# 环境文件

## C\_mem\_map.h

generating /home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1.2/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/tool\_data/compiler/include/c\_mem\_map.h

sray mem\_map -nocomdef -generate=h -ifdef\_name \_TB\_RESOURCES\_H\_ -verilog\_input /home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1.2/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/tool\_data/sray/mem\_map/soc\_tb.mem\_map -output\_file /home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1.2/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/tool\_data/compiler/include/c\_mem\_map.h

## cssi\_mem\_map.s

sray mem\_map -nocomdef -assembler gnu -comment\_delim @ -generate=s -ifdef\_name \_TB\_RESOURCES\_ASM\_ -verilog\_input /home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1.2/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/tool\_data/sray/mem\_map/soc\_tb.mem\_map -output\_file /home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1.2/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/tool\_data/compiler/cm85/cm85\_mem\_map.s

# 查找log

Target CPU :

riscv32-unknown-elf-g++ -DCSSI\_RESOURCES=1 -DS400\_RESOURCES=1

riscv32-unknown-elf-ld

Post-Compilation

Target CPU : cm85\_s

# Make

## 流程

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

### Run\_all.sh

Run\_make\_stim.sh

Log：



### 变量定义

#### BUILD\_DIR

/home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/tool\_data/verilog/verilog\_FILENAMES\_20240427\_182057\_apcw401-10-60-41-199.nxdi.kr-awsc1.nxp.com\_124108/sanity/

#### $(1)/$(2)

2 – zenv

1 -- CSSI

#### FILE\_NAMES\_DIR

Run\_make\_results.sh

="/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.25/testbench/blocks/soc\_tb/tool\_data/verilog/verilog\_FILENAMES\_apc7063.cn-sha01.nxp.com\_37453"; export FILE\_NAMES\_DIR

#### soc\_tb\_BLOCK\_DIR

/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.25/testbench/blocks/soc\_tb/tool\_data/verilog/verilog\_FILENAMES\_apc7063.cn-sha01.nxp.com\_37453/makefile.tbc

:= /home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.25/testbench/blocks/soc\_tb

#### v\_sc\_makefile\_BLOCK\_DIR

:=/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.25/testbench/common\_blocks/v\_sc\_makefile

RT2660:

v\_sc\_makefile\_BLOCK\_DIR := /home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1\_2\_backend/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/v\_sc\_xea1\_skeleton\_tb/v\_sc\_makefile

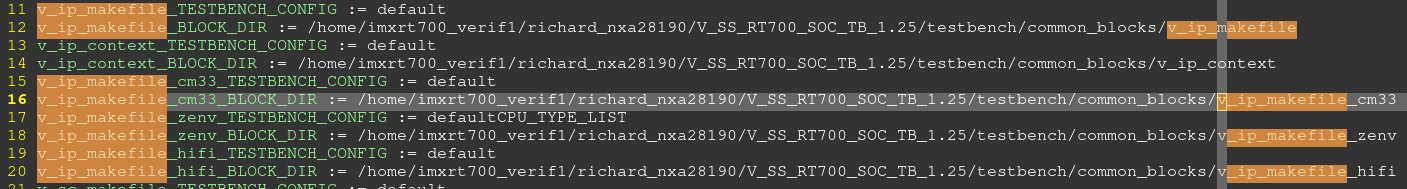
#### v\_ip\_makefile\_BLOCK\_DIR

:=/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.25/testbench/common\_blocks/v\_ip\_makefile

#### v\_ip\_makefile\_$(2)\_BLOCK\_DIR

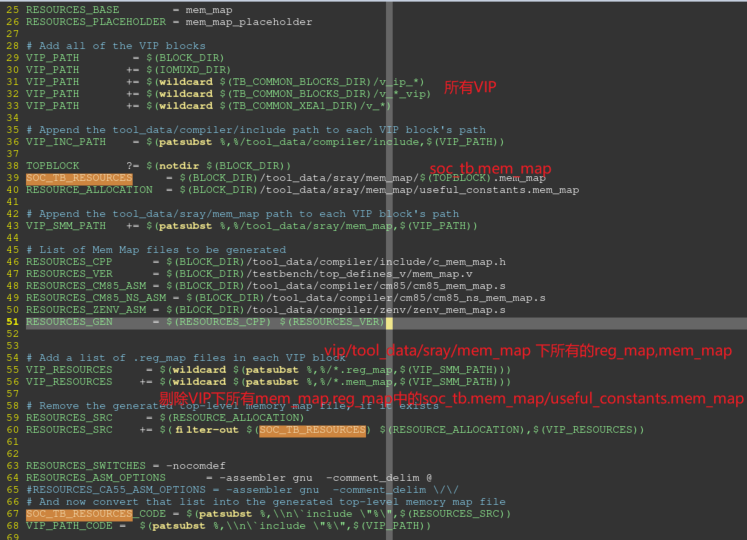
/home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1\_2\_backend/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/common\_blocks/v\_ip\_makefile\_cm85/

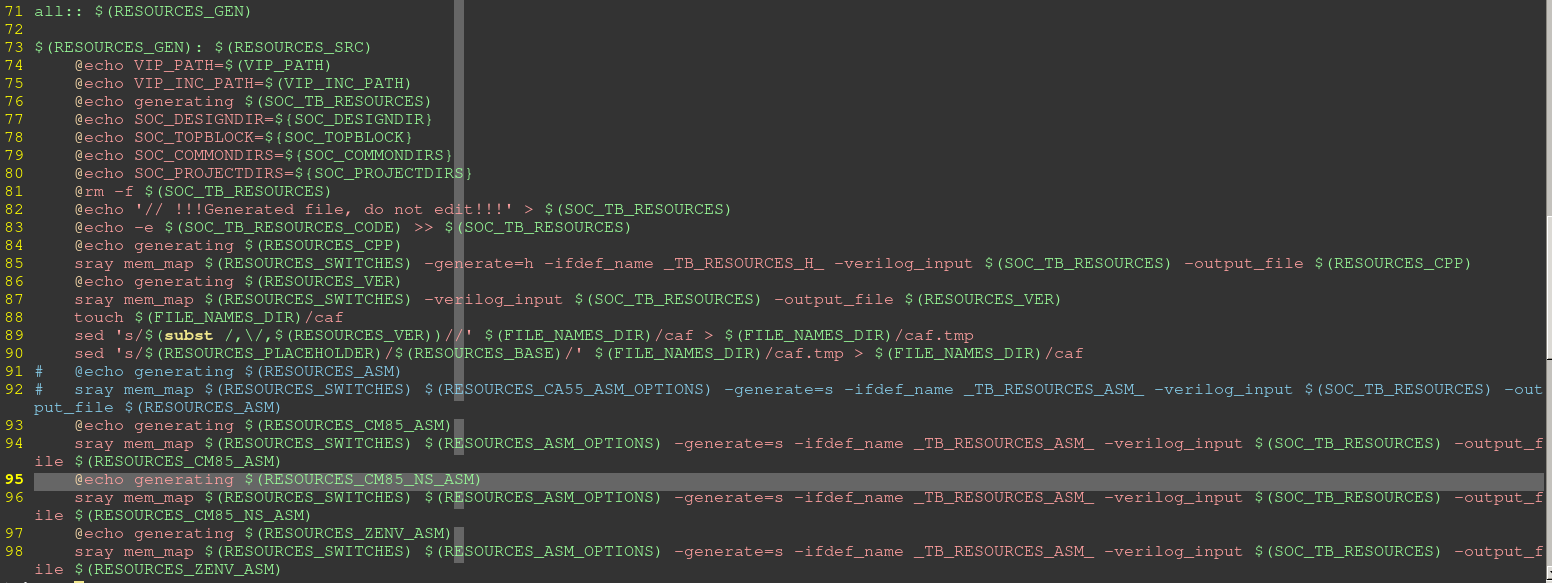
/home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1\_2\_backend/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/v\_sc\_xea1\_skeleton\_tb/v\_ip\_makefile\_zenv/



### soc\_tb/testbench/makefile.stimulus

生成soc\_tb.mem\_map, c\_mem\_map.h, mem\_map.v, cm85\_mem\_map.s等





105 # Skip C-code compile when not simulation phase

106 ifeq ($(SIMULATE),TRUE)

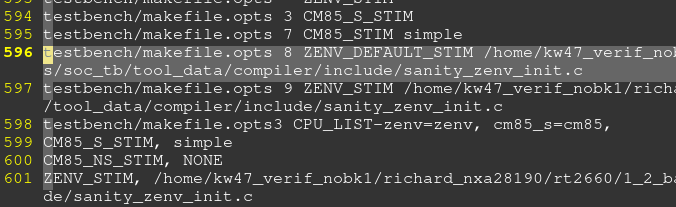
107

108 include $([FILE\_NAMES\_DIR](#_FILE_NAMES_DIR))/makefile.tbc

109 include $([soc\_tb\_BLOCK\_DIR](#_soc_tb_BLOCK_DIR))/testbench/makefile.defs

110 include $(soc\_tb\_BLOCK\_DIR)/testbench/makefile.opts

Log:



111 include $([v\_sc\_makefile\_BLOCK\_DIR](#_v_sc_makefile_BLOCK_DIR))/tool\_data/compiler/makefile.stimulus

#### 生成soc\_tb.mem\_map

从testbench/blocks/soc\_tb/tool\_data/sray/mem\_map， VIP/ tool\_data/sray/mem\_map,

$(IOMUXD\_DIR)/ tool\_data/sray/mem\_map

中抓取.mem\_map, .reg\_map

并在，这些文件绝对地址之前加上`include.

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

如下所有的.reg\_map, mem\_map都将

# Add all of the VIP blocks

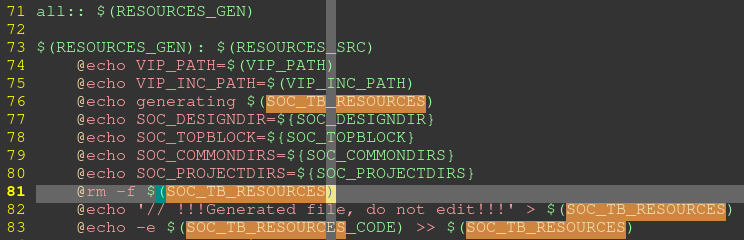
VIP\_PATH = $(BLOCK\_DIR)

VIP\_PATH += $(IOMUXD\_DIR)

VIP\_PATH += $(wildcard $(TB\_COMMON\_BLOCKS\_DIR)/v\_ip\_\*)

VIP\_PATH += $(wildcard $(TB\_COMMON\_BLOCKS\_DIR)/v\_\*\_vip)

VIP\_PATH += $(wildcard $(TB\_COMMON\_XEA1\_DIR)/v\_\*)



SOC\_TB\_RESOURCES\_CODE = $(patsubst %,\\n\`include \"%\",$(RESOURCES\_SRC))

//将/tool\_data/sray/mem\_map/useful\_constants.mem\_map ，tool\_data/sray/mem\_map中/以及 VIP中的mem\_map, reg\_map文件全路径前面加上`include

@echo '// !!!Generated file, do not edit!!!' > $(SOC\_TB\_RESOURCES)

@echo -e $(SOC\_TB\_RESOURCES\_CODE) >> $(SOC\_TB\_RESOURCES)

//生成soc\_tb.mem\_map

Log：

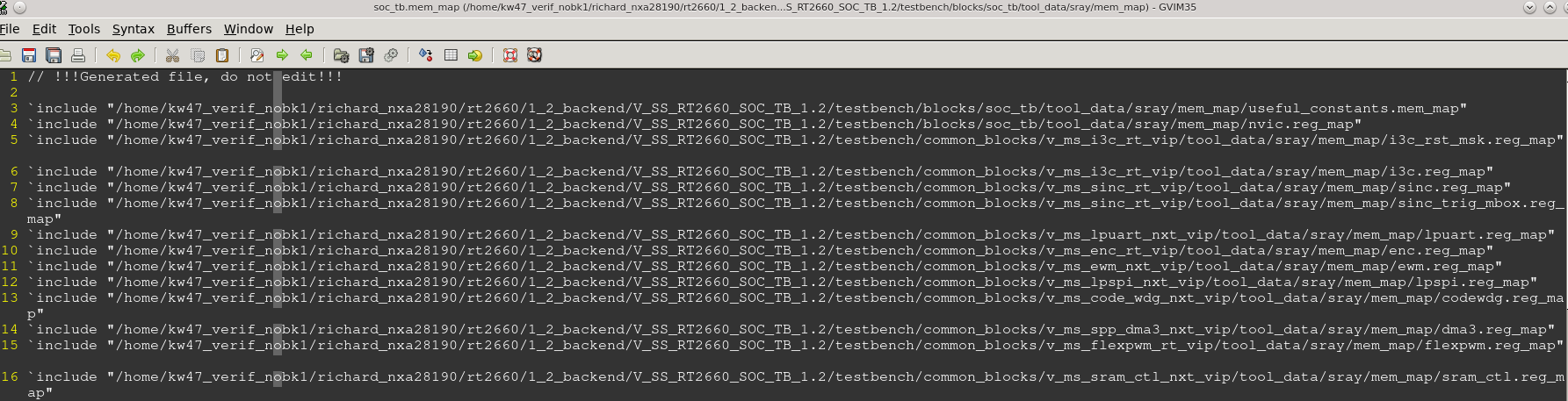
generating /home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1.2/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/tool\_data/sray/mem\_map/soc\_tb.mem\_map

SOC\_DESIGNDIR=/home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1.2/V\_SS\_RT2660\_SOC\_TB\_1.2/doc

