XCC 操作手册

V1.00

Pizer.Fan

Contents

前言

XCC 系统用于进一步提高软件代码质量，并加强流程管理的质量建设，尤其对软件开发阶段、验证阶段环节下的质量管理有着显著的帮助。

本文将在第一章讲述如何快速使用 XCC 系统。在后续的章节则是分别对 XCC 系统各个工具，诸如：XCC、XCovPost、XCovHTML 等进行详细的功能介绍。在本文的最后则是一些关联的附录，其中包括一些术语解释，如果您遇到一些不太理解的简称，则可以从术语的附录章节中获得帮助。

在这里，我重申一下 XCC 系统的功能，大致包括：

(一) 辅助静态扫描：
- ✓ 补充主流静态扫描工具不具备的；
- ✓ 代码可维护性度量；
- ✓ 代码风格规范；
- ✓ ASSERT 行管理；
- ✓ 函数栈开销；
- ✓ 函数依赖树；

(二) TRACE 扫描：
- ✓ 改善代码密度；
- ✓ 提供 LOG 性能；
- ✓ 提供前向兼容 ID 数据库管理，以及不修改源码的方式实现编译、链接；

(三) ASSERT 行扫描：
- ✓ 采用预编译方式扫描；
- ✓ 支持白名单数据库，过滤白名单之后提醒功能（警报）；

(四) 覆盖率度量：
- ✓ 主流 HOST 运行环境：windows、linux（ubuntu、centos、debian 等）；
- ✓ 主流芯片架构下各种编译器支持：例如 ARMCC、ARMCLANG、IAREWARM、ARM-NONE-EABI-GCC、ARC、RISCV 等等；
- ✓ 提供分支覆盖率、函数覆盖率，以及详细的 HTML 报告；

(五) XCC 系统的易用性
- ✓ 例如结合构建子系统（特指 LOS 平台的构建子系统），提供非常简单易用的操作。

欢迎大家提出建议或者意见！

最后祝您用的放心、舒心！

软件开发作者：Pizer.Fan

2022 年 8 月 15 日

1 快速开始

1.1 从构建开始

1.1.1 基于 LOS 构建子系统

如果您的项目已经基于 LOS 构建子系统进行编译、链接，那么您只需要将 LOS 构建子系统涉及到特定版本，即可无缝支持 XCC 的所有功能。

1.1.1.1 修改项目配置文件

例如您的项目配置文件：project_mysata.mak，如果期望构建的同时，执行辅助代码扫描，只需要增加如下一行：

```
…
XCCCODESCAN := on
…
```

在接下来的构建过程，您将同时看到辅助代码扫描的报告（通过或者告警）：

```
Remove output_clang_cov/obj/utest_xcc_cscan_rule/
output_clang_cov/dep/utest_xcc_cscan_rule/ ...
Compiling and update dependency ../code/iLib/iFileOps.c
Compiling and update dependency ../code/iLib/iString.c
Compiling and update dependency ../code/iLib/iStream.c
Compiling and update dependency ../code/iLib/iRbtree.c
Compiling and update dependency ../code/iLib/iList.c
Compiling and update dependency ../code/iLib/iAllocator.c
XccCScan ../code/iLib/iFileOps.c ... Okay
XccCScan ../code/iLib/iList.c ... 0 Errors, 29 Warnings
 [XccCScan] Warning 6: The iList_Create function did NOT match the naming rule
@../code/iLib/iList.c:198
 [XccCScan] Warning 6: The iList_Destroy function did NOT match the naming rule
@../code/iLib/iList.c:210
 [XccCScan] Warning 6: The iList_GetName function did NOT match the naming rule
@../code/iLib/iList.c:216
 [XccCScan] Warning 6: The iList_GetCount function did NOT match the naming rule
@../code/iLib/iList.c:222
…
```

1.1.1.2 查看静态扫描的结果

例如您的项目配置文件：project_mysata.mak，查看 output_mysata 目录：

```
\
|- cov
|- dep
|- lint
|- obj
|- out
|- tsc
|- xcov
|- build.log
|- macroes.option
|- xcc-cscan.log
|- xcc-protodb.dat
```

在 output_mysata 目录下，您可以看到 xcc-cscan.log 文件，其内容类似：

```
XccCScan ../code/xcc/tsc_strdb/tsc_strdb_lex.c : 0 Errors, 3 Warnings
  [XccCScan] Warning 1: Exceed cyclomatic complexity limit 70 > 15 in the tsc_strdb_yylex function
@../code/xcc/tsc_strdb/tsc_strdb_lex.c:9615
  [XccCScan] Warning 7: The tsc_strdb_yyleng global variable did NOT match the naming rule @../code/xcc/tsc_strdb/tsc_strdb_lex.c:215
  [XccCScan] Warning 7: The tsc_strdb_yyout global variable did NOT match the naming rule @../code/xcc/tsc_strdb/tsc_strdb_lex.c:264
XccCScan ../code/test_cov/test_cov.c : 0 Errors, 1 Warnings
  [XccCScan] Warning 1: Exceed cyclomatic complexity limit 55 > 15 in the func function @../code/test_cov/test_cov.c:172
XccCScan ../code/utest_clang_case1.c : 0 Errors, 7 Warnings
  [XccCScan] Warning 1: Exceed cyclomatic complexity limit 47 > 15 in the utest_base function @../code/utest_clang_case1.c:294
```

1.1.2 基于 Makefile 构建

1.1.2.1 修改编译过程

例如修改编译过程，通常找到%.0->%.c 的规则：

```
$(PROJECT_OUTDIR)/obj/$(MOD)/%.o : %.d
$(PROJECT_OUTDIR)/obj/$(MOD)/%.o : %.c
        $(ECHO) Compiling $<
        $(CC_EXEC) …
```

