|  |
| --- |
| 内部 |
| 软件库项目说明 |
| 乾芯科技 |
| 2023 09.5 |

[第一章 项目使用流程 1](#_Toc144833243)

[第二章 项目代码规范 6](#_Toc144833244)

[第三章 VCS仿真流程 7](#_Toc144833245)

[3.1 全芯片快速前仿 7](#_Toc144833246)

[3.2 全芯片带bootloader前仿 7](#_Toc144833247)

[3.3 核1仿真开关 8](#_Toc144833248)

[3.4 X态传播开关 8](#_Toc144833249)

[3.5 全芯片带网表仿真 8](#_Toc144833250)

[3.6 使用verdi替代DVE仿真 9](#_Toc144833251)

[3.7 全量DUMP开关 9](#_Toc144833252)

# 项目使用流程

software\_lib\_driver项目为qx\_c2000配套的软件项目，主要用于用户级lib库和系统级验证用例日常更新和维护。

项目文件和目录说明：

qx\_c2000/device: 存放所有提供给客户的使用的lib封装库

qx\_c2000/device/inc: 为lib库头文件集合，用户全局通过#include “devicelib.h”即可实现对整个lib库的引用；

qx\_c2000/device/src: 为lib库非内联函数的实现集合；

qx\_c2000/device/src/core0: 存放Core0的汇编启动文件；

qx\_c2000/device/src/core1: 存放Core1的汇编启动文件；

qx\_c2000/tests：存放所有系统级验证用例，其中common.h和common.c用于存放所有验证用例都可能用到的通用函数库封装。device.h和device.c用于存放板级设备初始化函数，主要为Device\_init函数（系统时钟配置在该函数中实现）；

software\_lib\_bootloader: bootloader的项目维护目录，bootloader程序会在系统启动初期由硬件自动搬运到ram并执行，bootloader运行后主要负责将core0和core1的可执行程序通过DMA搬运到各自的ram并跳转执行。bootloader正常不需要修改，编译一次产生可执行文件即可（Release/ outer\_insn\_mem.dat）；

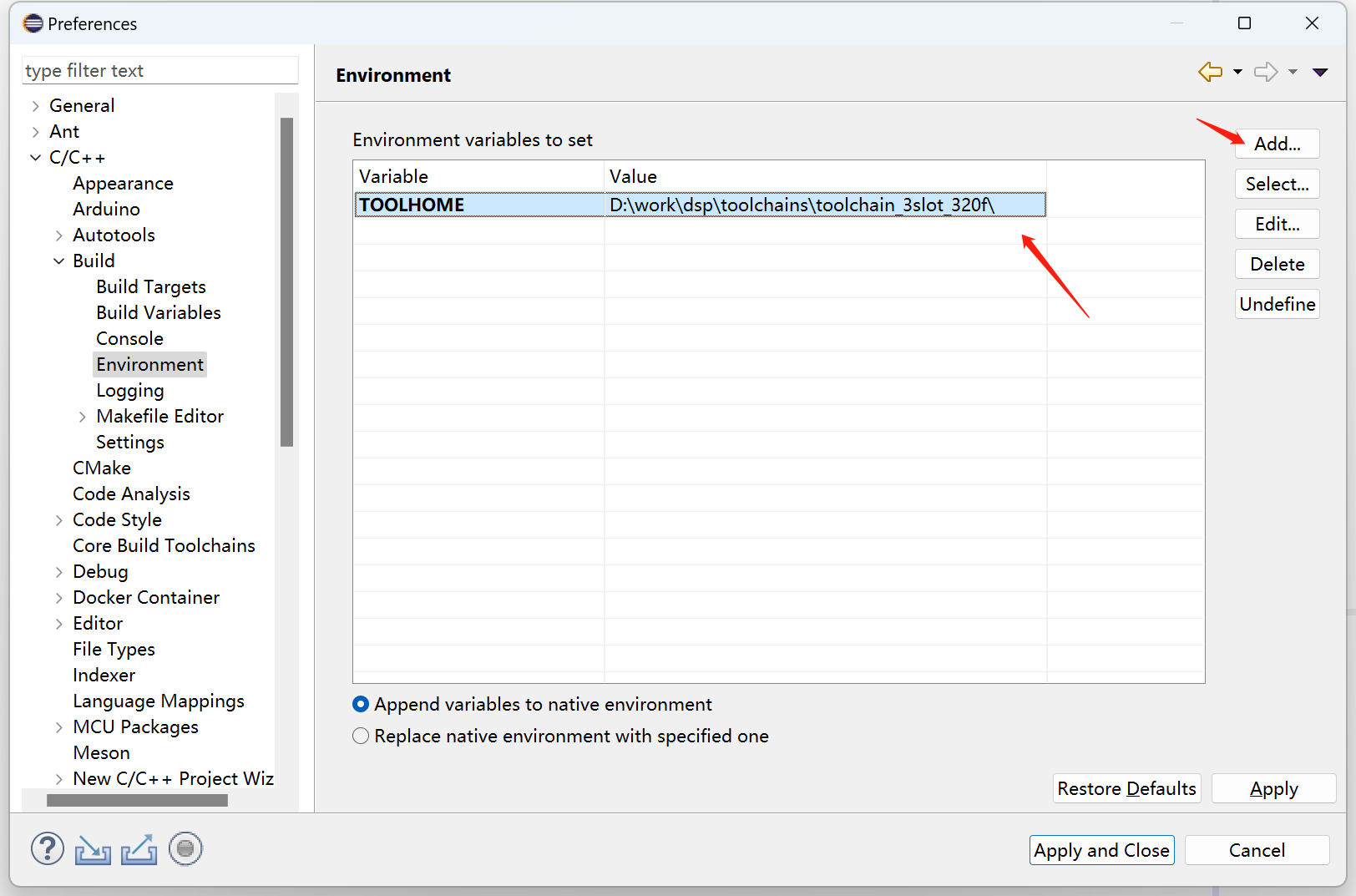
software\_lib\_driver\_core0：Core0的项目维护目录，main.c存放Core0的main函数入口，编译产生的Release/ outer\_data\_mem\_core0.dat为可执行程序的数据段，Release/ outer\_insn\_mem\_core0.dat为可执行程序的数据段；

