DW_ram_r_w_s_dff.v

File location:

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

```
//--------------------------------------------------------------------------------
//
//       This confidential and proprietary software may be used only
//     as authorized by a licensing agreement from Synopsys Inc.
//     In the event of publication, the following notice is applicable:
//
//                    (C) COPYRIGHT 1999 - 2004 SYNOPSYS INC.
//                           ALL RIGHTS RESERVED
//
//       The entire notice above must be reproduced on all authorized
//     copies.
//
// AUTHOR:    Rick Kelly    April 25, 2001
//
// VERSION:   DW_ram_r_w_s_dff Verilog Simulation Model
//
// DesignWare_version: 446b2915
// DesignWare_release: V-2004.06-DWF_0406
//
//--------------------------------------------------------------------------------
//-------------------------------------------------------------------------
// ABSTRACT:  Synch Write, Asynch Read RAM (Flip-Flop Based)
//            (flip flop memory array)
//            legal range:  depth        [ 2 to 256 ]
//            legal range:  data_width   [ 1 to 256 ]
//            Input data: data_in[data_width-1:0]
//            Output data : data_out[data_width-1:0]
//            Read Address: rd_addr[addr_width-1:0]
//            Write Address: wr_addr[addr_width-1:0]
//            write enable (active low): wr_n
//            chip select (active low): cs_n
//            reset (active low): rst_n
//            clock:clk
//
// MODIFIED:
//         09/22/99  Jay Zhu   Rewrote for STAR91151
//              10/18/00  RPH       Rewrote accoding to new guidelines
//                                  STAR 111067
//              05/25/01  RJK       Rewritten again
//------------------------------------------------------------------------
  module DW_ram_r_w_s_dff (clk, rst_n, cs_n, wr_n, rd_addr, wr_addr, data_in,
                           data_out);
   parameter data_width = 4;
   parameter depth = 8;
   parameter rst_mode = 0;


`define addr_width ((depth>16)?((depth>64)?((depth>128)?8:7):((depth>32)?6:5)):((depth>4)?((
   input [data_width-1:0] data_in;
   input [`addr_width-1:0] rd_addr;
   input [`addr_width-1:0] wr_addr;
   input             wr_n;
   input             rst_n;
   input             cs_n;
   input             clk;
   output [data_width-1:0] data_out;
// synopsys translate_off
   wire [data_width-1:0]   data_in;
   reg [depth*data_width-1:0]    next_mem;
   reg [depth*data_width-1:0]    mem;
  wire [depth*data_width-1:0]    mem_mux;

   wire             a_rst_n;



  initial begin : parameter_check
    integer param_err_flg;
    param_err_flg = 0;



    if ( (data_width < 1) || (data_width > 256) ) begin
      param_err_flg = 1;
      $display(
"ERROR: %m :\n  Invalid value (%d) for parameter data_width (legal range: 1 to 256)",
        data_width );
    end

    if ( (depth < 2) || (depth > 256 ) ) begin
      param_err_flg = 1;
      $display(
"ERROR: %m :\n  Invalid value (%d) for parameter depth (legal range: 2 to 256 )",
        depth );
    end

    if ( (rst_mode < 0) || (rst_mode > 1 ) ) begin
      param_err_flg = 1;
      $display(
"ERROR: %m :\n  Invalid value (%d) for parameter rst_mode (legal range: 0 to 1 )",
        rst_mode );
    end

    if ( param_err_flg == 1) begin
      $display(
        "%m :\n  Simulation aborted due to invalid parameter value(s)");
      $finish;
    end
  end // parameter_check

   assign mem_mux = mem >> (rd_addr * data_width);
   assign data_out = ((rd_addr ^ rd_addr) !== {`addr_width{1'b0}})? {data_width{1'bx}} : (
                             (rd_addr >= depth)? {data_width{1'b0}} :
                                mem_mux[data_width-1 : 0] );

   assign a_rst_n = (rst_mode == 0)? rst_n : 1'b1;


   always @ (posedge clk or negedge a_rst_n) begin : registers
      integer i, j;



      next_mem = mem;
      if ((cs_n | wr_n) !== 1'b1) begin

         if ((wr_addr ^ wr_addr) !== {`addr_width{1'b0}}) begin
            next_mem = {depth*data_width{1'bx}};
         end else begin
```

```
            if ((wr_addr < depth) && ((wr_n | cs_n) !== 1'b1)) begin
               for (i=0 ; i < data_width ; i=i+1) begin
                  j = wr_addr*data_width + i;
                  next_mem[j] = ((wr_n | cs_n) == 1'b0)? data_in[i] | 1'b0
                                        : mem[j];
               end // for
            end // if
         end // if-else
      end // if




      if (rst_n === 1'b0) begin
         mem <= {depth*data_width{1'b0}};
      end else begin
         if ( rst_n === 1'b1) begin
            mem <= next_mem;
         end else begin
            mem <= {depth*data_width{1'bX}};
         end
      end
   end // registers


  always @ (clk) begin : clk_monitor
    if ( (clk !== 1'b0) && (clk !== 1'b1) && ($time > 0) )
      $display( "WARNING: %m :\n  at time = %t, detected unknown value, %b, on clk input.",
                $time, clk );
    end // clk_monitor

// synopsys translate_on

endmodule // DW_ram_r_w_s_dff
```