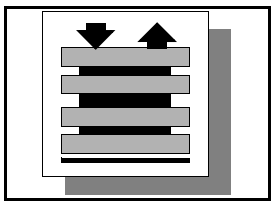
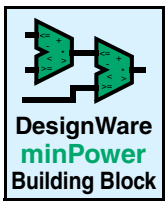




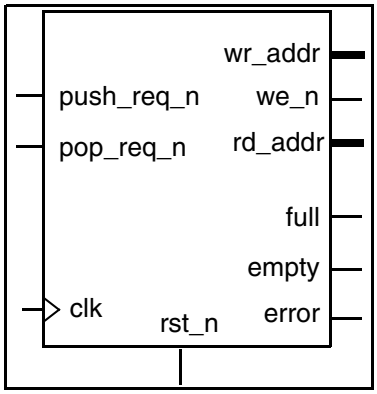
DW_stackctl

Synchronous (Single Clock) Stack Controller

Version, STAR and Download Information: IP Directory

Features and Benefits

- Parameterized word width and depth
- Stack empty and full status flags
- Stack error flag indicating underflow and overflow
- Fully registered synchronous address and flag output ports
- All operations execute in a single clock cycle
- Parameterized reset mode (synchronous or asynchronous)
- Interfaces with common hard macro or compiled ASIC dual-port synchronous RAMs
- Provides minPower benefits with the DesignWare-LP license.



Description

DW_stackctl is a stack RAM controller designed to interface with a dual-port synchronous RAM.

The RAM must have:

- A synchronous write port, and
- Either an asynchronous or synchronous read port.

The stack controller provides address generation, write-enable logic, flag logic, and operational error detection logic. Parameterizable features include stack depth (up to 24 address bits or 16,777,216 locations), and type of reset (either asynchronous or synchronous). You specify these parameters when the controller is instantiated in the design.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Reset input, active low asynchronous if `rst_mode` = 0, synchronous if `rst_mode` = 1
push_req_n	1 bit	Input	Stack push request, active low
pop_req_n	1 bit	Input	Stack pop request, active low
we_n	1 bit	Output	Write enable for RAM write port, active low
empty	1 bit	Output	Stack empty flag, active high
full	1 bit	Output	Stack full flag, active high
error	1 bit	Output	Stack error output, active high
wr_addr	ceil($\log_2$[*depth*]) bit(s)	Output	Address output to write port of RAM
rd_addr	ceil($\log_2$[*depth*]) bit(s)	Output	Address output to read port of RAM

Table 1-2 Parameter Description

Parameter	Values	Function
depth	2 to 2^{24} Default: None	Number of memory elements in the stack [used to size the address ports]
err_mode	0 or 1 Default: 0	Error mode 0 = underflow/overflow error, hold until reset, 1 = underflow/overflow error, hold until next clock.
rst_mode	0 or 1 Default: 0	Reset mode 0 = asynchronous reset, 1 = synchronous reset.

Table 1-3 Synthesis Implementations[a]

Implementation Name	Function	License Feature Required
rpl	Ripple carry synthesis model	DesignWare
cl2	Full carry look-ahead model	DesignWare

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use any architectures described in this table. For more, see *DesignWare Building Block IP User Guide*

Table 1-4 Simulation Models

Model	Function
DW03.DW_STACKCTL_CFG_SIM	Design unit name for VHDL simulation
dw/dw03/src/DW_stackctl_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_stackctl.v	Verilog simulation model source code

Table 1-5 Error Mode Description

error_mode	Error Types Detected	Error Output
0	Underflow/Overflow	Latched
1	Underflow/Overflow	Not Latched

