文档版本： Ver1.0
最后修改日期： 2015-05-30
修改人： William

E 课网 – UVM 实战培训

www.eecourse.com

klin@eecourse.com

SVV 实验 – Lab02_Guide

实验简介：

利用 SystemVerilog 中面向对象的编程（OOP）理念，将零散的输入信号进行抽象，并将它们封装进一个“Packet”类中，使得激励信号的产生和处理更加方便，我们将这个类称为“事务类”，将这个类的对象称为“事务”。利用 SystemVerilog 的随机化属性，可以很容易的产生大量的随机化测试事务，再结合“约束”，使得这些随机化发生在一定的范围之内，从而保证了事务既是“随机的”又是“合法的”。另一方面，升级 Testbench，使其激励信号的驱动和结果的收集并行执行，并添加结果检查功能。

需要注意的是，由于 DUT 是由两个子模块构成的两级流水系统，因此从输入到对应的输出之间相差两个时钟周期。

实验目的：

★ 熟练掌握如何将激励抽象为“事务类”，如何为类中的成员添加随机化属性以及如何为成员指定约束项；

★ 修改 Testbench 的结构，增添新的功能。

实验准备：

➢ 进入本次实验的实验目录

```
cd verification/svv/lab02
```

该目录中包含了本次实验所需要的相关代码和文档。

➢ 请确认你已经获得本次实验的 DUT 参考手《E 课网_UVM 实战培训_SVV 实验_DUT 手册.pdf》

➢ 如果在上述准备工作中遇到任何困难，请及时与讲师联系解决。

实验步骤：

1. 事务类 – Packet

打开 Packet.sv 文件，下来来一起认识一下这个文件中的内容。

① 使用 **rand** 关键字来为 Packet 类中的成员变量添加随机属性，这样在随机化 Packet 类型的对象时，凡是具有 **rand** 属性的成员均会随机化一个值。有些情况下，需要产生的测试量庞大，我们不可能一个一个的人工产生，随机化就很好的解决了这个问题。注意，只有具有了 **rand** 属性的成员才会在对象随机化时产生一个随机的值，没有该属性的成员则不会。

② 虽然随机化可以帮助我们轻松的产生大量的随机测试激励，但是如果不对随机范围做一些约束的话，那么产生的数据很可能是不合法的，这些不合法的数据并不是我们想要的。因此在 Packet 中为成员变量添加约束是十分必要的，它能使得变量在约束的范围内随机化，从而保证产生的数据是有效的。使用 **constraint** 来为变量指定约束。

③ 指定约束的方式几乎不受限制，只要约束之间不会发生冲突，可以使用：指定值的范围、依赖其他变量的条件约束等等。

```
class Packet;

    rand    reg [`REGISTER_WIDTH-1:0]   src1;
    rand    reg [`REGISTER_WIDTH-1:0]   src2;
    rand    reg [`REGISTER_WIDTH-1:0]   imm;
    rand    reg [`REGISTER_WIDTH-1:0]   mem_data;
    rand    reg                         immp_regn_op_gen;
    rand    reg [2:0]                   operation_gen;
    rand    reg [2:0]                   opselect_gen;

    string  name;

    constraint Limit {
        src1 inside {[0:65534]};
        src2 inside {[0:65534]};
        imm inside  {[0:65534]};
        mem_data inside {[0:65534]};

        opselect_gen inside {[1:1]};   //only arithmetic inputs

        if ((opselect_gen == `ARITH_LOGIC)){
            operation_gen inside {[0:7]};
        }
        else if ((opselect_gen == `SHIFT_REG)) {
            immp_regn_op_gen inside {0};
            operation_gen inside {[0:3]};
        }
        else if ((opselect_gen == `MEM_READ)) {
            immp_regn_op_gen inside {1};
            operation_gen inside {[0:4]};
        }
        else if ((opselect_gen == `MEM_WRITE)) {
            immp_regn_op_gen inside {1};
            operation_gen inside {[0:7]}; // just make sure it does not matter
        }
    }

    extern function new(string name = "Packet");
endclass

function Packet::new(string name = "Packet");
    this.name = name;
endfunction
```

赋予成员变量随机化属性

对成员变量的约束

将其 opselect 约束为 1，表示产生算术运算的指令。当然也可以更改该约束：inside{[0:1], [4:5]}

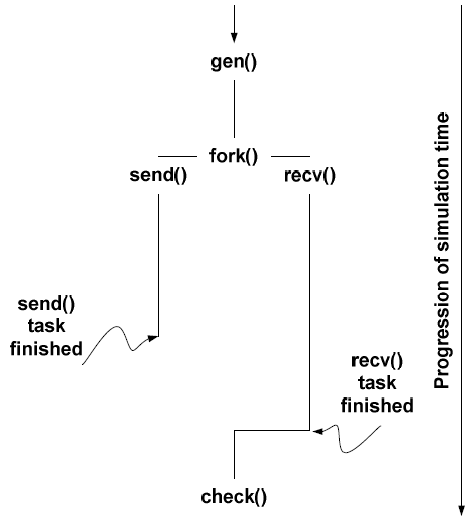
根据 opselect_gen 的不同值，来设定不同的约束，使得随机化的值是合法的。

