DW_fifo_s1_sf.v

File location:

$VCS_HOME/doc/examples/nativetestbench/systemverilog/vcs_quickstart/DW_fifo_s1_sf.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 1996 - 2004 SYNOPSYS INC.
//                           ALL RIGHTS RESERVED
//
//       The entire notice above must be reproduced on all authorized
//     copies.
//
// AUTHOR:    Rick Kelly         2/21/97
//
// VERSION:   Simulation Architecture
//
// DesignWare_version: 8cb3b816
// DesignWare_release: V-2004.06-DWF_0406
//
//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------
//
// ABSTRACT:  Synchronous with Static Flags
//            programmable almost empty and almost full flags
//
//               Parameters:     Valid Values
//               ==========      ============
//               width           [ 1 to 256 ]
//               depth           [ 2 to 256 ]
//               ae_level        [ 1 to depth-1 ]
//               af_level        [ 1 to depth-1 ]
//               err_mode        [ 0 = sticky error flag w/ ptr check,
//                                 1 = sticky error flag (no ptr chk),
//                                 2 = dynamic error flag ]
//               reset_mode      [ 0 = asynchronous reset,
//                                 1 = synchronous reset ]
```

```
//
//              Input Ports:    Size    Description
//              ===========     ====    ===========
//              clk             1 bit   Input Clock
//              rst_n           1 bit   Active Low Reset
//              push_req_n      1 bit   Active Low Push Request
//              pop_req_n       1 bit   Active Low Pop Request
//              diag_n          1 bit   Active Low diagnostic control
//              data_in         W bits  Push data input
//
//              Output Ports    Size    Description
//              ============    ====    ===========
//              empty           1 bit   Empty Flag
//              almost_empty    1 bit   Almost Empty Flag
//              half_full       1 bit   Half Full Flag
//              almost_full     1 bit   Almost Full Flag
//              full            1 bit   Full Flag
//              error           1 bit   Error Flag
//              data_out        W bits  Pop data output
//
//
// MODIFIED:
//         RJK 2/10/98
//         Added better handling of 'x' inputs and async rst
//
//-------------------------------------------------------------------------------
//
module DW_fifo_s1_sf (
    clk, rst_n, push_req_n, pop_req_n, diag_n, data_in, empty,
    almost_empty, half_full, almost_full, full, error, data_out );

parameter width  = 8;
parameter depth  = 4;
parameter ae_level = 1;
parameter af_level = 1;
parameter err_mode  =  0 ;
parameter rst_mode  =  0 ;

`define addr_width ((depth>16)?((depth>64)?((depth>128)? 8 : 7) : ((depth>32)? 6 : 5)) : ((d

`define ctl_rst_mode (rst_mode % 2)
`define ram_rst_mode ((rst_mode > 0)? 1 : 0)

 input  clk,  rst_n,  push_req_n,  pop_req_n,  diag_n;
 input [width-1 : 0] data_in;
 output  empty,  almost_empty,  half_full,  almost_full,  full, error;
```

```
 output [width-1 : 0 ]  data_out;
 wire clk, rst_n, push_req_n, pop_req_n, diag_n;
 wire empty, almost_empty, half_full, almost_full, full, error;
 wire [width-1 : 0 ]  data_out;

 wire [`addr_width-1 : 0] ram_rd_addr, ram_wr_addr;
 wire ram_rst_n, ram_we_n;



  initial begin : parameter_check
    integer param_err_flg;
    param_err_flg = 0;



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

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

```
    if ( (err_mode < 0) || (err_mode > 2 ) ) begin
      param_err_flg = 1;
      $display(
"ERROR: %m :\n  Invalid value (%d) for parameter err_mode (legal range: 0 to 2 )",
        err_mode );
    end

    if ( (rst_mode < 0) || (rst_mode > 3 ) ) begin
      param_err_flg = 1;
      $display(
"ERROR: %m :\n  Invalid value (%d) for parameter rst_mode (legal range: 0 to 3 )",
        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 ram_rst_n = (rst_mode < 2)? rst_n : 1'b1;

    DW_fifoctl_s1_sf #(depth, ae_level, af_level, err_mode, `ctl_rst_mode) FIFO_CTL(
                        .clk(clk),
                        .rst_n(rst_n),
                        .push_req_n(push_req_n),
                        .pop_req_n(pop_req_n),
                        .diag_n(diag_n),
                        .empty(empty),
                        .almost_empty(almost_empty),
                        .half_full(half_full),
                        .almost_full(almost_full),
                        .full(full),
                        .error(error),
                        .we_n(ram_we_n),
                        .wr_addr(ram_wr_addr),
                        .rd_addr(ram_rd_addr)
                        );

    DW_ram_r_w_s_dff #(width, depth, `ram_rst_mode) FIFO_MEM(
                        .clk(clk),
```

```
                        .rst_n(ram_rst_n),
                        .wr_n(ram_we_n),
                        .cs_n(1'b0),
                        .rd_addr(ram_rd_addr),
                        .wr_addr(ram_wr_addr),
                        .data_in(data_in),
                        .data_out(data_out)
                        );

endmodule
```