目录

[环境配置 3](#_Toc117234377)

[Groups 3](#_Toc117234378)

[获取VIP 3](#_Toc117234379)

[查看当前testbench 版本号 3](#_Toc117234380)

[TB\_1.6\_HOTFIX 3](#_Toc117234381)

[test command: 4](#_Toc117234382)

[几个重要的文件夹 4](#_Toc117234383)

[Spec 5](#_Toc117234384)

[Doc 5](#_Toc117234385)

[GCC 6](#_Toc117234386)

[编译过程 6](#_Toc117234387)

[参数详解 8](#_Toc117234388)

[常用参数 8](#_Toc117234389)

[预编译 -E 9](#_Toc117234390)

[编译-S 11](#_Toc117234391)

[汇编 -C 11](#_Toc117234392)

[链接 12](#_Toc117234393)

[arm-none-eabi交叉编译工具 13](#_Toc117234394)

[help 13](#_Toc117234395)

[arm-none-eabi-gcc 13](#_Toc117234396)

[arm-none-eabi-ld 13](#_Toc117234397)

[arm-none-eabi-objdump 14](#_Toc117234398)

[arm-none-eabi-objcopy 14](#_Toc117234399)

[arm-none-eabi-size 14](#_Toc117234400)

[RT700用到的gcc command 14](#_Toc117234401)

[查找log 关键词 14](#_Toc117234402)

[RT700 15](#_Toc117234403)

[Master ID 15](#_Toc117234404)

[code 16](#_Toc117234405)

[Boot 16](#_Toc117234406)

[Linkfile 解析 16](#_Toc117234407)

[C编译运行 17](#_Toc117234408)

[Pre\_main 、 post\_main 18](#_Toc117234409)

[CM33 NS boot 19](#_Toc117234410)

[CM33\_M boot 34](#_Toc117234411)

[CM33\_M\_NS boot 35](#_Toc117234412)

[HIFI4 boot 35](#_Toc117234413)

[HIFI1 Boot 36](#_Toc117234414)

[Zenv Boot 37](#_Toc117234415)

[Dual core boot 38](#_Toc117234416)

[可用的non-secure 地址 38](#_Toc117234417)

[指定code到section中 38](#_Toc117234418)

[DMA setup 38](#_Toc117234419)

[原理 40](#_Toc117234420)

[Testbench 40](#_Toc117234421)

[Spec文档 40](#_Toc117234422)

[Force 用法 40](#_Toc117234423)

[Arg用法 41](#_Toc117234424)

[Default tc 41](#_Toc117234425)

[Makefile defines 42](#_Toc117234426)

[Signals 42](#_Toc117234427)

# 环境配置

## Groups

* necessary unix groups:

mcudes

neutron

nstcc\_ec

risc5

* for EZHV need

riscv

## 获取VIP

* VIP 版本查询：

输入v\_ms\_spp\_dma3\_nxt\_vip 查询 [PDM](https://designpdm.nxp.com/3dspace/common/dpdm_WWTE_emxNavigator.jsp?serverId=SPACE.mcs01&ticket=ST-158635-tuicLoW3E5brCD7KwNWx-cas&collabSpace=Common%20Space)

* 创建名为ref\_dma的文件夹，获取VIP

pi ws ref\_dma -bom V\_MS\_SPP\_DMA3\_NXT\_VIP\_1.41

## 查看当前testbench 版本号

Testbench跟目录下查看如下文件，可以看到各个VIP，IP使用的版本

g .pi\_md/pi.bom

partid:V\_SS\_RT700\_SOC\_TB\_1.9 // verification的版本号

# Part SOC\_imxrt700\_CMOS28FDSOI\_1.9 //design的版本号

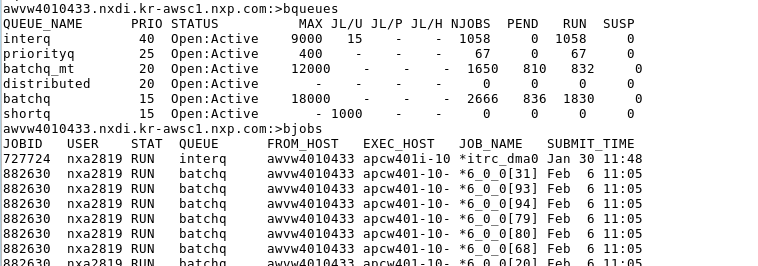
## Samba

[\\smb.nxdi.kr-awsc1.nxp.com\nxa28190](file:///\\smb.nxdi.kr-awsc1.nxp.com\nxa28190)

## Bsub

Bqueues， 查看服务器的job情况。Regression是batchq

Bjobs， 查看个人job情况



## TB\_1.6\_HOTFIX

可用的空间：

/home/imxrt700\_verif\_nobackup1

/home/imxrt700\_verif\_nobackup2

/home/imxrt700\_verif1

To build TB database for CN-SHA01 user:

cd /home/imxrt700\_verif1/<your\_workspace>

alias wa7 'source /home/imxrt700\_verif1/setup/w.csh'

wa7 V\_SS\_RT700\_SOC\_TB\_1.6 -bom V\_SS\_RT700\_SOC\_TB\_1.6

source /home/imxrt700\_verif1/Berk\_nxa12151/bin/create\_V\_SS\_RT700\_SOC\_TB\_1\_6.csh

Based on testbench V\_SS\_RT700\_SOC\_TB\_1.6, Please get updating by

source /home/imxrt700\_verif1/Berk\_nxa12151/bin/create\_V\_SS\_RT700\_SOC\_TB\_1\_6\_hotfix.csh

### test command:

* 当前项目根目录下下执行：

bsub -R rhel7 -q interq -Ip soc verilog -irun -block soc\_tb -bc rtl -collect\_arg -tc default -vectors sanity -test simple -sim\_arg "-gui" -shm -keeptemps -session default1 &

vgen -plan DMA.csv

bsub -R rhel7 -q interq -Ip soc verilog -irun -block soc\_tb -bc rtl -collect\_arg -tc default -vectors dma -test dma3\_reg\_\_inst0 -sim\_arg "-gui" -shm -keeptemps -session default2 &

gls：

bsub -q interq -I soc verilog -irun -block soc\_tb -bc postlayout -collect\_arg -tc default -vectors dma -test\_name dma3\_slave\_access\_sram\_p4\_p7\_\_inst0 -64bit -proj\_lib -keeptemps -shm -cond worst  -session worst&

bsub -q interq -I soc verilog -irun -block soc\_tb -bc postlayout -collect\_arg -tc default -vectors dma -test\_name dma3\_soft\_trigger\_\_zenv\_\_inst2 -sim\_arg "-gui" -64bit -proj\_lib -keeptemps -shm -cond best -session dma\_best &

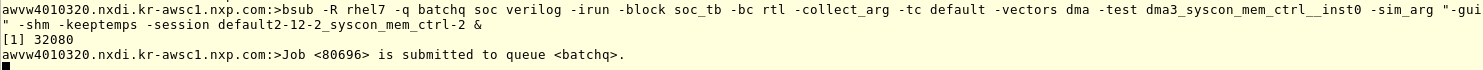
gls postlayout6:

bsub -q interq -I soc verilog -irun -block soc\_tb -bc postlayout6 -collect\_arg -tc default -vectors dma -test\_name dualcore\_dma2\_deepsleep\_wakeup\_\_cm33\_\_rstctl\_comp\_media\_comm\_srpg\_wakeup\_by\_rtc\_twice\_\_cm33\_m\_\_dma3\_lp\_cm33\_m -64bit  -proj\_lib -keeptemps -shm -cond best  -session best6&

* FSDB

bsub -q interq -I soc verilog -irun -block soc\_tb -bc postlayout -collect\_arg -tc default -vectors dma  -test\_name dma3\_soft\_trigger\_\_inst0 -64bit -proj\_lib -keeptemps -shm -cond worst  -fsdb -fsdb\_file "dma0\_soft\_trigger.fsdb" -session dma0\_soft\_trigger&

log不显示在console， batchq：



* 仿真波形图跑

bsub -R rhel7 -q interq -Ip soc verilog -irun -block soc\_tb -bc rtl -collect\_arg -tc default -vectors dma -test dma3\_reg\_\_cm33\_m\_\_inst2 -sim\_arg "-gui" -shm -keeptemps -session default\_0630\_1 &

* memory dump

bsub -R rhel7 -q interq -Ip soc verilog -irun -block soc\_tb -bc rtl -collect\_arg -tc default -vectors dma -test dma3\_interrupt\_\_inst0 -sim\_arg "-gui +MEMDUMP +DELTACYCLE " -shm -keeptemps -comp\_arg " +access+rwc -linedebug " -session default720-1

* 不编译.V

加上 -no\_compile

* 允许跑出所有错误

最好在command line 上跑， 加-sim\_arg "+MAX\_ERROR=200 +UVM\_ERROR=200"，这样把所有的error 都跑出来再 一次性加属性，速度快些

#### other project

K4W1：

Download：

SHA datacenter:

alias w 'source /home/k4w1\_bk/setup/set\_workarea'

PHX datacenter:

alias w 'source /proj/verif\_k4\_1/sivan\_nxa21236/setup/set\_workarea'

w SOC\_K4W1\_CLN40ULP\_1.13\_TB1\_TO -bom V\_SS\_K4W1\_SOC\_TB\_1.13 -pop chip -pop soc\_tb -pop k4\_common\_tb

环境配置：

/home/k4w1\_bk/richard\_nxa28190/SOC\_K4W1\_CLN40ULP\_1.13\_TB1\_TO

source /home/k4w1\_bk/setup/set\_workarea

test command：

bsub -q interq -I soc verilog -irun -tc default -bc rtl\_ams\_pads -vec dma3 -collect\_arg -test dma3\_stop\_ack\_\_dsleep\_\_inst0\_\_pclk\_cfg\_on\_run\_mode -sim\_arg '-gui' -shm -session cmp\_window

### SE RTL test command:

bsub -q batchq -n 3 soc verilog -irun -block soc\_tb -bc rtl -collect\_arg -tc default\_rom -vectors sys\_rom -64bit -proj\_lib -comp\_arg "-access +rwc" -sim\_arg " +RT700\_MEM\_LOADER\_HEX\_EN" -keeptemps -test na\_boot\_xspi0\_mx25um513\_octal\_xip\_50mhz\_redundancy -shm -session rom\_session\_dma3 &

### coverage

exclude 信号：/home/imxrt700\_verif\_nobackup2/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.31/testbench/blocks/soc\_tb/tool\_data/iccr/soc/exclude

soc\_tb/tool\_data/iccr/soc 145 $ ./vrefine2txt exclude/utick0.vRefine

ls exclude/

utick0.vRefine utick0\_toggle\_excludefile.txt

dssc ci utick0\* -new

#### regression test command

测试：rtl

vrun -cov &

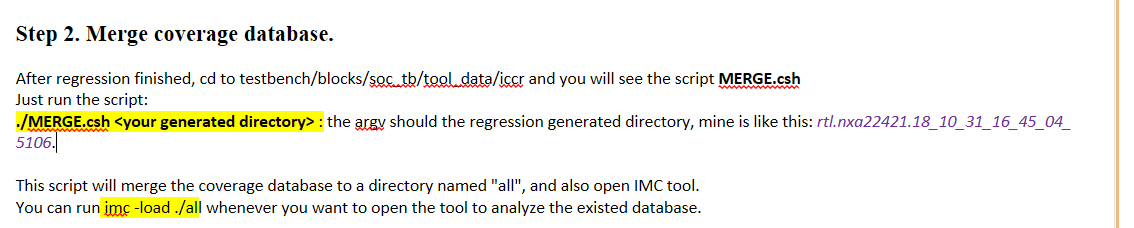
gls：

vgen -plan SANITY.csv -priority=0

vrun -view postlayout,postlayout6 -cond best -target gls

vrun -view postlayout,postlayout7,postlayout8,postlayout9 -cond worst -target gls

vrun -view postlayout\_clp -cond worst -target gls



[Backup--Coverage Analyzation](onenote:https://nxp1-my.sharepoint.com/personal/angela_liang_nxp_com/Documents/MCU_SZ_verification/Training%20materials.one#Backup--Coverage%20Analyzation&section-id={9730C98F-0ABD-4FE9-AFD2-EE7257D284FA}&page-id={BE7F5D54-2CA6-4D4F-BAFC-00DB3AF8E840}&end)  ([Web view](https://nxp1-my.sharepoint.com/personal/angela_liang_nxp_com/_layouts/OneNote.aspx?id=%2Fpersonal%2Fangela_liang_nxp_com%2FDocuments%2FMCU_SZ_verification&wd=target%28Training%20materials.one%7C9730C98F-0ABD-4FE9-AFD2-EE7257D284FA%2FBackup--Coverage%20Analyzation%7CBE7F5D54-2CA6-4D4F-BAFC-00DB3AF8E840%2F%29))

#### Run single test case

给coverage数据进行修改：

一个的话 可以命令行run 比如

bsub -R rhel7 -q interq -Ip soc verilog -irun -block soc\_tb -bc rtl -collect\_arg -tc default -vectors dma -test dma3\_soft\_trigger\_\_cm33\_m\_\_inst2 -shm -keeptemps -session default2-cov -icc\_cfg /home/imxrt700\_verif\_nobackup2/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.34/testbench/blocks/soc\_tb/tool\_data/iccr/soc/cov.cfg -icc\_overwrite -icc\_dut testbench &

打开coverage：

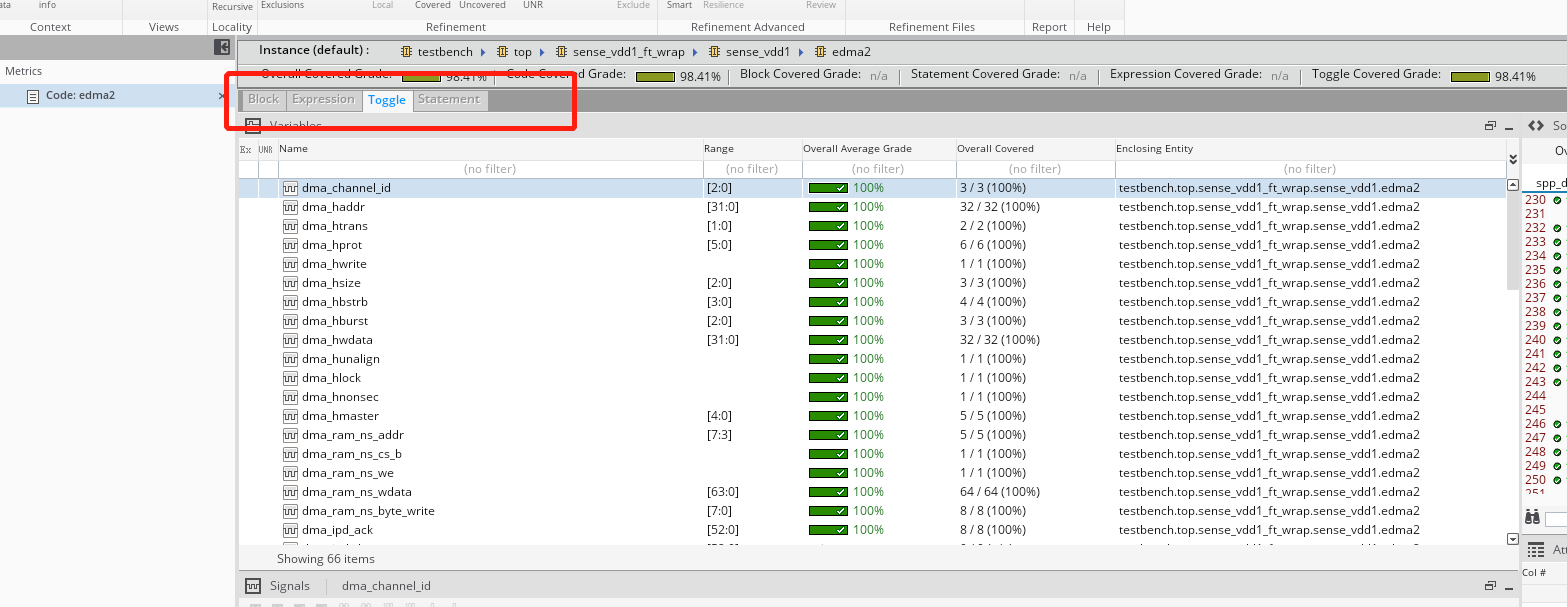


#### 加载coverage

加载别人的coverage 文件：

imc -load /home/imxrt700\_verif\_nobackup1/nxa12151/V\_SS\_RT700\_SOC\_TB\_1.26/testbench/blocks/soc\_tb/tool\_data/iccr/rtl.nxa12151.22\_11\_28\_20\_56\_39\_2754/all

#### coverage文件配置



V\_SS\_RT700\_SOC\_TB\_1.34/testbench/blocks/soc\_tb/tool\_data/iccr/soc/cov.cfg

这个文件定义了coverage的范围，需要exclude信号对应的文件

select\_coverage -all -instance testbench.top.compute\_vdd2\_ft\_wrap.compute\_vdd2.compute\_vdd2\_soc\_glue

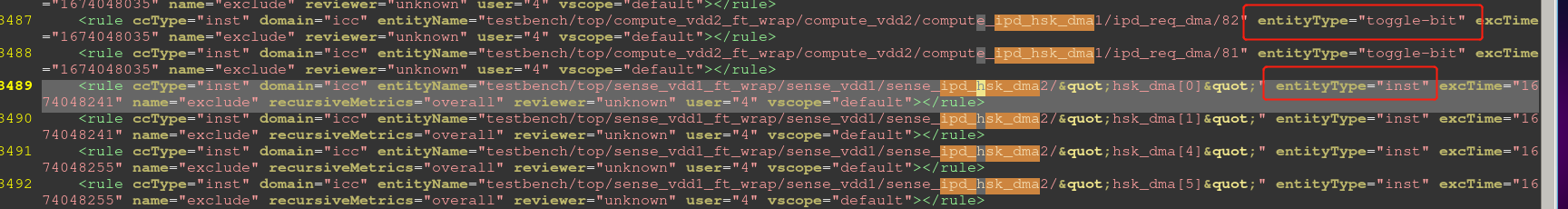
-all表示coverage所有的信号

-t 表示cover toggle信号

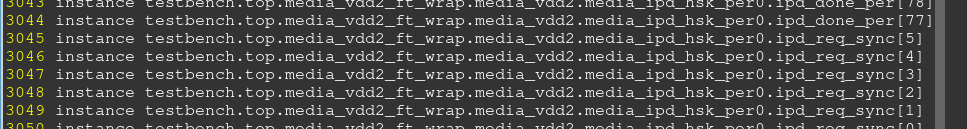
set\_toggle\_excludefile -bitexclude ${PIWORKSPACE}/testbench/blocks/soc\_tb/tool\_data/iccr/soc/exclude/dma\_related\_signals\_toggle\_excludefile.txt

这个表示需要exclude的信号，信号是GUI生成的vRefine文件中提取出来的。

如下是vRefine文件，是HTML格式



最终的目的是转换成cov.cfg需要的txt文件。转换的方法就是从entityName中提取到信号名称，前面加个instance



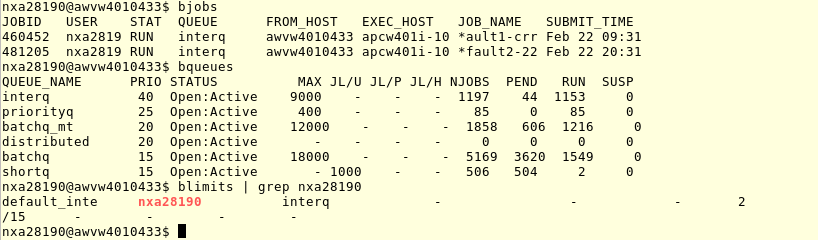
Attention：

如果generate出来的instance信号，需要在cov中用-t的方式，只cover toggle信号。使用-t参数后，如下所示，generate的instance信号全部被exclude



### Bsub

查看服务器使用情况：



Batchq查看后台log：bpeek+ID



## 几个重要的文件夹

* 存放baseaddr

/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.6/testbench/blocks/soc\_tb/tool\_data/sray/mem\_map

rt700\_periph.mem\_map // design 自动生成的文件，包含了DMA base address

dma\_params.mem\_map // 自己定义的文件，可以修改宏定义指向design自动生成的文件

interrupts.mem\_map //design 自动生成，中断向量入口编号

* 功能函数

/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.8/testbench/blocks/soc\_tb/tool\_data/compiler/include

ipc\_dummy\_declaration.h 存放DMA需要的API， ipc\_dma0\_release 之类的函数

/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.8/testbench/common\_blocks/mcu\_tb\_utils/tool\_data/compiler/include

存放pp\_repeat之类的函数

/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.7/testbench/common\_blocks/v\_ms\_spp\_dma3\_nxt\_vip/tool\_data/compiler/include

存放DMA VIP相关的函数

/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.8/testbench/common\_blocks/v\_ip\_capi/tool\_data/compiler/include

存放CAPI相关函数， BSET等

/home/imxrt700\_verif\_nobackup2/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.8/testbench/blocks/soc\_tb/testbench/forces\_v

