

# ASIC SVN环境介绍及RTL上传SVN流程

## 历史记录

版本	作者	修订日期	修订内容描述	审核人	批准人
V1.0	harry.ning	2021.5.15	初始版		
V1.1	harry.ning	2021.8.25	更新tags版本使用流程		
v1.2	harry.ning	2021.9.23	更新crg IP使用与替换流程		
V1.3	harry.ning	2021.10.21	更新lint规则waive流程		
V1.4	harry.ning	2021.10.25	更新cdc检查流程		

## ASIC SVN环境介绍及RTL上传SVN流程

1. SVN database下载和使用
  - 1.1 获取trunk版本工程
  - 1.2 获取tag版本工程
2. 研发环境及工作流程介绍
  - 2.1 svn switch
  - 2.2 memory规格申请
  - 2.3 G\_define
3. DE RTL上传SVN规范与流程
  - 3.1 RTL代码上传规范
  - 3.2 RTL检查 (Lint、CDC、RDC) 流程
    - 3.2.1 Lint检查
      - 3.2.1.1 Lint检查设置黑盒
      - 3.2.1.1 Lint检查waive方式
    - 3.2.2 CDC检查
      - 3.2.2.1 常见CDC检查错误
    - 3.2.2 RDC检查
  - 3.3 RTL上传SVN流程
  - 3.4 Xml寄存器生成流程
  - 3.5 Xml文件填写注释
  - 3.6 note生成寄存器流程
4. CRG IP使用流程
5. Memory规格生成及使用方法

## 1. SVN database下载和使用

第一次使用Linux需要配置相关环境变量。

```
cp /remote/public/upload/hw_home_bk/Makefile $HOME
```

使用make执行本地设计载入

```
make remote_load
```

## 1.1 获取trunk版本工程

Everest IC开发SVN database下载和使用方法如下：

首先，如何check out databse。

在你的home下的.cshrc文件中添加

```
setenv SVNROOT svn://172.16.50.24/asic
```

保存后退出，然后执行

```
source .cshrc
```

然后执行

```
cd /remote/project/everest/user  
  
mkdir your_name  
  
cd your_name  
  
mkdir everest_v100r001_ws0  
  
cd everest_v100r001_ws0  
  
svn co --depth=empty $SVNROOT/checkout_list/everest_v100r001/everest_v100r001 .  
###注意这里有个“.”，别漏了 : )  
  
svn up everest_v100r001_co_list  
  
./everest_v100r001_co_list
```

到这里，database就下载下来了。接下来就是如何使用。

使用database前先确认自己的IP目录在哪里。确认好后，source两个cshrc文件：

```
cd /remote/project/everest/user/your_name/everest_v100r001_ws0/database  
source cshrc_dbv1  
  
cd  
/remote/project/everest/user/your_name/everest_v100r001_ws0/database/soc/hw/ips/  
digital/your_ip/vx/  
source cshrc_your_ip_vx
```

如果你是第一次使用该database，请运行

```
cy_prep
```

每次新开一个terminal，仍然需要source上面两个cshrc，并执行 `cy_prep`

到这里，你的工作环境就OK了。接下来就是如何开始工作。

开始工作前先确认自己身份，是DE还是DV：

如果你是DE (RTL designer) , 你的工作区域主要在 \$RTL。所有为本模块新开发的RTL代码全部check in到这里, 同时维护一些methodology的东西;

如果你是DV (verification) , 你的工作区域主要在 \$TB, \$TS 和 \$VER。顾名思义, \$TB 里放的是testbench相关, \$TS 里是test case, \$VER里全部是一些验证组件和脚本等。

## 1.2 获取tag版本工程

tag版本供系统验证或调试团队等相关team使用, 否则使用正常版本。

首先, 如何check out databse。

在你的home下的.cshrc文件中添加

```
setenv SVNROOT svn://172.16.50.24/asic
```

保存后退出, 然后执行

```
source .cshrc
```

然后执行

```
cd /remote/project/everest/user

mkdir your_name

cd your_name

mkdir everest_v100r001_ws0_tags

cd everest_v100r001_ws0_tags

svn co --depth=empty $SVNROOT/checkout_list/everest_v100r001/everest_v100r001 .
###注意这里有个“.”, 别漏了 : )

svn up everest_v100r001_co_list_tags

./everest_v100r001_co_list_tags
```

到这里, database就下载下来了。接下来就是如何使用。

首先设置一个环境变量:

```
setenv SOC_PATH
/remote/project/everest/user/<your_name>/everest_v100r001_tags/database/soc
```

然后将编译器由默认工具切换到VCS

```
setenv SIM_TOOL VCSM
```

接下来进入自己需要的design层级:

顶层:

```
cd $SOC_PATH/hw/chips/everest/asic/full_chip
```

IP:

```
cd $SOC_PATH/hw/ips/digital/<your_ip>/vx
```

3rd party IP:

```
cd $SOC_PATH/hw/third_ips/<your_ip>/v1
```

source cshrc文件:

顶层:

```
source cshrc_chips_full_chip
```

IP/3rd party IP:

```
source cshrc_<your_ip>_vx
```

如果你是第一次使用该database, 请运行 `cy_prep`。后续使用就不需要再运行该命令了。但是新开一个 prompt, 你仍然需要 `source` 上述 `cshrc`。

如需编译, 进入上述目录下的rtl子目录,

```
make rtl_compile MESSAGE=1
```

之后, 如需使用verdi加载design

```
verdi -f dut.f
```

## 2. 研发环境及工作流程介绍

### 2.1 svn switch

项目开发到一定阶段, 不会所有人都停留在trunk分支上工作, trunk上随时都会有人更新代码, 影响当前系统的稳定, 所以集成同事需要将自己开发要用到的工作目录switch trunk, 其他目录停留在tags上。

在自己的工作目录和需要在trunk上的目录, 执行下面命令。

```
cydb_switch
```

在当前工作目录开发到新的feature成熟后, 将此目录打上新的tag发布。

### 2.2 memory规格申请

### 2.3 G\_define

编译仿真时, 如果使用的 `define`, 需要新建 `G_define.svh` 文件。在其中添加define名

注意该文件不能加到 `filelist.f` 中, 只保存在本地。

## 3. DE RTL上传SVN规范与流程

## 3.1 RTL代码上传规范

进入到你的工作目录 `$RTL`

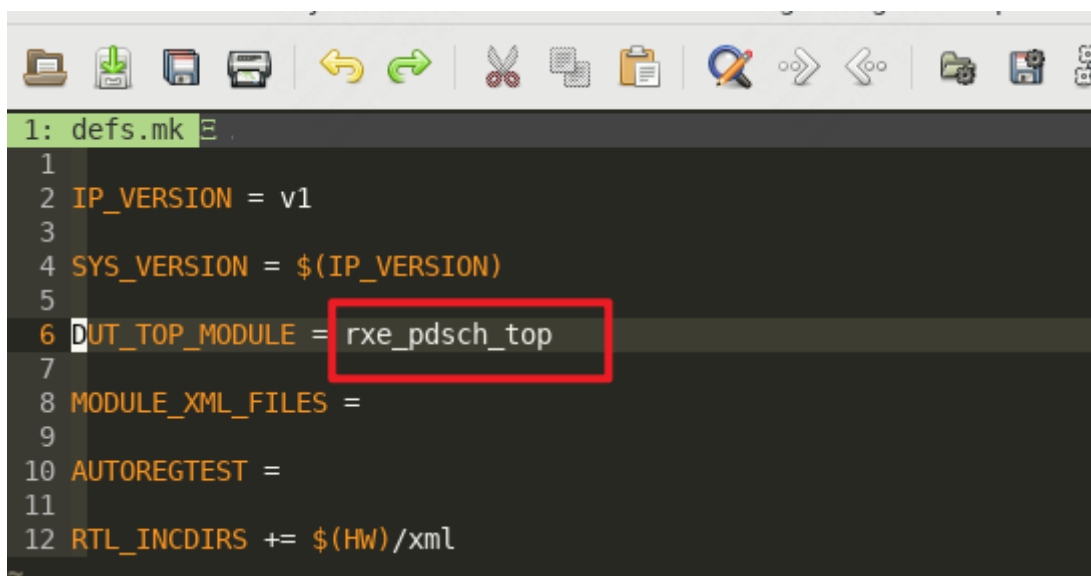
将本模块新开发的代码全部check in到这里，在上传至SVN服务器之前需要清除编译error和lint。

首先保证source了上面的两个cshrc，并且执行过 `cy_prep`，然后进入你的IP 目录

```
cd $HW/ips/digital/your_ip/vx/rtl/
```

目录下存放所有RTL文件和 `filelist.f`，top的命名方式必须为 `you_ip_vx_top`，否则无法编译通过。

将 `$HW/ips/digital/your_ip/v1/` 目录下的 `def.mk` 中的 `DUT_TOP_MODULE` 修改成自己的top名，例如 `rxp_pdsch_top`，然后将修改后的 `defs.mk` 也上传。



```
1: defs.mk
1
2 IP_VERSION = v1
3
4 SYS_VERSION = $(IP_VERSION)
5
6 DUT_TOP_MODULE = rxp_pdsch_top
7
8 MODULE_XML_FILES =
9
10 AUTOREGTEST =
11
12 RTL_INCDIRS += $(HW)/xml
```

