WCH QingKe 工具链使用说明

本文面向当基于上游riscv-toolchain重新构建的 `riscv32-wch-elf-*` 工具链，说明如何在工程里启用 **XW**、使用 **HPE 快速中断**属性，以及与 **MounRiver / 通用 RISC-V** 工具链混用时需注意的事项。

---

1. 安装路径与基本命令

安装完成后，将 `$prefix/bin` 加入 `PATH`（`$prefix` 与 configure 时 `--prefix` 一致），例如：

```
export PATH=/opt/riscv-wch/bin:$PATH
riscv32-wch-elf-gcc --version
```

常用命令前缀：`riscv32-wch-elf-`（`gcc`、`g++`、`as`、`ld`、`objdump`、`objcopy`、`size` 等）。

1.1 工具链各模块版本号

本仓库 `./build-wch-toolchain.sh` 默认下载并打补丁的 **GNU / Sourceware 上游发布包版本**如下，与 构建说明.md §2 及脚本开头的 `WCH_*_VERSION` 默认值一致。若构建时用环境变量改写了版本，或使用 `--disable-gdb` 跳过调试器，以你本机实际安装为准。

模块	默认上游版本	说明
GCC	**15.2.0**	编译器驱动、`cc1` / `cc1plus` 等
Binutils	**2.46.0**	`as`、`ld`、`objdump`、`objcopy`、`readelf` 等
Newlib	**4.6.0.20260123**	裸机 C 库（与 GCC 联编安装）
GDB	**17.1**	默认会构建；`--disable-gdb` 时不安装 `riscv32-wch-elf-gdb`
riscv-gnu-toolchain（编排层）	随仓库	目录 `scripts/riscv-gnu-toolchain/`，为上游精简快照，无独立发布版本号

安装后可用下列命令核对已安装工具的报告版本（`riscv32-unknown-elf-*` 前缀时同理替换）：

```
riscv32-wch-elf-gcc --version
riscv32-wch-elf-as --version
riscv32-wch-elf-ld --version
riscv32-wch-elf-gdb --version   # 若已安装 GDB
```

其中 `as` / `ld` 的版本信息对应 **Binutils**。**Newlib** 无单独命令前缀，其源码版本以构建时 `WCH_NEWLIB_VERSION` 为准；需要完整配置信息时可执行 `riscv32-wch-elf-gcc -v`（冗长模式）查看驱动与搜索路径。

---

2. 选择 `-march` 与 `-mabi`

- `-mabi`：与芯片及官方例程一致即可，常见为 `ilp32`（通用 RV32IMAC 等）、`ilp32f`（**QingKe V4F** 等带硬件单精度浮点），或 `ilp32e`（如 CH32V003 等 16 GPR 配置）。

- `-march`：可手写 `rv32imacxw`、`rv32ecxw` 等；更推荐 `-mcpu=wch-qingke-v3b` 等，由**内核型号**自动决定是否含 `xw`（见手册表 1-1）。
- `xw` **与** `zicsr` **/** `zifencei`：本仓库 GCC/Binutils 中 `xw` **扩展自动隐含** `zicsr` **和** `zifencei`，与 WCH 官方行为一致——含 `xw` 的 arch 无需手动添加 `_zicsr_zifencei`；**不含** `xw` 时仍建议在 `march` 中显式写出 `zicsr` / `zifencei`（或使用 `v3` / `v4a` 等无 **XW** 的 profile / `-mcpu`）。
- **QingKe V2–V5**：按**手册或对照表型号**选 `--profile` 或 `-mcpu`（如 `v003` / `v3a` / `v3b` / `v3v` / `v4f` / `v5f` / `v5a`），**XW** 随型号自动含于 `march`。详见 构建说明.md §3、Qingke-RISC-V差异.md **§1.2**。可用 `riscv32-wch-elf-gcc --print-supported-cpus` 查看本机是否含 `wch-qingke-*`。`v003` 与 `-mcpu=wch-qingke-v2a` 同 arch。**XW** 依赖 **C**，勿在 `v2` 上强开 `xw`。含 `f` 的工具链与工程须 `ilp32f` 一致，否则易与 `libnosys` / `libc` **浮点 ABI 冲突**。

`configure` 时的 `--with-arch` / `--with-abi` 只影响默认多库；单项目仍建议 **在 Makefile 里显式写** `CFLAGS += -march=... -mabi=...`，避免与默认不一致。

