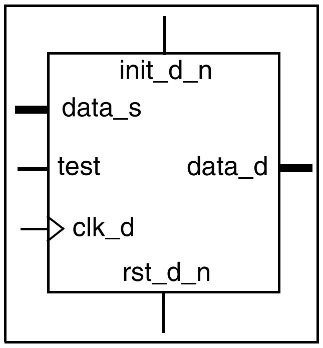
DW_sync

Single Clock Data Bus Synchronizer

Version, STAR and Download Information: IP Directory

Features and Benefits

- Parameterized data bus width
- Parameterized number of synchronizing stages
- Parameterized test feature
- Ability to model missampling of data on incoming clock domain



Description

DW_sync offers a solution for synchronizing data that crosses asynchronous clock boundaries. The DW_sync is versatile because it is configurable for the number of synchronizing stages (2, 3 or 4), the style of first-stage capturing flip-flop needed (negative or positive edge-triggered), and insertion of 'hold latches' to facilitate scan testing.

A unique built-in verification feature allows the designer to turn on a random sampling error mechanism that models skew between bits of the incoming data bus from the source domain (for more information, refer to the "Simulation Methodology" discussion). This facility provides an opportunity for determining system robustness during the early development phases and without having to develop special test stimulus. Figure 1-10 and Figure 1-11 are example timing diagrams depicting the behavior of the DW_sync when missampling is introduced.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
data_s	width	Input	Source Domain data vector
clk_d	1	Input	Destination Domain clock source
rst_d_n	1	Input	Destination Domain asynchronous reset (active low)
init_d_n	1	Input	Destination Domain synchronous reset (active low)
test	1	Input	Scan test mode select
data_d	width	Output	Destination Domain data vector

Table 1-2 Parameter Description

Parameter	Values	Description
width	1 to 1024 Default: 8	Vector width of input `data_s` and output `data_d`

Table 1-2 Parameter Description (Continued)

Parameter	Values	Description
f_sync_type	0 to 4 Default: 2	Forward Synchronization Type Defines type and number of synchronizing stages: 0 = single clock design, no synchronizing stages implemented 1 = 2-stage synchronization with 1st stage negative-edge capturing and 2nd stage positive-edge capturing 2 = 2-stage synchronization with both stages positive-edge capturing 3 = 3-stage synchronization with all stages positive-edge capturing 4 = 4-stage synchronization with all stages positive-edge capturing
tst_mode	0 to 2 Default: 0	Test Mode 0 = no 'latch' is inserted for scan testing 1 = insert negative-edge flip-flip on data_s input vector when `test` input is asserted 2 = for compatibility when latching is done outside of DW_sync (functions as with *tst_mode* = 0)
verif_en	0 to 4 Default: 1	Verification Enable Control 0 = no sampling errors inserted 1 = sampling errors are randomly inserted with 0 or up to 1 destination clock cycle delay 2 = sampling errors are randomly inserted with 0, 0.5, 1, or 1.5 destination clock cycle delays 3 = sampling errors are randomly inserted with 0, 1, 2, or 3 destination clock cycle delays 4 = sampling errors randomly inserted with 0 or up to 0.5 destination clock cycle delays * For more information about verif_en, refer to ""Simulation Methodology" on page 3.

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
rtl	Synthesis model	DesignWare

Table 1-4 Simulation Models

Model	Function
DW03.DW_SYNC_CFG_SIM	Design unit name for VHDL simulation
DW03.DW_SYNC_CFG_SIM_MS	Design unit name for VHDL simulation with mis-sampling enabled.
dw/dw03/src/DW_sync_sim.vhd	VHDL simulation model source code (modeling RTL) with no missampling
dw/dw03/src/DW_sync_sim_ms.vhd	VHDL simulation model source code with missampling
dw/sim_ver/DW_sync.v	Verilog simulation model source code

Simulation Methodology

For simulation, there are two methods available. One method is to use the simulation models since they emulate the RTL model. The other method is to enable modeling of random skew between bits on the `data_s` bus of the source domain by the destination domain (called "missampling" in this discussion). When using the simulation models purely to behave as the RTL model, no special configuration is required. When using the simulation models to enable missampling, you must consider the following.

- For Verilog simulation enabling missampling, a preprocessing variable named DW_MODEL_MISSAMPLES must be defined before compile:

  `define DW_MODEL_MISSAMPLES

  Once `DW_MODEL_MISSAMPLES is defined, the value of the *verif_en* parameter comes into play and configures the simulation model as described in Table 1-2. Note: If `DW_MODEL_MISSAMPLES is not defined, the Verilog simulation model behaves as if *verif_en* is set to '0'.

