UVM项目实战部分8

路 桑

Agenda

1. 寄存器模型深度应用
2. 寄存器覆盖率
3. 总线解析
4. 性能分析

寄存器模型深度应用
后门访问更新

- 为了能够使得大家能够进一步认识寄存器模型的应用，在实战模块新的MCDF寄存器模型（自动生成）之上，我们需要首先更新用来做后门访问的路径。
- 随堂练习：
  - 按照随堂代码示例，补齐其余部分的寄存器路径。
  - 随机选择测试某些寄存器的后门读写访问。

寄存器模型深度应用

寄存器的随机配置

- 通过对寄存器模型中的某些寄存器做随机化。
- 随后将一些必须指定的寄存器域做写操作。
- 最后做寄存器模型的update。
- 随堂练习：
  - 按照随堂代码的要求，对寄存器模型做随机化、对指定域做配置，最后对其做update，即按照要求更新mcdf_full_random_virtual_sequence::do_reg()。
  - 通过后门访问，直接取得某些硬件寄存器域的数值，与之前的配置做比对，检查寄存器配置是否成功。

寄存器模型深度应用

寄存器域值的前门获取

- 寄存器模型不支持域值的前门读写。
- 我们需要在既有寄存器模型基础上，通过添加新的方法，实现寄存器域的读写。
- 随堂练习：
  - 理解寄存器模型中寄存器域（register field）是如何构成寄存器的，乃至整个寄存器模型。
  - 思考如何通过索引方式来获取默写寄存器域所在的寄存器的地址、比特位等信息。
  - 继而在路桑给出部分代码的基础上，添加少量代码实现寄存器域值的前门读写，并且通过基本的单元测试（unit test）。

Agenda

1. 寄存器模型深度
2. 寄存器覆盖率
3. 总线解析
4. 性能分析

寄存器覆盖率
寄存器模型更新

- 在之前寄存器模型更新时，我们的寄存器模型并没有生成与寄存器覆盖率有关的代码。
- 首先来认识寄存器覆盖率的必要性，以及通过寄存器覆盖率我们可以量化硬件的功能已经被测试过。
- 随堂测试：
  - 通过执行mcdf_reg_builtin_test，来检查最终寄存器模型的覆盖率。
  - 在adapter和寄存器模型中设置断点，理解寄存器覆盖率收集的逻辑。
  - 执行mcdf_reg_builtin_test，分析寄存器覆盖率收集，理解哪些coverpoint被覆盖，哪些没有被覆盖。
  - 通过分析寄存器覆盖率模型中，哪些数值无法被覆盖，继而更新寄存器文件(excel），在对应的列中添加需要收集的bin。

Agenda

1. 寄存器模型深度
2. 寄存器覆盖率
3. 总线解析
4. 性能分析

总线解析
解析内容

- 总线monitor的信息收集可以告诉我们一个完整的数据transfer的起始和结束时间，当然也包括地址、指令操作符、数据和其它信息。
- 我们可以通过分析总线信息，来协助解析处理器的行为，例如在何时发起了何种操作（寄存器/存储的访问）。
- 对于大规模数据的传输，我们还可以收集某一段时间窗口中的数据来计算数据带宽和硬件处理数据的延时。

