目录

[File & folder 3](#_Toc128400376)

[Test case 3](#_Toc128400377)

[C查阅工具 6](#_Toc128400378)

[常用文件 6](#_Toc128400379)

[头文件关系 8](#_Toc128400380)

[Code analysis 8](#_Toc128400381)

[soc\_new\_request 8](#_Toc128400382)

[init\_ip\_request 9](#_Toc128400383)

[dma\_populate\_request 10](#_Toc128400384)

[soc\_allocate 10](#_Toc128400385)

[dma\_init\_context 11](#_Toc128400386)

[create\_context\_dma\_inst 11](#_Toc128400387)

[create\_context\_dma3 12](#_Toc128400388)

[soc\_get\_context\_pointer 13](#_Toc128400389)

[ip\_request\_t 13](#_Toc128400390)

[dma\_context\_t 14](#_Toc128400391)

[DMA3\_CONTEXT 14](#_Toc128400392)

[dma3\_reg\_check\_reset 15](#_Toc128400393)

[RE32 16](#_Toc128400394)

[dma3\_reg\_check\_complement\_32bit 16](#_Toc128400395)

[WRE32 18](#_Toc128400396)

[Register 18](#_Toc128400397)

[中断工作流程 18](#_Toc128400398)

[dma3\_setup\_int(dma3\_context); 18](#_Toc128400399)

[dma3\_setup\_hw 19](#_Toc128400400)

[Register definition 21](#_Toc128400401)

[DMA3\_TCD\_SADDR 21](#_Toc128400402)

[DMA3\_TCD\_SOFF 22](#_Toc128400403)

[DMA3\_TCD\_ATTR 23](#_Toc128400404)

[DMA3\_TCD\_NBYTES 24](#_Toc128400405)

[DMA3\_TCD\_SLAST 26](#_Toc128400406)

[DMA3\_TCD\_DADDR 28](#_Toc128400407)

[DMA3\_TCD\_CITER 29](#_Toc128400408)

[DMA3\_TCD\_DOFF 30](#_Toc128400409)

[DMA3\_TCD\_DLAST 31](#_Toc128400410)

[DMA3\_TCD\_BITER 32](#_Toc128400411)

[DMA3\_TCD\_CSR 34](#_Toc128400412)

[DMA3\_CH\_SBR 37](#_Toc128400413)

[DMA3\_CH\_CSR 39](#_Toc128400414)

[DMA3\_CH\_PRI 40](#_Toc128400415)

[DMA3\_GRPRI 41](#_Toc128400416)

[Makefile 42](#_Toc128400417)

[Testbench overview knowledge 43](#_Toc128400418)

[Module description 44](#_Toc128400419)

[TCD 44](#_Toc128400420)

[Channel preemption 45](#_Toc128400421)

[Scat/gather 46](#_Toc128400422)

[Example 46](#_Toc128400423)

[Channel linking 47](#_Toc128400424)

[Example 47](#_Toc128400425)

[Channel SBR 48](#_Toc128400426)

# File & folder

## Test case

* Test case所在文件夹：

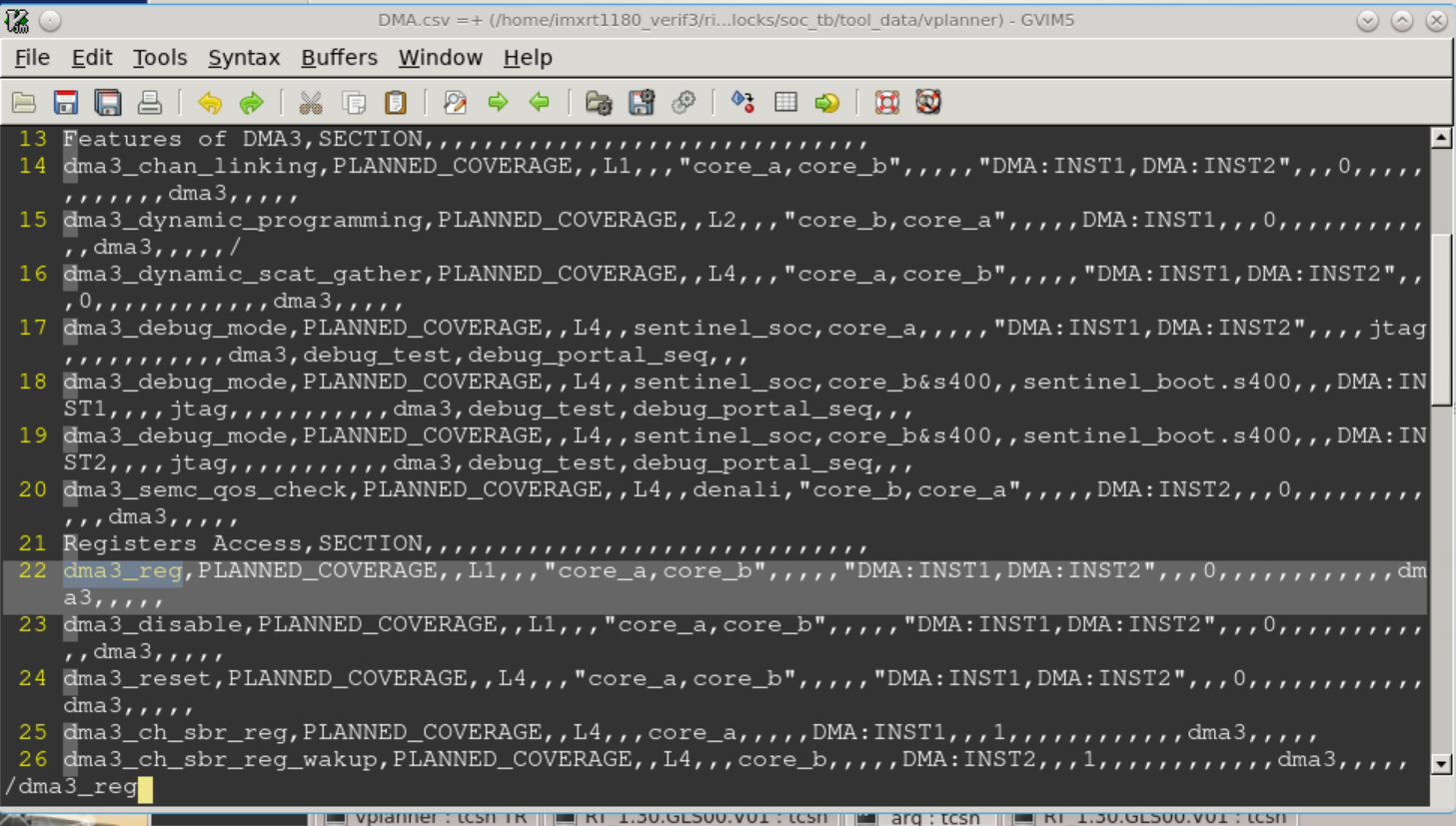
/home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/blocks/soc\_tb/vectors/dma3/stimulus

* Generate Plan

/home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/blocks/soc\_tb/tool\_data/vplanner

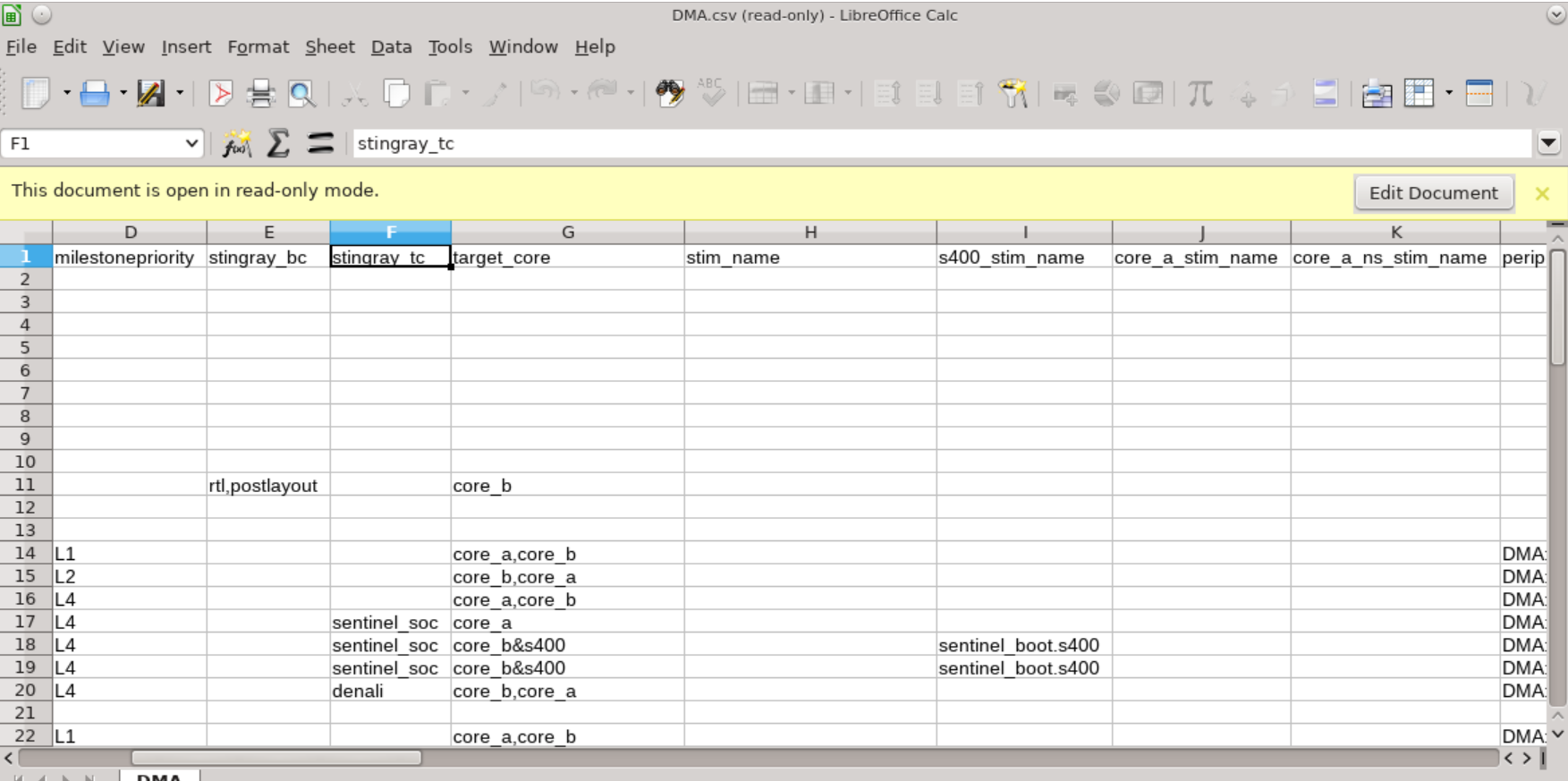
/home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01

vgen -plan DMA.csv



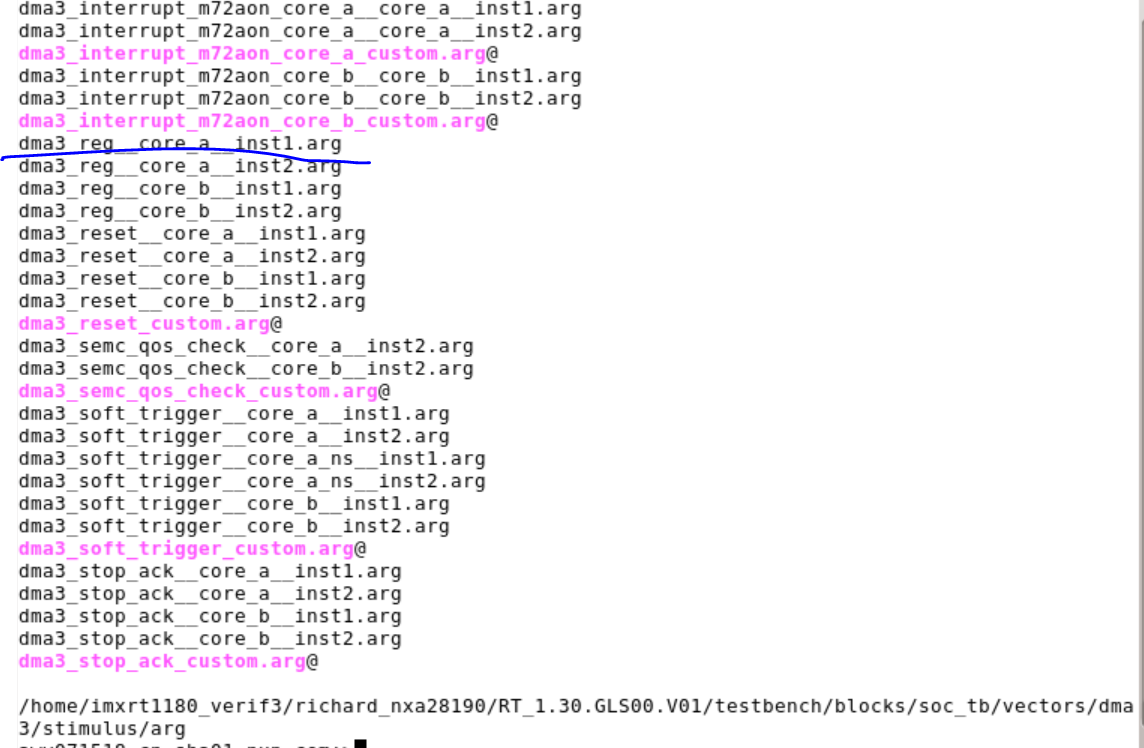
* 查看命令需要的-bc -tc参数，test case

/RT\_1.29/testbench/blocks/soc\_tb/tool\_data/vplanner]$ openoffice DMA.csv



Test case名称从如下文件夹中获取

/home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/blocks/soc\_tb/vectors/dma3/stimulus/arg



