Verilog-C++ co-simulation using VppSim

6.973 Tutorial 6

Ajay Joshi, Michael Perrott

Last update: November 5, 2008

In this tutorial you will learn about using VppSim to perform Verilog-C++ co-simulation. This tutorial will help you successfully complete checkpoint 2 of the project. Before you begin this tutorial, please complete tutorial 1 if you have not done it. For this tutorial, we assume that you are comfortable with the Cadence environment.

As described in the first tutorial, VppSim is a general behavioral framework that leverages the C++ language to achieve very fast simulation times, and a graphical framework to allow ease of design entry and modification.

The VppSim framework is a superset of three different, but related, simulation environments:

- CppSim: C++ only behavioral simulation
- VppSim: C++ and Verilog co-simulation
- AMS with VppSim modules: C++, Verilog, VHDL, and SPICE co-simulation

For more information on CppSim and VppSim, including manuals and documentation, consult http://www.cppsim.com.

Getting started

It is assumed that you have already installed VppSim on your system and read through the document VppSim Primer available at http://www.cppsim.com.

For this tutorial we will be using the scrambler as our test design. Figure 1 shows the block diagram of our system. We will first simulate this system using C++ behavioral models. For the second step we will use C++ model for the data source and verilog functional model for the scrambler.

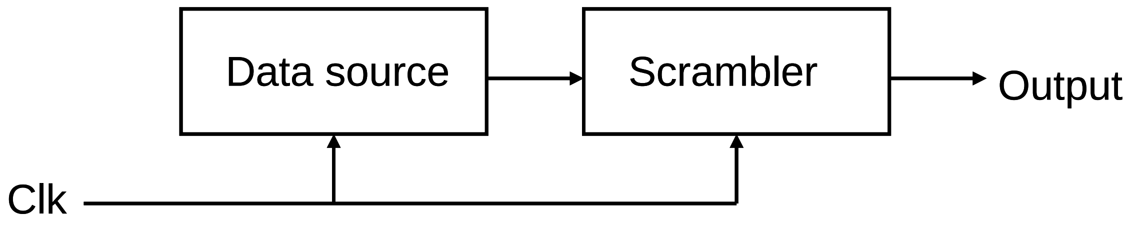


Figure 1. Block diagram of the target system

Once you have completed the setup described earlier, start Cadence Virtuoso Mixed Signal Design tool using the following steps at a Linux prompt

```
% cd VppSim/cds
%  icms&
```

This should open the Cadence Interpreter Window (CIW) and the Library Manager. If the Library Manager does not open then it can be opened using Tools → Library Manager in the CIW menu. Create a new library using File → New → Library in the menu of the Library Manager Window (LMW). In the New Library window that appears name the library as *MyCppSimLib*. After pressing *OK*, designate that you do not need a techfile within the form that appears. The next step is to create a new CppSim module schematic. Select *MyCppSimLib* within the Library Manager and then select File → New → Cell View. In the window that appears choose the Cell Name to be *source* and then press *OK*. Within the newly opened schematic window, create a one input pin named *clk* and one output pin named *out<25:0>*. Attach *noConn* instances from the *CppSimModules* library to each pin in order to avoid warning messages when you save the schematic (note that you'll need to specify an arrayed instance (i.e., *I1<25:0>*) for the *noConn* instance attached to pin *out<25:0>*). Press *check-and-save* (top left hand corner of the schematic window) button to save the schematic. Upon completion, your schematic should appear as shown in Figure 2.

