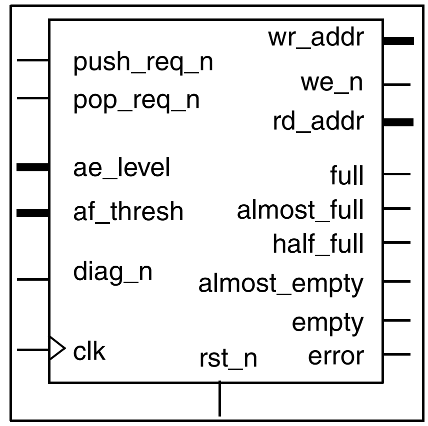
DW_fifoctl_s1_df

Synchronous (Single Clock) FIFO Controller with Dynamic Flags

Version, STAR and Download Information: IP Directory

Features and Benefits

- Fully registered synchronous address and flag output ports
- All operations execute in a single clock cycle
- FIFO empty, half full, and full flags
- FIFO error flag indicating underflow, overflow, and pointer corruption
- Dynamically programmable almost full and almost empty flags
- Parameterized word depth
- Parameterized reset mode (synchronous or asynchronous)
- Interfaces to common hard macro or compiled ASIC dual-port synchronous RAMs
- Provides minPower benefits with the DesignWare-LP license (Get the minPower version of this datasheet.)



Description

DW_fifoctl_s1_df is a FIFO RAM controller designed to interface with a dual-port synchronous RAM. The RAM must have the following:

- A synchronous write port
- Either an asynchronous or synchronous read port

The FIFO controller provides address generation, write-enable logic, flag logic, and operational error detection logic. Parameterizable features include FIFO depth (up to 24 address bits or 16,777,216 locations), level of error detection, 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	FIFO push request, active low

Table 1-1 Pin Description (Continued)

Pin Name	Width	Direction	Function
pop_req_n	1 bit	Input	FIFO pop request, active low
diag_n	1 bit	Input	Diagnostic control for *err_mode* = 0, NC for other *err_mode* values, active low
ae_level	ceil($\log_2$[*depth*]) bit(s)	Input	Almost empty level (the number of words in the FIFO at or below which the `almost_empty` flag is active)
af_thresh	ceil($\log_2$[*depth*]) bit(s)	Input	Almost full threshold (the number of words stored in the FIFO at or above which the `almost_full` flag is active)
we_n	1 bit	Output	Write enable output for write port of RAM, active low
empty	1 bit	Output	FIFO empty output, active high
almost_empty	1 bit	Output	FIFO almost empty output, active high
half_full	1 bit	Output	FIFO half full output, active high
almost_full	1 bit	Output	FIFO almost full output, active high
full	1 bit	Output	FIFO full output, active high
error	1 bit	Output	FIFO 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	Description
depth	2 to 2^{24}	Number of memory elements used in FIFO [used to size the address ports]
err_mode	0 to 2 Default: 0	Error mode 0 = underflow/overflow and pointer latched checking, 1 = underflow/overflow latched checking, 2 = underflow/overflow unlatched checking
rst_mode	0 or 1 Default: 0	Reset mode 0 = asynchronous reset, 1 = synchronous reset

Table 1-3 Synthesis Implementations

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

a. The implementation, “rtl” replaces the obsolete implementations “rpl,” “cl1,” and “cl2.” Information messages listing implementation replacements (SYNDB-37) may be generated by DC at compile time. Existing designs that specify an obsolete implementation (“rpl,” “cl1,” and “cl2”) will automatically have that implementation replaced by the new superseding implementation (“rtl”) noted by an information message (SYNDB-36) generated during DC compilation. The new implementation is capable of producing any of the original architectures automatically based on user constraints.

Table 1-4 Simulation Models

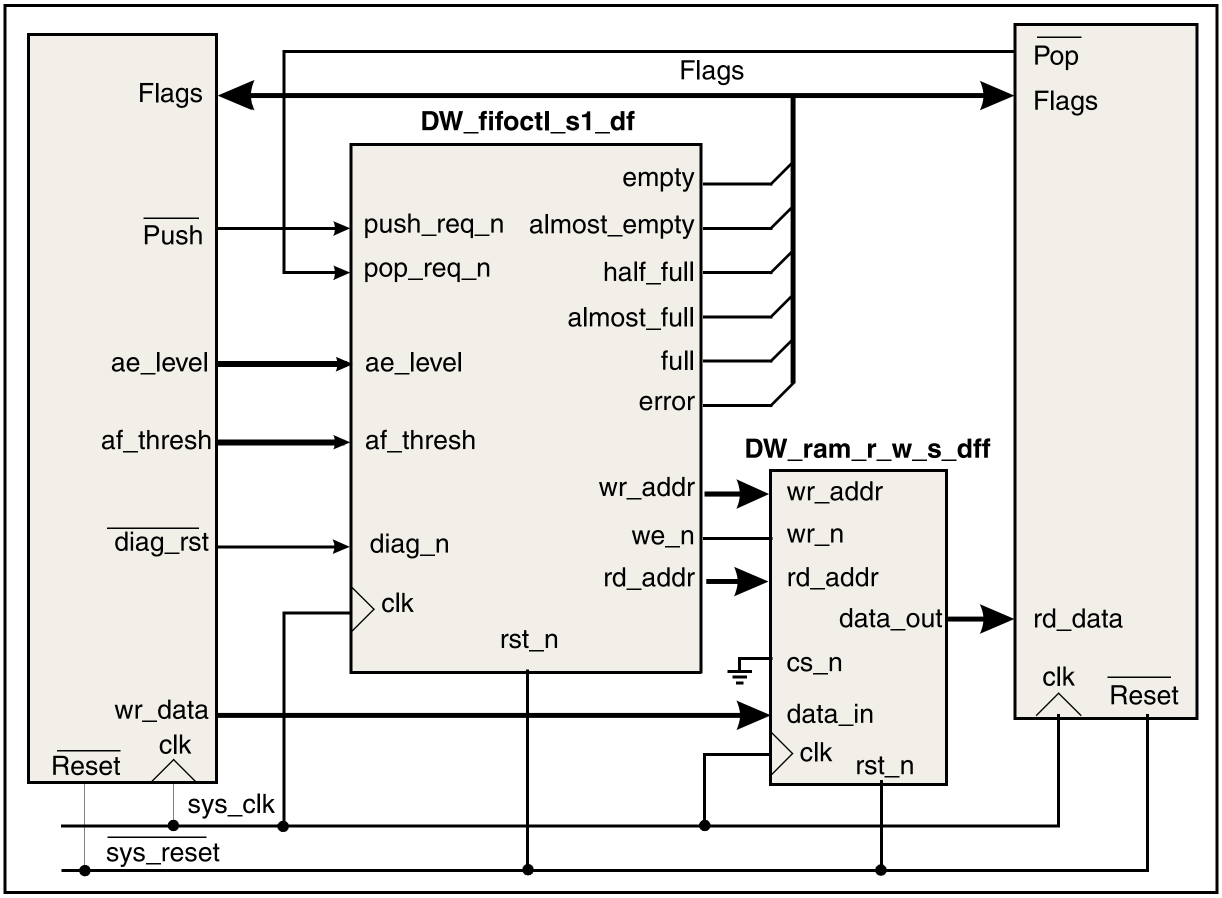
Model	Function
DW03.DW_FIFOCTL_S1_DF_CFG_SIM	Design unit name for VHDL simulation
dw/dw03/src/DW_fifoctl_s1_df_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_fifoctl_s1_df.v	Verilog simulation model source code

