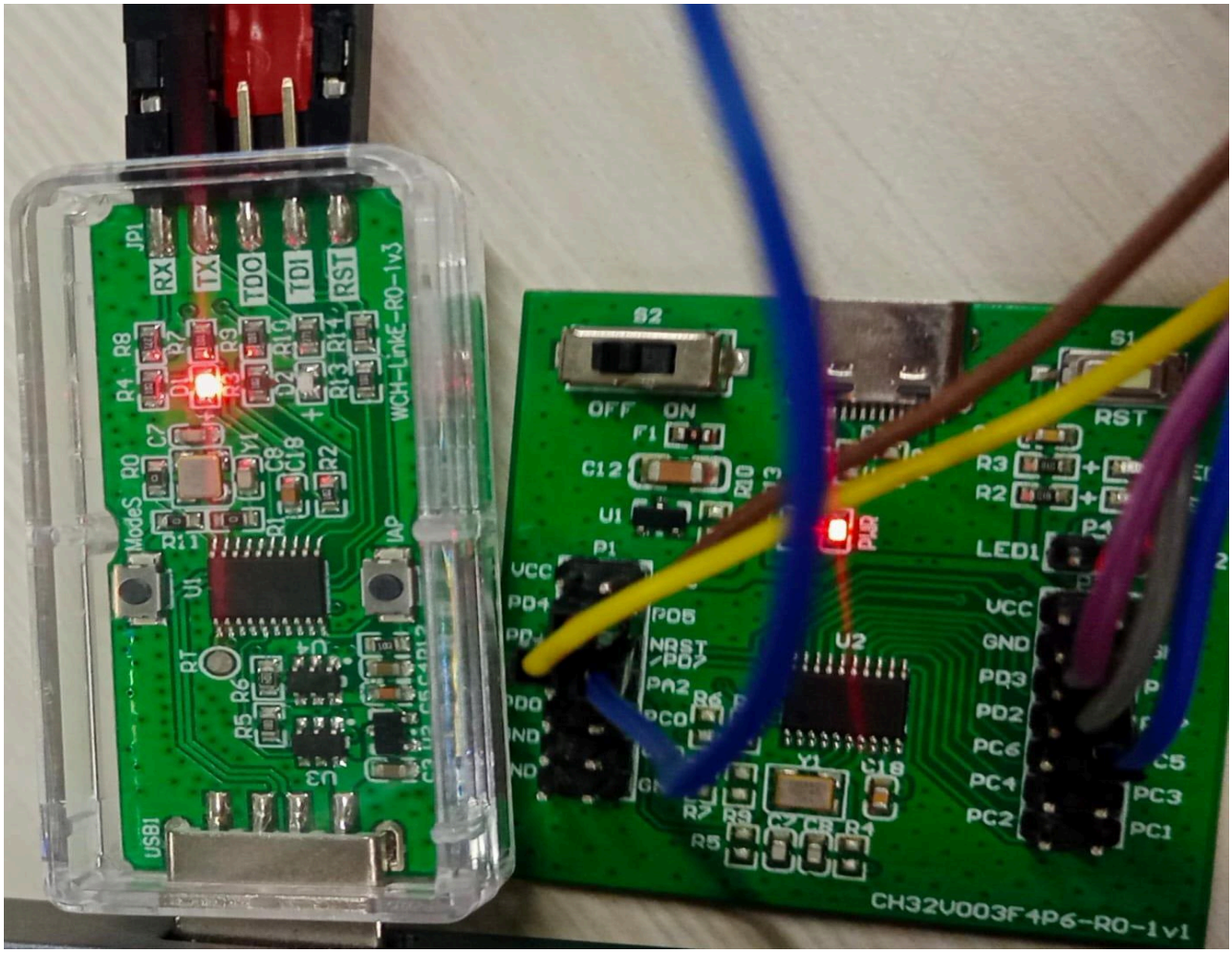
Exploring RISC-V: A Practical Guide with CH32V003 Microcontroller Development Board



INDEX

Sl. No.	**Title**	**Pg.No**
1	Chapter 1: RISC-V Architecture	2
2	Chapter 2: Microcontroller Basics	3
2.1	Introduction to Microcontrollers	3
2.2	Overview of CH32V003 Microcontroller	3
2.3	Microcontroller Components and Their Functions	5
2.4	Microcontroller vs. Microprocessor	6
2.5	CH32V003 Development Workflow	6
2.6	Why Use CH32V003 for Embedded Development?	6
3	Chapter 3: Development Tools	11
3.1	Introduction	11
3.2	Moun River Studio (MRS)	11
3.3	WCH-LinkE Debugger & Programmer	12
3.4	RISC-V Toolchain (Compiler & Debugger)	12
3.5	Development Workflow Summary	12
4	Chapter 4: Example Programs	13
I	Useful Tips	16

Chapter 1: RISC-V Architecture

RISC-V is an open-source, royalty-free instruction set architecture (ISA) based on the principles of reduced instruction set computing (RISC). It was developed at UC Berkeley and has gained widespread adoption due to its flexibility, modularity, and openness.

