DW_fifoctl_s1_sf.v

File location:

$VCS_HOME/doc/examples/nativetestbench/systemverilog/vcs_quickstart/DW_fifoctl_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 1999 - 2004 SYNOPSYS INC.
//                           ALL RIGHTS RESERVED
//
//       The entire notice above must be reproduced on all authorized
//     copies.
//
// AUTHOR:    Rick Kelly         10/15/99
//
// VERSION:   Simulation Architecture
//
// DesignWare_version: 29c29f78
// DesignWare_release: V-2004.06-DWF_0406
//
//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------
//
// ABSTRACT:  Synchronous with Static Flags
//           static programmable almost empty and almost full flags
//
//           This FIFO controller designed to interface to synchronous
//           true dual port RAMs.
//
//              Parameters:     Valid Values
//              ==========      ============
//              depth           [ 2 to 16777216 ]
//              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
//
//              Output Ports    Size    Description
//              ============    ====    ===========
//              we_n            1 bit   Active low Write Enable (to RAM)
//              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
//              wr_addr         N bits  Write Address (to RAM)
//              rd_addr         N bits  Read Address (to RAM)
//
//                Note: the value of N for wr_addr and rd_addr is
//                      determined from the parameter, depth.  The
//                      value of N is equal to:
//                              ceil( log2( depth ) )
//
//
//
// MODIFIED:
//
//  11/12/01  RJK  Fixed lint error by converting int to vector
//                 STAR #129582
//-------------------------------------------------------------------------------
//
module DW_fifoctl_s1_sf (
    clk, rst_n, push_req_n, pop_req_n, diag_n, we_n, empty,
    almost_empty, half_full, almost_full, full, error, wr_addr, rd_addr );

parameter depth  = 4;
parameter ae_level  =  1 ;
parameter af_level  =  1 ;
parameter err_mode  =  0 ;
```

```
parameter rst_mode  =  0 ;
`define DW_addr_width ((depth>4096)? ((depth>262144)? ((depth>2097152)? ((depth>8388608)? 24
 input  clk,  rst_n,  push_req_n,  pop_req_n,  diag_n;
 output  we_n,  empty,  almost_empty,  half_full,  almost_full,  full, error;
 output[`DW_addr_width-1 : 0 ]  wr_addr, rd_addr;

 wire a_rst_n, diag_n_int, we_n, empty, full, error;
 reg empty_int, almost_empty, half_full, almost_full, full_int, next_error_int, error_int;
wire [`DW_addr_width-1 : 0 ] wr_addr, rd_addr;
 integer wrd_count, next_wrd_count;
 integer wr_addr_int, next_wr_addr_int;
 integer rd_addr_int, next_rd_addr_int;
 wire [31:0] rd_addr_vec, wr_addr_vec;

  initial begin : parameter_check
    integer param_err_flg;
    param_err_flg = 0;



    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 > 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 ( (depth < 2) || (depth > 16777216 ) ) begin
      param_err_flg = 1;
      $display(
"ERROR: %m :\n  Invalid value (%d) for parameter depth (legal range: 2 to 16777216 )",
        depth );
    end
        wrd_count = -1;
        wr_addr_int = -1;
        rd_addr_int = -1;


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

    assign diag_n_int = (err_mode == 0)? diag_n : 1'b1;

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

    always @ (push_req_n or pop_req_n or full_int or wr_addr_int)
        begin : mk_next_wr_addr_int
        if ((push_req_n === 1'b0) && ((full_int === 1'b0)||(pop_req_n === 1'b0)))
            next_wr_addr_int = (wr_addr_int + 1) % depth;

        else
            next_wr_addr_int = ((push_req_n === 1'b1) || (full_int === 1'b1))?
                            wr_addr_int : -1;
    end // mk_next_wr_addr_int

    always @ (pop_req_n or empty_int or diag_n_int or rd_addr_int)
        begin : mk_next_rd_addr_int
        if (diag_n_int === 1'b0)
            next_rd_addr_int = 0;

        else begin
```

```
    if ((diag_n_int === 1'b1) && (pop_req_n === 1'b0) && (empty_int === 1'b0))
                next_rd_addr_int = (rd_addr_int + 1) % depth;

            else
next_rd_addr_int = (((pop_req_n === 1'b1) || (empty_int === 1'b1)) && (diag_n_int === 1'b1))
                                    rd_addr_int : -1;
        end
    end // mk_next_rd_addr_int

