ECEN 468   Advanced Logic Design
Department of Electrical and Computer Engineering
Texas A&M University

(Lab exercise created by Jaeyeon Won and Jiang Hu)

Lab 6

SystemC Verification Library (SCV)

Purpose:

In this lab, we will verify a module using SystemC Verification Library (SCV) which is an additional library given in SystemC standard. We will also use Vista as a tool for simulation and verification.

Preparation:

1. SystemC Verification Library (SCV)

SystemC Verification Library (SCV) which is SystemC Verification standard includes many add-on features to SystemC including data introspection, weighted randomization, transaction-based verification, exception handling, and various verification tasks. This allows us to implement a reusable and more easily readable testbench without developing by your own. For more information, you can download from www.systemc.org.

2. Randomization

In general, it is very important to verify hardware designs with more and distributed test coverage. Traditionally, the verification has been done by directed testing methodology, and its reliability also has been issued. To cover it with higher reliability, random testing methodology gives more benefits sometimes with directed testing methodology. When it uses randomization tests to verify, the stimulus (test-bench) is created through constrained randomization given by users. Users can set various constraints to verify their system models, and test many cases without further modification on their test-benches.

Absolutely, directed methodology is important as well to test the system model with the desired scenarios and the situations given. Also, a complete test can be achieved by directed test methodology for relatively small model if time for full verification is not too long and acceptable. However, the system volume has been increased nowadays, it requires faster and efficient methodology to verify and acquire similar reliability to the full test which has huge time consuming. In addition, it may be difficult and impossible for the human to generate random scenarios and possible situations by using direct verification methodology. In this sense, the randomization methodology is very powerful and efficient.

SCV provides the infrastructure for users to create basic randomization, constrained randomization, and weighted randomization tests. We will verify SRAM module which has been created in Lab1 with the randomization methodology. In this lab, it will cover the basics of the random demonstration of how to use the SCV library using the standard API and two **sc_interface_if** templates will be used for randomization: **scv_smart_ptr** and **scv_bag**.

3. sc_interface_if templates used in the lab

3.1. scv_smart_ptr

We can use templates provided by SCV to handle the scv_extensions pointer. The scv_smart_ptr class operates like a C++ pointer to the scv_extensions object. The scv_smart_ptr templates includes scv_extensions and scv_shared_ptr objects. The scv_shared_ptr can be used when multiple threads share same data objects with the necessary memory management. You should instantiate the object with the appropriate data type as shown below.

```
scv_smart_ptr<Packet> pPkt;
pPkt->address = 0;
pPkt->data = 0;
```

3.2. scv_bag

Scv_bag which is one of the template classes is used when you need more complicated distribution rules for data manipulation. We can define relative weights of particular values by using scv_bag as shown below.

```
scv_bag<int> intBag;
intBag.add(0, 25);          // add 25 objects of value 0 to bag
intBag.add(2, 75);          // add 75 objects of value 2 to bag
scv_smart_ptr<int> smart_int;
smart_int->set_mode(intBag);    // set smart_int distribution;
```

4. Functional Simulation

Once the design is completed it must be tested. The functionality of the design module can be tested by applying a testbench and checking the results. The testbench module can instantiate the design module and directly drive the signals in the design module. The testbench can be complied along with the design module. At the end of compilation the simulation results will be displayed.

You will reuse SRAM module which has been created in Lab1. Download only test_RAM.cpp and move it into the Lab6 working folder.

a. Make working folder for this lab.

```
        HOME]$ mkdir ECEN468/Lab6/SRC                    (make work folder)
        HOME]$ cd ECEN468/Lab6/SRC                       (move to work folder)
```

Download test_RAM.cpp from lab web page and move it into the working folder. Copy RAM.cpp file that you created into the same folder.

b. Please make a new project including the two files: RAM.cpp and test_RAM.cpp.