Key Features of RISC-V

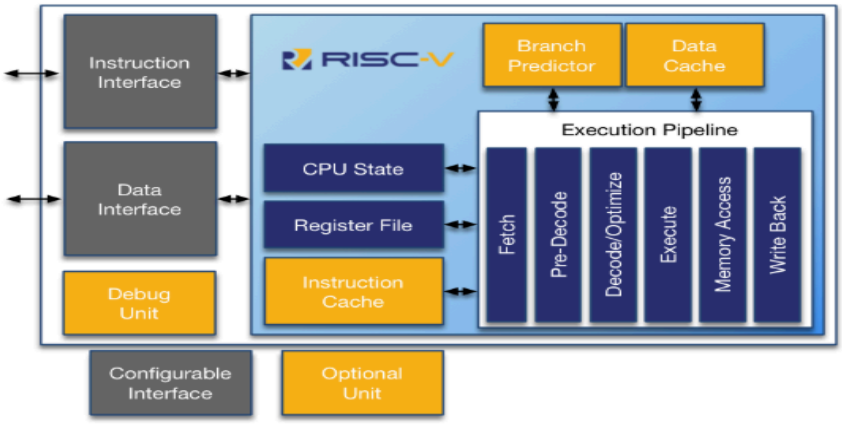
- **Open-Source & Free:** No licensing fees, allowing anyone to develop and modify processors.
- **Modular ISA:** Comes in base integer variants (e.g., RV32I, RV64I) with optional extensions (e.g., M for multiplication, A for atomic operations, F for floating-point, etc.).
- **Scalability:** Can be used in embedded systems, IoT devices, AI accelerators, and high-performance computing.
- **Growing Ecosystem:** Supported by organizations like RISC-V International, with hardware implementations from companies like SiFive, Andes, and Ventana.

Common RISC-V Implementations

- **SiFive Freedom Processors** (Used in embedded and industrial applications)
- **ESP32-C3** (Espressif's RISC-V-based Wi-Fi + BLE microcontroller)
- **HiFive Unmatched & Unleashed** (Development boards for high-performance applications)
- **StarFive VisionFive 2** (RISC-V-based single-board computer)
- **CH32V00X**

Applications of RISC-V

- IoT and embedded devices
- AI accelerators, Cloud computing and data centers
- Edge computing, Custom silicon development



Chapter 2: Microcontroller Basics

2.1 Introduction to Microcontrollers

A microcontroller (MCU) is a small computing device designed for embedded applications. Unlike general-purpose computers, microcontrollers integrate a **processor (CPU), memory (RAM/Flash), and peripherals (GPIO, timers, communication interfaces, etc.)** into a single chip.

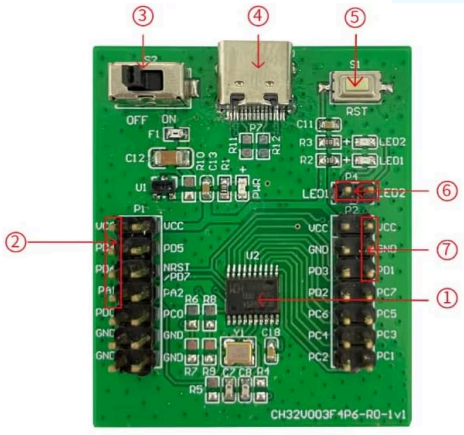
Key Features of a Microcontroller

- **CPU (Central Processing Unit)** – Executes instructions and processes data.
- **Flash Memory** – Stores the program (firmware) permanently.
- **RAM (Random Access Memory)** – Temporary memory for storing variables.
- **I/O Ports (Input/Output)** – Interface with external components like sensors and actuators.
- **Timers and PWM** – Used for time-based operations and motor control.
- **Communication Interfaces** – UART, SPI, I²C for communication.

