**SYSTEM VERILOG LAB**

**1. DECADE COUNTER**

// Code your design here

module decade\_counter(f,n,clk);

input [3:0]n;

input clk;

output reg [3:0]f;

initial f=4'b0000;

always@(posedge clk)

begin

f=(f==n-1'b1)?4'b0000:(f+1'b1);

end

endmodule

// Code your testbench here

// or browse Examples

module testbench\_decade\_counter;

reg [3:0]n;

reg clk;

wire [3:0]f;

decade\_counter g1(f,n,clk);

initial begin

clk=0;

n=4'b1010;

end

always #2clk=~clk;

initial $monitor($time," f=%b, clk=%b",f,clk);

initial #80 $stop;

initial begin

$dumpvars;

$dumpfile("dump.vcd");

end

endmodule

**2.101 Sequence Detector Mealy Machine**

module sequence\_detector(q,a,clk,rst);

output reg q;

input clk,rst,a;

parameter s0=2'b00;

parameter s1=2'b01;

parameter s2=2'b10;

reg [1:0]present\_state;

initial present\_state=s0;

always@(posedge clk)

begin

if(rst)

begin present\_state=s0; q=0;end

else

case(present\_state)

s0: begin present\_state=a?s1:s0; q=0; end

s1: begin present\_state=(~a)?s2:s1; q=0; end

s2: begin present\_state=a?s1:s0; q=a?1:0; end

endcase

end

endmodule

module sequence\_detector\_tb;

wire q;

reg a,clk,rst;

sequence\_detector g1(q,a,clk,rst);

initial begin

clk=0;

rst=1;

a=0;

#4 rst=0;

#4 a=1;

#4 a=0;

#4 a=1;

#4 a=0;

#4 a=1;

end

always #2 clk=~clk;

initial $monitor($time," clk=%b, rst=%b, a=%b, q=%b",clk,rst,a,q);

initial #30 $stop;

initial begin

$dumpvars;

$dumpfile("dump.vcd");

end

endmodule

**3. FIFO-First in First Out**

module fifobuffer(dataout, full, empty, datain, clk,rst, rd, wr);

output [7:0] dataout;

output full;

output empty;

input [7:0] datain;

input clk;

input rst;

input rd;

input wr;

reg [7:0]dataout;

reg [7:0]fifo[7:0];

reg [2:0]readcounter = 0;

reg [2:0]writecounter = 0;

// assign empty =((readcounter==0) && (writecounter==0) )?1'b1:1'b0;

assign empty=((readcounter==writecounter)?1'b1:1'b0);

assign full =((writecounter==7) && (readcounter==0))?1'b1:1'b0;

always@(posedge clk)

begin

if(rst)

begin

readcounter<=0;

writecounter<=0;

end

else

begin

if(rd==1'b1 && (!empty) )

begin

dataout<=fifo[readcounter];

readcounter<=readcounter+1;

end

else if(wr==1'b1 && (!full))

begin

fifo[writecounter]<=datain;

writecounter<=writecounter+1;

// $display("write count=%d",writecounter);

end

end

end

endmodule

module fifobuffer\_tb\_v;

reg [7:0] datain;

reg clk;

reg rst;

reg rd;

reg wr;

wire [7:0] dataout;

wire full;

wire empty;

initial

begin

datain = 8'h00;

rd = 1'b0;

wr = 1'b0;

rst = 1'b1;

clk = 1'b0;

forever #2 clk=~clk;

end

initial

begin

@ (posedge clk)

rst = 1'b0;

wr = 1'b1;

@ (posedge clk)

datain = 8'h11;

@ (posedge clk)

datain = 8'h22;

@ (posedge clk)

datain = 8'h45;

@ (posedge clk)

datain = 8'h23;

@ (posedge clk)

datain = 8'h17;

@ (posedge clk)

datain = 8'h66;