`filelist`中只添加你的RTL目录下的文件，文件格式如下

```
1 $HW/ips/digital/nr_phy/v1/rtl/axi_interconnect/rtl/nic400_1.v
2 $HW/ips/digital/nr_phy/v1/rtl/axi_interconnect/rtl/nic400_amib_master_0_1.v
3 $HW/ips/digital/nr_phy/v1/rtl/axi_interconnect/rtl/nic400_amib_master_0_chan_slice_1.v
4 $HW/ips/digital/nr_phy/v1/rtl/axi_interconnect/rtl/nic400_asib_slave_0_1.v
5 $HW/ips/digital/nr_phy/v1/rtl/axi_interconnect/rtl/nic400_asib_slave_0_chan_slice_1.v
6 $HW/ips/digital/nr_phy/v1/rtl/axi_interconnect/rtl/nic400_asib_slave_0_decode_1.v
7 $HW/ips/digital/nr_phy/v1/rtl/axi_interconnect/rtl/nic400_asib_slave_0_maskcntl_1.v
8 $HW/ips/digital/nr_phy/v1/rtl/axi_interconnect/rtl/nic400_asib_slave_0_rd_ss_cdas_1.v
9 $HW/ips/digital/nr_phy/v1/rtl/axi_interconnect/rtl/nic400_asib_slave_0_wr_ss_cdas_1.v
```

调用的别人的模块代码和第三方IP，需要在 `filelist.mk` 中 `RTL_INCIPS_DIR` 添加所调用模块/IP的路径，多个IP将本行复制多份修改

```
46 RTL_INCIPS_DIR += $(HW)/ips/digital/phy_tx/v1/rtl
47 RTL_INCIPS_DIR += $(HW)/ips/digital/tx_frm/v1/rtl
48 RTL_INCIPS_DIR += $(HW)/ips/digital/fft/v1/rtl
49 RTL_INCIPS_DIR += $(HW)/ips/digital/txb_pusch/v1/rtl
50 RTL_INCIPS_DIR += $(HW)/ips/digital/txs_pusch/v1/rtl
51 RTL_INCIPS_DIR += $(HW)/ips/digital/txf_map/v1/rtl
52 RTL_INCIPS_DIR += $(HW)/ips/digital/main_ctrl/v1/rtl
53 RTL_INCIPS_DIR += $(HW)/ips/digital/rfd/v1/rtl
54 RTL_INCIPS_DIR += $(HW)/ips/digital/axi_adaptor/v1/rtl/axi_conv_top
55 RTL_INCIPS_DIR += $(HW)/ips/digital/ip_common/v1/rtl
```

调用的头文件需要在 `filelist.mk` 中 `RTL_INCDIRS` 添加，所有用到的头文件都要添加，多个include将本行复制多份修改

```

57 RTL_INCDIRS += $(HW)/ips/digital/nr_phy/v1/rtl/axi_interconnect/rtl/def
58 RTL_INCDIRS += $(HW)/ips/digital/txb_pusch/v1/rtl/include
59 RTL_INCDIRS += $(HW)/ips/digital/nr_phy/v1/rtl/include
60 RTL_INCDIRS += $(HW)/ips/digital/ip_common/v1/rtl

```

如果与其他模块都共用的IP，如：mult\_asic.v TPSR.v 等，都在

```
$HW/ips/digital/ip_common/v1/rtl/
```

目录下，为避免重复编译，每个IP自己维护的 fileList.f 里面不要加这些IP，在 fileList.mk 中添加

```

54 RTL_INCIPS_DIR += $(HW)/ips/digital/axi_adaptor/v1/rtl/axi_conv_top
55 RTL_INCIPS_DIR += $(HW)/ips/digital/ip_common/v1/rtl
56
57 RTL_INCDIRS += $(HW)/ips/digital/nr_phy/v1/rtl/axi_interconnect/rtl/def
58 RTL_INCDIRS += $(HW)/ips/digital/txb_pusch/v1/rtl/include
59 RTL_INCDIRS += $(HW)/ips/digital/nr_phy/v1/rtl/include
60 RTL_INCDIRS += $(HW)/ips/digital/ip_common/v1/rtl

```

使用xml寄存器流程生成的regfile，默认在 \$XML 目录下，直接在 fileList.f 中添加。

```

1: fileList.f
1 $HW/ips/digital/main_ctrl/v1/rtl/main_ctrl_v1_top.v
2 $HW/ips/digital/main_ctrl/v1/rtl/main_ctrl.v
3 $HW/ips/digital/main_ctrl/v1/rtl/nr_phy_ahb_dec.v
4
5 $XML/MAIN_CTRL_V1_reg.sv
6 $XML/INT_MANAGMT_V1_reg.sv
7 $XML/PUBLIC_AHB_V1_reg.sv

```

关于 xml 生成regfile流程，查看3.4—3.5节xml格式文件生成寄存器流程，note格式生成寄存器流程。

**注意：**生成的regfile不要上传，只本地使用，只把修改过的.xml文件在 \$HW/xml 目录上传至svn版本库。

设置好filelist后在rtl目录下执行 make rtl\_compile，进行rtl编译，

```
make rtl_compile MESSAGE=1
```

编译的语法或filelist路径问题会直接在终端显示，信息较多时可以查看rtl目录下生成的 comp.log 查看编译信息。

想要切换到VCS编译器

```
setenv SIM_TOOL VCSM
```

然后再按上面的流程执行一遍。

## 3.2 RTL检查 (Lint、CDC、RDC) 流程

### 3.2.1 Lint检查

使用工具先需要在 \$HOME 目录下的 .cshrc 文件中添加下面的module load。

```

# load lint cdc rdc
module load AscentLint/2019.A.P20.2021.06.14
module load MeridianCDC/2019.A.P17.2021.06.08
module load MeridianRDC/2019.A.P16.RDC.2021.06.03

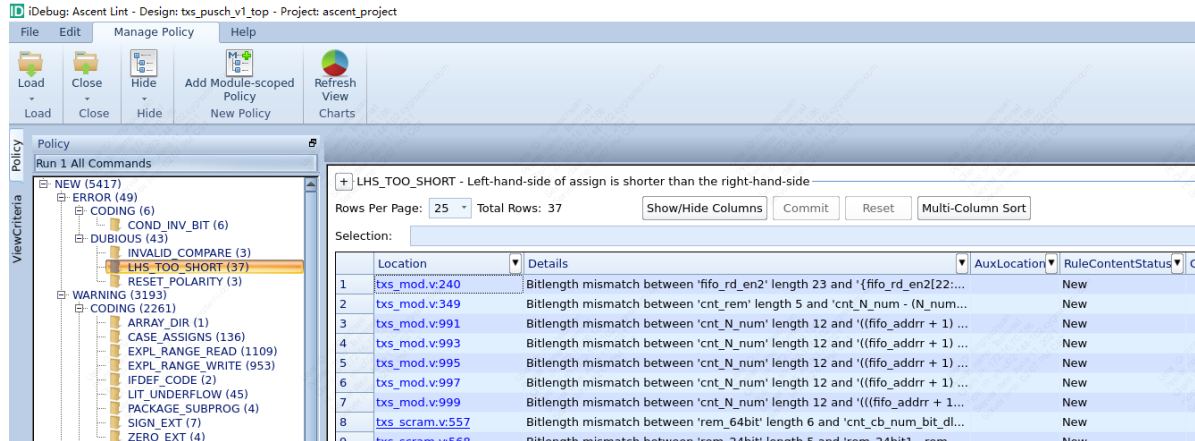
```

直接运行环境脚本。

运行命令前确保source过上述两个cshrc文件。

```
# 运行脚本
make alint

# 打开图形界面
idebug &
```



使用本地tcl脚本运行。

在目录 `$SOC_PATH/tools/aLint` 下将Real intent脚本

`alint_run.tcl` copy到自己的IP 目录 `rtl` 下

打开 `alint_run.tcl` 文件

```
1 # Set a top name
2 #set top dfi_top_wrapper
3
4
5 # Use a policy simply by sourcing the file
6 if { [file exists $env(ALINT_POLICY_FILE)] } {
7   source $env(ALINT_POLICY_FILE)
8 } elseif { [file exists alint.policy] } {
9   source alint.policy
10 } elseif { [file exists alint.local.policy] } {
11   source alint.local.policy
12 } elseif { [file exists alint.policy.local] } {
13   source alint.local.policy
14 } else {
15   source $env(SOC_PATH)/tools/aLint/alint.policy
16 }
17
18 # Compile design files
19 #-exclude_modules {dfi_top_wrapper}
20 read_liberty -f lib.f
21 #analyze /remote/project/everest/user/zhengzhewen/everest_v100r001_tags/database/soc/hw/third_ips/ddr_ctrl/v1/rtl/dfi_top_wrapper.v -sv
22 exclude_modules {tc5250_wrapper}
23 analyze -f dut.f -sv
24 #-analyze -f lint_lib.f -sv
25
26 #-elaborate $top -black_box {dfi_top_wrapper} -auto_black_box
27 elaborate $env(LINT_TOP) -auto_black_box
28
29 # Generate text report
30 report_policy ALL -verbose -compat -output $env(LINT_TOP).alint.rpt
```