SOC\_TOPBLOCK=soc\_tb

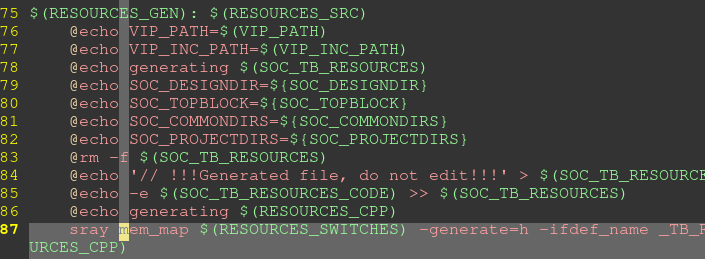
SOC\_COMMONDIRS=

SOC\_PROJECTDIRS=



#### 生成c\_mem\_map.h

RESOURCES\_CPP = $(BLOCK\_DIR)/tool\_data/compiler/include/c\_mem\_map.h



sray mem\_map $(RESOURCES\_SWITCHES) -generate=h -ifdef\_name \_TB\_RESOURCES\_H\_ -verilog\_input $(SOC\_TB\_RESOURCES) -output\_file $(RESOURCES\_CPP)

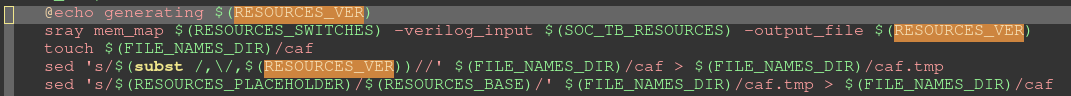
log：

generating /home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1.2/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/tool\_data/compiler/include/c\_mem\_map.h

sray mem\_map -nocomdef -generate=h -ifdef\_name \_TB\_RESOURCES\_H\_ -verilog\_input /home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1.2/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/tool\_data/sray/mem\_map/soc\_tb.mem\_map -output\_file /home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1.2/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/tool\_data/compiler/include/c\_mem\_map.h

#### 生成mem\_map.v

RESOURCES\_VER = $(BLOCK\_DIR)/testbench/top\_defines\_v/mem\_map.v



Log：

generating /home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1.2/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/testbench/top\_defines\_v/mem\_map.v

sray mem\_map -nocomdef -verilog\_input /home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1.2/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/tool\_data/sray/mem\_map/soc\_tb.mem\_map -output\_file /home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1.2/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/testbench/top\_defines\_v/mem\_map.v

#### 生成cm85\_mem\_map.s

生成各个core需要的mem\_map.s文件

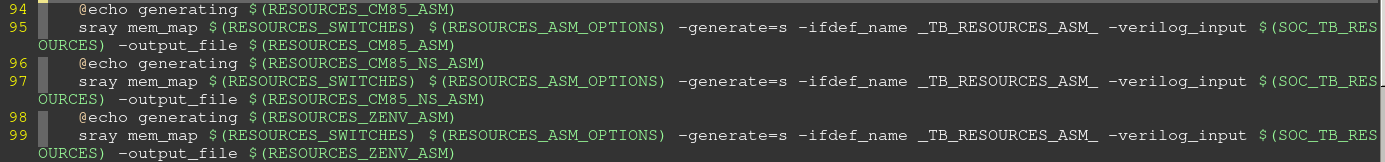
RESOURCES\_CM85\_ASM = $(BLOCK\_DIR)/tool\_data/compiler/cm85/cm85\_mem\_map.s

RESOURCES\_CM85\_NS\_ASM = $(BLOCK\_DIR)/tool\_data/compiler/cm85/cm85\_ns\_mem\_map.s

RESOURCES\_ZENV\_ASM = $(BLOCK\_DIR)/tool\_data/compiler/zenv/zenv\_mem\_map.s

RESOURCES\_CSSI\_ASM = $(BLOCK\_DIR)/tool\_data/compiler/zenv/cssi\_mem\_map.s

RESOURCES\_ZENVN\_ASM = $(BLOCK\_DIR)/tool\_data/compiler/zenv/zenvn\_mem\_map.s



Log:

generating /home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1.2/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/tool\_data/compiler/cm85/cm85\_mem\_map.s

sray mem\_map -nocomdef -assembler gnu -comment\_delim @ -generate=s -ifdef\_name \_TB\_RESOURCES\_ASM\_ -verilog\_input /home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1.2/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/tool\_data/sray/mem\_map/soc\_tb.mem\_map -output\_file /home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1.2/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/tool\_data/compiler/cm85/cm85\_mem\_map.s

### v\_sc\_makefile/tool\_data/compiler/makefile.stimulus

include $(v\_ip\_makefile\_BLOCK\_DIR)/tool\_data/compiler/makefile.stimulus.generic

$(info CPU\_TYPE\_LIST, $(CPU\_TYPE\_LIST))

$(info NO\_STIM\_PRESENT, $(NO\_STIM\_PRESENT))

$(info ACTIVE\_CORES, $(ACTIVE\_CORES))

Log :

CPU\_TYPE\_LIST, ZENV CM85

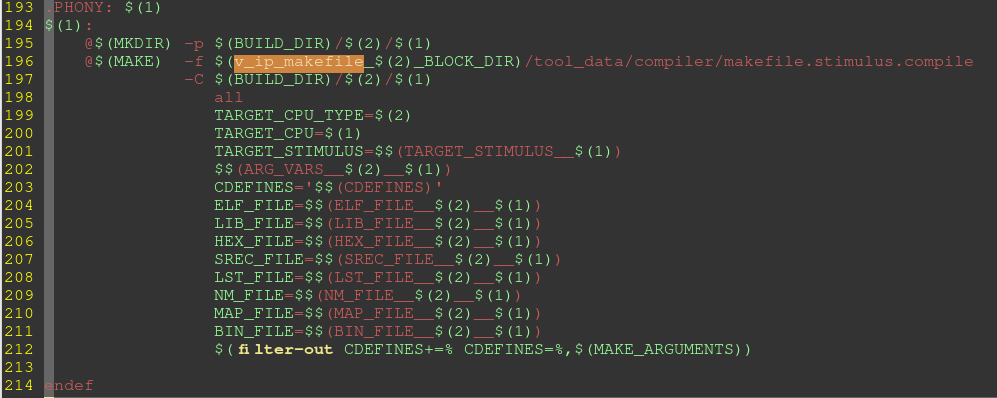
NO\_STIM\_PRESENT, 0

ACTIVE\_CORES, 0x3

TOTAL\_CORES, 2

make: Entering directory `/home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1\_2\_backend/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/vectors/sanity/stimulus'

### v\_ip\_makefile/tool\_data/compiler/makefile.stimulus.generic



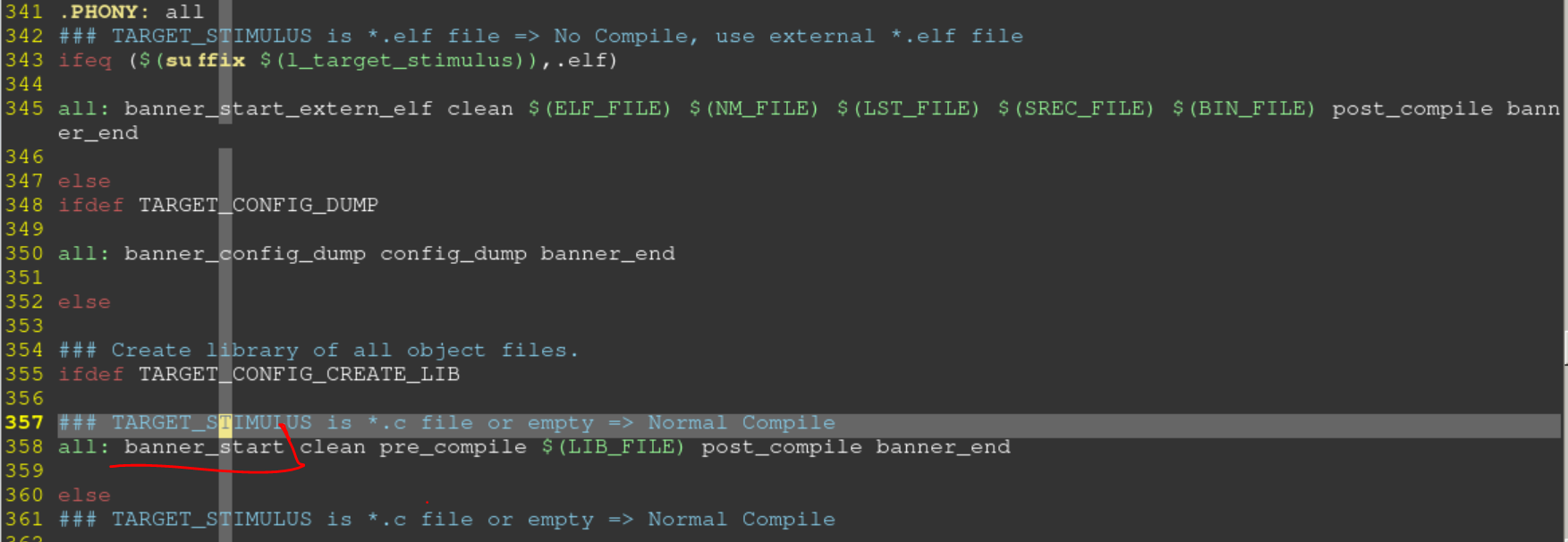
### v\_ip\_makefile\_$(2)\_BLOCK\_DIR/tool\_data/compiler/makefile.stimulus.compile



这里会根据$2选则响应的makefile VIP。 有CM33,HIFI,ZENV

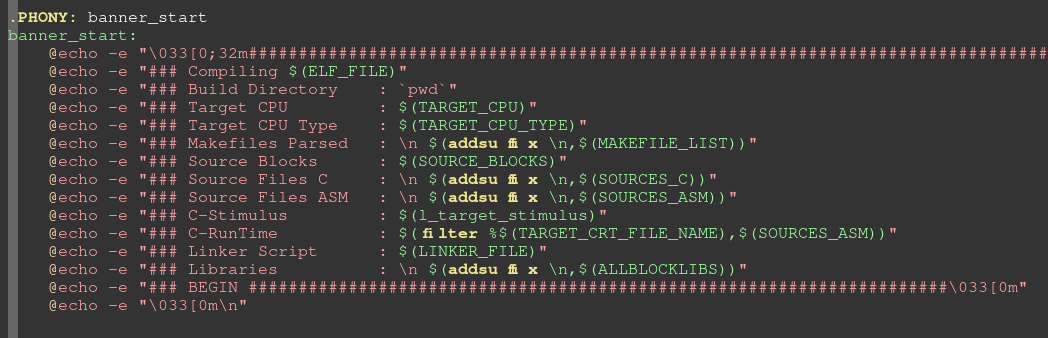
/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.25/testbench/common\_blocks/v\_ip\_makefile\_cm33/tool\_data/compiler/makefile.stimulus.compile

### v\_ip\_makefile/tool\_data/compiler/makefile.compile

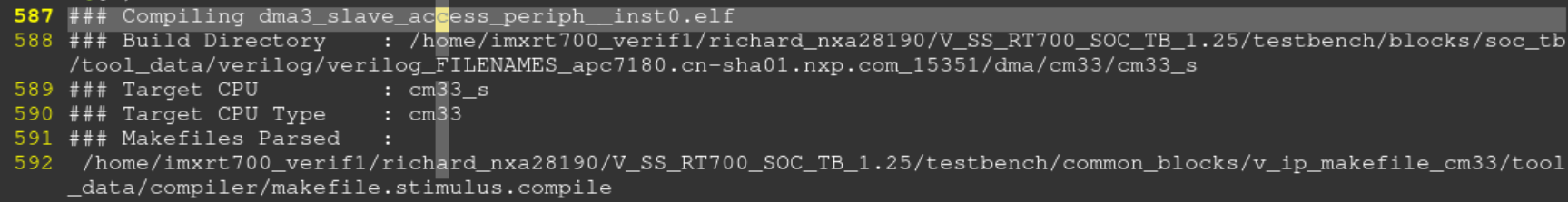


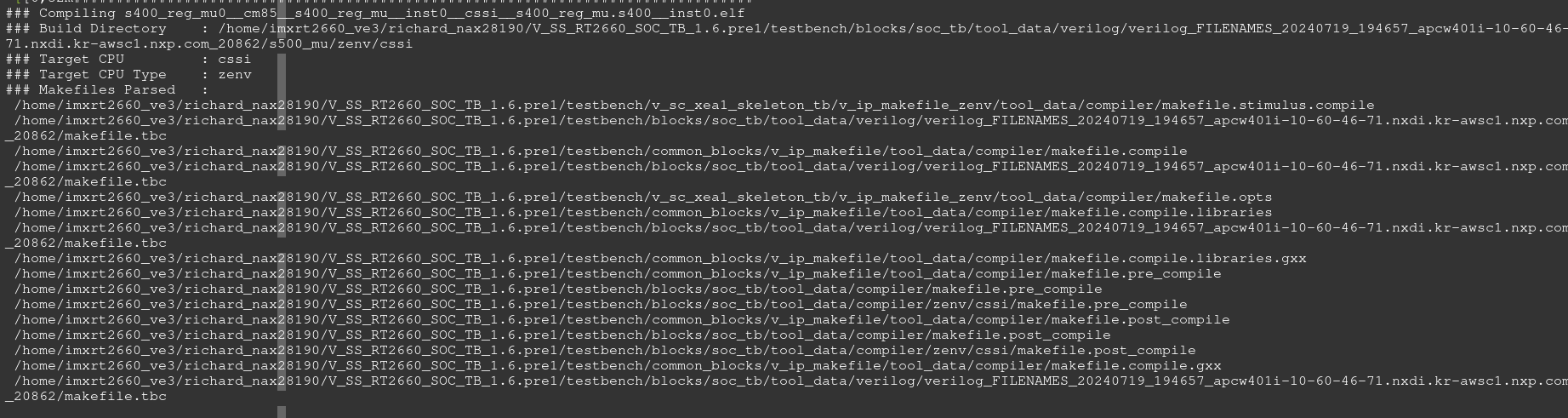
Log:

Compiling



Log:





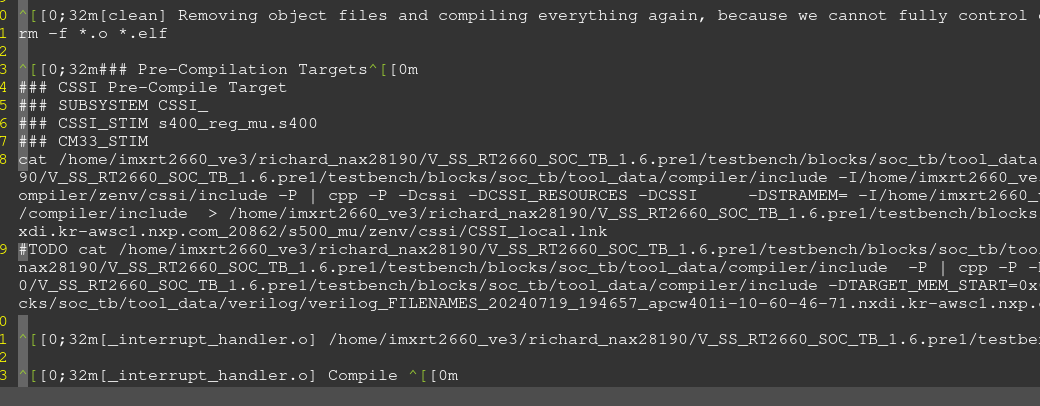
#### testbench/blocks/soc\_tb/tool\_data/compiler/zenv/cssi/makefile.pre\_compile

generate linkfile ..