@ (posedge clk)

wr = 1'b0;

rd=1;

end

//always #2 clk =~clk;

initial $monitor($time, "dataout=%b, full =%b,empty = %b, datain= %b,clk=%b,rst=%b,rd=%b,wr=%b",dataout,full,empty,datain,clk,rst,rd,wr);

initial #60 $stop;

initial begin

$dumpvars;

$dumpfile("test.vcd");

end

fifobuffer fbuf(dataout,full,empty,datain,clk,rst,rd,wr);

endmodule

**4. Semaphore:**

// Code your testbench here

// or browse Examples

module top();

semaphore sema = new(1); // Create semaphore with 1 key.

initial begin

repeat(3) begin

fork

////////// PROCESS 1 ////////////////

begin

$display("1: Waiting for key @time=%0t", $time);

sema.get(1);

$display("1: Got the Key @time=%0t", $time);

#(10);// Do some work

sema.put(1);

$display("1: Returning back key @time=%0t", $time);

#(10);

end

////////// PROCESS 2 ////////////////

begin

#1;

$display("2: Waiting for Key @time=%0t", $time);

sema.get(1);

$display("2: Got the Key @time=%0t", $time);

#(10);//Do some work

sema.put(1);

$display("2: Returning back key @time=%0t", $time);

#(10);

end

join

$display();

end

#1000;

end

endmodule : top

**5. Polymorphism**

// base class

class base\_class;

virtual function void display();

$display("Inside base class");

endfunction

endclass

// extended class 1

class ext\_class\_1 extends base\_class;

function void display();

$display("Inside extended class 1");

endfunction

endclass

// extended class 2

class ext\_class\_2 extends base\_class;

function void display();

$display("Inside extended class 2");

endfunction

endclass

// extended class 3

class ext\_class\_3 extends base\_class;

function void display();

$display("Inside extended class 3");

endfunction

endclass

// module

module class\_polymorphism;

initial begin

//declare and create extended class

ext\_class\_1 ec\_1 = new();

ext\_class\_2 ec\_2 = new();

ext\_class\_3 ec\_3 = new();

// base\_class bc=new();

// bc.display();

// base class handle

base\_class b\_c[3];

//assigning extended class to base class

b\_c[0] = ec\_1;

b\_c[1] = ec\_2;

b\_c[2] = ec\_3;

//accessing extended class methods using base class handle

b\_c[0].display();

b\_c[1].display();

b\_c[2].display();

end

endmodule

**7. FUNCTIONAL COVERAGE**

// Code your design here

module ALU (input wire [3:0] a,

input wire [3:0] b,

input wire [2:0] sel,

output logic [7:0] out);

always @ (\*)

begin

case (sel)

3'b000 : out <= a + b;

3'b001 : out <= a - b;

3'b010 : out <= a \* b;

3'b011 : out <= a & b;

3'b100 : out <= a ^ b;

3'b101 : out <= !a&b;

3'b110 : out <= a + !b;

default: out <= 0;

endcase

end

endmodule: ALU

// Code your testbench here

// or browse Examples

module tb;

logic [3:0] AA;

logic [3:0] BB;

logic [2:0] selector;

wire [7:0] out\_tb;

// logic clk;

ALU A1 ( .a (AA),

.b (BB),

.sel (selector),

.out(out\_tb)

);

covergroup cg;

operations: coverpoint selector{

bins algebra = {[0:2]};

bins logical = {[3:7]};

}

size\_A: coverpoint AA {

bins min\_A = {[0:4]};

bins med\_A = {[5:10]};

bins max\_A = {[11:15]};

}

size\_B: coverpoint BB {

bins min\_B = {[0:4]};

bins med\_B = {[5:10]};

bins max\_B = {[11:15]};

}

endgroup

cg c1;

task t1;

$display($time,"A : %d B : %d Sel: %d OUTPUT: %d",AA,BB,selector,out\_tb);

endtask

initial begin

c1 = new();

#1 selector <= 0; AA <= 0; BB <= 0;

#1 t1; c1.sample();