第22行如果是调用的第三方成熟的IP，可以忽略该IP，在 {} 内部加上IP的top名，没有用的话注释掉其余不变

Lint检查加define，Lint检查默认不加define，全部检查。

### 3.2.1.1 Lint检查设置黑盒

#### 3.2.1.1 Lint检查waive方式

Lint检查过程中对于第三方IP和需要waived掉的error或warning。提供下面waive的方式。

编写 `alint_exclude_module.tcl`，格式如下：

```

1 # Compile design files
2 # third ips
3 exclude_modules {nic400_nr_M}
4 exclude_modules {nic400_nr_S}
5 exclude_modules {tc5310}
6 exclude_modules {tc5210}
7 exclude_modules {tc5350}
8 exclude_modules {tc5250}
9 exclude_modules {DW_axi_gm_core}
10 exclude_modules {phy_clock_gate}
11 exclude_modules {scc14nsfp_90sdb_9tc16_rvt}

```

然后在将\$TOOLS/aLint/alint\_run.tcl, copy到本地。将alint\_exclude\_module.tcl添加到脚本下面位置。

```

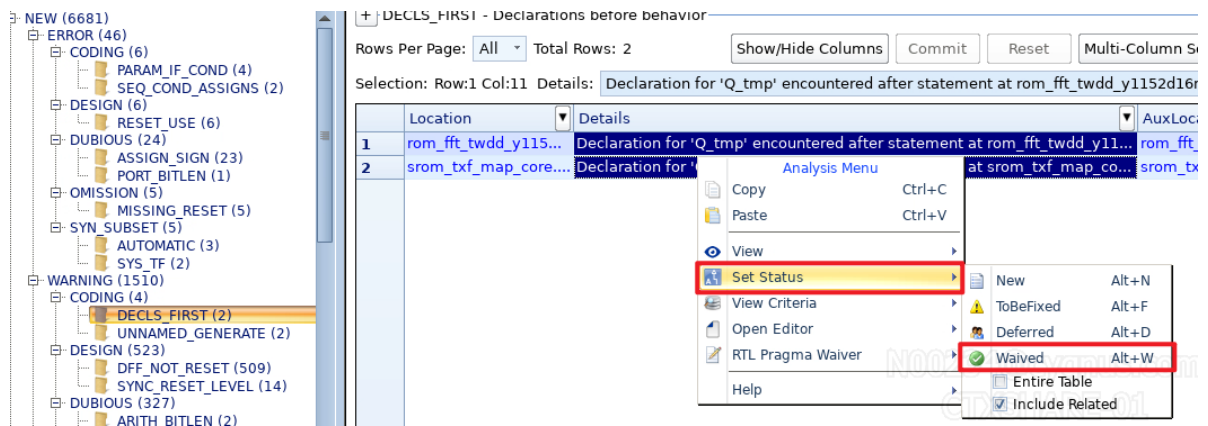
12 } elseif { [file exists alint.policy.local] } {
13     source alint.local.policy
14 } else {
15     source $env(SOC_PATH)/tools/aLint/alint.policy
16 }
17
18 source alint_exclude_module.tcl
19

```

对于需要waive的error/warning。跑完lint, 打开idebug。

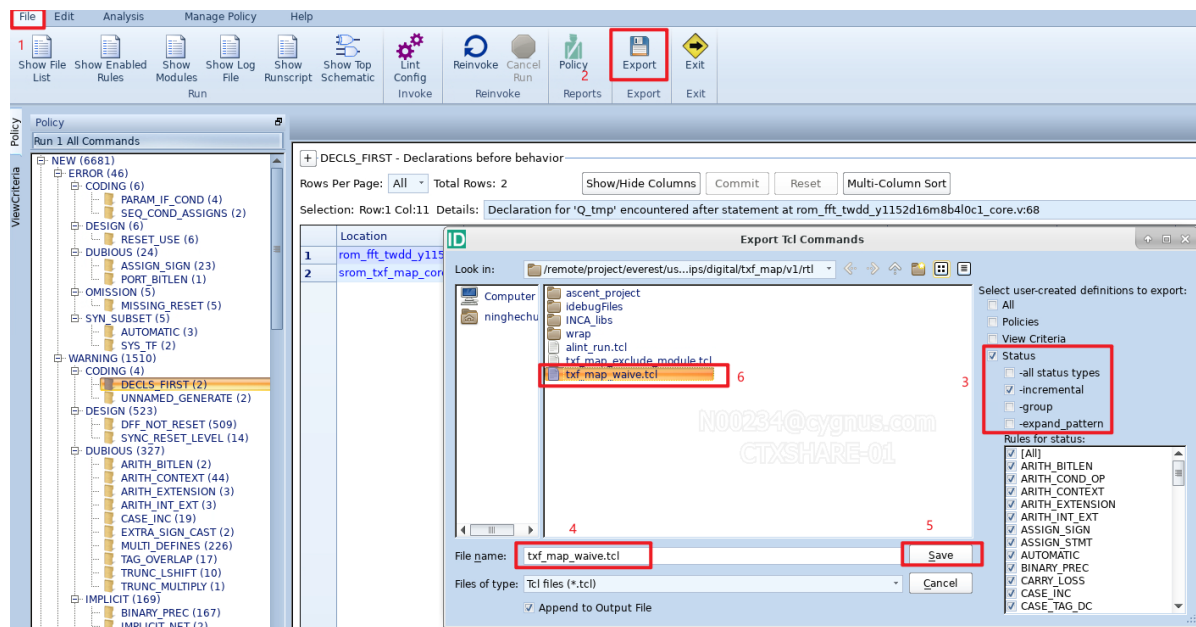
idebug &

选中需要waive的报告, 右键点击, 设置status为waived。



点击File->Export->勾选status & incremental->alint\_waive.tcl->save。





在alint\_run.tcl如下位置中添加，alint\_waive.tcl。

```

23 #analyze /remote/project/everest/user/zhengzhewen/everest_v100r001_tags/dat
24 analyze -f dut.f -sv -ignore_translate_off {synopsys}
25 ##analyze -f lint_lib.f -sv
26
27 #-#elaborate $top -black_box {dfi_top_wrapper} -auto_black_box
28 elaborate $env(LINT_TOP) -auto_black_box
29 source alint_waive.tcl
30 # Generate text report
31 report_policy ALL -verbose -compat -output $env(LINT_TOP).alint.rpt

```

重新跑一遍alint。

```
make alint
```

上文exclude module和waive lint rule就会生效。

每个IP的alint\_exclude\_module.tcl和alint\_waive.tcl文件需要上传至\$RTL目录。给上一级集成模块使用，进行lint清除。

harden级的模块直接调用子模块的alint\_exclude\_module.tcl和alint\_waive.tcl。

```

24
25 #-#elaborate $top -black_box {dfi_top_wrapper} -auto_black_box
26 elaborate $env(LINT_TOP) -auto_black_box
27
28 source $env(HW)/ips/digital/txf_map/v1/rtl/alint_waive.tcl
29
30 # Generate text report
31 report_policy ALL -verbose -compat -output $env(LINT_TOP).alint.rpt

```

### 3.2.2 CDC检查

使用工具先需要在 \$HOME 目录下的 .cshrc 文件中添加下面的module load。

```

# load lint cdc rdc
module load AscentLint/2019.A.P20.2021.06.14
module load MeridianCDC/2019.A.P17.2021.06.08
module load MeridianRDC/2019.A.P16.RDC.2021.06.03

```

直接运行环境脚本。

运行命令前确保source过上述两个cshrc文件。

准备好sdc，放在\$RTL目录下，sdc文件可以上传至svn

```
# 运行脚本
make mcdc

# 打开图形界面
idebug &
```

iDebug: Meridian CDC - Design: m\_phy\_v1\_top - Project: meridian\_project

File Edit Manage Policy Help

Load Close Hide Add Module-scoped Policy Refresh View Charts

Policy Run 1 sce\_env:All Commands

ViewCriteria

NEW (7072)

- MCDC\_SETUP\_CHECKS (6176)
  - ERROR (4)
    - S\_NORST (3)
    - S\_INPUT\_NO\_WAVE (1)
  - REVIEW (12)
    - CLK\_GROUPS (3)
    - BLACK\_BOX (9)
  - INFO (6160)
    - CONSTANT (6160)
  - MCDC\_ANALYSIS\_CHECKS (896)
    - ERROR (441)
      - W\_CNTL (2)
      - W\_DATA (404)
      - W\_GLITCH (24)
      - W\_FANOUT (2)
      - W\_RECON\_GROUPS (8)
      - W\_G\_CLK\_GLITCH (1)

W\_DATA - Reports unsafe (non-protected) non-synchronized crossings.

