**NoC自动化验证平台设计方案**

**一、修订记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本号** | **修订内容** | **执行时间** | **执行人** |
| V0.5 | 初稿完成 | 2024-4-17 09:59:53 | 葛小华 |
| V0.6 | 根据计划和步森、张虎反馈，做修改或补充 | 2024-6-6 16:03:15 | 葛小华 |

**二、背景**

NoC是目前我们项目用到的总线互连结构，其IP是由Arteris公司提供，我们的设计根据NoC架构表去配置产生具体的代码。当前在验证中，为搭建模块级的验证平台，需要很多手工处理的部分，包括挂接VIP、驱动时钟等等。往往从平台开始搭建，到sanity用例冒烟，需要三天到一周左右的时间（具体要视NoC的复杂度）。

根据上级指示，NoC要验证上减少人力的投入，首先在平台搭建上做好自动化。

本项事务具体由葛小华、步森、张虎、师田田负责，葛小华负责方案制定、实施过程中问题定位和解决和结果验收，步森、张虎、师田田负责该方案涉及到的脚步处理和版本发布。

另外，需要说明的是，该项工作是长期演进性质的。当前阶段化的目标包括如下四个（依时间顺序）：

1. 根据脚本的执行，输入所需要的参数后，一键执行生成整个NoC平台；
2. 平台生成包括NoC的基础用例，基础用例编译后可以直接执行；
3. 图形化GUI（仅作为远期计划，当前需求弱）。

该方案需要依赖如下输入件：

（1）[http://10.180.116.29/svn/XIAN\_Chip/九逸/02-架构设计/Top/SoC\_C/FlexNoC Info\_A0.96.xlsx](http://10.180.116.29/svn/XIAN_Chip/九逸/02-架构设计/Top/SoC_C/FlexNoC%20Info_A0.96.xlsx)

（2）<http://10.180.116.29/svn/XIAN_Chip/九逸/02-架构设计/Top/SoC_C/9E_ASIC_Address_Mapping_V0.991.xlsx>

（3）<http://10.180.116.29/svn/XIAN_Chip/九逸/02-架构设计/Top/SoC_C/Arch_v0.94.vsdx>

上述三个输入件皆来自架构老师，从上而下我们分别简称为“架构表”、“地址映射表”、“架构图”。该方案的制定和落地主要靠架构表和地址映射表，架构图仅作为参考以方便理解。请架构老师不要擅自修改架构表和地址映射表的格式，如果不是强需求，基本格式请沿用九逸项目，以避免给该方案带来冲击。下面分别介绍一下架构表和地址映射表的信息。

**三、输入件：架构表**

架构表，第一个sheet是修订记录，从第二个sheet往后是每个NoC或桥的信息，每个sheet的名字即为这个NoC模块的名字。NoC信息主要分为三个块。

第一块，在这个sheet的顶部位置，是每个NoC上挂接的interface的信息：编号、名字、类型（Initiator还是Target）、协议类型、地址位宽、数据位宽、时钟频率、ID位宽、支持的burst类型、最大的burst长度AxLEN/HBurst、是否支持narrow burst以及narrow burst最小粒度、MaxBurstLength(B)、nPendingTrans、读写使能。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Seq** | **Module** | **Interface** | **NIU type** | **protocol** | **AddrWidth** | **DataWidth** | **Frequence(MHz)** |
|
| 1 | Star SP | unCore\_C\_AHB32\_MST | Initiator NIU | AHB | 32 | 32 | 500 |
| 2 | unCore\_S\_AHB32\_MST | Initiator NIU | AHB | 32 | 32 | 500 |
| 3 | uncoreDMA | unCoreDMAC\_AXI32\_MST | Initiator NIU | AXI | 32 | 32 | 250 |
| 4 | unCoreDMAC\_CTL\_AHB32\_SLV | Target NIU | AHB | 24 | 32 | 125 |

表1 架构表中的interface信息格式举例

表1展示了四个interface的信息格式，篇幅所限，有些维度的信息未列出来。

关于这一块，需要注意如下几点：

* 上述所有信息都有用，需要脚本进行识别，并存储到数据结构中，下文会具体讲到；
* 协议类型是包含NSP的，不仅仅是AMBA类型协议；
* 上述几个维度的参数信息，是按照列从前往后展示的，每一行代表一个interface，每个interface都需要挂VIP。

第二块，位于第一块下面的位置，是每个Initiator interface和Target interface之间的路由关系，即是否某个Initiator interface可以有通路访问某个Target interface。其中所有的Initiator interface是列展出，所有的Target interface是按行展出。如果有通路的话，对应的行和列交叉的单元格位置有 “√”号标记，否则单元格是空的。所以，脚本需要根据这个标记将路由与否存储到数据结构中。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **列1** | **unCoreDMAC\_AXI32\_MST** | **unCore\_C\_AHB32\_MST** | **unCore\_S\_AHB32\_MST** | **JTAG2AHB\_AHB32\_MST** |
| **PER\_CTL\_AHB32\_SLV** | ✔ |  | ✔ | ✔ |
| **SYSCTL\_AHB32\_SLV** |  |  | ✔ | ✔ |
| **LMU0\_AHB32\_SLV** | ✔ | ✔ | ✔ | ✔ |