- For VHDL simulation enabling missampling, an alternative simulation architecture is provided. This architecture is named `sim_ms`. The parameter *verif_en* only has meaning when utilizing the `sim_ms`. That is, when binding the "sim" simulation architecture the *verif_en* value is ignored and the model effectively behaves as though *verif_en* is set to '0'. For an example on using each architecture, refer to "HDL Usage Through Component Instantiation - VHDL" on page 10.

Block Diagrams

Synchronization Type (*f_sync_type*) Diagrams

The following diagrams describe the behavior of the *f_sync_type* parameter.

Figure 1-1 Synchronization Type 1

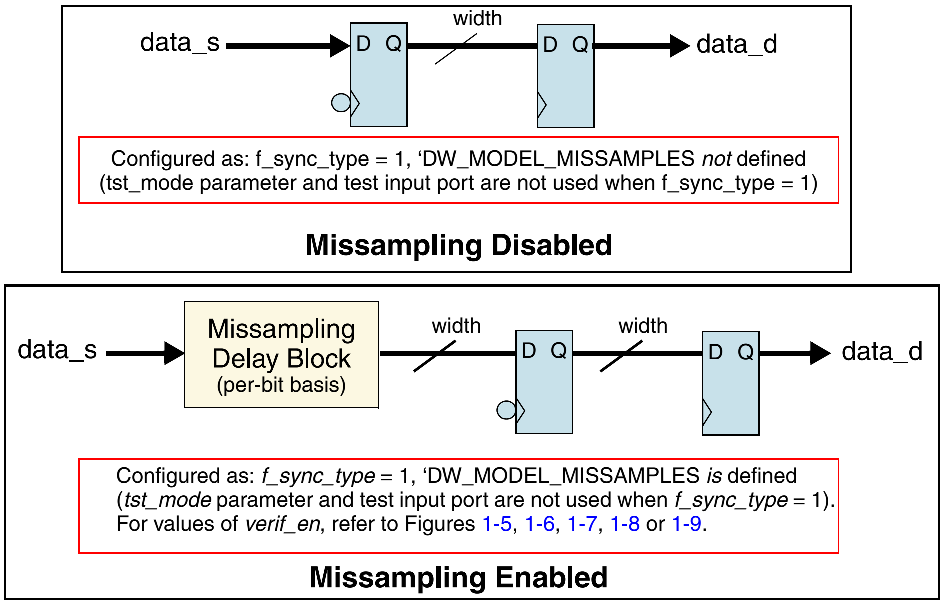
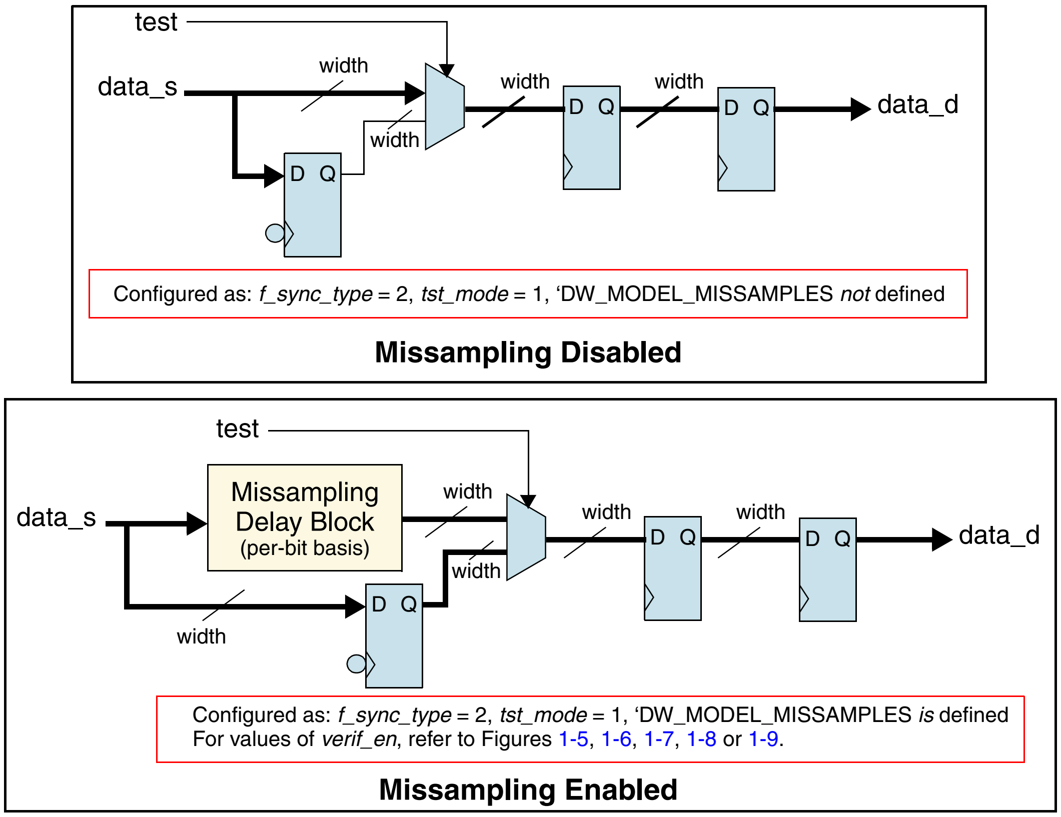

