Rust meets cheap bare-metal RISC-V

Marcel Ziswiler

Speaking today

Marcel Ziswiler

ROLE

Founder ZisiSoft GmbH

Senior Software Engineer
Codethink Ltd.

TENURE

Joined Codethink
in 2024

PAST ENGAGEMENTS

Senior Linux Expert, System Engineer, Technical Project Leader
Noser Engineering

Platform Manager Embedded Linux
Toradex

EDUCATION

MS Computer Science
ETH Zurich

Certificate in Embedded Systems Technologies UCI

Contents

WinChipHead (WCH) CH32V003

WinChipHead (WCH) CH32V003

- Nanjing Qinheng Microelectronics, company behind WCH (WinChipHead) brand, founded in 2004 in Nanjing
- 32-bit General-purpose RISC-V MCU
- QingKe 32-bit RISC-V2A processor, supports 2-level deep interrupt nesting
- Up to 48 MHz system main frequency
- 2 KB SRAM, 16 KB Flash
- Supply voltage: 3.3 or 5 Volts
- Multiple low-power modes: Sleep/Standby
- Power on/off reset
- Programmable voltage monitor
- 1 set of 1-channel general-purpose DMA
- Operational Amplifier/Comparator (OPAs)
- 1 set of 10-bit ADC

WinChipHead (WCH) CH32V003 cont.

- 1×16-bit advanced-control timer
- 1×16-bit general-purpose timer
- 2 Watchdog timers and 1×32-bit system time base timer
- 1 USART interface, 1 I2C interface, 1 SPI interface
- 18 GPIOs aka I/O ports, can be mapped to 1 external interrupt
- 96-bit chip unique ID
- 1-wire Serial Debug Interface (SDI)
- Industrial-grade temperature range of -40°C to 85°C
- Package: TSSOP20 (e.g. CH32V003F4P6), QFN20, SOP16, SOP8
- English Datasheet (37 pages) and Reference Manual (188 pages)

Part NO.	Freq	Flash	SRAM	GPIO	Timer				ADC (10bit) Unit/CH	OPA	U(S)ART	I²C	SPI	VDD	Package
					Adv (16bit)	GP (16bit)	WDOG	SysTick (32bit)							
CH32V003J4M6	48MHz	16K	2K	6	1	1	2	√	1/6	1	1	1	-	3.3/5.0	SOP8
CH32V003A4M6	48MHz	16K	2K	14	1	1	2	√	1/6	1	1	1	-	3.3/5.0	SOP16
CH32V003F4P6	48MHz	16K	2K	18	1	1	2	√	1/8	1	1	1	1	3.3/5.0	TSSOP20
CH32V003F4U6	48MHz	16K	2K	18	1	1	2	√	1/8	1	1	1	1	3.3/5.0	QFN20

