fifo_test.v

File location:

$VCS_HOME/doc/examples/nativetestbench/systemverilog/vcs_quickstart/fifo_test.v

```
`timescale 1 ns / 1 ps
/************************************************************
 *                        Chronologic VCS (TM)                *
 *              Copyright (c) 1991-2004 by Synopsys Inc.      *
 *                        ALL RIGHTS RESERVED                 *
 *                                                            *
 *    CONFIDENTIAL AND PROPRIETARY INFORMATION OF SYNOPSYS INC.   *
 ************************************************************/
`define WIDTH 16
`define DEPTH 128
`define ADEPTH 128
// start the program block, named fifo_test
program fifo_test ( rst_n,clk,data_in,data_out,
        push_req_n,pop_req_n,
        diag_n,empty,full,
        almost_empty,almost_full,half_full,
        error);
  output logic  rst_n;
  input clk;
  output logic [`WIDTH-1:0] data_in;
  input [`WIDTH-1:0] data_out;
  output logic push_req_n,pop_req_n,diag_n;
  input empty,full,almost_empty,almost_full,half_full,error;
////////////////////////////////////////////
// Definition: Random Write Data Class container
////////////////////////////////////////////
class random_write_data;
   rand logic [`WIDTH-1:0] data [`ADEPTH];
   rand logic [15:0]  data_rate;
   constraint reasonable {
        data_rate > 0;
        data_rate dist  { [1:8] := 10 , [128:135] := 10, [512:519] :=1 } ;
   }
endclass
////////////////////////////////////////////
```

```
// Definition: Random Read Data Class container
////////////////////////////////////////////
class random_read_data;
   rand logic [15:0]  data_rate;
   constraint reasonable {
        data_rate > 0;
        data_rate dist  { [1:8] := 10 , [128:135] := 10,  [512:519] :=1 } ;
   }
endclass
////////////////////////////////////////
// Definition: Coverage Group
////////////////////////////////////////
   covergroup Cvr ;
      RD: coverpoint READ_DATA.data_rate {
            bins LOW = { [1:127]};
            bins MED = { [128:511]};
            bins HIGH = { [512:1023]};
          }
      WD: coverpoint WRITE_DATA.data_rate {
            bins LOW = { [1:127]};
            bins MED = { [128:511]};
            bins HIGH = { [512:1023]};
          }
      RDxWD: cross RD,WD ;
   endgroup
////////////////////////////////////////////////
// Declaration of
//    one random_read_data instance
//    one random_write_data instance
//    one mailbox
//    one covergroup (Cvr) instance
//    one integer NUM  used for plus arguement
//    a queue of 16bit logic data
////////////////////////////////////////////////
random_read_data READ_DATA;
random_write_data WRITE_DATA;
mailbox mbox;
Cvr fifoCvr ;
int NUM;
logic [`WIDTH-1:0] cdata[$];
//////////////////////////////////////
```

```
//        reset sequence task
////////////////////////////////////////
task fifo_reset ();
  $write("FIFO_RESET: Task reset_fifo started\n");
  push_req_n <= 1'b1;
  pop_req_n <= 1'b1;
  diag_n <= 1'b1;
  rst_n <= 1'b1;
  @(posedge clk);
  rst_n <= 1'b0;
  @(posedge clk);
  rst_n <= 1'b1;
  @(posedge clk);
endtask
////////////////////////////////////////
//  task to check reset signals
////////////////////////////////////////
task fifo_reset_check ();
  $write("FIFO_RESET_CHECK: Task reset_check started \n");
  //all reseted signals must be at proper value
  E1: expect(@(posedge clk) ##[0:2]  empty === 1'b1);
  A1: assert(almost_empty  == 1'b1);
  A2: assert(half_full  == 1'b0);
  A3: assert(almost_full  == 1'b0);
  A4: assert(full  == 1'b0);
  A5: assert(error  == 1'b0);
endtask
////////////////////////////////////////////////////////
//  Task to drive signals for multiple word write (push)
////////////////////////////////////////////////////////
task fifo_mwrite (input [7:0] num_write = 0);
  // depending on input num_write randomize write_data
  // object with additional constraint or
  // just the constraint set defined in the object
  if (num_write != 0)
     WRITE_DATA.randomize() with { data_rate == num_write; }  ;
  else
     WRITE_DATA.randomize();
  $write("FIFO_MWRITE: Sending 128 words with data_rate of %4d to fifo at:   %12d\n",
                WRITE_DATA.data_rate,$time);
  for (int j = 0; j< `ADEPTH; j++)
     begin
```

```