Log:

Pre-Comp



Make Code Log：

Target CPU

riscv32-unknown-elf-ld

arm-none-eabi-ld

#### testbench/blocks/soc\_tb/tool\_data/compiler/zenv/cssi/makefile.post\_compile

Generate hex file and run simulation

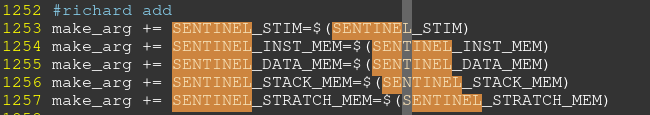


## 如何增加一个core的编译支持

### Block.Arg

/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.25/testbench/blocks/soc\_tb/arg/block.arg

添加SENTINEL\_STIM， 对应使用在vector下面的arg中



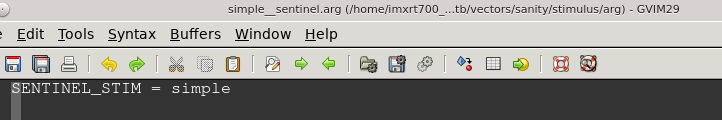
### test arg

test command:

bsub -R rhel7 -q interq -Ip soc verilog -irun -block soc\_tb -bc rtl -collect\_arg -tc default -vectors sanity -test simple\_\_sentinel -shm -keeptemps -session test\_sentinel\_compile &

/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.25/testbench/blocks/soc\_tb/vectors/sanity/stimulus/arg/simple\_\_sentinel.arg

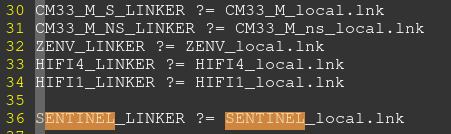
SENTINEL\_STIM 配置simple.c

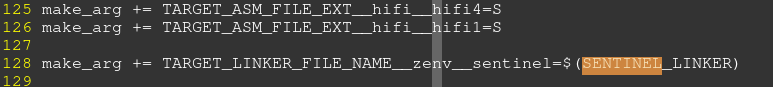


### Capi.arg

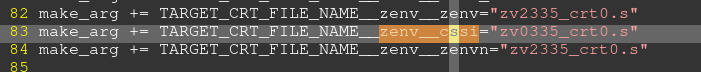
设置linkfile， 注意这里的命名规则 TARGET\_LINKER\_FILE\_NAME\_\_zenv\_\_sentinel

/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.25/testbench/blocks/soc\_tb/arg/capi.arg



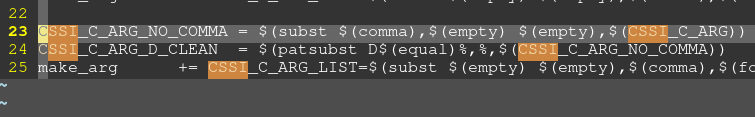


设置汇编



### imxrt\_C\_ARG.arg

/home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1.2/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/arg



### Makefile.opts

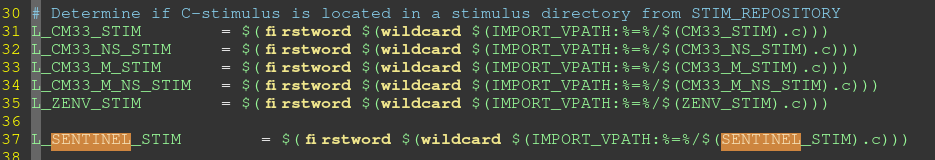
增加Core的编译支持：

/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.25/testbench/blocks/soc\_tb/testbench/makefile.opts

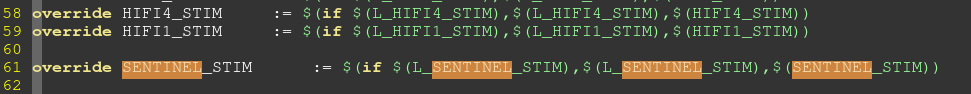
nn

显示当前用到的[SENTINEL\_STIM](#_test_arg)

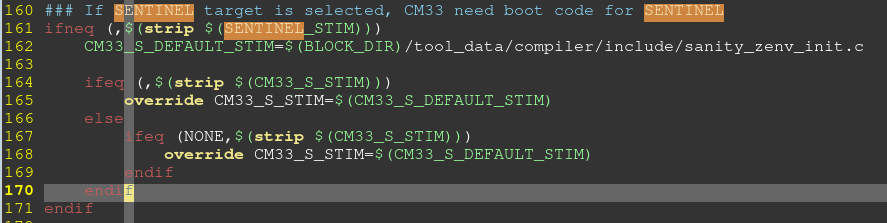




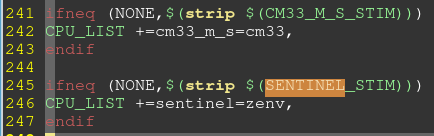




增加多core的编译



增加编译器



### Makefile.stimulus

生成mem\_map.s

/home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1.2/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/testbench/makefile.stimulus





### 新增core支持文件夹下

/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.25/testbench/blocks/soc\_tb/tool\_data/compiler/zenv/sentinel

该文件夹为自建文件夹，包含了编译所需的汇编文件，头文件，makefile，linkfile等等



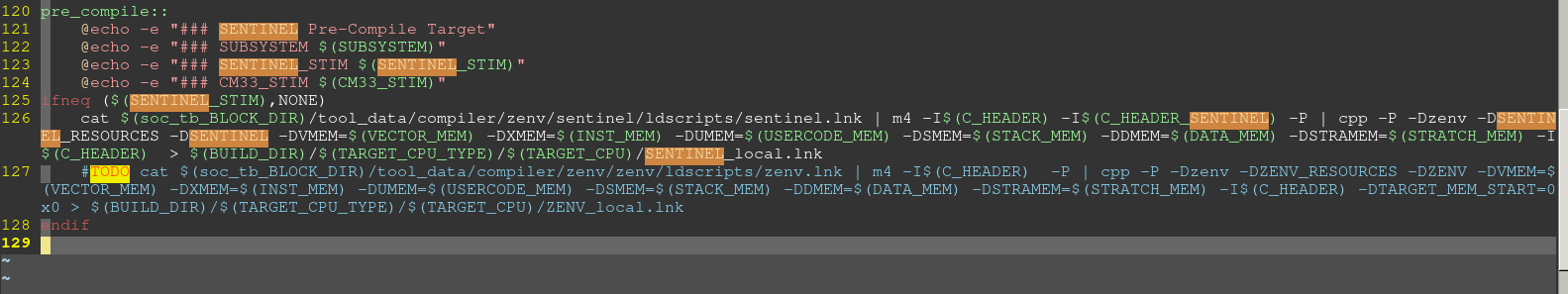
#### Lnk文件

/home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.25/testbench/blocks/soc\_tb/tool\_data/compiler/zenv/sentinel/ldscripts

#### makefile.pre\_compile

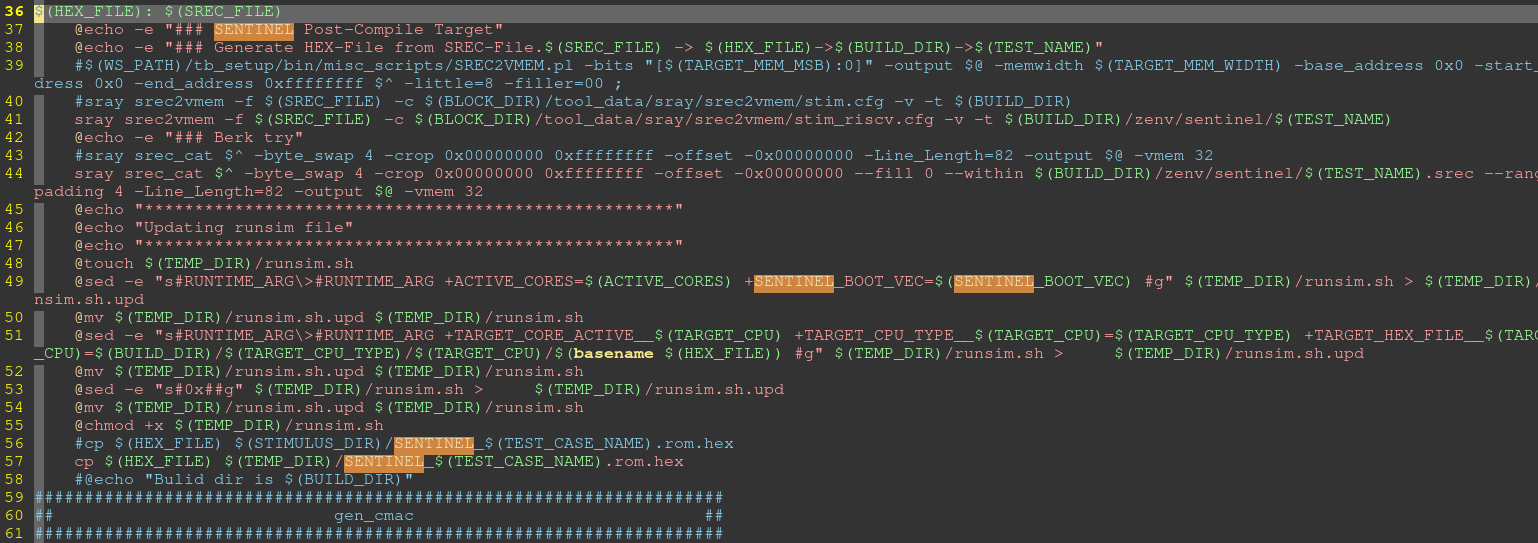
主要用来结合[lnk文件](#_Lnk文件)生成linkfile。

[Capi.arg](#_Capi.arg)中定义了linkfile



#### makefile.post\_compile

编译完成之后的一些动作，比如生成hex文件



## Makefile详解

<https://zhuanlan.zhihu.com/p/575852387>

## Make 函数

### filter-out

去除掉不需要的pattern



### Wildcard

显示所有的pattern文件



### Patsubst

如果pattern为空，直接显示replacement



例：

path = /home/kw47\_verif\_nobk1/richard\_nxa28190/rt2660/1\_2\_backend/V\_SS\_RT2660\_SOC\_TB\_1.2/testbench/blocks/soc\_tb/tool\_data/sray/mem\_map

obj = $(patsubst %, %/\*.mem\_map, $(path)) //返回path/\*mem\_map

content = $(wildcard $(patsubst %, %/\*.mem\_map, $(path))) // 显示所有后缀为mem\_map的文件

all:

@echo "obj = $(obj)"

@echo "content = $(content)"

Log：



## Make 命令

### -f

-f=<file>  
--file=<file>  
--makefile=<file>  
指定需要执行的makefile

### -I

-I <dir>  
--include-dir=<dir>  
指定一个被包含makefile的搜索目标。可以使用多个"-I"参数来指定多个目标

### -C

-C <dir>  
--directory=<dir>  
指定读取makefile的目录。如果有多个-C参数，make的解释是后面的路径以前面的作为相对路径，并以最后的目录作为被指定目录。如："make -C ~/test -C prog"等价于"make -C ~/test/prog"

## 常用文件

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

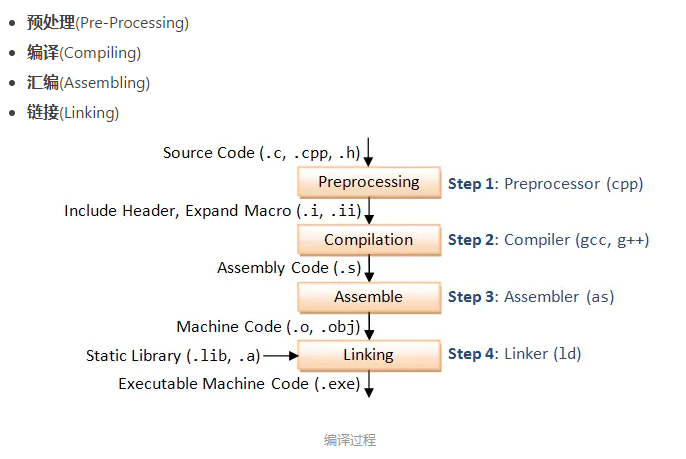
# GCC

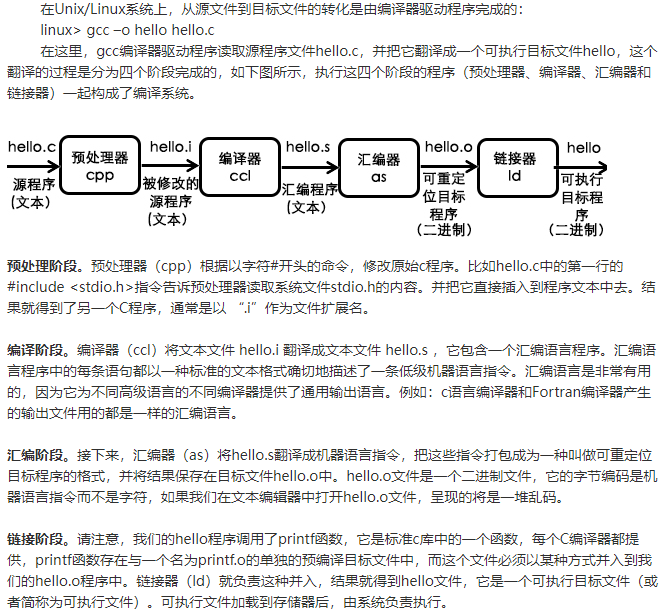
<https://www.jianshu.com/p/1bab86143f1c>