表2 架构表中的interface间路由结果格式举例

需要注意的时候，这里面涉及到NoC\_service，NoC\_service并不是NoC的interface而是内部的寄存器。目前可以暂不考虑让脚本处理NoC\_service，因为对于NoC寄存器的访问验证要手工方式特殊处理——考虑到寄存器有mask位，也有不同的R/W属性，验证起来相对复杂。但是，这个问题最终目标相信是可以解决的。

第三块，在第二块下面的位置，有一些额外的信息需要处理，这些信息包括但不限于——**这部分若交给自动化平台方案处理，难度略大（当然不是说实现不了，主要是表格的格式显得乱），建议后续版本升级的话考虑合入该需求，当前的版本可以先忽略**：

* 是否有ATB对外接口——验证：决定了是否要增加ATB接口用例。
* error probe的位置——验证：有较多用例和error probe相关，一方面有error probe的地方就有中断，另一方面error probe支持四种类型的error，原则上每种类型error都需要构造激励覆盖。
* packet probe的位置——验证：涉及到trace用例和带宽统计用例。
* transaction probe的位置——验证：涉及到统计链路利用率和delay的用例。

第三块的表格形式这样的：

|  |  |
| --- | --- |
| 是否需要ATB接口 | No |
| service targets数量 | 1 |
| observer被访问关系 | service target 的访问关系 |
| observer数量 | 1 |
| error probe位置 | pcieEP\_DATA\_AXI512\_W\_MST |
| pcieEP\_DATA\_AXI512\_R\_MST |
| PCIeEPNoC\_service |
|  |  |
| packet probe位置 | pcieEP\_DATA\_AXI512\_W\_SLV |
| pcieEP\_DATA\_AXI512\_R\_SLV |
|  |  |
| transaction probe位置 | No |

目前，这块的信息，架构给的格式相对没有那么整齐，如果对脚本处理带来困扰，需要给架构提格式上的需求。

**四、输入件：地址映射表**

地址映射表，主要反映了SoC给每个模块分配的地址区间。它的内容很多，有很多的列，对于本方案来说，主要关注以下的几列：

* Subsystem：子系统或block的名字。
* SubModule：模块的名字，从属于Subsystem，一个Subsystem可有多个SubModule。
* Start Address (Hex, 44b)：模块的起始地址，和SubModule对应。
* End Address (Hex, 44b)：模块的结束地址，同样和SubModule对应。
* FlexNoC port：该模块属于哪个slave interface，通过这个slave interface才可以访问该模块的内部空间，一般这个slave interface的命名和架构表里的命名一致（特殊情况是：如果架构表里区分了读写方向为两个不同的interface，例如：EP\_NoC的pcieEP\_DATA\_AXI512\_W\_SLV和pcieEP\_DATA\_AXI512\_R\_SLV，实际上它们包含的模块以及模块地址是相同的，但是同一个单元格里没有写入两个interface的名字，而是用类似pcieEP\_DATA\_AXI512\_SLV这样的名字统一表达的）。
* FlexNoC：该模块挂在哪个NoC底下。

上述信息，需要被脚本识别。

这里面有一个很重要的点，不是所有的interface都能在这地址映射表的“FlexNoC port”这一列找到——这里指的是NSP类型的，NSP类型的要特殊处理。因为NSP类型interface下游级联的只会是另外一个NoC的interface，它并不直接级联某个模块，反过来说，模块不属于任何一个NSP类型的interface，它只属于AMBA协议的。这个点引申出来另外一层含义——如果在架构表里表面某个Initiator interface可以访问某个NSP类型的Target interface，需要脚本将这个NSP类型的Target interface转为下游级联NoC的Initiator interface，然后看这个转化后的Initiator interface能通过被级联的NoC结果可访问哪些AMBA类型的Target interface，这些AMBA类型的Target interface里的所有模块即作为上面波折后的Initiator interface可访问对象。这里面的逻辑转化关系，需要脚本进行处理。



上图只有Slv 0、Slv 1和Slv 2才会在地址映射表里体现。

A NoC出口向NSP下游的slave interface叫A2B\_NSP\_SLV，那么B NoC入口自NSP上游的master interface就叫A2B\_NSP\_MST——脚本需要进行同一个架构表跨sheet操作，即由A NoC的master interface（出）名字得到B NoC的slave interface名字（入），而A NoC是一个sheet，B NoC是另一个sheet。

**五、脚本需要处理生成的文件**

**5.1 atb/assertion**

这个目录创建一个空的即可，默认NoC验证用户不需要断言。

**5.2 atb/cfg**

这个目录要配置VIP，包括AMBA（包括AXI、AHB、APB、ATB） VIP和NSP VIP。

* amba\_system\_configuration.sv

AMBA VIP协议的，建议单独放在一个文件中，不要和其它的config混在一起，文件起名：amba\_system\_configuration.sv。主要注意的是，xxx\_noc\_ahb\_system\_configuration这个类需设置this.ahb\_lite\_multilayer = 1，因为NoC可能需要多个AHB lite master VIP。其它的配置ATB工具应该已经主动可以配置好。