#1 selector <= 0; AA <= 'b1; BB <= 0;

#1 t1;c1.sample();

#1 selector <= 0; AA <= 'd10; BB <= 'd4;

#1 t1;c1.sample();

#1 selector <= 0; AA <= 'd11; BB <= 'd3;

#1 t1;c1.sample();

#1 selector <= 0; AA <= 'd14; BB <= 'd14;

#1 t1;c1.sample();

#1 selector <= 1; AA <= 0; BB <= 0;

#1 t1;c1.sample();

#1 selector <= 1; AA <= 'b1; BB <= 'd8;

#1 t1;c1.sample();

#1 selector <= 1; AA <= 'd10; BB <= 'd4;

#1 t1;c1.sample();

#1 selector <= 1; AA <= 'd11; BB <= 'd3;

#1 t1; c1.sample();

#1 selector <= 1; AA <= 'd14; BB <= 'd14;

#1 t1;c1.sample();

#1 selector <= 'd2; AA <= 0; BB <= 0;

#1 t1;c1.sample();

#1 selector <= 'd2; AA <= 'b1; BB <= 11;

#1 t1;c1.sample();

#1 selector <= 'd2; AA <= 'd10; BB <= 'd4;

#1 t1;c1.sample();

#1 selector <= 'd2; AA <= 'd11; BB <= 11;

#1 t1;c1.sample();

#1 selector <= 'd2; AA <= 'd14; BB <= 11;

#1 t1;c1.sample();

#1 selector <= 'd5; AA <= 0; BB <= 0;

#1 t1;c1.sample();

#1 selector <= 'd5; AA <= 'b1; BB <= 'd8;

#1 t1;c1.sample();

#1 selector <= 'd5; AA <= 'd10; BB <= 'd4;

#1 t1;c1.sample();

#1 selector <= 'd5; AA <= 'd11; BB <= 'd3;

#1 t1;c1.sample();

#1 selector <= 'd5; AA <= 'd14; BB <= 'd14;

#1 t1;c1.sample();

#10 $finish;

end

initial begin

$dumpvars(1);

$dumpfile("aludump.vcd");

end

initial begin

#50 $display ("Coverage = %0.2f %%", c1.get\_inst\_coverage());

end

endmodule

**8. FULL ADDER for monitor and scoreboard**

// Code your design here

module my\_adder (adder\_if \_if);

always @(posedge \_if.tb\_clk); begin

always\_comb begin

if (\_if.rstn) begin

\_if.sum<= 0;

\_if.carry<= 0;

end else begin

{\_if.carry, \_if.sum} <= \_if.a + \_if.b;

end

end

end

endmodule

// Code your testbench here

// or browse Examples

//Interface

// Adder interface contains all signals that the adder requires

// to operate

interface adder\_if();

logic rstn;

logic [7:0] a;

logic [7:0] b;

logic [7:0] sum;

logic carry;

logic tb\_clk;

initial tb\_clk<= 0;

always #10 tb\_clk = ~tb\_clk;

endinterface

// Although an adder does not have a clock, let us create a mock clock

// used in the testbench to synchronize when value is driven and when

// value is sampled. Typically combinational logic is used between

// sequential elements like FF in a real circuit. So, let us assume

// that inputs to the adder is provided at some posedge clock. But because

// the design does not have clock in its input, we will keep this clock

// in a separate interface that is available only to testbench components

interface clk\_if();

logic tb\_clk;

initial tb\_clk<= 0;

always #10 tb\_clk = ~tb\_clk;

endinterface

// To verify that the adder adds, we also need to check that it

// does not add when rstn is 0, and hence rstn should also be

// randomized along with a and b.

class Packet;

rand bit rstn;

rand bit[7:0] a;

rand bit[7:0] b;

bit [7:0] sum;

bit carry;

// Print contents of the data packet

function void print(string tag="");

$display ("T=%0t %S a=0x%0h b=0x%0h sum=0x%0h carry=0x%0h", $time, tag, a, b, sum, carry);

endfunction

// This is a utility function to allow copying contents in

// one Packet variable to another.

