



RISCV32 GCC 编译器

## 切换毕昇编译器迁移指南

文档版本 01

发布日期 2024-12-20

版权所有 © 海思技术有限公司2024。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

## 商标声明



**HISILICON**、海思和其他海思商标均为海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

## 注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

## 海思技术有限公司

地址：上海市青浦区虹桥港路2号101室 邮编：201721

网址：<https://www.hisilicon.com/cn/>

客户服务邮箱：[support@hisilicon.com](mailto:support@hisilicon.com)



# 前言

## 概述

本文档用于指导工程代码从RISCV32 GCC编译器切换到毕昇编译器进行开发。本文主要介绍RISCV32 GCC编译器和毕昇编译器的差异和代码迁移方法。






## 读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

## 符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 <b>危险</b>	表示如不避免则将会导致死亡或严重伤害的具有高等级风险的危害。
 <b>警告</b>	表示如不避免则可能导致死亡或严重伤害的具有中等级风险的危害。
 <b>注意</b>	表示如不避免则可能导致轻微或中度伤害的具有低等级风险的危害。
 <b>须知</b>	用于传递设备或环境安全警示信息。如不避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 <b>说明</b>	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。



# 修订记录

修订日期	版本	修订说明
2024-07-05	00B01	第1次临时版本发布。
2024-12-20	01	第1次正式版本发布。



# 目录

前言.....	i
1 概述.....	1
2 编译器.....	3
2.1 配置默认编译选项.....	3
2.2 编译选项对比.....	3
2.3 codesize 选项替换.....	6
3 汇编器以及链接器.....	8
4 编译问题.....	9
4.1 工程配置问题.....	9
4.1.1 使用 CMake 构建，编译器提示选项不识别.....	9
4.2 编译告警.....	9
4.2.1 Wreturn-type，返回值不正确告警.....	9
4.2.2 Warray-bounds，数组访问越界告警.....	10
4.2.3 Wunused-command-line-argument，选项未被使用告警.....	10
4.2.4 Wunused-variable，变量未被使用告警.....	10
4.2.5 Wunused-function，函数未被使用告警.....	10
4.2.6 Wunused-parameter，函数参数被使用告警.....	10
4.2.7 Wincompatible-pointer-types，指针类型不匹配告警.....	10
4.2.8 Wenum-conversion，枚举类型不匹配告警.....	10
4.2.9 Wstrict-prototypes，无参数函数没有使用 void 标识无参数告警.....	11
4.2.10 Wtypedef-redefinition，类型重复定义告警.....	11
4.2.11 Wsometimes-uninitialized，可能存在情况导致变量未初始化.....	11
4.2.12 Wunused-but-set-variable，变量被赋值但未使用.....	11
4.2.13 Wself-assign，赋值语句中将变量赋值给自身告警.....	11
4.2.14 Wmissing-field-initializers，结构体初始化不完整.....	11
4.2.15 Wtautological-pointer-compare，指针比较结果为固定值.....	12
4.2.16 Wparentheses-equality，判断条件有两层括号.....	12
4.2.17 Wtautological-constant-out-of-range-compare，与常量比较结果为固定值.....	12
4.2.18 Wlogical-not-parentheses，逻辑非的作用范围可能存在问题.....	12
4.2.19 Wmisleading-indentation，代码缩进不规范.....	12
4.2.20 Wuninitialized，变量未初始化.....	12
4.2.21 Wnull-pointer-subtraction，使用空指针进行减法运算.....	13



4.2.22 Wpointer-sign, 指针的数据类型不匹配.....	13
<b>5 运行问题.....</b>	<b>14</b>
5.1 链接脚本未包含新增段引起运行异常.....	14



## 表格目录

表 1-1 编译器常用编译工具命令迁移说明.....	1
表 1-2 运行时库迁移说明.....	2
表 1-3 支持指令集与浮点类型说明.....	2
表 2-1 常见编译选项.....	4
表 2-2 codesize 选项列表.....	7



