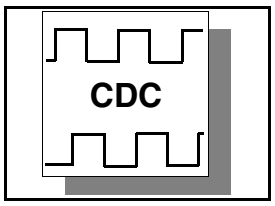
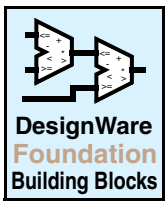




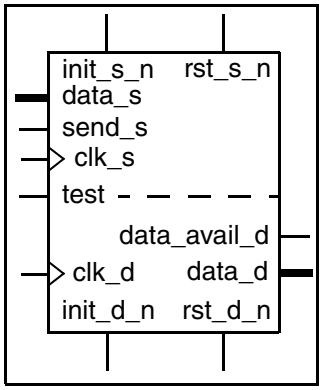
DW_data_sync_na

Data Bus Synchronizer without Acknowledge

Version, STAR and Download Information: IP Directory

Features and Benefits

- Interface between dual asynchronous clock domains
- Parameterized data bus width
- Parameterized number of synchronizing stages
- Parameterized test feature
- All output registered
- Ability to model missampling of data on incoming clock domain



Description

The DW_data_sync_na passes data values from the source domain to the destination domain. Full feedback handshake is not used, so there is no busy or done status in the source domain. This synchronizer is useful in situations where the destination domain is known to always have a significantly faster clock rate than the source domain—greater than three times. This means that every transaction started by the source domain is guaranteed to be completed in the destination domain before another one can be started in the source domain.

A unique built-in verification feature enables 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 "Simulation Methodology" on page 3). This facility provides an opportunity for determining system robustness during the early development phases without having to develop special test stimulus.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
clk_s	1	Input	Source Domain clock source
rst_s_n	1	Input	Source Domain asynchronous reset (active low)
init_s_n	1	Input	Source Domain synchronous reset (active low)
send_s	1	Input	Source initiate valid data vector control
data_s	*width*	Input	Source Domain data vector
clk_d	1	Input	Destination clock source
rst_d_n	1	Input	Destination asynchronous reset (active low)
init_d_n	1	Input	Destination synchronous reset (active low)

Table 1-1 Pin Description (Continued)

Pin Name	Width	Direction	Function
test	1	Input	Scan test mode select
data_avail_d	1	Output	Destination data update output
data_d	width	Output	Destination 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`
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 first stage negative-edge capturing and second 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 capturing register on `data_s` input vector when test input is asserted 2 = insert hold latch using active low latch
verif_en*	0 to 4 Default: 0	Verification enable control 0 = no sampling errors inserted 1 = sampling errors are randomly inserted with 0 or up to 1 destination clock cycle delays 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 are randomly inserted with 0 or up to 0.5 destination clock cycle delays * For more information about *verif_en*, see the Simulation Methodology section.
send_mode	0 to 3 Default: 1	0 = single clock cycle pulse in produces single clock cycle pulse out 1 = rising edge transition in produces single clock cycle pulse out 2 = falling edge transition in produces single clock cycle pulse out 3 = rising and falling transition each produce single clock cycle pulse out

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
rtl	Synthesis model	DesignWare

Table 1-4 Simulation Models

Model	Function
DW03.DW_DATA_SYNC_NA_CFG_SIM	Design unit name for VHDL simulation
dw/dw03/src/DW_data_sync_na_sim.vhd	VHDL simulation model source code (modeling RTL)—no mis-sampling
dw/sim_ver/DW_data_sync_na.v	Verilog simulation model source code

Simulation Methodology

For simulation, there are two methods available. One method is to use the simulation models as 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 "mis-sampling" in this discussion). When using the simulation models to behave only as the RTL model, no special configuration is required. When using the simulation models to enable mis-sampling, unique considerations must be made between Verilog and VHDL environments.

- For Verilog simulation enabling mis-sampling, a preprocessing variable named DW_MODEL_MISSAMPLES must be defined as follows:

  ```
  `define DW_MODEL_MISSAMPLES
  ```

