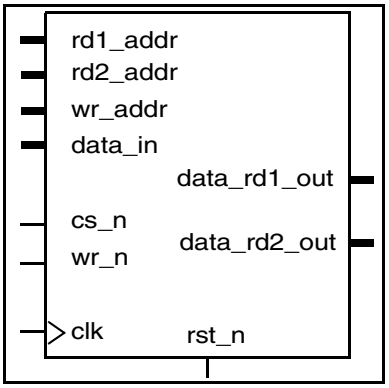
DW_ram_2r_w_s_dff

Sync. Write, Async. Dual Read RAM (Flip-Flop-Based)

Version, STAR and Download Information: IP Directory

Features and Benefits

- Parameterized word depth
- Parameterized data width
- Synchronous static memory
- Parameterized reset mode (synchronous or asynchronous)
- High testability using DFT Compiler



Description

DW_ram_2r_w_s_dff implements a parameterized, synchronous, three-port static RAM consisting of one write port and two read ports.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock
rst_n	1 bit	Input	Reset, active low
cs_n	1 bit	Input	Chip select, active low
wr_n	1 bit	Input	Write enable, active low
rd1_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Read1 address bus
rd2_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Read2 address bus
wr_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Write address bus
data_in	*data_width* bit(s)	Input	Input data bus
data_rd1_out	*data_width* bit(s)	Output	Output data bus for read1
data_rd2_out	*data_width* bit(s)	Output	Output data bus for read2

Table 1-2 Parameter Description

Parameter	Values	Description
data_width	1 to 2048 Default = none	Width of `data_in` and `data_out` buses
depth	2 to 1024 Default = none	Number of words in the memory array (address width)
rst_mode	0 or 1 Default = 1	Determines the reset methodology: 0 = `rst_n` asynchronously initializes the RAM 1 = `rst_n` synchronously initializes the RAM

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
rtl[a]	Synthesis model	DesignWare

a. The implementation, “rtl,” replaces the obsolete implementation, “str.” Existing designs that specify the obsolete implementation (“str”) will automatically have that implementation replaced by the new superseding implementation (“rtl”) as will be noted by an information message (SYNDB-36) generated during DC compilation.

Table 1-4 Simulation Models

Model	Function
DW06.DW_RAM_2R_W_S_DFF_CFG_SIM	VHDL simulation configuration
dw/dw06/src/DW_ram_2r_w_s_dff_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_ram_2r_w_s_dff.v	Verilog simulation model source code

The write operation of the RAM is fully synchronous with respect to the clock, `clk`, and takes one clock cycle to perform. The RAM can perform simultaneous read and write operations.

Write data enters the RAM through the `data_in` input port, and is read out through either the `data_rd1_out` port or the `data_rd2_out` port. The RAM is constantly reading regardless of the state of `cs_n`.

The `rd1_addr`, `rd2_addr`, and `wr_addr` ports are used to address the *depth* words in the memory. If `rd1_addr` or `rd2_addr` contains a value beyond the maximum depth, that corresponding output port is driven LOW. For example, if `rd1_addr` = 7 and *depth* = *6*, then the `data_rd1_out` bus is driven LOW. If `rd2_addr` is beyond the maximum depth, then `data_rd2_out` is driven LOW.

For `wr_addr` beyond the maximum depth (for example, `wr_addr` = 7 and `depth` = 6), nothing occurs and the data is lost. No warnings are given during simulations when an address beyond the scope of `depth` is used.

Chip Selection, Reading and Writing

The `cs_n` input is the chip select, active low signal that enables data to be written to the RAM. The RAM is constantly reading, regardless of the state of `cs_n`.

When `cs_n` is low and the RAM is enabled by `wr_n`, the active low write enable, data is written into the RAM on the rising edge of `clk`. If one of the read address buses is the same value as `wr_addr` and `wr_n` is LOW, `data_in` is transferred to appropriate data out (`data_rd1_out` or `data_rd2_out`) after the first rising edge of `clk`. When `cs_n` is high, writing to the RAM is disabled.