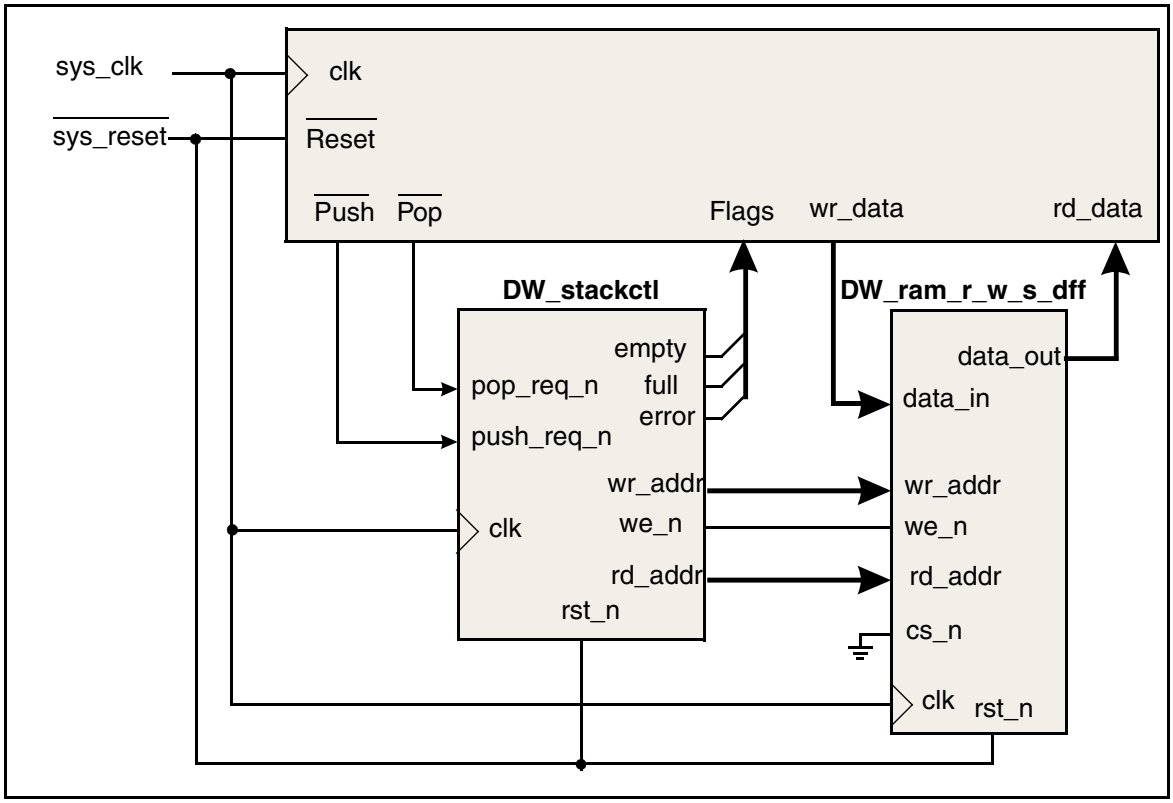
Table 1-6 Push and Pop Operation Function Table

push_req_n	full	pop_req_n	empty	Action	New Error
0	0	X	X	Push operation	No
0	1	X	0	Overrun; incoming data dropped (no action other than error generation)	Yes
1	X	0	0	Pop operation	No
1	0	0	1	Underrun; (no action other than error generation)	Yes
1	X	1	X	No action	No

Table 1-7 Write and Read Address Pointers Relationship

wr_addr	rd_addr	Memory Status
0	0	Empty (zero words in memory)
1	0	One word in memory
K	*K* - 1	K words in memory (1 < *K* < *depth*)
depth - 1	*depth* - 2	*depth* - 1 words in memory
depth - 1	*depth* - 1	full (depth words in memory)

Figure 1-1 Example Usage of DW_stackctl



Writing to the Stack (Push)

wr_addr and we_n
The `wr_addr` and `we_n` output ports of the stack controller provide the write address and synchronous write enable to the stack.

A push is executed when `push_req_n` is asserted (LOW) and the `full` flag is inactive (LOW). Asserting `push_req_n` causes the following events to occur:

- The `we_n` is immediately asserted in preparation for a write to the RAM on the next `clk`,
- The `wr_addr` increments on the next rising edge of `clk` if the stack is not full after pushing, and
- The `rd_addr` increments on the next rising edge of `clk` if the stack was not empty before pushing.