存放force的信号

/home/imxrt700\_verif\_nobackup2/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.8/testbench/blocks/soc\_tb/tool\_data/compiler/cm33/cm33\_s/include/pre\_post\_main.h

Pre\_post\_main, main函数之前的操作

### Spec

/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.11/testbench/blocks/soc\_tb/tool\_data/script/spec

# Doc

RT700 kick off meeting:

<https://nxp1-my.sharepoint.com/personal/allen_wang_nxp_com/_layouts/15/onedrive.aspx?sortField=DateShared&isAscending=false&groupBy=&id=%2Fpersonal%2Fbei%5Fxia%5Fnxp%5Fcom%2FDocuments%2FRecordings%2F%5BRT700%5D%20SOC%20verification%20kick%20off%20meeting%2D20220427%5F141731%2DMeeting%20Recording%2Emp4&listurl=%2Fpersonal%2Fbei%5Fxia%5Fnxp%5Fcom%2FDocuments&parentview=3>

# Make

common\_blocks/v\_ip\_makefile/tool\_data/compiler/makefile.compile

显示build elf需要的文件依赖

Log：Build Directory

common\_blocks/v\_ip\_makefile/tool\_data/compiler/makefile.compile.gxx

C编译

### C-Compile

%.o: %.c

@echo -e "\n\033[0;32m[$@] $^\033[0m"

@echo -e "\n\033[0;32m[$@] Compile \033[0m"

$(CC) $(CPPFLAGS) $(CFLAGS) $(ALL\_BLOCKS\_INCLUDE\_PATHS\_SWITCH) -MD -c $^ -o $@

testbench/blocks/soc\_tb/testbench/makefile.opts

定义C编译参数

-f /home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/testbench/makefile.stimulus

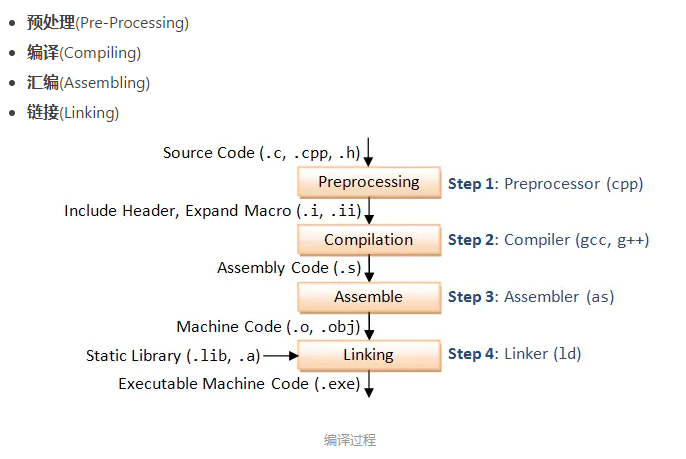
# GCC

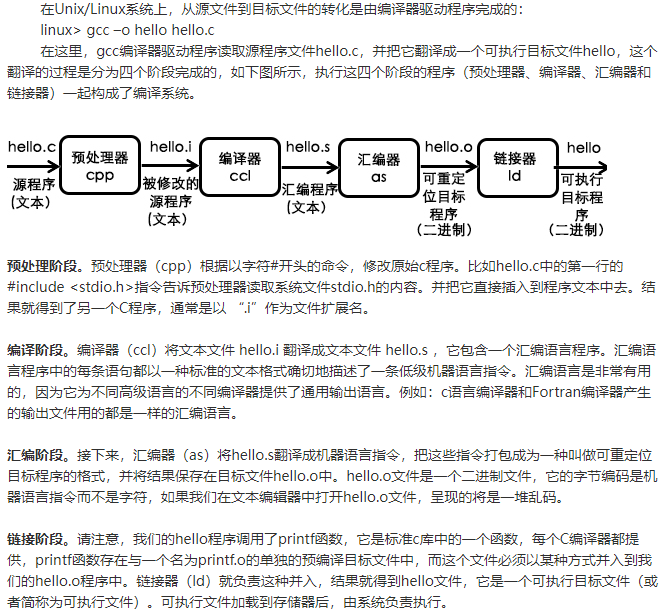
<https://www.jianshu.com/p/1bab86143f1c>

## thumb&ARM 指令集

<https://chan-shaw.github.io/2020/03/20/arm%E6%B1%87%E7%BC%96%E8%AF%AD%E8%A8%80%E5%AD%A6%E4%B9%A0%E7%AC%94%E8%AE%B0/>

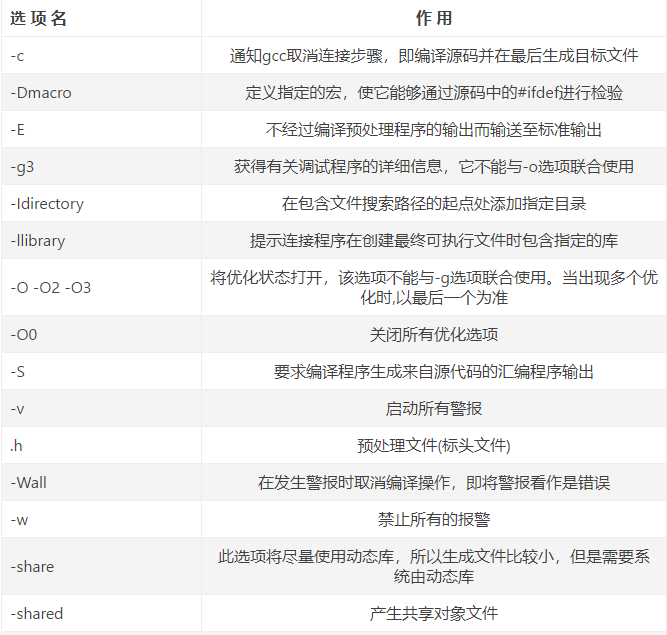
## 编译过程





## 参数详解

### 常用参数





### 预编译 -E

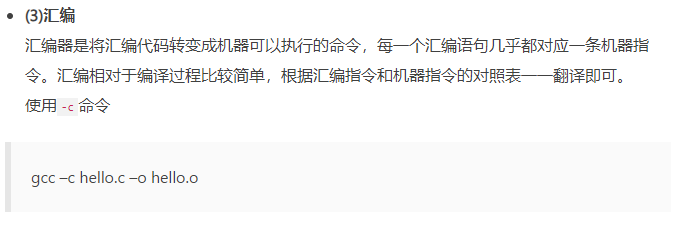


### 编译-S



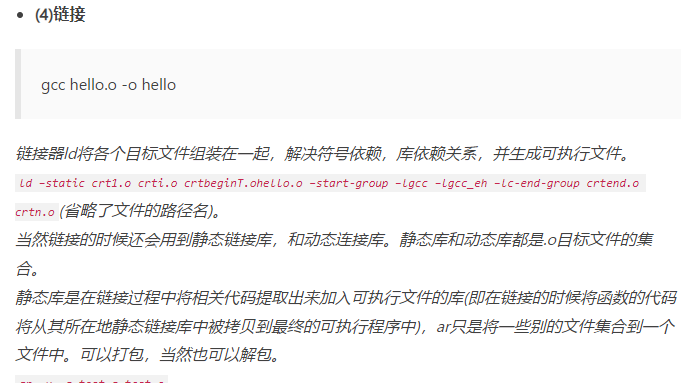
### 汇编 -C

汇编得到的结果没有链接，不可执行



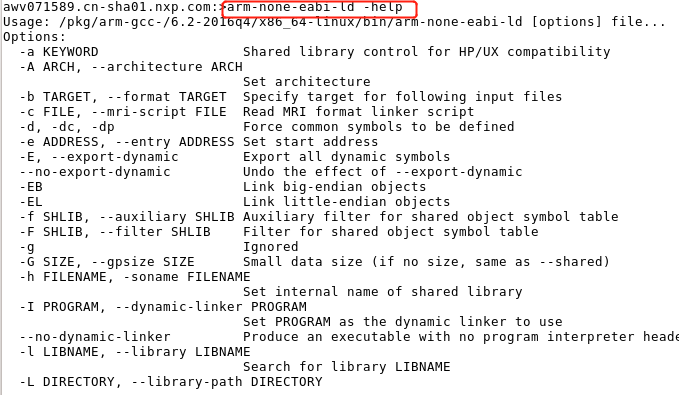
### 链接

也可以使用ld命令将汇编得到的.o连接成可执行文件

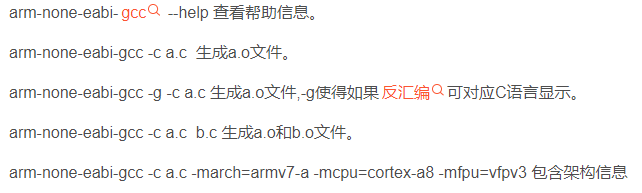


## arm-none-eabi交叉编译工具

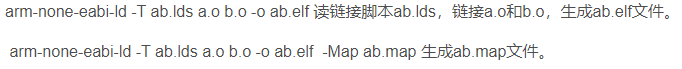
### help



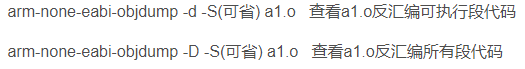
### [arm](https://so.csdn.net/so/search?q=arm&spm=1001.2101.3001.7020)-none-eabi-gcc



### arm-none-eabi-ld



### arm-none-eabi-objdump



### arm-none-eabi-objcopy

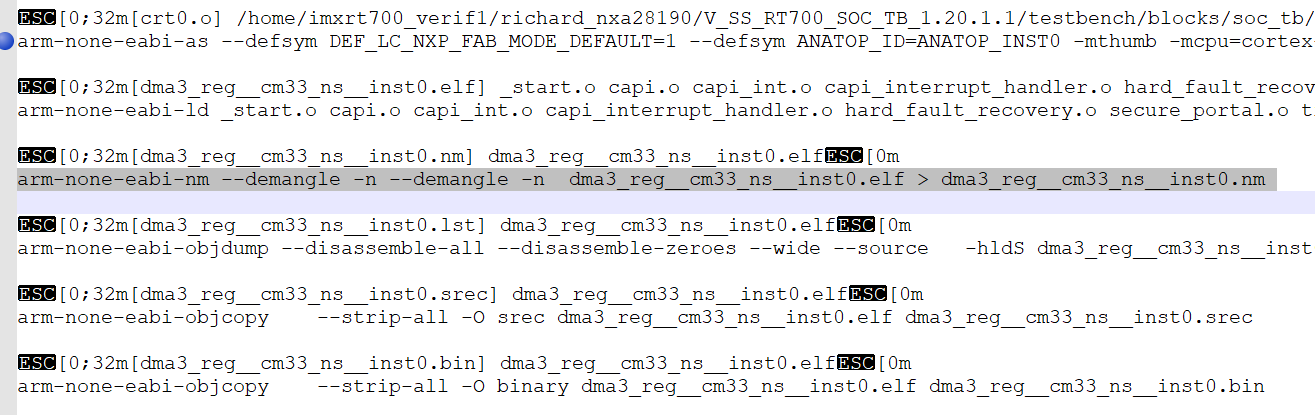


### arm-none-eabi-size

计算文件大小



## RT700用到的gcc command



## 查找log 关键词

Build Directory

arm-none-eabi-as

pre\_main: In pre main of

testbench.load\_memory\_cm33

testbench.load\_memory\_cm33\_m

# RT700

## Master ID

|  |  |  |
| --- | --- | --- |
| **MASTER** | **5-bit ID**  **Hex** | **4-bit ID**  **Hex** |
| CPU0\_S\_AHB\_MASTER\_ID | 00h | 0 |
| CPU0\_C\_AHB\_MASTER\_ID | 10h | 0 |
| DMA0\_AHB\_MASTER\_ID | 01h | 1 |
| DMA1\_AHB\_MASTER\_ID | 02h | 2 |
| PKC\_AHB\_MASTER\_ID | 0Ah | A |
| CSS\_AHB\_MASTER\_ID | 03h | 3 |
| HIFI4\_AXI\_MASTER\_ID | 04h | 4 |
| CPU1\_S\_AHB\_MASTER\_ID | 05h | 5 |
| CPU1\_C\_AHB\_MASTER\_ID | 15h | 5 |
| DMA2\_AHB\_MASTER\_ID | 06h | 6 |
| DMA3\_AHB\_MASTER\_ID | 07h | 7 |
| HIFI1\_AHB\_MASTER\_ID | 08h | 8 |
| EZH\_AHB\_MASTER\_ID | 09h | 9 |
| ETR\_AHB\_MASTER\_ID | 0Eh | E |
| EMMC0\_AXI\_MASTER\_ID | 0Bh | B |
| EMMC1\_AXI\_MASTER\_ID | 1Bh | B |
| USB\_HS\_AHB\_MASTER\_ID | 0Ch | C |
| USB\_E\_AHB\_MASTER\_ID | 1Ch | C |
| JPEG\_AXI\_MASTER\_ID | 16h | 6 |
| PNG\_AXI\_MASTER\_ID | 17h | 7 |
| GPU\_AXI\_MASTER\_ID | 0Dh | D |
| LCDIF\_AXI\_MASTER\_ID | 1Dh | D |
| DAP\_AHBAP\_MASTER\_ID | 0Fh | F |
| TEST\_PORT\_AHB\_MASTER\_ID | 1Fh | F |
| MTR\_AHB\_MASTER\_ID | 1Eh | E |

# code

## Boot

### Linkfile 解析

<https://blog.csdn.net/ymj321/article/details/116937630>

V\_SS\_RT700\_SOC\_TB\_1.10/testbench/blocks/soc\_tb/tool\_data/compiler/cm33/cm33\_s/ldscripts/cm33\_s.lnk

#### 代码段，数据段，BSS段，堆，栈

<https://www.cnblogs.com/zl1991/p/15039949.html>

#### makefile 解析

make过程中产生的make文件，bin，hex，反汇编，map文件都会存放到如下的目录中

V\_SS\_RT700\_SOC\_TB\_1.10/testbench/blocks/soc\_tb/tool\_data/Verilog

Run\_all.sh

1. run\_make\_stim.sh

Make -f testbench/blocks/soc\_tb/testbench/makefile.stimulus -I testbench/blocks/soc\_tb/testbench

-C testbench/blocks/soc\_tb/vectors/dma/stimulus

2. runsim.sh

3. run\_make\_results.sh

### C编译运行

testbench/blocks/soc\_tb/tool\_data/compiler/cm33/cm33\_s/ cm33\_s.lnk

/\* Set the initial value for the stack pointer \*/

/\* Leave 8 bytes at the top for the CAPI \*/

\_\_SP\_INIT = ADDR(stack) + SIZEOF(stack) - 8;

/\* GDB needs to be told where the entry-point is \*/

\_\_START\_ADDR = ADDR(.startup);

ENTRY(\_\_START\_ADDR);

Linkfile指定了入口地址，从.startup段中获取第一条指令。。。  
对应的crt0.S中会指定startup段。

#### Crt0.S

V\_SS\_RT700\_SOC\_TB\_1.9/testbench/blocks/soc\_tb/tool\_data/compiler/cm33/src/\_start.c

ROM offset 0开始存放0x400个中断数据，0x400处为\_\_startup:

CPU上电后，第一个运行的命令就是从这里获取。Link文件中指定了startup段为CPU获取的第一个指令

.section .vectors,"ax"

.globl \_\_vectors

\_\_vectors:

.word \_\_SP\_INIT @ 0 000: reset stack pointer val

.word \_\_startup+1 @ 1 004: reset start address

@ The +1 is to indicate Thumb mode

.ifdef MEMORY\_ate

.else

.rept 254

.word \_capi\_interrupt\_handler+1 @ 2-255 The remaining 254 vectors //伪指令rept重复执行254次。对应二进制binary word2-256都是一样的数值，为\_capi\_interrupt\_handler+1

.endr @ The +1 is to indicate Thumb mode

.endif @ MEMORY\_ate

.global vector\_table\_end

vector\_table\_end:

.section .text.startup,"ax" //linkfile 这段的数据被配置到了ROM中

.globl \_\_startup

\_\_startup:

MOV r0,#0 @ Initialize the GPRs

MOV r1,#0

MOV r2,#0

MOV r3,#0

MOV r4,#0

MOV r5,#0

MOV r6,#0

MOV r7,#0

CPSIE i @ Unmask interrupts

BL \_start @ call the C code

#### \_Start.c

V\_SS\_RT700\_SOC\_TB\_1.9/testbench/blocks/soc\_tb/tool\_data/compiler/cm33/src/\_start.c

C入口函数为\_start

第一个C函数：

void \_start()

{ int main\_status;

pre\_main();

main\_status = main();

post\_main();

CAPI\_END\_SIM(main\_status);

}

### Pre\_main 、 post\_main

V\_SS\_RT700\_SOC\_TB\_1.10/testbench/blocks/soc\_tb/tool\_data/compiler/cm33/cm33\_s/include/ pre\_post\_main.h

#### void pre\_main()

{

//SHOW32("pre\_main", "SHOW32= ", 0x32323232);

INFO("pre\_main","In pre main of cm33\_s.");

WR32(capi\_error\_counter, 0x0);//berk will remove it when we have GPR to replace

INFO("pre\_main","release\_all\_peripheral\_rst.");

release\_all\_peripheral\_rst();

INFO("pre\_main","initialize clock....");

clkctl\_initial\_config();

INFO("pre\_main","config sense cm33 vector....");

/\*to do design will change register from sense to common syscon

#ifdef CM33\_M\_RESET\_VECTOR

W32(REG32(0x40042098), CM33\_M\_RESET\_VECTOR>>7);//system\_svtor\_i

W32(REG32(0x4004209C), CM33\_M\_RESET\_VECTOR>>7);//system\_nsvtor\_i

#else

W32(REG32(0x40042098), S\_SSRAM\_P22\_BASE>>7);//system\_svtor\_i

W32(REG32(0x4004209C), NS\_SSRAM\_P22\_BASE>>7);//system\_nsvtor\_i

#endif \*/

TRIGGER\_SET(CM33\_TBCOMM\_BASE);

TRIGGER\_SET(CM33\_TBCOMM\_BASE+1);

#ifdef CM33\_NS\_RESOURCES\_SWITCH

INFO(\_\_FILE\_\_, "secure state switching configuration");

enable\_idau();

configure\_sau();

if (!tt\_get\_addr\_secure\_attr((address\_t)(&enable\_idau)))

ERROR("pre\_main", "Function address is Non-secure");

init\_nonsec\_stack( \_\_SP\_INIT\_\_SYMBOL\_\_ ); // initialize NS stack pointer

WR32(NSEC(SYSCTRL\_VTOR), \_\_vectors\_\_SYMBOL\_\_ ); // initialize NS vector table base

MB\_PUT32(SEC2NS\_NSC\_VNR\_ADDR\_MBOX, (address\_t)&nsc\_func\_veneer);

MB\_PUT32(SEC2NS\_NSC\_ADDR\_MBOX, (address\_t)&nsc\_func);

MB\_PUT32(NS2SEC\_POST\_MAIN\_HDSK\_MBOX, 0xdeaddead);

//sec2ns\_switch\_blxns(0x0fffe400 );

#endif

INFO("pre\_main","out pre main of cm33\_s.");

return;

}

#### void post\_main()

{

INFO("post\_main","In post main of cm33\_s ");

#ifdef CM33\_NS\_RESOURCES\_SWITCH

uint32\_t rdata;

MB\_GET32(NS2SEC\_POST\_MAIN\_HDSK\_MBOX, rdata);

SHOW32("CM33\_POST\_MAIN","rdata = ",rdata);

FLAG\_WAIT(9,1);

INFO("post\_main","End post main ");

//if (rdata>0xbabeface)

// ERROR("post\_main", "NS code did not finish executing");

#endif

SHOW32("post\_main", "capi\_error\_counter= ", capi\_error\_counter);

}

#### Main()

Test case中的main函数

V\_SS\_RT700\_SOC\_TB\_1.9/testbench/blocks/soc\_tb/vectors/dma/stimulus/dma3\_interrupt.c

### CM33 NS boot

Testbench V1.20.1.1为例

分两步，

1. testbench preload CM33 secure的code 到ROM跟secure SRAM中。 Preload non-secure demo 到NS ram中.
2. Secure demo pre\_main, 配置non-secure的vector，配置SAU,IDAU对non-secure空间进行配置

