RISC-V微处理器芯片设计实践

本实践完成RISC-V微处理器IP的仿真、工艺移植、逻辑综合和后仿真，以上内容构成了数字集成电路前端设计的主要内容，旨在帮助学生了解数字芯片设计流程，同时对RTL代码、标准单元库、时序分析等内容建立认识。

1. LAB1：搭建环境
   1. 准备流片数据

包括RISC-V处理器IP（RTL代码和仿真环境）、工艺库数据（标准单元库、IO库和存储器）、虚拟机和EDA。具体如下：

1. RISC-V处理器IP，包括RTL代码和仿真环境：.\e203\_hbirdv2-master.zip
2. 标准单元库：SCC018UG\_UHD\_RVT，版本V0.4，目录：.\SCC018UG\_UHD\_RVT\_V0p4a.zip
3. IO库：SP018，版本1.5b，目录：.\SP018\_V1.5b.zip
4. 存储器IP：.\MEM\_IP\_20MHz.zip
5. IC\_EDA\_XXXX虚拟机
   1. 建立虚拟机目录结构

文件夹命名解释：lib文件夹集中存放各个Fab不同节点的工艺数据。risc-v是projects文件夹中众多项目中的一个。20241209\_V1.0或20250105\_V2.0表示risc-v处理器的不同版本，例如不同存储器容量版本。data\_org是IP数据在虚拟机的备份。pre\_sim\_org是不修改IP功能，在本地搭建前仿真环境，复现IP功能。pre\_sim对RTL代码修改，将RTL移植至目标工艺，进行前仿真确保IP功能正确。syn逻辑综合文件夹。post\_sim后仿真目录。

虚拟机目录结构如下：

/home/ICer/projects

|----lib

|----SMIC180

|----MEM\_IP\_20MHz //MemroyCompiler生成的存储器IP

|----SCC018UG\_UHD\_RVT\_V0p4a //标准单元库

|----SP018\_V1.5b //IO库

|----risc-v

|----20241209\_V1.0\_groupxx //基于SMIC180nm的版本，groupxx表示第几小组

|----data\_org //处理器原始数据：代码、程序等

|----pre\_sim\_org //前仿，和工艺无关，即搭建代码仿真环境，确保数据完整。

|----rtl //rtl代码

|----tb //testbench

|----riscv-tools //处理器测试程序，直接拷贝e203\_hbirdv2-master/riscv-tools/

|----pre\_sim //前仿，将RTL移植至目标工艺，进行仿真保证功能正确。

|----rtl

|----tb

|----riscv-tools

|----syn //综合

|----scripts //脚本

|----netlist //网表

|----reports //报告

|----results //输出

|----post\_sim //后仿真，对综合后的网表进行功能仿真，不检查时序

|----tb

|----riscv-tools

|----20250105\_V2.0 //举例：ITCM为4K版本

* 1. 数据导入虚拟机

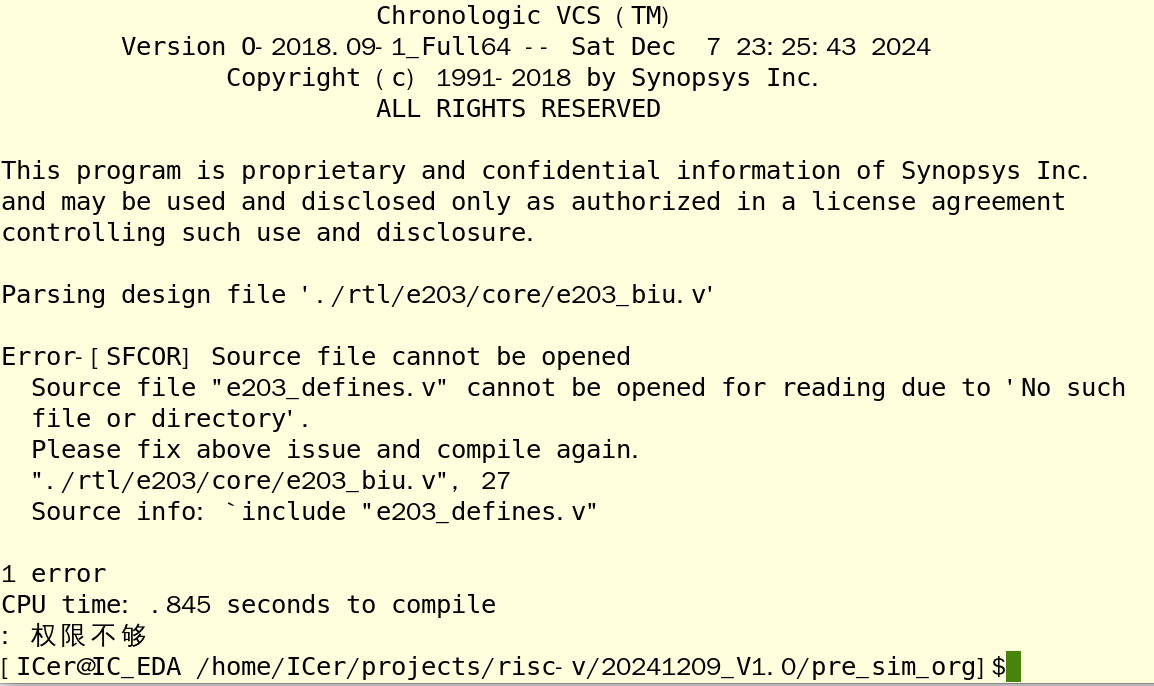
1. 将e203\_hbirdv2-master.zip、SP018\_V1.5b.zip和MEM\_IP\_20MHz.zip拷贝并解压在./projects/lib/SMIC180/目录下，这些工艺文件只读，不可修改。
2. 将e203\_hbirdv2-master.zip拷贝至项目目录下./projects/risc-v/20241209\_V1.0\_groupxx/data\_org/，并解压。
3. LAB2：pre\_sim\_org前仿真

当第三方IP设计数据移植至本地虚拟机后，建立pre\_sim\_org目录搭建本地仿真环境，这个环节只是为了搭建IP的VCS仿真环境，无需对RTL代码本身逻辑功能做修改。

* 1. pre\_sim\_org目录

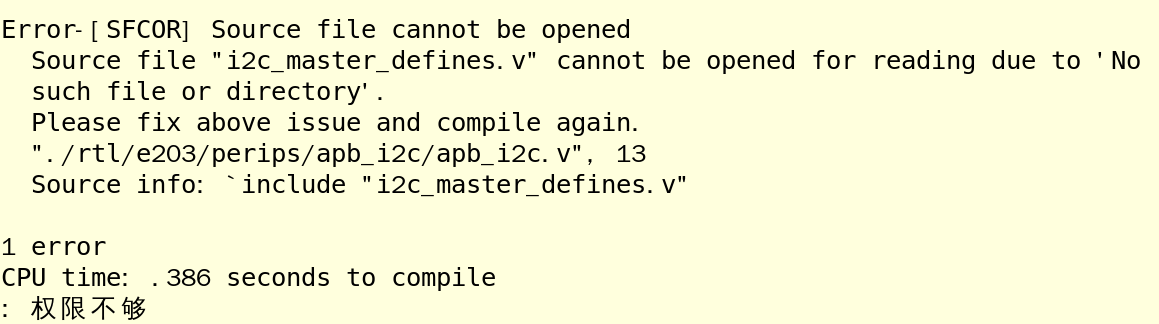
pre\_sim\_org目录下数据包括：

1. ./rtl/ 设计源代码所在目录，从./data\_org/e203\_hbirdv2-master/rtl/拷贝过来；
2. ./riscv-tools/ 处理器测试程序所在目录，直接从./data\_org/e203\_hbirdv2-master/riscv-tools/拷贝过来，./riscv-tools/riscv-tests/isa/generated/目录下有各类指令的测试程序。
3. ./tb/ testbench目录，直接从./data\_org/e203\_hbirdv2-master/tb/拷贝过来。
4. ./e203\_filelist.f 列出仿真所需要的所有rtl代码和testbench文件。
5. ./run\_vcs.csh VCS仿真命令脚本，VCS是RTL仿真工具。
6. ./run\_verdi.csh Verdi命令脚本，Verdi是波形查看工具。
   1. 前仿真
7. 打开terminal，进入pre\_sim\_org文件夹，执行source run\_vcs.csh，提示以下错误，请根据提示自己去分析原因。

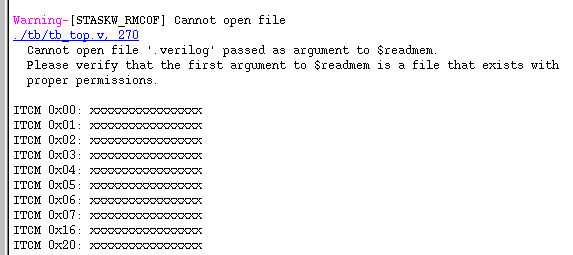


为了减小代码修改工作，将./rtl/e203/core/config.v和e203\_defines.v这两个文件剪切至./pre\_sim\_org/目录下。e203\_filelist.f文件中则无需再列出config.v和e203\_defines.v。

以下error按照同样方式处理，即xx\_define.v文件拷贝至./pre\_sim\_org目录，e203\_filelist.f文件不再列出该xx\_define.v。这样的好处是，不需要修改rtl代码。



1. 再执行source run\_vcs.csh，VCS工具启动，点击start，提示以下错误，原因是调用测试程序时出错。



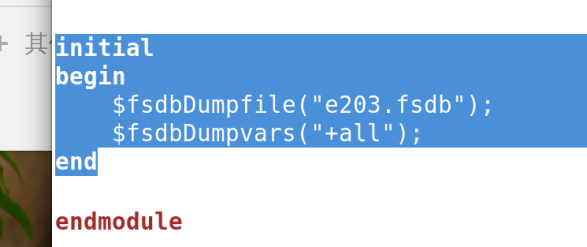
将tb\_top.v的line270改为如下即可，替换不同的.verilog文件，对不同指令进行测试。VCS界面会提示PASS或者FAIL，以此判断仿真是否正确。

1697681664065

指令集仿真正确时，VCS窗口显示如下信息。（思考题一：打印出PASS的条件是什么？结合tb\_top.v和测试程序去分析。）



1. 在tb\_top.v文件最后加入以下语句，仿真结束后会生成波形文件.fsdb。关闭VCS后，source run\_verdi.csh，可以启动Verdi工具查看波形。



* 1. 回答思考题一
  2. 学习要求

1. 熟悉虚拟机环境；
2. 熟悉VCS工具和仿真环境，熟悉Verdi工具；
3. 了解处理器基本工作原理，理解仿真过程。
4. LAB3：pre\_sim例化存储器IP

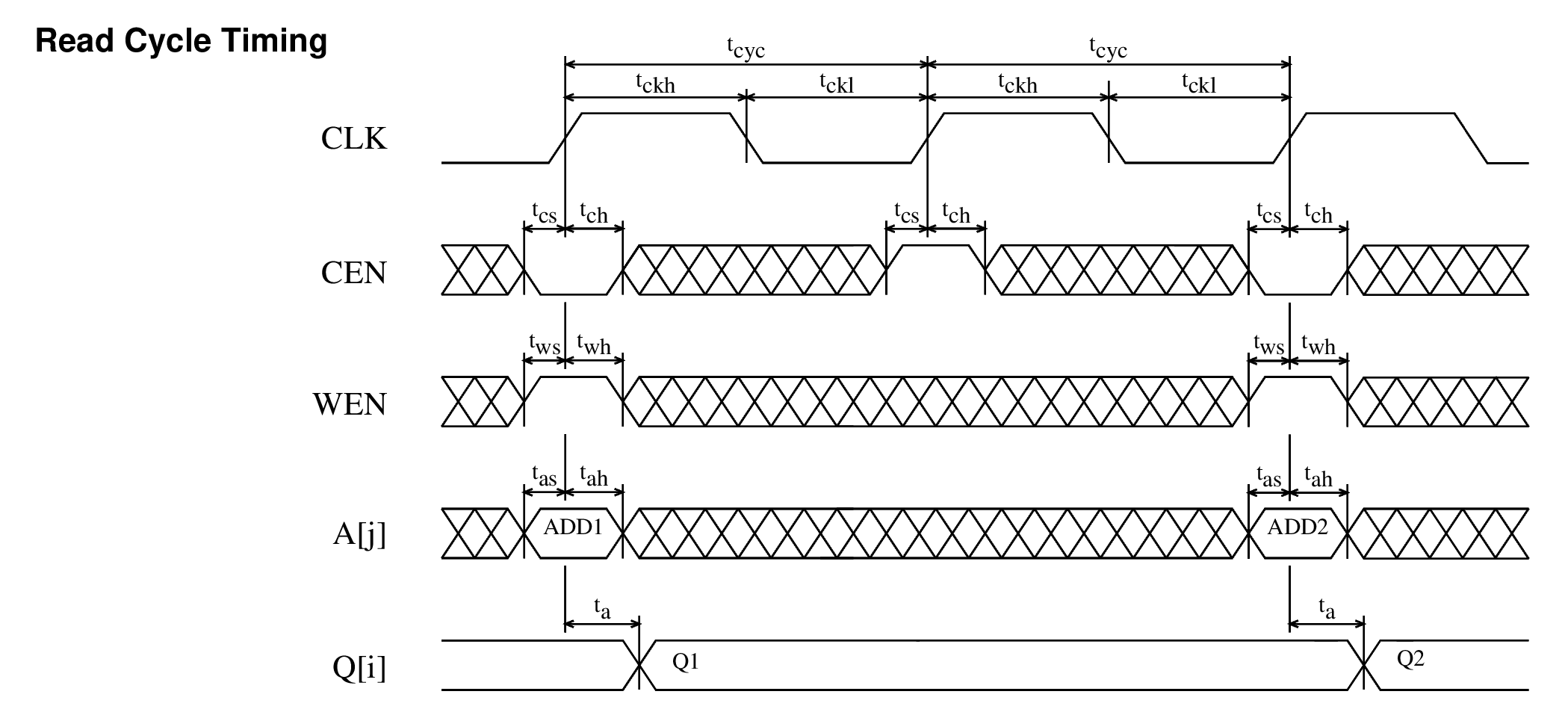
pre\_sim在pre\_sim\_org基础上，对RTL进行修改，目的是将设计移植至目标工艺。注意做好文档修改记录。建pre\_sim目录，拷贝pre\_sim\_org中的rtl, riscv-tools和tb文件夹过来。拷贝e203\_filelist.f, run\_vcs.csh和run\_verdi.csh过来。拷贝config.v, e203\_defines.v和ic\_master\_defines.v过来。

RTL代码修改的目的：用SMIC180nm工艺库中的存储器、门控时钟和IO单元替换RTL中的行为级描述。pre\_sim用来仿真验证工艺移植后，处理器功能是否正确。

* 1. 例化存储器

1. 存储器IP数据。数据目录./lib/SMIC180/MEM\_IP\_20MHz/文件夹中有两类不同容量的存储器，分别用作程序存储器ITCM和数据存储器DTCM。数字前端会用到的存储器数据包括.ps, .v, .lib。
2. 存储器IP *vs* 存储器行为级模型。分析存储器IP和行为级模型的差异？去读ra1shd.pdf 手册，去读sirv\_sim\_ram.v代码，对比手册和代码，结合仿真。

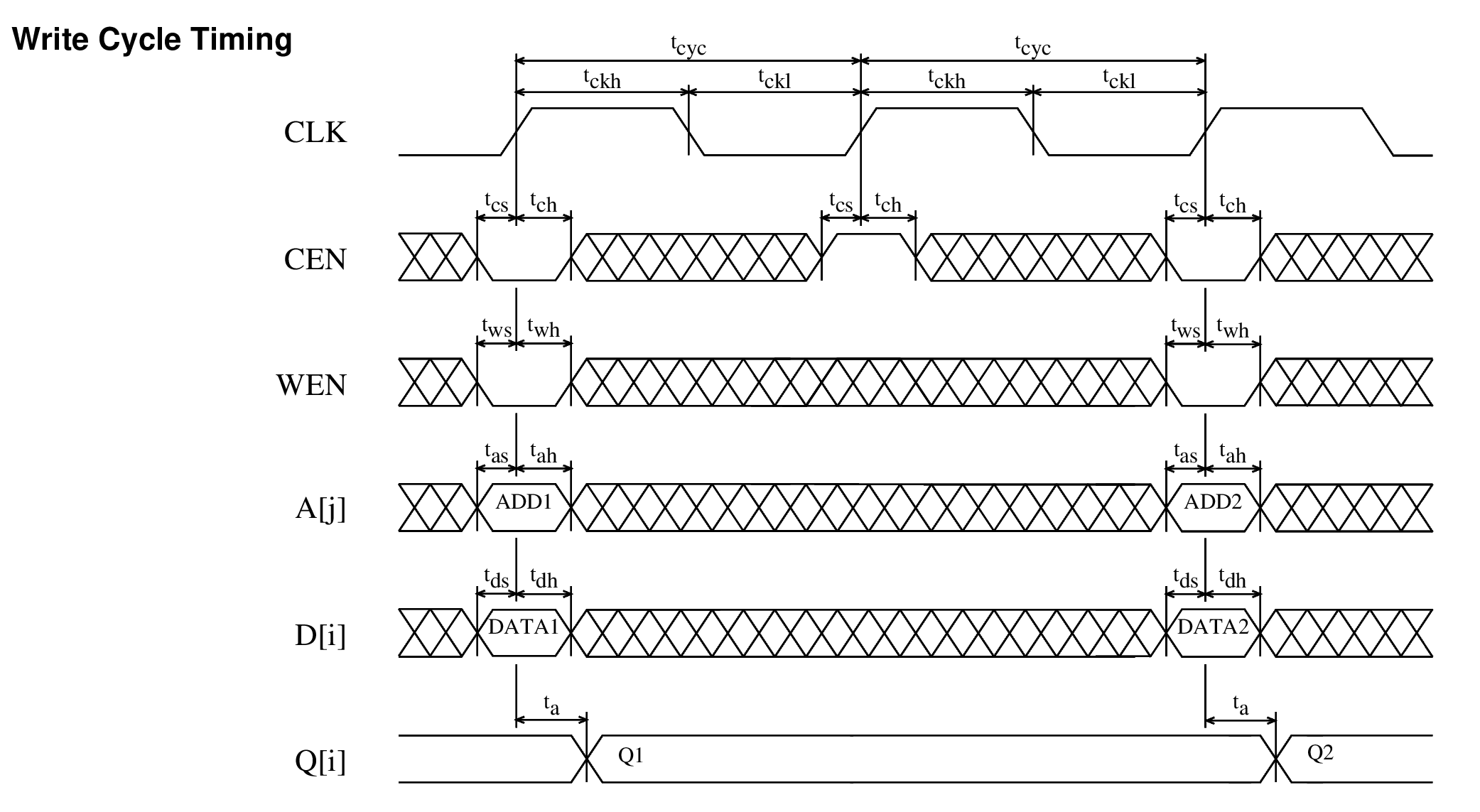
**存储器读操作：**



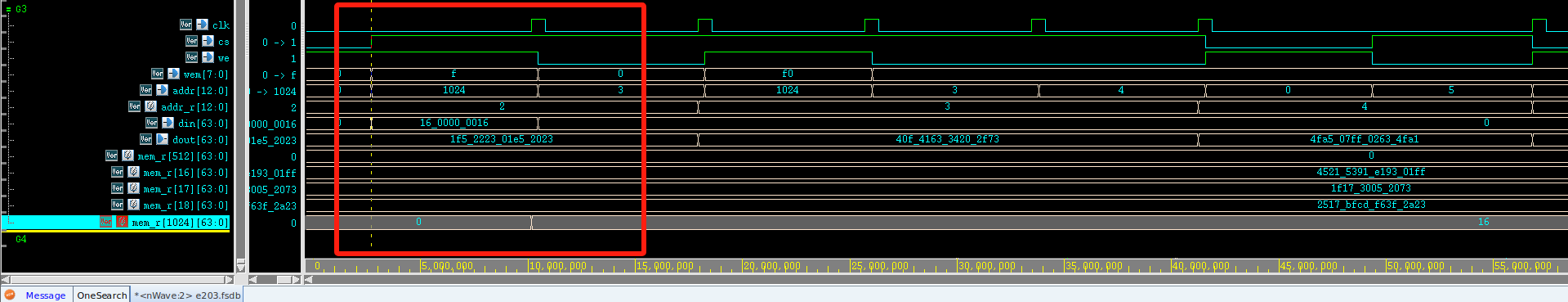
by .ps（手册）

使用sirv\_sim\_ram.v代码行为级仿真结果

**存储器写操作：**



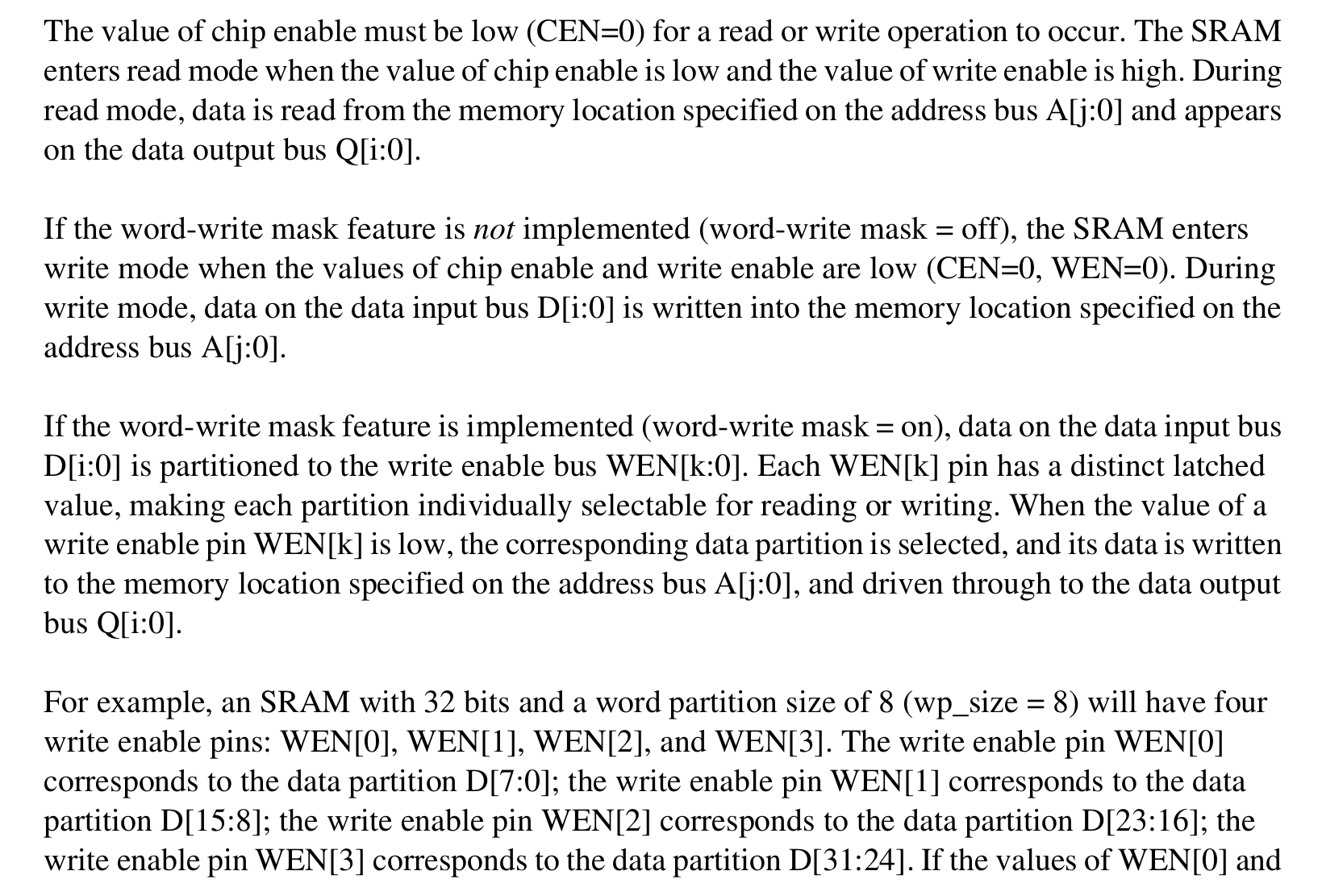
by .ps（手册）

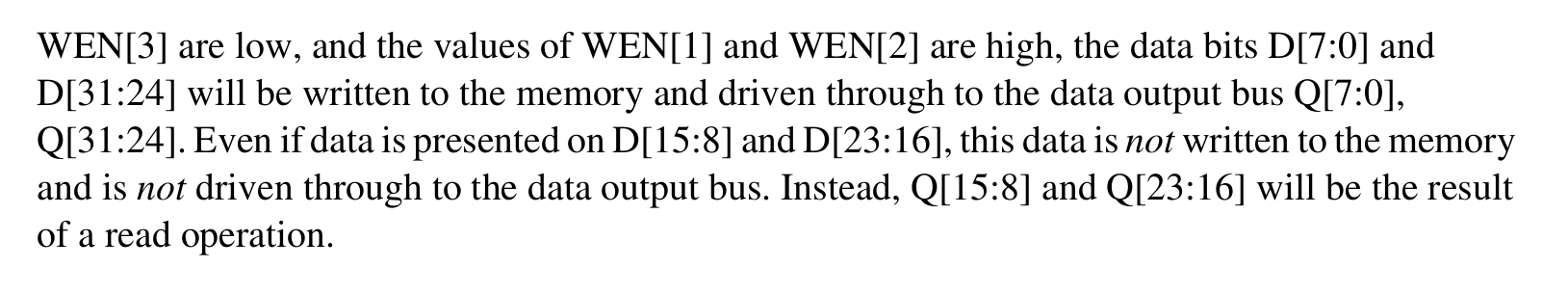


使用sirv\_sim\_ram.v代码行为级仿真结果

**OEN功能：**When OEN is active(low) the data out Q is valid. When OEN is inactive(high) then the data out Q is undriven (high-Z)