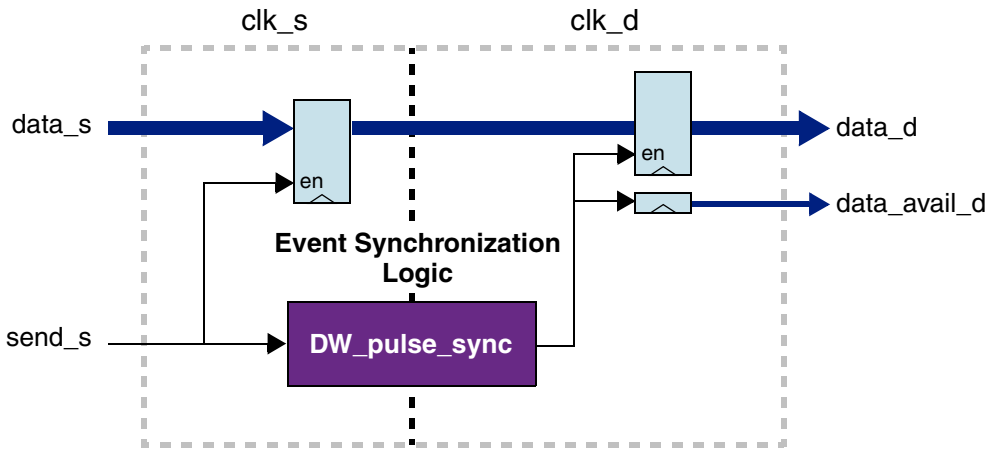
  Once `DW_MODEL_MISSAMPLES is defined, the value of the *verif_en* parameter, described in Table 2, configures the simulation model.

Note: If `DW_MODEL_MISSAMPLES is not defined, the Verilog simulation model behaves as if *verif_en* was set to 0.

- For VHDL simulation enabling mis-sampling, an alternative simulation architecture is provided. This architecture is named **sim_ms**. The *verif_en* parameter has meaning only 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 7.

Block Diagram

Figure 1-1 DW_data_sync_na Basic Block Diagram



Simulation Model Assertion for data_s Hold Violation

Both Verilog and VHDL simulation models contain an assertion to flag cases when the register that captures the `data_s` input vector changes while the `data_avail_d` is active. This implies that data presented at the inputs of the `data_d` output register are not held long enough for reliable capture, and missed `data_s` values could result. The models search during a window of time of one `clk_d` cycle just before capturing the `data_d` output register. The assertion message is in the form of a warning and looks like the following when running in a Verilog environment:

```
66520 TopModule.U1.SIM: WARNING: #### 'data_d' output register setup-time violation.
Captured data_s changes within one clk_d cycle before rising-edge. ####
```

An example of the assertion message for the VHDL simulation model is:

```
Assertion WARNING at 66520000 PS in design unit DW_DATA_SYNC_NA(SIM) from process
/MHDL_VH_TOP/\/DW_data_sync_na_top
/U1/DW_data_sync_na_vhdl\/SETUP_TIME_VIOLATION_MSG:

    "#### 'data_d' output register setup-time violation.  Captured data_s changes
within one clk_d cycle before rising-edge. ####"
```

The warning message is presented in terms of a setup-time violation with respect to the destination domain, but could also be represented as a hold-time violation from the source domain perspective. The waveform in Figure 1-3 on page 6 illustrates this.

Reset Considerations

The assertion of the `send_s` input is detected by the instantiated DW_pulse_sync, which internally converts it to a toggle event from the source domain. This toggle does not rely on state value, but rather on state change from the source domain. As a result, an assertion of `rst_s_n` or `init_d_n` could cause an erroneous toggle to occur that translates into the destination domain as a false assertion of `data_avail_d`. If this

behavior cannot be tolerated, it is recommended that both source and destination domains be reset simultaneously.

Timing Diagrams

Figure 1-2 illustrates how the destination domain clock, `clk_d`, is three times faster than the source domain clock, `clk_s`. With the `f_sync_type` equal to 2, a new value of `data_s` can occur every `clk_s cycle`, that is, `send_s` continuously active, without the destination domain dropping any data.