执行sanity\_cm\_init.c中的main函数，call Non-secure的C code。 \_start.c

Code 分布：

CM33 secure demo： 汇编-》ROM, code、data -》SRAM16, nsc-》sram0 末尾

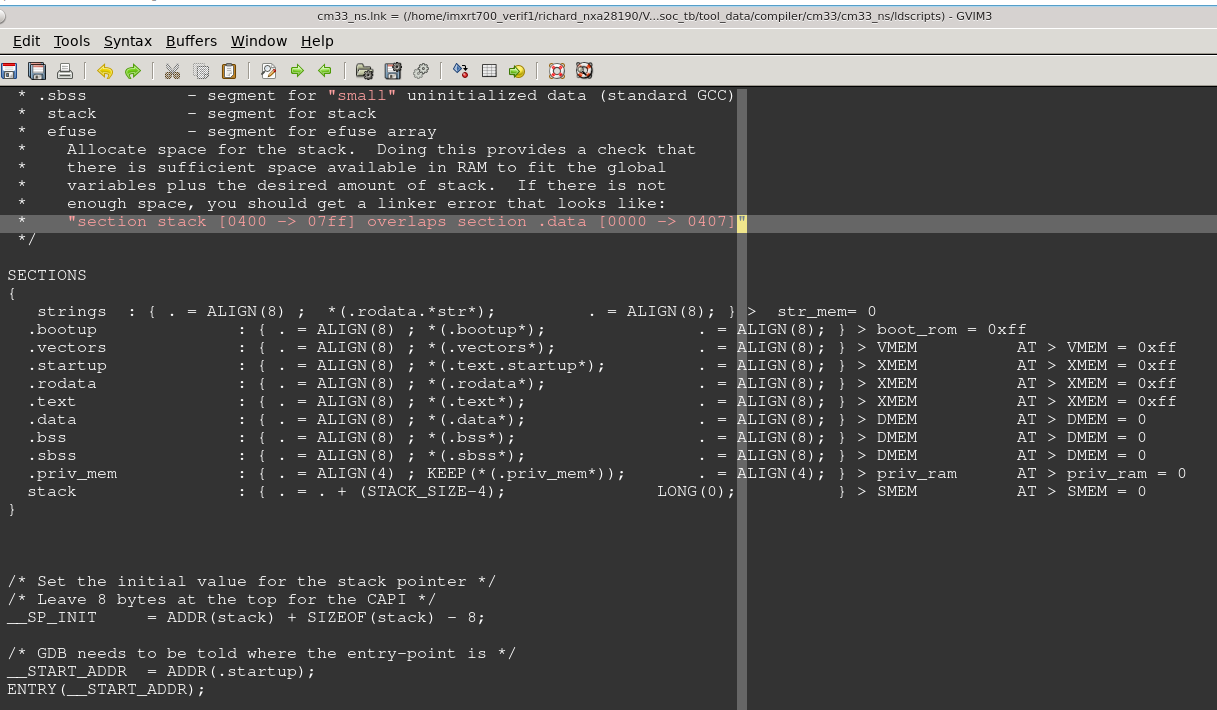
CM33 NS demo： code，data-》SRAM4 NS , vector -》SRAM1 NS

#### 生成linkfile

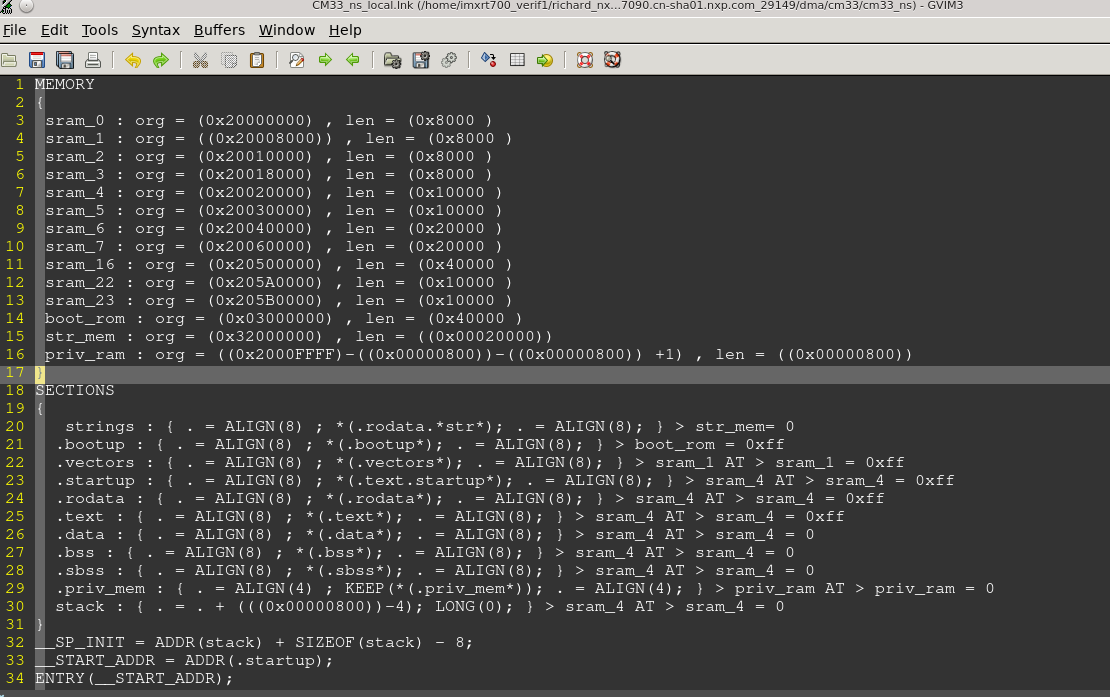
##### Non-secure demo linkfile

cat /home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/cm33/cm33\_ns/ldscripts/cm33\_ns.lnk | m4 -I/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/include -P | cpp -P -Dcm33\_ns -DCM33\_NS\_RESOURCES -DCM33\_NS -DVMEM=sram\_1 -DXMEM=sram\_4 -DUMEM=sram\_4 -DSMEM=sram\_4 -DDMEM=sram\_4 -DSTRAMEM=sram\_4 -I/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/include > /home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/verilog/verilog\_FILENAMES\_apc7092.cn-sha01.nxp.com\_24999/dma/cm33/cm33\_ns/CM33\_ns\_local.lnk

cm33\_ns.lnk



生成的linkfile， CM33\_ns\_local.lnk 。。 SRAM1放vector，sram4放code



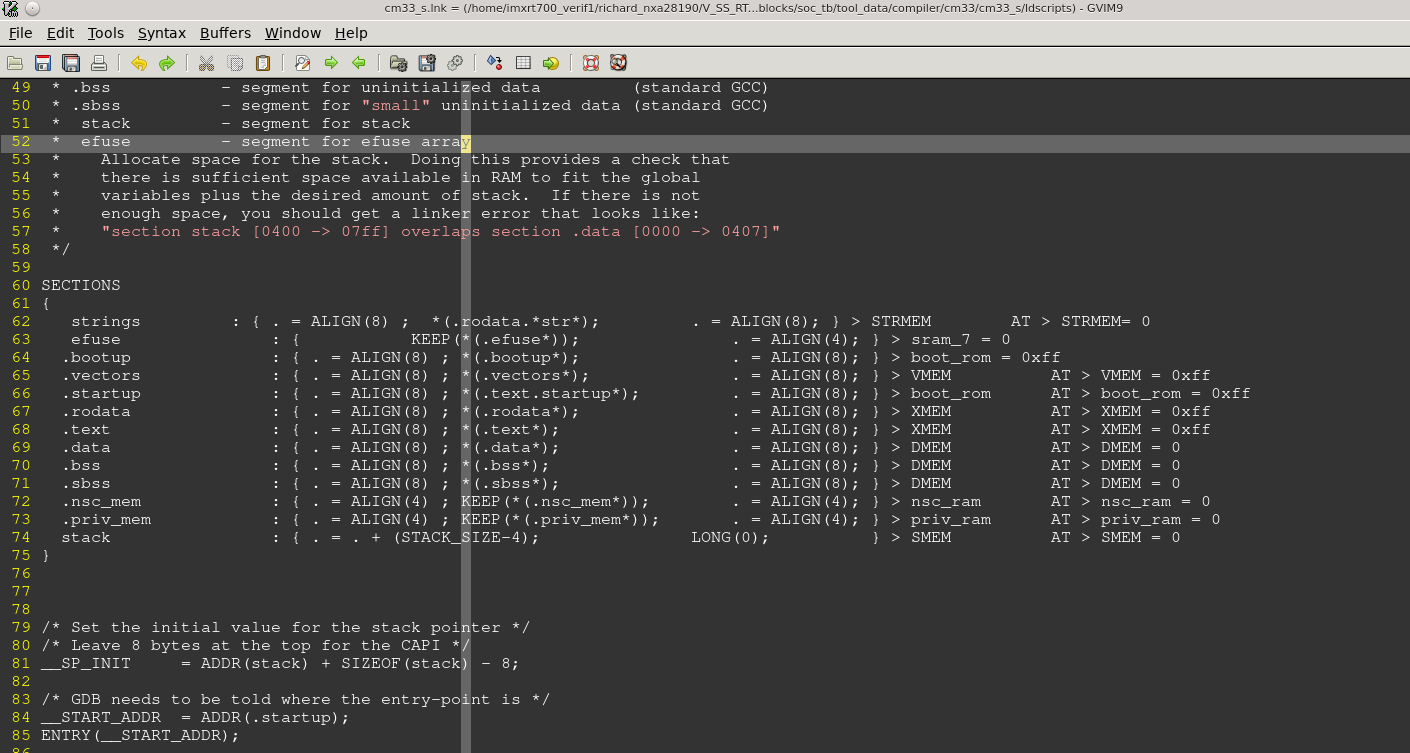
##### Secure demo linkfile

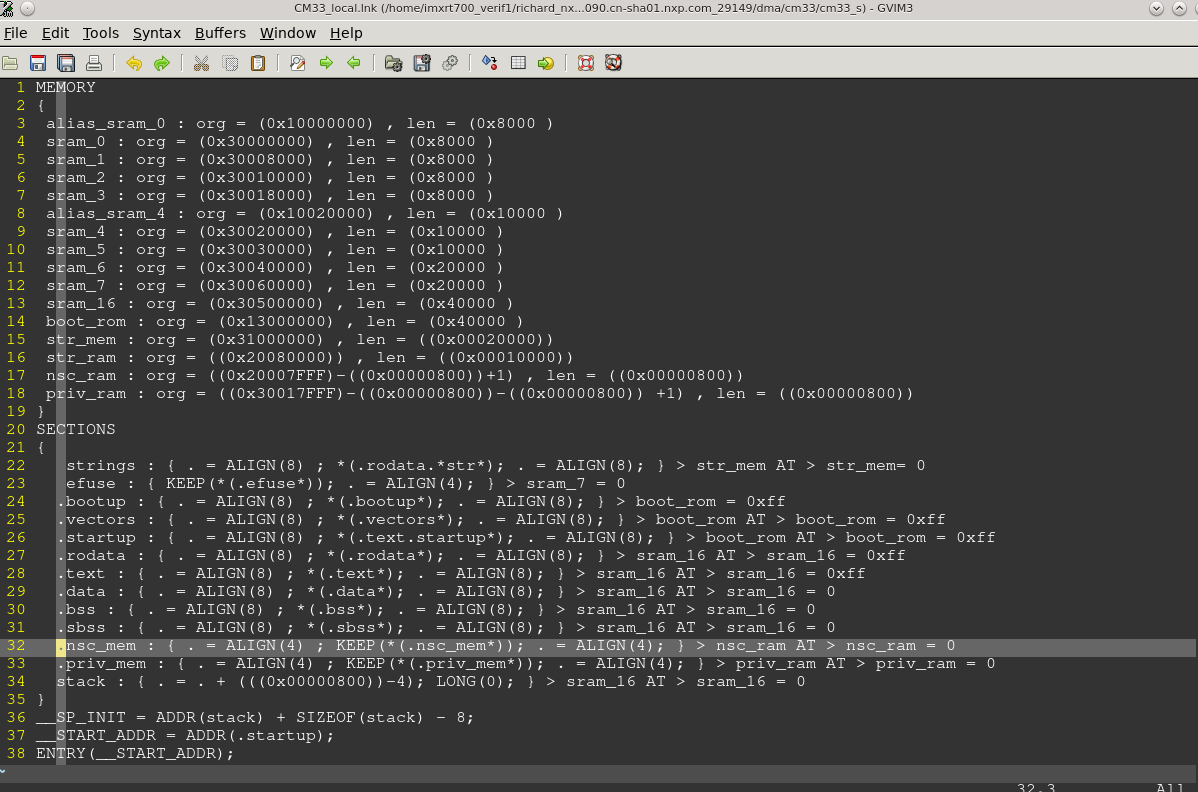
### C-Stimulus : /home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/include/sanity\_cm33\_init.c

### C-RunTime : /home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/cm33/cm33\_s/src/crt0.s

cat /home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/cm33/cm33\_s/ldscripts/cm33\_s.lnk | m4 -I/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/include -P | cpp -P -Dcm33 -DCM33\_RESOURCES -DCM33 -DVMEM=boot\_rom -DSTRMEM=str\_mem -DXMEM=sram\_16 -DUMEM=sram\_16 -DSMEM=sram\_16 -DDMEM=sram\_16 -DSTRAMEM=sram\_16 -I/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/include> /home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/verilog/verilog\_FILENAMES\_apc7092.cn-sha01.nxp.com\_24999/dma/cm33/cm33\_s/CM33\_local.lnk

SRAM16放code，data。 汇编启动代码放置在ROM中





ENTRY(\_\_START\_ADDR) //告诉链接器程序的启动地址，

#### 编译

##### Non-secure demo

编译.c

arm-none-eabi-g++ -D DEF\_LC\_NXP\_FAB\_MODE\_DEFAULT=1 -D ANATOP\_ID=ANATOP\_INST0 -D SRC=S\_SSRAM\_P10\_BASE -D DST=S\_SSRAM\_P10\_BASE -D SRC\_NS=NS\_SSRAM\_P10\_BASE -D DST\_NS=NS\_SSRAM\_P10\_BASE -D LEN=0x80000 -D ADDR\_START=0 -DCM33\_NS\_RESOURCES=1 -Dcm33\_ns -DCM33\_NS\_RESOURCES -DCM33\_NS -DVMEM=sram\_1 -DXMEM=sram\_4 -DUMEM=sram\_4 -DSMEM=sram\_4 -DDMEM=sram\_4 -DSTRAMEM=sram\_4 -I /home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/include/ -DCM33 -DCM33\_NS\_RESOURCES=1 -DCPRINT\_REGS\_BASE=0x2001FFE0 -DCPRINT\_CMD\_OFFSET=0x00 -DCPRINT\_ARG\_OFFSET=0x8 -DCPRINT\_RSP\_OFFSET=0x4 -Dcm33\_ns -DCM33\_NS\_RESOURCES -DCM33\_NS -DVMEM=sram\_1 -DXMEM=sram\_4 -DUMEM=sram\_4 -DSMEM=sram\_4 -DDMEM=sram\_4 -DSTRAMEM=sram\_4 -DDMA\_ID=INST0 -D CAPI\_INCLUDE\_C=0 -D CAPI\_INCLUDE\_STARTUP=0 -D STARTUP\_STIM=pre\_main -D SHUTDOWN\_STIM=post\_main -D CORE\_NUM=0 -D PH\_BASE=0x37FFFFF0 -D CPRINT\_REGS\_BASE=0x37FFFFF0 -D TARGET\_MEM\_START=0x34000000 -D TARGET\_MEM\_SIZE=0x00070000 -D TARGET\_MEM\_WIDTH=64 - -MD -c /home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/cm33/src/\_start.c -o \_start.o

编译.s

arm-none-eabi-as --defsym DEF\_LC\_NXP\_FAB\_MODE\_DEFAULT=1 --defsym ANATOP\_ID=ANATOP\_INST0 --defsym SRC=S\_SSRAM\_P10\_BASE --defsym DST=S\_SSRAM\_P10\_BASE --defsym SRC\_NS=NS\_SSRAM\_P10\_BASE --defsym DST\_NS=NS\_SSRAM\_P10\_BASE --defsym LEN=0x80000 --defsym ADDR\_START=0 -mthumb -mcpu=cortex-m33 --defsym CM33\_NS\_RESOURCES=1 --defsym cm33\_ns=1 --defsym CM33\_NS\_RESOURCES=1 --defsym CM33\_NS=1 --defsym cm33\_ns=1 --defsym CM33\_NS=1 --defsym VMEM=sram\_1 --defsym XMEM=sram\_4 --defsym UMEM=sram\_4 --defsym SMEM=sram\_4 --defsym DMEM=sram\_4 --defsym STRAMEM=sram\_4 --defsym DMA\_ID=INST0 --defsym CAPI\_INCLUDE\_C=0 --defsym CAPI\_INCLUDE\_STARTUP=0 --defsym STARTUP\_STIM=pre\_main --defsym SHUTDOWN\_STIM=post\_main --defsym CORE\_NUM=0 --defsym PH\_BASE=0x37FFFFF0 --defsym CPRINT\_REGS\_BASE=0x37FFFFF0 --defsym TARGET\_MEM\_START=0x34000000 --defsym TARGET\_MEM\_SIZE=0x00070000 --defsym TARGET\_MEM\_WIDTH=64 /home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/cm33/cm33\_ns/src/crt0.s -o crt0.o