Thus, the RAM is written and `wr_addr` (which always points to the address of the next word to be pushed) is incremented on the same rising edge of `clk` – the first clock after `push_req_n` is asserted. This means that `push_req_n` must be asserted early enough to propagate through the stack controller to the RAM before the next clock.

When the stack is empty, `wr_addr` points at the lowest RAM address. When the stack is full, `wr_addr` points at the highest stack address. When the stack is neither empty nor full, `wr_addr` = `rd_addr` + 1. Refer to Table 1-7 for internal address pointer information.

Write Errors

The `error` output is activated if a push is requested and the stack is full. That is, if

- The `push_req_n` input is asserted (LOW), and
- The `full` flag is active (HIGH)

at the next rising edge of `clk`.

Reading from the Stack (Pop)

The read port of the RAM can be either synchronous or asynchronous. In either case, the `rd_addr` output port of DW_stackctl provides the read address to the RAM. The `rd_addr` output bus always points to, thus prefetches, the next word of RAM read data to be popped.

A pop operation occurs when `pop_req_n` is asserted (LOW), and the stack is not empty. Asserting `pop_req_n` causes the following events to occur:

- The `rd_addr` pointer to be decremented on the next rising edge of `clk` if the stack is not empty after popping, and
- The `wr_addr` pointer to be decremented on the next rising edge of `clk` if the stack was not full before popping.

Thus, the RAM read data must be captured on the `clk` following the assertion of `pop_req_n`. For RAMs with a synchronous read port, the output data is captured in the output stage of the RAM. For RAMs with an asynchronous read port, the output data is captured by the next stage of logic after the stack.

When the stack is empty, `rd_addr` points at the lowest RAM address. When the stack is full, `rd_addr` points at the highest address. When the stack is neither empty nor full, `wr_addr` = `rd_addr` + 1. Refer to Table 1-7 on page 3 for internal address pointer information.

Read Errors

The `error` output is activated if a pop is requested and the stack is empty. That is, if:

- The `pop_req_n` input is active (LOW), and
- The `empty` flag is active (HIGH)

at the next rising edge of `clk`.

Simultaneous Push and Pop

DW_stackctl does not support simultaneous push and pop. If a push and pop occur at the same time when DW_stackctl is not full, only the push occurs, not the pop. DW_stackctl does not give an error. However, with the stack full, DW_stackctl activates the `error` output (due to overflow), and does not push or pop. Also refer to Table 1-7 on page 3.

Reset

rst_mode
The `rst_mode` parameter selects whether the DW_stackctl reset is asynchronous (`rst_mode` = 0) or synchronous (`rst_mode` = 1). If asynchronous mode is selected, asserting `rst_n` (setting it LOW) immediately causes the internal address pointers to be set to 0, and the flags and `error` output to be initialized. If synchronous mode is selected, the address pointers, flags, and `error` output are initialized at the rising edge of `clk`, following the assertion of `rst_n`.

The `error` output and flags are initialized as follows:

- The `empty` flag is initialized to 1, and
- The `full` flag and the `error` output are initialized to 0.

Errors

err_mode
The `err_mode` parameter determines whether the `error` output remains active until reset or for only the clock cycle in which the error is detected.

When `err_mode` = 0, overflow and underflow are detected, and the `error` output (once activated) remains active until reset. When `err_mode` = 1, overflow and underflow are detected, and the `error` output (once activated) remains active only for the clock cycle in which the `error` is detected. Refer to Table 1-5 on page 3 for error mode descriptions.

error
The `error` output indicates a fault in the operation of the stack control logic. There are two possible causes for the `error` output to be activated:

1. Overflow (push while full).
2. Underflow (pop while empty).

The `error` output is set LOW when `rst_n` is applied.

Controller Status Flag Outputs

empty
The `empty` output indicates that there are no words in the stack available to be popped. The `empty` output is set HIGH when `rst_n` is applied.