然后替换编译器命令 (参考如下颜色标记的修改)：

- ✓ 首先配置 XCC 执行程序的路径：XCC_EXEC := xcc.out (如果 windows 环境则是 xcc.exe，注意配置路径)
- ✓ 然后替换 $(CC_EXEC)编译器命令为：CC_EXEC := $(XCC_EXEC) $(XCC_CMD_OPTS) $(CC_EXEC)

```
# XCC
XCC_EXEC := xcc.out
ifeq ($(XCCCODESCAN),on)
XCC_CMD_OPTS += --cmd=codescan  --resultfile=./xcc-cscan.log --protodbfile=./xcc-protodb.dat
endif
ifeq ($(XCCCOV),on)
XCC_CMD_OPTS += --cmd=cov
endif
ifeq ($(XCCDRYSCAN),on)
XCC_CMD_OPTS += --cmd=dryscan
endif
ifneq ($(wildcard $(XCC_EXEC)),)
ifneq ($(XCC_CMD_OPTS),)
CC_EXEC := $(XCC_EXEC) $(XCC_CMD_OPTS) $(CC_EXEC)
endif
endif

$(PROJECT_OUTDIR)/obj/$(MOD)/%.o : %.d
$(PROJECT_OUTDIR)/obj/$(MOD)/%.o : %.c
        $(ECHO) Compiling $<
        $(CC_EXEC) …
```

1.1.2.2　修改链接过程（仅限 gcc 编译 x86 架构）

例如修改链接过程：

```
$(TARGET_OUT) : …
        $(ECHO) Link $@ …
        $(LINK_EXEC) …
        $(ECHO) Done
```

然后替换链接命令 (参考如下颜色标记的修改)：

```
$(TARGET_OUT) : …
        $(ECHO) Link $@ …
ifneq ($(wildcard $(XCC_EXEC)),)
        $(XCC_EXEC)  $(LINK_EXEC) …
else
        $(LINK_EXEC) …
endif
        $(ECHO) Done
```

1.1.3　基于 IDE 构建

暂未支持。

1.2　目标板的覆盖率数据输出

1.2.1　输出适配开发

XCC 系统提供 xcov_lib.c、xcov_lib_arm.c 等基础库文件，在目标板上，覆盖率数据的输出需要进行适配开发：

- 将数据输出重定向到 uart 输出或者保存到 ram、或者 flash 等存储介质；
- 同时还需要实现触发 dump 的过程——调用 xcov_dump；

1.2.2 临界适配

对于单核的临界适配，一般采用开关中断即可，有 RTOS 的也要避免采用 RTOS 层级的 MUTEX——基于性能考虑。对于多核的临界适配，则采用开关中断+自旋锁的方式，同样如果有 RTOS（SMP）支持，也要避免采用其 MUTEX 机制。

1.2.3 其它建议

对于启动模块、性能模块，如果想屏蔽覆盖率功能带来的影响，可以在源文件中增加：

```
#pragma XCov off
```

如果采用 makefile，例如基于 LOS 构建子系统方式，只需要在模块 makefile 中配置 XCCCOV = off 即可——该模块中所有文件将放弃覆盖率功能，例如：

```
TARGET := test_cov$(OUT_DEF_SUFFIX)

TARGET_OUT_DEPS_LIBS :=
TARGET_OUT_DEPS := $(TARGET_OUT_DEPS_LIBS)

CODE_ROOT_DIR := ../code
XCCCOV := off
```

1.3 覆盖率功能的开销

覆盖率功能的开销：

- ROM 开销：待定
- RAM 开销：待定

2 XCC 工具接口

2.1 命令帮助

```
>xcc -h
Usage: xcc [options] compiler [compiling-options]
  General options:
    -h/--help
      Prints this message
    -v/--version
      Prints the version info of the tool.
    --silent
      Disable outputing verbose info.
    --list-env
      List supported compilers.
    --color on|off
      Enable/disable outputing message with multi-color.

  Command options:
    --cmd <command>
      Execute XCC command:
      * dryscan      - just test XCC command.
        codescan     - scan codes to check if there're any warnings or not.
        assertscandb - scan assert lines database of C syntax, into precompiled database.
        assertscan   - scan assert lines to check if they're verified or not.
        tracescan    - convert constant strings to identifiers in the logging invocations.
        cov          - insert veneer codes for coverage.

  Command 'assertscandb' & 'assertscan' options:
    --assertfunc <assert-func-name>
      Specify the fucntion name of assert.
      For example :
        --assertfunc=my_assert1 --assertfunc=my_assert2
        or :
        --assertfunc=my_assert1:my_assert2
    --assertdbfile <precompiled-db-path>
      Specify the path of precompiled database.
    --assertresultfile <result-file>
      Specify the result file.

  Command 'codescan' options:
    --rulefile <rule-file>
      Specify the rule file.

      Rule options:
        --disable <rule-check-point>
        --enable <rule-check-point>
```

```
  Rule check point:
        cyclomatic_complexity_limit
        code_lines_limit
        for_max_depth_limit
        if_max_depth_limit
        include_files_limit
        loop_include_files
        func_naming_rule
        glb_var_naming_rule

      --cyclomatic_complexity_limit=<value>
      --code_lines_limit=<value>
      --for_max_depth_limit=<value>
      --if_max_depth_limit=<value>
      --include_files_limit=<value>

  --resultfile <result-file>
    Specify the result file.
  --protodbfile <protodb-file>
    Specify the prototype database file.

 Command 'tracescan' options:
  --tscdbfile <tsc-db-file>
    Specify the database file for trace scan.
  --create-mod <module list>
    Specify the module list to be created to the database.
    For example: "mod1 mod2 ..."
      The character of module name shall be one of [A-Z/a-z_0-9]
  --ls-log-header-file <file path>
    Specify the header file of LS_LOG/LS_LOG_ID macroes.

 Command 'cov' options:
  --zidata-section <section-name>
    Specify the section for the new XCov .bss data.
  --rodata-section <section-name>
    Specify the section for the new XCov .rodata data.

This XCC tool is using to scan codes for searching the possible defects, convert constant strings to
identifiers, and insert veneer codes for coverage.

########################################################################
XCC Release V1.00 Version, 2 July 2022.
Copyright (C) 2022 (Any question, please contact me: lzfzmpizer@sina.com).
########################################################################
```

2.2 静态扫描

2.2.1 输入规则文件

规则文件内容格式：

```
--rulefile <rule-file>
  Specify the rule file.

  Rule options:
    --disable <rule-check-point>
    --enable <rule-check-point>
      Rule check point:
        cyclomatic_complexity_limit
        code_lines_limit
        for_max_depth_limit
        if_max_depth_limit
        include_files_limit
        loop_include_files
        func_naming_rule
        glb_var_naming_rule
      --cyclomatic_complexity_limit=<value>
      --code_lines_limit=<value>
      --for_max_depth_limit=<value>
      --if_max_depth_limit=<value>
      --include_files_limit=<value>
```