Rows Per Page: 25 Total Rows: 404 Show/Hide Columns Commit Reset Multi-Column Sort

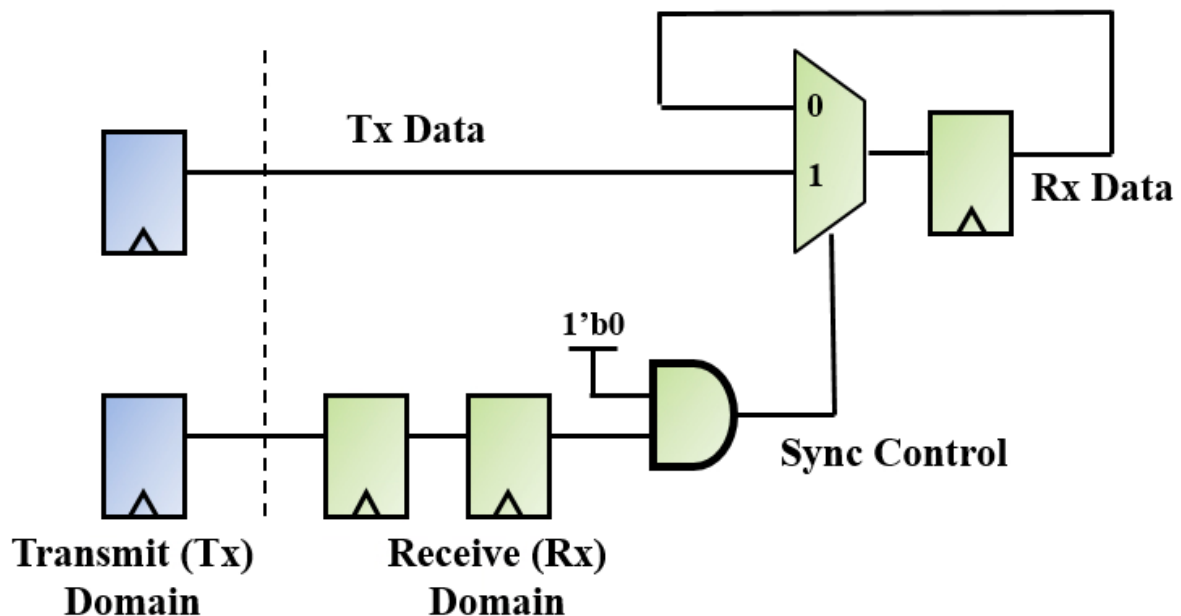
Selection:

RuleDataId	Signal	ReceivingFlop	ClockDomains	Location
1	tpu_trigger[4]	u_rfd_top.rxf_rfd_top_inst_inst...	main_ctrl_clk::clk_491p52m	rfd_rx_adapter.v:469
2	tpu_trigger[3]	u_rfd_top.rxf_rfd_top_inst_inst...	main_ctrl_clk::clk_491p52m	rfd_rx_adapter.v:458
+ 3(2)	tpu_trigger[0]	u_rfd_top.txs_inst.txs_rfd_inst...	main_ctrl_clk::clk_491p52m	tx_buffer_top.v:131
4	u_rxb_pdsch_to...	u_rxb_pdsch_top.process_inst...	clk_491p52m::clk_614p4m	tc5210_tcfifo_ctrl_sr.v:175
5	u_rxb_pdsch_to...	u_rxb_pdsch_top.process_inst...	clk_491p52m::clk_614p4m	tc5210_tcfifo_ctrl_sr.v:173
+ 6(2)	u_rxb_pdsch_to...	u_rxb_pdsch_top.process_inst...	clk_491p52m::clk_614p4m	tc5210_tcfifo_sa_sr_ds.v:1
+ 7(2)	u_rxb_pdsch_to...	u_rxb_pdsch_top.process_inst...	clk_491p52m::clk_614p4m	tc5210_tcfifo_sa_sr_ds.v:1
8	u_rxb_pdsch_to...	u_rxb_pdsch_top.process_inst...	clk_491p52m::clk_614p4m	tc5210_axi_stream_out_pr

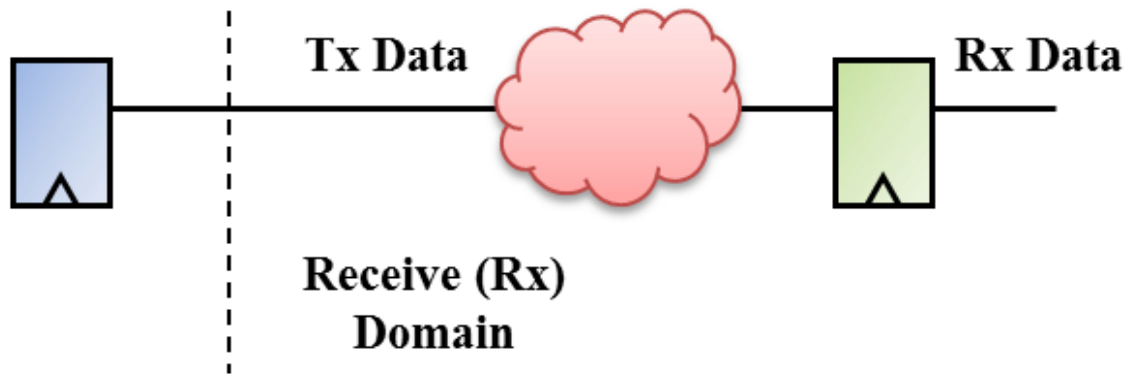
建议直接打开rpt查看。

### 3.2.2.1 常见CDC检查错误

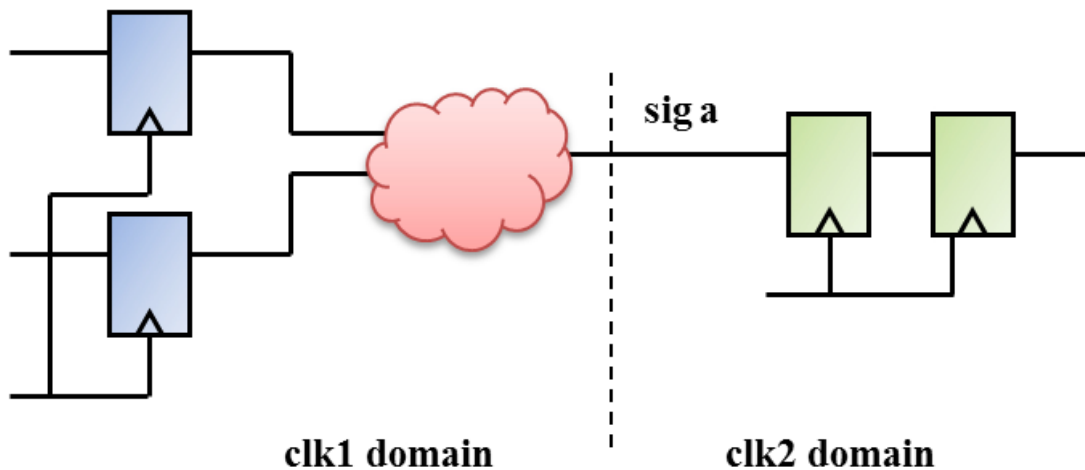
W\_CNTL 控制信号同步，数据未同步



W\_DATA 未做同步处理直接使用



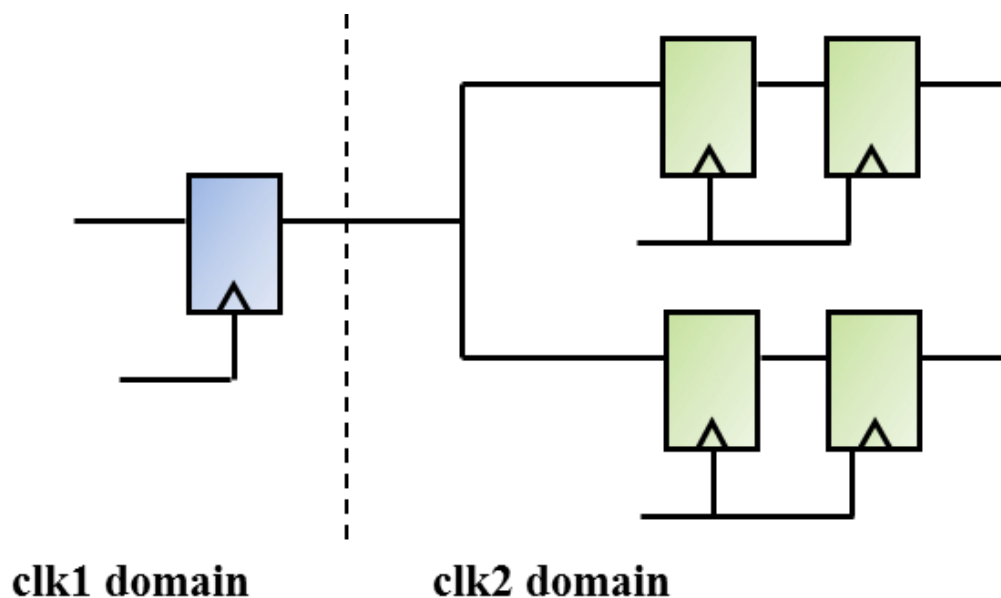
**W\_GLITCH 组合逻辑跨时钟**



组合逻辑输出，由于竞争冒险等原因出现毛刺，毛刺再经过同步跨时钟域处理导致逻辑错误。

解决办法：跨时钟域之间不能出现任何组合逻辑。

**W\_FANOUT 同一个信号在两处做跨时钟域处理**



同一个信号在两处做同步处理，表面上每一路同步路径都是安全的，但是由于需要跨时钟的信号扇出到不同同步器之间的走线延时不一致。同一个信号到达不同同步器第一级寄存器的时刻不一致，便导致不同同步器第一级寄存器是否会进入亚稳态的不确定，以及从亚稳态退出时输出的信号电平值不确定。