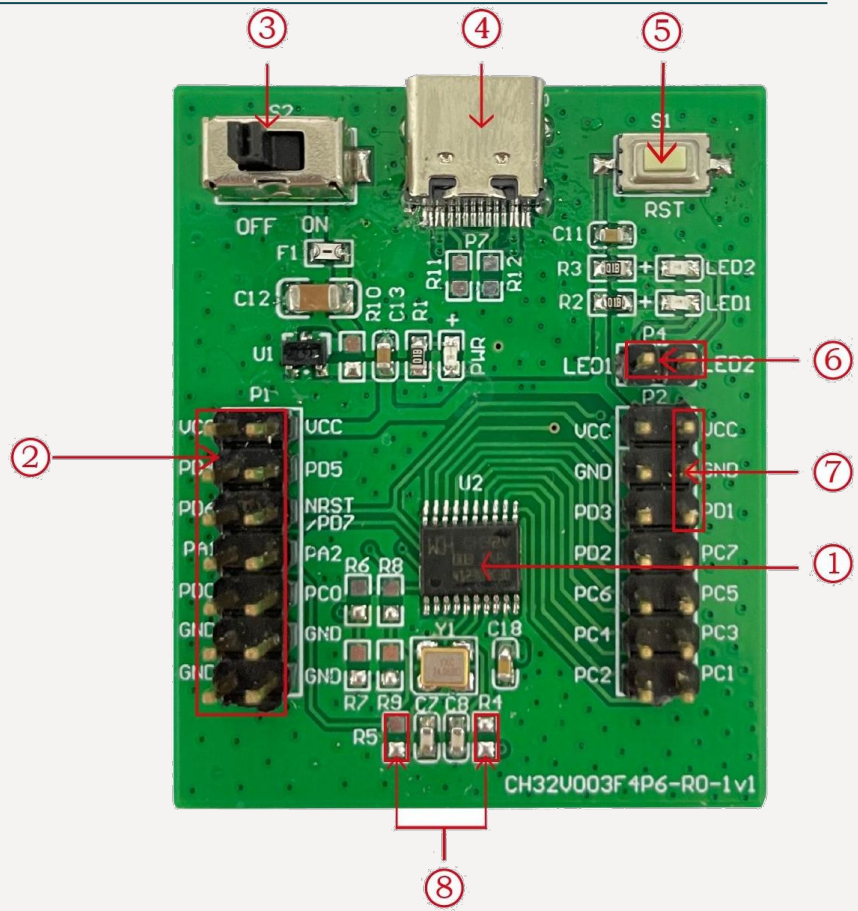
QingKe RISC-V2A Core

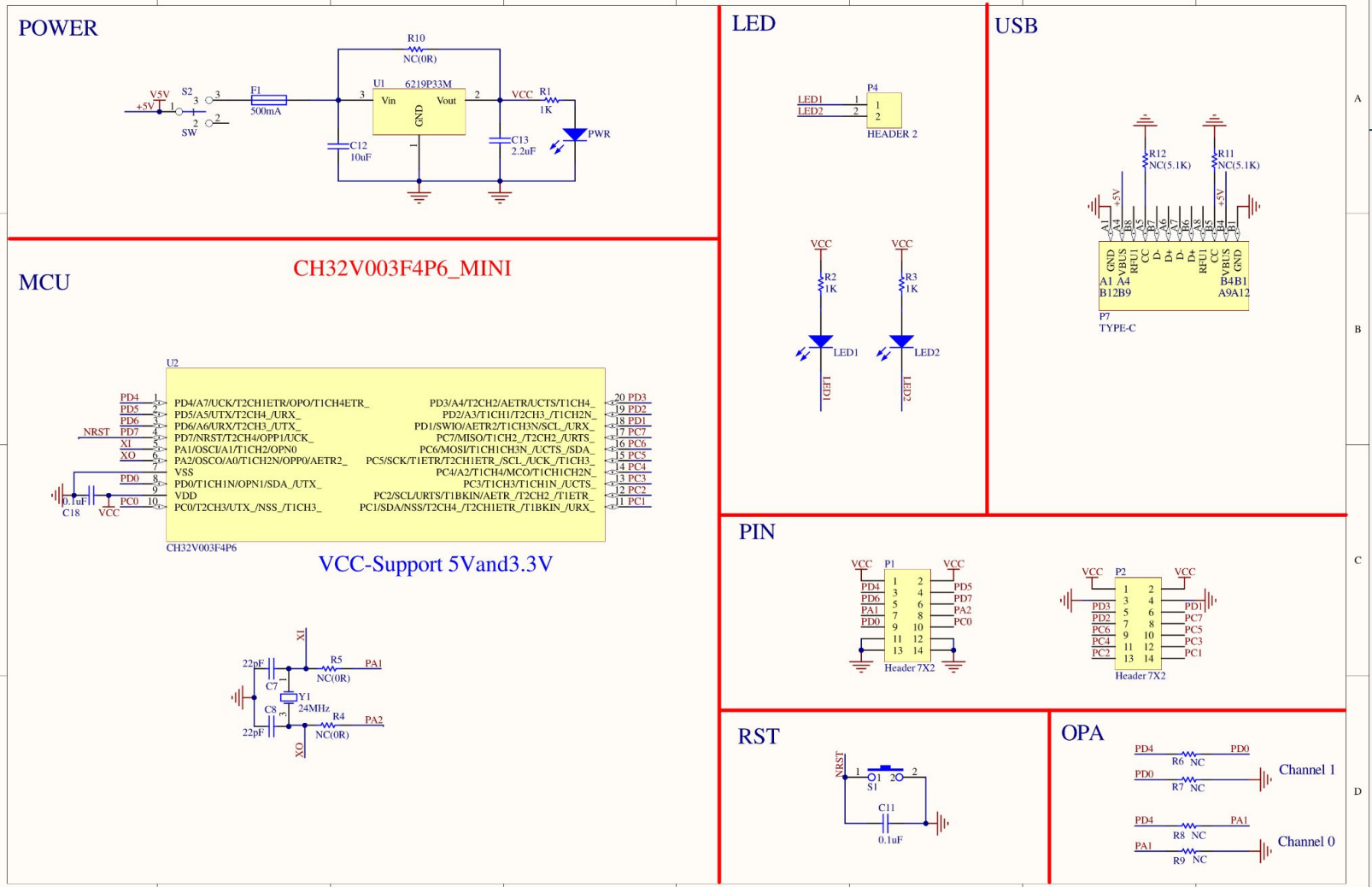
- Implements RV32EmC instruction set
- E: alternative base extension, specifically designed for very low spec processors
- Mostly identical to RV32I base extension, except only specifies 16 general-purpose registers instead of 32
- Smaller register file saves quite a lot of silicon space, reducing cost
- E extension still in development!
- Manufacturers nonetheless already implementing it in hardware
- Spec not officially frozen so compilers still catching up
- m: hardware multiplication (Zmmul) ext.
- C: Supports 16-bit compression instruction
- 2 deep pipeline
- Static branch prediction
- 256 interrupts including exceptions, and Vector Table Free (VTF) interrupts
