基于 SV+UVM 搭建 SOC/ASIC 验证平台

UVM-1.1 中提供了一个 UBUS 的例子，但是该例子对于刚刚入门的人来说还是需要一定时间去消化的，本文对该例子进行一步一步的简化，可以帮助理解。

[1-1] 如何顺序的写 UVM 平台（1）-Basic

1. 平台可以在前期规划好，但是对于搭建平台的人来说，调试永远是最大的问题，如果都将一个个 component 都写完了，调试起来还是有点痛苦的，所以我更倾向于一步一步的调试平台；先写一个可以 pass 的基本平台，然后在不断的扩展该平台，最后在各个 component 中加入所需要的 function 或者 task。当然，当对搭建平台数量以后，现在基本对平台中的 component 一次性搭建完成，然后调试并添加需要的 function 或者 task 即可。
2. 最简单的 UVM 平台，一个 interface，一个 DUT，一个 TOP，一个 test，一个 ENV 就可以工作了，然后慢慢的添加各个 component；
3. 写 interface
4. 写 top module，在 top 中例化 DUT，interface 和 DUT 在 top 中 include

```
1  `define UBUS_ADDR_WIDTH 16
2
3  `include "dut_dummy.v"
4  `include "ubus_if.sv"
5
6  module ubus_tb_top;
7      import uvm_pkg::*;
8
9      ubus_if    vif();
10
11   dut_dummy dut(
12     vif.sig_request[0],
13     vif.sig_grant[0],
14     vif.sig_request[1],
15     vif.sig_grant[1],
16     vif.sig_clock,
17     vif.sig_reset,
18     vif.sig_addr,
19     vif.sig_size,
20     vif.sig_read,
21     vif.sig_write,
22     vif.sig_start,
23     vif.sig_bip,
24     vif.sig_data,
25     vif.sig_wait,
26     vif.sig_error
27   );
28
29     initial begin
30         uvm_config_db#(virtual ubus_if)::set(uvm_root::get(),"*","vif",vif);
31         run_test();
32     end
```

uvm_config_db#(virtual ubus_if)::set(uvm_root::get(), "*", "vif", vif);

run_test();

5. 写 Makefile，此时编译可以通过

```
all: clean run

run:
        irun -access rw -uvmhome ${UVM_HOME} +UVM_VERBOSITY=${UVM_VERBOSITY} -quiet +define
+UVM_OBJECT_MUST_HAVE_CONSTRUCTOR -incdir . ubus_tb_top.sv

clean:
        rm -rf *~ core csrc simv* vc_hdrs.h ucli.key *.log
```

6. 写自定义的 package，然后在 top 中 include 该 package

```
1 package ubus_pkg;
2     import uvm_pkg::*;
3     `include "uvm_macros.svh"
4     typedef uvm_config_db#(virtual ubus_if) ubus_vif_config;
5     typedef virtual ubus_if ubus_vif;
6 endpackage
```

typedef uvm_config_db#(virtual ubus_if) ubus_vif_config;

typedef virtual ubus_if ubus_vif;

后来证明，这两句话在 ubus 的 env 中根本没有用上；

7. 定义 Environment，并将该文件加入到自定义的 package 中，这个时候编译不能通过

```
1 class ubus_env extends uvm_env;
2     protected virtual interface ubus_if vif;
3     `uvm_component_utils(ubus_env)
4
5
6     function new(string name, uvm_component parent);
7         super.new(name,parent);
8     endfunction
9
10    function void build_phase(uvm_phase phase);
11        super.build_phase(phase);
12        if(!uvm_config_db#(virtual ubus_if)::get(this,"","vif",vif))
13            `uvm_fatal("NO vif",{"virtual interface must be set for:",get_full_name(),"
  .vif"});
14    endfunction
15 endclass
```

此处的 get 和 top 中的 set 是一对，如果 top 中没有 set 则会报告 • uvm_fatal 中的错误

if(!uvm_config_db#(virtual ubus_if)::get(this, "", "vif", vif))

`uvm_fatal("NOVIF",{"virtual interface must be set for: ",get_full_name(),".vif"});

```
1 package ubus_pkg;
2     import uvm_pkg::*;
3     `include "uvm_macros.svh"
4     typedef uvm_config_db#(virtual ubus_if) ubus_vif_config;
5     typedef virtual ubus_if ubus_vif;
6     //`include "ubus_pkg.sv"
7     `include "ubus_env.sv"
8 endpackage
```

8. 定义 base_test，需要在 top 中 include 该 test 文件，并在 makefile 中加入编译该 test 的命令；此时可以再次编译通过，并运行最 basic 的 testcase

```
class ubus_base_test extends uvm_test;
    `uvm_component_utils(ubus_base_test)
    ubus_env env;
    uvm_table_printer printer;

    function new(string name="ubus_base_test",uvm_component parent=null);
        super.new(name,parent);
    endfunction

    virtual function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        env=ubus_env::type_id::create("env",this);
        printer =new();
        printer.knobs.depth=3;
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        uvm_report_info(get_full_name(),"start of test run_phase...",UVM_LOW);
    endtask
endclass
```

```
`define UBUS_ADDR_WIDTH 16

`include "dut_dummy.v"
`include "ubus_if.sv"
`include "ubus_pkg.sv"
module ubus_tb_top;
    import uvm_pkg::*;
    import ubus_pkg::*;
    `include "test_lib.sv"

    ubus_if    vif();

  dut_dummy dut(
```

编译第一次通过

总结一：在一个芯片的验证平台中，总会给一个最 **basic** 的 **base_test**，但是可能每个人负责验证的部分是不一样的，比如说我要验证 **USB**，那我一定会从 **base_test** 中派生一个 **usb_base_test** 来给自己用，这样我可以在 **usb_base_test** 加入任何我想要的函数，而不会影响到其他人。

总结二：如何从 **test** 中传递参数到 **top_tb**,如果用 **uvm_config_db** 在 **base_test** 中设置变量的值，那么这些变量在 **environment/agent/dirver** 等等中可以 **get** 到的，但是在 **top_tb** 中不能 **get** 到的？采用下面的方式就可以

uvm_config_db #(int)::set(null,"uvm_test_top","set_usb_single_test",1);

总结三：需要注意的是，不要把定义 **interface** 的文件 **include** 在 **package** 中，这会导致编译不过的。

总结四：**uvm_report_info 和宏`uvm_info 在用法是没有区别的，都是用来打印消息的。**

[1-2] 如何顺序的写 UVM 平台（2）-MasterAgent

9. 定义 sequence item，注意，一种类型的 transaction 需要对应一个 driver；

```
`uvm_object_utils_begin(ubus_transfer)
  `uvm_field_int        (addr,              UVM_DEFAULT)
  `uvm_field_enum       (ubus_read_write_enum, read_write, UVM_DEFAULT)
  `uvm_field_int        (size,              UVM_DEFAULT)
  `uvm_field_array_int(data,                UVM_DEFAULT)
  `uvm_field_array_int(wait_state,          UVM_DEFAULT)
  `uvm_field_int        (error_pos,         UVM_DEFAULT)
  `uvm_field_int        (transmit_delay, UVM_DEFAULT)
  `uvm_field_string     (master,            UVM_DEFAULT|UVM_NOCOMPARE)
  `uvm_field_string     (slave,             UVM_DEFAULT|UVM_NOCOMPARE)
`uvm_object_utils_end
```

```
typedef enum {NOP, READ, WRITE} ubus_read_write_enum;

class ubus_transfer extends uvm_sequence_item;
    rand bit[15:0]      addr;
    rand ubus_read_write_enum  read_write;
    rand int unsigned           size;
    rand bit[7:0]       data[];
    rand bit[3:0]       wait_state[];
    rand int unsigned       error_pos;
    rand int unsigned       transmit_delay=0;
    string          master="";
    string          slave="";

    constraint c_read_write{
        read_write inside {READ,WRITE};
    }
    constraint c_size{
        size inside{1,2,4,8};
    }
    constraint c_data_wait_size{
        data.size()==size;
        wait_state.size()==size;
    }
    constraint c_transmit_delay{
        transmit_delay<=10;
    }
    `uvm_object_utils_begin(ubus_transfer)
        `uvm_field_int(addr, UVM_DEFAULT)
        `uvm_field_enum(ubus_read_write_enus,read_write,UVM_DEFAULT)
        `uvm_field_int(size, UVM_DEFAULT)
        `uvm_field_array_int(data, UVM_DEFAULT)
        `uvm_field_array_int(wait_state, UVM_DEFAULT)
        `uvm_field_int(error_pos, UVM_DEFAULT)
        `uvm_field_int(transmit_delay,UVM_DEFAULT)
        `uvm_field_string(master, UVM_DEFAULT|UVM_NOCOMPARE)
        `uvm_field_string(slave, UVM_DEFAULT|UVM_NOCOMPARE)
    `uvm_object_utils_end

    function new(string name="ubus_transfer_inst");
        super.new(name);
    endfunction
endclass
```

需要思考哪些内容需要在 transaction 中定义？

10. 定义好了 transaction，就可以开始定义 base sequence 了；注意，在 base_sequence 的 pre_body 和 post_body 中定义 raise_objection 和 drop_objection 是很有好处的；将 base sequence 定义成虚基类，只有派生后才能进行实例化；

```
virtual class ubus_base_sequence extends uvm_sequence#(ubus_transfer);
    function new(string name="ubus_base_seq");
        super.new(name);
    endfunction

    //raise in pre_body so the objection is only raised for root sequences.
    //there is no need to raise for sub-sequences since the root sequence
    //will encapsulate the sub-sequence.

    virtual task pre_body();
        if(starting_phase!=null) begin
            `uvm_info(get_type_name(),$sformatf("%s pre_body raising %s objection",get_sequence_path(),starting_phase.get_name()),UVM_MEDIUM);
            starting_phase.raise_objection(this);
        end
    endtask

    //drop the objection in the post_body so the objection is removed when the root sequence is complete
    virtual task post_body();
        if(starting_phase!=null) begin
            `uvm_info(get_type_name(),$sformatf("%s post_body() dropping %s objection",get_sequence_path(),starting_phase.get_name()),UVM_MEDIUM);
            starting_phase.drop_objection(this);
    end
    endtask
endclass
```

11. 派生 sequence；先派生出一个 sequence 用于平台调试

```
class read_byte_seq extends ubus_base_sequence;
    function new(string name="read_byte_seq");
        super.new(name);
    endfunction

    `uvm_object_utils(read_byte_seq)

    rand bit[15:0] start_addr;
    rand int unsigned transmit_del=0;
    constraint transmid_del_ct{(transmit_del<=10);}

    virtual task body();
        `uvm_do_with(req,{req.addr==start_addr;
                          req.read_write==READ;
                          req.size==1;
                          req.error_pos== 1000;
                          req.transmit_delay==transmit_del;
                      })
        get_response(rsp);
        `uvm_info(get_type_name(),$sformtf("%s read: addr=`x%0h,data[0]= `x%0h",get_sequence_path(),rsp.addr,rsp.data[0]),UVM_HIGH);
    endtask
endclass
```

12. 用了 sequence，就该写 sequencer 了；sequencer 是整个环境中最简单的部分；

```
class ubus_master_sequencer extends uvm_sequencer #(ubus_transter);
    `uvm_component_utils(ubus_master_sequencer)

    function new(string name, uvm_component parent);
        super.new(name,parent);
    endfunction

endclass
```

13. 有了 sequencer，就可以定义 driver 了，定义好 driver，然后需要在 agent 中进行例化；在 driver 中一定要 get interface，因为 driver 直接和 DUT 做交互;

```
class ubus_master_driver extends uvm_driver #(ubus_transfer);

    protected virtual ubus_if vif;
    protected int master_id;

    `uvm_component_utils_begin(ubus_master_driver)
        `uvm_field_int(master_id,UVM_DEFAULT)
    `uvm_component_utils_end

    function new(string name, uvm_component parent);
        super.new(name,parent);
    endfunction

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        if(!uvm_config_db#(virtual ubus_if)::get(this,"","vif",vif))
            `uvm_fatal("NO vif",{"virtual interface must be set for: ",get_full_name(),".vif"});
    endfunction
```

14. 注意在 driver 中如何写 get sequence：

```
    virtual protected task get_and_drive();
        @(negedge vif.sig_reset);
        forever begin
            @(posedge vif.sig_clock);
            seq_item_port.get_next_item(req);
            $cast(rsp,req.clone());
            rsp.set_id_info(req);
            drive_transfer(rsp);
            seq_item_port.item_done();
            seq_item_port.put_response(rsp);
        end
    endtask
```

15. 有了 sequencer，driver，实际上就可以定义 master agent 了，在 agent 中要实现 dirver 和 sequencer 的连接

```
class ubus_master_agent extends uvm_agent;
    protected int master_id;

    ubus_master_driver driver;
    uvm_sequencer#(bus_transfer) sequencer;

    `uvm_component_utils_begin(ubus_master_agent)
        `uvm_field_int(master_id,UVM_DEFAULT)
    `uvm_component_utils_end

    function new(string name, uvm_component parent);
        super.new(name,parent);
    endfunction

    function build_phase(uvm_phase phase);
        super.build_phase(phase);

        if(get_is_active()==UVM_ACTIVE) begin
            squencer=uvm_sequencer#(ubus_transfer)::type_id::create("sequencer",this);
            driver=ubus_master_driver::type_id::create("driver",this);
        end
    endfunction

    function void connect_phase(uvm_phase phase);
        if(get_is_active()==UVM_ACTIVE) begin
            driver.seq_item_port.connect(sequencer.seq_item_export);
        end
    endfunction
endclass
```