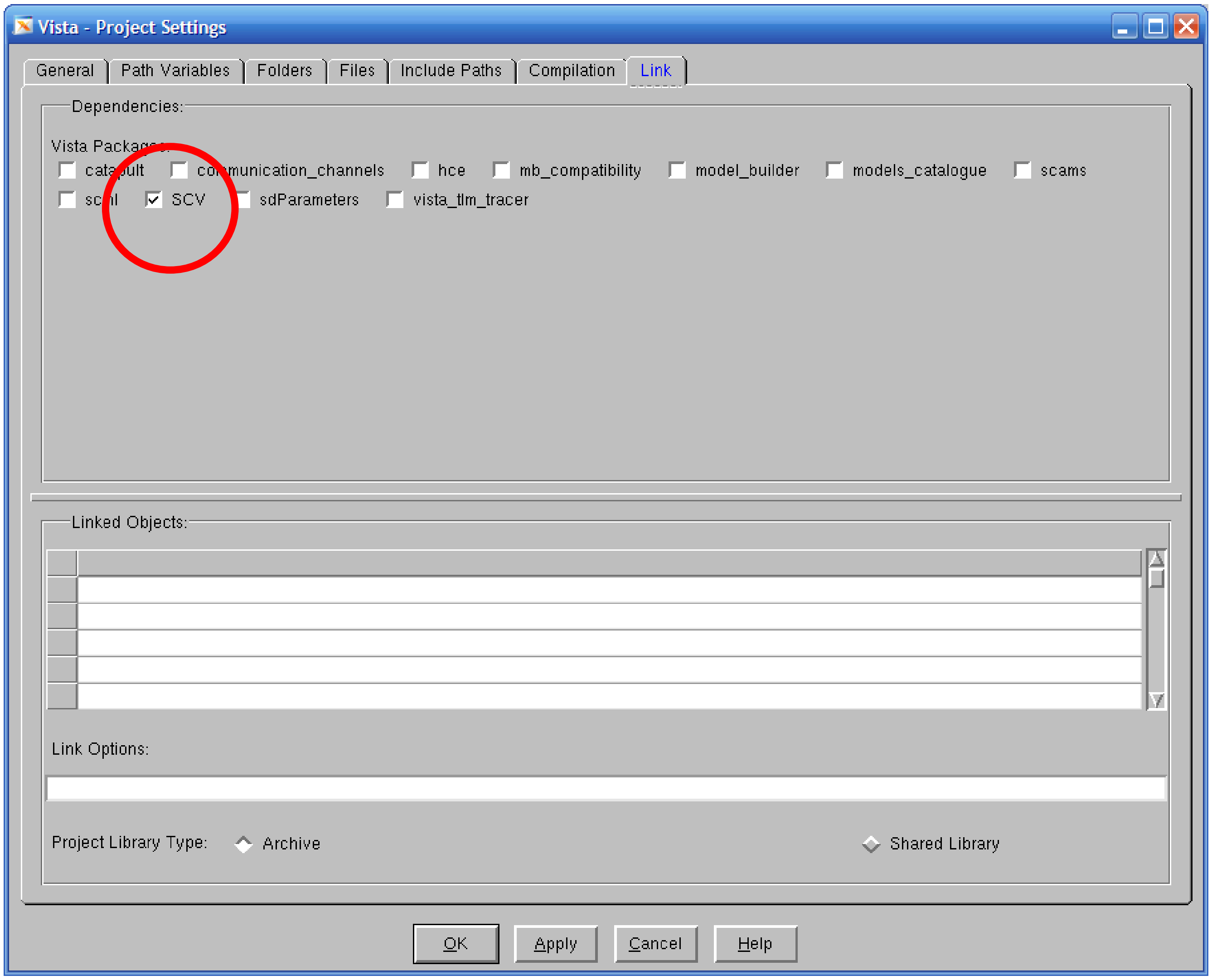
c. Link to SCV library.