##### Secure demo

编译.c

arm-none-eabi-g++ -D DEF\_LC\_NXP\_FAB\_MODE\_DEFAULT=1 -D ANATOP\_ID=ANATOP\_INST0 -D SRC=S\_SSRAM\_P10\_BASE -D DST=S\_SSRAM\_P10\_BASE -D SRC\_NS=NS\_SSRAM\_P10\_BASE -D DST\_NS=NS\_SSRAM\_P10\_BASE -D LEN=0x80000 -D ADDR\_START=0 -DCM33\_RESOURCES=1 -DNONE -DNONE -DNONE -Dcm33\_s -DCM33\_RESOURCES -DCM33 -DVMEM=boot\_rom -DXMEM=sram\_16 -DSTRMEM=str\_mem -DUMEM=sram\_16 -DSMEM=sram\_16 -DDMEM=sram\_16 -DSTRAMEM=sram\_16 -D CM33\_NS\_RESOURCES\_SWITCH=1 -I /home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/include/ -I DTEST\_NAME=dma3\_slave\_access\_sram\_\_cm33\_ns\_\_inst0 -DVECTOR\_SET=dma -DMAIN\_TEST=main -DDEF\_LC\_NXP\_FAB\_MODE\_DEFAULT=1 -DANATOP\_ID=ANATOP\_INST0 -DSRC=S\_SSRAM\_P10\_BASE -DDST=S\_SSRAM\_P10\_BASE -DSRC\_NS=NS\_SSRAM\_P10\_BASE -DDST\_NS=NS\_SSRAM\_P10\_BASE -DLEN=0x80000 -DADDR\_START=0 -fdata-sections -fno-exceptions -std=c99 -Wall -Werror=declaration-after-statement -Wno-unused-function -ffunction-sections -falign-functions=4 -fno-zero-initialized-in-bss -O1 -I/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/vectors/dma/stimulus -DACTIVE\_CORES=0x3 -DACTIVE\_CBFMS=0x0 -Wfatal-errors -Werror=implicit-function-declaration -Wmain -Werror=main -D ASM=\_\_asm\_\_ -fno-zero-initialized-in-bss -mthumb -mcpu=cortex-m33 -nostdlib -ffunction-sections -g -mthumb -march=armv8-m.main -mfpu=fpv5-sp-d16 -mfloat-abi=hard -fpermissive -Wno-narrowing -DCM33\_RESOURCES=1 -DCPRINT\_REGS\_BASE=0x3001FFE0 -DCPRINT\_CMD\_OFFSET=0x00 -DCPRINT\_ARG\_OFFSET=0x8 -DCPRINT\_RSP\_OFFSET=0x4 -DADDR\_START\_\_SYMBOL\_\_=0x00000000 -DANATOP\_ID\_\_SYMBOL\_\_=0x00000000 -DCAPI\_INCLUDE\_C\_\_SYMBOL\_\_=0x00000000 -DCAPI\_INCLUDE\_STARTUP\_\_SYMBOL\_\_=0x00000000 -DCORE\_NUM\_\_SYMBOL\_\_=0x00000000 -DDMA\_ID\_\_SYMBOL\_\_=0x00000000 -DDMEM\_\_SYMBOL\_\_=0x00000000 -DDST\_\_SYMBOL\_\_=0x00000000 -DDST\_NS\_\_SYMBOL\_\_=0x00000000 -DSHUTDOWN\_STIM\_\_SYMBOL\_\_=0x00000000 -DSMEM\_\_SYMBOL\_\_=0x00000000 -DSRC\_\_SYMBOL\_\_=0x00000000 -DSRC\_NS\_\_SYMBOL\_\_=0x00000000 -DSTARTUP\_STIM\_\_SYMBOL\_\_=0x00000000 -DSTRAMEM\_\_SYMBOL\_\_=0x00000000 -DUMEM\_\_SYMBOL\_\_=0x00000000 -DVMEM\_\_SYMBOL\_\_=0x00000000 -DXMEM\_\_SYMBOL\_\_=0x00000000 -DCM33\_NS\_\_SYMBOL\_\_=0x00000001 -DCM33\_NS\_\_SYMBOL\_\_=0x00000001 -DCM33\_NS\_RESOURCES\_\_SYMBOL\_\_=0x00000001 -DCM33\_NS\_RESOURCES\_\_SYMBOL\_\_=0x00000001 -DDEF\_LC\_NXP\_FAB\_MODE\_DEFAULT\_\_SYMBOL\_\_=0x00000001 -Dcm33\_ns\_\_SYMBOL\_\_=0x00000001 -Dcm33\_ns\_\_SYMBOL\_\_=0x00000001 -DTARGET\_MEM\_MSB\_\_SYMBOL\_\_=0x0000003f -DTARGET\_MEM\_WIDTH\_\_SYMBOL\_\_=0x00000040 -DTARGET\_MEM\_WIDTH\_\_SYMBOL\_\_=0x00000040 -DTARGET\_MEM\_STACK\_SIZE\_\_SYMBOL\_\_=0x00000800 -DTARGET\_MEM\_SIZE\_\_SYMBOL\_\_=0x00070000 -DTARGET\_MEM\_SIZE\_\_SYMBOL\_\_=0x00070000 -DLEN\_\_SYMBOL\_\_=0x00080000 -D\_\_vectors\_\_SYMBOL\_\_=0x20008000 -Dvector\_table\_end\_\_SYMBOL\_\_=0x20008400 -D\_\_START\_ADDR\_\_SYMBOL\_\_=0x20020000 -D\_\_startup\_\_SYMBOL\_\_=0x20020000 -D\_\_end\_\_SYMBOL\_\_=0x20020016 -D\_start\_\_SYMBOL\_\_=0x20020020 -DCAPI\_END\_SIM\_\_SYMBOL\_\_=0x20020038 -DUNIMPLEMENTED\_ISR\_\_SYMBOL\_\_=0x20020058 -DPROCESS\_INTERRUPT\_\_SYMBOL\_\_=0x20020098 -Dcore\_enable\_irq\_\_SYMBOL\_\_=0x200200c0 -D\_capi\_interrupt\_handler\_\_SYMBOL\_\_=0x200200f0 -D\_hard\_fault\_recovery\_\_SYMBOL\_\_=0x2002013c -D\_chk\_prec\_data\_err\_\_SYMBOL\_\_=0x20020146 -D\_chk\_instr\_type\_\_SYMBOL\_\_=0x2002014c -D\_rtn\_addr\_\_SYMBOL\_\_=0x2002016a -D\_chk\_instr\_err\_\_SYMBOL\_\_=0x2002016e -D\_end\_isr\_\_SYMBOL\_\_=0x20020194 -Dset\_interrupt\_secure\_state\_\_SYMBOL\_\_=0x200201b4 -Djump\_to\_nsc\_\_SYMBOL\_\_=0x200201cc -D\_\_NS\_RETURN\_ADDR\_\_SYMBOL\_\_=0x200201ea -Ddma3\_int\_isr\_\_SYMBOL\_\_=0x200201fc -Ddma3\_err\_int\_isr\_\_SYMBOL\_\_=0x20020320 -Ddma\_populate\_request\_\_SYMBOL\_\_=0x200203b4 -Dipc\_dma0\_assign\_\_SYMBOL\_\_=0x200203c4 -Dipc\_dma0\_release\_\_SYMBOL\_\_=0x2002040c -Dipc\_dma1\_assign\_\_SYMBOL\_\_=0x20020454 -Dipc\_dma1\_release\_\_SYMBOL\_\_=0x2002049c -Dipc\_dma2\_assign\_\_SYMBOL\_\_=0x200204e4 -Dipc\_dma2\_release\_\_SYMBOL\_\_=0x2002052c -Dipc\_dma3\_assign\_\_SYMBOL\_\_=0x20020574 -Ddma3\_pcc\_assign\_\_SYMBOL\_\_=0x200205bc -Dipc\_dma3\_release\_\_SYMBOL\_\_=0x20020614 -Ddma3\_pcc\_release\_\_SYMBOL\_\_=0x2002065c -Dipc\_dma0\_rstb\_set\_\_SYMBOL\_\_=0x200206b4 -Dipc\_dma0\_rstb\_clr\_\_SYMBOL\_\_=0x200206fc -Dipc\_dma1\_rstb\_set\_\_SYMBOL\_\_=0x20020744 -Dipc\_dma1\_rstb\_clr\_\_SYMBOL\_\_=0x2002078c -Dipc\_dma2\_rstb\_set\_\_SYMBOL\_\_=0x200207d4 -Dipc\_dma2\_rstb\_clr\_\_SYMBOL\_\_=0x2002081c -Dipc\_dma3\_rstb\_set\_\_SYMBOL\_\_=0x20020864 -Ddma3\_reset\_negate\_\_SYMBOL\_\_=0x200208ac -Dipc\_dma3\_rstb\_clr\_\_SYMBOL\_\_=0x20020904 -Ddma3\_reset\_assert\_\_SYMBOL\_\_=0x2002094c -Ddma3\_setup\_int\_\_SYMBOL\_\_=0x200209a4 -Ddma3\_check\_int\_\_SYMBOL\_\_=0x200209e4 -Ddma3\_setup\_err\_int\_\_SYMBOL\_\_=0x20020a38 -Ddma3\_check\_err\_int\_\_SYMBOL\_\_=0x20020a78 -Ddma3\_enable\_err\_int\_\_SYMBOL\_\_=0x20020ad0 -Ddma3\_validate\_channel\_\_SYMBOL\_\_=0x20020b1c -Ddma3\_setup\_complex\_\_SYMBOL\_\_=0x20020b68 -Ddma3\_setup\_hw\_\_SYMBOL\_\_=0x200212b4 -Ddma3\_start\_sw\_\_SYMBOL\_\_=0x20021304 -Ddma3\_prepare\_source\_data\_\_SYMBOL\_\_=0x20021350 -Ddma3\_check\_destination\_data\_\_SYMBOL\_\_=0x20021368 -Dsoc\_new\_request\_\_SYMBOL\_\_=0x20021bbc -Dsoc\_allocate\_ip\_\_SYMBOL\_\_=0x20021c28 -Dpre\_main\_\_SYMBOL\_\_=0x20021c8c -Dpost\_main\_\_SYMBOL\_\_=0x20021d18 -Dmain\_\_SYMBOL\_\_=0x20021d68 -Dns\_return\_addr\_\_SYMBOL\_\_=0x20022270 -Dinit\_ip\_request\_\_SYMBOL\_\_=0x20022274 -DtrSize\_\_SYMBOL\_\_=0x2002238c -DINTERRUPT\_CONTEXT\_TABLE\_\_SYMBOL\_\_=0x200223a0 -DINTERRUPT\_JUMP\_TABLE\_\_SYMBOL\_\_=0x200227a0 -Dns2sec\_comm\_isr\_msg\_\_SYMBOL\_\_=0x20022ba0 -Dnsc\_addr\_\_SYMBOL\_\_=0x20022ba4 -Dnsc\_veneer\_addr\_\_SYMBOL\_\_=0x20022ba8 -D\_\_SP\_INIT\_\_SYMBOL\_\_=0x20023470 -DTARGET\_MEM\_START\_\_SYMBOL\_\_=0x34000000 -DTARGET\_MEM\_START\_\_SYMBOL\_\_=0x34000000 -DTARGET\_MEM\_END\_\_SYMBOL\_\_=0x3406ffff -DCPRINT\_REGS\_BASE\_\_SYMBOL\_\_=0x37fffff0 -DPH\_BASE\_\_SYMBOL\_\_=0x37fffff0 -DNONE -DNONE -DNONE -Dcm33\_s -DCM33\_RESOURCES -DCM33 -DVMEM=boot\_rom -DXMEM=sram\_16 -DSTRMEM=str\_mem -DUMEM=sram\_16 -DSMEM=sram\_16 -DDMEM=sram\_16 -DSTRAMEM=sram\_16 -DCM33\_NS\_RESOURCES\_SWITCH=1 -D CAPI\_INCLUDE\_C=0 -D CAPI\_INCLUDE\_STARTUP=0 -D STARTUP\_STIM=pre\_main -D SHUTDOWN\_STIM=post\_main -D CORE\_NUM=1 -D PH\_BASE=0x3001FFE0 -D CPRINT\_REGS\_BASE=0x3001FFE0 -D TARGET\_MEM\_START=0x34070000 -D TARGET\_MEM\_SIZE=0x00070000 -D TARGET\_MEM\_WIDTH=64 -I/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/cm33/cm33\_s/include - -MD -c /home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/include/sanity\_cm33\_init.c -o sanity\_cm33\_init.o

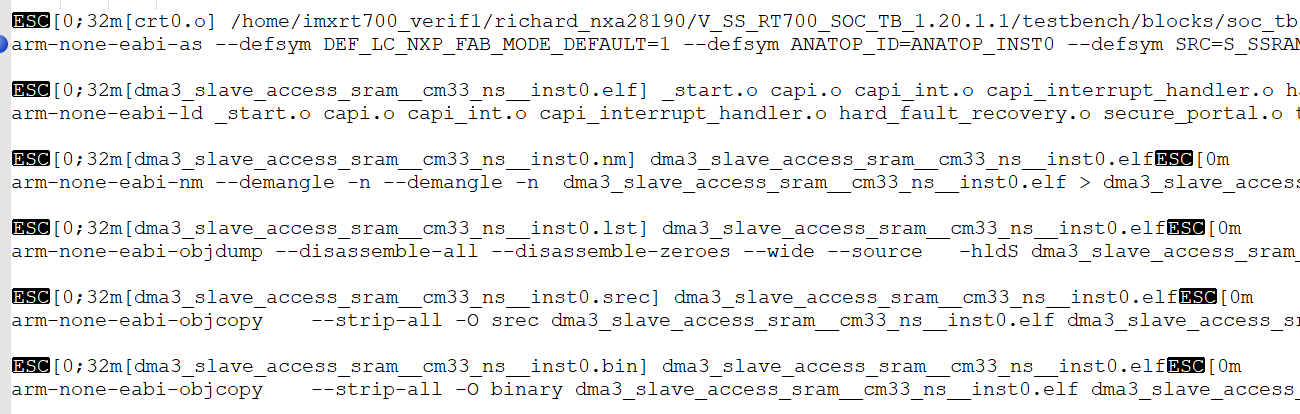
编译.S

arm-none-eabi-as --defsym DEF\_LC\_NXP\_FAB\_MODE\_DEFAULT=1 --defsym ANATOP\_ID=ANATOP\_INST0 --defsym SRC=S\_SSRAM\_P10\_BASE --defsym DST=S\_SSRAM\_P10\_BASE --defsym SRC\_NS=NS\_SSRAM\_P10\_BASE --defsym DST\_NS=NS\_SSRAM\_P10\_BASE --defsym LEN=0x80000 --defsym ADDR\_START=0 -mthumb -mcpu=cortex-m33 --defsym CM33=1 --defsym CM33\_RESOURCES=1 --defsym cm33\_s=1 --defsym CM33\_RESOURCES=1 --defsym CM33=1 --defsym cm33\_s=1 --defsym CM33=1 --defsym VMEM=boot\_rom --defsym XMEM=sram\_16 --defsym STRMEM=str\_mem --defsym UMEM=sram\_16 --defsym SMEM=sram\_16 --defsym DMEM=sram\_16 --defsym STRAMEM=sram\_16 --defsym CM33\_NS\_RESOURCES\_SWITCH=1 --defsym CAPI\_INCLUDE\_C=0 --defsym CAPI\_INCLUDE\_STARTUP=0 --defsym STARTUP\_STIM=pre\_main --defsym SHUTDOWN\_STIM=post\_main --defsym CORE\_NUM=1 --defsym PH\_BASE=0x3001FFE0 --defsym CPRINT\_REGS\_BASE=0x3001FFE0 --defsym TARGET\_MEM\_START=0x34070000 --defsym TARGET\_MEM\_SIZE=0x00070000 --defsym TARGET\_MEM\_WIDTH=64 /home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/cm33/cm33\_s/src/crt0.s -o crt0.o

#### 链接

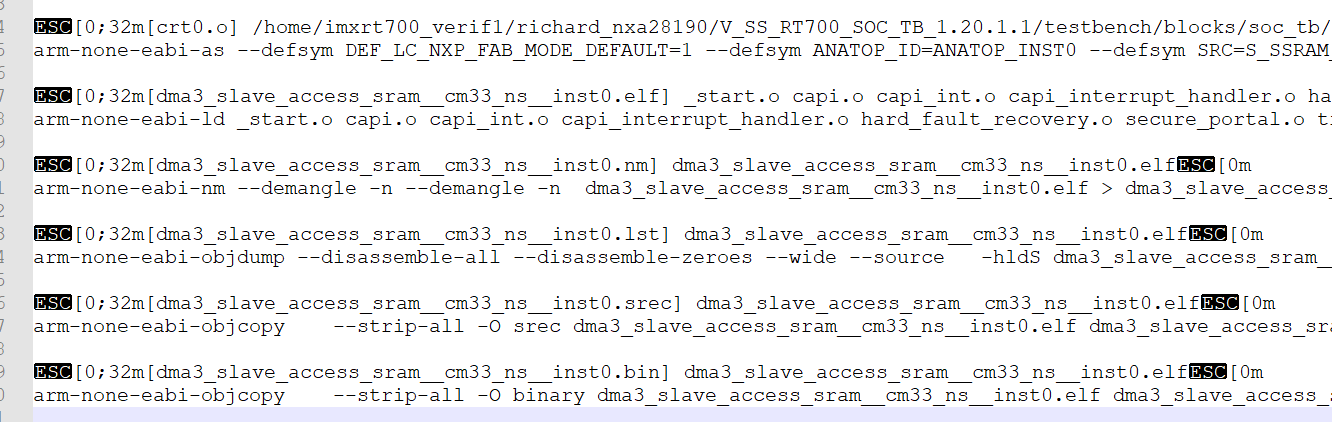
将.o文件按照link文件的要求整合起来，生成map跟elf文件

##### Non-secure demo



arm-none-eabi-ld \_start.o capi.o capi\_int.o capi\_interrupt\_handler.o hard\_fault\_recovery.o secure\_portal.o trustzone\_api.o dma3\_slave\_access.o crt0.o -L/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/cm33/cm33\_ns/lib -L/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/cm33/lib --whole-archive --no-whole-archive -nostdlib -gc-sections --gc-sections -nostdlib --entry=\_\_vectors --defsym TARGET\_MEM\_WIDTH=64 --defsym TARGET\_MEM\_START=0x34000000 --defsym TARGET\_MEM\_STACK\_SIZE=0x800 --defsym TARGET\_MEM\_END=0x3406ffff --defsym TARGET\_MEM\_MSB=63 --defsym TARGET\_MEM\_SIZE=0x00070000 -T./CM33\_ns\_local.lnk -Map dma3\_slave\_access\_sram\_\_cm33\_ns\_\_inst0.map -o dma3\_slave\_access\_sram\_\_cm33\_ns\_\_inst0.elf

##### Secure demo



arm-none-eabi-ld \_start.o capi.o capi\_int.o capi\_interrupt\_handler.o hard\_fault\_recovery.o secure\_portal.o trustzone\_api.o sanity\_cm33\_init.o crt0.o -L/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/cm33/cm33\_s/lib -L/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/cm33/lib --whole-archive --no-whole-archive -nostdlib -gc-sections --gc-sections -nostdlib --entry=\_\_vectors --defsym TARGET\_MEM\_WIDTH=64 --defsym TARGET\_MEM\_START=0x34070000 --defsym TARGET\_MEM\_STACK\_SIZE=0x800 --defsym TARGET\_MEM\_END=0x340dffff --defsym TARGET\_MEM\_MSB=63 --defsym TARGET\_MEM\_SIZE=0x00070000 -T./CM33\_local.lnk -Map dma3\_slave\_access\_sram\_\_cm33\_ns\_\_inst0.map -o dma3\_slave\_access\_sram\_\_cm33\_ns\_\_inst0.elf

#### 启动流程分析

##### Loadfile

Secure demo Load into ROM, sram16

UVM\_INFO @ 1085131.913200 ns --: [testbench.load\_memory\_cm33] load\_memory\_string: ==== File : /home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/verilog/verilog\_FILENAMES\_apc7092.cn-sha01.nxp.com\_24999/CM33\_dma3\_slave\_access\_sram\_\_cm33\_ns\_\_inst0.rom.hex

Non-secure demo, load into SRAM1,SRAM4

UVM\_INFO @ 1085131.913200 ns --: [testbench.load\_memory\_cm33\_ns] load\_memory\_string: ==== File : /home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/verilog/verilog\_FILENAMES\_apc7092.cn-sha01.nxp.com\_24999/CM33\_NS\_dma3\_slave\_access\_sram\_\_cm33\_ns\_\_inst0.rom.hex

##### Boot into non-secure demo

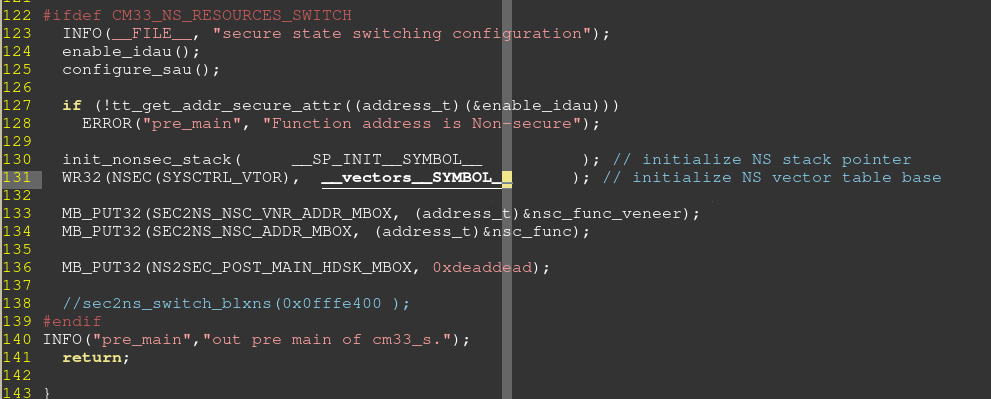
Secure demo从CM33 BOOT 起来后，执行pre\_main函数，跳转之前trustzone会对ram进行non-secure的配置

###### Non-secure 配置

1. 配置non-secure vctor， vector\_\_SYMBOL. Secure demo在编译的时候定义了这个变量，为non-secure demo的vector address



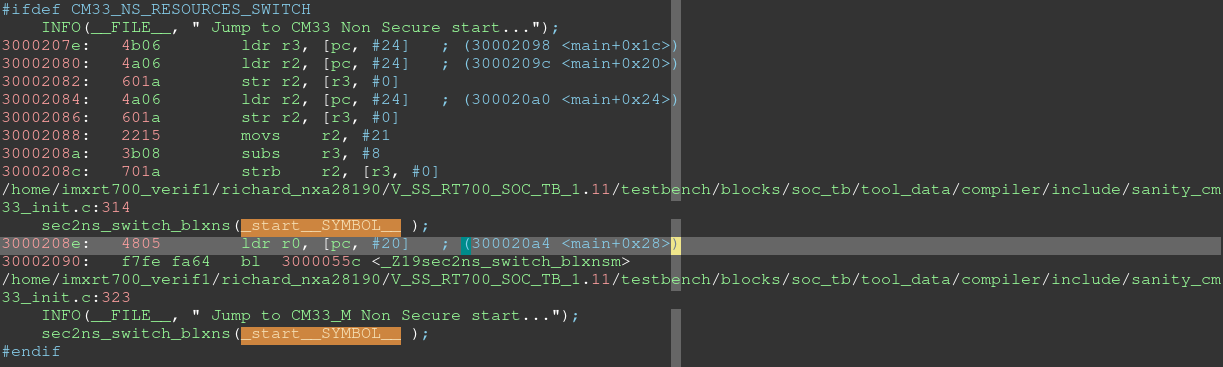
1. Config SAU,idau配置NON-SECURE address, NON-SECURE ram