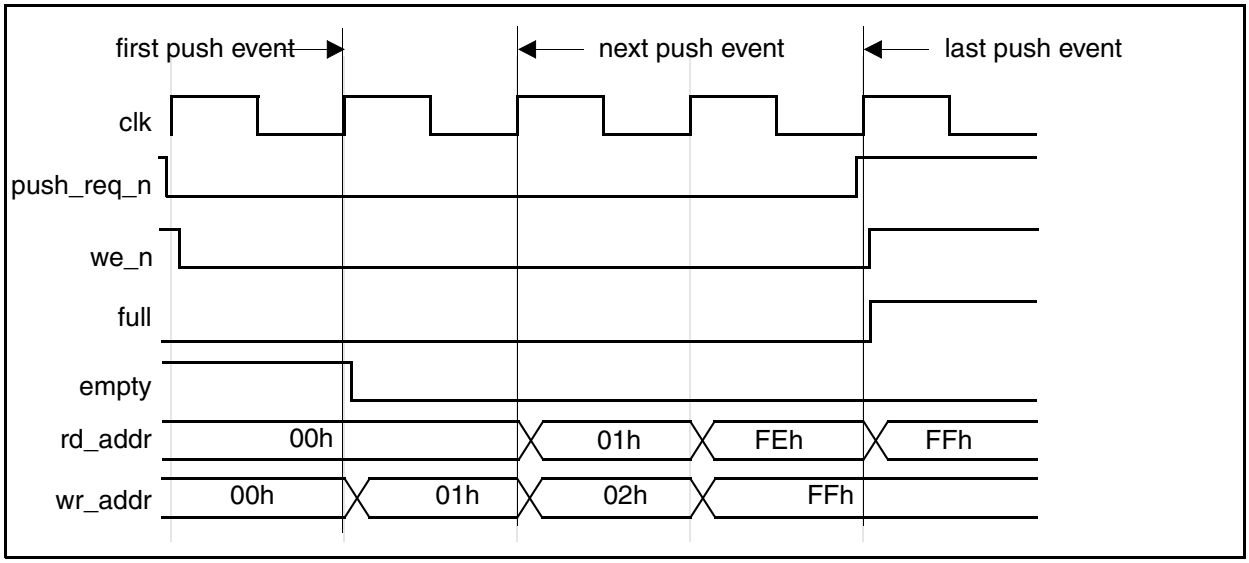
full
The `full` output indicates that the stack is full, and there is no space available for push data. The `full` output is set LOW when `rst_n` is applied.

Timing Waveforms

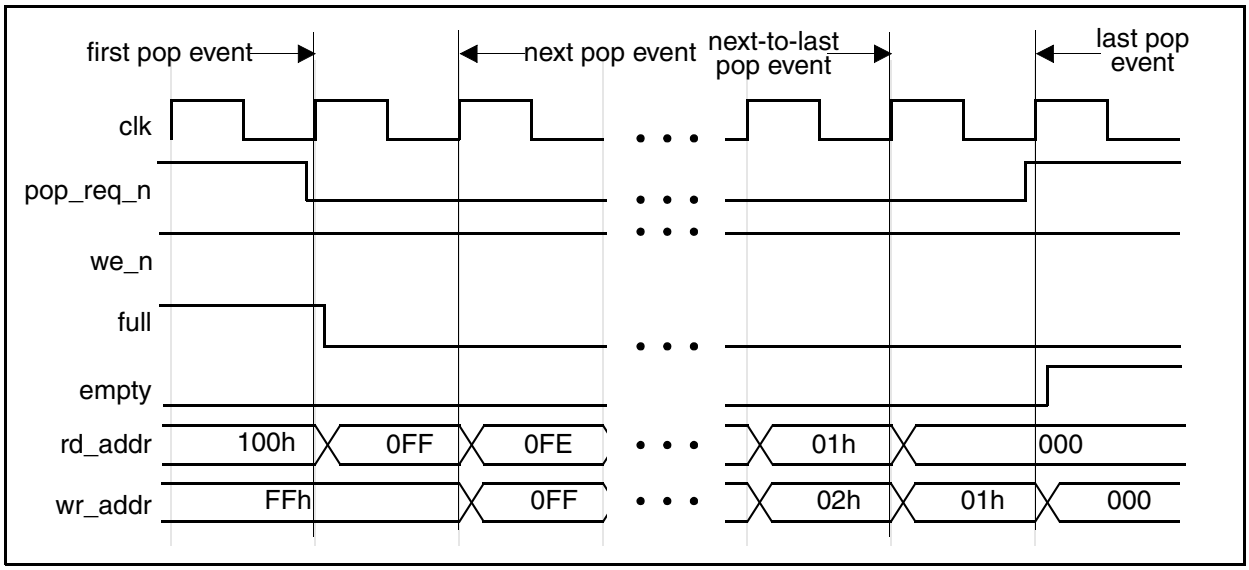
The following figure shows timing diagrams for various conditions of DW_stackctl.