## 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/>

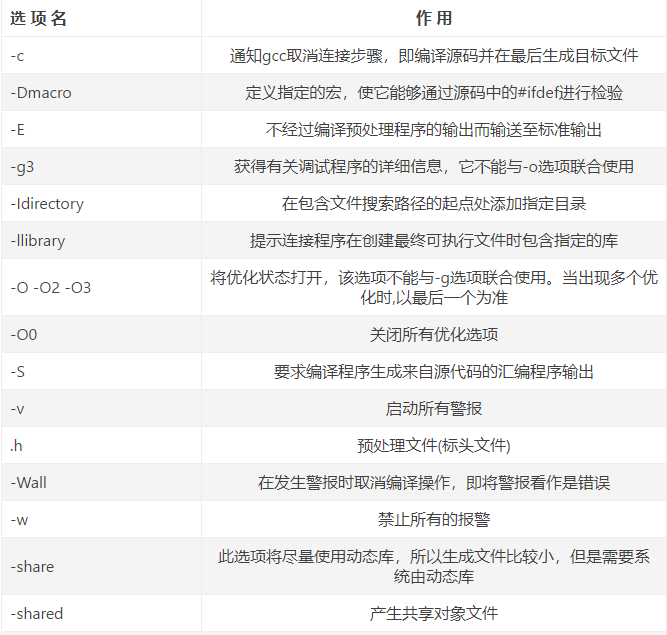
## 编译过程





## 参数详解

### 常用参数





### 预编译 -E

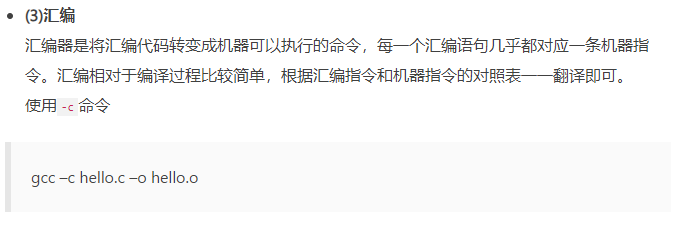


### 编译-S



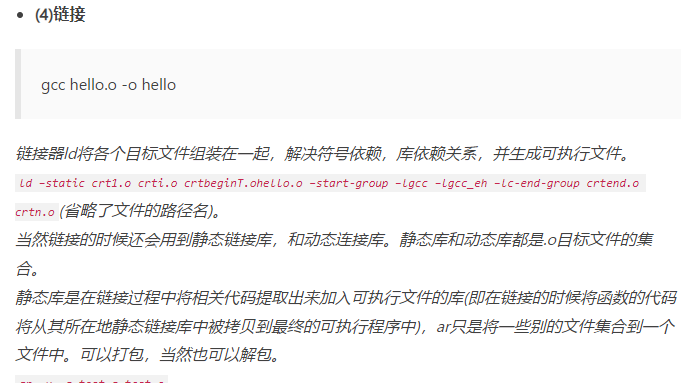
### 汇编 -C

汇编得到的结果没有链接，不可执行



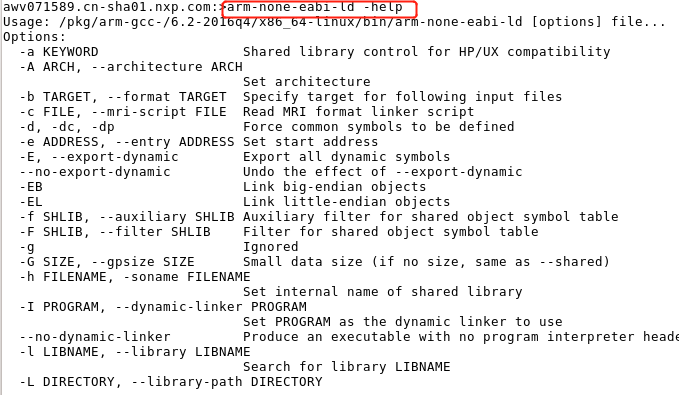
### 链接

也可以使用ld命令将汇编得到的.o连接成可执行文件

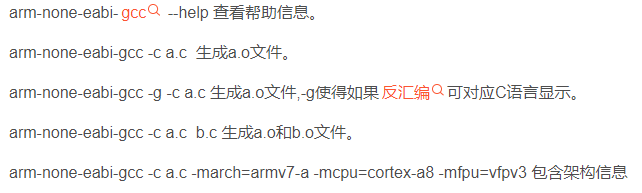


## arm-none-eabi交叉编译工具

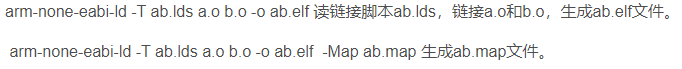
### help



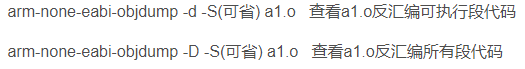
### [arm](https://so.csdn.net/so/search?q=arm&spm=1001.2101.3001.7020)-none-eabi-gcc



### arm-none-eabi-ld



### arm-none-eabi-objdump



### arm-none-eabi-objcopy

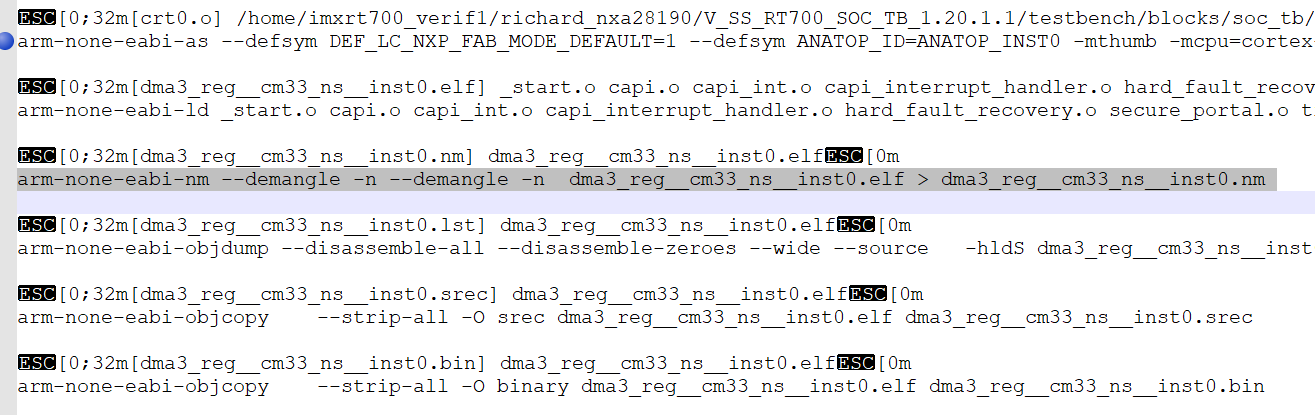


### arm-none-eabi-size

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

### 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

### 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

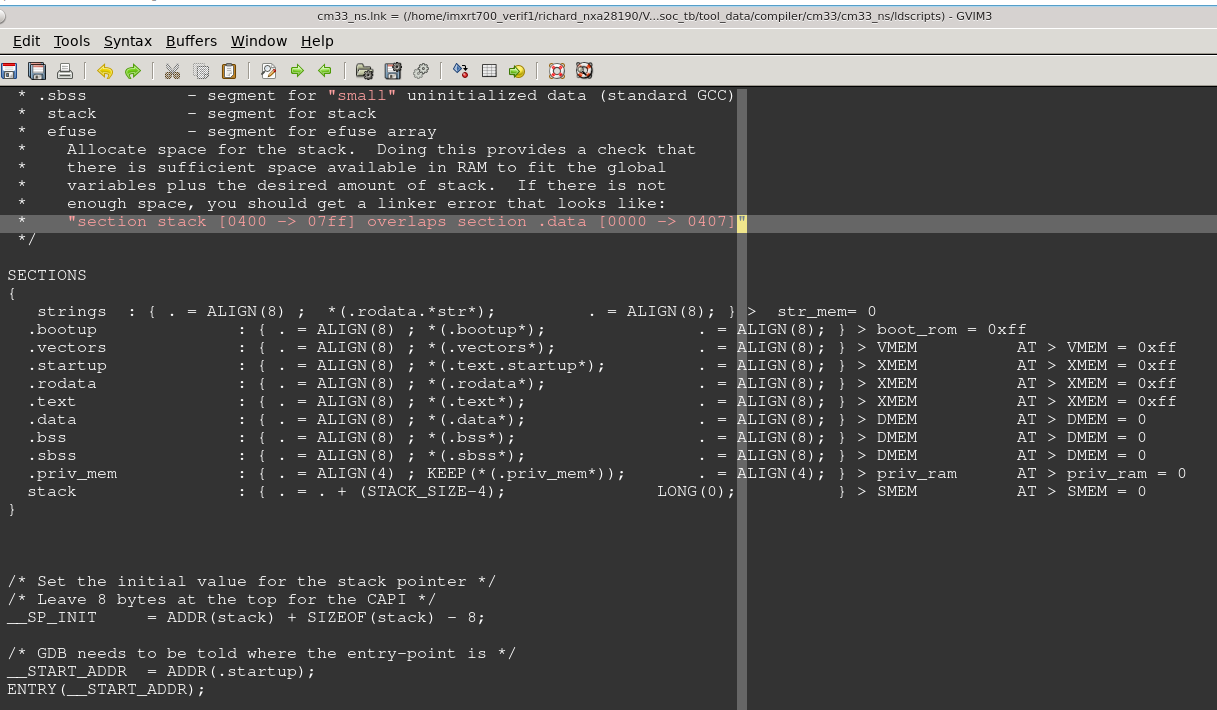
CM33 NS demo： code，data-》SRAM4 NS , vector -》SRAM1 NS