VIP的master和slave个数的配置，需要引用下面xxx\_noc\_definitions\_pkg.sv的内容——里面定义的全局变量，xxx\_noc\_definitions\_pkg.sv这部分会详细讲解。

* xxx\_noc\_nsp\_config.sv

NSP VIP也建议单独放在一个文件中，不要和其它的config混在一起，文件起名：xxx\_noc\_nsp\_config.sv，这里面定义了xxx\_noc\_nsp\_config类，需要脚本声明并初始化两个static变量数组 anvu\_nsp\_info initiator\_info[NspInitNb]和anvu\_nsp\_info target\_info[NspTargNb]——NspInitNb和NspTargNb这俩来自xxx\_noc\_definitions\_pkg.sv的全局变量——数组里面的各元素初值需要脚本进行处理：在自动化平台的用户界面让用户填入，最好提醒用户填入的格式，如果格式填入有问题告警并让用户重新填入——格式可以参考已有的NoC验证环境，这个格式是固定的。注意：提醒用户数组里元素的顺序是有要求的，顺序要参考下面common目录的xxx\_noc\_definitions\_pkg.sv内容。

**5.2 atb/common**

* xxx\_noc\_defines.svh

创建一个空壳文件，壳子的内容写法：

`ifndef XXX\_NOC\_DEFINES\_SVH

`define XXX\_NOC\_DEFINES\_SVH

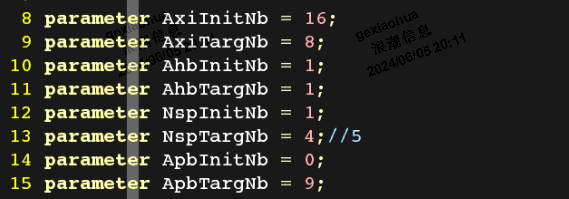
//put your code here

`endif

* xxx\_noc\_definitions\_pkg.sv

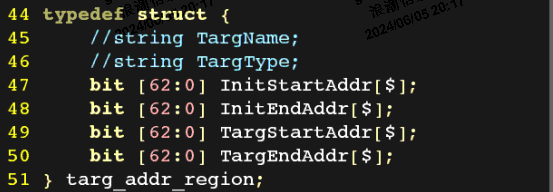
这个文件最复杂，因为要放很重要的全局变量，这些变量值在用例的sequence以及其它地方中调用。另外，这里面涉及到的数组的初始化，每一行代表其中一个元素，最后都要加注释（为了方便用户快速获取相关信息），表示出这行对应哪个interface名字。

首先，初始化各个类型master VIP和slave VIP的数量：



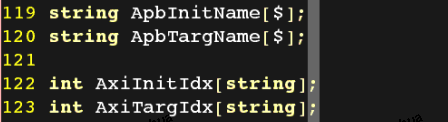
AxiInitNb表示Axi master数量，AxiTargNb表示Axi slave数量，其它的协议类型同理。分号前的数值需要从NoC架构表读入并计算：protocol这列确定协议类型，NIU type这列确定是master还是slave，然后组合起来数一下有多少行，就知道要填入的数量值是多少了。