例如输入规则文件 ./xcc-cscan-rule.txt：

```
--disable cyclomatic_complexity_limit
--disable code_lines_limit
--disable for_max_depth_limit
--disable if_max_depth_limit
--disable include_files_limit
--disable loop_include_files
--disable func_naming_rule
--disable glb_var_naming_rule

--enable cyclomatic_complexity_limit
--enable code_lines_limit
--enable for_max_depth_limit
--enable if_max_depth_limit
--enable include_files_limit
--enable loop_include_files
--enable func_naming_rule
--enable glb_var_naming_rule

--cyclomatic_complexity_limit=20
--code_lines_limit=2000
--for_max_depth_limit=4
--if_max_depth_limit=4
--include_files_limit=256
```

2.2.2 执行规则文件

如果是基于 LOS 构建子系统，只需要在项目配置文件中：

```
XCCCODESCAN_RULEFILE := ./xcc-cscan-rule.txt
```

如果是基于 Makefile 构建，则需要在替换编译命令过程，加上如下命令选项：

```
--rulefile=./xcc-cscan-rule.txt
```

2.2.3 代码中屏蔽规则

如果您期望在某个代码文件中屏蔽静态扫描的某些规则，例如：

```
#ifdef XccCScan
#pragma XccCScanSwitch off(maintainability_limit,code_lines_limit)
#endif
…
#ifdef XccCScan
#pragma XccCScanSwitch on
#endif
```

如果您期望在某个代码文件中屏蔽静态扫描的所有规则，例如：

```
#ifdef XccCScan
#pragma XccCScanSwitch off(*)
#endif
…
#ifdef XccCScan
#pragma XccCScanSwitch on
#endif
```

2.2.4 告警

2.2.4.1 ERROR 告警

ERROR 0 - Different function prototype

2.2.4.2 WARNING 告警

WARNING 128 - Exceed cyclomatic complexity limit in the function

WARNING 129 - Exceed code line limit in the function

WARNING 130 - The maintainability was bad in the function

WARNING 131 - The foreach maximum depth limit in the function

WARNING 132 - The branch maximum depth limit in the function

WARNING 133 - The function did NOT match the naming rule

WARNING 134 - The global variable did NOT match the naming rule

WARNING 135 - There's loop including codes

WARNING 136 - There's too many including files

2.2.4.3 INFO 告警

INFO 4096 - The %s global variable is recommended to rename with '_g' NOT 'g_'

2.3 ASSERT LINE 扫描

ASSERT LINE 扫描，基于 ASSERT LINE 白名单数据库，以及采用编译方式待编译的源文件进行扫描。

第一步、首先构建 ASSERT LINE 白名单数据库，例如构建一个 assertlinedb.c，其中创建一个 assertdb()函数，在函数中将各个 ASSERT LINE 加入其中：

```
#include <stdio.h>

void assertdb()
{
  MY_ASSERT(x > 0);
  MY_ASSERT2(x > 0);
}
```

第二步、首先要对 ASSERT LINE 白名单数据库进行预解析，输出解析后的白名单数据库，例如将 assertscandb.c 预解析为 assertscandb.c.i

- xcc --cmd=assertscandb，用于将 C 语法的 ASSERT LINE 白名单数据库，进行预处理。
- --assertdbfile=assertscandb.c.i，用来指明预处理后的白名单数据库。
- --assertfunc 用来指明 assert 函数，例如：my_assert, my_assert2。
- armcc -c assertscandb.c，表明是通过 armcc 编译器，其它包括一些编译选项，编译的目标 assertscandb.c

```
> xcc --cmd=assertscandb --assertfunc=my_assert --assertfunc=my_assert2
--assertdbfile=assertscandb.c.i armcc -c assertscandb.c
```

第三步、创建一个扫描结果文件，例如：result.txt

第四步、编译所有目标源文件

- 我们需要将原 <CC> -c xxxx.c 编译命令中<CC>替换为 xcc --cmd=assertscan --assertdbfile=assertscandb.c.i –assertfunc=my_assert <CC> 即可。
- xcc --cmd=assertscan，用于进行 assert 行扫描处理。
- --assertdbfile=assertscandb.c.i，用来指明预处理后的白名单数据库。
- --assertfunc 用来指明 assert 函数，例如：my_assert, my_assert2。
- --assertresultfile=result.txt 用来指明保存 assert 扫描结果的文件。

```
> xcc --cmd=assertscan --assertfunc=my_assert --assertfunc=my_assert2 --
assertdbfile=assertscandb.c.i --assertresultfile=result.txt armcc -c
xxxx.c
```

如果在源码中出现的 ASSERT 行，在白名单中没有找到，将会告警如下：

```
XccAssertScan xxxx.c ... 1 Warnings, 2 Passes
 [XccAssertScan] Pass : "my_assert((x>0))" @xxxx.c(25)
 [XccAssertScan] Pass : "my_assert2((x>0),"x > 0")" @xxxx.c(26)
 [XccAssertScan] Warning : "my_assert2((x!=0),"x != 0")" @xxxx.c(27)
```

如果--assertresultfile 参数指明了保存结果的文件，上述告警将保存到该文件中。

2.4 LOG ID 扫描

LOG ID 化重要的一些命令选项：

```
Command 'tracescan' options:
  --tscdbfile <tsc-db-file>
    Specify the database file for trace scan.
  --create-mod <module list>
    Specify the module list to be created to the database.
    For example: "mod1 mod2 ..."
      The character of module name shall be one of [A-Z/a-z_0-9]
  --ls-log-header-file <file path>
    Specify the header file of LS_LOG/LS_LOG_ID macroes.
```

2.4.1 输入数据库文件

建议我们指明 LOG ID 化的数据库文件，这样新增的 ID 就会以追加的方式加入——实现前向兼容，例如：

```
--tscdbfile=./tsc_db/tsc_db.cxx
```

如果没有指明数据库文件，那么 XCC 工具会缺省指明./tsc_db/tsc_db.cxx 作为当前的数据库文件。

2.4.2 输入 LOG API 重定向文件

考虑到 LS_LOG/LS_LOG_ID 等接口在不同的项目上可能有不同的重定向实现，因此要求开发者指明 LOG API 重定向文件，例如：

```
--ls-log-header-file=tools/runtime/xcc_ls_log.h
```