#### 生成linkfile

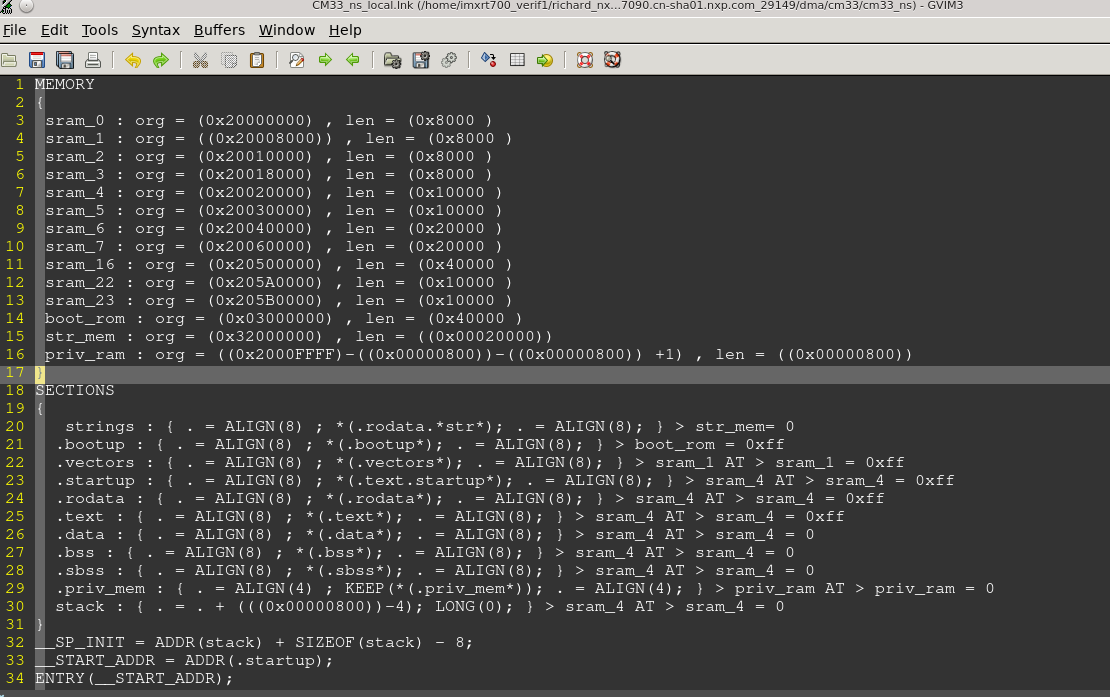
##### Non-secure demo linkfile

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



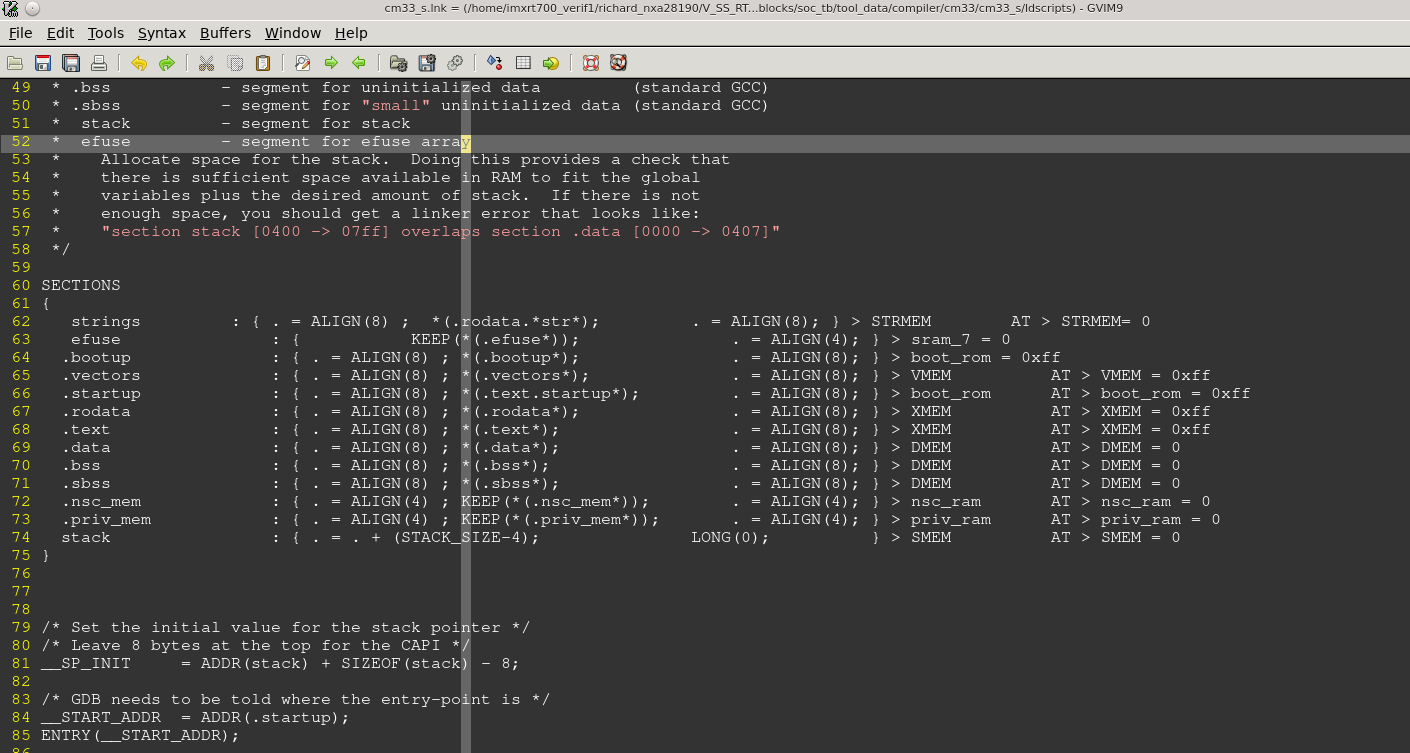
##### Secure demo linkfile

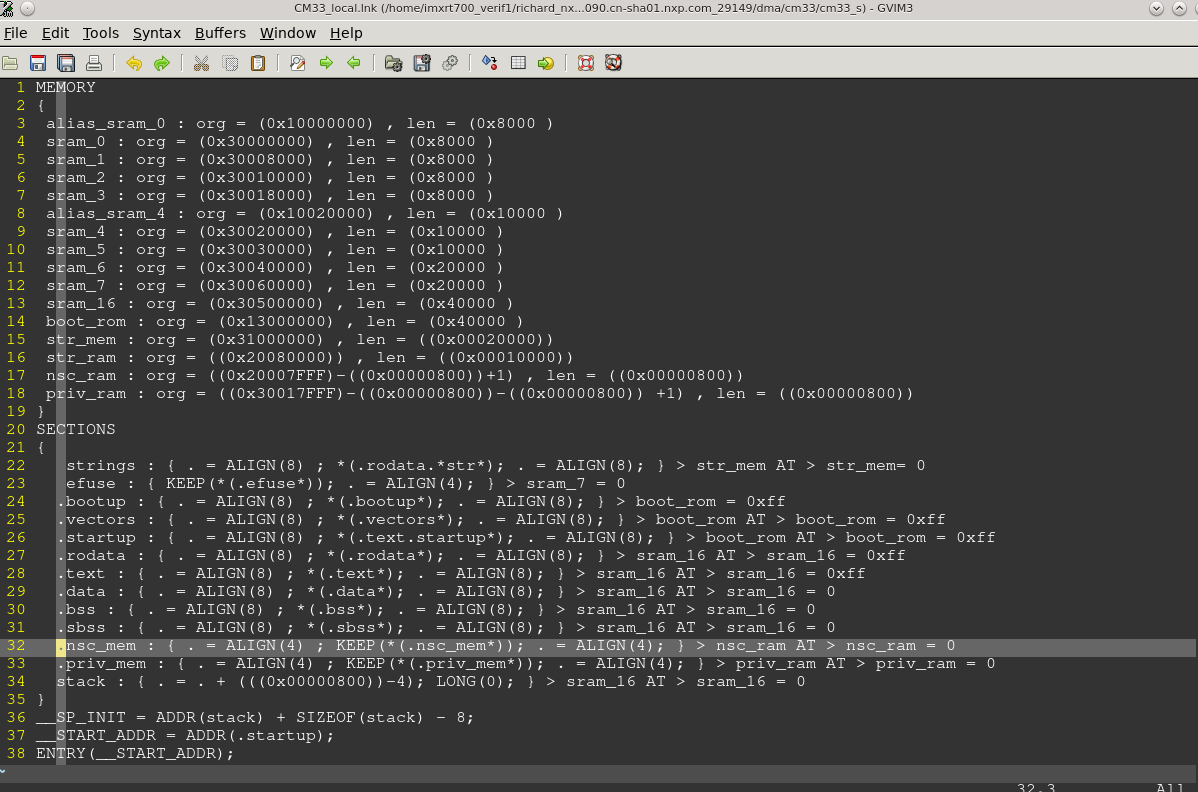
### C-Stimulus : /home/imxrt700\_verif1/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.20.1.1/testbench/blocks/soc\_tb/tool\_data/compiler/include/sanity\_cm33\_init.c

### 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

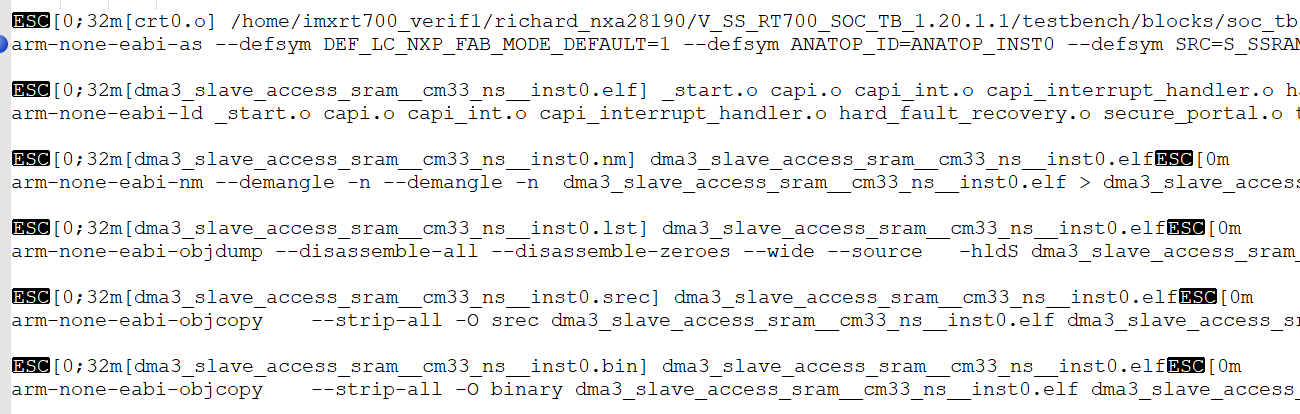
编译.S

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

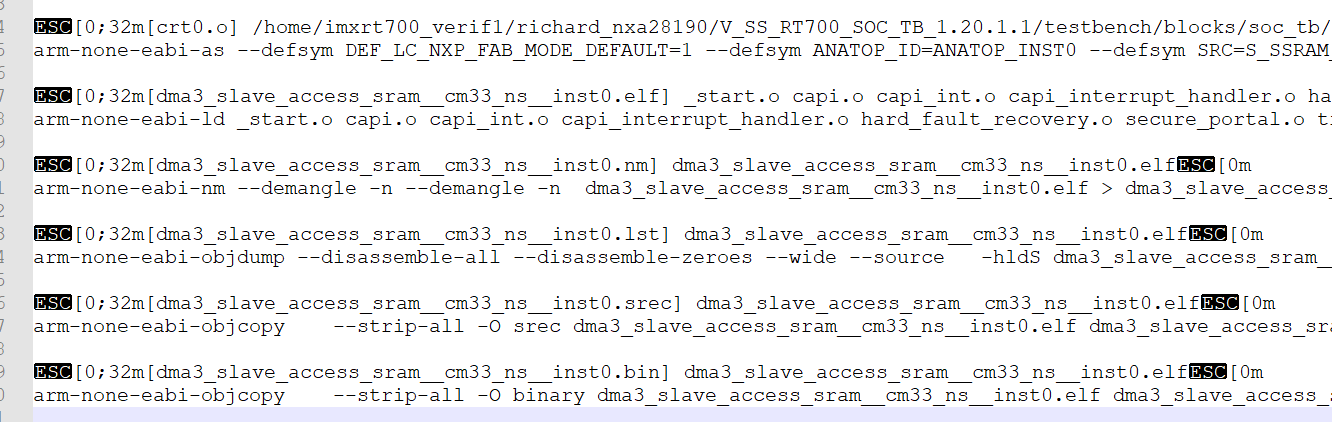
将.o文件按照link文件的要求整合起来，生成map跟elf文件

##### 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

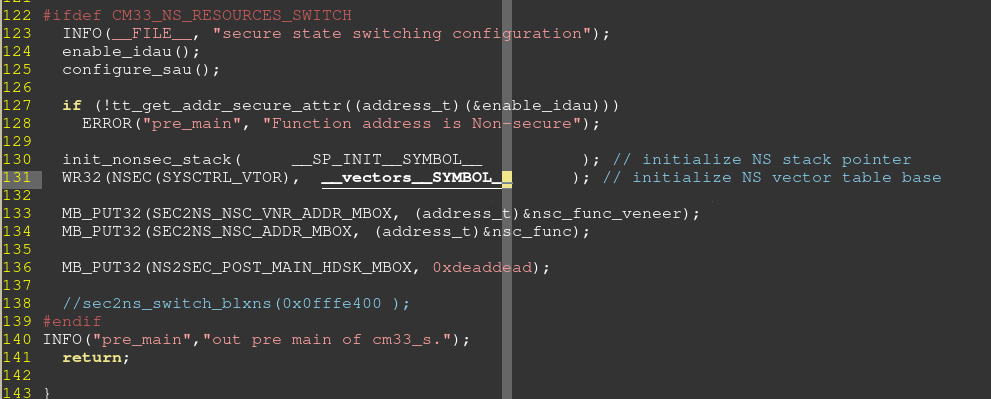
Secure demo从CM33 BOOT 起来后，执行pre\_main函数，跳转之前trustzone会对ram进行non-secure的配置

###### 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



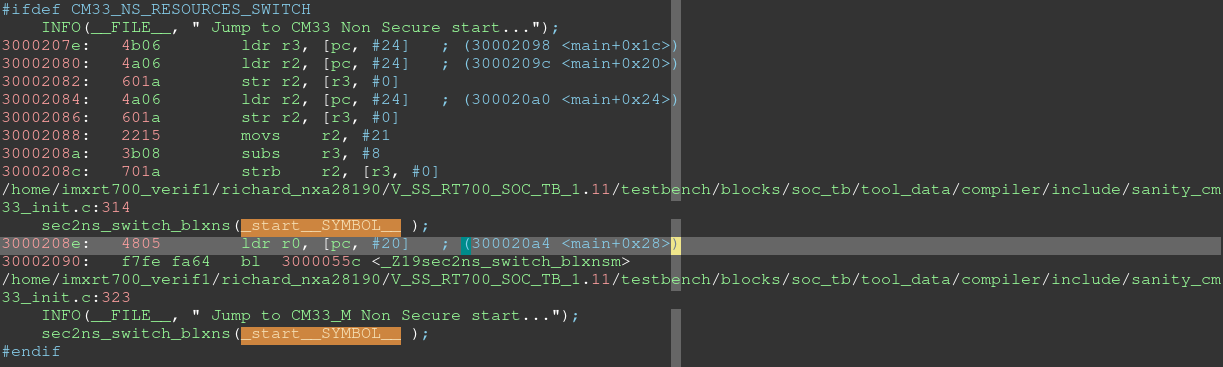
1. Config SAU,idau配置NON-SECURE address, NON-SECURE ram

