

ARM coresight 介绍

卢骏

目录

一、	coresight.....	4
1.	典型的一个 coresight 的环境.....	4
1.1	trace 通路	4
1.2	debug 的通路.....	5
1.3	trigger 通路.....	5
2.	coresight 组件的种类.....	5
2.1	control component.....	5
2.2	trace sources	5
2.3	trace links	6
2.4	trace sinks.....	6
2.5	debug access port	6
二、	coresight 的寄存器.....	8
1.	寄存器一览.....	8
2.	ITCTRL, integration mode control register	11
3.	CLAIM 寄存器	12
4.	DEVAFF, device affinity register	13
5.	lock 寄存器	14
6.	AUTHSTATUS, authentication status register	15
7.	DEVARCH, device architecture register	16
8.	DEVID, device configuration register	17
9.	DEVTYPE, device type identifier register.....	18
10.	PIDR0-PIDR7, peripheral identification registers	20
11.	CIDR0-CIDR3, component identification registers	21
三、	APB, ATB 总线.....	22
1.	APB 总线	22
2.	ATB 总线	24
2.1	全局信号.....	25
2.2	flow control	26
2.3	flusing.....	28
四、	channel interaface.....	30
1.	channel interface 信号.....	31
2.	CTI	33
2.1	对于 CIT 内部逻辑的 CTIAPP	35
2.2	外部的 CTM	38
2.3	CPU 发送的 input trigger.....	39
五、	rom table.....	39
3.	entry 寄存器	39
4.	例子.....	41
六、	power requestor.....	43
1.	CDBGPWRUPREQ	44
2.	CDBGPWRUPACK.....	44
3.	DEVID	45

七、	coresight 的两大功能.....	46
1.	debug	46
1.1、	单 core 的 debug 系统:	46
1.2、	多 core 的 debug 系统:	46
2.	trace	47
2.1、	单 core 的简单 trace 系统	47
2.2、	单 core 的高级 trace 系统	47
2.3、	多 core 的高级 trace 系统	48
3.	多 cluster 的完整 coresight 系统.....	49
八、	coresight soc-400	52
1.	DAP 组件.....	52
1.1、	SWJ-DP	54
1.2、	DAPBUS 互联	56
1.3、	AXI-AP.....	57
1.4、	APB-AP.....	57
2.	APB 互联组件	58
2.1、	rom table.....	58
2.2、	APB 异步桥	58
2.3、	APB 同步桥	59
3.	ATB 互联组件.....	59
3.1、	replicator	59
3.2、	funnel	59
3.3、	upsizer	60
3.4、	downsizer	61
3.5、	asynchronous bridge	61
3.6、	synchronous bridge	61
4.	timestamp 组件	62
5.	ECT 组件.....	62
5.1、	CTI	63
5.2、	CTM.....	63
6.	trace sink 组件	64
6.1、	TPIU.....	64
6.2、	ETB	65
7.	power requestor.....	66
8.	总结.....	66

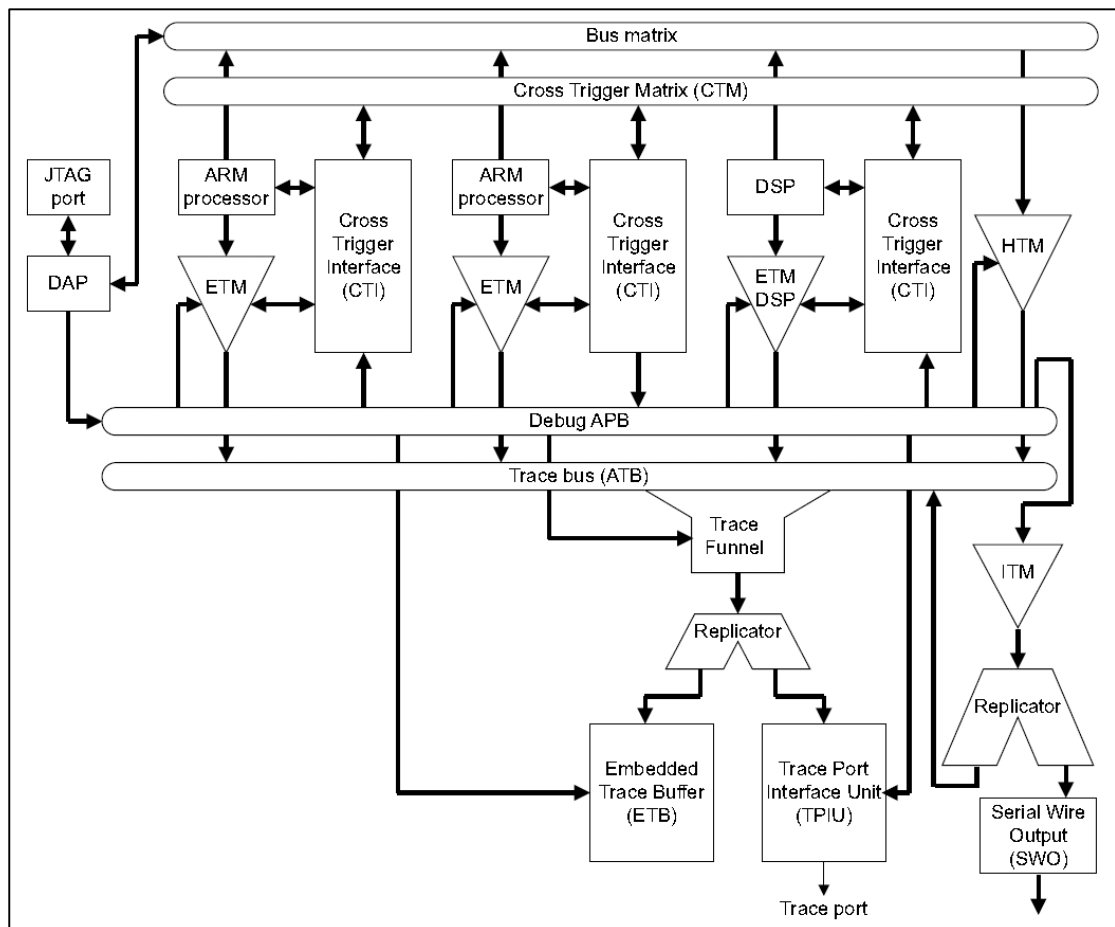
一、coresight

coresight 是 ARM 公司提出的, 用于对复杂的 SOC, 实现 debug 和 trace 的架构。该架构, 包含了多个 coresight 组件。众多的 coresight 组件, 构成了一个 coresight 系统。我们也可以根据 coresight 架构, 实现自己的 coresight 组件。

每个 coresight 的组件 (component), 都要遵循 coresight 架构的要求。

1. 典型的一个 coresight 的环境

以下是一个典型的 coresight 环境, 包含了两个 ARM core, 一个 DSP, 和众多的 coresight 组件。这个 coresight 组件, 实现对 core, DSP 的 debug 和 trace 功能。



环境中, 总共包括 3 个通路

- trace 通路: 将 core 和 DSP 内部信息输出到外部
- debug 通路: 对 core 和 DSP 实现 debug
- trigger 通路: 用于 core 和 core 之间, core 和 DSP 之间, 传输 trigger 信号

1.1 trace 通路

trace 通路, 实现对 master 组件的数据追踪功能, 使用 ETM 来追踪。

ETM 负责追踪处理器和 DSP 的信息, 将信息打包, 通过 ATB 总线发送到 trace bus 上。trace bus 上有 trace funnel, funnel 接收多个 ATB 总线数据, 然后合并成一个 ATB 总线数据,

发送给 replicator。

replicator 接收到 ATB 数据, 根据配置, 将 ATB 数据发送给 ETB 和 TPIU。

1.2 debug 的通路

debug 通路, 用于外部的 debugger, 对 ARM core 和 DSP 进行调试功能。

上图中, 只考虑了 JTAG 的 port。其实还有 SW 的 port。

DAP 接收外部端口的 JTAG 数据, 然后转化成对 DAP 内部的 AP 的访问, 然后 AP 再转化为 memory-mapped 的总线访问, 去访问 soc 内部的资源。

上图中, DAP 输出两个 memory-mapped 总线, 一个是 debug apb 总线, 连接到 debug APB 互联上, 用于访问 debug 组件的寄存器, 一个是 system bus, 连接到 bus matrix, 用于访问 soc 内部的资源。

debug APB 互联, 连接了有 CTI, ETM, HTM, ITM, ETB, TPIU 等 coresight 组件, 因此外部的 debugger 可以通过 JTAG port, 对这些 coresight 组件进行访问。

bus matrix 一般是连接 soc 的一些外设, 如 memory, 串口等, 因此外部的 debugger 可以通过 JTAG port 对这些外设设备进行访问。

1.3 trigger 通路

trigger 通路, 用于给指定的组件发送 trigger 信号, 或者接收指定的组件的 trigger 信号。这个功能由 CTI 和 CTM 来实现。

每个 core 和 DSP 都有一个 CTI 组件相连, CTI 可以给处理器 (DSP) 发送 trigger 信号, 也可以接收处理器 (DSP) 的 trigger 信号。

所有的 CTI 和 CTM 相连, 因此可以实现多个 CTI 之间的 trigger 信号的相互发送与接收。

2. coresight 组件的种类

2.1 control component

trigger 的 coresight 组件

- ECT (embedded cross trigger)
 - CTI (cross trigger interface): 接收和发送 trigger 信号
 - CTM (cross trigger matrix): CTI 之间的 trigger 信号传递

2.2 trace sources

产生 trace 的信息的 coresight 组件:

- ETM (embedded trace macrocells): 追踪指定设备 (处理器, DSP) 的 trace 信息, 每个设备 (处理器, DSP) 均有自己的 ETM。
- AMBA trace macrocells: 追踪 AMBA 总线的 trace 信息。
- PTM (program flow trace macrocells):
- STM (system trace macrocells): 追踪总线互联上的 trace 信息

2.3 trace links

trace 信息传递过程中所需要的中间组件:

- trace funnel: 将接收的多个 ATB 总线数据合并成一个 ATB 总线数据
- replicator: 将一个 ATB 总线数据, 分发成多个 ATB 总线数据发送
- ATB bridge: ATB 桥, 用于两个不同的 ATB 域之间数据传输

2.4 trace sinks

最终接收 trace 信息的 coresight 组件

- TPIU (trace port interface units): 将 ATB 数据通过 trace port 发送给外界
- ETB (embedded trace buffers): 存储 ATB 数据的 buffer
- TMC (trace memory controller):
每个 trace sink 可以有一个 trace formatter。

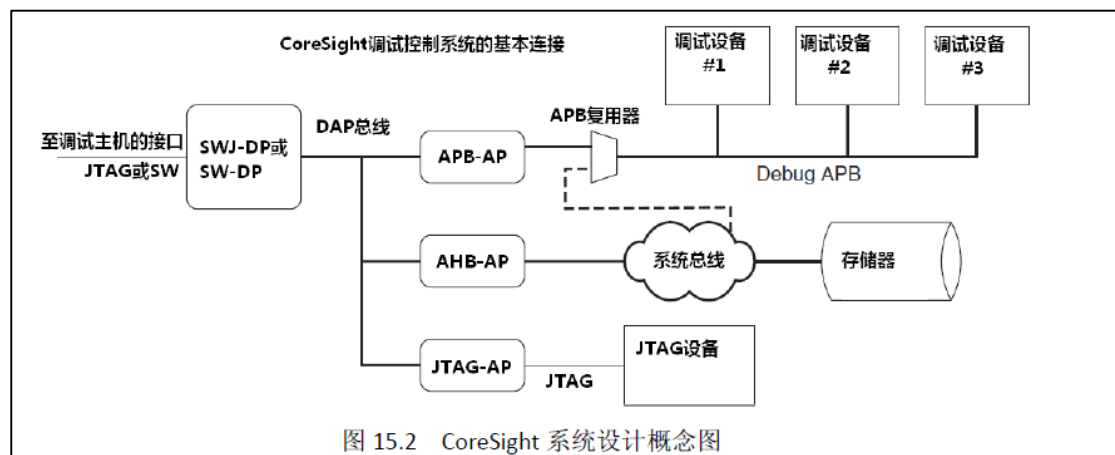
2.5 debug access port

DAP 不属于 coresight 的组件, 但是我们会通过 DAP 来对 coresight 的组件进行访问。

DAP 包括以下:

- APB access port(APB-AP)
- AHB access port(AHB-AP)
- AXI access port(AXI-AP)
- JTAG access port(JTAG-AP)
- serial wire JTAG debug port(SWJ-DP)
- JTAG debug port(JTAG-DP)
- ROM table

DAP 主要是由 DP 和 AP 组件。DP 负责接收外部的 JTAG 或 SW 数据, 然后转化为对 AP 的访问, 而对 AP 的访问, 是可以发起 memory-mapped 的访问。因此就可以对内部的资源进行访问。



如上图:

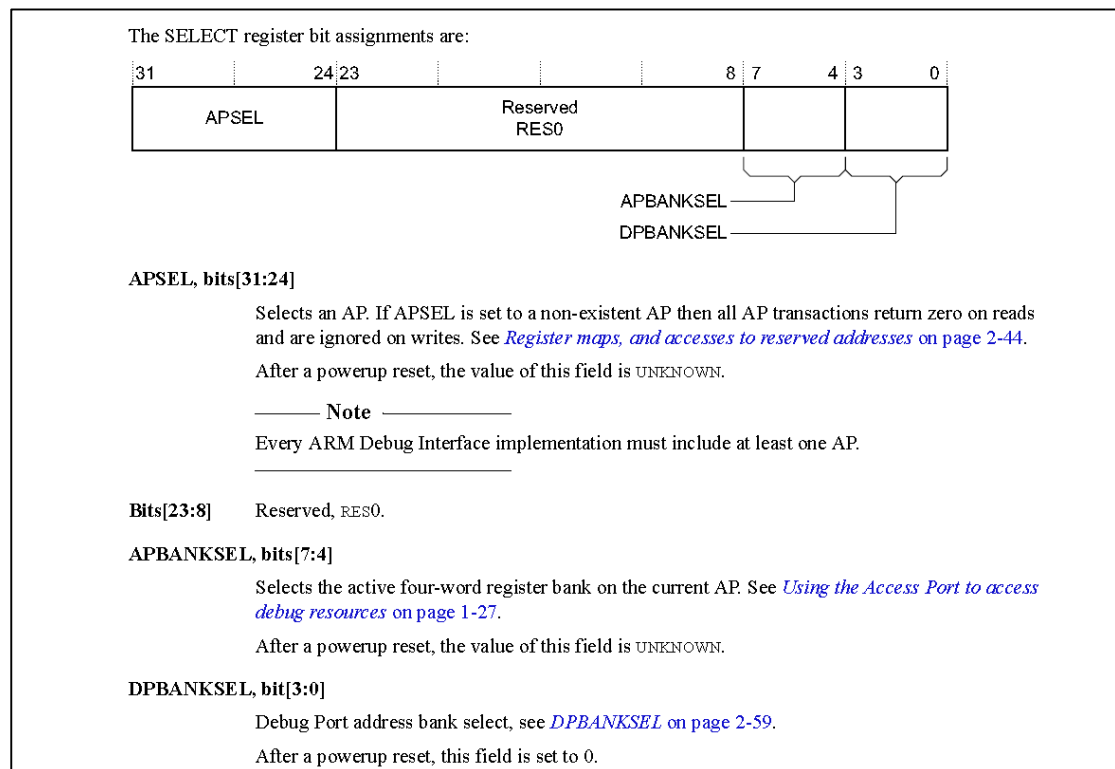
DAP 包括了三个 AP

- APB-AP: 对挂接到 debug APB 总线上的内部调试设备的访问
- AHB-AP: 对挂载在 AHB 系统总线上的设备的访问
- JTAG-AP: 对 JTAG 设备的访问。这个是兼容以前较早的 ARM 处理器, 如 ARM9。这些较早的处理器内部是用 JTAG 来调试的。但是现在的 ARM 处理器, 已经不用这种方式, 统一用 memory-mapped 方式进行调试。

目前的 ARM soc 中, 一般至少会包括一个 DAP。而一个 DAP 可以包括 1-256 个 AP(access port), AP 受 DP 的控制。只有对 AP 的访问, 才可以转化成 memory-mapped 总线, 对 soc 的内部资源进行访问。

DP 中有一个 SELECT 寄存器, 该寄存器用来选择, DP 对 AP 的访问, 是针对哪一个 AP 进行访问。

DAP 中, 是可以有多个 AP 的, 而每次, 只能对一个 AP 进行访问。因为需要对 AP 进行编号, 编号的值就在 APSEL 位域中。因为这个位域有 8 位, 因此 DAP 中可以最多有 256 个 AP。

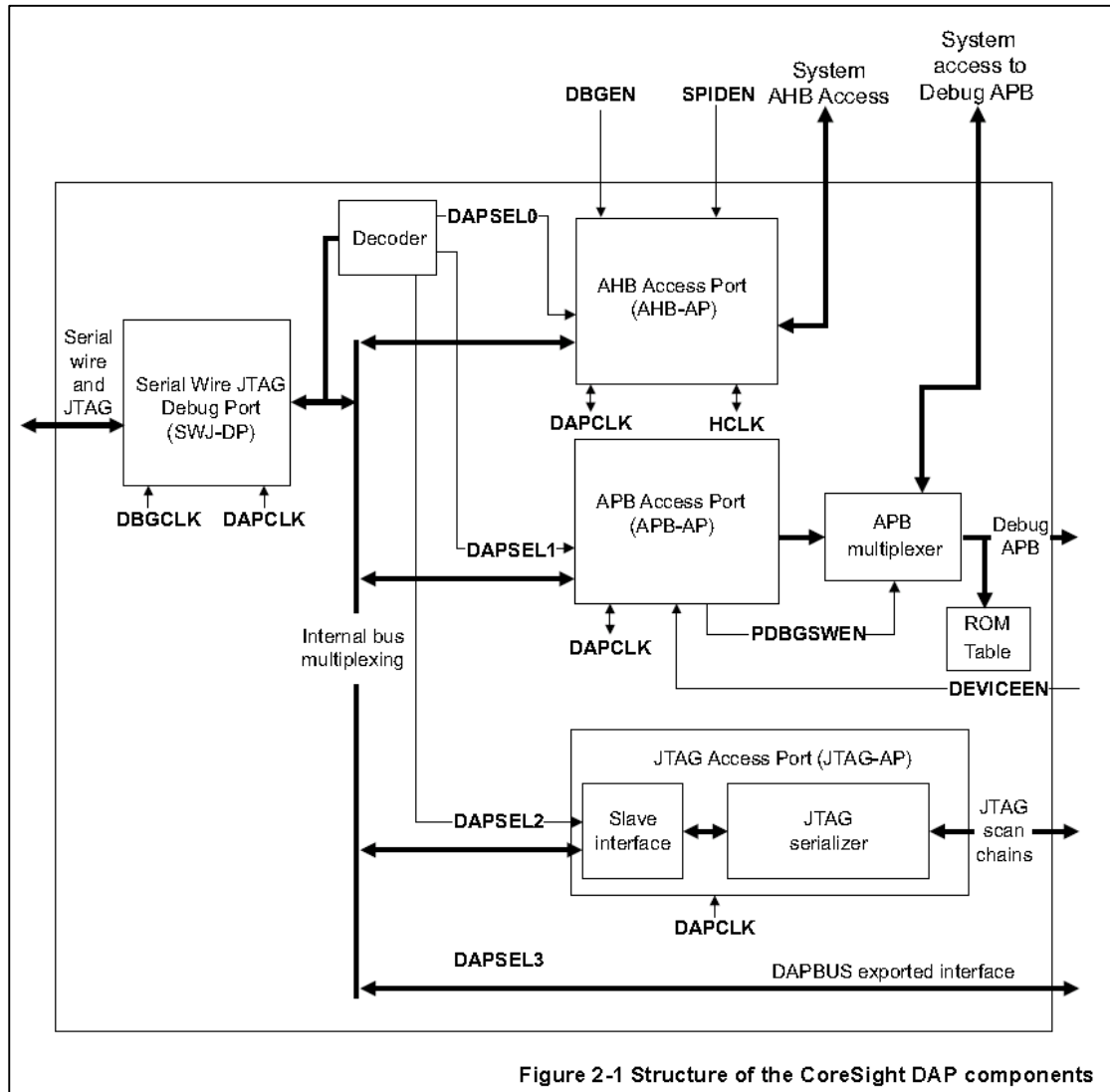


DAP 的内部结构如下图:

包括了一个 DP, 和 3 个 AP, 依次是 AHB-AP, APB-AP, JTAG-AP。

DP 通过 JTAG 或者 SW 管脚, 连接外部的 debugger, 和外部 debugger 进行通信。

DP 接收到外部 debugger 发送的 JTAG 或 SW 数据, 转化为对内部 AP 的访问。经过 decoder 模块, 判断是对哪一个 AP 进行访问, 然后将访问信息发送给对应的 AP。AP 接收到 DP 的访问后, 转化为对应的总线访问, 去访问内部资源。然后将访问的信息, 才回送给 DP, DP 再通过 JTAG 或 SW, 将访问信息返回给外部的 debugger。



二、coresight 的寄存器

coresight 对于每个 coresight 组件，规定了一些寄存器，这些寄存器的偏移是固定的，这些寄存器，是必须存在的。但是有的，可以不实现该寄存器功能。

1. 寄存器一览

coresight 架构，对于 coresight 的组件，定义了若干个固定的寄存器。第一个寄存器的偏移从 0xf00 开始，直到 0xffc。以下是寄存器列表：

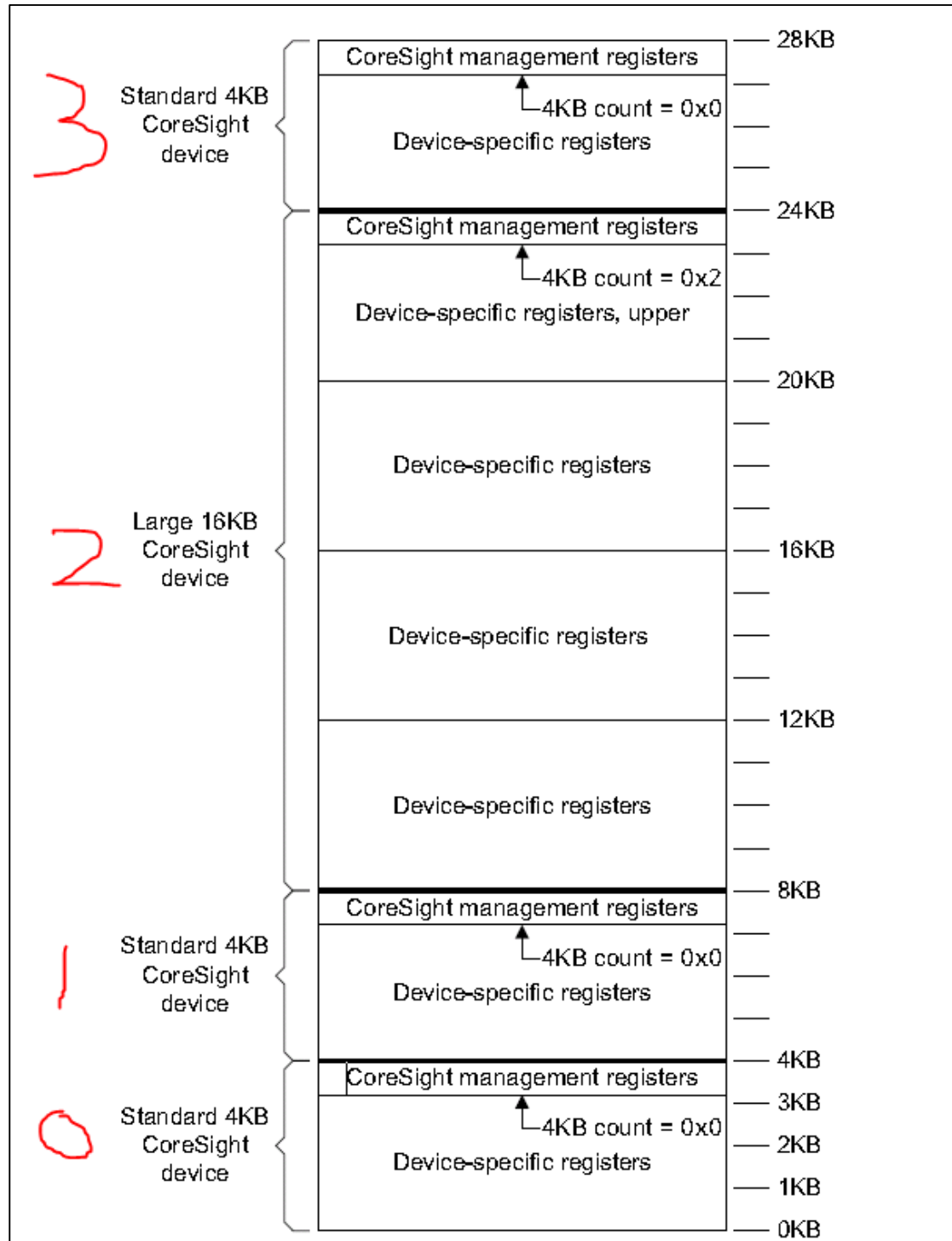
offset	type	valid bits	name	description
0xf00	RW	1	ITCTRL	integration mode control register
0xfa0	RW	0-32	CLAIMSET	claim tag set register
0xfa4	RW	0-32	CLAIMCLR	claim tag clear register
0xfa8	RO	32	DEVAFF0	device affinity register0
0xfac	RO	32	DEVAFF1	device affinity register1
0xfb0	WO	32	LAR	lock access register

0xfb4	RO	3	LSR	lock status register	
0xfb8	RO	8	AUTHSTATUS	authentication status register	
0xfbc	RO	32	DEVARCH	device architecture register	
0xfc0	RO	8-32	DEVID2	device configuration register2	
0xfc4	RO	8-32	DEVID1	device configuration register1	
0xfc8	RO	8-32	DEVID	device configuration register	
0xfcc	RO	8	DEVTYPE	device type identifier register	
0xfd0	RO	8	PIDR4	peripheral identification registers PIDR0-PIDR7	
0xfd4	RO	8	PIDR5		
0xfd8	RO	8	PIDR6		
0xfdc	RO	8	PIDR7		
0xfe0	RO	8	PIDR0		
0xfe4	RO	8	PIDR1		
0xfe8	RO	8	PIDR2		
0xfec	RO	8	PIDR3		
0xff0	RO	8	CIDR0	preamble	component identification registers, CIDR0-CIDR3
0xff4	RO	8	CIDR1	component class	
0xff8	RO	8	CIDR2	preamble	
0xffc	RO	8	CIDR3	preamble	

以上的寄存器的地址，在 coresight 的组件中，是不能当作其他功能使用的。如果该寄存器，在该组件没有实现，那么该寄存器地址要保留，读取要返回 0，写被忽略（read must return zero, and writes must be ignored），而不能当作其他功能使用。

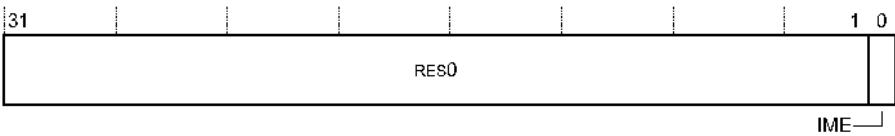
对于 coresight 的组件，占用 1 个 4k 或者整数倍的 4k 空间的 memory 空间。而 coresight 的寄存器，处于组件占用空间的最后一个 4K 空间的最后一部分。

以下是一个 coresight 组件占用的 memory 空间。占用一个 4k 空间。



2. ITCTRL, integration mode control register

工作模式寄存器。

The ITCTRL bit assignments are:	
	
Description	This register enables the component to switch between functional mode and integration mode. The default behavior is functional mode. In integration mode the inputs and outputs of the component can be directly controlled for the purpose of integration testing and topology solving.
Bits	<p>[31:1] RES0.</p> <p>[0] IME. When set, the component enters integration mode, enabling topology detection or integration testing to be performed. At reset the component must enter functional mode. If no integration functionality is implemented, this register might be RAZ.</p>
Short name	ITCTRL.
Location	0xF00.