16. Agent 实现了之后，就需要在 env 中例化

```
class ubus_env extends uvm_env;
    protected virtual interface ubus_if vif;
    //`uvm_component_utils(ubus_env)
    protected int num_masters=0;

    ubus_master_agent masters[];

    `uvm_component_utils_begin(ubus_env)
        `uvm_field_int(num_masters,UVM_DEFAULT)
    `uvm_component_utils_end

    function new(string name, uvm_component parent);
        super.new(name,parent);
    endfunction

    function void build_phase(uvm_phase phase);
        string inst_name;
        super.build_phase(phase);
        if(!uvm_config_db#(virtual ubus_if)::get(this,"","vif",vif))
            `uvm_fatal("NO vif",{"virtual interface must be set for:",get_full_name(),".vif"})

        void'(uvm_config_db#(int)::get(this,"","num_masters",num_masters));

        masters=new[num_masters];

        for(int i=0; i<num_masters;i++) begin
            $sformat(inst_name,"masters[%0d]",i);
            masters[i]=ubus_master_agent::type_id::create(inst_name,this);
            void'(uvm_config_db#(int)::set(this,{inst_name,".driver"},"master_id",i));
        end
    endfunction
endclass
```

17. 然后更改 test_lib.sv，写一个从 base_test 派生出来的 test，设置默认的 sequence。和 master 的数量；

```
class test_read_byte extends ubus_base_test;

    `uvm_component_utils(test_read_byte)

     function new(string name="test_read_byte",uvm_component parent=null);
        super.new(name,parent);
     endfunction

    virtual function void build_phase(uvm_phase phase);
        begin
            uvm_config_db#(int)::set(this,"env","num_masters",1);
            uvm_config_db#(uvm_object_wrapper)::set(this,"env.masters[0].sequencer.run_
phase","default_sequence",read_byte_seq::type_id::get());
            super.build_phase(phase);
        end
    endfunction
endclass
```

18. 然后在 ubus_pkg.sv 中 include 各个 component 的文件，然后编译，修改一些语法错误，但是一直会出现如下错误：make test_read_byte

```
irun -access rw -uvmhome /home/dengf/uvm-1.1 +UVM_VERBOSITY=UVM_LOW -quiet +define+UVM_OBJECT_
MUST_HAVE_CONSTRUCTOR -incdir . ubus_tb_top.sv +UVM_TESTNAME=test_read_byte
            uvm_config_db#(uvm_object_wrapper)::set(this,"env.masters[0].sequencer.run_phase",
"default_sequence",read_byte_seq::type_id::get());

                                |
ncvlog: *E,NOPBIND (test_lib.sv,36|125): Package read_byte_seq could not be bound.
            uvm_config_db#(uvm_object_wrapper)::set(this,"env.masters[0].sequencer.run_phase",
"default_sequence",read_byte_seq::type_id::get());

                                          |
ncvlog: *E,NOTFXX (test_lib.sv,36|139): Expecting a function name [10.3.3(IEEE)].
irun: *E,VLGERR: An error occurred during parsing.  Review the log file for errors with the co
de *E and fix those identified problems to proceed.  Exiting with code (status 1).
make: *** [read_byte] Error 1
```

19. 错误的意思是说这个 sequence 不能够 get 到，在屏蔽掉设置默认的 sequence 后，编译可以正确的通过，但是此时平台中没有数据的流动，需要 debug；

```
25 class test_read_byte extends ubus_base_test;
26
27     `uvm_component_utils(test_read_byte)
28
29      function new(string name="test_read_byte",uvm_component parent=null);
30         super.new(name,parent);
31      endfunction
32
33     virtual function void build_phase(uvm_phase phase);
34         begin
35             uvm_config_db#(int)::set(this,"env","num_masters",1);
36            // uvm_config_db#(uvm_object_wrapper)::set(this,"env.masters[0].sequencer.ru
   n_phase","default_sequence",read_byte_seq::type_id::get());
37             super.build_phase(phase);
38         end
39     endfunction
40 endclass
```

20. 打开上面的屏蔽，到 read_byte_seq 中去，将 task body()屏蔽掉也可以 pass

```
31     `uvm_object_utils(read_byte_seq)
32
33     rand bit[15:0] start_addr;
34     rand int unsigned transmit_del=0;
35     constraint transmid_del_ct{(transmit_del<=10);}
36
37     virtual task body();
38        uvm_report_info(get_full_name(),"sequence is okay now,please FIXME.\n",UVM_NONE);
39       //`uvm_do_with(req,
40       //                {req.addr==start_addr;
41       //                req.read_write==READ;
42       //                req.size==1;
43       //                req.error_pos== 1000;
44       //                req.transmit_delay==transmit_del;
45       //            })
46       //get_response(rsp);
47       //`uvm_info(get_type_name(),$sformtf("%s read: addr=`x%0h,data[0]= `x%0h",get_sequence_path(),rsp.addr,
   rsp.data[0]),UVM_HIGH);
48     endtask
49 endclass
```

21. 到目前为止，该平台中有 ENV，agent，driver，sequencer，sequence，transaction 了，可以进行最基本的 test 操作了，但是还需要 monitor，scoreboard 等等；

    总结一：driver（sequencer 也是）是和发送的 sequence 类型有关的，也就是一个 driver 要发送的 sequence 的类型是固定死了的，那么我们需要将多种类型的 sequence 通过一个 driver 来发送要怎么办？其实将多种类型的 sequence 不要做成 sequence，做成 uvm_object，然后在一个大的 uvm_object 中 random 这些小的 uvm_object（就是小的 sequence），然后用这个最大的 uvm_object 来做 sequence_item 就对了。

    总结二：sequence_item 中要具备哪些内容呢？严格的说是你当前 module 所要用到的所用 data flow 的数据类型。

    总结三：别忘记在 driver 中 get interface。

[1-3] 如何顺序的写 UVM 平台（3）-Master Monitor

22. 编写 master_monitor；该 component 的主要作用是收集 driver 发出的各种数据类型的 coverage。然后需要在 agent 中例化

```
42      `uvm_component_utils_begin(ubus_master_monitor)
43          `uvm_field_int(master_id,UVM_DEFAULT)
44          `uvm_field_int(checks_enable,UVM_DEFAULT)
45          `uvm_field_int(coverage_enable,UVM_DEFAULT)
46      `uvm_component_utils_end
47
48      function new(string name, uvm_component parent);
49          super.new(name,parent);
50          cov_trans=new();
51          cov_trans.set_inst_name({get_full_name(),".cov_trans"});
52          cov_trans_beat=new();
53          cov_trans_beat.set_inst_name({get_full_name(),".cov_trans_beat"});
54          trans_collected=new();
55          item_collected_port=new("item_collected_port",this);
56      endfunction
57
58      function void build_phase(uvm_phase phase);
59          super.build_phase(phase);
60          if(!uvm_config_db#(virtual ubus_if)::get(this,"","vif",vif))
61              `uvm_fatal("NO vif",{"virtual interface must be set for : ",get_full_name(),".vif"});
62      endfunction
63
```

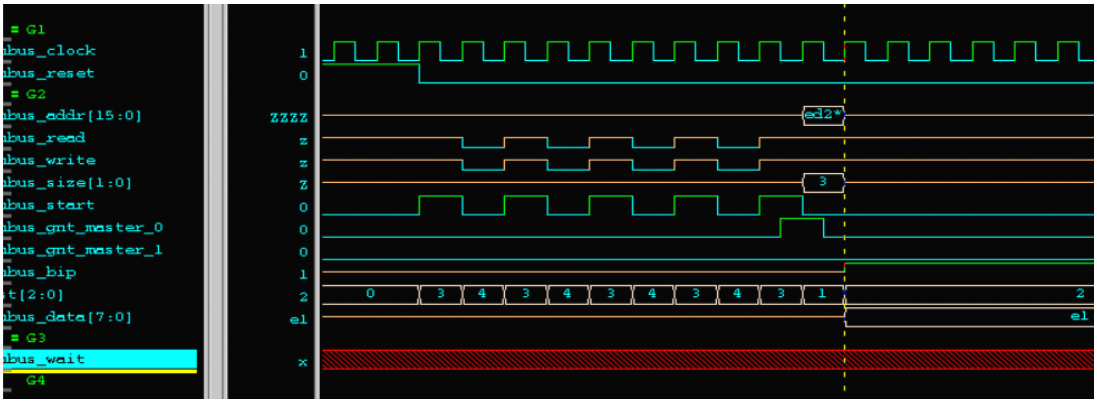
23. 在 agent 中对 master 的 monitor 进行例化之后，master 的部分就已经完成；

```
42      `uvm_component_utils_begin(ubus_master_monitor)
43          `uvm_field_int(master_id,UVM_DEFAULT)
44          `uvm_field_int(checks_enable,UVM_DEFAULT)
45          `uvm_field_int(coverage_enable,UVM_DEFAULT)
46      `uvm_component_utils_end
47
48      function new(string name, uvm_component parent);
49          super.new(name,parent);
50          cov_trans=new();
51          cov_trans.set_inst_name({get_full_name(),".cov_trans"});
52          cov_trans_beat=new();
53          cov_trans_beat.set_inst_name({get_full_name(),".cov_trans_beat"});
54          trans_collected=new();
55          item_collected_port=new("item_collected_port",this);
56      endfunction
57
58      function void build_phase(uvm_phase phase);
59          super.build_phase(phase);
60          if(!uvm_config_db#(virtual ubus_if)::get(this,"","vif",vif))
61              `uvm_fatal("NO vif",{"virtual interface must be set for : ",get_full_name(),".vif"});
62      endfunction
63
```

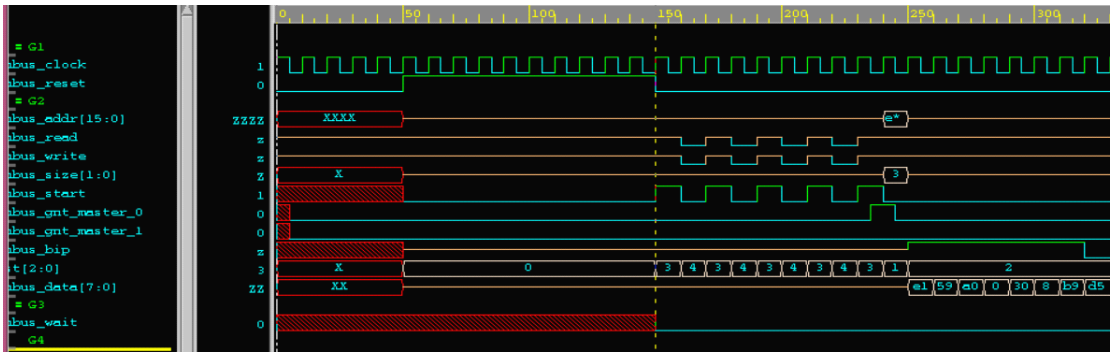
24. 到目前为止，平台能动，但是没有 sequence 在里面流动，所以时间是没有动的；需要在目前现有的资源下解决这个问题？这是因为在 sequence 中没有打开`uvm_do_*相关的函数；打开后时钟就开始动了，但是停止不下来，需要进一步 debug

```
37      virtual task body();
38         uvm_report_info(get_full_name(),"sequence is okay now,please FIXME.\n",UVM_NONE);
39         `uvm_do(req)
40        //`uvm_do_with(req,
41        //                  {req.addr==start_addr;
42        //                  req.read_write==READ;
43        //                  req.size==1;
44        //                  req.error_pos== 1000;
45        //                  req.transmit_delay==transmit_del;
46        //              })
47        //get_response(rsp);
48        //`uvm_info(get_type_name(),$sformtf("%s read: addr=`x%0h,data[0]= `x%0h",get_sequence_path(),rsp.addr,
   rsp.data[0]),UVM_HIGH);
49      endtask
50  endclass
```

25. 如何让平台停止下来呢？不能够停止是因为有一根信号没有驱动，导致一直等待



26. 在 top 中对该信号进行驱动后，该平台就能自动停止了



[1-4] 如何顺序的写 UVM 平台（4-1）-Slave Agent

27. 前面完成了该平台最基本的部分工作，这个时候这个平台会有一些动作了，但是还需要定义更多的东西，比如总线测试需要 master 和 slave，DMA 测试需要 slave model，DDR 控制器测试需要 slave model，SPI/UART/IIC/IIS/PCIE 等都需要另外的 model，我更倾向于 model 和验证方法学是独立开来的，这样在从一种验证方法学切换到另外一个验证方法学后不需要修改 model，比如像 micron 提供的 DDR model 和 NND model 都是独立于验证方法学的；当然对于 ubus 总线测试则无所谓，因为整个 DUT 外面的东西都是不固定的；当然，如果能让 slave 固定则最好。
28. Slave driver 的作用是返回 Response，那么这个 response 的内容是从 slave 的 sequencer 中来的，而 sequencer 的 item 则来自 slave sequence lib，所以需要首先写 slave 的 sequence；slave 有两种 sequence，第一种返回读写类型，第二种返回地址和数据等；
29. 先编写 sequence，在 package 中 include，编译可以通过；

```
class simple_response_seq extends uvm_sequence #(ubus_transfer);

    function new(string name="simple_response_seq");
        super.new(name);
    endfunction

    `uvm_object_utils(simple_response_seq)

    ubus_transfer util_transfer;

    virtual task body();
        `uvm_info(get_type_name(),$sformatf("%s starting...",get_sequence_path()),UVM_MEDIUM);
        forever begin
            uvm_test_done.raise_objection(this);
            `uvm_do_with(req,{req.read_write == util_transfer.read_write;
                req.size==util_transfer.size;req.error_pos==1000;})
            uvm_test_done.drop_objection(this);
        end
    endtask

endclass
```