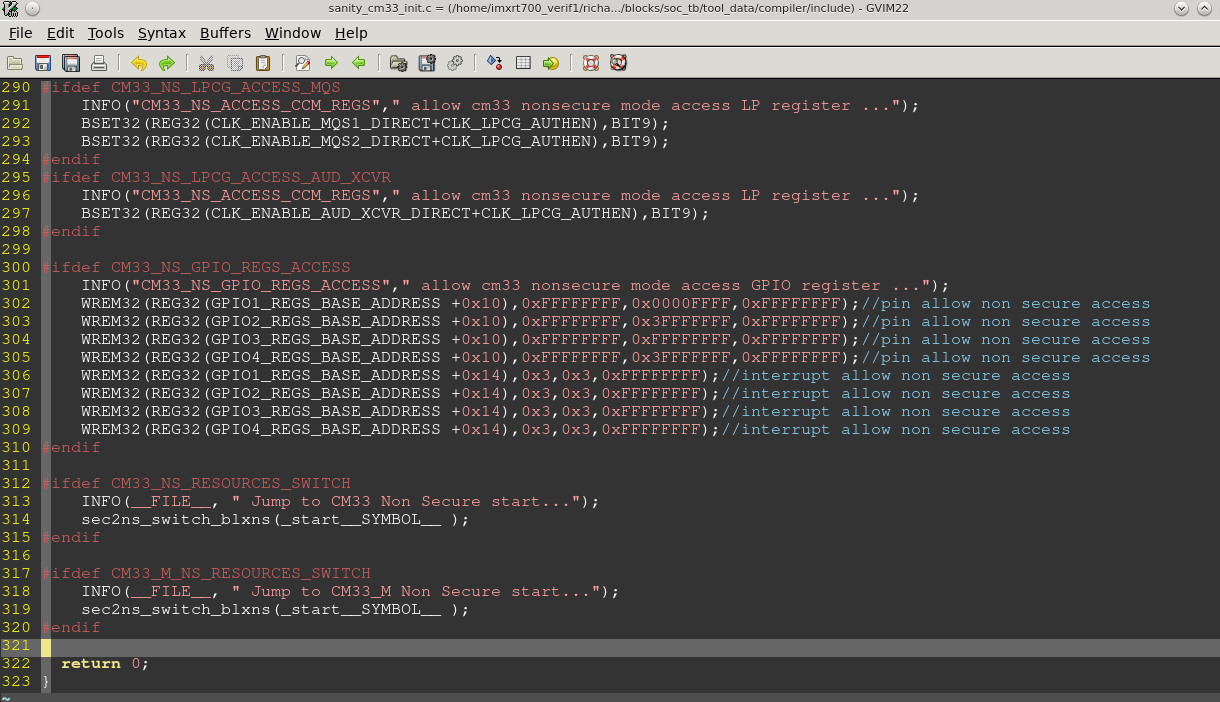


###### Boot into non-secure demo

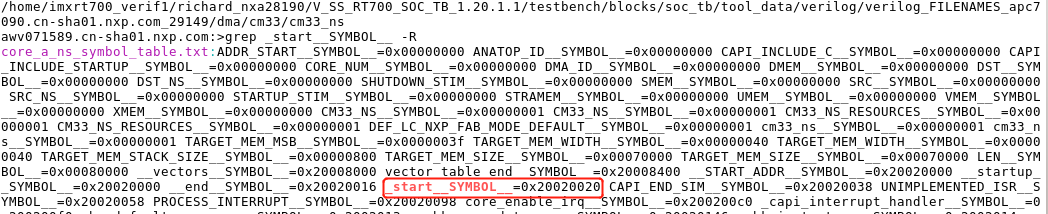
1. 执行Sanity\_cm33\_init.c中的main函数。

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)

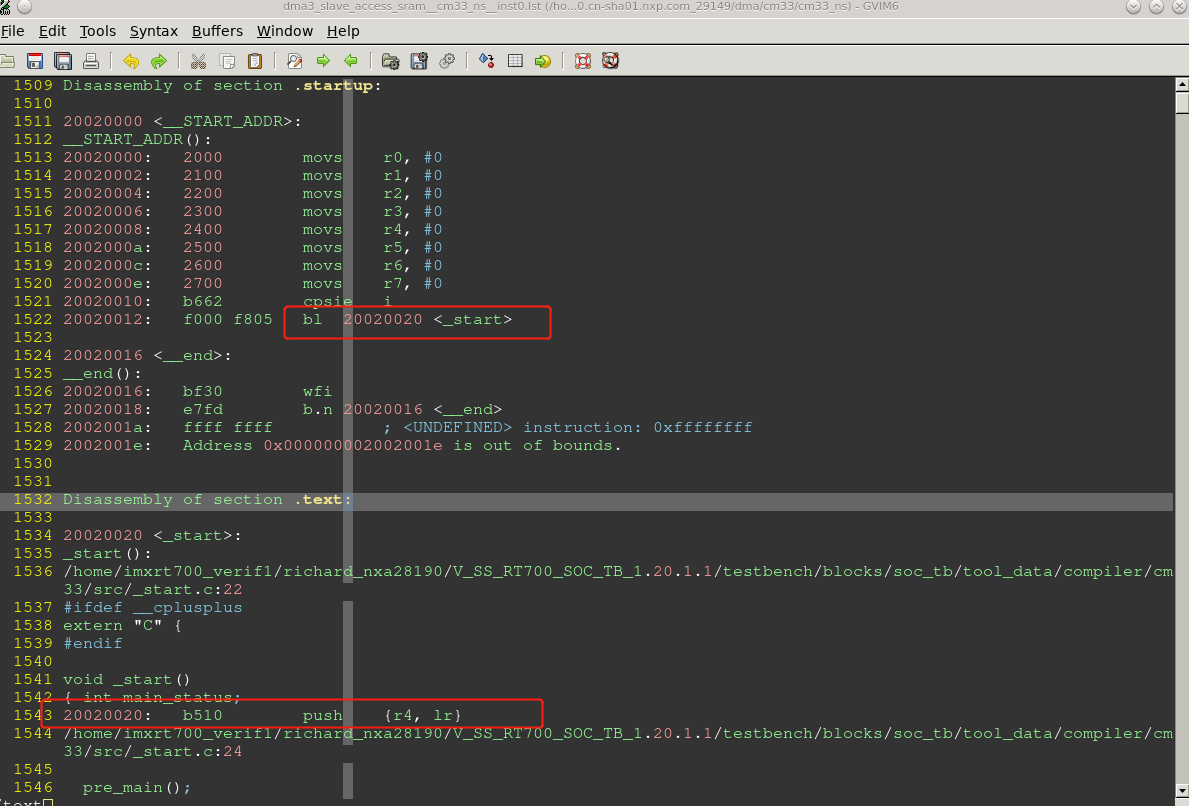




CM33\_ns\_local.lnk



CM33 ns demo反汇编lst文件



##### 信号分析

###### Vector重定向

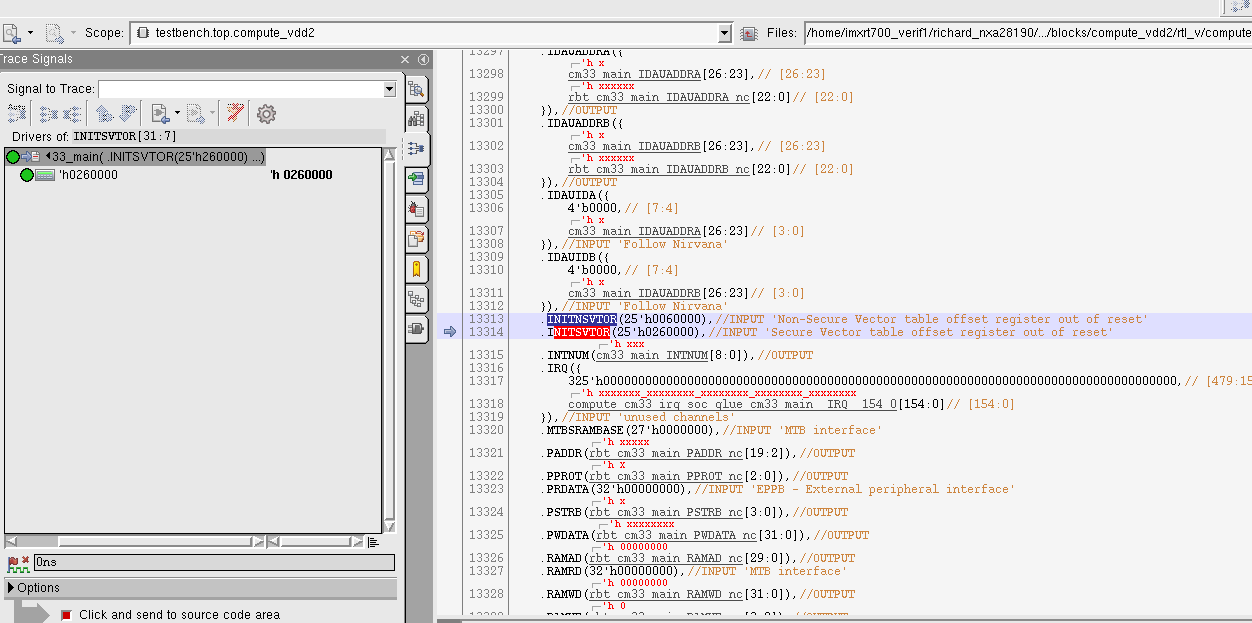
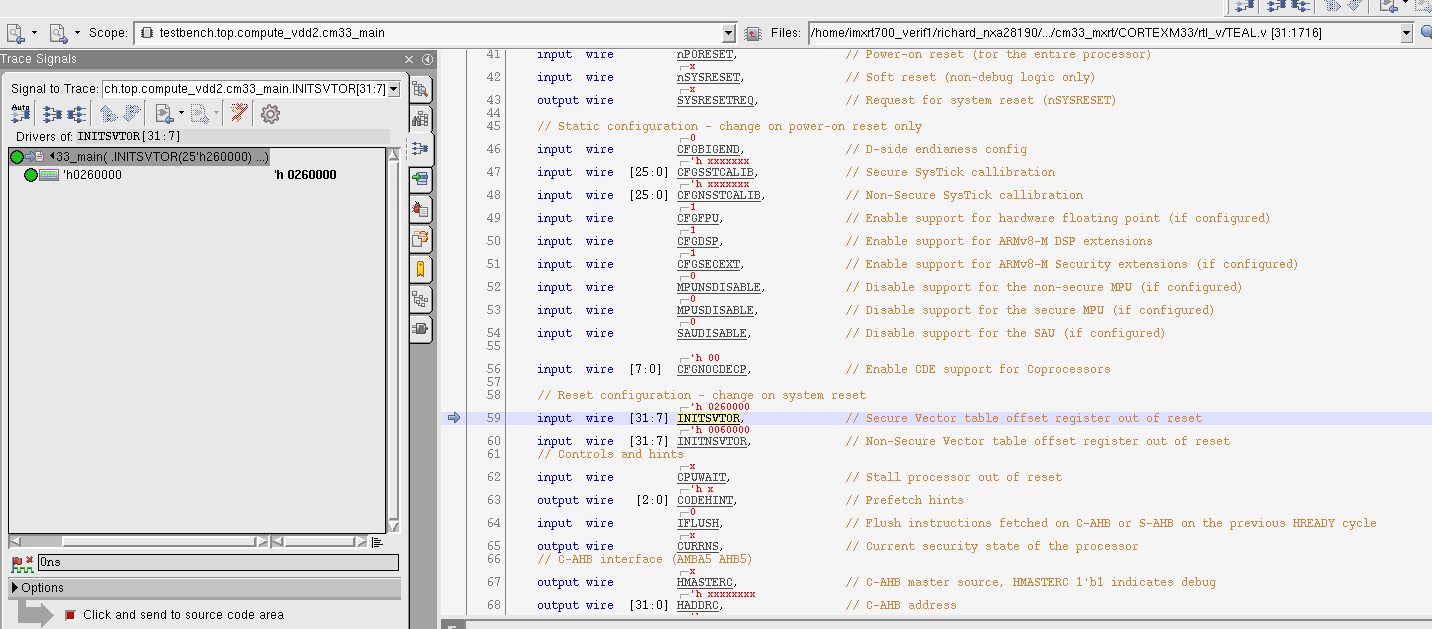
芯片上电后，secure vector会默认从0，重定向到0x13000000.