- 2 levels of Hardware Prologue/Epilogue (HPE)
- Supports Sleep and Deep sleep modes, and support WFI and WFE sleep methods
- Supports half-word and byte operation compression instructions
- 1-wire/2-wire SDI, standard RISC-V debug
- English Microprocessor Manual (32 pages)

CH32V003F4P6-EVT-R0-1v1.FP Board

EVT (Engineering Validation Test) board

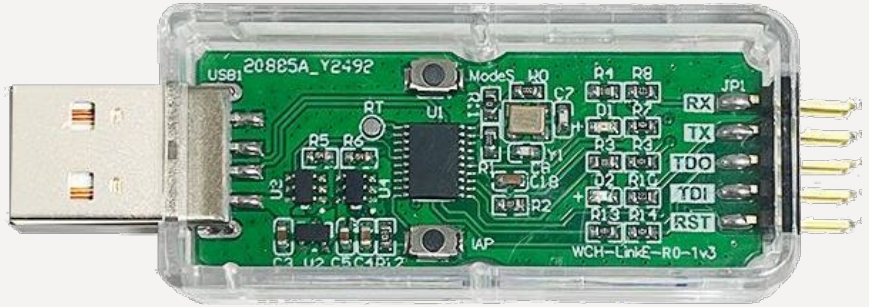
1. Actual MCU: CH32V003F4P6
2. MCU GPIOs: P1 and P2
3. Power switch S2: 5V USB power supply
4. USB-C interface: 5V power supply only
5. Reset button S1 (req. RST_MODE bits of the user select word register as non-11b)
6. LEDs are connected to main chip I/O port via LED row pins (P4)
7. 1-wire DEBUG interface:
   for downloading, simulation debugging, only needs SWDIO PD1 connection
8. Crystal pin PA1 and PA2 assembly option