解决办法：使用统一同步处理后再使用。

**信号跨时钟后逻辑再汇聚**

和上一个问题原因类似

解决办法：使用统一同步处理后再使用。

### 3.2.2 RDC检查

## 3.3 RTL上传SVN流程

要求清除完所有的编译error、warning和lint的error，才能开始上传文件。

注意只上传RTL代码，其他makefile文件和编译生成的中间文件不能上传，如果 `filelist.mk` 文件有修改也需要上传，

每次上传前先执行update，保证本地版本与库上一致，再开始进行上传

```
svn up
或者
svn update
```

查看文件状态信息

```
svn st
或者
svn status

?: #不在版本库上

A: #预定添加入到版本库上

M: #内容被修改

!: #本地删除，版本库上还有
```

向版本库添加文件

```
# 添加单个文件
svn add filename

# 添加文件夹
svn add path_dir

# 添加当前目录下所有的.v文件
svn add *.v
```

向版本库提交文件

```
svn commit -m "commit"
或者
svn ci -m "commit"
```

上传完成。

新添加的文件，库上没有的需要先执行 `svn add` 添加到版本库，然后再check in提交。

每次修改本地文件后，如果库上已经有历史版本，便不需要 `svn add` 操作，直接执行

```
svn ci -m "update"
```

# 删除版本库上的文件

```
svn delete file
或者
svn del file

svn ci -m "delete file"
```

注意：-m后面的注释里，简单的描述下本次提交是上传或更新了什么内容，不要只笼统的写个commit或update，增加描述，方便回溯历史版本。

## 3.4 Xml寄存器生成流程

xml文件目录在 `$HW/xml`

```
/remote/project/everest/user/ninghechuan/everest_v100r001_ws0/database/soc/hw/xml
ninghechuan@sh-ic02-144-102:xml% IP_SAMPLE_V1.xml
```

将 `IP_SAMPLE_V1.xml` 在当前目录copy出来一份命名为 `YOUR_IP_NAME_V1.xml` 要求命名大写。

```
1: IP_SAMPLE_V1.xml E
1 <?xml version="1.0" ?>
2 <spirit:component xmlns:spirit="http://www.cygnusemi.com">
3   <spirit:name>IP_SAMPLE</spirit:name>
4   <spirit:version>1.0</spirit:version>
5   <spirit:addressBlock>
6     <spirit:name>IP_SAMPLE_V1</spirit:name>
7     <spirit:description>DesignWare Cores DDR Gen2 multiPHY Utility Block</spirit:description>
8     <spirit:baseAddress>0x0000</spirit:baseAddress>
9     <spirit:range>0x1000</spirit:range>
10    <spirit:width>32</spirit:width>
11    <spirit:byteVisit>1</spirit:byteVisit>
12    <spirit:usage>register</spirit:usage>
13    <spirit:protocol>apb</spirit:protocol>
14    <spirit:register>
```

.xml 第13行，protocol中修改为apb会生成apb to regfile，修改为ahb生成ahb to regfile。

按照 .xml 文件的格式填写好寄存器列表

进入到自己的ip目录

```
cd $HW/ips/digital/your_ip/vx/
```

修改当前目录下的 `def.mk`，添加自己填写好的 xml 路径。

```

1: defs.mk
1
2 IP_VERSION = v1
3
4 SYS_VERSION = $(IP_VERSION)
5
6 DUT_TOP_MODULE = $(IP_NAME)_$(IP_VERSION)_top
7
8 MODULE_XML_FILES = $(XML)/NR_PHY_V1.xml
9
10
11
12 AUTOREGTEST =
13
14 RTL_INCDIRS += $(HW)/xml

```

如果子系统或Top集成时有多个xml文件使用两种方式添加。

```

MODULE_XML_FILES += $(XML)/IPNAME1_V1.XML
MODULE_XML_FILES += $(XML)/IPNAME2_V1.XML
MODULE_XML_FILES += $(XML)/IPNAME3_V1.XML

MODULE_XML_FILES = $(XML)/IPNAME1_V1.XML \
$(XML)/IPNAME2_V1.XML \
$(XML)/IPNAME3_V1.XML

```

def.mk 有修改需要checkin到svn环境。

修改完成后进入你的IP RTL目录

```
cd $HW/ips/digital/your_ip/vx/rtl
```

首先保证source了第1节上面的两个cshrc，以及执行过 cy\_prep，在rtl目录下执行

```
make prepregv
```

在 \$HW/xml 目录下就会生成apb/ahb转register的regfile

```

/remoteproject/everest/user/ninghechuan/everest_v100r001_ws0/database/soc/hw/xml
ninghechuan@sh-1c02-144-102:xml% NR_PHY_V1_reg.v

```

生成好的regfile可以直接添加到你的IP的 fileList.f 中。

```

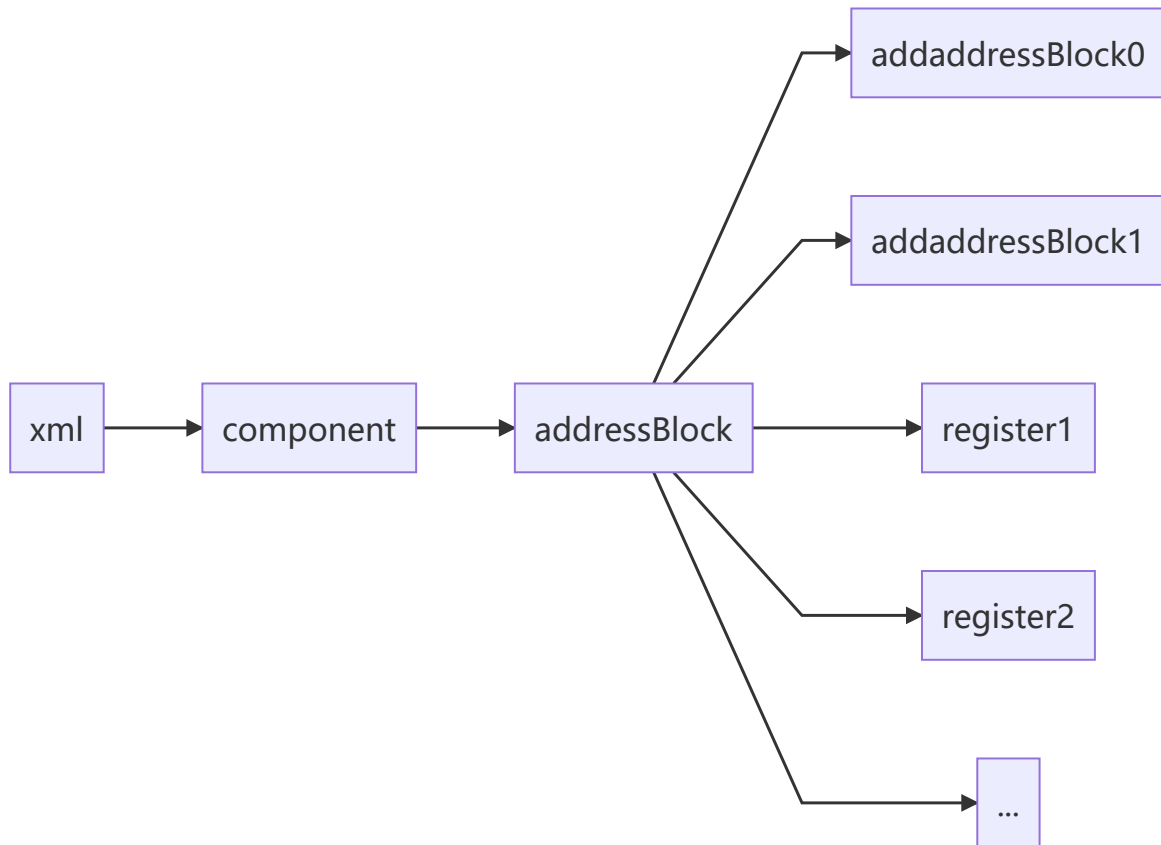
1: fileList.f
1 $HW/ips/digital/main_ctrl/v1/rtl/main_ctrl_v1_top.v
2 $HW/ips/digital/main_ctrl/v1/rtl/main_ctrl.v
3 $HW/ips/digital/main_ctrl/v1/rtl/nr_phy_ahb_dec.v
4
5 $XML/MAIN_CTRL_V1_reg.sv
6 $XML/INT_MANAGMT_V1_reg.sv
7 $XML/PUBLIC_AHB_V1_reg.sv