2.2 Overview of CH32V003 Microcontroller

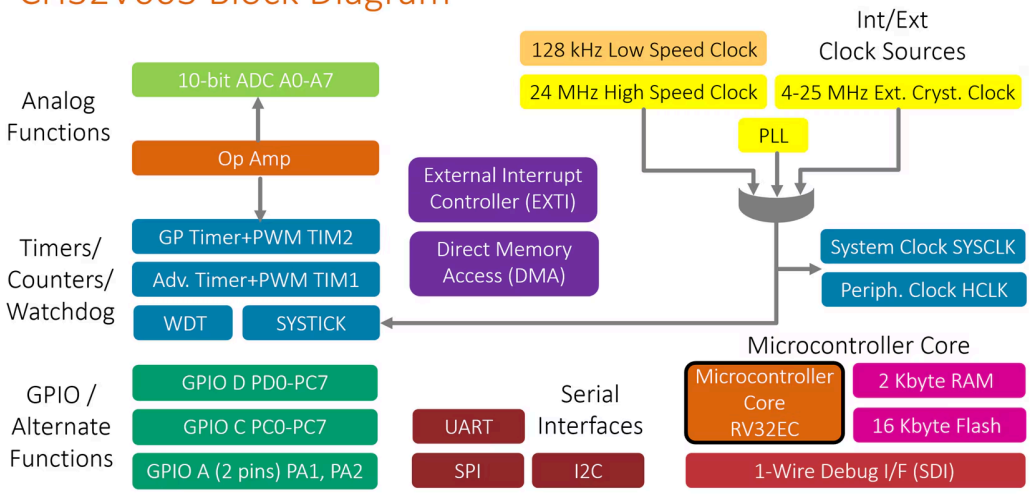
The **CH32V003F4P6** is an ultra-low-cost **RISC-V 32-bit** microcontroller developed by **WCH**. It is optimized for **embedded applications**, offering high performance with minimal power consumption.

Specifications of CH32V003F4P6



Feature	Description
CPU Core	RISC-V 32-bit (RV32EC)
Clock Speed	Up to 48 MHz
Flash Memory	16 KB
RAM	2 KB
GPIO	18 general-purpose I/O pins
Timers	2 general-purpose timers
Communication	UART, I²C, SPI
Analog Support	1 × 10-bit ADC
Power Modes	Sleep, Deep Sleep
Operating Voltage	3.3V
Programming Interface	SWD (via WCH-LinkE)

CH32V003 Block Diagram



No.	Interfaces/Devices	Description
1	Master control MCU	CH32V003F4P6
2	MCU I/O ports	I/O pinout interface of the master MCU
3	Power switch S2	Used for disconnecting or connecting external 5V power supply or USB power supply
4	USB interface	Power supply only, no USB function
5	Key S1	Reset key for external manual reset of the main MCU
6	LED	LEDs are connected to the main chip I/O port via LED row pins (P4)
7	DEBUG interface	Used for downloading, simulation debugging, single-wire communication, only need SWDIO to connect PD1

2.3 Microcontroller Components and Their Functions

1. Central Processing Unit (CPU)

- Executes instructions stored in Flash memory.
- Handles arithmetic operations, logical operations, and data movement.
- Works on **RISC-V RV32EC** instruction set in CH32V003.

2. Memory System

- **Flash Memory** – Stores the program permanently.
- **RAM (SRAM)** – Stores temporary data, variables, and stack.

3. General-Purpose Input/Output (GPIO)

- Used to interface with external components (LEDs, buttons, sensors, etc.).
- Configurable as **input or output**.

4. Timers and PWM

- **Timers**: Used for delays, counting, and event triggering.
- **PWM (Pulse Width Modulation)**: Used for motor control, LED dimming, etc.

5. Communication Interfaces

- **UART (Universal Asynchronous Receiver-Transmitter)** – Serial communication with PCs, sensors, or other MCUs.
- **SPI (Serial Peripheral Interface)** – Fast communication with peripherals like displays and sensors.
- **I²C (Inter-Integrated Circuit)** – Used to communicate with multiple devices over two wires.

6. Analog-to-Digital Converter (ADC)

- Converts analog signals (e.g., sensor data) into digital values.
- CH32V003 has a **10-bit ADC** for reading voltage levels.

7. Power Management

- Supports **Sleep and Deep Sleep modes** for power-saving applications.
- Works on **3.3V supply voltage**.

2.4 Microcontroller vs. Microprocessor

Feature	Microcontroller (MCU)	Microprocessor (MPU)
Definition	Small integrated system with CPU, memory, and peripherals	CPU that requires external memory and peripherals
Usage	Embedded systems, automation, IoT	Computers, servers, high-performance applications
Power Consumption	Low	High
Examples	CH32V003, ATmega328, STM32	Intel Core i7, AMD Ryzen

2.5 CH32V003 Development Workflow

1. **Writing Code**
   - Use **C/C++** programming with **Moun River Studio**.
2. **Compiling and Building**
   - Convert code into machine language using the **RISC-V toolchain**.
3. **Flashing Firmware**
   - Upload the compiled code to the CH32V003 microcontroller using **WCH-LinkE**.
4. **Debugging**
   - Use **SWD (Serial Wire Debug)** to debug and test the program.