**word-write mask：**





——ra1shd.pdf

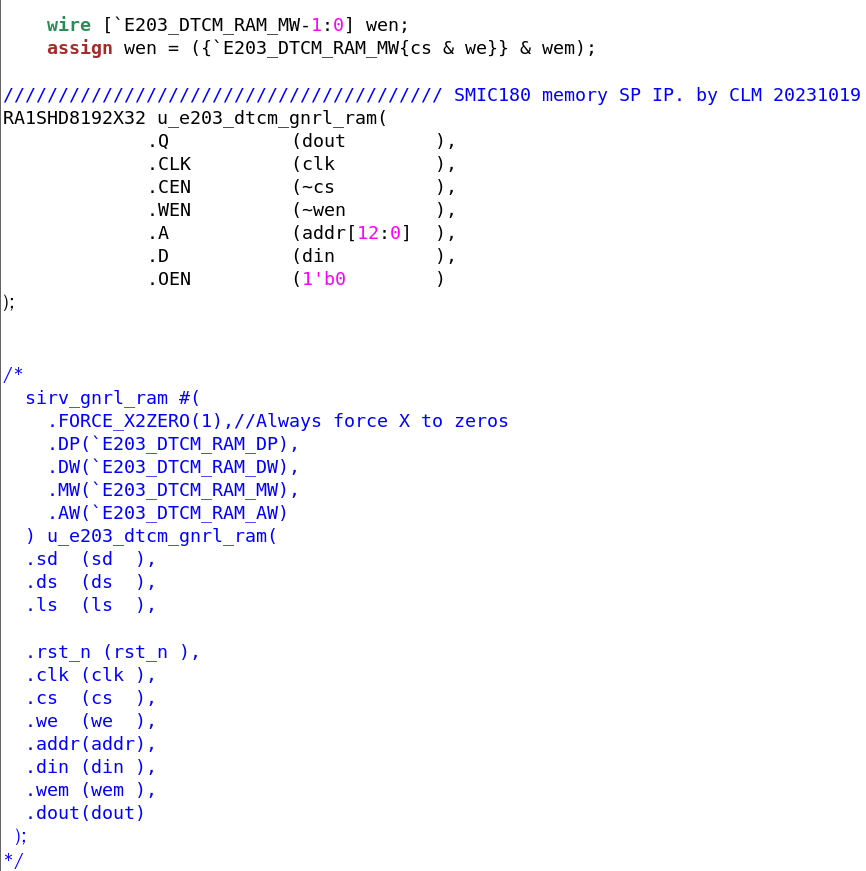
所以，存储器IP和行为级模型的差异是：a）片选信号和写信号的有效电平是反的；b）存储器IP有OEN，行为级模型没有；c）存储器IP是写穿透write through，而行为级模型不是。存储器的Write through是指存储器发生写操作时，写入的值也会出现在Q端。行为级模型中，写地址是addr，读地址是addr\_r。

1. RTL修改。e203\_itcm\_ram.v和e203\_dtcm\_ram.v，将存储器行为级模型，改为例化存储器IP。

e203\_itcm\_ram.v修改如下：



e203\_dtcm\_ram.v修改如下：

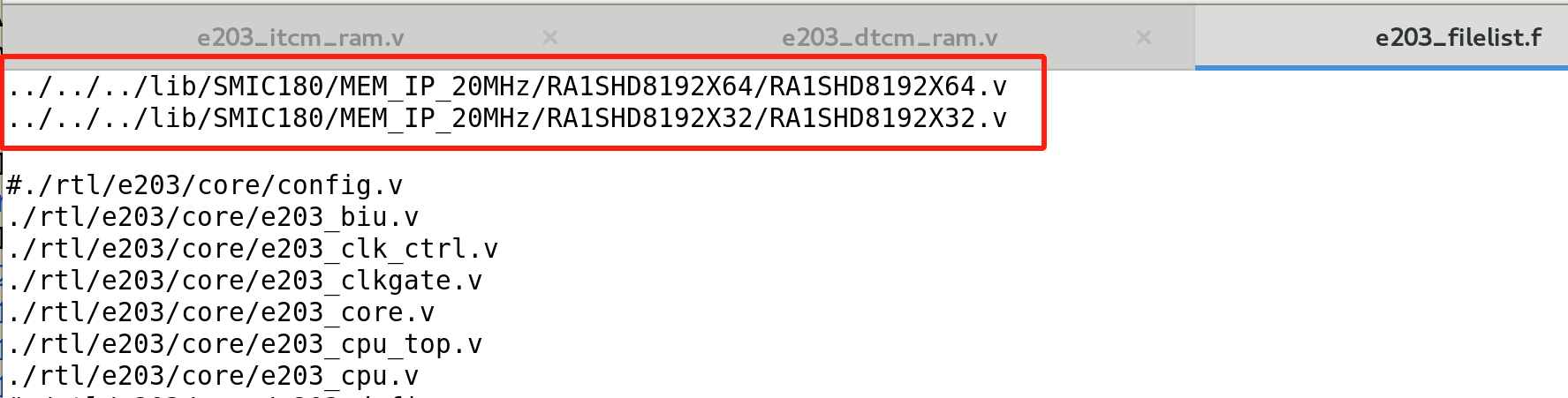


（思考题二：以上e203\_itcm\_ram.v和e203\_dtcm\_ram.v，将存储器行为级模型替换为存储器IP，是否改变原有设计的逻辑功能，为什么？）

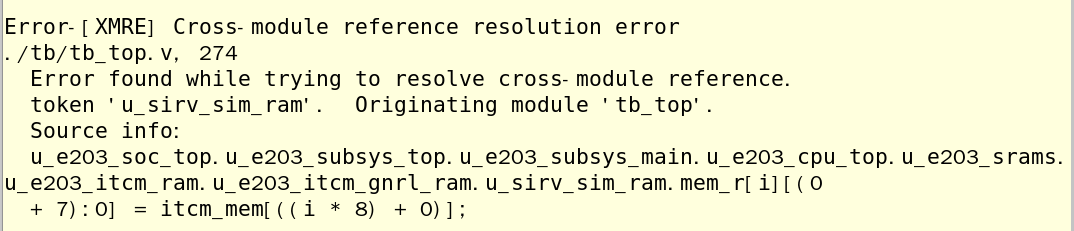
* 1. 仿真环境修改

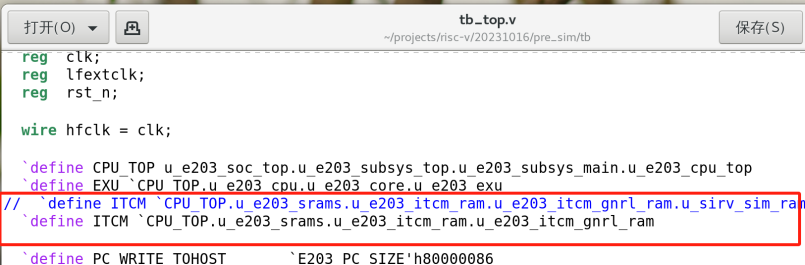
e203\_itcm\_ram.v和e203\_dtcm\_ram.v换成SMIC180nm工艺下的存储器IP后，要进行全面的功能验证，包括观察波形，以及大量测试程序仿真。首先对仿真环境做修改，包括：

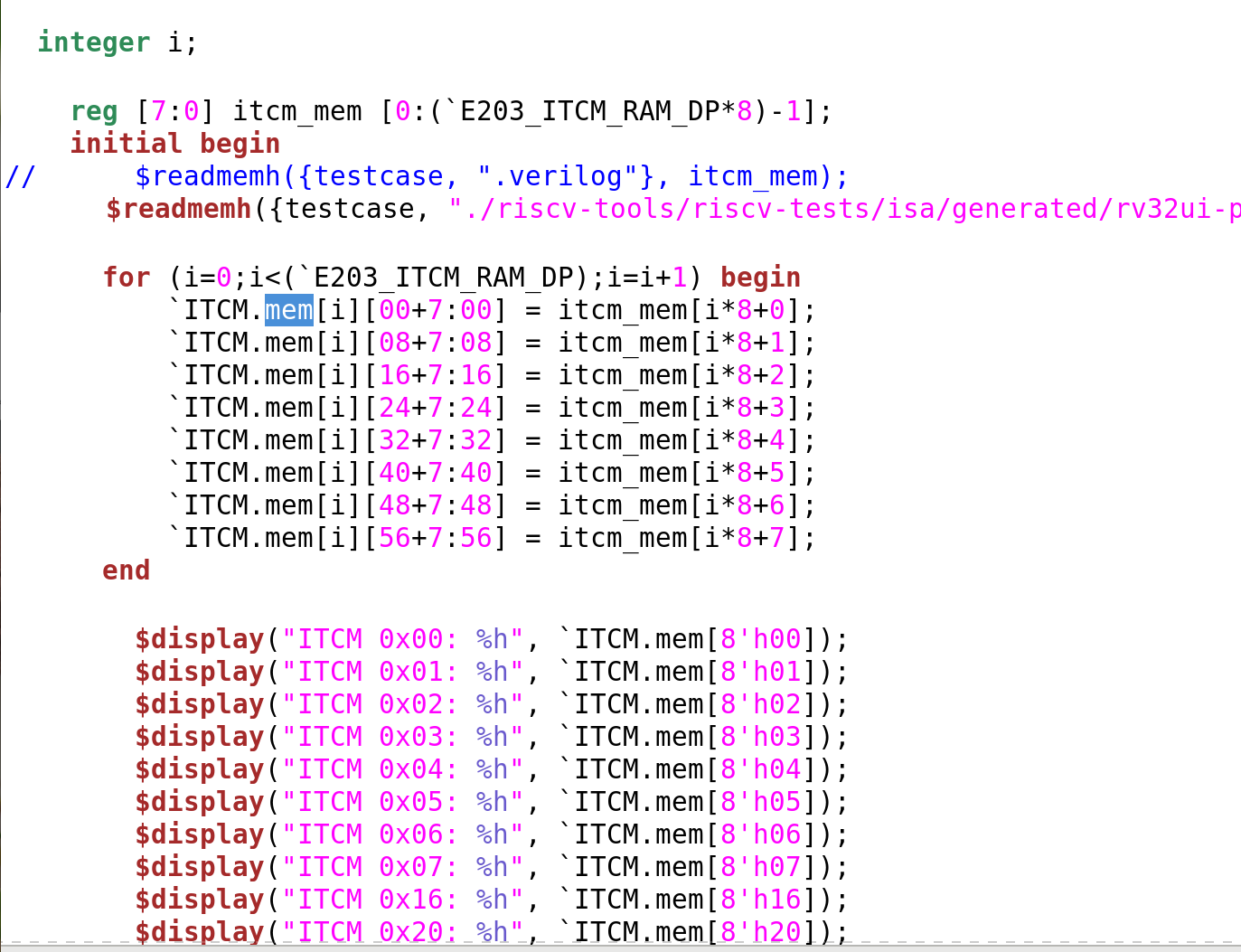
e203\_filelist.f把存储器IP的.v（存储器仿真模型）加进来，如下。



tb\_top.v中要做如下修改（要理解这些修改的原因）。



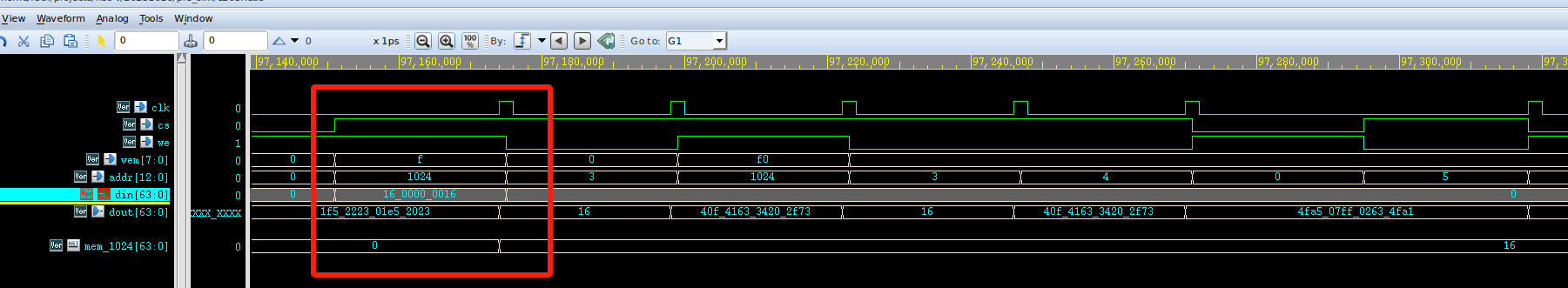




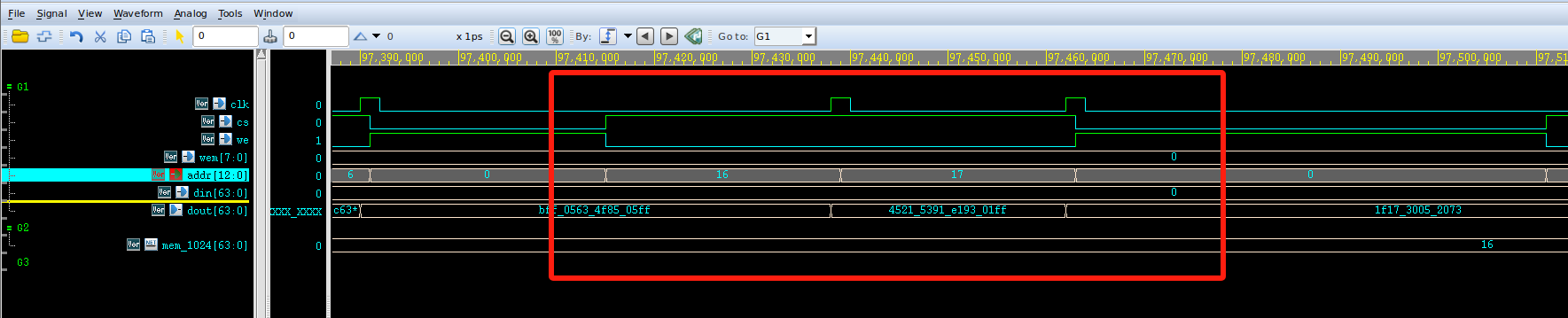
* 1. 观察仿真波形

观察存储器接口信号波形，验证读写是否正确

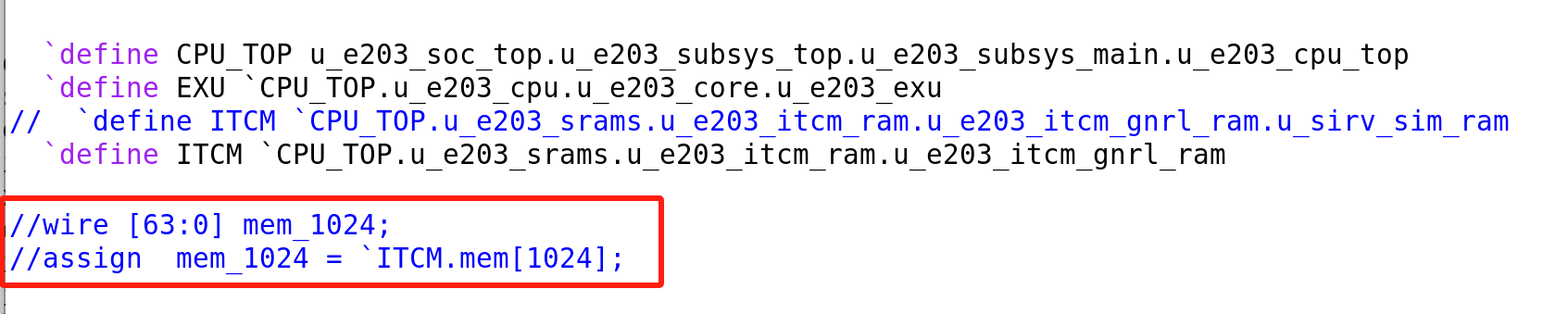
写操作：mem\_1024数据被更新，同时看到了write through



读操作：地址16和17的数据被正常读出。

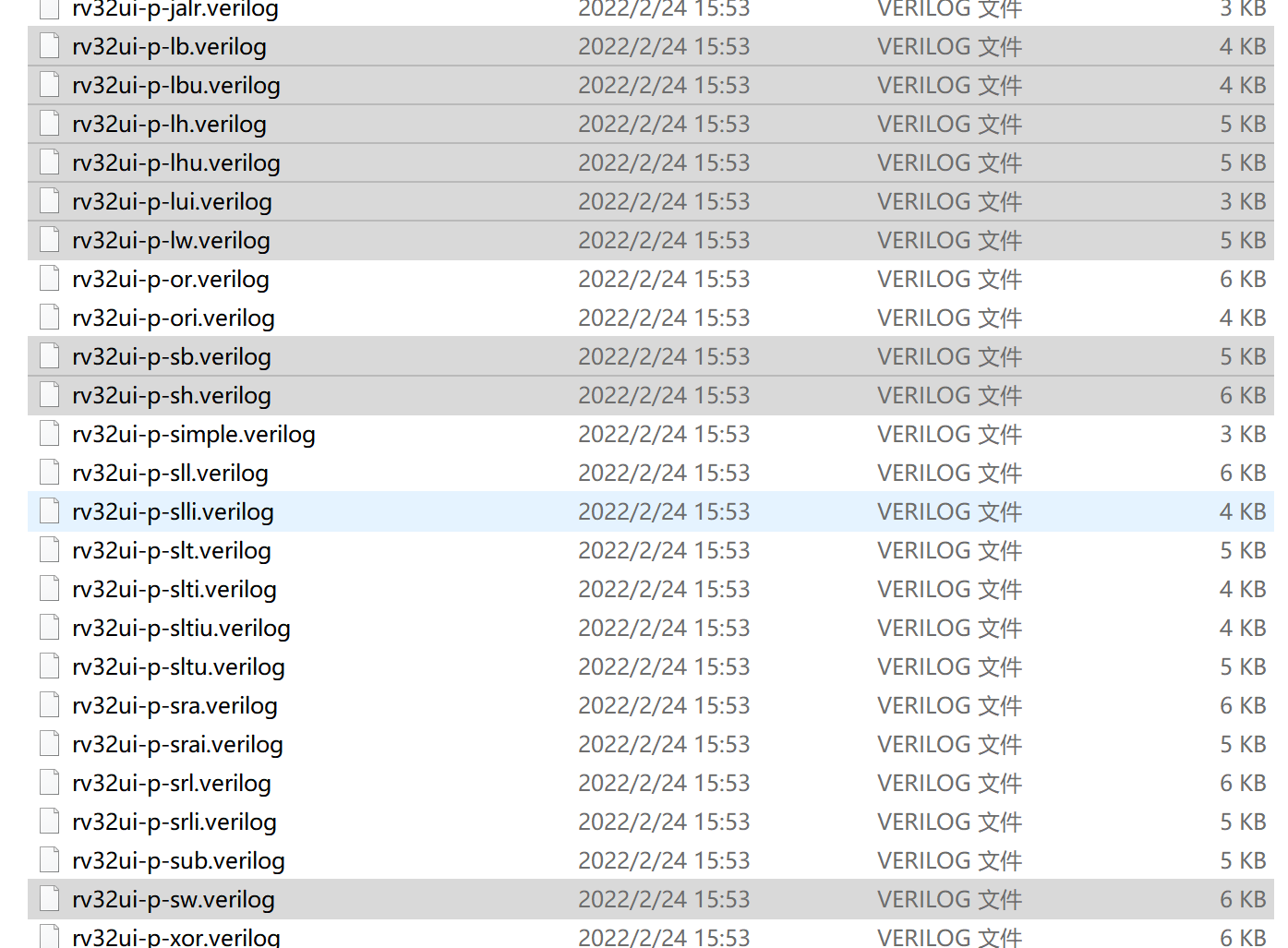


看波形时需要看存储器内部的mem阵列，数据太大，工具不支持，可以在tb\_top.v中这样操作，就可以观察mem阵列的第1024个数据。



* 1. 存储器测试程序

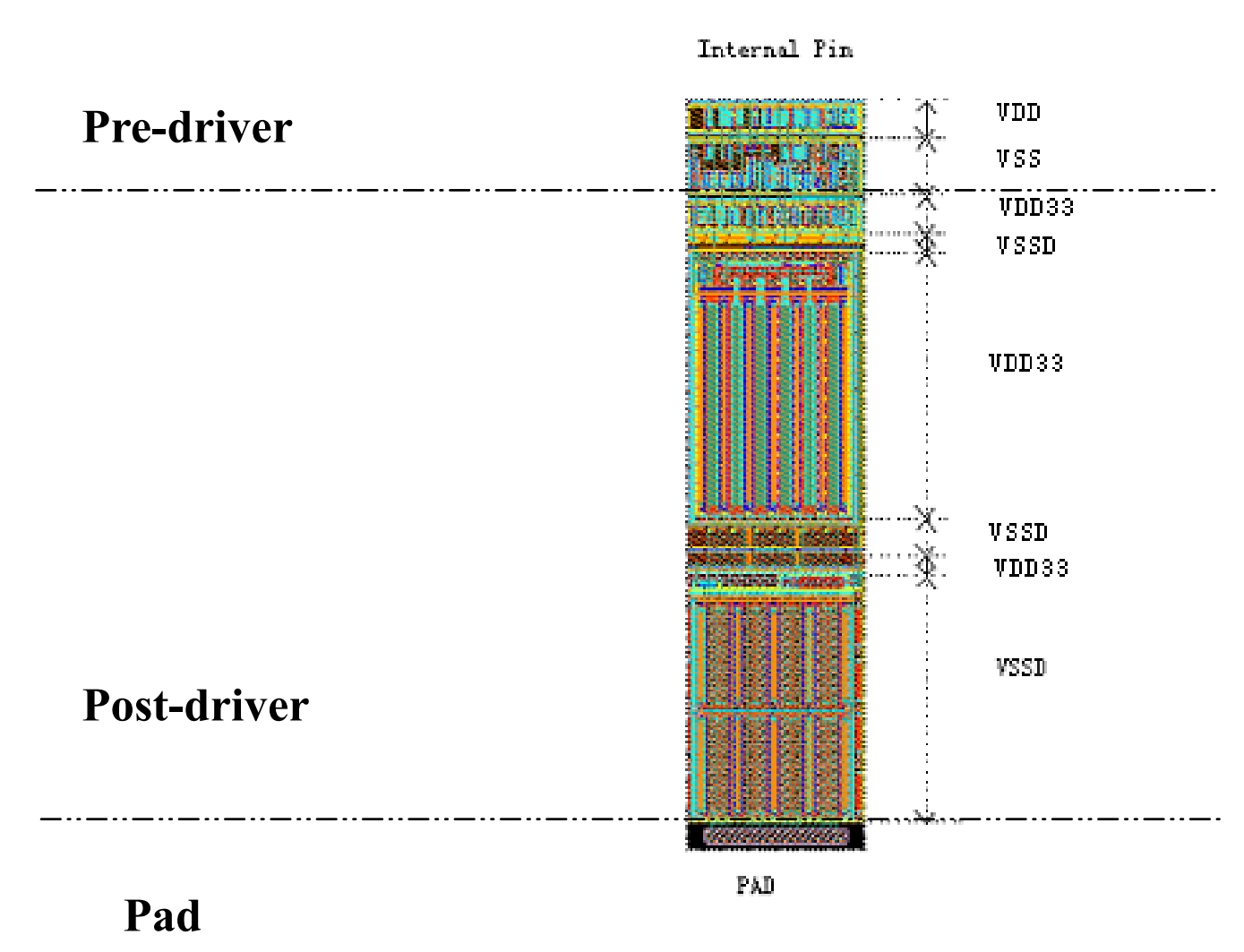
更换tb\_top.v中的测试程序，专门测试存储器的读写功能。这类测试程序都能够显示PASS，说明存储器IP例化正确。



