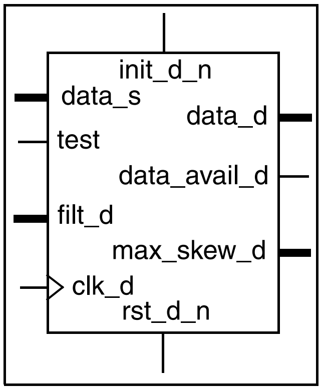
DW_data_sync_1c

Single Clock Filtered Data Bus Synchronizer

Version, STAR and Download Information: IP Directory

Features and Benefits

- Synchronizes on data without incoming clock source
- 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_1c can synchronize a data bus to a clock domain when only the data bus is available but not the source clock. The bus is double register synchronized (which is configurable up to thee stages of synchronization) and then monitored for changes. Once a change is detected, the module looks for a specified number of clocks of stability in the data bus before passing the synchronized data bus value out. The number of cycles required to declare the bus stable is specified by the input port `filt_d` and has a range that is determined by the parameter *filt_size*.

This synchronizer is especially well suited for synchronizing a bus from off-chip, where the input skew between bits of the bus may have a large variance due to board-level wiring and the skew also changes from board design to board design. With the possibility of high bit-to-bit skew and a relatively high frequency `clk_d`, this module is the prime candidate for the using a *verif_en* value of 2. Board-level routing skews can create bit-to-bit skews greater than one cycle of a fast `clk_d` signal which leads to the need for modeling missampling across more that a single clock delay.

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 “Simulation Methodology” on page 3). This facility provides an opportunity for determining system robustness during the early development phases and without having to develop special test stimulus.

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)

Table 1-1 Pin Description (Continued)

Pin Name	Width	Direction	Function
filt_d	filt_size	Input	Destination domain filter time specification
test	1	Input	Scan test mode select
data_avail_d	1	Output	Destination domain data update output
data_d	*width*	Output	Destination domain data vector
max_skew_d	*filt_size* +1	Output	Destination domain maximum skew detected between bits for any data_s bus transition

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 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
filt_size	1 to 8 Default: 1	filt_d vector size The width in bits for the filt_d vector input
tst_mode	0 or 1 Default: 0	Test Mode 0 = no 'latch' is inserted for scan testing 1 = insert negative-edge capturing flip-flip on data_s input vector when the test input is asserted.
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 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*, see the Simulation Methodology section.

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_1C_CFG_SIM	Design unit name for VHDL simulation (no missampling)
DW03.DW_DATA_SYNC_1C_CFG_SIM_MS	Design unit name for VHDL simulation (missampling enabled)
dw/dw03/src/DW_data_sync_1c_sim.vhd	VHDL simulation model source code (modeling RTL)
dw/sim_ver/DW_data_sync_1c.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:

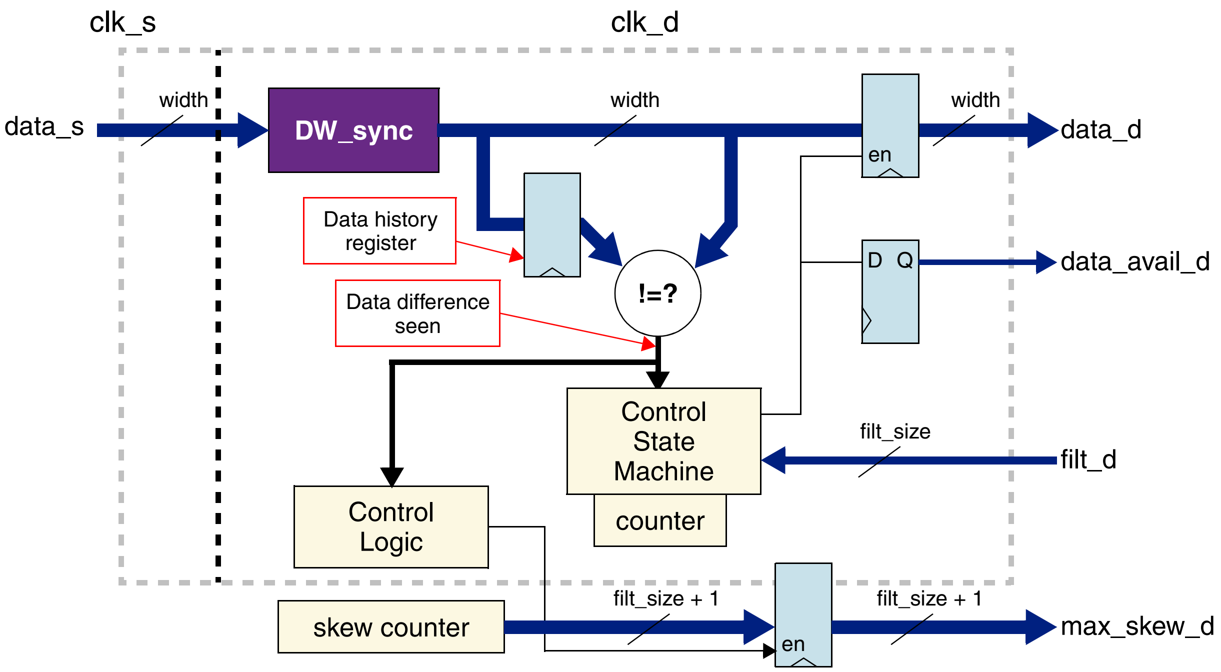
  ```
  `define DW_MODEL_MISSAMPLES
  ```