Reset

The `rst_n` port is an active low input that initializes the RAM to zeros. If the *rst_mode* parameter is set to 0, `rst_n` asynchronously resets the RAM. If the *rst_mode* parameter is set to 1, `rst_n` synchronously resets the RAM, independent of the value of `cs_n`. If the `rst_n` port is tied HIGH, the synthesis optimizes the logic and build a non-resetable RAM.

Application Notes

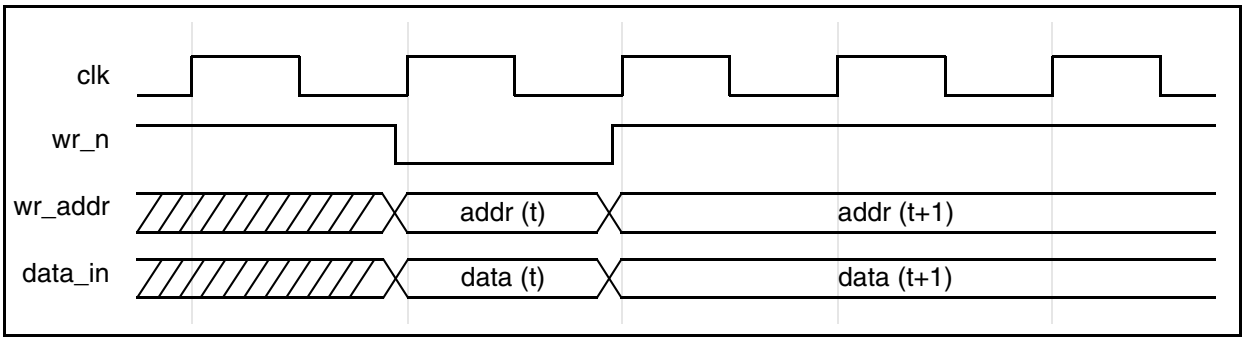
DW_ram_2r_w_s_dff is intended to be used as small scratch-pad memory, programmable lookup tables, and writable control storage. Because DW_ram_2r_w_s_dff is built from the cells within the ASIC cell library, it should be kept small to obtain an efficient implementation. If a larger memory is required, you should consider using a hard macro RAM from the ASIC library in use.

Timing Waveforms

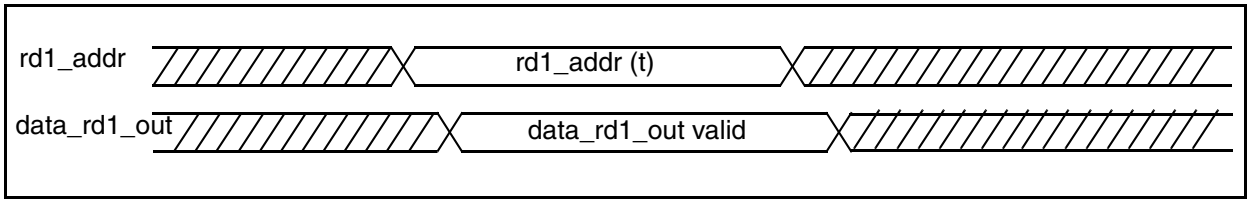
The figures in this section show timing diagrams for various conditions of DW_ram_2r_w_s_dff.

