Joshua Monson
ECEn 620 Assignment 6
10/13/2014

Overview

The purpose of this assignment was to create a test bench for the for an SRAM controller. Three tests were performed: 10 back-to-back writes; 10 back-to-back reads; and 10 random transactions. The 10 back-to-back writes were randomly generated values to randomly selected address between 0-4. The 10 back-to-back reads were to randomly selected addresses between 0-4. The 10 random transactions were reads and writes of random addresses and values and included randomly setting the reset and randomly sending idle transactions.

To accomplish my back-to-back writes I created a Transaction pipeline object. The purpose of this object was to keep track of which transaction was in each phase (address, data, read back data). The read-back data phase was required because the data line went through the clocking block. I also created a small scoreboard object to make the test bench self-checking.

I was unable to get the contents of the memory to print using a $display statement so I added the first five memory addresses (0-4) to each waveform.

I believe I found a bug in the Asynchronous Memory Simulation Model. The bug occurs when the SRAM controller transitions from the write state to the read state and the address changes. The delays on the WE_b signal and A (address) ports on the Asynchronous Memory Simulation model such that the change on A triggers another write before the internal write enable (we) is de-asserted. To work around this issue while I developed my testbench I inserted an idle transaction between the 10 writes and 10 reads. The idle transaction de-asserts CE_b which prevents the bad memory write from occurring. This situation was not encountered during my 10 random transactions.

The contents of the rest of this report are as follows:

- Code for the package, program, test-bench and interfaces.
- Three waveforms showing 10 writes, 10 reads, and 10 random transactions
- Snippets of Console outputs showing failures, success, and randomization distributions.