software\_lib\_driver\_core1：Core1的项目维护目录，main.c存放Core1的main函数入口，编译产生的Release/ outer\_data\_mem\_core1.dat为可执行程序的数据段，Release/ outer\_insn\_mem\_core1.dat为可执行程序的数据段；

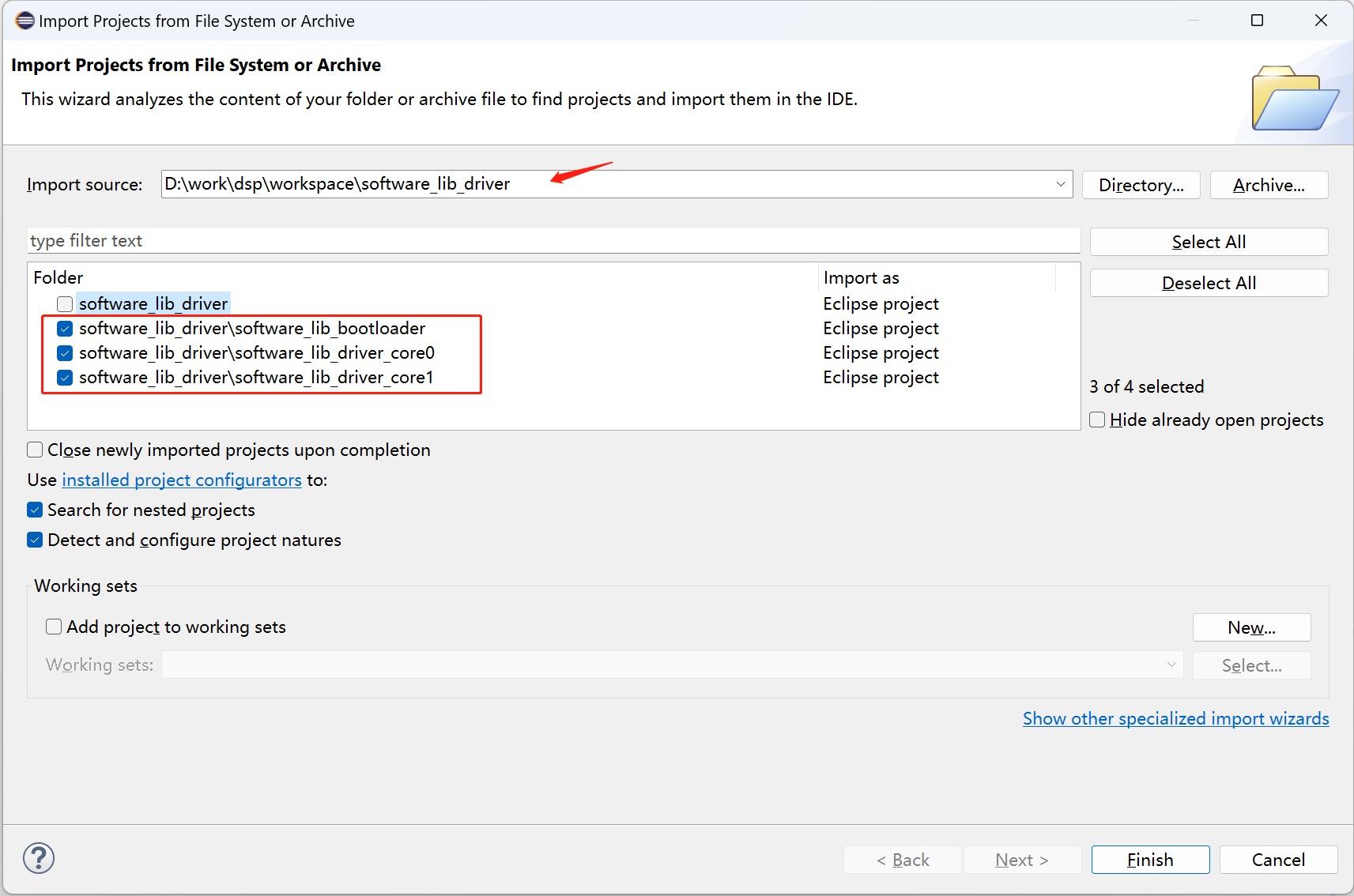
gen\_package.py：目标程序打包工具，负责将bootloader、core0、core1产生的二进制可执行文件（各自Release目录下的\*.out文件）汇总成init\_mem.txt文件，该文件在仿真阶段会用到。