对于每个 coresight 组件，可以工作在两种模式下：

- functional mode
- integration mode

两种模式的区别，在于对 coresight 组件的寄存器的访问，是否会引发寄存器相应的功能。

integration mode 是用来 topology detection 的。当一个 debugger 连接到一个 soc 后，此时 debugger 是不知道 soc 内部有哪些 coresight 组件的。因此就需要通过查询，来得知 soc 中有哪些 coresight 组件的。而查询，就是通过访问 coresight 组件的寄存器来实现的。此时 soc 还不知道组件是什么组件，因此也就不知道组件的寄存器是有什么功能。因此此时是不能随意对组件的寄存器进行访问的。

为了使访问的过程中，不影响组件的功能，就可以让组件工作在 integration mode 下，此时访问组件的寄存器，不会引发寄存器相应的功能。待 debugger 查询完毕后，获取到 soc 中各个 coresight 组件的信息后，再将组件的模式切换为 functional mode。

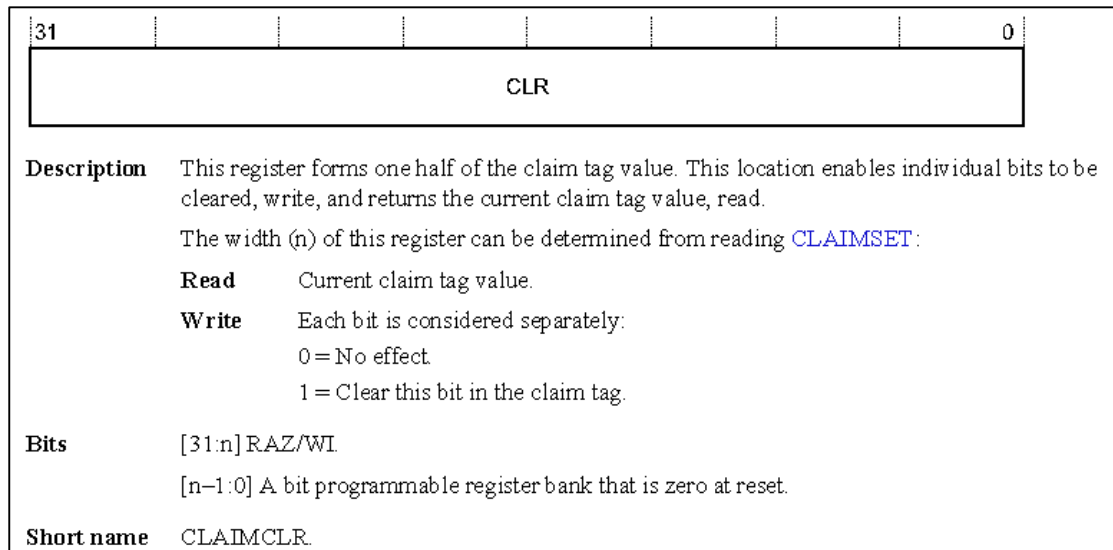
复位后，组件必须工作在 functional mode 下。因此外部 debugger 对组件查询完毕后，可以直接对组件进行复位，这样所有的组件就恢复到了 function mode 了。

3. CLAIM 寄存器

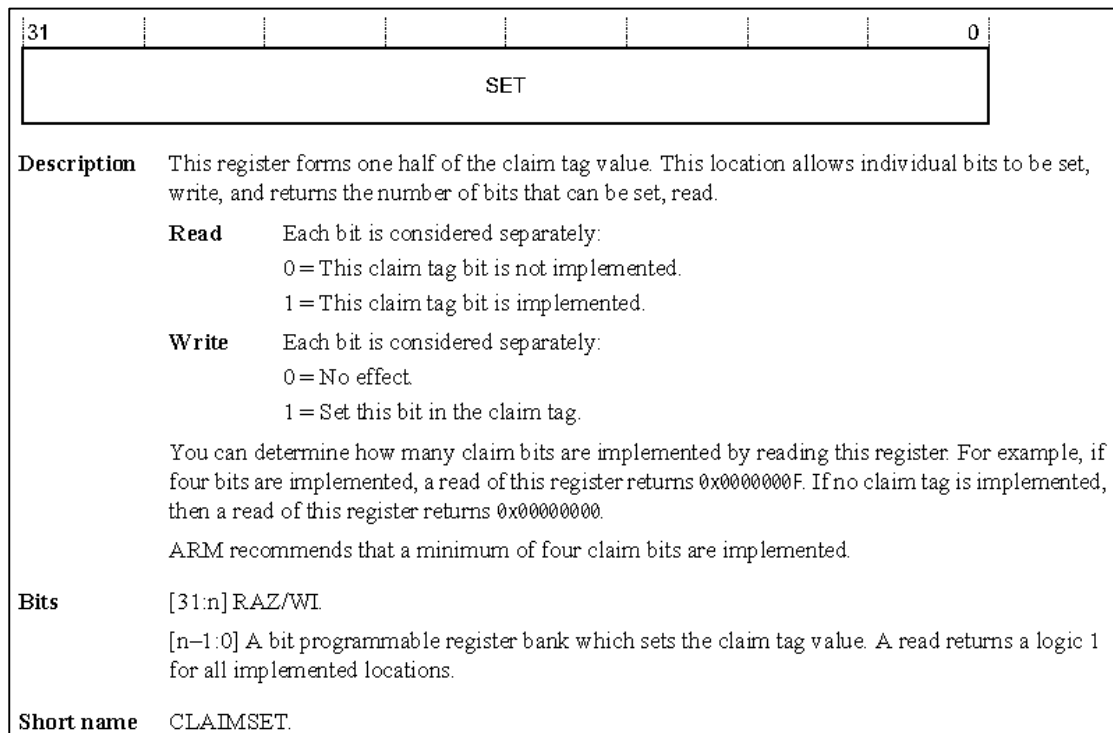
这个寄存器是一个 32 位的不可见寄存器。只能通过访问 CLAIMCLR 和 CLAIMSET 这两个寄存器，来设置或者获取该寄存器的值。

该寄存器，可以用来表示该组件的状态。这个是由实现来定义的，比如可以规定，该寄存器的最低位，表示最近该寄存器被读取过，第 1 位，表示最近该寄存器被写过。

CLAIMCLR 寄存器：



CLAIMSET 寄存器



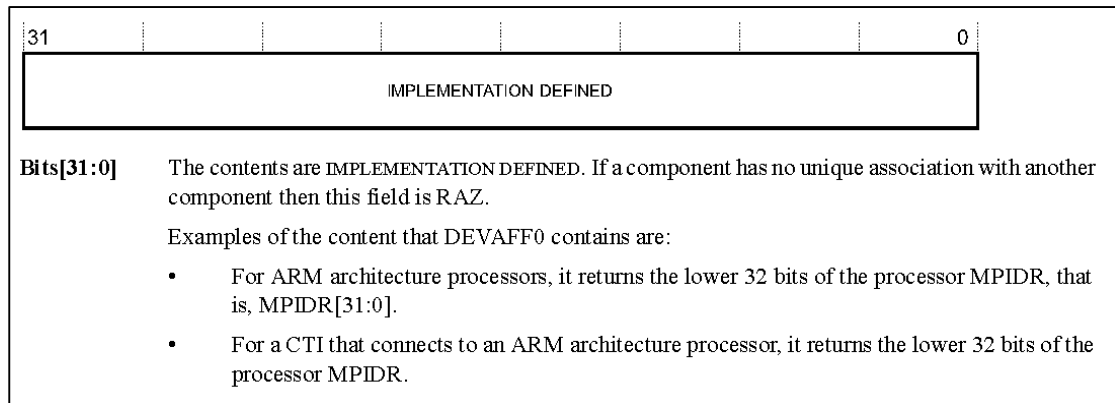
4. DEVAFF, device affinity register

组件关联功能寄存器。

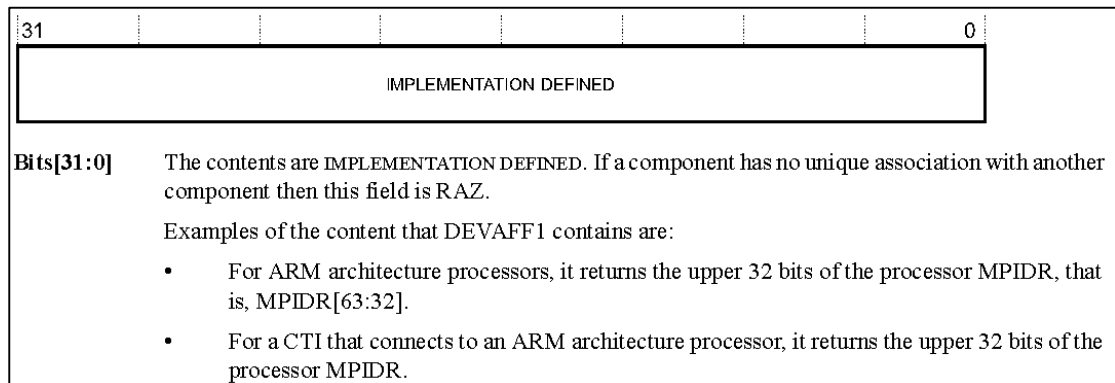
有时候，组件需要和其他组件，联合起来工作，这样，就需要指示该组件是和另外的什么组件进行关联，就可以用这寄存器。

比如一个 ETM，追踪一个 core 的 trace 信息，那么这个寄存器，就保存 core 的 MPIDR 寄存器信息，这样 debugger 就可以通过 DEVAFF 寄存器，得知这个 ETM 是关联的哪一个 core。

DEVAFF0 寄存器:

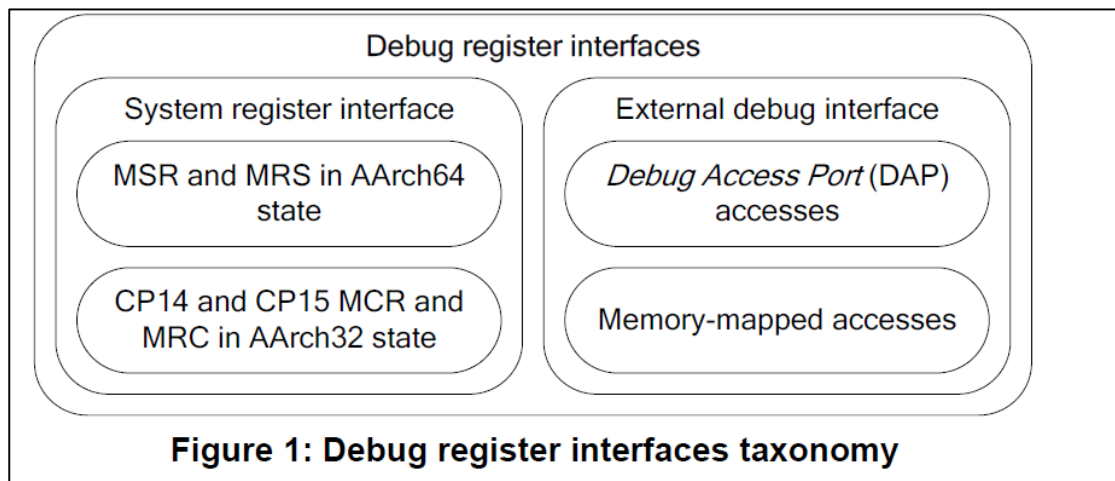


DEVAFF1 寄存器:



5. lock 寄存器

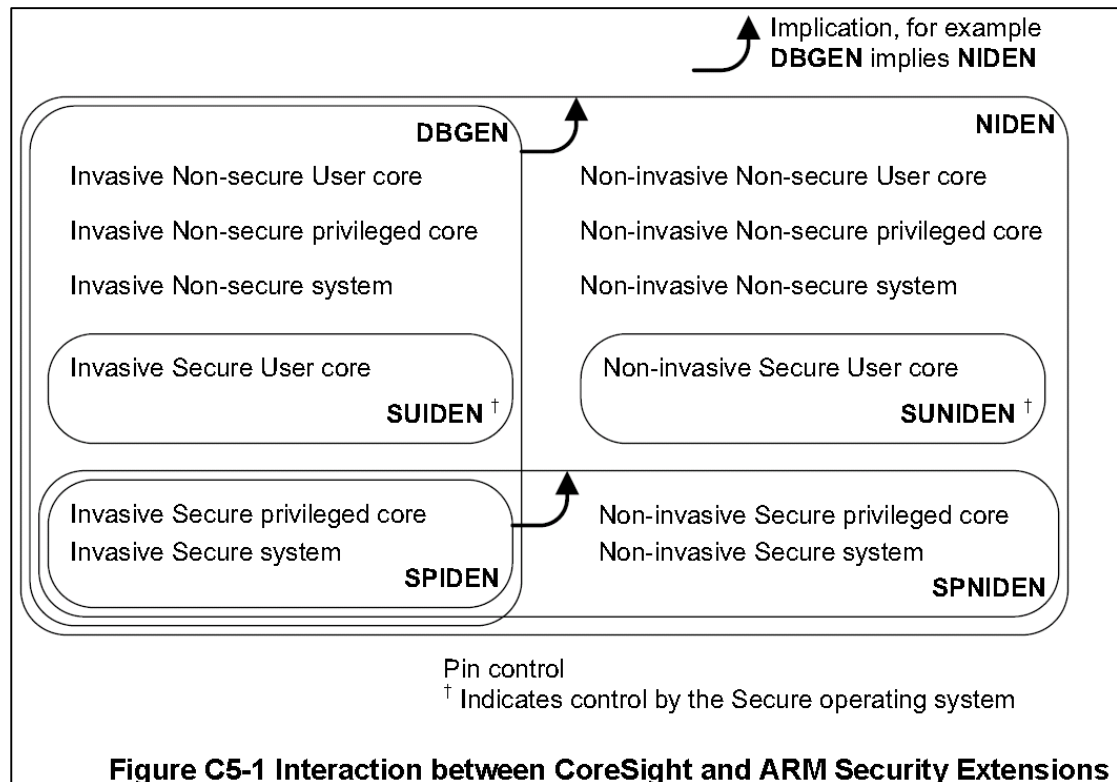
对于 coresight 组件的寄存器，ARM 定义了如下的一些访问方式:



可以分为两类访问:

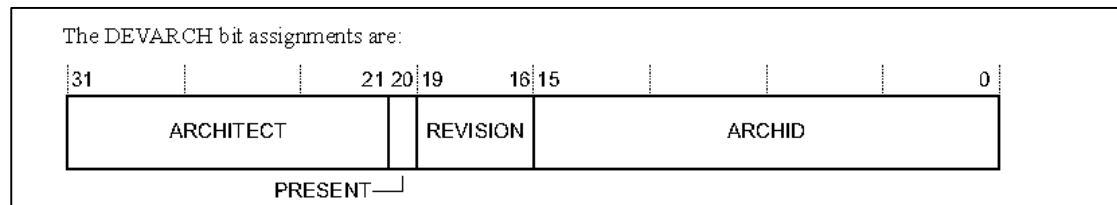
- 系统寄存器访问: 通过 MSR, MRS 指令 (aarch64), MCR, MRC 指令 (aarch32)
- external debug 接口访问: DAP 访问, 或者是 memory-mapped 访问, 也就是软件通过 load store 访问

对 coresight 组件寄存器的访问, 是有权限要求的。对于系统寄存器访问和 memory-mapped 访问, ARM 定义了 software lock 这个权限限制。当 software lock 有效的时候, 软件是不能访问 coresight 组件寄存器的。



7. DEVARCH, device architecture register

这个寄存器，标识了 coresight 组件的架构信息。



这里主要关心 ARCHID 这个位域。

Table B2-11 Example ARCHID values

ARCHID	Description
0x4A13	<i>Embedded Trace Macrocell</i> (ETMv4) architecture
0x1A14	<i>Cross Trigger Interface</i> (CTI) architecture
0x6A15	Processor debug architecture
0x2A16	Processor <i>Performance Monitor</i> (PMU) architecture
0x0A31	Basic trace router
0x0A34	Power requestor
0x0A63	<i>System Trace Macrocell</i> (STM) architecture

8. DEVID, device configuration register

这个寄存器的功能，由实现进行定义，总共包括 3 个寄存器，DEVID，DEVID1，DEVID2，每个寄存器 32 位，只读。

DEVID 寄存器：

31																													0
IMPLEMENTATION DEFINED																													
Description	This register is IMPLEMENTATION DEFINED for each Part Number and Designer. This indicates the capabilities of the component. The entire 32-bit field can be used because the data width is determined by the particular component. Unused bits must be RAZ. If the component is configurable then it is recommended that this register reflects any changes to a standard configuration.																												
Bits	[31:0] IMPLEMENTATION DEFINED.																												
Short Name	DEVID.																												

DEVID1 寄存器：

The DEVID1 bit assignments are:																													
31																													0
IMPLEMENTATION DEFINED																													
Bits[31:0]	The contents are IMPLEMENTATION DEFINED.																												

DEVID2 寄存器：

The DEVID2 bit assignments are:

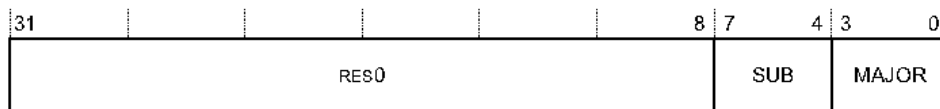
**Bits[31:0]**

The contents are IMPLEMENTATION DEFINED.

9. DEVTYPE, device type identifier register

组件的具体类型信息。依靠 MAJOR 位域和 SUB 位域来表示。

The DEVTTYPE bit assignments are:



Description	Indicates the type of functionality the component supports.
--------------------	---

Bits	[31:8] RES0
------	-------------

[7:4] SUB type

[3:0] MAJOR type.

Short Name DEVTYPE.

Location 0xFCC.

以下是组合情况:

Table B2-8 Device type encoding			
MAJOR type [3:0]		SUB type [7:4]	
Value	Description	Value	Description
0x0	Miscellaneous	0x0	Other, undefined
		0x1-0x3	Reserved.
		0x4	Validation component.
		0x5-0xF	Reserved.
		0x1	Trace Sink
0x1	Trace port, for example TPIU.		
0x2	Buffer, for example ETB.		
0x3	Basic trace router.		
0x4-0xF	Reserved.		

MAJOR type [3:0]		SUB type [7:4]	
Value	Description	Value	Description
0x2	Trace Link	0x0	Other.
		0x1	Trace funnel, Router.
		0x2	Filter.
		0x3	FIFO, Large Buffer.
		0x4-0xF	Reserved.
0x3	Trace Source	0x0	Other.
		0x1	Associated with a processor core.
		0x2	Associated with a DSP.
		0x3	Associated with a Data Engine or Coprocessor.
		0x4	Associated with a Bus, stimulus derived from bus activity.
		0x5	Reserved.
		0x6	Associated with software, stimulus derived from software activity.
		0x7-0xF	Reserved.

0x4	Debug Control	0x0	Other.
		0x1	Trigger Matrix, for example ECT.
		0x2	Debug Authentication Module. See Control of authentication interfaces on page D2-108
		0x3	Power requestor.
		0x4-0xF	Reserved.
0x5	Debug Logic	0x0	Other.
		0x1	Processor core.
		0x2	DSP.
		0x3	Data Engine or Coprocessor.
		0x4	Bus, stimulus derived from bus activity.
		0x5	Memory, tightly coupled device such as <i>Built In Self Test</i> (BIST).
		0x6-0xF	Reserved.

MAJOR type [3:0]		SUB type [7:4]	
Value	Description	Value	Description
0x6	Performance Monitor	0x0	Other.
		0x1	Associated with a processor.
		0x2	Associated with a DSP.
		0x3	Associated with a Data Engine or Coprocessor.
		0x4	Associated with a bus, stimulus derived from bus activity.
		0x5	Associated with a memory management unit that conforms to the ARM System MMU Architecture.
		0x6-0xF	Reserved.
0x7-0xF	Reserved	-	-

可以看出, arm 对组件分成了 7 大类:

- miscellaneous: 杂散类,
- trace sink: 最终接收 trace 信息的组件, 包括有 TPIU, ETB, router。
- trace link: trace 信息传递过程中需要的中间组件, 包括有 router, filter, FIFO
- trace source: 产生 trace 信息的 master
- debug control: debug 的控制器
- debug logic: 具有 debug 功能的 master
- performance monitor: 性能的检测器检测的 master

10. PIDR0-PIDR7, peripheral identification registers

外设识别寄存器。

Table B2-7 Peripheral Identification Registers					
Name	Offset	Bits	Field	Value	Description
PIDR7	0xFDC	[7:0]	-	RES0	Reserved.
PIDR6	0xFD8	[7:0]	-	RES0	Reserved.
PIDR5	0xFD4	[7:0]	-	RES0	Reserved.
PIDR4	0xFD0	[7:4]	SIZE	-	4KB count.
		[3:0]	DES_2	-	JEP106 continuation code.
PIDR3	0xFEC	[7:4]	REVAND	-	RevAnd.
		[3:0]	CMOD	-	Customer Modified.
PIDR2	0xFE8	[7:4]	REVISION	-	Revision.
		[3]	JEDEC	1	Always set. Indicates that a JEDEC assigned value is used.
		[2:0]	DES_1	-	JEP106 identification code, bits[6:4].
PIDR1	0xFE4	[7:4]	DES_0	-	JEP106 identification code, bits[3:0].
		[3:0]	PART_1	-	Part number, bits[11:8].
PIDR0	0xFE0	[7:0]	PART_0	-	Part number, bits[7:0].

这里面, 我们关心的是 SIZE, 和 Part number。因为其他的值在一个 soc 中, 所有的组件的值是固定的。

- **SIZE:** 表示这个组件, 占用 4 k 空间的块数。如果只占用一个块, 那么值是 0, 如果占用两个块, 值是 1。占用的块数为 2^{SIZE} 。

以下是 SIZE 对应的组件占用的大小。以及需要访问这个组件需要的地址宽度。

Number of 4KB blocks	Value of PADDR4.SIZE	Total memory window used	Component specific registers available	Expected PADDRDBG input ^a
1	0x0	4KB, 1K words	960 words	PADDRDBG[11:2]
2	0x1	8KB	1984 words	PADDRDBG[12:2]
4	0x2	16KB, 4K words	4032 words	PADDRDBG[13:2]
8	0x3	32KB, 8K words	8128 words	PADDRDBG[14:2]
16	0x4	64KB	16320 words	PADDRDBG[15:2]
32	0x5	128KB	32704 words	PADDRDBG[16:2]
64	0x6	256KB, 64K words	65472 words, 63.94K words	PADDRDBG[17:2]
128	0x7	512KB	127.94K words	PADDRDBG[18:2]
256	0x8	1MB, 256K words	255.9K words	PADDRDBG[19:2]
512	0x9	2MB	~512K words	PADDRDBG[20:2]
1024	0xA	4MB	~1M words	PADDRDBG[21:2]
2048	0xB	8MB	~2M words	PADDRDBG[22:2]
4096	0xC	16MB	~4M words	PADDRDBG[23:2]
8192	0xD	32MB	~8M words	PADDRDBG[24:2]
16384	0xE	64MB, 16M words	~16M words	PADDRDBG[25:2]
Reserved	0xF	-	-	-

a. PADDRDBG[1:0] are not required on blocks as all transfers under the AMBA 3 APB protocol are 32-bit, word, aligned. PADDRDBG[31] might also be required. For more information see [Use of PADDRDBG\[31\] on page C2-69](#).