function void copy(Packet tmp);

this.a = tmp.a;

this.b = tmp.b;

this.rstn = tmp.rstn;

this.sum = tmp.sum;

this.carry = tmp.carry;

endfunction

endclass

//Generator

// Sometimes we simply need to generate N random transactions to random

// locations so a generator would be useful to do just that. In this case

// loop determines how many transactions need to be sent

class generator;

int loop = 10;

event drv\_done;

mailbox drv\_mbx;

task run();

for (int i = 0; i< loop; i++) begin

Packet item = new;

item.randomize();

$display ("T=%0t [Generator] Loop:%0d/%0d create next item", $time, i+1, loop);

drv\_mbx.put(item);

$display ("T=%0t [Generator] Wait for driver to be done", $time);

@(drv\_done);

end

endtask

endclass

//Driver

class driver;

virtual adder\_if m\_adder\_vif;

virtual clk\_if m\_clk\_vif;

event drv\_done;

mailbox drv\_mbx;

task run();

$display ("T=%0t [Driver] starting ...", $time);

// Try to get a new transaction every time and then assign

// packet contents to the interface. But do this only if the

// design is ready to accept new transactions

forever begin

Packet item;

$display ("T=%0t [Driver] waiting for item ...", $time);

drv\_mbx.get(item);

@ (posedge m\_clk\_vif.tb\_clk);

item.print("Driver");

m\_adder\_vif.rstn<= item.rstn;

m\_adder\_vif.a<= item.a;

m\_adder\_vif.b<= item.b;

->drv\_done;

end

endtask

endclass

//Monitor

// The monitor has a virtual interface handle with which it can monitor

// the events happening on the interface. It sees new transactions and then

// captures information into a packet and sends it to the scoreboard

// using another mailbox.

class monitor;

virtual adder\_if m\_adder\_vif;

virtual clk\_if m\_clk\_vif;

mailbox scb\_mbx; // Mailbox connected to scoreboard

task run();

$display ("T=%0t [Monitor] starting ...", $time);

// Check forever at every clock edge to see if there is a

// valid transaction and if yes, capture info into a class

// object and send it to the scoreboard when the transaction

// is over.

forever begin

Packet m\_pkt = new();

@(posedge m\_clk\_vif.tb\_clk);

#1;

m\_pkt.a = m\_adder\_vif.a;

m\_pkt.b = m\_adder\_vif.b;

m\_pkt.rstn = m\_adder\_vif.rstn;

m\_pkt.sum = m\_adder\_vif.sum;

m\_pkt.carry = m\_adder\_vif.carry;

m\_pkt.print("Monitor");

scb\_mbx.put(m\_pkt);

end

endtask

endclass

//Scoreboard

// The scoreboard is responsible to check data integrity. Since the design

// simple adds inputs to give sum and carry, scoreboard helps to check if the

// output has changed for given set of inputs based on expected logic

class scoreboard;

mailbox scb\_mbx;

task run();

forever begin

Packet item, ref\_item;

scb\_mbx.get(item);

item.print("Scoreboard");

// Copy contents from received packet into a new packet so

// just to get a and b.

ref\_item = new();

ref\_item.copy(item);

// Let us calculate the expected values in carry and sum

if (ref\_item.rstn)

{ref\_item.carry, ref\_item.sum} = ref\_item.a + ref\_item.b;

else

{ref\_item.carry, ref\_item.sum} = 0;