Figure 1-2 Fundamental Case where clk_d is > 3 Times Faster than clk_s

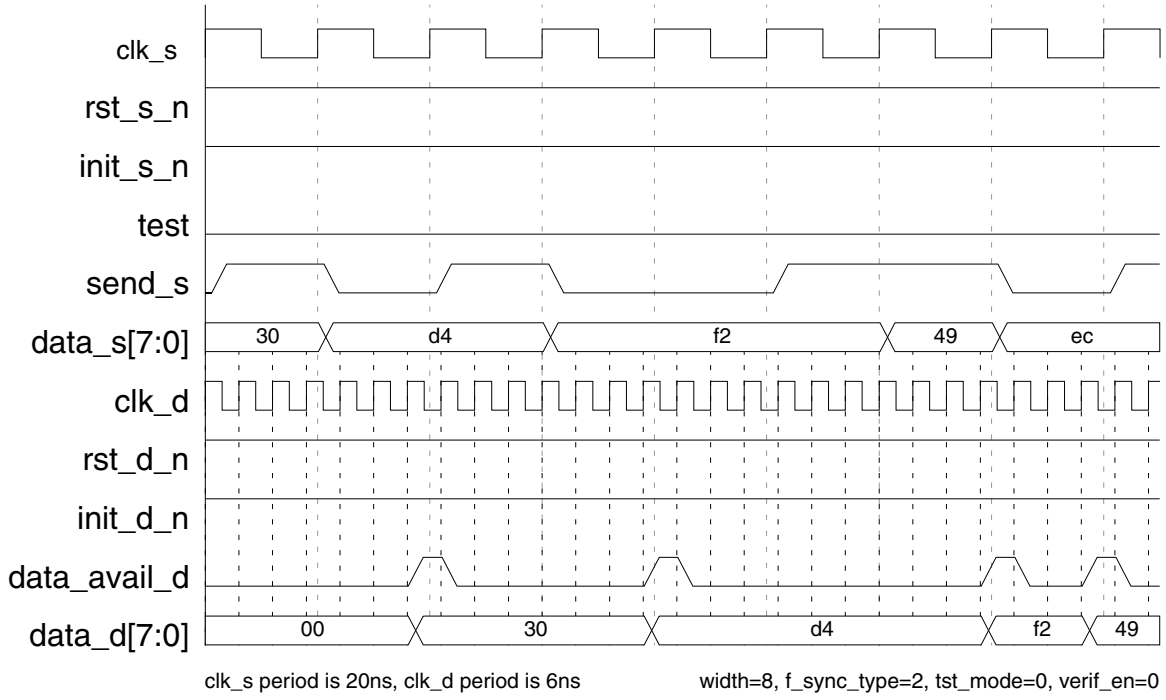
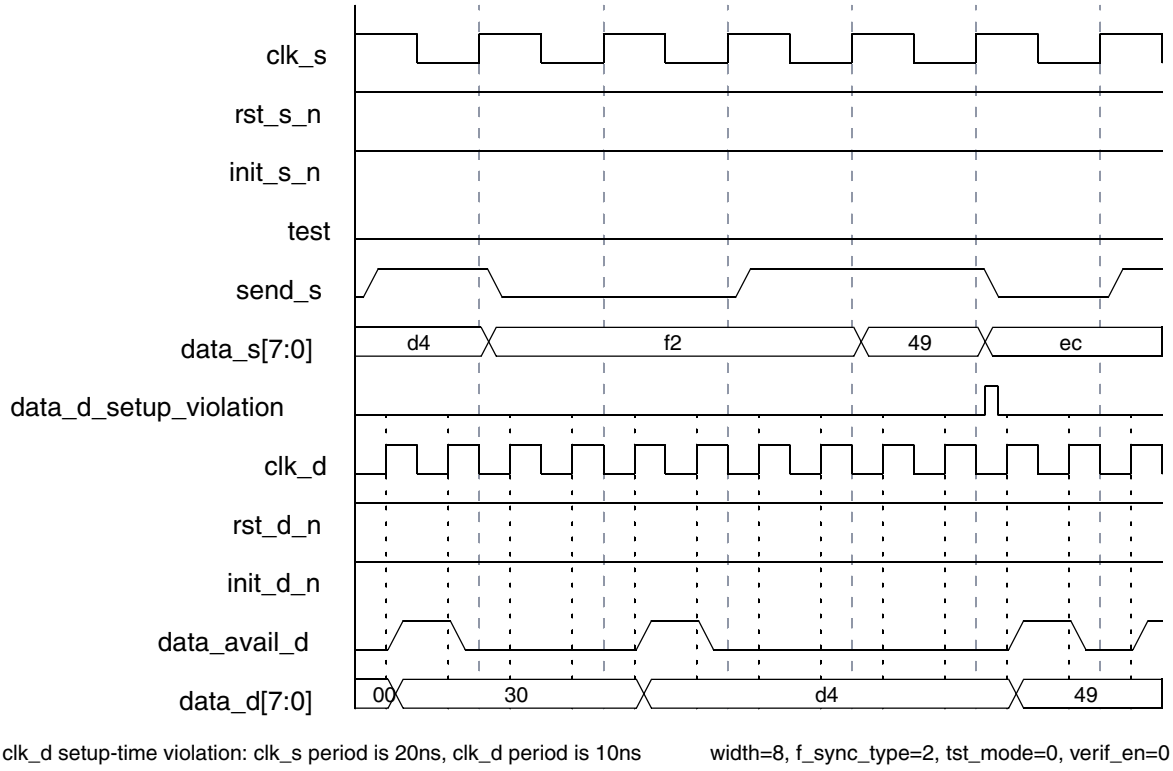