data_s
D Q
width
D Q
data_d
Configured as: f_sync_type = 1, 'DW_MODEL_MISSAMPLES not defined
(tst_mode parameter and test input port are not used when f_sync_type = 1)
Missampling Disabled
data_s
Missampling Delay Block
(per-bit basis)
width
D Q
width
D Q
data_d
Configured as: f_sync_type = 1, 'DW_MODEL_MISSAMPLES is defined
(tst_mode parameter and test input port are not used when f_sync_type = 1).
For values of verif_en, refer to Figures 1-5, 1-6, 1-7, 1-8 or 1-9.
Missampling Enabled

Figure 1-2 Synchronization Type 2



Configured as: *f_sync_type* = 2, *tst_mode* = 1, ‘DW_MODEL_MISSAMPLES *not* defined

Configured as: *f_sync_type* = 2, *tst_mode* = 1, ‘DW_MODEL_MISSAMPLES *is* defined
For values of *verif_en*, refer to Figures 1-5, 1-6, 1-7, 1-8 or 1-9.

Figure 1-3 Synchronization Type 3

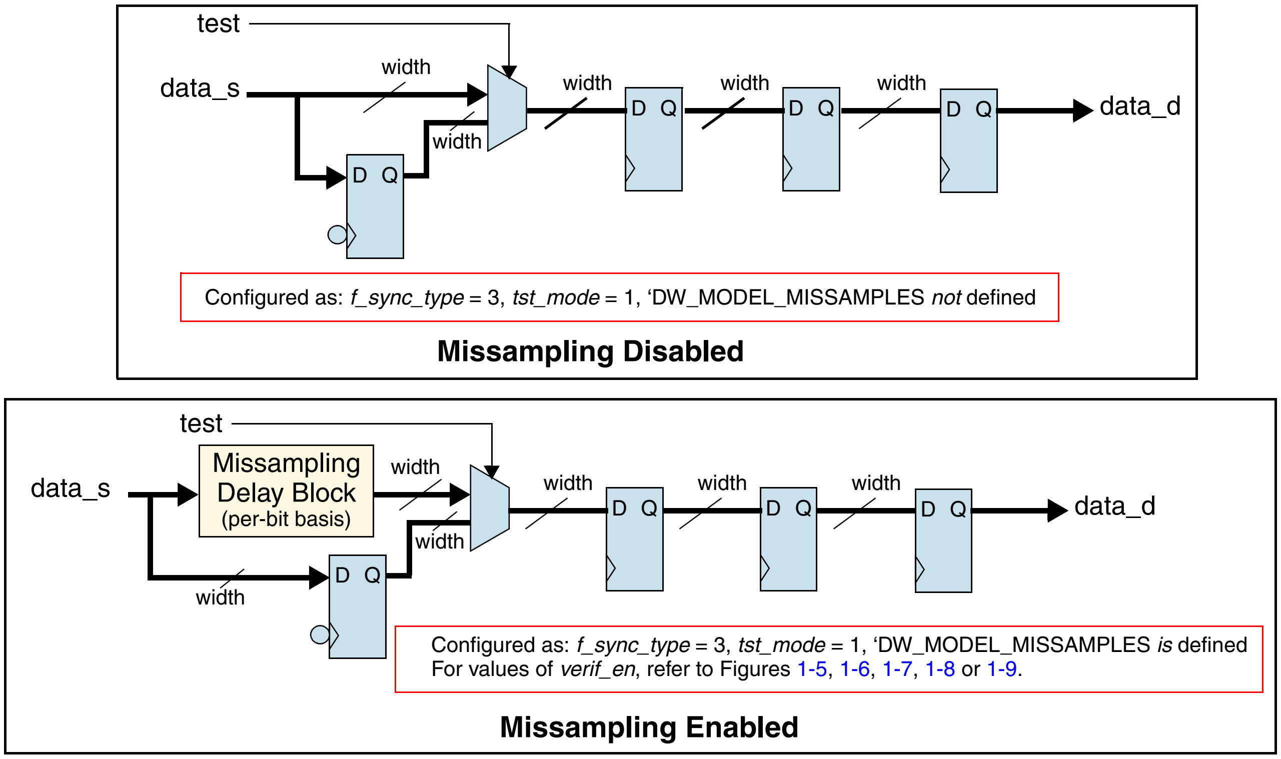
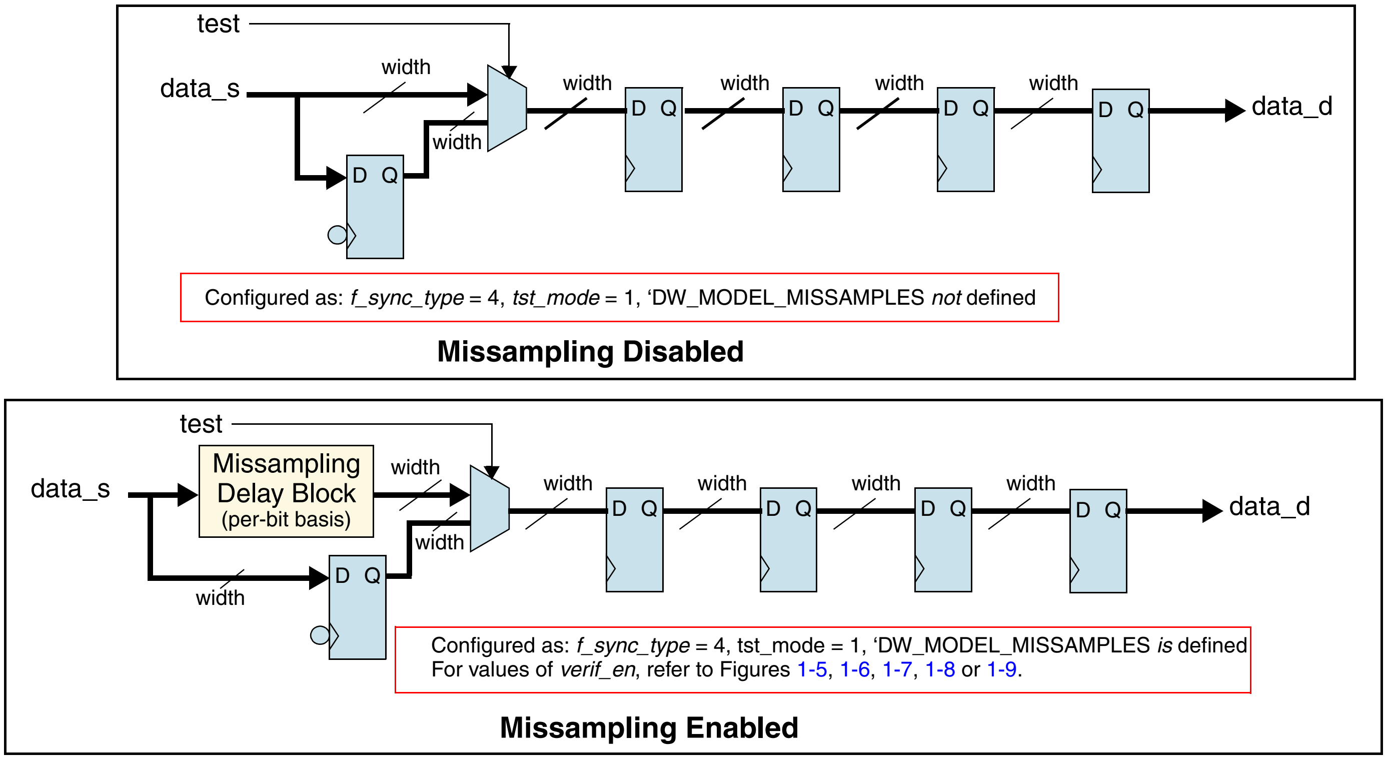

test
data_s
width
width
width
width
width
D Q
D Q
D Q
D Q
data_d
Configured as: *f_sync_type* = 3, *tst_mode* = 1, 'DW_MODEL_MISSAMPLES *not* defined
Missampling Disabled
test
data_s
Missampling Delay Block (per-bit basis)
width
width
width
width
width
width
D Q
D Q
D Q
D Q
data_d
Configured as: *f_sync_type* = 3, *tst_mode* = 1, 'DW_MODEL_MISSAMPLES *is* defined
For values of *verif_en*, refer to Figures 1-5, 1-6, 1-7, 1-8 or 1-9.
Missampling Enabled