Figure 1-2 Timing Waveforms

Push Operation
pop_req_n = 1, depth = 256



Pop Operation
push_req_n = 1



Related Topics

- Memory – Stacks Overview
- 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_stackctl_inst is
  generic (inst_depth    : INTEGER := 8;
           inst_err_mode : INTEGER := 0;
           inst_rst_mode : INTEGER := 0  );
  port (inst_clk       : in std_logic;
        inst_rst_n     : in std_logic;
        inst_push_req_n: in std_logic;
        inst_pop_req_n : in std_logic;
        we_n_inst   : out std_logic;
        empty_inst  : out std_logic;
        full_inst   : out std_logic;
        error_inst  : out std_logic;
        wr_addr_inst: out std_logic_vector(bit_width(inst_depth)-1 downto 0);
        rd_addr_inst: out std_logic_vector(bit_width(inst_depth)-1 downto 0)
        );
end DW_stackctl_inst;

architecture inst of DW_stackctl_inst is
begin

  -- Instance of DW_stackctl
  U1 : DW_stackctl
    generic map (depth => inst_depth,   err_mode => inst_err_mode,
                 rst_mode => inst_rst_mode )
    port map (clk => inst_clk,   rst_n => inst_rst_n,
              push_req_n => inst_push_req_n,   pop_req_n => inst_pop_req_n,
              we_n => we_n_inst,   empty => empty_inst,   full => full_inst,
              error => error_inst,   wr_addr => wr_addr_inst,
              rd_addr => rd_addr_inst );
end inst;

-- pragma translate_off
configuration DW_stackctl_inst_cfg_inst of DW_stackctl_inst is
  for inst
  end for; -- inst
end DW_stackctl_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_stackctl_inst(inst_clk, inst_rst_n, inst_push_req_n,
                        inst_pop_req_n, we_n_inst, empty_inst, full_inst,
                        error_inst, wr_addr_inst, rd_addr_inst );

  parameter depth = 8;
  parameter err_mode = 0;
  parameter rst_mode = 0;
  `define bit_width_depth 3 // ceil(log2(depth))

  input inst_clk;
  input inst_rst_n;
  input inst_push_req_n;
  input inst_pop_req_n;
  output we_n_inst;
  output empty_inst;
  output full_inst;
  output error_inst;
  output [`bit_width_depth-1 : 0] wr_addr_inst;
  output [`bit_width_depth-1 : 0] rd_addr_inst;

  // Instance of DW_stackctl
  DW_stackctl #(depth, err_mode, rst_mode)
    U1 (.clk(inst_clk),   .rst_n(inst_rst_n),   .push_req_n(inst_push_req_n),
        .pop_req_n(inst_pop_req_n),   .we_n(we_n_inst),   .empty(empty_inst),
        .full(full_inst),   .error(error_inst),   .wr_addr(wr_addr_inst),
        .rd_addr(rd_addr_inst) );
endmodule
```

Copyright Notice and Proprietary Information

© 2018 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

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