项目使用：

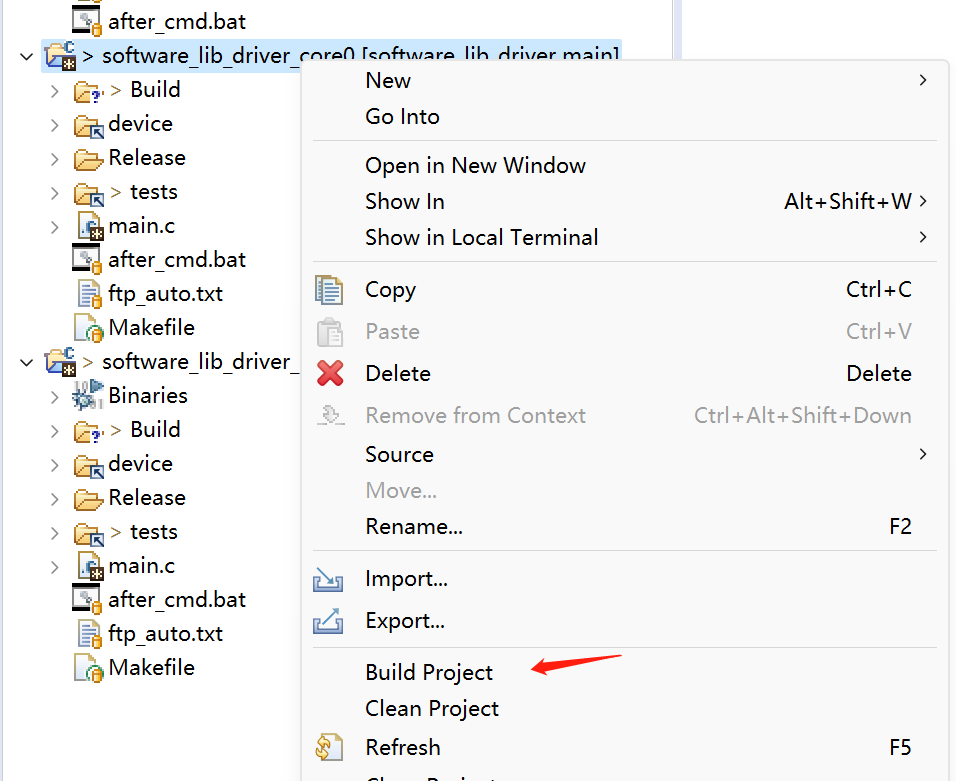
首次启动IDE需要配置TOOLHOME环境变量，该环境变量需要正确的指定到编译工具链所在目录才能正常编译（windows -> Preferences -> C/C++ -> Build -> Environment -> Add…）：



通过IDE的file -> Open Projects From File System… 可以直接一次性导入三个项目（注意不要勾选第一个software\_lib\_driver项目）：



导入后通过在项目目录上右键-> Build Project即可编译：



为减少编译后代码大小，一般建议编译时只添加需要参与编译的文件，项目中所有qx\_xxx.c的用户lib库文件以及test\_xxx.c验证用例文件都可以通过如下方法在编译时加入或排除编译：

1. 选择需要排除或者加入的文件，右键 -> Resource Configurations -> Exclude from Build…

图形用户界面, 应用程序

描述已自动生成

1. 箭头位置勾选为排除编译，取消勾选为加入编译：

图形用户界面, 文本, 应用程序, Word

描述已自动生成

为加速编译仿真速度，项目集成了编译后目标文件通过ftp自动上传功能，通过如下配置可实现编译并自动上传功能：

1. 编辑after\_cmd.bat文件，确保ftp -s:..\ftp\_auto.txt命令正常开启：

图形用户界面, 应用程序, Word

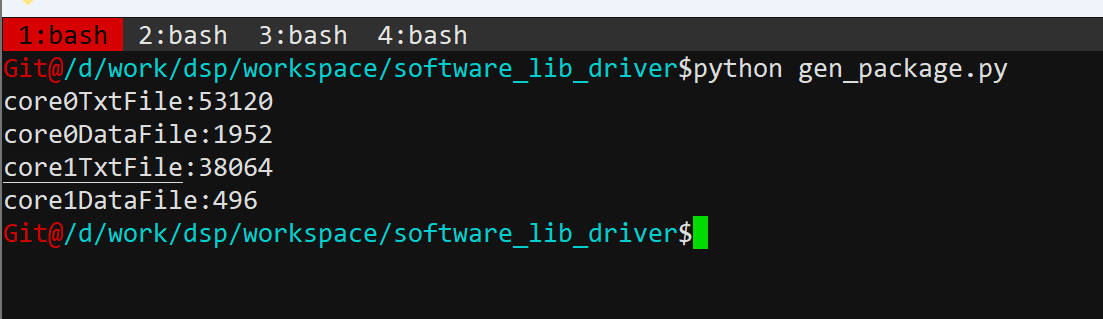
描述已自动生成

1. 编辑ftp\_auto.txt文件，修改[your\_ftp\_name]、[your\_ftp\_password]为自己的ftp账户和密码（需要去掉中括号），cd qx\_c2000/vcs是相对自己服务器家目录的qx\_c2000/vcs目录路径，修改成自己的qx\_c2000项目路径即可。

