ECEN 468/719 Advanced Logic Design

Department of Electrical and Computer Engineering

Texas A&M University

Lab 6: SystemC Verification Library

Objectives

In this lab, we will verify a module using SystemC Verification Library (SCV), which is an additional library given in SystemC standard. SCV provides the infrastructure to create basic randomization, constrained randomization, and weighted randomization tests. In this lab, we will verify the SRAM module from Lab 1 using randomization methodology. We will cover the basics of the random demonstration. Specifically, two **sc_interface_if** templates will be used for randomization: **scv_smart_ptr** and **scv_bag**.

Introduction

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 easily readable test bench. More information can be found at www.systemc.org.

Randomization

In general, it is essential to verify hardware designs with randomly distributed test coverage. Traditionally, the verification has been done by directed testing methodology. To cover it with higher reliability, random testing methodology gives more benefits. 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 modules and test many cases without further modification on their test benches.

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

sc_interface_if Templates

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 include **scv_extensions** and **scv_shared_ptr** objects. The scv_shared_ptr can be used when multiple threads share the 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;
```

scv_bag is one of the template classes used when you need more complicated distribution rules for data manipulation. We can define the relative weights of particular values 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;
```

Implementation & Simulation

Please login to the Olympus server and create a working directory for this lab using the following commands.

```
## Create and navigate to the working directory.
mkdir -p $HOME/ECEN468/Lab6/src
cd $HOME/ECEN468/Lab6/src
```

Download file test_RAM.cpp from Piazza. Copy it to the working directory. Please also copy RAM.cpp from Lab 1 to the working directory. Please create a new project including those two files.

Link to SCV library:

1. In the menu bar, click on **Project -> Setting**;
2. Click on the tab **Link**;
3. Check the box for **SCV,** as shown in Figure 1, and click on OK.

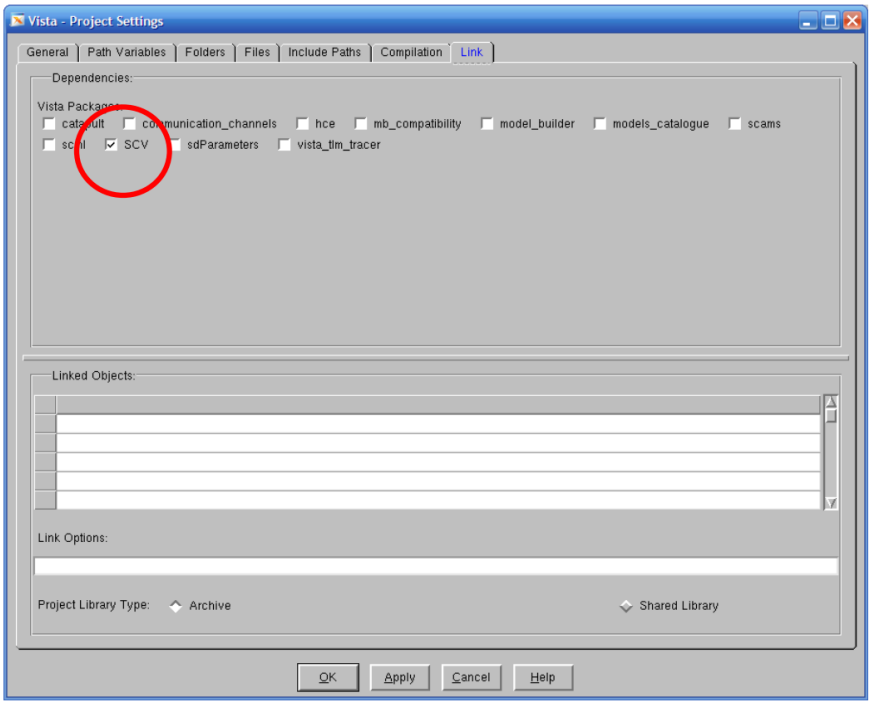

Figure 1. Project settings window

In test_RAM.cpp, you will need to complete **Verification I & II**. Below are the requirements.

- For both Verifications I & II
  - Please use "[name].print()" instead of "cout" for printing.
  - Please remove "cout" in the SRAM module if there is any.
  - Insert timing delay properly using **sc_start(#)** to control input and output.
- For Verification I
  - The value of **Addr** should be greater than 0, and lower than 10.
  - The value of **InData** should be higher than or equal to 80.
  - The value of **Addr** and **InData** should be reloaded per test cycle.
- For Verification II
  - The value of **InData** should be different from Verification I, with 5% from 80 to 99 and 95% from 100 to 120 generated with **scv_bag**.

For your reference, a code example is given at the end of this manual.

Please take screenshots of the simulation output and include them in the report.

Commands for reference:

```
load-ecen-468
source /opt/coe/mentorgraphics/vista312/setup.vista312.linux.bash
vista &
source /opt/coe/synopsys/wv/O-2018.09/setup.wv.sh
wv &
```

Submission

Please only submit one PDF file containing the following items:

1. Screenshots of the waveform with analysis.
2. Screenshots of the simulation output in Vista.
3. Screenshots of your code in this design with reasonable comments.

Code Example

```
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
}
```

Figure 2. A code example

More references:

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