然后声明几个结构体变量类型，举例：

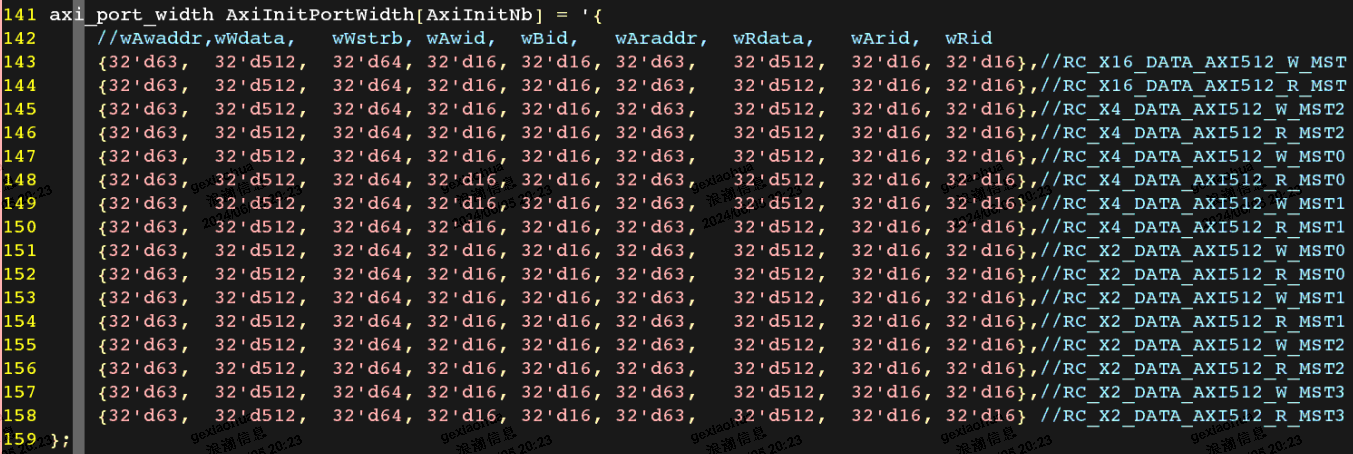


我们系统中最大的地址位宽就是63，所以这里是用最大的。其它的结构体变量类型参考当前的NoC验证环境即可，全部复制过来无需修改，本文不再赘述。

然后声明几个变量，拿其中一部分示意一下，其余参考当前的NoC验证环境即可，全部复制过来无需修改：



然后声明这个下面数组，顺便做初始化。



有多少的Axi master就有多少行，从上而下顺序是有要求的（后面会讲顺序这回事，后面所有的数组都要符合这个要求），另外每一列尽量对齐，这样观感好一些。

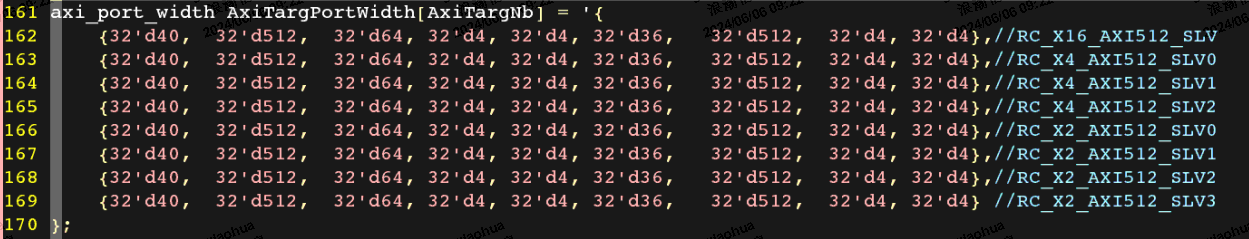
wAwaddr、wAraddr：从架构表AddrWidth列读进来；

wWdata、wRdata：从架构表DataWidth列读进来；

wWstb：需要计算，公式为wWdata÷8；

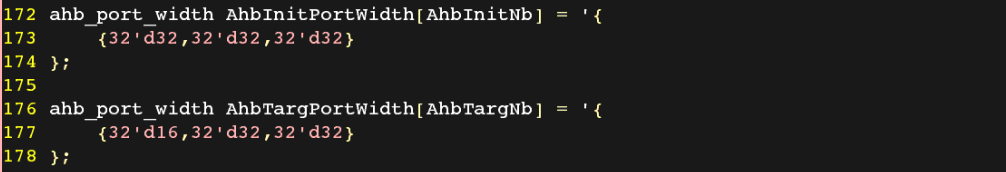
wAwid、wBid、wArid、wRid：从架构表ID Width/SeqID列读进来，取“/”前面那个值，如果单元格是空的，直接赋0值即可——如果脚本处理不了这种情况，请提前知会，我们让架构老师调整输入件的格式。