# 1 概述

本文档主要是描述C/C++代码从RISCV32 GCC编译器切换到毕昇编译器进行开发。RISCV32 GCC编译器与毕昇LLVM编译器的差异和迁移说明如表1-1、表1-2和表1-3所示。

表 1-1 编译器常用编译工具命令迁移说明

工具	RISCV32 GCC编译器	毕昇LLVM编译器	迁移说明
Compiler	riscv32-linux-musl-gcc for C riscv32-linux-musl-g++ for C++ 基于开源软件GCC-7.3.0构建，支持标准C89/C90/C++03/C++11/C++14	clang for C clang++ for C++ 基于开源软件LLVM-15.0.4构建，支持标准到C17/C++17	将riscv32-linux-musl-gcc命令修改为clang 将riscv32-linux-musl-g++命令修改为clang++
Assembler	riscv32-linux-musl-as	clang内置汇编器 汇编语言对应RISCV指令集要求，汇编器的伪指令满足GNU风格，具体差异可见下文描述	将riscv32-linux-musl-as命令修改为clang
Linker	riscv32-linux-musl-ld	ld.lld 链接脚本是满足GNU Linker script定义的语法要求。	将riscv32-linux-musl-ld命令修改为ld.lld
Archiver	riscv32-linux-musl-ar	llvm-ar	将riscv32-linux-musl-ar命令修改为llvm-ar





工具	RISC-V32 GCC编译器	毕昇LLVM编译器	迁移说明
Image Conversion Utility	riscv32-linux-musl-objcopy	llvm-objcopy	将riscv32-linux-musl-objcopy命令修改为llvm-objcopy

表 1-2 运行时库迁移说明

描述	RISC-V32 GCC编译器	毕昇编译器	迁移说明
运行时库	libgcc.a	libclang_rt.builtins-riscv32.a	替换链接的库，例如：-lgcc修改为-lclang_rt.builtins-riscv32
异常处理库	libgcc_eh.a	libunwind.a	替换链接的库，例如：-lgcc_eh修改为-lunwind
C++ 标准库	libstdc++.a	libc++.a	替换链接的库，例如：-lstdc++修改为-lc++
C++ ABI 库	libsupc++.a	libc++abi.a	替换链接的库，例如：-lsupc++修改为-lc++abi

表 1-3 支持指令集与浮点类型说明

描述	RISC-V32 GCC编译器	毕昇编译器
131核软浮点	选择cc_riscv32_musl编译器，指定-march=rv32imc	指定-march=rv32imc_xhimideer，链接riscv32-elf/lib/rv32imc_ilp32路径下的库
131核硬浮点	选择cc_riscv32_musl_fp编译器，指定-march=rv32imfc	指定-march=rv32imfc_xhimideer，链接riscv32-elf/lib/rv32imfc_ilp32f路径下的库
170核软浮点	NA	指定-march=rv32imc_xlinxma_xlinxmb_xlinxmc_xlinxmd，链接riscv32-elf/lib/rv32imc_xlinxma_xlinxmb_xlinxmc_xlinxmd_ilp32路径下的库
170核硬浮点	NA	指定-march=rv32imfc_xlinxma_xlinxmb_xlinxmc_xlinxmd，链接riscv32-elf/lib/rv32imfc_xlinxma_xlinxmb_xlinxmc_xlinxmd_ilp32f路径下的库



# 2 编译器

## 2.1 配置默认编译选项

参考<https://clang.llvm.org/docs/UsersManual.html#configuration-files>

1. 在编译命令所在目录linux-llvm-binary-release-musl/bin, 按照<triple>-<driver>规则创建软链接。

比如riscv32 创建软链接: ln -s clang riscv32-clang 或者 ln -s clang riscv32-none-eabi-clang。

2. 在编译命令所在目录linux-llvm-binary-release-musl/bin创建编译配置文件。

编译配置文件和创建的编译命令软链接名称一致, 加上.cfg后缀。

C编译: 比如riscv32-clang.cfg或riscv32-none-eabi-clang.cfg。

C++编译: 比如riscv32-clang++.cfg或riscv32-none-eabi-clang++.cfg。

3. 默认编译选项配置文件内容。

```
# Several options on line
-O2
# other config files may be included
@linux.options
```