Package Code:

```
package my_package;

enum {READ=0,WRITE=1} HWRITE_VALUE;
enum {IDLE=0,NONSEQ=2} HTRANS_VALUE;

class AHB_TRANSACTION;
   static int next_id = 0;
   static int stat_haddr[bit [20:0]];
   static int stat_htrans[bit [1:0]];
   static int stat_hwrite[bit];
   static int stat_rst_count=0;
```

```
   int id;

   rand bit [20:0] HADDR;
   rand bit [1:0]       HTRANS;
   rand bit       HWRITE;
   rand bit [7:0]       HWDATA;
   logic [7:0]    HRDATA;

   rand bit       rst;

   constraint cHADDR {HADDR dist {[0:4]:/40, [5:2097146]:/20,
[2097147:2097151]:/40};};
   constraint cHTRANS {HTRANS == NONSEQ || HTRANS == IDLE;};
   constraint cRST {rst dist {0:=90, 1:=10};};

   function new();
      this.id = next_id++;
   endfunction // new

   function void post_randomize();
      if(this.rst)
        stat_rst_count++;
      stat_htrans[this.HTRANS]++;
      stat_haddr[this.HADDR]++;
      stat_hwrite[this.HWRITE]++;
   endfunction // post_randomize

   function print();
      $display("AHB Transaction Id=%d HADDR=%d HWRITE=%d HWDATA=%d
HRDATA=%d", this.id, this.HADDR, this.HWRITE, this.HWDATA, this.HRDATA);
   endfunction // print

   static function void print_coverage_stats();



      $display("Printing Coverage Stats");

      $display("\tReset Coverage: %0d of %0d transactions", stat_rst_count,
next_id);

      $display("\tHTRANS Coverage (0=IDLE, 2=NONSEQ):");
      foreach(stat_htrans[i]) begin
       $display("\t\t%0d: %0d", i,stat_htrans[i]);
      end

      $display("\tHADDR Coverage:");
      foreach(stat_haddr[i]) begin
       $display("\t\t%0d: %0d", i,stat_haddr[i]);
      end

      $display("\tHWRITE Coverage:");
      foreach(stat_hwrite[i]) begin
       $display("\t\t%0d: %0d", i,stat_hwrite[i]);

      end
```

```
   endfunction

endclass // AHBTransaction

class AHB_SCOREBOARD;
   int            ErrorCount;

   AHB_TRANSACTION MemoryModel[bit [20:0]];

   function new ();
      ErrorCount = 0;
   endfunction

   function print_error_count();
      $display("Scoreboard Error Count @ time %0d: %0d", $time,
this.ErrorCount);
   endfunction // print_error_count

   function void update(AHB_TRANSACTION T);
      if(T.HWRITE == WRITE && T.HTRANS != IDLE) begin
       $display("Updating Scoreboard");
       T.print();
       this.MemoryModel[T.HADDR] = T;
      end
   endfunction // update_scoreboard

   function void check(AHB_TRANSACTION T);

      if(T.HWRITE == READ && T.HTRANS != IDLE) begin
       $display("Checking:");

       if(this.MemoryModel.exists(T.HADDR)) begin
          if(this.MemoryModel[T.HADDR].HWDATA != T.HRDATA) begin
             ErrorCount++;
             T.print();
             $display("Unexpected Read Error @ time=%0d: Read Address: %0d
Read Value: %0d Expected Read Value: %0d", $time, T.HADDR, T.HRDATA,
MemoryModel[T.HADDR].HWDATA);
          end
        else
          $display("Scoreboard Check Passed.");
        end
      end
   endfunction // check_results
endclass // ScoreBoard

class AHB_CMD_PIPELINE;

   AHB_TRANSACTION Idle;
   AHB_TRANSACTION AddressPhase;
   AHB_TRANSACTION DataPhase0;
   AHB_TRANSACTION DataPhase1;
   AHB_TRANSACTION CmdPipeline[$];

   function new();

      //Initialize Idle Packet
```

```
      Idle = new();
      Idle.HADDR = 0;
      Idle.HTRANS = 0;
      Idle.HWRITE = 1;
      Idle.HWDATA = 0;
      Idle.rst = 0;
      //Initialize Current Address and Data Packets
      //
      AddressPhase = Idle;
      DataPhase0 = Idle;
      DataPhase1 = Idle;

   endfunction // new

   function void inject(AHB_TRANSACTION T);
      this.CmdPipeline.push_back(T);
   endfunction // enqueue_trans

   function void advance();

      DataPhase1 = DataPhase0;
      DataPhase0 = AddressPhase;
      if(CmdPipeline.size() > 0)
      AddressPhase = CmdPipeline.pop_front();
      else
      AddressPhase = Idle;
      $display("Pipeline Advancing @ %d", $time());
      AddressPhase.print();
      DataPhase0.print();
      DataPhase1.print();
   endfunction // advance

endclass // AHB_DRIVER

endpackage // my_package
```