2. 修改 Testbench，完善其功能

在该实验中，我们修改并添加了一些方法：

- 修改 gen() 任务，使它可以随机化产生 packets 对象；
- 修改 send_payload()任务，负责将由 gen()产生的所有 packets 对象转化为激励信号驱动给 DUT，并将这些对象寄存下来用作后面的结果对比；
- 新增 recv()任务，该任务会与 send()任务并行执行，用来同步的收集 DUT 的输出结果；
- 新增 check()任务，该任务会在 send()和 recv()任务都结束之后执行，负责对 DUT 的输出结果做检查来判断 DUT 是否正确。

① Testbench 的执行过程如下图所示：



下面是实现上述过程的代码：

```
initial begin
    run_n_times = 21;
    reset();
    repeat(run_n_times) begin
        $display($time, "ns: Sending Another Packet");
        gen();
        fork
            send();
            recv();
        join
        check();
    end
    repeat(5) @(Execute.cb);
end
```

使用 fork join 来启动两个并行的线程，使得激励的驱动和结果的收集同时进行。

② 激励发生器 – gen()

```
task gen();

    number_packets = $urandom_range(3,12);

    GenPackets = new[number_packets];
    for (i=0; i<number_packets; i++) begin
     GenPackets[i] = new();
    end
    for (i=0; i<number_packets; i++)    begin
      pkt2send.name = $psprintf("Packet[%0d]", i);
      if (!pkt2send.randomize()) begin
        $display("\n%m\n[ERROR]%0d gen(): Randomization Failed!", $time);
        $finish;
       end
      GenPackets[i].src1 = pkt2send.src1;
      GenPackets[i].src2 = pkt2send.src2;
      GenPackets[i].imm = pkt2send.imm;
      GenPackets[i].mem_data = pkt2send.mem_data;
      GenPackets[i].immp_regn_op_gen = pkt2send.immp_regn_op_gen;
      GenPackets[i].operation_gen = pkt2send.operation_gen;
      GenPackets[i].opselect_gen = pkt2send.opselect_gen;
    end
    $display($time, "ns: [GENERATE] Generate Finished Creating %d Packets
", number_packets);
endtask
```

为动态数组分配内存空间，空间的个数取决于随机出来的 packet 的个数。

调用 randomiz()函数来随机化 packet 对象，这样 packet 中所有具有 rand 属性的成员都会随机化出一个值。

将产生的 packet 对象存入动态数组中

pkt2send.randomize()函数使得 pkt2send 中具有 rand 属性的成员变量随机化一个值。pkt2send.randomize()函数有返回值，当随机化成功时，将会返回非 0；当失败时，会返回 0，因此可以通过判断它的返回值来确定是否随机化成功。

③ 激励驱动器 – send()

任务 send()负责将 gen()产生的激励驱动到 DUT 端口上。而 send()的所有功能是通过 send_payload()任务来实现的，其过程如下：

每过一个时钟周期，从 GenPackets 动态数组中取出一个 packet 对象，然后按照 DUT 的输入时序要求将 packet 中的数据驱动到 DUT 对应的输入端口上。

上述的过程会一直持续直到 GenPackets 中的 packets 都被发送出去为止。

当所有的 packet 发送完毕之后，清空 GenPackets 数组。

```
task send_payload();

    Execute.cb.enable_ex <= 1'b1;
    packets_sent = 0;
    i = 0;
```

```
    while(i < GenPackets.size()) begin

        pkt2send = GenPackets[i];

        Execute.cb.src1              <= pkt2send.src1;
        Execute.cb.src2              <= pkt2send.src2;
        Execute.cb.imm               <= pkt2send.imm;
        Execute.cb.mem_data_read_in <= pkt2send.mem_data;
        Execute.cb.control_in        <= {pkt2send.operation_gen,
               pkt2send.immp_regn_op_gen, pkt2send.opselect_gen};

        packets_sent++;

        Inputs.push_back(pkt2send);

        Enables.push_back(1'b1);
        i++;
        @(Execute.cb);
    end

    GenPackets.delete();
endtask
```

从 GenPackets 取出 Packet

将 Packet 对象中成员的值按照对应关系驱动到接口上去。

将发送的 Packet 存储下来。

清空数组中的 Packet，为存储下一组指令做准备。

代码 Inputs.push_back(pkt2send) 是将 Testbench 发送给 DUT 的激励保存在队列 Inputs 中，目的是为了在仿真的最后使用 check() 函数来检查 DUT 输出的结果是否正确。