- part number: 组件的唯一编号。soc 中有多个 coresight 的组件，为了较好方便的管理这些组件，给每个组件分配了唯一的编号。这个编号就保存在 part number 中。

对于 JEP106，这个是 JEDEC 标准，可以查阅以下网站；

www.jedec.org

11.CIDR0-CIDR3， component identification registers

这四个寄存器，每个寄存器只有最低 8 位有效。这四个寄存器的组合，用来标识组件的类型。

对于 ARM 的组件，CIDR 寄存器的有些位是固定的。比如对于 CIDR0, CIDR1[3:0], CIDR2, CIDR3，是有固定值的。

Table B2-5 Component Identification Registers					
Name	Offset	Bits	Field	Value	Description
CIDR3	0xFFC	[7:0]	PRMBL_3	0xB1	Preamble
CIDR2	0xFF8	[7:0]	PRMBL_2	0x05	Preamble
CIDR1	0xFF4	[7:4]	CLASS	See Table B2-6	Component class
		[3:0]	PRMBL_1	0x0	Preamble
CIDR0	0xFF0	[7:0]	PRMBL_0	0x00	Preamble

CIDR1 寄存器中有一个 CLASS 位域, 用来表示组件属于哪一个类。ARM 对自己的组件, 也划分了若个的类。

Table B2-6 CLASS field encodings	
Value	Description
0x0	Generic verification component.
0x1	ROM table. See <i>Class 0x1 ROM table on page B2-39</i> .
0x2-0x8	Reserved.
0x9	CoreSight component. See <i>Class 0x9 CoreSight component on page B2-40</i> .
0xA	Reserved.
0xB	Peripheral Test Block.
0xC-0xD	Reserved.
0xE	Generic IP component.
0xF	CoreLink, PrimeCell, or system component with no standardized register layout, for backwards compatibility. See <i>Class 0xF CoreLink, PrimeCell, or system component on page B2-51</i> .

这里, 我们关心如下的 class:

- 0x1: rom table
- 0x9: coresight 组件。
- 0xf: corelink 组件

读取组件的 CIDR 寄存器, 即可知道这个组件是属于哪一类。

对于 rom table, 固定为 0xb105_100d

对于 coresight 组件, 固定为 0xb105_900d

对于 corelink 组件, 固定为 0xb105_f00d

三、 APB, ATB 总线

APB 和 ATB 总线, 是 coresight 中常用的 2 个总线。

对于 coresight 组件的访问, 使用 debug APB 总线进行访问。而对于 trace 数据的传输, 使用 ATB 总线进行传输。

1. APB 总线

以下是信号列表。

Table C2-1 Signals on the Debug APB interface

Name	Direction			Clamp value	Description
	Master	Slave			
PCLKDBG	Input	Input	-		The rising edge of PCLKDBG times all transfers on the AMBA 3 APB interface.
PRESETDBGn	Input	Input	-		This signal resets the interface and is active LOW.
PADDRDBG[31:2]	Output	Input	0		This bus indicates the address of the transfer. You do not have to implement unused bits. For information on the special use of bit[31] see Use of PADDRDBG[31] on page C2-69.
PSELDBG	Output	Input	0		This signal indicates that the slave device is selected and a data transfer is required. There is a PSELDBG signal for each slave.
PENABLEDBG	Output	Input	0		This signal indicates the second and subsequent cycles of an AMBA 3 APB interface transfer.
PWRITEDBG	Output	Input	0		When HIGH this signal indicates a write access and when LOW a read access.
PWDATADBG[31:0]	Output	Input	0		The write data bus is driven by the master during write cycles, when PWRITEDBG is HIGH. The write data bus can be up to 32-bits wide.
PREADYDBG	Input	Output	1		The ready signal is used by the slave to extend an AMBA 3 APB interface transfer.
PRDATADBG[31:0]	Input	Output	0		The read data bus is driven by the selected slave during read cycles, when PWRITEDBG is LOW. The read data bus can be up to 32-bits wide.
PSLVERRDBG	Input	Output	1		This signal is returned in the second cycle of the transfer, and indicates an error response. CoreSight components should only use this signal to indicate that the component is unavailable, for example because of power down.

clamp value, 是指当一个组件是 power down 或者是 disabled, 输出的固定值。

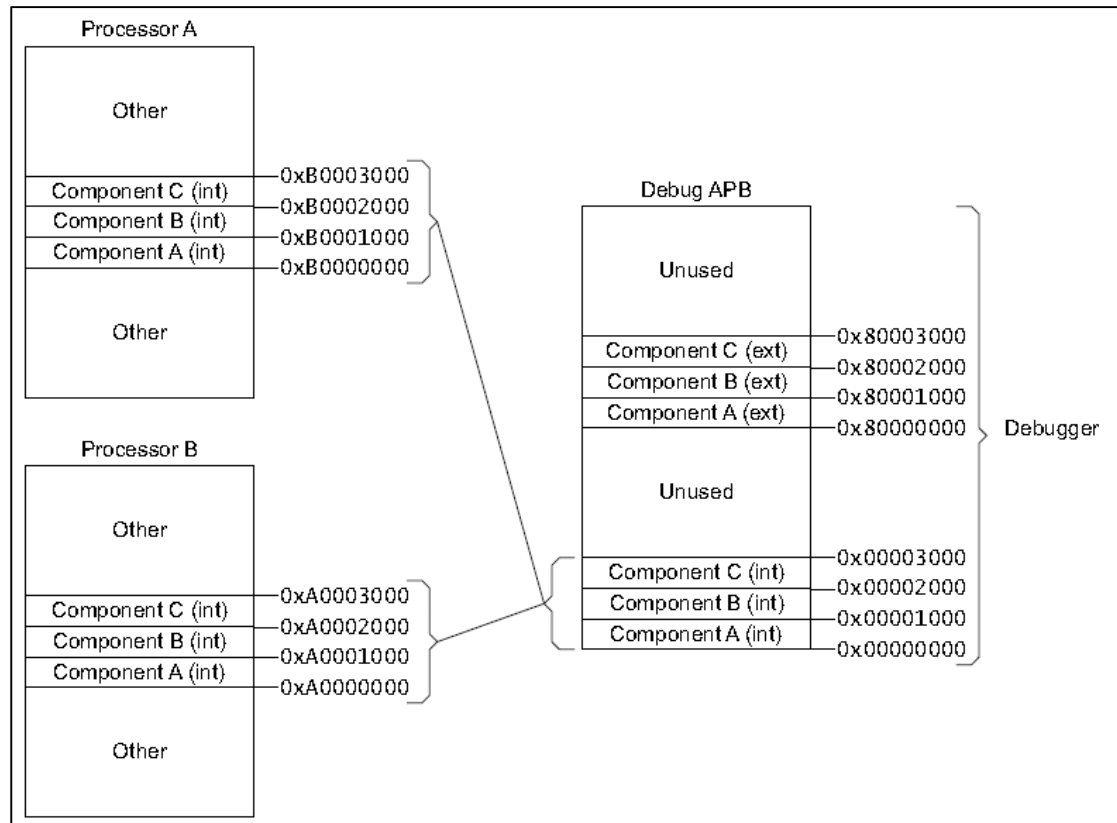
APB 访问, 数据最多是 32bit, 也就是 coresight 组件的寄存器的位宽最多是 32bit 的。

对于 PADDRDBG[31], 地址的最高位, 表示当前的访问是 internal access, 还是 external access。

- internal access, 是指处理器执行指令的访问, 比如 load/store 去访问, 或者是外部 debugger 通过 memory map 的访问。
- external access, 是指外部的访问, 比如 debugger, external access 比 internal access 有更高的权限。

如下图, 两个处理器, 均有自己的 coresight 组件。每个组件的地址是不一样的。对于外部的 debugger 而言, 使用地址 0x8000_0000 和地址 0x0000_0000 都是访问的同一个 coresight 组件的寄存器, 要不就是处理器 A 的组件 A, 要不就是处理器 B 的组件 A。

但是如果 debugger 使用地址 0x0000_0000 访问, 是 internal access, 此时受限于 software lock 的影响。而使用地址 0x8000_0000 访问, 是 external access, 不受限于 software lock。



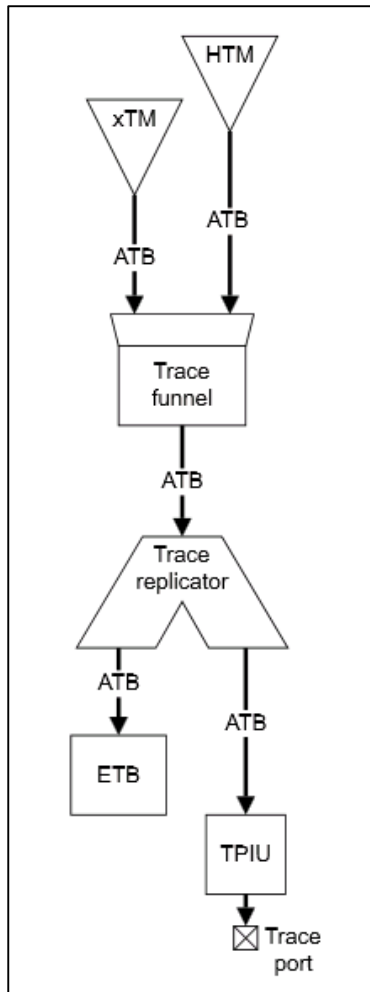
2. ATB 总线

ATB 总线用来 coresight 组件之间传递 trace 信息用。包括两部分：

- master: 在 ATB 总线上产生 trace 信息的接口。
- slave: 在 ATB 总线上接收 trace 信息的接口。

ATB 总线以 AT 作为信号的前缀。

如以下：HTM 和 xTM 是产生 trace 信息的 master，通过 ATB 总线，发送给 funnel，funnel 将收到的两路 ATB 数据合并成一路 ATB 数据发送给 replicator，replicator 再将接收的一路 ATB 数据，重复以两路 ATB 数据分别发送给 ETB 和 TPIU，TPIU 通过 trace port 发送到外部。



trace 信息，通过 ATB 总线进行传输。

- ETB: embedded trace buffer。片内存储 trace 数据的大容量设备。
- xTM: trace macrocells。从连接的处理器的中，追踪数据，产生 trace 信息输出。
包括两个:
 - ETM: embedded trace macrocell，可以产生指令 trace 信息，也可以产生数据 trace 信息
 - PTM: program flow macrocell，产生指令 trace 信息
- HTM: advanced high-performance bus trace macrocell。连接到 AHB 互联总线上，产生总线的 trace 信息
- TPIU: trace port interface unit。接收 ATB trace 数据，传输到 trace port 上，将 trace 信息输出到片外。
- trace replicator: 将一个 ATB 数据发送给独立的两个 ATB slave。
- trace funnel: 将多个 trace 源信息，合并成一个 trace 数据输出。

2.1 全局信号

时钟和复位信号。复位信号低电平有效。数据在时钟的上升沿采样。

Table 2-1 Global signals

Name	Master	Slave	Description
ATCLK	Input	Input	Global ATB clock.
ATCLKEN	Input	Input	Enable signal for ATCLK domain.
ATRESETn	Input	Input	ATB interface reset when LOW. This signal is asserted LOW asynchronously, and deasserted HIGH synchronously.

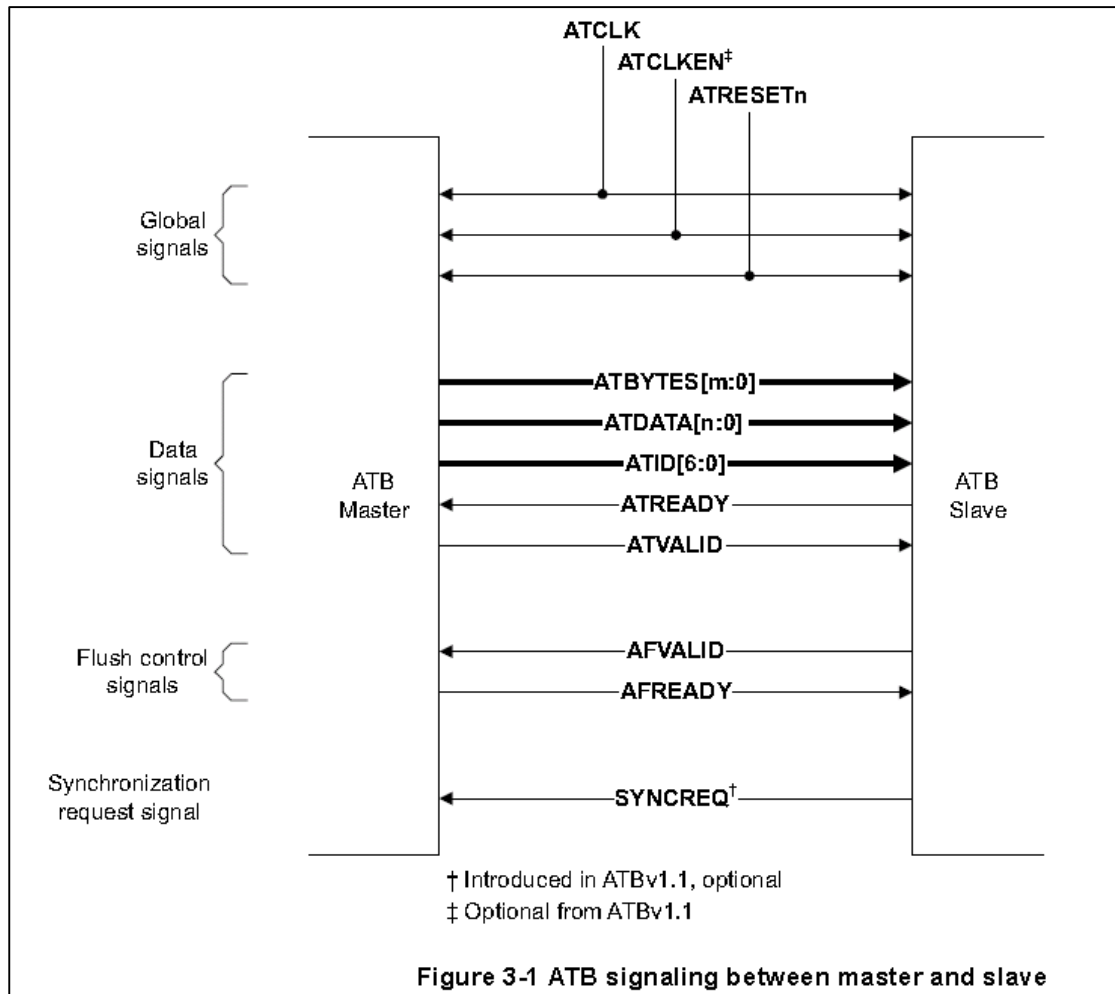
2.2 flow control

数据信号。

Table 2-2 Data signals

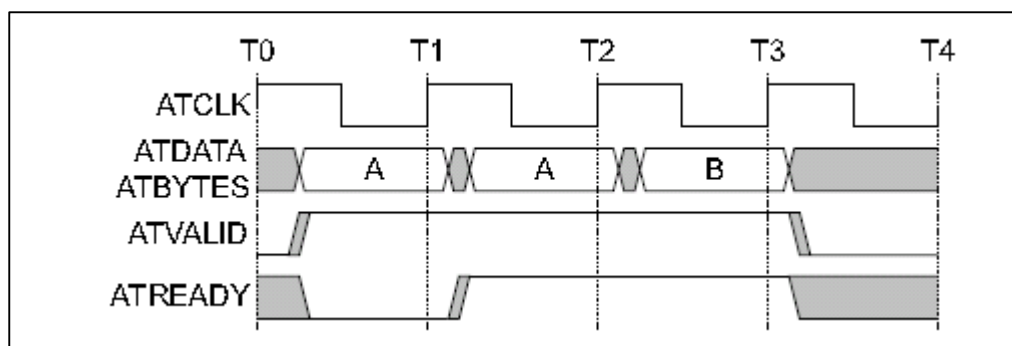
Name	Master	Slave	Clamp value	Description
ATBYTES[m:0] ^a	Output	Input	LOW	The number of bytes on ATDATA to be captured, minus 1.
ATDATA[n:0]	Output	Input	LOW	Trace data.
ATID[6:0]	Output	Input	LOW	An ID that uniquely identifies the source of the trace. See <i>ATIDs</i> on page 3-7.
ATREADY	Input	Output	HIGH	Slave is ready to accept data. See Chapter 3 <i>Flow Control</i> .
ATVALID	Output	Input	LOW	A transfer is valid during this cycle. If LOW, all other AT signals must be ignored this cycle. See Chapter 3 <i>Flow Control</i> .

master 和 slave 之间的通信，通过 **ATVALID** 和 **ATREADY** 作为握手信号。



master 将 ATVALID 拉高，表示 master 传输的数据有效。slave 将 ATREADY 信号拉高，表示 slave 准备好接收 master 的数据。如果 master 发送数据，发现 slave 的 ATREADY 没有拉高，那么就要将数据重新发送一遍。

时序图如下：



T1, master 发送数据 A，ATVALID 拉高。但是 master 检测到 ATREADY 没有拉高。

T2, master 检测到上一次数据传输，ATREADY 没有拉高，因此需要将数据 A 重新发送。此时 ATREADY 为 1，表示此时数据发送成功。因此下一次可以发送新的数据。

T3, master 发送数据 B，ATREADY 为 1，表示此时数据发送成功。下一拍可以发送新的数据。

T4, master 将 ATVALID 拉低，表示没有数据发送。

如果 slave 在 master 发送数据的时刻, 不能及时接收 master 发送的数据 (ATREADY 不能在 master 发送数据时刻拉高), 那么在 slave 端, 建议实现 internal buffer, 接收 master 发送的数据。然后再处理。这样的话, 当 buffer 没有满的话, ATREADY 信号就可以一直为高, master 就不用一直重新发数据。

对于 ATBYTES[m:0] 和 ATDATA[n:0]。

$$m = \log_2(n+1) - 4$$

ATDATA[n:0]	ATBYTES[m:0]
n = 7	ATBYTES not required
n = 15	m = 0
n = 31	m = 1
n = 63	m = 2
n = 127	m = 3

对于 ATID:

trace 的数据要伴随着 ATID 进行发送的。因为产生 trace 的组件有很多, 因此需要一个 ID 来识别这些组件, ID 就保存在 ATID 中发送。

ATID 是一个 7 位的值。0x00 和 0x70-0x7f 是 coresight 中规定的保留值, 在 soc 实现中, coresight trace 组件不应该使用这些 ID。

在一个 soc 中, 对于每个 trace 组件, ID 必须是唯一的。对于 ID, 有两种选择:

- fixed ID: 在 soc 设计的时候, 就将 ID 值给固定了。
- programmed ID: trace 组件的 ID 可以通过外部的 debugger 进行更改, 但是 debugger 要保证, 组件的 ID 必须是唯一的。

ATID 和 ATDATA, 在同一时刻被 slave 所接收。

2.3 flusing

一般, trace 数据都是从 trace source 发送到 trace sink。但是在某些情况下, trace sink 需要主动要求 trace source 发送数据, 此时就会用到 flusing。sink 发送 flush 信号给 source, source 将内部所有的 trace 数据全部发送出去, 发送完毕后, 回应 sink flush complete 信号。

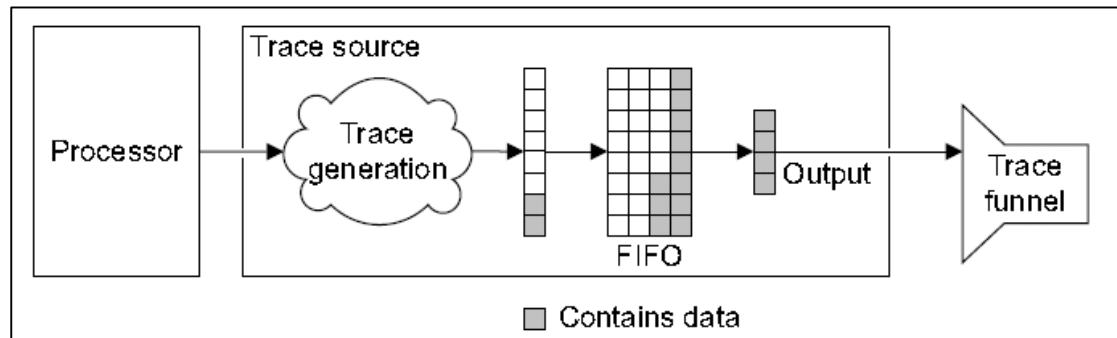
flush 使用以下两个信号来作为握手信号。

Name	Master	Slave	Clamp value	Description
AFVALID	Input	Output	LOW	This is the flush signal. All buffers must be flushed because trace capture is about to stop. See <i>Buffer flush</i> on page 4-2.
AFREADY	Output	Input	HIGH	This is a flush acknowledge. Asserted when buffers are flushed. See <i>Buffer flush</i> on page 4-2.

如下图: trace source 追踪处理器的数据, 通过 trace generation, 发送到下一级的 buffer 中, 然后 buffer 将数据发送给 FIFO 中进行暂存。FIFO 中一旦有数据, FIFO 就负责将数据通

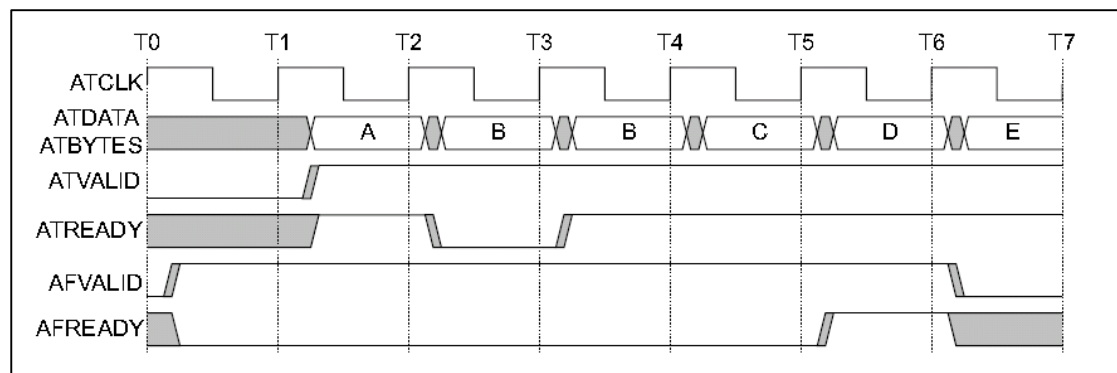
过 output 发送给 trace funnel。

因此对于一个 trace 组件，中间是会存在多级的 buffer 对数据进行缓存的。



假设此时系统要 power down，而 source 的 FIFO 中还有 trace 信息，那这些 trace 信息，就要在 power down 之前，发送出去。此时 funnel 就可以向 source 发送 flush 信息，也就是将 AFVALID 信号拉高，trace source 接收到该信号，就将自己 FIFO 中的所有剩余 trace 信息，全部发送给 funnel，发送完毕后，将 AFREADY 信号拉高，表示数据发送完毕。此时，就可以将系统给 power down。

时序图：



T1 时刻，AFVALID 为高，表示要求 master 进行 flush 操作。

T2 开始，就进行 master 的 FIFO 的 trace 数据排空操作。

从 T2-T6，master 将自己内部的 FIFO 数据发送出去。在 T6 时刻，master 将 AFREADY 信号拉高，表示 master 将数据发送外部，也就是 flush 完成。

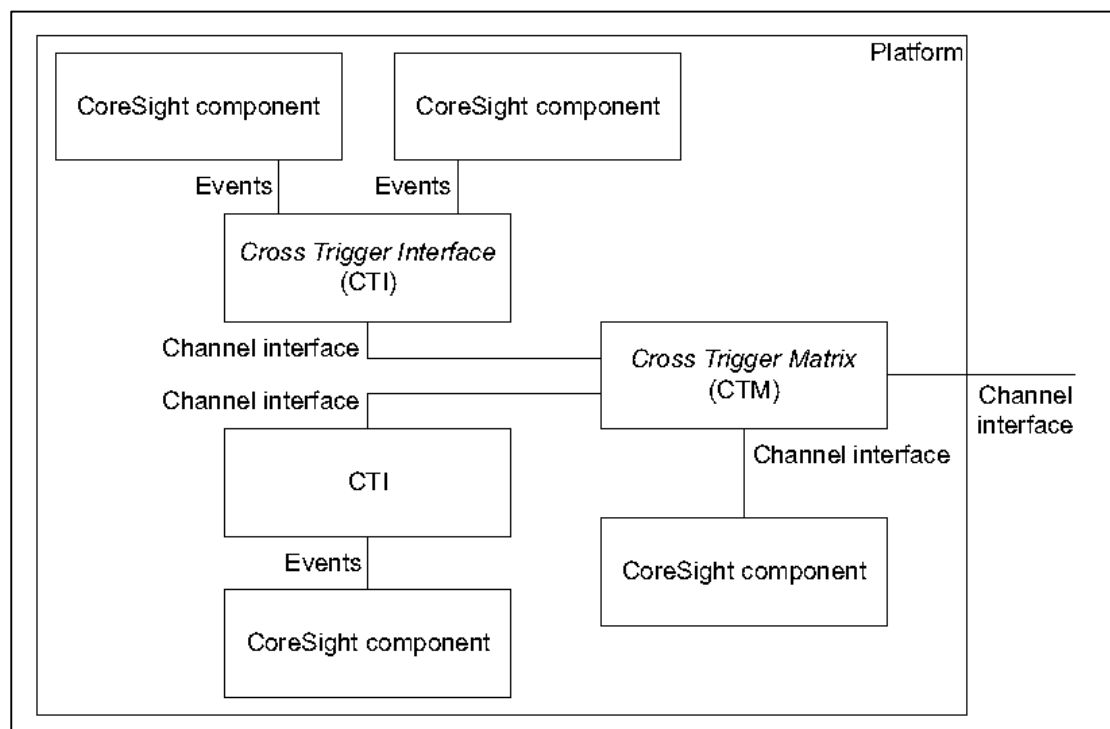
Table 4-1 Flush events	
Clock cycle	Event
T1-T2	Flush requested. A, B, C, in FIFO.
T2	Draining begins.
T3-T5	Stalls still possible.
T6	Flush complete. Output continues.
T6	AFVALID deasserted.