// Now, carry and sum outputs in the reference variable can be compared

// with those in the received packet

if (ref\_item.carry != item.carry) begin

$display("[%0t] Scoreboard Error! Carry mismatch ref\_item=0x%0h item=0x%0h", $time, ref\_item.carry, item.carry);

end else begin

$display("[%0t] Scoreboard Pass! Carry match ref\_item=0x%0h item=0x%0h", $time, ref\_item.carry, item.carry);

end

if (ref\_item.sum != item.sum) begin

$display("[%0t] Scoreboard Error! Sum mismatch ref\_item=0x%0h item=0x%0h", $time, ref\_item.sum, item.sum);

end else begin

$display("[%0t] Scoreboard Pass! Sum match ref\_item=0x%0h item=0x%0h", $time, ref\_item.sum, item.sum);

end

end

endtask

endclass

//Environment

// Lets say that the environment class was already there, and generator is

// a new component that needs to be included in the ENV.

class env;

generator g0; // Generate transactions

driver d0; // Driver to design

monitor m0; // Monitor from design

scoreboard s0; // Scoreboard connected to monitor

mailbox scb\_mbx; // Top level mailbox for SCB <-> MON

virtual adder\_if m\_adder\_vif; // Virtual interface handle

virtual clk\_if m\_clk\_vif; // TB clk

event drv\_done;

mailbox drv\_mbx;

function new();

d0 = new;

m0 = new;

s0 = new;

scb\_mbx = new();

g0 = new;

drv\_mbx = new;

endfunction

virtual task run();

// Connect virtual interface handles

d0.m\_adder\_vif = m\_adder\_vif;

m0.m\_adder\_vif = m\_adder\_vif;

d0.m\_clk\_vif = m\_clk\_vif;

m0.m\_clk\_vif = m\_clk\_vif;

// Connect mailboxes between each component

d0.drv\_mbx = drv\_mbx;

g0.drv\_mbx = drv\_mbx;

m0.scb\_mbx = scb\_mbx;

s0.scb\_mbx = scb\_mbx;

// Connect event handles

d0.drv\_done = drv\_done;

g0.drv\_done = drv\_done;

// Start all components - a fork join\_any is used because

// the stimulus is generated by the generator and we want the

// simulation to exit only when the generator has finished

// creating all transactions. Until then all other components

// have to run in the background.

fork

s0.run();

d0.run();

m0.run();

g0.run();

join\_any

endtask

endclass

//Test

// The test can instantiate any environment. In this test, we are using

// an environment without the generator and hence the stimulus should be

// written in the test.

class test;

env e0;

mailbox drv\_mbx;

function new();

drv\_mbx = new();

e0 = new();

endfunction

virtual task run();

e0.d0.drv\_mbx = drv\_mbx;

e0.run();

endtask

endclass

//Testbench Top

module tb;

clk\_if m\_clk\_if ();

adder\_if m\_adder\_if ();

my\_adder u0(m\_adder\_if);

initial begin

test t0;

t0 = new;

t0.e0.m\_adder\_vif = m\_adder\_if;

t0.e0.m\_clk\_vif = m\_clk\_if;

t0.run();

// Once the main stimulus is over, wait for some time

// until all transactions are finished and then end

// simulation. Note that $finish is required because

// there are components that are running forever in

// the background like clk, monitor, driver, etc

#50 $finish;

end

initial begin

$dumpvars;

$dumpfile ("dump.vcd");

end

endmodule

**9. FULL ADDER for packet class,mailbox, generator and driver**

// Code your design here

module my\_adder (adder\_if \_if);

always @(posedge \_if.tb\_clk); begin

always\_comb begin

if (\_if.rstn) begin

\_if.sum<= 0;

\_if.carry<= 0;

end else begin

{\_if.carry, \_if.sum} <= \_if.a + \_if.b;

end

end

end

endmodule

// Code your testbench here

// or browse Examples

//Interface

// Adder interface contains all signals that the adder requires

// to operate

interface adder\_if();

logic rstn;

logic [7:0] a;

logic [7:0] b;

logic [7:0] sum;

logic carry;

logic tb\_clk;

initial tb\_clk<= 0;

always #10 tb\_clk = ~tb\_clk;

endinterface

// Although an adder does not have a clock, let us create a mock clock

// used in the testbench to synchronize when value is driven and when

// value is sampled. Typically combinational logic is used between

// sequential elements like FF in a real circuit. So, let us assume

// that inputs to the adder is provided at some posedge clock. But because

// the design does not have clock in its input, we will keep this clock

// in a separate interface that is available only to testbench components

interface clk\_if();

logic tb\_clk;

initial tb\_clk<= 0;

always #10 tb\_clk = ~tb\_clk;

endinterface

// To verify that the adder adds, we also need to check that it

// does not add when rstn is 0, and hence rstn should also be

// randomized along with a and b.