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

拼凑完成的command:

bsub -R rhel7 -q interq -Ip soc verilog -irun -block soc\_tb -bc rtl -tc default -vectors dma3 -test dma3\_reg\_\_core\_a\_\_inst1 -shm -keeptemps -session default &

bsub -R rhel7 -q interq -Ip soc verilog -irun -block soc\_tb -bc rtl -tc default -vectors dma3 -test dma3\_interrupt\_core\_a\_inst2\_test\_1 -shm -keeptemps -session default &

batchq -n 3

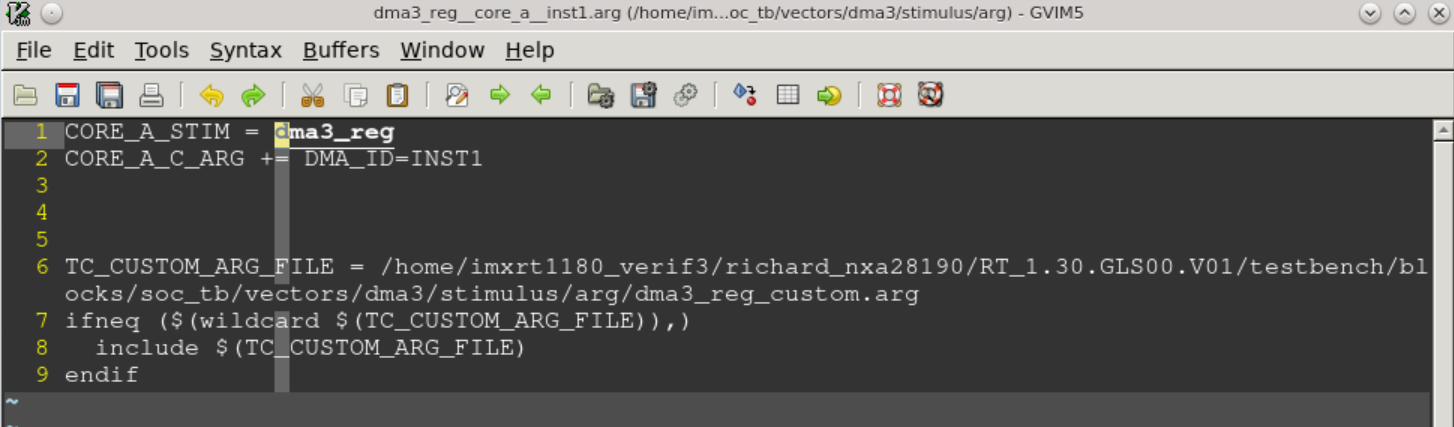
batchq\_mt -n 8

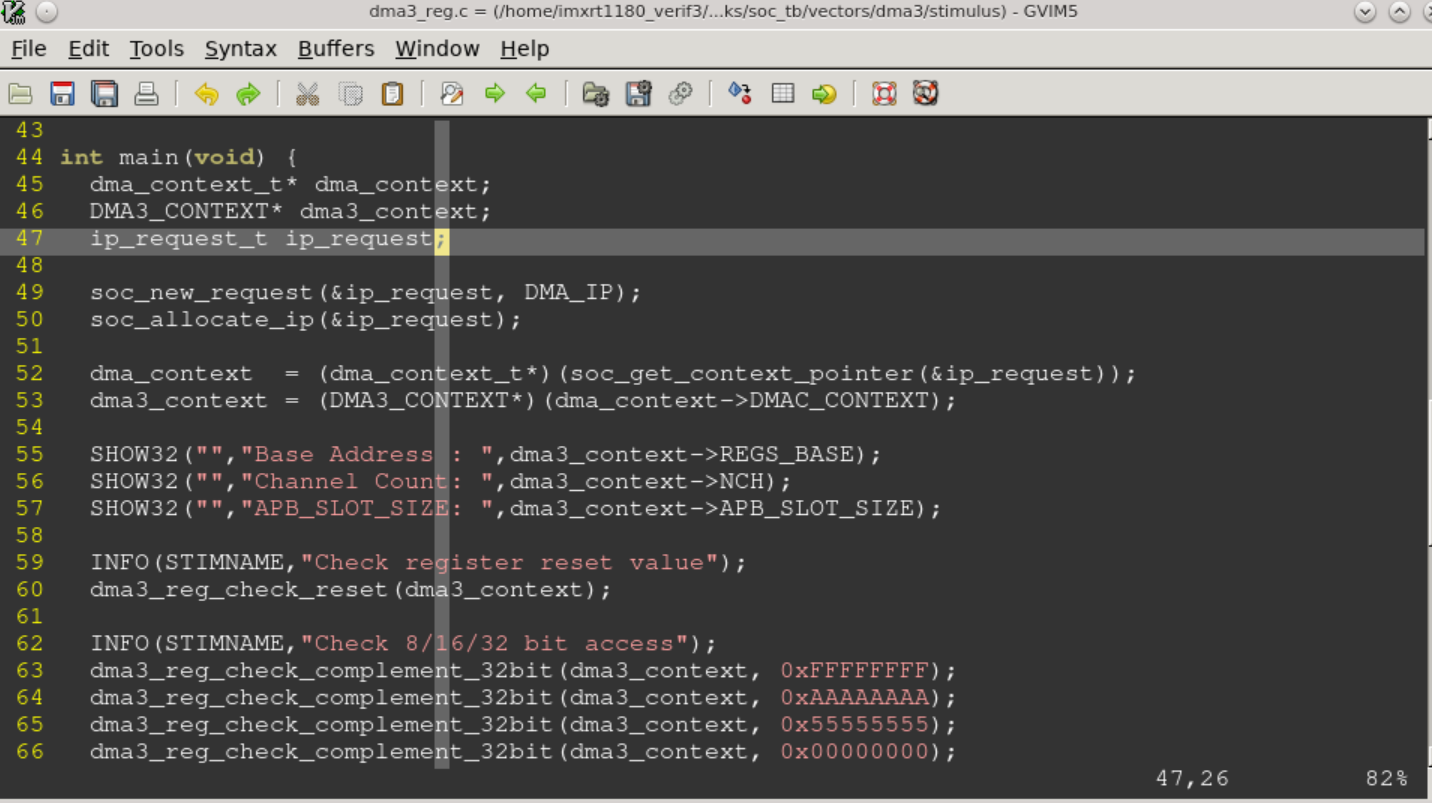
-sim\_arg “-gui”

-no\_compile

* c文件

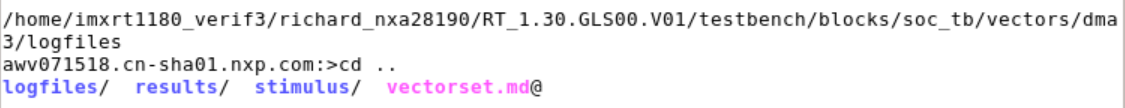
.arg文件里面指定用到的.c





* 运行结束后，会生成logfile文件夹，result文件夹

Logfile文件夹中存放的test log以及测试命令，result文件夹中存放的事波形跟tarmac文件



## C查阅工具

/home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/blocks/soc\_tb/vectors/dma3/stimulus/arg

tr.pkg OSS-sourcenavigator\_NG-/4.5 snavigator &

## 常用文件

* Arg 文件

这里指定了需要编译的test case

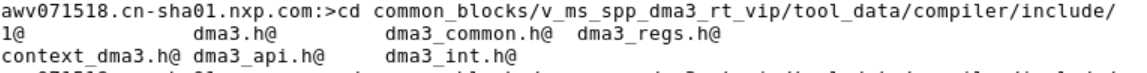
/home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/blocks/soc\_tb/vectors/dma3/stimulus/arg

* Test case .c文件

/home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/blocks/soc\_tb/vectors/dma3/stimulus

* Test case 调用的common function， create\_context\_dma3函数的位置，对DMA的属性进行初始化

/home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/common\_blocks/v\_ms\_spp\_dma3\_rt\_vip/tool\_data/compiler/include/



* DMA context ，API 定义, case里面在调用到create\_context\_dma3 之前的所有函数，变量都从这里找

/home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/blocks/soc\_tb/tool\_data/compiler/include

* NUM\_DMA\_INSTS DMA个数定义

testbench/blocks/soc\_tb/tool\_data/compiler/include

* ip\_request->context\_id = DMA\_ID;

stimulus/arg文件中的定义DMA\_ID会被传到这里，code根据DMA\_ID的index，进行对应DMA的赋值。比如base\_addr，channels等等

* ipc\_dma##num##\_assign() clock的定义，

需要clock的owner进行API的撰写，这里没有相关函数，编译会报错

* design 定义好的DMA的属性：

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

testbench/blocks/soc\_tb/tool\_data/compiler/include

dma\_context\_t(cl) [dma\_context\_t](#_dma_context_t)

testbench/blocks/soc\_tb/tool\_data/compiler/include/dma\_context.h

DMA3\_CONTEXT(cl) [DMA3\_CONTEXT](#_DMA3_CONTEXT) testbench/common\_blocks/v\_ms\_spp\_dma3\_rt\_vip/tool\_data/compiler/include/dma3.h

ip\_request\_t(cl) [ip\_request\_t](#_ip_request_t)

testbench/blocks/soc\_tb/tool\_data/compiler/include/ip\_request.h

DMA\_IP \ soc\_new\_request \ soc\_allocate\_ip \ soc\_get\_context\_pointer

testbench/blocks/soc\_tb/tool\_data/compiler/include/context\_api.h

Mem\_map.h中定义了DMA3寄存器的各类属性信息，包括DMA base addr， DMA的各类寄存器

* Map文件

/home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/blocks/soc\_tb/tool\_data/sray/mem\_map

* Plan

/home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/blocks/soc\_tb/tool\_data/vplanner

* 编译C需要的头文件，environment.h之类

/home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/blocks/soc\_tb/tool\_data/compiler/include

testbench/blocks/soc\_tb/tool\_data/compiler/include/environment.h

* PP 文件

cd testbench/common\_blocks/mcu\_tb\_utils/tool\_data/compiler/

* 编译log

RT\_1.30.GLS00.V01/testbench/blocks/soc\_tb/tool\_data/verilog/verilog\_FILENAMES\_apc7116.cn-sha01.nxp.com\_7203

## 头文件关系

Environment.h

/home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/blocks/soc\_tb/tool\_data/compiler/include

# Code analysis

dma\_context\_t\* dma\_context;

DMA3\_CONTEXT\* dma3\_context;

ip\_request\_t ip\_request;

根据传入的IP的序列号，进行IP的初始化。DMA是0

soc\_new\_request(&ip\_request, DMA\_IP);

soc\_allocate\_ip(&ip\_request);

获取DMA相关的内容

dma\_context = (dma\_context\_t\*)(soc\_get\_context\_pointer(&ip\_request));[soc\_get\_context\_pointer](#_soc_get_context_pointer)

dma3\_context = (DMA3\_CONTEXT\*)(dma\_context->DMAC\_CONTEXT);

## soc\_new\_request

.\testbench\ blocks\soc\_tb\tool\_data\compiler\include\context\_api.h

soc\_new\_request(fu) void (ip\_request\_t \*,ip\_type\_e) testbench/blocks/soc\_tb/tool\_data/compiler/include/context\_api.h

根据传入的IP 的ID，获取初始化函数

void soc\_new\_request(ip\_request\_t\* ip\_request, ip\_type\_e ip\_type){

ip\_request->ip\_type = ip\_type;

ip\_request->init\_context = NULL;

ip\_request->clock\_config = NULL;

ip\_request->clock\_disable = NULL;

ip\_request->trig\_config = NULL;

SHOW8("soc\_new\_request","requested ip\_type enumeration number is: 0x", ip\_type);

if(init\_ip\_request[ip\_type])

init\_ip\_request[ip\_type](ip\_request);

else

FATAL("soc\_new\_request", "You need to define --> <CORE>\_C\_ARG += <BLOCK>\_ID=X <-- in order to use this function");