2.4.3 创建 LOG 模块

LS LOG 支持模块化，考虑到 LOG 模块的不可随意自动创建，需要手动创建，XCC 工具兼容 TSC 部分功能，例如创建 LOG 模块（将更新到数据库中），例如：

```
> tools\xcc.exe --cmd=tracescan --create-mod="ABC CDE"
XccTraceScan : Warning: The ABC module was already existed!
XccTraceScan : Warning: The CDE module was already existed!
```

值得一提的是：

- 建议待创建的模块名称：采用大写字母、下划线和数字组成；
- 如果模块已经创建，XCC 会报告相关告警；

2.5 代码覆盖率插装

2.5.1 项目配置或者模块配置

代码覆盖率插装，例如在模块 makefile 中（基于 LOS 构建子系统）配置 XCCCOV=on 即可：


```
>make p=clang_cov m="test_cov2 test_cov" build job=8
Update makefile dependency for the test_cov2 module ...
Remove output_clang_cov/obj/test_cov2/ output_clang_cov/dep/test_cov2/ ...
Update makefile dependency for the test_cov module ...
Remove output_clang_cov/obj/test_cov/ output_clang_cov/dep/test_cov/ ...
Compiling and update dependency ../code/test_cov/test_cov_filem.c
Compiling and update dependency ../code/test_cov/test_cov_file2.c
Compiling and update dependency ../code/test_cov/test_cov_file1.c
XccCov ../code/test_cov/test_cov_file2.c ... Okay [1 checkpoints]
XccCov ../code/test_cov/test_cov_filem.c ... Okay [1 checkpoints]
XccCov ../code/test_cov/test_cov_file1.c ... Okay [3 checkpoints]
Link output_clang_cov/out/test_cov2.exe ...
Done
Compiling and update dependency ../code/test_cov/test_cov.c
XccCov ../code/test_cov/test_cov.c ... Okay [122 checkpoints]
Link output_clang_cov/out/test_cov.exe ...
Done
Generate image ...
Making binaries ...
Done
```

2.5.2 代码中屏蔽

如果您期望在某个代码文件中屏蔽覆盖率插装，例如：

```
#ifdef XccCov
#pragma XccCovSwitch off
#endif

void func3(int a, int b, int c)
{
}


#ifdef XccCov
#pragma XccCovSwitch on
#endif

void func3_2(int a, int b, int c)
{
}
```

图 1 代码屏蔽覆盖率插装后的报告

如果您期望只屏蔽某一段代码，而不影响其它代码：

```
#ifdef XccCov
#pragma XccCovSwitch push(off)
#endif

void func3(int a, int b, int c)
{
}


#ifdef XccCov
#pragma XccCovSwitch pop
#endif

void func3_2(int a, int b, int c)
{
}
```

亦或者您期望只开启对某一段代码的覆盖率插装，而不影响其它代码：

```
#ifdef XccCov
#pragma XccCovSwitch push(on)
#endif

void func3(int a, int b, int c)
{
}


#ifdef XccCov
#pragma XccCovSwitch pop
#endif

void func3_2(int a, int b, int c)
{
}
```

2.5.3 指明插装数据的 section

如果您期望插装数据放入指定的 section，参考 cov 的相关参数：

```
 Command 'cov' options:
  --zidata-section <section-name>
    Specify the section for the new XCov .bss data.
  --rodata-section <section-name>
    Specify the section for the new XCov .rodata data.
```

例如项目配置 makefile 中：

```
# project_test.mak
XCC_CMD_OPTS := --zidata-section=”MYSEC” --rodata-section=”MYSEC”
XCCCOV := on
```

或者在命令行中输入相关参数：

```
xcc --cmd=cov --zidata-section=”MYSEC” --rodata-section=”MYSEC” …
```

如果你期望指定特定的文件的插装数据放入指定的 section，可以如下：

```
#ifdef XccCov
#pragma XccCovSection rodata = "cov.rodata", zidata="cov.zidata"
#endif
```

例如上我们在某个文件加入上述行，链接完成之后，我们可以查看其 section 信息：

```
>nm test.exe | grep cov\.
0000000140017000 r cov.rodata
0000000140014000 d cov.zidata
```

2.5.4 Bank/Overlay Linking

在嵌入式系统中，Bank 或 Overlay Link 是指将静态内存划分为多个不同的部分（称为 bank、overlay 等），并且允许程序在运行时从一个部分切换到另一个部分。通常这种方式被用于解决内存容量不足的问题，同时也可以提高程序的效率。

- 在 Bank Linking 中，程序员负责手动将静态内存划分为多个 bank，并将程序的各个部分存储在不同的 bank 中。然后，链接器根据程序员指定的 bank 分配方案，将每个模块分配到对应的 bank 中。程序只需要在需要时切换 bank 即可。
- 在 Overlay Linking 中，程序员将代码分成多个功能模块，并将每个模块放在内存中的不同区域。然后，链接器需要能够识别每个模块，并确定哪些模块可以共享同一片内存。例如，如果两个模块的内存空间没有重叠部分，那么它们可以被分配到同一片内存中。当程序执行时，只有当前需要的模块才会加载到内存中。如果需要切换到另一个模块，则会将当前的模块从内存中卸载，并加载所需的另一个模块。

Bank Linking 和 Overlay Linking 都需要链接器和编译器的支持，以便将代码正确地分配到不同的 bank 或 overlay 中，并维护好程序之间的跳转关系。根据具体的应用场景，程序员可以选择使用其中一种技术，或者将两种技术结合使用，以实现内存管理的最佳效果。

- Bank/ Overlay Linking 下 Bank 的 XCOV 数据要放入常驻内存中
- 在程序启动过程，需要将 Bank 的 XCOV 段复制到执行试图对应的位置