* 1. 回答思考题二
  2. 学习要求

1. RTL代码修改时，先想好，再仔细做，最后做好验证，形成闭环。
2. 掌握CMOS工艺下存储器的类型、工作原理、读写时序。
3. RTL代码版本管理。将当前pre\_sim文件夹另存为pre\_sim(memory)，作为例化存储器IP后的版本暂存。
4. LAB4：pre\_sim增加full\_chip.v顶层

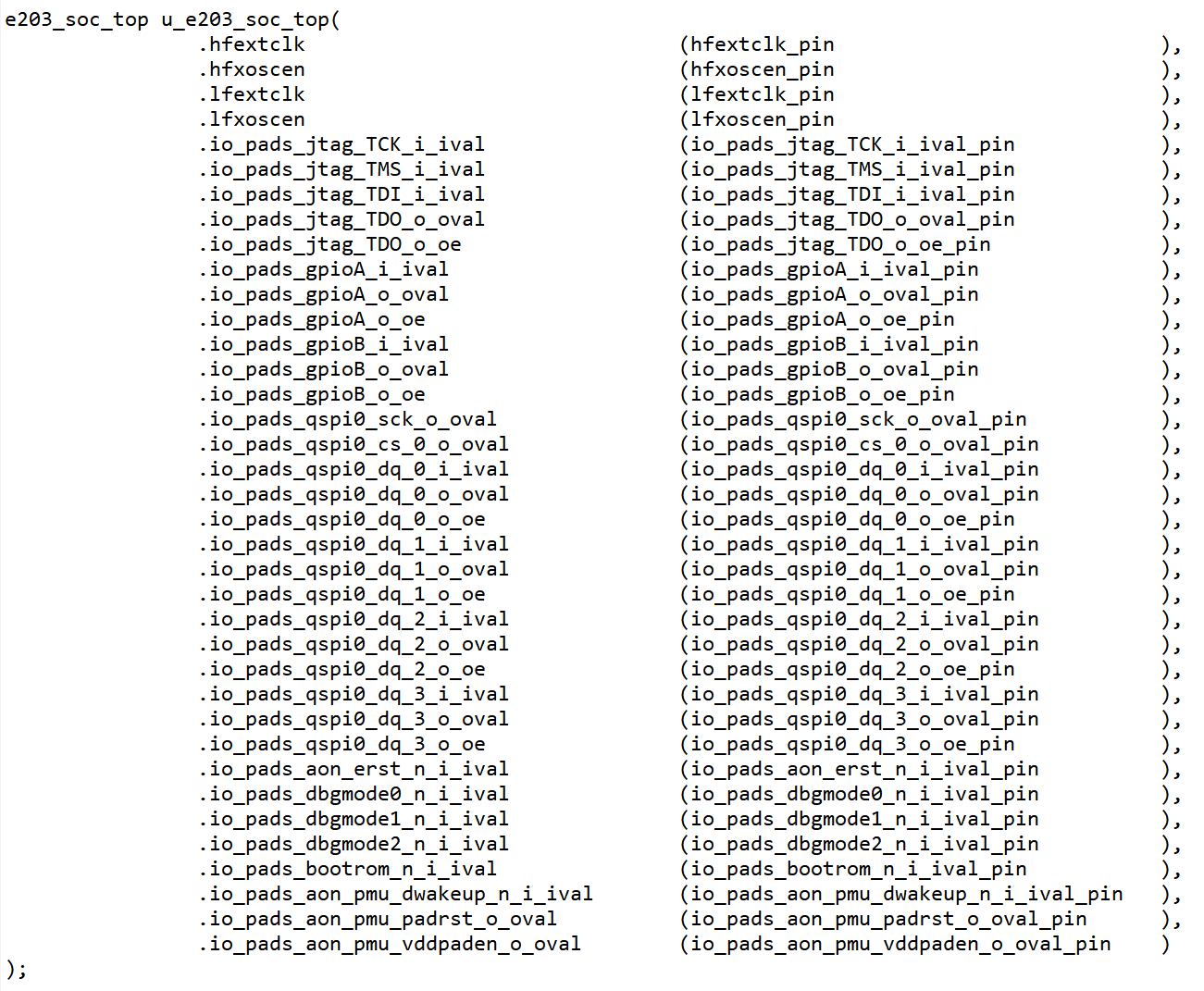
RISC-V代码的最顶层是e203\_soc\_top，不包含IO，即顶层所有信号都是内部pin，需要通过IO cell才可以和芯片外部进行互连。IO的作用：完成core电平和IO电平转换，例如core是1.8V，IO是3.3V；完成特殊逻辑功能，例如双向IO、三态IO；提供ESD保护。读IO手册《SMIC\_SP018\_IO\_DataBook\_Ver1p3.pdf》和《SMIC\_Standard\_IO\_Application\_Note\_Ver3p0.pdf》



增加full\_chip.v模块，例化IO cell和e203\_soc\_top模块，即将e203\_soc\_top模块的所有输入输出外围包一圈IO。IO选型就是选择：输入、输出、双向、是否需要上拉或下拉、输出IO的驱动能力等，看手册和.v了解IO功能。

* 1. 增加full\_chip.v文件

增加full\_chip.v文件，放在./rtl/e203/soc/目录下，例化e203\_soc\_top模块及信号命名如下：



* 1. 例化IO cell

1. hfextclk：输入，16MHz时钟，接有源晶振输出。

选择PI（Input pad），不要带上拉/下拉，不要施密特。测试时直接从hfextclk输入16MHz测试时钟，hfxoscen不使用。真实应用中，板上使用有源晶振，通过hfxoscen控制有源晶振是否输出时钟给hfextclk。此处不使用无源晶振，原因：IO要使用晶振专用IO（PXWE），没用过，功能和物理设计上有风险。另外无源晶振需要外围电路配合，不熟悉，也不是我们的重点。

PI u\_pad\_hfextclk ( .PAD(hfextclk), .C(hfextclk\_pin) );

1. hfxoscen：输出，接有源晶振输出使能端。

选择PO（CMOS output pad），用作控制信号，驱动能力选择4.

PO4 u\_pad\_hfxoscen ( .I(hfxoscen\_pin), .PAD(hfxoscen) );

1. lfextclk, lfxoscen与hfextclk, hfxoscen类似

PI u\_pad\_lfextclk ( .PAD(lfextclk), .C(lfextclk\_pin) );

PO4 u\_pad\_lfxoscen ( .I(lfxoscen\_pin), .PAD(lfxoscen) );

1. io\_pads\_jtag\_TCK\_i\_ival：输入，时钟，频率MHz级别。

选择PIU（Input pad with pull-up），上拉的原因是非JTAG模式时，该端口悬空即可，同时上拉电阻不会对JTAG时钟产生影响，因为频率很低。

PIU u\_pad\_io\_pads\_jtag\_TCK\_i\_ival ( .PAD(io\_pads\_jtag\_TCK\_i\_ival), .C(io\_pads\_jtag\_TCK\_i\_ival\_pin) );

1. io\_pads\_jtag\_TMS\_i\_ival和io\_pads\_jtag\_TDI\_i\_ival：输入，数据。

选择PIU（Input pad with pull-up），上拉的原因是非JTAG模式时，该端口悬空即可。

PIU u\_pad\_io\_pads\_jtag\_TMS\_i\_ival ( .PAD(io\_pads\_jtag\_TMS\_i\_ival), .C(io\_pads\_jtag\_TMS\_i\_ival\_pin) );

PIU u\_pad\_io\_pads\_jtag\_TDI\_i\_ival ( .PAD(io\_pads\_jtag\_TDI\_i\_ival), .C(io\_pads\_jtag\_TDI\_i\_ival\_pin) );

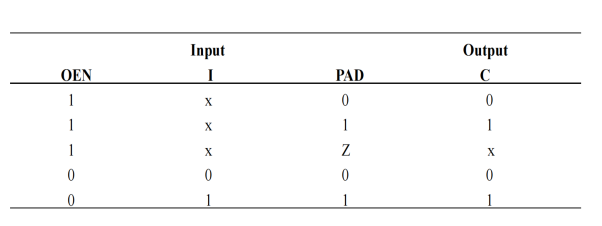
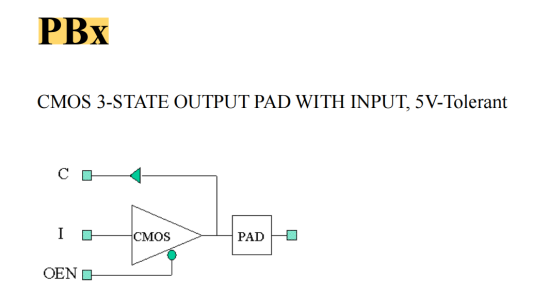
1. io\_pads\_jtag\_TDO\_o\_oval和io\_pads\_jtag\_TDO\_o\_oe打包成一个三态输出。

选择POT16（CMOS 3-state output pad），当oe为高时，输出有效；当oe为低时，开漏。注意io\_pads\_jtag\_TDO\_o\_oe是高有效，而IO是OEN为低时输出有效，所以要反向。驱动选16，要和上位机通信。

POT16 u\_pad\_io\_pads\_jtag\_TDO\_o (.OEN(~io\_pads\_jtag\_TDO\_o\_oe\_pin), .I(io\_pads\_jtag\_TDO\_o\_oval\_pin), .PAD(io\_pads\_jtag\_TDO\_o) );

1. io\_pads\_gpioA\_i\_ival，io\_pads\_gpioA\_o\_oval和io\_pads\_gpioA\_o\_oe打包成一个双向端口，命名为io\_pads\_gpioA。

选择PB8（CMOS 3-state output pad with input），OEN为低时，输出有效PAD=I；OEN为高时，PAD为开漏输出；输入C=PAD。不需要上拉和下拉，驱动选择8。未来有可能接串行总线或者ADC数据总线。代码中GPIO模块复位后是开漏输出，io\_pads\_gpioA\_o\_oe为1时，输出有效。



PB8 u\_pad\_gpioAX (.OEN(~io\_pads\_gpioA\_o\_oe\_pin[X]), .C(io\_pads\_gpioA\_i\_ival\_pin[X]), .PAD(io\_pads\_gpioA[X]), .I(io\_pads\_gpioA\_o\_oval\_pin[X]));

1. io\_pads\_gpioB\_i\_ival，io\_pads\_gpioB\_o\_oval和io\_pads\_gpioB\_o\_oe打包成一个双向端口，命名为io\_pads\_gpioB。与io\_pads\_gpioA一样处理。
2. io\_pads\_qspi0\_sck\_o\_oval：输出时钟，与片外spi flash通信。

选择PO16（CMOS output pad），处理器作为master，提供sck作为时钟，与spi flash通信，不选上拉和下拉，驱动能力选16.

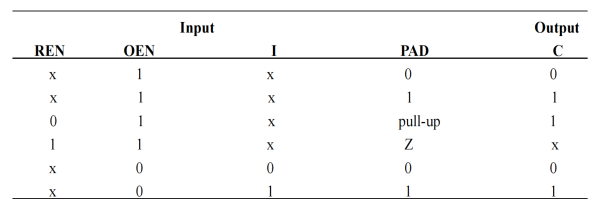
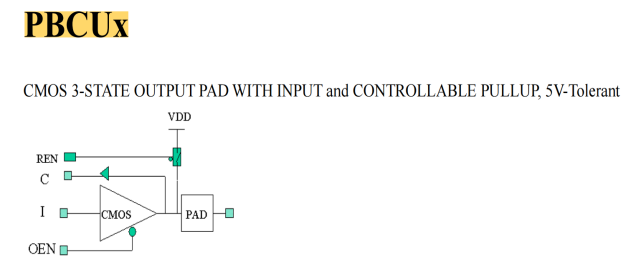
PO16 u\_pad\_io\_pads\_qspi0\_sck\_o\_oval (.I(io\_pads\_qspi0\_sck\_o\_oval\_pin), .PAD(io\_pads\_qspi0\_sck\_o\_oval));

1. io\_pads\_qspi0\_cs\_0\_o\_oval和io\_pads\_qspi0\_sck\_o\_oval一样处理。

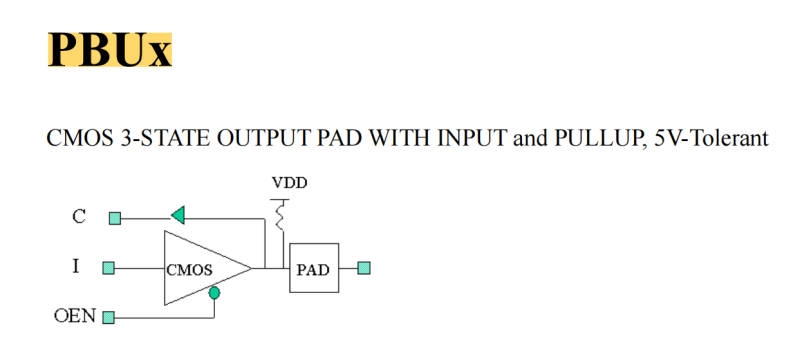
PO16 u\_pad\_io\_pads\_qspi0\_cs\_0\_o\_oval (.I(io\_pads\_qspi0\_cs\_0\_o\_oval\_pin), .PAD(io\_pads\_qspi0\_cs\_0\_o\_oval));

1. io\_pads\_qspi0\_dq\_0\_i\_ival，io\_pads\_qspi0\_dq\_0\_o\_oval和io\_pads\_qspi0\_dq\_0\_o\_oe打包成一个双向端口，命名为io\_pads\_qspi0\_dq\_0。

选择PBCU16（3-state output pad with input and enable controlled pull-up），这是spi flash的双向数据线，当总线空闲时，主从双方都应该将端口设置为开漏输出，并其中一方将端口上拉至电源。如果主从双方都没有上拉，则在板上需要上拉。驱动能力选16.



IO库中还有PBU，更合适，但是RTL仿真时功能没通过，所以选择PBCU，同时强制让REN=0，即强制上拉。



PBCU16 u\_pad\_io\_pads\_qspi0\_dq\_0 (.PAD(io\_pads\_qspi0\_dq\_0), .OEN(~io\_pads\_qspi0\_dq\_0\_o\_oe\_pin), .REN(1'b0), .I(io\_pads\_qspi0\_dq\_0\_o\_oval\_pin), .C(io\_pads\_qspi0\_dq\_0\_i\_ival\_pin));

1. io\_pads\_qspi0\_dq\_1，io\_pads\_qspi0\_dq\_2和io\_pads\_qspi0\_dq\_3一样处理方法。
2. io\_pads\_aon\_erst\_n\_i\_ival：输入，异步复位信号。

选择PIU（Input pad with pull-up），上拉的原因是正常工作模式时，该端口悬空即可。

PIU u\_pad\_io\_pads\_aon\_erst\_n\_i\_ival ( .PAD(io\_pads\_aon\_erst\_n\_i\_ival), .C(io\_pads\_aon\_erst\_n\_i\_ival\_pin) );

1. io\_pads\_dbgmode0\_n\_i\_ival：输入，debug模块控制信号

选择PIU（Input pad with pull-up），端口悬空时，提供逻辑1进入芯片。

PIU u\_pad\_io\_pads\_dbgmode0\_n\_i\_ival ( .PAD(io\_pads\_dbgmode0\_n\_i\_ival), .C(io\_pads\_dbgmode0\_n\_i\_ival\_pin) );

1. io\_pads\_dbgmode1\_n\_i\_ival和io\_pads\_dbgmode2\_n\_i\_ival一样处理
2. io\_pads\_bootrom\_n\_i\_ival：输入，boot启动时PC的控制信号

选择PIU（Input pad with pull-up），端口悬空时，提供逻辑1进入芯片。

PIU u\_pad\_io\_pads\_bootrom\_n\_i\_ival ( .PAD(io\_pads\_bootrom\_n\_i\_ival), .C(io\_pads\_bootrom\_n\_i\_ival\_pin) );

1. u\_pad\_io\_pads\_aon\_pmu\_dwakeup\_n\_i\_ival：输入，always-on模块控制信号

选择PIU（Input pad with pull-up），端口悬空时，提供逻辑1进入芯片。

PIU u\_pad\_io\_pads\_aon\_pmu\_dwakeup\_n\_i\_ival ( .PAD(io\_pads\_aon\_pmu\_dwakeup\_n\_i\_ival), .C(io\_pads\_aon\_pmu\_dwakeup\_n\_i\_ival\_pin) );

1. io\_pads\_aon\_pmu\_padrst\_o\_oval和io\_pads\_aon\_pmu\_vddpaden\_o\_oval：输出，always-on模块控制信号

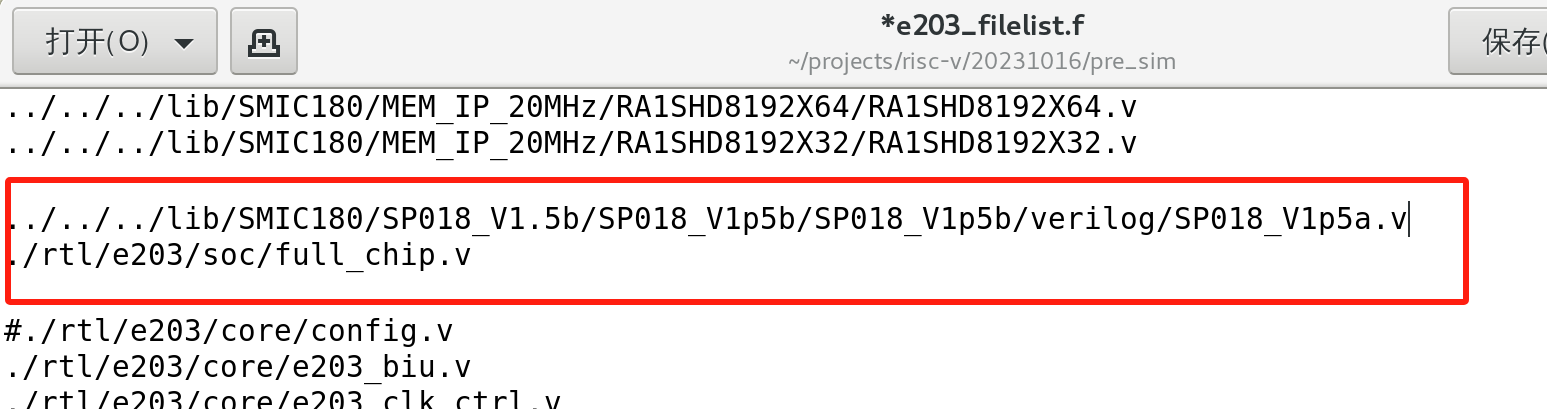
选择PO16（CMOS output pad），用作控制信号，驱动能力选择4.

PO16 u\_pad\_io\_pads\_aon\_pmu\_padrst\_o\_oval ( .I(io\_pads\_aon\_pmu\_padrst\_o\_oval\_pin), .PAD(io\_pads\_aon\_pmu\_padrst\_o\_oval) );

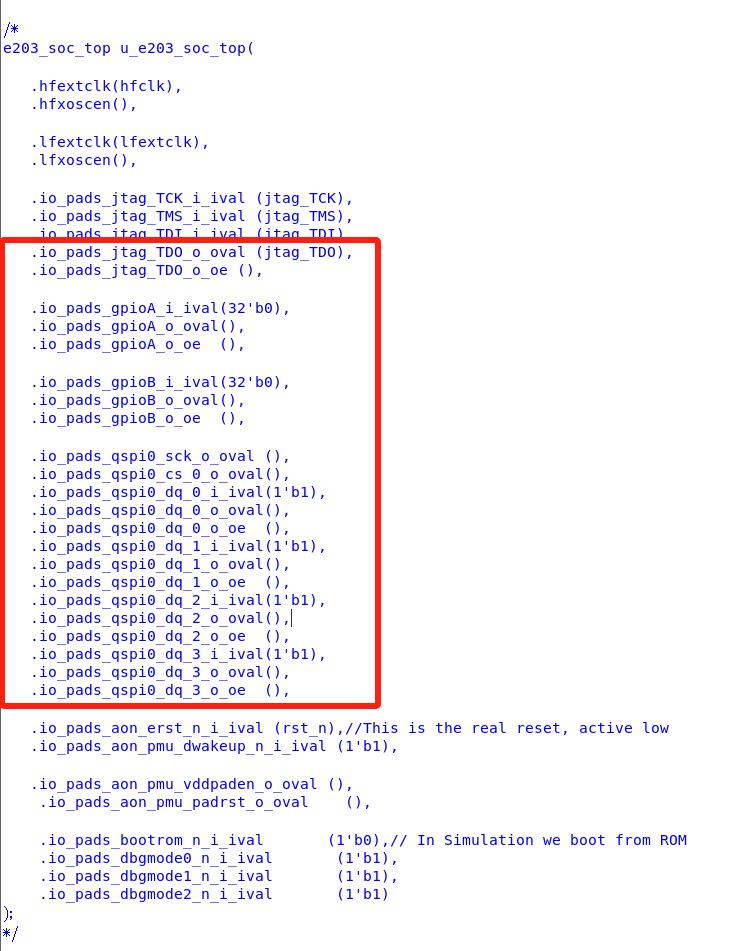
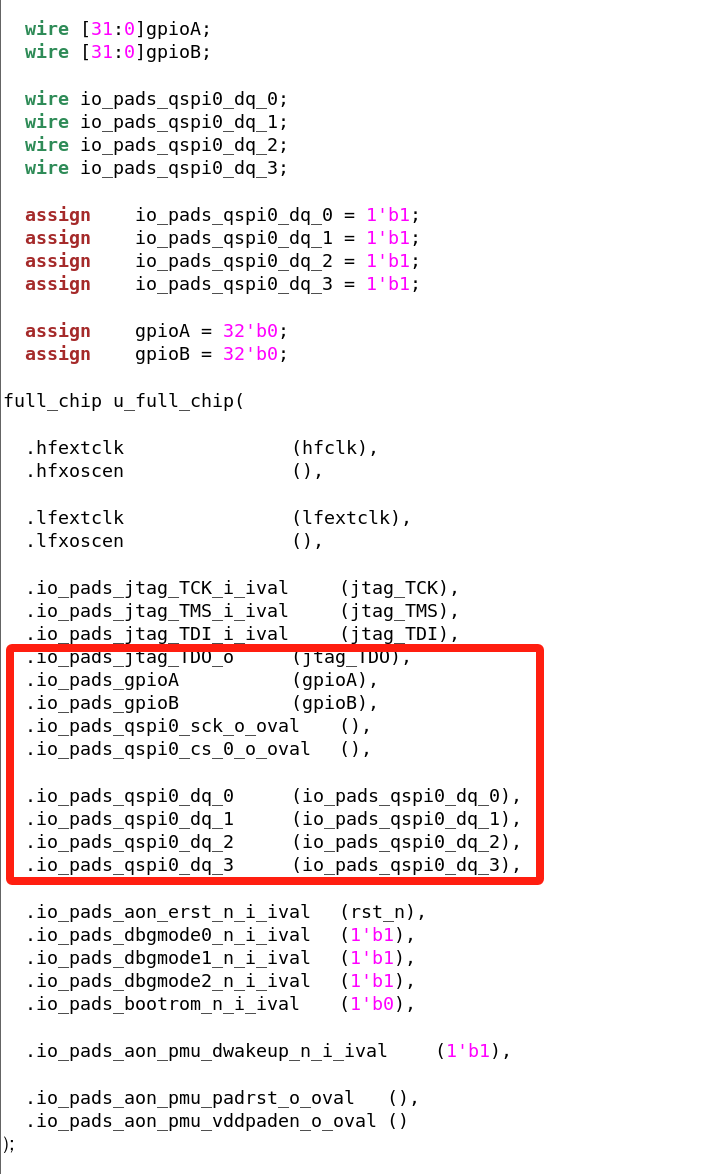
PO16 u\_pad\_io\_pads\_aon\_pmu\_vddpaden\_o\_oval ( .I(io\_pads\_aon\_pmu\_vddpaden\_o\_oval\_pin), .PAD(io\_pads\_aon\_pmu\_vddpaden\_o\_oval) );