}

### init\_ip\_request

.\testbench\ blocks\soc\_tb\tool\_data\compiler\include\context\_api.h

init\_ip\_request\_t init\_ip\_request[SUPPORTED\_IP\_TYPES\_IN\_SOC] = {

[DMA\_IP] = dma\_populate\_request, //0

[BLK\_CTRL\_IP] = blk\_ctrl\_populate\_request, //1

[MU\_IP] = mu\_populate\_request, //2

[SEMA\_IP] = sema\_populate\_request, //3

[TRDC\_MGR\_IP] = trdc\_mgr\_populate\_request, //4

[TRDC\_MC\_IP] = trdc\_mc\_populate\_request, //5

[SYSCTR\_IP] = sysctr\_populate\_request, //6

[TSTMR\_IP] = tstmr\_populate\_request, //7

[LPWDOG\_IP] = lpwdog\_populate\_request, //8

[LPIT\_IP] = lpit\_populate\_request, //9

[LPTMR\_IP] = lptmr\_populate\_request, //10

[LPTPM\_IP] = lptpm\_populate\_request, //11

[I3C\_IP] = i3c\_populate\_request, //12

[LPI2C\_IP] = lpi2c\_populate\_request, //13

[LPSPI\_IP] = lpspi\_populate\_request, //14

[LPUART\_IP] = lpuart\_populate\_request, //15

[FLEXCAN\_IP] = flexcan3\_populate\_request, //16

[SAI\_IP] = sai\_populate\_request, //17

[IPC\_IP] = ipc\_populate\_request, //18

[M33\_IP] = m33\_populate\_request, //19

[M33\_SYSPM\_IP] = m33\_syspm\_populate\_request, //20

[M33\_TCM\_MECC\_IP] = m33\_tcm\_mecc\_populate\_request, //21

[ROMCP\_IP] = romcp\_populate\_request, //22

[BBNSM\_IP] = bbnsm\_populate\_request, //23

[MTR\_MSTR\_IP] = mtr\_mstr\_populate\_request, //24

[MTR\_DCA\_IP] = mtr\_dca\_populate\_request, //25

[TCU\_IP] = tcu\_populate\_request, //26

[BASIC\_TROUT\_IP] = basic\_trout\_populate\_request, //27

[AXBS\_IP] = axbs\_populate\_request, //65

[DCDC\_IP] = dcdc\_populate\_request, //29

[FLEXSPI\_IP] = flexspi\_populate\_request, //30

[XRIOCU\_IP] = xriocu\_populate\_request, //31

[MOTOR\_IP] = motor\_populate\_request, //32

[GPT\_IP] = gpt\_populate\_request, //33

[FLEXIO\_IP] = flexio\_populate\_request, //34

[ADC\_IP] = adc\_populate\_request, //35

[FLEXPWM\_IP] = flexpwm\_populate\_request, //36

[QTIMER\_IP] = qtimer\_populate\_request, //37

[ENC\_IP] = enc\_populate\_request, //38

[XBAR\_IP] = xbar\_populate\_request, //39

[AOI\_IP] = aoi\_populate\_request, //40

[EWM\_IP] = ewm\_populate\_request, //41

[GPV\_MEGA\_IP] = gpv\_mega\_populate\_request, //42

[USDHC\_IP] = usdhc\_populate\_request, //43

[MSGINTR\_IP] = msgintr\_populate\_request, //44

[XSPI\_SLV\_IP] = xspi\_slv\_populate\_request, //45

[SEMC\_IP] = semc\_populate\_request, //46

[OCRAM\_MECC\_IP] = ocram\_mecc\_populate\_request, //47

[ASRC\_IP] = asrc\_populate\_request, //48

[KPP\_IP] = kpp\_populate\_request, //49

[ECAT\_IP] = ecat\_populate\_request, //50

[SPDIF\_IP] = spdif\_populate\_request, //51

[MICFIL\_IP] = micfil\_populate\_request, //52

[SINC\_IP] = sinc\_populate\_request, //53

[USB\_IP] = usb\_populate\_request, //54

[USB\_PHY\_IP] = usb\_phy\_populate\_request, //55

[ACMP\_IP] = acmp\_populate\_request, //56

[DAC\_IP] = dac\_populate\_request, //57

[VREF\_IP] = vref\_populate\_request, //58

[IEE\_IP] = iee\_populate\_request, //59

[SPTP\_IP] = sptp\_populate\_request, //60

[SERDES\_IP] = serdes\_populate\_request, //61

[SECSUBSYS\_IP] = secsubsys\_populate\_request, //62

[NVIC\_IP] = nvic\_populate\_request, //63

[LPCAC\_IP] = lpcac\_populate\_request, //64

[NETC\_IP] = netc\_populate\_request, //66

[XCEL\_IP] = xcel\_populate\_request, //67

[GPIO\_IP] = gpio\_populate\_request, //68

[SECSUBSYS\_FSB\_IP]= s400\_fsb\_populate\_request, //69

[SEMA42\_IP] = sema42\_populate\_request //70

};

### dma\_populate\_request

.\testbench\ blocks\soc\_tb\tool\_data\compiler\include\dma\_contexti.h

#ifdef DMA\_ID

void dma\_populate\_request(ip\_request\_t\* ip\_request){

ip\_request->init\_context = dma\_init\_context; [dma\_init\_context](#_dma_init_context)

ip\_request->context\_id = DMA\_ID;

}

#else

#define dma\_populate\_request NULL

#endif

## soc\_allocate

soc\_allocate\_ip(fu) void (ip\_request\_t \*) testbench/blocks/soc\_tb/tool\_data/compiler/include/context\_api.h

调用soc\_new\_request获取的初始化函数，进行IP的初始化

void soc\_allocate\_ip(ip\_request\_t\* ip\_request){

if(!ip\_request->init\_context)

FATAL(CONTEXT\_API\_HEADER,"init\_context function must be provided !!!");

ip\_request->init\_context(ip\_request); [dma\_populate\_request](#_dma_populate_request)

if(ip\_request->skip\_configure)

return;

#ifndef SKIP\_CLOCK\_CFG

if(ip\_request->clock\_config)

ip\_request->clock\_config(ip\_request);

#endif

#ifndef SKIP\_RESET\_CFG

if(ip\_request->reset\_negate)

ip\_request->reset\_negate(ip\_request);

#endif

#ifndef SKIP\_PIN\_CFG

if(ip\_request->pin\_config)

ip\_request->pin\_config(ip\_request);

#endif

#ifndef SKIP\_TRIG\_CFG

if(ip\_request->trig\_config)

ip\_request->trig\_config(ip\_request);

#endif

// Interrupts

// Resource Allocation

}

void soc\_allocate\_ip(ip\_request\_t\* ip\_request){

if(!ip\_request->init\_context)

FATAL(CONTEXT\_API\_HEADER,"init\_context function must be provided !!!");

ip\_request->init\_context(ip\_request);

if(ip\_request->clock\_config)

ip\_request->clock\_config(ip\_request);

if(ip\_request->trig\_config)

ip\_request->trig\_config(ip\_request);

// Interrupts`

// Resource Allocation

}

### dma\_init\_context

.\testbench\ blocks\soc\_tb\tool\_data\compiler\include\dma\_contexti.h

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Function:

//

// Initialize and return context pointer associated with ip\_request->context\_id

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void dma\_init\_context(ip\_request\_t\* ip\_request){

dma\_inst\_id\_e id = (dma\_inst\_id\_e)ip\_request->context\_id;

create\_context\_dma\_inst(id, (dma\_context\_t \*\*)(&ip\_request->context\_ptr));

ip\_request->clock\_config = dma3\_pcc\_assign;

ip\_request->clock\_disable = dma3\_pcc\_release;

}

### create\_context\_dma\_inst

.\testbench\ blocks\soc\_tb\tool\_data\compiler\include\dma\_contexti.h

void create\_context\_dma\_inst(dma\_inst\_id\_e inst\_num, dma\_context\_t\*\* context){

static dma\_context\_t static\_context;

DMA3\_CONTEXT\* dma3 = NULL;

create\_context\_dma3(inst\_num, &dma3);

static\_context.DMAC\_CONTEXT = dma3;

static\_context.DMA\_MUX\_CONTEXT = NULL;

static\_context.LOGICAL\_CHANNEL = -1;

static\_context.PHYSICAL\_CHANNEL = -1;

static\_context.ALWAYS\_EN = 0;

\*context = &static\_context;

}

### create\_context\_dma3

.\testbench\common\_blocks\dma\ context\_dma3.h

给DMA3\_CONTEXT结构体进行赋值

#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; \

}

PP\_REPEAT(DMA\_INSTS, CREATE\_CONTEXT\_DMA3\_INST)

#define DMA\_CREATE\_CONTEXT\_CASE(num) case DMA3\_INST##num: create\_context\_dma3\_inst##num(context); break;

void create\_context\_dma3(dma\_inst\_id\_e inst\_num, DMA3\_CONTEXT\*\* context)

{

switch (inst\_num){

PP\_REPEAT(DMA\_INSTS, DMA\_CREATE\_CONTEXT\_CASE)

default:

SHOW8(CONTEXT\_DMA3\_HEADER,"DMA\_INST: ", (int)inst\_num);

FATAL(CONTEXT\_DMA3\_HEADER,"DMA DOES NOT EXIST");

}

}

#### PP\_REPEAT如何获取IP属性

DMA为例：

1. CREATE\_CONTEXT\_DMA3\_INST 函数中会给DMA3\_CONTEXT进行赋值，传参是DMA的inst

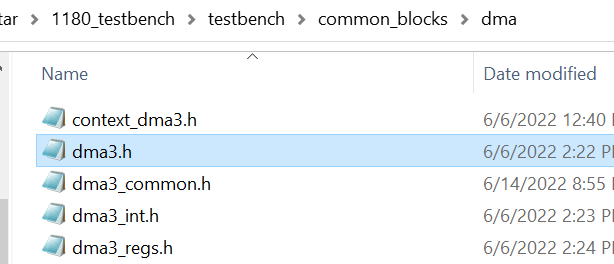
static\_context.REGS\_BASE = DMA##num##\_RBASE;

1. 如果如上num为1则dma3\_context->REGS\_BASE赋值为 DMA1\_RBASE
2. /testbench/blocks/soc\_tb/tool\_data/sray/mem\_map/dma\_params.mem\_map:71:`define DMA1\_RBASE `EDMA1\_CH0\_RBASE
3. /testbench/blocks/soc\_tb/tool\_data/sray/mem\_map/aips\_cm33\_sec.mem\_map:282:`define EDMA1\_CH0\_RBASE `CM33\_EDMA1\_CH0\_RBASE // AIPS1 slot 0
4. ./testbench/blocks/soc\_tb/tool\_data/sray/mem\_map/aips\_cm33\_sec.mem\_map:5:`define CM33\_EDMA1\_CH0\_RBASE 'h54000000 // AIPS1 slot 0

DMA register定义：

.\testbench\common\_blocks\dma\ dma3.h

DMA3\_CONTEXT的赋值函数在.\testbench\common\_blocks\dma\ context\_dma3.h



DMA register base addr, auto generated from xlsx file

./testbench/blocks/soc\_tb/tool\_data/sray/mem\_map/aips\_cm33\_sec.mem\_map

## soc\_get\_context\_pointer

IP初始化成功后，就可以获取IP相关的属性参数。

//Function to get the pointer to the context

void\* soc\_get\_context\_pointer(ip\_request\_t\* ip\_request)

{

return ip\_request->context\_ptr;

}

dma\_context = (dma\_context\_t\*)(soc\_get\_context\_pointer(&ip\_request));

dma3\_context = (DMA3\_CONTEXT\*)(dma\_context->DMAC\_CONTEXT);

SHOW32("","Base Address : ",dma3\_context->REGS\_BASE);

SHOW32("","Channel Count: ",dma3\_context->NCH);

SHOW32("","APB\_SLOT\_SIZE: ",dma3\_context->APB\_SLOT\_SIZE);

### ip\_request\_t

testbench/blocks/soc\_tb/tool\_data/compiler/include/ip\_request.h

// This is a SoC specific function. Future implementations may need to initialize more elements

typedef struct ip\_request\_t ip\_request\_t;

struct ip\_request\_t{

ip\_type\_e ip\_type;

int context\_id;

void\* context\_ptr;

pinmux\_alt\_e pinmux\_opt;

uint8\_t skip\_configure;

void (\*init\_context)(ip\_request\_t\* ip\_request);

void (\*clock\_config)(ip\_request\_t\* ip\_request);

void (\*clock\_disable)(ip\_request\_t\* ip\_request);

void (\*pin\_config)(ip\_request\_t\* ip\_request);

void (\*pin\_disable)(ip\_request\_t\* ip\_request);

void (\*trig\_config)(ip\_request\_t\* ip\_request);

void (\*reset\_assert)(ip\_request\_t\* ip\_request);

void (\*reset\_negate)(ip\_request\_t\* ip\_request);

};

#endif

### dma\_context\_t

/home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/blocks/soc\_tb/tool\_data/compiler/include/ dma\_context.h

//------------------------------------------------------------------------------

// SoC-specific DMA Controller Context

// -----------------------------------------------------------------------------

typedef struct

{ void\* DMAC\_CONTEXT;

void\* DMA\_MUX\_CONTEXT;

uint8\_t LOGICAL\_CHANNEL;

uint8\_t PHYSICAL\_CHANNEL;

uint8\_t ALWAYS\_EN;

} dma\_context\_t;

### DMA3\_CONTEXT

/home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/common\_blocks/v\_ms\_spp\_dma3\_rt\_vip/tool\_data/compiler/include/dma3.h

// DMA3 Context struct

// Note: Variables in ALL\_CAPS are for the SoC team to fill in before calling

// any of the functions, and should be considered constants.

// Variables in lower\_case are private variables for the functions to use.

typedef struct {

char\* NAME;

// ===============Parameter Info of IP================

uint8\_t NCH;

uint8\_t BUS\_WIDTH;

vuint8\_t ENB\_MID\_REPL;

vuint8\_t ASW;

vuint8\_t ENB\_CH\_BUFFWR;

vuint8\_t ENB\_MP\_BUFFWR;

vuint8\_t AHB\_MASTER\_ID;

vuint8\_t COMBINED\_IRQ;

vuint8\_t ENB\_MP\_INT;

vuint8\_t NHR;

vuint8\_t DMA\_VERSION;

vuint8\_t DMA\_ENB\_SW\_SECURITY\_CTRL;

vuint8\_t AHB\_PAL\_DEFAULT;

vuint8\_t AHB\_SEC\_DEFAULT;

// ===============BASE Info===========================

address\_t REGS\_BASE;

uint32\_t APB\_SLOT\_SIZE;

tbcomm\_t TBCOMM\_BASE;

// ===============Interrupt===========================

vector\_t INTERRUPT\_VECTOR;

vector\_t TEA\_INT\_VECTOR;

vector\_t ERROR\_INTERRUPT\_VECTOR;

vuint8\_t interrupt\_cnt;

vuint8\_t tea\_int\_cnt;

vuint8\_t err\_int\_cnt;

vuint32\_t last\_logged\_error;

// ==================DMA==============================

uint32\_t INUSE;

vuint8\_t eop\_enable;

} DMA3\_CONTEXT;

## dma3\_reg\_check\_reset

// TCD reset values are not tested, due to they are from sram

void dma3\_reg\_check\_reset(DMA3\_CONTEXT\* dma3\_context)

{

uint32\_t channel, sbr\_reset\_val;

sbr\_reset\_val = (dma3\_context->DMA\_ENB\_SW\_SECURITY\_CTRL)? ((dma3\_context->AHB\_PAL\_DEFAULT<<15)|(dma3\_context->AHB\_SEC\_DEFAULT<<14)|dma3\_context->AHB\_MASTER\_ID):((dma3\_context->AHB\_PAL\_DEFAULT<<15)|dma3\_context->AHB\_MASTER\_ID);

SHOW32("sbr\_reset\_val is", " 0x", sbr\_reset\_val);

INFO("Start of ", "Checking reset register values");

RE32(REG32(DMA3\_MP\_CSR),DMA3\_MP\_CSR\_RESET |(dma3\_context->DMA\_VERSION <<16));

RE32(REG32(DMA3\_MP\_ES),DMA3\_MP\_ES\_RESET);

if (dma3\_context->ENB\_MP\_INT) {

RE32(REG32(DMA3\_MP\_INT), DMA3\_MP\_INT\_RESET);

}

if(DMA\_ID==INST1)

RE32(REG32(DMA3\_MP\_HRS),DMA3\_MP\_HRS\_RESET);//the INST2'S HRS is reserved;

SHOW8("DMA CHANNEL count is ", " 0x", dma3\_context->NCH);

for (channel=0; channel<dma3\_context->NCH; channel++)

RE32(DMA3\_GRPRI(channel), DMA3\_CH\_GRPRI\_RESET);

for (channel=0; channel<dma3\_context->NCH; channel++)

{

SHOW8("DMA3 CHANNEL index is ", " 0x", channel);

RE32(DMA3\_CH\_CSR(channel), DMA3\_CH\_CSR\_RESET);

RE32(DMA3\_CH\_ES(channel), DMA3\_CH\_ES\_RESET);

RE32(DMA3\_CH\_INT(channel), DMA3\_CH\_INT\_RESET);

RE32(DMA3\_CH\_SBR(channel), sbr\_reset\_val);

RE32(DMA3\_CH\_PRI(channel), DMA3\_CH\_PRI\_RESET);

if (dma3\_context->NHR>dma3\_context->NCH) {

RE32(DMA3\_CH\_MUX(channel), DMA3\_CH\_MUX\_RESET);

}

}

}

### RE32

//============================================================================

// Read & Compare - 32 bits

// Read a value from address and print an error message if the value read

// does not match expected\_data. Otherwise, print the read operation.

