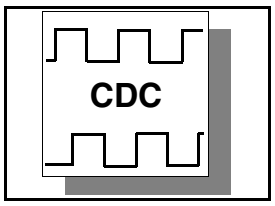
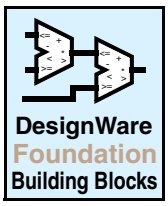




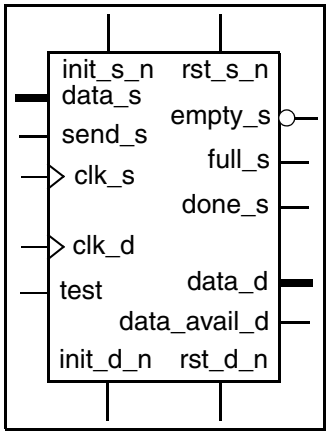
DW_data_sync

Data Bus Synchronizer with Acknowledge

Version, STAR and Download Information: IP Directory

Features and Benefits

- Fully Tested clock domain crossing
- Fully parametrized
- Selectable clock edge
- Parameterized output registration: either combinatorial or registered outputs
- Applications: data bus controllers, bus-based communication circuits, any interface sending parallel data between two clock domains



Description

The DW_data_sync passes data values from the source domain to the destination domain through a hand-shake protocol which includes an acknowledge back to the source domain. You can use an optional pending data buffer register to save pending data presented to the source interface while an earlier data transfer transaction is in progress.

The first transferred data loads into the transmit register. While the transmit register is busy transferring data (before receiving the acknowledge for that data), the optional pending data register (selected by parameter *pend_mode* = 1) holds data until the current data transfer is complete, freeing the transit register to receive the pending data for a subsequent transmission. Multiple writes to the pending register will overwrite earlier data, resulting in the last data written (until the current transfer is complete) being the data that is transferred.

A one-clock-cycle pulse (in the source clock domain) on the `done_s` output indicates the completion of one data word transfer.

The active low `empty_s` output goes inactive (high) when the transmit register is currently in use transferring a word of data.

When the *pend_mode* parameter is set to '0', then the active high `full_s` output goes active (high) when the transmit register is in use transferring a word of data. Thus, when *pend_mode* = 0, the active high `full_s` output has the same value as the active low `empty_s` output.

When the *pend_mode* parameter is set to '1', then the active high `full_s` output goes active (high) when a data word is written to the pending data register and goes inactive (low) when the current data transfer transaction is complete and no new send request is present.

Spurious pulses caused by reset

The source domain transmit register semaphore is based on a toggle type transfer; therefore, a reset in the source domain after an odd number of bus transactions results in a spurious 'available' flag bit in the destination domain. If these are unacceptable, you must reset both domains simultaneously.

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
empty_s	1	Output	Source domain transaction reg empty output (active low)
full_s	1	Output	Source domain transaction reg full output
done_s	1	Output	Source domain transaction done output
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)
data_avail_d	1	Output	Destination data update output
data_d	width	Output	Destination data vector
test	1	Input	Scan test mode select

Table 1-2 Parameter Description

Parameter	Values	Description
width	1 to 1024 Default: 8	Vector width of input `data_s` and output `data_d`
pend_mode	0 to 1 Default: 1	Buffer pending data
ack_delay	0 to 1 Default: 0	The acknowledge signal returned from the destination domain occurs either (0) before the second edge register, or (1) after the second edge detect register.
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

Table 1-2 Parameter Description (Continued)

Parameter	Values	Description
r_sync_type	0 to 4 Default: 2	0 = Single clock design (that is, `clk_s` == `clk_d`) 1 = First synchronization in `clk_s` domain is done on the negative edge and the rest on positive edge. This reduces latency req. of synchronization slightly but quicker metastability resolution for the negative edge sensitive FF. It also requires the technology library to contain an acceptable negative edge sensitive FF. 2 = All synchronization in `clk_s` domain is done on positive edges - 2 D flip flops in source domain. 3 = All synchronization in `clk_s` domain is done on positive edges - 3 D flip flops in source domain. 4 = All synchronization in `clk_s` domain is done on positive edges - 4 D flip flops in source domain.
tst_mode	0 or 1 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
verif_en*	0 to 4 Default: 0	Verification enable 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 in the DW_sync datasheet.
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_CFG_SIM	Design unit name for VHDL simulation
dw/dw03/src/DW_data_sync_sim.vhd	VHDL simulation model source code (modeling RTL)—no missampling
dw/sim_ver/DW_data_sync.v	Verilog simulation model source code

Figure 1-1 illustrates *send_mode* = 0, every detected 'high' logic level in produces a single clock cycle pulse out, timing from `send_s` to `done_s`, with `data_avail_d` (*r_sync_type* = 2, *f_sync_type* = 2).