* 1. 仿真验证

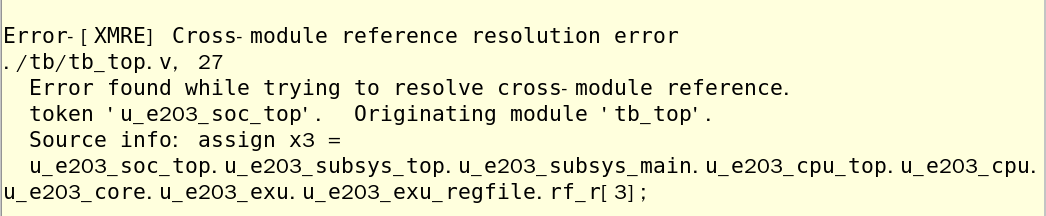
1. e203\_filelist.f文件将full\_chip.v和IO库的.v文件加进来。



1. tb\_top.v。例化full\_chip模块，对于输入或者输出端口，施加的激励不变。对于双向端口gpioA，gpioB和qspi0\_dq\_x，以及三态输出jtag\_TD0\_o，加激励的方式要修改，直接对最终的PAD施加激励。这样赋值后，对于双向IO单元gpioA，PAD上有激励，对应的io\_pads\_gpioA\_i\_ival信号也有激励，即可。

1. 仿真报错如下。tb\_top.v中例化的顶层变成了full\_chip，对层次调用要修改。如何改？

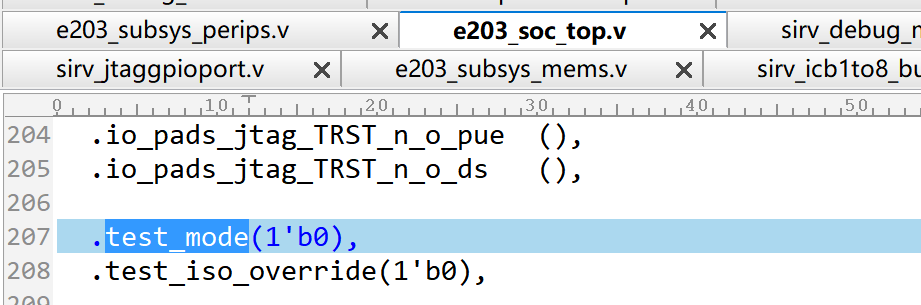
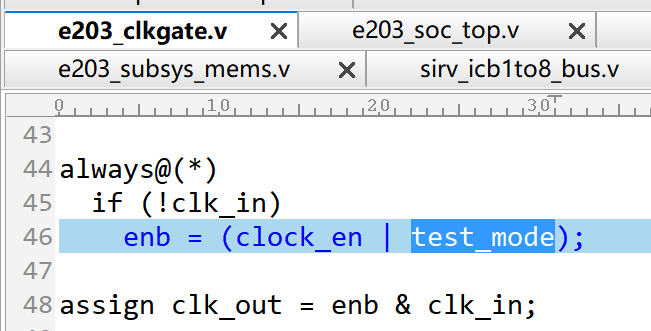


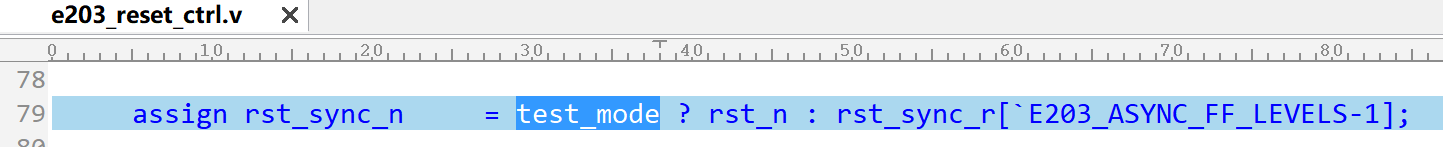
1. 仿真验证。对测试程序进行仿真验证，结果显示PASS，但是这些程序功能是否正确与IO功能并没有直接关系。事实上，对于双向IO的功能还没有验证。QSPI0的双向IO功能可以通过“Flash下载功能”去验证。
   1. 学习要求
2. 熟悉数字IO类型和功能：输入/输出/双向/三态/驱动能力/上拉/下拉。
3. RTL代码版本管理。将当前pre\_sim文件夹另存为pre\_sim(full\_chip)。
4. LAB5：full\_chip顶层增加test\_mode端口

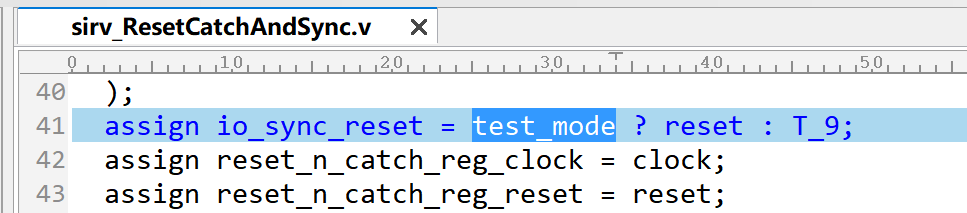
从e203\_subsys\_top到e203\_soc\_top，并不是所有端口都引出芯片，对这些未引出的端口不做详细解释，因为这和处理器的具体功能相关，不是本实践课的重点。这里仅以test\_mode端口为例，尝试修改IP的逻辑功能，将原先未引出芯片的test\_mode端口，最终引出芯片。

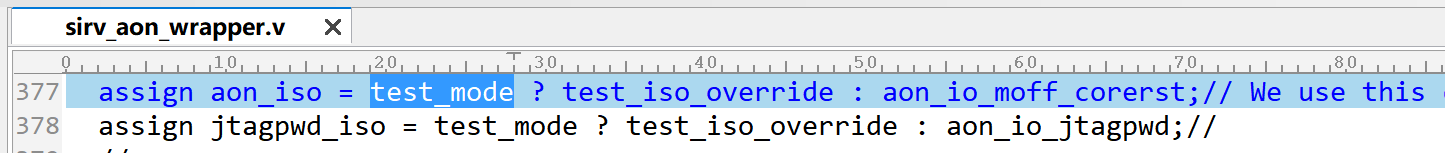
* 1. test\_mode功能分析

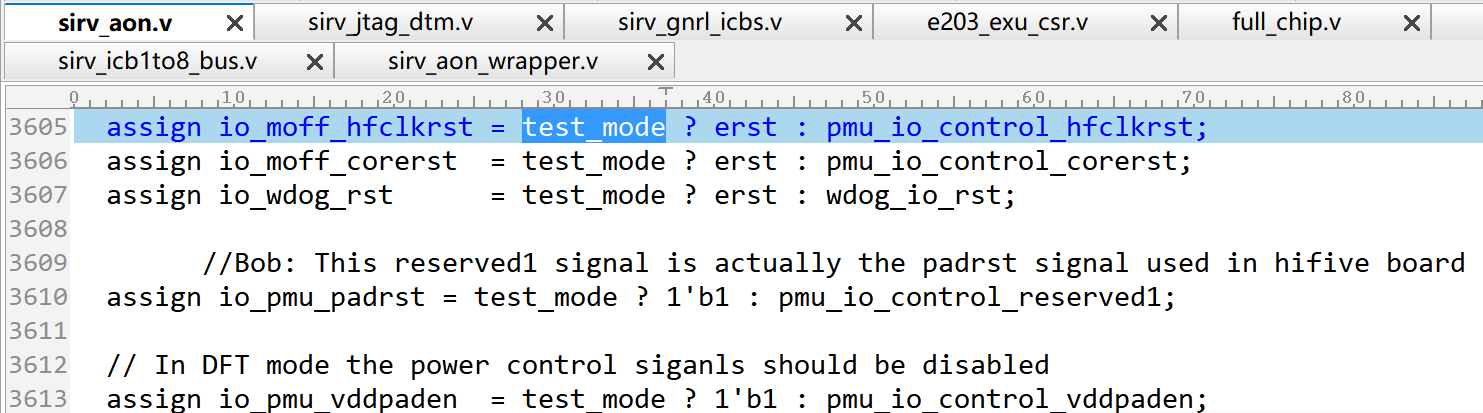
通过追踪test\_mode在代码中的逻辑功能，发现它是测试模式控制信号，代码中被强制为0。test\_mode的作用是在debug模式下，让整个芯片处于非复位状态、关闭所有门控时钟（即让时钟常开）、所有电路都上电（如果有门控电源或者隔离单元的话），然后通过JTAG接口去访问芯片内部寄存器和存储器。下面尝试将这个信号改为引出芯片，在片外进行0或1选择。

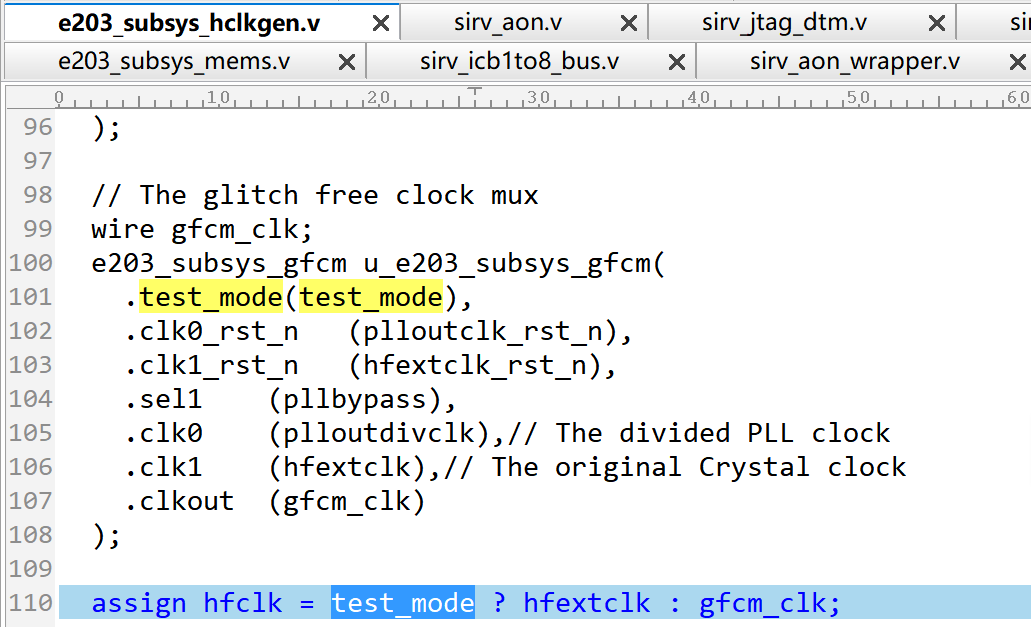
 

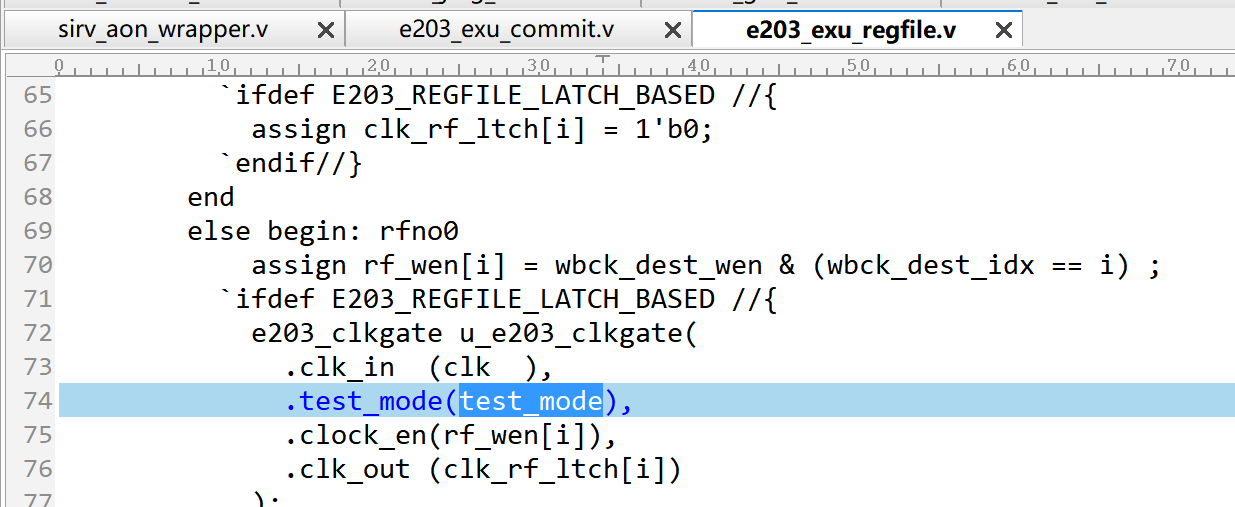












* 1. test\_mode端口引出片外

（1）修改e203\_soc\_top.v，把test\_mode引出e203\_soc\_top模块。

（2）修改full\_chip.v，full\_chip.v中为test\_mode选择输入带下拉的IO，即PID（Input pad with pull down），当test\_mode端口悬空时，输入逻辑0进入芯片，芯片处于正常工作模式，而非测试模式。

（3）修改tb\_top.v，因为full\_chip.v多了一个test\_mode输入端口，例化full\_chip时要修改。

（4）最后，完成仿真，验证修改test\_mode没有影响原有功能。

* 1. 学习要求

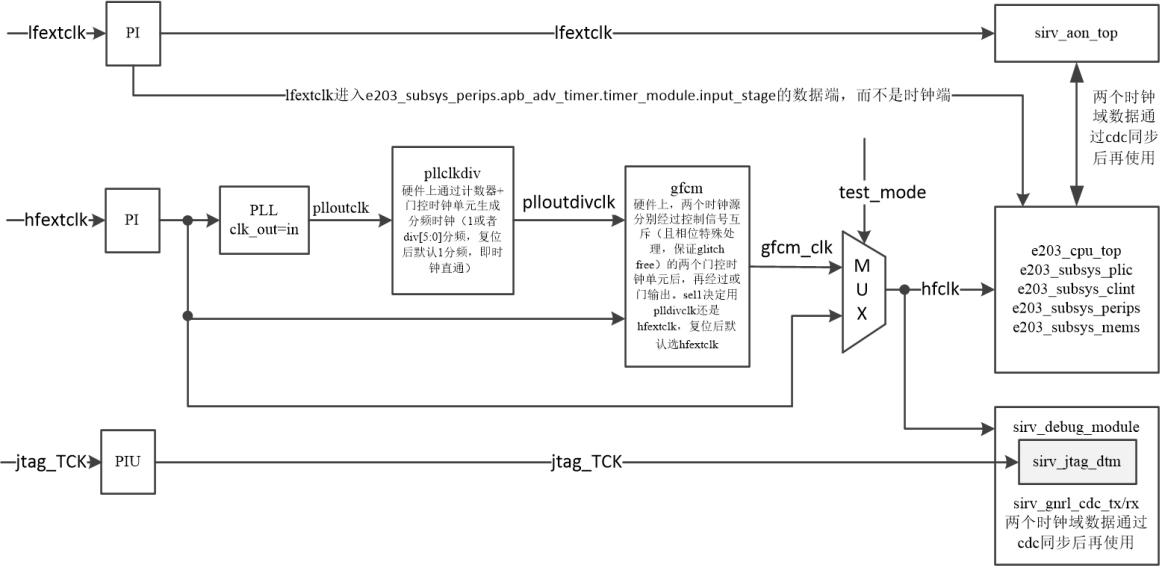
1. 掌握将芯片内部信号引出端口的方法
2. 保存为pre\_sim(test\_mode)
3. LAB6：时钟网络分析

时钟网络分析的目的是弄清楚系统时钟网络结构，为逻辑综合阶段的时序约束和时序检查做好准备。

* 1. 时钟网络

时钟网络如下，hfextclk为系统主时钟，给内核和外设提供时钟。jtag\_TCK为debug模块的主时钟，和hfextclk是异步的，两个时钟域传输数据通过cdc模块同步。lfextclk为always-on模块的主时钟，和hfextclk是异步的，两个时钟域传输数据通过cdc模块同步。

根据以上时钟网络结构，逻辑综合的时钟约束大致如下：hfextclk、lfextclk和jtag\_TCK频率分别设置为20MHz、50KHz和1MHz，三个时钟之间属于异步关系，即穿越时钟域的路径可不做时序检查，由设计保证时序可靠性。



* 1. hfextclk传播路径

时钟源hfextclk连接到触发器CK端口之前，经过了一些硬件逻辑（包括组合逻辑和时序逻辑），要考虑时序分析时，hfextclk时钟能否穿过这些硬件逻辑。如果时钟信号被这些逻辑门“卡住”，则后续的D触发器CK端口无时钟约束，是不允许的。

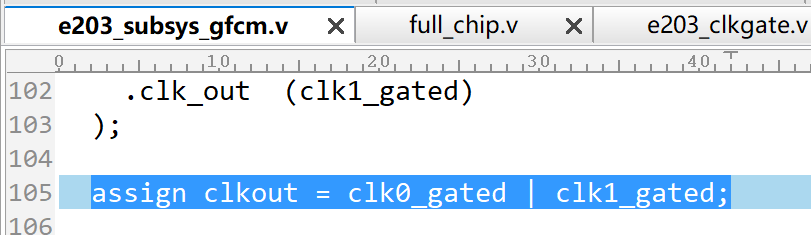
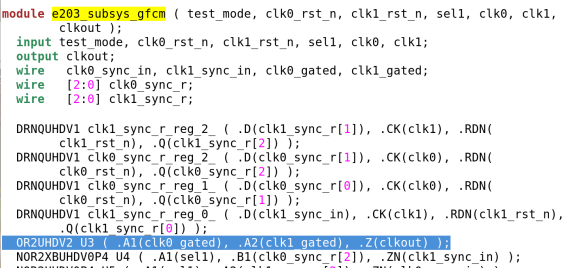
1. PI、PLL是将时钟直通，pllclkdiv模块是用计数器+门控时钟的方式实现时钟分频，这两个模块的时钟都是可以穿过硬件逻辑的，无需特殊设置。
2. gfcm模块硬件结构如下，时钟会被这个OR cell隔断吗？综合没有报告时钟被这个OR隔断，原因是OR的两个输入都是确定的、而且已经被约束上的时钟，所以工具可以计算出OR的输出。假设OR的一个输入是时钟，另一个输入来自于片内和片外的控制信号，则OR的输出取决于控制信号的状态，工具无法计算OR的输出，此时约束需要强制设定OR控制信号的状态，或者在OR输出处设置generated clock。

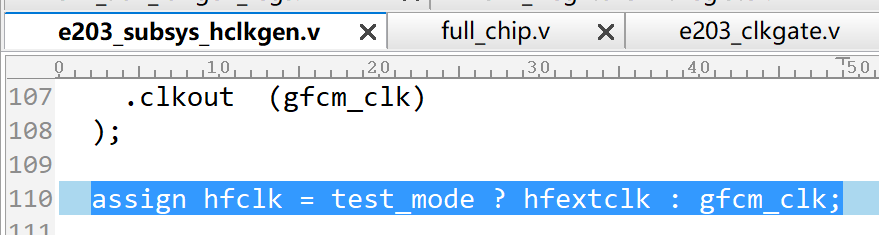
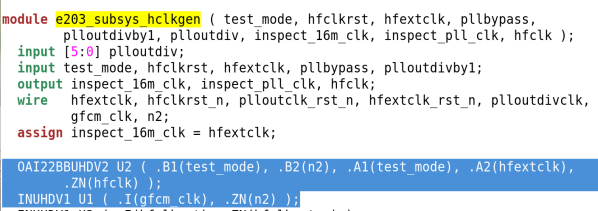


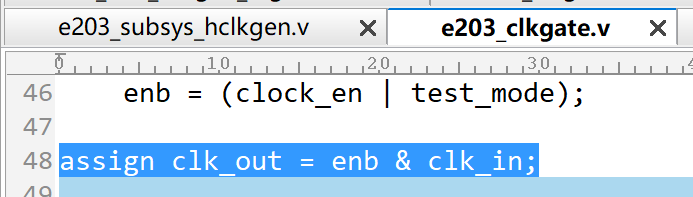
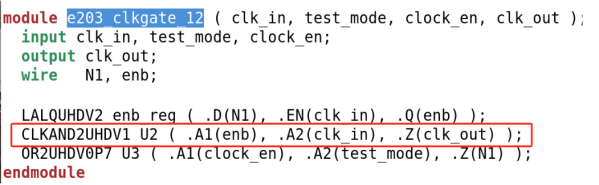
1. clock MUX的输出取决于test\_mode。综合时会报告时钟信号无法传过去，可以通过set\_case\_analysis 0 [get\_ports test\_mode]强制让test\_mode为0或1，让MUX的控制信号处于确定的状态。
   1. 时钟网络上的逻辑门

在RTL代码中，时钟网络上有一系列逻辑操作，是否需要用标准单元库中的时钟专用cell去代替？答案是：需要。

如果不替换，让工具自动去综合，则对应的硬件如右边所示。

（思考题三：上图中，左边是RTL代码中时钟信号的相关逻辑操作，右边是逻辑综合后对应的网表，综合的结果符合预期吗？为什么？应该怎么处理？）

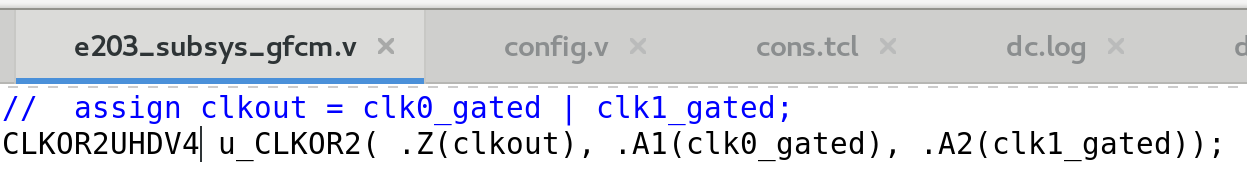
以上，EDA工具自动综合选用的cell都不是我们想要的，对于时钟或操作，有CLKOR2UHD单元可以用。对于时钟mux操作，有CLKMUX2UHD可以用。对于时钟与操作（即门控时钟），有CLKLANQUHD可以用。对于时钟信号，要使用标准单元库中的专用cell。

* 1. 例化时钟专用cell

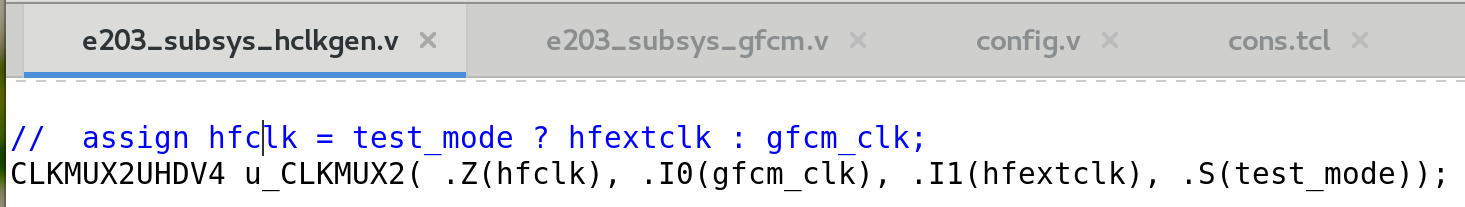
由于时钟信号的特殊性，时钟网络上的逻辑操作，只能使用特定的cell实现。如果交由EDA工具自动完成，得到的网表一般不符合预期。因此，我们需要在RTL代码里直接例化这些专用cell，然后在逻辑综合约束中设置dont\_touch属性。

修改RTL之前，需要先弄清楚RTL的功能，再查看CLK cell的功能（见标准单元手册、 .v和.lib），保证两者逻辑的一致性。具体修改如下：

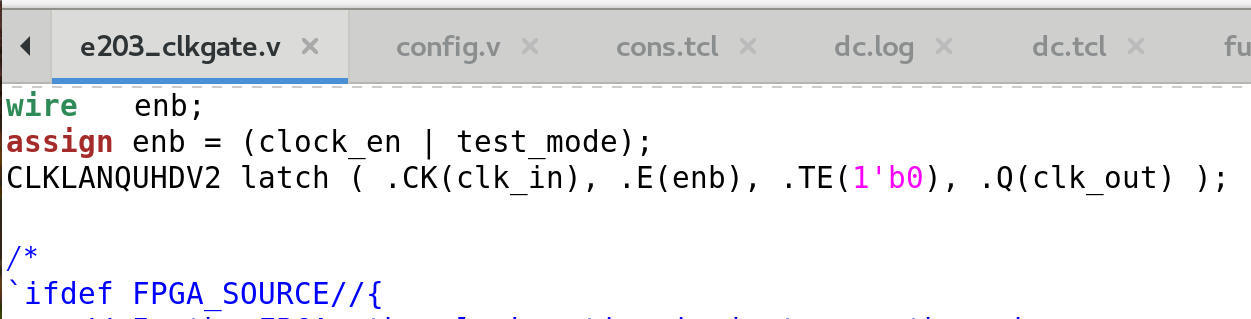
1. e203\_subsys\_gfcm.v将时钟或操作改为用CLKOR2UHDV4代替。