2.6 Why Use CH32V003 for Embedded Development?

- Low-cost and open-source RISC-V architecture.
- Compact and efficient for IoT and automation projects.
- Rich set of peripherals (UART, SPI, I²C, ADC, Timers, etc.).
- Easy to program using Moun River Studio and WCH-LinkE.

WCH LINK E

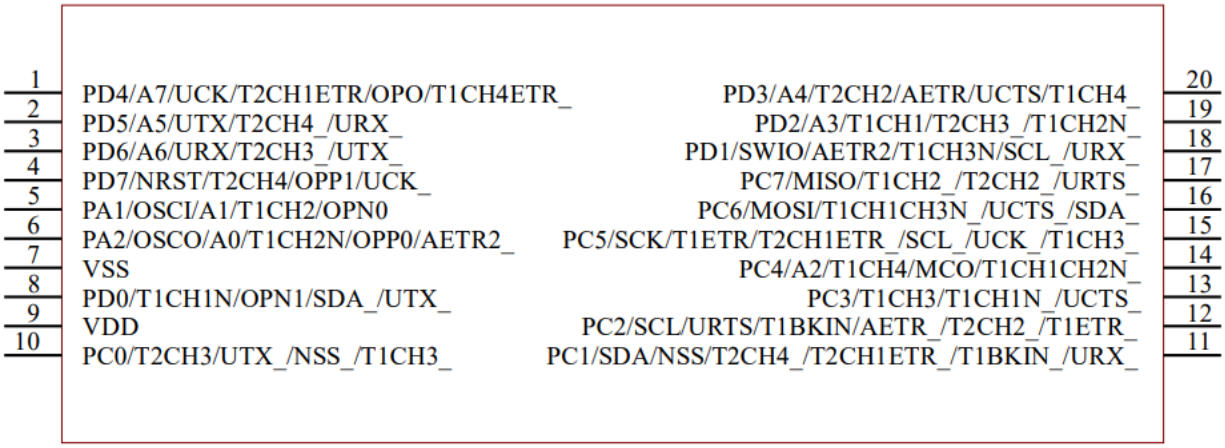
The **WCH-LinkE** is a low-cost **debugger and programmer** designed for WCH's RISC-V and ARM microcontrollers, including the **CH32V003** series. It is widely used for **flashing firmware and debugging RISC-V MCUs** from WCH.

🔹 Key Features of WCH-LinkE

✅ **Supports RISC-V (CH32V) and ARM (CH32F) MCUs**
✅ **SWD and WCH-Link debugging protocols**
✅ **USB interface** – Plugs directly into your PC
✅ **Supports OpenOCD** – Can be used in open-source toolchains
✅ **Compatible with Moun River IDE** – WCH's official development environment

🔹 How It Works

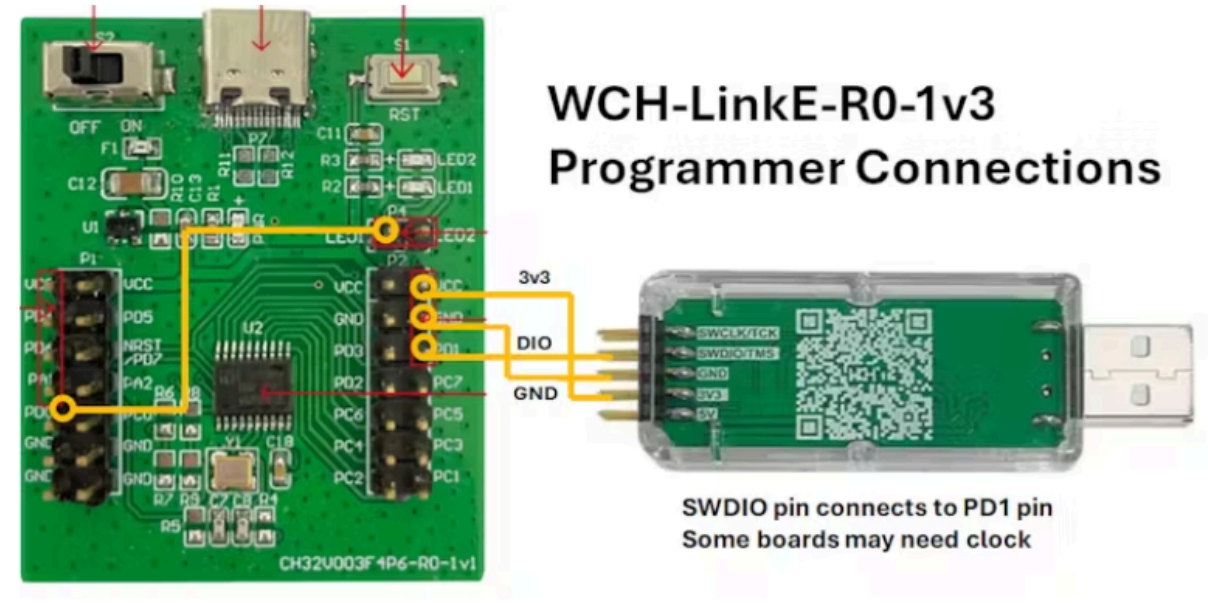
- **Programming & Debugging:** Used to flash firmware onto **CH32V003** and other WCH MCUs.
- **SWD Interface:** Allows debugging using breakpoints and stepping through code.
- **Works with Moun River Studio:** The official WCH IDE for RISC-V development.