四、 channel interaface

channel interface 是用来使不同 coresight 组件之间传递 event 使用。使用两个组件来实现:

- CTM: cross trigger matrix, 接收 CTI 的 channel 信号, 然后广播给其他 CTI
- CTI: cross trigger interface, 接收 trigger 信号, 发送 trigger 信号, 接收 channel 信号, 发送 channel 信号

channel interface 的典型应用。



每个 coresight 组件和对应的 CTI 相连，那这个 CTI 就可以采集组件的 trigger 信号，或者发送 trigger 信号给组件。

CTI 将接收的 trigger 信号，发送到 channel interface 上，或者从 channel interface 上接收 trigger 信号，发送给和自己相连的 coresight 组件。

所有的 CTI 的 channel interface 和 CTM 相连，这样各个 CTI 就可以通过 CTM，相互传输 trigger 信号。

1. channel interface 信号

channel interface 上传递的信号，使用握手机制进行信号的传递。不同 CTI 运行的时钟是不一样的，因此需要握手机制，来保证不同 CTI 之间能够相互传递 event。

信号总共有以下 4 个，direction 是相对 CTI 而言的。

Table C4-1 Asynchronous channel interface signals

Name	Direction	Clamp value	Description
CHIN[n-1:0]	Input	-	Channel input
CHINACK[n-1:0]	Output	1	Channel input acknowledge
CHOUT[n-1:0]	Output	0	Channel output
CHOUTACK[n-1:0]	Input	-	Channel output acknowledge

clamp value，是指设备 power down 或者 disable，output 上的固定值。

以下是时序图：

Figure C4-3 shows how the asynchronous interface uses a basic 4-phase handshaking protocol. The same protocol is used by **CHOUT** and **CHOUTACK**.



Figure C4-3 Channel interface handshaking

两种连接方式，异步的和同步的。

Figure C4-4 shows how to connect two asynchronous channel interfaces together.



Figure C4-4 Asynchronous channel interface connection

Figure C4-5 shows how to connect two synchronous channel interfaces together.

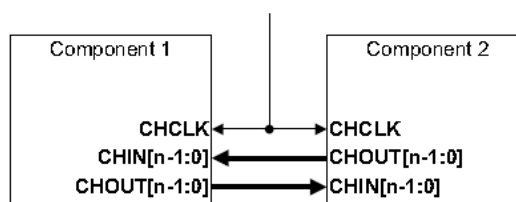
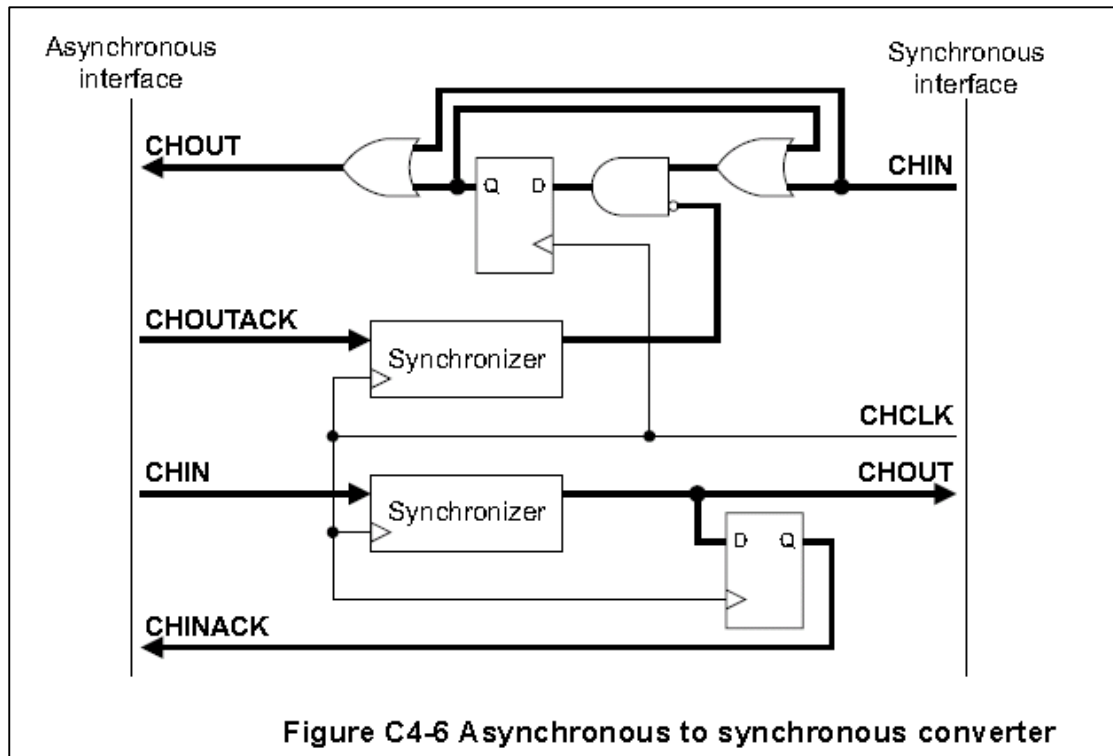


Figure C4-5 Synchronous channel interface connection

同步与异步接口之间的转换。



2. CTI

CTI 有 input trigger 和 output trigger。ARM 对这些 trigger 均有定义：

以下是 output trigger:

Numbers	Source	Destination	Event description
0	CTI	Processor	<i>Debug Request trigger event on page 182</i>
1	CTI	Processor	<i>Restart Request trigger event on page 183</i>
2	CTI	IRQ controller	<i>Generic CTI Interrupt trigger event on page 184</i>
3	-	-	Reserved
4, 5, 6, 7	CTI	Trace extension	<i>OPTIONAL Generic Trace External Input trigger events on page 183</i>

Table 57: Allocation of CTI output trigger events

trigger0: 连接 CPU, debug request 请求, 当这个信号有效, 表示使连接的 CPU 进入 debug state。

trigger1: 连接 CPU, restart request 请求, 当这个信号有效, 可以使连接的 PE 退出 debug state。

trigger2: 连接中断控制器, CTI interrupt, 当这个信号有效, 会发出 CTIIRQ 信号, 给中断控制器发送中断。

以下是 input trigger:

Numbers	Source	Destination	Event description
0	Processor	CTI	<i>Cross-halt trigger event on page 183</i>
1	Processor	CTI	<i>Performance Monitors Overflow trigger event on page 183</i>
2, 3	-	-	Reserved
4, 5, 6, 7	Trace extension	CTI	<i>OPTIONAL Generic Trace External Output trigger events on page 184</i>

Table 58: Allocation of CTI input trigger events

trigger0: 连接 CPU, cross-halt trigger, 当这个信号有效, 表示连接的 PE 进入 debug state。

以下是 CTI 的内部结构:

CTI 左边连接 CPU, 有 2 个通道, 一个是接收 CPU 发送的 input trigger, 一个是发送给 CPU 的 output trigger。

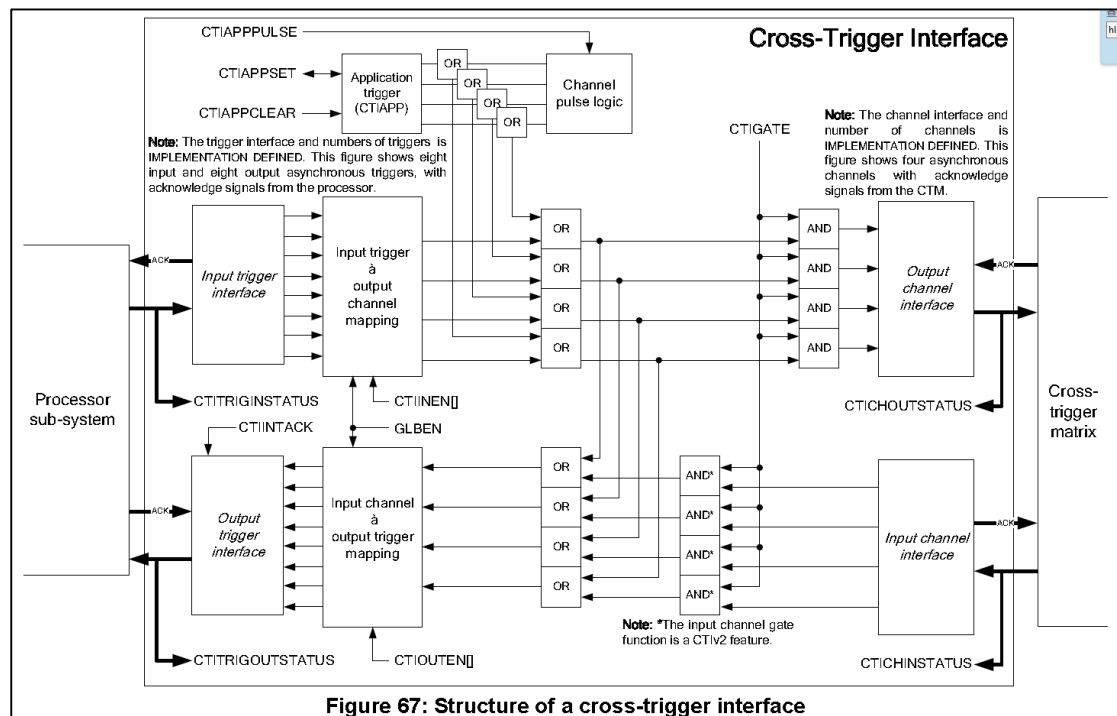
CTI 右边连接 CTM, 用于发送 output channel, 以及接收 input channel。

CTI 接口信号共有 4 种:

- Input triggers: trigger event inputs from the processor to the CTI。trigger 信号从 PE 发出给 CTI。
- Output triggers: trigger event outputs from the CTI to the processor。trigger 信号从 CTI 发出给 PE。
- Input channels: channel event inputs from the CTM(cross-trigger matrix) to CTI。channel 信号从 CTM 发出给 CTI。
- Output channels: channel event outputs from CTI to CTM。channel 信号从 CTI 发出给 CTM。

input trigger & output channel mapping: 根据 CTIINEN 信号, 将 input trigger, 连接到指定的 output channel 上。

input channel & output trigger mapping: 根据 CTIOUTEN 信号, 将 input channel, 连接到指定的 output trigger 上。



对于每一个 output trigger, 有单独的寄存器和其对应, 如 CTIOUTEN0, 就和 output trigger0 对应, 控制该寄存器, 就可以将指定的 input channel 连接到 output trigger0 上, 这样指定的 input channel 上的 trigger 就可以传递到 output trigger0 上。

16.14.7 Input Channel to Output Trigger Enable registers, CTIOUTEN_n

The Input Channel to Output Trigger Enable registers are read/write registers which define which input channels generate trigger outputs. There is one register for each of the trigger outputs. If output trigger n is not connected, the behavior of CTIOUTEN_n is IMPLEMENTATION DEFINED. These registers do affect the mapping from application trigger to trigger outputs.

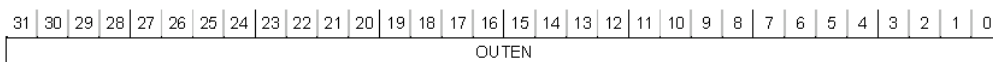


Figure 95: CTIOUTEN_n register bit assignment

OUTEN[x], bit [x]

If $x \geq \text{CTIDEVID.NUMCHAN}$, the number of implemented ECT channels, OUTEN[x] is RAZ/WI.

Otherwise, permitted values for OUTEN[x] are:

- 0 An event on input channel x will not cause output trigger n to be asserted.
- 1 An event on input channel x will cause output trigger n to be asserted.

寄存器有 32 位，表示可以从 32 个 input channel 中选择一个（armv8 的输出 trigger 最多 32 个）。如果寄存器的值为 0x1，就表示将 output trigger0 连接到 input channel0，如果寄存器的值为 0x4，就表示 output trigger0 连接到 channel2。

对于 CTI，可以接收三种输入信号，一个是 core 的 trigger inputs，一个是 CTM 的 input channel，另一个就是外部 debugger 通过 APB 访问的寄存器从而产生的 trigger 信号。

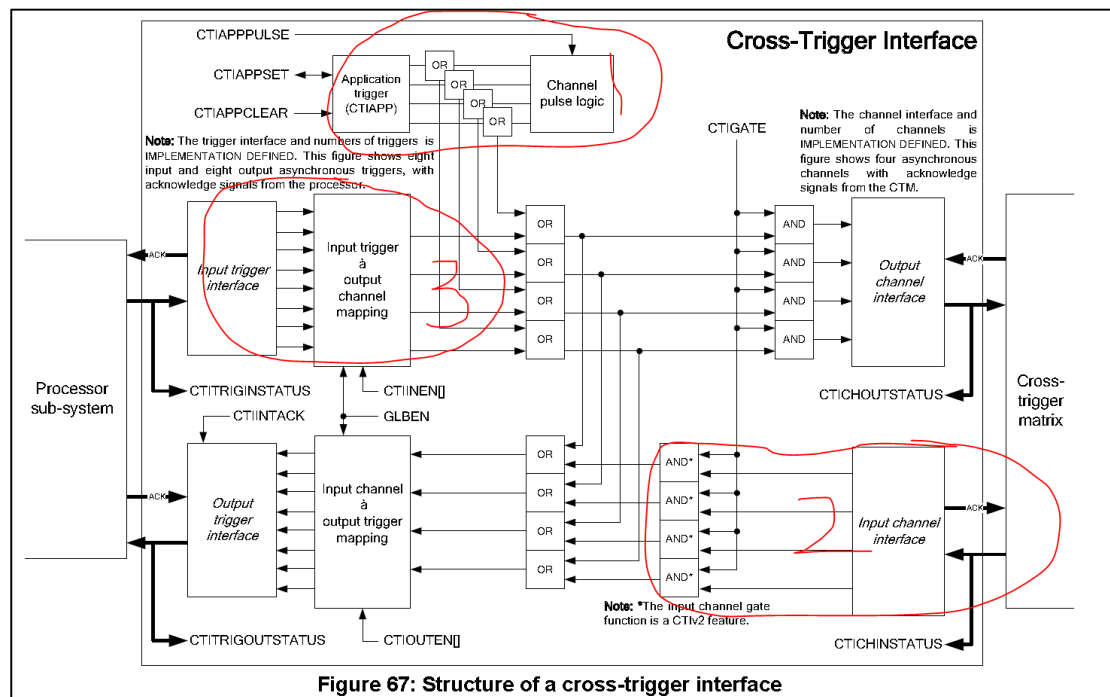


Figure 67: Structure of a cross-trigger interface

总共有 3 个来源：

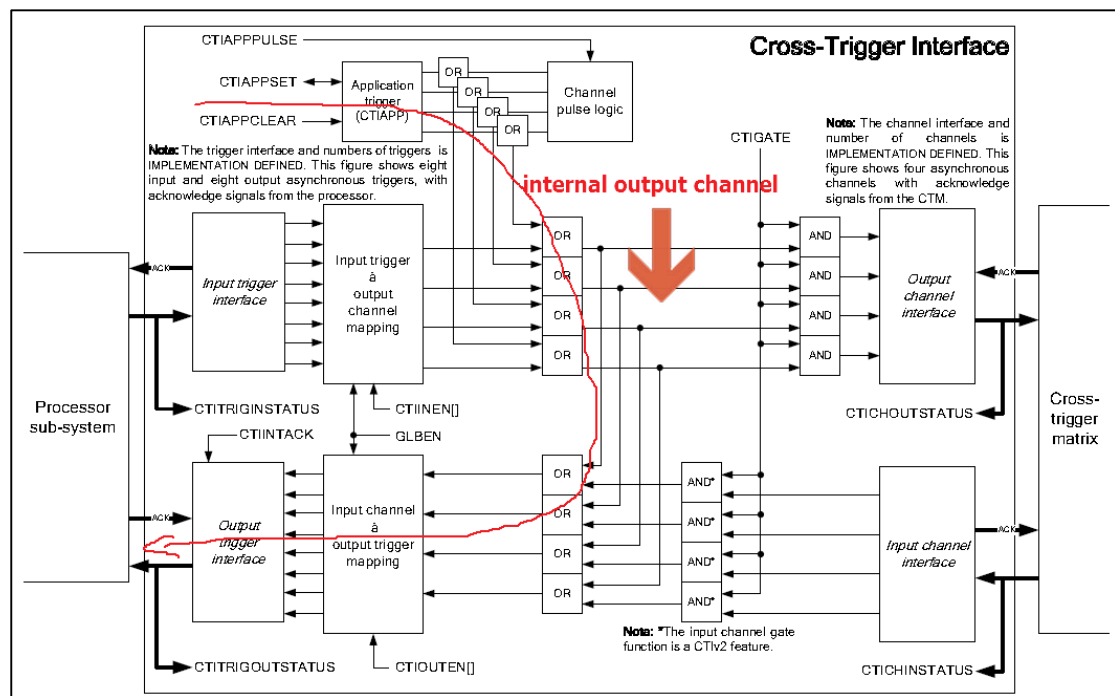
- CTI 内部逻辑的 CTIAPP，外部可以通过 APB 总线访问
- 外部的 CTM
- CPU 发送的 input trigger

2.1 对于 CIT 内部逻辑的 CTIAPP

这部分提供了若干个寄存器，外部通过 APB 总线可以访问这些寄存器，从而控制发生，取消 trigger 信号。

外部的 debugger 可以控制这些寄存器，从而使对应的 internal output channel 有效，然后控制 CTIOUTEN[]，就可以使 internal output channel 的数据输出到指定的 output trigger 上。

以下是 trigger 走向图。



armv8 定义了 3 个寄存器，CTIAPPSET，CTIAPPCLEAR，CTIAPPULSE。

- CTIAPPSET: 指定的 internal channel 上产生 channel event
- CTIAPPCLEAR: 指定的 internal channel 上取消 channel event
- CTIAPPULSE: 指定的 internal channel 上产生只持续一个周期的 channel event

16.14.3 Application Trigger Set register, CTIAPPSET

The Application Trigger Set register is read/write. A write to this register causes bits of the Application Trigger register, CTIAPPTRIG, to be set to 1. CTIAPPTRIG generates channel events. A read from this register returns the current value of CTIAPPTRIG.

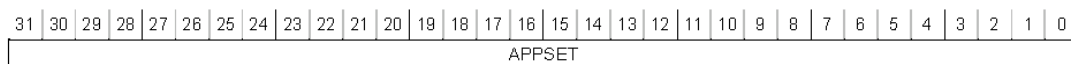


Figure 91: CTIAPPSET register bit assignment

APPSET[x], bit [x]

If $x \geq \text{CTIDEVID.NUMCHAN}$, the number of implemented ECT channels, APPSET[x] is RAZ/WI.

Otherwise, the behavior on reads and writes of APPSET[x] is:

	On reads	On writes
0	Application trigger inactive	No effect.
1	Application trigger active.	Set CTIAPPTRIG[x] to 1 and generate the corresponding channel event.

If the ECT does not support multicycle channel events, use of CTIAPPSET is deprecated and the debugger should only use CTIAPPULSE.

16.14.4 Application Trigger Clear register, CTIAPPCLEAR

The Application Trigger Clear register is write-only. A write to this register causes bits of the Application Trigger register, CTIAPPTRIG, to be cleared to 0.

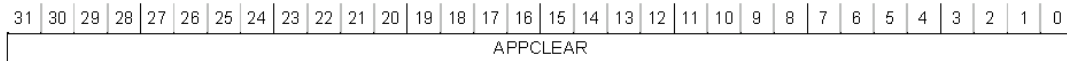


Figure 92: CTIAPPCLEAR register bit assignment

APPCLEAR[x], bit [x]

If $x \geq \text{CTIDEVID.NUMCHAN}$, the number of implemented ECT channels, APPCLEAR[x] is W1. Otherwise, the behavior on writes to APPCLEAR[x] is:

- 0 No effect.
- 1 Clear CTIAPPTRIG[x] to 0 and clear the corresponding channel event.

If the ECT does not support multicycle channel events, use of CTIAPPCLEAR is deprecated and the debugger should only use CTIAPPULSE.

16.14.5 Application Pulse register, CTIAPPULSE

The Application Pulse register is write-only. A write to this register causes an event pulse to be generated, on the channel corresponding to the bit written to.

It is **CONSTRAINED UNPREDICTABLE** whether a write to CTIAPPULSE generates an event on a channel if CTICONTROL.GLBEN is zero.



Figure 93: CTIAPPULSE register bit assignment

APPPULSE[x], bit [x]

If $x \geq \text{CTIDEVID.NUMCHAN}$, the number of implemented ECT channels, APPPULSE[x] is W1. Otherwise, the behavior on writes to APPPULSE[x] is:

- 0 No effect.
- 1 Channel x event pulse generated for one clock period.

Notes:

- The CTIAPPULSE operation does not affect the state of the Application Trigger register, CTIAPPTRIG. If the channel is active, either because of an earlier event or from the application trigger, then the value written to CTIAPPULSE has no effect.
- The clock period for a pulse may be unrelated to any other system clock period, for example, the processor clock period. Although the channel event is a single cycle inside the CTI, it may be extended to a longer pulse by the **IMPLEMENTATION DEFINED** trigger and channel interfaces. Multiple pulse events that occur close together might be merged into single pulse events.

CTIINTACK。这个寄存器，是用来控制将 output trigger 上的信号给无效掉的。比如，当前 output trigger0 是有效的，表示 debug request，那么如果要将这个 debug request 给无效掉的话，那么就可以往这个寄存器写 0x1，就无效掉了。

寄存器的每一位对应相应的 output trigger。

- CTIINTACK[0] 对应 output trigger0
- CTIINTACK[1] 对应 output trigger1
- CTIINTACK[2] 对应 output trigger2
- ...
- CTIINTACK[31] 对应 output trigger31

因为 ARMv8 规定了，output trigger 的最大数目是 32，因此也就用一个 32 位的寄存器就可以表示所有了。

16.14.2 Output Trigger Acknowledge register, CTIINTACK

The Output Trigger Acknowledge register, CTIINTACK, is write-only. Writing 1 to a bit of CTIINTACK creates a soft acknowledge for an output trigger, deactivating the trigger.

Note: Whether an output trigger requires a soft acknowledge is a property of the trigger.

A soft acknowledge of entry to Debug state to clear a Debug Request trigger event from the CTI is required before it is possible to restart the processor.

A debugger must read CTITRIGOUTSTATUS to confirm that the output trigger has been acknowledged before generating any event that must be ordered after the write to CTIINTACK.

Example: A debugger writes to CTIINTACK to clear an Debug Request trigger event from the CTI and then writes to CTIAPPULSE to generate a Restart Request trigger event. These writes complete in order. A read of CTITRIGOUTSTATUS is required between the two writes to force the clearing of the first trigger event to happen strictly before the second trigger event. Otherwise they might happen simultaneously, leading to UNPREDICTABLE behavior.

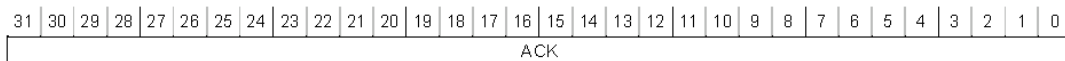


Figure 90: CTIINTACK register bit assignment

ACK[n], bit [n]

If any of the following is true, writes to ACK[n] are ignored:

- $n \geq \text{CTIDEVID.NUMTRIG}$, the number of implemented triggers
- output trigger n is not active
- the channel mapping function output, as controlled by CTIOUTEN n , is still active.

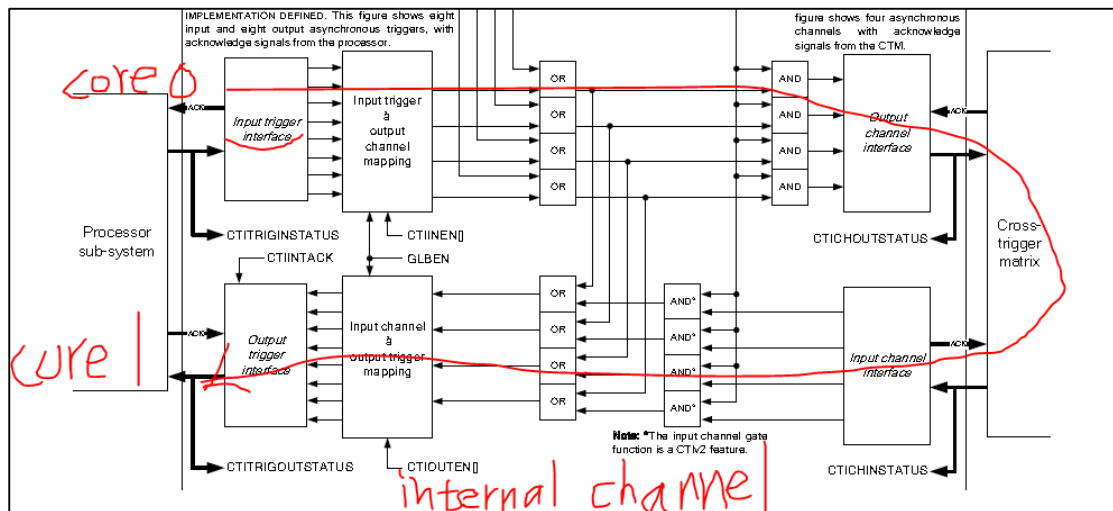
Otherwise it is IMPLEMENTATION DEFINED whether writes to ACK[n] are ignored if any of:

- output trigger n is not implemented
- output trigger n is not connected
- output trigger n is self-acknowledging and does not requires software acknowledge.

Otherwise, the behavior on writes to ACK[n] is:

- 0 No effect.
- 1 Deactivate the trigger.