④ 输出监视器 – recv()

为了获取 DUT 的输出结果，我们需要一个用于采集输出信号的监视器，在该实验中使用 recv() 任务来完成。需要注意的是，本次实验并没有对子模块“预处理器”的输出进行结果采集，它属于 DUT 的内部信号。

可以看到，在 recv() 内部，还有一个用于从接口上采集信号的 get_payload() 任务。

```
task recv();
    int i;
    // delay for synchronization with the outputs from DUT
    @ (Execute.cb);
    repeat(number_packets+1) begin
      @ (Execute.cb);
      get_payload();
    end
endtask
```

```
task get_payload();
    aluout2cmp = Execute.cb.aluout;
    mem_en2cmp = Execute.cb.mem_write_en;
    memout2cmp = Execute.cb.mem_data_write_out;

    aluout_q.push_back(aluout2cmp);
    memout_q.push_back(memout2cmp);
    mem_en_q.push_back(mem_en2cmp);
endtask
```

从接口采集 DUT 的输出

将采集到的输出信号存储下来

任务 recv()中的第一个@(Execute.cb)是为了保证采集到的结果是有效的输出。值得注意的是，任务 send()和 recv()使用 fork join 启动的两个并行的线程，由于 DUT 是一个二级流水系统，因此从某一组信号的输入到对应的输出有 2 个时钟周期的延迟，所以必须保证在正确的时间点去采集 DUT 的输出结果。

⑤ 计分板 – check()

计分板或者叫比较器 check()是为了检查 DUT 的输出是否正确。为了实现这个功能它必须具有模拟 DUT 功能的能力。

check()模拟 DUT 的功能，将输入给 DUT 的激励作为怎么的输入数据，然后根据这些数据模拟 DUT 的功能计算出对应的结果，这个结果我们称为是“期望结果”，也就是什么期望 DUT 的输出结果。将这个期望的结果与 DUT 的实际输出结果做比较，如果相同，我们就认为 DUT 的功能是正确的；如果不相同，我们认为 DUT 存在功能上的错误。在本次实验的代码中，我们仅仅实现了 check()的算术运算的能力。请认真阅读 check()中的代码，弄清楚它们的含义。

⑥ Makefile

复制 Lab01 中的 Makefile，使用 Makefile 来进行本次实验的仿真。

实验问题：

请将以下问题以回帖的方式在论坛上作答。

1. 请根据实际的代码，将 Packet 中的成员变量与 DUT 的输入信号一一对应起来。

答：

2. 请说明在 Packet 中定义的 constraint 中被约束的变量的随机范围。

答：

3. 在 Testbench 中，任务 send()和 recv()的执行顺序是什么。

答：

4. 请结合波形说明，recv()任务中为什么要 repeat(number_packets+1)次。

答：

5. 请结合波形说明，check()任务中，为什么要在 while(Inputs.size()!=0) begin 之后先执行 check_arith()函数。

答：

6. 如何只产生 MEM_READ 类型的激励。

答：

7. 在 check() 任务中加入打印语句：当“期望值”与 DUT 的输出不一致时，打印出相关错误信息（打印出错误信息即可，信息的内容自定）

答：

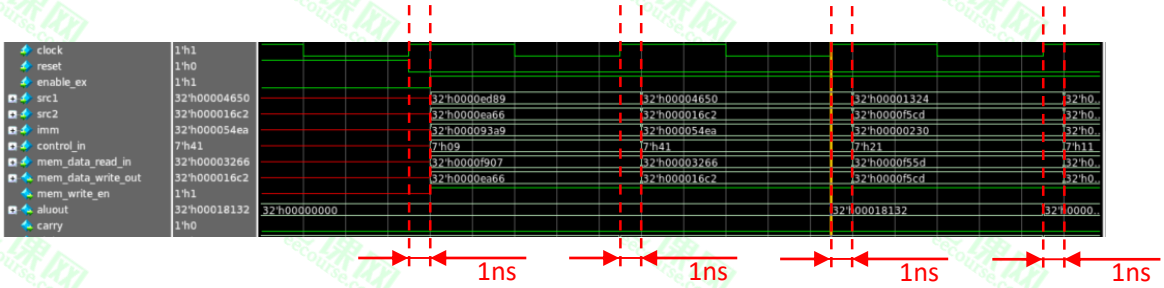
8. 修改 check() 任务，使它可以完成所有操作类型的检查工作。

答：

9. 随机化产生出多种操作类型的 packet。

答：

10. 运行仿真，打开 DUT 端口的波形文件，观察：



请解释一下，为什么 DUT 输入端的信号（reset 信号除外）都是在时钟 clock 的上升沿之后延迟 1ns 后才被驱动新的值，这一动作为什么没有刚好发生在时钟的上升沿处？

答：