## 2.2 编译选项对比

RISCv32 GCC编译器仅支持Himideer指令架构, 毕昇编译器除了支持Himideer指令架构外, 还支持Linux170指令架构, 如果不指定march, 毕昇LLVM默认使用Linux170指令架构。



表 2-1 常见编译选项

RISCV32 GCC编译器	毕昇编译器
<b>-O0/-O1/-O2/-O3/-Os</b> 设置编译优化等级	<b>-O0/-O1/-O2/-O3/-Os/-Oz</b> 除了支持O0~O3优化选项外，还支持Oz选项，专门针对CodeSize进行优化，但性能不如O2/O3
<b>-march=rv32i[mfdc]</b> 'i' means integer, 'f' means float, 'd' means double. 'c' means compress	<b>-march=</b> 指定指令架构，默认使用linux170指令架构。默认为-march=rv32imc_xlinuxma_xlinuxmb_xlinuxmc_xlinuxmd.如果要使用Himeedr指令架构，则需要时设置成march=rv32imc_xhimideer
<b>-mabi</b> It will automatic choose according march, also can use -mabi=ilp32/ilp32f/ilp32d set.	<b>-mabi</b> It will automatic choose according march, also can use -mabi=ilp32/ilp32f/ilp32d set.
<b>--enjal16</b> Optimization for code size, the original function call that is larger than +/-1M will need to use auipc + jalr two insn. Using the link option, we can combine the auipc + jalr of the function call to jal16/j16. If you are using ld for link, please add option -enjal16. If you are using clang for link, please add option -WL,--enjal16	<b>--enjal16</b> Optimization for code size, the original function call that is larger than +/-1M will need to use auipc + jalr two insn. Using the link option, we can combine the auipc + jalr of the function call to jal16/j16. If you are using ld for link, please add option -enjal16. If you are using clang for link, please add option -WL,--enjal16
<b>-enable-c-lbu-sb</b> Optimization for assembler, which is disabled by default. If enable this optimization, assembler will use compressed lbu & sb to replace lbu & sb.	<b>-enable-c-lbu-sb</b> Optimization for assembler, which is disabled by default. If enable this optimization, assembler will use compressed lbu & sb to replace lbu & sb. You should use the -Wa,-enable-c-lbu-sb option when you are assembly with clang and use the -enable-c-lbu-sb when you are assembly with as
<b>-enable-c-lhu-sh</b> Optimization for assembler, which is disabled by default. If enable this optimization, assembler will use compressed lhu & sh to replace lhu & sh. You should use the -Wa,-enable-c-lbu-sb option when you are assembly with gcc and use the -enable-c-lbu-sb when you are assembly with as.	<b>-enable-c-lhu-sh</b> Optimization for assembler, which is disabled by default. If enable this optimization, assembler will use compressed lhu & sh to replace lhu & sh. You should use the -Wa,-enable-c-lhu-sh option when you are assembly with clang and use the -enable-c-lhu-sh when you are assembly with as