  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 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 8.

Block Diagram

Figure 1-1 DW_data_sync_1c Basic Block Diagram



filt_d Interpretation

The filt_d value setting specifies the number of clk_d cycles in which the data_d output is stable after the first occurrence of that data_d value. Once stable data is observed based on the filt_d value, the data_d and data_avail_d outputs are registered. For example, if you want to see two consecutive clk_d cycles of stable output data from DW_sync, set filt_d to 1.

data_s Bus Transition Guidelines as Related to filt_d

To prevent the dropping of data_s values, choosing a filt_d value as it relates to the frequency of data_s transitions and the potential bit skew within data_s in terms of clk_d is important. For the simple case where no skew between bits is expected on data_s in which the bus value is captured in its entirety by clk_d, data_s cannot change more often than every 'filt_d+1' clk_d cycles and filt_d would typically be set to '1' for optimum throughput.

In the case where there is less than a clk_d cycle skew between bits of data_s but the skew is greater than 0, then data_s cannot change more often than every 'filt_d+2' clk_d cycles. Again, filt_d can be set as low as '1' for optimum performance without dropping data.

However, in the more severe bit skew cases on data_s of greater than one clk_d cycle, the data_s bus value changes should not be made more often than the number of clk_d cycles based on the formula:

max_skew_d + 2

where max_skew_d is the maximum number of clk_d cycles of data_s bit skew (see immediately below for the detailed description of max_skew_d) and filt_d is recommended to be set to 'max_skew_d + 1'.