30. 编写 slave_sequencer，在 package 中 include，注意，并没有地方例化它，编译可以通过

```
class ubus_slave_sequencer extends uvm_sequencer #(ubus_transfer);
    //TLM port to peek the address phase from the slave monitor

    uvm_blocking_peek_port#(ubus_transfer) addr_ph_port;

    `uvm_component_utils(ubus_slave_sequencer)

    function new(string name, uvm_component parent);
        super.new(name,parent);
        addr_ph_port =new("addr_ph_port",this);
    endfunction

endclass
```

31. 编写 slave_agent，并在 slave_agent 中例化 sequencer，在 package 中 include，注意，并没有地方例化这个 agent，编译可以通过

```
class ubus_slave_agent extends uvm_agent;
    ubus_slave_sequencer sequencer;

    `uvm_component_utils(ubus_slave_agent)

    function new(string name, uvm_component parent);
        super.new(name,parent);
    endfunction

    virtual function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        if(get_is_active()==UVM_ACTIVE) begin
            sequencer =ubus_slave_sequencer::type_id::create("sequencer",this);
        end
    endfunction

    function void connect_phase(uvm_phase phase);
        if(get_is_active()==UVM_ACTIVE) begin
        end
    endfunction
endclass
```

32. 然后需要做一系列动作，来设置 slave 的数量，并在 env 中例化 slave_agent；

```
        slaves=new[num_slaves];
        for(int i=0;i<num_slaves;i++)begin
            $sformat(inst_name,"slaves[%0d]",i);
            slaves[i]=ubus_slave_agent::type_id::create(inst_name,this);
        end
    endfunction
```

33. 出现在 slave 部分,目前只有 agent,sequencer,sequence,出现如下错误;

UVM_ERROR @ 0: uvm_test_top.ubus_example_tb0.ubus0.slaves[0].sequencer.addr_ph_port [Connection Error] connection count of 0 does not meet required minimum of 1

这是因为在 sequencer 中声明了一个 TLM port,但是一直没有连接它,所以需要屏蔽掉才可以;

```
 2 class ubus_slave_sequencer extends uvm_sequencer #(ubus_transfer);
 3     //TLM port to peek the address phase from the slave monitor
 4
 5     //uvm_blocking_peek_port#(ubus_transfer) addr_ph_port;
 6
 7     `uvm_component_utils(ubus_slave_sequencer)
 8
 9     function new(string name, uvm_component parent);
10         super.new(name,parent);
11         //addr_ph_port =new("addr_ph_port",this);
12     endfunction
13
14 endclass
```

34. 完成上面的修改后，可以得到下面的结果：

```
----------------------------------------------------------------------
Name                            Type                    Size  Value
----------------------------------------------------------------------
uvm_test_top                    test_read_byte          -     @238
  ubus_example_tb0              ubus_example_tb         -     @4859
    ubus0                       ubus_env                -     @4858
      masters[0]                ubus_master_agent       -     @4971
        driver                  ubus_master_driver      -     @5150
          rsp_port              uvm_analysis_port       -     @6593
            recording_detail    uvm_verbosity           32    UVM_FULL
          sqr_pull_port         uvm_seq_item_pull_port  -     @6519
            recording_detail    uvm_verbosity           32    UVM_FULL
          master_id             integral                32    'h0
          recording_detail      uvm_verbosity           32    UVM_FULL
        monitor                 ubus_master_monitor     -     @4974
          item_collected_port   uvm_analysis_port       -     @5313
            recording_detail    uvm_verbosity           32    UVM_FULL
          master_id             integral                32    'h0
          checks_enable         integral                1     'h1
          coverage_enable       integral                1     'h1
          recording_detail      uvm_verbosity           32    UVM_FULL
        sequencer               uvm_sequencer           -     @5218
          rsp_export            uvm_analysis_export     -     @5476
            recording_detail    uvm_verbosity           32    UVM_FULL
          seq_item_export       uvm_seq_item_pull_imp   -     @6372
            recording_detail    uvm_verbosity           32    UVM_FULL
          recording_detail      uvm_verbosity           32    UVM_FULL
          arbitration_queue     array                   0     -
          lock_queue            array                   0     -
          num_last_reqs         integral                32    'd1
          num_last_rsps         integral                32    'd1
        master_id               integral                32    'h0
        recording_detail        uvm_verbosity           32    UVM_FULL
```

```
    slaves[0]                ubus_slave_agent          -     @4972
      sequencer              ubus_slave_sequencer      -     @6364
        rsp_export           uvm_analysis_export       -     @6762
          recording_detail   uvm_verbosity             32    UVM_FULL
        seq_item_export      uvm_seq_item_pull_imp     -     @7658
          recording_detail   uvm_verbosity             32    UVM_FULL
        recording_detail     uvm_verbosity             32    UVM_FULL
        arbitration_queue    array                     0     -
        lock_queue           array                     0     -
        num_last_reqs        integral                  32    'd1
        num_last_rsps        integral                  32    'd1
      recording_detail       uvm_verbosity             32    UVM_FULL
    num_masters              integral                  32    'h1
    num_slaves               integral                  32    'h1
    recording_detail         uvm_verbosity             32    UVM_FULL
  recording_detail           uvm_verbosity             32    UVM_FULL
----------------------------------------------------------------------
```

```
----------------------------------------------------------------------
Name                         Type                    Size  Value
----------------------------------------------------------------------
ubus_transfer_inst           ubus_transfer           -     @5284
  addr                       integral                16    'h2c7
  read_write                 ubus_read_write_enum    32    READ
  size                       integral                32    'h1
  data                       da(integral)            1     -
    [0]                      integral                8     'h0
  wait_state                 da(integral)            0     -
  error_pos                  integral                32    'h0
  transmit_delay             integral                32    'h0
  master                     string                  10    masters[0]
  slave                      string                  0     ""
  begin_time                 time                    64    260
  end_time                   time                    64    280
----------------------------------------------------------------------
```

[1-5] 如何顺序的写 UVM 平台（4-2）-Slave Driver

35. Slave driver 的作用是针对 master 的读写访问返回数据或者响应；当然还要驱动 slave 方面该驱动的信号，首先需要获得 Virtual interface。然后获得 slave_sequence_lib 中的 sequence 并驱动到总线上；

```
10      function void build_phase(uvm_phase phase);
11          if(!uvm_config_db#(virtual ubus_if)::get(this,"","vif",vif))
12              `uvm_fatal("NOVIF",{"virtual interface must be set for :",get_full_name(),".vif"});
13      endfunction
14
15      virtual task run_phase(uvm_phase phase);
16          fork
17              get_and_drive();
18              reset_signals();
19          join
20      endtask
21
22      //get_and_drive
23
24      virtual protected task get_and_drive();
25          @(negedge vif.sig_reset);
26          forever begin
27              @(posedge vif.sig_clock);
28              seq_item_port.get_next_item(req);
29              respond_to_transfer(req);
30              seq_item_port.item_done();
31          end
32      endtask
33
34      virtual protected task reset_signals();
35          forever begin
36             @(posedge vif.sig_reset);
37              vif.sig_error <=1'bz;
38              vif.sig_wait <=1'bz;
39              vif.rw      <=1'b0;
40          end
41      endtask
```

36. 驱动总线这个地方还是有点不明白？resp 获得到写的数据后会存放起来吗？待解决，这个确实会存放起来，在 ubus_slave_req_lib.sv 中的 slave_memory_seq 中的 post_do 函数中可以看到。

```
43      virtual protected task respond_to_transfer(ubus_transfer resp);
44          if(resp.read_write !=NOP) begin
45              vif.sig_error<=1'b0;
46              for(int i=0;i<resp.size;i++) begin
47                  case(resp.read_write)
48                      READ:begin
49                          vif.rw<=1'b1;
50                          vif.sig_data_out<=resp.data[i];
51                      end
52                      WRITE:begin
53                      end
54                  endcase
55                  if(resp.wait_state[i]>0) begin
56                      vif.sig_wait<=1'b1;
57                      repeat(resp.wait_state[i])
58                          @(posedge vif.sig_clock);
59                  end
60                  vif.sig_wait<=1'b0;
61
62                  @(posedge vif.sig_clock);
63                  resp.data[i] =vif.sig_data;
64              end
65
66              vif.rw <=1'b0;
67              vif.sig_wait <=1'bz;
68              vif.sig_error <=1'bz;
69          end
70      endtask
```

37. Slave_driver 写完后，就需要在 slave_agent 中例化它，并将 slave_driver 和 slave_sequencer 连接起来；在 package 中 include ubus_slave_driver.sv；此时 slave 并不会发送任何数据

出来，因为没有在 test 中设置 default sequence；

```
class ubus_slave_agent extends uvm_agent;
    ubus_slave_sequencer sequencer;
    ubus_slave_driver    driver;
    `uvm_component_utils(ubus_slave_agent)

    function new(string name, uvm_component parent);
        super.new(name,parent);
    endfunction

    virtual function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        if(get_is_active()==UVM_ACTIVE) begin
            sequencer =ubus_slave_sequencer::type_id::create("sequencer",this);
            driver =ubus_slave_driver::type_id::create("driver",this);
        end
    endfunction

    function void connect_phase(uvm_phase phase);
        if(get_is_active()==UVM_ACTIVE) begin
            driver.seq_item_port.connect(sequencer.seq_item_export);
        end
    endfunction
endclass
```

38. 我们需要让 slave 动起来，因为我们的 case 是读操作，那么就需要 slave 提供需要，所以需要在 slave_sequence_lib.sv 中加入一个 memory 的 sequence；在 pre_do 中将要读取的数据准备好，在 post_do 中将写入的数据存起来；

```
    virtual task pre_do(bit is_item);
        req.size =util_transfer.size;
        req.addr =util_transfer.addr;
        req.read_write =util_transfer.read_write;
        req.error_pos=1000;
        req.transmit_delay=0;
        req.data=new[util_transfer.size];
        req.wait_state=new[util_transfer.size];

        for(int unsigned i=0; i< util_transfer.size;i++)begin
            req.wait_state[i]=2;
            if(req.read_write==READ) begin
                if(!m_mem.exists(util_transfer.addr+i)) begin
                    m_mem[util_transfer.addr+i] =$urandom;
                end
                req.data[i]=m_mem[util_transfer.addr +i];
            end
        end
    endtask

    function void post_do(uvm_sequence_item this_item);
        if(util_transfer.read_write ==WRITE) begin
            for(int unsigned i=0 ;i<req.size; i++) begin
                m_mem[req.addr+i] =req.data[i];
            end
        end
    endfunction
```

39. 在 task body 中将 sequencer 中的 trans peek 出来，不知道有什么用处，但是这个地方得到的 req 又是通过 sequencer 发送给 driver 的，需要更多的理解；

```
65      virtual task body();
66          `uvm_info(get_type_name(),$sformatf("%s starting ...",get_sequence_path()),UVM_MEDIUM);
67          $cast(req,create_item(ubus_transfer::get_type(),p_sequencer, "req"));
68
69          forever begin
70              p_sequencer.addr_ph_port.peek(util_transfer);
71              starting_phase.raise_objection(this);
72              start_item(req);
73              finish_item(req);
74              starting_phase.drop_objection(this);
75          end
76      endtask
77
```

40. 既然有了读写返回的 sequence，就可以再 test 中设置 slave 的默认 sequence 了；

```
                read_modify_write_seq::type_id::get());
        uvm_config_db#(uvm_object_wrapper)::set(this,
                "ubus_example_tb0.ubus0.slaves[0].sequencer.run_phase",
                        "default_sequence",
                    slave_memory_seq::type_id::get());
```

设置后出现如下错误：

ncvlog: *E,NOIPRT (ubus_slave_seq_lib.sv,31|49): Unrecognized declaration 'ubus_slave_sequencer' could be a spelling mistake [SystemVerilog].

(`define macro: uvm_declare_p_sequencer [/home/dengf/uvm-1.1/src/macros/uvm_sequence_defines.svh line 446], `include file: ubus_slave_seq_lib.sv line 31, `include file: ubus_pkg.sv line 12, file: ubus_tb_top.sv line 5)

```
1  package ubus_pkg;
2      import uvm_pkg::*;
3      `include "uvm_macros.svh"
4      typedef uvm_config_db#(virtual ubus_if) ubus_vif_config;
5      typedef virtual ubus_if ubus_vif;
6      `include "ubus_transfer.sv"
7      `include "ubus_master_seq_lib.sv"
8      `include "ubus_master_sequencer.sv"
9      `include "ubus_master_driver.sv"
10     `include "ubus_master_monitor.sv"
11     `include "ubus_master_agent.sv"
12     `include "ubus_slave_sequencer.sv"
13     `include "ubus_slave_seq_lib.sv"
14     `include "ubus_slave_driver.sv"
15     `include "ubus_slave_agent.sv"
16     `include "ubus_env.sv"
17 endpackage
```

在 package 中将 sequencer 的顺序调整到 seq_lib 的前面即可。

41. 另外 seq 中用到了 sequencer 中的 addr_ph_port，所以需要将原来的屏蔽打开，并且需要连接 addr_ph_port 才可以；但是目前的 monitor 还没写好，所以先屏蔽掉所有用到这个 port 的地方吧，屏蔽后编译通过，但是出现如下错误：

UVM_INFO @ 0: uvm_test_top.ubus_example_tb0.ubus0.masters[0].sequencer@@read_byte_seq [uvm_test_top.ubus_example_tb0.ubus0.masters[0].sequencer.read_byte_seq] sequence is okay

now,please FIXME.