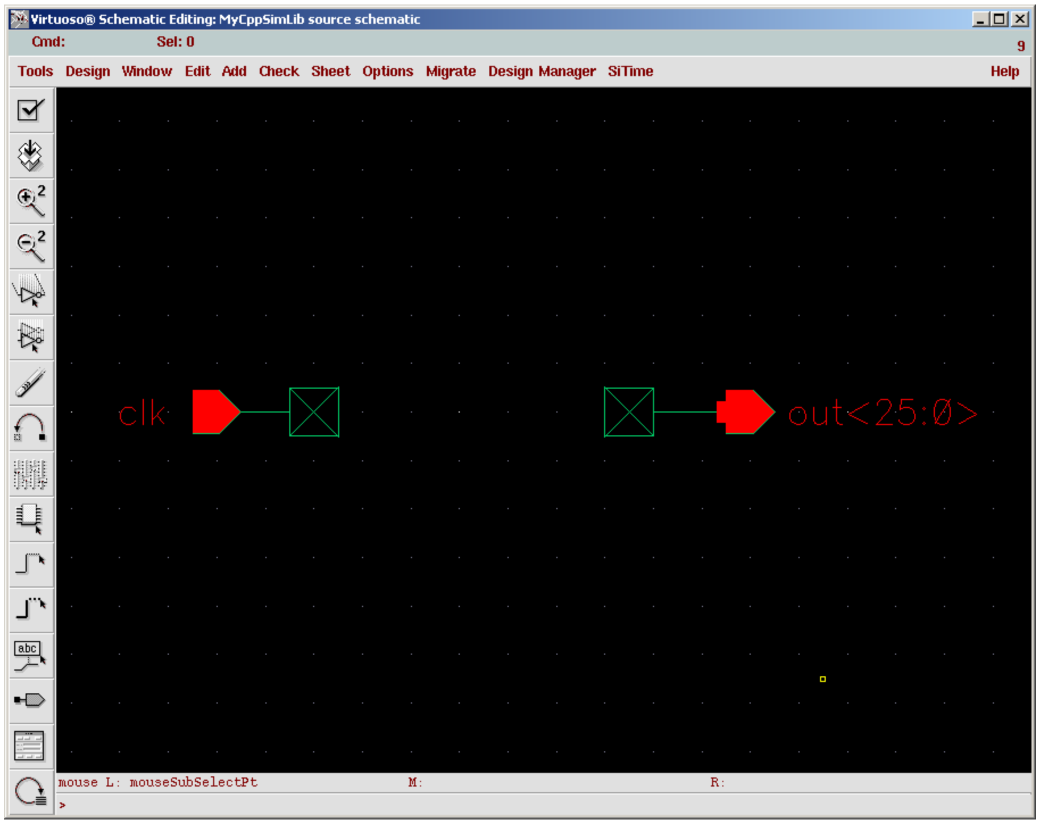


Figure 2. Schematic view of the *source* cell

Next create a new CppSim module symbol. To do this within the schematic window menu, select Design → Create Cellview → From Cellview. Press *OK* on the forms that follow. A default symbol window should appear. Please make the necessary cosmetic changes described in tutorial 1.

The next step is to create the CppSim module code. In the Cadence Library Manager, select *MyCppSimLib* and then in the menu select File → New → Cell View. Within the Create New File window that opens, select *Text-Editor* as the tool. Designate the View Name to be *cppsim* and then push *OK*. A blank editor window should appear. Copy the following CppSim module code description.

```
module: source
parameters:
inputs: double_interp clk;
outputs: bool out[25:0];
classes: EdgeDetect pos_edge();
        IntVector out_buffer();
static_variables: int count;
init:
count = 0;
out_buffer.set_length(out.get_length());
// set seed of rand() function
srand(17);

code:
int i;

out.copy(out_buffer);

if (pos_edge.inp(clk))
   {
   if (count == 0)
      {
      out_buffer.set_elem(0,1);
      out_buffer.set_elem(1,1);
      }
   else
      {
      out_buffer.set_elem(0,0);
      out_buffer.set_elem(1,0);
      }
   for (i = 2; i < out_buffer.get_length(); i++)
      {
       out_buffer.set_elem(i,rand() % 2);
      }

   count++;
   if (count > 10)
      count = 0;
  }
```