    always @ (push_req_n or pop_req_n or wrd_count)
        begin : mk_next_wrd_count
        if ( wrd_count < 0 )
            next_wrd_count = wrd_count;

        else
            if ((push_req_n === 1'b1) && (pop_req_n === 1'b0) && (wrd_count != 0))
                next_wrd_count = wrd_count - 1;

            else
if ( ((push_req_n === 1'b0) && (pop_req_n === 1'b0) && (wrd_count == 0)) ||
     ((push_req_n === 1'b0) && (pop_req_n === 1'b1) && (wrd_count < depth)) )
                    next_wrd_count = wrd_count + 1;

                else
    if ( ((push_req_n === 1'b0) && (pop_req_n === 1'b0) && (wrd_count > 0)) ||
 ((push_req_n === 1'b0) && (pop_req_n === 1'b1) && (wrd_count == depth)) ||
 ((push_req_n === 1'b1) && (pop_req_n === 1'b0) && (wrd_count == 0)) ||
                         ((push_req_n === 1'b1) && (pop_req_n === 1'b1)) )

                        next_wrd_count = wrd_count;

                    else
                        next_wrd_count = -1;
    end // mk_next_wrd_count

    always @ (push_req_n or pop_req_n or rd_addr_int or
                    wr_addr_int or wrd_count or next_wrd_count or error_int)
        begin : mk_next_error
        if ((err_mode < 2) && (error_int !== 1'b0))
            next_error_int = error_int;

        else
            if ((err_mode == 0) && (rd_addr_int >= 0) && (wr_addr_int >= 0) &&
```

```
( ((rd_addr_int == wr_addr_int) && (wrd_count > 0) && (wrd_count < depth))  ||
  ((rd_addr_int != wr_addr_int) && ((wrd_count == 0) || (wrd_count == depth))) ))

            next_error_int = 1'b1;

        else
            next_error_int =
                        (wrd_count == 0)? ~pop_req_n :
                        (wrd_count == depth)? (pop_req_n & ~push_req_n) :
  ((wrd_count < 0)&&((push_req_n & pop_req_n)==1'b0))? 1'bx : 1'b0;

  end // mk_next_error

  always @ (posedge clk or negedge a_rst_n)
      begin : clk_registers
      if (rst_n === 1'b0) begin
          wr_addr_int <= 0;
          rd_addr_int <= 0;
          wrd_count <= 0;
          error_int <= 1'b0;
      end
      else
          if (rst_n === 1'b1) begin
              wr_addr_int <= next_wr_addr_int;
              rd_addr_int <= next_rd_addr_int;
              wrd_count <= next_wrd_count;
              error_int <= next_error_int;
          end
          else begin
              wr_addr_int <= -1;
              rd_addr_int <= -1;
              wrd_count <= -1;
              error_int <= 1'bx;
          end
  end // clk_registers

  always @ (wrd_count)
      begin : mk_flags
      if ( wrd_count < 0 ) begin
          empty_int = 1'bx;
          almost_empty = 1'bx;
          half_full = 1'bx;
          almost_full = 1'bx;
```

```
        full_int = 1'bx;
    end
    else begin
        if (wrd_count == 0)
            empty_int = 1'b1;

        else
            empty_int = 1'b0;

        if (wrd_count > ae_level)
            almost_empty = 1'b0;

        else
            almost_empty = 1'b1;

        if ( wrd_count < ((depth  +  1 )/ 2 ) )
            half_full = 1'b0;

        else
            half_full = 1'b1;

        if ( wrd_count < (depth  - af_level ) )
            almost_full = 1'b0;

        else
            almost_full = 1'b1;

        if ( wrd_count == depth )
            full_int = 1'b1;

        else
            full_int = 1'b0;
    end
end // mk_flags

assign wr_addr_vec = wr_addr_int;
assign rd_addr_vec = rd_addr_int;
assign wr_addr = (wr_addr_int < 0)? {`DW_addr_width{1'bx}} : wr_addr_vec[`DW_addr_width-
assign rd_addr = (rd_addr_int < 0)? {`DW_addr_width{1'bx}} : rd_addr_vec[`DW_addr_width-

assign  we_n  = (push_req_n  | (pop_req_n  & full_int ));
assign  empty = empty_int;
```

```
    assign  full = full_int;
    assign  error = error_int;


  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

`undef DW_addr_width

endmodule
```