DW_fifo_s1_sf_sim.vhd

File location:

$VCS_HOME/doc/examples/nativetestbench/systemverilog/vcs_quickstart/DW_fifo_s1_sf_sim.vhd

```
--------------------------------------------------------------------------------
--
--       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         1/23/97
--
-- VERSION:   VHDL Simulation Model
--
-- DesignWare_version: 3366ad93
-- 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 ]
--               rst_mode        [ 0 = asynchronous reset control & memory,
--                                 1 = synchronous reset control & memory,
--                                 2 = asynchronous reset control only,
--                                3 = synchronous reset control only ]
--
--              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 - 2/3/98
--         Added better handling of 'X' and 'U' inputs
--
---------------------------------------------------------------------------------
--
--
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
--use DWARE.DWpackages.all;
use WORK.DWpackages.all;
architecture sim of DW_fifo_s1_sf is
constant addr_width : INTEGER := bit_width( depth );
signal push_req_n_int : std_logic;
signal pop_req_n_int : std_logic;
signal rst_n_int : std_logic;
signal diag_n_int : std_logic;
    begin
    rst_n_int <= To_01X( rst_n );
```

```
push_req_n_int <= To_01X( push_req_n );
pop_req_n_int <= To_01X( pop_req_n );
diag_n_int <= To_01X( diag_n );
sim_mdl : process
    variable wrd_count, wr_addr, rd_addr : integer range -1 to depth;
    variable error_int : std_logic;
    type mem_array is array(0 to depth-1) of std_logic_vector(width-1 downto 0);
    variable memory : mem_array;
    begin
    if ( err_mode > 1 ) then
        error_int := '0';
    end if;
    if ( (rst_mode mod 2) = 1 ) then
        wait until (clk'event and (clk'last_value = '0') and (clk = '1'));
    end if;
    if  (rst_n_int = '0') then
        wrd_count := 0;
        wr_addr := 0;
        rd_addr := 0;
        error_int := '0';
        error <= '0';
        empty <= '1';
        almost_empty <= '1';
        half_full <= '0';
        almost_full <= '0';
        full <= '0';
        if ( ((rst_mode / 2) mod 2) = 0 ) then
            for i in 0 to depth-1 loop
                memory(i) := (others => '0');
            end loop;
        end if;
        data_out <= memory(0);
        wait until (rst_n_int /= '0');
      elsif ( rst_n_int = 'X' ) then
        wrd_count := -1;
        wr_addr := -1;
        rd_addr := -1;
        error_int := 'X';
        error <= 'X';
        empty <= 'X';
        almost_empty <= 'X';
        half_full <= 'X';
        almost_full <= 'X';
    full <= 'X';
    if ( ((rst_mode / 2) mod 2) = 0 ) then
        for i in 0 to depth-1 loop
            memory(i) := (others => 'X');
        end loop;
    end if;
    data_out <= (others => 'X');
    wait until (rst_n_int /= 'X');
  else
    if ((rst_mode mod 2) = 0) then
        wait until (clk'event and (clk'last_value = '0') and (clk = '1'))
                        or ( rst_n_int /= '1');
    end if;
    if ( rst_n_int = '1' ) then
        if ( err_mode = 0 ) then
            if ( (wrd_count = 0) or (wrd_count = depth) ) then
                if ( rd_addr /= wr_addr ) then
                    error_int := '1';
                end if;
              elsif ( rd_addr = wr_addr ) then
                error_int := '1';
            end if;
        end if;
        if ( (wrd_count < depth) or (pop_req_n_int = '0') ) then
            if ( wr_addr < 0 ) then
                if ( push_req_n_int /= '1' ) then
                    for i in 0 to depth-1 loop
                        memory(i) := (others => 'X');
                    end loop;
                end if;
             else
                if (push_req_n_int = '0') then
                    memory( wr_addr ) := data_in;
                  elsif (push_req_n_int /= '1') then
                    memory( wr_addr ) := (others => 'X');
                end if;
            end if;
        end if;
        if (pop_req_n_int = '0') and (push_req_n_int = '0') then
            if ( wrd_count < 0 ) then
                rd_addr := -1;
                wr_addr := -1;
                error_int := 'X';
```

```
     else
        if ( wrd_count = 0 ) then
            error_int := '1';
            wrd_count := 1;
          else
            rd_addr := (rd_addr + 1) mod depth;
        end if;
        wr_addr := (wr_addr + 1) mod depth;
    end if;
end if;
if (pop_req_n_int = '0') and (push_req_n_int = '1') then
    if ( wrd_count < 0 ) then
        rd_addr := -1;
        error_int := 'X';
     else
        if ( wrd_count = 0 ) then
            error_int := '1';
          else
            wrd_count := wrd_count - 1;
            rd_addr := (rd_addr + 1) mod depth;
        end if;
    end if;
end if;
if (pop_req_n_int = '1') and (push_req_n_int = '0') then
    if ( wrd_count < 0 ) then
        wr_addr := -1;
        error_int := 'X';
     else
        if ( wrd_count = depth ) then
            error_int := '1';
          else
            wrd_count := wrd_count + 1;
            wr_addr := (wr_addr +1) mod depth;
        end if;
    end if;
end if;
if ( pop_req_n_int = 'X' ) then
    if ( wrd_count = 0 ) then
        error_int := 'X';
     else
        wrd_count := -1;
        rd_addr := -1;
    end if;
```

```
        end if;
        if ( push_req_n_int = 'X' ) then
            if ( wrd_count = depth ) and ( pop_req_n_int /= '0' ) then
                error_int := 'X';
             else
                wrd_count := -1;
                wr_addr := -1;
            end if;
        end if;
    end if;
end if;
if ( wrd_count = 0 ) then
    empty <= '1';
  elsif ( wrd_count < 0 ) then
    empty <= 'X';
  else
    empty <= '0';
end if;
if ( wrd_count > ae_level ) then
    almost_empty <= '0';
  elsif ( wrd_count < 0 ) then
    almost_empty <= 'X';
  else
    almost_empty <= '1';
end if;
if ( wrd_count >= (depth +1)/2 ) then
    half_full <= '1';
  elsif ( wrd_count < 0 ) then
    half_full <= 'X';
  else
    half_full <= '0';
end if;
if ( wrd_count < (depth - af_level) ) then
    almost_full <= '0';
  elsif ( wrd_count < 0 ) then
    almost_full <= 'X';
  else
    almost_full <= '1';
end if;
if ( wrd_count = depth ) then
    full <= '1';
  elsif ( wrd_count < 0 ) then
    full <= 'X';
```

```vhdl
          else
            full <= '0';
        end if;
        if ( err_mode = 0 ) and ( diag_n_int /= '1' ) then
            if ( diag_n_int = '0' ) then
                rd_addr := 0;
             else
                rd_addr := -1;
            end if;
        end if;
        if ( rd_addr < 0 ) then
            data_out <= (others => 'X');
         else
            data_out <= memory( rd_addr );
        end if;
        error <= error_int;
        if ( (rst_mode mod 2) = 1 ) then
            wait until (clk = '0');
          elsif ( rst_n_int = '1' ) then
            wait until ( (clk = '0') or (rst_n_int /= '1') );
        end if;
    end process;
end sim;
--------------------------------------------------------------------------------
-- auto generated configuration
configuration DW_fifo_s1_sf_cfg_sim of DW_fifo_s1_sf is
 for sim
 end for; -- sim
end DW_fifo_s1_sf_cfg_sim;
```