HACKING RISC-V

CHAPTER 1: WHY RISC-V

Introduction

As technology evolves, the importance of understanding the underlying hardware architecture becomes increasingly crucial for those interested in hacking and security. RISC-V, an open-source instruction set architecture (ISA), has gained significant traction in the industry due to its flexibility, scalability, and openness. Unlike proprietary ISAs, RISC-V provides the freedom to modify and extend the architecture, making it an ideal platform for experimentation and innovation. In this chapter, we will explore why learning hacking for RISC-V is essential, particularly in the context of using the ESP32-C3-DevKitM-1 development board and PlatformIO with Visual Studio Code. These tools provide an easy and accessible setup that will allow you to focus on practical projects and hands-on experience.

The Rise of RISC-V

RISC-V is not just another ISA; it represents a paradigm shift in the world of computing. Born out of a desire to create a simple, clean, and extensible architecture, RISC-V has attracted a growing community of developers, researchers, and companies. The open nature of RISC-V allows anyone to implement and modify the ISA without paying royalties or licensing fees, fostering innovation and collaboration. As RISC-V continues to gain popularity, especially in embedded systems, IoT devices, and security-critical applications, the demand for skilled professionals who understand this architecture is on the rise. Learning to hack on RISC-V not only equips you with knowledge of a cutting-edge technology but also positions you at the forefront of a rapidly expanding field.

Why Hacking?

Hacking, in its truest sense, is about understanding systems at a fundamental level, finding creative solutions to problems, and pushing the boundaries of what is possible. In the context of RISC-V, hacking can involve anything from reverse engineering binaries to exploiting vulnerabilities in hardware or firmware. By learning to hack RISC-V, you gain insight into how modern computing systems work, how to identify and mitigate security risks, and how to leverage the unique features of RISC-V for your own projects. Whether you're interested in cybersecurity, embedded systems, or just enjoy the challenge of solving complex problems, hacking on RISC-V offers a rich and rewarding experience.

Why the ESP32-C3-DevKitM-1?

The ESP32-C3-DevKitM-1 development board is a powerful yet cost-effective platform that is perfect for hacking and experimentation. Based on the ESP32-C3, a low-power, single-core RISC-V microcontroller with integrated Wi-Fi and Bluetooth, this development board offers a wide range of features for IoT and embedded applications. Its RISC-V core makes it an ideal candidate for learning and hacking RISC-V, while its built-in peripherals provide ample opportunities for hands-on projects. Whether you're interested in building a custom IoT device, experimenting with low-level firmware, or exploring the security of wireless communications, the ESP32-C3-DevKitM-1 offers a versatile platform to bring your ideas to life.

Why PlatformIO and Visual Studio Code?

Setting up a development environment can often be a daunting task, especially for those new to embedded systems or hacking. PlatformIO, combined with Visual Studio Code, simplifies this process by providing a user-friendly interface and a powerful set of tools for developing, debugging, and deploying firmware. PlatformIO supports a wide range of microcontrollers, including the ESP32-C3, and integrates seamlessly with Visual Studio Code, one of the most popular code editors in the world. With PlatformIO, you can easily configure your projects, manage dependencies, and even upload your code to the ESP32-C3-DevKitM-1 with just a few clicks. This streamlined workflow allows you to focus on what matters most: learning and hacking RISC-V.

Practical Projects: The Key to Mastery

Theory alone is not enough to master hacking on RISC-V. To truly understand the architecture and develop your skills, you need to engage in practical projects that challenge you to apply what you've learned. The combination of the ESP32-C3-DevKitM-1 and PlatformIO provides the perfect environment for these projects. From writing custom firmware to exploiting vulnerabilities in existing systems, the hands-on experience you gain will deepen your understanding and prepare you for real-world challenges. Throughout this book, we will guide you through a series of projects that will not only teach you the fundamentals of RISC-V hacking but also inspire you to explore new ideas and create your own innovations.

Conclusion

Learning to hack RISC-V is not just about mastering a new ISA; it's about gaining the skills and knowledge needed to thrive in a world where open-source hardware is becoming increasingly important. The ESP32-C3-DevKitM-1 development board, combined with PlatformIO and Visual Studio Code, provides an accessible and powerful platform for your journey into RISC-V hacking. As you progress through this book, you will gain the confidence to tackle complex challenges, the creativity to develop innovative solutions, and the expertise to contribute to the growing RISC-V community. Welcome to the world of RISC-V hacking—your adventure begins here.

CHAPTER 2: SETUP ENVIRONMENT

Introduction

Before diving into the exciting world of RISC-V programming on the ESP32-C3, it's essential to have the right tools and setup. This chapter will guide you through purchasing the ESP32-C3-DevKitM-1 development board, setting up your development environment with Visual Studio Code (VS Code), installing PlatformIO and C++ plugins, and configuring OpenOCD for debugging. By the end of this chapter, you'll be ready to start coding and debugging your projects.

1. Purchase the ESP32-C3-DevKitM-1 Development Board and Accessories

The ESP32-C3-DevKitM-1 is a versatile and powerful development board that is perfect for exploring RISC-V architecture. This board is compact, feature-rich, and provides everything you need to get started with RISC-V on the ESP32-C3.