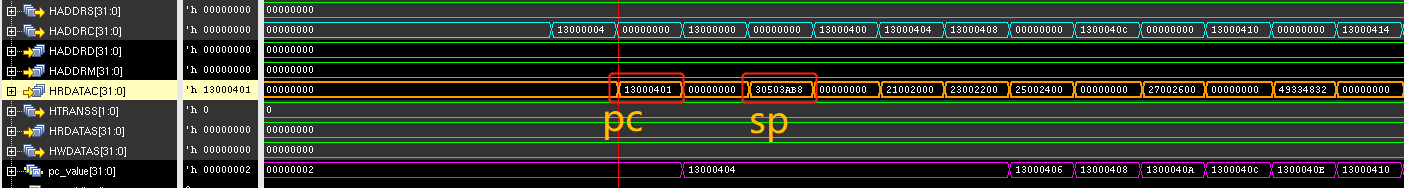
然后从0x13000400处取指令执行

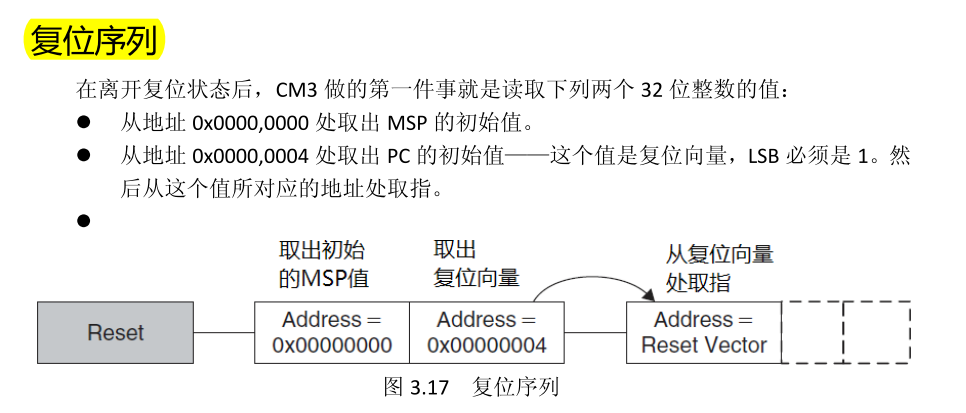
Secure vctor如下所示

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





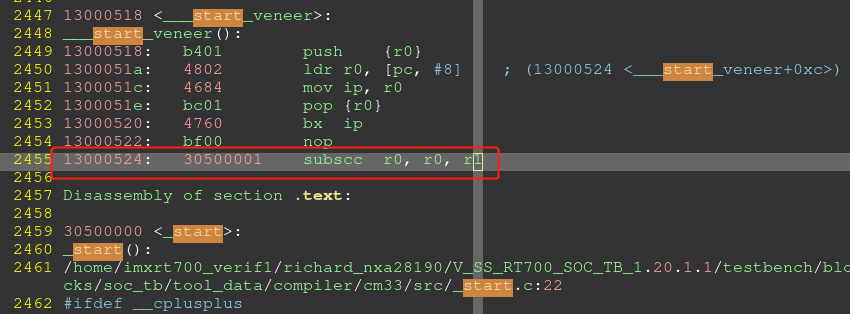


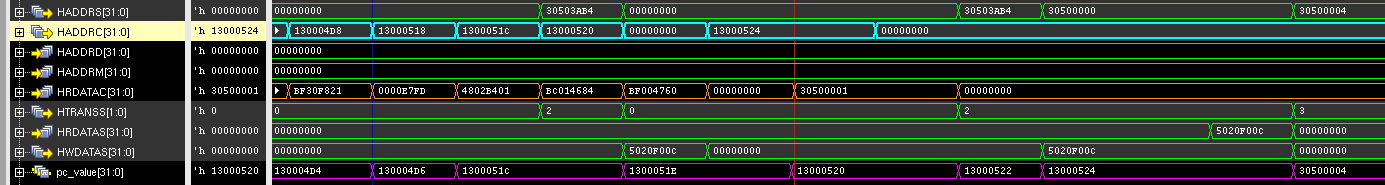


###### 跳转到c

Crt0.s





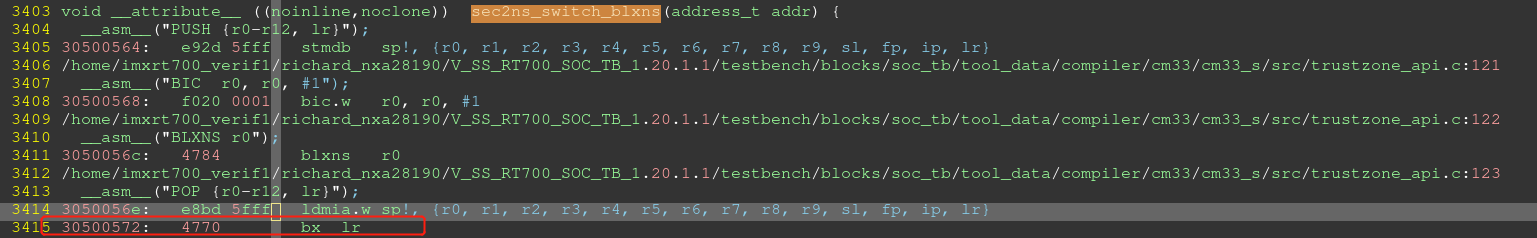


###### 跳转到NON-SECURE DEMO

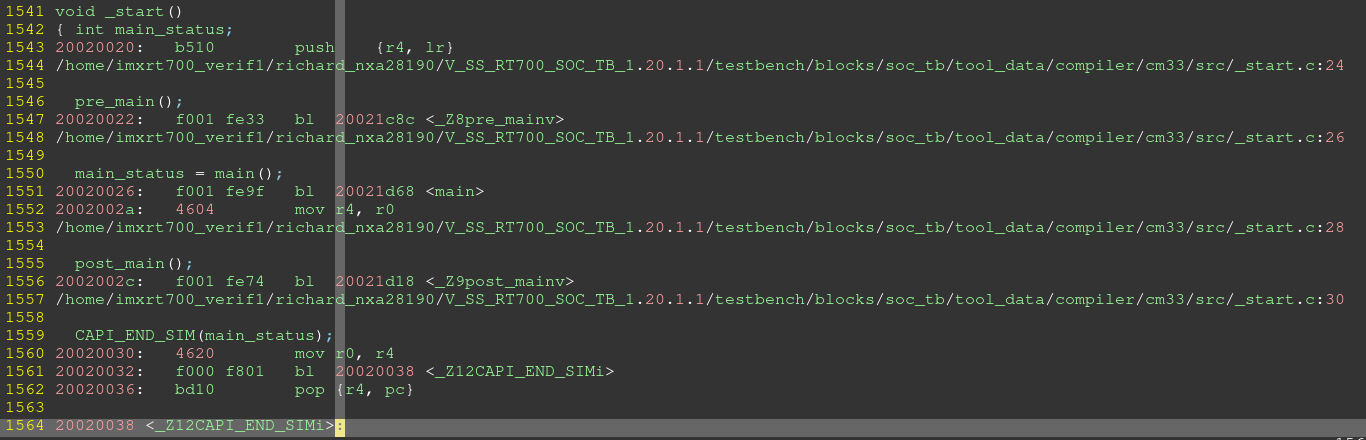
跳转到non-secure demo的\_start.c

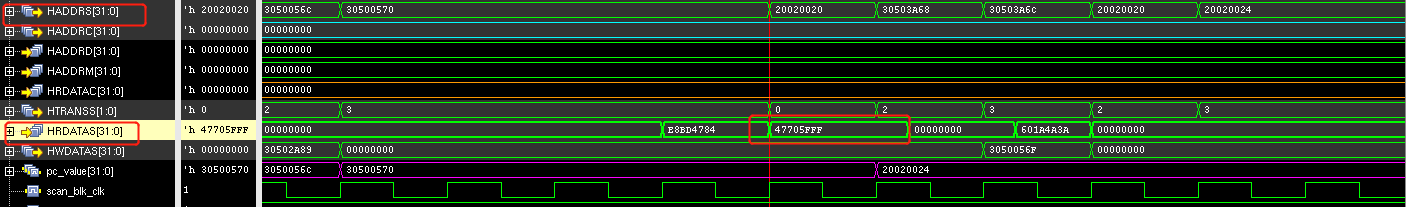
[Boot into non-secure demo](#_Boot_into_non-secure)

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

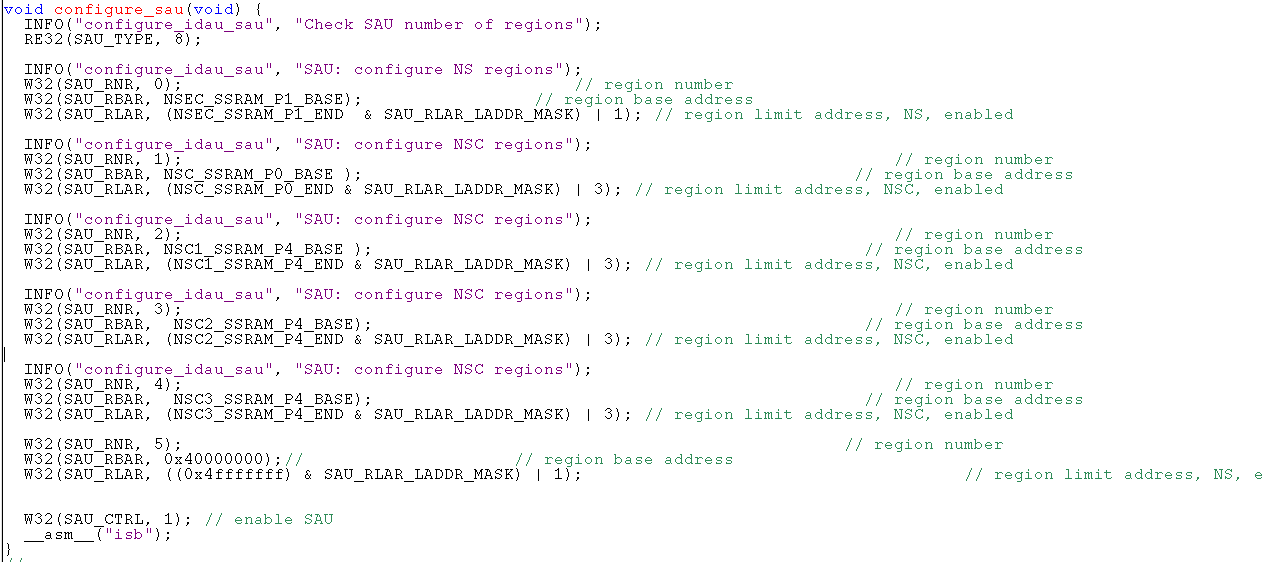
以CM33&CM33\_M为例：

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

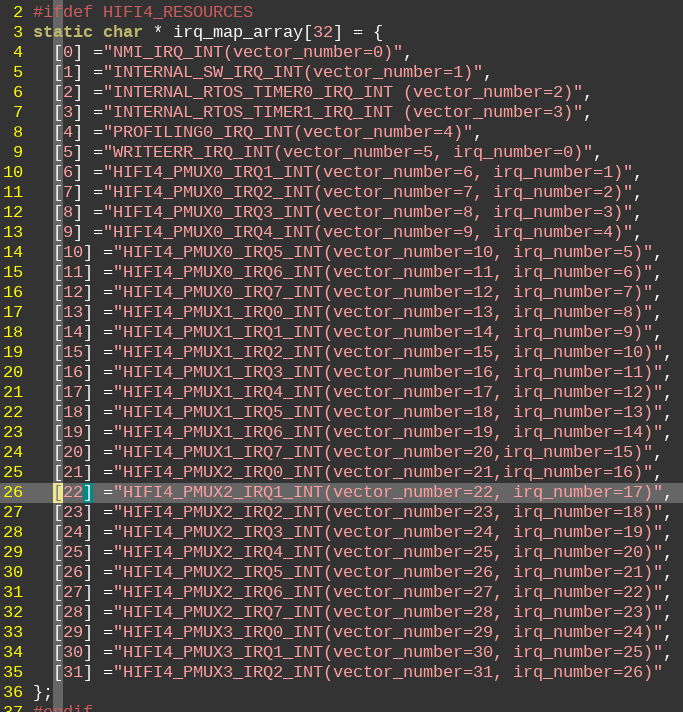
111b - Reserved

# 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

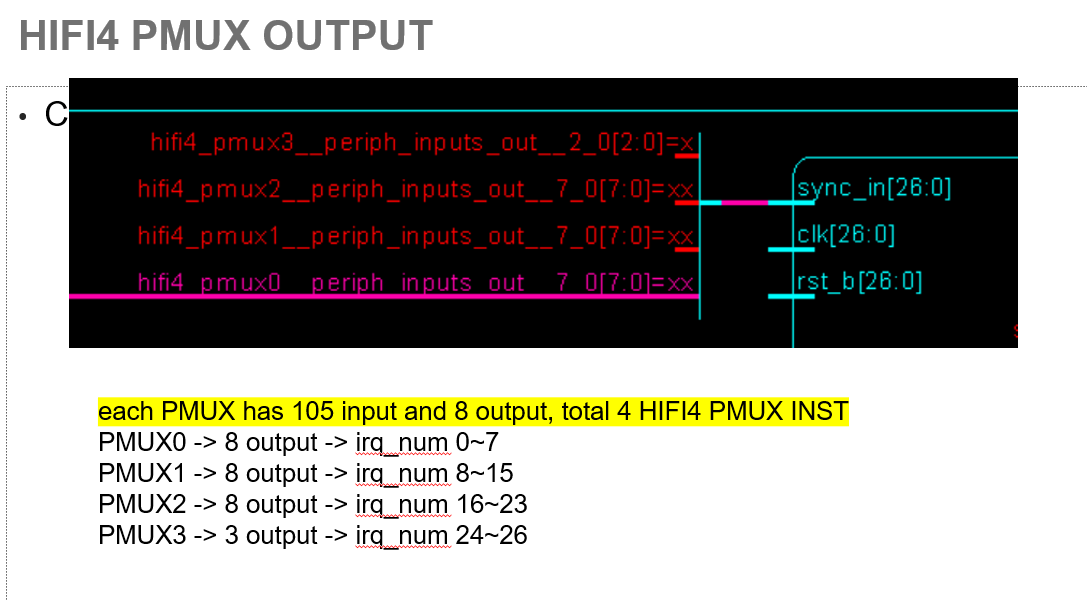
第一个参数2，表示使用第二个mux（一共4个mux，每个8个输出端口）。第三个参数1，表示使用mux的output口为1。。对应就是2\*8+1=17.。

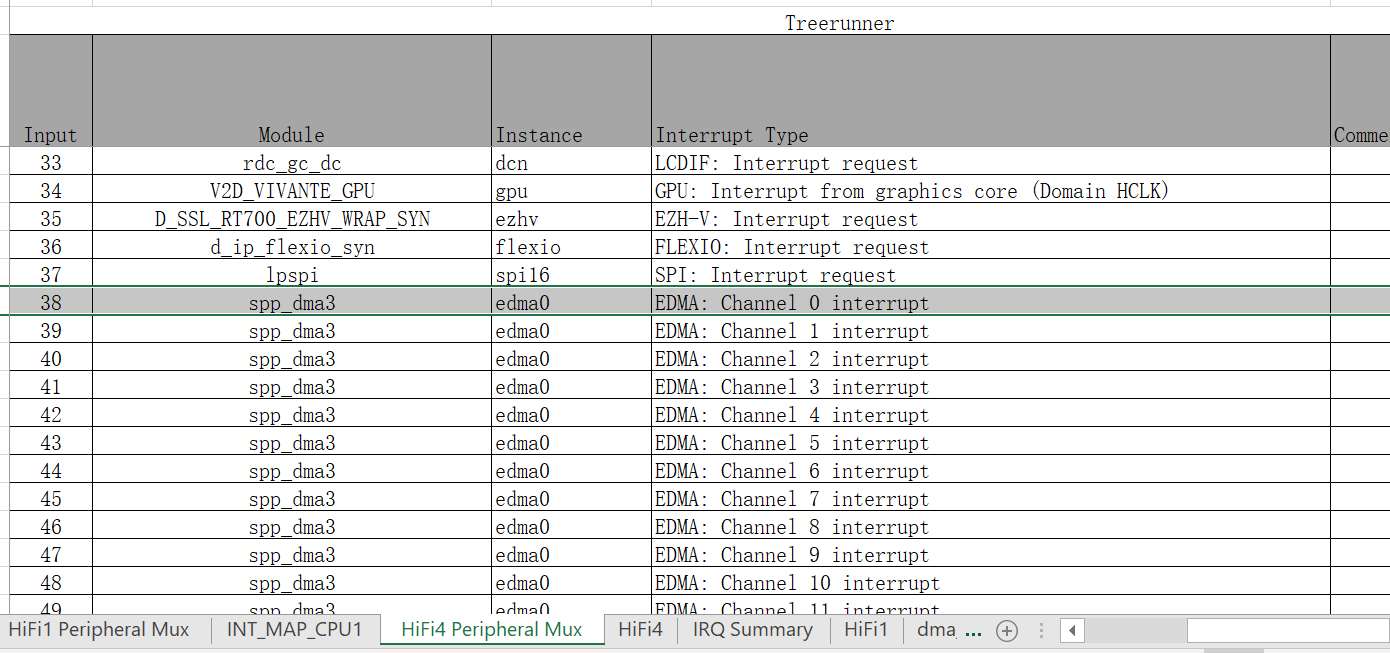
17的意思就是HIFI的Binterrupt使用的是bit17.