Program Code:

```
import my_package::*;

`define SV_RAND_CHECK(r) \
   do begin \
      if (!(r)) begin \
       $display("%s:%0d: Randomization failed \"%s\"", \
              `__FILE__, `__LINE__, `"r`");\
       $finish; \
      end \
   end while(0)

program automatic test(ahb_if.TEST ahb_bus, output bit rst);
   AHB_CMD_PIPELINE AhbCmdPipe;
   AHB_SCOREBOARD Scoreboard;

   AHB_TRANSACTION T,A,D0, D1;
```

```
   initial begin
      AhbCmdPipe = new();
      Scoreboard = new();

      //Inject 10 Write to Memory
      for(int i=0; i<10; i++) begin
       T = new();
       `SV_RAND_CHECK(T.randomize() with {HWRITE == 1; HTRANS == NONSEQ; rst
== 0; HADDR inside {[0:4]};});
       AhbCmdPipe.inject(T);
      end

      //Inject Idle to Prevent Simulation model Error
      AhbCmdPipe.inject(AhbCmdPipe.Idle);

      //Inject 10 Read From Memory
      for(int i=0; i<10; i++) begin
       T = new();
       `SV_RAND_CHECK(T.randomize() with {HWRITE == 0; HTRANS == NONSEQ; rst
== 0; HADDR inside {[0:4]};});
       AhbCmdPipe.inject(T);
      end

      //Inject 10 Random AHB Transactions
      for(int i=0; i<10; i++) begin
       T = new();
       `SV_RAND_CHECK(T.randomize());
       AhbCmdPipe.inject(T);
      end

      //Reset the Controller
      rst <= 1;
      @ahb_bus.cb;
      @ahb_bus.cb;
      @ahb_bus.cb;
      rst <= 0;
      @ahb_bus.cb;

      //Main Test Loop
      for(int i=0; i<31; i++) begin
         //Advance the Driver Pipeline
       AhbCmdPipe.advance();
       A = AhbCmdPipe.AddressPhase;
         D0 = AhbCmdPipe.DataPhase0;
       D1 = AhbCmdPipe.DataPhase1;
       //$display("Address Phase @ time %d", $time);
       //Apply/Receive SRAM Controller Stimulus
       apply_receive_stimulus(A,D0,D1);
       //Update Scoreboard and Check Read Data
       if(rst == 0)
        Scoreboard.update(D0);
       if(D0.rst == 0 && D1.rst == 0)
          Scoreboard.check(D1);
       //Advance the Clock
       @ahb_bus.cb;
```

```
      end

      @ahb_bus.cb;
      @ahb_bus.cb;


      AHB_TRANSACTION::print_coverage_stats();


      Scoreboard.print_error_count();

      for(int i=0; i<5; i++) begin
       //$display("mem_arry0: %d %d", i, $root/top/memory/mem_array0);
      end

   end

   final begin
      $display("Program Complete");
   end

   function void apply_receive_stimulus(ref AHB_TRANSACTION A,
                                ref AHB_TRANSACTION D0,
                                ref AHB_TRANSACTION D1);
      rst <= A.rst;

      //Apply Address Phase
      ahb_bus.cb.HADDR <= A.HADDR;
      ahb_bus.cb.HTRANS<= A.HTRANS;
      ahb_bus.cb.HWRITE<= A.HWRITE;
      //Apply/Receive Data Phase
      ahb_bus.cb.HWDATA<= D0.HWDATA;
      D1.HRDATA = ahb_bus.cb.HRDATA;

   endfunction // apply_stimulus


endprogram // test
```

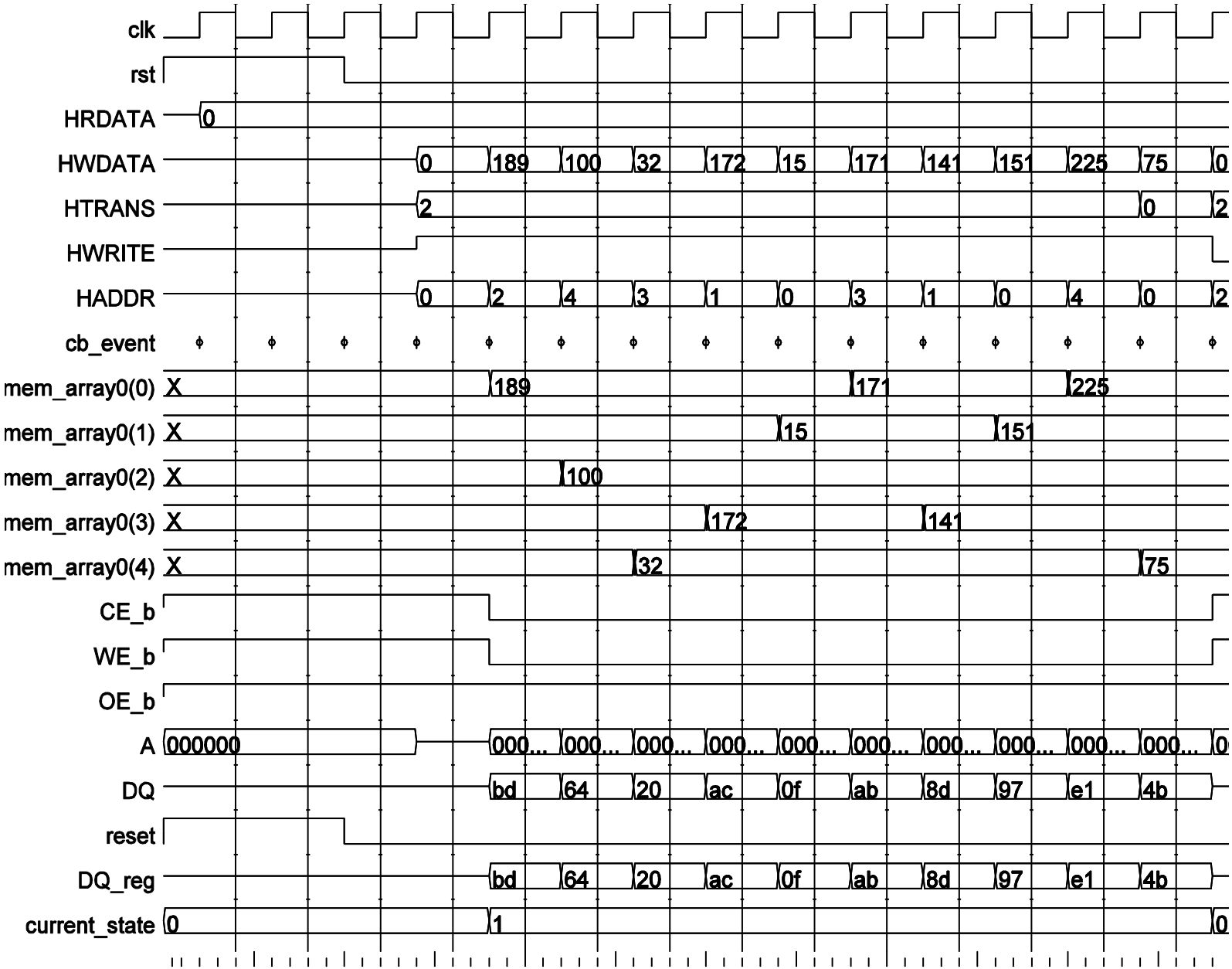
Testbench (top) And Interfaces Code:

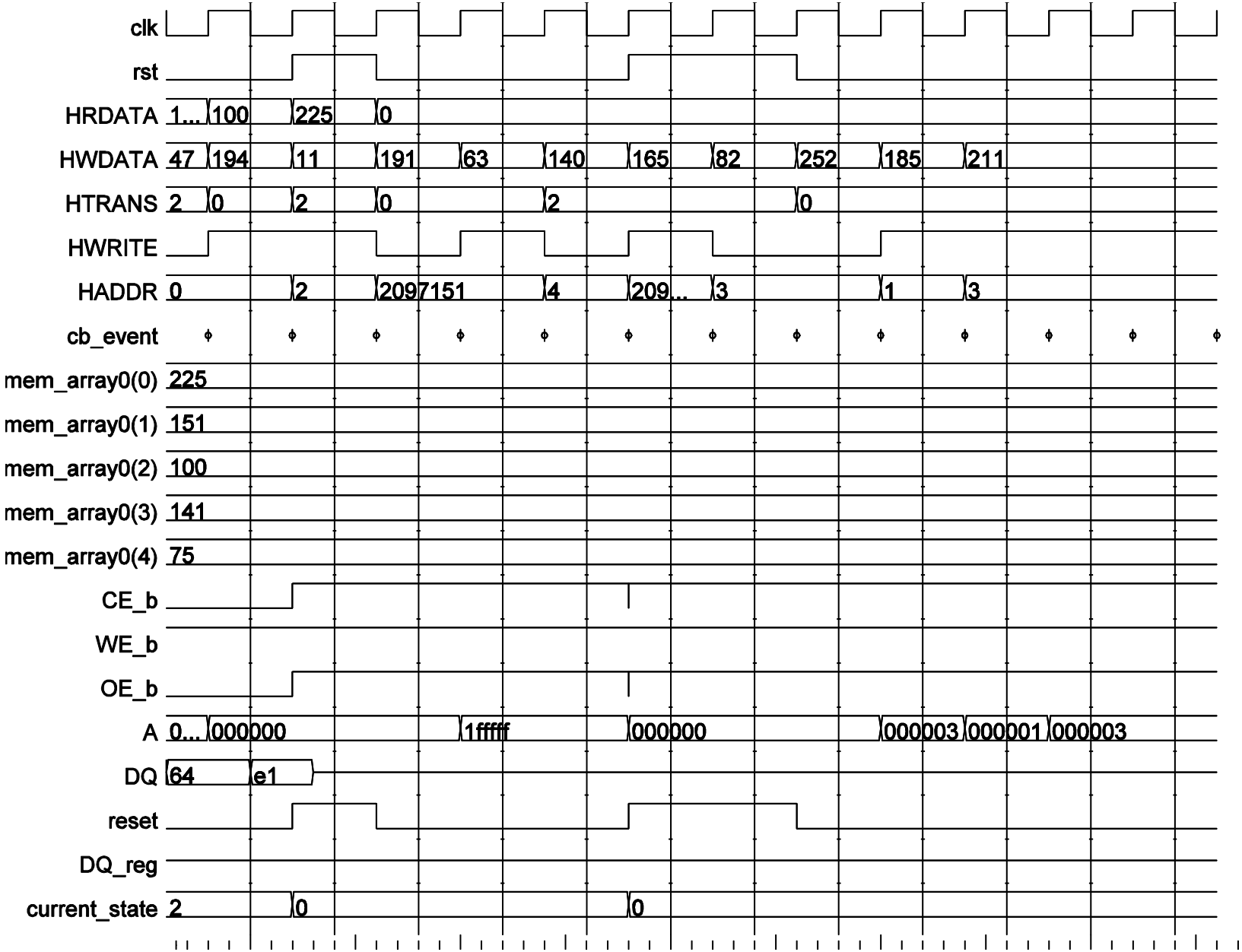
```
`timescale 1ns/100ps

import my_package::*;

interface ahb_if(input bit HCLK);
   logic [20:0]   HADDR;
   logic      HWRITE;
   logic [1:0]      HTRANS;
   logic [7:0]      HWDATA;
   logic [7:0]      HRDATA;

   clocking cb @(posedge HCLK);
      output        HADDR;
      output        HWRITE;
```

```
      output        HTRANS;
      output        HWDATA;
      input         HRDATA;
   endclocking

   modport TEST(clocking cb);
   modport DUT(input HCLK,
               input HADDR,
               input  HWRITE,
               input  HTRANS,
               input  HWDATA,
               output HRDATA);

endinterface // ahb_if

interface sram_if();
   logic      CE_b;
   logic      WE_b;
   logic      OE_b;
   logic [20:0]   A;
   wire [7:0]       DQ;
endinterface // sram_if

module top();

   bit clk;
   wire rst;

   always #10ns clk = ~clk;

   ahb_if ahb_bus(clk);
   sram_if sram_bus();

   //Test Program
   test test_bench(ahb_bus.TEST, rst);
   //DUT
   sram_control sram_ctl(ahb_bus, sram_bus, rst);
   //Memory Model
   async memory(.CE_b(sram_bus.CE_b),
            .WE_b(sram_bus.WE_b),
            .OE_b(sram_bus.OE_b),
            .A(sram_bus.A),
            .DQ(sram_bus.DQ));

Endmodule
```

