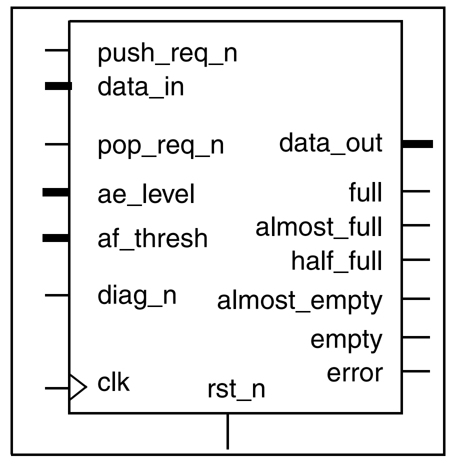
DW_fifo_s1_df

Synchronous (Single Clock) FIFO with Dynamic Flags

Version, STAR and Download Information: IP Directory

Features and Benefits



- Fully registered synchronous flag output ports
- D flip-flop-based memory array for high testability
- 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 width
- Parameterized word depth
- Parameterized reset mode (synchronous or asynchronous, memory array initialized or not)

Description

DW_fifo_s1_df is a fully synchronous, single-clocked FIFO. It combines the DW_fifoctl_s1_df FIFO controller and the DW_ram_r_w_s_dff flip-flop-based RAM DesignWare Building Blocks.

The FIFO provides parameterized width and depth, and a full complement of flags: full, almost full, half full, almost empty, empty, and error.

Reset can be selected at instantiation to be either synchronous or asynchronous, and can either include or exclude the RAM array.

The DW_fifo_s1_df is recommended for relatively small configurations. For large FIFOs, you should consider using the DW_fifoctl_s1_df in conjunction with a compiled, full-custom RAM array.

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 or 2, synchronous if *rst_mode* = 1 or 3)
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, 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)
data_in	*width* bit(s)	Input	FIFO data to push
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
data_out	*width* bit(s)	Output	FIFO data to pop

Table 1-2 Parameter Description

Parameter	Values	Description
width	1 to 2048, Default: 8	Width of data_in and data_out buses
depth	2 to 1024, Default: 4	Number of memory elements used in FIFO
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 to 3 Default: 0	Reset mode 0 = asynchronous reset including memory, 1 = synchronous reset including memory, 2 = asynchronous reset excluding memory, 3 = synchronous reset excluding memory

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
DW06.DW_FIFO_S1_DF_CFG_SIM	Design unit name for VHDL simulation
dw/dw06/src/DW_fifo_s1_df_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_fifo_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

Writing to the FIFO (Push)

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 data at the `data_in` port to be written to the next available location in the FIFO. This write occurs on the `clk` following the assertion of `push_req_n`. The data at the `data_in` port must be stable for a setup time before the rising edge of `clk`.

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)

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 internal read 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`.

Refer to the timing diagrams for details of the pop operation.

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, the internal read pointer points to the next address to be popped, and the pop data is available at the `data_out` 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. However, push data is captured in the FIFO.

Reset

rst_mode
This parameter selects whether reset is asynchronous (*rst_mode* = 0 or 2) or synchronous (*rst_mode* = 1 or 3). If an 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 a synchronous mode is selected, the address pointers, flags, and error outputs are initialized at the rising edge of `clk` after the assertion of `rst_n`.

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.

If *rst_mode* = 0 or 1, the RAM array is also initialized when `rst_n` is asserted. If *rst_mode* = 2 or 3, only the address pointers, and error and flag outputs are initialized; the RAM array is not initialized.

Errors

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 is 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 read pointer. 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 as follows:

1. Overflow (push and no pop while full).
2. Underflow (pop while empty).
3. Empty pointer mismatch (read pointer ≠ write pointer when empty).
4. Full pointer mismatch (read pointer ≠ write pointer when full).
5. In between pointer mismatch (read pointer = write pointer 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 *err_mode* descriptions. The error output is set LOW when rst_n is applied.

Controller Status Flag Outputs

Refer to Figure 1-1 on page 6 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 time 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.