1. 中断处理完成后清除中断，否则不停进入中断服务程序

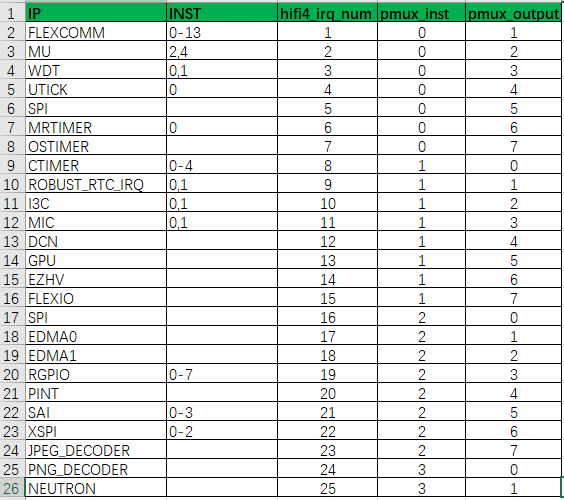
BSET32(DMA3\_CH\_INT(channel\_index),DMA3\_CH\_INT\_INT\_MASK);

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

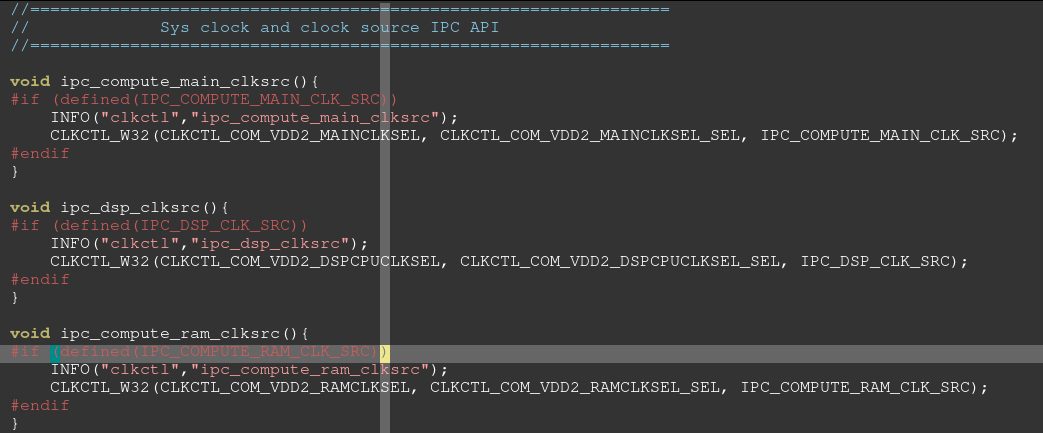


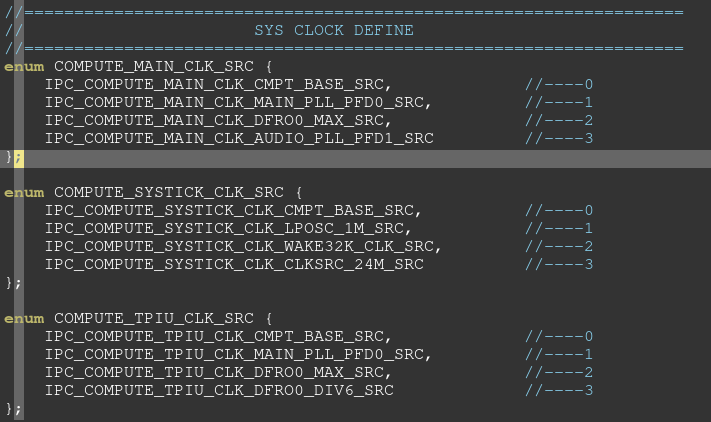
# CLOCK

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



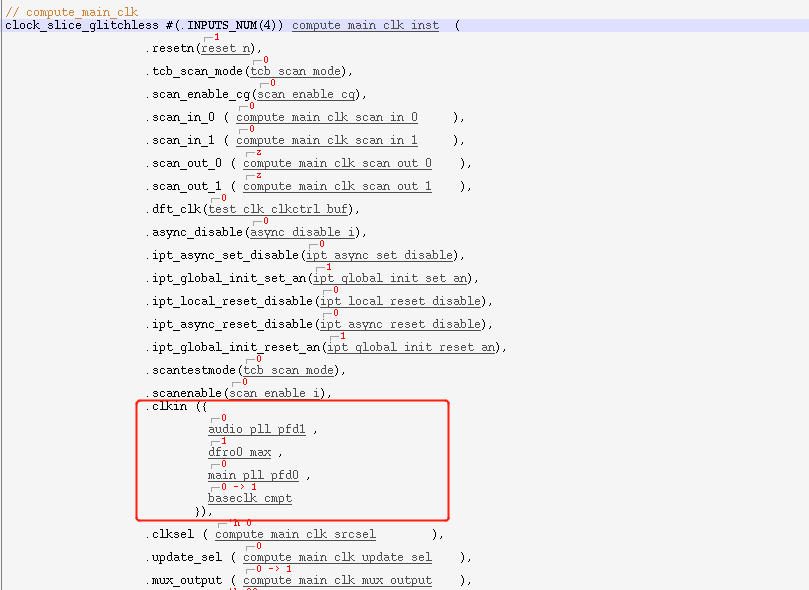


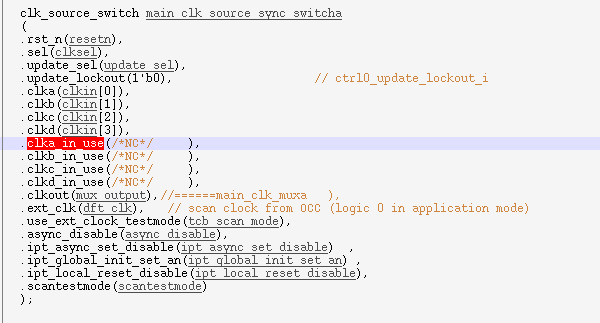
## Clock选信号

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

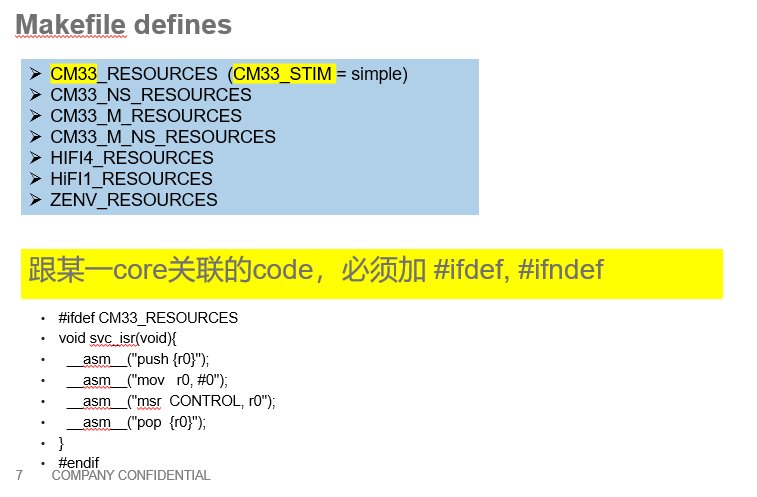
CM33\_M\_C\_ARG += DEBUG\_RUN

## 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：

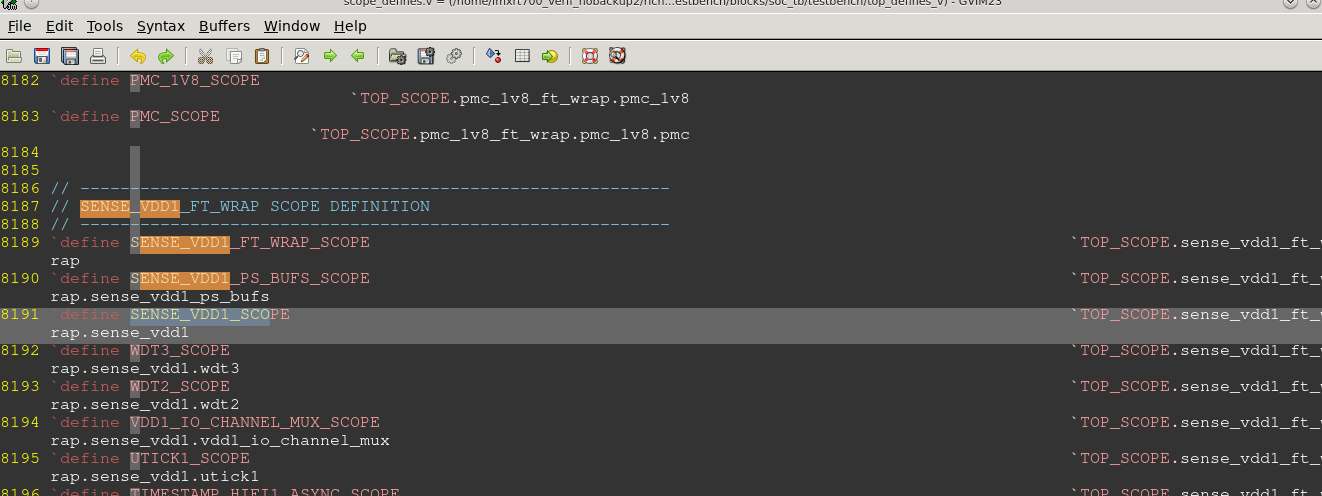
/home/imxrt700\_verif\_nobackup2/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.33/testbench/blocks/soc\_tb/testbench/instances\_v

SV变量的定义：

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

## scope 定义

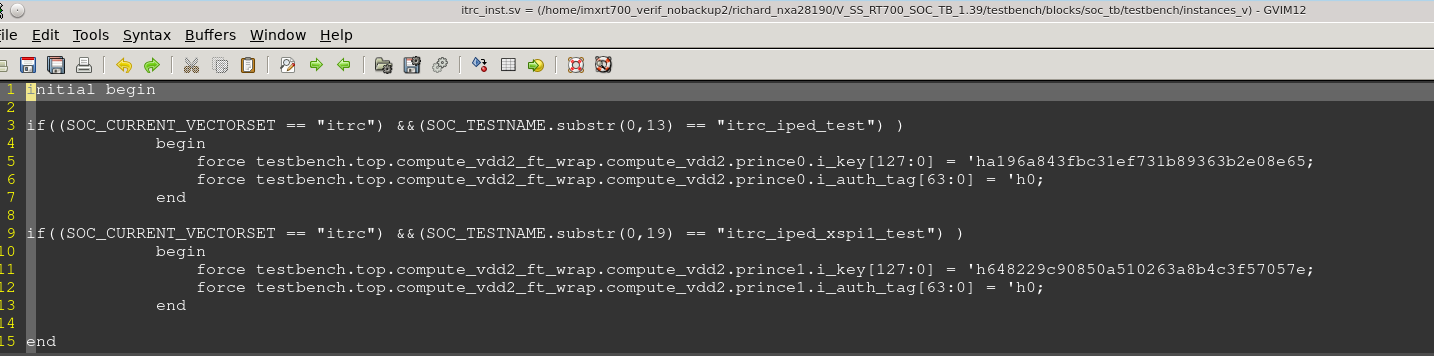
/home/imxrt700\_verif\_nobackup2/richard\_nxa28190/V\_SS\_RT700\_SOC\_TB\_1.43/testbench/blocks/soc\_tb/testbench/top\_defines\_v/ scope\_defines.v



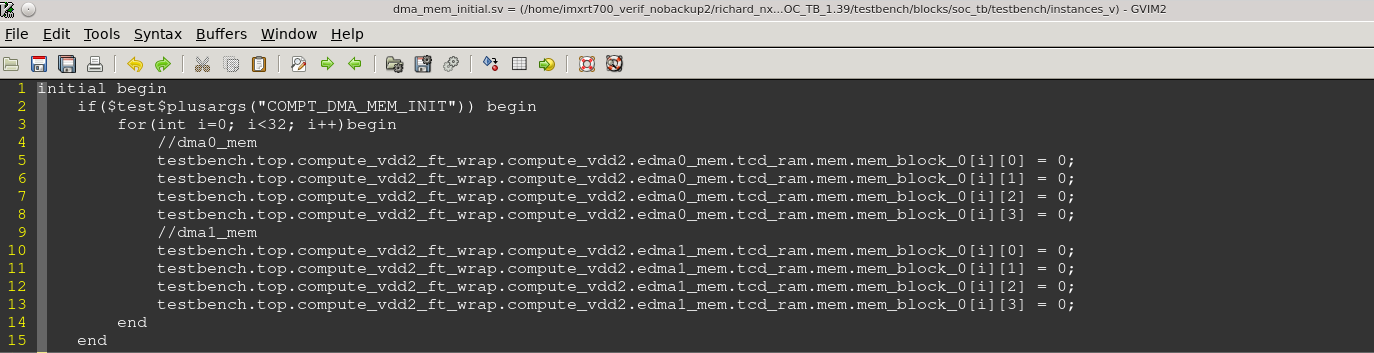
## modules\_v/instances\_v

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