2.2 外部的 CTM



从 core0 发出的 input trigger 信号，通过 output channel，然后通过 CTM 到达其他所有 core 的 output trigger 信号上的。

从 core0 发出 input trigger 信号，然后通过 input trigger mux（通过 CTIINEN[] 选择），到相应的 internal input channel 上，如果后面的 AND 与门打开，也就是 CTIGATE 信号有效，那么 channel 上的 trigger 信号就会通过 output channel interface 传输到 CTM。

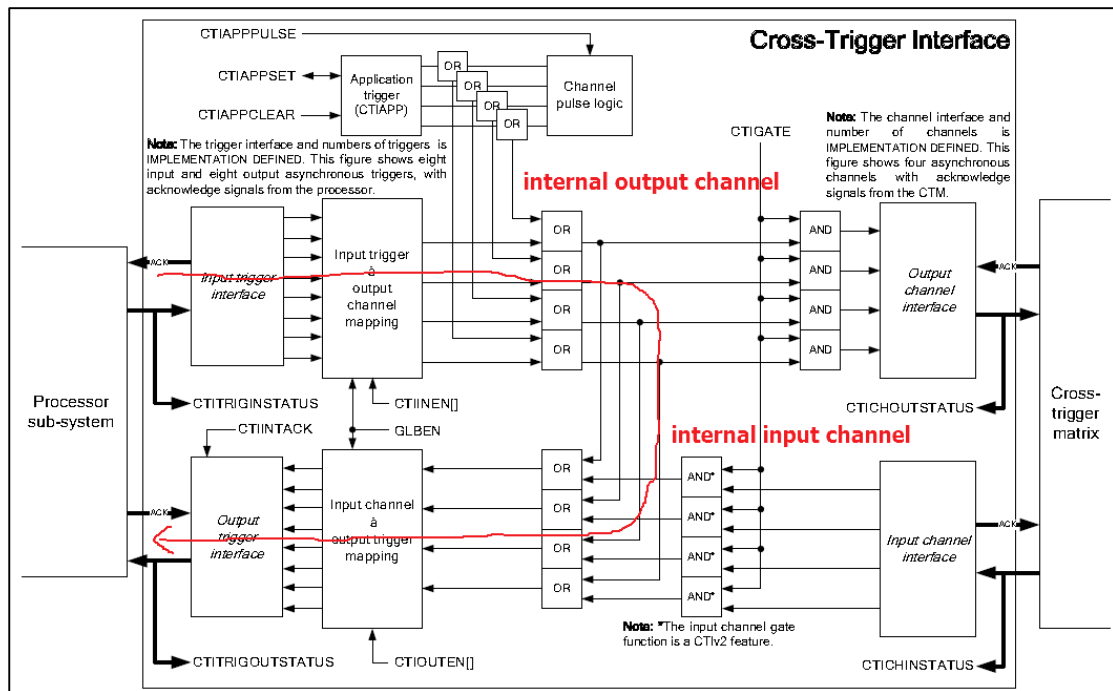
CTM 将 trigger，发送给其他 core（这里是 core1）的 CTI 外部 input channel 上，如果其

他 core 的 AND 与门打开, 也就是 CTIGATE 信号有效, 就会传递到 CTI 的 internal input channel 上, 通过 output trigger mux (通过 CTIOUTEN[] 选择), 通过 output trigger, 发送给 core1。

以上就是 CTI 的不同 core 之间的 trigger 信号传递的机制。通过该机制, 可以使一个 core 给另外的 core 发 trigger 信号, 从而可以控制使其他 core 进入 debug state, 退出 debug state 等。

2.3 CPU 发送的 input trigger

接收 CPU 的 input trigger, 通过内部的 output channel, 发送给内部的 input channel 上, 然后在发送到 output trigger 上。



五、 rom table

在一个 soc 中, 有多个 coresight 组件, 但是软件怎么去识别这些 coresight 组件, 去获取这些 coresight 组件的信息了? 这个时候, 就需要靠 coresight 组件中, 一个重要的组件, 这个组件就是 rom table。

ARM 规定, 在一个 soc 中, 必须要实现至少 1 个 rom table, 该 rom table, 保存了 soc 中各个 coresight 组件的信息, 包括组件格式以及组件的基地址。

rom table 只会占用一个 4K 空间大小, 也就是 PIDR4 寄存器的 SIZE 为 0。

3. entry 寄存器

对于 rom table, 从 0x000-0xefc, 是 entry 寄存器。每个 entry 保存了一个 coresight 组件的信息。

Bits	Name	Description
[31:12]	Address offset	Base address of the highest 4KB block for that component, relative to the ROM address. Negative values are permitted using twos complement. ComponentAddress = ROMAddress + (AddressOffset SHL 12).
[11:9]	-	RES0.
[8:4]	Power domain ID	Indicates the power domain ID of the component. This field: <ul style="list-style-type: none"> Is only valid if bit[2]==1, otherwise this field must be RAZ. Supports up to 32 power domains using values 0x00 to 0x1F.
[3]	-	RES0.

[2]	Power domain ID valid	Indicates if the Power domain ID field contains a power domain ID: 0 = Power domain ID not provided. 1 = Power domain ID provided in bits[8:4].
[1]	Format	0 = 8-bit format. 1 = 32-bit format. For valid ROM table entries, as indicated by bit[0]==1, this bit is always 1.
[0]	Entry present	This bit indicates whether an entry is present at this location in the ROM table. Its possible values are: 0 = Entry not present. 1 = Entry present. See <i>Empty entries and the end of the ROM table</i> for more information.

[31:12]是 coresight 组件，基于 ROM table 基地址的偏移地址。例如 ROM table 的基地址为 0x2000_0000，而[31:12]为 1，那么这个 coresight 组件的基地址就是 0x2000_1000。

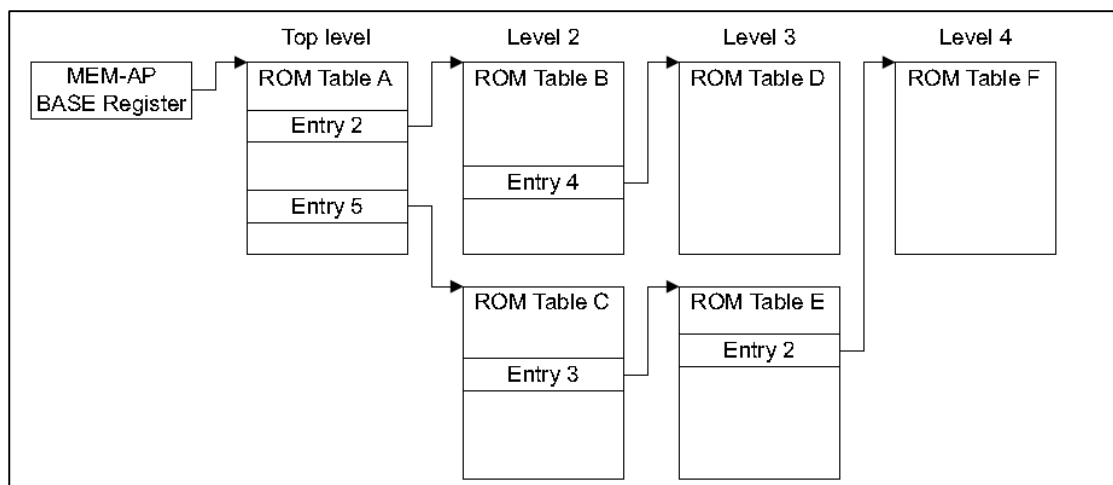
[8:4]和[2]是用来说明该 coresight 组件所处的 power domain。因为一个 soc 中，有多个 power domain，而每个组件可能会处于不同的 power domain 中。就依靠这两个位域说明。

[1]，表示 coresight 的寄存器的数据有效是 8bit，还是 32bit。

[0]，表示这个 entry 代表的组件是否有有效的。

一个 rom table 中，最多有 960 个 entry，也就是一个 rom table 中，可以最多保存 960 个 coresight 组件的信息，但是在一个 entry 中，可以指向下一个 rom table，这样，就扩展了保存 coresight 组件信息的个数。

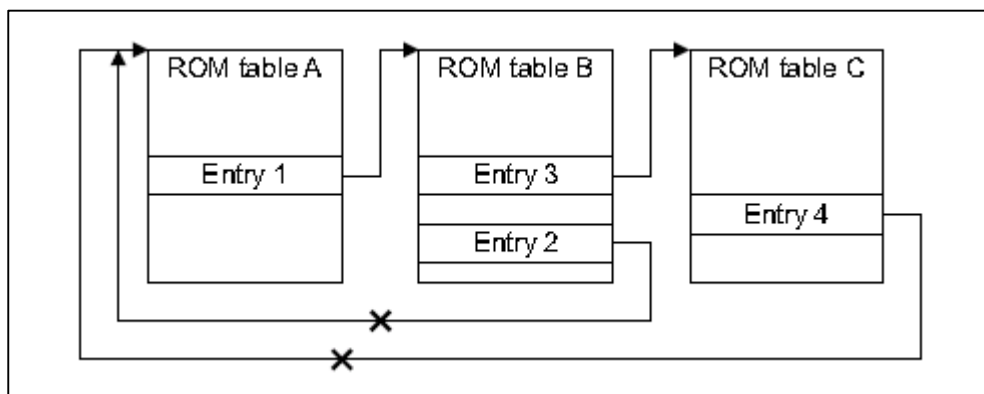
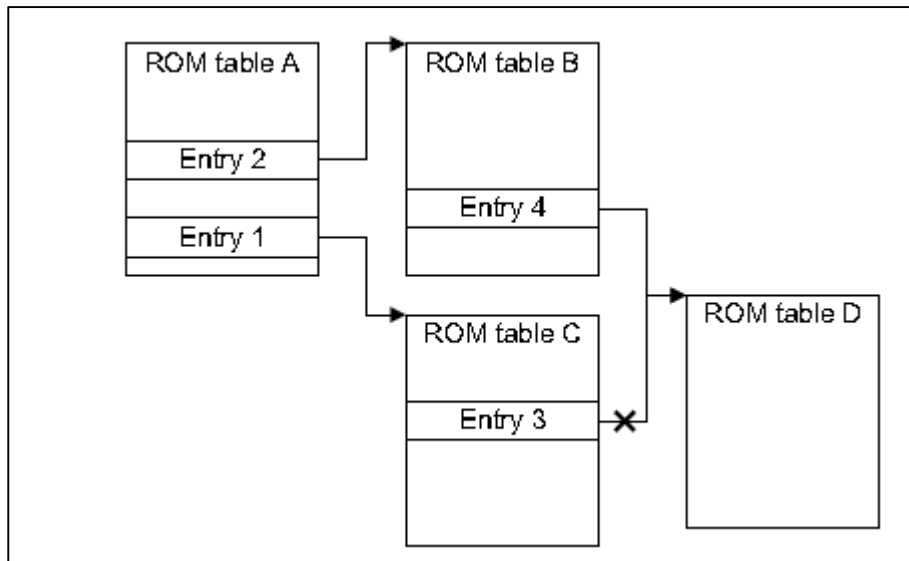
如果 rom table 的 entry 没有用完，那最后一个有效的 entry 后的下一个 entry，entry 值为全 0。



rom table 的基地址，保存在 DAP 的 AP 的 base addr 寄存器中，这样 debugger 通过访问 DAP 的 AP，就可以获取到 rom table 的基地址，然后在访问 rom table，从而获取到整个 soc

中所有的 coresight 组件的信息。

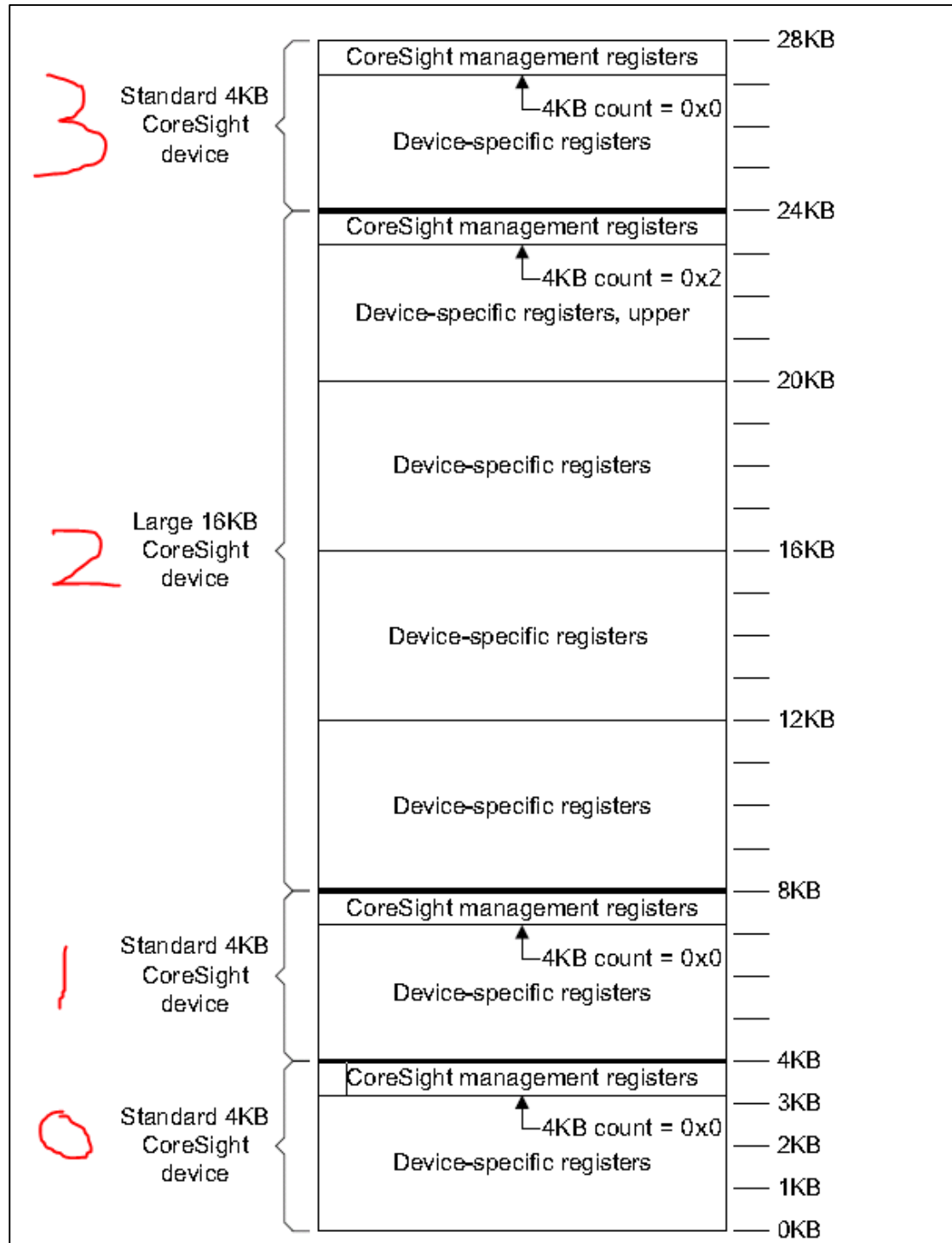
rom table 的 entry 指向,可以理解是一个链表,但是链表中,不能有环。如以下的 entry 指向是错误的。



如果一个 coresight 组件,占用的空间,超过了 4K,但 coresight 有规定,coresight 的寄存器,要实现在最后一个 4K 的最后 1K 位置,因此 rom table 中的该 coresight 组件的基地址,为最后一个 4K 空间的基地址。

4. 例子

例如如下的 coresight 系统,共 4 个组件,假设第一个组件是 rom table。假设 rom table 的基地址是 0x8000_0000。



那么:

组件	基地址	占用 4K 空间个数
rom table	0x8000_0000	1
组件 1	0x8000_1000	1
组件 2	0x8000_2000	4
组件 3	0x8000_6000	1

rom table 的基地址, 存在 DAP 的 AP 的 base addr 寄存器中, 外部通过访问 DAP 的这个寄存器, 获取到 rom table 的基地址, 然后就可以访问 rom table 各个 entry 寄存器的值。

组件 1, 组件 2, 组件 3 的基地址信息, 存放在 rom table 的 entry0, entry1, entry2 中。

对于组件 1, entry0 的[31:12]为 1, 表示组件 1 的基地址是 0x8000_1000, 外部根据这个地址, 就可以访问这个组件的 coresight 寄存器, 从而获取到这个组件的信息。

对于组件 2, 因为这个组件, 占用了 4 个 4K 空间大小, rom table 中存放占用最后 1 个 4K 空间的基地址, 因此 entry0 的[31:12]为 5, 表示组件 1 的基地址是 0x8000_5000, 外部根据这个地址, 就可以访问这个组件的 coresight 寄存器, 从而获取到这个组件的信息。通过读取 PIDR4 寄存器的 SIZE 信息, 获取到该组件占用 4 个 4K 空间, 从而反推, 可以得到该组件的基地址是 0x8000_2000。

对于组件 3, entry0 的[31:12]为 6, 表示组件 1 的基地址是 0x8000_6000, 外部根据这个地址, 就可以访问这个组件的 coresight 寄存器, 从而获取到这个组件的信息。

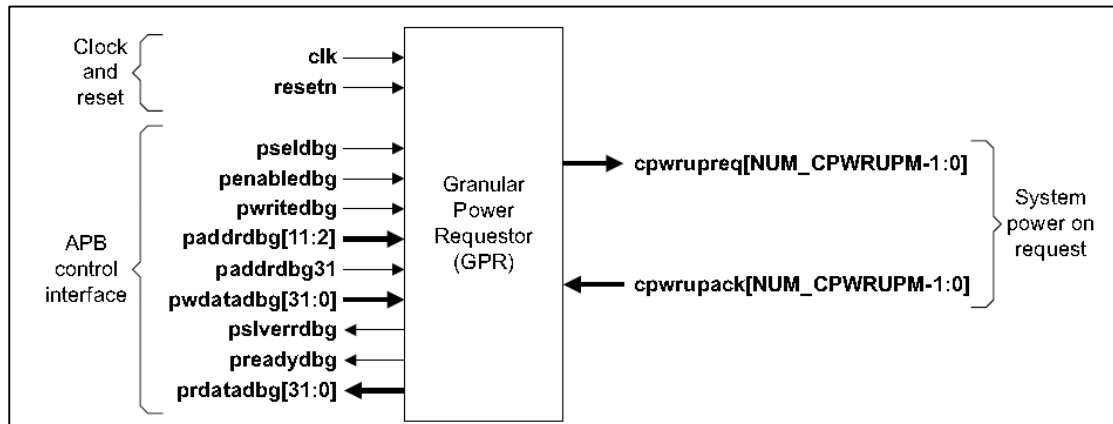
这样, 外部就通过 rom table, 就可以获取到 soc 中, 所有 coresight 组件的基地址。有了基地址, 就可以对其进行访问。

六、 power requestor

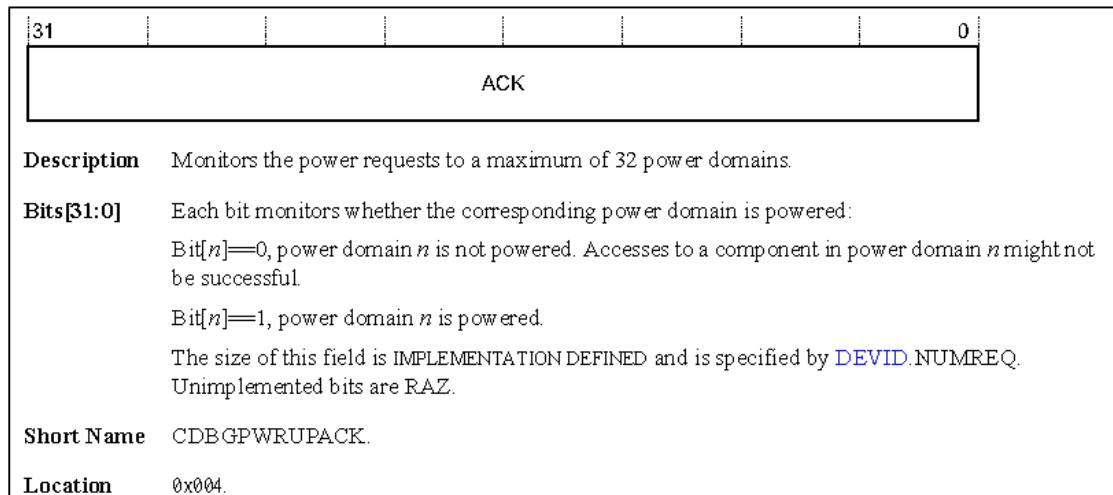
power requestor 属于 coresight 组件。这个组件用来控制系统的 power domain, 最多可以控制 32 个。

如果没有 power requestor, 通过 DAP, 只能对整个 coresight 系统进行上下电操作, 但是有了 power requestor, 可以对某些关心的组件, 进行上下电操作, 实现 power 的精细操作。

以下是 power requestor 的框图, 通过 apb 总线访问该组件, 该组件通过 cpwrupreq 信号, 向系统 power 发送请求, 通过 cpwrupack 获取到系统 power 的状态。



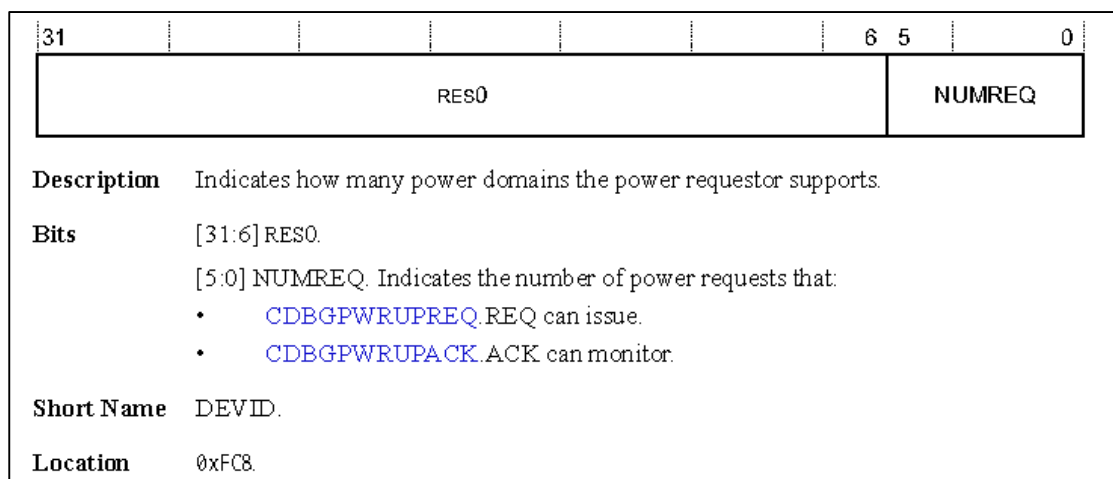
以下是 power requestor 的寄存器。



如这个寄存器值为 0x3，表示 domain0 和 domain1 是上电的。

3. DEVID

这个寄存器的低 6bit，保存了系统中有多少个 power domain。



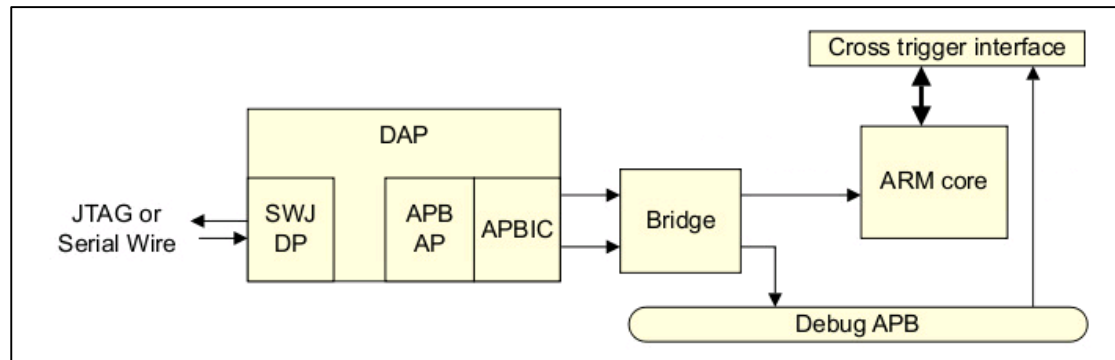
七、coresight 的两大功能

coresight 具有两大功能，一个是 debug，一个是 trace。

1. debug

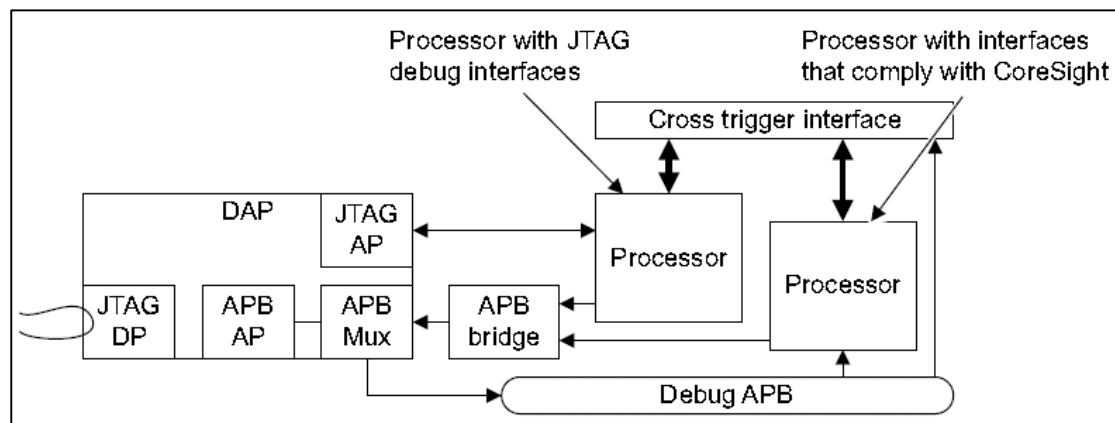
debugger 通过 DAP，来实现 debug 功能。

1.1、单 core 的 debug 系统:



一个 DAP，加上一个 AP 和 APBIC。外部对 DP 访问，DAP 将 DP 访问，转化为 AP 访问，AP 通过 APBIC，生成 AP 总线，通过 bridge，对 ARM core 中的 debug 资源，或者挂接在 debug APB 上的 coresight 组件，进行访问。

1.2、多 core 的 debug 系统:

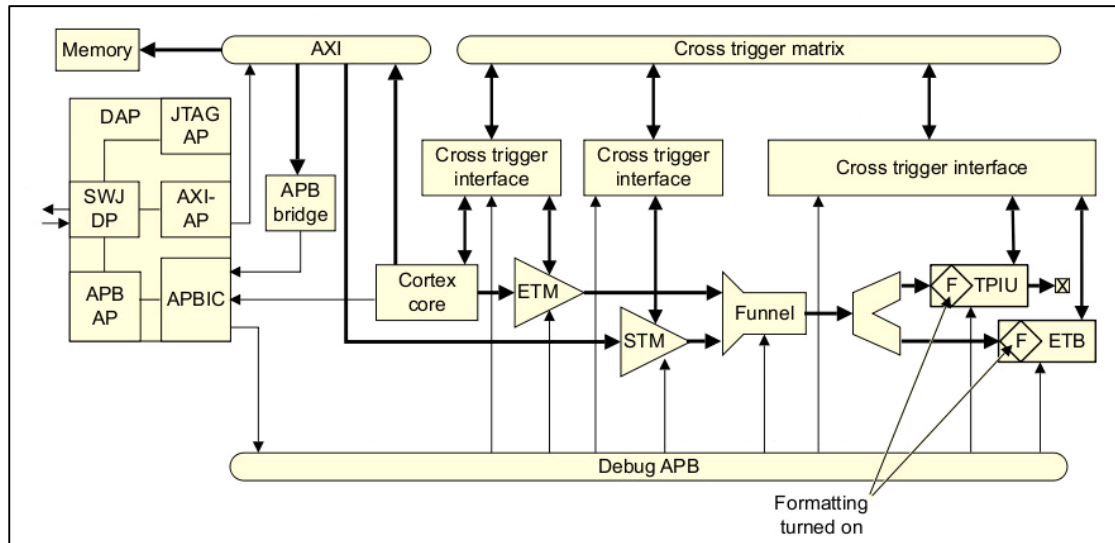


一个 DAP，DAP 内实现了多个 AP，这些 AP 实现 jtag 或 memory-mapped 方式访问 debug 资源。

外部对 DP 访问，DAP 将 DP 访问，转化为 APB AP 或者 JTAG AP 访问，如果是 jtag 访问，直接通过 JTAG AP，以 jtag 方式，对连接到该 jtag 上的处理器进行访问。

如果是 APB 访问，通过 APX mux，判断对处理器访问呢，还是挂接在 debug APB 总线上的 coresight 组件进行访问，如果对处理器访问，经过 APB bridge 对处理器进行访问。如果对挂接在 debug APB 总线上的 coresight 组件进行访问，那么就直接进行访问。

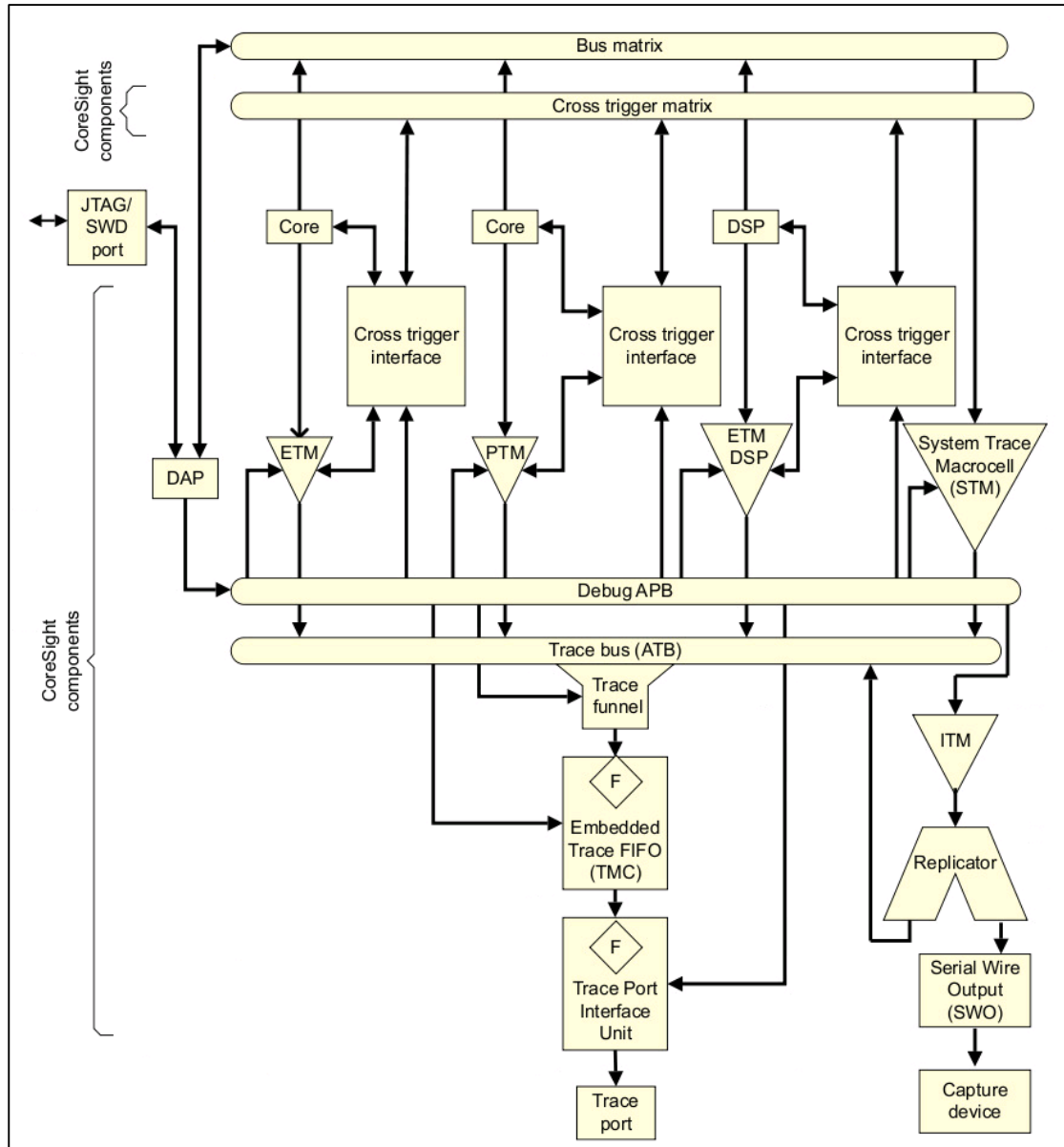
下图是 debug 组件，在 coresight 系统中的位置。



有多个 trace 源 (ETM, STM), 因此中间需要 trace link 组件 (funnel), funnel 合并 trace 信息, 发送给 replicators, replicators 再分发给 trace sink (TPIU, ETB)。因为有 trace link 组件, 因此 trace sink 需要 trace formatters, 将 trace 数据格式化, 得到真正的数据。

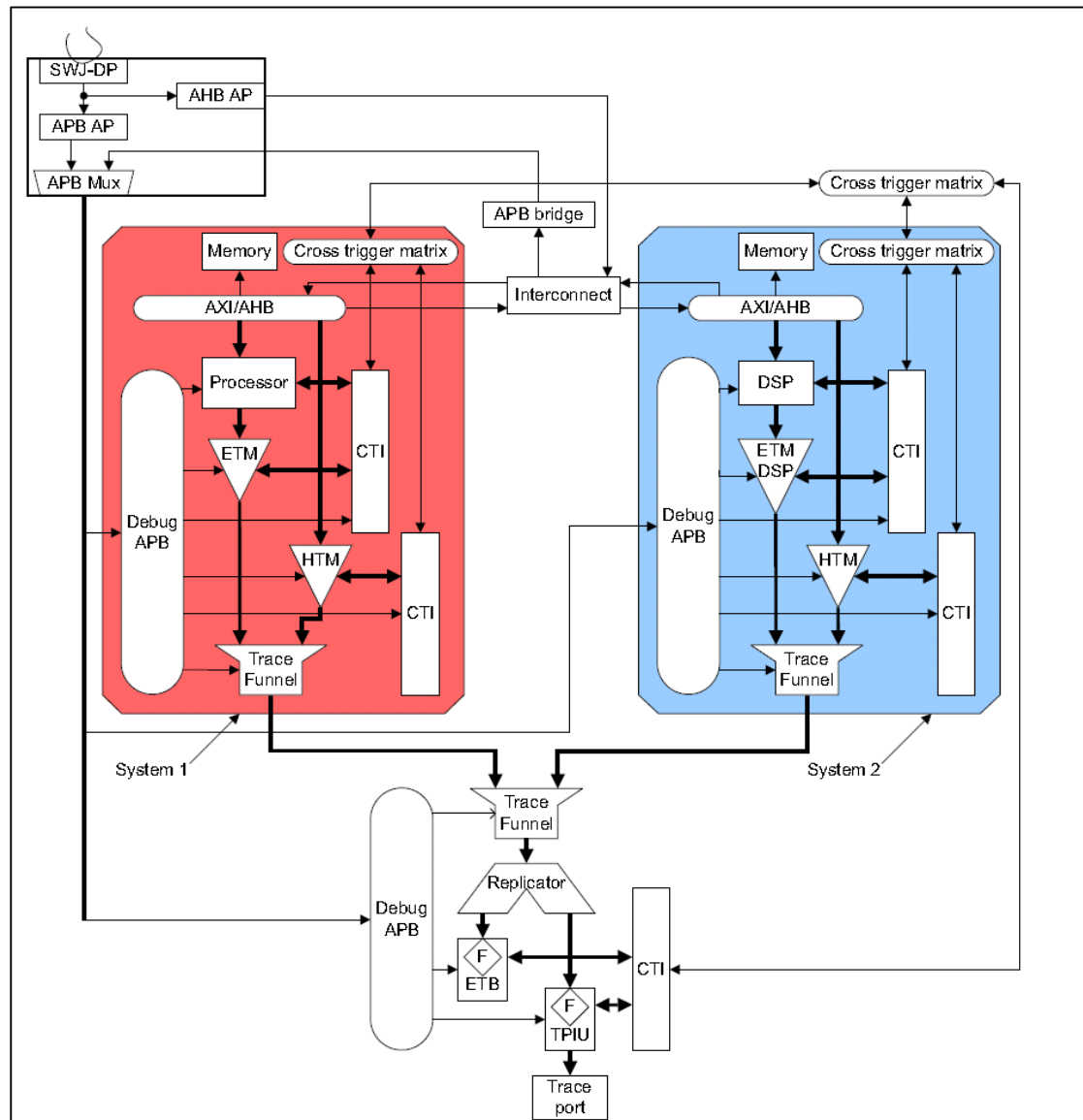
2.3、多 core 的高级 trace 系统

有多个 core, 每个 core 有自己的 trace 组件。因此会包含很多 trace 功能的 coresight 组件。



3. 多 cluster 的完整 coresight 系统

下图是一个多 cluster 的完整的 coresight 系统。



系统中有两个 cluster:

- system1, 以 processor 作为主设备。这个系统中包括了 coresight 的多个组件, debug 组件, trace 组件, trigger 组件。
- system2, 以 DSP 作为主设备。这个系统中包括了 coresight 的多个组件, debug 组件, trace 组件, trigger 组件。

两个子系统通过 interconnect 连接到一起, 实现相互间的通信以及访问外部外设。同时两个子系统的 CTM 和外部的 CTM 连接到一起, 实现两个子系统之间的 event 的相互传输。

整个系统中, 包括一个 DAP, DAP 中有一个 DP, SWJ-DP (jtag 和 sw 协议转化), 和两个 AP, AHB-AP (产生 AHB 总线), APB-AP (产生 APB 总线)。

AHB-AP 产生的 AHB 总线, 直接连接到系统中的 interconnect 上, 就可以访问连接到 interconnect 的外部外设 (如 memory)。

APB-AP 产生的 APB 总线, 连接到两个子系统的 debug apb 上, 实现对子系统的 coresight 组件的寄存器访问。如 ETM, CTI, HTM, funnel 等。同时 APB 总线还连接到了系统的 debug apb 上, 实现对系统的 coresight 组件的寄存器访问。如 funnel, ETB, TPIU, CTI 等。

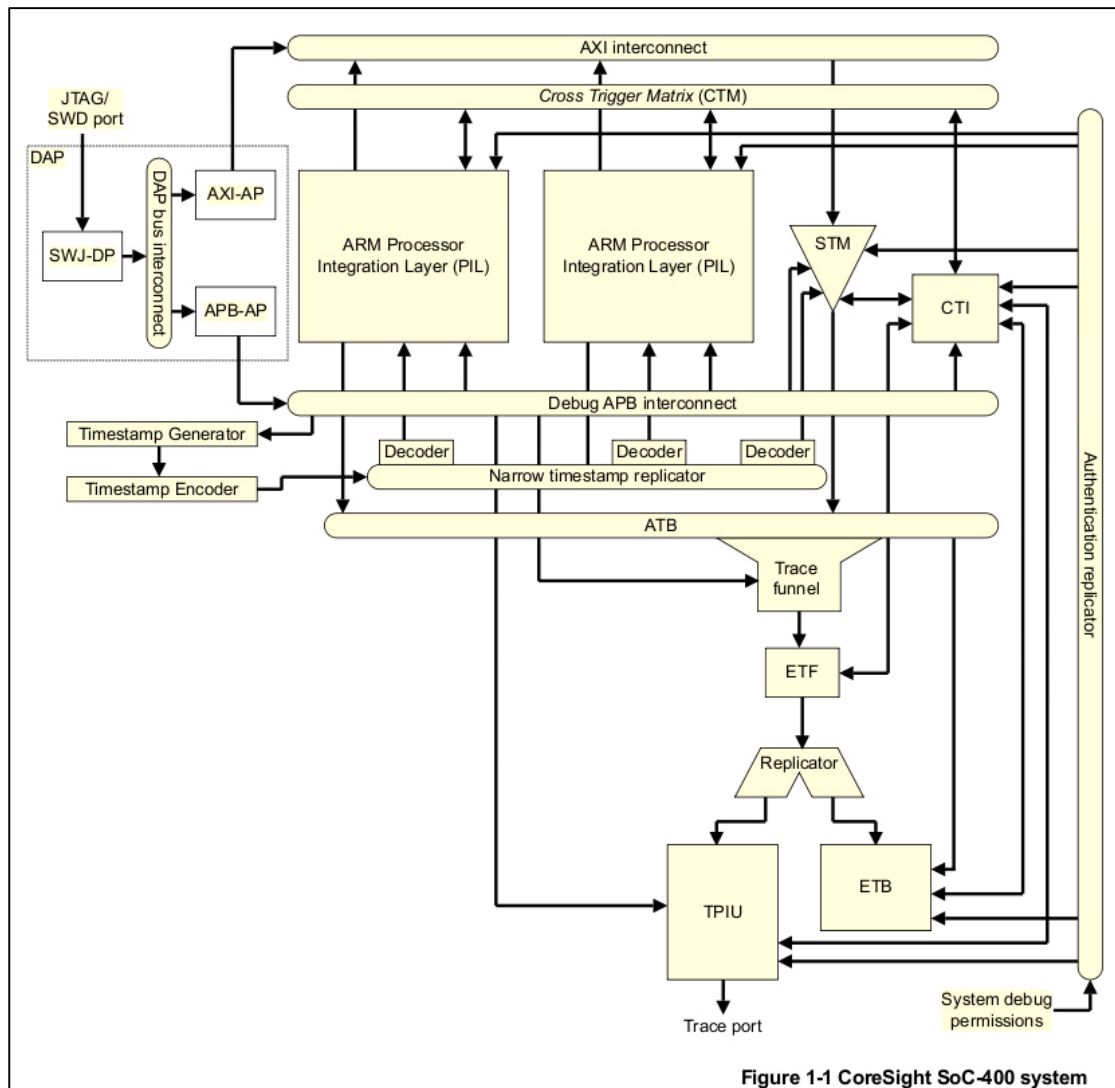
两个子系统的 trace 信息, 通过各自的 funnel 输出到系统的 funnel 上, funnel 对两个子系统的 trace 信息进行合并, 然后输出给 replicator。replicator 将接收的 trace 信息, 广播给

ETB 和 TPIU。TPIU 在通过 trace port 将信息输出到外部。

八、coresight soc-400

因为 coresight 属于 ARM 制定的标准，因此 ARM 针对 coresight，设计出来 soc-400 套件。设计人员可以利用这个套件，快速的生成 coresight 系统，并且生成相应的 case，对 coresight 系统进行验证。

coresight soc-400 系统框图:



这个套件中，可以利用 **AMBA-designer** 来自动生成 **coresight** 的组件，只需要更改一些配置信息即可自动生成。

1. DAP 组件

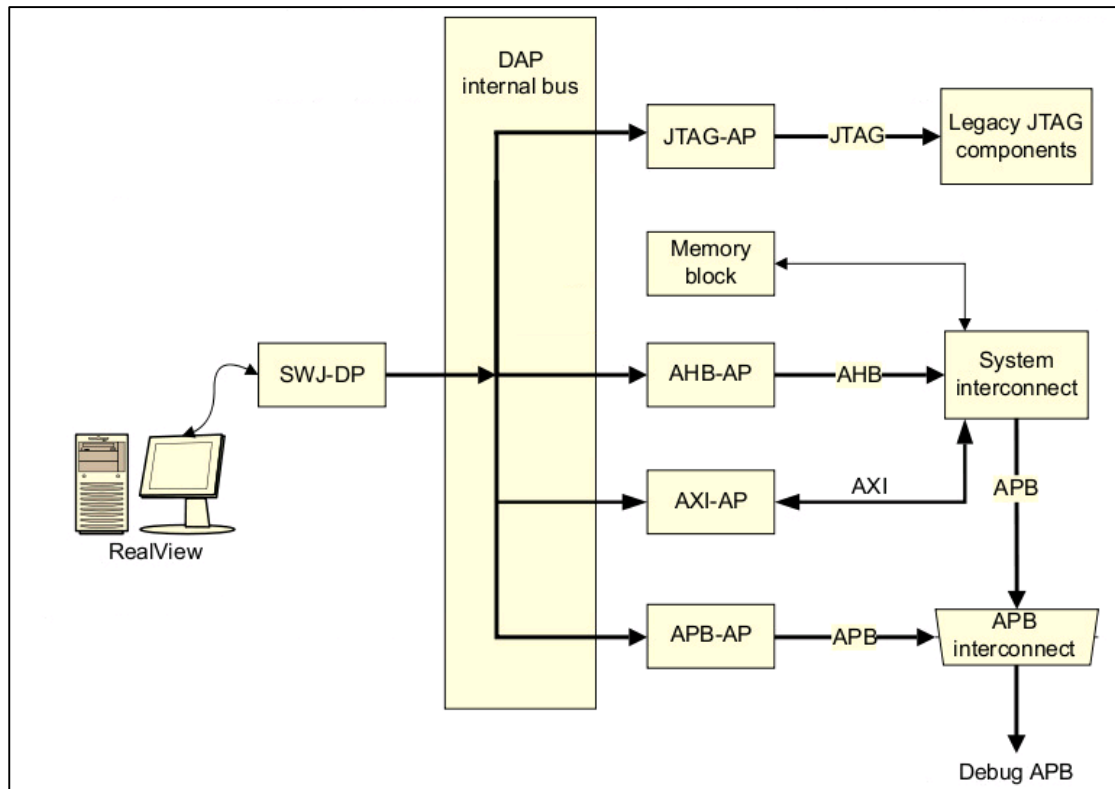
DAP 的一般结构:

SWJ-DP 和外部的 sw 或 jtag 通信，然后和 DAPBUS 通信。实现对各个 AP 的访问。然后各个 AP 再对片内内部资源进行访问。

SWJ-DP 包括两个 DP，一个是 SW-DP，一个是 JTAG-DP。SW-DP 负责和外部的 sw 通信，JTAG-DP 负责与外部的 itag 通信。

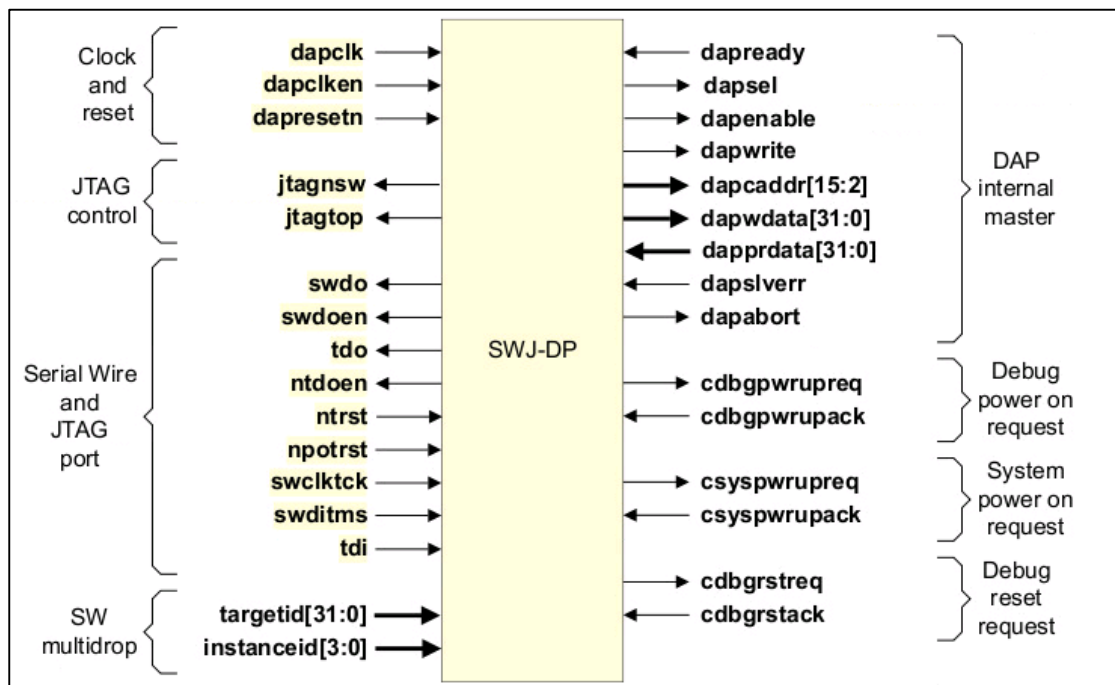
下图是 DAP 的内部结构，包含一个 DP，5 个 AP。

DAP 将外部接口数据 (external interface format), 也就是 SW 协议数据或者 JTAG 协议数据, 转化为内部的接口数据(internal interface), 也就是 AP 访问数据。



1.1、 SWJ-DP

将 jtag 或 sw 总线协议，转化为 dap 总线。



接收 jtag 或 sw 数据，如果是对 DP 访问，直接在内部对 DP 的寄存器进行访问。如果是对 AP 的访问，转化为 dap 总线，对下级所接的 AP 进行访问。

组件，还提供了两个 power 域的上电请求（system power 和 debug power），以及 debug 域的复位请求。

对于两个 power 域的信号，每个信号都是 1bit 信号。

信号	作用
cdbgpwrupreq	DAP 向 power 控制器发送的 debug power 域的上电请求以及时钟使能信号
cdbgpwrupack	power 控制器向 DAP 回应的 debug power 域的上电请求以及时钟使能响应信号
csyspwupreq	DAP 向 power 控制器发送的 system power 域的上电请求以及时钟使能信号
csyspwupack	power 控制器向 DAP 回应的 system power 域的上电请求以及时钟使能响应信号

debugger 通过控制这些信号，来实现对 debug power 域和 system power 域的上电以及时钟使能请求操作。而控制这些信号，是通过写 DA 的 CRTL/STAT 寄存器来实现。

该寄存器的 31-28bit。

CSYSPWRUPACK, bit[31]

System powerup acknowledge. Indicates the status of the CSYSPWRUPACK signal. See [Power control requirements and operation on page 2-64](#).

This bit is RO, meaning it ignores writes.

CSYSPWRUPREQ, bit[30]

System powerup request. This bit controls the CSYSPWRUPREQ signal. See [Power control requirements and operation on page 2-64](#).

After a powerup reset, this bit is set 0.

CDBGPWRUPACK, bit[29]

Debug powerup acknowledge. Indicates the status of the CDBGPWRUPACK signal. See [Power control requirements and operation on page 2-64](#).

This bit is RO, meaning it ignores writes.

CDBGPRWUPREQ, bit[28]

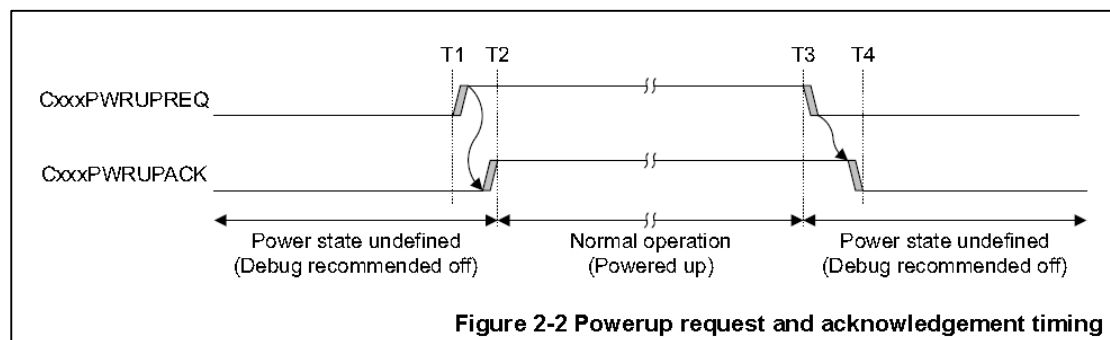
Debug powerup request. This bit controls the CDBGPRWUPREQ signal. See [Power control requirements and operation on page 2-64](#).

After a powerup reset, this bit is set to 0.