Table 1-5 Error Mode Description

error_mode	Error Types Detected	Error Output	diag_n
0	Underflow/Overflow and Pointer Corruption	Latched	Connected
1	Underflow/Overflow	Latched	N/C
2	Underflow/Overflow	Not Latched	N/C

Figure 1-1 shows a typical application of the controller.

Figure 1-1 Example Usage of DW_fifoctl_s1_df



Writing to the FIFO (Push)

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

A push is executed when the `push_req_n` input is asserted (LOW,) and either:

- The `full` flag is inactive (LOW),

or:

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

Thus, a push can occur even if the FIFO is full, as long as a pop is executed in the same cycle.

Asserting `push_req_n` in either of the above cases causes the following events to occur:

- The `we_n` is asserted immediately, preparing for a write to the RAM on the next clock, and
- On the next rising edge of `clk`, `wr_addr` is incremented.

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 FIFO controller to the RAM before the next clock.

An error occurs if a push is attempted while the FIFO is full. That is, if:

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

Reading from the FIFO (Pop)

The read port of the RAM can be either synchronous or asynchronous. In either case, the `rd_addr` output port of the DW_fifoctl_s1_sf 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), as long as the FIFO is not empty. Asserting `pop_req_n` causes the `rd_addr` pointer to be incremented on the next rising edge of `clk`. 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 FIFO.

Refer to the timing diagrams for details of the pop operation for RAMs with synchronous and asynchronous read ports.

An error occurs if:

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

Simultaneous Push and Pop

Push and pop can occur at the same time if there is data in the FIFO, even when the FIFO is full. With the FIFO not empty, `rd_addr` is pointing to the next address to be popped and the pop data is available to be prefetched at the RAM output. When `pop_req_n` and `push_req_n` are both asserted, the following events occur on the next rising edge of `clk`:

- Pop data is captured by the next stage of logic after the FIFO, and
- The new data is pushed into the same location from which the data was popped.

Thus, there is no conflict in a simultaneous push and pop when the FIFO is full. A simultaneous push and pop cannot occur when the FIFO is empty, since there is no pop data to prefetch.

Reset

rst_mode

This parameter selects whether 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 outputs to be initialized. If synchronous mode is selected, the address pointers, flags, and error outputs are initialized at the rising edge of `clk` after `rst_n` is asserted.

The error outputs and flags are initialized as follows:

- The `empty` and `almost_empty` are initialized to 1, and
- All other flags and the `error` output are initialized to 0.

Errors

err_mode

The `err_mode` parameter determines which possible fault conditions are detected, and whether the `error` output remains active until reset or for only the clock cycle in which the error was detected.

When the `err_mode` parameter is set to 0 at design time, the `diag_n` input provides an unconditional synchronous reset to the value of the `rd_addr` output port. This can be used to intentionally cause the FIFO address pointers to become corrupted, forcing a pointer inconsistency-type error.

For normal operation when `err_mode` = 0, `diag_n` should be driven inactive (HIGH). When the `err_mode` parameter is set to 1 or 2, the `diag_n` input is ignored (unconnected).

error

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

1. Overflow (push and no pop while full).
2. Underflow (pop while empty).
3. Empty pointer mismatch (`rd_addr` ≠ `wr_addr` when empty).
4. Full pointer mismatch (`rd_addr` ≠ `wr_addr` when full).
5. In between pointer mismatch (`rd_addr` = `wr_addr` when neither empty nor full).

When `err_mode` = 0, all five causes are detected, and the `error` output (once activated) remains active until reset. When `err_mode` = 1, only causes 1 and 2 are detected, and the `error` output (once activated) remains active until reset. When `err_mode` = 2, only causes 1 and 2 are detected, and the `error` output only stays active for the clock cycle in which the `error` is detected. Refer to Table 1-5 on page 3 for error mode descriptions. The `error` output is set LOW when `rst_n` is applied.

Controller Status Flag Outputs

Refer to Figure 1-2 on page 8 for operation of the status flags.

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

almost_empty
The `almost_empty` output is asserted when there are no more than `ae_level` words currently in the FIFO available to be popped. The value present on the `ae_level` port defines the almost empty threshold. The `almost_empty` output is updated only on the rising edge of `clk`. This signal is useful for preventing the FIFO from underflowing. The `almost_empty` output is set HIGH when `rst_n` is applied.

half_full
The `half_full` output is active HIGH when at least half the FIFO memory locations are occupied. The `half_full` output is set LOW when `rst_n` is applied.

almost_full
The `almost_full` output is asserted when there are no more than *depth* – *af_thresh* empty locations in the FIFO. The value present on the `af_thresh` port defines the almost full threshold. The `almost_full` output is updated only on the rising edge of `clk`. This signal is useful for preventing the FIFO from overflowing. The `almost_full` output is set LOW when `rst_n` is applied.

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

Application Notes

The `ae_level` value is supplied by the application and is chosen:

- To allow input flow control logic to interrupt the pushing of data into the FIFO, or
- To give output flow control logic enough time to begin popping data.

Systems can characterize their own response times dynamically against the data stream. This allows you to set the `ae_level` as tight as practical on the fly for optimal utilization of FIFO memory.