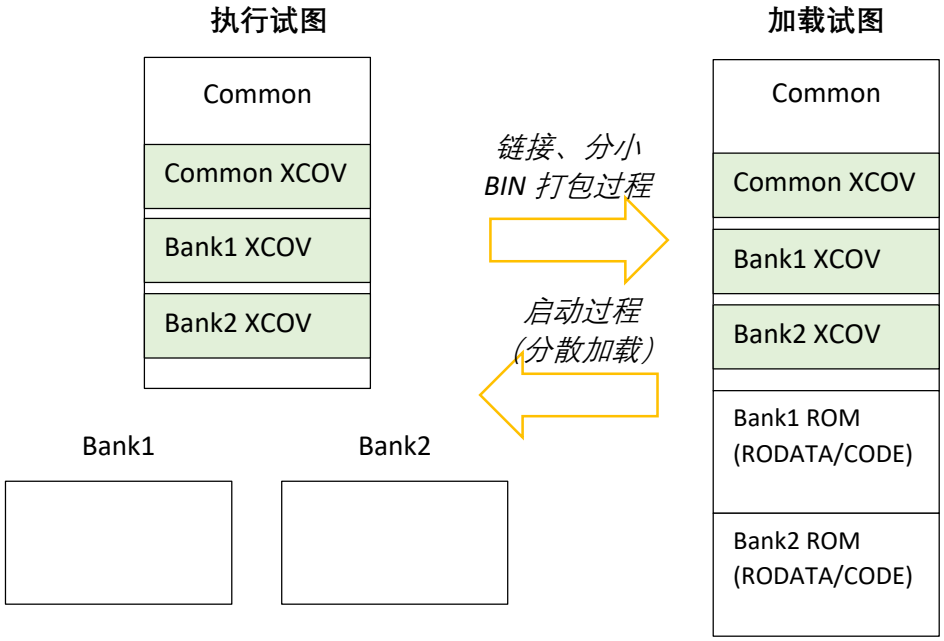

图 2 Bank/ Overlay Linking 下 Bank 的 XCOV 数据要放入常驻内存中

2.6 lib-xcov 库在目标环境下的适配

2.6.1 目标环境为 ARM/ARC/RISCV 等环境

XCC 系统提供 xcov_lib.c、xcov_lib_arm.c 等基础库文件，在目标为 ARM/ARC/RISCV 等环境下（以 xcov_lib_arm.c 为参考），lib-xcov 库有 3 个函数需要适配实现：

- xcov_dump_write: 将覆盖率数据输出到 host，具体实现可以暂存到 target 的 flash 或者 ram 中，后续过程获取出来，也可以通过 uart 等总线直接输出。
  static void xcov_dump_write(const void *data, int len)

```
{
  …
}
```

- xcov_lock/xcov_unlock：对于单核的临界适配，一般采用开关中断即可（注意可能的嵌套，例如在关闭中断的前置情况下），有 RTOS 的也要避免采用 RTOS 层级的 MUTEX——基于性能考虑。对于多核的临界适配，则采用开关中断+自旋锁的方式，同样如果有 RTOS（SMP）支持，也要避免采用其 RTOS 层面的 MUTEX 机制。

```
static void *xcov_lock()
{
  …
}
static void xcov_unlock(void *status)
{
  …
}
```

最后还需要实现触发覆盖率数据的 dump 过程——调用 xcov_dump。

2.6.2 目标环境为 HOST 环境

例如在 windows 或者 linux 上通过 gcc 编译目标，目前 XCC 自动提供（链接）host 环境下的 lib-xcov 代码库，该库函数直接不再调用 xcov_dump_write 而是直接 xcov_dump_file 到 host 文件系统中，另外在进程退出时候自动调用 xcov_dump。目前直接输出覆盖率数据到 .xcovdat 文件是编译时决定的绝对路径，因此建议复制到其它 host 环境下执行时：

- 建立相同路径，并复制编译时所有的.xcovdat 文件和执行程序
- 执行完成之后，需要将各个 host 的.xcovdat 通过 XCovPost 进行合并

3 XCovPost 工具接口

3.1 命令帮助

```
>xcovpost -h

Usage: xcovpost [options] <input-data-file>
  General options:
    -h/--help
      Prints this message
    -v/--version
      Prints the version info of the tool.
    -i <directory>
      Specify the input *.xcovdat directory to be merged.
    -o <directory>
      Specify the target directory of the database.
    --export[-force]
      Use exporting mode instead of merging one.
      In the exporting mode, the *.xcovsrc/*.xcovdat will be exported.
      --export-force: the existed files will be replace without prompt.
  <input-data-file>:
    Specify the input coverage data file path, it may be encoded or a .xcovdat file.

This XCovPost tool is using to merge the coverage data to the database.

################################################################################
XCovPost Release V1.00 Version, 2 July 2022.
Copyright (C) 2022 (Any question, please contact me: lzfzmpizer@sina.com).
################################################################################
```

3.2 导出功能

如果您编译完成之后，需要将*.xcovsrc/*.xcovdat 文件进行备份，可以通过导出功能进行完成，例如：xcovpost.exe -i <input-dir> -o <export-dir> --export-force

导出还有一个功能：汇聚。例如：一个工程分成若干独立子部分进行覆盖率插装，通过导出功能将它们汇聚一起：xcovpost.exe -i <dir1> -i <dir2> -i <dir3> -o <export-dir> --export

```
>xcovpost.exe -i output_clang_cov –o out_clang_export --export-force
XCovPost Scanning the input directories ...
  [XCovPost] Input 64 valid .xcovdat files
  [XCovPost] Export to myexport\obj\clang\clang_allocator.xcovdat ...
  [XCovPost] Export to myexport\obj\clang\clang_compound_statement_stat.xcovdat ...
  [XCovPost] Export to myexport\obj\clang\clang_func_body_stat.xcovdat ...
  [XCovPost] Export to myexport\obj\clang\clang_ir_common.xcovdat ...
  [XCovPost] Export to myexport\obj\clang\clang_ir_declarator.xcovdat ...
  [XCovPost] Export to myexport\obj\clang\clang_ir_expression.xcovdat ...
...
  [XCovPost] Export to myexport\obj\utest_xcc_main_general_6_listenv\test_api.xcovdat ...
  [XCovPost] Export to myexport\obj\xcc_lib\xcc_crc16.xcovdat ...
  [XCovPost] Export complete (64)!
```