1. e203\_subsys\_hclkgen.v将时钟2选1操作改为用CLKMUX2UHDV4代替。



1. e203\_clkgate.v将门控时钟逻辑用CLKLANQUHDV2代替。



1. 仿真验证，保证功能一致性。
   1. 回答思考题三
   2. 学习要求
2. 时钟网络分析的目的；
3. 将时钟网络上的逻辑操作用时钟专用cell来实现；
4. 保存为pre\_sim(clock)
5. LAB7：逻辑综合1：syn

逻辑综合是将RTL代码映射为由标准单元组成的网表netlist，EDA工具保证RTL和netlist逻辑功能的一致性，同时，满足时序、面积和功耗约束。工具：Synopsys之DesignCompiler。

* 1. 逻辑综合目录

1. 建立综合目录./syn。子目录scripts用于存放setup\_dc，constraints，dc综合脚本。子目录netlist用于存放综合网表。子目录reports用于存放时序、面积、功耗等各类报告。子目录results用于存放sdc，ddc等文件。
2. 生成memory的db文件。标准单元库和IO均由Foundry同时提供lib和db文件，但是MemoryCompiler生成的memory只有lib文件，没有db文件，要自己生成。

→ 在./lib/SMIC180/MEM\_IP\_20MHz/目录下建立文件夹db，用于保存所有memory的三个工艺角的db文件。

→ Memory的lib转为db命令如下：启动library compiler，命令行输入：lc

read\_lib RA1SHD8192X32\_ss\_1.08\_125.0\_syn.lib

write\_lib -format db USERLIB -output RA1SHD8192X32\_ss\_1.08\_125.0\_syn.db

→ 注意：每次生成完一种memory，要退出lc，进入另一个memory文件夹，再启动lc才可以。

* 1. setup\_dc.tcl脚本

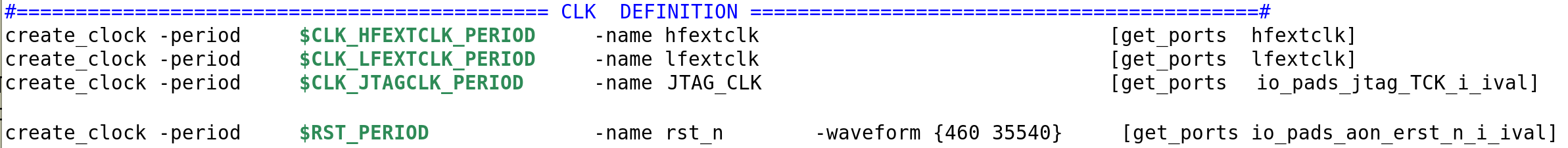
./syn/scripts/setup\_dc.tcl脚本，设定set search\_path/set target\_library/set synthetic\_library/set link\_library，读入所有RTL代码，并规定顶层模块。

综合要使用ss工艺角的db文件，即125度，90%工作电压。

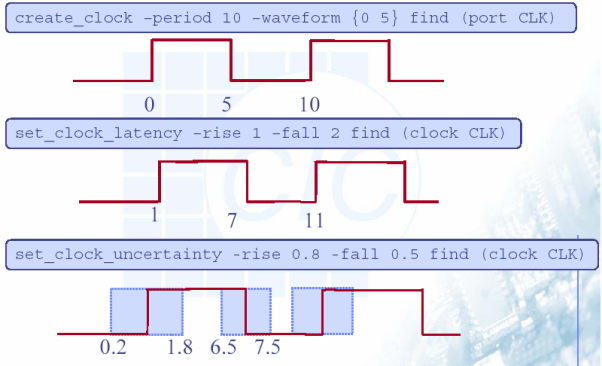
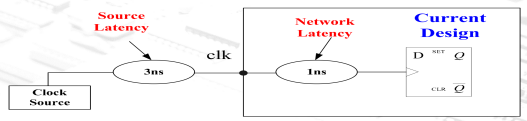
* 1. cons脚本

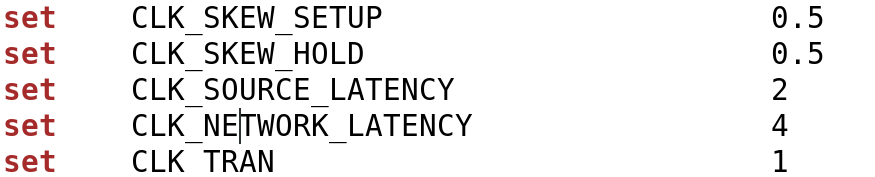
./syn/scripts/cons.tcl脚本，包含时钟约束、输入输出约束、其他。

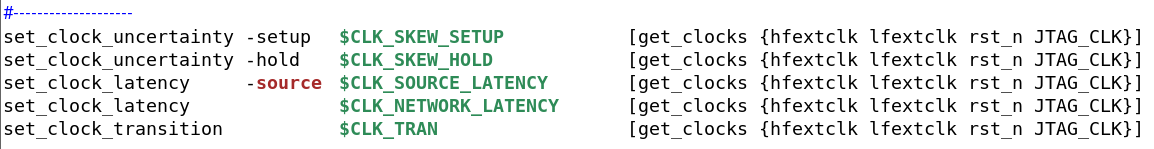
1. 时钟约束create\_clock：名字、周期、占空比等



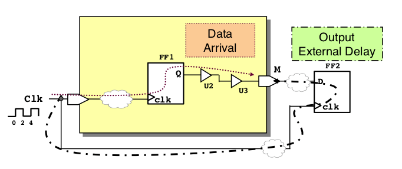
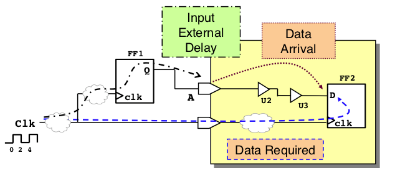
1. 时钟约束create\_generated\_clock：时钟网络上有组合逻辑（类似与、或、异或门、选择器等）被卡住，需要在逻辑门的输出端create\_generated\_clock，让时钟继续传下去。
2. 时钟约束：transition、uncertainty、latency

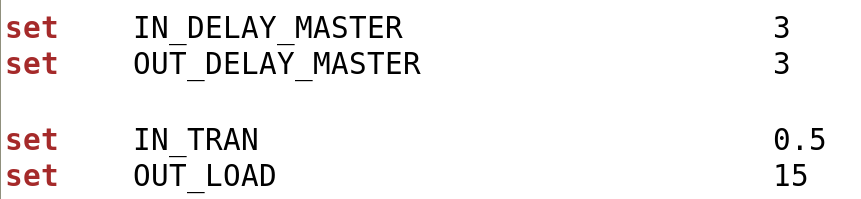
 

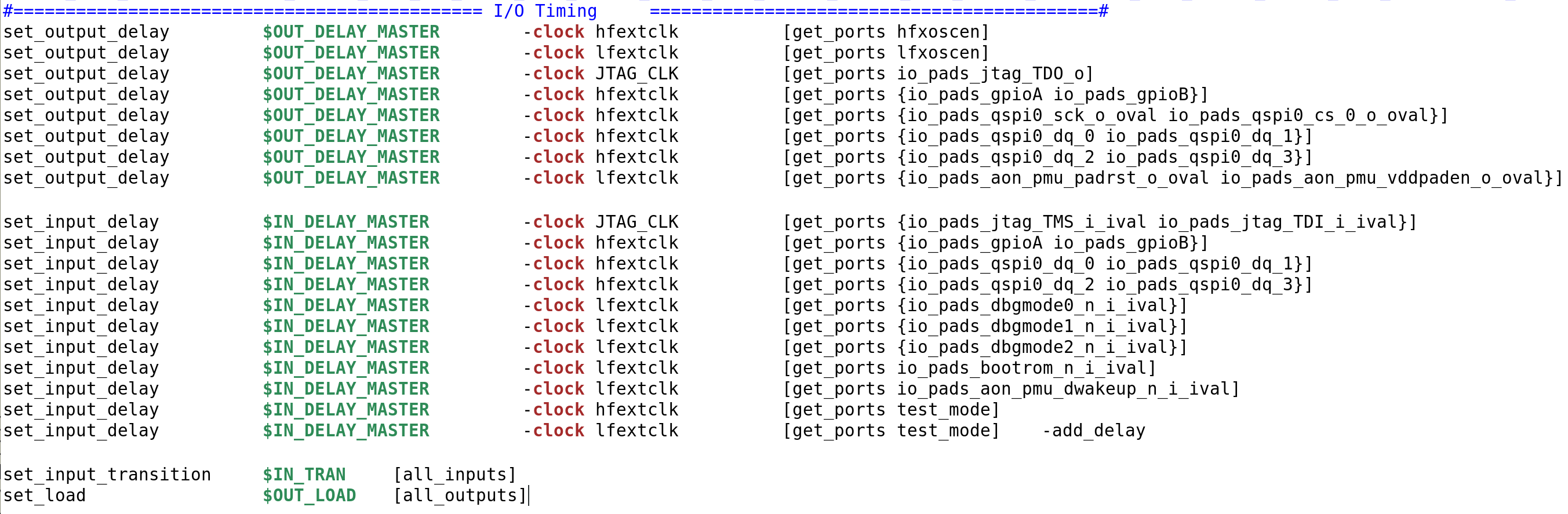




1. IO约束：input/output delay，input transition，output load。信号和哪个时钟域相关（即由哪个时钟域采集或者发射），就关联到哪个时钟域上。

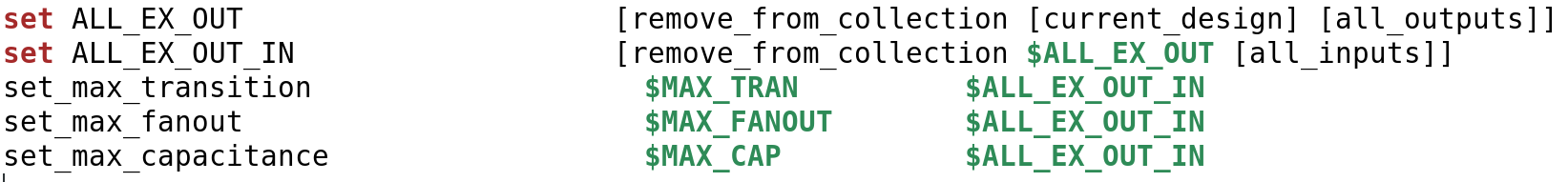






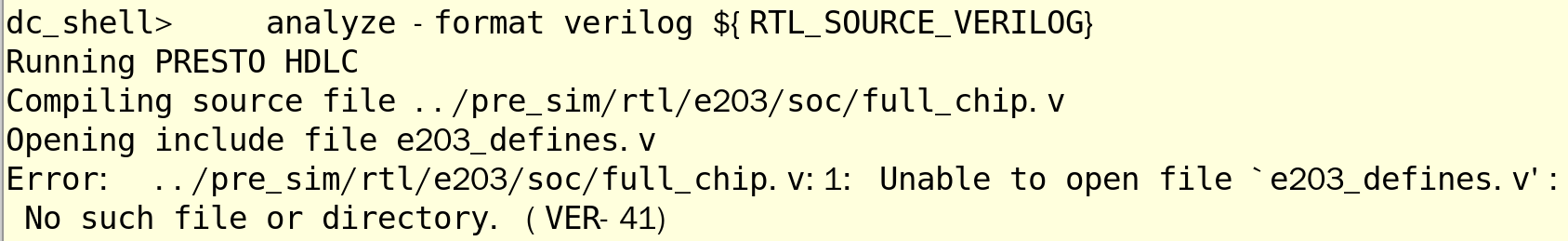
1. 环境约束：set\_max\_transtion，set\_max\_fanout，set\_max\_capacitance





* 1. 启动dc

1. 命令行中输入cd ./syn，然后输入dc\_shell-t
2. source ./scripts/dc.tcl > ./dc.log时报错，这时需要将./pre\_sim/目录下的config.v, e203\_defines.v和i2c\_master\_defines.v拷贝到./syn/目录下，否则报错如下。



综合可以跑起来了，打开dc.log文件，挨个确认error和warning，所有的erro都要解决，warning需要逐个分析。

* 1. 学习要求

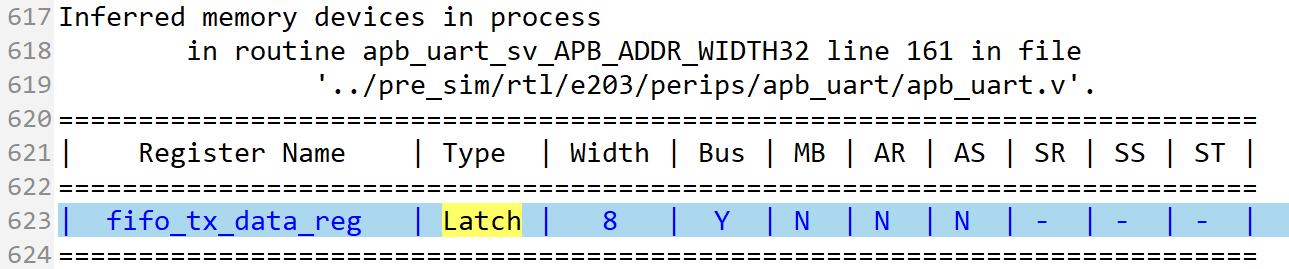
1. 工艺库文件.v, .lib, doc解读；
2. 综合的setup脚本基本内容；
3. 时序约束脚本的基本内容；
4. LAB8：逻辑综合2：dc.log

初次跑逻辑综合，需要仔细看综合报告dc.log，修正RTL代码中的bug，主要是latch或者logic loop之类的错误。其次看timing report，主要是看触发器是否都约束上。这个环节先不考虑约束是否严格，主要是看整个综合流程是否能跑下来：一方面检查RTL代码合规性（Latch）以及综合环境（工艺库），另一方面检查时序约束是否完备（时钟是否都约束上了？timing\_report中有没有异常的路径，比如严重的slack违反?）

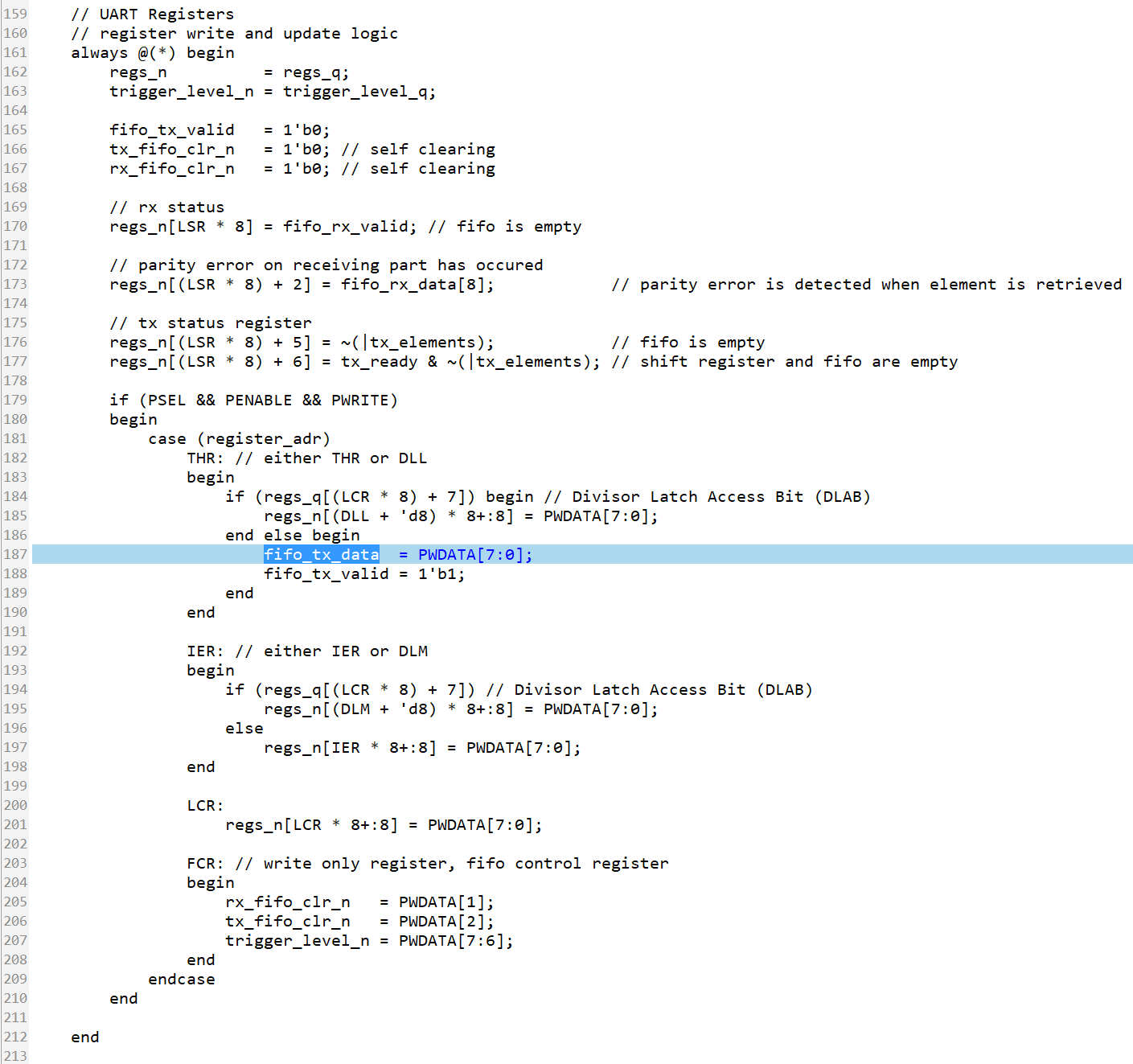
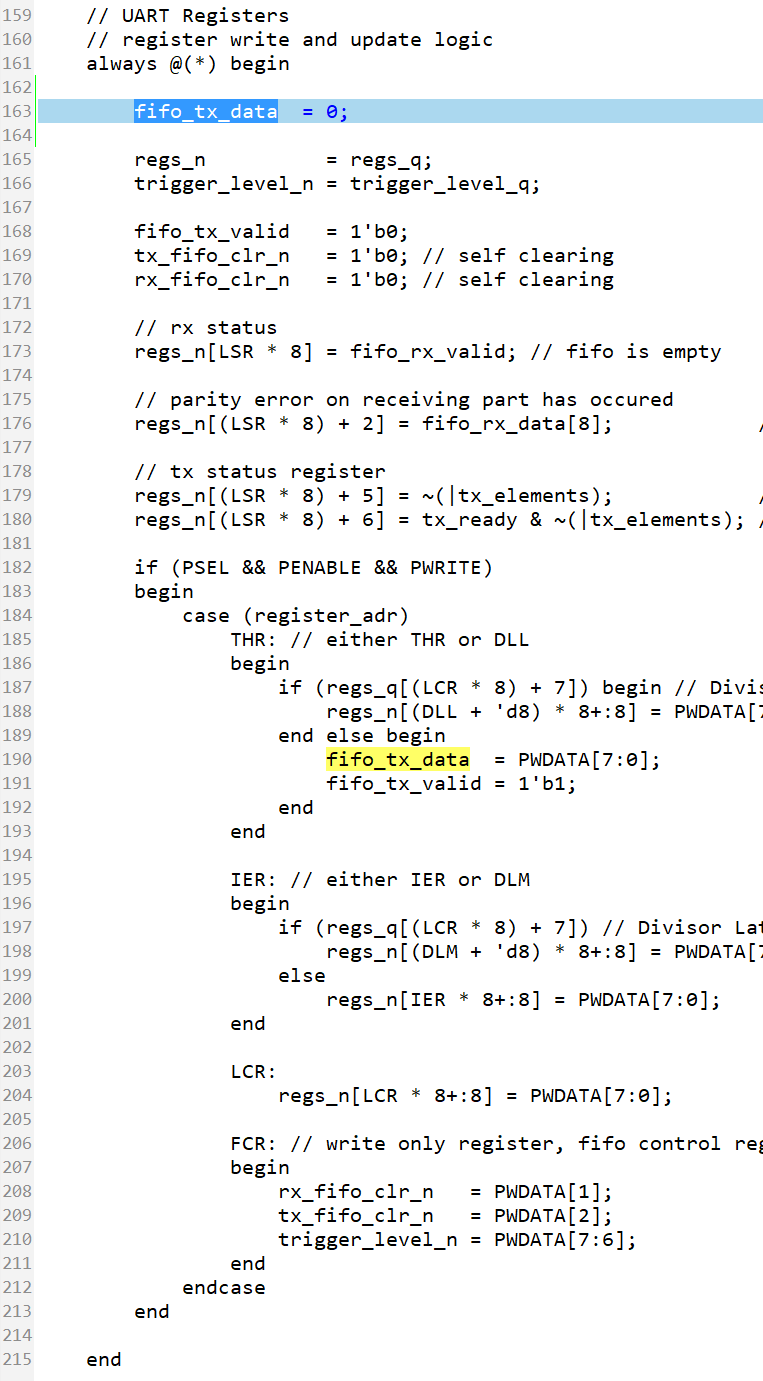
dc.log中搜索error, latch和warning信息，逐个分析解决。

* 1. 锁存器Latch

dc.log中搜索latch，报告显示fifo\_tx\_data被综合成了Latch。



查看RTL代码，并对apb\_uart.v模块做以下修改，消除Latch。

（思考题四：Latch产生的原因是什么？如上修改后为什么能消除Latch？）

* 1. 可以忽略的warning

Warning: ../pre\_sim/rtl/e203/perips/apb\_uart/apb\_uart.v:35: Parameter keyword used in local parameter declaration. (VER-329)

Warning: ../pre\_sim/rtl/e203/perips/apb\_uart/apb\_uart.v:113: Net regs\_n connected to instance uart\_rx\_fifo\_i is declared as reg data type but is not driven by an always block. (VER-1004)

Warning: ../pre\_sim/rtl/e203/perips/apb\_i2c/i2c\_master\_byte\_ctrl.v:220: Case statement is not a full case. (ELAB-909)

Warning: The trip points for the library named SP018\_V1p4\_max differ from those in the library named scc018ug\_uhd\_rvt\_ss\_v1p62\_125c\_basic. (TIM-164)

Warning: IO pad 'PDIODE8' is unusable: unknown logic function. (OPT-1022)

Warning: Operating condition ss\_v1p62\_125c set on design full\_chip has different process,

voltage and temperatures parameters than the parameters at which target library SP018\_V1p4\_max is characterized. Delays may be inaccurate as a result. (OPT-998)

Warning: Design 'full\_chip' contains 4 high-fanout nets. A fanout number of 1000 will be used for delay calculations involving these nets. (TIM-134)

Net 'u\_e203\_soc\_top/u\_e203\_subsys\_top/u\_e203\_subsys\_main/u\_e203\_subsys\_plic/u\_wdg\_irq\_sync/sync\_gen[0].i\_is\_0.sync\_dffr/clk': 8664 load(s), 1 driver(s)

Net 'u\_e203\_soc\_top/u\_e203\_subsys\_top/u\_e203\_subsys\_main/u\_e203\_subsys\_plic/u\_wdg\_irq\_sync/sync\_gen[0].i\_is\_0.sync\_dffr/rst\_n': 6471 load(s), 1 driver(s)

Net 'u\_e203\_soc\_top/u\_e203\_subsys\_top/u\_e203\_subsys\_main/u\_e203\_cpu\_top/u\_e203\_cpu/u\_e203\_core/u\_e203\_exu/u\_e203\_exu\_alu/u\_e203\_exu\_alu\_dpath/sbf\_1\_dffl/clk': 1641 load(s), 1 driver(s)

Net 'u\_e203\_soc\_top/u\_e203\_subsys\_top/u\_sirv\_aon\_top/u\_aon\_icb\_cdc\_rx/u\_i\_vld\_sync/sync\_gen[0].i\_is\_0.sync\_dffr/clk': 1039 load(s), 1 driver(s)

Warning: Current implementation 'rpl' of module 'DW01\_add' was not found in the files of synthetic\_library. Implementation selection of synthetic design

'e203\_subsys\_nice\_core\_DW01\_add\_2' (cell 'u\_e203\_soc\_top/u\_e203\_subsys\_top/u\_e203\_subsys\_main/u\_e203\_cpu\_top/u\_e203\_cpu/u\_e203\_nice\_core/add\_349') will take longer. (SYNH-22)