Figure 1-4 Synchronization Type 4


test
data_s
width
width
width
width
width
width
D Q
D Q
D Q
D Q
D Q
data_d
Configured as: *f_sync_type* = 4, *tst_mode* = 1, 'DW_MODEL_MISSAMPLES *not* defined
Missampling Disabled
test
data_s
Missampling Delay Block (per-bit basis)
width
width
width
width
width
width
width
D Q
D Q
D Q
D Q
D Q
data_d
Configured as: *f_sync_type* = 4, tst_mode = 1, 'DW_MODEL_MISSAMPLES *is* defined
For values of *verif_en*, refer to Figures 1-5, 1-6, 1-7, 1-8 or 1-9.
Missampling Enabled

Verification Type (*verify_en*) Diagrams

The following diagrams describe the behavior of the *verify_en* parameter.

Figure 1-5 Missampling Model Type 0

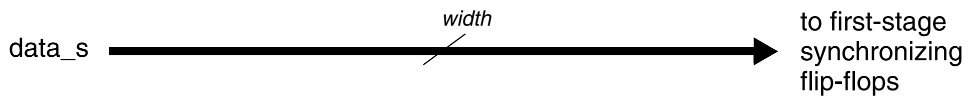


Figure 1-6 Missampling Model Type 1

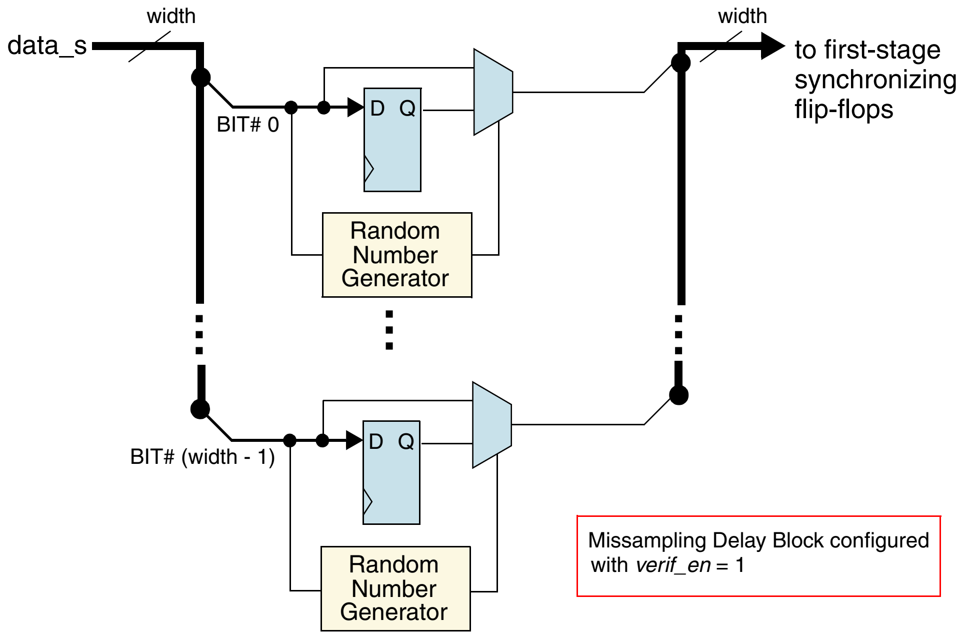

width
data_s
BIT# 0
D Q
Random Number Generator
BIT# (width - 1)
D Q
Random Number Generator
width
to first-stage synchronizing flip-flops
Missampling Delay Block configured with *verif_en* = 1

