“基于AMBA总线的CRC运算核IP设计”验证报告

一、引言

对于我们设计的IP核，为了验证其功能正确性，我们进行了TestBench仿真调试，并通过了测试。但是，由于在TestBench仿真中，激励方式是通过人工输入的有限个测试用例，且需要人工判断输出波形是否正确，这导致了测试数据的覆盖率不高，且测试效率不高。

为了解决这个问题，提高IP核的可靠性，我们进一步搭建了UVM(Universal Verification Methodology)验证平台，来对我们设计的IP核进行算法验证。一方面，UVM可以生成大量的随机测试激励，覆盖率高；另一方面，UVM可以根据从软件角度编写的参考模型，自动判断待测硬件模块的输出结果是否正确，并输出验证报告。基于UVM的特点，这有效地改善了使用TestBench验证IP核功能的不足之处。

二、UVM验证平台搭建

2.1 软件平台

我们所使用的软件平台如表1所示。

表1 软件平台

|  |  |
| --- | --- |
| 操作系统 | Windows 7 64bit |
| 仿真器 | Questa Sim 10.6c |
| UVM库版本 | 1.1d |
| 开发语言 | System Verilog |

2.2 UVM模块设计

在我们的设计中，所搭建的UVM验证平台框图如图1所示。

Device Under Test(DUT)是我们的待验证的硬件模块，environment模块中，实例化了我们UVM的各个部件。输入代理包含用于激励的driver模块和用于收集激励数据的monitor模块，driver模块的激励来自于sequencer模块生成的数据包。输出代理包含用于收集DUT数据的monitor模块。scoreboard模块用于比较对于同一激励，来自reference model（RM）模块的理论输出和来自DUT的实际输出。

driver模块通过seq\_item\_port接口请求并接收由sequencer模块发送的数据包。我们例化了两个virtual interface（VIF），一个用在输入端，用于传输32位待计算数据，一个用在输出端，用于传输16位校验结果，这将UVM环境和DUT进行了连接。monitor模块通过VIF收集数据，此外，声明了analysis\_port端口用于输出收集到的包。agent模块将自己的analysis\_port输出端口地址指向monitor的analysis\_port端口。输入代理将自己的analysis\_port端口和RM模块的blocking\_get输入端口采用FIFO隔离，间接相连。同理，输出代理的analysis\_port输出端口和scoreboard的act\_port输入端口(blocking\_get类型)也是采用FIFO隔离，间接相连。最后，RM模块的analysis\_port输出端口和scoreboard的exp\_port输入端口(blocking\_get类型)采用FIFO隔离，间接相连。