RISCV32 GCC编译器	毕昇编译器
<b>-fimm-compare</b> Optimization for code size, which can combine the two instructions for no zero immediate compare (li, bxx) to one instruction (bxxi). The combine is only happened with option above -O1.	<b>-mllvm -imm-compare</b> Optimization for code size, which can combine the two instructions for no zero immediate compare (li, bxx) to one instruction (bxxi).
<b>-fmerge-immsfh</b> Optimization for code size, which can combine the immediate shift to one instructions. The combine is only happened with option above -O1.	<b>-mllvm -merge-immsfh</b> Optimization for code size, which can combine the immediate shift to one instructions.
<b>-femit-muliadd</b> Optimization for code size, which can combine the mul li add tree instruction to one instructions.	<b>-mllvm -merge-immsfh</b> Optimization for code size, which can combine the immediate shift to one instructions.
<b>-femit-muliadd</b> Optimization for code size, which can combine the mul li add tree instruction to one instructions.	<b>-mllvm -emit-muliadd</b> Optimization for code size, which can combine the mul li add tree instruction to one instructions.
<b>-femit-uxtb-uxth</b> Optimization for code size, optimization the unsigned extend byte and unsigned extend half word to uxtb, uxth(16 byte). The combine is only happened with option above -O1.	<b>-mllvm -emit-uxtb-uxth</b> Optimization for code size, optimization the unsigned extend byte and unsigned extend half word to uxtb, uxth(16 byte).
<b>-mpush-pop</b> Emit c.push/c.pop/c.popret when handling epilogue and prologue for each function.  Attention: Please do not use the -msave-restore option at the same time.	<b>-mllvm -himideer-push-pop</b> Emit push/pop/popret when handling epilogue and prologue for each function.
<b>-fldm-stm-optimize</b> Enable the optimization of replacing consecutive WORD load/store with ldmia/stmia, which is disabled by default.	<b>-mllvm -fldm-stm-optimize</b> Enable himideer ldmia/stmia instructions



RISCV32 GCC编译器	毕昇编译器
<p><b>-femit-lli</b></p> <p>Use the 48 bit l.li instruction to replace the 64 bit instruction lui + addi for 32 bit long immediate load.This optimization is combined with insn combine, please use optimization option higher than O1 to enable it.</p>	<p><b>-mllvm -enable-lli</b></p> <p>Use the 48 bit l.li instruction to replace the 64 bit instruction lui + addi for 32 bit long immediate load.</p> <p>This optimization is combined with insn combine.</p>

## 2.3 codesize 选项替换

RISCV32 GCC编译器与毕昇编译器均有自定义codesize选项，具体选项说明可查看《RISCV32 GCC工具链使用指南》与《BiSheng Mobile MCU 201.2.0 Toolchain User Manual》。



表 2-2 codesize 选项列表

RISCV32 GCC编译器	毕昇编译器
-madjust-regorder -madjust-const-cost -freorder-commu-args -fimm-compare-expand -frmv-str-zero -mfp-const-opt -mswitch-jump-table -frtl-sequence-abstract -frtl-hoist-sink -fsafe-alias-multipointer -finline-optimize-size -fmuliadd-expand -mlli-expand -WL,--cjal-relax -Wa,-mcjal-expand -foptimize-reg-alloc -mstack-protector-abstract -fsplit-multi-zero-assignments -floop-optimize-size -WL,--dslf -Wa,-mlli-relax -WL,--lli-relax -mpattern-abstract -foptimize-pro-and-epilogue -WL,--jal-transfer	-mllvm --jump-is-expensive=false -mllvm --min-jump-table-entries=1000000 -mllvm --enable-lli=true -mllvm --enable-oz-inline-threshold=true -mllvm --allow-unalign-ldst=true -mllvm --enable-machine-outliner=always -mllvm -machine-outliner-funcbytes=4 -mllvm -enable-ldst-multiple-opt=true -mllvm -enable-rodata-sections=true -mllvm -enable-sequence-abstract=true -mllvm -enable-branch-imm-no-zero=true -mllvm --riscv-enable-copyelim=true -WL,--jal-relax -WL,--jal-transfer



# 3 汇编器以及链接器

汇编器、链接器与HCCRISCV32 GCC编译器工具链一致，使用的binutils工具。默认支持binutils工具集。除此之外毕昇编译器还支持LLVM内置链接器（ld.lld），与binutils相比，它的运行速度更快，性能更好。ld.lld链接器使用说明详情可参考《BiSheng Mobile MCU 201.2.0 Toolchain User Manual》。

使用clang编译链接时增加“-lld”选项，毕昇编译器将从ld链接器切换到ld.lld链接器，编译器默认链接库的路径也将从riscv32-elf/lib切换到riscv32-elf/lib-lld。但有以下几点需要注意：