Figure 1-7 Missampling Model Type 2

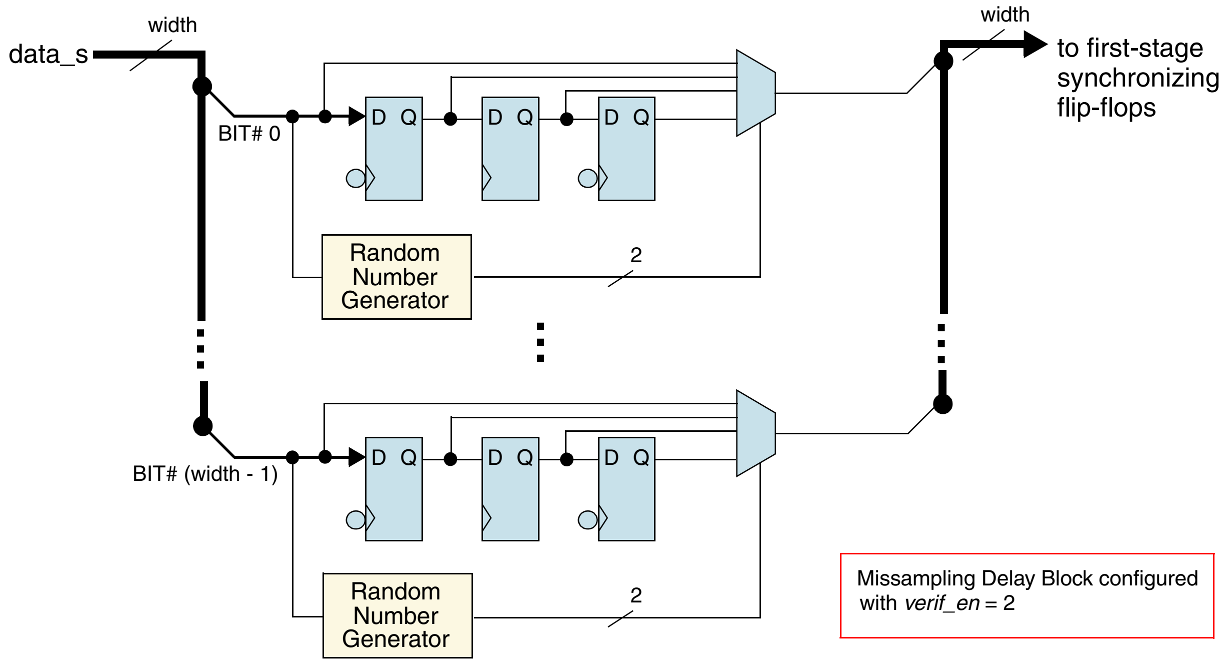


Figure 1-8 Missampling Model Type 3

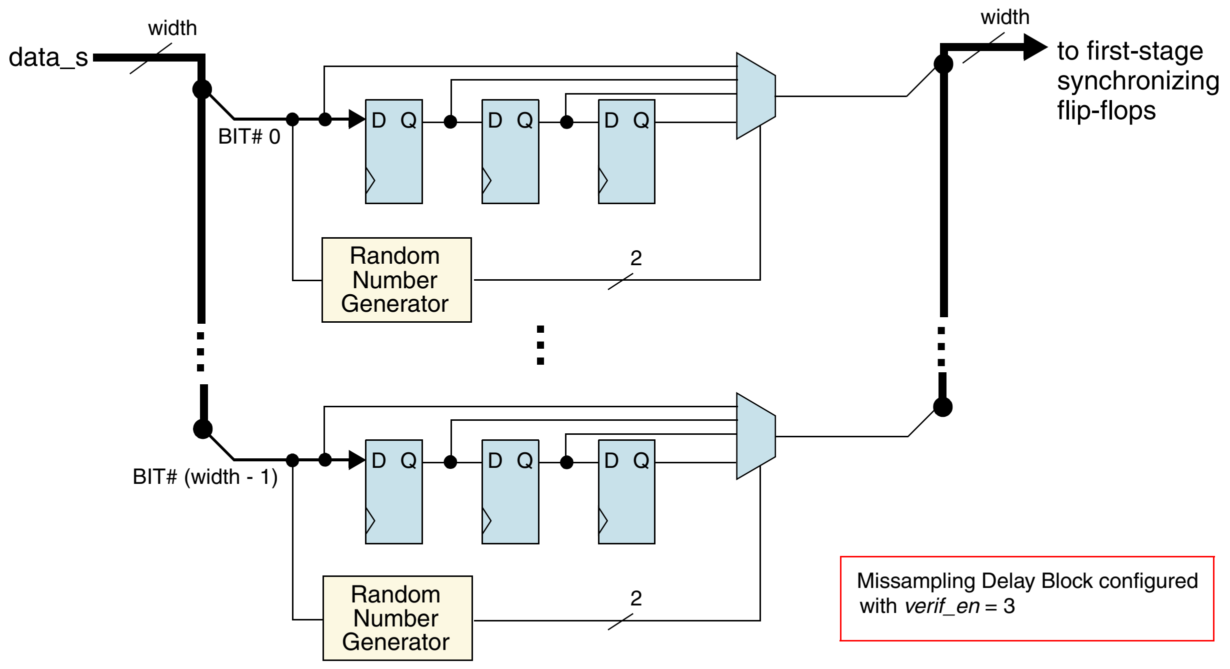
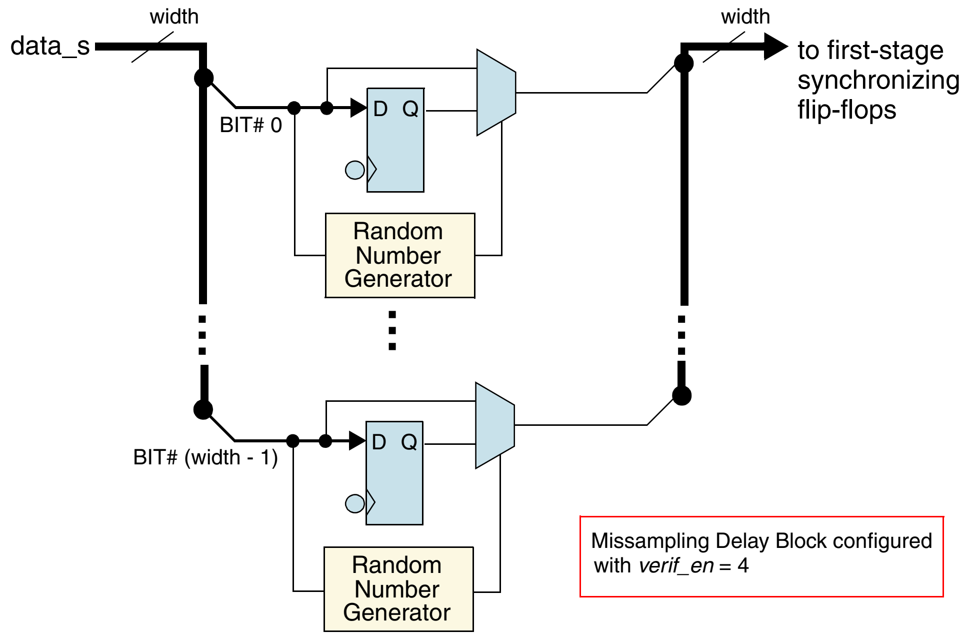