We now have a *source* cell and its corresponding CppSim module code defined. Adopt the same steps described above to create a *scrambler* cell and its corresponding CppSim module code. The scrambler cell will have two input pins – *clk* and *input_data<25:0>*, and one output pin – *output_data<25:0>*. The code for the scrambler cell is given below.

```
module: scrambler
description:
parameters:
inputs:
        double_interp clk;
        bool input_data[25:0];
outputs:
        bool output_data[25:0];
classes:
        EdgeDetect pos_edge();
        IntVector scram_seq();
        IntVector scram_seed();
         IntVector output_buffer();
static_variables:
init:
       scram_seq.set_length(24);
       scram_seed.set_length(7);
       output_buffer.set_length(output_data.get_length());
        // initialize constants
       int scram_seed_init[7] = {0, 1, 1, 0, 1, 1, 0}; // seed = 1011101
       for (int i = 0; i < scram_seed.get_length(); i++)
       scram_seed.set_elem(i,scram_seed_init[i]);
code:
         int i;
        // wires
        int new_message = input_data.get_elem(0) | input_data.get_elem(1);

         output_data.copy(output_buffer);
        // clock edge
        if(pos_edge.inp(clk)){
                // print debugging info
                printf("G scrambler in(");
                printf("%i%i ", input_data.get_elem(0), input_data.get_elem(1));
                for (i = 2; i < input_data.get_length(); i++)
                      printf("%i", input_data.get_elem(i));
                printf(")");

                printf(" out(");
                printf("%i%i ", output_buffer.get_elem(0), output_buffer.get_elem(1));
                for (i = 2; i < output_buffer.get_length(); i++)
                      printf("%i", output_buffer.get_elem(i));
                printf(")\n");


                // process input
                // latch rate
                output_buffer.set_elem(0,input_data.get_elem(0));
                output_buffer.set_elem(1,input_data.get_elem(1));

                // re-initialize scramble sequence if necessary
                if (new_message){
                         for(i = 0; i < scram_seed.get_length(); i++)
                scram_seq.set_elem(i,scram_seed.get_elem(i));
                         for(i = scram_seed.get_length(); i < scram_seq.get_length(); i++)
                scram_seq.set_elem(i,scram_seq.get_elem(i-4)^scram_seq.get_elem(i-7));
                 }
                 // scramble and latch input
                 for (i = 2; i < output_buffer.get_length(); i++) {
                         output_buffer.set_elem(i,input_data.get_elem(i) ^ scram_seq.get_elem(i-2));
                 }
                 // create new scramble sequence
                for (i = 0; i < 4; i++)
                   scram_seq.set_elem(i,scram_seq.get_elem(24 + i - 4) ^ scram_seq.get_elem(24 + i - 7));
                for(i = 4; i < 7; i++)
                   scram_seq.set_elem(i,scram_seq.get_elem(i - 4) ^ scram_seq.get_elem(24 + i - 7));
                for(i = 7; i < 24; i++)
                   scram_seq.set_elem(i,scram_seq.get_elem(i - 4) ^ scram_seq.get_elem(i - 7));
        }
end:
```

The next step will be to create our test system. To do this, from the Cadence Library Manager, create a schematic cell view named *top_level*. Build the structure shown in Figure 3, in the schematic window for the *top_level* cell. The *top_level* structure has one instance of constant cell, vco cell, source cell and scrambler cell. The final output of the system is the *scram_out<25:0>* pin. (Note: All modules that are not part of your library can be found in the CppSimModules library). For the *constant* cell, the value of parameter val to be 0, while for the *vco* cell set the value of parameter fc = 200e6 Hz and Kvco = 1 Hz/V.