//============================================================================

void CAPI\_Base::RE32 (U32 address, U32 expected\_data)

{

U32 destvar;

isr\_escape();

stdtsk\_quiet++; // suppress messages from base function

R32(address, destvar);

stdtsk\_quiet--;

if (destvar == expected\_data)

{

print\_portable\_task\_msg("RE32", address, DONT\_CARE\_Z, (U64)stdtsk\_read\_value,

(U64)expected\_data, DONT\_CARE\_Z, STDTSK\_SZ\_32,

stdtsk\_current\_function\_code,

(STDTSK\_SUCCESS | STDTSK\_PRNT\_READ | STDTSK\_PRNT\_EXP));

}

else

{

print\_portable\_task\_msg("RE32", address, DONT\_CARE\_Z, (U64)stdtsk\_read\_value,

(U64)expected\_data, DONT\_CARE\_Z, STDTSK\_SZ\_32,

stdtsk\_current\_function\_code,

(STDTSK\_FAIL | STDTSK\_PRNT\_READ | STDTSK\_PRNT\_EXP));

}

}

## dma3\_reg\_check\_complement\_32bit

dma3\_reg\_check\_complement\_8bit(fu) void (DMA3\_CONTEXT \*,uint16\_t) testbench/common\_blocks/v\_ms\_spp\_dma3\_rt\_vip/tool\_data/compiler/include/dma3\_regs.h

// Method used for checking DMA registers R/W by 32bit access

void dma3\_reg\_check\_complement\_32bit(DMA3\_CONTEXT\* dma3\_context,uint32\_t wr\_val)

{

uint8\_t channel;

INFO("Start of ", "Checking rw register values 32 bit access");

WRE32(REG32(DMA3\_MP\_CSR),wr\_val,(wr\_val & (DMA3\_MP\_CSR\_MASK | dma3\_context->ENB\_MP\_BUFFWR))|(dma3\_context->DMA\_VERSION <<16));

WRE32(REG32(DMA3\_MP\_ES),wr\_val,wr\_val & DMA3\_MP\_ES\_MASK);

if (dma3\_context->ENB\_MP\_INT) {

WRE32(REG32(DMA3\_MP\_INT),wr\_val,0x00000000);

}

if(DMA\_ID==INST1)

WRE32(REG32(DMA3\_MP\_HRS),wr\_val,wr\_val & DMA3\_MP\_HRS\_MASK);

SHOW8("DMA CHANNEL count is ", " 0x", dma3\_context->NCH);

for (channel=0; channel<dma3\_context->NCH; channel++)

if(DMA\_ID==INST1)

WRE32(DMA3\_GRPRI(channel), wr\_val, wr\_val & DMA3\_CH\_GRPRI\_MASK);

else

WRE32(DMA3\_GRPRI(channel), wr\_val, wr\_val & 0x3f);//this is because the dma4 has six bit

for (channel=0; channel<dma3\_context->NCH; channel++)

{

SHOW8("DMA3 CHANNEL index is ", " 0x", channel);

if(DMA\_ID==INST1)

WRE32(DMA3\_CH\_CSR(channel), wr\_val, wr\_val & (DMA3\_CH\_CSR\_MASK | dma3\_context->ENB\_CH\_BUFFWR<<3));

else{

SHOW8("channel for debug ", " 0x", channel);

WRE32(DMA3\_CH\_CSR(channel), wr\_val, wr\_val & DMA3\_CH\_CSR\_MASK\_INST2);

}

WRE32(DMA3\_CH\_ES(channel), wr\_val, wr\_val & DMA3\_CH\_ES\_MASK);

WRE32(DMA3\_CH\_INT(channel), wr\_val, wr\_val & DMA3\_CH\_INT\_MASK);

WRE32(DMA3\_CH\_PRI(channel), wr\_val, wr\_val & DMA3\_CH\_PRI\_MASK);

if (dma3\_context->NHR>dma3\_context->NCH) {

WRE32(DMA3\_CH\_MUX(channel), dma3\_context->NHR-1 , dma3\_context->NHR-1);

WRE32(DMA3\_CH\_MUX(channel), 0, 0);

}

WRE32(DMA3\_TCD\_SADDR(channel), wr\_val, wr\_val & DMA3\_TCD\_SADDR\_SADDR\_MASK);

WRE32(DMA3\_TCD\_SOFF(channel), wr\_val, wr\_val &((DMA3\_TCD\_ATTR\_MASK<<16)|(DMA3\_TCD\_SOFF\_SOFF\_MASK)));

WRE32(DMA3\_TCD\_NBYTES(channel), wr\_val, wr\_val & DMA3\_TCD\_NBYTES\_MASK);

WRE32(DMA3\_TCD\_SLAST(channel), wr\_val, wr\_val & DMA3\_TCD\_SLAST\_MASK);

WRE32(DMA3\_TCD\_DADDR(channel), wr\_val, wr\_val & DMA3\_TCD\_DADDR\_DADDR\_MASK);

WRE32(DMA3\_TCD\_DOFF(channel), wr\_val, wr\_val &((DMA3\_TCD\_CITER\_MASK<<16)|(DMA3\_TCD\_DOFF\_DOFF\_MASK)));

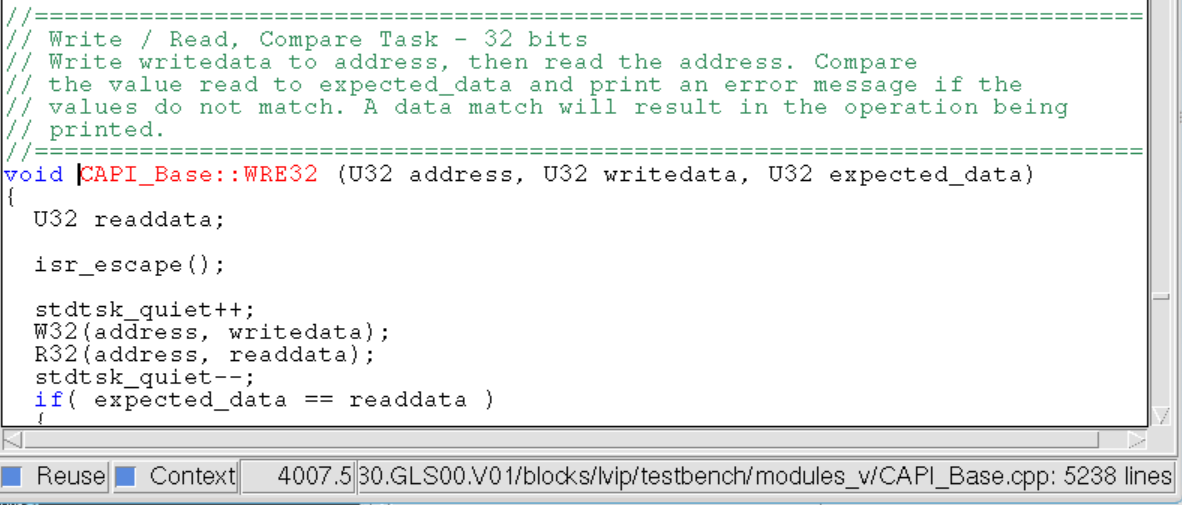
WRE32(DMA3\_TCD\_DLAST(channel), wr\_val, wr\_val & DMA3\_TCD\_DLAST\_DLAST\_MASK);

WRE32(DMA3\_TCD\_CSR(channel), wr\_val & 0xFFFFFFFE, wr\_val & 0xFFFFFFFE &((DMA3\_TCD\_BITER\_MASK<<16)|(DMA3\_TCD\_CSR\_MASK)));

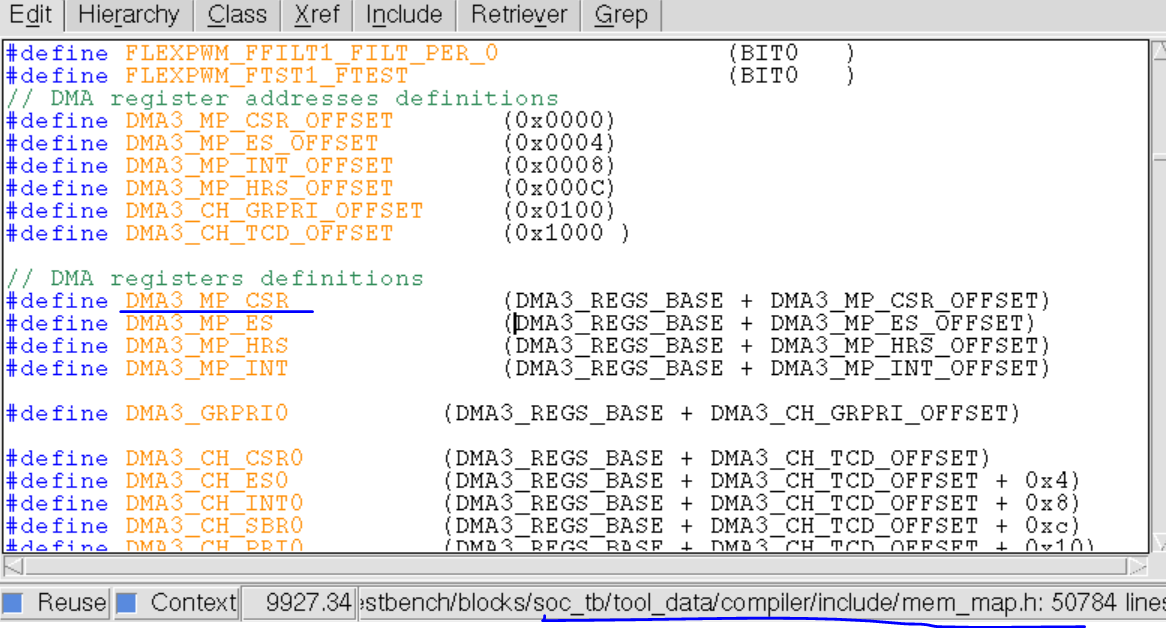
}

}

### WRE32



## Register



## 中断工作流程

### dma3\_setup\_int(dma3\_context);

配置中断服务函数

#### dma3\_int\_isr

//DMA3 interrupt service routine

void dma3\_int\_isr(DMA3\_CONTEXT\* dma3\_context) {

uint32\_t channel\_status, channel\_csr;

uint32\_t channel\_tcd\_csr;

uint8\_t channel\_index;

// Clear interrupt

R32(REG32(DMA3\_MP\_CSR), channel\_status);

channel\_index = (channel\_status&DMA3\_MP\_CSR\_CHID\_MASK)>>24;

SHOW8(dma3\_context->NAME, "dma3\_int\_isr: severed channel: 0x", channel\_index);

R32(DMA3\_CH\_CSR(channel\_index), channel\_csr);

if(channel\_csr&DMA3\_CH\_CSR\_DONE\_MASK)

{

BSET32(DMA3\_CH\_CSR(channel\_index),DMA3\_CH\_CSR\_DONE\_MASK);

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

++(dma3\_context->interrupt\_cnt);

INFO(dma3\_context->NAME, "dma3\_int\_isr: dma done interrupt serviced");

} else {

R32(DMA3\_TCD\_CSR(channel\_index), channel\_tcd\_csr);

if(channel\_tcd\_csr&DMA3\_TCD\_CSR\_INTHALF\_MASK)

{

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

++(dma3\_context->interrupt\_cnt);

INFO(dma3\_context->NAME, "dma3\_int\_isr: dma half done interrupt serviced");