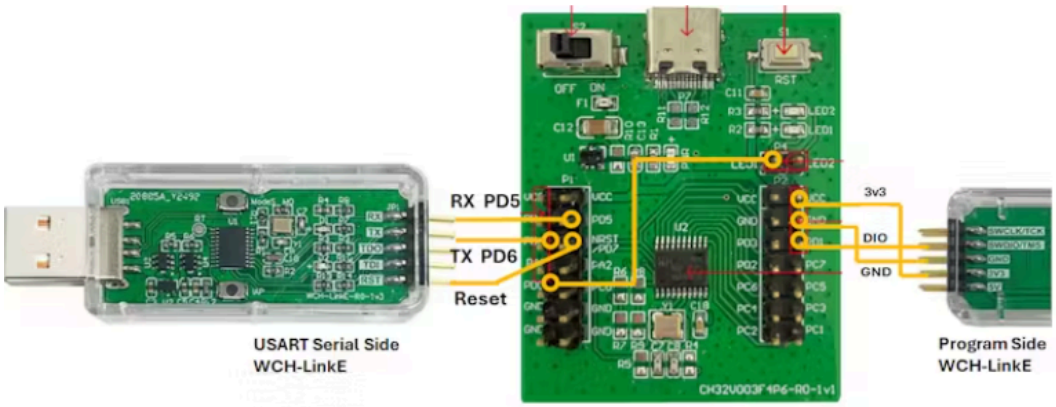


[PINOUT CH32V003F]

Connections WCH Link E to CH32V003F

WCH LinkE	CH32V003F4P6	WCH LinkE	CH32V003F4P6
5V	vcc	3v3	vcc
GND	GND	SWDIO/TMS	PD1
SWCLK/TCK	-	Rx	PD5
Tx	PD6	RST	NRST/PD7
TDO	-	TDI	-


WCH-LinkE-R0-1v3
Programmer Connections
3v3
DIO
GND
SWDIO pin connects to PD1 pin
Some boards may need clock


RX PD5
TX PD6
Reset
USART Serial Side
WCH-LinkE
3v3
DIO
GND
Program Side
WCH-LinkE

WCH-LinkE physical picture

Module Pin Interface	Target Board Interface	Remarks
5V	MCU-5V power interface	Connect 5V power supply of the target board
3.3V	MCU-3.3V power interface	Connect the 3.3V power supply of the target board
GND	MCU-GND	Connect GND of the target board
SWCLK	MCU-SWCLK	Connect SWCLK of the target board
SWDIO	MCU-SWDIO	Connect SWDIO of the target board
RX	MCU-UART-TX	Connect the UART - RX of the target board
TX	MCU-UART-RX	Connect the UART - RX of the target board
ModeS	\	Press and hold the ModeS key to power up until the status indicator lights up to complete the mode switch
IAP	\	WCH - LinkE manual update key

Chapter 3: Development Tools

3.1 Introduction

To work with the **CH32V003F4P6** microcontroller, we need a proper development environment. This includes an **Integrated Development Environment (IDE)** for writing and compiling code, a **debugger/programmer** for flashing firmware, and a **toolchain** to convert code into machine-readable instructions.

The primary development tools for CH32V003 include:

1. **Moun River Studio (MRS)** – IDE for coding, compiling, and debugging.
2. **WCH-LinkE** – A programmer/debugger for flashing and debugging firmware.
3. **RISC-V Toolchain** – Compiler and linker to convert code into executable firmware.

