SYSTEM VERILOG TESTBENCH

EXAMPLE 2

JAIRAJ MIRASHI
DESIGN VERIFICATION ENGINEER

SystemVerilog Testbench Example 2

Example of a SystemVerilog testbench using OOP concepts like inheritance, polymorphism to build a functional testbench for a simple design.

Design

This Verilog module named "switch" is designed to handle two sets of address and data pairs based on a specified condition. Here's a brief explanation of the design:

- Parameters:
  - `ADDR_WIDTH`: Specifies the width of the address bus.
  - `DATA_WIDTH`: Specifies the width of the data bus.
  - `ADDR_DIV`: Represents a dividing value for determining the condition.

- Inputs:
  - `clk`: Clock input.
  - `rstn`: Active-low asynchronous reset.
  - `vld`: Valid signal indicating a valid input condition.
  - `addr`: Address input.
  - `data`: Data input.

- Outputs:
  - `addr_a`: Address output for condition met.
  - `data_a`: Data output for condition met.
  - `addr_b`: Address output for condition not met.
  - `data_b`: Data output for condition not met.

- Behavior:
  - On the positive edge of the clock (`posedge clk`), the module processes the input data.
  - During a reset condition (`!rstn`), all outputs are set to zero.
  - If the valid signal (`vld`) is active, the module checks whether the address is within the specified range (`0` to `ADDR_DIV`).
    - If true, it assigns the input address and data to `addr_a` and `data_a`, respectively. `addr_b` and `data_b` are set to zero.
    - If false, it assigns the input address and data to `addr_b` and `data_b`, respectively. `addr_a` and `data_a` are set to zero.

This design effectively separates input data into two sets based on the condition specified by the dividing value `ADDR_DIV`. The outputs `addr_a`, `data_a`, `addr_b`, and `data_b` represent the results of this conditional separation.

```
module switch #
(
  parameter ADDR_WIDTH = 8,
  parameter DATA_WIDTH = 16,
  parameter ADDR_DIV = 8'h3F
)
(
  input clk,
  input rstn,
  input vld,
  input [ADDR_WIDTH-1:0] addr,
  input [DATA_WIDTH-1:0] data,
  output reg [ADDR_WIDTH-1:0] addr_a,
  output reg [DATA_WIDTH-1:0] data_a,
  output reg [ADDR_WIDTH-1:0] addr_b,
  output reg [DATA_WIDTH-1:0] data_b
);
  always @(posedge clk) begin
    if (!rstn) begin
      addr_a <= 0;
      data_a <= 0;
      addr_b <= 0;
      data_b <= 0;
    end
    else begin
      if (vld) begin
        if (addr >= 0 && addr <= ADDR_DIV) begin
          addr_a <= addr;
          data_a <= data;
          addr_b <= 0;
          data_b <= 0;
        end
        else begin
          addr_a <= 0;
          data_a <= 0;
          addr_b <= addr;
          data_b <= data;
        end
      end
    end
  end
endmodule
```

Interface

```
// Design interface used to monitor activity and capture/drive
// transactions
interface switch_if (
  input bit clk
);
  logic rstn;
  logic vld;
  logic [7:0] addr;
  logic [15:0] data;
  logic [7:0] addr_a;
  logic [15:0] data_a;
  logic [7:0] addr_b;
  logic [15:0] data_b;
endinterface
```

Transaction Object

```
// This is the base transaction object that will be used
// in the environment to initiate new transactions and
// capture transactions at DUT interface
class switch_item;
  rand bit [7:0] addr;
  rand bit [15:0] data;
  bit [7:0] addr_a;
  bit [15:0] data_a;
  bit [7:0] addr_b;
  bit [15:0] data_b;

  // This function allows us to print contents of the data
  // packet so that it is easier to track in a logfile
  function void print(string tag = "");
    $display("T=%0t %s addr=0x%0h data=0x%0h addr_a=0x%0h data_a=0x%0h
addr_b=0x%0h data_b=0x%0h",
             $time, tag, addr, data, addr_a, data_a, addr_b, data_b);
  endfunction
endclass
```

Generator

```
// The generator class is used to generate a random
// number of transactions with random addresses and data
// that can be driven to the design
class generator;
  mailbox drv_mbx;
  event drv_done;
  int num = 20;

  task run();
    for (int i = 0; i < num; i++) begin
      switch_item item = new;
      item.randomize();
      $display("T=%0t [Generator] Loop:%0d/%0d create next item", $time, i +
1, num);
      drv_mbx.put(item);
      @(drv_done);
    end
    $display("T=%0t [Generator] Done generation of %0d items", $time, num);
  endtask
endclass
```