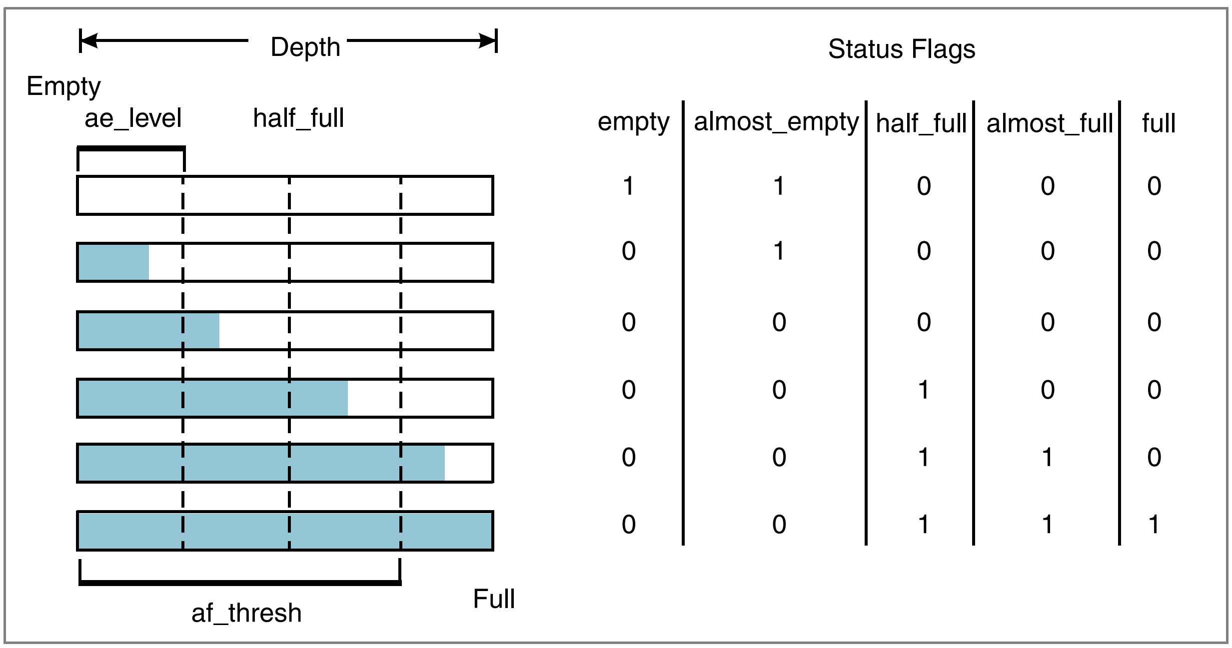
The `af_thresh` value is supplied by the application, and is chosen:

- 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 time 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 1-1 on page 6 shows the status flags of the DW_fifo_s1_df FIFO at various FIFO storage levels.

Figure 1-1 DW_fifo_s1_df FIFO Status Flags



Timing Waveforms

The following figures show timing diagrams for various conditions of DW_fifo_s1_df.