在实际中，可能 power 域是断电，或者时钟是关掉的。此时 debugger 要对这个 power 域中的组件进行访问，就需要将该 power 域给开启以及将时钟给开启。此时就需要这些信号。

两个 power 域的请求信号是独立的，因为两个 power 域都是独立的，互不干扰。

当 REQ 信号变高后，表示要 power up，power 控制器应该将 ACK 信号拉高，表示响应该请求。而 REQ 信号变低后，表示要 power down，power 控制器应该将 ACK 信号拉低。



对于 debugger, 可以访问该寄存器, 读取 ACK 的值, 即可知道该 power 域是否有上电。reset 也是一样的。DAP 可以请求复位 debug 域的寄存器。也是通过 CTRL/STAT 寄存器来控制。

CDBGSTACK, bit[27]

Debug reset acknowledge. Indicates the status of the CDBGSTACK signal. See [Debug reset control on page 2-66](#).

This bit is RO, meaning it ignores writes.

CDBGSTREQ, bit[26]

Debug reset request. This bit controls the CDBGSTREQ signal. See [Debug reset control on page 2-66](#).

It is IMPLEMENTATION DEFINED whether this bit is RW or RAZ/WI. See [Emulation of debug reset request on page 2-67](#).

After a powerup reset, this bit is set to 0.

时序如下: 会驱动 PRESETDBGn 信号为低, 从而实现 debug 复位。

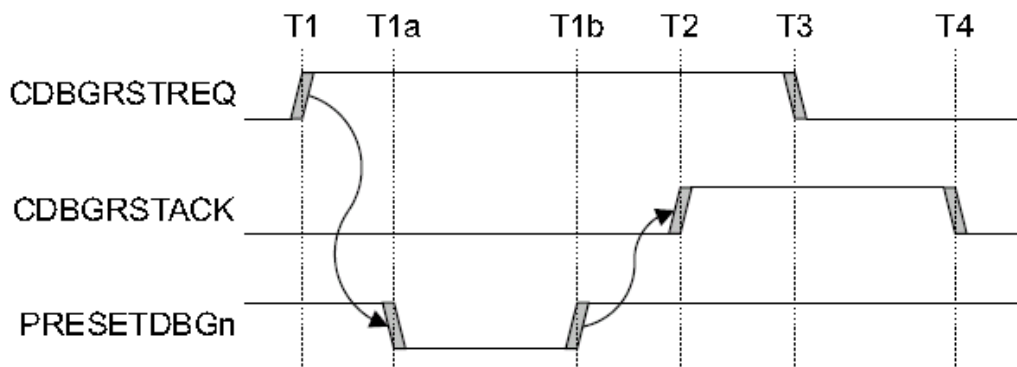
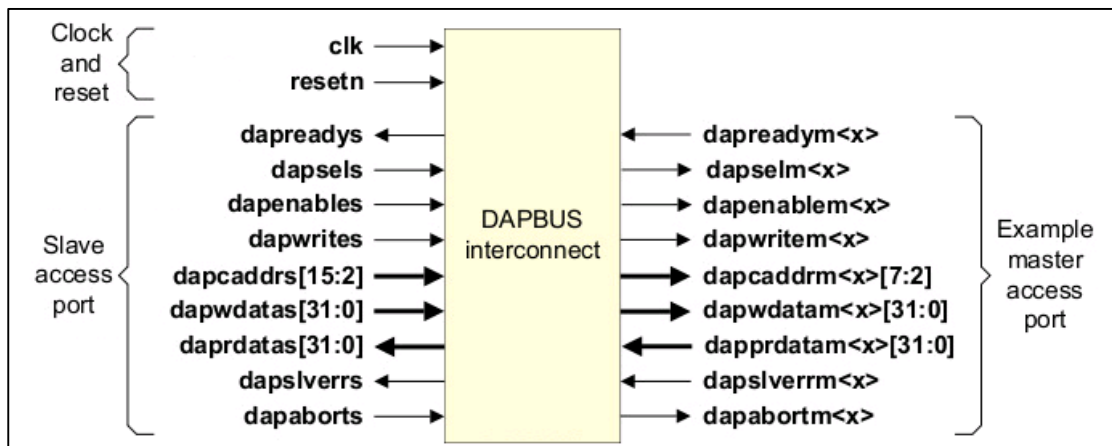


Figure 2-5 Reset request and acknowledge timing

1.2、 DAPBUS 互联

连接 DP 和后续的所有 AP。组件会根据 DP 的 select 寄存器, 决定是对哪一个 AP 进行访问, 从而生成对该 AP 访问的总线。



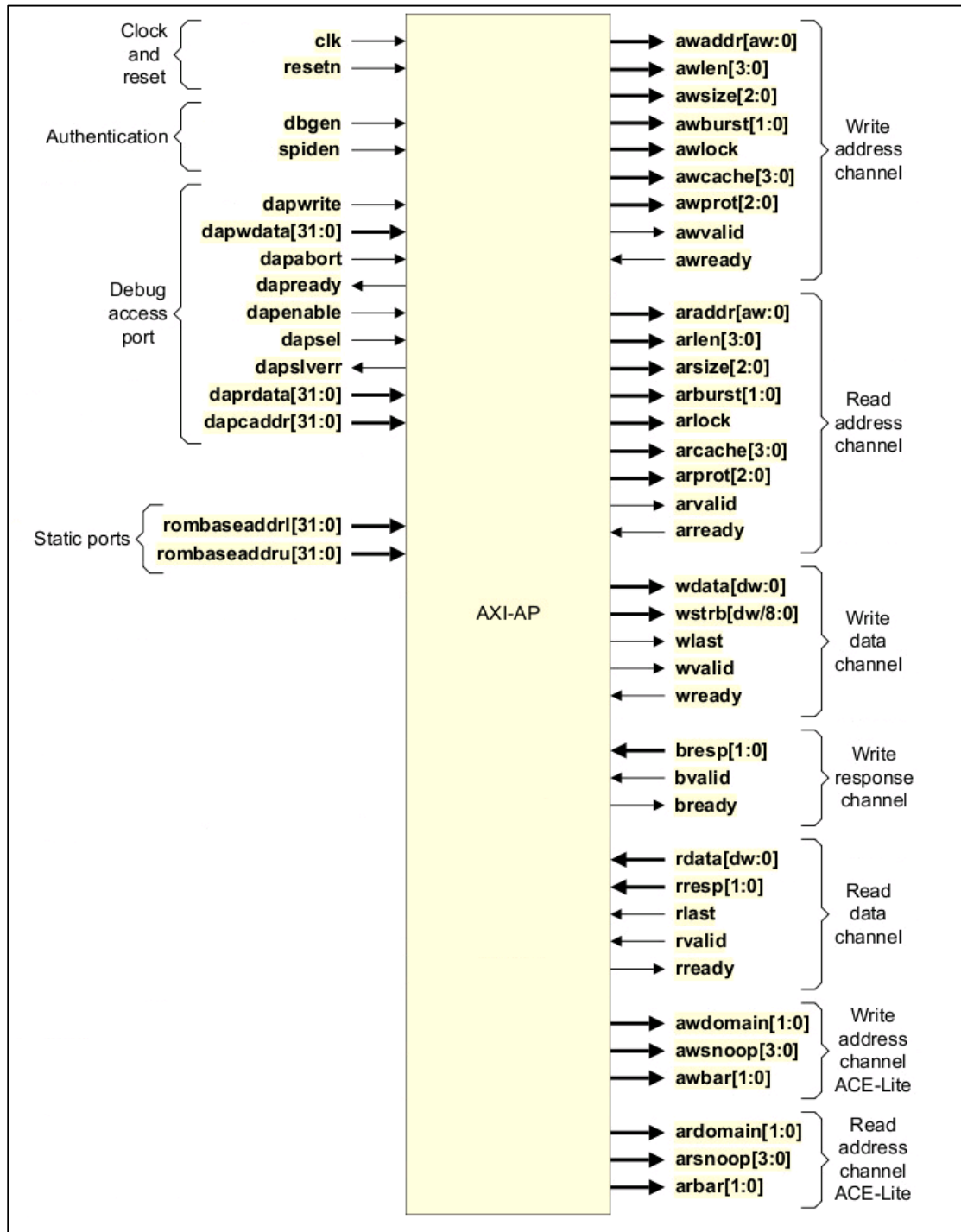
对于地址 dapcaddrs[15:2]:

- dapcaddrs[15:8]: 是 select 寄存器的最高 8 位的值, 也就是 AP 的选择

- dapcaddr[7:2]: 访问 AP 寄存器的地址

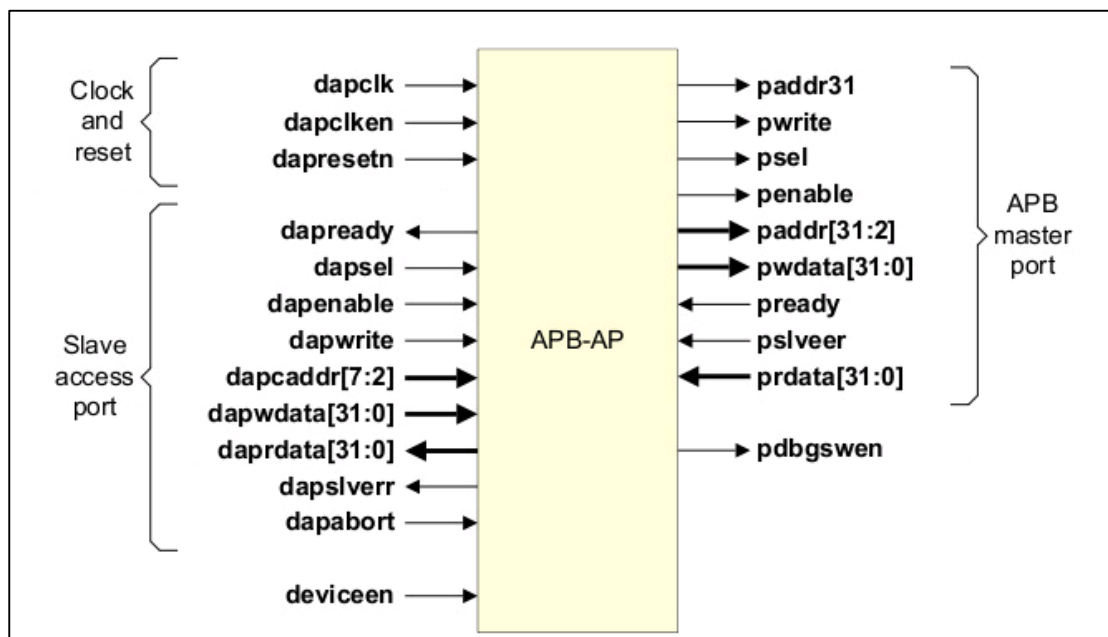
1.3、 AXI-AP

AXI 的 master, 访问该 AP, 可以发起 AXI 访问。输入 DAP 总线, 输出 AXI 总线。



1.4、 APB-AP

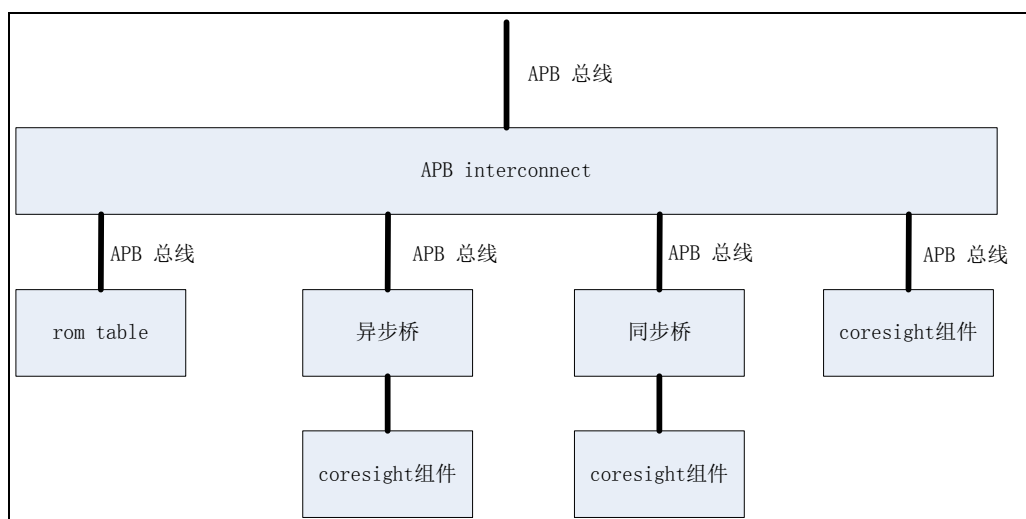
APB 的 master, 访问该 AP, 可以发起 APB 访问。输入 DAP 总线, 输出 APB 总线。



2. APB 互联组件

APB 互联组件，连接了众多的 coresight 组件，外部可以通过 APB 互联，实现对连接到 APB 互联上的 coresight 组件的访问。

以下是 APB 互联组件框图：



APB 互联组件包括以下的一些互联组件。

2.1、 rom table

每个 APB 互联组件，至少连接一个 rom table 组件，并且该组件的地址为 0x0000_0000，这样外部通过 rom table，在能知道连接到该 APB 互联组件上的所有 coresight 组件信息。

2.2、 APB 异步桥

coresight 组件，和 APB 互联的时钟，可能是异步的，因此需要一个异步桥，进行转换。

2.3、 APB 同步桥

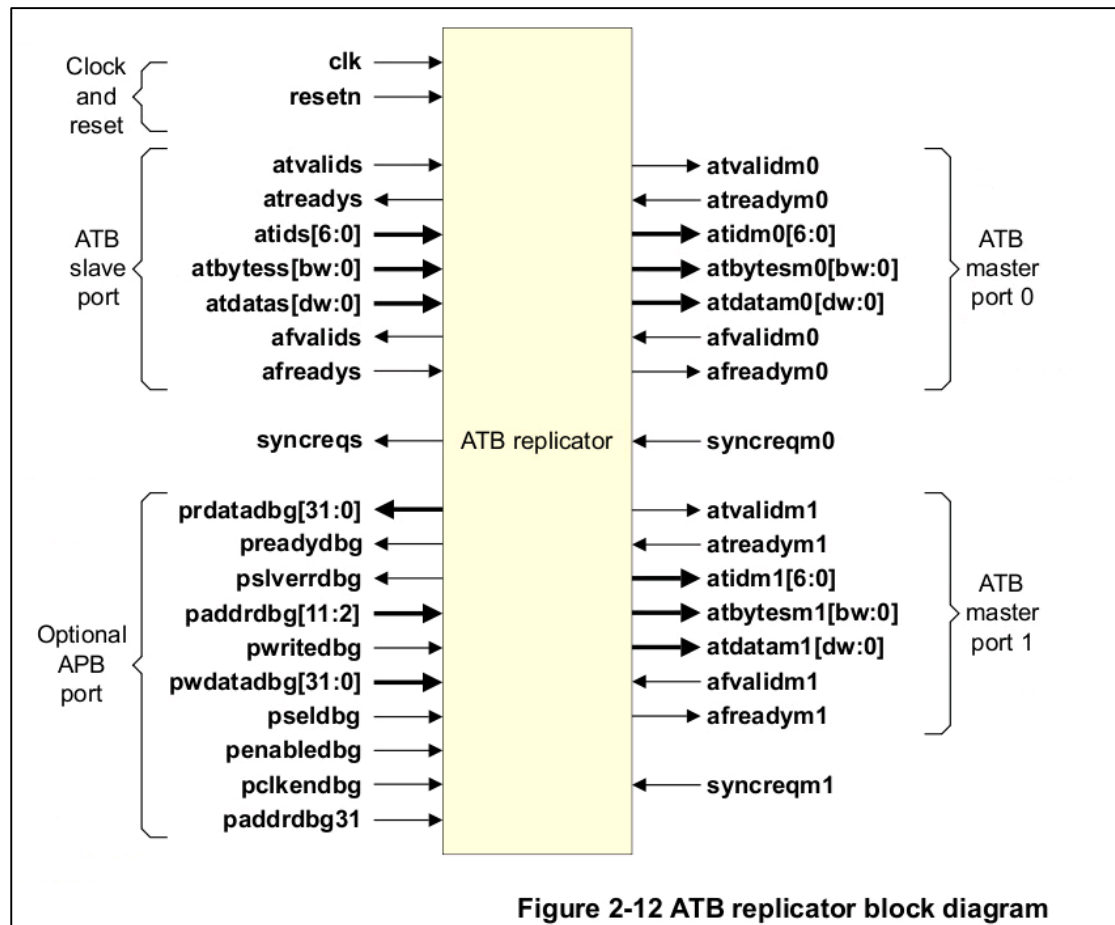
coresight 组件, 和 APB 互联的时钟, 可能是同步, 但是不是同频, 因此需要一个同步桥, 进行转换。

3. ATB 互联组件

ATB 互联组件包括以下的一些互联组件

3.1、 replicator

replicator 用来将上级的 master 发送的 ATB 数据, 传输给下级的两个 ATB slave 组件。
结构如下图所示:

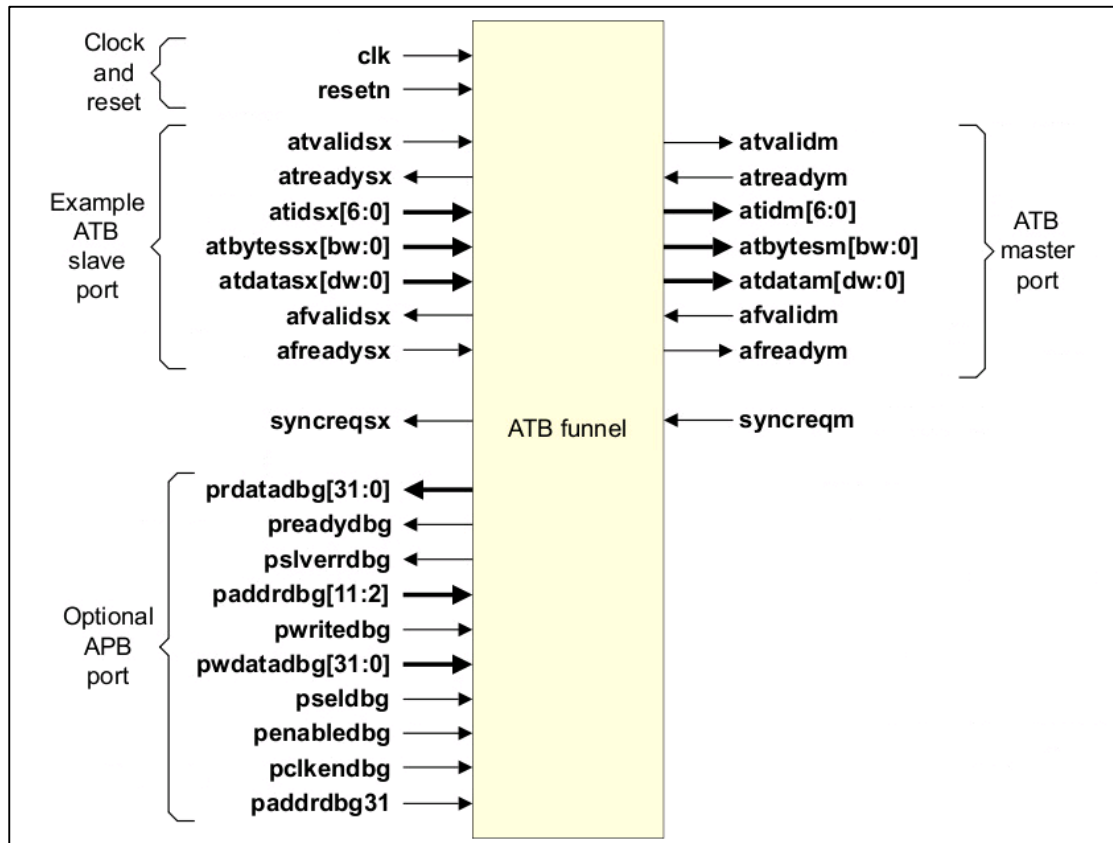


总共有 4 个 port:

- ATB slave port: 接收上一级的 ATB master 的 ATB 数据
- optional APB port: 配置 replicator 的 APB 总线端口, 外部通过该 APB 总线设置 replicator。
- ATB master port0: 输出给 master0 的 ATB 总线
- ATB master port1: 输出给 master1 的 ATB 总线

3.2、 funnel

将多个 ATB 输入, 合并成一个 ATB 输出。
结构如下图所示:



3 个 port:

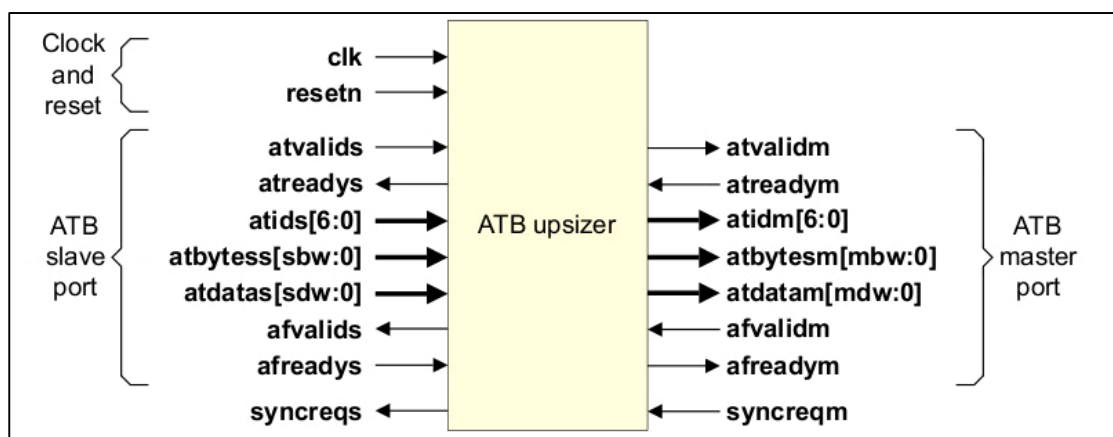
- ATB slave port: 接收上级的 ATB 总线，至少有两组
- optional APB port: 配置 funnel 的 APB 总线端口，外部通过该 APB 总线设置 funnel。
- ATB master port: 输出给 master 的 ATB 总线