- 1.当前Windows版本工具链暂不支持ld.lld链接器，后续版本将默认使用LLD链接器。
- 2.不要使用ld.lld链接binutils编译的文件和库，反之亦然，这是因为两个工具在链接阶段使用的重定位类型不兼容。
- 3.binutils可以同时支持“-Wl,--gc-section”和“-Wl,--gc-sections”这两种写法，而ld.lld只支持带s的版本，即“-Wl,--gc-sections”，这也是gcc官方文档中的标准写法。
- 4.如果需要生成动态库，一定要加上选项“-Wl,allow-shlib-undefined”，而binutils默认启用了这个选项。
- 5.从文件获取符号以及地址时，binutils使用“--just-symbols=filename”选项，而ld.lld使用“-R filename”。
- 6.ld.lld链接脚本中冒号前后要保留空格，即不要写成“os:ALIGN(64)”，而是写成“os : ALIGN(64)”。
- 7.汇编文件中自定义的section必须定义段属性，否则默认使用non-alloc属性。即使链接脚本指定该section存储在内存区域中，该部分实际上也会被丢弃。
- 8.ld.lld目前不支持带有立即数的call指令的汇编输入。
- 9.如果要使用ld.lld -r 生成可重定位的文件，则需要添加“-static”选项。
- 10.如果使用“jal 0x2222”这样的跳转指令，binutils将0x2222视为目标地址，而ld.lld则将其视为目标地址的偏移量。
- 11.在ld.lld中，'b0p92'和'zbpbo'不能同时启用，'b1.0'和'zbpbo'可以同时启用。
- 12.不要丢弃控制section，比如section .shstrtab，ld.lld会给出错误“discarding .shstrtab section is not allowed”，而GNU ld会忽略并保留它们。



# 4 编译问题

## 4.1 工程配置问题

### 4.1.1 使用 CMake 构建，编译器提示选项不识别

增加两个LLVM选项时不识别，如-mllvm optionA -mllvm optionB，编译器提示optionB选项不识别。这是因为多个具有重复字段的编译选项时，由于CMake自身具有自动去重的机制，会造成传递的编译选项的字段损失，导致编译失败。

```
list(APPEND GLOBAL_COMPILE_OPTIONS -Wall -Os ... -mllvm -fldm-stm-optimize -mllvm -imm-compare)
编译器提示
error: unknown argument: '-imm-compare'
clang-15: error: unknown argument: '-imm-compare'
```

解决方法1：在参数前面加上"SHELL: "，让cmake把该参数作为shell数据处理。

```
list(APPEND GLOBAL_COMPILE_OPTIONS -Wall -Os ..."SHELL:-mllvm -fldm-stm-optimize" "SHELL:-mllvm
-imm-compare" "SHELL:-mllvm -disable-shrink-to-bool=true")
```

解决方法2：使用CMakeList原生CMAKE\_C\_FLAGS单独设置。

```
list(APPEND CMAKE_C_FLAGS "-mllvm -fldm-stm-optimize -mllvm -imm-compare -mllvm -disable-shrink-
to-bool=true")
```

## 4.2 编译告警

切换毕昇编译器后可能会新增编译告警信息，编译告警是编译器在编译代码时发现的一些可能会导致程序运行出现问题的警告信息。这些警告信息虽然不会导致程序无法编译或运行，但是可能会影响程序的正确性和性能。建议仔细阅读告警信息，了解告警的原因和影响，并根据需要修改代码。对于一些无法修改的告警信息，可以使用-Wno-选项来屏蔽这些告警信息。例如提示告警-Wenum-conversion，则可使用-Wno-enum-conversion选项进行屏蔽。

### 4.2.1 Wreturn-type，返回值不正确告警

```
warning: non-void function does not return a value [-Wreturn-type]
示例：
int foo(int w)
{
```





```
w++;  
}
```

建议：函数定义中明确指定返回值类型。

## 4.2.2 Warray-bounds，数组访问越界告警

warning: array index 4 is past the end of the array (which contains 4 elements) [-Warray-bounds]

示例：

```
#define MAX_INDEX 4  
  
void Fool()  
{  
    int table[MAX_INDEX] = { 1, 2, 3, 4 };  
    table[MAX_INDEX] = 1; // 越界  
}
```