Figure 1-2 Status Flag Timing Waveforms While Pushing

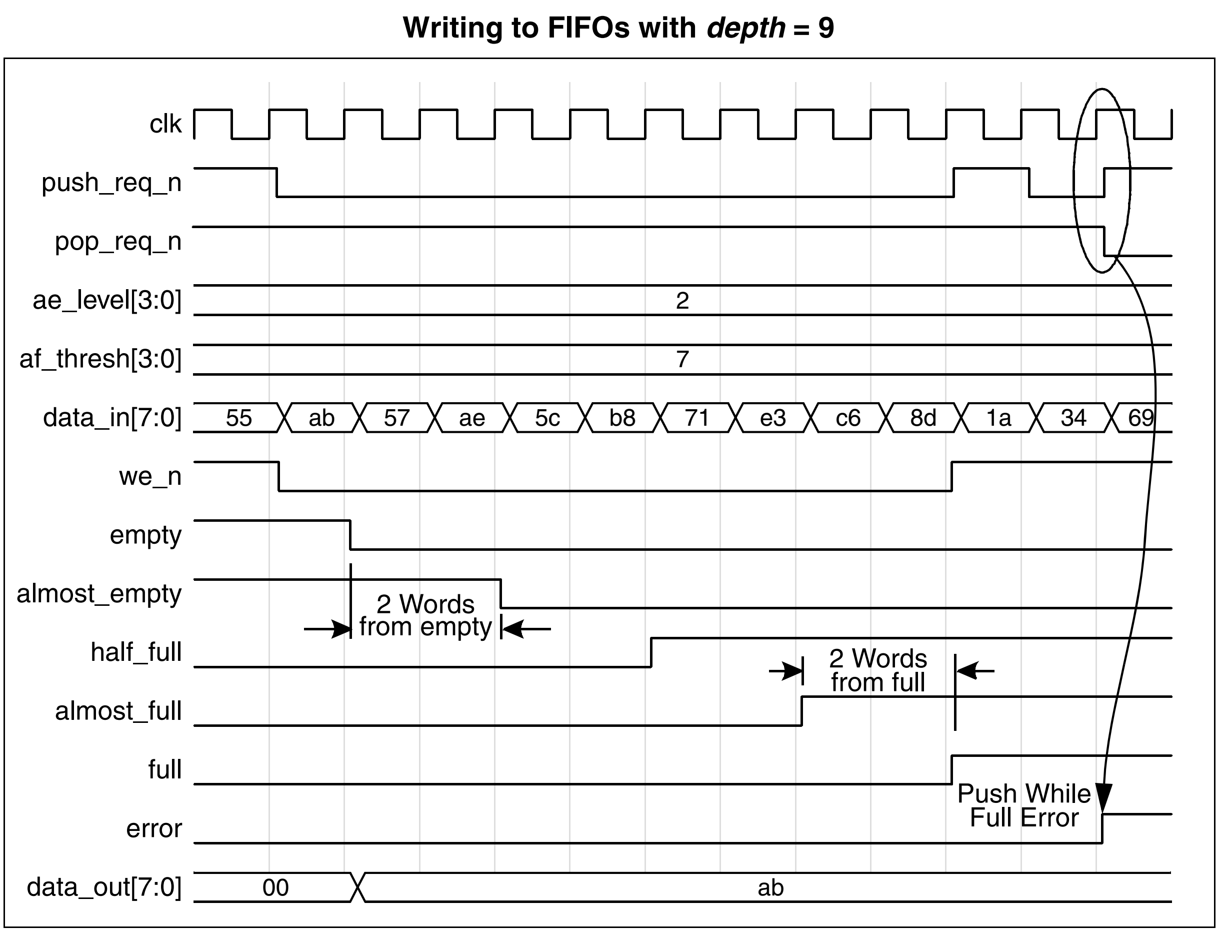


Figure 1-3 Status