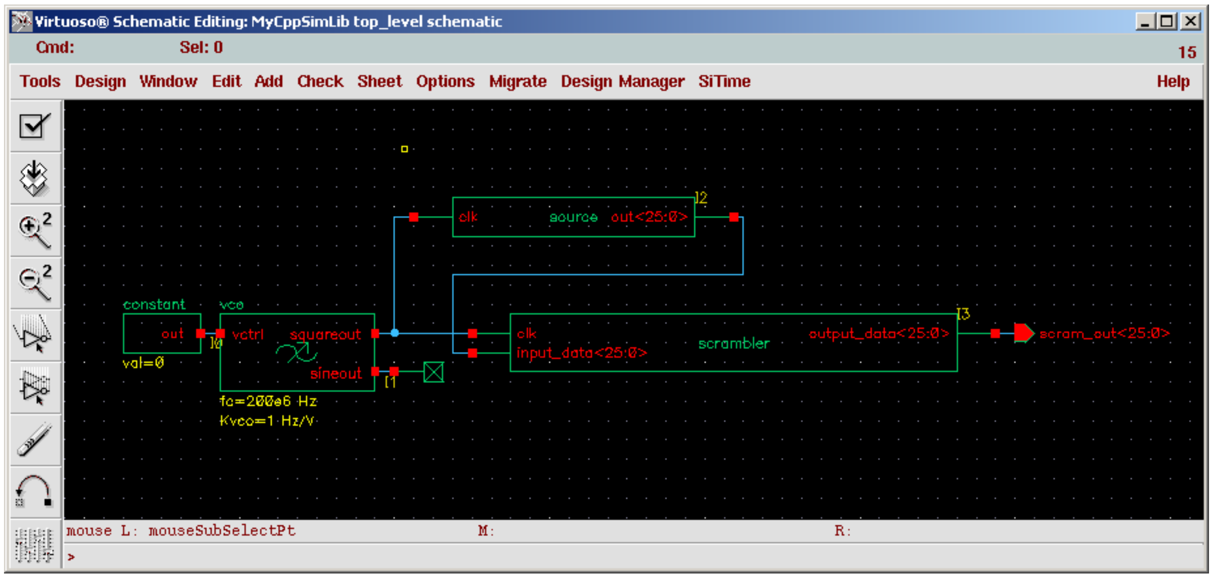
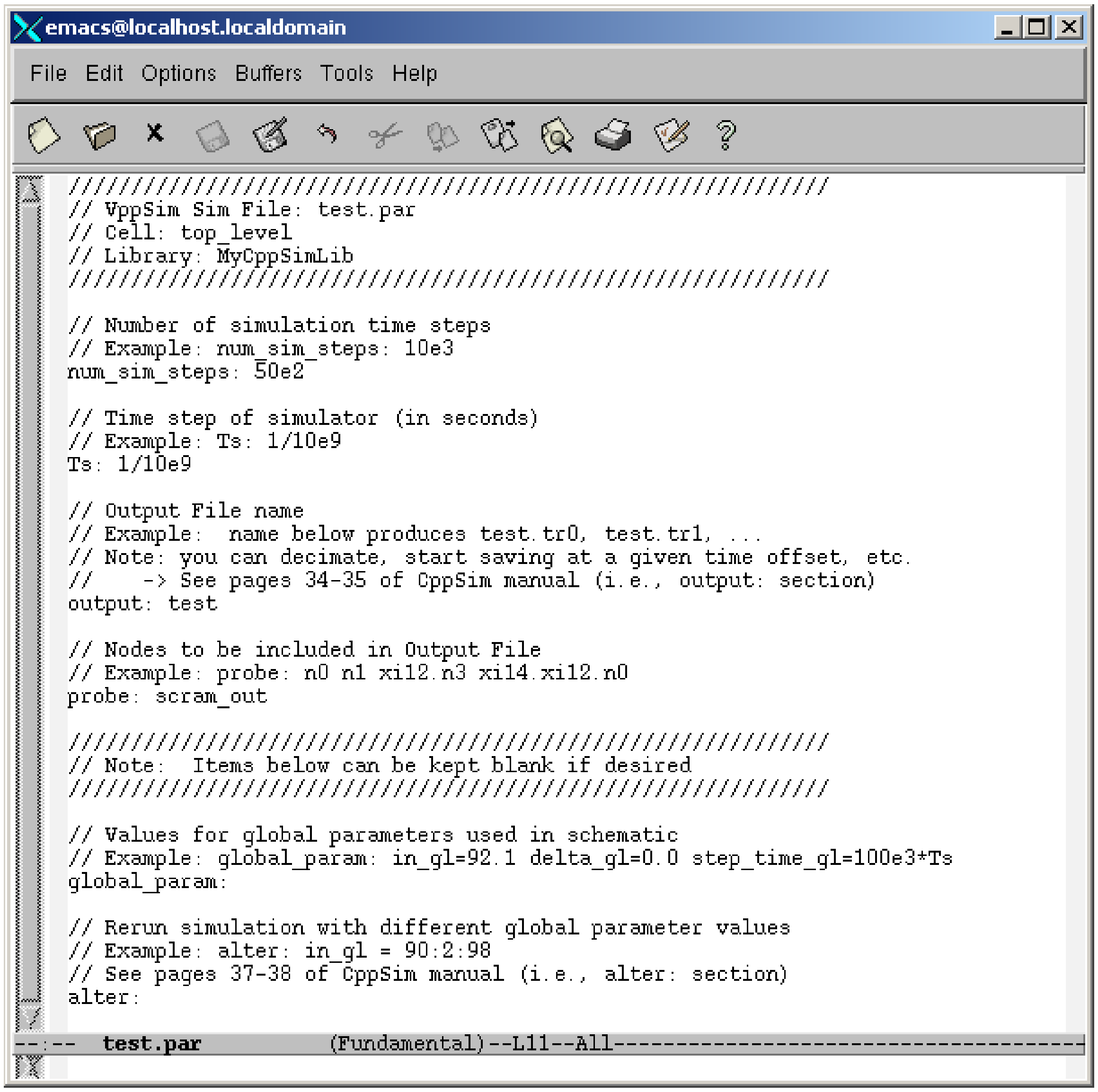