然后声明这个下面数组，顺便做初始化。



该数组处理准则同上面的AxiInitPortWidth数组，不再赘述。

然后声明这个下面这俩数组，顺便做初始化。

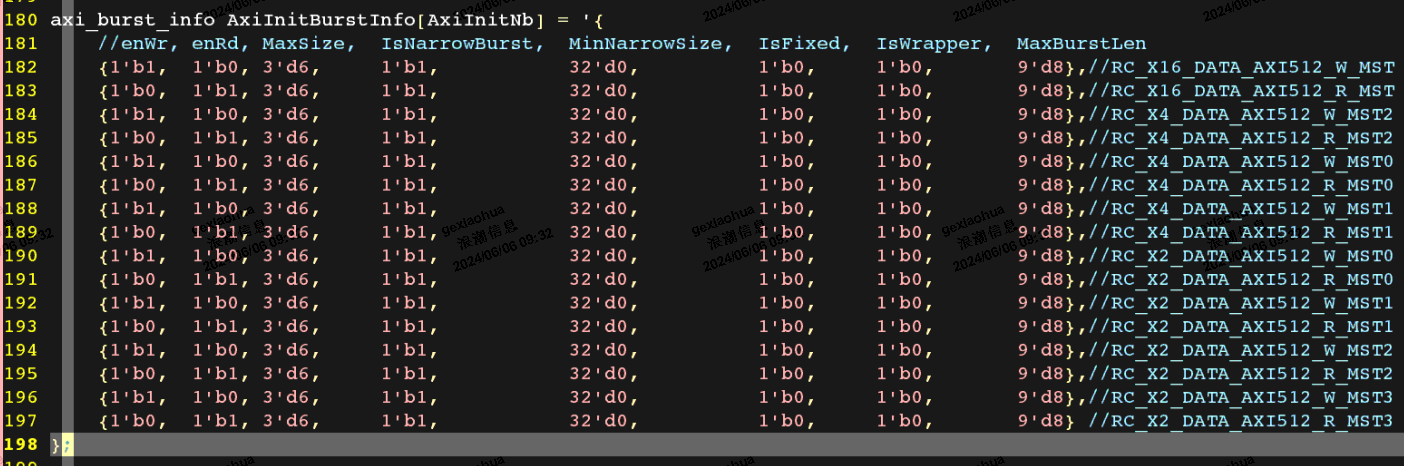


这俩数组格式是一样的。

第一个元素：从架构表AddrWidth列读进来；

第二个、第三个元素：从架构表DataWidth列读进来。

然后声明下面这个数组，顺便做初始化。



enWr：从架构表Write列读进来；

enRd：从架构表Read列读进来；

MaxSize：需要计算，计算方式为：从架构表DataWidth列读进来，然后以2为底对其取对数，即log2(DataWidth)；

IsNarrowBurst：从架构表Narrow Transfer: support列读进来，如果结果是√则赋值为1，否则赋值为0；

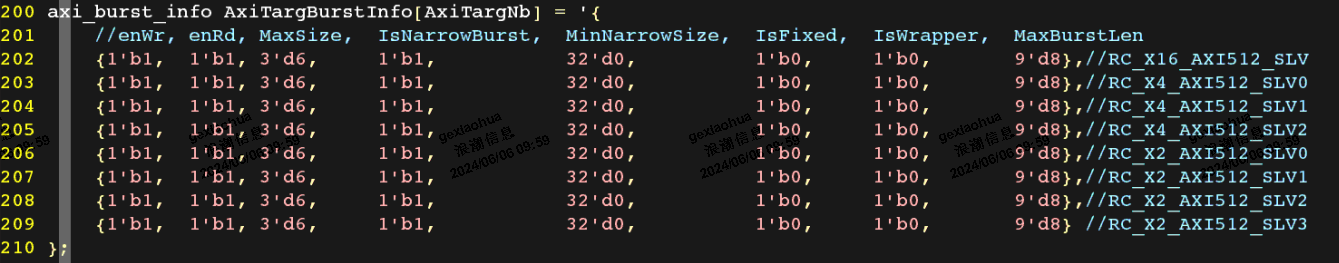
MinNarrowSize：如果IsNarrowBurst=0则令MinNarrowSize= MaxSize，否则需要计算：从架构表Narrow Transfer: min Size列读进来然后以2为底对其取对数；

IsFixed：从架构表Fixed列读进来；

IsWrapper：从架构表Wrap列读进来；

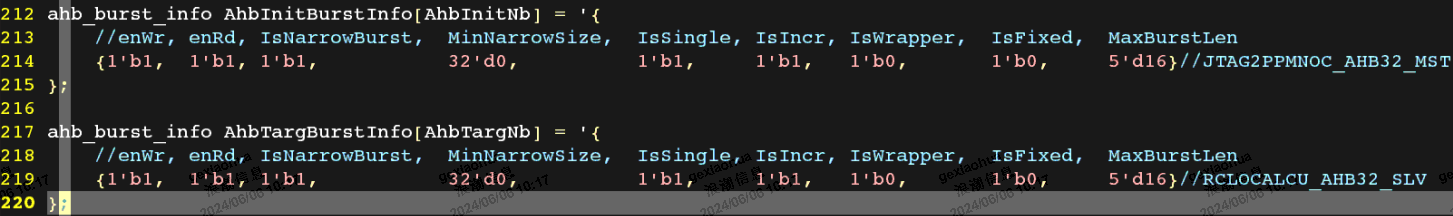
MaxBurstLen：从架构表AxLEN/Hburst列读进来；

然后声明下面这个数组，顺便做初始化。



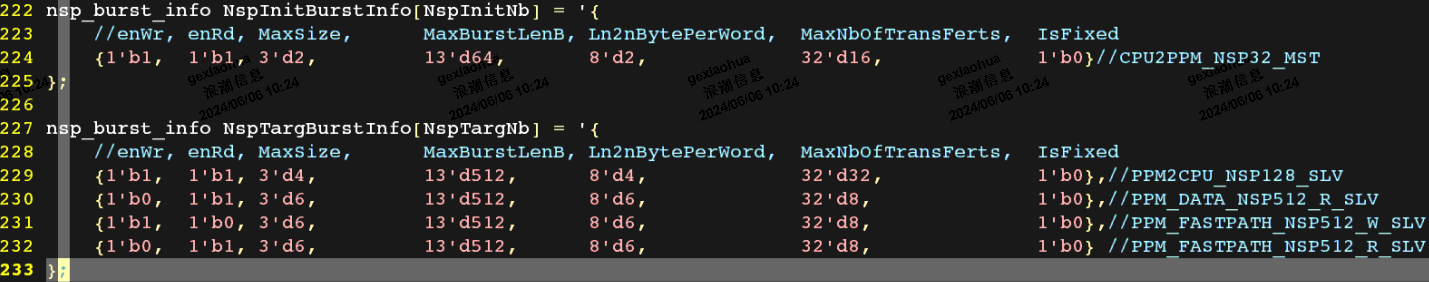
该数组处理准则同上面的AxiInitBurstInfo数组，不再赘述。

然后声明下面这俩数组，顺便做初始化。



这俩数组格式是一样的。初始化值的来源同AxiInitBurstInfo和AxiTargBurstInfo，处理方式此处不再赘述，参考AxiInitBurstInfo即可。