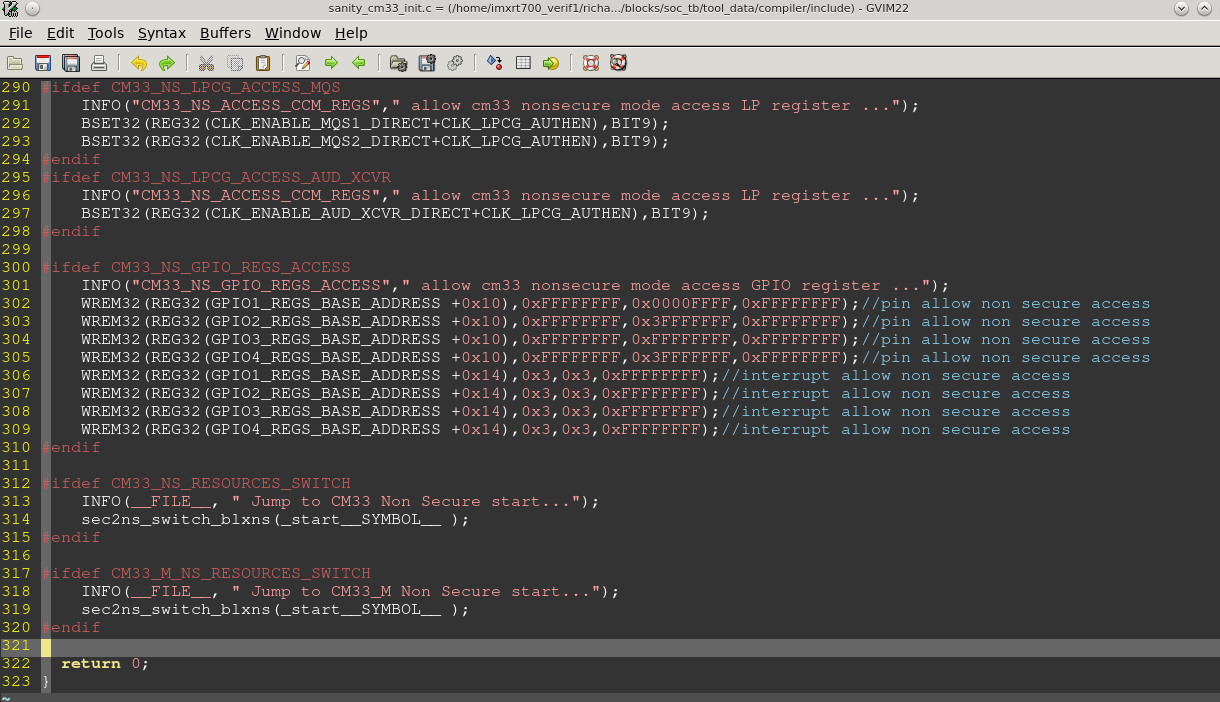


###### Boot into non-secure demo

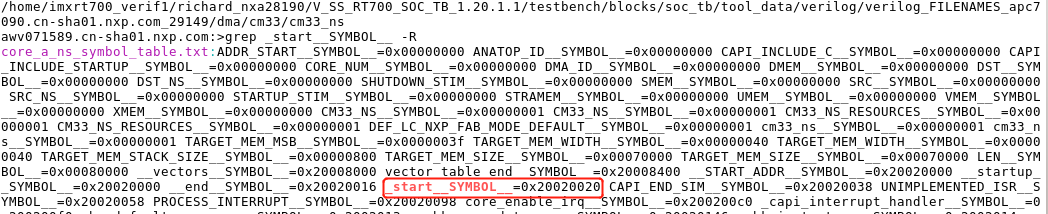
1. 执行Sanity\_cm33\_init.c中的main函数。

CM33\_NS\_RESOURCES\_SWITCH<#ns_source_switch> 存在，跳转到non-secure demo中，跳转地址<#start_symbol> (NON-SECURE code 的\_start.c)

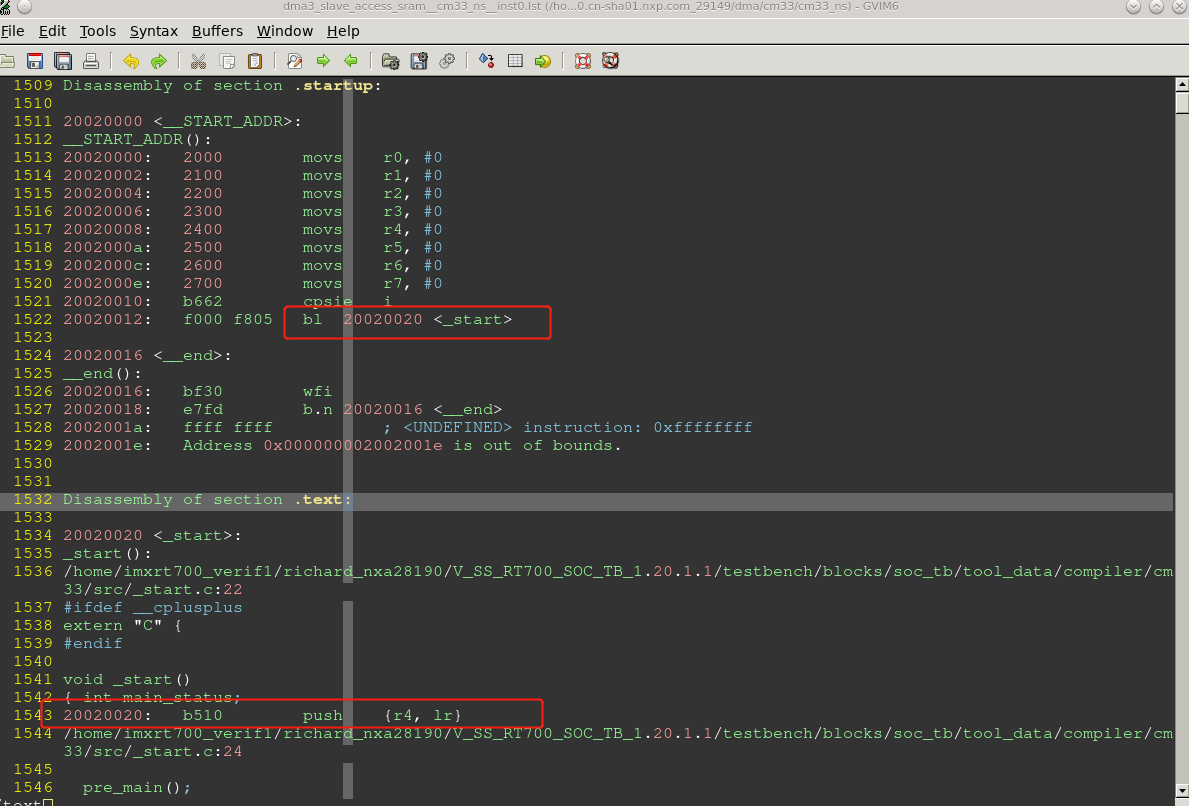




CM33\_ns\_local.lnk



CM33 ns demo反汇编lst文件



##### 信号分析

###### Vector重定向

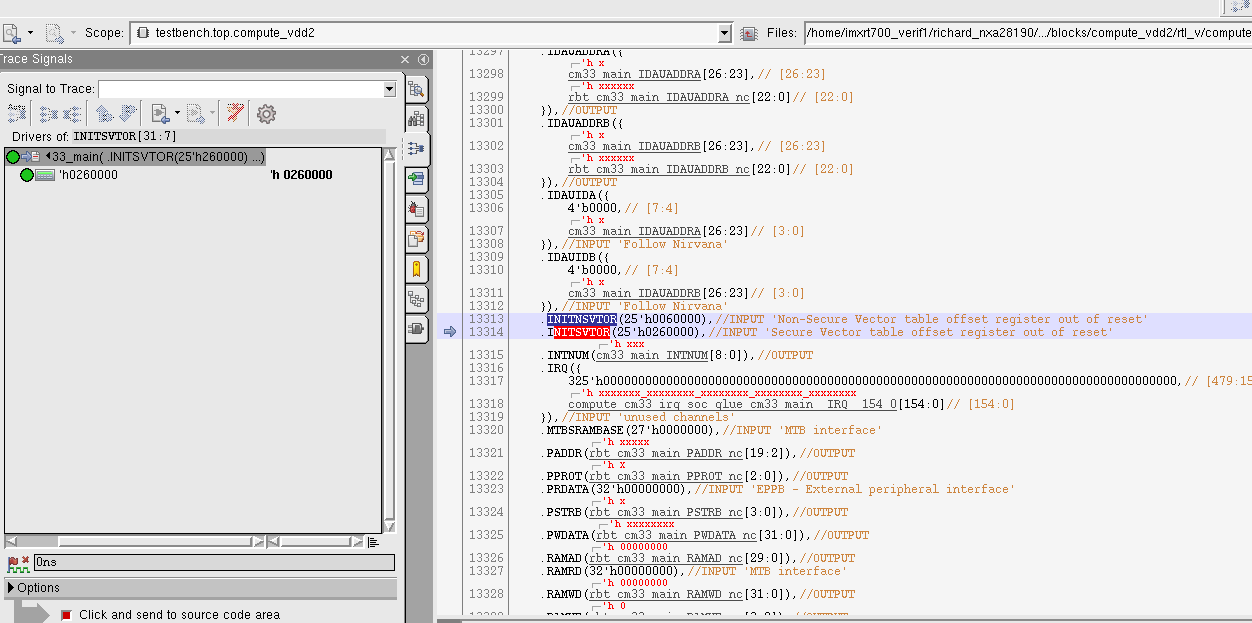
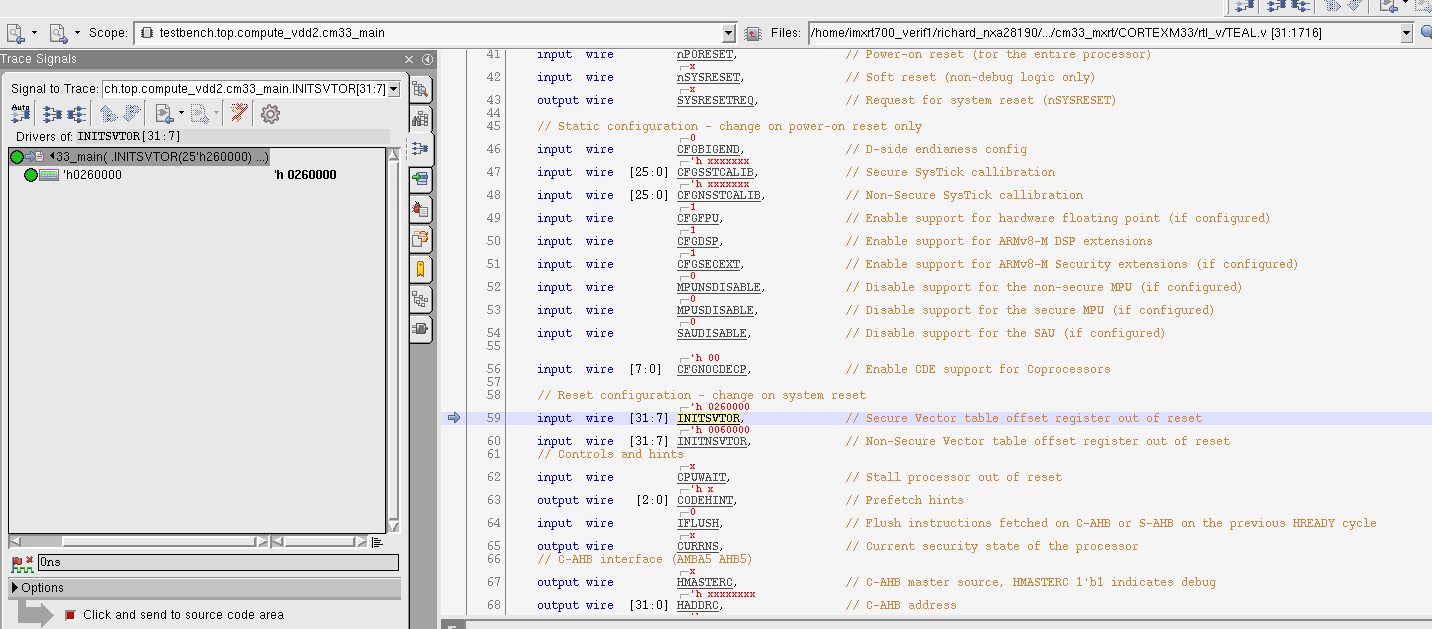
芯片上电后，secure vector会默认从0，重定向到0x13000000.

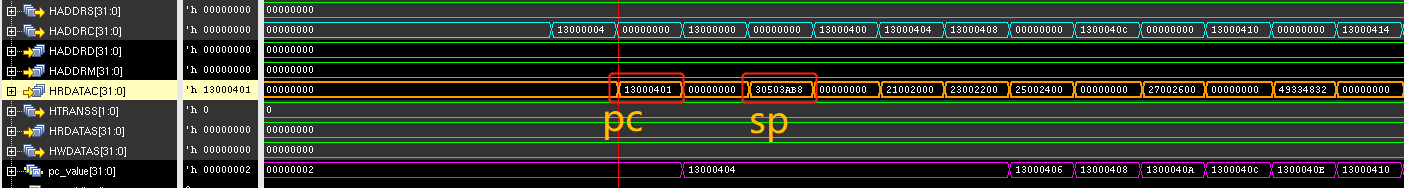
然后从0x13000400处取指令执行

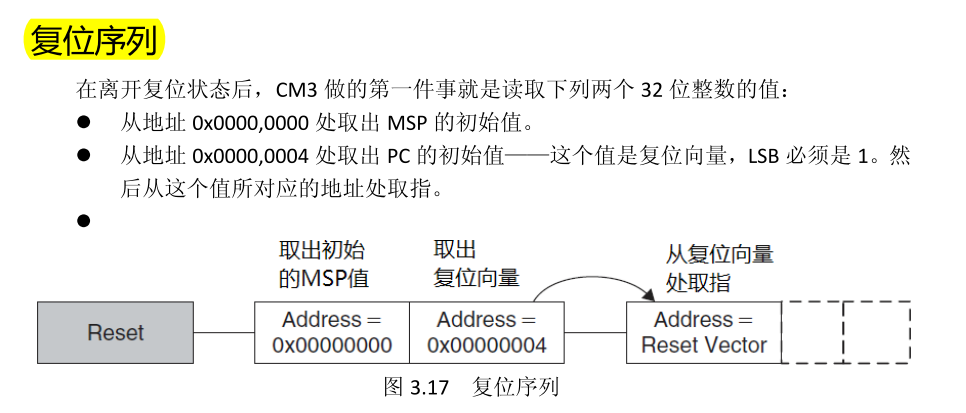
Secure vctor如下所示

testbench.top.compute\_vdd2.cm33\_main.INITSVTOR[31:7] = 0x26000





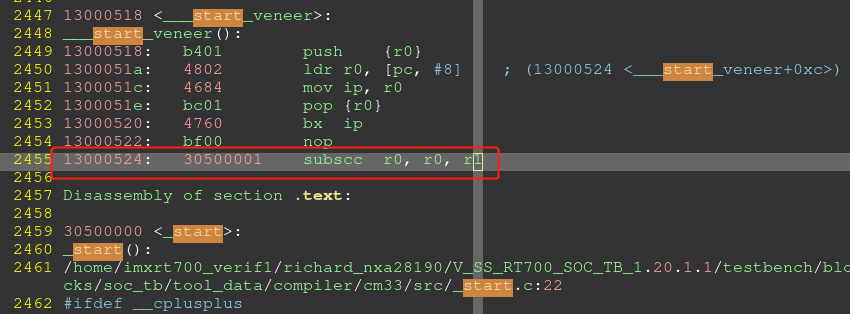


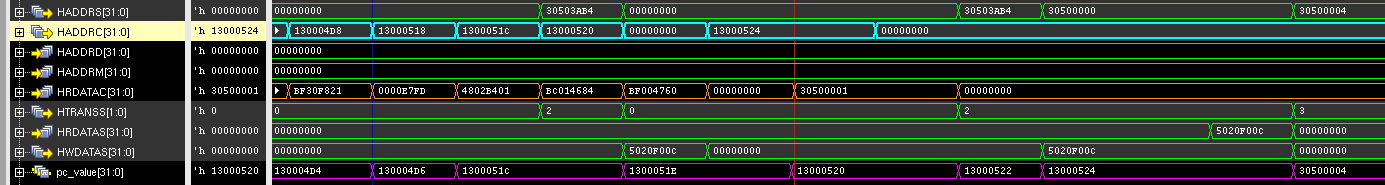


###### 跳转到c

Crt0.s





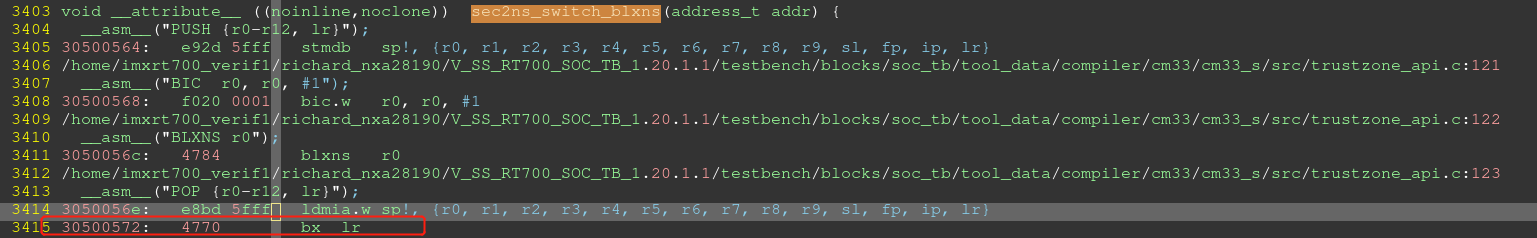


###### 跳转到NON-SECURE DEMO

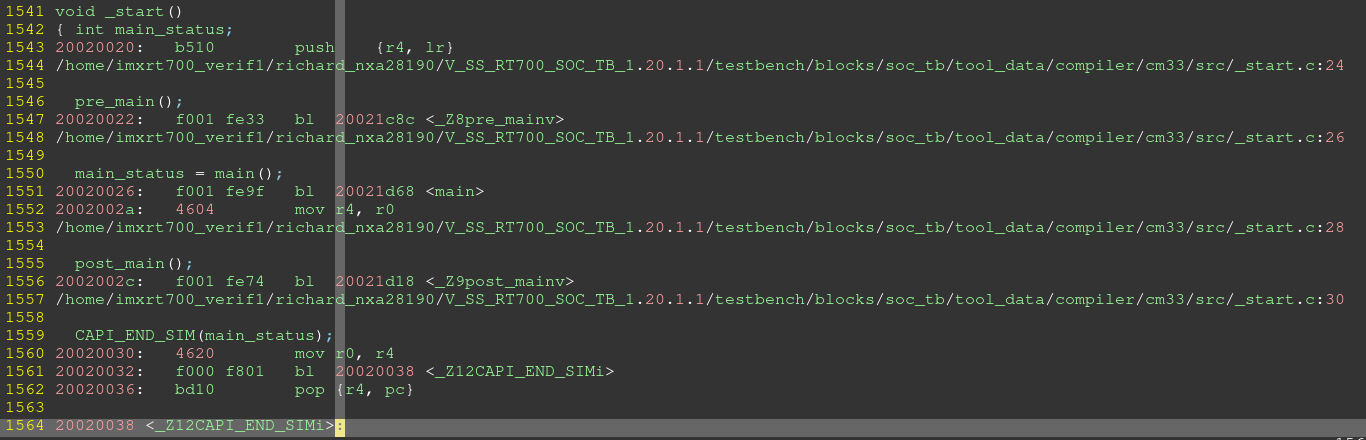
跳转到non-secure demo的\_start.c

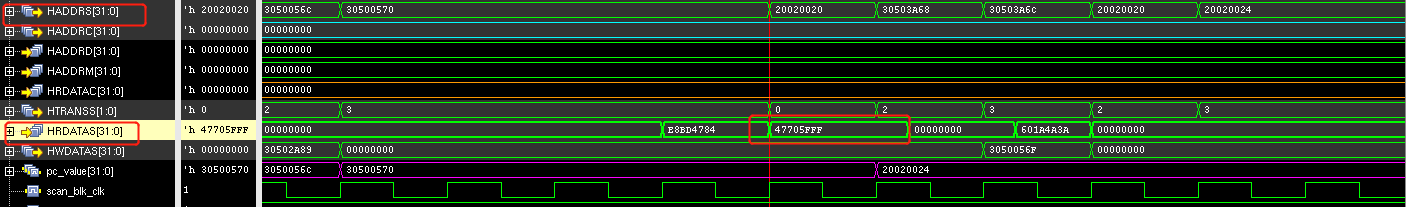
[Boot into non-secure demo](#_Boot_into_non-secure)

Secure demo cdoe:



Non-secure demo code





### CM33\_M boot

CM33\_S : sanity\_cm33\_m\_init.c, cm33\_s/src/crt0.s

CM33\_M\_S: dma3\_reg.c, cm33\_m\_s/src/crt0.s

1. CM33\_S demo: vector/crt0.s-》ROM， code/data-》sram16

Pre\_main中配置CM33\_M的vector

#ifdef CM33\_M\_BOOT\_ENABLE

INFO("pre\_main","config sense cm33 vector....");

#ifdef CM33\_M\_RESET\_VECTOR

W32(REG32(SYSCTL\_SHA\_VDD1\_BASE + 0x98), CM33\_M\_RESET\_VECTOR>>7);//system\_svtor\_i

W32(REG32(SYSCTL\_SHA\_VDD1\_BASE + 0x9C), CM33\_M\_RESET\_VECTOR>>7);//system\_nsvtor\_i

#else

W32(REG32(SYSCTL\_SHA\_VDD1\_BASE + 0x98), S\_SSRAM\_P28\_BASE>>7);//system\_svtor\_i

W32(REG32(SYSCTL\_SHA\_VDD1\_BASE + 0x9C), S\_SSRAM\_P28\_BASE>>7);//system\_nsvtor\_i

#endif CM33\_M\_BOOT\_ENABLE

INFO("pre\_main","release sense cm33 reset....");//CM33\_M demo loading

BCLR32(REG32(RSTCTL\_SHA\_VDD1\_BASE + 0x0010), BIT31);

INFO("pre\_main","release sense cm33 CPUWAIT....");

BCLR32(REG32(SYSCTL\_SHA\_VDD1\_BASE + 0xA0), BIT0);

#endif //CM33\_M\_BOOT\_ENABLE

1. Release CM33\_M reset， testbench load CM33\_M demo, 运行CM33\_M demo

CM33\_M demo, vector/code/data->SRAM28

### CM33\_M\_NS boot

CM33\_M\_NS: dma3\_interrupt.c , cm33\_m\_ns/src/crt0.s, cm33/src/\_start.c

CM33\_S: sanity\_cm33\_m\_init.c, cm33/cm33\_s/src/crt0.s, cm33/src/\_start.c

CM33\_M\_S: sanity\_cm33\_init.c, cm33\_m\_s/src/crt0.s, cm33/src/\_start.c

1. CM33 vector/crt0.s-> ROM, code/data->SRAM16

程序跑起来后，识别到CM33\_M\_BOOT\_ENABLE， 配置CM33\_M的vector， 然后release CM33\_M reset

1. CM33\_M vector/code/data -> SRAM28, CM33\_M\_NS -> SRAM1/SRAM4

CM33\_M demo识别到CM33\_M\_NS\_RESOURCES\_SWITCH， 配置SAU/IDAU non-secure空间，

然后main函数中跳转到CM33\_M\_NS demo。

跳转地址\_start\_\_SYMBOL\_\_， 为CM33\_M\_NS demo的C入口地址

### HIFI4 boot

HIFI使用的编译器是xt-clang

HIFI4: dma3\_reg.c, hifi4/src/vectors.S, /hifi4/src/\_start.c

CM33\_S: cm33\_s/src/crt0.s , compiler/include/sanity\_hifi4\_init.c, cm33/src/\_start.c

1. CM33\_S vector/crt0.s -> ROM, C code/data->SRAM16

Cm33\_s/include/pre\_post\_main.h 检测到 HIFI4\_BOOT\_ENABLE。Release HIFI4 reset.