```

**注意：**生成的regfile不要上传，只本地使用，只把修改过的 \$HW/xml 目录下的.xml文件和修改过的 def.mk 文件上传至svn版本库。

### 3.5 Xml文件填写注释

对于Top和子系统的集成，designer维护addaddressBlock。



#top或子系统集成时

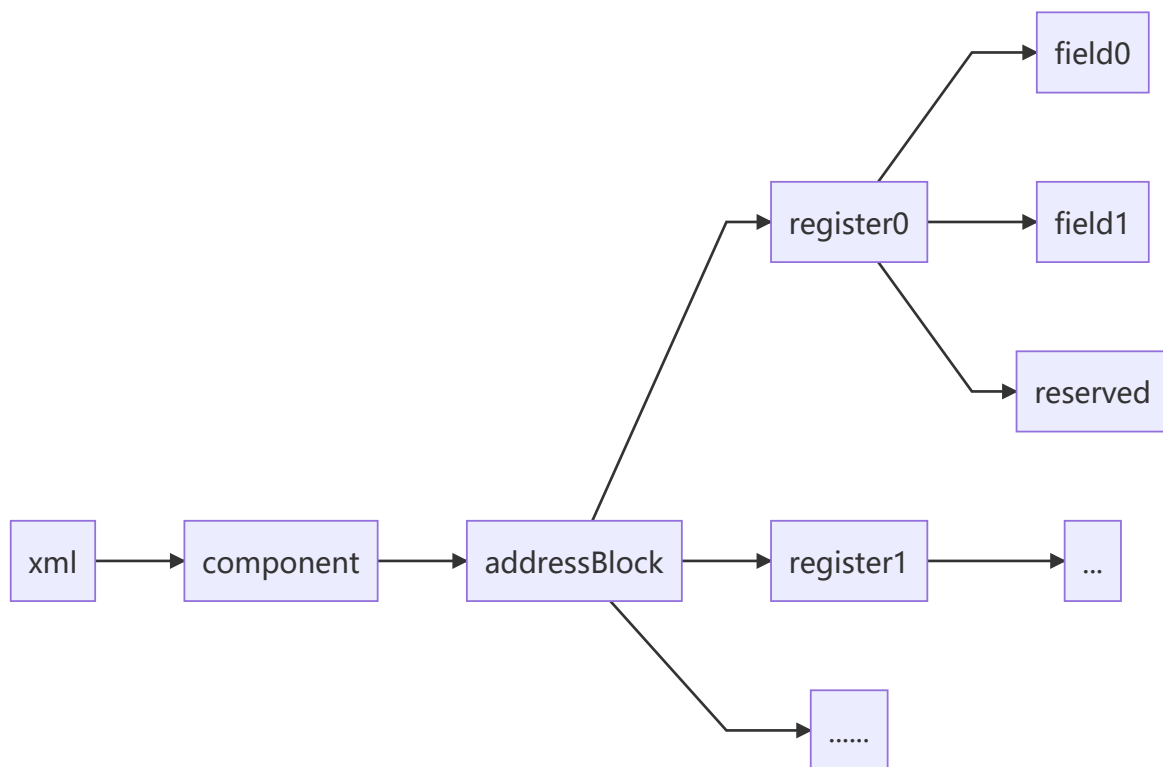
```
<spirit:component xmlns:spirit="http://www.cygnusemi.com">
  <spirit:name>nr_phy</spirit:name>
  <spirit:version>1.0</spirit:version>
  <spirit:addressBlock>
    <spirit:name>NR_PHY_V1</spirit:name>
    <spirit:description>nr phy register file</spirit:description>
    <spirit:baseAddress>0x0</spirit:baseAddress>
    <spirit:range>0x300</spirit:range>
    <spirit:width>32</spirit:width>
    <spirit:byteVisit>1</spirit:byteVisit>
    <spirit:usage>register</spirit:usage>
    <spirit:addaddressBlock>
      <spirit:name>TXF_SUB</spirit:name>
      <spirit:description>sub module registers of isp
sensor3</spirit:description>
      <spirit:baseAddress>0x00</spirit:baseAddress>
      <spirit:range>0x0300</spirit:range>
      <spirit:width>32</spirit:width>
      <spirit:byteVisit>1</spirit:byteVisit>
      <spirit:usage>register</spirit:usage>
    </spirit:addaddressBlock>
  </spirit:addressBlock>
  <spirit:register>
    <spirit:name>ver_date</spirit:name> #修改寄存器名
```

```

    <spirit:description>Revision Identification
Register</spirit:description> #寄存器注释
    <spirit:addressOffset>0x0</spirit:addressOffset> #寄存器偏移地址
    <spirit:size>32</spirit:size>
    <spirit:access>RO</spirit:access> #寄存器权限
    <spirit:reset>
        <spirit:value>0x20210604</spirit:value> #寄存器初始值
    </spirit:reset>
    <spirit:field>
        <spirit:name>PUBMNR</spirit:name> #寄存器分段名，可以占满32bit，也可以多
个寄存器分段沾满32bit
        <spirit:description>asic version</spirit:description> #寄存器段描述
        <spirit:bitOffset>0</spirit:bitOffset> #寄存器段偏移地址
        <spirit:bitwidth>32</spirit:bitwidth>
        <spirit:access>RO</spirit:access> #寄存器段权限
    </spirit:field>
</spirit:register>
</spirit:addressBlock>
</spirit:component>

```

对于单个IP模块设计，模块的owner维护自己的xml生成寄存器文件。有自己的寄存器块。



xml文件的结构如图所示包含一个component组件，组件包含一个地址块addressBlock，每个地址块包含多个寄存器，位宽为32bit，每个寄存器包含多个寄存器段field，每个寄存器段的位宽可以自定义，所有的寄存器段合并起来为一个32bit的寄存器。

有多少个寄存器，就将寄存器块赋值多少份进行填写。



```

<?xml version="1.0" ?>
<spirit:component xmlns:spirit="http://www.cygnusemi.com">
  <spirit:name>nr_phy</spirit:name>    #修改成自己的IP名，必须为小写
  <spirit:version>1.0</spirit:version> # xml的版本号
  <spirit:addressBlock>
    <spirit:name>NR_PHY_V1</spirit:name>    #修改成自己的IP名+版本号，要求大写
    <spirit:description>nr phy register file</spirit:description>    #寄存器文
件的注释
    <spirit:baseAddress>0x0</spirit:baseAddress>    #基地址，每个IP默认从0开始
    <spirit:range>0xb</spirit:range>    #地址范围，表示所有的寄存器占用的地址数
量
    <spirit:width>32</spirit:width>    #寄存器位宽32bit
    <spirit:byteVisit>1</spirit:byteVisit>
    <spirit:usage>register</spirit:usage>
    <spirit:protocol>ahb</spirit:protocol>    #支持ahb和apb协议，修改后生成对应协议
的regfile
    #第一个寄存器，如果设置为RO，会默认为版本寄存器，没有input，如果没有版本寄存器，建议第
一个寄存器别设置成RO
    <spirit:register>
      <spirit:name>ver_date</spirit:name>    #修改寄存器名
      <spirit:description>Revision Identification
Register</spirit:description>    #寄存器注释
      <spirit:addressOffset>0x0</spirit:addressOffset>    #寄存器偏移地址
      <spirit:size>32</spirit:size>
      <spirit:access>RO</spirit:access>    #寄存器权限
      <spirit:reset>
        <spirit:value>0x20210604</spirit:value>    #寄存器初始值
      </spirit:reset>
      <spirit:field>
        <spirit:name>PUBMNR</spirit:name>    #寄存器分段名，可以占满32bit，也可以多
个寄存器分段沾满32bit
        <spirit:description>asic version</spirit:description>    #寄存器段描述
        <spirit:bitOffset>0</spirit:bitOffset>    #寄存器段偏移地址
        <spirit:bitWidth>32</spirit:bitWidth>
        <spirit:access>RO</spirit:access>    #寄存器段权限
      </spirit:field>
    </spirit:register>
    #第二个寄存器
    <spirit:register>
      <spirit:name>test</spirit:name>
      <spirit:description>test register example</spirit:description>
      <spirit:addressOffset>0x4</spirit:addressOffset>    #寄存器的偏移地址，接着上
一个寄存器的偏移地址
      <spirit:size>32</spirit:size>
      <spirit:access>RW</spirit:access>
      <spirit:reset>
        <spirit:value>0x0</spirit:value>    #寄存器初始值为寄存器段的初始值组合后的值
      </spirit:reset>
      <spirit:field>
        <spirit:name>write</spirit:name>
        <spirit:description>test write register</spirit:description>
        <spirit:bitOffset>0</spirit:bitOffset>
        <spirit:bitWidth>16</spirit:bitWidth>
        <spirit:access>RW</spirit:access>
      </spirit:field>
    </spirit:register>
    #一个寄存器中可以分多个寄存器段
    <spirit:field>
      <spirit:name>read</spirit:name>

```

```

        <spirit:description>test read register</spirit:description>
        <spirit:bitOffset>16</spirit:bitOffset>
        <spirit:bitwidth>16</spirit:bitwidth>
        <spirit:access>RO</spirit:access>
    </spirit:field>
</spirit:register>