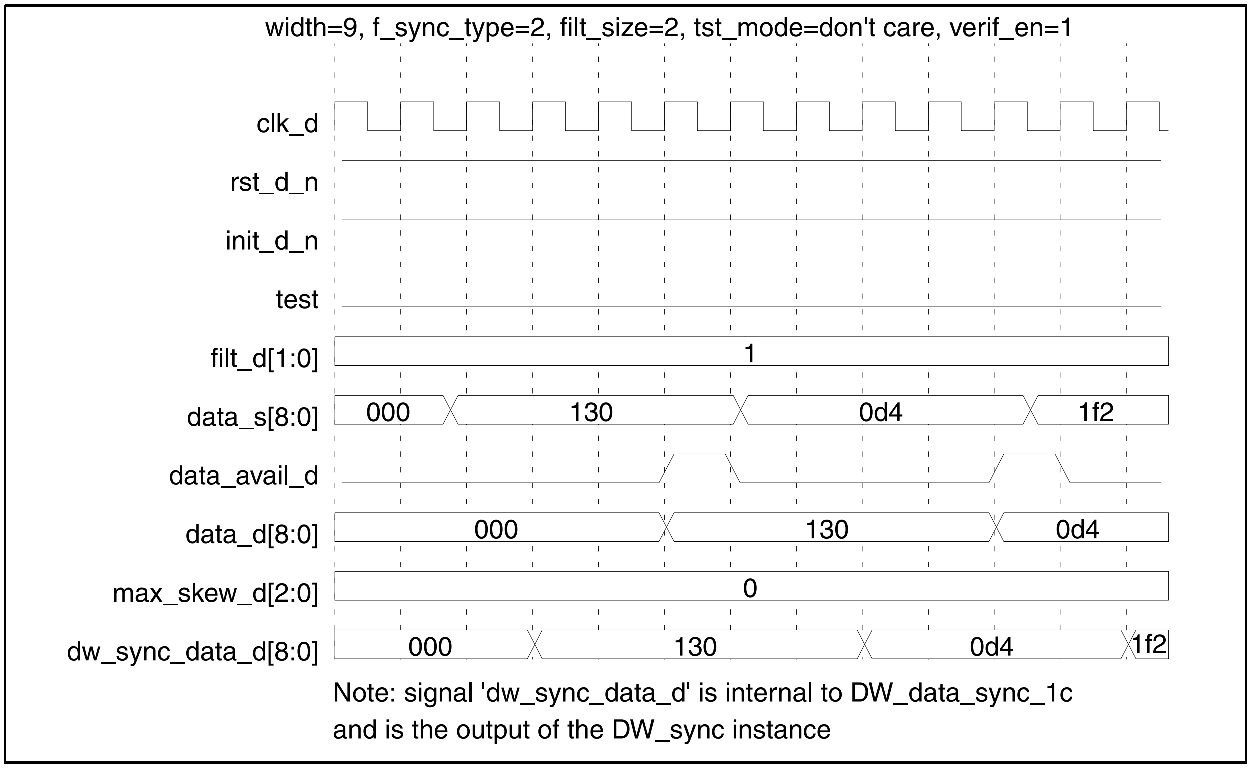
max_skew_d Interpretation

max_skew_d is an ongoing result which defines the maximum skew (in clk_d cycles) between bits within data_s per data_s bus transition that has been collected over time. The idea is to use max_skew_d to determine an optimized filt_d value. One approach to determining max_skew_d is run the system through an initialization/test period which is representative of normal behavior for data_s. Then, filt_d can be set to the recommended value of max_skew_d + 1 for max_skew_d values '2' or higher. When max_skew_d is 1, filt_d can be set to '1' for optimal throughput as long as 'data_s bus transition guidelines' are maintained as described above. It is not recommended to set filt_d to less than max_skew_d. Note: if filt_d is set to 0 that implies there is no skew between bits within data_s and, therefore, no skew measurements will be taken. The resulting max_skew_d will be 0 in this case.

Timing Diagrams

Figure 1-2 depicts the case in which data_s contains on intra-bit skew meaning all bits arrive into the component at exactly the same time. The important concept to note here is that since filt_d is 1, the number of clk_d cycles of stable data on the internal signal dw_sync_data_d[8:0] must 2 before data is considered valid. 2 clk_d cycles defined 1 cycle when first observed at the DW_sync output and 1 cycle (as defined by filt_d = 1), which has the same value immediately after the first observed occurrence. Note that in the diagram, signal dw_sync_data_d[8:0] is provided as reference only and it represents the data_d of the instance of DW_sync contained in DW_data_sync_1c.