3.3 合入目标板的覆盖率数据

目标板的覆盖率数据，可以通过重定向到 UART、或者 FLASH 中，后续再提取出来。各个文件的覆盖率数据可以分别保存为文件，也可以将它们汇聚为一个文件。

我们需要通过 XCovPost 命令将上面覆盖率数据合入到覆盖率数据库中——在构建的时候，产生的*.xcovsrc 和*.xcovdat 文件。XCovPost 命令提供目标数据库文件的自动查找（只需要指明数据库所在的目录），例如：

```
>xcovpost.exe -o output_clang_cov test_cov_data.txt
XCovPost Scanning the target directories ...
  [XCovPost] Merge the "test_cov_data.txt" to ... [58 valid .xcovdat target files]
  [XCovPost] Merge the "test_cov_data.txt" … Okay [1 files]
```

3.4 合入.xcovdat 文件

尤其是 HOST 上的应用程序，可能会在多个服务器上测试，那么它们各自运行的覆盖率数据更新输出到各个服务器上（.xcovdat 文件）。最后我们需要将各个服务器上的.xcovdat 汇聚一起，生成一个最终统一的报告。

XCovPost 命令支持输合入单个.xcovdat 到目标目录，也同时支持合入一个源目录中所有的.xcovdat 文件到目标目录。XCovPost 命令会自动配备它们编译的时间等，确保一一正确合入。

```
>xcovpost.exe -o output_clang_cov -i output_clang_cov
XCovPost Scanning the input directories ...
  [XCovPost] Input 58 valid .xcovdat files
XCovPost Scanning the target directories ...
  [XCovPost] Merge 58 .xcovdat files to ... [58 .xcovdat target files]
  [XCovPost] Merge output_clang_cov\obj\clang\clang_allocator.xcovdat ... Okay
  [XCovPost] Merge output_clang_cov\obj\clang\clang_compound_statement_stat.xcovdat ... Okay
  [XCovPost] Merge output_clang_cov\obj\clang\clang_func_body_stat.xcovdat ... Okay
  [XCovPost] Merge output_clang_cov\obj\clang\clang_ir_common.xcovdat ... Okay
  [XCovPost] Merge output_clang_cov\obj\clang\clang_ir_declarator.xcovdat ... Okay
  [XCovPost] Merge output_clang_cov\obj\clang\clang_ir_expression.xcovdat ... Okay
  [XCovPost] Merge output_clang_cov\obj\clang\clang_ir_func.xcovdat ... Okay
  [XCovPost] Merge output_clang_cov\obj\clang\clang_ir_tp.xcovdat ... Okay
…
```

4 XCovHTML 工具接口

4.1 命令帮助

```
>xcovhtml -h

Usage: xcovhtml [options]
  General options:
    -h/--help
      Prints this message
    -v/--version
      Prints the version info of the tool.
    -d <directory>
      Specify the input directory.
    -o <directory>
      Specify the output directory.

This XCovHtml tool is using to generate HTMLs reports for coverage info.

################################################################################
XCovHtml Release V1.00 Version, 2 July 2022.
Copyright (C) 2022 (Any question, please contact me: lzfzmpizer@sina.com).
################################################################################
```

4.2 输出覆盖率数据库目录的 HTML 报告

覆盖率数据库一般按模块区分目录，通过 XCovHTML 命令，我们可以指明目录输出 HTML 报告，例如：

```
>xcovhtml.exe -d output_clang_cov -o output_clang_cov/xcov
XCovHtml There's 42 .xcovdat files found!
  [XCovHtml] processing file ../code/clang/clang_func_body_stat.h ... decisions 0/10, functions: 0/6
  [XCovHtml] processing file ../code/clang/clang_ir.h ... decisions 0/145, functions: 0/143
  [XCovHtml] processing file ../code/clang/clang_ir_expression.h ... decisions 0/3, functions: 0/3
  [XCovHtml] processing file ../code/clang/clang_pars.h ... decisions 0/9, functions: 0/9
...
  [XCovHtml] processing file c:\mingw\include\wchar.h ... decisions 0/1, functions: 0/1
  [XCovHtml] processing file c:\mingw\include\winnt.h ... decisions 0/4, functions: 0/4
  [XCovHtml] processing file c:\mingw\include\winsock.h ... decisions 0/14, functions: 0/4
  [XCovHtml] total ... decisions 2679/4882, functions: 514/913
...
```

4.3 输出 HTML 报告时的过滤配置

在 XCovHTML 命令同级目录下的 xcovhtml-skips.conf 为过滤配置文件，在该文件可以指明过滤的前缀目录，例如：

```
c:/msys64/mingw64/
c:/MinGW/include/
c:/MinGW/lib/
```

4.4 查看 HTML 报告

通过 XCovHTML 命令输出的 HTML 报告放在了指明的输出目录下面（如果指明，则会放在当前工作目录下），在输出目录中我们可以找到 index.html，通过浏览器打开。

4.4.1 查看 HTML 概要报告

我们通过浏览器打开 index.html：

- Title: 总体概要的 title 显示为 XCOV，而各个文件详细的 title 则为 source 路径；
- 全部统计信息：总体概要提供总和信息，包括所有文件的所有条件/判定的覆盖情况，以及所有函数的覆盖情况；
- 文件列表：总体概要列举所有的 source（源文件或者有内联函数的头文件），以及每一个文件的条件/判定、函数的覆盖情况；