Virtuoso® Schematic Editing: MyCppSimLib top_level schematic
Cmd:
Sel: 0
15
Tools Design Window Edit Add Check Sheet Options Migrate Design Manager SiTime Help
constant
out
val=0
vco
I0
vctrl
squareout
sineout
I1
fc=200e6 Hz
Kvco=1 Hz/V
I2
clk
source
out<25:0>
I3
clk
input_data<25:0>
scrambler
output_data<25:0>
scram_out<25:0>
mouse L: mouseSubSelectPt
M:
R:
>

Figure 3. Schematic view of the *top_level* cell

Save the *top_level* schematic view that you have created. The next step is to run the CppSim simulation. For this step, in the schematic editor window for the *top_level* cell select Options → VppSim. This should open the VppSim GUI. Click on the *CppSim* radio button within the VppSim GUI. Push the *Netlist Only* button. Push the *Edit Sim File* button VppSim GUI window. A text editor as shown in Figure 4 should appear. Set the *num_sim_steps* to 50e2, *Ts* to 1/10e9 and *probe* to *scram_out*. Close the text editor once you have made the changes.


emacs@localhost.localdomain
File Edit Options Buffers Tools Help

```
//////////////////////////////////////////////////////////////
// VppSim Sim File: test.par
// Cell: top_level
// Library: MyCppSimLib
//////////////////////////////////////////////////////////////

// Number of simulation time steps
// Example: num_sim_steps: 10e3
num_sim_steps: 50e2

// Time step of simulator (in seconds)
// Example: Ts: 1/10e9
Ts: 1/10e9

// Output File name
// Example:  name below produces test.tr0, test.tr1, ...
// Note: you can decimate, start saving at a given time offset, etc.
//     -> See pages 34-35 of CppSim manual (i.e., output: section)
output: test

// Nodes to be included in Output File
// Example: probe: n0 n1 xi12.n3 xi14.xi12.n0
probe: scram_out

//////////////////////////////////////////////////////////////
// Note:  Items below can be kept blank if desired
//////////////////////////////////////////////////////////////

// Values for global parameters used in schematic
// Example: global_param: in_gl=92.1 delta_gl=0.0 step_time_gl=100e3*Ts
global_param:

// Rerun simulation with different global parameter values
// Example: alter: in_gl = 90:2:98
// See pages 37-38 of CppSim manual (i.e., alter: section)
alter:
--:--  test.par         (Fundamental)--L11--All------------------------------------
```

Figure 4. File to set the simulation parameters

Our next step is to compile the C++ code and simulate our system. Use the following commands at a Linux prompt to do this step

% cd VppSim/SimRuns/MyCppSimLib/top_level

```
% net2code –cpp
% make | grep "^[G]" > cpp_sim_out.txt
```

The above commands compile the C++ code and simulate our system. Open the *cpp_sim_out.txt* file to see the output of the system. It should show you the input and output of the scrambler during each clock cycle. Now that we have a working C++ model of our system, the next step is to replace the C++ model of the scrambler with the verilog model. The way in which the C++ implementations of the modules live in the *cppsim* subdirectory, the Verilog implementations will live in the *verilog* subdirectory of each module. You will have to create a *verilog* directory. You can do so by either creating the verilog cell view from Cadence, or use the following Linux commands to do this

```
% cd VppSim/cds/MyCppSimLib/scrambler
% mkdir verilog
% cd verilog
```