图形用户界面, 文本, 应用程序, Word

描述已自动生成

默认情况下，VCS仿真时载\*.dat文件，带bootloader 前仿时需要将bootloader、core0、core1三个项目产生的dat文件打包成init\_mem.txt文件，打包方式为在三个项目编译完成后在software\_lib\_driver目录下执行python gen\_package.py：



生成的init\_mem.txt上传到qx\_c2000/vcs目录下即可。

# 项目代码规范

为增加代码编写和用户LIB库交付质量，项目代码需要遵循如下基本的代码规范：

1. 为避免Tab带来的格式化问题，项目中所有地方禁止使用真实的Tab，所有需要使用Tab的地方统一使用4个空格取代，IDE需要做如下配置来实现该特性：

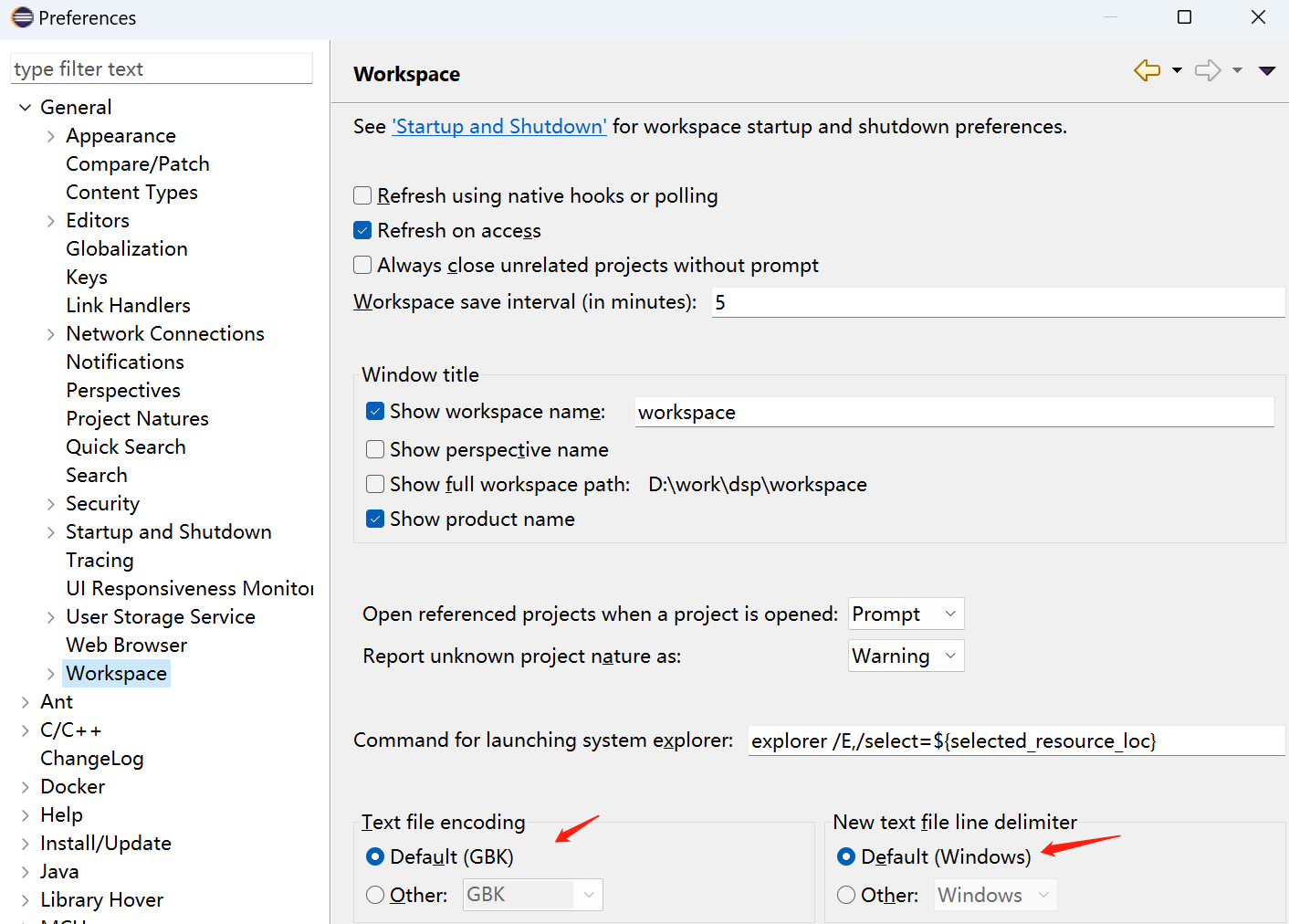
windows -> Preferences -> General -> Editors -> Text Editors:

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

箭头位置实现配置一个Tab 等于 4个空格并实现在输入Tab键时自动扩展为4个空格。

1. 中文编码统一采用GBK编码，回车符采用windows风格(“\r\n”)，如果出现乱码，请修改默认文本编码格式配置（Windows -> Prefereces ->General->Workspace）：