Figure 1-1 Instantiated RAM Timing Waveforms

Write Timing, rst_n = 1, cs_n = 0



Read Port 1 Timing, address controlled, rst_n = 1, cs_n = don't care



Read Port 2 Timing, address controlled, rst_n = 1, cs_n = don't care

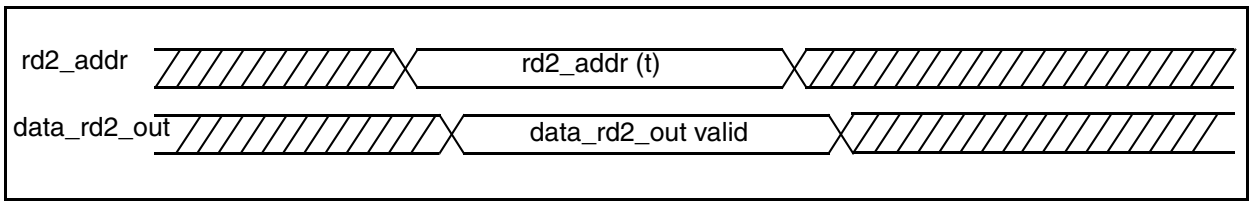
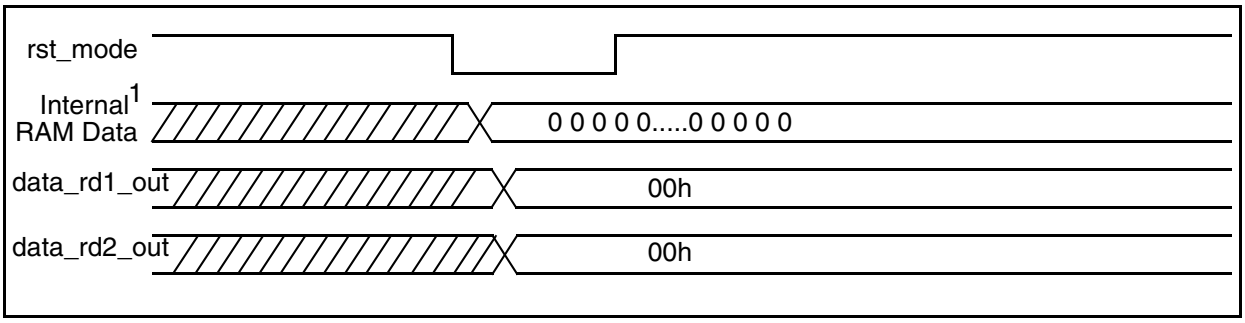
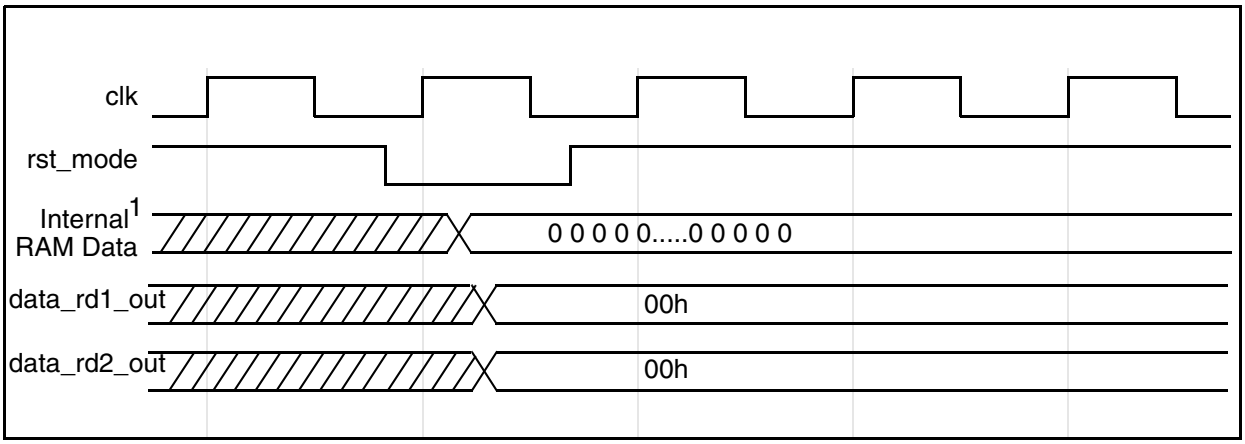


Figure 1-2 Instantiated RAM Timing Waveforms (continued)

Asynchronous Reset, *rst_mode* = 0, cs_n = don't care



Synchronous Reset, *rst_mode* = 1, cs_n = don't care



[1] Internal RAM Data is the array of memory bits; the memory is not available to users.