一旦release HIFI4 reset， testbench 将HIFI4 DEMO load到HIFI4的ITCM,DTCM中。 Demo开始运行

#ifdef HIFI4\_BOOT\_ENABLE

INFO("pre\_main","release hifi4 reset....");//

//W32(REG32(RSTCTL\_COM\_VDD2\_BASE + 0x0024), 0x0000000);

BCLR32(REG32(RSTCTL\_COM\_VDD2\_BASE + 0x0024), BIT0);

#endif //HIFI4\_BOOT\_ENABLE

MEMORY

{

sram0\_seg : org = 0x20040004, len = 0x3FFFC

dram0\_0\_seg : org = 0x24000000, len = 0x10000

iram0\_0\_seg : org = 0x24020000, len = 0x2E0

iram0\_1\_seg : org = 0x240202E0, len = 0x120

iram0\_2\_seg : org = 0x24020400, len = 0x178

iram0\_4\_seg : org = 0x2402057C, len = 0x1C

iram0\_5\_seg : org = 0x24020598, len = 0x4

iram0\_6\_seg : org = 0x2402059C, len = 0x1C

iram0\_7\_seg : org = 0x240205B8, len = 0x4

iram0\_8\_seg : org = 0x240205BC, len = 0x1C

iram0\_10\_seg : org = 0x240205DC, len = 0x1C

iram0\_12\_seg : org = 0x240205FC, len = 0x1C

iram0\_13\_seg : org = 0x24020618, len = 0x4

iram0\_14\_seg : org = 0x2402061C, len = 0x1C

iram0\_16\_seg : org = 0x2402063C, len = 0x1C

iram0\_17\_seg : org = 0x24020658, len = 0xF9A8

string\_0\_seg : org = 0x36000000, len = 0x20000

}

UVM\_INFO @ 1559524.603122 ns --: [testbench.load\_memory\_hifi4] Loading DTCM ......

UVM\_INFO @ 1559524.603122 ns --: [testbench.memory\_map\_load\_gnu\_hex] platform:HIFI4 Loading into mem from address 24000000 to 2400ffff

UVM\_INFO @ 1559524.603122 ns --: [testbench.memory\_map\_load\_gnu\_hex] platform: HIFI4 address=9008000 over range

UVM\_INFO @ 1559524.603122 ns --: [testbench.memory\_map\_load\_gnu\_hex] platform: HIFI4 address >= 24000f94 Load default value into mem...

UVM\_INFO @ 1559524.603122 ns --: [testbench.load\_memory\_hifi4] Loading ITCM ......

UVM\_INFO @ 1559524.603122 ns --: [testbench.memory\_map\_load\_gnu\_hex] platform:HIFI4 Loading into mem from address 24020000 to 2402ffff

UVM\_INFO @ 1559524.603122 ns --: [testbench.memory\_map\_load\_gnu\_hex] platform: HIFI4 address=d800000 over range

UVM\_INFO @ 1559524.603122 ns --: [testbench.memory\_map\_load\_gnu\_hex] platform: HIFI4 address >= 24020628 Load default value into mem...

UVM\_INFO @ 1559524.603122 ns --: [testbench.load\_memory\_hifi4] Loading Shared SRAM6 ......

UVM\_INFO @ 1559524.603122 ns --: [testbench.memory\_map\_load\_gnu\_hex] platform:HIFI4 Loading into mem from address 20040000 to 2005ffff

UVM\_INFO @ 1559524.603122 ns --: [testbench.memory\_map\_load\_gnu\_hex] platform: HIFI4 address=9000000 over range

UVM\_INFO @ 1559524.603122 ns --: [testbench.memory\_map\_load\_gnu\_hex] platform: HIFI4 address >= 20043f1c Load default value into mem...

UVM\_INFO @ 1559524.603122 ns --: [testbench.load\_memory\_hifi4] Loading Shared SRAM7 ......

UVM\_INFO @ 1559524.603122 ns --: [testbench.memory\_map\_load\_gnu\_hex] platform:HIFI4 Loading into mem from address 20060000 to 2007ffff

UVM\_INFO @ 1559524.603122 ns --: [testbench.memory\_map\_load\_gnu\_hex] platform: HIFI4 address=9000000 over range

UVM\_INFO @ 1559524.603122 ns --: [testbench.memory\_map\_load\_gnu\_hex] platform: HIFI4 address >= 0xxxxxxxxX Load default value into mem...

UVM\_INFO @ 1559524.603122 ns --: [testbench.load\_memory\_hifi4] Loading string ......

UVM\_INFO @ 1559524.603122 ns --: [testbench.memory\_map\_load\_gnu\_hex] platform:HIFI4 Loading into mem from address 36000000 to 3607ffff

UVM\_INFO @ 1559524.603122 ns --: [testbench.memory\_map\_load\_gnu\_hex] platform: HIFI4 address >= 36010634 Load default value into mem...

### HIFI1 Boot

HIFI1的vector table是RTL设置好的，为HIFI1 ITCM的start address



HIFI1: hifi/hifi1/src/vectors.S, dma/stimulus/dma3\_soft\_trigger.c

CM33: sanity\_hifi1\_init.c, cm33\_s/src/crt0.s

CM33\_M: sanity\_hifi1\_init.c, cm33\_m\_s/src/crt0.s

1. CM33 vector-> ROM, data/code->SRAM16

上电后，ROM启动，执行CM33的code，识别到CM33\_M\_BOOT\_ENABLE，对CM33\_M进行vector的配置，然后release CM33\_M reset

1. Testbench load CM33 demo， code/data -> SRAM28

CM33\_M 启动后，CM33\_M\_S/include/pre\_post\_main.h识别到HIFI1\_BOOT\_ENABLE，release HIFI1 reset

#ifdef HIFI1\_BOOT\_ENABLE

INFO("pre\_main","release hifi1 reset....");

W32(REG32(RSTCTL\_SEN\_VDD1\_BASE + 0x0010), 0x00000000);

W32(REG32(SYSCTL\_SEN\_VDD1\_BASE + 0x0300), 0x00000000);

#endif //HIFI1\_BOOT\_ENABLE

1. Testbench load HIFI1 demo，code->HIFI1 ITCM ,data->SRAM23

运行HIFI1 demo

### Zenv Boot

ZENV使用的编译器riscv32-unknown-elf-g++

Zenv: dma3\_reg.c zenv/src/zv0335\_crt0.s

CM33: cm33\_s/src/crt0.s sanity\_zenv\_init.c

1. CM33 VECTOR->ROM, code/data->SRAM16,

程序识别到ZENV\_BOOT\_ENABLE， 配置zenv的vector，release zenv的reset

#ifdef ZENV\_BOOT\_ENABLE

INFO("pre\_main","config EZH vector....");

#ifdef ZENV\_RESET\_VECTOR

INFO("pre\_main","config EZH reset vector to ZENV\_RESET\_VECTOR");

W32(REG32(SYSCTL\_MED\_VDD2\_BASE + 0xD4), ZENV\_RESET\_VECTOR); // MTVEC

W32(REG32(SYSCTL\_MED\_VDD2\_BASE + 0xD8), ZENV\_RESET\_VECTOR+0x400); // STVEC

W32(REG32(SYSCTL\_MED\_VDD2\_BASE + 0xD0), (ZENV\_RESET\_VECTOR+0x800)>>2); // RSTBASE

#else

W32(REG32(SYSCTL\_MED\_VDD2\_BASE + 0xD4), 0x24100000); // MTVEC

W32(REG32(SYSCTL\_MED\_VDD2\_BASE + 0xD8), 0x24100400); // STVEC

W32(REG32(SYSCTL\_MED\_VDD2\_BASE + 0xD0), 0x24100800>>2); // RSTBASE

#endif

INFO("pre\_main","release ezhv reset....");

BCLR32(REG32(RSTCTL\_MED\_VDD2\_BASE + 0x0010), BIT5);

#endif //ZENV\_BOOT\_ENABLE

1. Testbench load ZENV demo,运行zenv demo

Vector、code -> ZENV\_ITCM , data->SRAM17

### Dual core boot

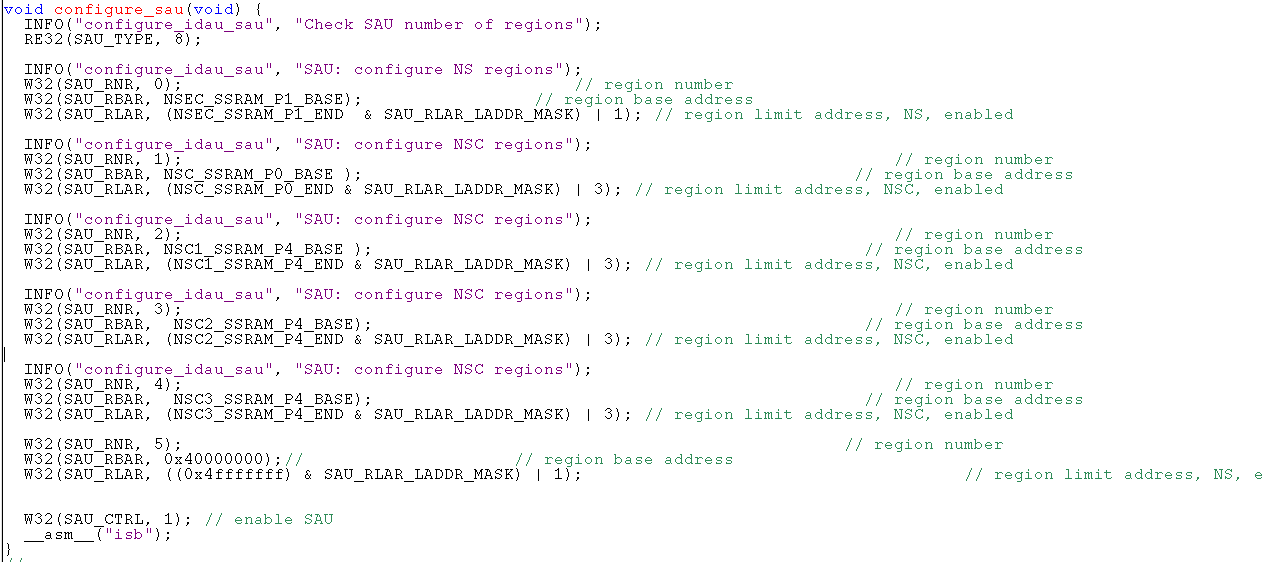
以CM33&CM33\_M为例：

1. VPLAN中填入CM33&CM33\_M，生成的arg文件中就会指定两者的stimulus为相同的main
2. CM33跑的最快，可以在CM33\_M 的main跑起来之前，通过CM33做一些配置，实现CM33\_M对相关功能的访问

如果两个core需要交互控制，可以通过trigge\_get/trigger\_set的方式进行

## 可用的non-secure 地址

CM33\_NS



## 指定code到section中

Gcc \_\_attribute\_\_用法

<https://blog.csdn.net/qlexcel/article/details/92656797>

uint32\_t src\_sysram[4] \_\_attribute\_\_ ((section(".stcm\_last\_test"))) \_\_attribute\_\_ ((aligned (256)));

## DMA setup

DMA context

#define CREATE\_CONTEXT\_DMA3\_INST(num) \

void create\_context\_dma3\_inst##num(DMA3\_CONTEXT\*\* context) \

{ static DMA3\_CONTEXT static\_context; \

static\_context.NAME = "DMA3 INST" CAPI\_STRINGIFY(num); \

static\_context.REGS\_BASE = DMA##num##\_RBASE; \

static\_context.TBCOMM\_BASE = DMA##num##\_TBCOMM\_BASE; \

static\_context.NCH = DMA##num##\_CHANNELS; \

static\_context.BUS\_WIDTH = DMA##num##\_BUS\_WIDTH; \

static\_context.INTERRUPT\_VECTOR = DMA##num##\_CH0\_INT; \

static\_context.TEA\_INT\_VECTOR = HARD\_FAULT\_XCP; \

static\_context.interrupt\_cnt = 0; \

static\_context.tea\_int\_cnt = 0; \

static\_context.err\_int\_cnt = 0; \

static\_context.last\_logged\_error = 0; \

static\_context.APB\_SLOT\_SIZE = DMA##num##\_SLOT\_SIZE; \

static\_context.ENB\_MID\_REPL = DMA##num##\_ENABLE\_MID\_REPL; \

static\_context.ASW = DMA##num##\_ASW; \

static\_context.ENB\_CH\_BUFFWR = DMA##num##\_ENB\_CH\_BUFFWR; \

static\_context.ENB\_MP\_BUFFWR = DMA##num##\_ENB\_MP\_BUFFWR; \

static\_context.AHB\_MASTER\_ID = DMA##num##\_AHB\_MASTER\_ID; \

static\_context.COMBINED\_IRQ = DMA##num##\_COMBINED\_IRQ; \

static\_context.ENB\_MP\_INT = DMA##num##\_ENB\_MP\_INT; \

static\_context.NHR = DMA##num##\_NHR; \

static\_context.DMA\_VERSION = DMA##num##\_VERSION; \

static\_context.DMA\_ENB\_SW\_SECURITY\_CTRL = DMA##num##\_ENB\_SW\_SECURITY\_CTRL; \

static\_context.AHB\_PAL\_DEFAULT = DMA##num##\_AHB\_PAL\_DEFAULT; \

static\_context.AHB\_SEC\_DEFAULT = DMA##num##\_AHB\_SEC\_DEFAULT; \

static\_context.INUSE = 0; \

static\_context.eop\_enable = 0; \

\*context = &static\_context; \

}

dma3\_setup\_hw( dma3\_context,DMA\_CH\_SEL,

(uint32\_t)src, offset: 0x10020 + (a\*1000)

(uint32\_t)addrErr, 0x10030

4, // src Transfer Size 0x10026 bit[10:8]

4, // dst Trassfer Size 0x10026 bit[2:0]

4, // src Buff Offset ( srcAddress Incr ) 0x10024

4, // dst Buff Offset ( dstAddress Incr ) 0x10034

4, // Transfer Byte Count 传输的字节数， 0x10028

1); // Iterations 传输次数，citer=biter biter 0x1003e, citer 0x0x10036

Ssize, dsize – definition:

000b - 8-bit

001b - 16-bit

010b - 32-bit

011b - 64-bit

100b - 16-byte

101b - 32-byte

110b - 64-byte

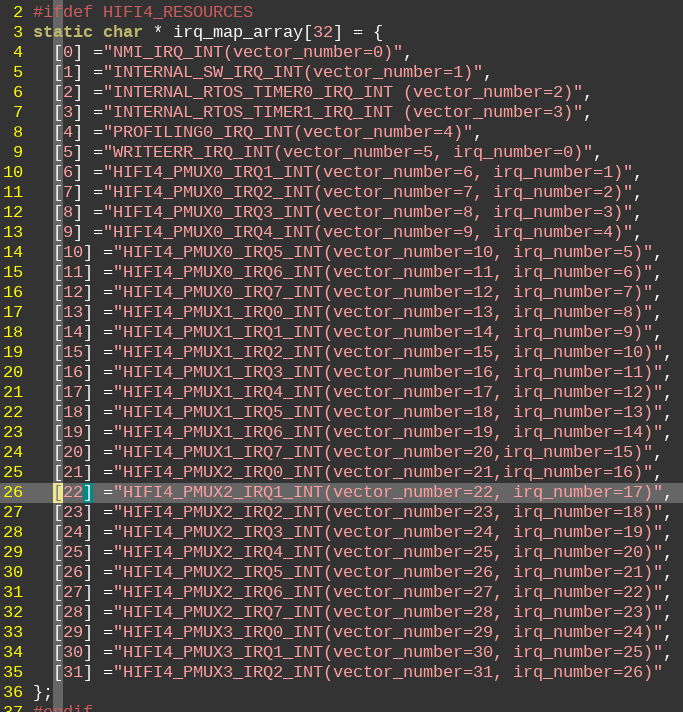
111b - Reserved

# 原理



# HIFI interrupt

* HIFI 一共支持31个中断，26+5.。其中26个外部中断



## 中断mux配置

26个外部中断对应中断号5-31，外部中断通过mux的方式实现，以DMA为例.

DMA0 mux input号为38-53，分别对应DMA0 CH0-CH15.HIFI只认Binterrupt中断信号。

如何让DMA 的mux input对应HIFI的中断信号呢，如下是配置：

1. . 假设我们HIFI使用中断号为22，对应的Binterrupt应该是第17个bit
2. 程序在注册中断限量表的时候 INSTALL\_ISR\_WITH\_CONTEXT(dma3\_context->INTERRUPT\_VECTOR, dma3\_int\_isr, dma3\_context);

第一个参数dma3\_context->INTERRUPT\_VECTOR，应该设置为22.表示我们想用的hifi中断号为22

1. 配置hifi mux寄存器

pmux\_hifi4\_source\_sel(2,38,1);

第二个参数是DMA channel0在mux表格中对应的input号

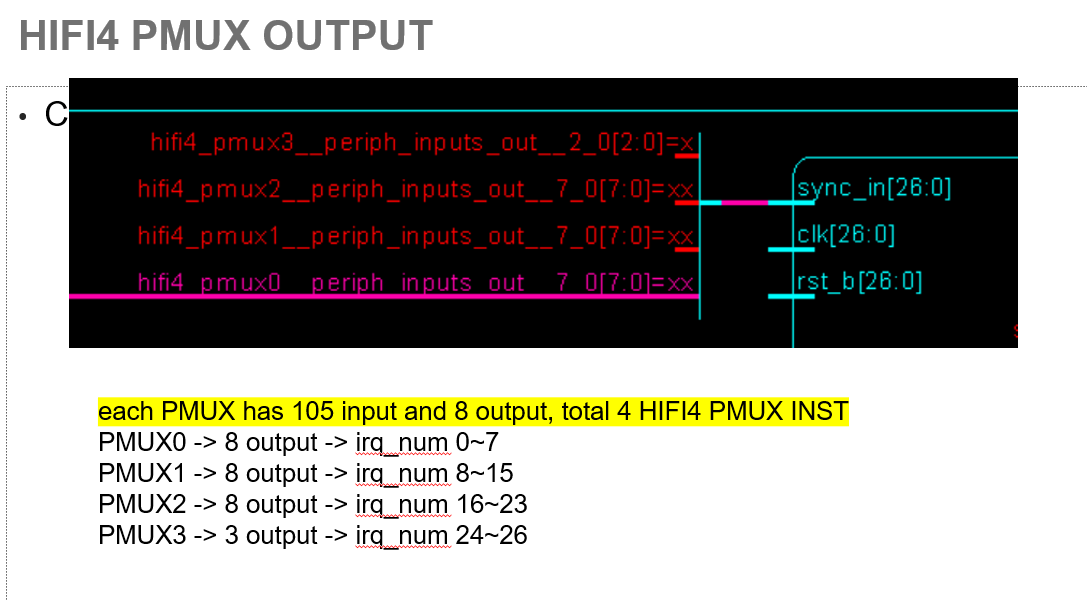
第一个参数2，表示使用第二个mux（一共4个mux，每个8个输出端口）。第三个参数1，表示使用mux的output口为1。。对应就是2\*8+1=17.。

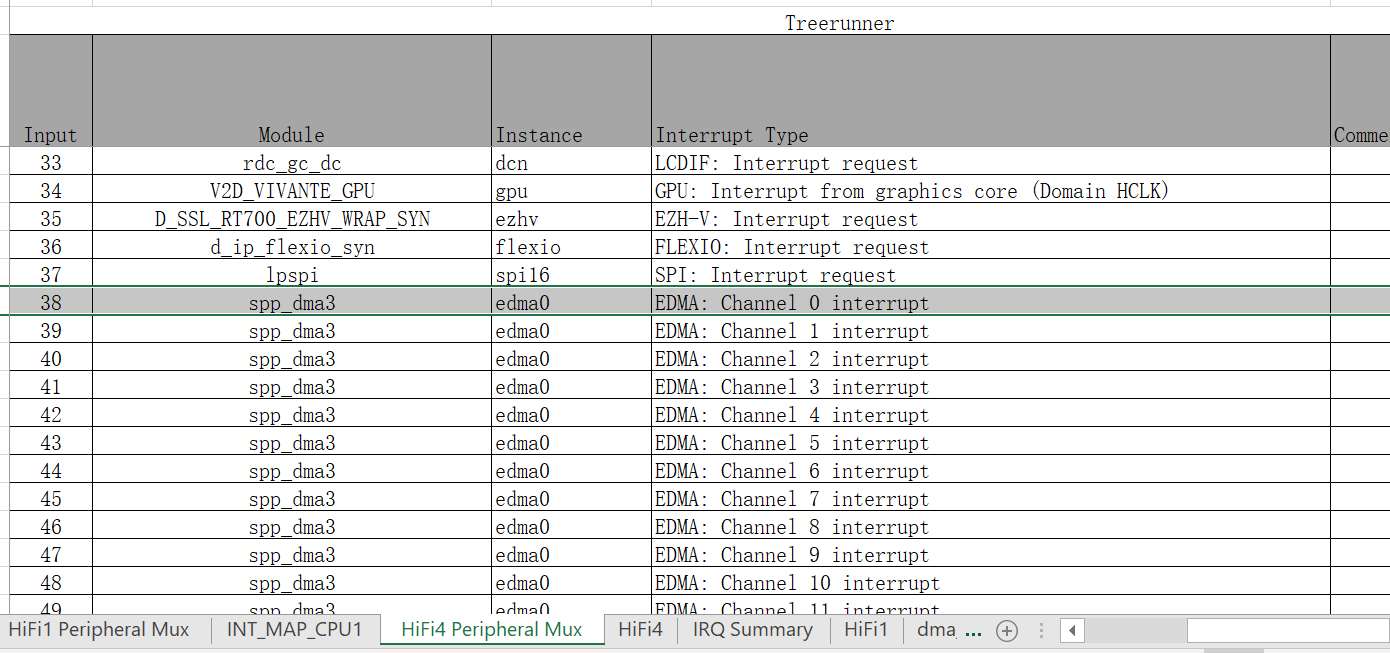
17的意思就是HIFI的Binterrupt使用的是bit17.