1. 行尾尽量不要出现多余空格，注释、枚举和宏等定义尽量做到对齐；
2. 为方便代码注释，除了函数头说明等特殊注释体，通常对代码内容的注释，需统一使用“//”作为默认代码注释风格，“//”后面需要有且只有一个空格；
3. 函数和函数之间需要有且只有一行空格进行分割，函数逻辑中根据逻辑块分布适量加入空行分割；
4. 用户lib库的封装风格和命名规范需要统一遵循TI C2000的原始风格，尽量做到和TI库的统一性和兼容性；
5. 测试用例的编写建议能调用用户lib库函数的地方调用lib库函数以实现同时对lib库的验证，另外不同用例之间建议通过封装函数的方式区分开，函数名要充分说明用例的主要验证点，风格上建议使用“小写 + 下划线”的方式命名；
6. 由于TI原始的lib库中通过位与、位或操作实现的bit位操作可读性差，新的用户lib库封装时能用结构位域操作完成的操作建议尽量使用结构体位域操作取代原有风格以提高代码可读性。

注意：项目根目录下的.clang-format文件是用clang-format工具定义的代码自动格式化规范，可以通过clang-format工具或插件实现对代码的一键格式化。

# VCS仿真流程

## 全芯片快速前仿

1. 将qx\_c2000项目克隆到本地：

cd $HOME && git clone [git@10.18.30.198:qx\_dsp/qx\_c2000.git](mailto:git@10.18.30.198:qx_dsp/qx_c2000.git)

1. 切换到编译目录并编译qx\_c2000项目：

cd qx\_c2000/vcs

make comp\_fullchip

注意：make vclean用于清除编译结果并全量重新编译

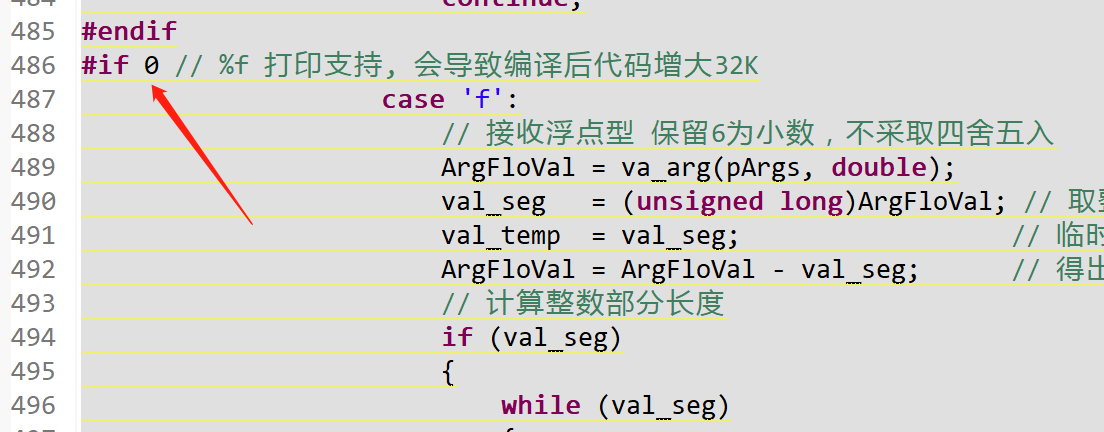
1. 更新vcs目录下的outer\_data\_mem\_core0.dat、outer\_insn\_mem\_core0.dat、outer\_data\_mem\_core1.dat、outer\_insn\_mem\_core1.dat文件
2. 运行vcs并仿真：

make run

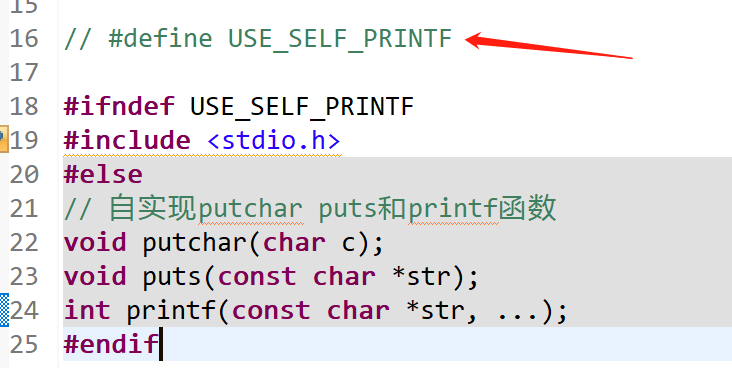
## 仿真输出打印方法切换

全芯片C环境中串口打印输出采用标准函数puts和printf实现，两个函数在软件上存在两个版本：

1. 自实现版本，该版本的puts和printf逻辑在common.c中自定义实现，其中printf只支持简单的%c、%s、%x、%d格式化打印，为减少目标文件大小，%o、%b、%f作为可选项支持，默认关闭，开启方法是解开对应代码注释：



1. libc库内建版本，该版本是工具链自带的libc库实现版本，printf支持全量的格式化打印，缺点是输出目标文件大，打印速度变慢，为避免目标程序过大导致的问题，暂时不建议采用该方法打印。
2. 项目在comon.h中采用USE\_SELF\_PRINTF宏来实现以上两个版本打印支持的切换，默认开启，如果希望使用libc库版本，需要手动屏蔽该宏：