2.1 Multilib（多库变体）

本工具链默认构建 **22 个 multilib 变体**（与 WCH 官方 GCC 15 一致），覆盖 QingKe V003～V5 全系列。**关键行为**：

- 指定 **不含** `xw` 的 `-march`（如 `rv32imac`）时，链接器自动选择 **不含 XW 指令的库**（如 `rv32imac_zaamo_zalrsc/ilp32/`），确保 `memset`、`memcpy` 等库函数不含 16-bit XW 编码。
- 指定 **含** `xw` 的 `-march`（如 `rv32imacxw`）时，链接器选择 **含 XW 优化的库**（如 `rv32imac_zaamo_zalrsc_xw/ilp32/`），获得 XW 压缩带来的代码密度提升。

可通过以下命令查看当前 toolchain 的 multilib 变体列表和选择：

```
riscv32-wch-elf-gcc -print-multi-lib
riscv32-wch-elf-gcc -march=rv32imac -mabi=ilp32 -print-multi-directory
```

---

3. 各 QingKe 内核推荐 GCC 优化参数

不同 QingKe 内核在流水线深度、发射宽度、Flash/RAM 容量等方面差异显著，优化策略也应有所不同。下表给出**推荐的基础编译参数**，用户可根据实际工程需求调整。

3.1 通用参数说明

参数	说明
`-mcpu=wch-qingke-*`	自动展开对应的 `-march` + `-mtune`，**推荐始终使用**
`-Os`	优化代码体积（MCU 首选，Flash 有限）
`-O2`	优化运行速度（适合 Flash/RAM 充裕的场景）
`-mabi=ilp32` / `ilp32f` / `ilp32e`	与芯片 GPR 数及 FPU 匹配（见 §2）
`-msave-restore`	使用库函数辅助保存/恢复寄存器，缩小 prologue/epilogue 体积（牺牲少量性能换体积）
`-mshorten-memrefs`	默认开启；将大偏移地址重写为基址+小偏移，提升 RVC/XW 压缩率
`-msmall-data-limit=N`	GP 相对寻址阈值（默认 8 字节）；增大可减少全局变量访问指令数，但 `.sdata` 段不可超过 ±2KiB
`-mstrict-align`	禁止非对齐访存（QingKe 硬件不支持非对齐访问，建议始终开启；`-mcpu=wch-qingke-*` 已隐含）
`-mrelax`	默认开启；允许链接器松弛（将 `lui+addi` 缩减为 `c.li` 等），减少代码体积
`-ffunction-sections -fdata-sections`	每个函数/数据独立 section，配合链接器 `--gc-sections` 删除未引用代码
`-Wl,--gc-sections`	链接时删除未引用的 section（与上条配合使用）
`-flto`	链接时优化（跨文件内联、死代码消除；增加编译时间，但效果显著）

3.2 按内核型号推荐

QingKe V2A（CH32V003 等）— RV32EC + XW，16 GPR，2 级流水线

资源极度受限（通常 16KB Flash / 2KB RAM），**体积优化为第一优先级**。

```
CFLAGS += -mcpu=wch-qingke-v2a -mabi=ilp32e \
          -Os -msave-restore \
          -ffunction-sections -fdata-sections -fno-common
LDFLAGS += -Wl,--gc-sections -nostartfiles
```

- `-Os` + `-msave-restore`：对仅 2KB RAM 的 V003 尤其重要
- `-msmall-data-limit=8`（默认）通常够用；若全局变量密集可适当增大
- 避免 `-O2` / `-O3`：展开循环和内联会显著增大 Flash 占用

QingKe V2C（CH32V002/004/005/006/007）— RV32EC + Zmmul + XW，16 GPR

与 V2A 类似但有硬件乘法（无除法），资源仍受限。

```
CFLAGS += -mcpu=wch-qingke-v2c -mabi=ilp32e \
          -Os -msave-restore \
          -ffunction-sections -fdata-sections -fno-common
LDFLAGS += -Wl,--gc-sections -nostartfiles
```

QingKe V3A（CH32V103、CH32L103、CH571/573 等）— RV32IMAC，无 XW，3 级流水线

32 GPR，较充裕的 Flash（64–128KB），无 XW。蓝牙系列 CH571/CH573（BLE 4.2）也使用 V3A 内核。

```
CFLAGS += -mcpu=wch-qingke-v3a -mabi=ilp32 \
          -Os \
          -ffunction-sections -fdata-sections
LDFLAGS += -Wl,--gc-sections -nostartfiles
```