UVM_INFO @ 150: uvm_test_top.ubus_example_tb0.ubus0.masters[0].driver [uvm_test_top.ubus_example_tb0.ubus0.masters[0].driver] enter get_and_drive.

Error! NULL pointer dereference

File: ./ubus_slave_seq_lib.sv, line = 38, pos = 30

Scope: worklib.ubus_pkg::slave_memory_seq@8242_5.pre_do

Time: 160 NS + 2

问题的原因就是在 sequence 中没有 port 来 get 到现有的 sequence，所以需要重新例化一个，写一个 util_transfer 的 new 函数即可；

```
virtual task body();
    `uvm_info(get_type_name(),$sformatf("%s starting ...",get_sequence_path()),UVM_MEDIUM);
    $cast(req,create_item(ubus_transfer::get_type(),p_sequencer, "req"));

    forever begin
        //p_sequencer.addr_ph_port.peek(util_transfer);
        util_transfer=new();
        starting_phase.raise_objection(this);
        start_item(req);
        finish_item(req);
        starting_phase.drop_objection(this);
    end
endtask
```

42. 后面，testcase 不能结束，问题的原因需要 debug；问题的原因是在 slave_sequence_lib.sv 中的 task body 中在 forever begin 。。。end 中调用了 starting_phase.raise_objection 和 starting_phase.drop_objection，将 forever 去掉后，能正确的结束，但是想想 slave 正常的工作吗？整个平台后来又是怎么停止的呢？拭目以待吧

```
64
65      virtual task body();
66          `uvm_info(get_type_name(),$sformatf("%s starting ...",get_sequence_path()),UVM_MEDIUM);
67          $cast(req,create_item(ubus_transfer::get_type(),p_sequencer, "req"));
68
69          begin
70              //p_sequencer.addr_ph_port.peek(util_transfer);
71              util_transfer=new();
72              starting_phase.raise_objection(this);
73              start_item(req);
74              finish_item(req);
75              starting_phase.drop_objection(this);
76          end
77      endtask
```

[1-6] 如何顺序的写 UVM 平台（4-3）-Slave Monitor

43. 个人觉得 slave monitor 和 master monitor 一个就足够了，我不知道为什么这个地方需要用两个，因为看起来都是对 coverage 做统计而已；另外，monitor 中有一个 peek 函数，这个函数是在 seq_lib 中，用来将 monitor 收到的 trans 送入到 seq_lib 中存储用的，采用的是 addr_ph_port 进行通信；

```
97    virtual protected task collect_transactions();
98        bit range_check;
99        forever begin
100         if(m_parent !=null)
101           trans_collected.slave =m_parent.get_name();
102         collect_address_phase();
103         range_check =check_addr_range();
104         if(range_check) begin
105           void'(this.begin_tr(trans_collected));
106           -> address_phase_grabbed;
107           collect_data_phase();
108           `uvm_info(get_type_name(),$sformatf("Transfer collected : \n%s",trans_collected.sprint()),UVM_FULL)
109           if(checks_enable)
110               perform_transfer_checks();
111           if(coverage_enable)
112               perform_transfer_coverage();
113           item_collected_port.write(trans_collected);
114         end
115       end
116   endtask
```

```
190       task peek(output ubus_transfer trans);
191             @address_phase_grabbed;
192             trans =trans_collected;
193       endtask
```

44. 然后需要将 ubus_slave_monitor.sv 在 package 中 include，并在 agent 中例化 slave_monitor，需要打开 sequencer 中屏蔽的 addr_ph_port，并在 seq_lib.sv 中采用 addr_ph_port 来 get transaction，而不是自己采用一个 new（）的构造函数；还要将 sequencer 的 addr_ph_port 和 monitor 的 addr_ph_imp 连接起来；将 slave_seq_lib 中的 slave_mem_seq 中的 task body 中的 forever 加上；

```
65    virtual task body();
66        `uvm_info(get_type_name(),$sformatf("%s starting ...",get_sequence_path()),UVM_MEDIUM);
67        $cast(req,create_item(ubus_transfer::get_type(),p_sequencer, "req"));
68
69        forever begin
70            p_sequencer.addr_ph_port.peek(util_transfer);
71            //util_transfer=new();
72            starting_phase.raise_objection(this);
73            start_item(req);
74            finish_item(req);
75            starting_phase.drop_objection(this);
76        end
77    endtask
```

这次加上 forever 后能停止，是因为在 peek 函数中会等待一个 event，如果没有等待 event，就不会 raise_objection。

```
class ubus_slave_agent extends uvm_agent;
    ubus_slave_sequencer sequencer;
    ubus_slave_driver    driver;
    ubus_slave_monitor   monitor;
    `uvm_component_utils(ubus_slave_agent)

    function new(string name, uvm_component parent);
        super.new(name,parent);
    endfunction

    virtual function void build_phase(uvm_phase phase);
        super.build_phase(phase);
            monitor =ubus_slave_monitor::type_id::create("monitor",this);
        if(get_is_active()==UVM_ACTIVE) begin
            sequencer =ubus_slave_sequencer::type_id::create("sequencer",this);
            driver =ubus_slave_driver::type_id::create("driver",this);
        end
    endfunction

    function void connect_phase(uvm_phase phase);
        if(get_is_active()==UVM_ACTIVE) begin
            driver.seq_item_port.connect(sequencer.seq_item_export);
            sequencer.addr_ph_port.connect(monitor.addr_ph_imp);
        end
    endfunction
endclass
```

到目前为止可以看到 master_monitor 和 slave_monitor 中都能出现 coverage 的统计结果了。

UVM_INFO ubus_master_monitor.sv(153) @ 300: uvm_test_top.ubus_example_tb0.ubus0.masters[0].monitor [uvm_test_top.ubus_example_tb0.ubus0.masters[0].monitor] Covergroup 'cov_trans' coverage: 15.000000

UVM_INFO ubus_slave_monitor.sv(196) @ 300: uvm_test_top.ubus_example_tb0.ubus0.slaves[0].monitor [uvm_test_top.ubus_example_tb0.ubus0.slaves[0].monitor] Covergroup 'cov_trans' coverage:15.000000

[1-7] 如何顺序的写 UVM 平台（5-1）-Scoreboard

45. 到目前为止，环境中最基本的 component 都有了，DUT 可以正常工作了，但是还需要一个 scoreboard 来判断 DUT 工作是否正确；首先 scoreboard 应该声明一个 TLP imp，用来接收 master_monitor 和 slave_monitor 传递过来的 collected_trans；

```
uvm_analysis_imp#(ubus_transfer, ubus_example_scoreboard) item_collected_export;

protected bit disable_scoreboard =0;
protected int num_writes =0;
protected int num_init_reads =0;
protected int num_uninit_reads =0;
int sbd_error =0;

protected int unsigned m_mem_expected[int unsigned];

`uvm_component_utils_begin(ubus_example_scoreboard)
    `uvm_field_int(disable_scoreboard, UVM_DEFAULT)
    `uvm_field_int(num_writes,UVM_DEFAULT|UVM_DEC)
    `uvm_field_int(num_init_reads,UVM_DEFAULT|UVM_DEC)
    `uvm_field_int(num_uninit_reads,UVM_DEFAULT|UVM_DEC)
`uvm_component_utils_end

function new(string name, uvm_component parent);
    super.new(name,parent);
endfunction

function void build_phase(uvm_phase phase);
    item_collected_export =new("item_collected_export",this);
endfunction

virtual function void write(ubus_transfer trans);
    if(!disable_scoreboard)
        memory_verify(trans);
endfunction
```

这个 write 函数是 uvm_analysis_port 调用的，默认就存在 write 函数，所以这个地方是在重载，即没有写这个函数的时候整个环境也是可以运行的；uvm_analysis_port 的作用就是让 slave_monitor 中调用 memory_verify 函数，在读的时候将收到的 transaction 中的 data（从 DUT 收到的 data）和存在于 mem 中的 data（送入 DUT 之前的 data）进行比较；（注意，master_monitor 中的 write 函数不是来自 scoreboard!! 那么来自哪儿呢）

注意：使用宏的时候中间不能换行；

```
33    protected function void memory_verify(intput ubus_transfer trans);
34        int unsigned data,exp;
35        for(int i=0;i<trans.size;i++) begin
36            if(m_mem_expected.exists(trans.addr+i)) begin
37                if(trans.read_write==READ) begin
38                    data=trans.data[i];
39                    `uvm_info(get_type_name(),$formatf("%s to existing address ...
40                      Checking address:%0h with data:%0h", trans.read_write.name(),trans.addr,data),UVM_LOW)
41                    assert(m_mem_expected[trans.addr +i ]==trans.data[i]) else begin
42                        exp=m_mem_expected[trans.addr +i];
43                        `uvm_error(get_type_name(),$sformatf("Read data mismacth,Expected:%0h. Actual: %0h",exp,data))
44                        sbd_error =1;
45                    end
46                    num_init_reads++;
47                end
48                if(trans.read_write==WRITE) begin
49                    data =trans.data[i];
50                    `uvm_info(get_type_name(),$sformatf("%s to existing address ...
51                      Updating address: %0h with data: %0h",trans.read_write.name(),trans.addr+i,data),UVM_LOW)
52                    m_mem_expected[trans.addr+i] =trans.data[i];
53                    num_writes++;
54                end
55            end else begin
56                data =trans.data[i];
57                `uvm_info(get_type_name(),$sformatf("%s to empty address ... Updating address :%0h with data :%0h",
58                  trans.read_write.name(),trans.addr+i,data),UVM_LOW)
59                 m_mem_expected[trans.addr+i]=trans.data[i];
60                if(trans.read_write==READ)
61                    num_uninit_reads++;
62                else if(trans.read_write==WRITE)
63                    num_writes++;
64            end
65        end
66    endfunction
```

46. Scoreboard 写完后，需要在 ubus_example_tb 中进行例化并连接 TLM 端口到 slave，并在 package 中 include；

```
class ubus_example_tb extends uvm_env;
    `uvm_component_utils(ubus_example_tb)

    ubus_env ubus0;

    ubus_example_scoreboard scoreboard0;

    function new(string name, uvm_component parent=null);
        super.new(name, parent);
    endfunction

    virtual function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        uvm_config_db#(int)::set(this,"ubus0","num_masters",1);
        uvm_config_db#(int)::set(this,"ubus0","num_slaves",1);
        ubus0=ubus_env::type_id::create("ubus0",this);
        scoreboard0 =ubus_example_scoreboard::type_id::create("scoreboard0",this);
    endfunction

    function void connect_phase(uvm_phase phase);
        ubus0.slaves[0].monitor.item_collected_port.connect(scoreboard0.item_collected_export);
    endfunction

endclass
```

47. 到目前为止，master，slave，scoreboard 就全了；

    对应文件：uvm_ubus_4.tar.gz

[1-8] 如何顺序的写 UVM 平台（5-2）-Bus Monitor

48. Bus Monitor 主要来决定是哪个 salve 对 master 进行响应；需要和 environment 配合使用；相当于 bus_monitor 起到了一个地址译码的作用；而具体 slave 的地址设置则是在 ubus_example_tb 中完成。
49. 首先在 bus_monitor 中提供地址设置的 class；

```
30 class slave_address_map_info extends uvm_object;
31
32   protected int min_addr;
33   protected int max_addr;
34
35   function new(string name = "slave_address_map_info");
36     super.new(name);
37   endfunction
38
39   `uvm_object_utils_begin(slave_address_map_info)
40     `uvm_field_int(min_addr, UVM_DEFAULT)
41     `uvm_field_int(max_addr, UVM_DEFAULT)
42   `uvm_object_utils_end
43
44   function void set_address_map (int min_addr, int max_addr);
45     this.min_addr = min_addr;
46     this.max_addr = max_addr;
47   endfunction : set_address_map
48
49   // get the min addr
50   function bit [15:0] get_min_addr();
51     return min_addr;
52   endfunction : get_min_addr
53
54   // get the max addr
55   function bit [15:0] get_max_addr();
56     return max_addr;
57   endfunction : get_max_addr
58
59 endclass : slave_address_map_info
```

50. 在 ubus_monitor 中用设置地址的 class 例化对象；并调用 slave_address_map_info 中的成员函数 set_address_map 来进行具体的地址设置；

```
185   // The following property is used to store slave address map
186   protected slave_address_map_info slave_addr_map[string];
187   // set_slave_configs
188   function void set_slave_configs(string slave_name,
189     int min_addr, int max_addr);
190     slave_addr_map[slave_name] = new();
191     slave_addr_map[slave_name].set_address_map(min_addr, max_addr);
192   endfunction : set_slave_configs
```

51. 在 ubus_slave_driver 在进行响应返回的时候，ubus_monitor 会根据当前 slave 的地址空间来决定是哪个 slave 在返回响应；

```
303   // check_which_slave
304   function void check_which_slave();
305     string slave_name;
306     bit slave_found;
307     slave_found = 1'b0;
308     if(slave_addr_map.first(slave_name))
309       do begin
310         if (slave_addr_map[slave_name].get_min_addr() <= trans_collected.addr
311           && trans_collected.addr <= slave_addr_map[slave_name].get_max_addr())
312         begin
313           trans_collected.slave = slave_name;
314           slave_found = 1'b1;
315         end
316         if (slave_found == 1'b1)
317           break;
318       end
319     while (slave_addr_map.next(slave_name));
320       assert(slave_found) else begin
321         `uvm_error(get_type_name(),
322           $sformatf("Master attempted a transfer at illegal address 16'h%0h",
323           trans_collected.addr))
324       end
325   endfunction : check_which_slave
```