}

}

} // dma3\_int\_isr

### dma3\_setup\_hw

dma3\_setup\_hw(dma3\_context,i,(address\_t)source, (address\_t)dest,4,4,4,4,16,2);

void dma3\_setup\_hw(DMA3\_CONTEXT\* dma3\_context,

uint8\_t channel,

uint32\_t source,

uint32\_t destination,

uint8\_t source\_size, //4

uint8\_t destination\_size, //4

int16\_t source\_offset, //4

int16\_t destination\_offset //4,

uint32\_t bytes\_per\_transfer, //16

uint16\_t iterations) //2

{

//fix setup\_hw with fixed ID, remote read/write

uint32\_t proc\_id = 0;

uint32\_t local\_read = 0;

uint32\_t local\_write = 0;

dma3\_setup\_complex(dma3\_context, // context

channel, // channel

source , // source

0, // source\_modulo

source\_size, // source\_size

0, // destination\_modulo

destination\_size, // destination\_size

source\_offset, // source\_offset

bytes\_per\_transfer, // bytes\_per\_transfer

0, // source\_adjustment

destination, // destination

0, // link

0, // link\_channel

iterations, // iterations

destination\_offset, // destination\_offset

0, // destination\_adjustment

0x0008|(dma3\_context->eop\_enable<<6), // control status - d\_req = 1 - disable HW request once iterations are complete

local\_read,

local\_write,

proc\_id); //process id to use

}

#### dma3\_setup\_complex

配置channel寄存器

void dma3\_setup\_complex(DMA3\_CONTEXT\* dma3\_context,

uint8\_t channel,

uint32\_t source,

uint8\_t source\_modulo,

uint8\_t source\_size, // bytes

uint8\_t destination\_modulo,

uint8\_t destination\_size, // bytes

int16\_t source\_offset,

uint32\_t bytes\_per\_transfer,

uint32\_t source\_adjustment,

uint32\_t destination,

uint8\_t link,

uint8\_t link\_channel,

uint16\_t iterations,

int16\_t destination\_offset,

uint32\_t destination\_adjustment,

int16\_t control\_status,

uint32\_t local\_read,

uint32\_t local\_write,

uint32\_t proc\_id)

{

// Word 0/1

WR32(DMA3\_TCD\_SADDR (channel), source);

WR16(DMA3\_TCD\_ATTR (channel), (source\_modulo << 11) | ssize | (destination\_modulo << 3) | dsize);

WR16(DMA3\_TCD\_SOFF (channel), source\_offset);

// Word 2/3

WR32(DMA3\_TCD\_NBYTES(channel), bytes\_per\_transfer);

WR32(DMA3\_TCD\_SLAST (channel), source\_adjustment);

// Word 4/5

WR32(DMA3\_TCD\_DADDR(channel), destination);

WR16(DMA3\_TCD\_CITER(channel), (link << 15) | (link\_channel << 9) | iterations);

WR16(DMA3\_TCD\_DOFF (channel), destination\_offset);

// Word 6/7

WR32(DMA3\_TCD\_DLAST(channel), destination\_adjustment);

WR16(DMA3\_TCD\_BITER(channel), (link << 15) | (link\_channel << 9) | iterations);

WR16(DMA3\_TCD\_CSR (channel), control\_status);

WRM32(DMA3\_CH\_SBR(channel), (local\_read<<22)|(local\_write<<21)|((proc\_id&0xF)<<17), (local\_read<<22)|(local\_write<<21)|((proc\_id&0xF)<<17) );

WRE32(DMA3\_CH\_CSR(channel), DMA3\_CH\_CSR\_DONE\_MASK|DMA3\_CH\_CSR\_ERQ\_MASK|DMA3\_CH\_CSR\_EARQ\_MASK|DMA3\_CH\_CSR\_EEI\_MASK, DMA3\_CH\_CSR\_ERQ\_MASK|DMA3\_CH\_CSR\_EARQ\_MASK|DMA3\_CH\_CSR\_EEI\_MASK);

// Send Channel and iterations to sv to do dma trans checking

MB\_PUT16(DMA3\_ITERATIONS\_MBOX, iterations);

# Register definition

### DMA3\_TCD\_SADDR

WR32(DMA3\_TCD\_SADDR (channel), source);

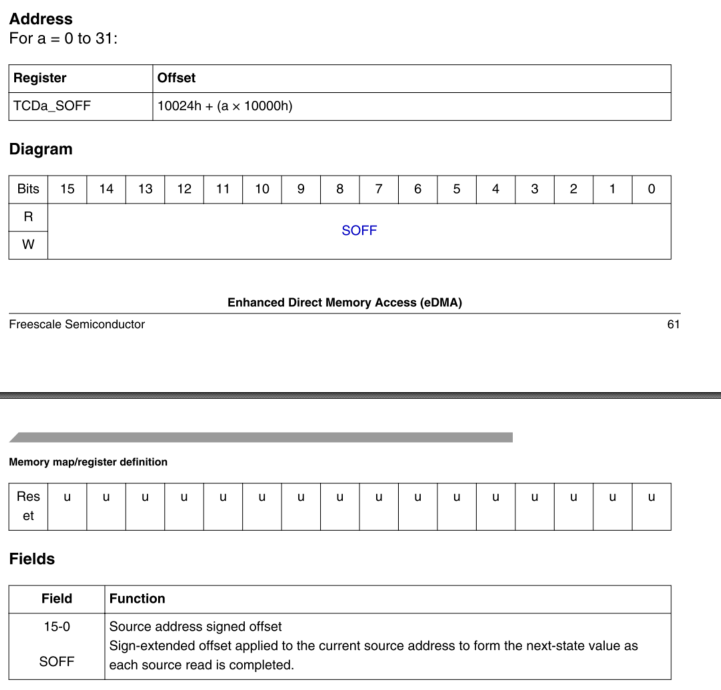
数据的源地址写入此寄存器



### DMA3\_TCD\_SOFF

相对于源地址的偏移量

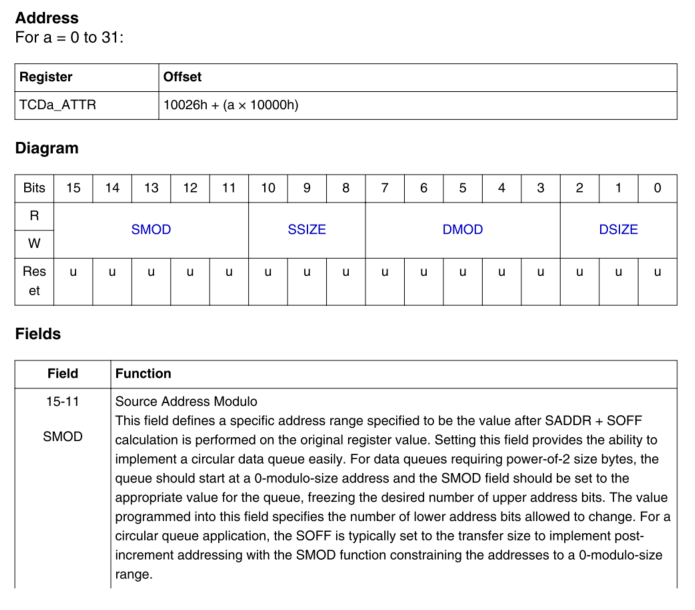
WR16(DMA3\_TCD\_SOFF (channel), source\_offset); //4

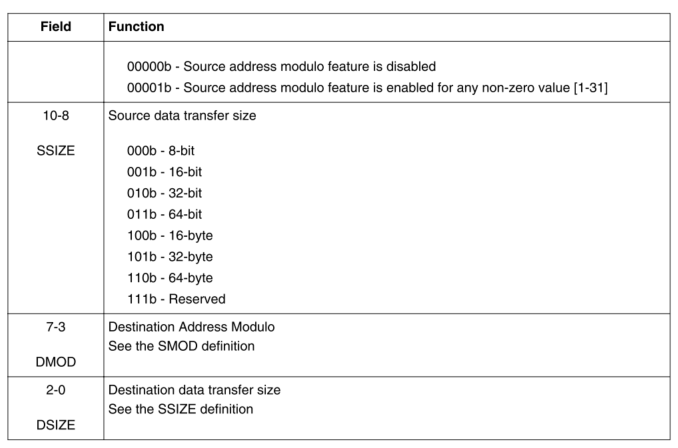


### DMA3\_TCD\_ATTR

WR16(DMA3\_TCD\_ATTR (channel), (source\_modulo << 11) | ssize | (destination\_modulo << 3) | dsize);

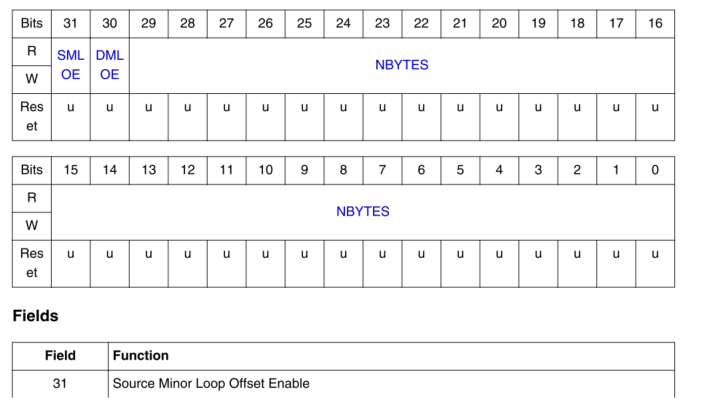
0<<11 | 0x200 | 0<<3 | 0x2 source /destination size transfer : 16bytes

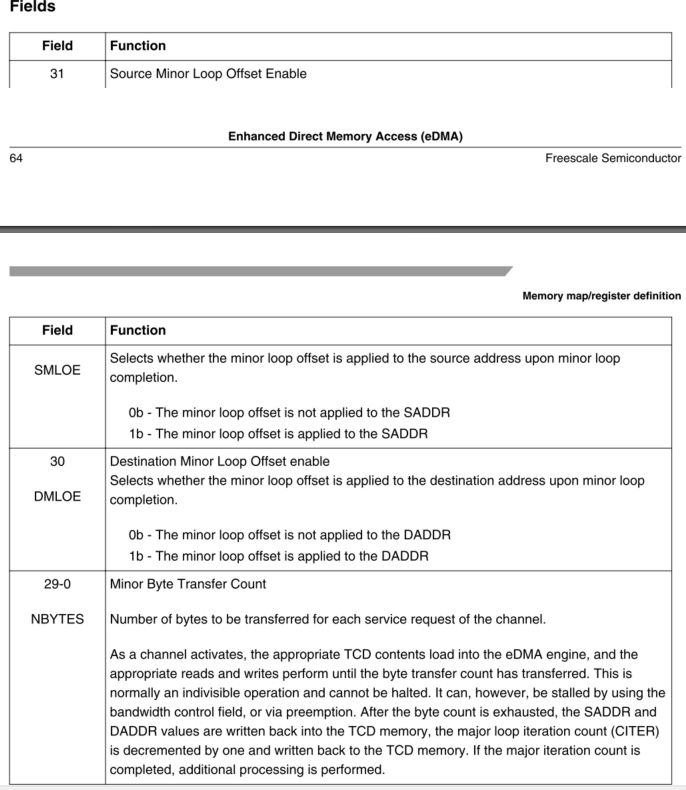




### DMA3\_TCD\_NBYTES

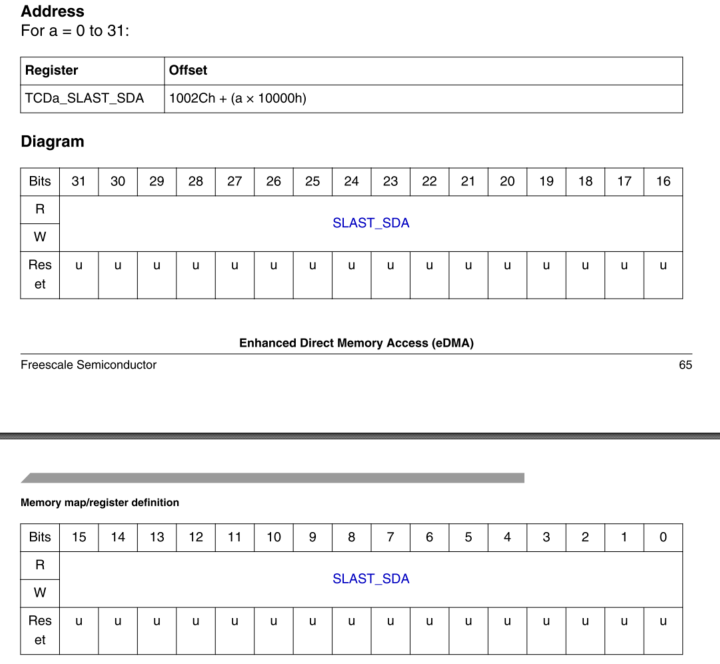
WR32(DMA3\_TCD\_NBYTES(channel), bytes\_per\_transfer); //16 Nbytes表示channel每次传输的字节数