- Click Project – Setting in Menu bar.
- Click Link tab.
- Check the check box for SCV as shown in Figure 1.

d. Add your testbench for Verification Part I to satisfy with the requirements. Please refer to an example and some references sites at the end of the manual.

e. Build your code and testbench and do simulation. For how to build and simulate, please refer to the previous labs. Please show your result of Verification Part I to TA within your regular lab hour to get credit for lab evaluation.

f. Add your testbench for Verification Part II to satisfy with the requirements, build and simulate. Please show your result of Verification Part II to TA within your regular lab hour to get credit for lab evaluation.


Vista - Project Settings

General | Path Variables | Folders | Files | Include Paths | Compilation | Link

Dependencies:

Vista Packages:

- [ ] catapult
- [ ] communication_channels
- [ ] hce
- [ ] mb_compatibility
- [ ] model_builder
- [ ] models_catalogue
- [ ] scams
- [ ] scml
- [✓] SCV
- [ ] sdParameters
- [ ] vista_tlm_tracer

Linked Objects:

Link Options:

Project Library Type: Archive / Shared Library

OK | Apply | Cancel | Help

Figure 1 Project setting window

5. Reference

Chapter.15 in "SystemC: From the Ground Up", *David C. Black, Jack Donovan, Bill Bunton, Anna Keist*, Springer, 2nd Edition, 2009.

6. Simulation Tips

- Insert timing delay properly using sc_start(#) to control input and output signals.
- Delete "cout" commands at your SRAM module.

Report Requirements:

1. Your design modules should satisfy constraints below.
   a. It should verify SRAM module correctly using sc_interface_if templates.
   b. Detailed comments.
   c. Please do not change the names of the signals given and the other functions not mentioned in the report. (You will lose some points if you change any names of the input or output signals.)
   d. Please implement below two verification parts in one testbench file.
   e. Late penalty : 20% of total score will be deducted on each subsequent weekday after due date.

2. Submit hardcopy of your report to TA. The report should include contents below.
   a. Please use "*[Name]*.print()" instead of "cout" for printing results. (5 points)
   b. Constraints of Verification Part I (10 points)
      - The value of Addr signal should be lower than 10.
      - The value of InDdata signal should be higher than 80.
      - The value of Addr and InData should be reloaded per every test cycles.
      - Please use "*[Name]*.print()" instead of "cout" for printing.
   c. Constraints of Verification Part II (10 points)
      - Do not regenerate the value of Addr and keep using the previous value of Part I.
      - Please generate the value of InData with 5% from 80 to 99 and 95% from 100 to 120 using **scv_bag**.
   d. sim.out (test message output) with analysis. (5 points)
      - The message should be able to show all requirements.
   e. Captured waveform with analysis. (5 points)

Note : Please send your source codes, testbench, results(sim.out and *.vcd) to TA's email address. (5 points)

An example in the reference book

```
class Pkt_constraint : virtual public scv_constraint_base
{
        public:
        scv_smart_ptr<sc_uint<16>> address;
        scv_smart_ptr<sc_uint<32>> data;

        SCV_CONSTRAINT_CTOR(Pkt_constraint) {
        // define constraints
                SCV_CONSTRAINT(
                        (address() != 0x00000000) &&
                        (address() < 0x00001000));
                SCV_CONSTRAINT(data() >= 0x1000);
        }
};

void test() {
        typedef pair <sc_uint<32,> sc_u int<32>> data_range;
        scv_bag<data_range> data_dist;

        //Set range distribution for data
        //data range (0x1000, 0xffff) occurs 30%
        //data range (0x10000, 0x20000) occurs 70%)
        data_dist.add(data_range(0x1000, 0xffff), 30);
        data_dist.add(data_range(0x10000, 0x20000), 70);

        Pkt_constraint cPkt("name_of_class");
        cPkt.next();            //generate addr and data using constraints
        cPkt.addr->next();      //generate addr only using constraints

        cPkt.data->set_mode(data_dist);
        cPkt.next();            //generate addr using constraints
                                // and generate data using 'data_dist' distribution
}
```

Fig. 15.18 Changing randomization modes

Verification Using *SystemC* Part VII

Verification Using *SystemC* Part VIII