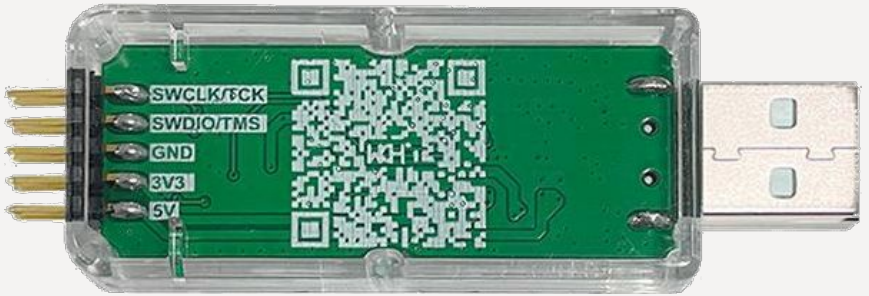



POWER
R10
NC(0R)
V5V
+5V
S2
SW
F1
500mA
U1
6219P33M
Vin
GND
Vout
VCC
R1
1K
PWR
C12
10uF
C13
2.2uF
MCU
CH32V003F4P6_MINI
U2
PD4
PD5
PD6
NRST
PD7
XI
XO
PA1/OSCI/A1/T1CH2/OPN0
PA2/OSCO/A0/T1CH2N/OPP0/AETR2_
VSS
PD0
PD0/T1CH1N/OPN1/SDA_/UTX_
VDD
PC0
PC0/T2CH3/UTX_/NSS_/T1CH3_
PD3/A4/T2CH2/AETR/UCTS/T1CH4_
PD2/A3/T1CH1/T2CH3_/T1CH2N_
PD1/SWIO/AETR2/T1CH3N/SCL_/URX_
PC7/MISO/T1CH2_/T2CH2_/URTS_
PC6/MOSI/T1CH1CH3N_/UCTS_/SDA_
PC5/SCK/T1ETR/T2CH1ETR_/SCL_/UCK_/T1CH3_
PC4/A2/T1CH4/MCO/T1CH1CH2N_
PC3/T1CH3/T1CH1N_/UCTS_
PC2/SCL/URTS/T1BKIN/AETR_/T2CH2_/T1ETR_
PC1/SDA/NSS/T2CH4_/T2CH1ETR_/T1BKIN_/URX_
20 PD3
19 PD2
18 PD1
17 PC7
16 PC6
15 PC5
14 PC4
13 PC3
12 PC2
11 PC1
CH32V003F4P6
0.1uF
C18
VCC
VCC-Support 5Vand3.3V
XI
22pF
C7
R5
NC(0R)
PA1
Y1
24MHz
C8
R4
NC(0R)
PA2
22pF
XO
LED
LED1
LED2
P4
HEADER 2
VCC
R2
1K
R3
1K
LED1
LED2
USB
R12
NC(5.1K)
R11
NC(5.1K)
+5V
GND
VBUS
RFU1
CC
D-
D+
RFU1
CC
VBUS
GND
A1 A4
B12B9
B4B1
A9A12
P7
TYPE-C
PIN
P1
VCC
PD4
PD6
PA1
PD0
PD5
PD7
PA2
PC0
Header 7X2
P2
PD3
PD2
PC6
PC4
PC2
PD1
PC7
PC5
PC3
PC1
Header 7X2
RST
NRST
S1
C11
0.1uF
OPA
PD4
PD0
R6 NC
R7 NC
Channel 1
PA1
R8 NC
R9 NC
Channel 0

WCH-LinkE-R0-1v3.FP Debug Probe

- USB debug probe dongle
- Online debugging and downloading of WCH RISC-V MCUs
- Online debugging and downloading of ARM MCUs with SWD/JTAG interface
- Serial port for easy debugging output
- wlink:
  WCH-Link(RV) command line tool in Rust
- probe-rs:
  The de facto embedded toolkit in Rust
- English User Manual (29 pages)
- CH32V003-LinkE Kit: EVT, USB probe and 5 MCU chips for less than 7 ½ bucks