class Packet;

rand bit rstn;

rand bit[7:0] a;

rand bit[7:0] b;

bit [7:0] sum;

bit carry;

// Print contents of the data packet

function void print(string tag="");

$display ("T=%0t %s a=0x%0h b=0x%0h sum=0x%0h carry=0x%0h", $time, tag, a, b, sum, carry);

endfunction

// This is a utility function to allow copying contents in

// one Packet variable to another.

function void copy(Packet tmp);

this.a = tmp.a;

this.b = tmp.b;

this.rstn = tmp.rstn;

this.sum = tmp.sum;

this.carry = tmp.carry;

endfunction

endclass

//Generator

// Sometimes we simply need to generate N random transactions to random

// locations so a generator would be useful to do just that. In this case

// loop determines how many transactions need to be sent

class generator;

int loop = 10;

event drv\_done;

mailbox drv\_mbx;

task run();

for (int i = 0; i< loop; i++) begin

Packet item = new;

item.randomize();

$display ("T=%0t [Generator] Loop:%0d/%0d create next item", $time, i+1, loop);

drv\_mbx.put(item);

$display ("T=%0t [Generator] Wait for driver to be done", $time);

@(drv\_done);

end

endtask

endclass

//Driver

class driver;

virtual adder\_if m\_adder\_vif;

virtual clk\_if m\_clk\_vif;

event drv\_done;

mailbox drv\_mbx;

task run();

$display ("T=%0t [Driver] starting ...", $time);

// Try to get a new transaction every time and then assign

// packet contents to the interface. But do this only if the

// design is ready to accept new transactions

forever begin

Packet item;

$display ("T=%0t [Driver] waiting for item ...", $time);

drv\_mbx.get(item);

@ (posedge m\_clk\_vif.tb\_clk);

item.print("Driver");

m\_adder\_vif.rstn<= item.rstn;

m\_adder\_vif.a<= item.a;

m\_adder\_vif.b<= item.b;

->drv\_done;

end

endtask

endclass

//Monitor

// The monitor has a virtual interface handle with which it can monitor

// the events happening on the interface. It sees new transactions and then

// captures information into a packet and sends it to the scoreboard

// using another mailbox.

// class monitor;

// virtual adder\_if m\_adder\_vif;

// virtual clk\_if m\_clk\_vif;

// mailbox scb\_mbx; // Mailbox connected to scoreboard

// task run();

// $display ("T=%0t [Monitor] starting ...", $time);

// // Check forever at every clock edge to see if there is a

// // valid transaction and if yes, capture info into a class

// // object and send it to the scoreboard when the transaction

// // is over.

// forever begin

// Packet m\_pkt = new();

// @(posedge m\_clk\_vif.tb\_clk);

// #1;

// m\_pkt.a = m\_adder\_vif.a;

// m\_pkt.b = m\_adder\_vif.b;

// m\_pkt.rstn = m\_adder\_vif.rstn;

// m\_pkt.sum = m\_adder\_vif.sum;

// m\_pkt.carry = m\_adder\_vif.carry;

// m\_pkt.print("Monitor");

// scb\_mbx.put(m\_pkt);

// end

// endtask

// endclass

// Scoreboard

// The scoreboard is responsible to check data integrity. Since the design

// simple adds inputs to give sum and carry, scoreboard helps to check if the

// output has changed for given set of inputs based on expected logic

// class scoreboard;

// mailbox scb\_mbx;

// task run();

// forever begin

// Packet item, ref\_item;

// scb\_mbx.get(item);

// item.print("Scoreboard");

// // Copy contents from received packet into a new packet so

// // just to get a and b.