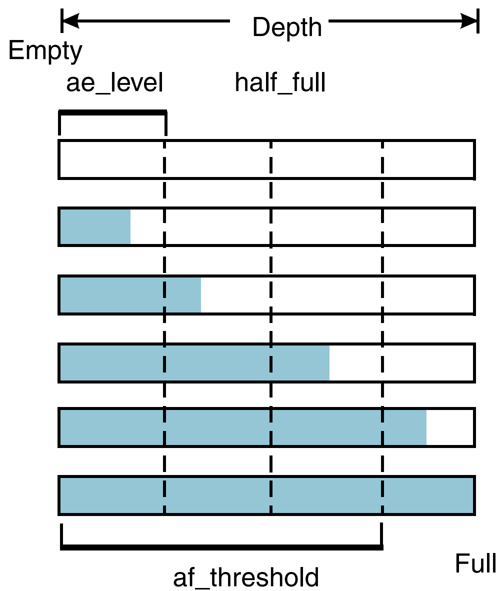
The `af_thresh` value is supplied by the application and is chosen:

- To give output flow control logic enough time to begin popping data, or
- To allow input flow control logic to interrupt the pushing of data into the FIFO.

Systems can characterize their own response times dynamically against the data stream. This allows you to set the `almost_full` flag trip point on the fly for optimal utilization of FIFO memory.

Figure 1-2 shows the status flags of the DW_fifoctl_s1_df FIFO controller at various FIFO storage levels.

Figure 1-2 DW_fifoctl_s1_df FIFO Status Flags



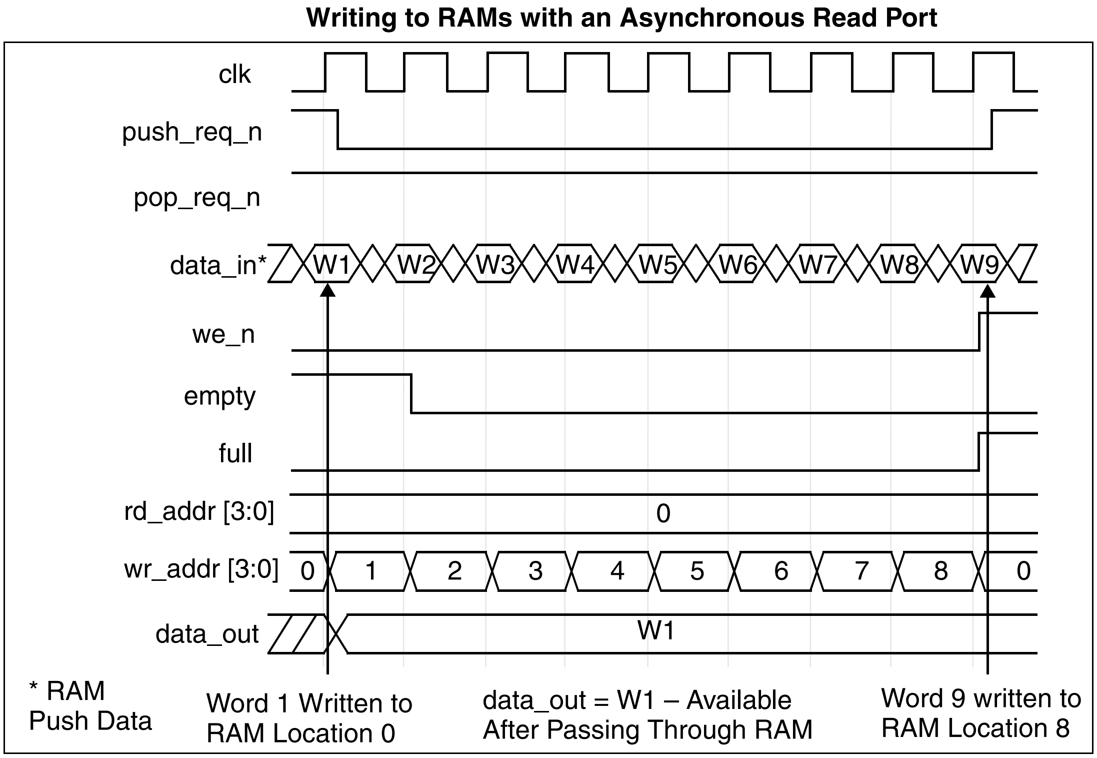
Status Flags

empty	almost_empty	half_full	almost_full	full
1	1	0	0	0
0	1	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	1	1	0
0	0	1	1	1

Timing Waveforms

The figures in this section show timing diagrams for various conditions of DW_fifoctl_s1_df.

Figure 1-3 Push and Pop Timing Waveforms (Asynchronous Read Port RAMs)