        repeat (WRITE_DATA.data_rate) @(posedge clk);
        push_req_n <= 1'b0;
        // place the data into queue for checking
        cdata.push_back(WRITE_DATA.data[j]);
        // drive data on to data input signals
        data_in <= WRITE_DATA.data[j];
        @(posedge clk);
        // remove the push request signal
        push_req_n <= 1'b1;
      end
endtask

////////////////////////////////////////////////////////
//  Task to drive signals for multiple word read (pop)
////////////////////////////////////////////////////////
task fifo_mread (input [7:0] num_read = 0);
  logic [`WIDTH-1:0] temp;
  // depending on input num_read randomize read_data
  // object with additional constraint or
  // just the constraint set defined in the object
  if (num_read != 0 )
     READ_DATA.randomize() with { data_rate == WRITE_DATA.data_rate; } ;
  else
     READ_DATA.randomize() ;
  $write("FIFO_MREAD:  Reading 128 words with data_rate of %4d from fifo at: %12d\n",
           READ_DATA.data_rate,$time);

  for (int j = 0; j< `ADEPTH; j++)
     begin
       repeat(READ_DATA.data_rate) @(posedge clk);
       while (empty) @(posedge clk);
       pop_req_n <= 1'b0;
       // start the read (pop) and place data in mailbox
       mbox.put(data_out);
       @(posedge clk);
       // remove the pop request signal
       pop_req_n <= 1'b1;
     end
endtask

////////////////////////////////////////////////////////
//  Task to check data is read correctly from FIFO
////////////////////////////////////////////////////////
task fifo_check();
```

```
  logic [`WIDTH-1:0] tempIn;
  logic [`WIDTH-1:0] tempOut;
  $write("FIFO_CHECK: Task check started \n");
  while (1)
     begin
        // Get the data that is read(poped) out of the mailbox
        mbox.get(tempOut);
        // Get the expected data from top of the queue
        tempIn = cdata.pop_front();
        // Compare the two
        assert( tempIn == tempOut )
            // $write("TEST: FIFO read data 'h%0h matched expected\n",tempOut);
   else $write("TEST: FIFO read data 'h%0h DID NOT match expcted data 'h%0h\n",
                tempOut, tempIn);
     end
endtask

////////////////////////////////////////
////////////////////////////////////////
//   Main initial block of program
////////////////////////////////////////
////////////////////////////////////////
initial begin : main_prog
  ////////////////////////////////
  //    Instantiation of objects
  ////////////////////////////////
  mbox = new();
  fifoCvr = new();
  rst_n <= 1'b0;
  @(posedge clk);
  WRITE_DATA = new();
  READ_DATA = new();

  ////////////////////////////////////////
  //    Read in NUM value - how many sets of
  //    Data we want to simulate
  ////////////////////////////////////////
  if (!$value$plusargs("NUM+%0d",NUM))
    NUM = 10;
  $write("FIFO_TEST: Start simulation %d sets of data to the FIFO \n",NUM);
  ////////////////////////////////////////
```

```
//    Reset  and check for proper
//    signals from DUT
////////////////////////////////////////

fork
  fifo_reset();
  fifo_reset_check();
join

////////////////////////////////////////
//  The checker block is spawn in the
//  background ( fork join_none construct)
////////////////////////////////////////
fork
  fifo_check();
join_none
////////////////////////////////////////
//  Basic Test, read/write every clock cycle
//  start write and read in parallel and
//  sample the covergroups initially
////////////////////////////////////////
fork
  fifo_mwrite(1);
  fifo_mread(1);
  @(posedge clk) fifoCvr.sample();
join
////////////////////////////////////////
//  Now read/write with variable data_rate
//  start write and read in parallel and
//  sample the covergroups initially
////////////////////////////////////////
repeat(NUM)
fork
  fifo_mwrite();
  fifo_mread();
  @(posedge clk) fifoCvr.sample();
join
////////////////////////////////////////
//  Fill the fifo, then start write/read
//  will provide for all assertion coverage
////////////////////////////////////////
fifo_mwrite(1);
```

```
  fork
    fifo_mwrite(1);
    fifo_mread(1);
    @(posedge clk) fifoCvr.sample();
  join
end : main_prog                 // initial block
endprogram : fifo_test         // program block
////////////////////////////////////////////////
```