Driver

```
// The driver is responsible for driving transactions to the DUT
// All it does is to get a transaction from the mailbox if it is
// available and drive it out into the DUT interface.
class driver;
  virtual switch_if vif;
  event drv_done;
  mailbox drv_mbx;

  task run();
    $display("T=%0t [Driver] starting ...", $time);
    @(posedge vif.clk);

    // 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
      switch_item item;
      $display("T=%0t [Driver] waiting for item ...", $time);
      drv_mbx.get(item);
      item.print("Driver");
      vif.vld <= 1;
      vif.addr <= item.addr;
      vif.data <= item.data;
      // When transfer is over, raise the done event
      @(posedge vif.clk);
      vif.vld <= 0;
      ->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 switch_if vif;
  mailbox scb_mbx;
  semaphore sema4;

  function new();
    sema4 = new(1);
  endfunction

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

    // To get a pipeline effect of transfers, fork two threads
    // where each thread uses a semaphore for the address phase
    fork
      sample_port("Thread0");
      sample_port("Thread1");
    join
  endtask

  task sample_port(string tag = "");
    // This task monitors the interface for a complete
    // transaction and pushes into the mailbox when the
    // transaction is complete
    forever begin
      @(posedge vif.clk);
      if (vif.rstn && vif.vld) begin
        switch_item item = new;
        sema4.get();
        item.addr = vif.addr;
        item.data = vif.data;
        $display("T=%0t [Monitor] %s First part over", $time, tag);
        @(posedge vif.clk);
        sema4.put();
        item.addr_a = vif.addr_a;
        item.data_a = vif.data_a;
        item.addr_b = vif.addr_b;
        item.data_b = vif.data_b;
        $display("T=%0t [Monitor] %s Second part over", $time, tag);
        scb_mbx.put(item);
        item.print({"Monitor_", tag});
      end
    end
  endtask
endclass
```

Environment

```
// The environment is a container object simply to hold
// all verification components together. This environment can
// then be reused later and all components in it would be
// automatically connected and available for use
class env;
  driver d0;            // Driver handle
  monitor m0;           // Monitor handle
  generator g0;         // Generator Handle
  scoreboard s0;        // Scoreboard handle
  mailbox drv_mbx;      // Connect GEN -> DRV
  mailbox scb_mbx;      // Connect MON -> SCB
  event drv_done;        // Indicates when driver is done
  virtual switch_if vif; // Virtual interface handle

  function new();
    d0 = new;
    m0 = new;
    g0 = new;
    s0 = new;
    drv_mbx = new();
    scb_mbx = new();
    d0.drv_mbx = drv_mbx;
    g0.drv_mbx = drv_mbx;
    m0.scb_mbx = scb_mbx;
    s0.scb_mbx = scb_mbx;
    d0.drv_done = drv_done;
    g0.drv_done = drv_done;
  endfunction

  virtual task run();
    d0.vif = vif;
    m0.vif = vif;
    fork
      d0.run();
      m0.run();
      g0.run();
      s0.run();
    join_any
  endtask
endclass
```

Scoreboard

```
// The scoreboard is responsible to check data integrity. Since
// the design routes packets based on an address range, the
// scoreboard checks that the packet's address is within valid
// range.
class scoreboard;
  mailbox scb_mbx;

  task run();
    forever begin
      switch_item item;
      scb_mbx.get(item);

      if (item.addr inside {[0:'h3f]}) begin
        if (item.addr_a != item.addr || item.data_a != item.data)
          $display("T=%0t [Scoreboard] ERROR! Mismatch addr=0x%0h data=0x%0h
addr_a=0x%0h data_a=0x%0h", $time, item.addr, item.data, item.addr_a,
item.data_a);
        else
          $display("T=%0t [Scoreboard] PASS! Mismatch addr=0x%0h data=0x%0h
addr_a=0x%0h data_a=0x%0h", $time, item.addr, item.data, item.addr_a,
item.data_a);
      end
      else begin
        if (item.addr_b != item.addr || item.data_b != item.data)
          $display("T=%0t [Scoreboard] ERROR! Mismatch addr=0x%0h data=0x%0h
addr_b=0x%0h data_b=0x%0h", $time, item.addr, item.data, item.addr_b,
item.data_b);
        else
          $display("T=%0t [Scoreboard] PASS! Mismatch addr=0x%0h data=0x%0h
addr_b=0x%0h data_b=0x%0h", $time, item.addr, item.data, item.addr_b,
item.data_b);
      end
    end
  endtask
endclass
```

Test

```
// Test class instantiates the environment and starts it.
class test;
  env e0;

  function new();
    e0 = new;
  endfunction

  task run();
    e0.run();
  endtask
endclass
```

Testbench TOP

```
// Top level testbench module to instantiate design, interface
// start clocks and run the test
module tb;
  reg clk;
  always #10 clk =~clk;

  switch_if _if(clk);
  switch u0(
    .clk(clk),
    .rstn(_if.rstn),
    .addr(_if.addr),
    .data(_if.data),
    .vld(_if.vld),
    .addr_a(_if.addr_a),
    .data_a(_if.data_a),
    .addr_b(_if.addr_b),
    .data_b(_if.data_b)
  );

  test t0;
  initial begin
    {clk, _if.rstn} <= 0;
    // Apply reset and start stimulus
    #20 _if.rstn <= 1;
    t0 = new;
    t0.e0.vif = _if;
    t0.run();
    // Because multiple components and clock are running
    // in the background, we need to call $finish explicitly
    #50 $finish;
  end

  // System tasks to dump VCD waveform file
  initial begin
    $dumpvars;
    $dumpfile("dump.vcd");
  end
endmodule
```