Purchase Links:

https://www.amazon.com/Espressif-ESP32-C3-DevKitM-1-Development-Board/dp/B08W2J9B8J

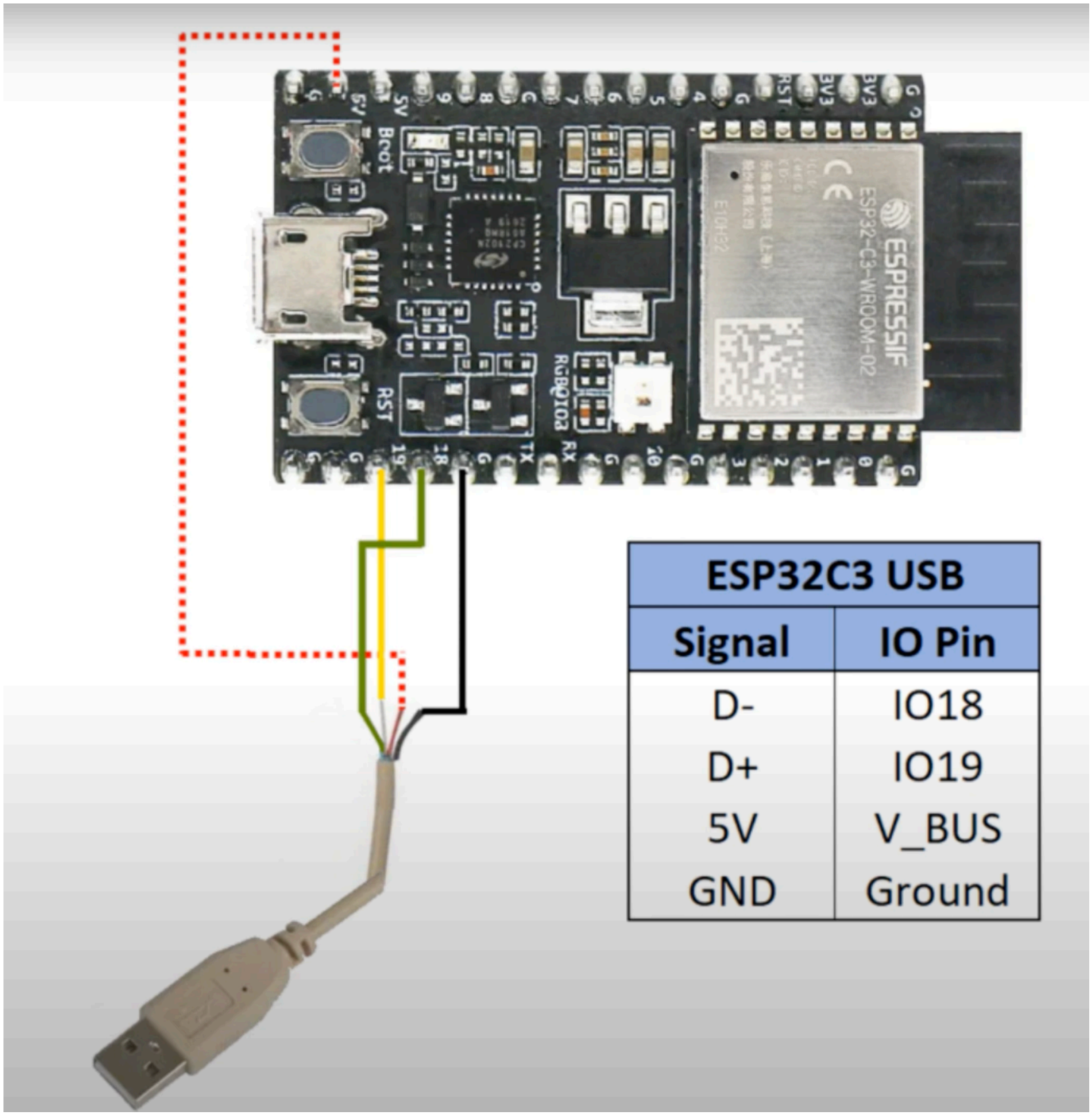
https://www.amazon.com/DGZZI-PL2303TA-Console-Serial-Raspberry/dp/B07W42V16T

https://www.amazon.com/Syntech-Adapter-Thunderbolt-Compatible-MacBook/dp/B07CVX3516

https://www.amazon.com/EL-CK-002-Electronic-Breadboard-Capacitor-Potentiometer/dp/B01ERP6WL4

2. Setup USB Cable

Follow the diagram to wire up your ESP32-C3 for flashing and debugging.



ESP32C3 USB	
Signal	IO Pin
D-	IO18
D+	IO19
5V	V_BUS
GND	Ground

2. Set Up Visual Studio Code (VS Code)

Visual Studio Code (VS Code) is a powerful and popular code editor that is widely used for embedded systems development. It supports multiple programming languages, including C++, and provides excellent tools for debugging and code management.