Flag Timing Waveforms While Popping

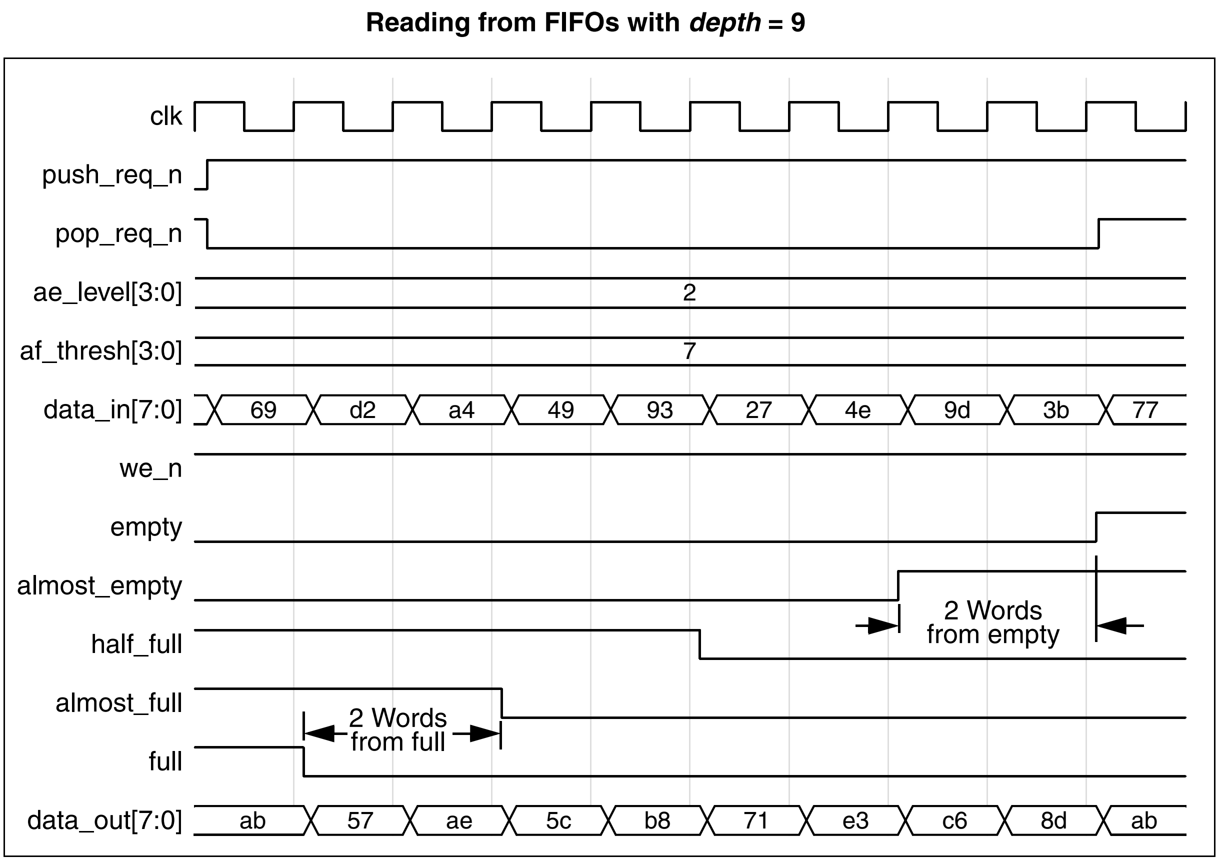


Figure 1-4 Status Flag Timing Waveforms