建议：在访问数组元素之前，应该先检查数组的大小，确保不会访问超出数组范围的元素。

## 4.2.3 Wunused-command-line-argument，选项未被使用告警

选项在当前执行过程中未使用到，例如编译汇编文件时使用了cflags中的编译选项。

建议：去掉无用的选项或选择屏蔽告警。

## 4.2.4 Wunused-variable，变量未被使用告警

通常是由于变量被定义但未被使用，或者是在某些条件下未被使用的情况引起的。

建议：删除未使用的变量，如果变量是有意未使用的，可以使用注释或者特殊的宏定义来标记。

## 4.2.5 Wunused-function，函数未被使用告警

通常是由于代码中存在未被使用的函数而导致的。

建议：删除未使用的函数，以减少代码冗余。

## 4.2.6 Wunused-parameter，函数参数被使用告警

通常是由于函数定义时的形参与实际使用时的需求不一致所导致的。

建议：如果该形参确实没有被使用到，可以将其删除，以避免告警。如果该形参是有意义的，可以屏蔽该告警。

## 4.2.7 Wincompatible-pointer-types，指针类型不匹配告警

毕昇编译器对函数入参检查更为严格，通常是由于不同类型的指针之间的转换引起的。

建议：在使用指针时，确保它们指向相同类型的数据。如果必须在不同类型的指针之间进行转换，请使用正确的类型转换。

## 4.2.8 Wenum-conversion，枚举类型不匹配告警

通常是因为枚举类型的值在赋值给其他类型的变量时，没有进行强制类型转换。这可能导致潜在的类型不匹配和数据损坏问题。

建议：在确保两个枚举是等价的情况下进行类型强转换。



### 4.2.9 Wstrict-prototypes, 无参数函数没有使用 void 标识无参数警告

在C语言中函数声明没有指定参数类型，导致编译器无法确定函数的参数类型，从而引起的警告。

建议：在函数声明中指定函数的参数类型，如果函数不接受任何参数，则可以使用void关键字表示。

### 4.2.10 Wtypedef-redefinition, 类型重复定义警告

通常是由于在多个源文件中定义了相同的类型名称，或者在同一源文件中定义了相同的类型名称的多个类型定义之间引起的。

建议：根据业务情况选择需要类型，正确包含头文件。

### 4.2.11 Wsometimes-uninitialized, 可能存在情况导致变量未初始化

通常是某个变量可能未被初始化就被使用了。这种情况可能会导致程序运行时出现不可预知的结果，因此应该尽量避免。

建议：可以在定义变量时进行初始化，确保变量在使用之前已经被赋值。如果变量的值在后续的代码中被修改，也要确保每次修改后变量的值都是有意义的。

### 4.2.12 Wunused-but-set-variable, 变量被赋值但未使用

可能是因为代码中存在一些无用的变量或者是因为代码中的某些变量被错误地赋值了。

建议：删除未使用的变量，如果变量是有意未使用的，可以使用注释或者特殊的宏定义来标记。

### 4.2.13 Wself-assign, 赋值语句中将变量赋值给自身警告

建议：自我赋值无意义，可以删除。

### 4.2.14 Wmissing-field-initializers, 结构体初始化不完整

```
warning: missing field 'y' initializer [-Wmissing-field-initializers]
typedef struct {
    td_u8 *x;
    td_u8 *y;
    td_u32 length;
} drv_pke_ecc_point;
.....
drv_pke_ecc_point A = {TD_NULL};
```

通常是在结构体或联合体初始化时，某些字段没有被显式初始化。这可能会导致未定义的行为。

建议：在定义结构体或联合体时，显式初始化所有字段。



## 4.2.15 Wtautological-pointer-compare, 指针比较结果为固定值

```
warning: comparison of array 'wpa_s->current_bss->ssid' not equal to a null pointer is always true [-Wtautological-pointer-compare]
(wpa_s->current_bss->ssid != NULL)) {
```