2.1 Download and Install VS Code

1. Visit the Visual Studio Code website - https://code.visualstudio.com.

2. Download the installer for your operating system (Windows, macOS, or Linux).

3. Follow the on-screen instructions to install VS Code.

2.2 Install the PlatformIO and C++ Plugins

PlatformIO is an open-source ecosystem that supports multiple development platforms, including the ESP32. It simplifies the process of writing, testing, and debugging embedded software.

1. Open VS Code.

2. Go to the Extensions view by clicking on the Extensions icon in the Activity Bar on the side of the window or by pressing `Ctrl+Shift+X`.

3. In the search box, type `PlatformIO IDE` and click "Install."

4. Next, search for `C/C++` and click "Install" to add the C++ extension, which provides language support for C++.

3. Install OpenOCD

OpenOCD (Open On-Chip Debugger) is essential for debugging your projects. Below are the installation steps for Windows, macOS, and Linux.

3.1 Windows

1. Download the latest release of OpenOCD for Windows from the [official website](http://openocd.org/getting-openocd).

2. Extract the downloaded ZIP file to a directory of your choice.

3. Add the OpenOCD *bin* directory to your system's *PATH* environment variable:

- Right-click *This PC* or *Computer*, and select *Properties*.

- Click on *Advanced system settings*, and then *Environment Variables*.

- Find the *Path* variable, click *Edit*, and add the path to the OpenOCD *bin* directory.

3.2 macOS

1. Open Terminal.

2. Install OpenOCD using Homebrew:

```
brew install openocd
```

3.3 Linux

1. Open Terminal.

2. Install OpenOCD using your package manager:

- Debian/Ubuntu:

```
sudo apt-get install openocd
```

- Fedora:

```
sudo dnf install openocd
```

- Arch Linux:

```
sudo pacman –S openocd
```

Conclusion

By following the steps outlined in this chapter, you will have all the necessary tools and configurations to start developing RISC-V projects on the ESP32-C3-DevKitM-1. With your development environment set up, you're ready to dive into the code and explore the capabilities of RISC-V on this powerful platform.

CHAPTER 3: HELLO, WORLD

Introduction

In this chapter, we will walk through creating your first "Hello World" project using PlatformIO with the ESP32-C3 and the Arduino framework. This project will demonstrate how to integrate an assembly function into a C++ project.

1. Open Visual Studio Code and Create a New PlatformIO Project

1.2 Create a New PlatformIO Project

1. Click on the PlatformIO icon in the left sidebar.

2. Click on the “New Project” button.

3. In the "Project Name" field, enter *0x01_hello_world*.

4. For "Board," select *Espressif ESP32-C3-DevKitM-1*.

5. For "Framework," select *Arduino*.

6. Choose a location to save the project and click “Finish.”

PlatformIO will now create a new project with the necessary folders and files.

2. Modify the *main.cpp* File

Once your project is created, you'll find a *main.cpp* file inside the *src* folder. Replace the contents of this file with the following code:

```
#include <Arduino.h>


// Declare the function implemented in assembly
extern "C" void hello_world();


void setup() {
    Serial.begin(115200);
    hello_world();  // Call the assembly function directly
}

void loop() {
}
```

This *main.cpp* file initializes the serial communication at a baud rate of 115200 and calls the *hello_world* function, which will be implemented in assembly language.

3. Create the Assembly Function

Next, you'll create a new file to hold your assembly code.

3.1 Create the *main.S* File

1. In the *src* folder, create a new file named *main.S*.

2. Add the following assembly code:

```
.global hello_world


hello_world:
    li      t0, 0
    ret
```

This simple assembly code defines a function named *hello_world* that does nothing and returns immediately.

4. Build and Upload the Project

With both the *main.cpp* and *main.S* files in place, you're ready to build and upload the project to your ESP32-C3 board.

1. Click on the checkmark icon in the bottom toolbar of VS Code to build the project.

2. If the build is successful, click on the right arrow icon to upload the code to your ESP32-C3 board.

5. Configure PlatformIO for the ESP32-C3

Once you have PlatformIO installed, you need to configure your project for the ESP32-C3. Below is a sample *platformio.ini* file that you can use for your projects:

```
[env:esp32-c3-devkitm-1]

platform = espressif32

board = esp32-c3-devkitm-1

framework = arduino

monitor_speed = 115200

debug_tool = esp-builtin

debug_init_break = break setup

build_type = debug
```

- **platform**: Specifies the platform you are working with, in this case, *espressif32*.

- **board**: The specific board you are using, which is *esp32-c3-devkitm-1*.

- **framework**: Indicates that you are using the Arduino framework.

- **monitor_speed**: Sets the serial monitor speed to *115200* baud.

- **debug_tool**: Specifies the built-in debugger for the ESP32-C3.

- **debug_init_break**: Sets an initial breakpoint at the `setup()` function.

- **build_type**: Configures the build for debugging.

Conclusion

Congratulations! You've successfully created a "Hello World" project that integrates a simple assembly function into your C++ code on the ESP32-C3 using the Arduino framework. This setup forms the foundation for more advanced projects that combine the power of C++ with the efficiency of assembly language.