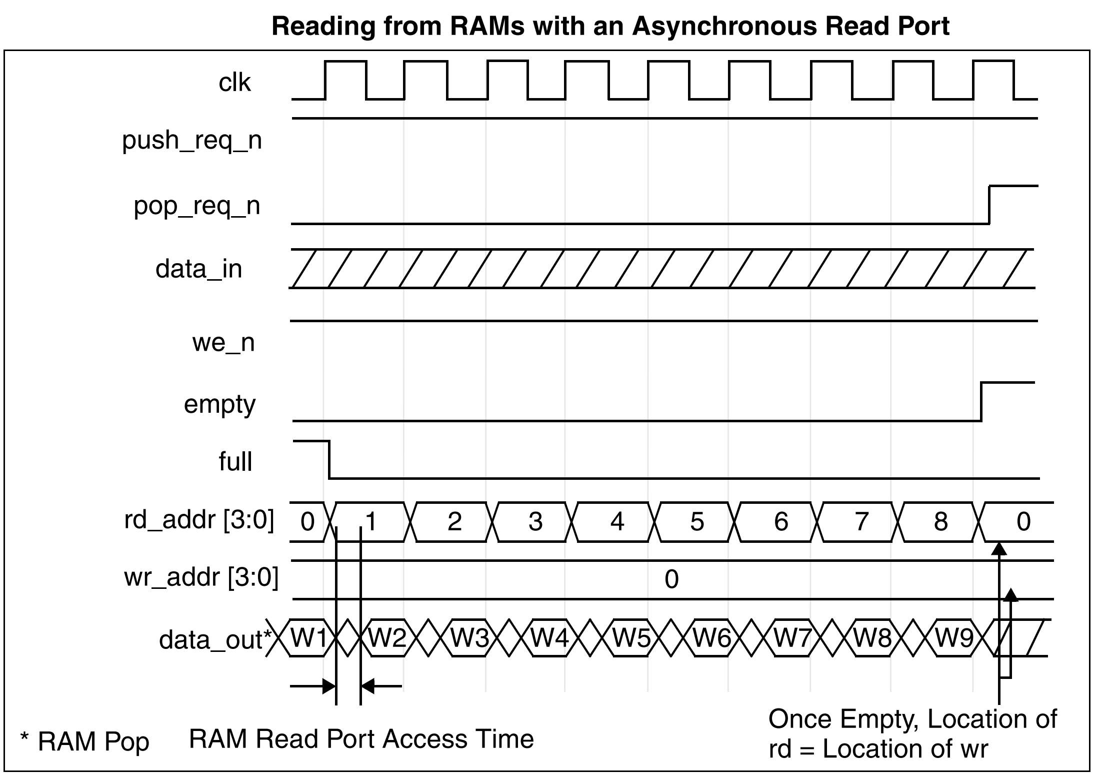
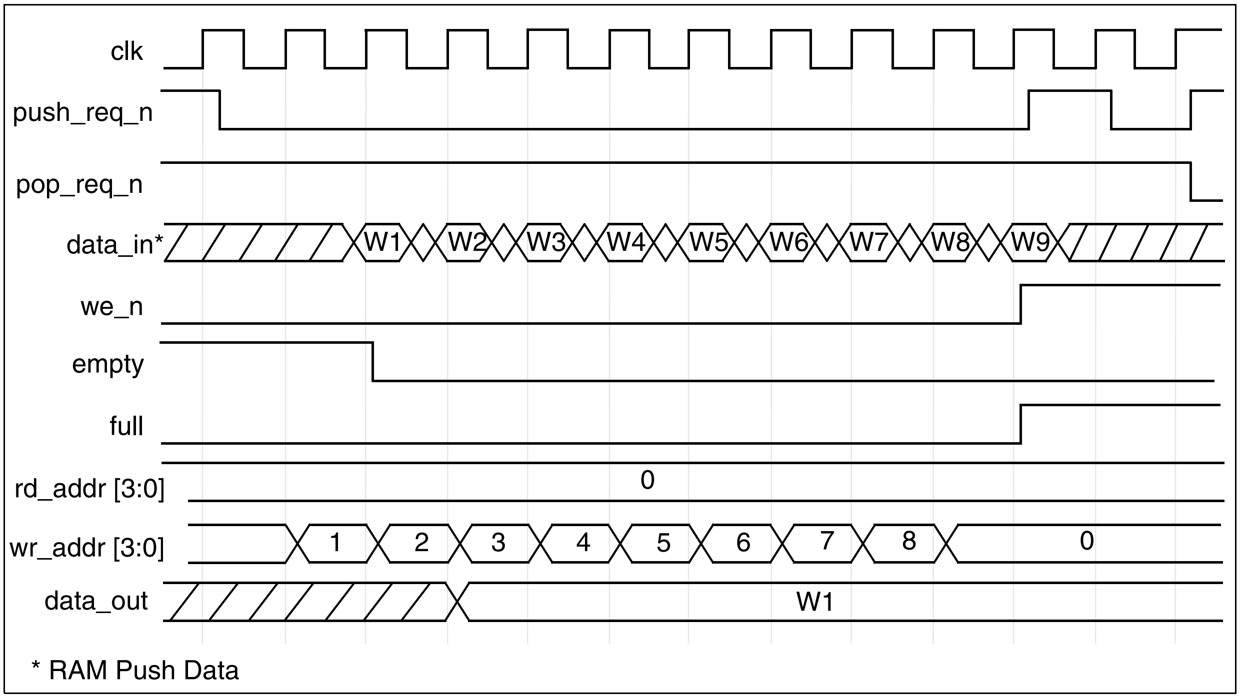


Figure 1-4 Push and Pop Timing Waveforms (Synchronous Read Port RAMs)