Figure 1-3 shows an example where `clk_d` is only two times faster than `clk_s`, with *f_sync_type* equal to 2. Because the `clk_d` rate is less than three times as fast as `clk_s`, there is a potential for `data_s` values from the source domain being dropped at the destination domain if there are no gaps in the assertion of `send_s`. Here, `data_s` values of 0xf2 and 0x42 are sent back-to-back—`send_s` active two consecutive cycles. During the internal evaluation of the `data_s` bus, the 'data_d_setup_violation' flag goes active when it sees captured `data_s` change a `clk_d` cycle before `data_avail_d` goes active. At the time of the setup-time violation, data internal to DW_data_sync_na as it is presented to the destination domain registers has the value 0x49 ready for launching. Unfortunately, the 0xf2 value has been overwritten during this violation. Hence, the sequence of `data_d` values is 0xd4 and 0x49, rather than the sequence presented originally at the `data_s` input of 0xd4, 0xf2, and 0x49.

Figure 1-3 Setup Violation at clk_d (clk_d rate < 3 times faster than clk_s rate)



Note that the ‘data_d_setup_violation’ signal is an internal signal available in the simulation models only. The Verilog implementation model does not contain this signal and as a result does not provide a warning message should this event occur. Only the two simulation models contain this functionality.

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_data_sync_na_inst is
      generic (
        inst_width : INTEGER := 8;
        inst_f_sync_type : INTEGER := 2;
        inst_tst_mode : INTEGER := 0;
        inst_verif_en : INTEGER := 2
        );
      port (
        inst_clk_s : in std_logic;
        inst_rst_s_n : in std_logic;
        inst_init_s_n : in std_logic;
        inst_send_s : in std_logic;
        inst_data_s : in std_logic_vector(inst_width-1 downto 0);
        inst_clk_d : in std_logic;
        inst_rst_d_n : in std_logic;
        inst_init_d_n : in std_logic;
        inst_test : in std_logic;
        data_avail_d_inst : out std_logic;
        data_d_inst : out std_logic_vector(inst_width-1 downto 0)
        );
    end DW_data_sync_na_inst;


architecture inst of DW_data_sync_na_inst is
begin

    -- Instance of DW_data_sync_na
    U1 : DW_data_sync_na
    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_s => inst_clk_s, rst_s_n => inst_rst_s_n, init_s_n =>
                inst_init_s_n,
    send_s => inst_send_s, data_s => inst_data_s, clk_d => inst_clk_d,
    rst_d_n => inst_rst_d_n, init_d_n => inst_init_d_n, test => inst_test,
    data_avail_d => data_avail_d_inst, data_d => data_d_inst );


end inst;

-- Configuration for use with a VHDL simulator
-- pragma translate_off
library DW03;
configuration DW_data_sync_na_inst_cfg_inst of DW_data_sync_na_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_data_sync_na use configuration DW03.DW_data_sync_na_cfg_sim_ms;  end
for;
  end for; -- inst
end DW_data_sync_na_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_data_sync_na_inst( inst_clk_s, inst_rst_s_n, inst_init_s_n, inst_send_s,
inst_data_s,
          inst_clk_d, inst_rst_d_n, inst_init_d_n, inst_test, data_avail_d_inst,
          data_d_inst );

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


input inst_clk_s;
input inst_rst_s_n;
input inst_init_s_n;
input inst_send_s;
input [width-1 : 0] inst_data_s;
input inst_clk_d;
input inst_rst_d_n;
input inst_init_d_n;
input inst_test;
output data_avail_d_inst;
output [width-1 : 0] data_d_inst;

    // Instance of DW_data_sync_na
    DW_data_sync_na #(width, f_sync_type, tst_mode, verif_en)
      U1 ( .clk_s(inst_clk_s), .rst_s_n(inst_rst_s_n),
.init_s_n(inst_init_s_n), .send_s(inst_send_s), .data_s(inst_data_s),
.clk_d(inst_clk_d), .rst_d_n(inst_rst_d_n), .init_d_n(inst_init_d_n),
.test(inst_test), .data_avail_d(data_avail_d_inst), .data_d(data_d_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