1. 中断处理完成后清除中断，否则不停进入中断服务程序

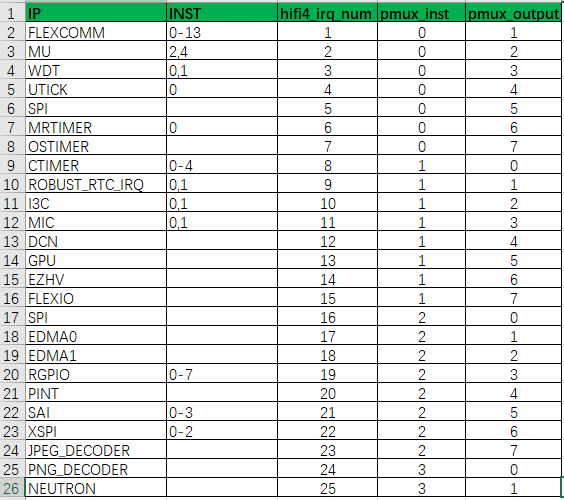
BSET32(DMA3\_CH\_INT(channel\_index),DMA3\_CH\_INT\_INT\_MASK);

1. HIFI的外部中断6-31可以随意给外设配置中断，HIFI在意的只要外部中断的Binterrupt跟注册中断时候的中断号对的上就可以了。中断号=Binterrupt+5





## Testbench 外部中断规定表格

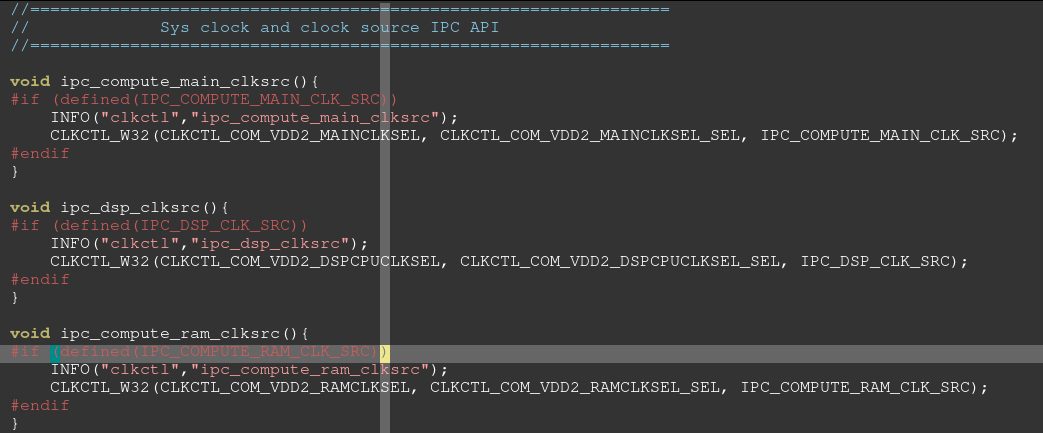


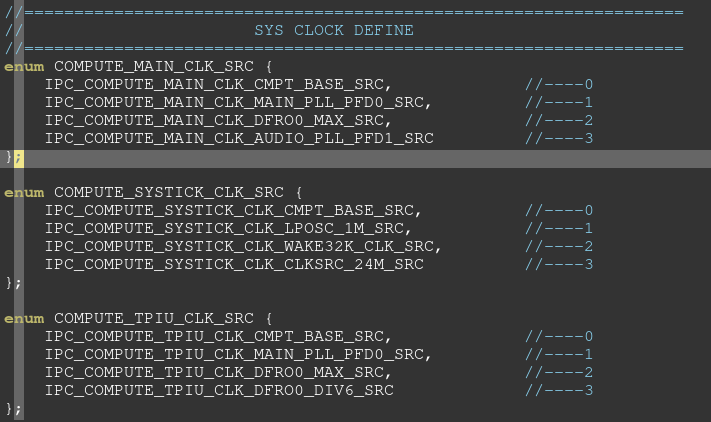
# CLOCK

Testbench/blocks/soc\_tb/tool\_data/compiler/include/ipc\_api.h

Clkctl\_ipc\_def.h

C\_ARG+=IPC\_COMPUTE\_MAIN\_CLK\_SRC=IPC\_COMPUTE\_MAIN\_CLK\_CMPT\_BASE\_SRC



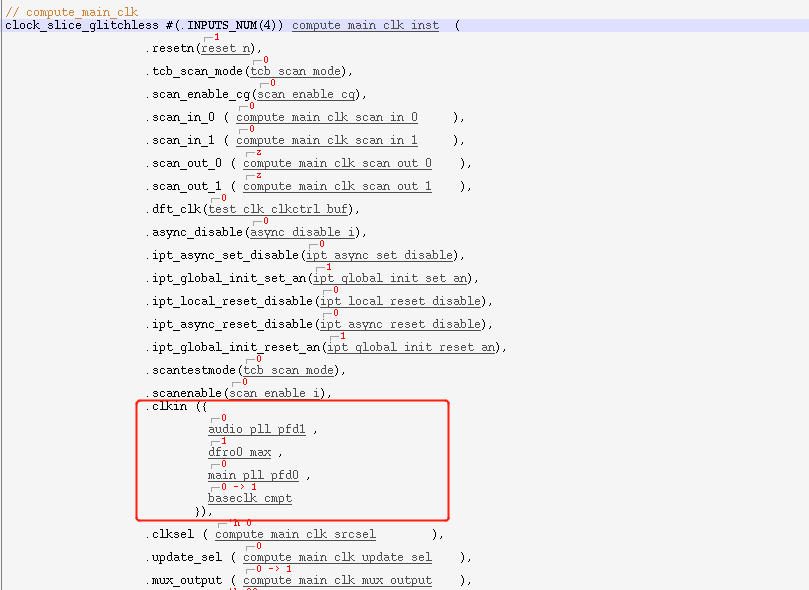


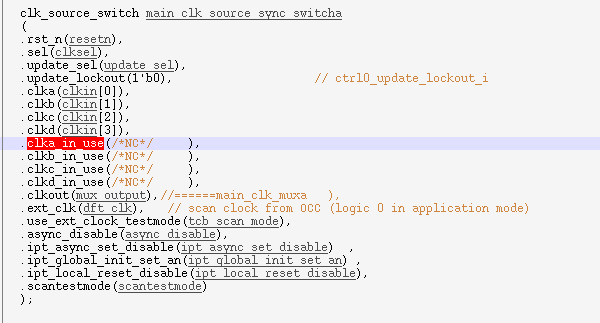
## Clock选信号

DMA0 ipg\_clock：



例化时，四个source clock





# Testbench

## Spec文档

V\_SS\_RT700\_SOC\_TB\_1.11/testbench/blocks/soc\_tb/tool\_data/script/spec

## Force 用法

V\_SS\_RT700\_SOC\_TB\_1.11/testbench/blocks/soc\_tb/testbench/forces\_v

中创建一个richard\_dma\_temp\_force.sv， 内容如下：

/\*initial begin

//disable the unused interrupt for DMA interrupt test

if($test$plusargs("RICHARD\_ENABLE"))

// DMA0,DMA1

//1.10 force testbench.top.compute\_vdd2.compute\_cm33\_irq\_soc\_glue.cpu0\_irq[151:89]= 0;

//1.10 force testbench.top.compute\_vdd2.compute\_cm33\_irq\_soc\_glue.cpu0\_irq[56:0]= 0;

// DMA2,DMA3

//1.10 force testbench.top.sense\_vdd1.sense\_cm33\_irq\_soc\_glue.cpu1\_irq[93:60] = 0;

//1.10 force testbench.top.sense\_vdd1.sense\_cm33\_irq\_soc\_glue.cpu1\_irq[43:0] = 0;

end

\*/

对应case的dma3\_interrupt\_custom.arg 中加上。。

sim\_arg := +RICHARD\_ENABLE $(sim\_arg)

这样只有执行该case的时候，才会调用这个force

## Arg用法

ifeq ($(findstring \_\_cm33\_m,$(TEST\_NAME)),\_\_cm33\_m)

sim\_arg:= +CAPI\_AP=1 + $(sim\_arg)

CM33\_C\_ARG += CAPI\_AP=1

CM33\_C\_ARG += SPIDEN=1

CM33\_C\_ARG += SPNIDEN=1

CM33\_C\_ARG += DBGEN=1

CM33\_C\_ARG += NIDEN=1

else

sim\_arg:= +CAPI\_AP=0 + $(sim\_arg)

CM33\_C\_ARG += CAPI\_AP=0

CM33\_C\_ARG += SPIDEN=1

CM33\_C\_ARG += SPNIDEN=1

CM33\_C\_ARG += DBGEN=1

CM33\_C\_ARG += NIDEN=1

endif

sim\_arg:= +DEBUG\_RUN + $(sim\_arg)

CM33\_C\_ARG += DEBUG\_RUN

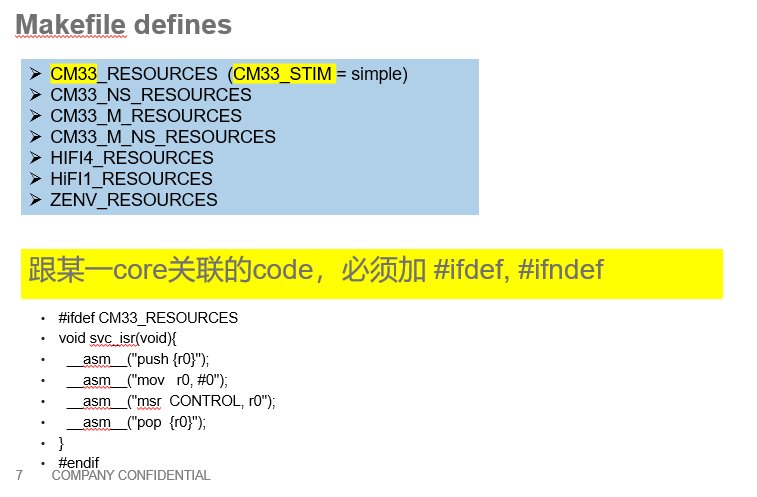
CM33\_M\_C\_ARG += DEBUG\_RUN

## Default tc

用到的VIP文件夹需要写到这里

/home/imxrt700\_verif\_nobackup2/kevin\_nxf88718/V\_SS\_RT700\_SOC\_TB\_1.15.1.1/testbench/blocks/soc\_tb/global/default.tc

## Makefile defines



## V 如何中止testbench

top\_defines\_v/tb\_scope\_define.v:`define REPORTER\_SCOPE `TB\_SCOPE.cm33\_ahb\_reporter.ahb\_reporter.capi\_reporter

module定义：

/home/imxrt700\_verif\_nobackup2/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.33/testbench/blocks/soc\_tb/testbench/modules\_v

实例化：

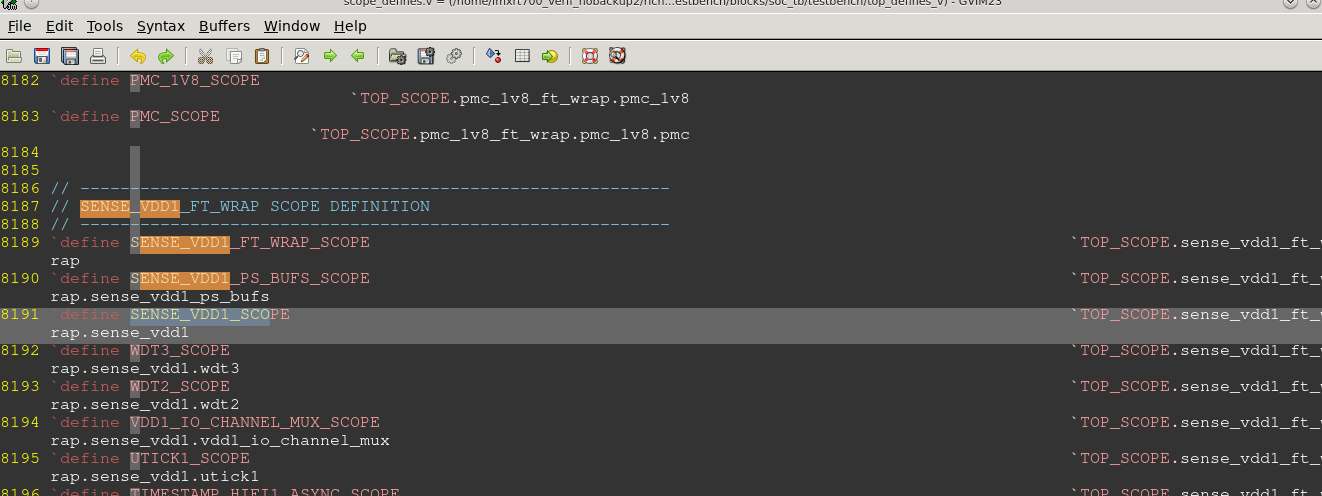
/home/imxrt700\_verif\_nobackup2/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.33/testbench/blocks/soc\_tb/testbench/instances\_v

SV变量的定义：

testbench/blocks/soc\_tb/tool\_data/sray/mem\_map/

## scope 定义

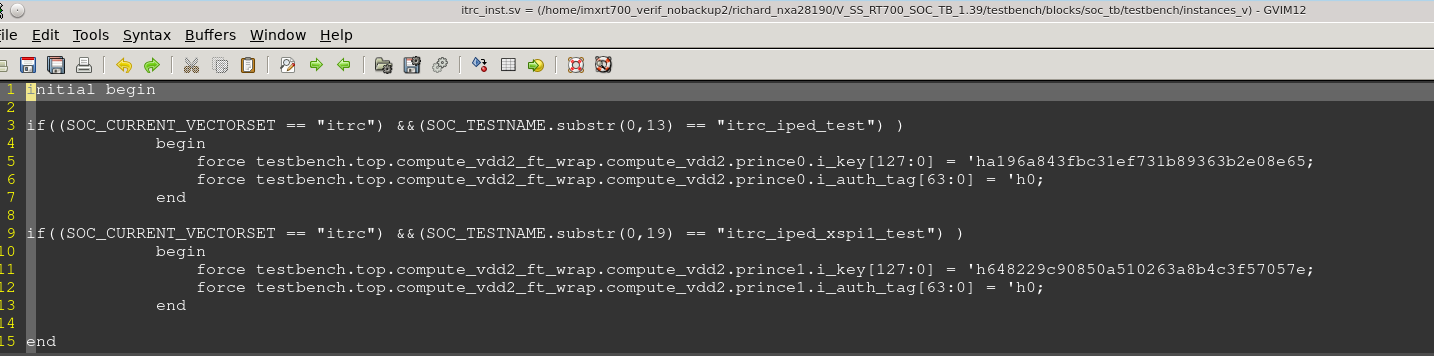
/home/imxrt700\_verif\_nobackup2/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.43/testbench/blocks/soc\_tb/testbench/top\_defines\_v/ scope\_defines.v



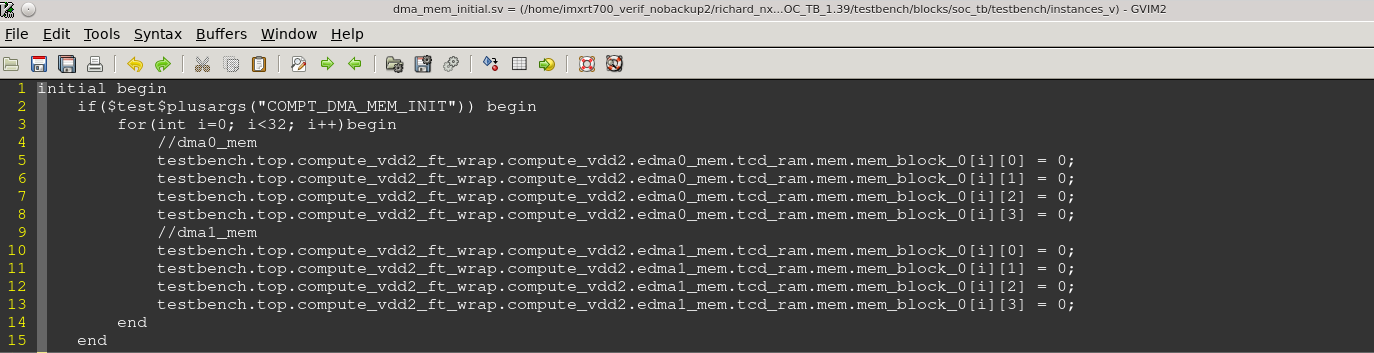
## modules\_v/instances\_v

条件选择用法：

1. 指定vector跟test case



1. Sim\_arg



## testbench task

/home/imxrt700\_verif\_nobackup2/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.36/testbench/blocks/soc\_tb/testbench/tasks\_v

tb\_mem\_loader.sv // load rom fuse，初始化memory

/home/imxrt700\_verif\_nobackup2/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.36/testbench/blocks/soc\_tb/testbench/tasks\_v

tb\_mem\_tasks.sv // 初始化memory的task，memory的reg数组赋值

## load fuse

* testbench/blocks/soc\_tb/testbench/top\_tasks\_v/tb\_mem\_loader.sv

`TB.ocotp\_sram\_init\_data;

`TB.load\_efuse;