Figure 1-9 Missampling Model Type 4


data_s
width
BIT# 0
D Q
Random Number Generator
BIT# (width - 1)
D Q
Random Number Generator
width
to first-stage synchronizing flip-flops
Missampling Delay Block configured with *verif_en* = 4

Timing Diagrams

Figure 1-10 Enable Missampling of data_s input

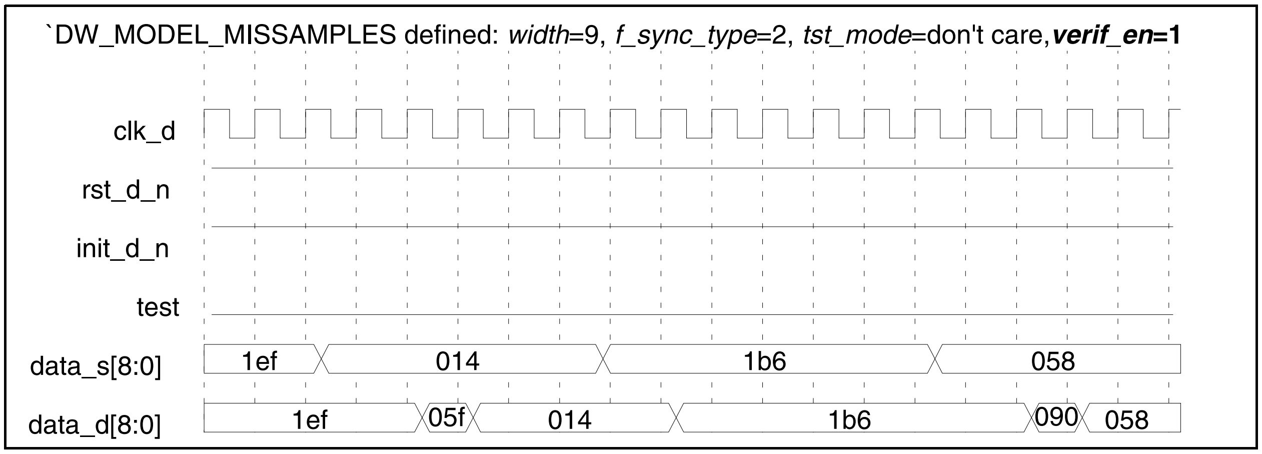
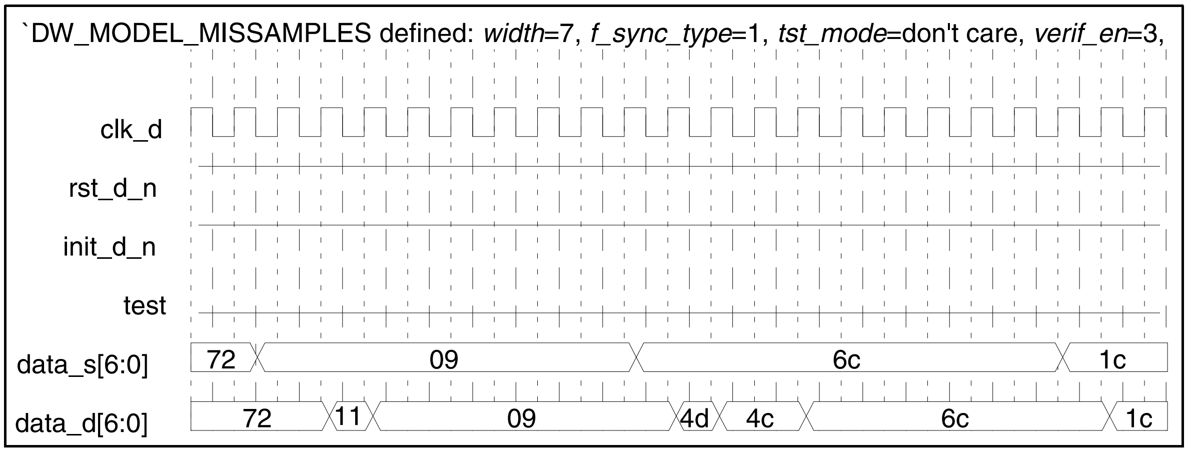