此外，为加快仿真速度，串口打印功能在硬件上也存在两个版本：

1. 特殊寄存器快速打印版本，该版本采用向专门为仿真验证准备的特殊寄存器0x007f0000连续写数据实现在DVE、verdi界面的console窗口中输出puts、printf函数的输出功能，优点是仿真速度快，尤其是在中断使用时，对中断的处理速度影响很小，由于该方法可以很大程度上加快仿真，默认处于开启状态。

注意：快速打印版本只能在全芯片前仿以及门级网表前仿、后仿阶段使用，FPGA以及真实的芯片验证中不能使用。

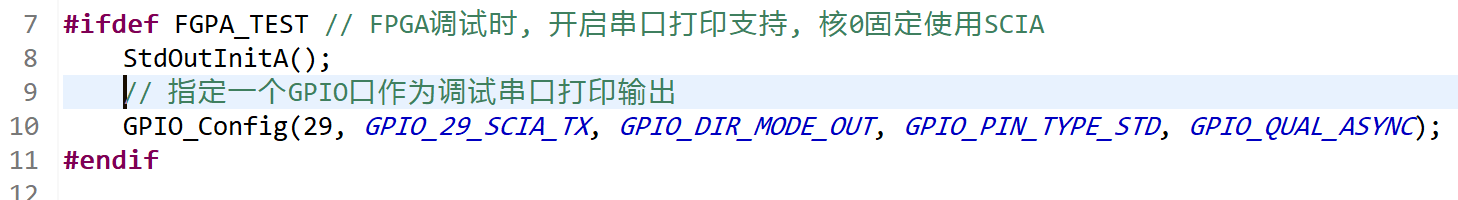
1. 真实串口打印版本，该版本采用向真实的SCIA、SCIB两个串口发送打印数据的方式实现打印，仿真阶段由tb\_capture\_0和tb\_capture\_1模块实现串口数据的接收并向DVE、verdi窗口的console窗口重定向输出打印结果：

图形用户界面, 文本, 应用程序

描述已自动生成

为避免双核向同一个串口打印导致输出串扰问题，项目默认情况下人为约定核0总是向SCIA输出，核1总是向SCIB输出。

software\_lib\_driver\_core0\main.c有对SCIA打印的初始化逻辑：



software\_lib\_driver\_core1\main.c有对SCIB打印的初始化逻辑：



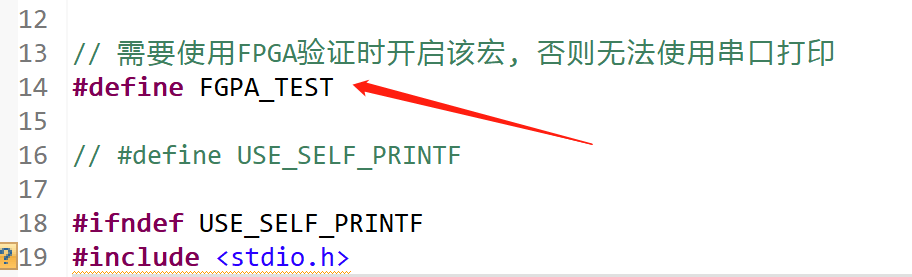
注意：在FPGA验证时还需要指定一个GPIO作为串口输出口，默认核0输出到GPIO\_29，核1输出到GPIO\_40。

StdOutInitA和StdOutInitB函数中存在对串口波特率的配置（默认为50Mhz环境中的921600）：



注意：在仿真阶段SciaRegs.DLL/ScibRegs.DLL参数配置不能低于2，否则会出现无法打印问题。

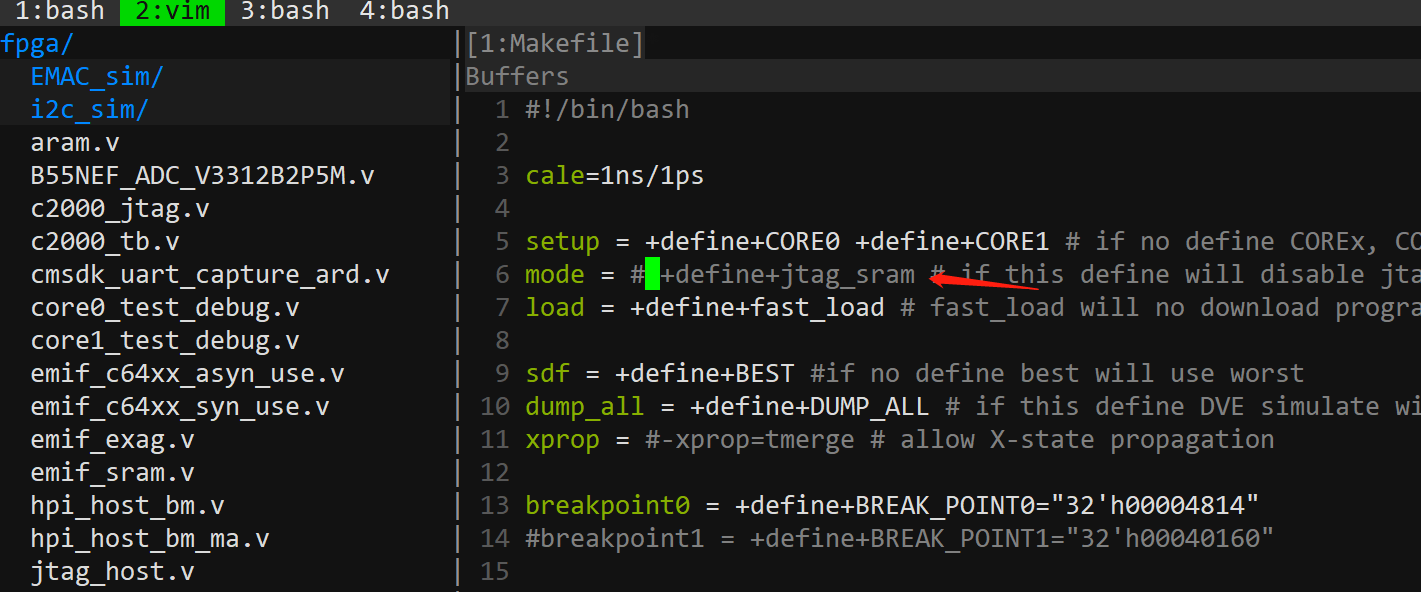
项目使用FPGA\_TEST宏作为真实串口打印的开关，默认关闭，需要时在common.h中开启：