Writing to RAMs with a Synchronous Read Port



Reading from RAMs with a Synchronous Read Port

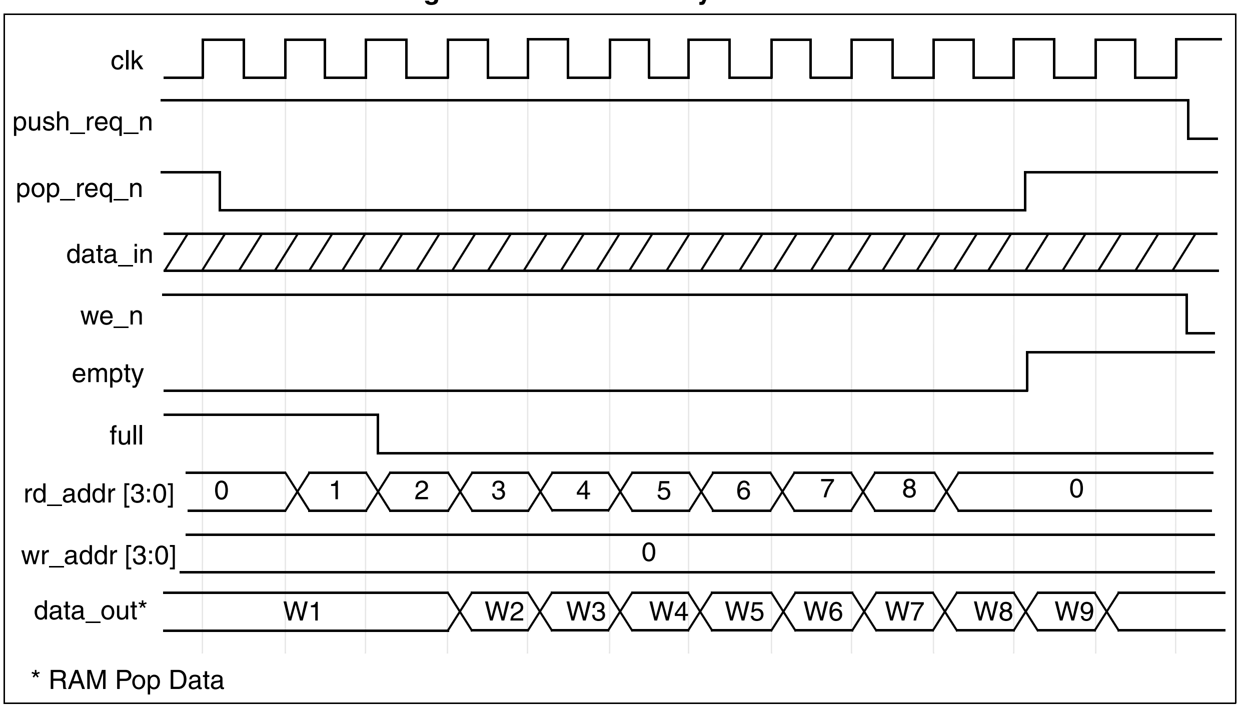


Figure 1-5 Status Flag Timing Waveforms While Pushing

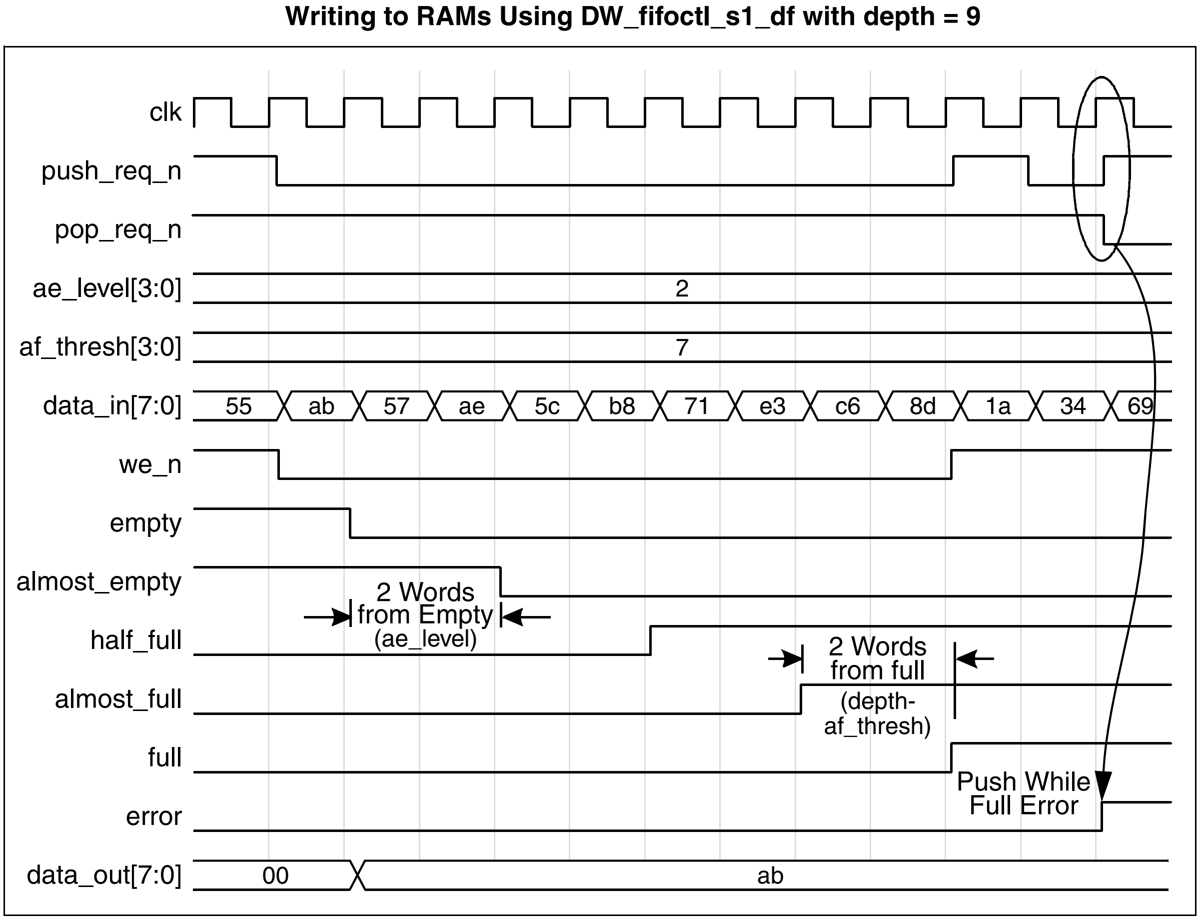


Figure 1-6 Status Flag Timing Waveforms While Popping