Figure 1-2 Fundamental case where no intra-bit skew on data_s

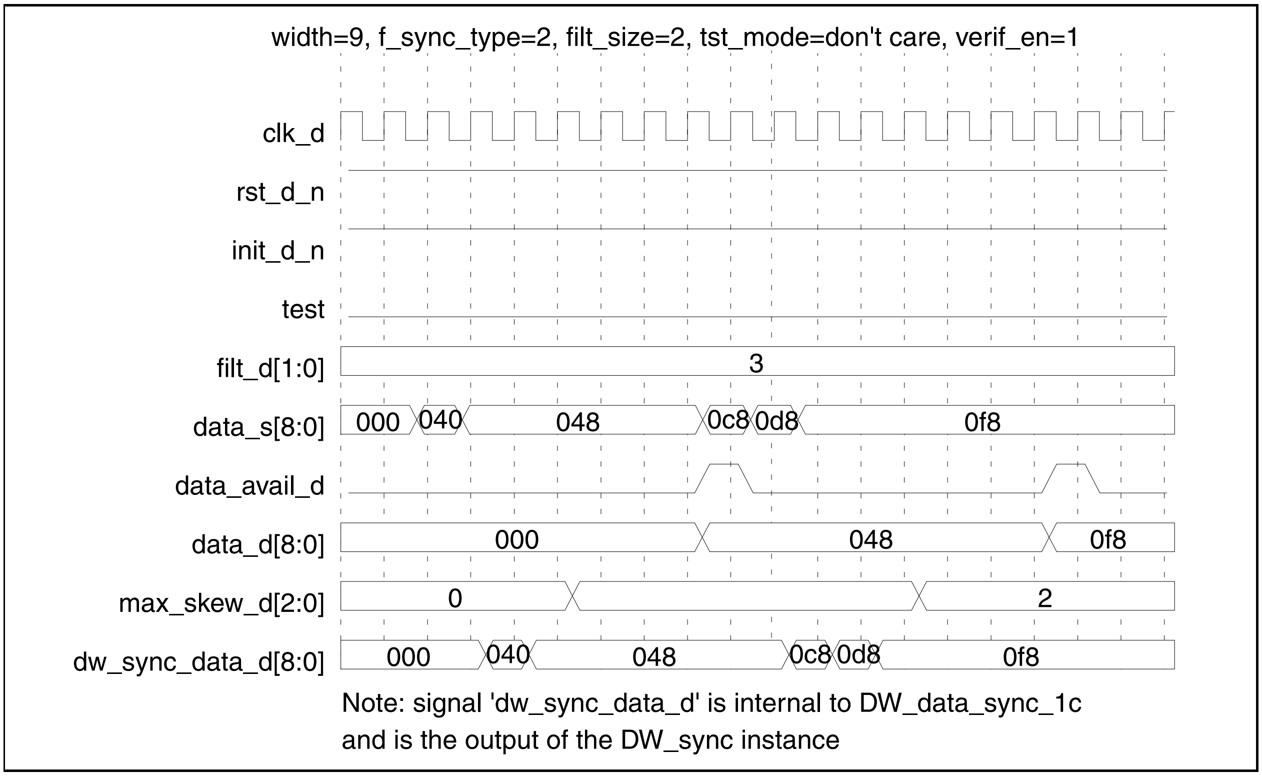


Note: signal 'dw_sync_data_d' is internal to DW_data_sync_1c
and is the output of the DW_sync instance

Figure 1-3 shows an example of how intra-bit skew on data_s as seen by the destination domain (`clk_d`) contributes to the result of the output max_skew_d. Note for illustration purposes, the intra-bit skew on `data_s` is grossly exaggerated to clearly cross `clk_d` sampling boundaries. At beginning of time, bits on `data_s` arrive as all 0s then change to 0x040 then to 0x048 effectively one `clk_d` sampling later. The value of 0x048 settles long enough (satisfying `filt_d`) for `data_avail_d` to activate and the captured `data_s` value of 0x048 to be presented to the destination domain at that time. Meanwhile, due to the `data_s` intermediate changes before settling (once after the previously settled value of all 0s) a new `max_skew_d` value is calculated to be 1.

In the subsequent `data_s` value changes after 0x048 and before settling on 0x0f8 the `max_skew_d` value gets updated to 2 because data_s underwent intra-bit skew across two `clk_d` cycles as it arrived into the component as 0x0c8, then 0x0d8, then finally settling at 0x0f8.

Figure 1-3 Intra-bit skew on data_s affecting max_skew_d results


width=9, f_sync_type=2, filt_size=2, tst_mode=don't care, verif_en=1
clk_d
rst_d_n
init_d_n
test
filt_d[1:0]
3
data_s[8:0]
000
040
048
0c8
0d8
0f8
data_avail_d
data_d[8:0]
000
048
0f8
max_skew_d[2:0]
0
2
dw_sync_data_d[8:0]
000
040
048
0c8
0d8
0f8
Note: signal 'dw_sync_data_d' is internal to DW_data_sync_1c
and is the output of the DW_sync instance

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_1c_inst is
      generic (
            inst_width : INTEGER := 8;
            inst_f_sync_type : INTEGER := 2;
            inst_filt_size : INTEGER := 1;
            inst_tst_mode : INTEGER := 0;
            inst_verif_en : INTEGER := 2
            );
      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_filt_d : in std_logic_vector(inst_filt_size-1 downto 0);
            inst_test : in std_logic;
            data_avail_d_inst : out std_logic;
            data_d_inst : out std_logic_vector(inst_width-1 downto 0);
            max_skew_d_inst : out std_logic_vector(inst_filt_size downto 0)
            );
    end DW_data_sync_1c_inst;


architecture inst of DW_data_sync_1c_inst is
begin

    -- Instance of DW_data_sync_1c
    U1 : DW_data_sync_1c
    generic map ( width => inst_width, f_sync_type => inst_f_sync_type,
                filt_size => inst_filt_size, 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,
                filt_d => inst_filt_d,test => inst_test,
                data_avail_d => data_avail_d_inst, data_d => data_d_inst,
                max_skew_d => max_skew_d_inst );
end inst;

-- Configuration for use with a VHDL simulator
-- pragma translate_off
library DW03;
configuration DW_data_sync_1c_inst_cfg_inst of DW_data_sync_1c_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_1c use configuration DW03.DW_data_sync_1c_cfg_sim_ms;  end
 for;
   end for; -- inst
 end DW_data_sync_1c_inst_cfg_inst;
 -- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_data_sync_1c_inst( inst_clk_d, inst_rst_d_n, inst_init_d_n, inst_data_s,
inst_filt_d,
                                                      inst_test,  data_avail_d_inst,
data_d_inst, max_skew_d_inst );

parameter width = 8;
parameter f_sync_type = 2;
parameter filt_size = 1;
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 [filt_size-1 : 0] inst_filt_d;
input inst_test;
output                      data_avail_d_inst;
output [width-1 : 0] data_d_inst;
output [filt_size : 0] max_skew_d_inst;

    // Instance of DW_data_sync_1c
    DW_data_sync_1c #(width, f_sync_type, filt_size, 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), .filt_d(inst_filt_d), .test(inst_test),
                  .data_avail_d(data_avail_d_inst), .data_d(data_d_inst),
                  .max_skew_d(max_skew_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