## 全芯片带bootloader前仿

默认情况下make comp\_fullchip会使用jtags+快速载入方式实现最快的前仿流程，如果需要带bootloader 加 flash加载的全流程仿真（真实的芯片启动流程），需要通过如下方式进行：

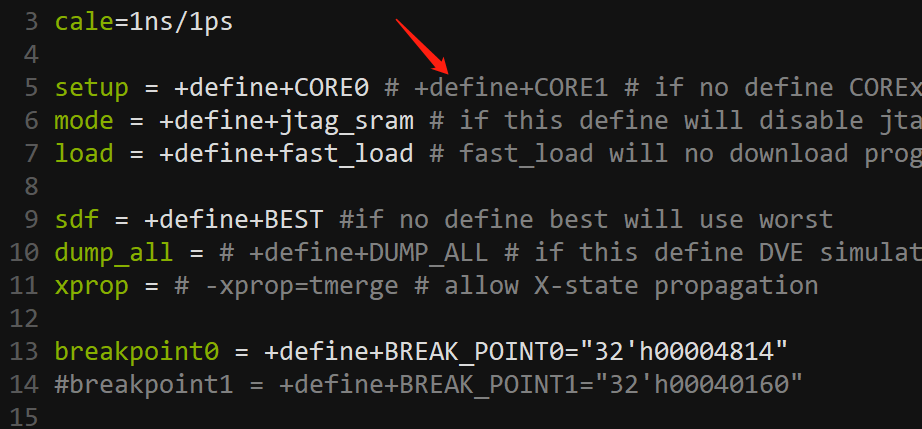
1. 修改Makefile，注释掉+define+jtag\_sram宏：



1. 重新使用make comp\_fullchip编译
2. 带bootloader前仿时不再需要outer\_data\_mem\_core0.dat、outer\_insn\_mem\_core0.dat、outer\_data\_mem\_core1.dat、outer\_insn\_mem\_core1.dat文件，取而代之的是init\_mem.txt，正确更新init\_mem.txt到qx\_c2000/vcs后make run即可仿真。

## 核1仿真开关

默认情况下，Makefile开启了CORE1的仿真支持，如果只对CORE0进行仿真测试，可以通过关闭CORE1仿真支持来实现加速，关闭方法为在Makefile中关闭+define+CORE1宏：

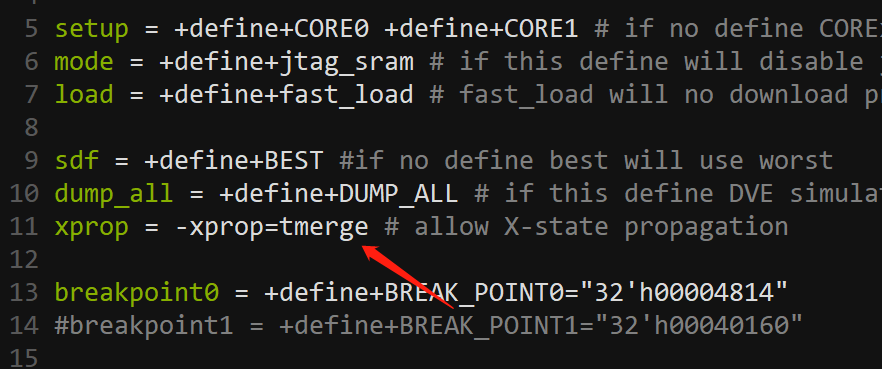


注意：CORE1宏只在全芯片快速前仿阶段起作用

## X态传播开关

在项目的前期，由于很多模块处于半成品状态，X态传播特性会导致各种不必要的麻烦，一般不建议打开，但在项目后期，所有模块稳定后，为避免X态导致的未知的问题，有必要对所有用例在开启X态传播开启的情况下做回归验证。

Makefile中X态传播特性默认处于关闭状态了，通过开启-xprop=tmerge宏后重新make comp\_fullchip编译即可实现开启X态传播特性情况下进行前仿：