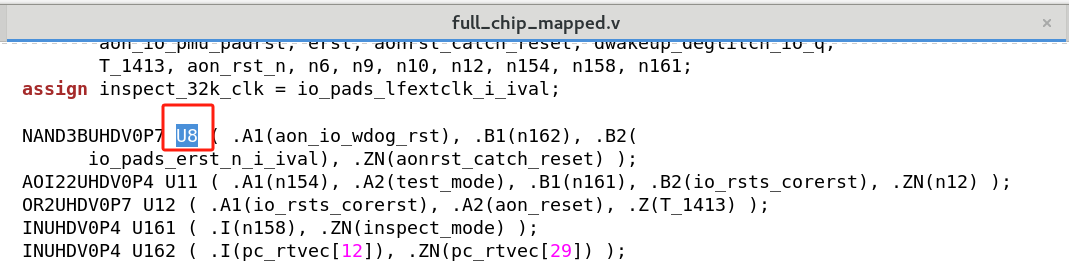
Warning: Design has unannotated black box outputs. (PWR-428)

Warning: The specified replacement character (\_) is conflicting with the specified allowed or restricted character. (UCN-4)

Warning: In the design e203\_subsys\_hclkgen, net 'hfextclk' is connecting multiple ports. (UCN-1)

* 1. 需要处理的warning：(TIM-052)

Warning: A non-unate path in clock network for clock 'rst\_n' from pin 'u\_e203\_soc\_top/u\_e203\_subsys\_top/u\_sirv\_aon\_top/u\_sirv\_aon\_wrapper/U8/ZN' is detected. (TIM-052)



复位网络rst\_n传到U8这里卡住了，工具无法判断输出端口ZN和输入rst\_n网络的相位关系，U8的B1固定为1。B2端口在cons.tcl中约束了时钟属性。

A1端口： assign io\_wdog\_rst = test\_mode ? erst : wdog\_io\_rst;

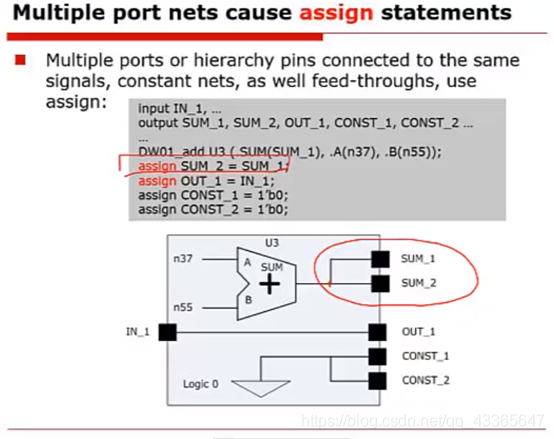
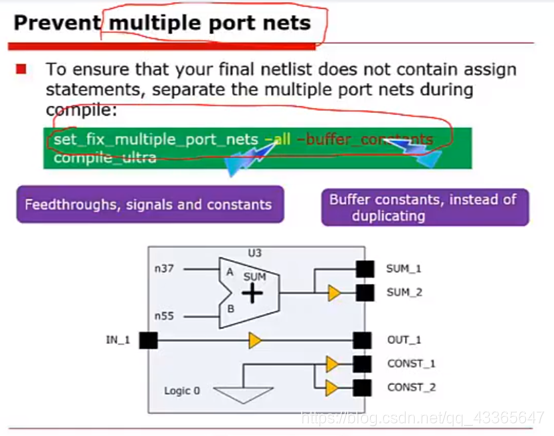
在cons.tcl 中如下设置，即设置test\_mode=0，就可以消除该warning。

set\_case\_analysis 0 [get\_ports test\_mode]

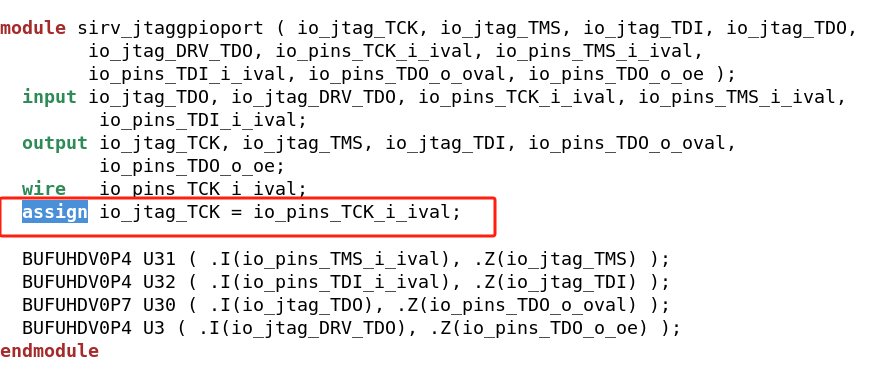
* 1. 需要处理的warning：(VO-4)

Warning: Verilog 'assign' or 'tran' statements are written out. (VO-4)

后端工具不支持网表中的assign，所以要消除网表中的assign。

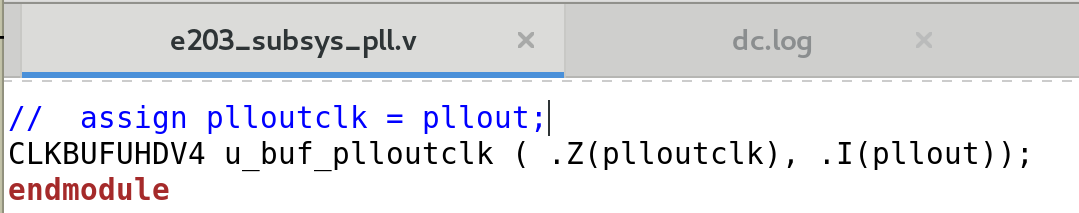
 

在dc.tcl中加入命令：Set\_fix\_multiple\_port\_nets -all -buffer\_constants

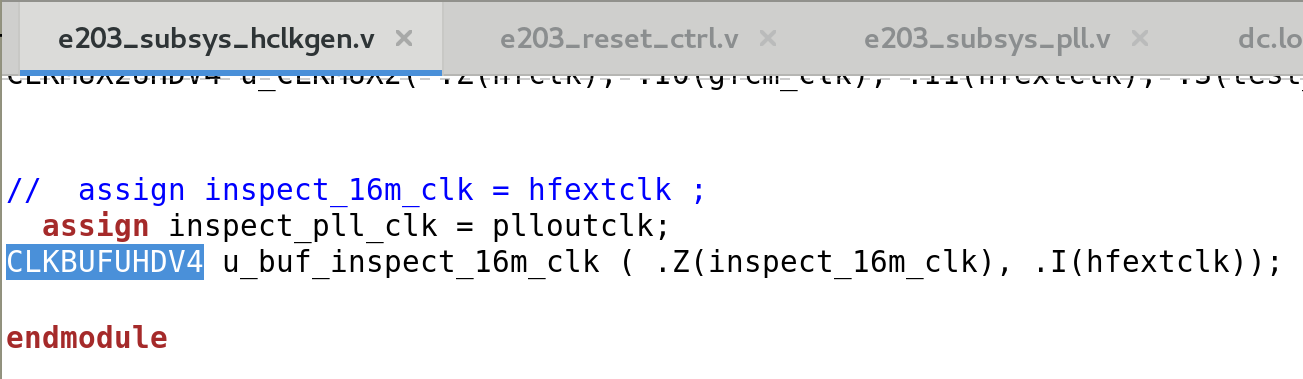


以上命令，对于数据输入直接赋值给输出的情况，EDA工具使用BUF cell解决。但是对于时钟信号，工具仍然使用assign，如上图。网表中一共有10处，都是和时钟、复位相关的信号。

解决办法：手工将RTL中这些assign赋值改为CLKBUFUHDV4，如下。这样处理之后，网表中就没有assign语句了。





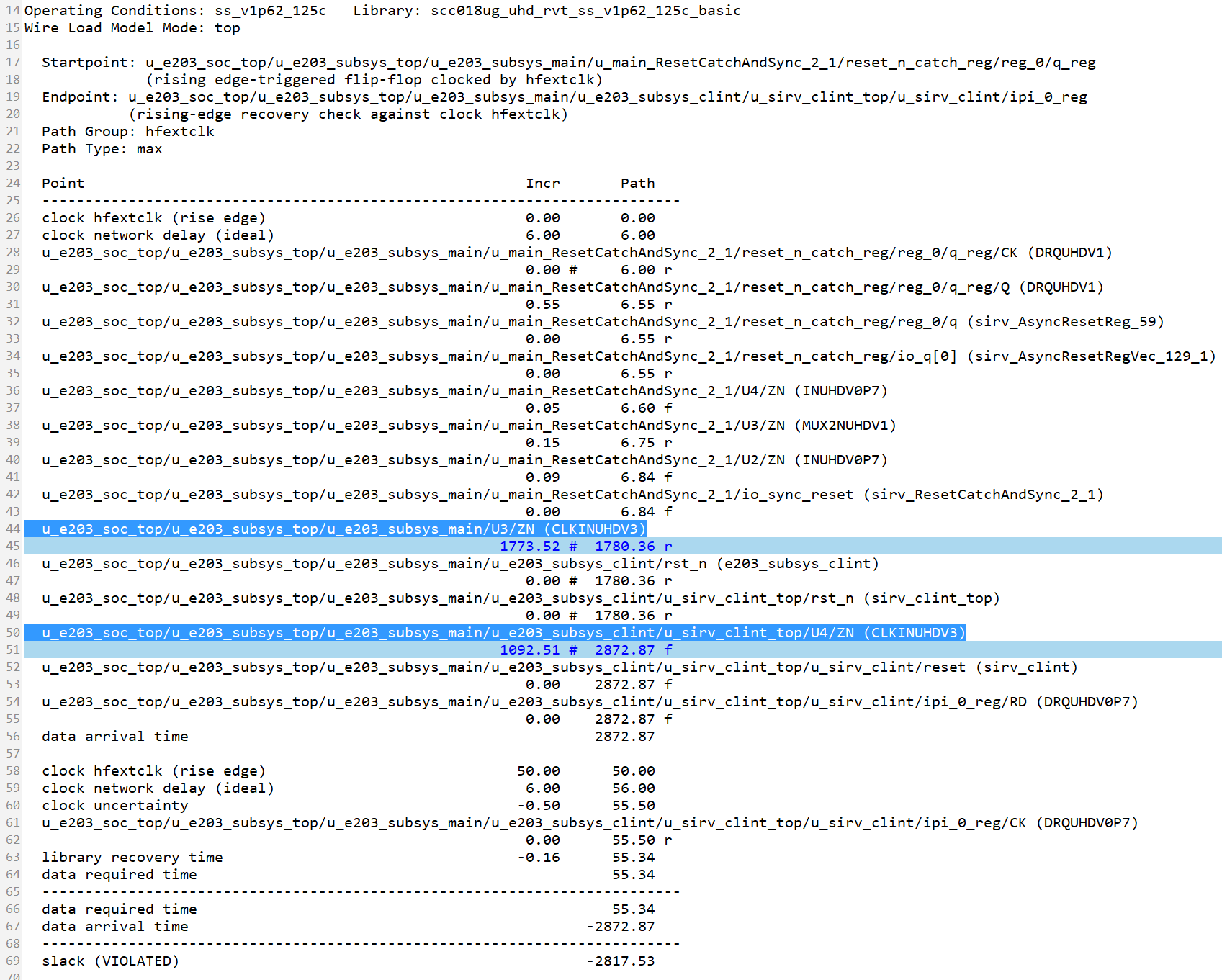


剩余5处assign，同学们查找网表中assign位置，并找到RTL中对应代码，做同样处理。

* 1. 查看时序报告max\_delay

看full\_chip\_constraint\_all\_violators.rpt，发现slack非常严重，用以下命令报告第一个路径：这条路径是从一个触发器的Q端到另一个触发器的RD端，因此这条路径是复位网络，可以看到U3和U4上的延时非常大，这符合复位网络的特点。约束中没有将该复位网络设为ideal\_network，因此延时超级大。

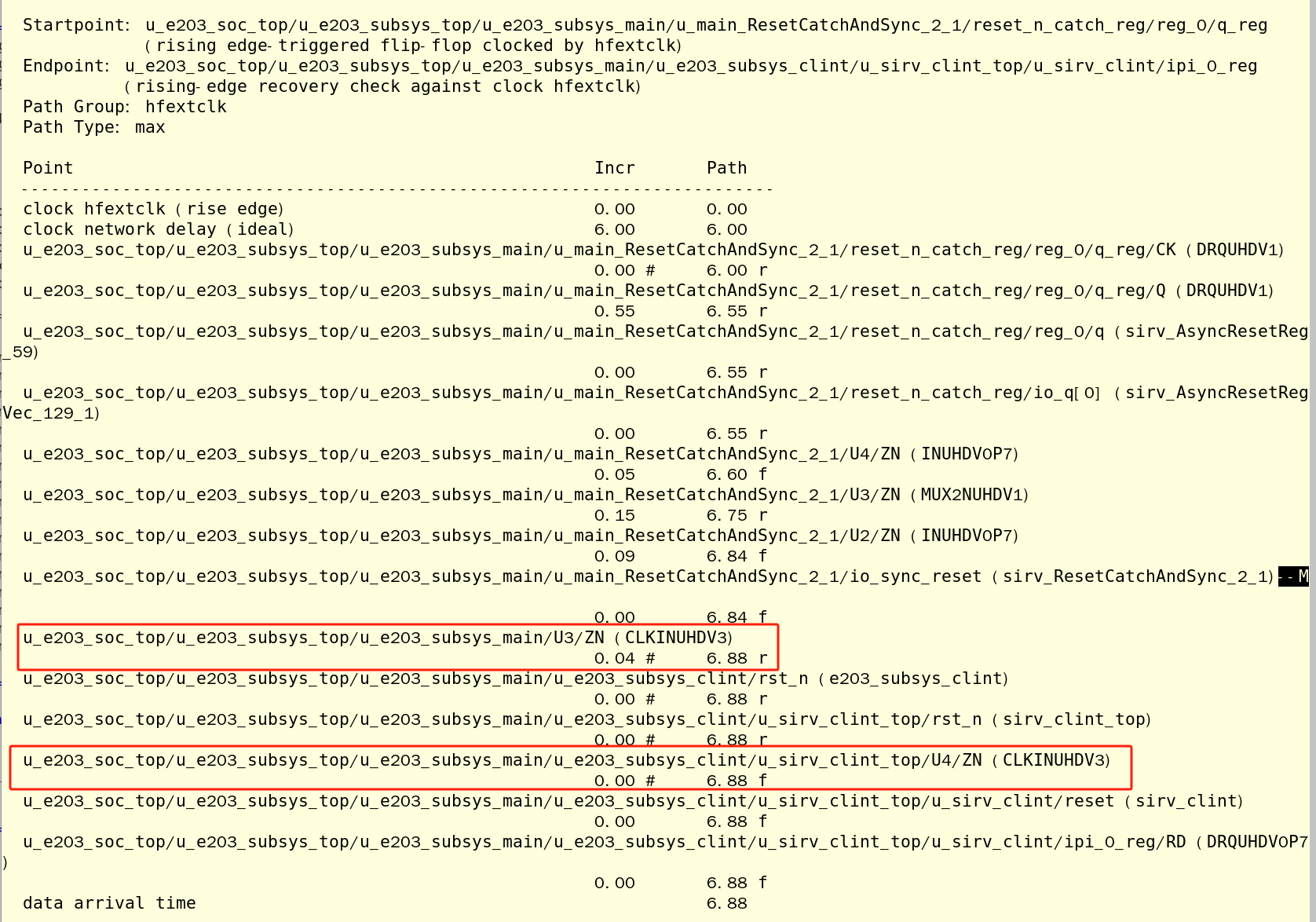
report\_timing -to xxx/RD



后端要去优化复位网络，增强其驱动能力。综合阶段设置复位网络为ideal\_network，是为了暂时忽略复位网络的延时，认为其是理想的，看看其他路径上是否有问题。因此，在cons.tcl中增加约束：

set\_ideal\_network [get\_pins u\_e203\_soc\_top/u\_e203\_subsys\_top/u\_e203\_subsys\_main/U3/ZN]

增加以上约束后，先source cons.tcl（更新约束），再报同一条路径，如下。可以看出当认为复位网络是理想后，路径时序就正常了。而且可以看到设置ideal\_network后，该属性可以传递下去，即设置了U3/ZN为ideal\_network后，U4/ZN也是ideal的了。

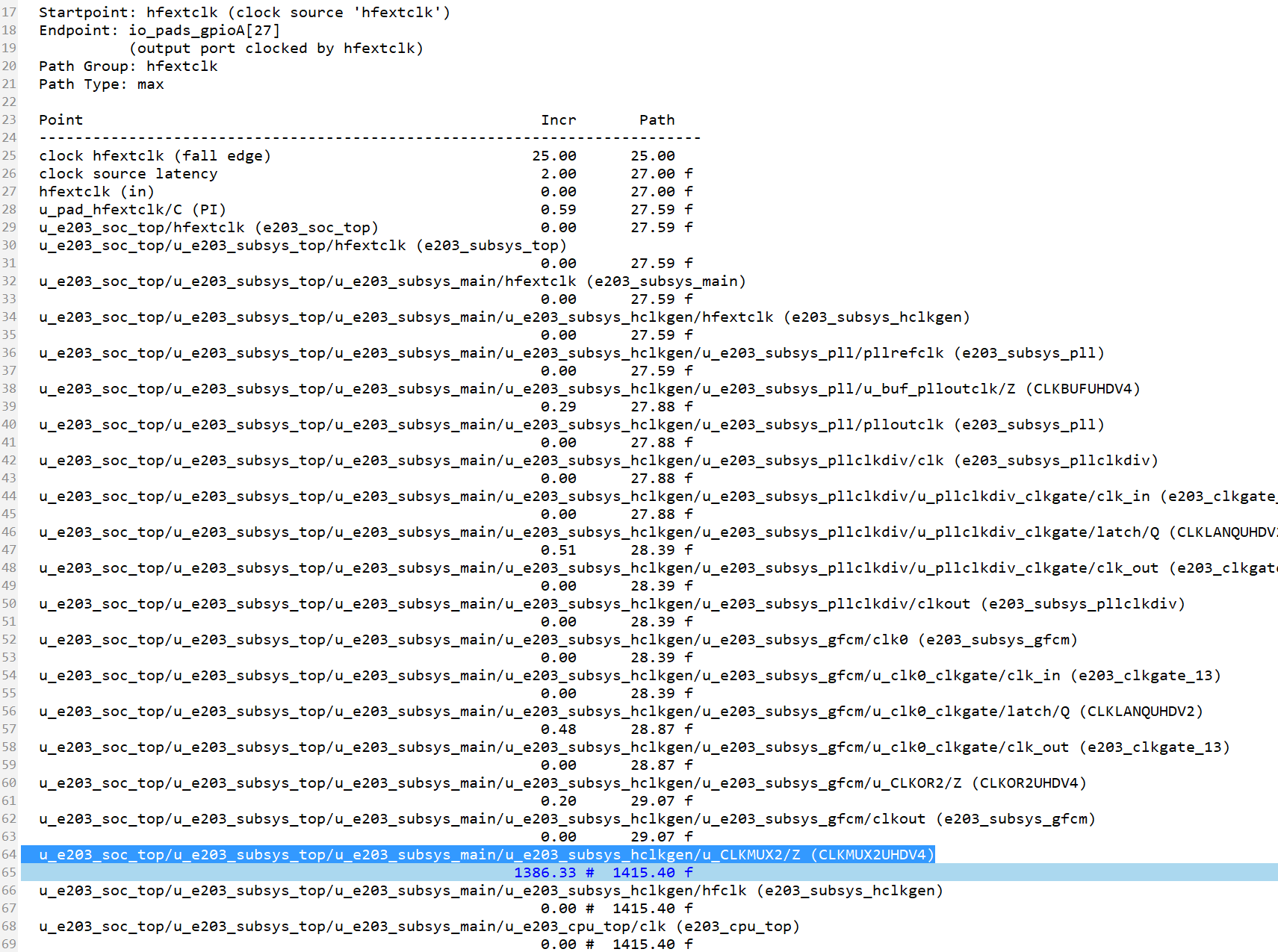


加了以上约束后，再报时序full\_chip\_constraint\_all\_violators.rpt，发现还有一条slack很严重的路径，如下。这条路径起始无需关注，该路径是将时钟输入信号hfextclk送至gpioA[27]输出，然后假设片外再用hfextclk去采集gpioA[27]。

看这条路径，slack这么严重，主要u\_CLKMUX2上的延时大，这符合时钟网络的特点，扇出大，负载大，延时大，需要后端去增加时钟网络驱动能力。

综合阶段设置时钟网络为ideal\_network，是为了暂时忽略时钟网络的延时，认为其是理想的，看看其他路径上是否有问题。因此，在cons.tcl中增加约束：

set\_ideal\_network [get\_pins u\_e203\_soc\_top/u\_e203\_subsys\_top/u\_e203\_subsys\_main/u\_e203\_subsys\_hclkgen/u\_CLKMUX2/Z]

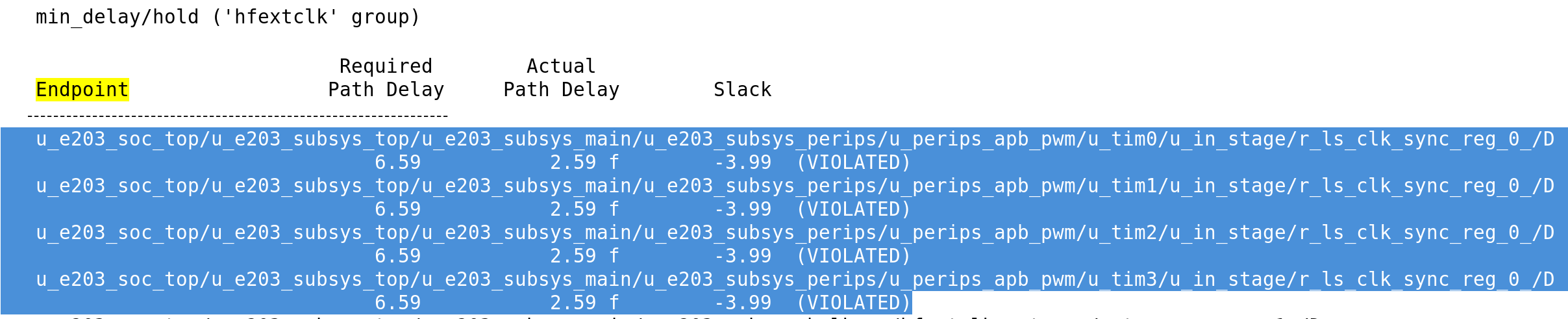


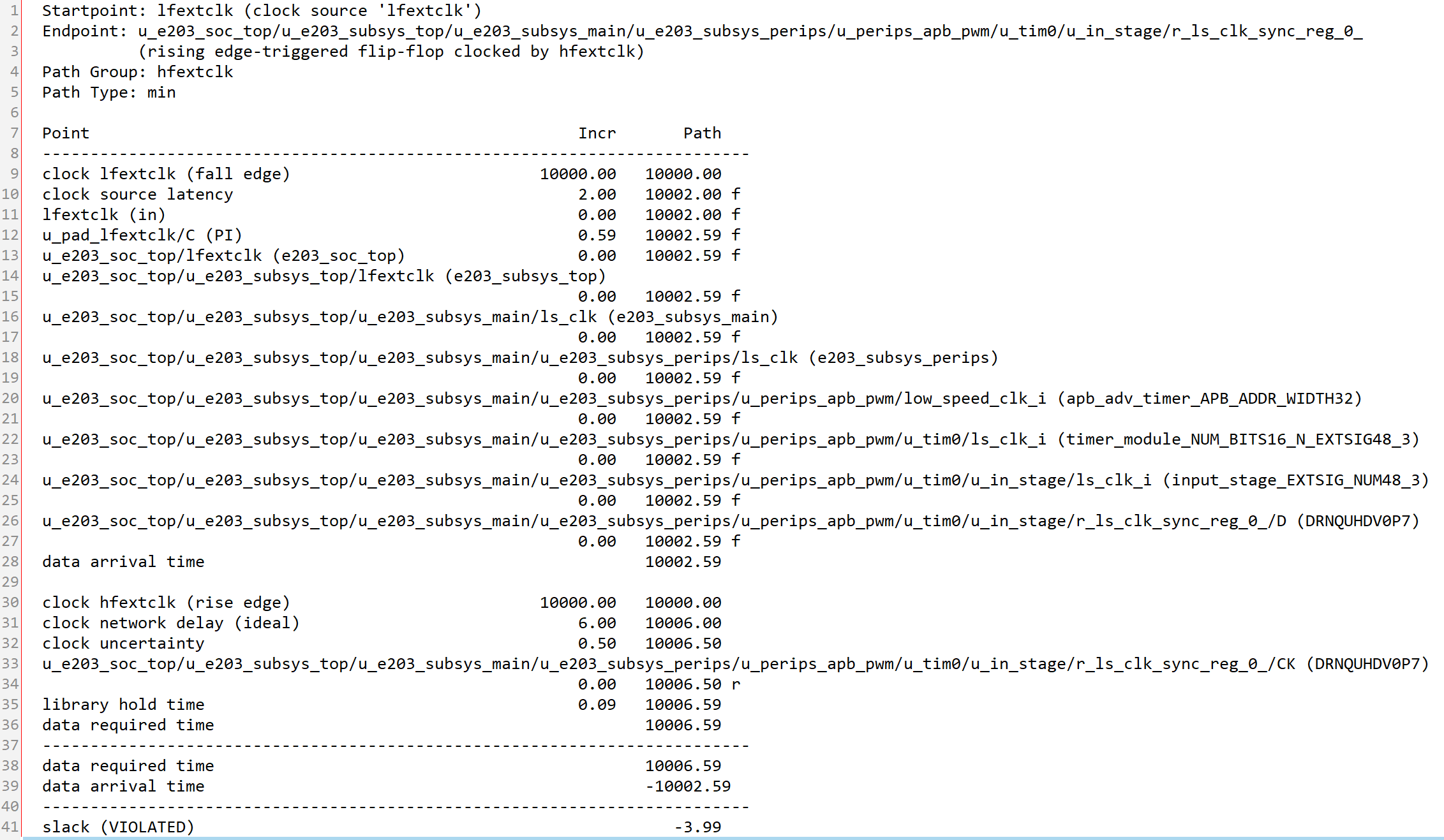
增加以上两处set\_ideal\_network设置后，剩余的路径延时要去仔细检查，随便看一条路径的时序报告。看full\_chip\_constraint\_all\_violators.rpt报告，max\_delay最严重的路径slack为-1.32，这是符合要求的。

