically integrated, mass-manufactured, microscopic robots*. They invented actuators on a nano scale that you literally cannot see with your eyes. The super tiny structure has two solar cells on it to move the legs, and when laser beams are dropped on those solar cells, they generate enough voltage to activate the legs. Although not ready for any practical application as yet, this research is definitely on my follow-up list as a technologist and IoT expert.

> **Important note**
>
> If you want to see them in action, there is a video on YouTube: `https://www.youtube.com/watch?v=2TjdGuBK9mI`.

These examples are certainly extremes in terms of technology application, but I hope they provide a glimpse of the future in terms of IoT technologies and inspire you in your next IoT project. Let's now continue with some common features of IoT solutions.

# Understanding the basic structure of IoT solutions

An IoT solution combines many different technologies into a single product, starting from a physical device and covering all layers up to end user applications. Each layer of the solution aims to implement the same vision set by the business, but requires a different approach while designing and developing. We definitely cannot talk about one-size-fits-all solutions in IoT projects, but we still can apply an organized approach to develop products. Let's see which layers a solution has in a typical IoT product:

- **Device hardware**: Every IoT project requires hardware with a **System-On-Chip** (**SoC**) or **Microcontroller Unit** (**MCU**) and sensors/actuators to interact with the physical world. In addition to that, every IoT device is connected, so we need to select the optimal communication medium, such as wired or wireless. Power management is also another consideration under this category.
- **Device firmware**: We need to develop device firmware to run on the SoC in order to fulfill the project's requirements. This is where we collect data and transfer it to the other components in the solution.

- **Communication**: Connectivity issues are handled in this category of the solution architecture. The physical medium selection corresponds to one part of the solution, but we still need to decide on the protocol between devices as a common language for sharing data. Some protocols may provide a whole stack of communication by defining both the physical medium up to the application layer. If this is the case, you don't need to worry about anything else, but if your stack leaves the context management at the application layer up to you, then it is time to decide on what IoT protocol to use.
- **Backend system**: This is the backbone of the solution. All data is collected on the backend system and provides the management, monitoring, and integration capabilities of the product. Backend systems can be implemented on on-premises hardware or cloud providers, again depending on the project requirements. Moreover, this is where IoT encounters other disruptive technologies. You can apply big data analytics to extract deeper information from data coming from sensors, or you can use AI algorithms to feed your system with more smart features, such as anomaly detection or predictive maintenance.
- **End user applications**: You will very likely require an interface for your end users to let them access the functionality. 10 years ago, we were only talking about desktop, web, or mobile applications. But today we have voice assistants. You can think of them as a modern interface for human interaction, and it might be a good idea to add voice assistant integration as a feature, especially in the consumer segment.

The following diagram depicts the general structure of IoT solutions:

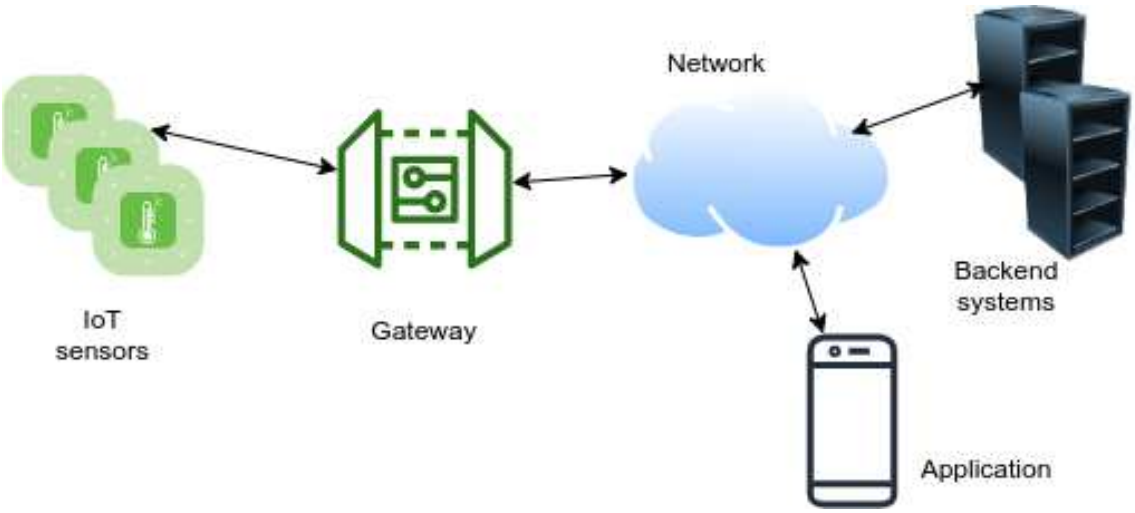


Figure 1.1 – Basic structure

This is the list of aspects, more or less, that we need to take into account in many types of IoT projects before starting.

## IoT security

One important consideration that remains is security. Actually, it is all about security. I cannot overemphasize its importance whatever I write. IoT devices are connected to the real world and any security incident has the potential for serious damage in the immediate environment, let alone other cybersecurity crimes. Therefore, it should always be in your checklist while designing any hardware or software components of the solution. Although security, as a subject, definitely deserves a book by itself, I can list some golden rules for devices in the field:

- Always look to reduce the attack surface for both hardware and firmware.
- Prevent physical tampering wherever possible. No physical port should be open if this is not necessary.
- Keep secret keys on a secure medium.
- Implement secure boot, secure firmware updates, and encrypted communication.
- Do not use default passwords; TCP/IP ports should not be open unnecessarily.
- Put health check mechanisms in place along with anomaly detection where possible.

We should embrace secure design principles in general as IoT developers. Since an IoT product has many different components, end-to-end security becomes the crucial point while designing the product. A risk impact analysis should be done for each component to decide on the security levels of data in transit and data at rest. There are many national/international institutions and organizations that provide standards, guidelines, and best practices regarding cybersecurity. One of these, which works specifically on IoT technology is the IoT Security Foundation. They are actively developing guidelines and frameworks on the subject and publishing many of those guidelines, which are freely available.

> **Important note**
>
> If you want to check those guidelines, you can visit the IoT Security Foundation website for their publications here: `https://www.iotsecurityfoundation.org/best-practice-guidelines/`.

Now, that we are equipped with sufficient knowledge of IoT and its applications, we can propel our journey with ESP32, a platform perfectly suited for beginner-level projects as well as end products. In the remaining sections of this chapter, we are going to talk about the ESP32 hardware, development frameworks, and RTOS options available on the market.

# Introduction to ESP32 platform and modules

The first ESP32 chip was launched in 2016 when I was working for a smart home company as a technical product manager. The wireless communication technology that we had chosen for our product line was Z-wave on account of its technical features (sub-gigahertz wireless communication, mesh networking, interoperability, and suchlike) and market status (many vendors, thousands of certified products, and so on).

The vision was not to be yet another device vendor, but to be a platform where every other vendor meets with end users. The most crucial step was to develop the most affordable Z-wave gateway on the market such that any smart home enthusiast would prefer our gateway as the access point of other Z-wave devices in their home. Our first prototype was a high-performance, embedded Linux board with an ARM-CortexA SoC; however, in terms of pricing, it was certainly not the most affordable one in our segment. Then we discovered ESP32 from Espressif. This was a game-changer.

ESP32 allowed us to slash the price of the gateway to a quarter of what it was originally. Having an ESP32 as the main computing unit, we attached a Z-wave module to it as the network co-processor. The other end was Wi-Fi, a built-in feature of ESP32, to connect the backend system. We didn't worry about security requirements because there was a cryptographic hardware accelerator in the ESP32 chip for encryption/decryption purposes. That was all that we needed. However, as always, life is not that easy. The Z-wave library that we procured from the market had targeted Linux-based boards, not a resource-constrained SoC like ESP32. So we started to port the whole Z-wave library for ESP32 and succeeded. Finally, we had the most compact and most affordable Z-wave gateway on the market.