然后声明下面这俩数组，顺便做初始化。



这俩数组格式是一样的，另外enWr、enRd、MaxSize和IsFixed同前面的处理方式，不再赘述。

MaxBurstLenB：从架构表MaxBurstLength(B)列读进来；

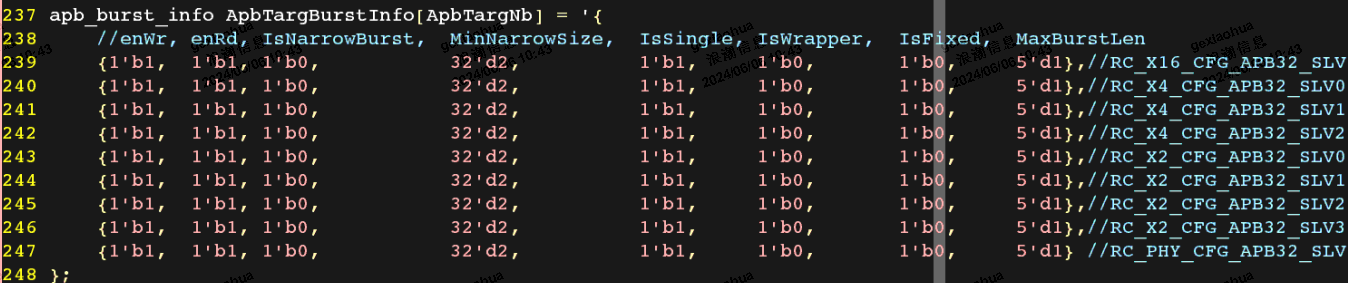
Ln2nBytePerWord：令其值等于MaxSize即可；

MaxNbOfTransFerts：需要计算，计算公式：MaxBurstLenB÷2Ln2nBytePerWord。

然后声明这个数组，照抄即可——目前NoC结构不存在APB的master。



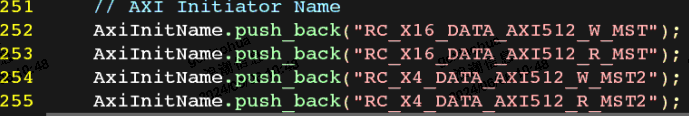
然后声明下面这俩数组，顺便做初始化。



初始化值的来源同AxiInitBurstInfo和AxiTargBurstInfo，处理方式此处不再赘述，参考AxiInitBurstInfo即可。

下面要定义一个原型为function void xxx\_noc\_definitions\_init()的函数，下面将该函数的流程步骤化。

Step1：将所有的Axi master interface名字推入队列（注意顺序要和前面的AxiInit开头的数组元素顺序一致），示意如下：



Step2：将所有的Axi slave interface名字推入队列（注意顺序），处理方式同上；

Step3：将所有的Ahb master interface名字推入队列（注意顺序），处理方式同上；

Step4：将所有的Ahb slave interface名字推入队列（注意顺序），处理方式同上；

Step5：将所有的Nsp master interface名字推入队列（注意顺序），处理方式同上；

Step6：将所有的Nsp slave interface名字推入队列（注意顺序），处理方式同上；

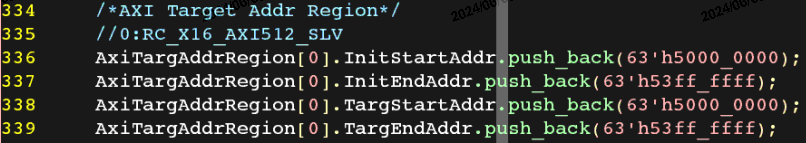
Step7：将所有的Apb master interface名字推入队列（注意顺序），处理方式同上；

Step8：将所有的Apb slave interface名字推入队列（注意顺序），处理方式同上；

Step9：给所有的master和slave索引初始化，以Axi master举例：

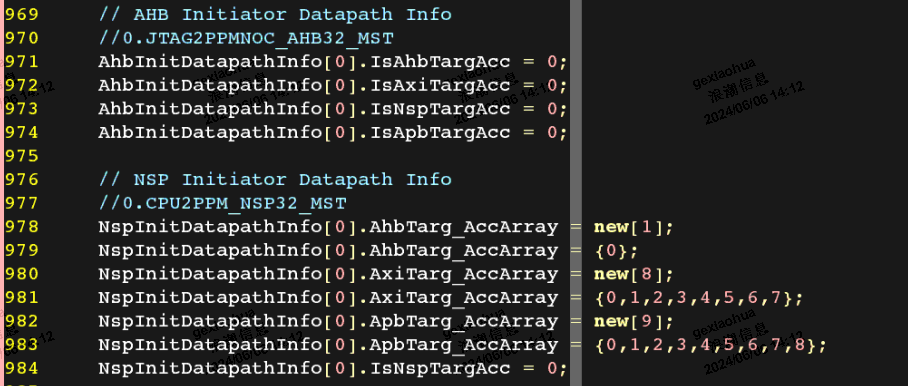


Step10：将所有slave的interface设置可以访问地址区间，区间是多个且不连续的，每一个包括起始地址和结束地址。包括InitStartAddr和InitEndAddr以及TargStartAddr和TargEndAddr，Init开头的表示进NoC地址，Targ开头的表示出NoC地址。自动化平台可以把InitStartAddr和InitEndAddr设置一样，同理把TargStartAddr和TargEndAddr设置一样——实际情况可能会不一样，因为NoC可能或过滤高位地址，如果遇到这个问题，需要用户在环境生成后去修改TargStartAddr和TargEndAddr，脚本无需关注这个问题。以AXI类型第一个访问区间为例示意一下设置可访问地址区间的方法：