3.2 Moun River Studio (MRS)

What is Moun River Studio?

Moun River Studio is an **Integrated Development Environment (IDE)** designed for WCH microcontrollers. It provides:

- **Code Editor** – For writing C/C++ programs.
- **Compiler and Debugger** – Built-in support for RISC-V development.
- **Flashing Utility** – Directly upload code to CH32V003 using WCH-LinkE.

Installing Moun River Studio

1. Download MRS from the **WCH official website**.
2. Install and configure the IDE.
3. Set up the **CH32V003 SDK** for project development.

Creating a New Project in MRS

1. Open Moun River Studio.
2. Select **File → New Project**.
3. Choose **CH32V003F4P6** as the target microcontroller.
4. Write your program in **main.c**.
5. Build and flash the firmware using **WCH-LinkE**.

3.3 WCH-LinkE Debugger & Programmer

What is WCH-LinkE?

The **WCH-LinkE** is a hardware debugging tool that allows:

- **Programming (Flashing)** firmware into the CH32V003 microcontroller.
- **Debugging** using **SWD (Serial Wire Debug)** interface.

3.4 RISC-V Toolchain (Compiler & Debugger)

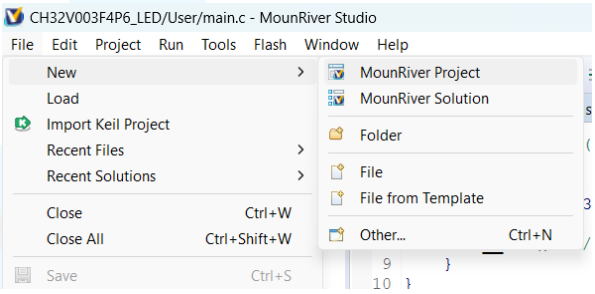
What is a Toolchain?

A **toolchain** is a set of programs that convert human-readable code into machine language. The RISC-V toolchain for CH32V003 includes:

- **Compiler (GCC-RISC-V)** – Converts C/C++ code into machine instructions.
- **Linker** – Combines compiled files into an executable.
- **Debugger (GDB)** – Helps debug programs on the microcontroller.

Installing the RISC-V Toolchain

The RISC-V toolchain comes bundled with **Moun River Studio**, so no separate installation is needed.



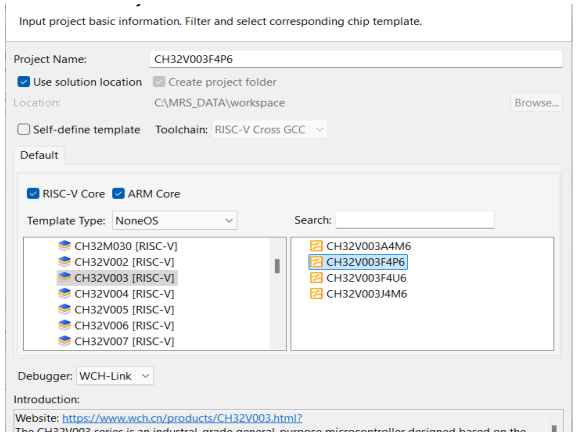
3.5 Development Workflow Summary

1. **Write Code** in Moun River Studio.
2. **Compile** using the RISC-V toolchain.
3. **Flash Firmware** using WCH-LinkE.
4. **Debug** using SWD and Moun River Studio.

   C:\MounRiver\MounRiver_Studio\toolchain\RISC-V Embedded GCC\bin

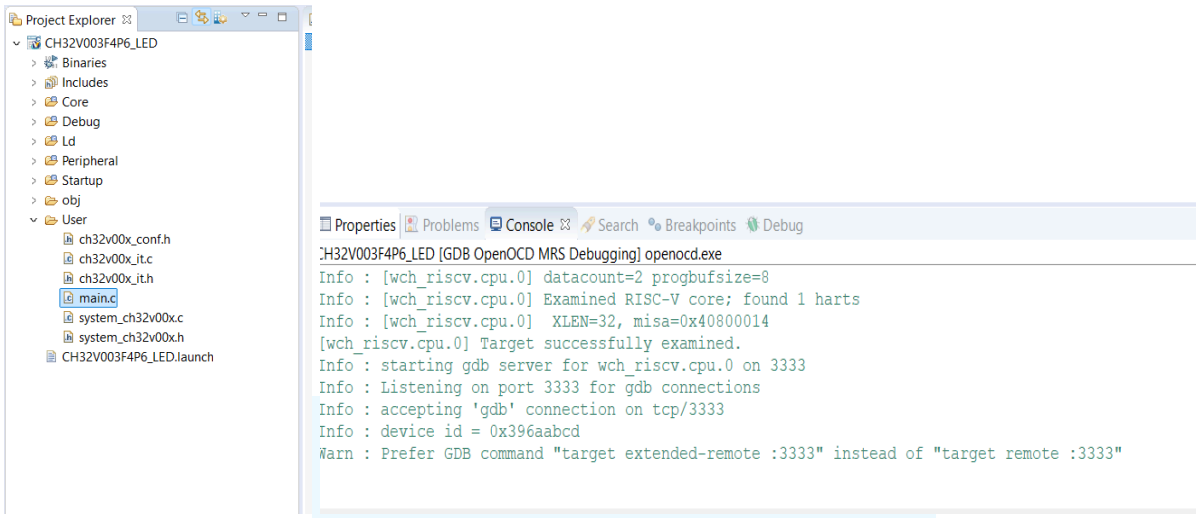
   Ensure the bin has RISC -V embed gcc.exe