```

#第三个寄存器

```

<spirit:register>
    <spirit:name>enable</spirit:name>
    <spirit:description>test enable example</spirit:description>
    <spirit:addressOffset>0x8</spirit:addressOffset>
    <spirit:size>32</spirit:size>
    <spirit:access>RW</spirit:access>
    #寄存器段的权限始终是寄存器的子集，例如，寄存器段的权限为RW，寄存器的权限必须为RW
    <spirit:reset>
        <spirit:value>0x0</spirit:value>
    </spirit:reset>
    <spirit:field>
        <spirit:name>test1</spirit:name>
        <spirit:description>test1 enable register</spirit:description>
        <spirit:bitOffset>0</spirit:bitOffset>
        <spirit:bitwidth>1</spirit:bitwidth>
        <spirit:access>RC</spirit:access>
    </spirit:field>
    <spirit:field>
        <spirit:name>test2</spirit:name>
        <spirit:description>test2 enable register</spirit:description>
        <spirit:bitOffset>1</spirit:bitOffset>
        <spirit:bitwidth>1</spirit:bitwidth>
        <spirit:access>WC</spirit:access>
    </spirit:field>

```

#一个寄存器没有占满，需要设置保留bit，整个寄存器文件中的保留bit的命名需要依次为reserved\_0—reserved\_n排序，reserved寄存器权限为RO

```

    <spirit:field>
        <spirit:name>reserved_0</spirit:name>
        <spirit:description>reserved register</spirit:description>
        <spirit:bitOffset>2</spirit:bitOffset>
        <spirit:bitwidth>30</spirit:bitwidth>
        <spirit:access>RO</spirit:access>
    </spirit:field>
</spirit:register>

```

#中断控制器 第一组中断 建议将中断控制器块放在xml的最后

```

<spirit:interrupt>
    <spirit:name>INT_DWC0</spirit:name>#中断信号名
    <spirit:description>interrupt for DWC</spirit:description>
    <spirit:addressOffset>0xC</spirit:addressOffset>
    <spirit:size>32</spirit:size>
    <spirit:access>RW</spirit:access>#中断信号权限
    <spirit:reset>
        <spirit:value>0x0</spirit:value>
    </spirit:reset>
    <spirit:field>
        <spirit:name>DWC1_DONE</spirit:name>
        <spirit:description>DWC1 completed.</spirit:description>
        <spirit:bitOffset>0</spirit:bitOffset>
        <spirit:bitwidth>1</spirit:bitwidth>
        <spirit:access>RW</spirit:access>
    </spirit:field>

```

```

    <spirit:field>
      <spirit:name>DWC2_DONE</spirit:name>
      <spirit:description>DWC2 completed.</spirit:description>
      <spirit:bitOffset>4</spirit:bitOffset>
      <spirit:bitWidth>1</spirit:bitWidth>
      <spirit:access>RW</spirit:access>
    </spirit:field>
  </spirit:interrupt>
  #第二组中断
  <spirit:interrupt>
    <spirit:name>INT_DWC1</spirit:name>#中断信号名
    <spirit:description>interrupt for DWC</spirit:description>
    <spirit:addressOffset>0x28</spirit:addressOffset># 每新加一组中断需要预留出
7个中断的状态寄存器
    <spirit:size>32</spirit:size>
    <spirit:access>RW</spirit:access>#中断信号权限
    <spirit:reset>
      <spirit:value>0x0</spirit:value>
    </spirit:reset>
    <spirit:field>
      <spirit:name>DWC1_DONE</spirit:name>
      <spirit:description>DWC1 completed.</spirit:description>
      <spirit:bitOffset>0</spirit:bitOffset>
      <spirit:bitWidth>1</spirit:bitWidth>
      <spirit:access>RW</spirit:access>
    </spirit:field>
    <spirit:field>
      <spirit:name>DWC2_DONE</spirit:name>
      <spirit:description>DWC2 completed.</spirit:description>
      <spirit:bitOffset>4</spirit:bitOffset>
      <spirit:bitWidth>1</spirit:bitWidth>
      <spirit:access>RW</spirit:access>
    </spirit:field>
  </spirit:interrupt>
</spirit:addressBlock>
</spirit:component>

```

目前xml支持寄存器的权限有

"RO"	W:no effect, R:as-is
"WO"	W:as-is, R:no effect
"RW"	W:as-is, R:as-is
"WC"	W:clears all bits, R:no effect
"RC"	W:no effect, R:clears all bits

WC为写寄存器后，一个cycle后自清为reset值。RC在读一次寄存器后，一个cycle后自清为reset值。

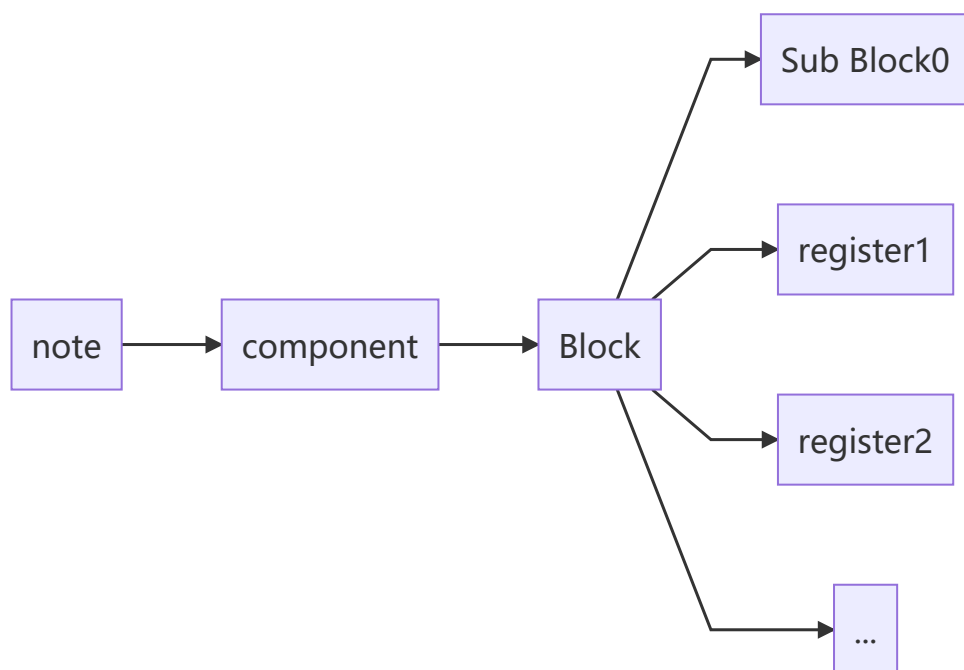
寄存器的权限只能是RW、WO、RO三种，寄存器段（field）的权限可以为RW、RO、WO、WC、RC。而且寄存器的权限必须包含寄存器段的权限，比如，如果register field的权限为RW，那么register的权限就只能为RW。如果register field的权限为WC，那么register的权限可以是RW，也可以是WO。

中断控制器对Designer侧接口建议输入脉冲中断，对软件侧支持配置电平触发或脉冲触发中断。

中断控制寄存器	偏移地址	说明
RAW	interrupt偏移地址	IP模块中断信号原始值
STAT	interrupt偏移地址+0x4	中断状态寄存器
MASK	interrupt偏移地址+0x8	掩码寄存器，高电平掩码
SET	interrupt偏移地址+0xc	设置特定中断触发寄存器
CLR	interrupt偏移地址+0x10	中断清除寄存器，仅支持写清
MODE	interrupt偏移地址+0x14	中断类型mode=1：电平触发，mode=0：脉冲触发
POLAR	interrupt偏移地址+0x18	设置中断寄存器极性polar=1：上升沿触发，polar=0：下降沿触发

### 3.6 note生成寄存器流程

复制路径 `/remote/public/ninghechuan/note_xml_reg/isp_mnt.note` 到自己的IP目录下



```

#修改ISP_SUB为自己的IP名  v1.0是note的版本号，无需修改
// Component
ISP_SUB      -- v1.0

#修改成自己的IP名+版本号，修改自己模块的地址范围，第三列--后为注释
// Block

```

```

ISP_SUB_V1  -- 0x8000:0x8fff      -- ISP sub Block monitor      -- ahb

#调用其他模块的寄存器块时需要填写，没有时删除
//Sub Block
ISP_MNT_TST -- 0x8c00:0x8fff      -- ISP monitor sub Block Test
//Sub block end

#从左到右 --区分列
#列1: 寄存器偏移地址
#列2: 寄存器初始值，寄存器不需要填写，内部的填写初始值，工具会自动拼接
#列3: 寄存器权限，寄存器权限支持RW、RO、WO，寄存器段的权限支持RW、RO、WO、WC、RC
#列4: 寄存器名，寄存器段名
#列5: 寄存器位宽，寄存器段范围
#列6: 注释
//register
0x0          -- RW  -- MNT0      -- [31:0] -- Here is the comment
-- 'h0       -- RW  -- ENABLE   -- [0]   -- Here is the comment
-- 'h0       -- RW  -- CLR       -- [1]   -- Here is the comment

0x4          -- RW  -- MNT1      -- [31:0] -- Here is the comment
-- 'h0       -- RW  -- FC        -- [0]   -- Here is the comment

0x8          -- RW  -- MNT2      -- [31:0] -- Here is the comment
-- 'h0       -- RW  -- FLAG      -- [0]   -- Here is the comment
//register end

//Interrupt
0xc          -- RW  -- INPT0    -- [31:0] -- Here is the comment
-- 'h0       -- RW  -- DONE0    -- [0]   -- Here is the comment
-- 'h0       -- RW  -- DONE1    -- [1]   -- Here is the comment
#第二个中断组与第一个中断中间需要空7个寄存器地址给中断寄存器状态
0x18        -- RW  -- INPT1    -- [31:0] -- Here is the comment
-- 'h0       -- RW  -- DONE     -- [0]   -- Here is the comment
//Interrupt end

//Block end
//Component end