通常是由于代码中存在不必要的指针比较操作引起的。

建议：为了避免这种告警，应该检查代码中的指针比较操作，并确保它们是有意义的。

## 4.2.16 Wparentheses-equality, 判断条件有两层括号

```
warning: equality comparison with extraneous parentheses [-Wparentheses-equality]
if ((local_ip == NULL)) {
```

建议：删除多余括号。

## 4.2.17 Wtautological-constant-out-of-range-compare, 与常量比较结果为固定值

```
warning: result of comparison of constant 18446744073709551615 with expression of type 'size_t' (aka 'unsigned int') is always true [-Wtautological-constant-out-of-range-compare]
else if ( data_len < 0xFFFFFFFFFFFFFFFF )
```

建议：确认判断是否冗余，如果冗余可以删除。

## 4.2.18 Wlogical-not-parentheses, 逻辑非的作用范围可能存在问题

```
warning: logical not is only applied to the left hand side of this comparison [-Wlogical-not-parentheses]
if (!(netif->flags & NETIF_FLAG_DRIVER_RDY) != 0)) {
```

建议：在逻辑非运算符周围使用括号，以明确指定优先级。另外，建议养成良好的编码习惯，避免在代码中出现模糊的运算符优先级问题。

## 4.2.19 Wmisleading-indentation, 代码缩进不规范

```
warning: misleading indentation; statement is not part of the previous 'if' [-Wmisleading-indentation]
    if ( in_frames->first )
        res = in_frames->first->content;
        opcode = WebSocket_OP_BINARY;
```

通常是由于代码缩进不当引起的。缩进是代码中非常重要的一部分，它可以使代码更易读，更易于维护。但是，当缩进不正确时，编译器可能会误解你的代码意图，导致错误。

建议：使用一致的缩进风格。无论是使用制表符还是空格，都要保持一致，这样可以避免混淆。

## 4.2.20 Wuninitialized, 变量未初始化

```
warning: variable 'sys_config' is uninitialized when used here [-Wuninitialized]
    sys_config = bt_config_set_mask_bit(sys_config, BT_CONFIG_SYS_MASK_KEY_FROM_BT, 1);
```

建议：在使用变量前，确保其已经被初始化。



### 4.2.21 Wnull-pointer-subtraction, 使用空指针进行减法运算

```
warning: performing pointer subtraction with a null pointer has undefined behavior [-Wnull-pointer-subtraction]
    alg_aggr_param_offset(enable), alg_aggr_config_enable),
```

建议：在对指针进行减法操作之前，先进行非空判断。如果指针为空，就不应该进行减法操作。

### 4.2.22 Wpointer-sign, 指针的数据类型不匹配

```
warning: passing 'osal_s8[16]' (aka 'signed char[16]') to parameter of type 'osal_u8 *' (aka 'unsigned char *') converts between pointers to integer types with different sign [-Wpointer-sign]
    chba_cap_info_len = g_achba_get_chba_cap_func(hmac_vap->net_device, chba_cap_info,
    MAC_CHBA_CAP_INFO_LEN);
```

通常是因为在代码中使用了不匹配的指针类型，即将指针从一个类型转换为另一个类型时，可能会丢失一些信息。

建议：了解指针的类型和大小，显式地进行类型转换，确保指针类型匹配。



# 5 运行问题

## 5.1 链接脚本未包含新增段引起运行异常

相较于RISC-V32 GCC编译器，毕昇编译器最终生成的elf文件新增了.sdata段与.sbss段。切换毕昇LLVM后，如果未使用编译器自带的链接脚本，则需要确认链接脚本中是否包含新增的段，以免出现运行异常。

解决办法：修改链接脚本，匹配sdata段、sbss段内容。

```
.data : ALIGN(4)
{
    __data_load = LOADADDR(.data);
    __data_start = .;
    *(.data*)
    *(.sdata*)
    . = ALIGN(4);
    __data_end = .;
    .....
.bss (NOLOAD) : ALIGN(4)
{
    __bss_begin__ = .;
    *(.bss*)
    *(.sbss*)
    *(COMMON)
    .....
}
```