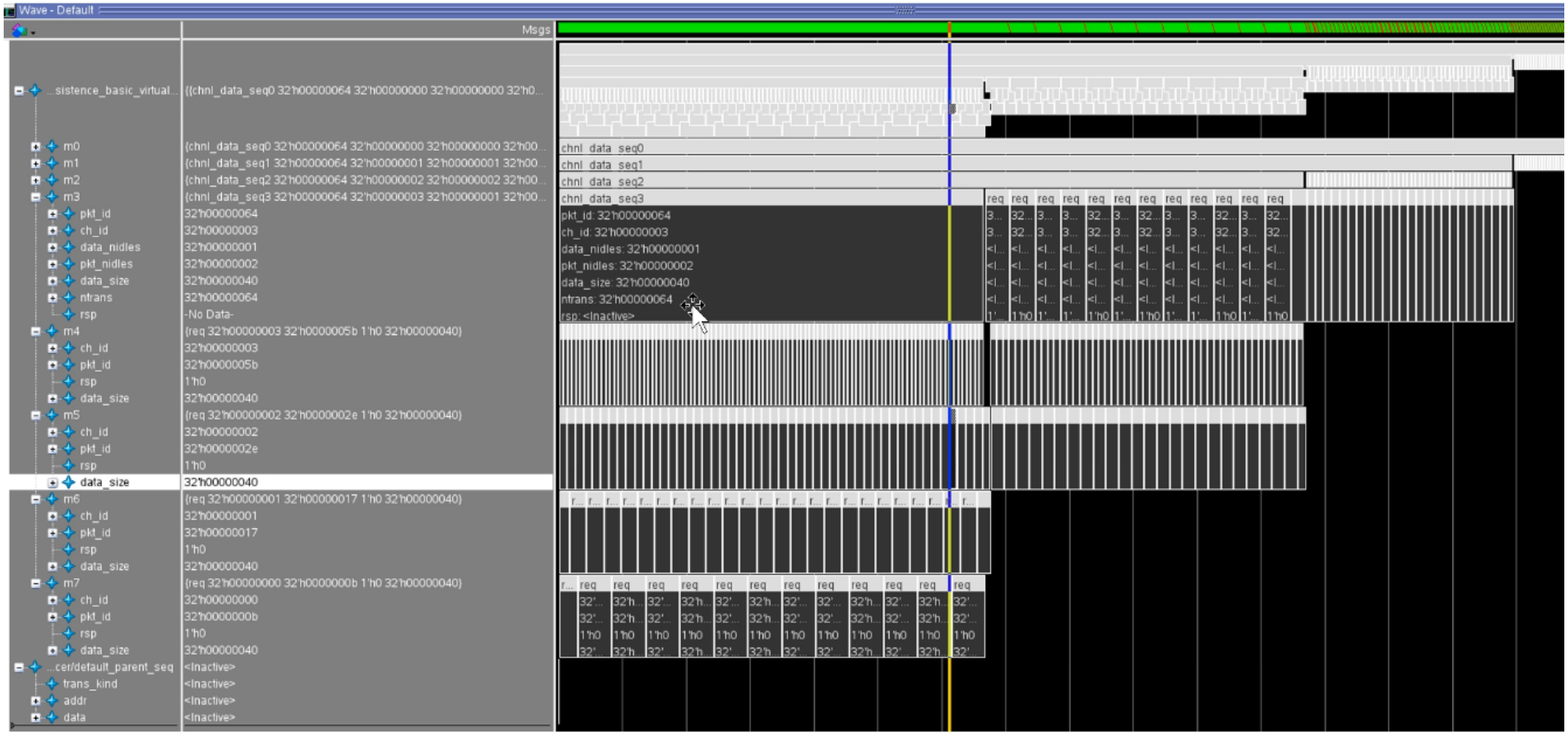
总线解析
UVM record使用

- 在通常硬件时序调试中，我们看到的数据总线即会将总线中的各个成员收集在波形窗口中，这给我们的调试带来了不便。
- 在UVM中我们可以通过在monitor中实现transaction record的功能，结合UVM record方法和EDA仿真器的支持，使得总线数据传输在时序和波形上的调试更为方便。
- 随堂练习：
  - 根据路桑给的随堂代码，跑仿真添加transaction到波形中，从波形窗口中理解transaction抽象级别的信息内容和调试的便捷性。

总线解析
UVM record使用

- 在tb.sv中设置
  uvm_config_db#(int)::set(uvm_root::get(), "*", "recording_detail", 1);
- 在仿真时添加仿真选项
  vsim -novopt -classdebug -msgmode both -uvmcontrol=all work.tb
- log -r /*
- View -> UVM Details，在SIM窗口中选中相应的UVM组件，继而在UVM Details窗口中选中对应的transaction stream，右键点击添加到波形窗口，观察波形窗口中的transaction。

UVM record使用



处理器总线在线解析

- 结合monitor采样bus transaction和寄存器模型，在仿真时通过采样的总线信息，从寄存器模型中查找对应的寄存器，继而将读写信息、寄存器信息乃至域的信息都一并打印出来。
- 这种方式便于摆脱波形调试，在前期利用仿真信息（simulation log）即能够完成调试，这有利于多数不了解UVM验证环境的人可以展开有处理器在内的系统调试。
- 这种总线在线解析的方式与transaction波形展示的方式可以互相配合。
- 随堂练习：
  - 根据路桑给的代码，跑仿真注意仿真log文件，并且与对应的寄存器访问序列对应，检查解析出的信息是否一致，查找关键字“REGANA”。
  - 阅读在线解析代码mcdf_bus_analyzer::do_reg_analysis()，理解其解析的逻辑。

Agenda

1. 寄存器模型深度
2. 寄存器覆盖率
3. 总线解析
4. 性能分析

性能分析
介绍

- 正如路桑在SV模块的验证方法介绍中讲到，功能验证目前已经不能充分验证芯片系统，与之而来的是性能验证(performance)和效能(power)验证。
- 效能验证的两部分无论是UPF用来做电源域的模拟开关验证还是与能耗收集评估验证，均需要对应的EDA工具，而且考虑到专业性较强，我们暂不在实战模块中展开。
- 性能验证需要考虑到两部分环境：
  - 提供性能验证的平台需要足够快，继而使得软件、固件代码可以在其之上运行。
  - 性能验证需要结合将来会运用的软件、固件代码，这指的是我们需要采用真实或者贴近实际用例的软件代码。

性能分析
介绍

- 用来做性能分析的平台不单包括simulator，还会在FPGA和emulator上进行，这是基于它们出色的仿真速度的考量。
- 我们在实战模块中主要介绍的是性能分析的思想，至于在何种平台、是否使用实际使用的软件代码都不会妨碍我们理解性能分析的核心，它们包括：
  - 对目标硬件发送大量的数据。
  - 在固定的周期中采集数据量并且计算带宽。
  - 从数据的接收时间到数据的送出的延时计算。

性能分析
性能计算

- 创建一个性能收集组件，置于UVM顶层环境中。
- 从各个monitor中收集transaction，并进行performance计算。
- 对于MCDF的数据吞吐量计算，以channel所有通道的数据输入总和为输入带宽，以formatter的数据输出为输出带宽。
- 通过标记输入数据和输出数据，对同一个数据包在MCDF的输入和输出计算其延时，作为performance的又一个标准。

性能分析
性能计算

- 随堂练习
  - 根据路桑给的随堂代码，对各个monitor的analysis port连接到performance subscriber组件的analysis fifo中。
  - 从各个装载不同数据的analysis fifo中分析数据，将其packet开始时间和长度信息统一提取到定义的performance transaction类型中。
  - 设置带宽计算的时间窗口长度。
  - 根据路桑给出的部分代码，实现用来计算输出带宽的方法mcdf_bus_analyzer::calculate_bandwidth()。
  - 根据输出packet中的头部数据的开始时间和对应的输入packet中数据的开始时间，计算其大致的延时，实现对应方法mcdf_bus_analyzer::calculate_delay()。