### ocotp\_sram\_init\_data

testbench/blocks/soc\_tb/testbench/tasks\_v/tb\_mem\_tasks.sv

task ocotp\_sram\_init\_data; // OCOTP 512 words清零

0: begin `SRAM\_OCOTP.mem\_block\_0 [Row] [Column] = data; end

### load\_efuse

编译hex的时候，指定FUSE的值到数组中，编译到C-V协商好的地址（SRAM7）

V启动时，从hex对应的位置获取到fuse值，然后load到OCOTP中

testbench/blocks/soc\_tb/testbench/tasks\_v/tb\_mem\_tasks.sv

task load\_efuse; memory\_map\_load\_gnu\_hex

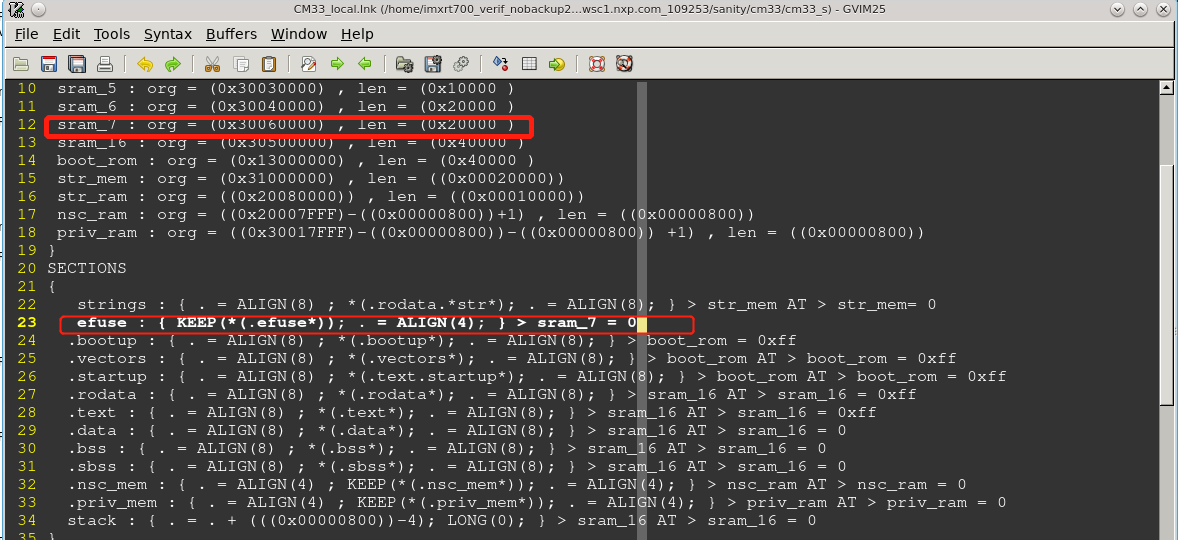
#### V 指定load 地址

V中指定的fuse 存储地址：（C编译hex的时候，也需要讲fuse 数组link到这个地址）

top\_defines\_v/mem\_map.v:`define EFUSE\_BASE `S\_SSRAM\_P7\_BASE

#### C 指定fuse array保存地址

C link文件中指定的fuse array 地址

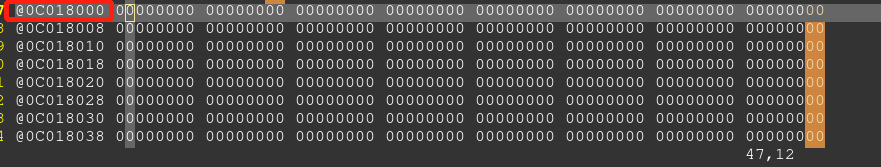


#### Hex中fuse array的地址

C编译生成的hex文件

testbench/blocks/soc\_tb/tool\_data/verilog/verilog\_FILENAMES\_apc\*/ CM33\_simple.rom.hex

0x30060000 >> 2 = 0x0c180000

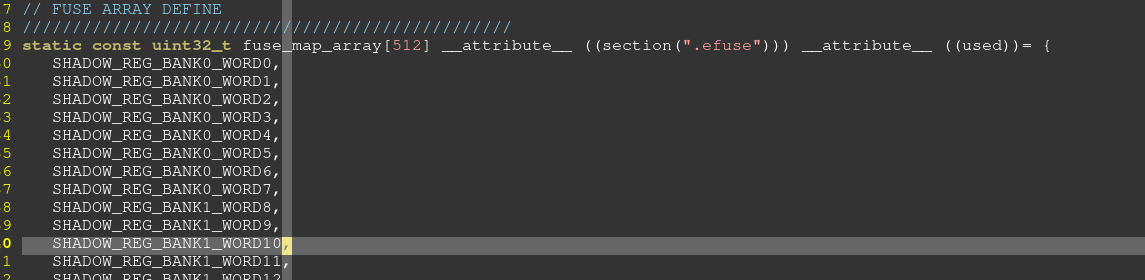


#### Fuse array

512 words

Testbench/blocks/soc\_tb/tool\_data/compiler/include/fuse\_map.h

static const uint32\_t fuse\_map\_array[512] \_\_attribute\_\_ ((section(".efuse"))) \_\_attribute\_\_ ((used))=



#### Fuse macros&generation

Testbench/blocks/soc\_tb/tool\_data/compiler/include/fuse\_map.h

Fuse\_map.h由如下command生成：

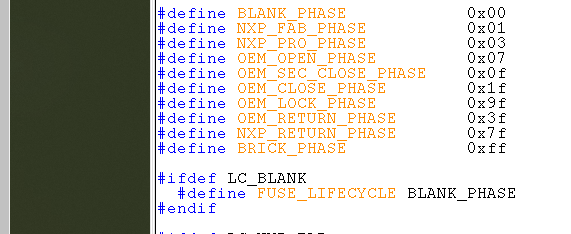
blocks/soc\_tb/tool\_data/script/fuse\_map/fuse\_map\_generator.py

/fuse\_map\_generator.py -src /home/imxrt700\_verif\_nobackup1/hong\_nxf47715/V\_SS\_RT700\_SOC\_TB\_1.34/testbench/blocks/soc\_tb/tool\_data/script/spec/i.MXRT700\_fusemap\_v1.00.xlsx -dst\_file fuse\_map.h -rename\_csv fuse\_rename.csv -sheet\_name OTP\_CONFIG -verbose

#define FUSE\_FIELD(name, low\_bit, high\_bit, reg\_bit) ((((name) >> low\_bit) & (0xffffffff >> (31 - (high\_bit - low\_bit)))) << reg\_bit)

// name的数据，从low bit开始取到high\_bit，得到的数值后左移reg\_bit

// 比如一个数据是0xA，寄存器中对应的reg bit是28。如果配置low\_bit=0， high\_bit=3. 就是讲0xA从bit0取到bit3，也就是A，然后左移到bit28处。就是0xa0000000

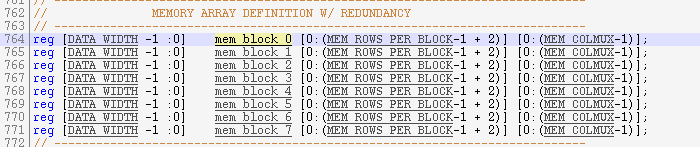


## Signals

### RAM:

`TOP\_SCOPE.common\_vdd2.shared\_mem\_vdd2.sram0\_1.mem

testbench.top.common\_vdd2\_ft\_wrap.common\_vdd2.shared\_mem\_vdd2.sram7\_0.mem



**ROM:**

testbench.top.compute\_vdd2.romcp\_mem.rom0.mem

**Zenv Bus:**

//写数据

//ready信号1为有效，可以进行数据传输

testbench.top.media\_vdd2.nic400\_media\_1.AWREADY\_p\_c\_1

testbench.top.media\_vdd2.nic400\_media\_1.AWADDR\_p\_c\_1[31:0]

testbench.top.media\_vdd2.nic400\_media\_1.WDATA\_p\_c\_1[31:0]

//读

testbench.top.media\_vdd2.nic400\_media\_1.ARADDR\_p\_c\_1[31:0]

testbench.top.media\_vdd2.nic400\_media\_1.RDATA\_p\_c\_1[31:0]

Sense：

testbench.top.sense\_vdd1\_ft\_wrap.sense\_vdd1.sense\_matrix.hready\_m0

testbench.top.sense\_vdd1\_ft\_wrap.sense\_vdd1.sense\_matrix.haddr\_p10[31:0]

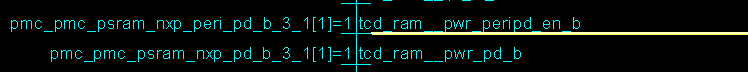
### DMA RAM

如果不能产生中断，确认下DMA的clock，reset是否存在。Power是否在。

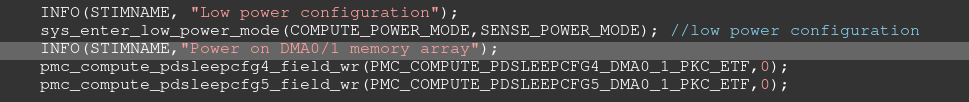
确认下DAM mem的power是否在.

Dma mem 中PMC控制的这两个信号需要是1，为0时，中断没法产生。

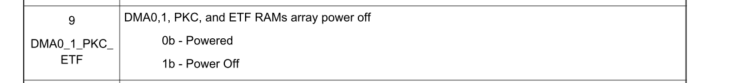
simulator::testbench.top.compute\_vdd2\_ft\_wrap.compute\_vdd2.edma0\_mem.tcd\_ram\_\_pwr\_peripd\_en\_b



寄存器定义







对应的PMC的信号

simulator::testbench.top.pmc\_1v8\_ft\_wrap.pmc\_1v8.pmc.A\_pmc\_dig\_vdd1v8.pmc\_top.pmc\_aggregate.pmc\_pdcfg\_aggreg4[31:0]



### DMA temp wakeup

DMA temp wakeup功能就是，low power下，peripheral想要DMA 传输数据的话。  
通过peripheral的ipd\_async线给sleepcon 发送信号，告知等会要DMA传输了，请你打开DMA的时钟以便传输数据，等待数据传输完毕后，关闭DMA的时钟。

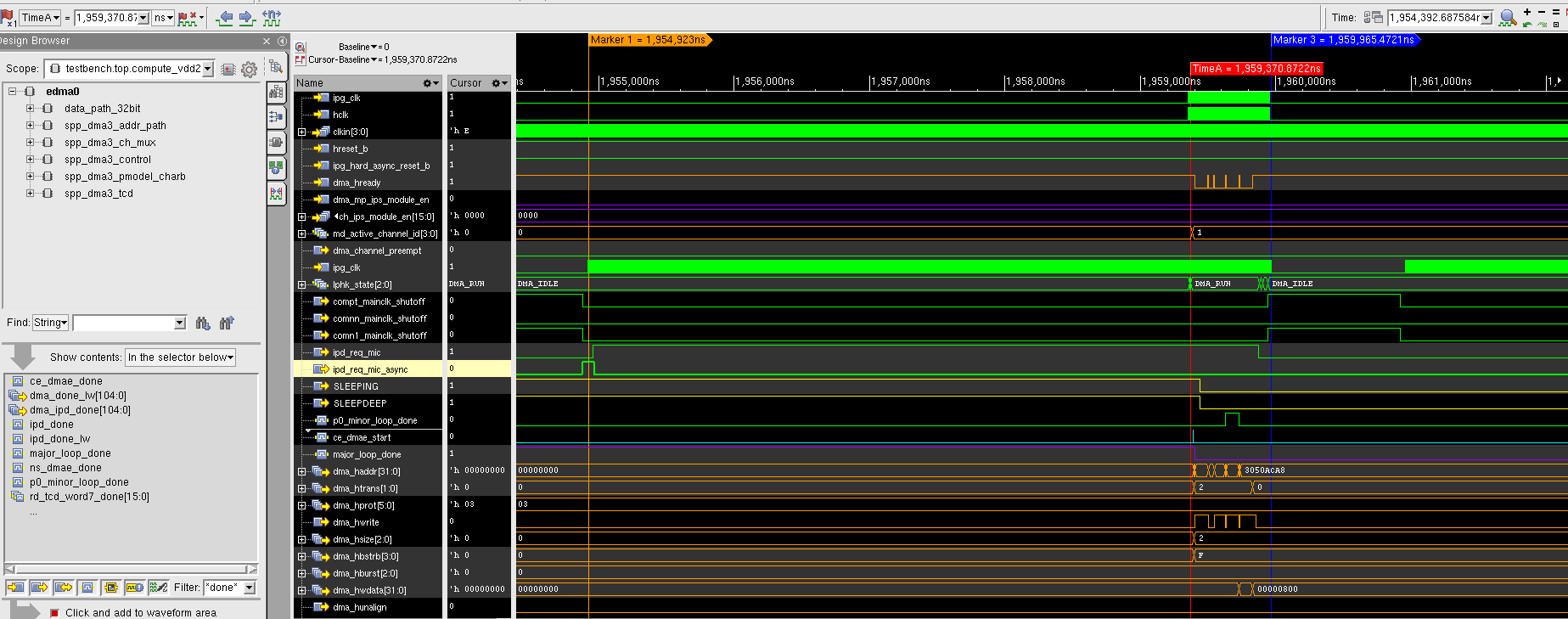
如下是MIC DMA temp wakeup的case。 MIC在进入Low power之前做了配置，不停采集数据，当MIC的FIFO达到一个water mark后，产生一个ipd\_async的信号，

sleepcon得到这个信号后，给DMA跟MIC的时钟打开，

MIC有了时钟后会产生ipd\_req的信号。DMA得到这个信号后，进行数据传输工作。

传输的过程中，sleepcon有个状态机，显示DMA run。（testbench.top.common\_vdd2\_ft\_wrap.common\_vdd2.sleepcon\_cpu0.lphk\_state[2:0]）

停止工作后，状态机变成DMA idle，关闭时钟。



#### Supported peripherals

Flexcomm0-13（45-72）， flexio0-7（37-44）， micfil（1）， sai0-2（81-86），lpspi spi14/spi16（73-76）

### Signals

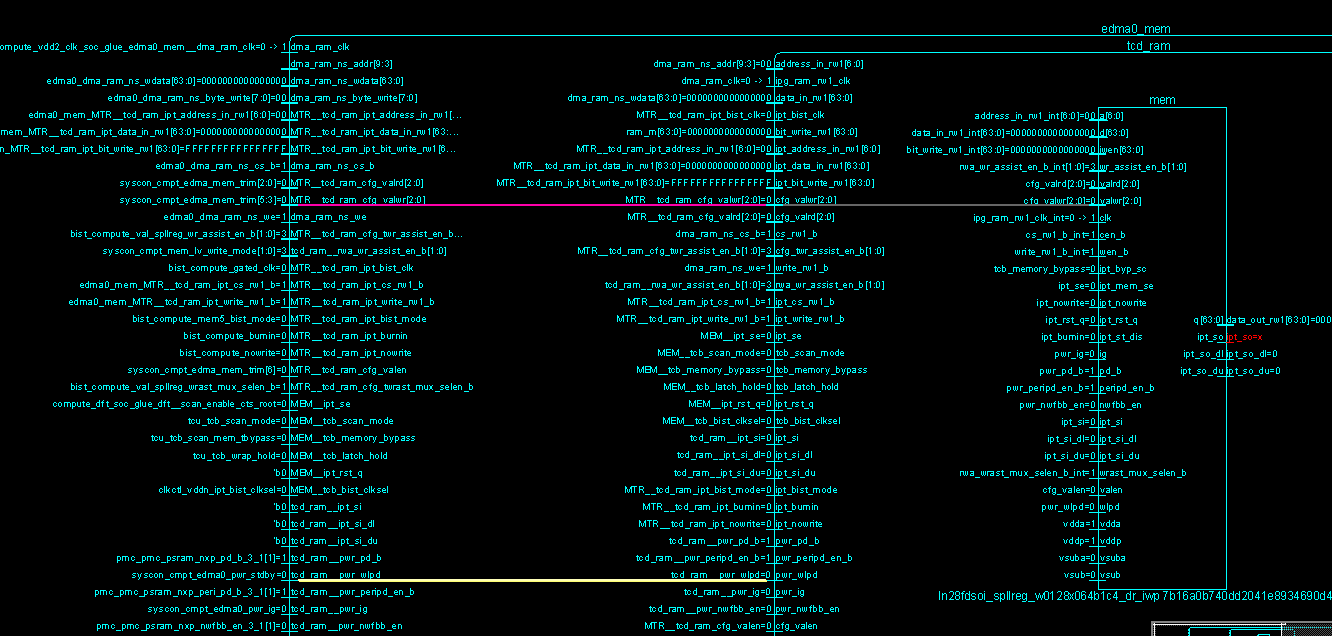
testbench.top.common\_vdd2\_ft\_wrap.common\_vdd2.sleepcon\_cpu0.lphk\_state[2:0]

显示DMA\_RUN,DMA\_IDLE,DMA\_LPHK\_REL,DMA\_LPHK\_REC,

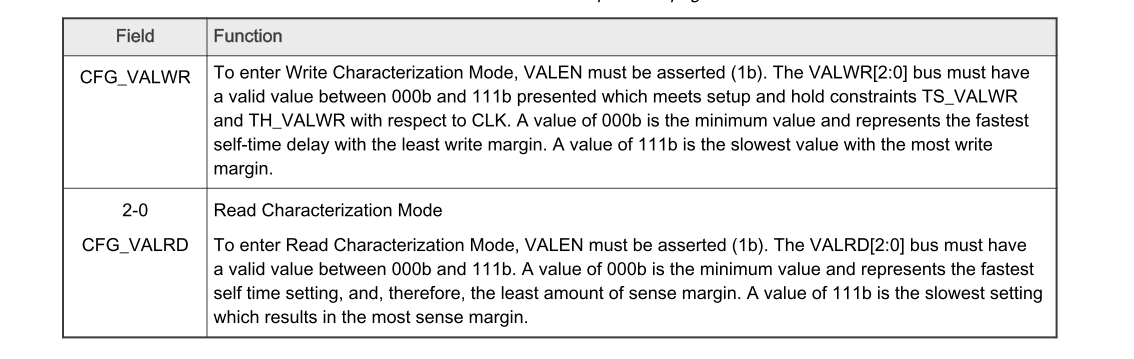
### DMA force signals summary

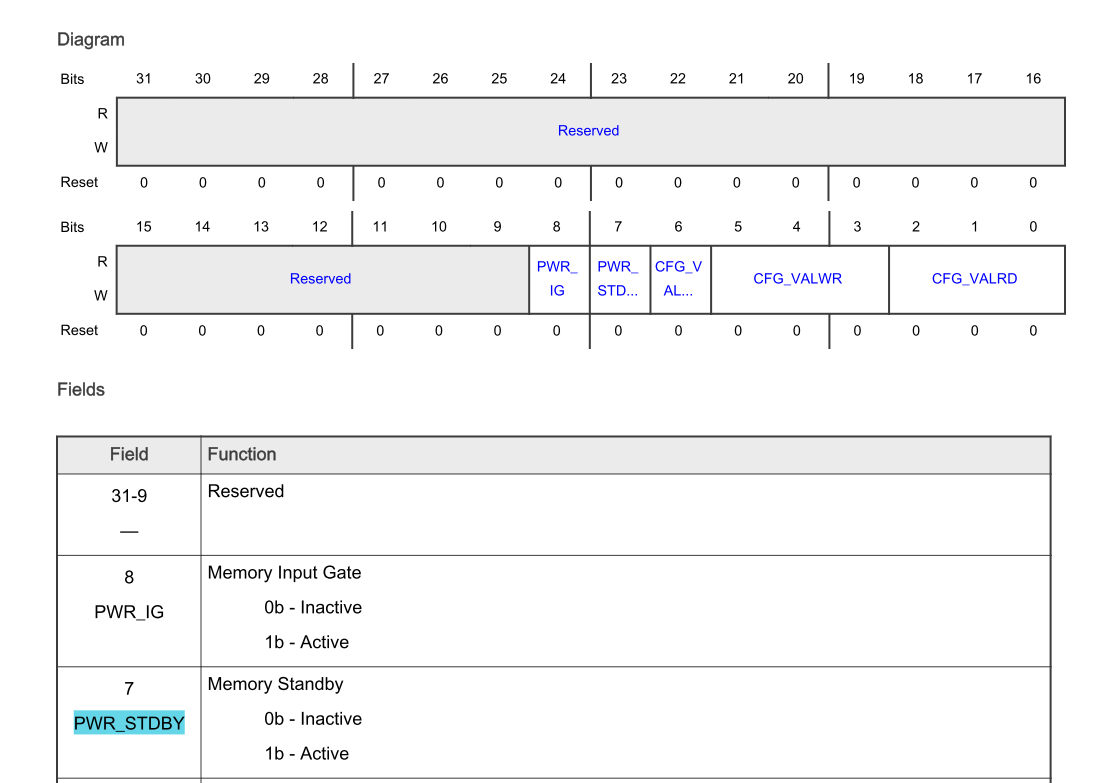
1. testbench.top.compute\_vdd2\_ft\_wrap.compute\_vdd2.edma0\_mem.MTR\_\_tcd\_ram\_cfg\_valwr[2:0] = 3`b0

testbench.top.compute\_vdd2\_ft\_wrap.compute\_vdd2.edma0\_mem.tcd\_ram\_\_pwr\_wlpd =0



(EDMA0\_MEM\_CTRL





1. 中断冲突

DMA0/1 channel 1-16 interrupt doesn't work well. Need to force below signals:  
force testbench.top.compute\_vdd2.compute\_cm33\_irq\_soc\_glue.cpu0\_irq[151:89]= 0;  
force testbench.top.compute\_vdd2.compute\_cm33\_irq\_soc\_glue.cpu0\_irq[56:0]= 0;

1. Memory没有上电

DMA0/1/2/3 access DMA memory failed due to the DMA mem VDDA/VDDP is 0



1. 寄存器地址错误

In compute\_vdd2.v, aips\_lite1.haddr has been changed to pbus\_matrix\_haddr\_p1[25:0], this will cause dma\_mp\_ips\_module\_en always be 0x0.

1. DMA ram clock 跟DMA clock不同步