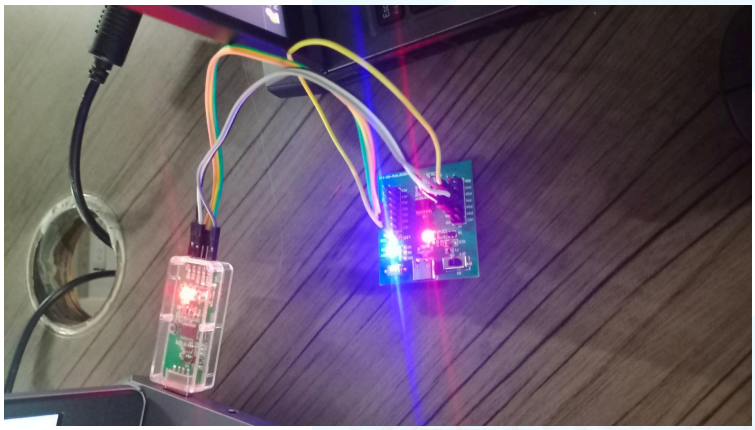
   Set toolchain path in windows environment variables and preferences risc v toolchain

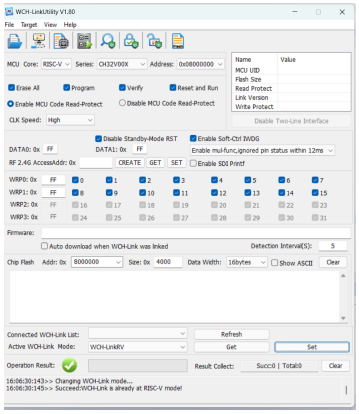


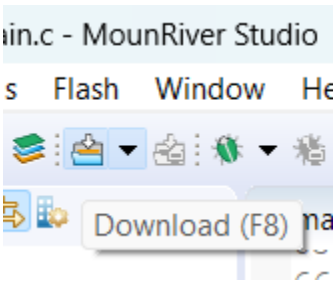
Chapter 4: Example Programs

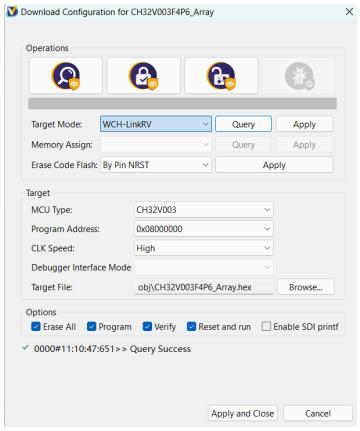
Program 1: LED Blinking:https://github.com/Neethu-Jaisan/MounRiver/blob/main/LED.c





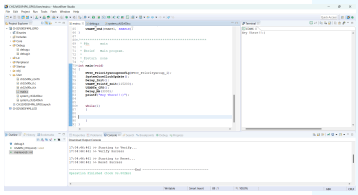




In the linkE programmer gets disconnected and won't show the status.. In moun river use download option(F8) 

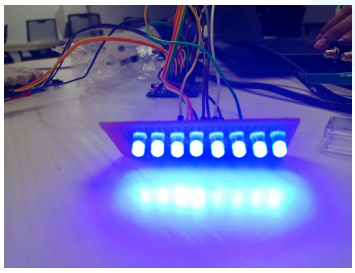
[#failed to configure MCU]