52. Ubus_bus_monitor 只是提供了地址空间的一些函数，具体的调用则是在 ubus_env 中完成的，因为 ubus_bus_monitor 是在 env 中例化的；需要在 env 中例化 ubus_monitor

```
if(has_bus_monitor == 1) begin
    bus_monitor = ubus_bus_monitor::type_id::create("bus_monitor", this);
end
```

```
 95   // set_slave_address_map
 96   function void set_slave_address_map(string slave_name,
 97     int min_addr, int max_addr);
 98     ubus_slave_monitor tmp_slave_monitor;
 99     if( bus_monitor != null ) begin
100       // Set slave address map for bus monitor
101       bus_monitor.set_slave_configs(slave_name, min_addr, max_addr);
102     end
103     // Set slave address map for slave monitor
104     $cast(tmp_slave_monitor, lookup({slave_name, ".monitor"}));
105     tmp_slave_monitor.set_addr_range(min_addr, max_addr);
106   endfunction : set_slave_address_map
```

53. ubus_example_tb 来完成 slave 地址设置；

```
62   function void connect_phase(uvm_phase phase);
63     // Connect slave0 monitor to scoreboard
64     ubus0.slaves[0].monitor.item_collected_port.connect(
65       scoreboard0.item_collected_export);
66   endfunction : connect_phase
67
68   function void end_of_elaboration_phase(uvm_phase phase);
69     // Set up slave address map for ubus0 (basic default)
70     ubus0.set_slave_address_map("slaves[0]", 0, 16'hffff);
71   endfunction : end_of_elaboration_phase
```

54. 此时整个环境已经完成，只需要不断的扩展 sequence 和扩展 test_case 即可；

[1-9] 如何顺序的写 UVM 平台（6）-Sequence

55. Sequence 是整个平台中的血液，有了 sequence 才能够组合出更多的 testcase；所以的 sequence 的实例化和随机化都是在 task body 中进行的，而平台的开始和停止则是在 sequence 的 task pre_body 和 task post_body 中进行的。

```
31 virtual class ubus_base_sequence extends uvm_sequence #(ubus_transfer);
32
33   function new(string name="ubus_base_seq");
34     super.new(name);
35   endfunction
36
37   // Raise in pre_body so the objection is only raised for root sequences.
38   // There is no need to raise for sub-sequences since the root sequence
39   // will encapsulate the sub-sequence.
40   virtual task pre_body();
41     if (starting_phase!=null) begin
42       `uvm_info(get_type_name(),
43         $sformatf("%s pre_body() raising %s objection",
44               get_sequence_path(),
45               starting_phase.get_name()), UVM_MEDIUM);
46       starting_phase.raise_objection(this);
47     end
48   endtask
49
50   // Drop the objection in the post_body so the objection is removed when
51   // the root sequence is complete.
52   virtual task post_body();
53     if (starting_phase!=null) begin
54       `uvm_info(get_type_name(),
55         $sformatf("%s post_body() dropping %s objection",
56               get_sequence_path(),
57               starting_phase.get_name()), UVM_MEDIUM);
58     starting_phase.drop_objection(this);
59     end
60   endtask
61
62 endclass : ubus_base_sequence
```

56. 派生出最基本的 sequence，这种最基本的 sequence 不同的就是约束不一样，其他的几乎一致；我一致不太理解的就是这个地方的 get_response 得到的是什么?从什么地方传递过来的呢？这个应该是在 master_driver 的函数中执行的。

```
27     virtual protected task get_and_drive();
28         @(negedge vif.sig_reset);
29         uvm_report_info(get_full_name(),"enter get_and_drive.\n",UVM_MEDIUM);
30         forever begin
31             @(posedge vif.sig_clock);
32             seq_item_port.get_next_item(req);
33             $cast(rsp,req.clone());
34             rsp.set_id_info(req);
35             drive_transfer(rsp);
36             seq_item_port.item_done();
37             seq_item_port.put_response(rsp);
38         end
39     endtask
```

```
70 class read_byte_seq extends ubus_base_sequence;
71
72   function new(string name="read_byte_seq");
73     super.new(name);
74   endfunction
75
76   `uvm_object_utils(read_byte_seq)
77
78   rand bit [15:0] start_addr;
79   rand int unsigned transmit_del = 0;
80   constraint transmit_del_ct { (transmit_del <= 10); }
81
82   virtual task body();
83     `uvm_do_with(req,
84       { req.addr == start_addr;
85         req.read_write == READ;
86         req.size == 1;
87         req.error_pos == 1000;
88         req.transmit_delay == transmit_del; } )
89     get_response(rsp);
90     `uvm_info(get_type_name(),
91       $sformatf("%s read : addr = `x%0h, data[0] = `x%0h",
92       get_sequence_path(), rsp.addr, rsp.data[0]),
93       UVM_HIGH);
94   endtask
95
96 endclass : read_byte_seq
```

57. 在 sequence 中使用`uvm_do*的时候，默认的例化的 sequence 的 name 为 req，这个 req 是可以再`uvm_do*之外访问的，但是想要知道这个 req 的详细信息，只能去 driver 那边打印，因为`uvm_do*中包含一系列操作，只有等到 driver 那边完成后才会接着执行`uvm_do*后面的程序；
58. 如何用简单的 sequence 组合出复杂的 sequence 呢？uvm_do 系统宏可以直接接受一个 sequence 的变量作为参数；下面就是将 2 个 read_byte_seq 和一个 write_byte_seq 进行合并成一个复杂的读-写-读的 sequence；利用这种原理可以进行错误注入等其他操作。

```
170 //------------------------------------------------------------------------------
171 //
172 // SEQUENCE: read_modify_write_seq
173 //
174 //------------------------------------------------------------------------------
175
176 class read_modify_write_seq extends ubus_base_sequence;
177
178   function new(string name="read_modify_write_seq");
179     super.new(name);
180   endfunction : new
181
182   `uvm_object_utils(read_modify_write_seq)
183
184   read_byte_seq read_byte_seq0;
185   write_byte_seq write_byte_seq0;
186
187   rand bit [15:0] addr_check;
188   bit [7:0] m_data0_check;
189
190   virtual task body();
191     `uvm_info(get_type_name(),
192       $sformatf("%s starting...",
193       get_sequence_path()), UVM_MEDIUM);
194     // READ A RANDOM LOCATION
195     `uvm_do_with(read_byte_seq0, {read_byte_seq0.transmit_del == 0; })
196     addr_check = read_byte_seq0.rsp.addr;
197     m_data0_check = read_byte_seq0.rsp.data[0] + 1;
198     // WRITE MODIFIED READ DATA
199     `uvm_do_with(write_byte_seq0,
200       { write_byte_seq0.start_addr == addr_check;
201         write_byte_seq0.data0 == m_data0_check; } )
202     // READ MODIFIED WRITE DATA
203     `uvm_do_with(read_byte_seq0,
204       { read_byte_seq0.start_addr == addr_check; } )
205     assert(m_data0_check == read_byte_seq0.rsp.data[0]) else
206       `uvm_error(get_type_name(),
207         $sformatf("%s Read Modify Write Read error!\n\tADDR: %h, EXP: %h, ACT: %h",
208         get_sequence_path(),addr_check,m_data0_check,read_byte_seq0.rsp.data[0]));
209   endtask : body
210
211 endclass : read_modify_write_seq
```

[1-10] 如何顺序的写 UVM 平台（7）-ubus_example_tb

我认为这个 example_tb 并不是那么必要，毕竟我们在前面不用这个也是可以做一切事情的，那么这个存在的意义呢？没有这个的时候直接在 test 中例化 ENV 和 scoreboard 即可，有这个后，在 test 中直接例化 example_tb 即可，也许这个 example_tb 主要是用于向上 integrate 的作用吧。

相当于场景 tb1 有 4 个 master 4 个 slave

tb2 只有一个 master 2 个 slave

这些都是你自己去呼叫的

每个场景下 又分不同的 pattern

刚开始调试的时候，只有 master，没有 monitor 和 slave 和 scoreboard；

```
1
2  class ubus_example_tb extends uvm_env;
3      `uvm_component_utils(ubus_example_tb)
4
5      ubus_env ubus0;
6
7      function new(string name, uvm_component parent=null);
8          super.new(name, parent);
9      endfunction
10
11     virtual function void build_phase(uvm_phase phase);
12         super.build_phase(phase);
13         uvm_config_db#(int)::set(this,"ubus0","num_master",1);
14         ubus0=ubus_env::type_id::create("ubus0",this);
15     endfunction
16
17 endclass
```

由于在 test 和 ENV 中间加入了 ubus_example_tb，所以需要修改 test_lib.sv

```
2  class ubus_base_test extends uvm_test;
3      `uvm_component_utils(ubus_base_test)
4      ubus_example_tb ubus_example_tb0;
5      uvm_table_printer printer;
6
7      function new(string name="ubus_base_test",uvm_component parent=null);
8          super.new(name,parent);
9      endfunction
10
11     virtual function void build_phase(uvm_phase phase);
12         super.build_phase(phase);
13         //enable transaction recording for everything
14         uvm_config_db#(int)::set(this,"*","recording_detail",UVM_FULL);
15         ubus_example_tb0=ubus_example_tb::type_id::create("ubus_example_tb0",this);
16         printer =new();
17         printer.knobs.depth=3;
18     endfunction
19
20     task run_phase(uvm_phase phase);
21         super.run_phase(phase);
22         uvm_report_info(get_full_name(),"start of test run_phase...",UVM_LOW);
23         uvm_top.print_topology();
24     endtask
25 endclass
26
27 class test_read_byte extends ubus_base_test;
28
29     `uvm_component_utils(test_read_byte)
30
31      function new(string name="test_read_byte",uvm_component parent=null);
32         super.new(name,parent);
33      endfunction
34
35     virtual function void build_phase(uvm_phase phase);
36         begin
37             uvm_config_db#(int)::set(this,"env","num_masters",1);
38             uvm_config_db#(uvm_object_wrapper)::set(this,"ubus_example_tb0.ubus0.masters[0].sequencer.
   yte_seq::type_id::get());
```

运行结果为：

```
UVM_INFO @ 0: reporter [UVMTOP] UVM testbench topology:
-------------------------------------------------------
Name                    Type             Size  Value
-------------------------------------------------------
uvm_test_top            test_read_byte   -     @235
  ubus_example_tb0      ubus_example_tb  -     @4844
    ubus0               ubus_env         -     @4843
      num_masters       integral         32    'h0
      num_slaves        integral         32    'h0
      recording_detail  uvm_verbosity    32    UVM_FULL
    recording_detail    uvm_verbosity    32    UVM_FULL
-------------------------------------------------------
```

[1-11] 如何顺序的写 UVM 平台（8）-平台总结

[1-12] 如何顺序的写 UVM 平台（8-1）-top.sv

1. 只需要 include DUT, Interface，ubus_pkg；其他的都在 package 中 include

```
`include "dut_dummy.v"
`include "ubus_if.sv"
`include "ubus_pkg.sv"
```

2. 需要 import uvm_pkg，和自定义的 pkg；

```
import uvm_pkg::*;
import ubus_pkg::*;
```

虽然在 ubus_pkg 中 import 了 uvm_pkg，但是在这个地方还必须 import 一次，否则会出现如下错误：

```
uvm_config_db#(virtual ubus_if)::set(uvm_root::get(),"*","vif",vif);
                                  |
ncvlog: *E,NOPBIND (ubus_tb_top.sv,31|20): Package uvm_config_db could not be bound.
```

3. 设置接口，run_test

```
initial begin
  uvm_config_db#(virtual ubus_if)::set(uvm_root::get(), "*", "vif", vif);
  run_test();
end
```

4. 设置波形

```
initial begin
    $fsdbDumpfile("test.fsdb");
    $fsdbDumpvars(0,ubus_tb_top);
end
```

[1-13] 如何顺序的写 UVM 平台（8-2）-pkg.sv

5. ，有两个 typedef，本来是想在 get virtual interface 中用的，但是也没有用，所以就可以去掉了，注意的是，这个 include 是有顺序的，被调用的一定要放在前面。

```
package ubus_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    //typedef uvm_config_db#(virtual ubus_if) ubus_vif_config;
    //typedef virtual ubus_if ubus_vif;
    `include "ubus_transfer.sv"
    `include "ubus_master_seq_lib.sv"
    `include "ubus_master_sequencer.sv"
    `include "ubus_master_driver.sv"
    `include "ubus_master_monitor.sv"
    `include "ubus_master_agent.sv"
    `include "ubus_slave_sequencer.sv"
    `include "ubus_slave_seq_lib.sv"
    `include "ubus_slave_driver.sv"
    `include "ubus_slave_monitor.sv"
    `include "ubus_slave_agent.sv"
    `include "ubus_bus_monitor.sv"
    `include "ubus_env.sv"
    `include "ubus_example_scoreboard.sv"
    `include "ubus_example_tb.sv"
    `include "test_lib.sv"
endpackage
```