for ae_level and af_thresh Inputs

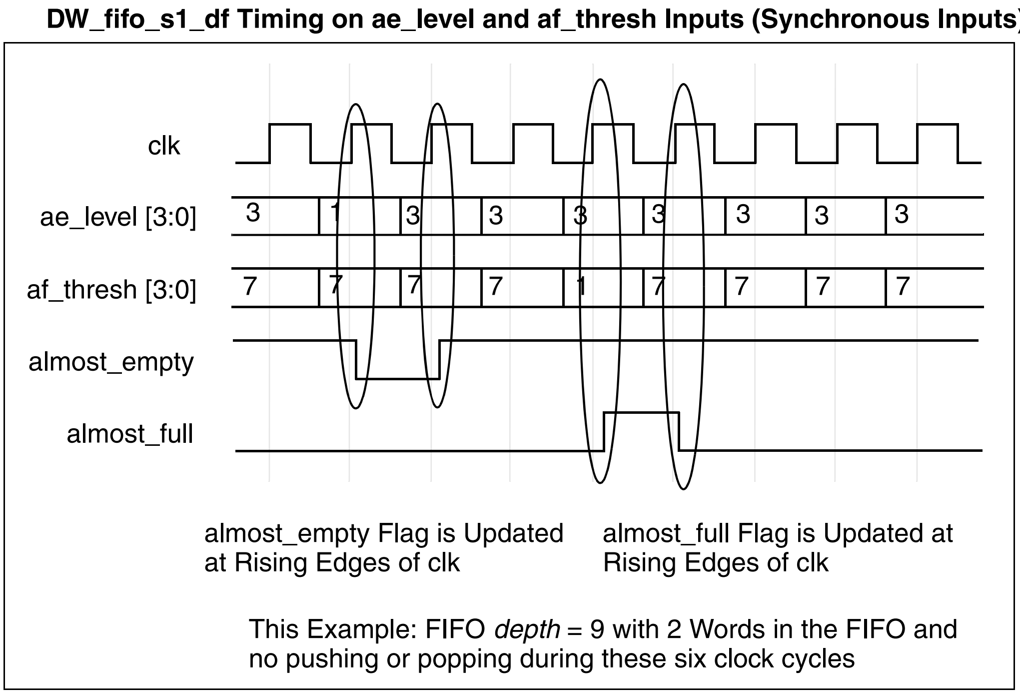
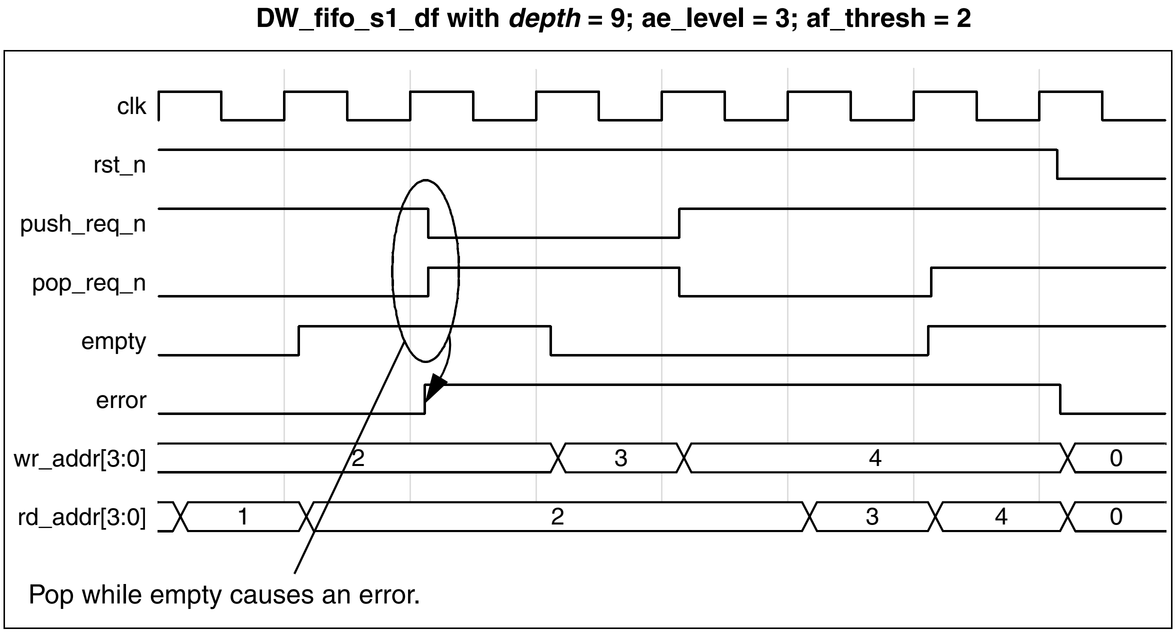