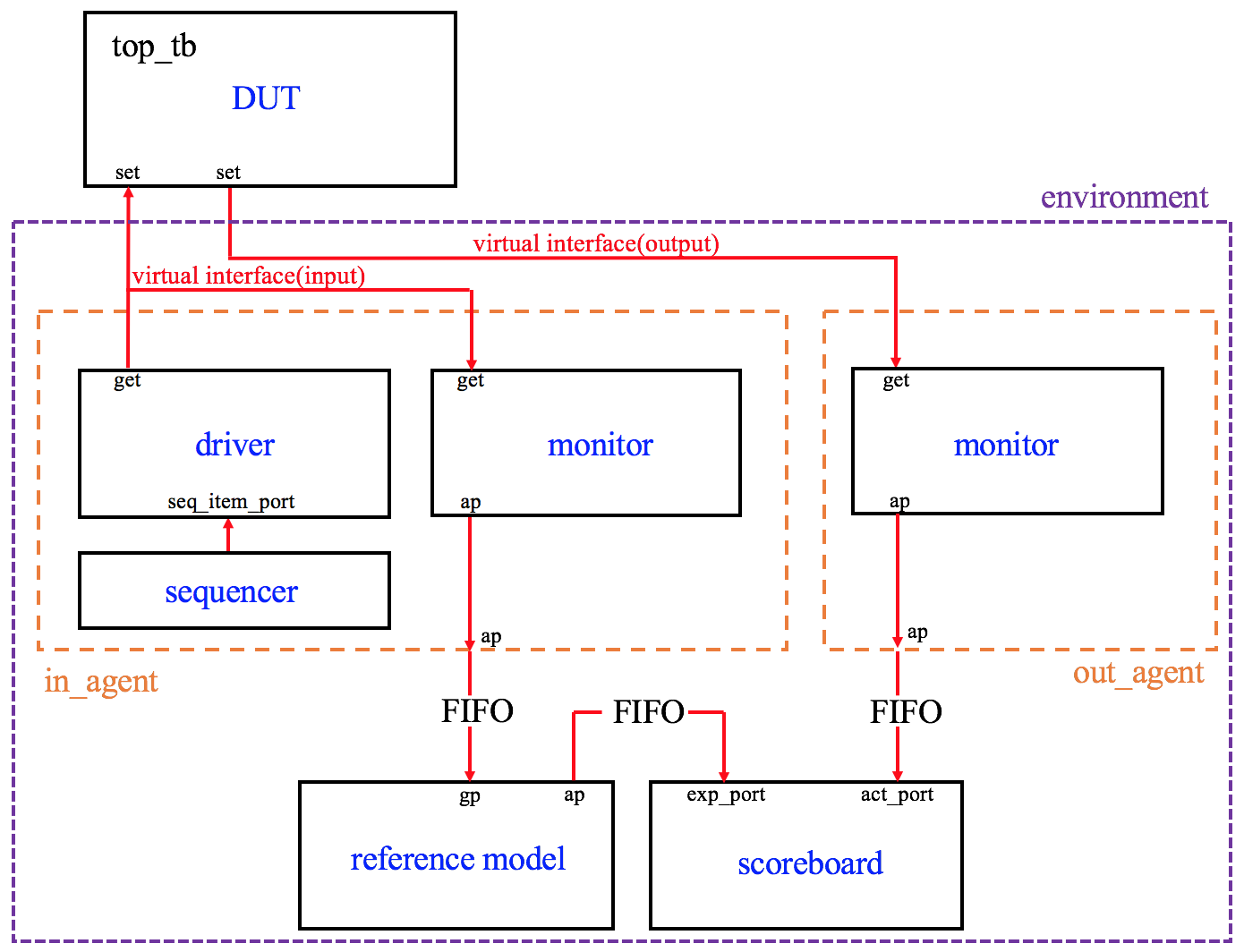


图1 所搭建的UVM框图

我们将数据以transaction包的形式传输，为了匹配我们的IP核设计，我们设计一个transaction包含如下内容：（1）32位输入数据（2）16位校验结果。

我们采用宏的方式，将DUT和RM的参数进行匹配，包括数据长度、CRC模式。同时，在driver和monitor中，采用无限循环阻塞，复位信号消失时跳出循环的方式，这使得复位信号还没有结束的时候，monitor不会驱动和收集数据包。

我们的RM模块从软件的角度实现CRC运算，计算过程无需考虑时序，而DUT模块是硬件实现CRC，两者出发点不同，但最后的计算结果应该相同。如果两者匹配，那么说明DUT模块通常正确。如果两者不匹配，那么说明两者至少有一个错误。

三、参考模型设计

根据CRC定义，我们基于递归，根据四种不同的CRC模式，采用串行按位计算CRC。算法流程图如图2所示。对于不同的CRC模式，采用不同的生成多项式，具体的生成多项式选择如表2所示。基于此算法，可以计算任意长度数据的CRC结果。

根据我们设计的IP核的功能描述，每个时钟周期计算16位数据，而一个transaction包含32位数据，因此我们采用1位bit变量作为标志位，据此选择取用前16位或者后16位来计算，计算后取反标志位即可。