[1-14] 如何顺序的写 UVM 平台（8-3）-test_lib.sv

6. 首先定义一个 base_test，定义各种打印函数；

```
    Uvm_table_printer printer;
```

在 build_phase 中 new printer 和设置 print knobs

```
    Printer=new();
    Printer.knobs.depth=3;
```

在 end_of_elaboration_phase 中设置打印级别，和打印 test_topology

```
    // Set verbosity for the bus monitor for this demo
     if(ubus_example_tb0.ubus0.bus_monitor != null)
```

```
ubus_example_tb0.ubus0.bus_monitor.set_report_verbosity_level(UVM_FULL);
`uvm_info(get_type_name(),
  $sformatf("Printing the test topology :\n%s", this.sprint(printer)), UVM_LOW)
```

```
UVM_INFO test_lib.sv(25) @ 0: uvm_test_top [test_read_byte] Printfing the test topology :
 ----------------------------------------------------------------------
Name                          Type                      Size  Value
----------------------------------------------------------------------
uvm_test_top                  test_read_byte            -     @245
  ubus_example_tb0            ubus_example_tb           -     @4915
    scoreboard0               ubus_example_scoreboard   -     @5023
      item_collected_export   uvm_analysis_imp          -     @5172
      disable_scoreboard      integral                  1     'h0
      num_writes              integral                  32    'd0
      num_init_reads          integral                  32    'd0
      num_uninit_reads        integral                  32    'd0
      recording_detail        uvm_verbosity             32    UVM_FULL
    ubus0                     ubus_env                  -     @4914
      bus_monitor             ubus_bus_monitor          -     @4985
      masters[0]              ubus_master_agent         -     @5250
      slaves[0]               ubus_slave_agent          -     @5024
      num_masters             integral                  32    'h1
      num_slaves              integral                  32    'h1
      has_bus_monitor         integral                  1     'h1
      intf_checks_enable      integral                  1     'h1
      intf_coverage_enable    integral                  1     'h1
      recording_detail        uvm_verbosity             32    UVM_FULL
    recording_detail          uvm_verbosity             32    UVM_FULL
----------------------------------------------------------------------
```

用 uvm_top.print_topology()函数打印效果更好；

7. 在 run_phase 中设置 drain_time;

```
task run_phase(uvm_phase phase);
  //set a drain-time for the environment if desired
  phase.phase_done.set_drain_time(this, 50);
endtask : run_phase
```

8. 在 extract phase 中查看 test run 完后错误产生吗?

```
function void extract_phase(uvm_phase phase);
  if(ubus_example_tb0.scoreboard0.sbd_error)
    test_pass = 1'b0;
endfunction // void
```

9. 在 report phase 中打印 PASSED or FAILED 的 message;

```
function void report_phase(uvm_phase phase);
  if(test_pass) begin
    `uvm_info(get_type_name(), "** UVM TEST PASSED **", UVM_NONE)
  end
  else begin
    `uvm_error(get_type_name(), "** UVM TEST FAIL **")
  end
endfunction
```

10. 派生具体的 test，设置 master 和 slave 的数量

```
uvm_config_db#(int)::set(this,"ubus_example_tb0.ubus0",
                          "num_masters", 2);
uvm_config_db#(int)::set(this,"ubus_example_tb0.ubus0",
                          "num_slaves", 4);
```

11. 设置 test 的 default sequence

```
        // Control the number of RMW loops

uvm_config_db#(int)::set(this,"ubus_example_tb0.ubus0.masters[0].sequencer.loop_read_modify_write_seq", "itr", 6);

uvm_config_db#(int)::set(this,"ubus_example_tb0.ubus0.masters[1].sequencer.loop_read_modify_write_seq", "itr", 8);

            // Define the sequences to run in the run phase

uvm_config_db#(uvm_object_wrapper)::set(this,"*.ubus0.masters[0].sequencer.main_phase",
                                        "default_sequence",
                                        loop_read_modify_write_seq::type_id::get());
        lrmw_seq = loop_read_modify_write_seq::type_id::create();
        uvm_config_db#(uvm_sequence_base)::set(this,
                                        "ubus_example_tb0.ubus0.masters[1].sequencer.main_phase",
                                        "default_sequence",
                                        lrmw_seq);
    for(int i = 0; i < 4; i++) begin
        string slname;
        $swrite(slname,"ubus_example_tb0.ubus0.slaves[%0d].sequencer", i);
        uvm_config_db#(uvm_object_wrapper)::set(this, {slname,".run_phase"},
                                        "default_sequence",
                                        slave_memory_seq::type_id::get());
    end
```

12. 使用 sequence 机制之后，在不同的 case 中，把不同的 sequence 设置成 sequencer 的 main_phase（一定要是 main_phase 吗？为什么上面设置的是 run_phase 呢？）的 default_sequence，当 sequencer 执行到 main_phase 时，发现有 default_sequence，那么它就会把这个 sequence 启动起来。通过在 test_read_byte 中测试，在 main_phase 和在 run_phase 中设置 default_sequence 都是正确的；

[1-15] 如何顺序的写 UVM 平台（8-4）-transfer.sv

13. Transfer 中主要定义一个 driver 中所需要的数据类型，都是从 uvm_sequence_item 中来，还需要提供一些基本的约束；自定义枚举类型等；需要注意的是一个 transfer 只能对应一种类型的 driver，比如说有配置寄存器的 APB，还有驱动 DUT 的 master，则需要两种 transfer，两个 driver；

```
typedef enum { NOP,
               READ,
               WRITE
             } ubus_read_write_enum;
```

14. 约束定义

```
constraint c_read_write {
   read_write inside { READ, WRITE };
}
constraint c_size {
   size inside {1,2,4,8};
}
constraint c_data_wait_size {
   data.size() == size;
   wait_state.size() == size;
}
constraint c_transmit_delay {
   transmit_delay <= 10 ;
}
```

15. 工厂模式注册

```
`uvm_object_utils_begin(ubus_transfer)
   `uvm_field_int         (addr,                UVM_DEFAULT)
   `uvm_field_enum        (ubus_read_write_enum, read_write, UVM_DEFAULT)
   `uvm_field_int         (size,                UVM_DEFAULT)
   `uvm_field_array_int(data,                UVM_DEFAULT)
   `uvm_field_array_int(wait_state,        UVM_DEFAULT)
   `uvm_field_int         (error_pos,           UVM_DEFAULT)
   `uvm_field_int         (transmit_delay, UVM_DEFAULT)
   `uvm_field_string      (master,              UVM_DEFAULT|UVM_NOCOMPARE)
   `uvm_field_string      (slave,               UVM_DEFAULT|UVM_NOCOMPARE)
`uvm_object_utils_end
```

[1-16] 如何顺序的写 UVM 平台（8-5）-sequence.sv

16. 定义 base 的 sequence 都是从 uvm_sequence 派生而来，注意 transfer 是从 uvm_sequence_item 派生而来，这是由区别的，而 sequence 是依赖 transfer 类型的；另外我们会将 base_sequence 声明成虚基类，只有派生后才可以进行实例化；

```
virtual class ubus_base_sequence extends uvm_sequence #(ubus_transfer);
```

17. Sequence 是控制平台的启动和关闭，主要是在 task pre_body 和 task post_body 中控制；

```
virtual task pre_body();
  if (starting_phase!=null) begin
    starting_phase.raise_objection(this);
  end
endtask

// Drop the objection in the post_body so the objection is removed when
// the root sequence is complete.
virtual task post_body();
  if (starting_phase!=null) begin
    starting_phase.drop_objection(this);
  end
endtask
```

18. 然后就是通过 base_sequence 来产生具体的基本的 sequence；在使用`uvm_do*系列宏的时候，默认例化的 sequence 的名字为 req；这个 req 是可以在`uvm_do*之外使用；

```
`uvm_do_with(req,
  { req.addr == start_addr;
    req.read_write == READ;
    req.size == 1;
    req.error_pos == 1000;
    req.transmit_delay == transmit_del; } )
```

19. 接收返回类型，有两种方式，一种是直接采用 UVM 提供的 put_response 和 get_respone，另外一种则是在 sequencer 中定义一个 port 并采用 p_sequencer 的方式，先说第一种方式；

在`uvm_do 之后 sequencer 中等待接收 rsp，这个 rsp 是通过 driver 发送过来的；

```
get_response(rsp);
```

那么在 driver 中会出现如下代码:

```
seq_item_port.get_next_item(req);
$cast(rsp, req.clone());
rsp.set_id_info(req);
```

```
        drive_transfer(rsp);     //具体的 DUT 这个地方实现不同
        seq_item_port.item_done();
        seq_item_port.put_response(rsp);
```

20. 另外一种方式同 ubus_slave_seq_lib.sv 中的那样；

首先一定要在 sequencer 中定义一个 TLM port；

```
class ubus_slave_sequencer extends uvm_sequencer #(ubus_transfer);
  uvm_blocking_peek_port#(ubus_transfer) addr_ph_port;
  // Provide implementations of virtual methods such as get_type_name and create
  `uvm_component_utils(ubus_slave_sequencer)
  function new (string name, uvm_component parent);
    super.new(name, parent);
    addr_ph_port = new("addr_ph_port", this);
  endfunction : new
endclass : ubus_slave_sequencer
```

注意，这个地方要用到 p_sequencer 和 m_sequencer 的知识，简单的将，m_sequencer 指向的 uvm_sequencer_base 的指针， p_sequencer 指向的是 uvm_sequencer_base 派生对象 ubus_slave_sequencer 的指针；所以我们要使用 ubus_slave_sequencer 中的 port，就一定要用 p_sequencer 才可以。

```
  `uvm_declare_p_sequencer(ubus_slave_sequencer)
  ubus_transfer util_transfer;
  virtual task body();
      $cast(req, create_item(ubus_transfer::get_type(), p_sequencer, "req"));
      forever
      begin
        p_sequencer.addr_ph_port.peek(util_transfer);
        starting_phase.raise_objection(this);
        start_item(req);
        finish_item(req);
        starting_phase.drop_objection(this);
      end
    endtask : body
```

peek 函数当然是在于 ubus_slave_sequencer 的 component 中实现，本例子中是和 ubus_slave_monitor 中通信的，所以 peek 函数定义如下：

```
    uvm_blocking_peek_imp#(ubus_transfer,ubus_slave_monitor) addr_ph_imp;
    task peek(output ubus_transfer trans);
      @address_phase_grabbed;
      trans = trans_collected;
```

```
endtask : peek
```

当然还需要在 slave_agent 中将 port 之间进行连接

```
sequencer.addr_ph_port.connect(monitor.addr_ph_imp);
```

[1-17] 如何顺序的写 UVM 平台（8-6）-driver.sv

21. Driver，Monitor，Environment 都需要 get interface；virtual interface 的设置则是在 top 中完成的；

```
function void build_phase(uvm_phase phase);
    if(!uvm_config_db#(virtual ubus_if)::get(this, "", "vif", vif))
        `uvm_fatal("NOVIF",{"virtual interface must be set for: ",get_full_name(),".vif"});
endfunction: build_phase
```

22. 然后就是从 sequencer 中接收 sequence，发送到 interface 上，还需要对 sequence 返回响应，前面说了，返回响应有两种方式，见 sequence.sv

```
// run phase
virtual task run_phase(uvm_phase phase);
  fork
    get_and_drive();
    reset_signals();
  join
endtask : run_phase

// get_and_drive
virtual protected task get_and_drive();
  @(negedge vif.sig_reset);
  forever begin
    @(posedge vif.sig_clock);
    seq_item_port.get_next_item(req);
    $cast(rsp, req.clone());
    rsp.set_id_info(req);
    drive_transfer(rsp);
    seq_item_port.item_done();
    seq_item_port.put_response(rsp);
  end
endtask : get_and_drive
```

23. 注意，driver 一般也可以和 scoreboard 直接采用 TLM 接口直接通信，将 trans 发送到 scoreboard 中进行比较，但是在 ubus 中做的比较复杂，用了三个 monitor；

[1-18] 如何顺序的写 UVM 平台（8-7）-Monitor.sv

24. 同 driver 一样，monitor 的第一件事是 get interface；

```
function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  if(!uvm_config_db#(virtual ubus_if)::get(this, "", "vif", vif))
    `uvm_fatal("NOVIF",{"virtual interface must be set for: ",get_full_name(),".vif"});
endfunction: build_phase
```

25. 然后定义一个 TLP 接口，用于和 scoreboard 进行通信；和进行数据比较

```
uvm_analysis_port #(ubus_transfer) item_collected_port;
  item_collected_port.write(trans_collected);
```

在 ubus_example_tb.sv 中：

```
ubus0.slaves[0].monitor.item_collected_port.connect(scoreboard0.item_collected_export);
```

在 scoreboard 中实现 write 函数；在 scoreboard 中也有一个 memory 模型，将 slave_monitor 收到的写操作转化成保存数据或者是数据更新；将 slave_monitor 收到的读操作转化成数据比较操作或者数据更新；而数据的返回操作时有 slave_seq_lib 中的 slave_memory_seq 来完成的，注意着两个的区别；

```
// write
virtual function void write(ubus_transfer trans);
  if(!disable_scoreboard)
    memory_verify(trans);
endfunction : write
```

26. Monitor 的另外一个重要作用就是进行 coverage 的统计；首先要定义各个 cover point；

```
// Transfer collected covergroup
covergroup cov_trans;
  option.per_instance = 1;
  trans_start_addr : coverpoint trans_collected.addr {
    option.auto_bin_max = 16; }
  trans_dir : coverpoint trans_collected.read_write;
  trans_size : coverpoint trans_collected.size {
    bins sizes[] = {1, 2, 4, 8};
    illegal_bins invalid_sizes = default; }
  trans_addrXdir : cross trans_start_addr, trans_dir;
  trans_dirXsize : cross trans_dir, trans_size;
endgroup : cov_trans
```