注：中括号里的数字[0]表示第几个区间，而推入的具体数值从地址映射表读取，实现方式见本文第四章“输入件：地址映射表”的描述。AHB、APB、NSP的设置方法同理。

Step11：以master为视角建立路由关系，即某个master可以访问哪些类型的哪几个slave，脚本需要根据架构表的第二块信息确定谁可以访问谁，我们以下面的结果进行示例。



978~984行表示第0个Nsp master可以访问什么类型的哪个slave，其中：

AhbTarg\_AccArray = new[1]；//表示该master只可以访问1个AHB slave

AhbTarg\_AccArray = {0}；//表示只有第0个AHB slave属于该master的访问范围

AxiTarg\_AccArray = new[8]；//表示该master只可以访问8个Axi slave

AxiTarg\_AccArray = {0,1,2,3,4,5,6,7}；//表示这个8个Axi slave的所有编号

IsNspTargAcc = 0；//表示该master没有可以访问的Nsp slave

971~974行表示第0个AHB master可以访问什么类型的哪个slave，我们看到：

IsAhbTargAcc = 0；

IsAxiTargAcc = 0；

IsNspTargAcc = 0；

IsApbTargAcc = 0；

所以该master没有任何可以访问的interface（它只可以访问NoC内部service）。

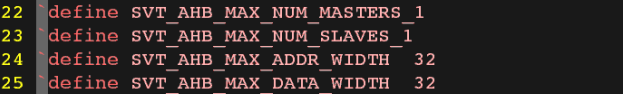
至此，xxx\_noc\_definitions\_pkg.sv的处理全部完毕。

* registermap.txt

生成一个空文件即可

* svt\_ahb\_user\_defines.svi

特别地，要关注4个宏定义，前面两个是master和slave个数，后面两个是位宽，它们define的格式有所差异，示意如下：



具体的个数数值需要脚本根据输入件信息填入以产生完整的宏定义，其余的宏定义从现有的NoC环境复制过来即可，不需要修改。

* svt\_apb\_user\_defines.svi

同理，不再赘述。

* svt\_axi\_user\_defines.svi

同理，不再赘述。

**5.3 atb/env**

* xxx\_noc\_interface.svi

按照interface格式创建一个文件，声明如下几个信号：

logic xxx\_noc\_clk；

logic end\_simulation；

logic xxx\_noc\_rst\_n；

logic xxx\_noc\_interrupt；

logic [1:0] axi\_resp\_cfg = 2’h0；

logic [1:0] ahb\_resp\_cfg = 2’h0；

logic axi\_slave\_delay\_flag = 1’b0；

logic power\_Idle；

logic power\_IdleAck；

logic power\_IdleReq = 1’b0；

int reset\_cnt；

bit [31:0] dbg\_flag；

logic sim\_end;

**5.4 atb/sim/vcs**

* coverage.hier

该文件填入内容：+tree top\_tb.th.dut\_inst\_xxx\_noc 1

**5.5 atb/tb**

* amba\_vip\_conn\_for\_noc.sv

该文件是挂接AMBA VIP的，挂接的方式是先声明中间信号（声明位置：同目录noc\_dut\_inst.sv），然后利用中间信号将DUT和VIP interface绑定起来，具体可以参考已有的NoC环境。如果有其它更好更方便的挂接方式，也可以不采用当前这个方案。