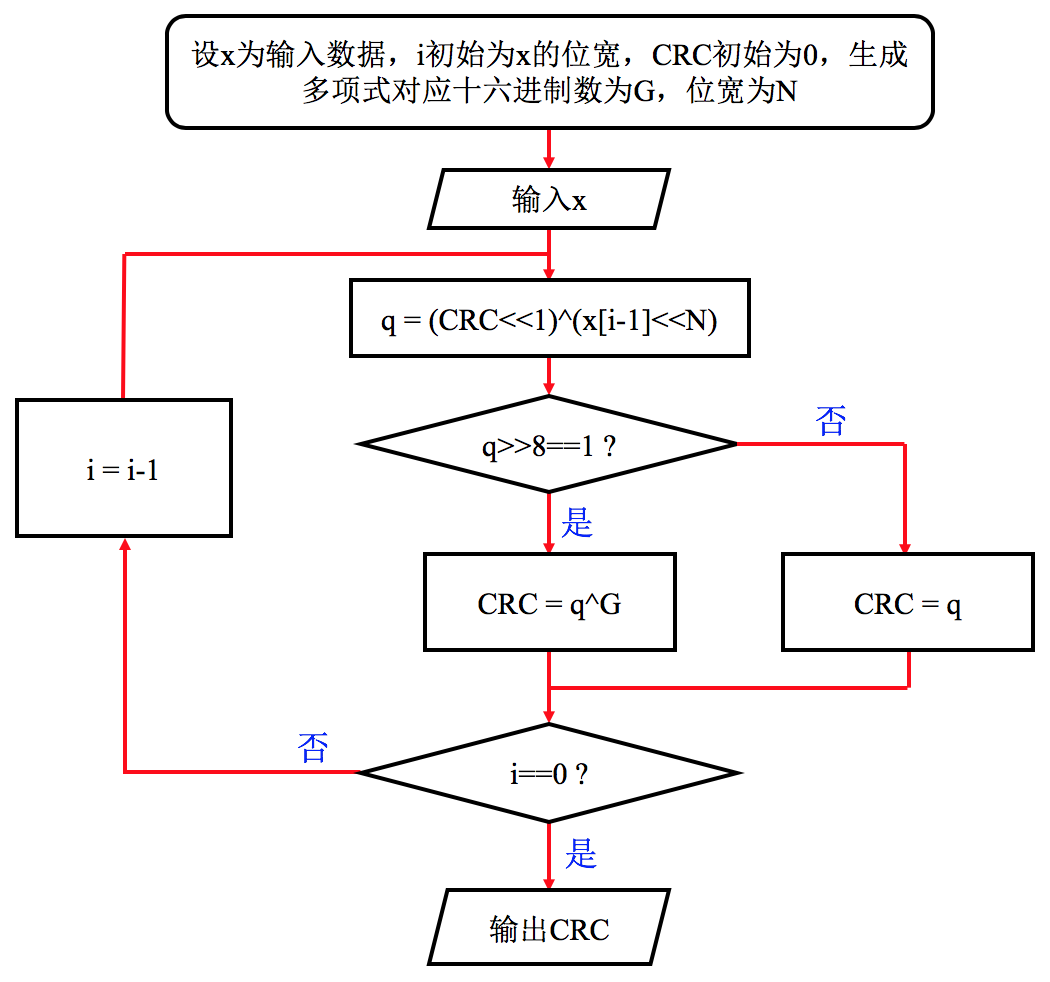


图2 参考模型所用的CRC算法流程图

根据我们设计的IP核的输出时序描述，将会在2个时钟周期后输出CRC值，但是参考模型从软件角度，是没有时序的，因此我们设计了移位寄存器对参考模型的输出结果进行缓存。由于输出结果最大为16位，因此定义48位bit变量pFIFO，赋初值为0，每一次计算，pFIFO向左移16位，低16位刷新为最新计算值，高16位进行输出，这样就实现了软硬件时序一致。

表2 不同CRC模式对应的参数

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CRC模式 | | 生成多项式 | | 对应十六进制数 | |
| CRC-8 |  | | ‘h131 | |
| CRC-12 |  | | ‘h180f | |
| CRC-16 |  | | ‘h18005 | |
| CRC-16CCITT |  | | ‘h11021 | | |

四、UVM验证结果

为了避免伪随机数导致的随机序列相同的问题，首先我们在仿真命令中加入-sv\_seed参数（Questa Sim中），这使得我们可以手动确定随机数种子，避免了序列相同。