Waveforms

1. 10 Writes
2. 10 Reads
3. 10 Random Transactions


clk
rst
HRDATA
HWDATA
HTRANS
HWRITE
HADDR
cb_event
mem_array0(0)
mem_array0(1)
mem_array0(2)
mem_array0(3)
mem_array0(4)
CE_b
WE_b
OE_b
A
DQ
reset
DQ_reg
current_state
0 189 100 32 172 15 171 141 151 225 75 0
2 0 2
0 2 4 3 1 0 3 1 0 4 0 2
189 171 225
15 151
100
172 141
32 75
000000 000... 000... 000... 000... 000... 000... 000... 000... 000... 000... 0
bd 64 20 ac 0f ab 8d 97 e1 4b
0 1 0


clk
rst
HRDATA
HWDATA
HTRANS
HWRITE
HADDR
cb_event
mem_array0(0)
mem_array0(1)
mem_array0(2)
mem_array0(3)
mem_array0(4)
CE_b
WE_b
OE_b
A
DQ
reset
DQ_reg
current_state
1...
100
225
0
47
194
11
191
63
140
165
82
252
185
211
2
0
2
0
2
0
0
2
2097151
4
209...
3
1
3
225
151
100
141
75
000000
1fffff
000000
000003
000001
000003
64
e1
2
0
0