27. 然后就是进行 coverage 的计算和报告；

```
// perform_transfer_coverage
```

```
  protected function void perform_transfer_coverage();
    cov_trans.sample();
    for (int unsigned i = 0; i < trans_collected.size; i++) begin
      addr = trans_collected.addr + i;
      data = trans_collected.data[i];
//Wait state inforamtion is not currently monitored.
//        wait_state = trans_collected.wait_state[i];
      cov_trans_beat.sample();
    end
  endfunction : perform_transfer_coverage

  task peek(output ubus_transfer trans);
    @address_phase_grabbed;
    trans = trans_collected;
  endtask : peek

  virtual function void report_phase(uvm_phase phase);
    `uvm_info(get_full_name(),$sformatf("Covergroup 'cov_trans' coverage: %2f",
                        cov_trans.get_inst_coverage()),UVM_LOW)
  endfunction
```

28. Ubus_monitor 中还有一个作用，就是进行 slave 的译码设置；具体见前面的 ubus_monitor 部分；

[1-19] 如何顺序的写 UVM 平台（8-8）-Agent.sv

29. Agent 做的事情是最简单的，主要是负责 sequencer，driver，monitor 的例化和连接，注意的是为了向上集成，我们需要判断 UVM_ACTIVE 是否有效，因为向上集成的过程中可能不在需要 sequencer 和 driver，而只需要 monitor；

```
  // build_phase
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    monitor = ubus_slave_monitor::type_id::create("monitor", this);
    if(get_is_active() == UVM_ACTIVE) begin
      driver = ubus_slave_driver::type_id::create("driver", this);
      sequencer = ubus_slave_sequencer::type_id::create("sequencer", this);
    end
  endfunction : build_phase
```

```
// connect_phase
function void connect_phase(uvm_phase phase);
  if(get_is_active() == UVM_ACTIVE) begin
    driver.seq_item_port.connect(sequencer.seq_item_export);
    sequencer.addr_ph_port.connect(monitor.addr_ph_imp);
  end
endfunction : connect_phase
```

[1-20] 如何顺序的写 UVM 平台（8-9）-Scoreboard.sv

30. Scoreboard 主要接收来自 monitor 中的数据，并判断是否是正确的，如果是错误的则会输出一个 error 的指示，在 test_lib 的 base_test 中会用到这个指示，在 base_test 的 report_phase 中报告 FAILED，否则报告 PASSED，要接收数据，首先要声明 TLM port；还要重载 write 函数；

```
uvm_analysis_imp#(ubus_transfer, ubus_example_scoreboard) item_collected_export;
//build_phase
function void build_phase(uvm_phase phase);
  item_collected_export = new("item_collected_export", this);
endfunction

// write
virtual function void write(ubus_transfer trans);
  if(!disable_scoreboard)
    memory_verify(trans);
endfunction : write
```

31. Write 函数内部一般都是调用比较函数，这个比较函数的写法要具体问题具体分析了；大多数时候都会用到队列或者 memory 的方式来进行比较；

[1-21] 如何顺序的写 UVM 平台（8-10）-Env.sv

32. Env 主要负责例化 master agent，slave agent，monitor；

```
    void'(uvm_config_db#(int)::get(this, "", "num_masters", num_masters));

    masters = new[num_masters];
    for(int i = 0; i < num_masters; i++) begin
      $sformat(inst_name, "masters[%0d]", i);
```

```
            masters[i] = ubus_master_agent::type_id::create(inst_name, this);
            void'(uvm_config_db#(int)::set(this,{inst_name,".monitor"},
                        "master_id", i));
            void'(uvm_config_db#(int)::set(this,{inst_name,".driver"},
                        "master_id", i));
        end
```

33. 还有会做 coverage 和 check 的变量控制，当然在 ubus 中还有对 slave 地址空间的配置函数实现；至于 slave 地址空间的配置，后面专门给个总结吧；

```
    // set_slave_address_map
    function void set_slave_address_map(string slave_name,
        int min_addr, int max_addr);
        ubus_slave_monitor tmp_slave_monitor;
        if( bus_monitor != null ) begin
            // Set slave address map for bus monitor
            bus_monitor.set_slave_configs(slave_name, min_addr, max_addr);
        end
        // Set slave address map for slave monitor
        $cast(tmp_slave_monitor, lookup({slave_name, ".monitor"}));
        tmp_slave_monitor.set_addr_range(min_addr, max_addr);
    endfunction : set_slave_address_map
```

[1-22] 如何顺序的写 UVM 平台（8-11）-example_tb.sv

34. 我曾经一度怀疑这个文件存在的作用性，因为对我们来说直接在 test 中例化 env，然后就可以了，后来得到一个解释是，有可能需要模拟多个场景，特别是对于总线这种东西可能需要同时模拟，我用 AXI 总线来说明这个问题，比如说我在一个 chip 中，用了三组 AXI 总线，分别是 2x4,5x8,3x6，我希望的是在同一个平台中同时模拟着三组总线的行为，那么 example_tb 就很有用了，因为可以直接在 test 中例化三个 example_tb 即可。
35. Example_tb 主要用来设置 master 的数量，slave 的数量，slave 的地址分配，当然也包含 scoreboard 和 monitor 的连接。

```
    // build_phase
    virtual function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        uvm_config_db#(int)::set(this,"ubus0","num_masters", 1);
        uvm_config_db#(int)::set(this,"ubus0","num_slaves", 1);
```

```
    ubus0 = ubus_env::type_id::create("ubus0", this);
    scoreboard0 = ubus_example_scoreboard::type_id::create("scoreboard0", this);
  endfunction : build_phase

  function void connect_phase(uvm_phase phase);
    // Connect slave0 monitor to scoreboard
ubus0.slaves[0].monitor.item_collected_port.connect(scoreboard0.item_collected_export);
  endfunction : connect_phase

  function void end_of_elaboration_phase(uvm_phase phase);
    // Set up slave address map for ubus0 (basic default)
    ubus0.set_slave_address_map("slaves[0]", 0, 16'hffff);
  endfunction : end_of_elaboration_phase
```

[1-23] 如何顺序的写 UVM 平台（9）-RAL

1. 以一个 APB 访问寄存器的 dut 为例子，由于 APB 的 Agent 在 UVM 中已经提供，我们只是需要些 regmodel 和在 env 中例化 regmodel 的相关工作等，但是我们首先还是要理解 APB agent 的原理；
2. Callbacks

在 APB 的 driver 中在 get trans 后和执行读写操作后都存在一个 callback 的调用；

```
typedef class apb_master;
class apb_master_cbs extends uvm_callback;
    virtual task trans_received (apb_master xactor , apb_rw cycle);endtask
    virtual task trans_executed (apb_master xactor , apb_rw cycle);endtask
endclass

        @ (this.sigs.mck);

        this.trans_received(tr);
        `uvm_do_callbacks(apb_master,apb_master_cbs,trans_received(this,tr))

      case (tr.kind)
        apb_rw::READ:  this.read(tr.addr, tr.data);
        apb_rw::WRITE: this.write(tr.addr, tr.data);
      endcase
```

this.trans_executed(tr);

`uvm_do_callbacks(apb_master,apb_master_cbs,trans_executed(this,tr))

3. reg2apb_adapter

一个转换器要定义好两个函数：

一个是 reg2bus，作用就是把 register model 通过 sequence 发出的 uvm_reg_bus_op 类型的变量转换成 sequencer 能够接受的形式；

另一个是 bus2reg，作用是当监测到总线上有操作时，把收集来的 transaction 转换成 register model 能够接受的形式，一遍 register model 能够更新相应的寄存器的值。

```
52 class reg2apb_adapter extends uvm_reg_adapter;
53
54   `uvm_object_utils(reg2apb_adapter)
55
56    function new(string name = "reg2apb_adapter");
57       super.new(name);
58    endfunction
59
60   virtual function uvm_sequence_item reg2bus(const ref uvm_reg_bus_op rw);
61     apb_rw apb = apb_rw::type_id::create("apb_rw");
62     apb.kind = (rw.kind == UVM_READ) ? apb_rw::READ : apb_rw::WRITE;
63     apb.addr = rw.addr;
64     apb.data = rw.data;
65     return apb;
66   endfunction
67
68   virtual function void bus2reg(uvm_sequence_item bus_item,
69                                 ref uvm_reg_bus_op rw);
70     apb_rw apb;
71     if (!$cast(apb,bus_item)) begin
72       `uvm_fatal("NOT_APB_TYPE","Provided bus_item is not of the correct type")
73       return;
74     end
75     rw.kind = apb.kind == apb_rw::READ ? UVM_READ : UVM_WRITE;
76     rw.addr = apb.addr;
77     rw.data = apb.data;
78     rw.status = UVM_IS_OK;
79   endfunction
80
81 endclass
```

Register model 发起的读操作的数值是如何返回给 register model 的？monitor 监测到读操作后，把读操作的数据封装成 trans 的形式发送出去，uvm_reg_predictor 的类会接收这个 transaction，并会调用 adapter bus2reg，把 trans 转换成 uvm_reg_bus_op，regsiter model 从后者获取读操作的数值。

uvm_reg_predictor#(apb_rw) apb2reg_predictor;

apb2reg_predictor = new("apb2reg_predictor", this);

reg2apb_adapter reg2apb = new;

regmodel.default_map.set_sequencer(apb.sqr,reg2apb);

apb2reg_predictor.map = regmodel.default_map;

apb2reg_predictor.adapter = reg2apb;

regmodel.default_map.set_auto_predict(0);

apb.mon.ap.connect(apb2reg_predictor.bus_in);

4. Regmodel

一般先一个寄存器（extends uvm_reg）定义成一个 class，每个寄存器中可能包含多个 uvm_reg_filed，new 函数中输入的是总线的宽度，build 函数中包含 create 和 configure 的调用；

Configure 参数列表为：parent,有效位宽，从第几 bit 开始有效，访问方式，volatile，复位值，是否复位，是否随即，是否可以单独存取；

```
24 class dut_ID extends uvm_reg;
25
26    uvm_reg_field REVISION_ID;
27    uvm_reg_field CHIP_ID;
28    uvm_reg_field PRODUCT_ID;
29
30    function new(string name = "dut_ID");
31       super.new(name,32,UVM_NO_COVERAGE);
32    endfunction
33
34    virtual function void build();
35       this.REVISION_ID = uvm_reg_field::type_id::create("REVISION_ID");
36           this.CHIP_ID = uvm_reg_field::type_id::create("CHIP_ID");
37        this.PRODUCT_ID = uvm_reg_field::type_id::create("PRODUCT_ID");
38
39       this.REVISION_ID.configure(this, 8,  0, "RO",   0, 8'h03, 1, 0, 1);
40           this.CHIP_ID.configure(this, 8,  8, "RO",   0, 8'h5A, 1, 0, 1);
41        this.PRODUCT_ID.configure(this, 10, 16,"RO", 0, 10'h176, 1, 0, 1);
42    endfunction
43
44    `uvm_object_utils(dut_ID)
45
46 endclass
```

5. 对于 UVM_MEM 的 extends，只有 new 函数，没有 build 函数，new 函数中会定义长度与宽度；

```
89 class dut_RAM extends uvm_mem;
90
91    function new(string name = "dut_RAM");
92       super.new(name,'h400,32,"RW",UVM_NO_COVERAGE);
93    endfunction
94
95    `uvm_object_utils(dut_RAM)
96
97 endclass
```

6. 将所有的寄存器都组合在一个 uvm_reg_block 中；在 block 中首先例化这些寄存器，然后调用 configure()和 build()函数

```
100 class dut_regmodel extends uvm_reg_block;
101
102    rand dut_ID       ID;
103    rand dut_DATA     DATA;
104
105    rand dut_SOCKET   SOCKET[256];
106
107    rand dut_RAM      RAM;
108
109    function new(string name = "slave");
110       super.new(name,UVM_NO_COVERAGE);
111    endfunction
112
113    virtual function void build();
114
115       // create
116       ID         = dut_ID::type_id::create("ID");
117       DATA       = dut_DATA::type_id::create("DATA");
118       foreach (SOCKET[i])
119          SOCKET[i] = dut_SOCKET::type_id::create($sformatf("SOCKET[%0d]",i));
120       RAM    = dut_RAM::type_id::create("DMA_RAM");
121
122       // configure
123       ID.configure(this,null,"ID");
124       ID.build();
125       DATA.configure(this,null,"DATA");
126       DATA.build();
127       foreach (SOCKET[i]) begin
128          SOCKET[i].configure(this,null,$sformatf("SOCKET[%0d]",i));
129          SOCKET[i].build();
130       end
131       RAM.configure(this,"DMA");
132
133       // define default map
134       default_map = create_map("default_map", 'h0, 4, UVM_LITTLE_ENDIAN, 1);
135       default_map.add_reg(ID,   'h0,  "RW");
136       default_map.add_reg(DATA, 'h24, "RW");
137       foreach (SOCKET[i])
138          default_map.add_reg(SOCKET[i], 'h1000 + 16 * i, "RW");
139       default_map.add_mem(RAM,  'h2000, "RW");
140
141    endfunction
142
143    `uvm_object_utils(dut_regmodel)
144
145 endclass : dut_regmodel
```