Reading from RAMs Using DW_fifoctl_s1_df with depth = 9

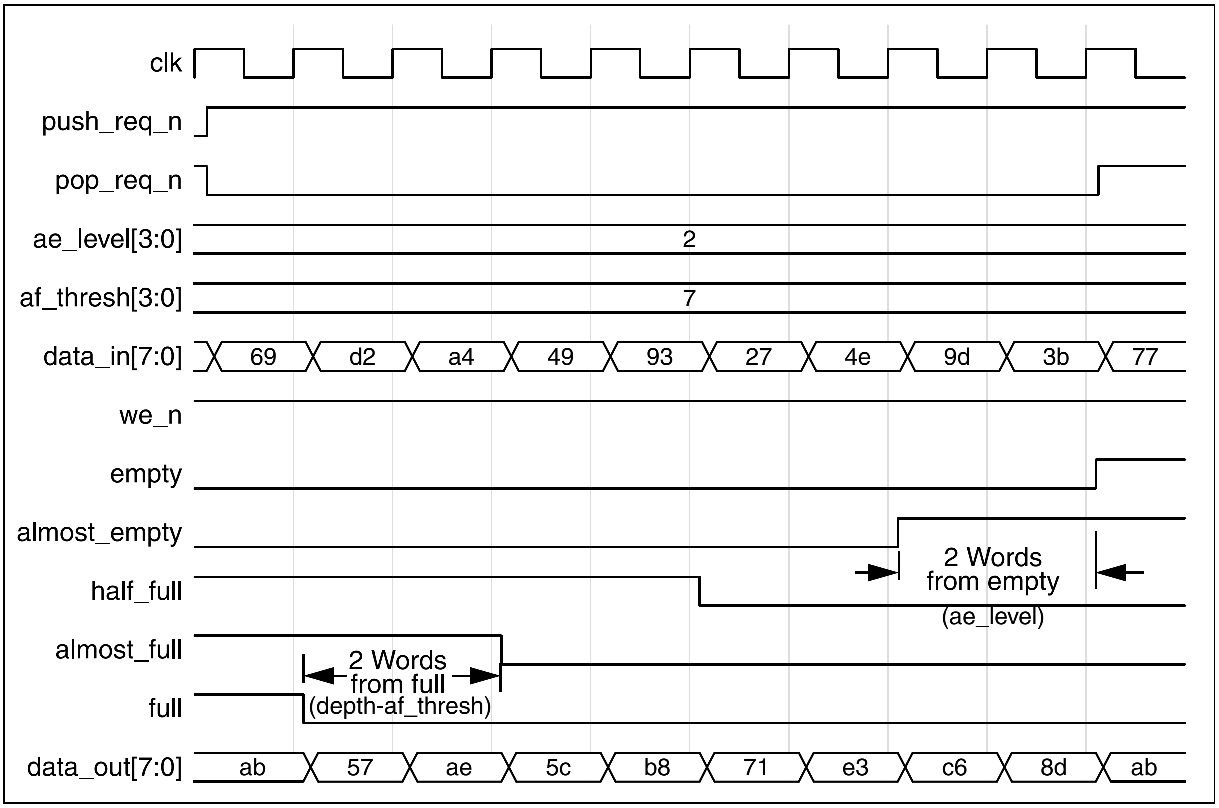


Figure 1-7 Status Flag Timing Waveforms for ae_level and af_thresh Inputs

DW_fifoctl_s1_df Timing on ae_level and af_thresh Inputs (Synchronous Inputs)

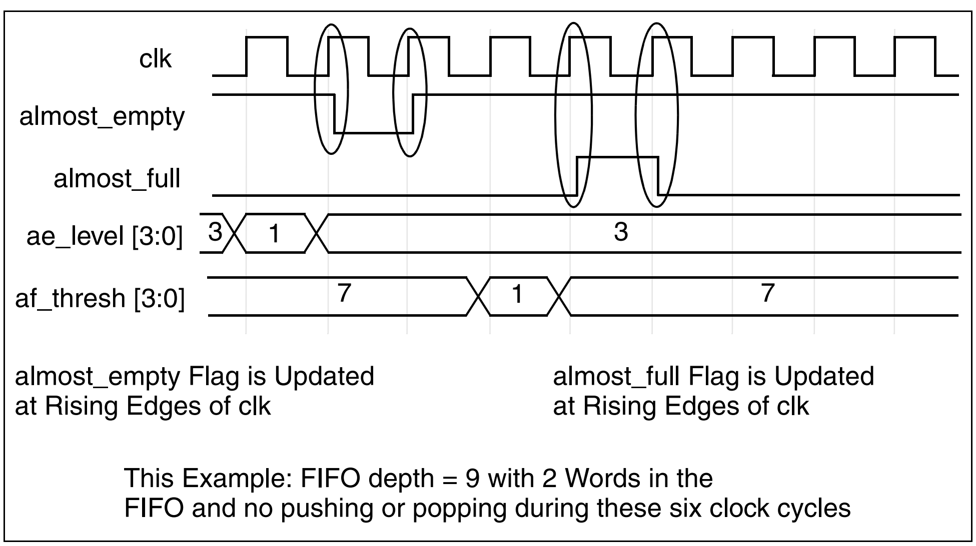
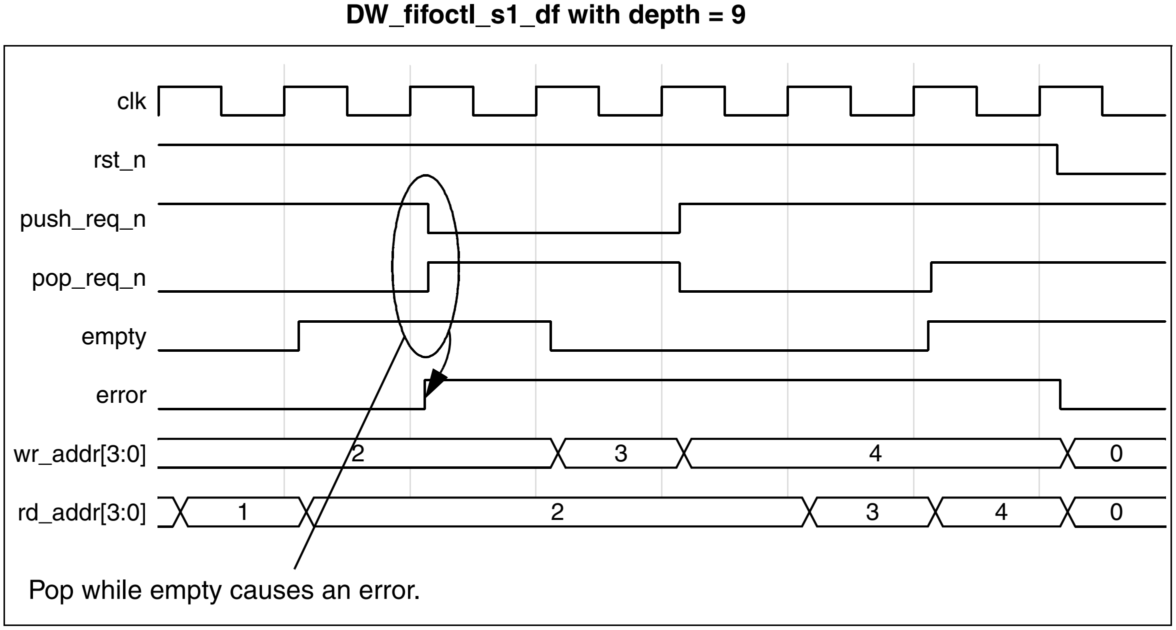