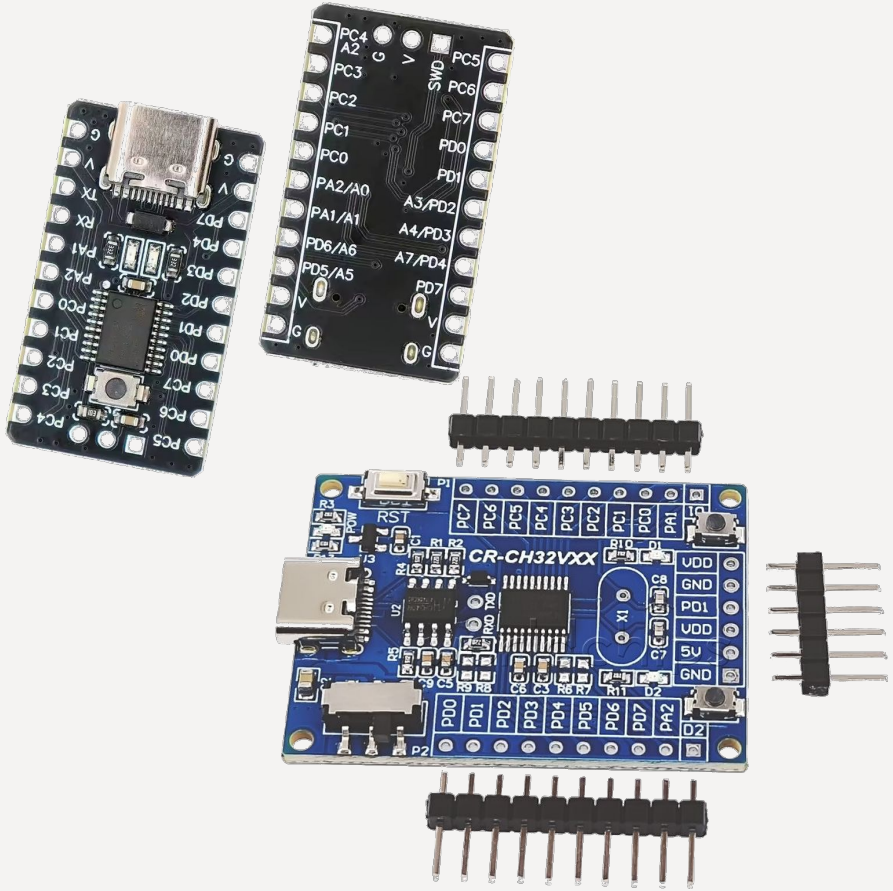


WCH-LinkE-R0-1v3.FP Debug Probe cont.

```
[27236.590351] usb 1-1: new full-speed USB device number 4 using xhci_hcd
[27236.736817] usb 1-1: New USB device found, idVendor=1a86, idProduct=8010, bcdDevice= 2.18
[27236.736831] usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[27236.736837] usb 1-1: Product: WCH-Link
[27236.736842] usb 1-1: Manufacturer: wch.cn
[27236.736847] usb 1-1: SerialNumber: F67E8F067BB0
[27236.793384] cdc_acm 1-1:1.1: ttyACM0: USB ACM device
[27236.793408] usbcore: registered new interface driver cdc_acm
[27236.793409] cdc_acm: USB Abstract Control Model driver for USB modems and ISDN adapters
```

Many More Cheap Boards Available

- TENSTAR Robot TS2160Y-CH32V003 Raspberry Pi Pico/Teensy-style
  - Very simple
  - USB-C just for power
  - 11-pin header pins on each side
- CR-CH32Vxxx also known as MiniTool CH32V003 (GXFA0260-001)
  - More advanced
  - USB-C with serial UART for debug
  - Function and test keys
  - Power switch
  - 10-pin header pins on each side