Figure 1-5 Error Flag Timing Waveforms



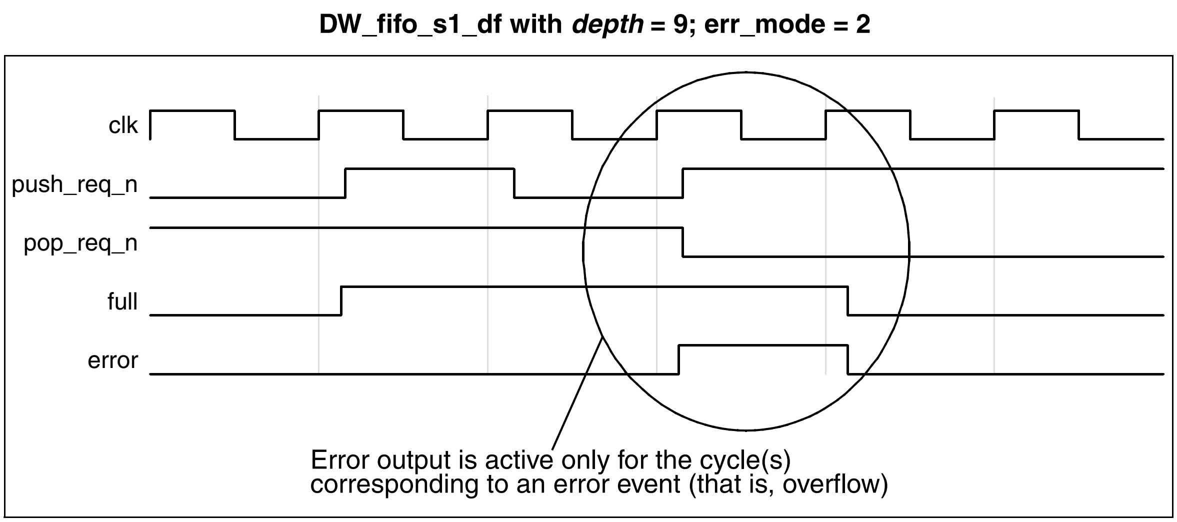


Figure 1-6 Error Flag Timing Waveforms (continued)

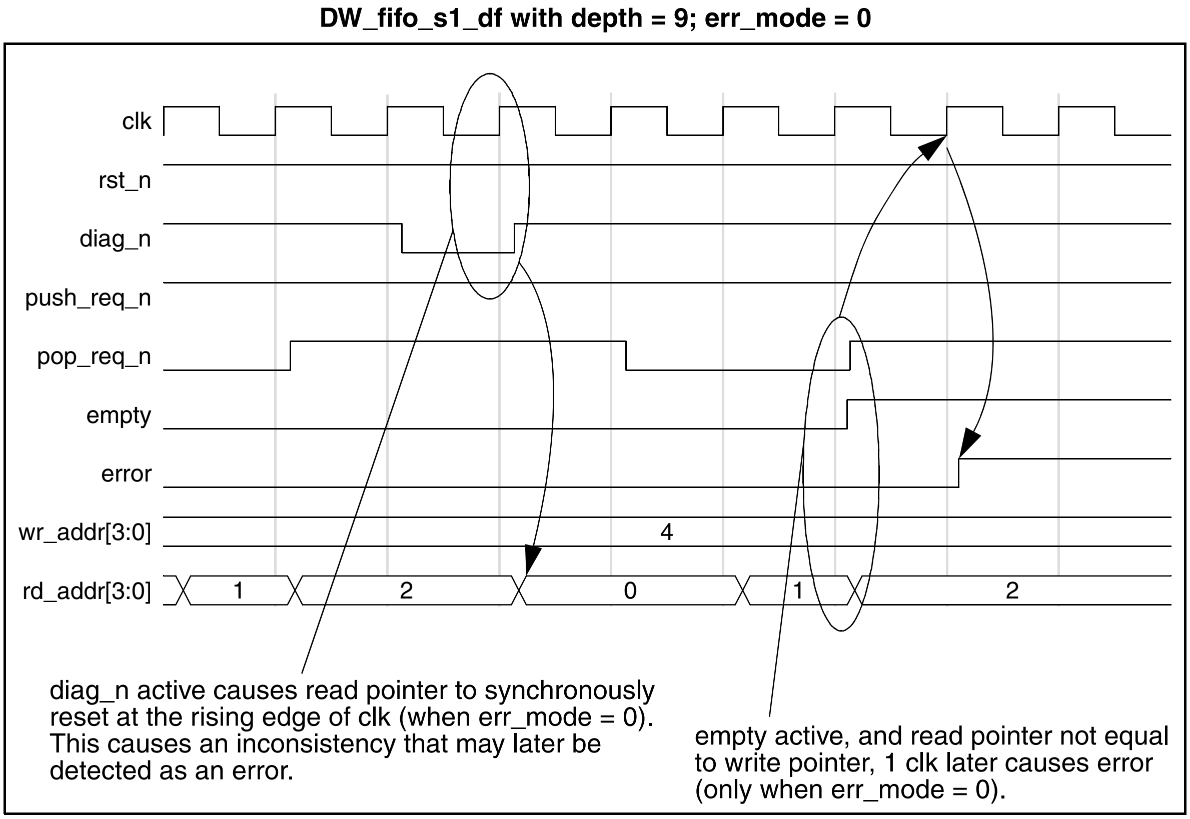


Figure 1-7 Error Flag Timing Waveforms (continued)