Related Topics

- Memory – Synchronous RAMs Listing
- DesignWare Building Block IP Documentation Overview

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DWARE.DW_foundation_comp.all;

entity DW_ram_2r_w_s_dff_inst is
  generic (inst_data_width : INTEGER := 8;
           inst_depth : INTEGER := 8;
           inst_rst_mode : INTEGER := 0 );
  port (inst_clk      : in std_logic;
        inst_rst_n    : in std_logic;
        inst_cs_n     : in std_logic;
        inst_wr_n     : in std_logic;
        inst_rd1_addr : in std_logic_vector(bit_width(inst_depth)-1
                                                              downto 0);
        inst_rd2_addr : in std_logic_vector(bit_width(inst_depth)-1
                                                              downto 0);
        inst_wr_addr : in std_logic_vector(bit_width(inst_depth)-1 downto 0);
        inst_data_in : in std_logic_vector(inst_data_width-1 downto 0);
        data_rd1_out_inst : out std_logic_vector(inst_data_width-1 downto 0);
        data_rd2_out_inst : out std_logic_vector(inst_data_width-1 downto 0)
        );
end DW_ram_2r_w_s_dff_inst;

architecture inst of DW_ram_2r_w_s_dff_inst is
begin

  -- Instance of DW_ram_2r_w_s_dff
  U1 : DW_ram_2r_w_s_dff
    generic map (data_width => inst_data_width,   depth => inst_depth,
                 rst_mode => inst_rst_mode )
    port map (clk => inst_clk,   rst_n => inst_rst_n,   cs_n => inst_cs_n,
              wr_n => inst_wr_n,   rd1_addr => inst_rd1_addr,
              rd2_addr => inst_rd2_addr,   wr_addr => inst_wr_addr,
              data_in => inst_data_in,   data_rd1_out => data_rd1_out_inst,
              data_rd2_out => data_rd2_out_inst );
end inst;

-- pragma translate_off
configuration DW_ram_2r_w_s_dff_inst_cfg_inst of DW_ram_2r_w_s_dff_inst is
  for inst
  end for; -- inst
end DW_ram_2r_w_s_dff_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_ram_2r_w_s_dff_inst( inst_clk, inst_rst_n, inst_cs_n, inst_wr_n,
                  inst_rd1_addr, inst_rd2_addr, inst_wr_addr, inst_data_in,
                  data_rd1_out_inst, data_rd2_out_inst );
  parameter data_width = 8;
  parameter depth = 8;
  parameter rst_mode = 0;
  `define bit_width_depth 3 // ceil(log2(depth))

  input inst_clk;
  input inst_rst_n;
  input inst_cs_n;
  input inst_wr_n;
  input [`bit_width_depth-1 : 0] inst_rd1_addr;
  input [`bit_width_depth-1 : 0] inst_rd2_addr;
  input [`bit_width_depth-1 : 0] inst_wr_addr;
  input [data_width-1 : 0] inst_data_in;
  output [data_width-1 : 0] data_rd1_out_inst;
  output [data_width-1 : 0] data_rd2_out_inst;

  // Instance of DW_ram_2r_w_s_dff
  DW_ram_2r_w_s_dff #(data_width,   depth, rst_mode)
    U1 (.clk(inst_clk),   .rst_n(inst_rst_n),   .cs_n(inst_cs_n),
        .wr_n(inst_wr_n),   .rd1_addr(inst_rd1_addr),
        .rd2_addr(inst_rd2_addr),   .wr_addr(inst_wr_addr),
        .data_in(inst_data_in),   .data_rd1_out(data_rd1_out_inst),
        .data_rd2_out(data_rd2_out_inst) );
endmodule
```

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.

All other product or company names may be trademarks of their respective owners.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043

www.synopsys.com