## Why ESP32?

IoT technologies have proven their worth over the years and, as developers, we have a great many tools available to us today for developing exceptional IoT products compared to 5 or 10 years ago. ESP32 is definitely one of those tools and there are many reasons as to why this is so:

- Its price tag and availability
- Wi-Fi and Bluetooth in a single SoC

- Great hardware features with many peripheral interfaces, different power modes, and cryptographic hardware acceleration
- Variants for different requirements, in terms of both chips and modules
- Advanced development platforms and frameworks
- A huge community
- And finally, native integration with top cloud infrastructures

These are the reasons for putting ESP32 at the top of your SoC selection list if you require a Wi-Fi SoC in your project.

## ESP32 features

Since the introduction of the first ESP32 on the market, Espressif launched several variants of ESP32 and most recently, in 2020, they introduced ESP32-S2 series chips. The ESP32 family is a general-purpose, feature-rich, and versatile SoC solution that you can use in many different types of IoT projects where you require Wi-Fi connectivity.
Let's have a quick look at the main features:

- **CPU and memory**: 32-bit Xtensa® LX6 microprocessor with a clock frequency/MIPS of up to 240 MHz/600 MIPS. Single- or dual-core variants. 448 KB ROM, 520 KB SRAM, and 16 KB RTC memory. Support for external SPI flash and SPI RAM for module variants. DMA for peripherals.
- **Connectivity**: Wi-Fi 802.11 n (2.4 GHz) up to 150 Mbps (STA and softAP modes) and Bluetooth-compliant with Bluetooth v4.2 BR/EDR and BLE specifications.
- **Peripheral interfaces**: GPIOs, ADC, DAC, SPI, I2C, I2S, UART, eMMC/SD (chip variants), CAN, IR, PWM, touch sensor, and hall sensor.
- **Security**: Cryptographic hardware acceleration (random number, hash, AES, RSA, and ECC), 1024-bit OTP, secure boot, and flash encryption.
- **Power modes**: Different power modes with the help of an **Ultra-Low-Power** (**ULP**) co-processor and a **Real-Time Clock** (**RTC**). 100 µA power consumption in deep-sleep mode (ULP active).

The new ESP32-S2 series is a bit different, with some notable differences including the following:

- Single core.
- No Bluetooth.

- No support for SD/eMMC, but USB OTG has been added.
- Enhanced security features.

To make hardware design easier, Espressif provides different ESP32 modules with different configurations. Variable parameters for the modules are the ESP32 chip variant, external flash (4, 8, or 16 MB), external SRAM, and the antenna type. We can select among modules with a PCB antenna, or there is an external antenna option realized with the help of a U.FL/IPEX connector. On the ESP32-S2 side, we have only one module option as of the time of writing this book. Most of the time, it is enough to choose one of those modules in your projects. However, if you require a specific ESP32 chip, for example, one with high-temperature operation, then you need to use a corresponding chip variant such as ESP32-U4WDH and design your PCB accordingly. You can find available modules on the Espressif website here: `https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/index.html`.

The following photo shows an ESP32-WROOM-32D module with an integrated onboard antenna:


Figure 1.2 – ESP32-WROOM-32D module

As a development kit, we can find many boards from different vendors on the market. We can easily start to develop with such a kit without the need for the actual hardware design and prototype of the final product. All models integrate a USB-UART bridge chip and a USB port, so we only need to plug the kit into our development PC to flash and test the firmware:

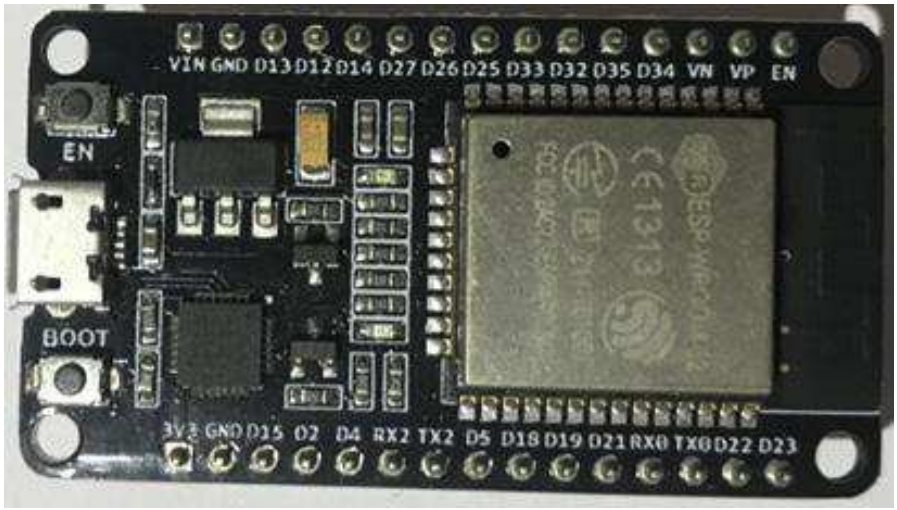


Figure 1.3 – DOIT ESP32 Devkit v1

Following this introduction to the hardware, we can continue with the firmware development platforms and frameworks.

# Development platforms and frameworks

ESP32 is quite popular. Therefore, there are a good number of options that you can select as your development platform and framework.

The first one, of course, comes directly from Espressif itself. They call it the **Espressif IoT Development Framework** (**ESP-IDF**). It supports all three main OS environments – Windows, macOS, and Linux. After installing some prerequisite packages, you can download the ESP-IDF from the GitHub repository, and install it on your development PC. They have collected all the necessary functionality into a single Python script, named `idf.py`, for developers. You can configure project parameters and a final binary image by using this command-line tool. You can also use it in every step of your project, starting from the build phase to connecting and monitoring your ESP32 board from the serial port of your computer. But as I said, it is a command-line tool, so if you are a more graphical UI person, then you need to install Visual Studio Code and install an ESP-IDF extension in it. Here is the link to ESP-IDF: `https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html`.

The second option is the Arduino IDE. As you might expect. Arduino provides its own library to work with ESP32 boards. If you have experience with the Arduino IDE, you know how easy it is to use. However, it comes at the cost of development flexibility compared to ESP-IDF. You are constricted in terms of what Arduino allows you to do and you need to obey its rules.

The third alternative you can choose is PlatformIO. This is not a standalone IDE or tool, but comes as an extension in Visual Studio Code as an open source embedded development environment. It supports many different embedded boards, platforms, and frameworks, including ESP32 boards and ESP-IDF. Following installation, it integrates itself with the VSCode UI, where you can find all the functionality that `idf.py` of ESP-IDF provides. In addition to VSCode IDE features, PlatformIO has an integrated debugger, unit testing support, static code analysis, and remote development tools for embedded programming. PlatformIO is a good choice for balancing ease of use and development flexibility.

The programming language for those three frameworks is C/C++, so you need to know C/C++ in order to develop within those frameworks. However, C/C++ is not the only programming language for ESP32. You can use MicroPython for Python programming or Espruino for JavaScript programming. They both support ESP32 boards, but to be honest, I wouldn't use them to develop any product to be launched on the market. Although you may feel more comfortable with them because of your programming language preferences, you won't find ESP-IDF capabilities in any of them.

# RTOS options

Basically, an RTOS provides a deterministic task scheduler. Although the scheduling rules change depending on the scheduling algorithm, we know that the task we create will complete in a certain time frame within those rules. The main advantages of using an RTOS are the reduction in complexity and improved software architecture for easier maintenance.

The main real-time operating system supported by ESP-IDF is FreeRTOS. ESP-IDF uses its own version of the Xtensa port of FreeRTOS. The fundamental difference compared with the vanilla FreeRTOS is the dual-core support. In ESP-IDF FreeRTOS, you can choose one of two cores to assign a task or you can let FreeRTOS choose it. Other differences compared with the original FreeRTOS mostly stem from the dual-core support. FreeRTOS is distributed under an MIT license: `https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html`.