Embedded Tooling

wlink: WCH-Link(RV) command line tool in Rust

- Flash firmware, support Intel HEX, ELF and raw binary format
- Erase chip
- Halt, resume, reset support
- Read chip info
- Read chip memory(flash)
- Read/write chip register - very handy for debugging
- Code-Protect & Code-Unprotect for supported chips
- Enable or Disable 3.3V, 5V output
- SDI print support, requires 2.10+ firmware
- Serial port watching for a smooth development experience
- Windows native driver support, no need to install libusb manually (requires x86 build)

wlink: WCH-Link(RV) command line tool in Rust cont.

- cargo install --git https://github.com/ch32-rs/wlink

```
cargo install --git https://github.com/ch32-rs/wlink
   Installed package `wlink v0.1.1 (https://github.com/ch32-rs/wlink#e80f286e)` (executable
 `wlink`)
```

- wlink list

```
• [zim@toolbx ~]$ wlink list
<WCH-Link#0 nusb device> ID 1a86:8010(USB-FS 12 Mbps) (RV mode)
```

- wlink status

```
• [zim@toolbx ~]$ wlink status
17:25:33 [INFO] Connected to WCH-Link v2.18(v38) (WCH-LinkE-CH32V305)
17:25:33 [INFO] Attached chip: CH32V003 [CH32V003F4P6] (ChipID: 0x00300510)
17:25:33 [INFO] Chip ESIG: FlashSize(16KB) UID(cd-ab-11-95-2e-bd-0c-fe)
17:25:33 [INFO] Flash protected: false
17:25:33 [INFO] RISC-V ISA(misa): Some("RV32CEX")
17:25:33 [INFO] RISC-V arch(marchid): Some("WCH-V2A")
17:25:33 [WARN] The halt status may be incorrect because detaching might resume the MCU
17:25:33 [INFO] Dmstatus {
...
```

wlink: WCH-Link(RV) command line tool in Rust cont.

- wlink regs

```
● [root@toolbx ch32v003-blinky-rust]# wlink regs
01:31:53 [INFO] Connected to WCH-Link v2.18(v38) (WCH-LinkE-CH32V305)
01:31:53 [INFO] Attached chip: CH32V003 [CH32V003F4P6] (ChipID: 0x00300510)
01:31:53 [INFO] Dump GPRs
dpc(pc):   0x0000097c
x0   zero: 0x00000000
x1     ra: 0x00000154
x2     sp: 0x2000075c
x3     gp: 0x20000800
x4     tp: 0xd0010030
x5     t0: 0x00000000
x6     t1: 0x00000b98
x7     t2: 0x20000034
x8     s0: 0x00004000
x9     s1: 0x00003fce
x10    a0: 0x000000b0
x11    a1: 0x0000009c
x12    a2: 0x00000000
x13    a3: 0x000000ff
```

wlink: WCH-Link(RV) command line tool in Rust cont.

- wlink regs cont.

```
x14    a4: 0x2000001c
x15    a5: 0x0000097c
marchid  : 0xdc68d841
mimpid   : 0xdc688001
mhartid  : 0x00000000
misa     : 0x40800014
mtvec    : 0x00000003
mscratch : 0x2b809c19
mepc     : 0x000009ac
mcause   : 0x00000005
mtval    : 0x00000000
mstatus  : 0x00801888
dcsr     : 0x400008c3
dpc      : 0x0000097c
dscratch0: 0x000000b0
dscratch1: 0x0000009c
gintenr  : 0x00000000
intsyscr : 0x00000003
corecfgr : 0x00000000
```

probe-rs: Embedded programming made easy

- User-friendly and flexible embedded toolkit that just works
- Run programs on your microcontroller with ease of native applications
- Easily print to STDOUT via RTT and defmt encoding when using probe-rs run
- cargo-flash to just flash a target
- cargo-embed to get full RTT terminal to send commands and view multiple channels
- Easy debugging in VSCode
- cargo install probe-rs-tools

```
[zim@toolbx ~]$ cargo install probe-rs-tools
  Installed package `probe-rs-tools v0.31.0` with `probe-rs-tools v0.31.0` (executables
`cargo-embed`, `cargo-flash`, `probe-rs`)
```