Figure 1-8 Error Flag Timing Waveforms



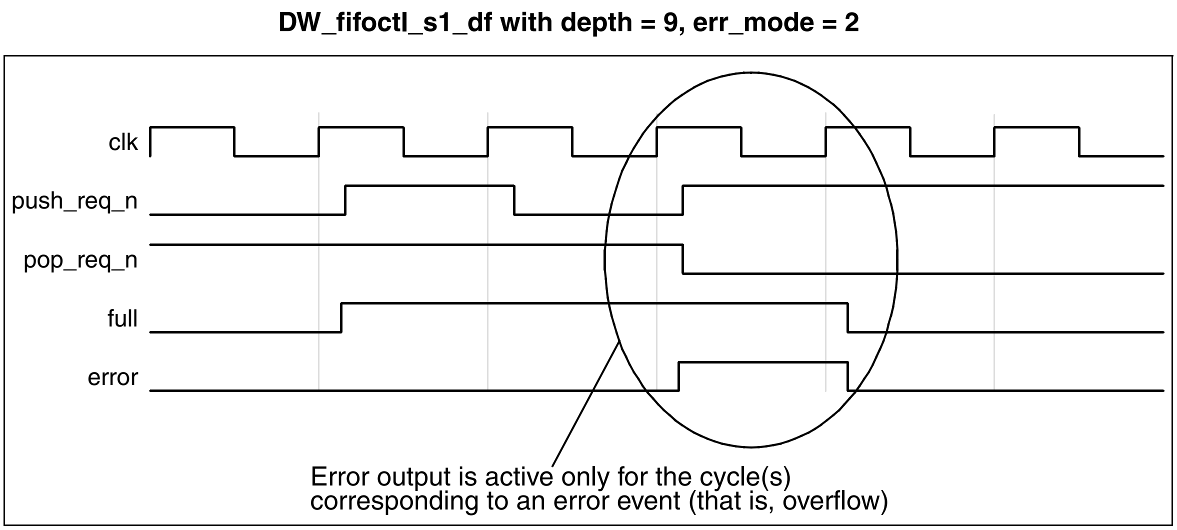


Figure 1-9 Error Flag Timing Waveforms (continued)

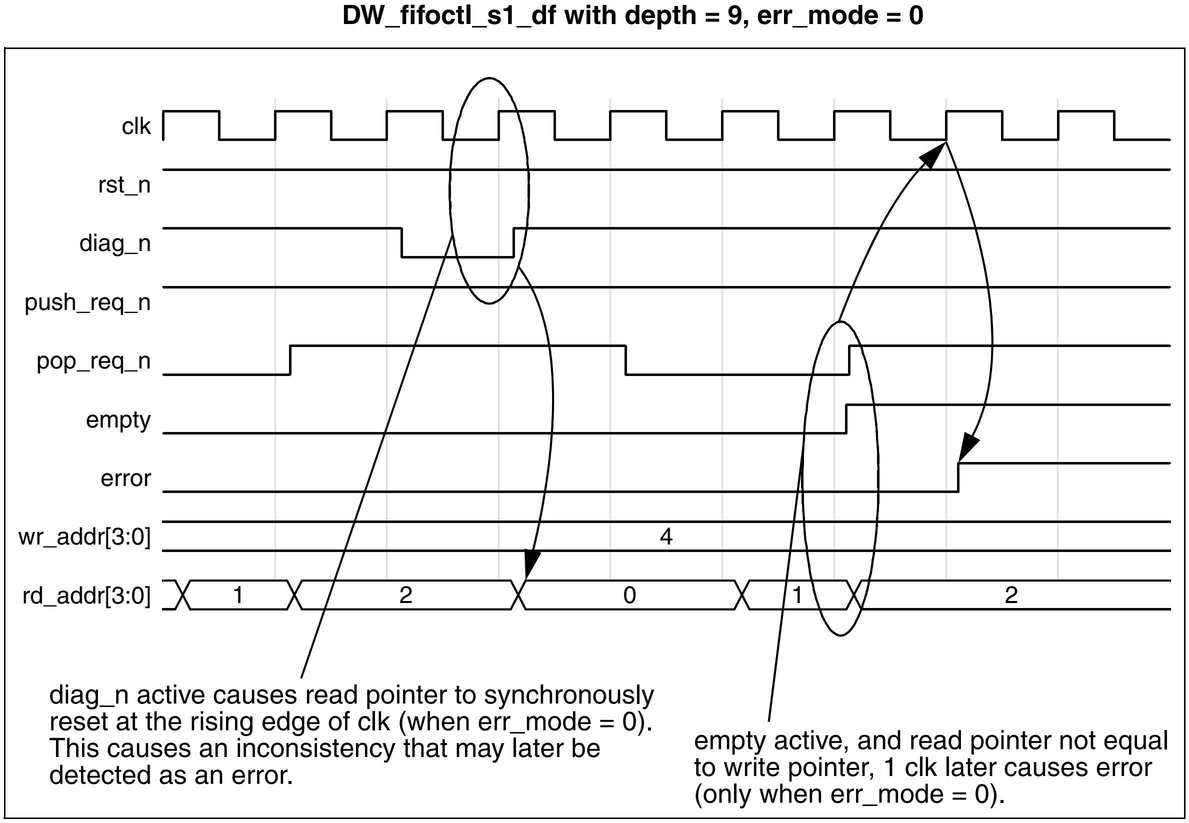


Figure 1-10 Error Flag Timing Waveforms (continued)