我们在configv.sv文件中配置参数，分别设定CRC模式为CRC-8、CRC-12、CRC-16、CRC-16CCITT，随机生成1000组激励。

添加输入波形，运行仿真后，scoreboard输出结果如图3所示。我们产生了1000组32位随机激励，但是计算是按16位计算，因此一共有2000组输出。2000组计算全部正确，这验证了我们DUT基于四种CRC算法运行通常正确。

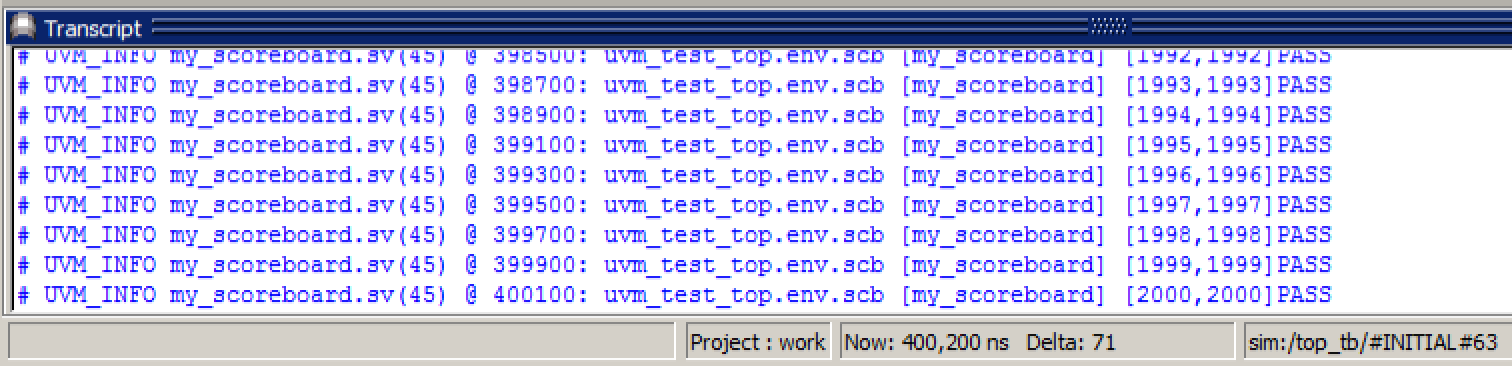


图3 scoreboard进行比较，通过了算法验证

查看输入波形，如图4所示，这表明了我们的driver的确是生成了1000组随机的32位激励。

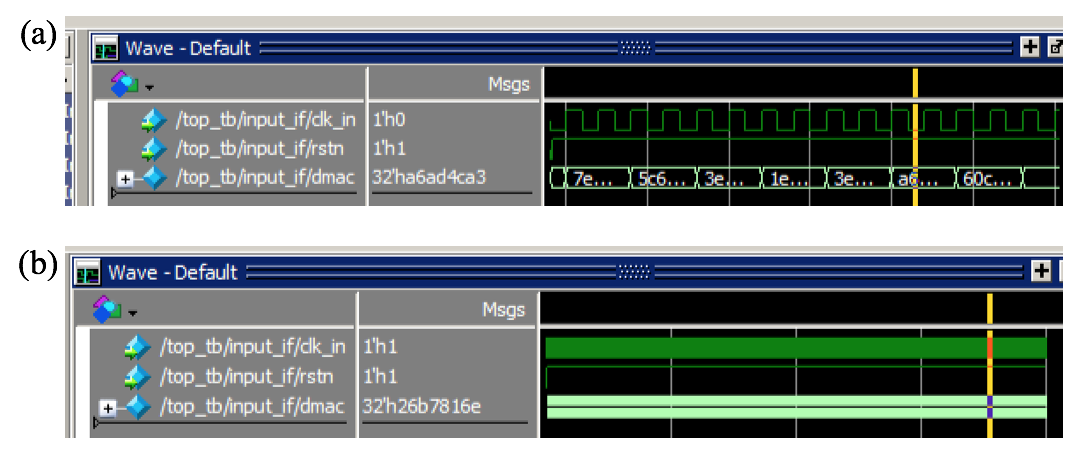


图4 driver产生的随机激励波形 (a)前几组激励;(b)整体波形