7. 在 env 中例化 uvm_reg_block，例化 apb_agent，例化 uvm_reg_sequence，在 build_phase 中 create regmodel, apb_agent,apb2reg_predictor 等；

uvm_reg_sequence seq; 目前还不知道为什么一定要定义这个 sequence?

uvm_reg_predictor#(apb_rw) apb2reg_predictor;

下面是设置 regmodel 中所以寄存器的路径前缀；

```
begin
  string hdl_root = "tb_top.dut";
  void'($value$plusargs("ROOT_HDL_PATH=%s",hdl_root));
  regmodel.set_hdl_path_root(hdl_root);
end
```

```
32 class tb_env extends uvm_component;
33
34    `uvm_component_utils(tb_env)
35
36    dut_regmodel regmodel;
37    apb_agent    apb;
38    uvm_reg_sequence seq;
39 `ifdef EXPLICIT_MON
40    uvm_reg_predictor#(apb_rw) apb2reg_predictor;
41 `endif
42
43    function new(string name, uvm_component parent=null);
44       super.new(name,parent);
45    endfunction
46
47    virtual function void build_phase(uvm_phase phase);
48       if (regmodel == null) begin
49          regmodel = dut_regmodel::type_id::create("regmodel",,get_full_name());
50          regmodel.build();
51          regmodel.lock_model();
52
53          apb = apb_agent::type_id::create("apb", this);
54 `ifdef EXPLICIT_MON
55          apb2reg_predictor = new("apb2reg_predictor",this);
56 `endif
57       end
58
59       begin
60         string hdl_root = "tb_top.dut";
61         void'($value$plusargs("ROOT_HDL_PATH=%s",hdl_root));
62         regmodel.set_hdl_path_root(hdl_root);
63       end
64
65    endfunction
```

```
    virtual function void connect_phase(uvm_phase phase);
        if (apb != null) begin
            reg2apb_adapter reg2apb = new;
            regmodel.default_map.set_sequencer(apb.sqr,reg2apb);
`ifdef EXPLICIT_MON
            apb2reg_predictor.map = regmodel.default_map;
            apb2reg_predictor.adapter = reg2apb;
            regmodel.default_map.set_auto_predict(0);
            apb.mon.ap.connect(apb2reg_predictor.bus_in);
`else
            regmodel.default_map.set_auto_predict(1);
`endif
        end
        regmodel.print();
    endfunction
```

```
67    virtual function void connect_phase(uvm_phase phase);
68       if (apb != null) begin
69          reg2apb_adapter reg2apb = new;
70          regmodel.default_map.set_sequencer(apb.sqr,reg2apb);
71 `ifdef EXPLICIT_MON
72          apb2reg_predictor.map = regmodel.default_map;
73          apb2reg_predictor.adapter = reg2apb;
74          regmodel.default_map.set_auto_predict(0);
75          apb.mon.ap.connect(apb2reg_predictor.bus_in);
76 `else
77          regmodel.default_map.set_auto_predict(1);
78 `endif
79       end
80       regmodel.print();
81    endfunction
82
83    virtual task run_phase(uvm_phase phase);
84      phase.raise_objection(this);
85      if (seq == null) begin
86        uvm_report_fatal("NO_SEQUENCE","Env's sequence is not defined. Nothing to do. Exiting.");
87        return;
88      end
89
90      begin : do_reset
91        uvm_report_info("RESET","Performing reset of 5 cycles");
92        tb_top.rst <= 1;
93        repeat (5) @(posedge tb_top.clk);
94        tb_top.rst <= 0;
95      end
96
97      #100;
98
99      uvm_report_info("START_SEQ",{"Starting sequence '",seq.get_name(),"'"});
100     seq.model = regmodel;
101     seq.start(null);
102     phase.drop_objection(this);
103   endtask
104
105
106 endclass
```

```
virtual task run_phase(uvm_phase phase);
  phase.raise_objection(this);
  if (seq == null) begin
    uvm_report_fatal("NO_SEQUENCE","Env's sequence is not defined. Nothing to do. Exiting.");
    return;
  end
  #100;

  uvm_report_info("START_SEQ",{"Starting sequence '",seq.get_name(),"'"});
  seq.model = regmodel;
  seq.start(null);
  phase.drop_objection(this);
endtask
```

8. 在 test 中需要定义一个 sequence 赋值给 env 中的 sequence；

```
36 initial
37 begin
38    static tb_env env = new("env");
39
40    begin
41      uvm_report_server svr;
42      svr = _global_reporter.get_report_server();
43      svr.set_max_quit_count(10);
44    end
45
46    begin
47      string seq_name;
48      if ($value$plusargs("UVM_SEQUENCE=%s",seq_name)) begin
49        uvm_reg_sequence seq;
50        seq = uvm_utils #(uvm_reg_sequence)::create_type_by_name(seq_name,"tb");
51        if (seq == null)
52          uvm_report_fatal("NO_SEQUENCE",
53            "This env requires you to specify the sequence to run using UVM_SEQUENCE=<name>");
54        env.seq = seq;
55      end
56    end
57
58    uvm_config_db#(apb_vif)::set(env, "apb", "vif", $root.tb_top.apb0);
59
60    run_test();
61 end
```

seq = uvm_utils #(uvm_reg_sequence)::create_type_by_name(seq_name,"tb");//不知道这是个什么意思呢？应该就是创建一个 uvm_reg_sequence

9. 如果 regmodel 中的寄存器 create 函数没有写全或者写对，会出现下面的错误；

```
ncelab: *F,INTERR: INTERNAL EXCEPTION
-------------------------------------------------------------
The tool has encountered an unexpected condition and must exit.
Contact Cadence Design Systems customer support about this
problem and provide enough information to help us reproduce it,
including the logfile that contains this error message.
  TOOL: ncelab  10.20-s009
  HOSTNAME: cdlc03
  OPERATING SYSTEM: Linux 2.6.18-238.el5 #1 SMP Thu Jan 13 15:51:15 EST 2011 x86_64
  MESSAGE: sv_seghandler - trapno -1 addr(0x00000000)
-------------------------------------------------------------
csi-ncelab - CSI TRIAL: Cadence Support Investigation, sending details to ncelab.err
csi-ncelab - CSI TRIAL: investigation complete, send ncelab.err to Cadence Support
irun: *E,ELBERR: Error during elaboration (status 255), exiting.
make: *** [test] Error 1
```

[1-24] 如何顺序的写 UVM 平台（10）-打印信息汇总

1. 在 ubus_base_test 中：

`uvm_info(get_type_name(),$sformatf("Printfing the test topology : \n %s",this.sprint(printer)),UVM_LOW)

```
UVM_INFO test_lib.sv(25) @ 0: uvm_test_top [test_read_byte] Printfing the test topology :
 ----------------------------------------------------------------------
Name                          Type                     Size  Value
----------------------------------------------------------------------
uvm_test_top                  test_read_byte           -     @249
  ubus_example_tb0            ubus_example_tb          -     @4935
    scoreboard0               ubus_example_scoreboard  -     @5043
      item_collected_export   uvm_analysis_imp         -     @5192
      disable_scoreboard      integral                 1     'h0
      num_writes              integral                 32    'd0
      num_init_reads          integral                 32    'd0
      num_uninit_reads        integral                 32    'd0
      recording_detail        uvm_verbosity            32    UVM_FULL
    ubus0                     ubus_env                 -     @4934
      bus_monitor             ubus_bus_monitor         -     @5005
      masters[0]              ubus_master_agent        -     @5270
      slaves[0]               ubus_slave_agent         -     @5044
      num_masters             integral                 32    'h1
      num_slaves              integral                 32    'h1
      has_bus_monitor         integral                 1     'h1
      intf_checks_enable      integral                 1     'h1
      intf_coverage_enable    integral                 1     'h1
      recording_detail        uvm_verbosity            32    UVM_FULL
    recording_detail          uvm_verbosity            32    UVM_FULL
----------------------------------------------------------------------
```

2. 在 ubus_base_test 中：

```
uvm_report_info(get_full_name(),"start of test run_phase...",UVM_LOW);
uvm_top.print_topology();
```

```
UVM_INFO @ 0: uvm_test_top [uvm_test_top] start of test run_phase...
UVM_INFO @ 0: reporter [UVMTOP] UVM testbench topology:
----------------------------------------------------------------------
Name                            Type                     Size  Value
----------------------------------------------------------------------
uvm_test_top                    test_read_byte           -     @249
  ubus_example_tb0              ubus_example_tb          -     @4935
    scoreboard0                 ubus_example_scoreboard  -     @5043
      item_collected_export     uvm_analysis_imp         -     @5192
        recording_detail        uvm_verbosity            32    UVM_FULL
      disable_scoreboard        integral                 1     'h0
      num_writes                integral                 32    'd0
      num_init_reads            integral                 32    'd0
      num_uninit_reads          integral                 32    'd0
      recording_detail          uvm_verbosity            32    UVM_FULL
    ubus0                       ubus_env                 -     @4934
      bus_monitor               ubus_bus_monitor         -     @5005
        item_collected_port     uvm_analysis_port        -     @5375
          recording_detail      uvm_verbosity            32    UVM_FULL
        state_port              uvm_analysis_port        -     @5449
          recording_detail      uvm_verbosity            32    UVM_FULL
        checks_enable           integral                 1     'h1
        coverage_enable         integral                 1     'h1
        num_transactions        integral                 32    'h0
        slave_addr_map          aa(object,string)        1     -
          [slaves[0]]           slave_address_map_info   -     @8768
            min_addr            integral                 32    'h0
            max_addr            integral                 32    'hffff
        recording_detail        uvm_verbosity            32    UVM_FULL
      masters[0]                ubus master agent        -     @5270
```

3. 在 master_monitor 中：

`uvm_info({get_full_name()," MASTER ID"},$sformatf(" = %0d",master_id),UVM_MEDIUM)

UVM_INFO ubus_master_monitor.sv(65) @ 0: uvm_test_top.ubus_example_tb0.ubus0.masters[0].monitor [uvm_test_top.ubus_example_tb0.ubus0.masters[0].monitor MASTER ID] =0

4. 在 master_sequence_lib 中：

```
if (starting_phase!=null) begin
    `uvm_info(get_type_name(), $sformatf("%s pre_body() raising %s objection",
            get_sequence_path(), starting_phase.get_name()), UVM_MEDIUM);
```

UVM_INFO ubus_master_seq_lib.sv(12) @ 0: uvm_test_top.ubus_example_tb0.ubus0.masters[0].sequencer@@read_byte_seq [read_byte_seq] read_byte_seq pre_body raising run objection

5. 在 slave_sequence_lib 中：

```
`uvm_info(get_type_name(), $sformatf("%s starting...",
  get_sequence_path()), UVM_MEDIUM);
```

UVM_INFO ubus_slave_seq_lib.sv(66) @ 0: uvm_test_top.ubus_example_tb0.ubus0.slaves[0].sequencer@@slave_memory_seq [slave_memory_seq] slave_memory_seq starting ...

6. 在 master_seq_lib 中：

```
`uvm_info(get_type_name(),$sformatf("%s read : addr = `x%0h, data[0] = `x%0h",
  get_sequence_path(), rsp.addr, rsp.data[0]), UVM_HIGH);
```

UVM_INFO ubus_master_seq_lib.sv(48) @ 300: uvm_test_top.ubus_example_tb0.ubus0.masters[0].sequencer@@read_byte_seq [read_byte_seq] read_byte_seq read: addr=`x2c7,data[0]= `x39

[1-25] 如何顺序的写 UVM 平台（11）-疑惑

1. 使用宏的时候中间不要换行；换行的话加 “\”。
2. 在 ubus_slave_seq_lib.sv 中；没明白下面的意思；

```
ubus_slave_sequencer p_sequencer;
  $cast(p_sequencer, m_sequencer);
   p_sequencer.addr_ph_port.peek(util_transfer);
```

一般来说我们在 sequencer 中都不增加任何变量，但是在 ubus_slave_sequencer 中增加了一个 umv_blocking_peek_port，例化成了 addr_ph_port，这 addr_ph_port 是 sequencer 用来和 slave_monitor 进行通信的。通信的主要原因是 slave_monitor 收到 transaction 的写操作后需要发送给 slave_memory_seq 中进行保存；所以在 slave_memory_seq 中通过 TLP 接口调用了 slave_monitor 中的 peek 函数；

再来看看 p_sequencer 和 m_sequencer 的区别： m_sequecer 是指向基类 uvm_sequencer_base 的指针，而 p_sequencer 是指向从 uvm_sequencer_base 派生出来的 ubus_slave_sequencer 的指针；所以如果要用到子对象 ubus_slave_sequencer 中的成员变量，就必须声明一个 p_sequencer 指针才能调用；

3. 在 ubus_slave_seq_lib.sv 中；没明白下面的意思；

```
`uvm_declare_p_sequencer(ubus_slave_sequencer)
$cast(req, create_item(ubus_transfer::get_type(), p_sequencer, "req"));
    p_sequencer.addr_ph_port.peek(util_transfer);
```

4. 在 ubus_slave_monitor.sv 中；没明白下面的意思；

```
void'(this.begin_tr(trans_collected));
```

5. Get_full_name()和 get_type_name()的区别

Get_full_name()：

uvm_test_top.ubus_example_tb0.ubus0.slaves[0].driver [uvm_test_top.ubus_example_tb0.ubus0.slaves[0].driver]

Get_type_name()：

uvm_test_top.ubus_example_tb0.scoreboard0 [ubus_example_scoreboard]