```

## 格式要求

1. 不能使用tab键，对齐只能使用空格
2. Register/Interrupt的注释不要含有“-"

note格式的文件对空格的个数没有要求，为了易读使用空格对齐。

目前xml支持寄存器的权限有

"RO"	W:no effect, R:as-is
"WO"	W:as-is, R:no effect
"RW"	W:as-is, R:as-is
"WC"	W:clears all bits, R:no effect
"RC"	W:no effect, R:clears all bits

寄存器的权限只能是RW、WO、RO三种，寄存器段（field）的权限可以为RW、RO、WO、WC、RC。而且寄存器的权限必须包含寄存器段的权限，比如，如果register field的权限为RW，那么register的权限就只能为RW。如果register field的权限为WC，那么register的权限可以是RW，也可以是WO。

中断控制器对Designer侧接口建议输入脉冲中断，对软件侧支持配置电平触发或脉冲触发中断。

中断控制寄存器	偏移地址	说明
RAW	interrupt偏移地址	IP模块中断信号原始值
STAT	interrupt偏移地址+0x4	中断状态寄存器
MASK	interrupt偏移地址+0x8	掩码寄存器，高电平掩码
SET	interrupt偏移地址+0xc	设置特定中断触发寄存器
CLR	interrupt偏移地址+0x10	中断清除寄存器，仅支持写清
MODE	interrupt偏移地址+0x14	中断类型mode=1：电平触发，mode=0：脉冲触发
POLAR	interrupt偏移地址+0x18	设置中断寄存器极性polar=1：上升沿触发，polar=0：下降沿触发

填写好note文件，首先保证source了上面的两个cshrc，然后进入你的IP目录。执行

```
gen_xml.pl xxx.note
```

然后把本地生成的xxx.xml文件复制到xml文件目录在 `$HW/xml` 下

进入到自己的ip目录

```
cd $HW/ips/digital/your_ip/vx/
```

修改当前目录下的 `def.mk`，添加自己填写好的 `xml` 路径。

```
1: defs.mk
1
2 IP_VERSION = v1
3
4 SYS_VERSION = $(IP_VERSION)
5
6 DUT_TOP_MODULE = $(IP_NAME)_$(IP_VERSION)_top
7
8 MODULE_XML_FILES = $(XML)/NR_PHY_V1.xml
9
10
11
12 AUTOREGTEST =
13
14 RTL_INCDIRS += $(HW)/xml
```

如果子系统或Top集成时有多个xml文件使用两种方式添加。

```
MODULE_XML_FILES += $(XML)/IPNAME1_V1.XML
MODULE_XML_FILES += $(XML)/IPNAME2_V1.XML
MODULE_XML_FILES += $(XML)/IPNAME3_V1.XML

MODULE_XML_FILES = $(XML)/IPNAME1_V1.XML \
                  $(XML)/IPNAME2_V1.XML \
                  $(XML)/IPNAME3_V1.XML
```

修改完成后进入你的IP RTL目录

```
cd $HW/ips/digital/your_ip/vx/rtl
```

首先保证source了第1节上面的两个cshrc，以及执行过 `cy_prep`，在rtl目录下执行

```
make prepregv
```

在 `$HW/xml` 目录下就会生成apb/ahb转register的regfile

```
/remote/project/everest/user/ninghechuan/everest_v100r001_ws0/database/soc/hw/xml
ninghechuan@sh-ic02-144-102:xml% NR_PHY_V1_reg.v
```

生成好的regfile可以直接添加到你的IP的 `filelist.f` 中。

```
1: filelist.f
1 $HW/ips/digital/main_ctrl/v1/rtl/main_ctrl_v1_top.v
2 $HW/ips/digital/main_ctrl/v1/rtl/main_ctrl.v
3 $HW/ips/digital/main_ctrl/v1/rtl/nr_phy_ahb_dec.v
4
5 $XML/MAIN_CTRL_V1_reg.sv
6 $XML/INT_MANAGMT_V1_reg.sv
7 $XML/PUBLIC_AHB_V1_reg.sv
```

**注意：**生成的regfile不要上传，只本地使用，只把修改过的.xml文件在 `$HW/xml` 目录上传至svn版本库。

## 4. CRG IP使用流程

NR的crg IP已经上传至svn ip\_common路径。

i\_scan\_en 所有用了icg模块的IP，都需要留出端口，所有模块共用一个i\_scan\_en，nr\_phy会从顶层引出至soc。

端口说明

Port name	I/O	描述
i_clk	I	系统时钟输入
i_rst_n	I	系统复位输入
i_scan_en	I	测试使能，高电平软件clock gating 使能失效
i_soft_clk_en	I	软件clock gating 使能，高电平有效
soft_rstn_in	I	软件复位使能
soft_rstn_out	O	复位信号输出
o_gclk	O	Clock gating输出

## 5. Memory规格生成及使用方法

memory规格申请，填写memory规格申请表格。

<http://pc.cygnus.com/wiki/spaces/60b6db7bb8a85fa307692bd6/pages/60dec62ee401642cb63ea9d5>

IP	your_ip_name
Number	申请个数
TYPE	memory类型, spram、dpram、rom
Number of Words	深度
Bits in Word	宽度
Mux	default空，中端填写
Bank	default空，中端填写
Bit-Write	bit使能，TURE/FALSE
Center Decode	default TURE
Power Gating	rentention memory填写TURE, default FALSE
Dual Power	rentention memory填写TURE, default FALSE
Light Sleep	rentention memory填写TURE, default FALSE
Periphery Vt	default LOW
info	备注信息，rom需要备注rom的txt文件。

memory的规格如下。（注意：建议优先使用spram，dpram资源大于spram）

dpram: dual port memory, 伪双口memory。A口写，B口读，memory输入信号必须为寄存器，输出必须锁存一拍使用。



<b>dpram</b>		
output	QB	B口输出数据
input	ADRA	A口写地址
input	DA	A口写输入
input	WEA	A口写使能, 1有效
input	MEA	A口片选, 1有效
input	CLKA	A口时钟
input	ADRB	B口写地址
input	MEB	B口片选, 1有效
input	CLKB	B口时钟

spram: single port memory, 单口memory。WE和ME同时为1写, WE高, ME低为读。。memory输入信号必须为寄存器, 输出必须锁存一拍使用。

<b>spram</b>		
output	Q	输出数据
input	ADR	写地址
input	D	写输入
input	WE	写使能, 1有效
input	ME	片选, 1有效
input	CLK	时钟

rom: 输出必须锁存一拍使用。

<b>rom</b>		
output	Q	输出数据
input	ADR	写地址
input	D	写输入
input	ME	片选, 1有效
input	CLK	时钟

`$HW/lib/mem/nr_phy/v1/mem_wrapper`: 需要拿去在自己的rtl中替换的wrapper  
`/remote/project/everest/smic14_lib/asic_mem_lib/nr_phy/v1/mem_db`: 综合使用真实mem db  
`/remote/project/everest/smic14_lib/asic_mem_lib/nr_phy/v1/mem_model`: 生成的仿真模型  
`/remote/project/everest/smic14_lib/asic_mem_lib/nr_phy/v1/mem_syn`: 带延时的仿真模型  
`/remote/project/everest/smic14_lib/asic_mem_lib/nr_phy/v1/srom_hex_file`: rom的hex 文件

每个IP使用在 `filelist.mk` 中添加如下。

```
51 RTL_INCDIRS += $(HW)/ips/digital/ip_common/v1/rtl
52 RTL_INCDIRS +=
53 RTL_INCIPS_DIR += $(HW)/lib/mem/v1/nr_phy
54 RTL_INCIPS_DIR += /remote/project/everest/smic14_lib/asic_mem_lib/nr_phy/v1/mem_model
55 RTL_INCIPS_DIR += /remote/project/everest/smic14_lib/asic_mem_lib/nr_phy/v1/mem_syn
```

然后取 `$HW/lib/mem/v1/nr_phy` 中寻找自己的mem\_wrapper进行调用替换。

mem\_wrapper中提供了三个define分支

```
`ifdef FPGA

`elsif ASIC_MEM_LIB

`else
```

FPGA分支为原型验证使用FPGA MEM IP的替换。ASIC\_MEM\_LIB分支为综合和后仿 MEM IP分支。esle分支为默认的ASIC仿真模型分支。