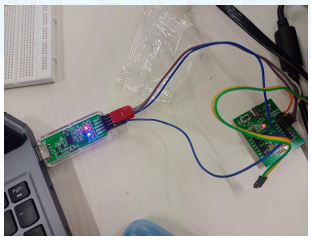
Program 2: U(s)art; "Hey There!!:)" Print in terminal : https://github.com/Neethu-Jaisan/MounRiver/blob/main/u(s)art_printf_pgm.c



Program 3: Blink all 8 LED Array

https://github.com/Neethu-Jaisan/MounRiver/blob/main/LED8ArrayBlinkall.c



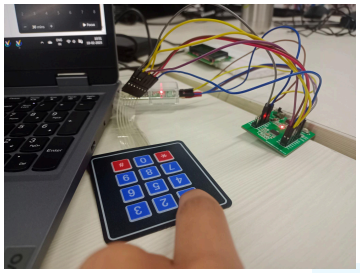


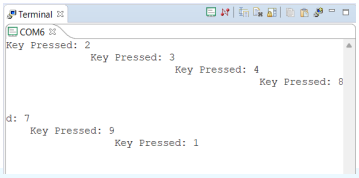
Program 4: LED Chaser(1/2/4 pattern)

https://github.com/Neethu-Jaisan/MounRiver/blob/main/LEDchaser.c

Program 5: 4*3 matrix Keypad

https://github.com/Neethu-Jaisan/MounRiver/blob/main/4*3matrixkeypad.c

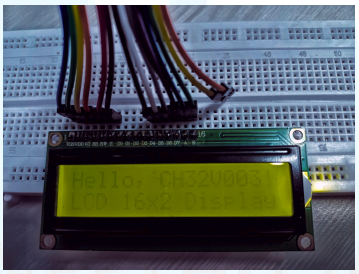




Program 6: 16*2 LCD Display Interfacing

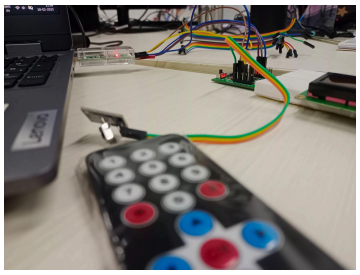
https://github.com/Neethu-Jaisan/MounRiver/blob/main/16*2%20CD%20Display%20Interfacing.c

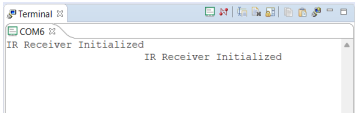
LCD Pin	Microcontroller Pin
VSS	GND
VDD	5V
V0	Potentiometer (Contrast)
RS	PC0
RW	GND (Always Write Mode)
E	PC1
D4	PC2
D5	PC3
D6	PC4
D7	PC5
A (LED+)	5V
K (LED-)	GND



Program 7: IR Receiver Detection

https://github.com/Neethu-Jaisan/MounRiver/blob/main/IR%20Reciever.c





Useful Tips

🔹 Setup & Development

✅ **Use the Latest WCH Drivers & Firmware** – Ensure WCH-LinkE is updated for stable programming and debugging.
✅ **Moun River Studio (MRS) Configuration** – Set the correct **target MCU (CH32V003F4P6)** before compiling.
✅ **Use F8 (Download Option)** – If **MCU fails to configure**, use **F8 (Download)** instead of **Run**.

🔹 Hardware & Debugging

✅ **Check WCH-LinkE Connections** – Properly connect **SWDIO (PD1), SWCLK (PD5), and NRST (PD7)** for stable flashing.
✅ **Avoid Using PC5 & PC6** – These are **SWD debugging pins**, not general-purpose GPIOs.
✅ **Use UART for Debugging** – If WCH-LinkE is unreliable, use **PD6 (TX) & PD5 (RX)** for serial debugging.

🔹 Power & Performance

✅ **Use a 3.3V Power Supply** – CH32V003 operates at **3.3V**, using **5V peripherals may damage it**.
✅ **Enable Low-Power Modes** – Use **Sleep & Deep Sleep** to reduce power consumption in battery-powered projects.

🔹 Code Optimization & Troubleshooting

✅ **Use Interrupts Efficiently** – Avoid blocking delays (`delay()`), use **timers & interrupts** for real-time performance.
✅ **Check Peripheral Registers** – Incorrect **I²C, SPI, UART, or ADC** configurations can cause malfunctions.
✅ **Refer to Official WCH Documentation** – Use the **CH32V003 datasheets & reference manuals** for accurate register settings.

🔗 **Additional Resources:**
📌 **GitHub:** CH32V003 Official Repo

📌 **GitHub:** https://github.com/Neethu-Jaisan/MounRiver

CH32V003F4P6_YT_Tutorials: https://curiousscientist.tech/blog/ch32v003f4p6-setup-and-gpios