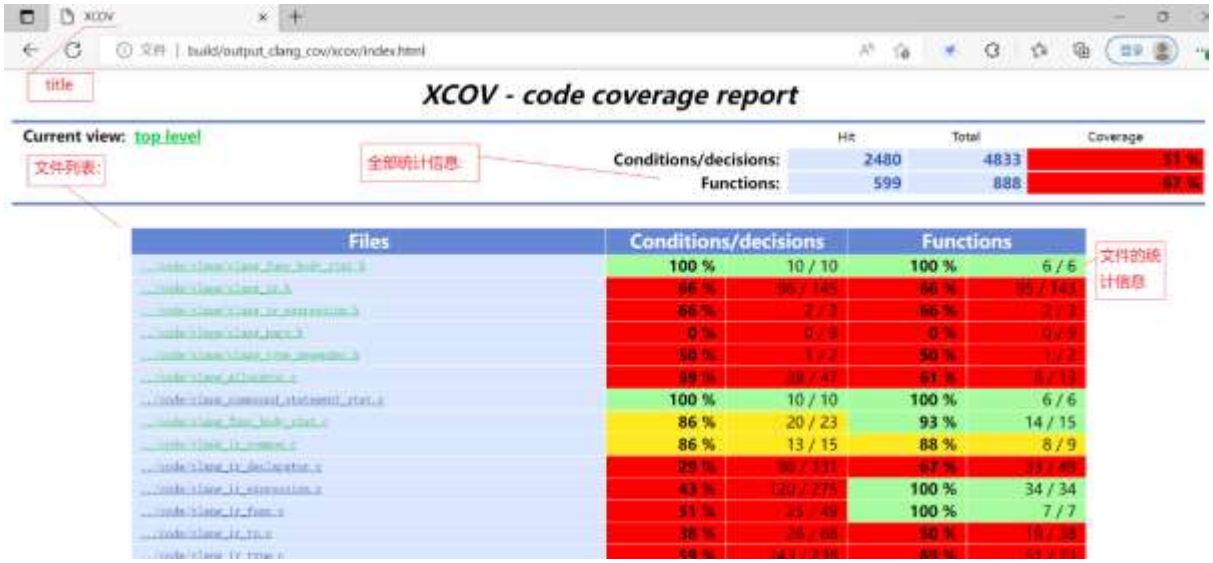

图 3 查看 HTML 概要报告

4.4.2 查看具体文件的详细报告

打开 index.html 之后，选择指定文件打开：

- 覆盖率统计信息：文件的条件/判定、函数的覆盖情况；
- 文件的详细覆盖率信息：
  - ✓ 代码行栏：对应源代码的行号。
  - ✓ Status 栏：共有三种状态——'T'、'F'和'!'，其中'T'表示条件为真的判定已经覆盖，'F'表示条件为假的判定已经覆盖，而'!'则是表示该条件的某种判定没有覆盖

——对于单个状态而言，一般表示真的判定没有覆盖，对于结对状态，则按照 TF 这样的信息交替对应那种判定没有覆盖。

- ✓ Source code 栏：对于源代码的具体内容。

- XCov 版本和版权信息：指示 XCov 版本信息和版权信息。

图 4 查看具体文件的详细报告

5 XCScanView 工具接口

5.1 命令帮助

```
> xcscanview -h
  General options:
    -h/--help
      Prints this message
    -v/--version
      Prints the version info of the tool.
    -i <directory>
      Specify the input *.cscandat directory.
    --top <top-level>
      Specify the top level, such as 10.
    --by <mt|cl|cc|all>
      Sort by:
       *mt - maintainability
        cl - code lines
        cc - cyclomatic complexity

This XCScanView tool is using to view the codescan info.

################################################################################
XCScanView Development V1.00 Version, 2 July 2022.
Copyright (C) 2022 (Any question, please contact me: lzfzmpizer@sina.com).
################################################################################
```

5.2 查看函数的可维护性排行

查看函数的可维护性排行，例如：

```
>xcscanview -i output_clang_cov --top=20 –by=mt
XCScanView Scanning the input directories ...
  [XCScanView] Top 20 by maintainability
    [01]  0 : clang_pars_yyparse in ../code/clang_yacc.c
    [02]  6 : yylex_ in ../code/clang_lex.c
    [03] 33 : tsc_strdb_yylex in ../code/xcc/tsc_strdb/tsc_strdb_lex.c
...
    [20] 56 : yy_get_next_buffer in ../code/clang_lex.c
```

函数的可维护性数值越低，表明其可维护的难度越大，因此其排行越靠前——用于警示。

5.3 查看函数的代码行数排行

查看函数的代码行数排行，例如：

```
>xcscanview -i output_clang_cov --top=20 –by=cl
XCScanView Scanning the input directories ...
  [XCScanView] Top 20 by code lines
    [01] 2549 : clang_pars_yyparse in ../code/clang_yacc.c
    [02] 1168 : yylex_ in ../code/clang_lex.c
    [03] 400 : tsc_strdb_yylex in ../code/xcc/tsc_strdb/tsc_strdb_lex.c
…
    [20]  87 : iRbtree_TestMain in ../code/iLib/test_irbtree.c
```

函数的代码行数数值越大，通常表明其可维护的难度越大，因此其排行越靠前——用于警示。

5.4 查看函数的圈路复杂度排行

查看函数的圈路复杂度排行，例如：

```
>xcscanview -i output_clang_cov --top=20 –by=cc
XCScanView Scanning the input directories ...
  [XCScanView] Top 20 by cyclomatic complexity
    [01] 395 : clang_pars_yyparse in ../code/clang_yacc.c
    [02] 196 : yylex_ in ../code/clang_lex.c
    [03] 77 : utest_type_depender in ../code/utest_clang_case1.c
 …
    [20] 15 : clang_relational_expr_get_value in ../code/clang_ir_expression.c
```

函数的圈路复杂度数值越大，通常表明其可维护的难度越大，因此其排行越靠前——用于警示。

附录 1 术语

编号	简称	注释
1	XCC	eXtended C Compiler，本文设计目标工具名称。也指 XCC 系统，即相关工具链集合。
2	LOS	Love Operating System——指本文作者 Pizer.Fan 设计的操作系统（支持轮询调度、单核实时抢占调度、SMP 抢占调度等）。LOS 通常又泛指整个 LOS 平台——除 LOS 微内核之外，还包括各个基础软件模块，包括内存管理、核间通信、调试组件、有限状态机引擎，等各类基础中间件。
3	C99	C99 是 ISO / IEC 9899：1999 的非正式名称，在 1999 年推出，被 ANSI 于 2000 年 3 月采用。
4	ANSI C	ANSI C 是由美国国家标准协会（ANSI）及国际标准化组织（ISO）推出的关于 C 语言的标准。ANSI C 主要标准化了现存的实现， 同时增加了一些来自 C++ 的内容并支持多国字符集。 ANSI C 标准同时规定了 C 运行期库例程的标准。狭义的 ANSI C: 把 C89 标准叫做 ANSI C，因为这个标准是美国国家标准协会（ANSI）发布的。
5	TSC	Trace SCan ，本文主要特指 TSC 工具，LOS 平台下的一款 TRACE ID 化扫描工具，以修改源码的方式实现的。
6	XCScanView	XCC Code Scan View：用于查看代码扫描中间数据库的工具；
7	XCov	XCC Coverage: 本文特指测试覆盖率，如代码行覆盖率、分支覆盖率等；
8	XCovPost	XCC Coverage Post: 将测试目标输出的 Coverage 运行时片段数据，合入到 Coverage 数据库；
9	XCovHtml	XCC Coverage Html generator: 将 Coverage 数据库生成 HTML 格式的覆盖率报告；
10	XAssertScan	XCC Assert Scan: 用于查看代码中 ASSERT 行的工具，并支持过滤白名单的提醒功能；
11	CC	本文有两种缩写含义： ● C Compiler：C 语言编译器； ● Cyclomatic complexity：圈复杂度，也称为条件复杂度，是一种衡量代码复杂度的标准。