* 1. 查看时序报告min\_delay

full\_chip\_constraint\_all\_violators.rpt报告中min\_delay处，有4条路径的min\_delay的slack为-3.99，这个不正常。看了一下，这条路径无需检查时序，因为是从lfextclk到hfextclk域触发器的D端（两个时钟是异步的，从设计上保证不出亚稳态，时序检查就没必要了），如下：

（思考题五：为什么min\_delay的-3.99的slack需要我们去关注？而max\_delay的-1.32的slack则可以认为是正常的？）





将这些路径设为false\_path，即不需要工具去优化时序。这个约束要传到后端，告诉后端工具无需优化这些路径.

set\_false\_path -from lfextclk -to u\_e203\_soc\_top/u\_e203\_subsys\_top/u\_e203\_subsys\_main/u\_e203\_subsys\_perips/u\_perips\_apb\_pwm/u\_tim0/u\_in\_stage/r\_ls\_clk\_sync\_reg\_0\_/D

set\_false\_path -from lfextclk -to u\_e203\_soc\_top/u\_e203\_subsys\_top/u\_e203\_subsys\_main/u\_e203\_subsys\_perips/u\_perips\_apb\_pwm/u\_tim1/u\_in\_stage/r\_ls\_clk\_sync\_reg\_0\_/D

set\_false\_path -from lfextclk -to u\_e203\_soc\_top/u\_e203\_subsys\_top/u\_e203\_subsys\_main/u\_e203\_subsys\_perips/u\_perips\_apb\_pwm/u\_tim2/u\_in\_stage/r\_ls\_clk\_sync\_reg\_0\_/D

set\_false\_path -from lfextclk -to u\_e203\_soc\_top/u\_e203\_subsys\_top/u\_e203\_subsys\_main/u\_e203\_subsys\_perips/u\_perips\_apb\_pwm/u\_tim3/u\_in\_stage/r\_ls\_clk\_sync\_reg\_0\_/D

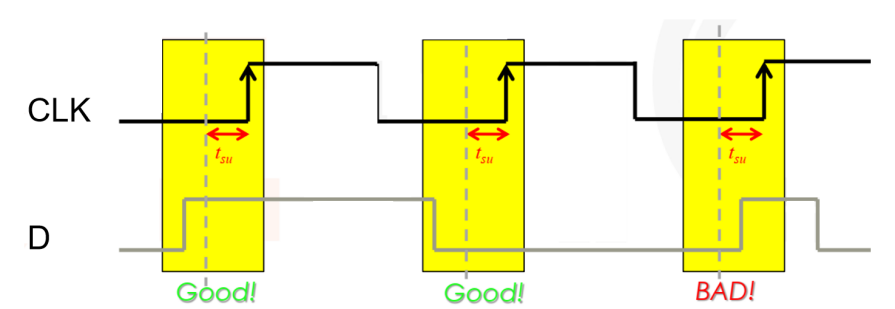
* 1. 回答思考题四、五
  2. 学习要求

1. 综合报告中的各类warning如何处理？
2. 网表中的assign如何处理？
3. Latch如何处理？
4. 综合时序报告中时钟网络和复位网络上的超大延时如何处理？
5. LAB9：逻辑综合3：cons.tcl

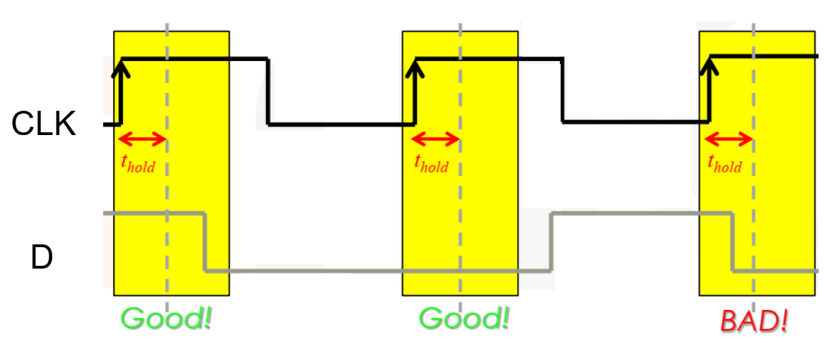
逻辑综合1搭建环境，逻辑综合2消除warning，逻辑综合3设定时序约束。

* 1. 时序检查类型

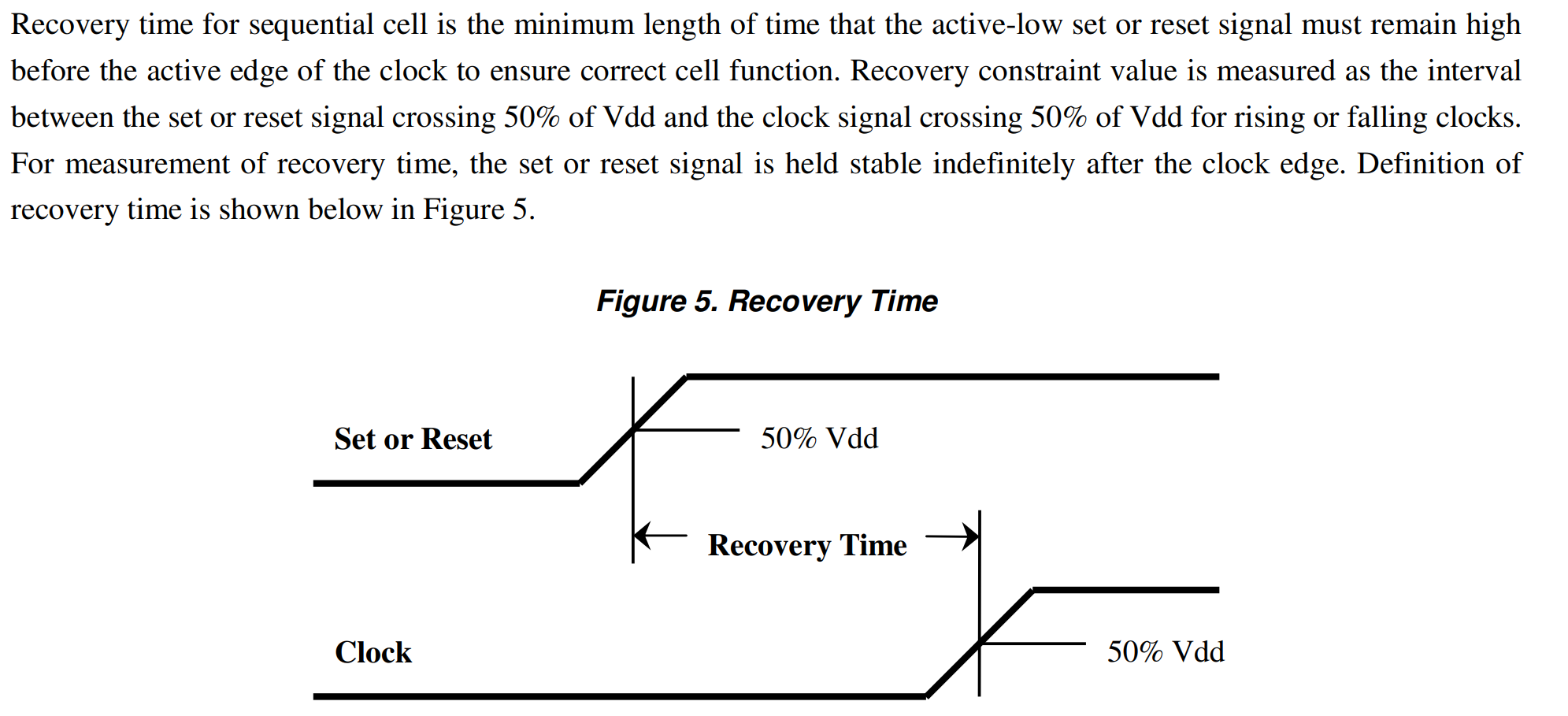
1. 建立时间setup



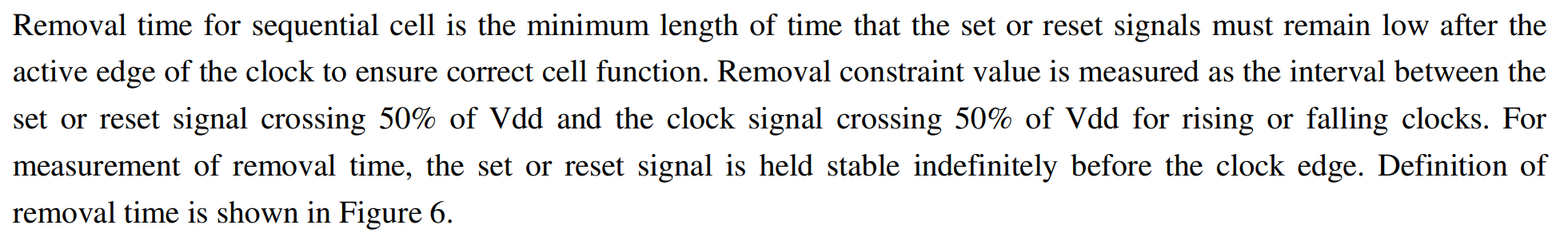
1. 保持时间hold

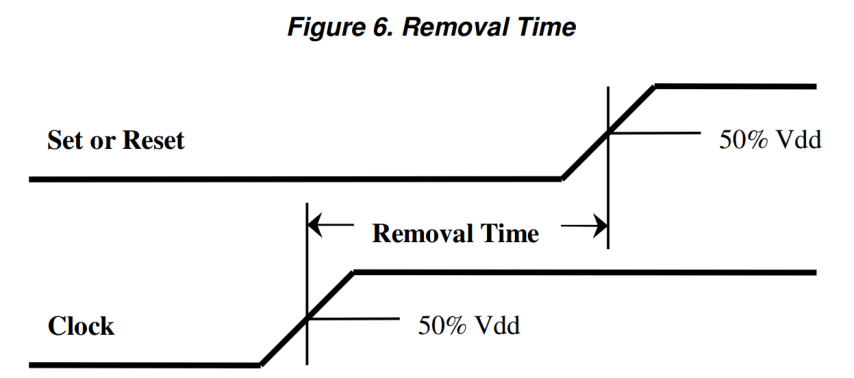


1. Recovery Time

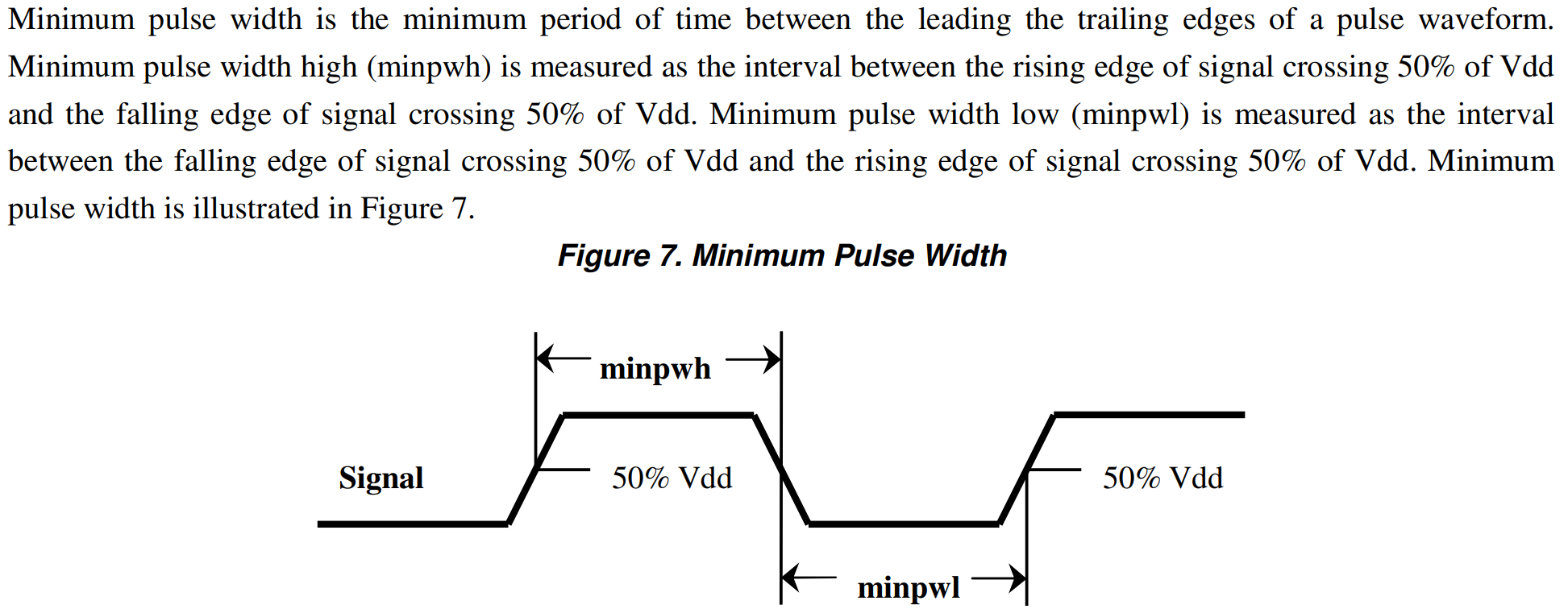


1. Removal Time

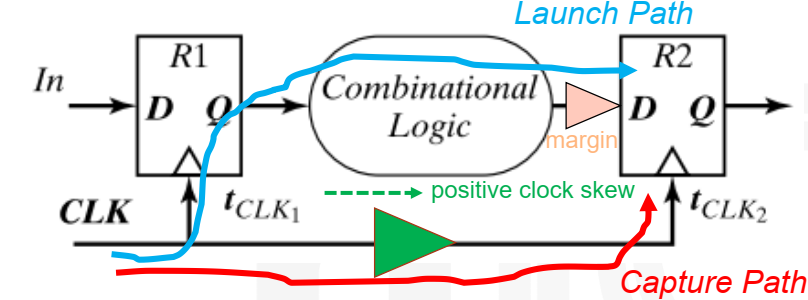




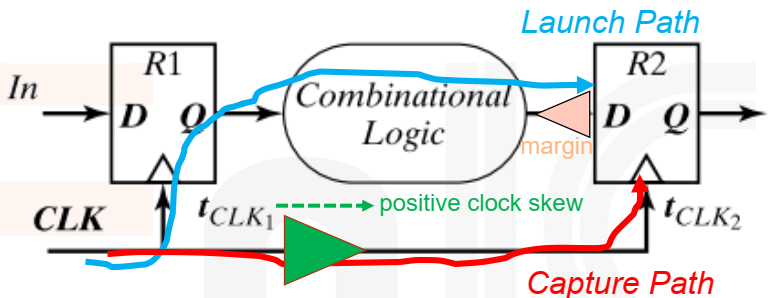
1. Clock Gating
2. Minimum Pulse Width



* 1. 时序约束

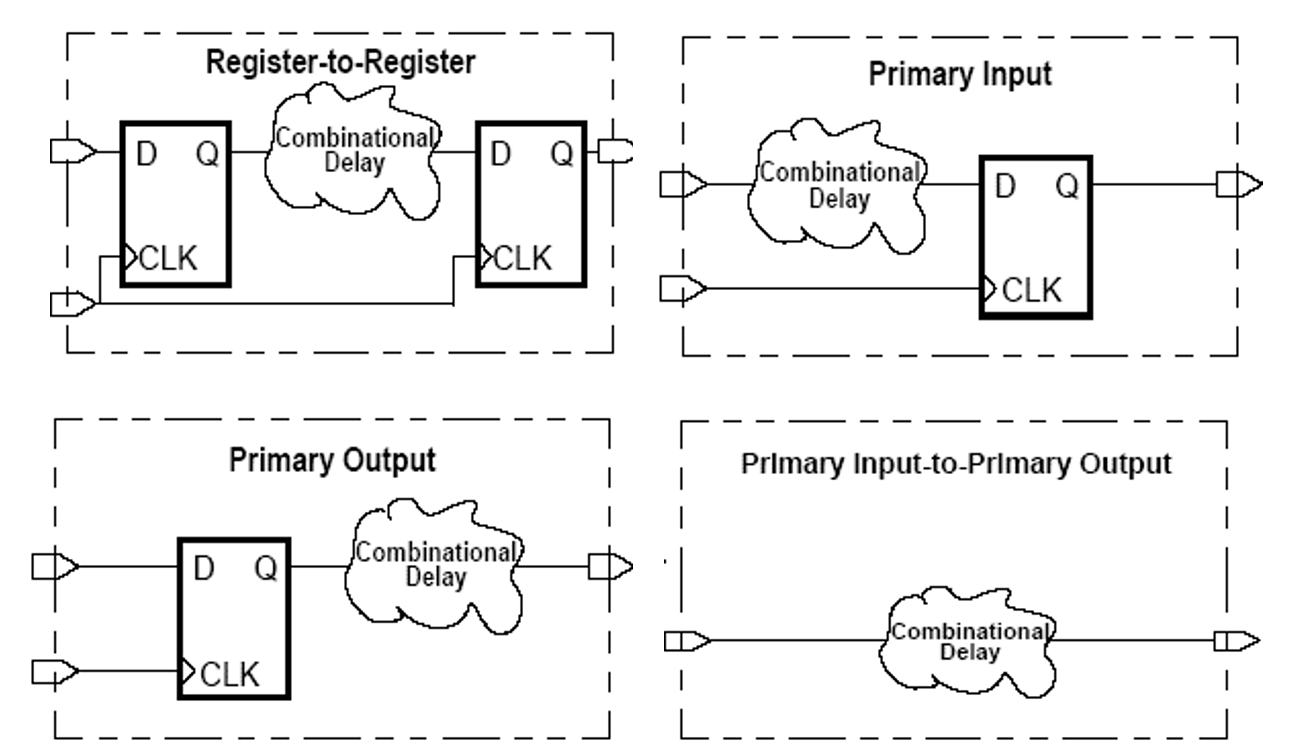


*T + tskew > tcq + tlogic + tsetup + tmargin*



*tcq + tlogic - tmargin > thold + tskew*

* 1. Timing Path: 4类



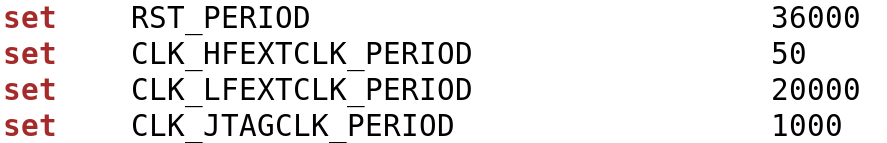
* 1. 脚本cons.tcl

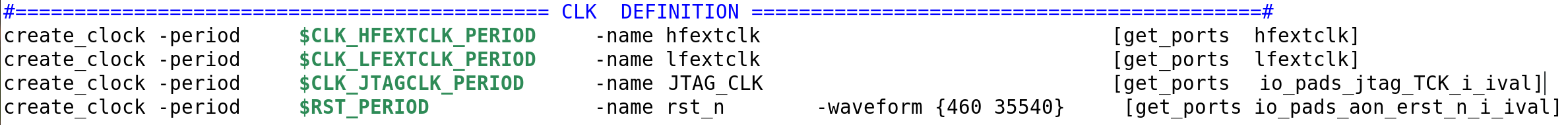
1. 时钟周期的设定

时钟周期反映了芯片的峰值频率，当我们设定了一个时钟周期值，工具会保证数据路径的延时小于单个时钟周期（先简单这么理解）。当时钟周期很长时，工具会选用那些驱动能力不大的门来实现两级触发器之间的组合逻辑，因为这样面积更小，功耗也更小。当我们把时钟周期设定的更小，原先小驱动能力门组成的逻辑路径的延时就超过了时钟周期，就发生建立时间违反setup violation。这是工具就会逐渐选择大驱动能力的门，增加驱动能力，降低路径延时，保证数据路径的延时仍然小于单个时钟周期。这样芯片就可以工作在更高的频率下，代价是面积会增加，功耗会增加。继续降低时钟周期值，工具选择了最强驱动能力的门，还是无法保证所有的路径延时都小于时钟周期，那么这就是当前设计和当前工艺下，芯片性能的极限。如果还想提高芯片性能，一方面可以换更先进的工艺，标准单元门电路的延时天然就会降低；另一方面去修改RTL设计，将路径延时最长的关键路径用寄存器从中间隔断，这样路径延时就变成二分之一。事实上我们在RTL时，要有一种模糊的概念，即整个设计中任意两个寄存器之间的路径延时应该保持平衡，芯片的峰值频率是由最长的那条路径决定的。最后我们还可以牺牲一些设计余量，例如把set\_clock\_uncertainty值改小等，但这种方式提供非常有限，而且是以牺牲良率和稳定性为代价。