Embedded Rust

Rust Embedded Devices Working Group

- Build an ergonomic and composable ecosystem for embedded Rust developers
- Peripheral access crates
  - How to access hardware peripheral registers?
  - Machine readable hardware definitions in System View Description (SVD) format
  - svd2rust: automatic tooling to generate API for peripherals
  - WinChipHead case distributed as part of Eclipse-derived IDE MounRiver Studio
- Embedded Hardware Abstraction Layer (embedded-hal)
  - Abstraction crate
  - Defines a set of traits that describe common peripheral functionality
  - Interfaces for different peripheral types like GPIOs, timers, busses etc.
  - Making sure abstractions are zero-cost
  - Model platform differences reasonably well

WCH CH32V003 with Rust

- WCH CH32V003 embedded-hal available as part of ch32-rs project
- Same project wlink comes from
- embedded-hal requires Rust Nightly
  rustup install nightly
  rustup override set nightly

```
● [zim@toolbx ~]$ rustup install nightly
  nightly-x86_64-unknown-linux-gnu updated - rustc 1.95.0-nightly (a293cc4af 2026-01-30)
 (from rustc 1.95.0-nightly (d940e5684 2026-01-19))
● [zim@toolbx ch32v003-blinky-rust]$ rustup override set nightly
info: override toolchain for '/var/home/zim/Downloads/WinChipHead
 (WCH)/CH32V003/ch32v003-blinky-rust' set to 'nightly-x86_64-unknown-linux-gnu'
```

- Also needs sources of standard libraries in order to build core later on
  rustup component add rust-src

```
● [zim@toolbx ch32v003-blinky-rust]$ rustup component add rust-src
info: downloading component 'rust-src'
info: installing component 'rust-src'
```

Target Specification File

riscv32ec-unknown-none-elf.json

```
{
   "arch": "riscv32",
   "atomic-cas": false,
   "cpu": "generic-rv32",
   "crt-objects-fallback": "false",
   "data-layout": "e-m:e-p:32:32-i64:64-n32-S32",
   "eh-frame-header": false,
   "emit-debug-gdb-scripts": false,
   "features": "+e,+c,+forced-atomics",
   "linker": "rust-lld",
   "linker-flavor": "gnu-lld",
   "llvm-target": "riscv32",
   "llvm-abiname": "ilp32e",
   "max-atomic-width": 32,
   "panic-strategy": "abort",
   "relocation-model": "static",
   "target-pointer-width": "32"
}
```

Cargo Configuration File

.cargo/config.toml

```
[build]
target = "riscv32ec-unknown-none-elf.json"
[target.riscv32ec-unknown-none-elf]
runner = "probe-rs run --chip ch32v003"
rustflags = ["-C", "link-arg=-Tlink.x"]
[unstable]
build-std = ["core"]
```

Dependencies and Profile Settings

Cargo.toml

```
[package]
name = "ch32v003-blinky-rust"
version = "0.1.0"
edition = "2021"
[dependencies]
panic-halt = "1.0.0"
ch32-hal = { git = "https://github.com/ch32-rs/ch32-hal", features = [
        "ch32v003f4u6",
    ] }
qingke-rt = "0.4.0"
qingke = "0.4.0"
embedded-hal = "1.0.0"
[profile.dev]
strip = false
lto = true
opt-level = "s"
```