表 1 术语

附录 2 TSC 工具

TSC（TraceSCan）工具将转换源文件中的日志打印为 “TRACEID” 形式，其目的用于节省目标程序的代码尺寸、提高 LOGGING 性能等等。TSC 工具的详细指南，请参考《LS-TSC 工具用户手册》。

下面是一个使用 TSC 工具的简单例子：

```
> ls-tsc.exe --db-path "..\tsc_db\tsc_db.cxx" -l ftl_file_list.txt
```

(1) ls-tsc.exe : 是 TSC 执行程序

(2) --db-path “..\tsc_db\tsc_db.cxx”：指明 ID 化数据库，该数据库是 C++语言文件，记录了 TSC 转换的所有“TRACEID”，为了前向兼容，一般情况下该数据库只能追加。

(3) -l ftl_file_list.txt ：通过 ftl_file_list.txt 指明了 ftl 模块的所有文件，当然也可以直接指定待扫描的文件们。

上述命令将 ftl_file_list.txt 列举文件中的 LS_LOG 转成为 ID 打印，例如某个文件：

```
/* ftl_sample.c */
...
  LS_LOG(CRIT, FTL, "Test FTL LOG!");
  LS_LOG(CRIT, FTL, "Test FTL LOG with p1 = %d!", 10);
->
  LS_LOG_ID(CRIT, FTL, "Test FTL LOG!", 0,(), Test_FTL_LOG_);
  LS_LOG_ID(CRIT, FTL, "Test FTL LOG with p1 = %d !", 1,(10), Test_FTL_LOG_with_p1__d_);
...
```

此时，若是去检查 ID 数据库——“..\tsc_db\tsc_db.cxx”，你会发现增加的“TRACEID”们。

附录 3 代码度量 - 可维护性指数范围和含义【微软】

源于【微软】官网——可维护性指数 - 计算介于 0 到 100 之前的指数值，它表示维护代码的相对难易程度。 较高的值表示可维护性更高。 颜色编码评级可用于快速标识代码中的故障点。 绿色评级介于 20 到 100 之间，表示代码可维护性良好。 黄色评级介于 10 到 19 之间，表示代码可维护性适中。 红色评级是介于 0 到 9 之间的评级，表示可维护性低。 有关详细信息，请参阅可维护性指数的范围和含义（参考 Code metrics - Maintainability index range and meaning - Visual Studio (Windows) | Microsoft Docs）。

The metric originally was calculated as follows:

```
Maintainability Index = 171 - 5.2 * ln(Halstead Volume) - 0.23 * (Cyclomatic Complexity) - 16.2 * ln(Lines of Code)
```

The use of this formula meant that it ranged from 171 to an unbounded negative number. As code tended toward 0, it was clearly hard to maintain code and the difference between code at 0 and some negative value was not useful. As a result of the decreasing usefulness of the negative numbers and a desire to keep the metric as clear as possible, we decided to treat all 0 or less indexes as 0 and then rebase the 171 or less range to be from 0 to 100. For this reason, the formula we use is:

```
Maintainability Index = MAX(0,(171 - 5.2 * ln(Halstead Volume) - 0.23 * (Cyclomatic Complexity) - 16.2 * ln(Lines of Code))*100 / 171)
```

In addition to that, we decided to be conservative with the thresholds. The desire was that if the index showed red then we would be saying with a high degree of confidence that there was an issue with the code. This gave us the following thresholds:

For the thresholds, we decided to break down this 0-100 range 80-20 to keep the noise level low and we only flagged code that was suspicious. We've used the following thresholds:

- 0-9 = Red
- 10-19 = Yellow
- 20-100 = Green

附录 4 圈复杂度的简单介绍

参考 Code metrics - Cyclomatic complexity - Visual Studio (Windows) | Microsoft Docs 或者 Introducing Cyclomatic Complexity in C# | CodeGuru.com， Cyclomatic complexity can be calculated as follows:

```
M = E − N + 2 * P
```

Where:

- M = cyclomatic complexity
- E = number of edges of the graph
- N = number of nodes of the graph
- P = number of connected components

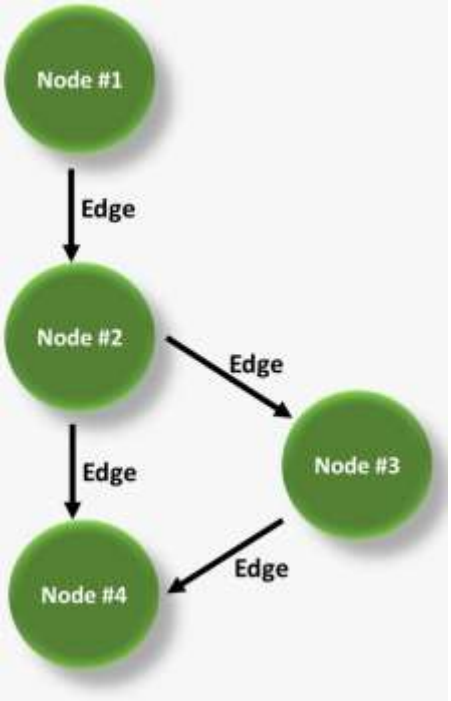

图 5 Illustrating Cyclomatic Complexity

附录 5 圈复杂度的衡量标准【网友】

源于【网友】的一段摘抄：代码复杂度低，代码不一定好，但代码复杂度高，代码一定不好:

圈复杂度	代码状况	可测性	维护成本
1 - 10	清晰、结构化	高	低
10 - 20	复杂	中	中
20 - 30	非常复杂	低	高
>30	不可读	不可测	非常高

表 2 圈复杂度的衡量标准