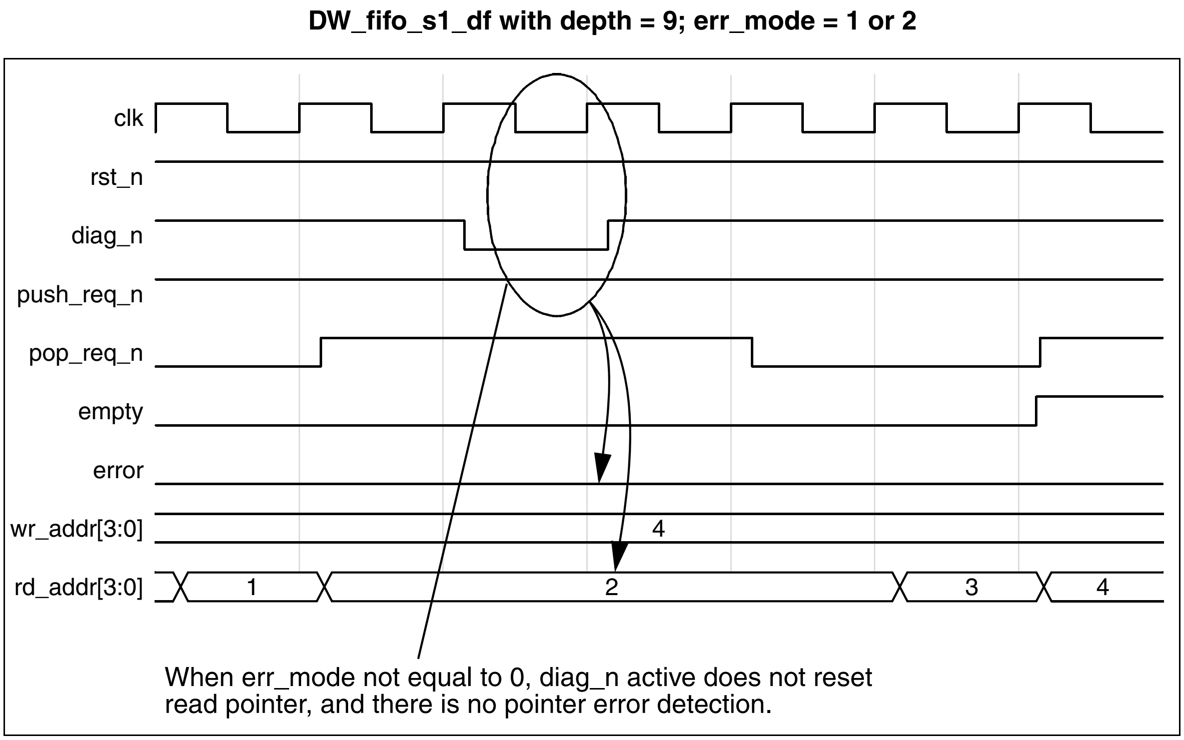
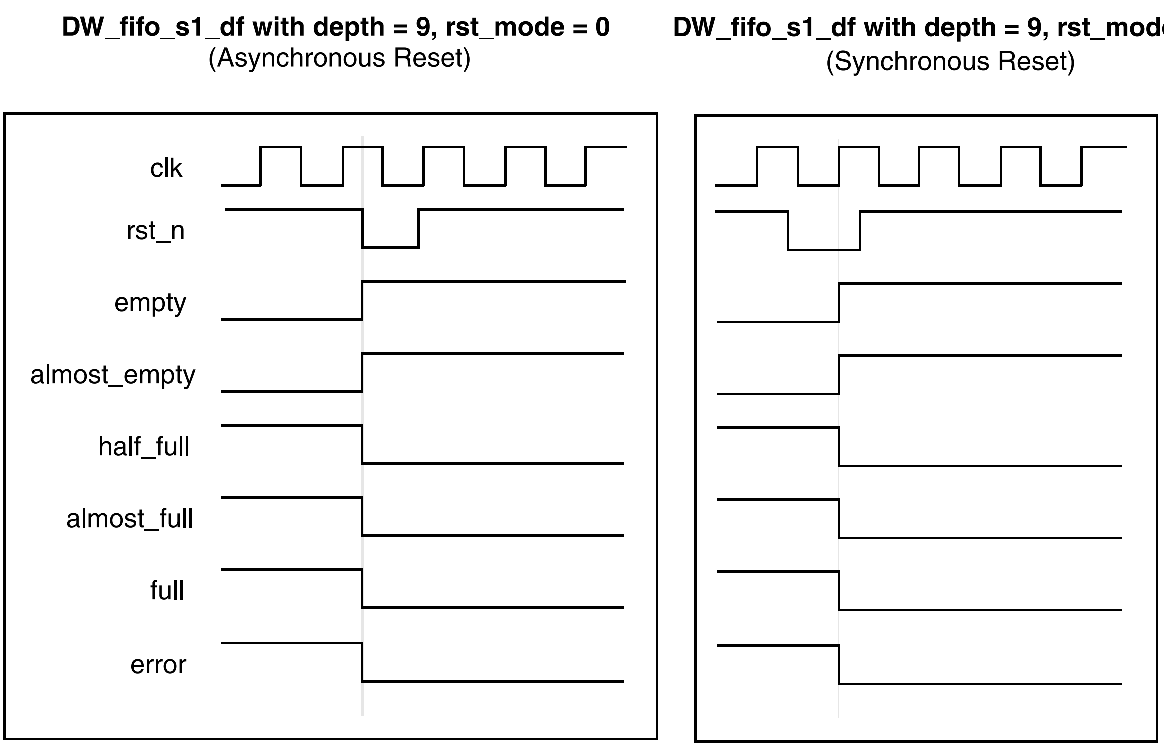