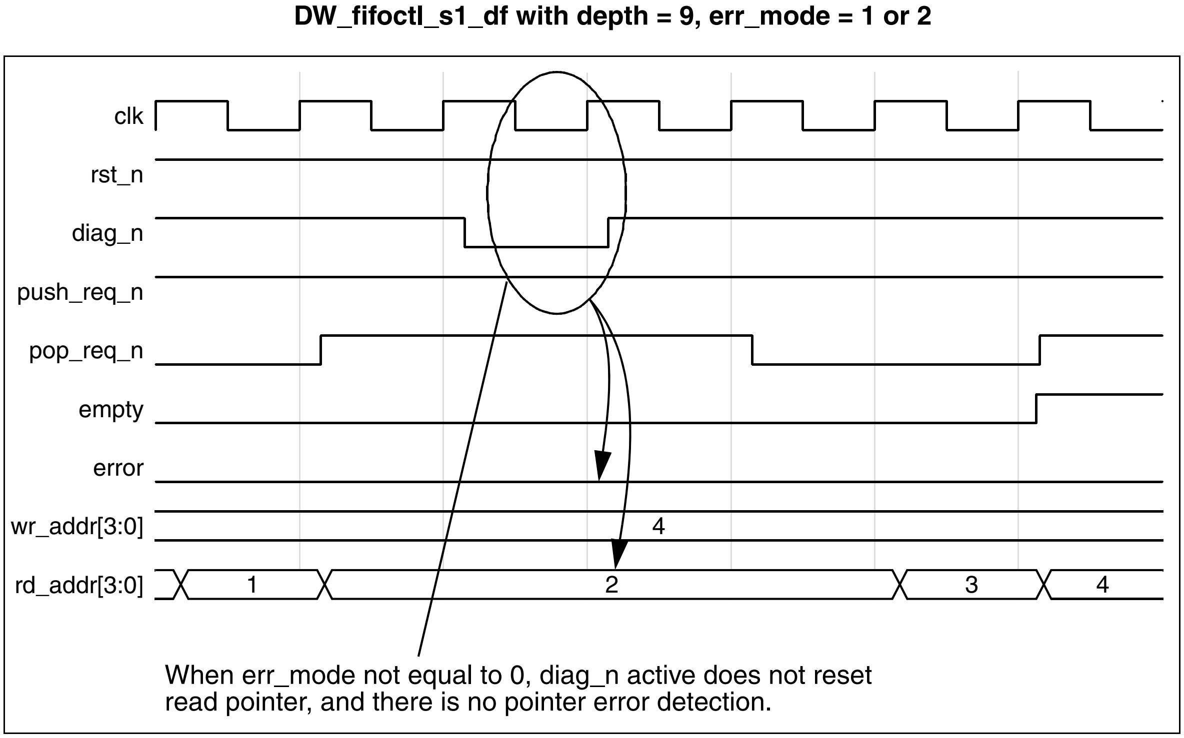
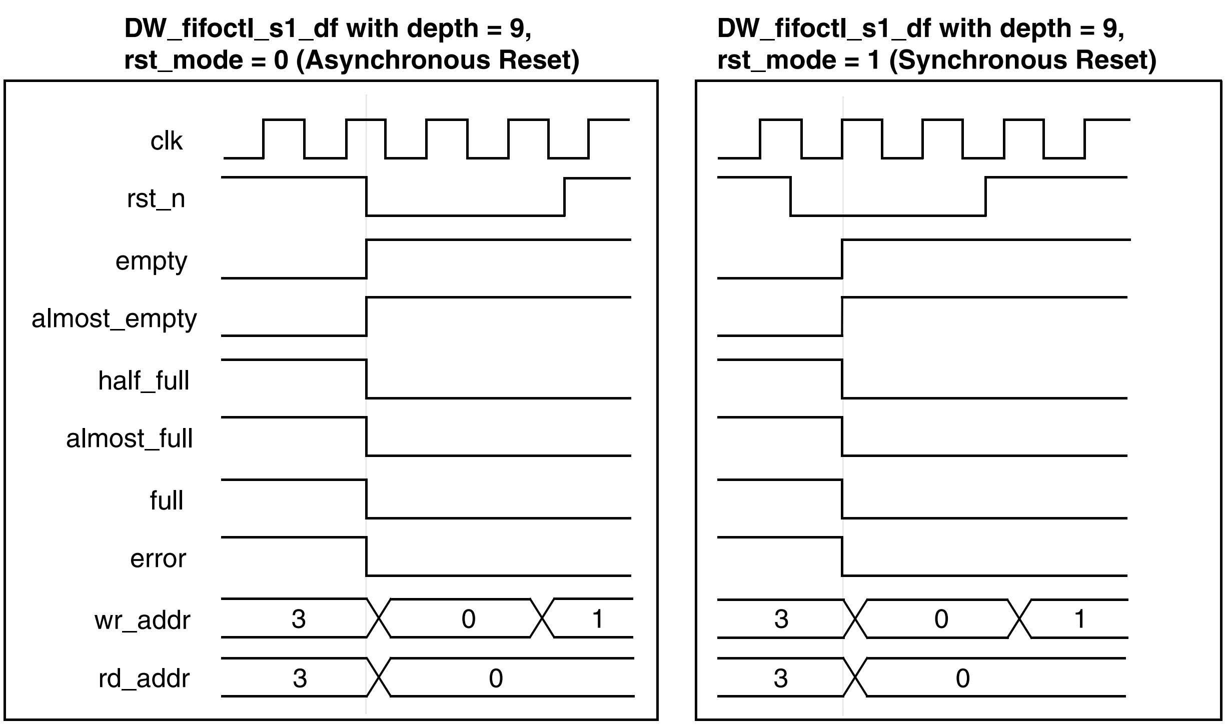


Figure 1-11 Reset Timing Waveforms



Related Topics

- Memory – FIFO 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_fifoctl_s1_df_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;
        inst_diag_n       : in std_logic;
        inst_ae_level     : in std_logic_vector(bit_width(inst_depth)-1
                                                               downto 0);
        inst_af_thresh    : in std_logic_vector(bit_width(inst_depth)-1
                                                               downto 0);
        we_n_inst         : out std_logic;
        empty_inst        : out std_logic;
        almost_empty_inst : out std_logic;
        half_full_inst    : out std_logic;
        almost_full_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_fifoctl_s1_df_inst;

architecture inst of DW_fifoctl_s1_df_inst is
begin
```

```
    -- Instance of DW_fifoctl_s1_df
    U1 : DW_fifoctl_s1_df
      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,
                diag_n => inst_diag_n,   ae_level => inst_ae_level,
                af_thresh => inst_af_thresh,   we_n => we_n_inst,
                empty => empty_inst,   almost_empty => almost_empty_inst,
                half_full => half_full_inst,   almost_full => almost_full_inst,
                full => full_inst,   error => error_inst,
                wr_addr => wr_addr_inst,   rd_addr => rd_addr_inst );
  end inst;

  -- pragma translate_off
  configuration DW_fifoctl_s1_df_inst_cfg_inst of DW_fifoctl_s1_df_inst is
    for inst
    end for; -- inst
  end DW_fifoctl_s1_df_inst_cfg_inst;
  -- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_fifoctl_s1_df_inst(inst_clk, inst_rst_n, inst_push_req_n,
                             inst_pop_req_n, inst_diag_n, inst_ae_level,
                             inst_af_thresh, we_n_inst, empty_inst,
                             almost_empty_inst, half_full_inst,
                             almost_full_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;
  input inst_diag_n;
  input [`bit_width_depth-1 : 0] inst_ae_level;
  input [`bit_width_depth-1 : 0] inst_af_thresh;
  output we_n_inst;
  output empty_inst;
  output almost_empty_inst;
  output half_full_inst;
  output almost_full_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_fifoctl_s1_df
  DW_fifoctl_s1_df #(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),   .diag_n(inst_diag_n),
        .ae_level(inst_ae_level),   .af_thresh(inst_af_thresh),
        .we_n(we_n_inst),   .empty(empty_inst),
        .almost_empty(almost_empty_inst),   .half_full(half_full_inst),
         .almost_full(almost_full_inst),   .full(full_inst),
         .error(error_inst),   .wr_addr(wr_addr_inst),
         .rd_addr(rd_addr_inst)  );
endmodule
```

Copyright Notice and Proprietary Information

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

www.synopsys.com