3.3、 upsizer

将输入的数据位宽为 SBW+1 的 ATB 总线，转换为数据为数据位宽为 MDW+1 的 ATB 总线。MDW >= SBW。

ATB_DATA_WIDTH_SLAVE: 8,16,32,64

ATB_DATA_WIDTH_MASTER: 8,16,32,64

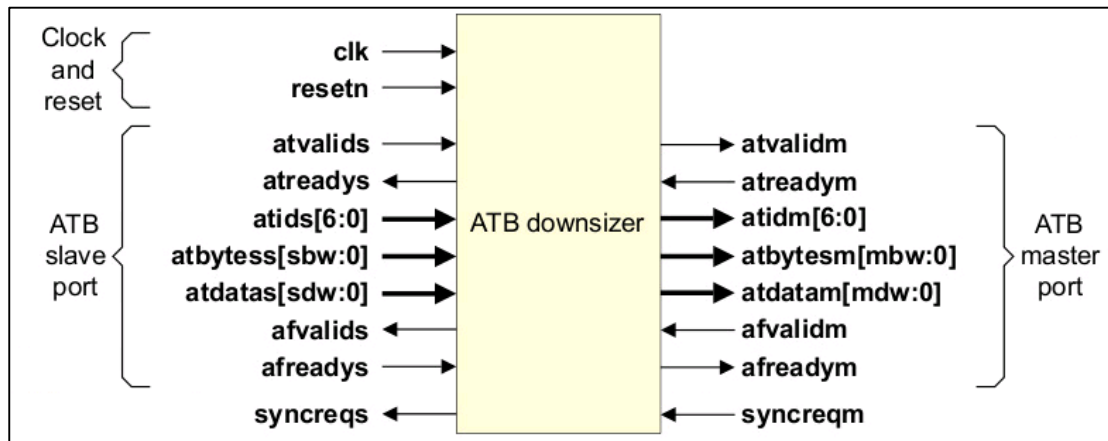


3.4、downsizer

将输入的数据位宽为 SBW+1 的 ATB 总线，转换为数据为数据位宽为 MDW+1 的 ATB 总线。MDW <= SBW。

ATB_DATA_WIDTH_SLAVE: 8,16,32,64

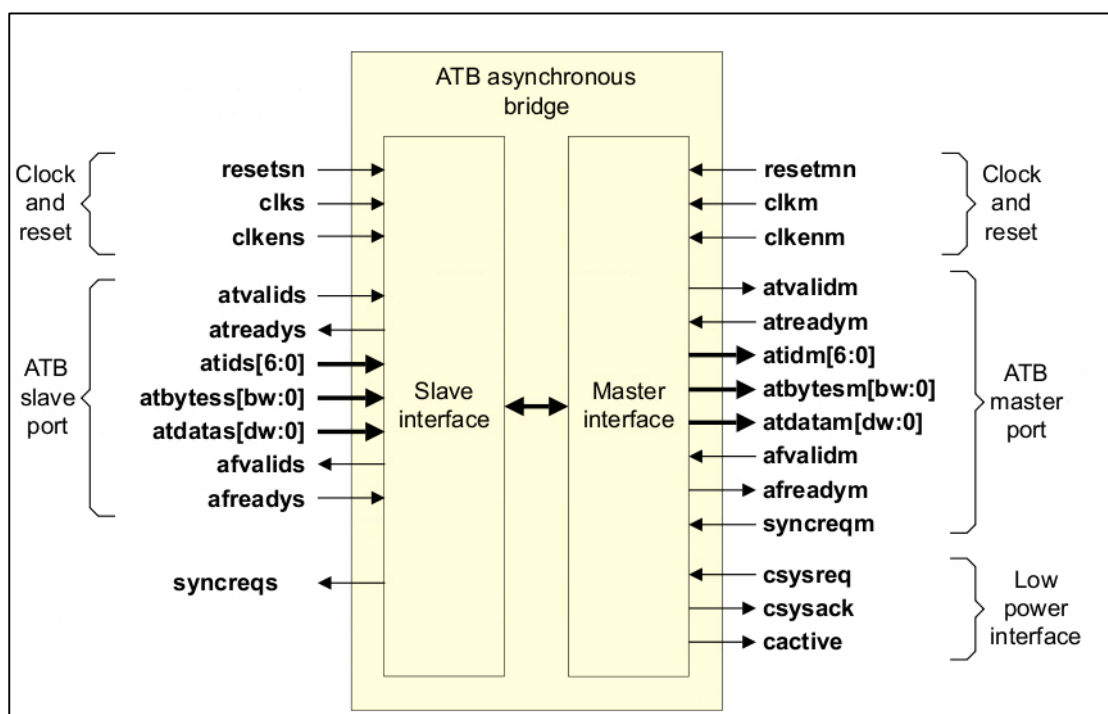
ATB_DATA_WIDTH_MASTER: 8,16,32,64



3.5、asynchronous bridge

跨时钟域（异步时钟）的数据转换桥。

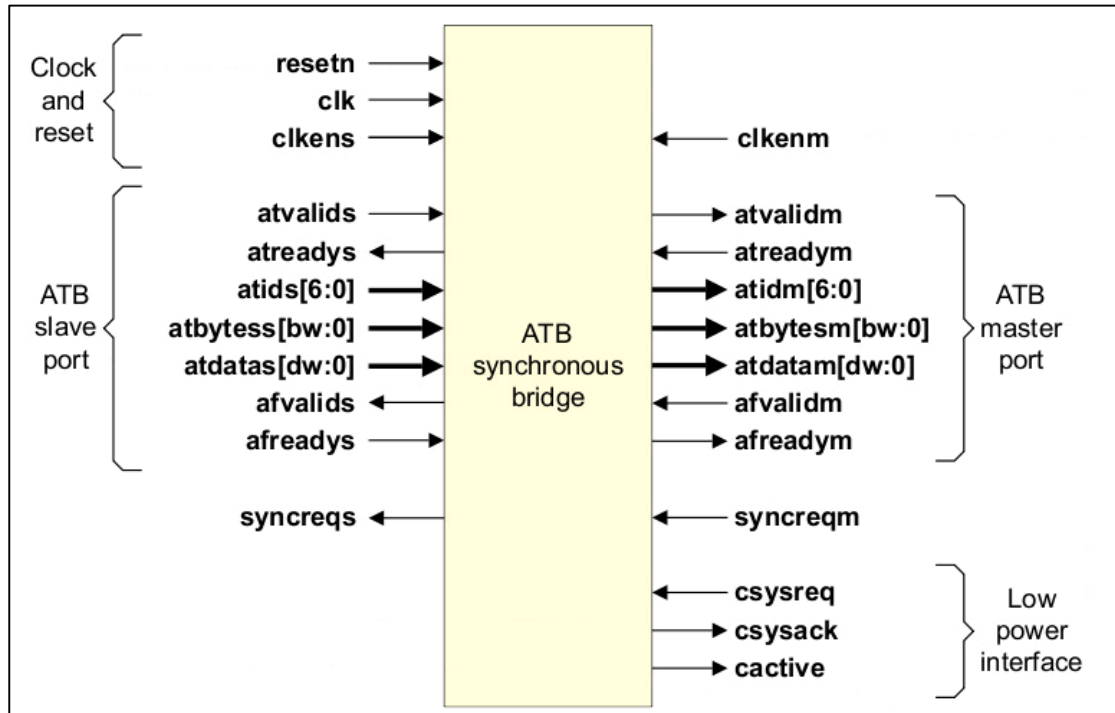
将时钟为 clks 的 ATB 总线，转换为时钟为 clk 的 ATB 总线。



3.6、synchronous bridge

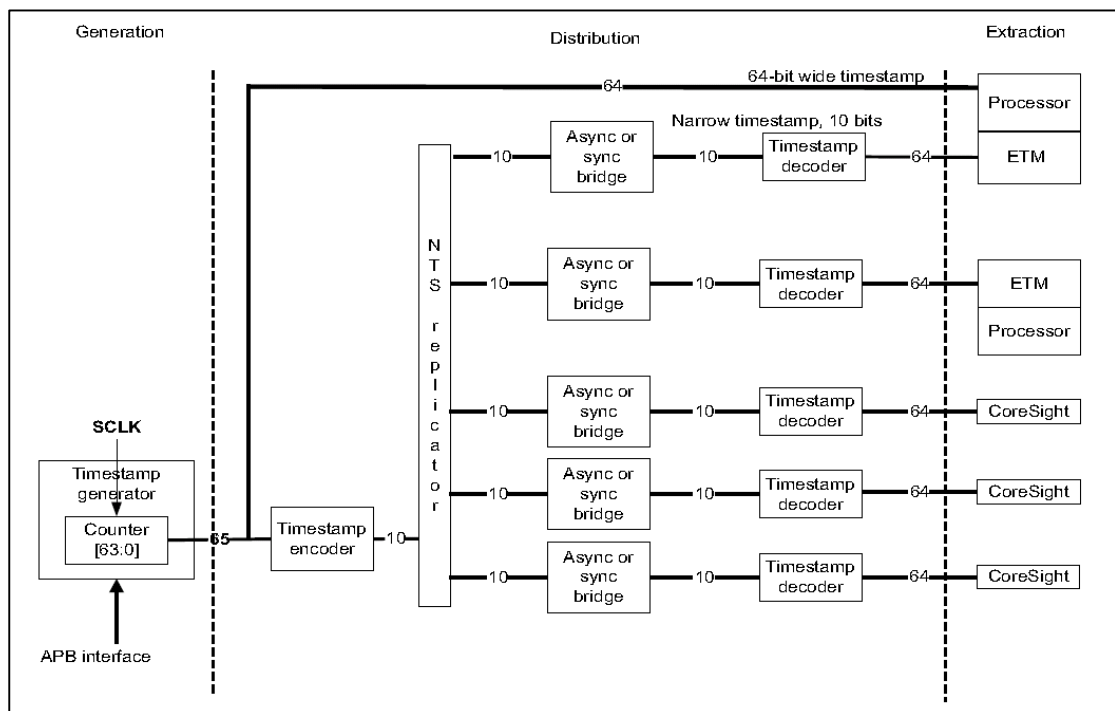
跨时钟域（同步时钟）的数据转换桥。

将时钟为 clks 的 ATB 总线，转换为时钟为 clk 的 ATB 总线。



4. timestamp 组件

timestamp 组件，用来生成时间信息的。



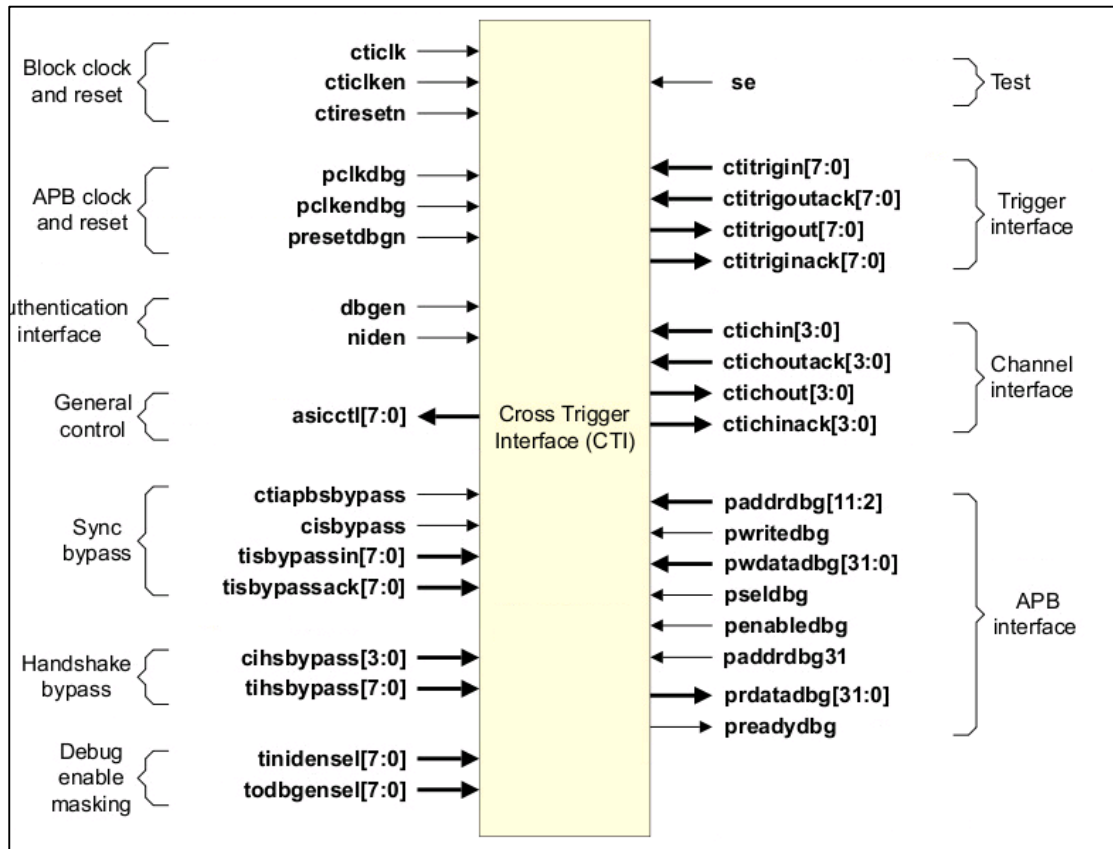
组件，根据 SCLK，产生计数值，然后发送给各个 coresight 组件，这样各个组件就有了时间信息。

5. ECT 组件

ECT 包括 CTI 和 CTM。

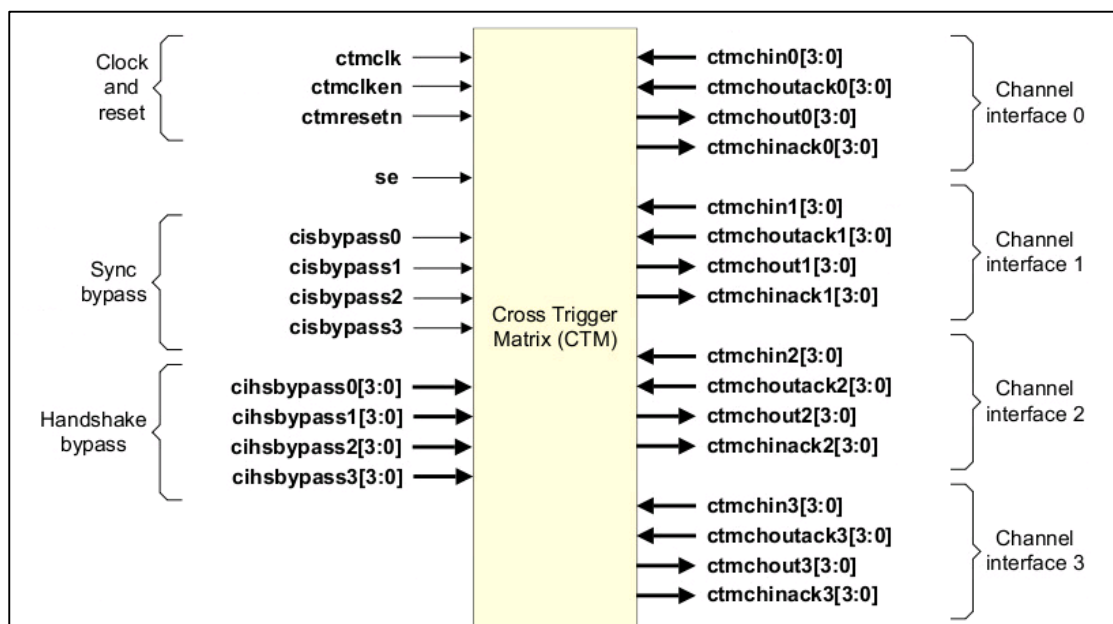
5.1、 CTI

CTI 用来接收和发送 trigger, channel 信号用。



- trigger interface: 连接需要发送 trigger, 接收 trigger 的组件
- channel interface: 连接 CTM, 接收 CTM 发送的 channel, 以及发送 channel 到 CTM 上
- APB interface: 配置 CTI 的 APB 总线, 外部通过该 APB 总线, 设置 CTI

5.2、 CTM

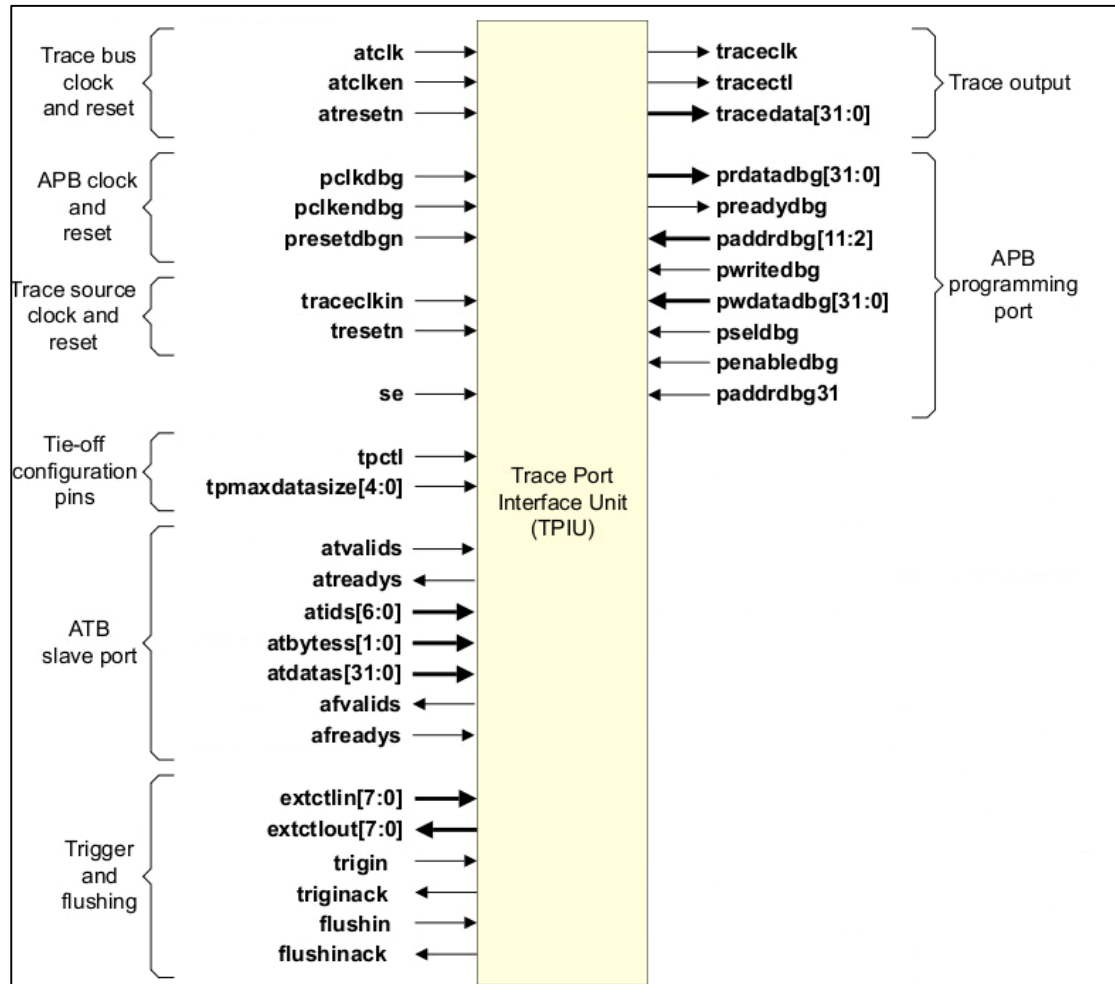


CTM, 连接各个 CTI。

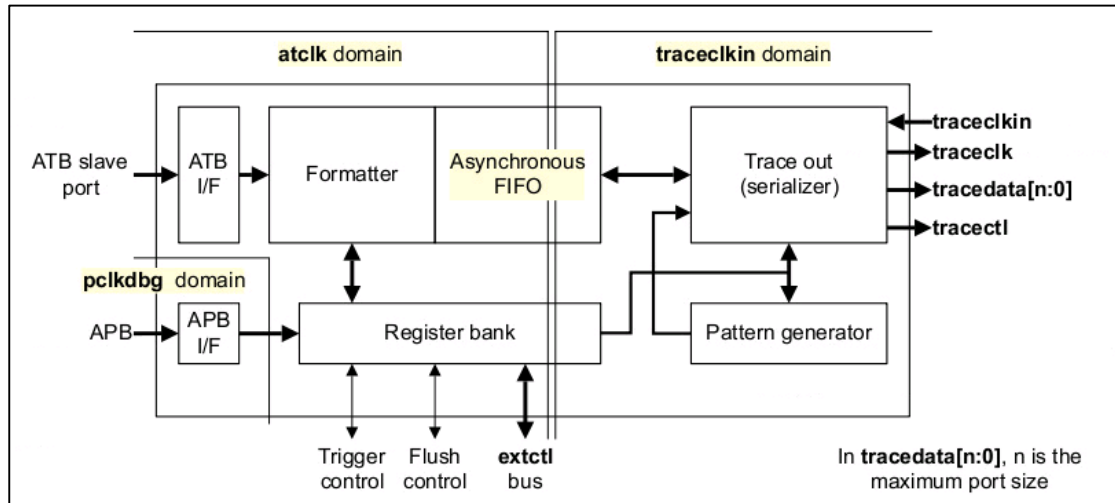
6. trace sink 组件

6.1、 TPIU

trace port interface , 接收 trace 信息, 发送 trace 信息到片外。



- debug apb port: 配置 TPIU 的 APB 接口, 外部通过 APB 总线, 设置 TPIU。
 - ATB slave port: 接收 trace source 或 trace link 的 trace 数据
 - trace port: 芯片的输出管脚, 输出信息给外界
 - trigger port: 连接 CTI。
- 内部结构如下图:



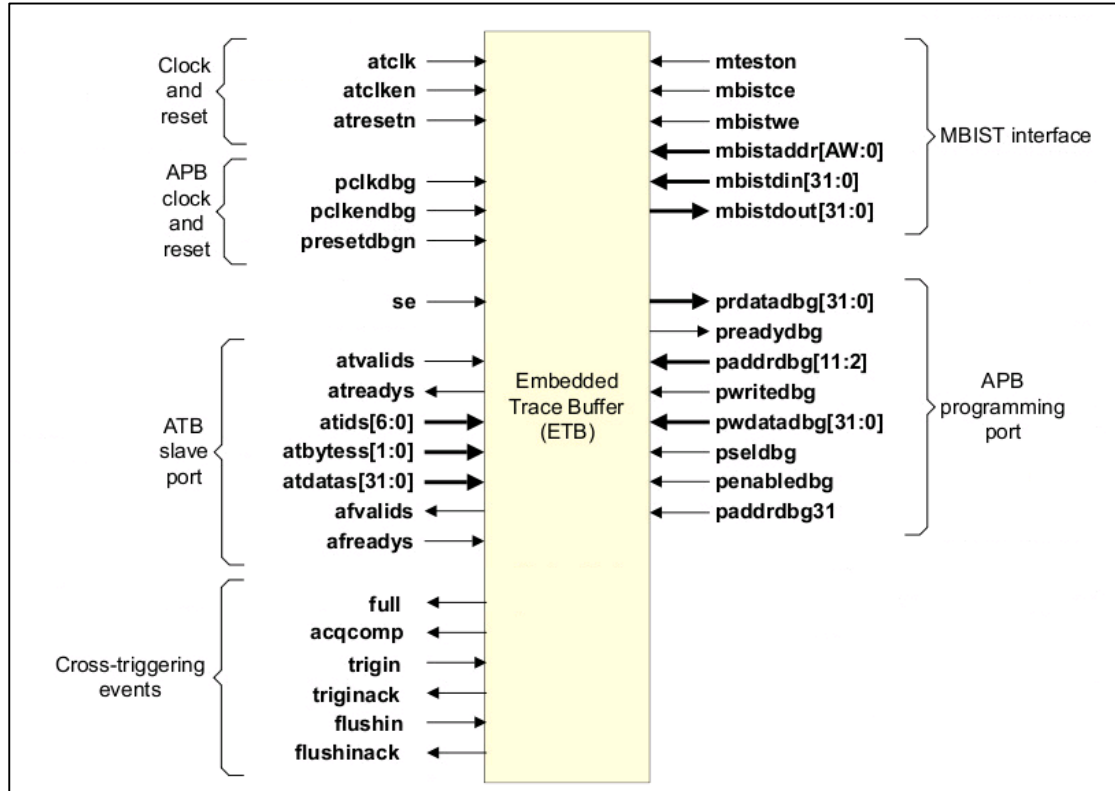
接收 ATB 发送的 trace 信息，通过 formatter，将数据转化，得到真正的数据信息，保存在 FIFO 中。

因为有 2 个时钟域，一个是片内的时钟域，一个是片外的时钟域，因此该 FIFO 是异步 FIFO，写是在 atclk 时钟域写入，读是在 traceclk 时钟域读取。读取之后，通过 trace out，将数据以串行方式，从接口发送出去。

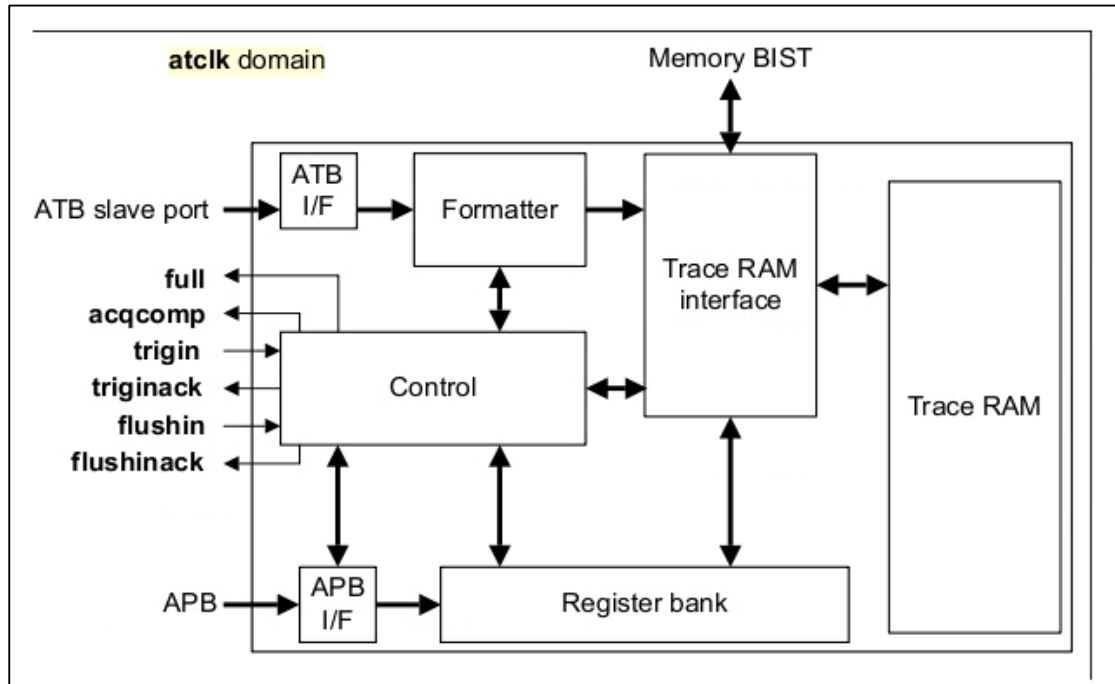
APB 接口，是 TPIU 向外部提供了配置 TPIU 寄存器的 APB 接口。

6.2、ETB

embedded trace buffer。存储 trace 信息的 buffer。



内部结构如下：

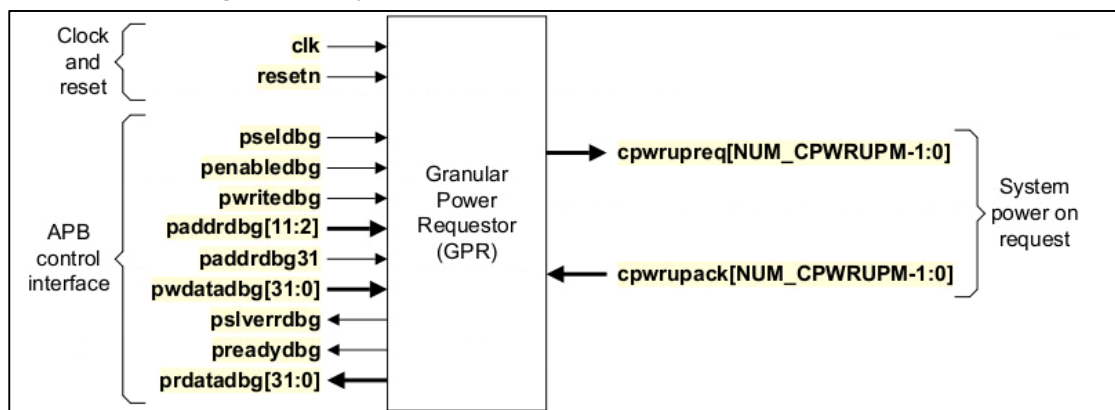


接收 ATB 发送的 trace 信息，通过 formatter，将数据转化，得到真正的数据信息，然后通过 trace RAM interface，将数据，保存到 trace RAM 中。

APB 接口，是 ETB 向外部提供了配置 ETB 寄存器的 APB 接口。

7. power requestor

power requestor 可以让外部通过 APB 总线控制指定 power domain 的上电和断电。从而控制指定的 coresight 组件的 power。



8. 总结

coresight-400, 其实就是 ARM 实现 coresight 系统的套件, 包含了 coresight 的各个组件, 我们利用这个套件, 就不再需要自己单独去设计以及验证这些 coresight 组件, 直接拿过来, 搭建 soc 环境。并且 coresight-400 组件, 还提供了一些测试 case, 可以用来验证搭建的 coresight 系统, 是否正确。

更多的信息, 查看 ARM 提供的 coresight-400 组件文档。