Figure 1-8 Reset Timing Waveforms



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_fifo_s1_df_inst is
  generic (inst_width    : INTEGER := 8;
           inst_depth    : INTEGER := 4;
           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);
        inst_data_in      : in std_logic_vector(inst_width-1 downto 0);
        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;
        data_out_inst : out std_logic_vector(inst_width-1 downto 0) );
end DW_fifo_s1_df_inst;

architecture inst of DW_fifo_s1_df_inst is
begin
```

```
    -- Instance of DW_fifo_s1_df
    U1 : DW_fifo_s1_df
      generic map (width => inst_width,   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,   data_in => inst_data_in,
                empty => empty_inst,   almost_empty => almost_empty_inst,
                half_full => half_full_inst,   almost_full => almost_full_inst,
                full => full_inst,   error => error_inst,
                data_out => data_out_inst );
  end inst;

  -- pragma translate_off
  configuration DW_fifo_s1_df_inst_cfg_inst of DW_fifo_s1_df_inst is
    for inst
    end for;
  end DW_fifo_s1_df_inst_cfg_inst;
  -- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_fifo_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, inst_data_in, empty_inst,
                          almost_empty_inst, half_full_inst,
                          almost_full_inst, full_inst, error_inst,
                          data_out_inst );
  parameter width = 8;
  parameter depth = 4;
  parameter err_mode = 0;
  parameter rst_mode = 0;
  `define bit_width_depth 2 // 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;
  input [width-1 : 0] inst_data_in;
  output empty_inst;
  output almost_empty_inst;
  output half_full_inst;
  output almost_full_inst;
  output full_inst;
  output error_inst;
  output [width-1 : 0] data_out_inst;

  // Instance of DW_fifo_s1_df
  DW_fifo_s1_df #(width, 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),
        .data_in(inst_data_in),   .empty(empty_inst),
        .almost_empty(almost_empty_inst),   .half_full(half_full_inst),
        .almost_full(almost_full_inst),   .full(full_inst),
        .error(error_inst),   .data_out(data_out_inst) );
endmodule
```

Copyright Notice and Proprietary Information

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

www.synopsys.com