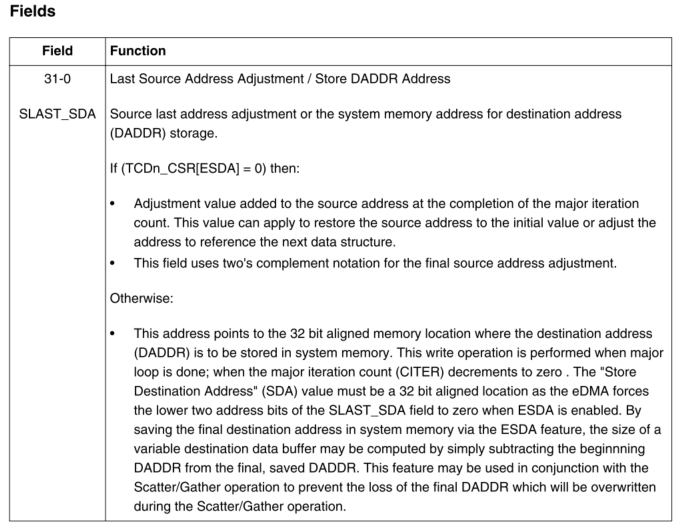




### DMA3\_TCD\_SLAST

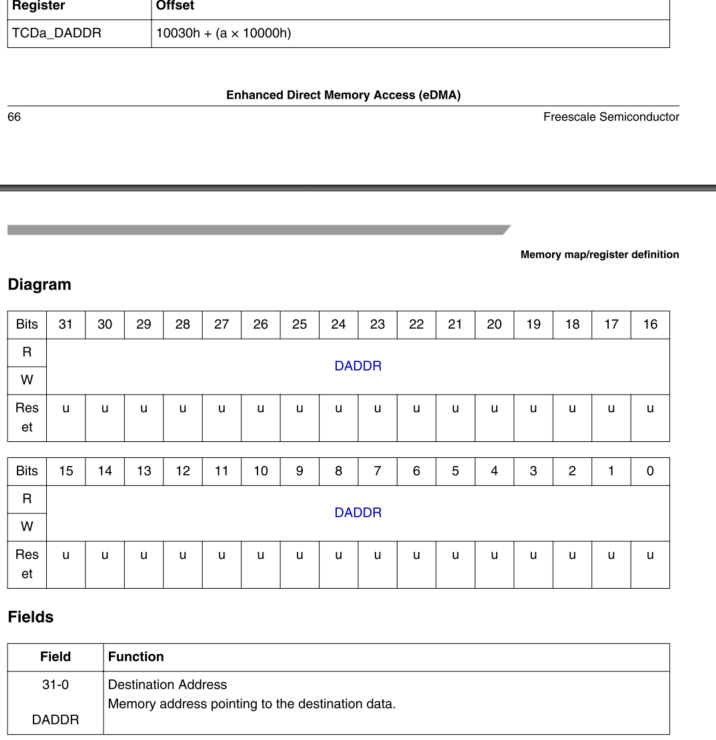
WR32(DMA3\_TCD\_SLAST (channel), source\_adjustment); //0





### DMA3\_TCD\_DADDR

WR32(DMA3\_TCD\_DADDR(channel), destination);



### DMA3\_TCD\_CITER

WR16(DMA3\_TCD\_CITER(channel), (link << 15) | (link\_channel << 9) | iterations);

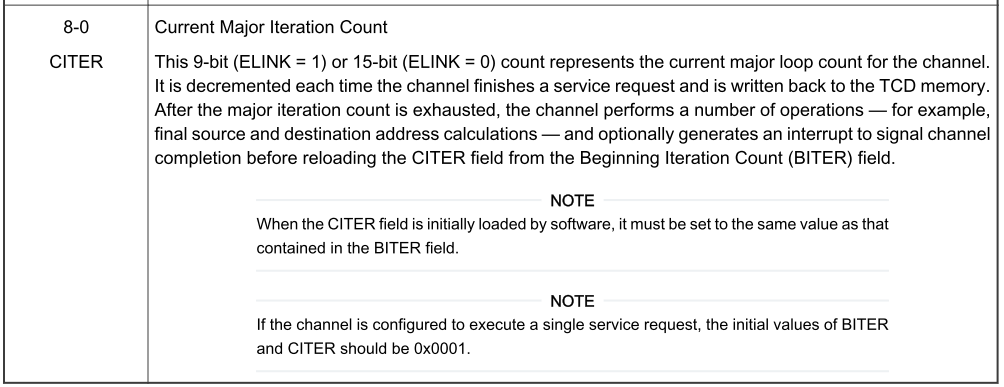
Interatios表示该channel 的major Loop次数

0<15|0<<9|2

如果bit15 ELINK=1，bit[13:9]当作LINKCH使用。

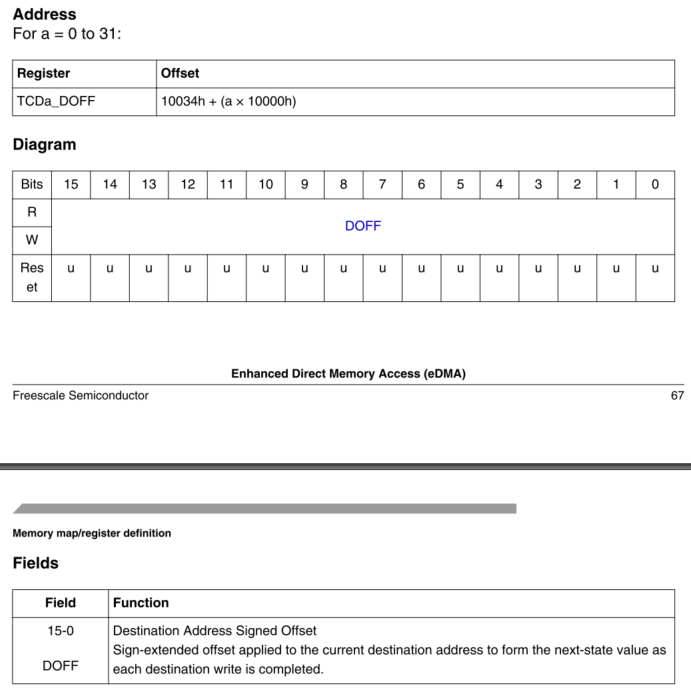
如果bit15 ELINK=0, bit[14:0]当作CITER使用





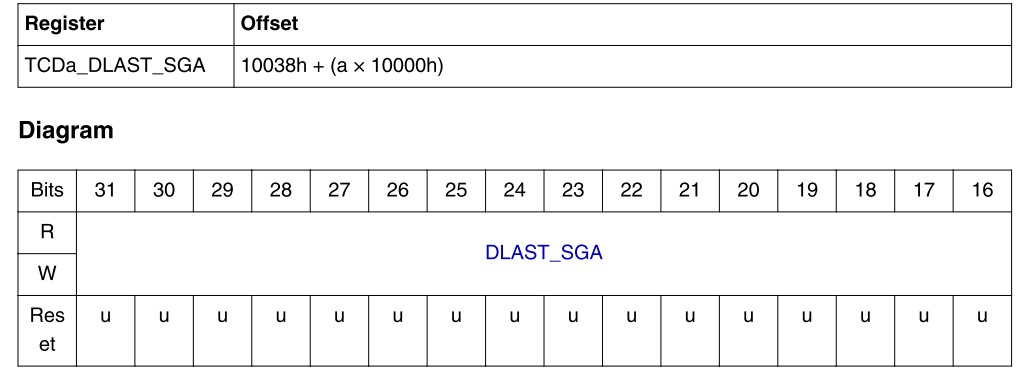
### DMA3\_TCD\_DOFF

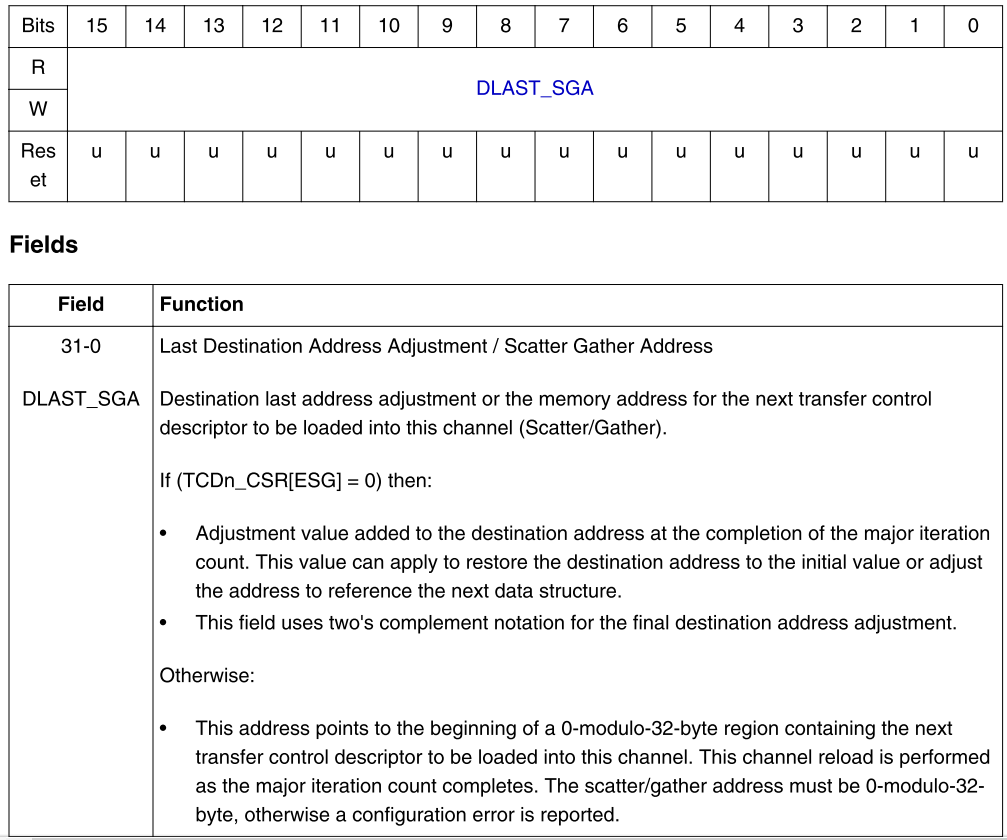
WR16(DMA3\_TCD\_DOFF (channel), destination\_offset); //4