Figure 1-11 Enable Missampling of data_s input


`DW_MODEL_MISSAMPLES defined: *width*=7, *f_sync_type*=1, *tst_mode*=don't care, *verif_en*=3,
clk_d
rst_d_n
init_d_n
test
data_s[6:0]
72
09
6c
1c
data_d[6:0]
72
11
09
4d
4c
6c
1c

Related Topics

- Memory – Registers Overview
- DesignWare Building Block IP Documentation Overview

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DW_Foundation_comp.all;

entity DW_sync_inst is
      generic (
            inst_width : INTEGER := 8;
            inst_f_sync_type : INTEGER := 2;
            inst_tst_mode : INTEGER := 0;
            inst_verif_en : INTEGER := 1
            );
      port (
            inst_clk_d : in std_logic;
            inst_rst_d_n : in std_logic;
            inst_init_d_n : in std_logic;
            inst_data_s : in std_logic_vector (inst_width-1 downto 0);
            inst_test : in std_logic;
            data_d_inst : out std_logic_vector (inst_width-1 downto 0)
            );
    end DW_sync_inst;

architecture inst of DW_sync_inst is
begin

    -- Instance of DW_sync
    U1 : DW_sync
        generic map ( width => inst_width, f_sync_type => inst_f_sync_type,
    tst_mode => inst_tst_mode, verif_en => inst_verif_en )
    port map ( clk_d => inst_clk_d, rst_d_n => inst_rst_d_n,
          init_d_n => inst_init_d_n,
          data_s => inst_data_s, test => inst_test, data_d => data_d_inst );
end inst;

-- Configuration for use with a VHDL simulator
-- pragma translate_off
library DW03;
configuration DW_sync_inst_cfg_inst of DW_sync_inst is
  for inst
    -- NOTE: If desiring to model missampling, uncomment the following
    -- line.  Doing so, however, will cause inconsequential errors
    -- when analyzing or reading this configuration before synthesis.
    -- for U1 : DW_sync use configuration DW03.DW_sync_cfg_sim_ms; end for;
  end for; -- inst
end DW_sync_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_sync_inst( inst_clk_d, inst_rst_d_n, inst_init_d_n, inst_data_s, inst_test,
data_d_inst );

parameter width = 8;
parameter f_sync_type = 2;
parameter tst_mode = 0;
parameter verif_en = 1;


input inst_clk_d;
input inst_rst_d_n;
input inst_init_d_n;
input [width-1 : 0] inst_data_s;
input inst_test;
output [width-1 : 0] data_d_inst;

    // Instance of DW_sync
    DW_sync #(width, f_sync_type, tst_mode, verif_en)
          U1 ( .clk_d(inst_clk_d), .rst_d_n(inst_rst_d_n), .init_d_n(inst_init_d_n),
.data_s(inst_data_s), .test(inst_test), .data_d(data_d_inst) );

endmodule
```

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

www.synopsys.com