- 无 XW 时 `-mshorten-memrefs` 仍有效（改善标准 RVC 压缩率）
- 若 Flash 充裕且对性能敏感（如协议栈），可尝试 `-O2`

QingKe V3B/V3C（CH32X033/X035、CH591/592 等）— RV32IMAC + XW [+ B]，3 级流水线

```
# V3B
CFLAGS += -mcpu=wch-qingke-v3b -mabi=ilp32 \
          -Os \
          -ffunction-sections -fdata-sections
# V3C（额外含 B 扩展：zba/zbb/zbc/zbs）
CFLAGS += -mcpu=wch-qingke-v3c -mabi=ilp32 \
          -Os \
          -ffunction-sections -fdata-sections
LDFLAGS += -Wl,--gc-sections -nostartfiles
```

- V3C 含 B 扩展，GCC 会自动使用 `sh1add` / `clz` / `bext` 等替代多指令序列
- XW 在 `-Os` 下自动发挥最优效果

QingKe V3F — RV32IMAFC + XW + B，单精度浮点，3 级流水线

```
CFLAGS += -mcpu=wch-qingke-v3f -mabi=ilp32f \
          -Os \
          -ffunction-sections -fdata-sections
LDFLAGS += -Wl,--gc-sections -nostartfiles
```

- **`-mabi=ilp32f`** 必须与库一致，否则浮点 ABI 不匹配导致链接错误或运行时异常
- 若大量浮点运算，可尝试 `-O2 -ffast-math`（注意 `-ffast-math` 会放宽 IEEE 754 合规性）

QingKe V3V — RV32IMAC + XW + B + Zve64x + Zvbb，向量扩展

```
CFLAGS += -mcpu=wch-qingke-v3v -mabi=ilp32 \
          -Os \
          -ffunction-sections -fdata-sections
LDFLAGS += -Wl,--gc-sections -nostartfiles
```

- GCC 可自动向量化（`-ftree-vectorize`，`-O2` 以上默认开启）
- 手动向量化可使用 RISC-V Vector Intrinsics（`<riscv_vector.h>`）

QingKe V4A（CH581/582/583 等）— RV32IMAC，无 XW，3 级流水线

蓝牙系列 CH581/CH582/CH583（BLE 5.x）使用 V4A 内核，与 V3A 编译参数相同。

```
CFLAGS += -mcpu=wch-qingke-v4a -mabi=ilp32 \
          -Os \
          -ffunction-sections -fdata-sections
LDFLAGS += -Wl,--gc-sections -nostartfiles
```

QingKe V4B/V4C/V4J（CH32V203、CH32V208、CH584/585、CH643/645 等）— RV32IMAC + XW，3 级流水线

```
CFLAGS += -mcpu=wch-qingke-v4b -mabi=ilp32 \
          -Os \
          -ffunction-sections -fdata-sections
LDFLAGS += -Wl,--gc-sections -nostartfiles
```

- V4B（CH32V203）/ V4C（CH32V208、CH584/585、CH643/645）/ V4J（CH569/565）的 `-march` 展开一致，差异在微架构（I-Cache、MPU 区数等）
- V4J 有 I-Cache，循环密集型代码可受益于 `-O2 -funroll-loops`
- CH584/CH585（BLE 5.4）使用 V4C 内核，含 XW 和 mcpy 支持

QingKe V4F（CH32V307、CH32V317 等）— RV32IMAFC + XW，单精度浮点，3 级流水线

Flash 通常 256KB，RAM 较充裕（64KB+），是 QingKe 系列中**最常用的高性能型号之一**。

```
CFLAGS += -mcpu=wch-qingke-v4f -mabi=ilp32f \
          -O2 \
          -ffunction-sections -fdata-sections
LDFLAGS += -Wl,--gc-sections -nostartfiles
```

- Flash 充裕时推荐 `-O2`（平衡速度与体积）；体积敏感场景仍用 `-Os`
- **`-mabi=ilp32f`**：启用硬件浮点传参，避免不必要的 softfloat 转换
- HPE 中断处理函数使用 `__attribute__((interrupt("WCH-Interrupt-fast")))`，详见 §6
- 对性能极端敏感的 USB/以太网处理：可局部使用 `-O3` 或 `__attribute__((optimize("O3")))`