### DMA3\_TCD\_DLAST

**WR32(DMA3\_TCD\_DLAST(channel), destination\_adjustment); //0**

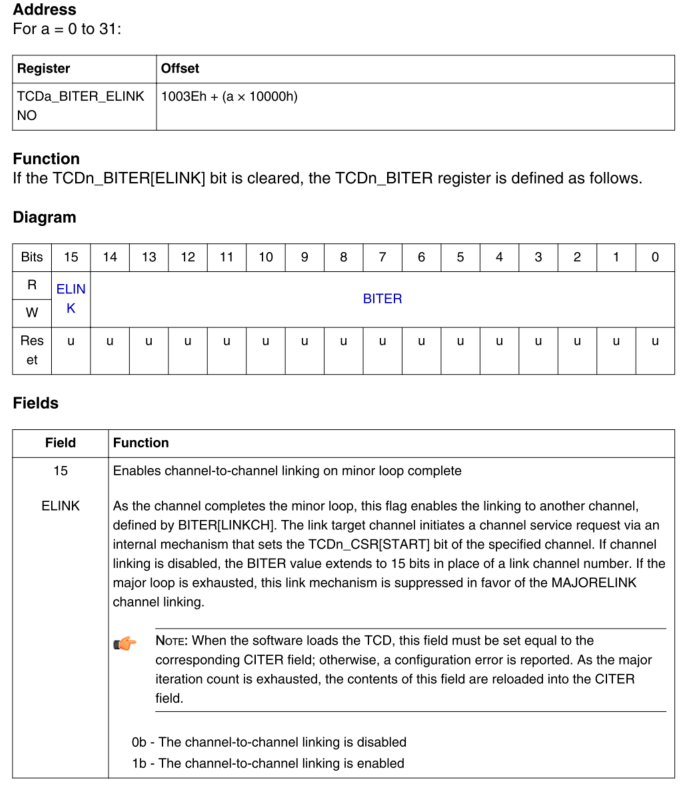


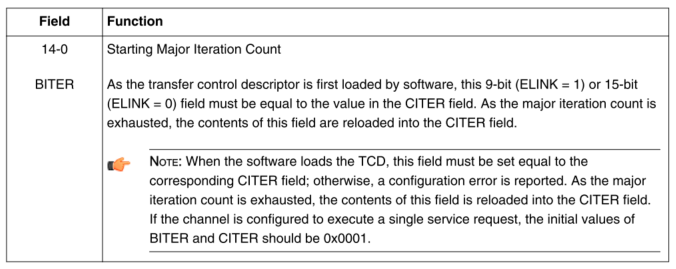


### DMA3\_TCD\_BITER

**WR16(DMA3\_TCD\_BITER(channel), (link << 15) | (link\_channel << 9) | iterations);**

**0<<15|0<<9|2**

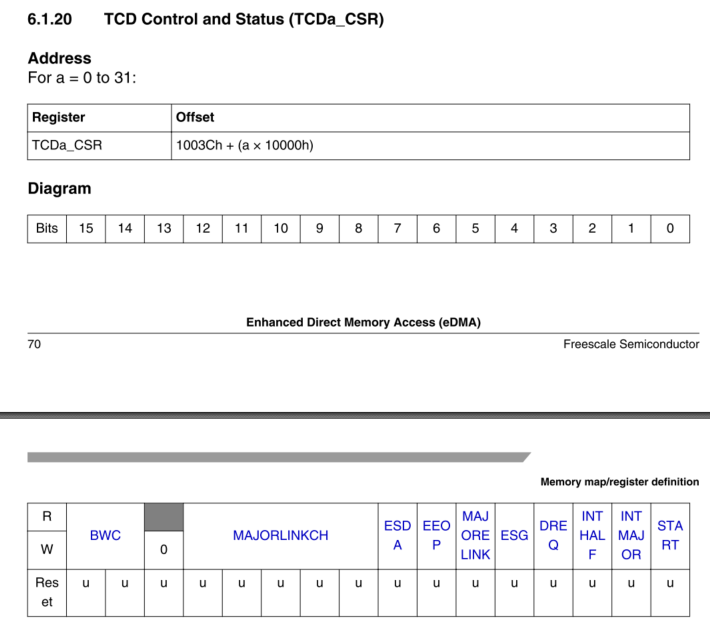


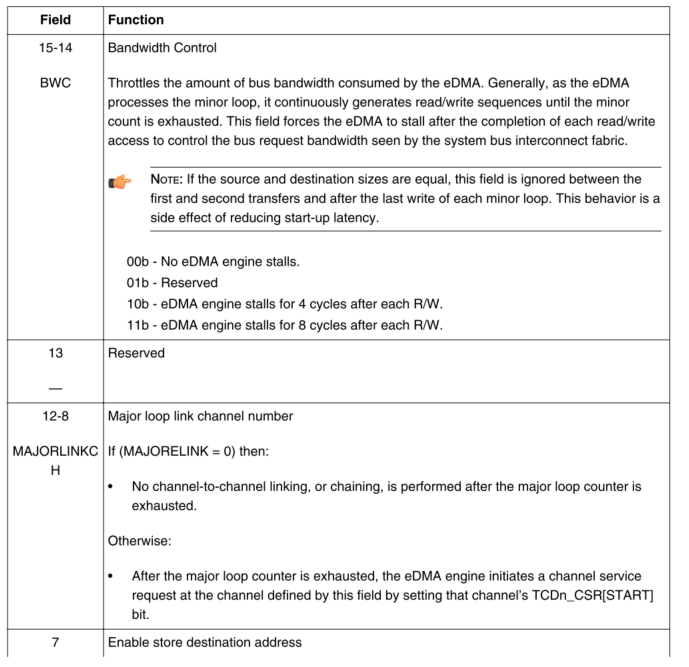


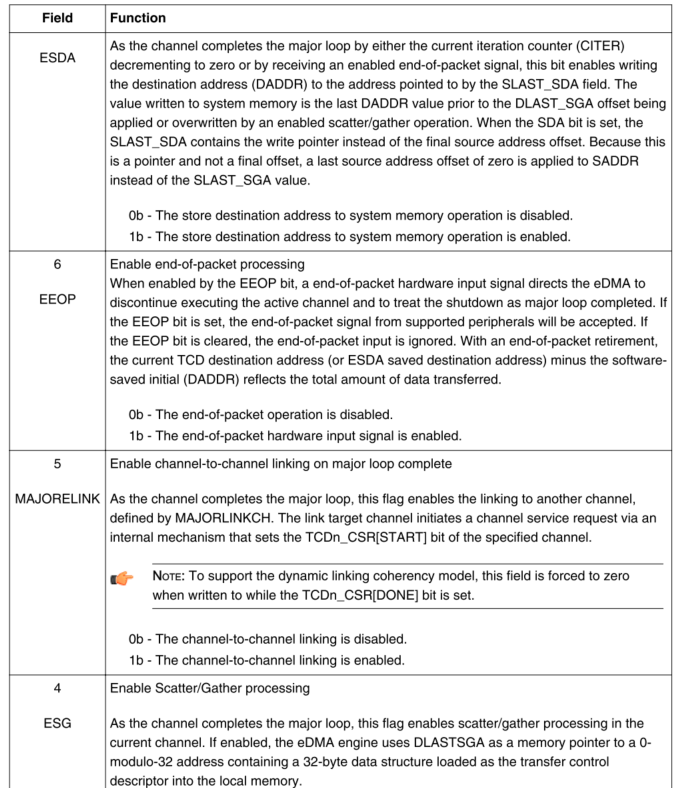
### DMA3\_TCD\_CSR

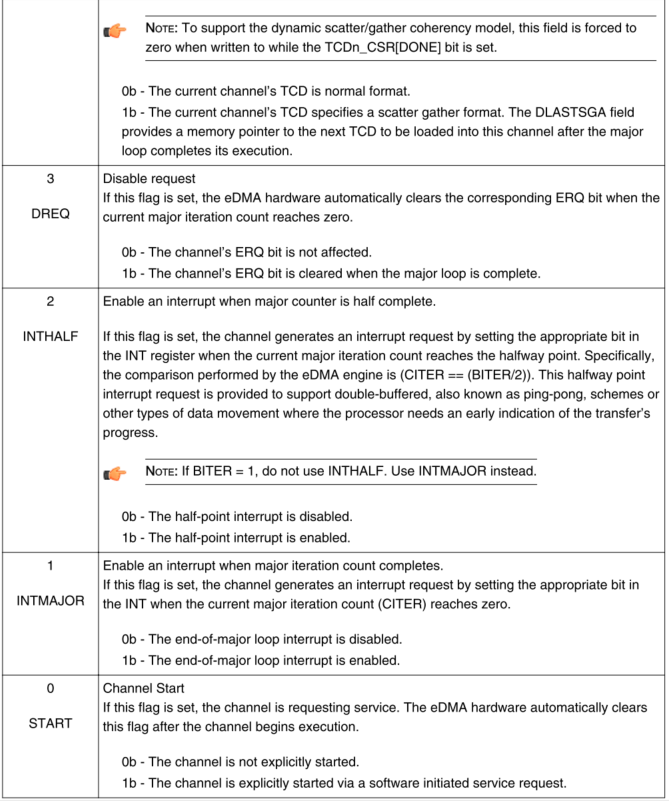
**WR16(DMA3\_TCD\_CSR (channel), control\_status);**

0x0008|(dma3\_context->eop\_enable<<6),







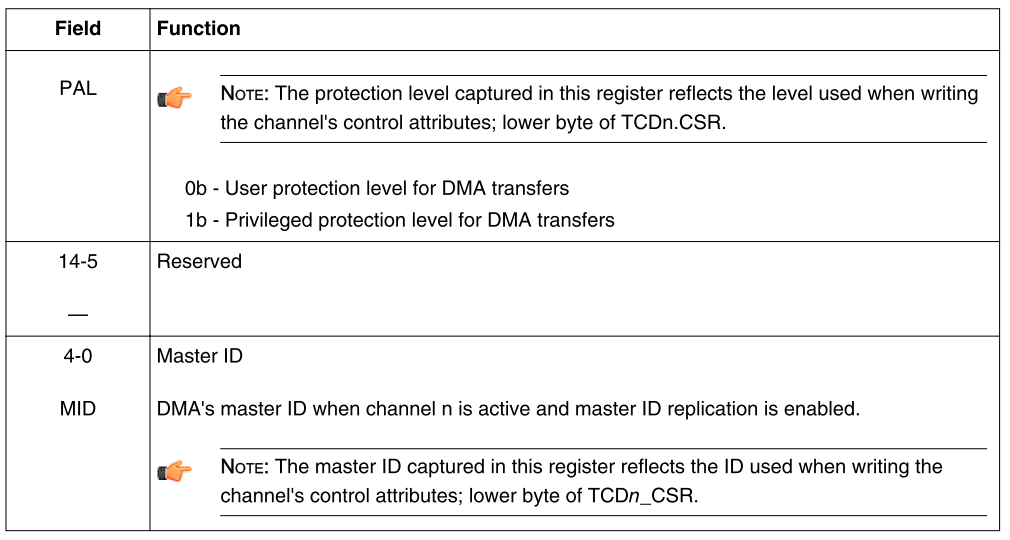


### DMA3\_CH\_SBR

**WRM32(DMA3\_CH\_SBR(channel), (local\_read<<22)|(local\_write<<21)|((proc\_id&0xF)<<17), (local\_read<<22)|(local\_write<<21)|((proc\_id&0xF)<<17) );**

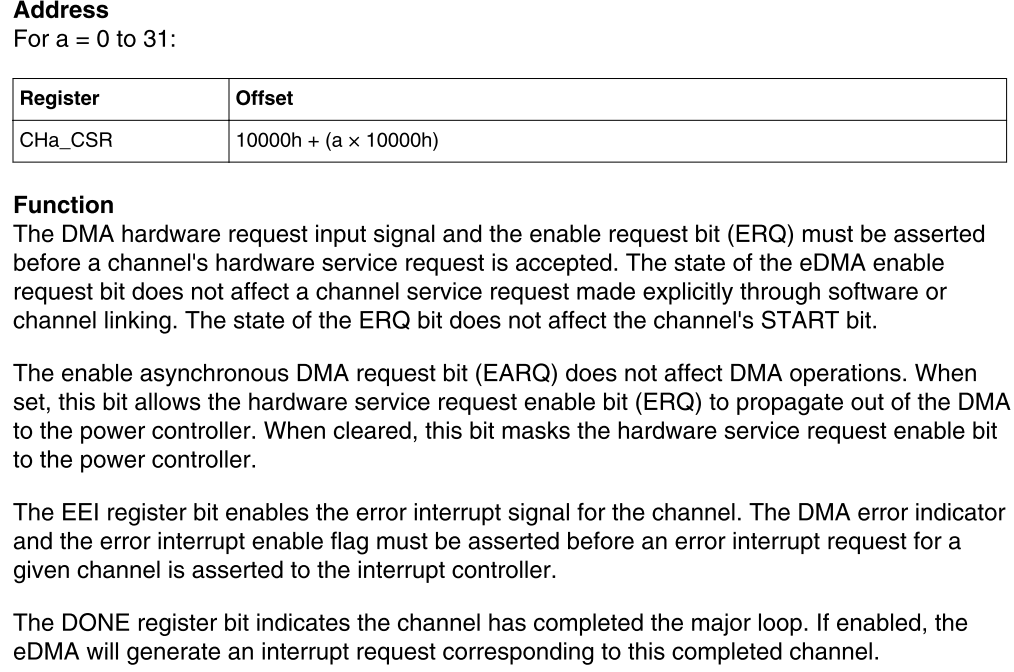
**0**





### DMA3\_CH\_CSR

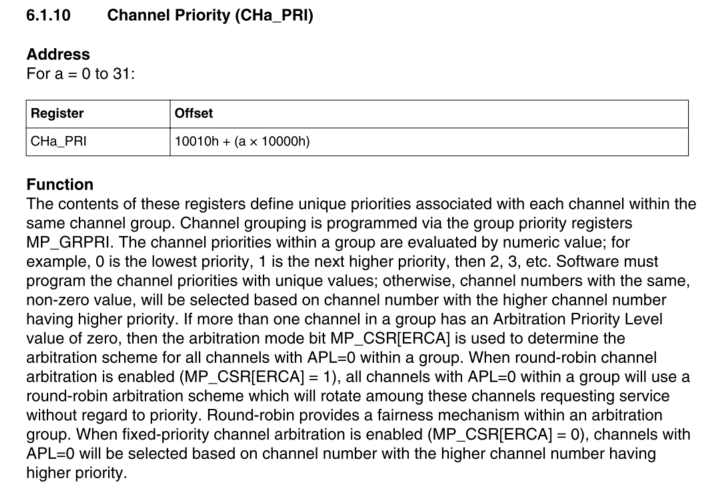
**WRE32(DMA3\_CH\_CSR(channel), DMA3\_CH\_CSR\_DONE\_MASK|DMA3\_CH\_CSR\_ERQ\_MASK|DMA3\_CH\_CSR\_EARQ\_MASK|DMA3\_CH\_CSR\_EEI\_MASK, DMA3\_CH\_CSR\_ERQ\_MASK|DMA3\_CH\_CSR\_EARQ\_MASK|DMA3\_CH\_CSR\_EEI\_MASK);**

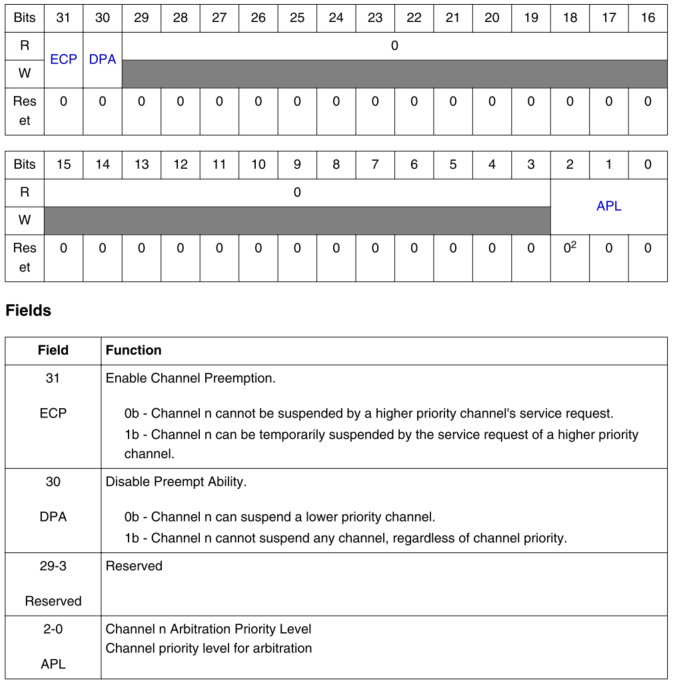




### DMA3\_CH\_PRI

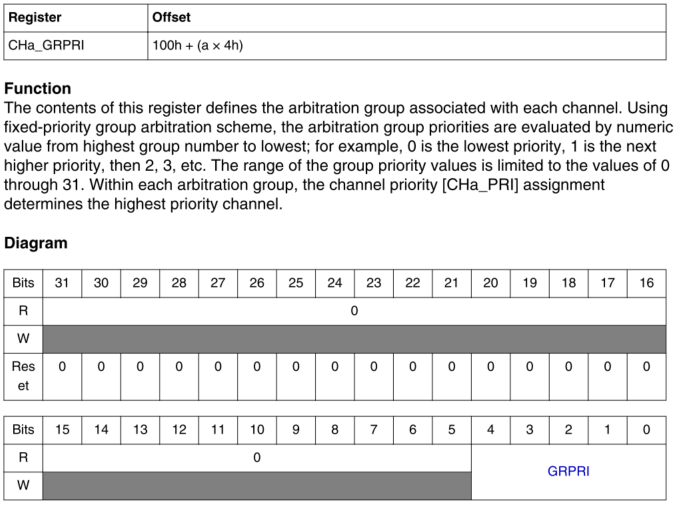
通道优先级设置， 0-7， 7最大





### DMA3\_GRPRI

设置Group优先级,优先级0-31,31最大



# Makefile

#!/bin/sh

cd /home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/blocks/soc\_tb/tool\_data/verilog

set -e

# Invoke stimulus makefile

/home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/blocks/soc\_tb/tool\_data/verilog/verilog\_FILENAMES\_awv071676.cn-sha01.nxp.com\_85014/run\_make\_stim.sh

# Invoke the simulator

/home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/blocks/soc\_tb/tool\_data/verilog/verilog\_FILENAMES\_awv071676.cn-sha01.nxp.com\_85014/runsim.sh

# Echo the command line string to stdout and the simulation log file

echo "Stingray command line: soc verilog -irun -block soc\_tb -bc rtl -tc default -vectors dma3 -test dma3\_interrupt\_core\_a\_inst2\_test\_5 -shm -keeptemps -session default\_5" | tee -a /home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/blocks/soc\_tb/vectors/dma3/logfiles/dma3\_interrupt\_core\_a\_inst2\_test\_5verilog\_default\_5.log