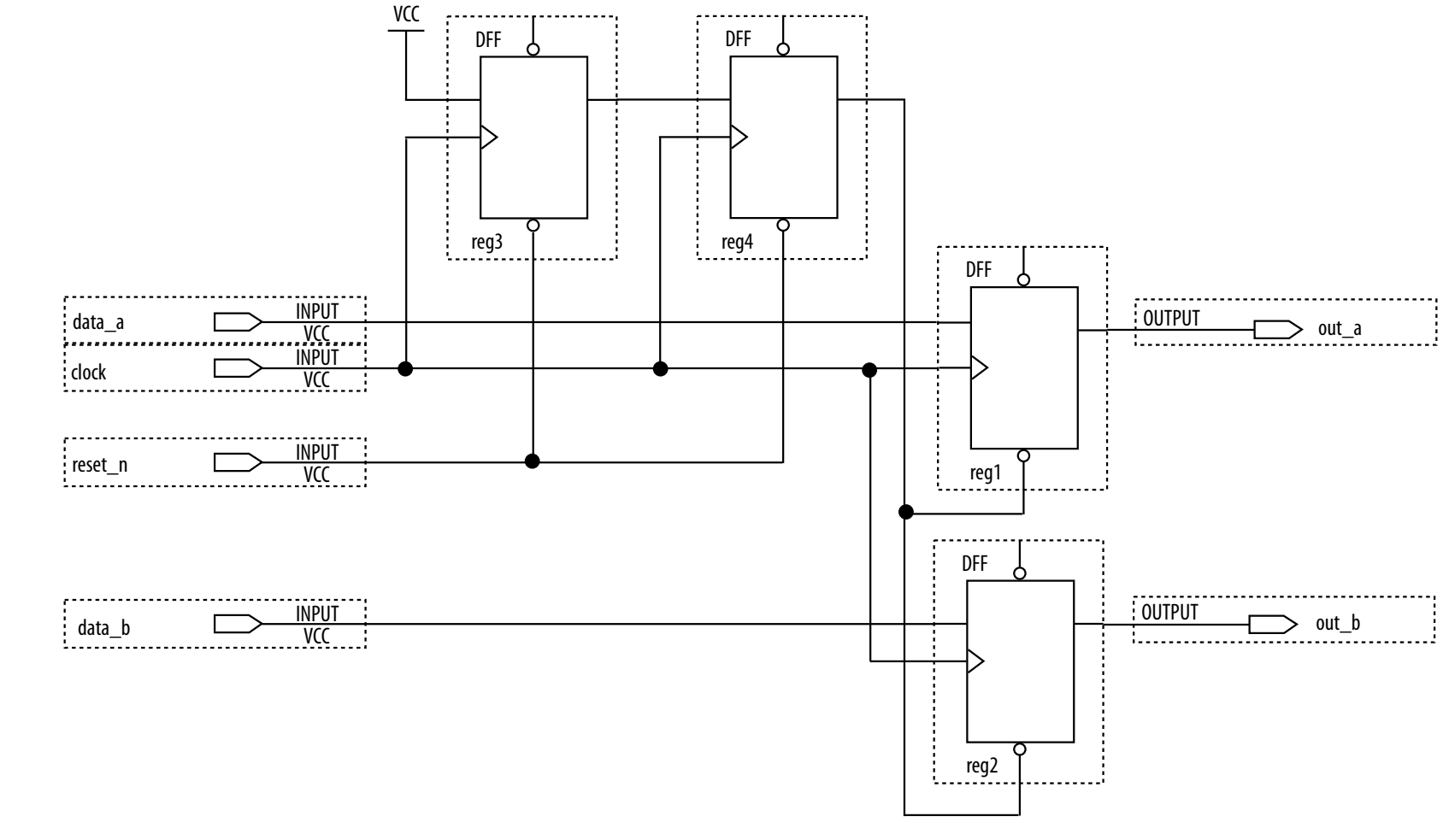
可见，综合时时钟周期的设定，决定了芯片的峰值频率，某种程度上对芯片面积和功耗也有影响。该如何设定时钟周期？我们芯片的目标频率是16MHz，综合阶段我们可以把时钟周期设定为50ns，即频率20Mhz（比目标频率提高20%）。综合结果显示最严重的setup violation的slack为-1.32ns，意味着只要频率稍微降低1.32ns，整个时序就pass了。可见对于SMIC180nm下我们的设计，综合工具几乎没有压力。还有一点，综合阶段将目标频率提高的目的：后端做完时钟树综合和寄生参数提取后，时序情况会恶化，所以综合阶段的目标频率要留出余量。

这个芯片三个时钟hfextclk，lfextclk和jtag\_TCK，时钟周期分别如下

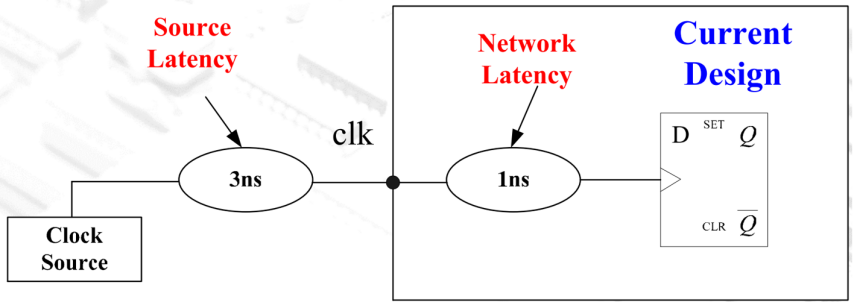




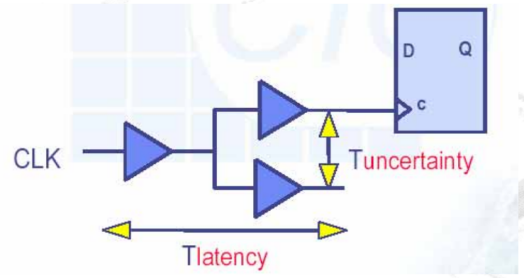
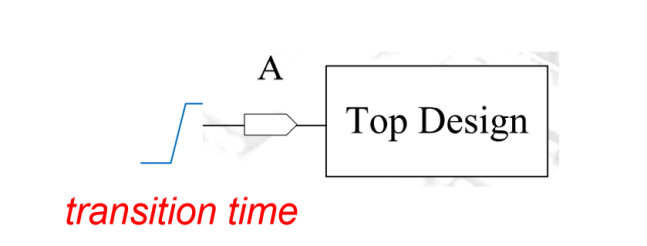
D触发器除了检查setup和hold以外，还需要检查recovery和removal，这里把复位信号也当成时钟，不管是否报recovery和removal问题，都要清楚实际应用中，复位信号随时可能进入芯片或者离开芯片（和三个时钟是异步的），所以recovery和removal violation无法完全避免。如何解决由于复位信号引起的时序问题？答案：异步复位，同步释放。



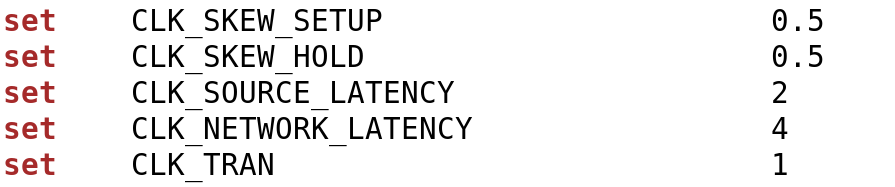
1. 时钟skew, latency, transition

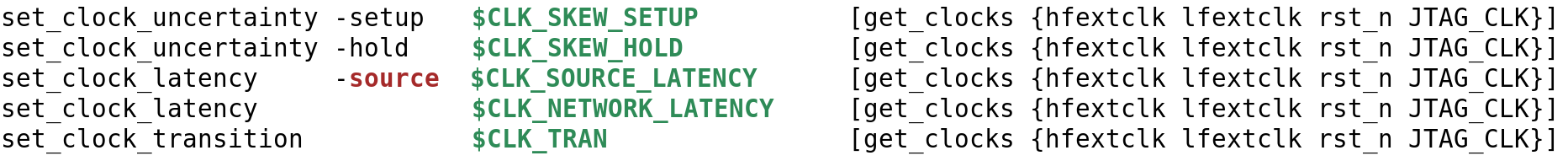




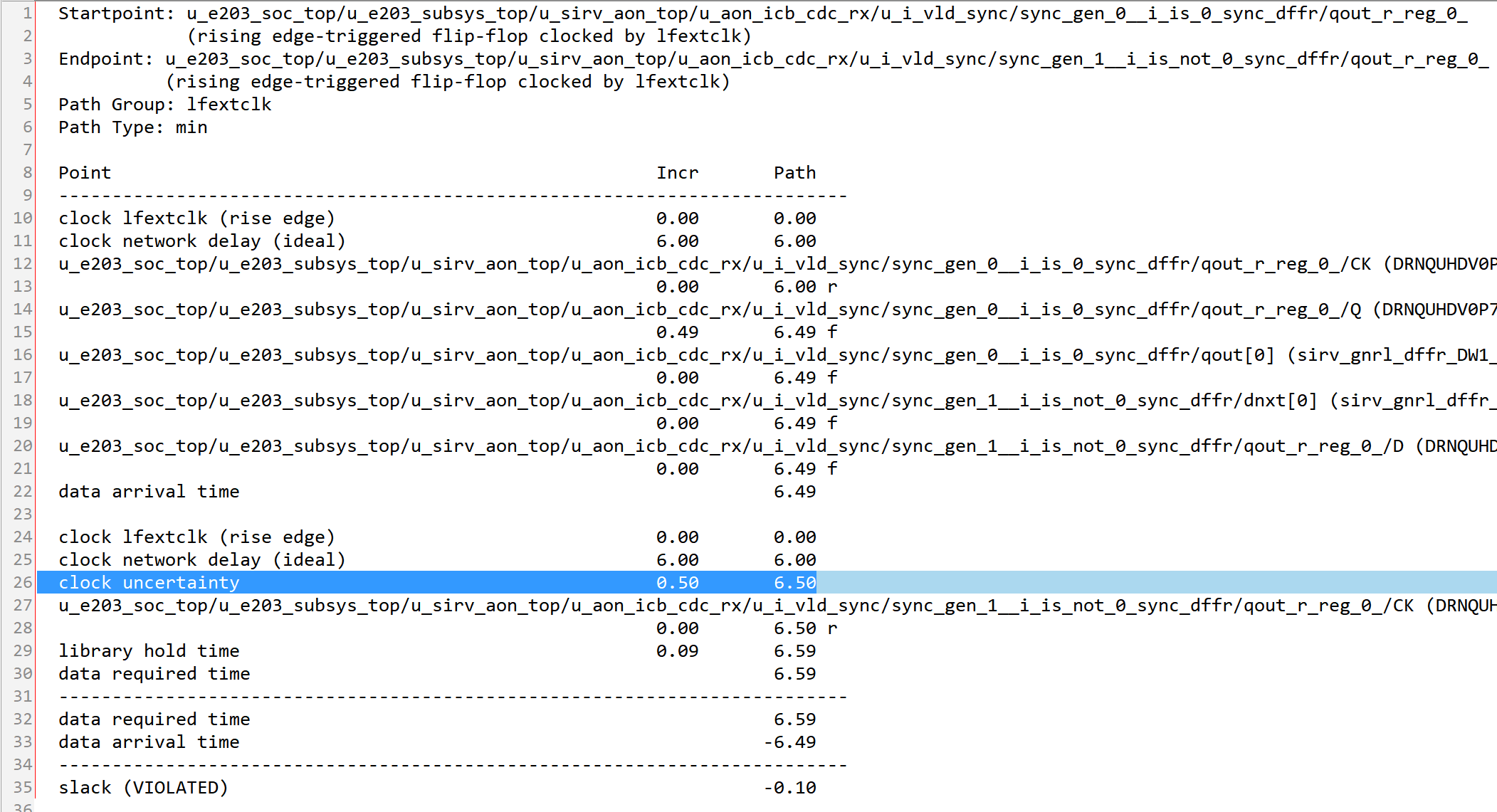
 

这个芯片时钟参数设定如下：



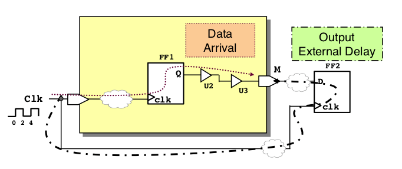
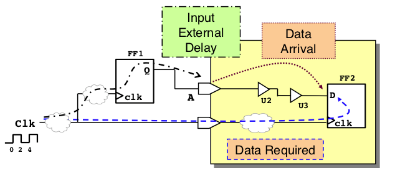


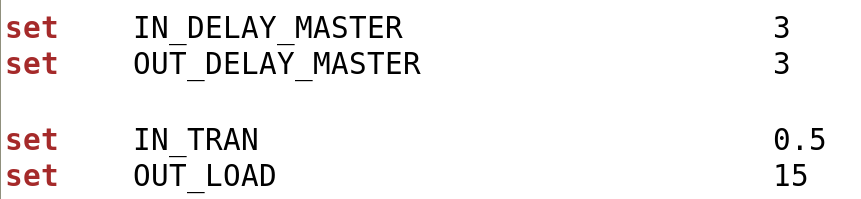
发现hold的slack值和set\_clock\_uncertainty直接相关，这个值不可太大，否则工具需要加一堆buffer，保证数据路径延时足够大，不发生hold violation。

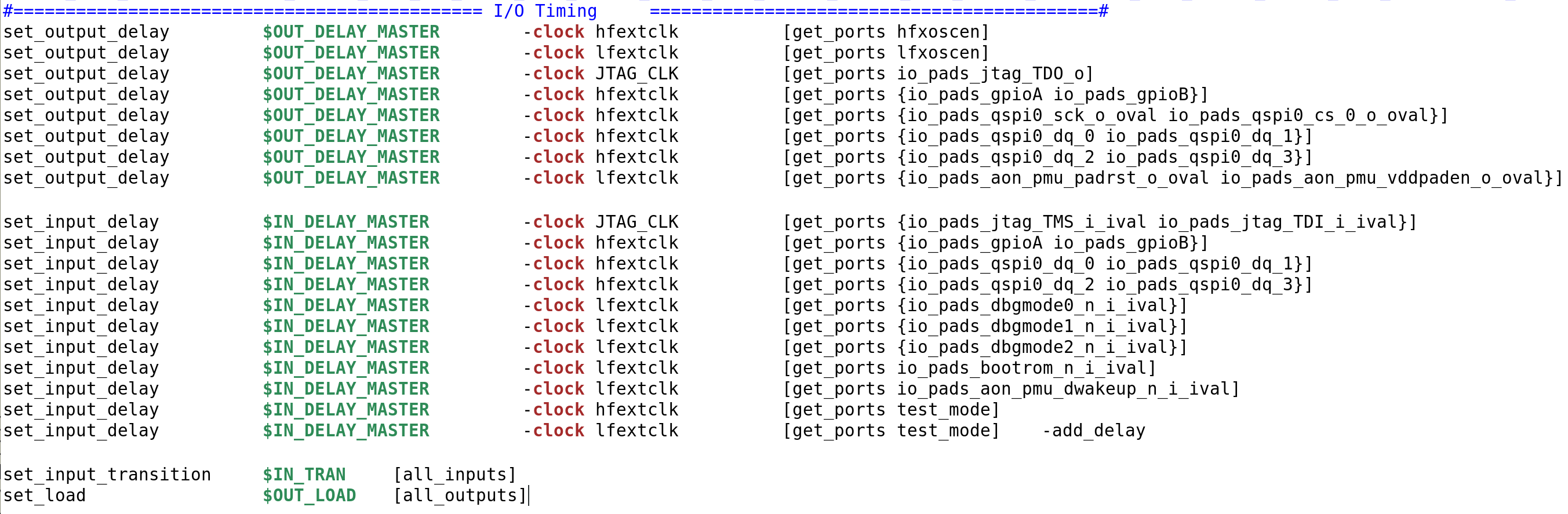


1. Input/Output Delay, Input Transition, Output Load

信号和哪个时钟域相关（即由哪个时钟域采集或者发射），就关联到哪个时钟域上。

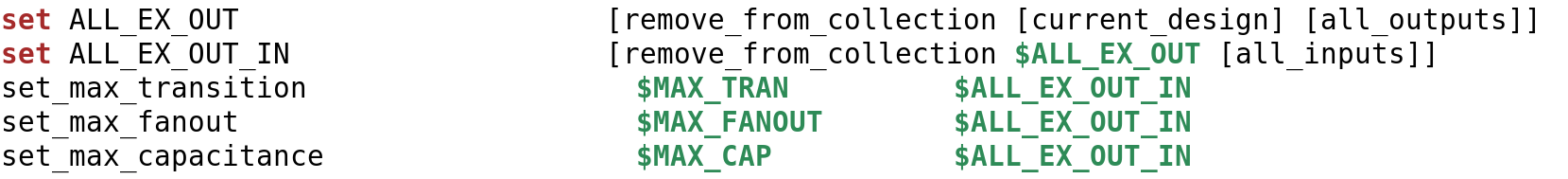






1. set\_max\_transition, set\_max\_fanout和set\_max\_capacitance





标准单元库中有default\_max\_transition约束，tt下为2.75。ss下为3.03。ff下为2.47。我们设定一个相对严格的值1.5。和2.4的transition相比，综合结果会付出一些面积代价，小于1%。

标准单元中有max\_capacitance约束，规定cell输出能驱动的最大负载，但是每个cell的max\_capacitance是不一样的，从0.20156到3.2378不等。我们这里统一设定为0.25。后端完成后，cons.tcl中去掉max\_capacitance约束，就让工具按照库里的max\_capacitance去查就好了。

（思考题六：修改设计约束，看看综合结果会有什么变化？并尝试解释该变化。例如，通过set\_clock\_uncertainty设置时钟偏斜情况，对比修改前后的时序报告和面积报告。）

* 1. 回答思考题六
  2. 学习要求：

1. 时序检查
2. 时序约束
3. 设计余量
4. LAB10：逻辑综合4：提交网表和约束
   1. 数据检查

完成逻辑综合，提交后端数据之前，再做以下检查：

1. dc.log中的Latch，error和warning信息，再次确认。
2. full\_chip\_mapped.v中为了避免assign，增加的10个CLKBUFUHDV4是否都在？时钟网络上手工加的cell（CLKOR，CLKMUX，CLKLANQ）是否都在？检查sirv\_clint模块是否例化CLKINUHDV3，只有输入，没有输出？
3. full\_chip\_mapped\_power.rpt和full\_chip\_mapped\_area.rpt报告的面积和功耗没有明显增加。
4. full\_chip\_report\_timing\_requirements.rpt报告high-fanout网络，不管。
5. full\_chip\_mapped\_clock\_gating.rpt可以查看门控时钟报告。
6. full\_chip\_check\_timing\_final.rpt没有warning，没有未约束的寄存器。
7. full\_chip\_mapped\_timing.rpt报告每个时钟域中max\_delay的关键路径。
8. full\_chip\_constraint\_all\_violators.rpt报告显示有很严重的DRC（max\_tran，max\_cap，max\_fanout）违反。通过逐个分析，发现这些违反多是由于时钟网络、复位网络、iso网络负载太大导致，以及有4条false path导致。综合阶段无需对这些违反做任何操作，交由后端处理即可。
   1. 数据交付后端

将./netlist/full\_chip\_mapped.v和./results/full\_chip\_mapped.sdc打包交付后端。同时，综合文件夹./syn备份，待查。

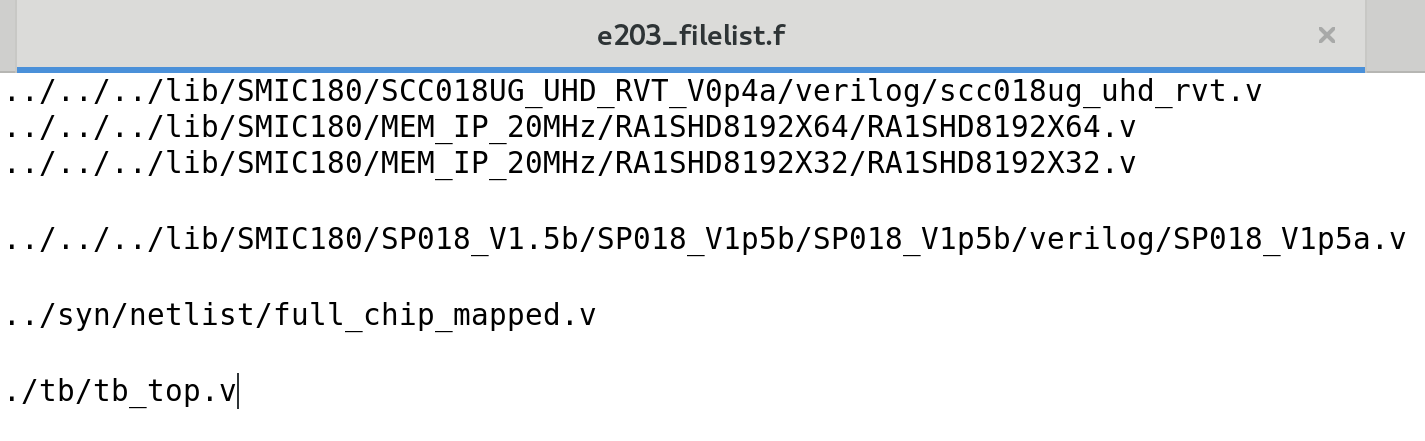
* 1. 学习要求

1. 逻辑综合过程检查
2. 逻辑综合交付数据检查
3. LAB11：后仿真post\_sim

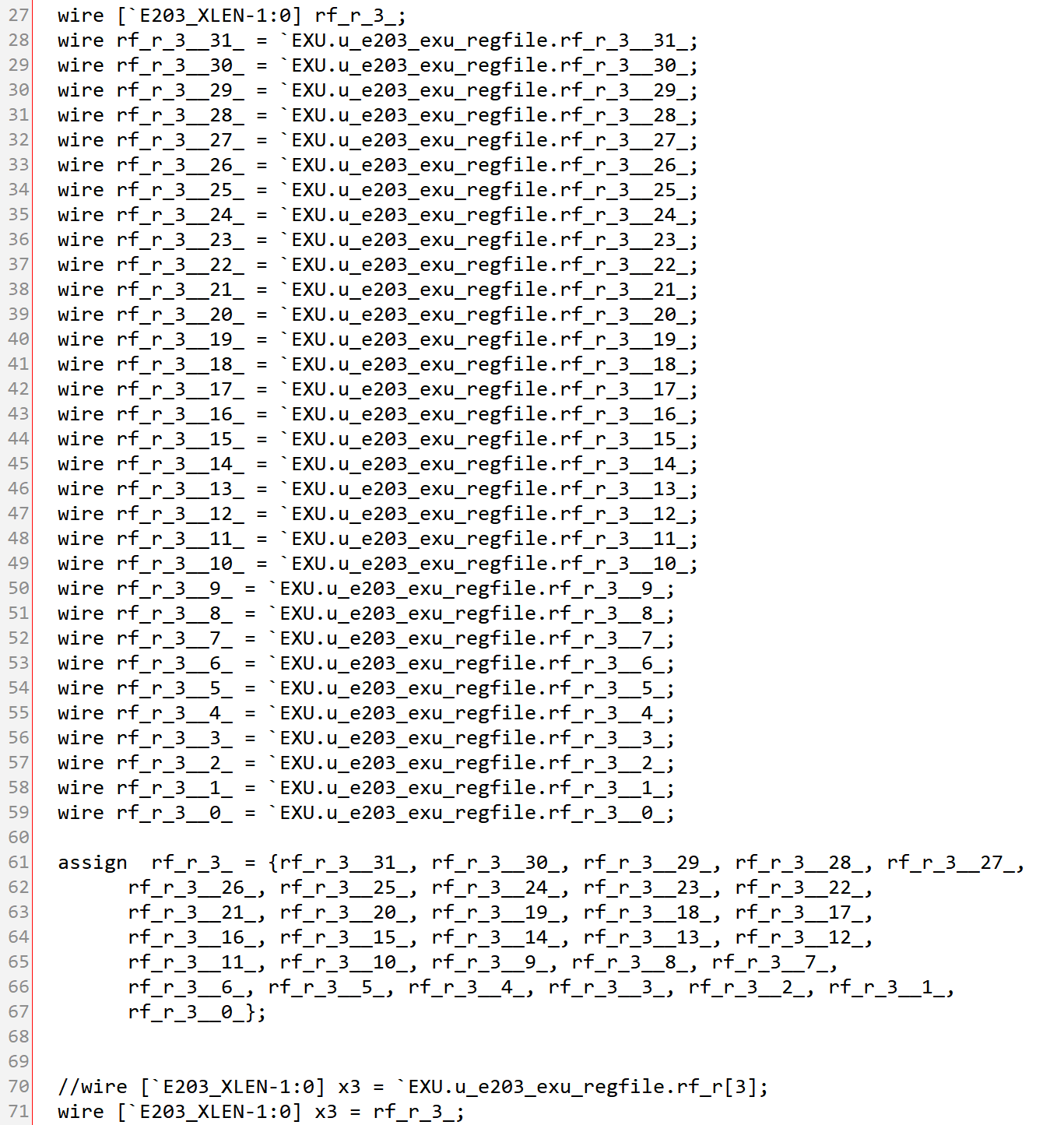
对综合后的网表进行后仿真。

* 1. 建立后仿环境

1. 建立目录./post\_sim/。拷贝./pre\_sim/config.v, e203\_defines.v过来，这些文件tb会用到。拷贝./pre\_sim/tb/。拷贝./pre\_sim/riscv-tools/。拷贝run\_vcs.csh，run\_verdi.csh和e203\_filelist.f。
2. e203\_filelist.f修改如下



1. tb\_top.v修改如下，就可以仿真了。



1. 课程实践报告

以小组为单位提交word格式的课程实践报告，报告命名为：BUPT\_SoC实践报告\_小组xx\_20241216.docx，要求书写规范，格式统一。实践报告内容上应至少包含以下几方面：

→ 完成以上所有LAB，记录实践过程，越详细越好。

→ 完成六道思考题。

→ 随实践报告一并提交的数据包括：./pre\_sim/rtl/，./post\_sim/tb/，./syn/netlist/，./syn/reports/，./syn/dc.log，以上数据打包压缩，命名为：BUPT\_SoC\_groupxx\_20241216.zip

→ 小组成员合影靓照。