QingKe V5F / V5A（CH32H417 等）— RV32IMAFC + XW + B，双发射，7～9 级流水线

V5 系列为 QingKe 中**性能最强的内核**，双发射 + 深流水线，I/D-Cache。CH32H417 为目前已知使用 V5F 内核的典型型号。

```
CFLAGS += -mcpu=wch-qingke-v5f -mabi=ilp32f \
          -O2 \
          -ffunction-sections -fdata-sections
LDFLAGS += -Wl,--gc-sections -nostartfiles
```

- **推荐 `-O2`**：双发射核可充分利用指令级并行，`-O2` 的调度和内联比 `-Os` 更有利
- 含 B 扩展（`zba` / `zbb` / `zbc` / `zbs`），GCC 自动使用位操作指令
- I/D-Cache 使循环展开（`-funroll-loops`）收益更大
- 大型应用可考虑 `-flto` 做跨文件优化
- 性能调优时可用 `-O3`（激进内联与循环优化），但需关注 Flash 增量

3.3 优化策略速查表

内核	`-mcpu`	`-mabi`	推荐优化	侧重	典型芯片
V2A	`wch-qingke-v2a`	`ilp32e`	`-Os -msave-restore`	极致体积	CH32V003
V2C	`wch-qingke-v2c`	`ilp32e`	`-Os -msave-restore`	极致体积	CH32V002/004/005/006/007
V3A	`wch-qingke-v3a`	`ilp32`	`-Os`	体积优先	CH32V103, CH32L103, CH571/573
V3B	`wch-qingke-v3b`	`ilp32`	`-Os`	体积+XW	CH32X033/X035, CH591/592
V3C	`wch-qingke-v3c`	`ilp32`	`-Os`	体积+XW+B	—
V3F	`wch-qingke-v3f`	`ilp32f`	`-Os`	体积+FPU	—
V3V	`wch-qingke-v3v`	`ilp32`	`-Os` / `-O2`	体积/向量	—
V4A	`wch-qingke-v4a`	`ilp32`	`-Os`	体积优先	CH581/582/583
V4B/C/J	`wch-qingke-v4b`	`ilp32`	`-Os`	体积+XW	CH32V203, CH32V208, CH584/585, CH643/645
V4F	`wch-qingke-v4f`	`ilp32f`	`-O2` / `-Os`	速度/体积	CH32V303/305/307, CH32V317
V5F/V5A	`wch-qingke-v5f`	`ilp32f`	`-O2`	速度优先	CH32H417

3.4 注意事项

- **`-mstrict-align`**：QingKe 硬件不支持非对齐访存。使用 `-mcpu=wch-qingke-*` 时已通过 `rocket_tune_info` 设置 `slow_unaligned_access=true`，GCC 不会生成非对齐访问。若手写 `-march`/`-mtune`，建议显式加 `-mstrict-align`。
- **`-msave-restore`**：在 V2（16 GPR）系列效果最显著；V4F/V5F 上因寄存器充裕且性能敏感，通常不建议开启。
- **`-flto`**：链接时优化对所有内核均有效，但会增加编译时间和内存消耗。小型工程（< 50 个源文件）推荐使用。
- **`-ffast-math`**：仅在浮点精度要求不高时使用（如音频/图形）；涉及金融计算或 ADC 校准时应避免。
- **`-msmall-data-limit`**：增大该值可减少全局变量的访问开销，但全局 `.sdata` + `.sbss` 总大小不可超过 GP 可达范围（±2KiB）。V2 系列 RAM 极小，建议保持默认值 8。
- **HPE 中断**：HPE ISR 内部的优化级别与全局一致。若个别 ISR 需要不同优化，可使用函数属性 `__attribute__((optimize("Os")))` 单独控制。

---

4. XW 扩展：作用与编译选项

XW 在汇编层提供 16 位编码的 **字节/半字** 访存别名（如 `wchqk.c.lbu` 等，由 GAS 根据操作数解析生成）。本仓库中的 GCC 还会在 `-Os` 等优化下配合 `shorten_memrefs`，尽量把 **QI/HI** 访存整理成更易触发上述别名的地址形式，并略提高压缩 GPR 分配优先级，以减小代码体积。

启用 XW 时请务必保证 `-march` **含** `xw`，否则不会定义 `__riscv_xw`，也不会按 XW 目标做上述优化。

在 C/C++ 中可用预处理器判断是否启用 XW（宏值为版本号编码的整数，与 GCC 对其它扩展宏一致）：

```
#ifdef __riscv_xw
/* 已启用 XW */
#endif
```