In this directory, open your favorite text editor and copy the code below in that editor. Save the file as *verilog.v*. This file contains the verilog implementation of the scrambler unit.

```
module scrambler (clk, input_data, output_data);

    input clk;
    input [25:0] input_data;

    output [25:0] output_data;

    reg [25:0] output_data;

    reg [6:0] scram_seed;
    reg new_msg;
    reg [25:0] scram_seq;
    reg [25:0] output_buffer;

    integer i;
    integer j;

        initial
                begin
                        scram_seed = 7'b0110110;
                end

        always @(input_data) begin
                new_msg = input_data[0] | input_data[1];
        end

        always @(posedge clk) begin
                output_data = output_buffer;
        end

        always @ (posedge clk) begin
```

```
                $display("G scrambler in(%b%b %b%b%b%b%b%b%b%b%b%b%b%b%b%b%b%b%b%b%b%b%b%b%b%b)",input_data[0],input_data[1],input_data[2],input_data[3],input_data[4],input_data[5],input_data[6],input_data[7],input_data[8],input_data[9],input_data[10],input_data[11],input_data[12],input_data[13],input_data[14],input_data[15],input_data[16],input_data[17],input_data[18],input_data[19],input_data[20],input_data[21],input_data[22],input_data[23],input_data[24],input_data[25]);

        $display("out(%b%b  %b%b%b%b%b%b%b%b%b%b%b%b%b%b%b%b%b%b%b%b%b%b%b%b)",output_data[0],output_data[1],output_data[2],output_data[3],output_data[4],output_data[5],output_data[6],output_data[7],output_data[8],output_data[9],output_data[10],output_data[11],output_data[12],output_data[13],output_data[14],output_data[15],output_data[16],output_data[17],output_data[18],output_data[19],output_data[20],output_data[21],output_data[22],output_data[23],output_data[24],output_data[25]);

                // process input
                output_buffer[0] = input_data[0];
                output_buffer[1] = input_data[1];

                // re-initialize scramble sequence if necessary
                if (new_msg == 1'b1) begin
                        for (i=0;i<7;i=i+1) begin
                                scram_seq[i]  = scram_seed[i];
                        end
                        for (i=7;i<24;i=i+1) begin
                                scram_seq[i] = scram_seq[i-4] ^ scram_seq[i-7];
                        end
                end

                // Scramble and latch input
                for (i=0;i<24;i=i+1) begin
                        output_buffer[i+2] = input_data[i+2] ^ scram_seq[i];
                end

                // create new scramble sequence
                for (i = 0; i < 4; i=i+1) begin
                        scram_seq[i] = scram_seq[24 + i - 4] ^ scram_seq[24 + i - 7];
                end
                for (i = 4; i < 7; i=i+1) begin
                        scram_seq[i] = scram_seq[i - 4] ^ scram_seq[24 + i - 7];
                end
                for (i = 7; i < 24; i=i+1) begin
                        scram_seq[i] = scram_seq[i - 4] ^ scram_seq[i - 7];
                end
        end
endmodule
```

To run the simulation using our newly created scrambler implementation, first cd into the directory where we make and run the simulator (*VppSim/SimRuns/MyCppSimLib/top_level*) and issue the command *net2code –vpp* (note the change from -*cpp* to -*vpp*). Note that simulating Verilog modules takes longer than simulating C++ modules, hence we should run the simulator on smaller amounts of data. The value for *num_sim_steps* in the *test.par* file in the current folder can be edited to do this. The Linux commands to do the steps described above are below

```
% cd VppSim/SimRuns/MyCppSimLib/top_level
% net2code -vpp
```

Now we need to setup the CppSim tool such that it will use the verilog implementation for the scrambler while running the simulation. To do this, within the same directory (*VppSim/SimRuns/MyCppSimLib/top_level*), edit the file *test.hier_v*. In this file you will see the names of all the modules you are simulating preceded by either a “c” or a “v”. Any module with its name preceded by a “c” will be simulated using its C++ implementation and any module preceded by a “v” will be simulated using its Verilog implementation. Find the *scrambler* module and switch it to “v” (note the words cppsim and verilog in parentheses next to the module’s name; this means that both implementations are available for this module). Save the file, return to a command prompt in the simulation directory, run *net2code -vpp* again and *make* to build and run the simulator.

```
% cd VppSim/SimRuns/MyCppSimLib/top_level
% net2code –vpp
% make | grep “^[G]” > veri_sim_out.txt
```

Note that this simulation takes longer than the simulation using C++ models. Open the *veri_sim_output.txt* file to check the input and output of the scrambler in every clock cycle. Compare this file with the output obtained using the C++ models (*cpp_sim_output.txt*). Note any differences in the outputs. The only difference should be that the C++ models output 0’s during the initial setup cycles, while the verilog models output X’s (don’t care).