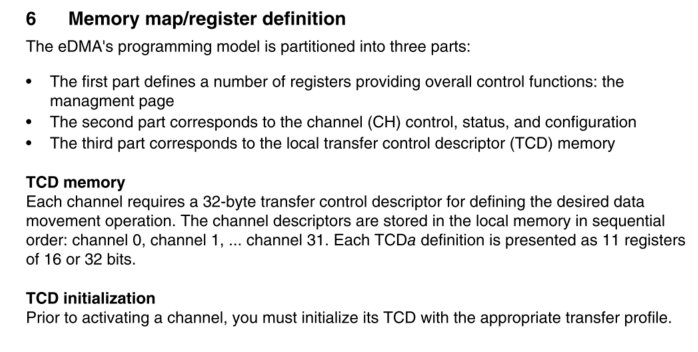
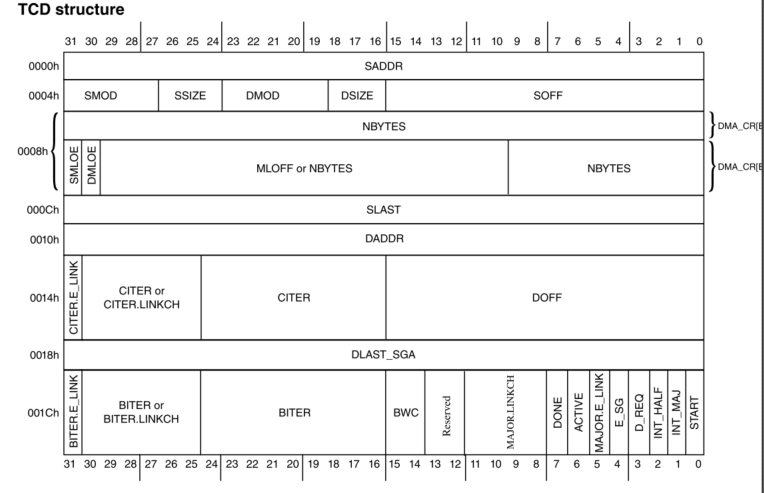
第一个makefile /home/imxrt1180\_verif3/richard\_nxa28190/RT\_1.30.GLS00.V01/testbench/blocks/soc\_tb/testbench/makefile.stimulus

# Testbench overview knowledge

<https://nxp1-my.sharepoint.com/personal/bill_cui_nxp_com/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fbill%5Fcui%5Fnxp%5Fcom%2FDocuments%2FRecordings%2F2022%5F05%20onboard%20New%20hire%20Q%26A%20serial%2D20220614%5F164854%2DMeeting%20Recording%2Emp4&parent=%2Fpersonal%2Fbill%5Fcui%5Fnxp%5Fcom%2FDocuments%2FRecordings&ct=1656311680338&or=Teams%2DHL&ga=1>

# Module description

## TCD

## Channel preemption

通道抢占.DMA允许某个channel在执行的时候，被另一个高级别的channel抢占通信，被抢占的通道挂起，等待抢占成功的channel完成传输后再继续没有完成的数据传输。

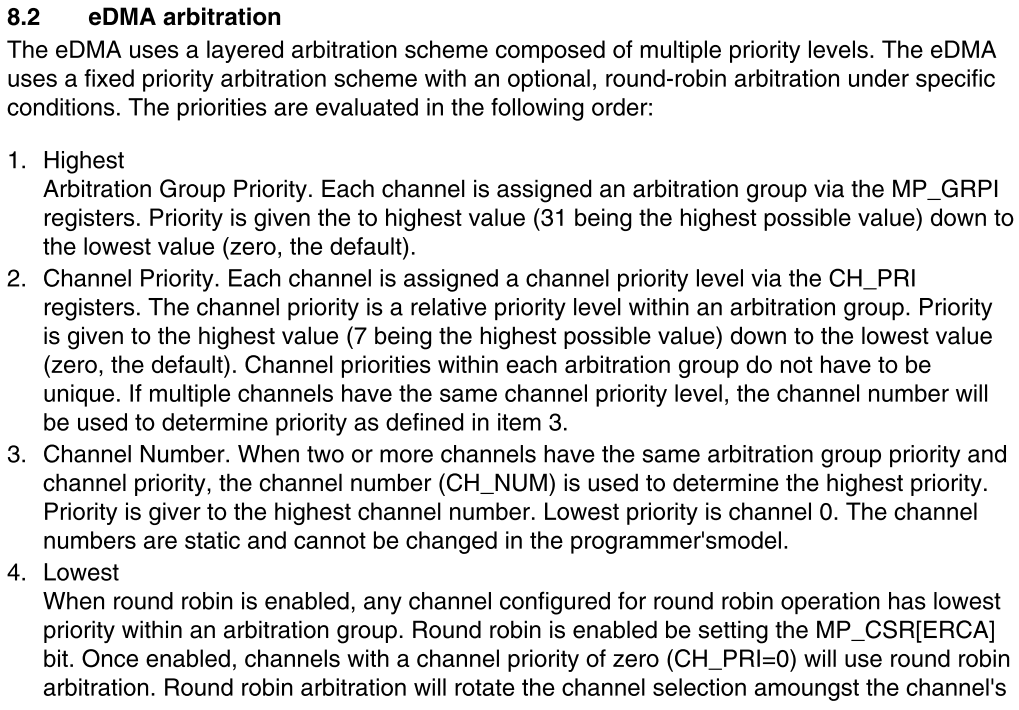
1. Channel preemption is enabled on a per-channel basis by setting the CHa\_PRI[ECP] bit.
2. A channel’s ability to preempt another channel can be disabled by setting CHa\_PRI[DPA].
3. When Round-Robin channel arbitration mode is enabled, MP\_CSR[ERCA] = 1, any channel with a priority level equal to zero, CHa\_PRI[APL] = 0, has preemption disabled and will not preempt another

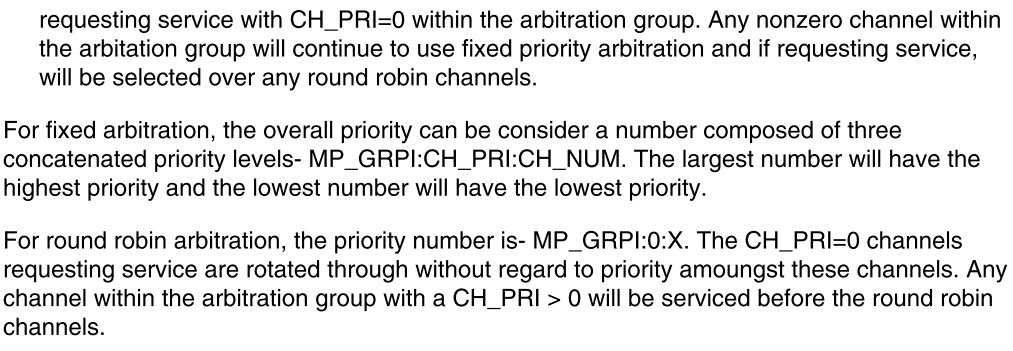
channel.

#### Channel priority

Group priority -> channel priority

[DMA3\_GRPRI](#_DMA3_GRPRI) [DMA3\_CH\_PRI](#_DMA3_CH_PRI)

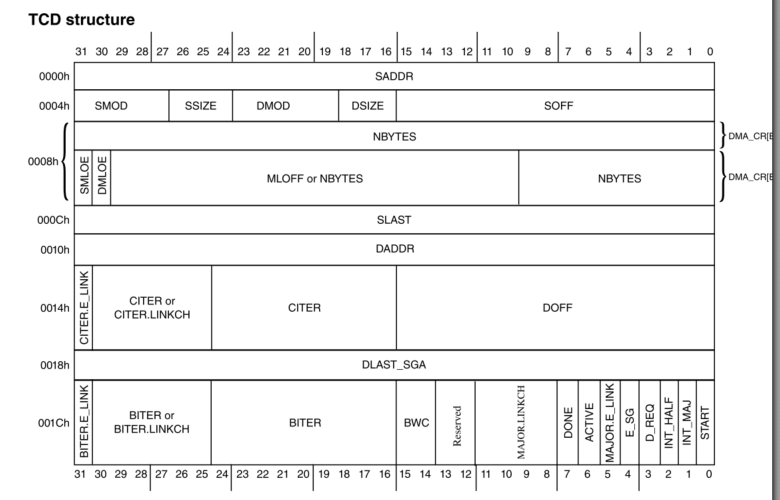




## Scat/gather

DMA支持将TCD的配置写入RAM中，DMA从RAM中获取TCD配置并写入TCD寄存器。TCD的结构参考[TCD](#_TCD)

当正在执行的channel major loop结束后，DMA会根据配置将TCD的内容写入TCD寄存器。



### Example

// Set TCD content into the RAM

// Loading TCD content to memory

WR32(REG32(TCM\_SYS\_MIDDLE), OCRAM2\_BASE); // for TCD\_SADDR

WR32(REG32(TCM\_SYS\_MIDDLE+0x04),0x02020004); // for TCD\_SOFF and TCD\_ATTR

WR32(REG32(TCM\_SYS\_MIDDLE+0x08),0x00000040); // for TCD\_NBYTES

WR32(REG32(TCM\_SYS\_MIDDLE+0x0c),0); // for TCD\_SLAST

WR32(REG32(TCM\_SYS\_MIDDLE+0x10),OCRAM2\_BASE+0x1000); // for TCD\_DADDR

WR32(REG32(TCM\_SYS\_MIDDLE+0x14),0x00010004); // for TCD\_DOFF and TCD\_CITER

WR32(REG32(TCM\_SYS\_MIDDLE+0x18), 0); // for TCD\_DLASTSGA

WR32(REG32(TCM\_SYS\_MIDDLE+0x1c), 0x00010008); // for TCD\_CSR and TCD\_BITER

RE16(REG16(TCM\_SYS\_MIDDLE+0x1e), 0x0001); // for TCD\_CSR and TCD\_BITER

RE16(REG16(TCM\_SYS\_MIDDLE+0x1c), 0x0008); // for TCD\_CSR and TCD\_BITER

//Enable E\_SG in TCD\_CSR register [DMA3\_TCD\_CSR](#_DMA3_TCD_CSR)

dma3\_ch\_set\_csr(dma3\_context, 0, 0x10); // enable e\_sg

// set the scat/gather RAM address [DMA3\_TCD\_DLAST](#_DMA3_TCD_DLAST)

W32(DMA3\_TCD\_DLAST(0), TCM\_SYS\_MIDDLE+0x400000); // scatter\_gathering address, set the aliased address into the register

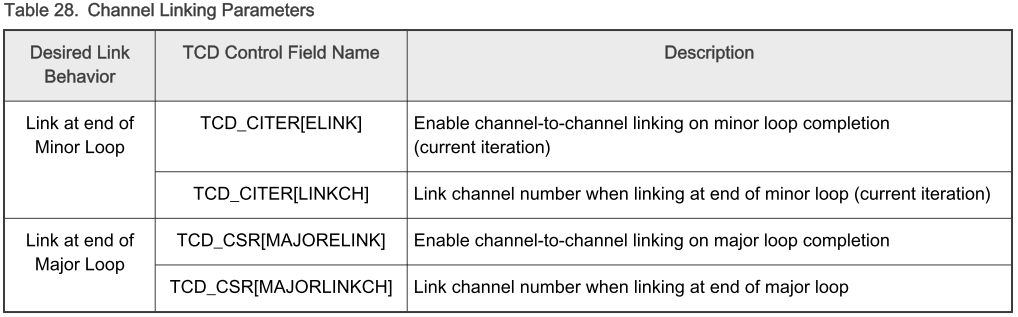
当一个channel major loop结束后，根据如上的配置会重新加载TCD的数据到寄存器中。

然后手动启动TCDn\_CSR[start]，再次执行该channel的时候，就会按照重新load的TCD进行数据传输。

## Channel linking

DMA支持使用channel linking的方式来启动相关channel的数据传输。

当手动设置TCDn\_CSR[start]启动一个channel后，在该channel的minor/major loop结束后，根据TCD\_CITER[ELINK]或者TCD\_CSR[MAJORELINK]的配置，DMA自动启动该linking的channel，而不需要手动配置TCDn\_CSR[start]



### Example

[DMA3\_TCD\_CITER](#_DMA3_TCD_CITER) [DMA3\_TCD\_CSR](#_DMA3_TCD_CSR)



如上配置了4个minor loop（TCDnCITR[citer] = 4）.

TCDn\_CSR[start] 启动该nchannel后，当第一个minor loop结束。自动连接到TCDn\_CITER[ELINK]设置的channel C进行数据传输

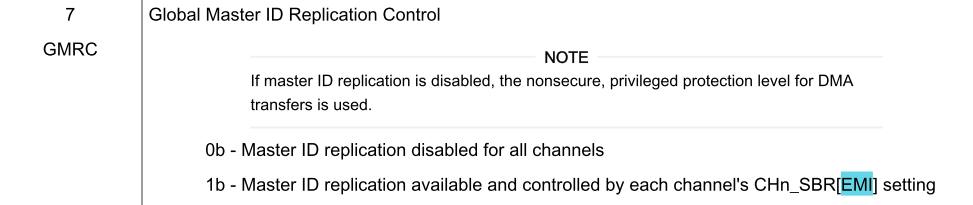
每个minor loop结束，Citer的值减少1，执行一次channel 12的数据传输

当major loop结束，也就是达到了TCDnCITR[citer] 设置的4次loop。自动启动TCDn\_CSR[MAJORELINKCH]对应的channel 7，进行channel 7的数据传输。

## Channel SBR

1. MP\_CSR （control status register）寄存器GMRC(global master ID replication control)置1





1. CHn\_SBR(channel system bus)寄存器，EMI(enable master ID replication)置1