Coverage Statistics

```
# Printing Coverage Stats
#       Reset Coverage: 3 of 31 transactions

#       HTRANS Coverage (0=IDLE, 2=NONSEQ):
#               0: 6
#               2: 24

#       HADDR Coverage:
#               0: 6
#               1: 4
#               2: 4
#               3: 8
#               4: 5
#               2097149: 1
#               2097151: 2
#       HWRITE Coverage:
#               0: 14
#               1: 16
```

Transcript Snippet Showing 0 Errors

```
# Pipeline Advancing @          6300
# AHB Transaction Id=      28 HADDR=     3 HWRITE=0 HWDATA=185 HRDATA=  x
# AHB Transaction Id=      27 HADDR=     3 HWRITE=0 HWDATA=252 HRDATA=  x
# AHB Transaction Id=      26 HADDR=2097149 HWRITE=1 HWDATA= 82
HRDATA=  x
# Pipeline Advancing @          6500
# AHB Transaction Id=      29 HADDR=     1 HWRITE=1 HWDATA=211 HRDATA=  x
# AHB Transaction Id=      28 HADDR=     3 HWRITE=0 HWDATA=185 HRDATA=  x
# AHB Transaction Id=      27 HADDR=     3 HWRITE=0 HWDATA=252 HRDATA=  #
Pipeline Advancing @          6700
# AHB Transaction Id=      30 HADDR=     3 HWRITE=1 HWDATA= 87 HRDATA=  x
```

AHB Transaction Id= 29 HADDR= 1 HWRITE=1 HWDATA=211 HRDATA= x
AHB Transaction Id= 28 HADDR= 3 HWRITE=0 HWDATA=185 HRDATA= x

Scoreboard Error Count @ time 7300: 0
Program Complete

Transcript Snippet Showing what happens during an error

Pipeline Advancing @ 4500
AHB Transaction Id= 19 HADDR= 2 HWRITE=0 HWDATA= 47 HRDATA= x
AHB Transaction Id= 18 HADDR= 1 HWRITE=0 HWDATA= 4 HRDATA= x
AHB Transaction Id= 17 HADDR= 3 HWRITE=0 HWDATA=177 HRDATA= x
Checking:
AHB Transaction Id= 17 HADDR= 3 HWRITE=0 HWDATA=177 HRDATA= 75
 # Unexpected Read Error @ time=4500: Read Address: 3 Read Value: 75 Expected Read Value: 141
Pipeline Advancing @ 4700
AHB Transaction Id= 20 HADDR= 0 HWRITE=0 HWDATA=194 HRDATA= x
AHB Transaction Id= 19 HADDR= 2 HWRITE=0 HWDATA= 47 HRDATA= x
AHB Transaction Id= 18 HADDR= 1 HWRITE=0 HWDATA= 4 HRDATA= x
Checking:
AHB Transaction Id= 18 HADDR= 1 HWRITE=0 HWDATA= 4 HRDATA= 75
 # Unexpected Read Error @ time=4700: Read Address: 1 Read Value: 75 Expected Read Value: 151

Pipeline Advancing @ 4900
AHB Transaction Id= 21 HADDR= 0 HWRITE=1 HWDATA= 11 HRDATA= x
AHB Transaction Id= 20 HADDR= 0 HWRITE=0 HWDATA=194 HRDATA= x
AHB Transaction Id= 19 HADDR= 2 HWRITE=0 HWDATA= 47 HRDATA= x
Checking:
AHB Transaction Id= 19 HADDR= 2 HWRITE=0 HWDATA= 47 HRDATA= 75
 # Unexpected Read Error @ time=4900: Read Address: 2 Read Value: 75 Expected Read Value: 100

 # Scoreboard Error Count @ time 7300: 8

(Transcript Snippet only shows 3 of the 8 errors)