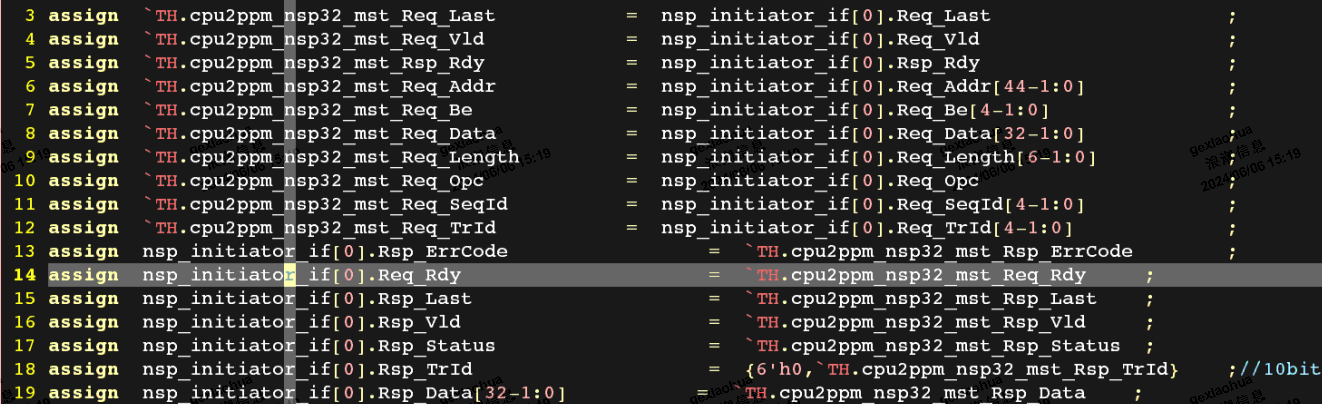
* noc\_dut\_inst.sv

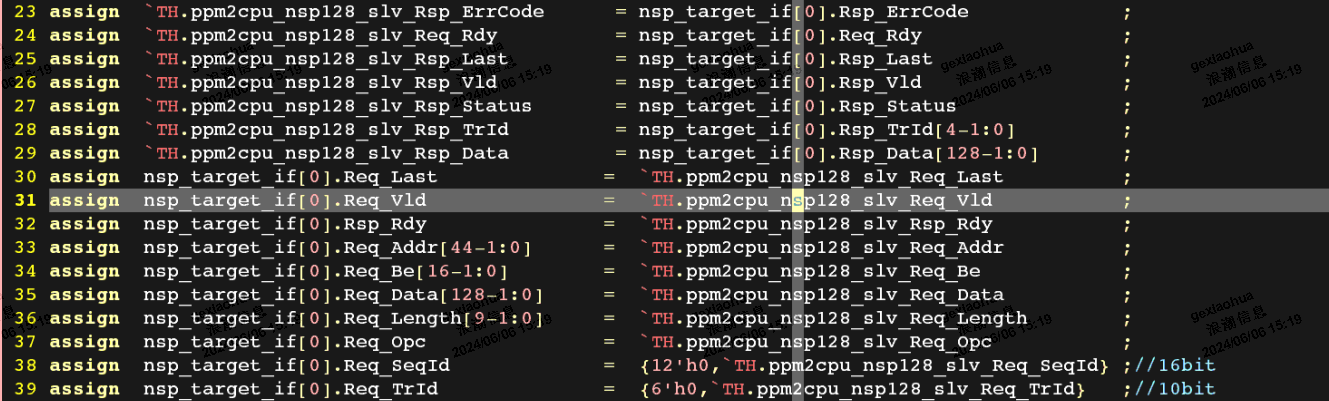
该文件是声明了用于连接DUT的中间信号，然后例化DUT代码，比较简单。

* nsp\_vip\_conn\_for\_noc.sv

该文件是挂接NSP VIP的，挂接的方式是先声明中间信号（声明位置：同目录noc\_dut\_inst.sv），然后利用中间信号将DUT和VIP interface绑定起来。

挂接NSP master和slave VIP的方法：





如果有其它更好更方便的挂接方式，也可以不采用当前上面这个方案。

* xxx\_noc\_general\_file.lst

该文件是整个DUT的filelist，脚本先把公共的库文件放进去。而DUT的RTL代码由用户在自动化平台交互界面进行输入，然后平台进行将其补充到文件中。公共的库文件包括：stdcc\_sync2\_rstn.sv、SynchronizerCell\_RstAsync.v、GaterCell\_RstAsync.v、stdcc\_rstn\_jtag.sv、stdcc\_rstn\_sync2.sv、stdcc\_clkicg.sv、stdcc\_mux.sv。

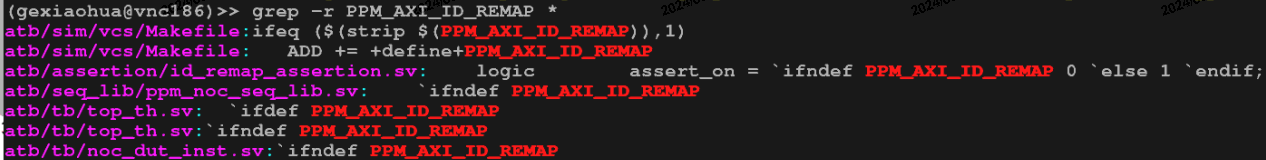
* top\_tb.sv

该文件大部分内容可以从当前已有NoC环境复制，但是这个地方涉及到了NSP VIP，需要修改为循环的方式，把VIP从第0个开始遍历到最后一个，位置如下：



* top\_th.sv

这个文件唯一需要特别提到的是，当前PPM\_NoC环境有使用PPM\_AXI\_ID\_REMAP这个宏，这个宏是为了区分单PPM\_NoC的UNIT测试环境还是PPM\_NoC+ID\_REMAP的BLOCK测试环境。处理前需要把这个宏相关的代码处理掉，处理方式单独沟通，此处不体现——最终的结果就只有NoC的UNIT测试环境。同时也涉及到的其它文件，如下：



**5.6 其它**

其它的目录或文件，可以直接从已有NoC环境进行复制，做好关键词替换即可。

**六、对于架构、设计的约束**

**6.1 对于架构的约束**

|  |  |
| --- | --- |
| 序号 | 描述 |
| 1 | 架构表格式上请继承九逸项目，避免修改 |
| 2 | 有些interface命名方式是xxx\_mst0、xxx\_mst1、xxx\_slv0这种，需要修改一下，把数字挪到mst前面 |

**6.2 对于设计的约束**

|  |  |
| --- | --- |
| 序号 | 描述 |
| 1 | 协议信号命名的时候，统一用XXX\_Ar\_Addr这样的驼峰+下划线组合方式 |
| 2 | 命名时钟的时候，请统一使用xxx\_regimeN\_clk类似这种方式，不使用带其它命名格式不统一的方式，不统一的方式脚本处理不了时钟连接 |

**--------------------------End--------------------------**