---

5. `xw` 与其他扩展的兼容性

本仓库与 **WCH 官方 GCC 15** 行为对齐，**不再**禁止以下组合：

组合	GCC 行为	说明
`xw` + `zcb`	**允许**（不报错）	编码空间有重叠但 GCC 不强制互斥；LLVM `xwchc` 仍禁止此组合
`xw` + `d`	**允许**（不报错）	实际青稞 MCU 通常不实现 D 扩展，链接含 D 代码会在运行时非法指令

从 MRS 工程迁移时，`march` **字符串**中的 `zcb` 或 `d` 不再与 `xw` 冲突。但请确认芯片实际支持的扩展，避免链接不兼容的代码。

---

6. HPE: `interrupt("WCH-Interrupt-fast")`

硬件压栈扩展（HPE） 下，部分寄存器由硬件在中断入口保存，ISR 软件序言/尾声与 `interrupt("machine")` 等不同。

本工具链支持 GCC 属性：

```
void __attribute__((interrupt("WCH-Interrupt-fast")))
irq_handler(void)
{
    /* 用户代码 */
}
```

要点：

- **所有退出路径均使用** `mret`：无论 ISR 有多少个 `return` 分支（包括 GCC shrink-wrapping 优化产生的提前退出），编译结果均以 `mret` 返回，不会出现 `ret`。这保证了中断上下文通过 `mepc` 正确恢复，而非使用可能无效的 `ra` 值。可用 `objdump -d` 核对所有退出路径。
- 空 ISR 或仅使用 **硬件已保存的 GPR** 时，prologue/epilogue 更精简——编译器不会为 HPE 已保存的 caller-saved 整型寄存器生成多余的 `sw`/`lw`。
- 编译器将 **仅对部分通用整数寄存器** 按 HPE 约定视为“已由硬件保存”；**不要假定浮点寄存器**由硬件保存。若 ISR 使用 **FPU**，仍需按 ABI 由编译器生成完整的浮点保存/恢复（具体以生成汇编为准）。
- 可用宏判断 **本编译器是否支持该属性字符串**（与是否带 XW 无关）：

```
#ifdef __riscv_wch_interrupt_fast
void __attribute__((interrupt("WCH-Interrupt-fast"))) isr(void);
#endif
```

6.1 多分支 ISR 验证

含多个早退出分支的 HPE ISR（如 USB/网络等复杂中断处理），建议编译后确认无 `ret`：

```
riscv32-wch-elf-gcc -march=rv32imafcxw -mabi=ilp32f -O2 -c isr.c
riscv32-wch-elf-objdump -d isr.o | grep -E '\bret$'
# 预期：无匹配（所有返回均为 mret）
```

---

7. 汇编与反汇编

- 手写汇编时，使用与本工程一致的 **`-march=...xw`**，以便 GAS 识别 XW 助记与别名。
- 查看 XW 指令时，对 `.o` 或 `.elf` 执行：

```
riscv32-wch-elf-objdump -d your.elf
```

与 WCH 官方 objdump 的差异：

特性	本仓库 objdump	WCH 官方 objdump
默认解码 XW	**自动根据 ELF `xw2p0` 属性解码**	需手动加 `-M xw`
不加 `-M xw` 时	正确显示 `lbu` / `sb` / `lhu` / `sh`	显示为 `.insn 2, 0xNNNN`

本仓库的 objdump 在 `riscv_dis_parse_subset` 中检测 ELF 的 `Tag_RISCV_arch` 属性，若含 `xw2p0` 则自动启用 XW 解码，无需 `-M xw`。也可显式使用 `-M xw` 强制启用（例如对不含 `xw2p0` 属性的旧 ELF 文件）。

---

8. 与 MounRiver（`riscv-none-elf`）工程对齐

项目	本仓库产物	常见 MRS 产物
工具前缀	`riscv32-wch-elf-`（本仓库推荐）	`riscv-none-elf-`
对齐方式	**`-march` / `-mabi` 与链接脚本、启动文件一致即可**	同上

将 Makefile 中的 **`CC`** 等改为 `riscv32-wch-elf-gcc` 后，重点检查：**`ARCH_FLAGS`**、**`LDFLAGS`**、**`specs`** / **`nano.specs`** 是否与芯片匹配；triplet 名称不同 **不代表** ISA 不同。