Main File to Blink an LED

```
#![no_std]
#![no_main]

use hal::delay::Delay;
use hal::gpio::{Level, Output};
use ch32_hal as hal;
use panic_halt as _;

#[qingke_rt::entry]
fn main() -> ! {
    let config = hal::Config::default();

    let peripherals = hal::init(config);

    let mut led = Output::new(peripherals.PD6, Level::Low, Default::default());

    let mut delay = Delay;

    loop {
        led.toggle();
        delay.delay_ms(1000);
    }
}
```

Run to Compile and Upload

- cargo run

```
● [zim@toolbx ch32v003-blinky-rust]$ cargo run
   Compiling compiler_builtins v0.1.160 (/var/home/zim/.rustup/toolchains/nightly-x86_64-
unknown-linux-gnu/lib/rustlib/src/rust/library/compiler-builtins/compiler-builtins)
   Compiling core v0.0.0 (/var/home/zim/.rustup/toolchains/nightly-x86_64-unknown-linux-gnu
/lib/rustlib/src/rust/library/core)
...
    Finished `dev` profile [optimized + debuginfo] target(s) in 9.26s
     Running `probe-rs run --chip ch32v003 target/riscv32ec-unknown-none-elf/debug/
ch32v003-blinky-rust`
      Erasing ✓ 100% [####################]   4.00 KiB @   9.53 KiB/s (took 0s)
  Programming ✓ 100% [####################]   3.06 KiB @   1.70 KiB/s (took 2s)
                                  Finished in 2.33s
Blink!
01:39:13.271: Blink!
```

CODETHINK.CO.UK | OPEN SOURCE

VS Code

VS Code

- Install a few extensions
  - rust-analyzer
  - Debugger for probe-rs
- Given I'm on Fedora Silverblue it uses the Flatpak version of VS Code
  - Problem, can't access host tools from sandbox
  - That's where Dev Containers come in handy, so install that extension as well
  - Use a Toolbox container configured with your Rust environment
  - Configure a few things like dockerPath and dockerSocketPath

settings.json

```
{
    "rust-analyzer.check.command": "clippy",
    "rust-analyzer.workspace.discoverConfig": null,
    "dev.containers.dockerPath": "/var/home/zim/.local/bin/podman-host",
    "dev.containers.dockerSocketPath": "unix:///run/user/1000/podman/podman.sock",
}
```

VS Code cont.

- For dockerPath you can use a wrapper script

```
zim@fedora:~$ cat ~/.local/bin/podman-host
#!/bin/sh
exec flatpak-spawn --host podman "${@}"
```

- For USB debugging probe to work you need access to plugdev
- Find out what user VS Code dev container uses by spawning a sleep 1000
- Outside, in native environment, check what user that is

```
[root@toolbx ch32v003-blinky-rust]# sg plugdev -c 'sleep 1000'

zim@fedora:~$ ps -eo user,group,args | grep sleep
524288   524288   sleep 1
```

- And configure it accordingly in your /etc/group

```
plugdev:x:995:524288,zim
```

VS Code cont.

- Alternatively run your container privileged

registry.fedoraproject.org%2ffedora-toolbox%3a43.json

```
{
	"workspaceFolder": "/var/home/zim/Downloads/WinChipHead
(WCH)/CH32V003/ch32v003-blinky-rust",
	"extensions": [
		"probe-rs.probe-rs-debugger",
		"rust-lang.rust-analyzer"
	],
	"privileged": true,
   "runArgs": [
		"--privileged",
		"--security-opt=label=disable",
	]
}
```

Live Demo

References

- WinChipHead CH32V003
  https://www.wch-ic.com/products/CH32V003.html
- WCH MCU for Rust
  https://github.com/ch32-rs
- probe-rs
  https://probe.rs
- Rust on the CH32V003
  https://noxim.xyz/blog/rust-ch32v003
- Getting Started with CH32V003 Firmware in Rust
  https://albertskog.se/ch32v-in-rust

Thank You.

Codethink LTD

3rd Floor Dale House,
35 Dale Street,
MANCHESTER,
M1 2HF
United Kingdom