## 门级网表前仿

在DC综合之后，PR布局布线之前，可以产生一个单文件门级网表（存放于libs/DC\_netlist/ c2000\_full\_top.mapped.v），为避免VCS前仿编译和DC编译流程之间的差异导致可能存在的问题，此阶段可以进行门级网表前仿，注意此阶段不带SDF时序反标文件（零延时仿真，因为在布局布线之前，hold没有修，所以正常会出现不少时序违例），仿真方法如下：

1. 通过make comp\_post\_dc命令编译RTL代码
2. 带网表仿真时只支持bootloader+flash全流程方式仿真，因此需要注意更新init\_mem.txt，并更新init\_mem.txt后通过make run即可开始门级网表前仿。

DC阶段还会加入DFT的支持，如果希望带着DFT信息进行网表前仿，则需要使用make comp\_post\_dc\_dft命令来完成编译。

注意：

1. 门级网表前仿比较耗时（一次运行在半个小时以上），为加快仿真速度，一个模块建议挑选典型功能用例并将这些功能用例全部一次性仿真完成，多个用例同时运行可能存在上一个用例的配置影响下一个用例的情况下，建议在每个用例前面添加模块复位或关键参数还原等操作以规避用例间的影响。

## 门级网表后仿

在DC综合和PR布局布线之后可以产生一个SDF文件和一个全量的网表级RTL文件（存放于libs/PR\_netlist目录），SDF（Standard Delay Format）文件是把布局布线过程中器件延时和线延时的信息保留下来，据此就可以在路径时序分析时将整条路径的时序计算出来，再判断时序约束条件就能知道是否满足时序要求。SDF文件包括设计中每个单元的时序信息，基本的时序数据由IOPATH INTERCONNECT SETUP HOLD时序组成。仿真方法如下：

1. 通过make comp\_post\_pr命令编译RTL代码
2. 带网表仿真时只支持bootloader+flash全流程方式仿真，因此需要注意更新init\_mem.txt，更新init\_mem.txt后通过make run即可开始门级网表后仿。

注意：

1. 门级网表后仿比较耗时（一次运行在半天以上），为加快仿真速度，一个模块建议挑选典型功能用例并将这些功能用例全部一次性仿真完成，多个用例同时运行可能存在上一个用例的配置影响下一个用例的情况下，建议在每个用例前面添加模块复位等操作以规避用例间的影响。
2. 关闭核1加快仿真方法参见“门级网表前仿”章节。
3. 通过make comp\_post\_pr\_notime命令可以实现后仿阶段去掉SDF时序反标文件的引入，此时将不带时序信息仿真，可以将仿真时间提升到门级前仿水平，此时和门级前仿的区别是网表文件是经过布局布线之后的结果，更加接近真实的物理电路。

## 使用verdi替代DVE仿真

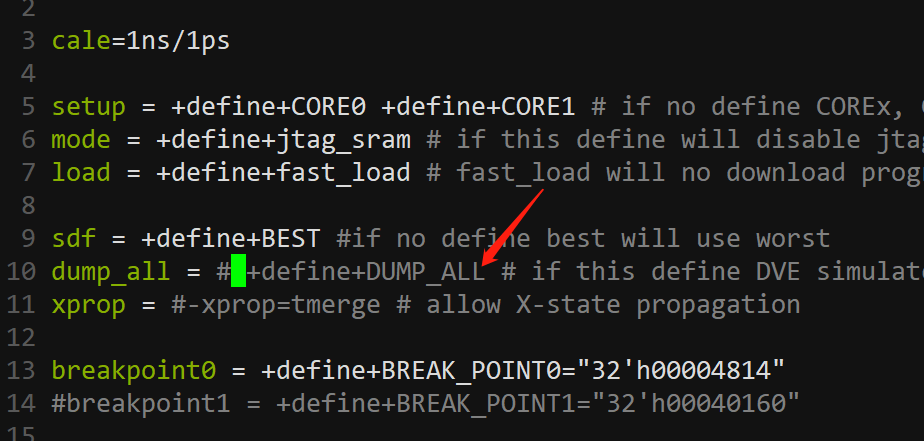
默认情况下，VCS使用DVE作为波形查看界面完成仿真流程，qx\_c2000支持通过verdi替代DVE的方式完成仿真，如果希望使用verdi来仿真，可以通过如下流程实现：

1. 通过make comp\_verdi替代make comp\_fullchip完成RTL编译；
2. 通过make runv替代make run完成VCS启动并仿真，此时将调用verdi替代DVE实现仿真。

## 全量DUMP开关

默认情况下，为方便仿真波形的查看，无论是DVE还是verdi仿真都默认开启了仿真波形的全量dump特性，使用DVE时，输出的波形文件存放于qx\_c2000/vcs/inter.vpd文件中，使用verdi时存放于qx\_c2000/vcs/test.fsdb文件中。全量dump特性绝大多数情况下对仿真速度的影响很小，在部分特殊场景中如果希望关闭该特性来加快仿真，可以通过如下方式实现关闭：

如果使用DVE仿真，通过Makefile中禁用+define+DUMP\_ALL宏来关闭:



如果使用verdi，一般不建议关闭全量dump特性，如果非要关闭可以编辑fpga/c2000\_tb.v来实现关闭（注释掉红框中的代码即可）：

文本

描述已自动生成