// ref\_item = new();

// ref\_item.copy(item);

// // Let us calculate the expected values in carry and sum

// if (ref\_item.rstn)

// {ref\_item.carry, ref\_item.sum} = ref\_item.a + ref\_item.b;

// else

// {ref\_item.carry, ref\_item.sum} = 0;

// // Now, carry and sum outputs in the reference variable can be compared

// // with those in the received packet

// if (ref\_item.carry != item.carry) begin

// $display("[%0t] Scoreboard Error! Carry mismatch ref\_item=0x%0h item=0x%0h", $time, ref\_item.carry, item.carry);

// end else begin

// $display("[%0t] Scoreboard Pass! Carry match ref\_item=0x%0h item=0x%0h", $time, ref\_item.carry, item.carry);

// end

// if (ref\_item.sum != item.sum) begin

// $display("[%0t] Scoreboard Error! Sum mismatch ref\_item=0x%0h item=0x%0h", $time, ref\_item.sum, item.sum);

// end else begin

// $display("[%0t] Scoreboard Pass! Sum match ref\_item=0x%0h item=0x%0h", $time, ref\_item.sum, item.sum);

// end

// end

// endtask

// endclass

//Environment

// Lets say that the environment class was already there, and generator is

// a new component that needs to be included in the ENV.

class env;

generator g0; // Generate transactions

driver d0; // Driver to design

// monitor m0; // Monitor from design

// scoreboard s0; // Scoreboard connected to monitor

mailbox scb\_mbx; // Top level mailbox for SCB <-> MON

virtual adder\_if m\_adder\_vif; // Virtual interface handle

virtual clk\_if m\_clk\_vif; // TB clk

event drv\_done;

mailbox drv\_mbx;

function new();

d0 = new;

// m0 = new;

// s0 = new;

scb\_mbx = new();

g0 = new;

drv\_mbx = new;

endfunction

virtual task run();

// Connect virtual interface handles

d0.m\_adder\_vif = m\_adder\_vif;

// m0.m\_adder\_vif = m\_adder\_vif;

d0.m\_clk\_vif = m\_clk\_vif;

// m0.m\_clk\_vif = m\_clk\_vif;

// Connect mailboxes between each component

d0.drv\_mbx = drv\_mbx;

g0.drv\_mbx = drv\_mbx;

// m0.scb\_mbx = scb\_mbx;

// s0.scb\_mbx = scb\_mbx;

// Connect event handles

d0.drv\_done = drv\_done;

g0.drv\_done = drv\_done;

// Start all components - a fork join\_any is used because

// the stimulus is generated by the generator and we want the

// simulation to exit only when the generator has finished

// creating all transactions. Until then all other components

// have to run in the background.

fork

// s0.run();

d0.run();

// m0.run();

g0.run();

join\_any

endtask

endclass

//Test

// The test can instantiate any environment. In this test, we are using

// an environment without the generator and hence the stimulus should be

// written in the test.

class test;

env e0;

mailbox drv\_mbx;

function new();

drv\_mbx = new();

e0 = new();

endfunction

virtual task run();

e0.d0.drv\_mbx = drv\_mbx;

e0.run();

endtask

endclass

//Testbench Top

module tb;

clk\_if m\_clk\_if ();

adder\_if m\_adder\_if ();

my\_adder u0(m\_adder\_if);

initial begin

test t0;

t0 = new;

t0.e0.m\_adder\_vif = m\_adder\_if;

t0.e0.m\_clk\_vif = m\_clk\_if;

t0.run();

// Once the main stimulus is over, wait for some time

// until all transactions are finished and then end

// simulation. Note that $finish is required because

// there are components that are running forever in

// the background like clk, monitor, driver, etc

#50 $finish;

end

initial begin

$dumpvars;

$dumpfile ("dump.vcd");

end

endmodule