Figure 1-1 Timing for pend_mode = 1, ack_delay = 1, send_mode = 0

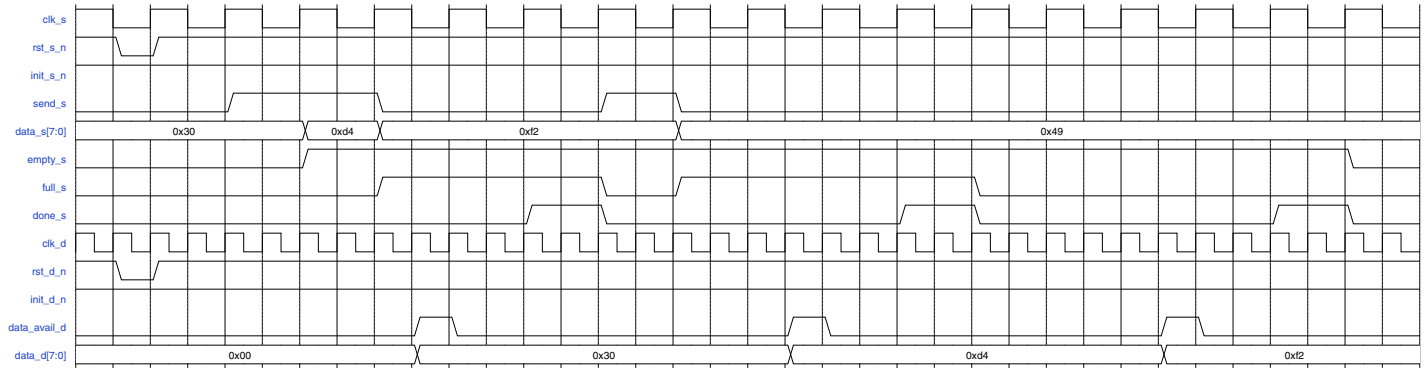
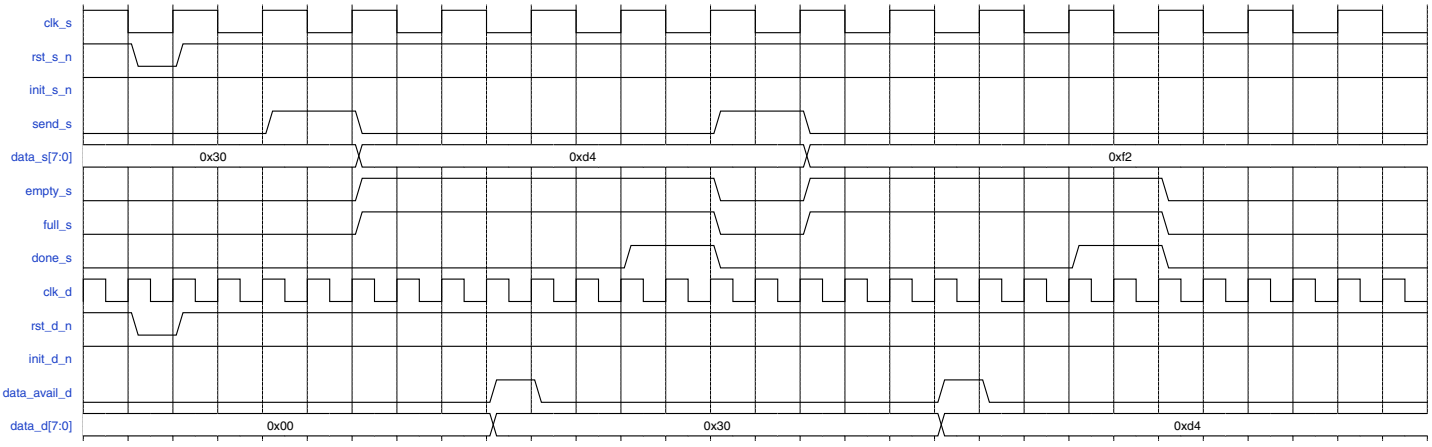


Figure 1-2 illustrates *send_mode* = 1, rising edge transition in produces a single clock cycle pulse out, timing from `send_s` to `done_s`, with `data_avail_d` (*r_sync_type* = 2, *f_sync_type* = 2).

Figure 1-2 Timing for pend_mode = 0, ack_delay = 0, send_mode = 1



Related Topics

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

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE,dw03;
use IEEE.std_logic_1164.all;
use DWARE.DW_Foundation_comp.all;

entity DW_data_sync_inst is
      generic (
        inst_width : NATURAL := 8;
        inst_pend_mode : NATURAL := 0;
        inst_ack_delay : NATURAL := 1;
        inst_f_sync_type : NATURAL := 2;
        inst_r_sync_type : NATURAL := 2;
        inst_tst_mode : NATURAL := 0;
        inst_verif_en : NATURAL := 1;
        inst_send_mode : NATURAL := 0
        );
      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;
        empty_s_inst : out std_logic;
        full_s_inst : out std_logic;
        done_s_inst : out 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_inst;


architecture inst of DW_data_sync_inst is

begin

    -- Instance of DW_data_sync
    U1 : DW_data_sync
    generic map ( width => inst_width,
                  pend_mode => inst_pend_mode,
                ack_delay => inst_ack_delay,
                f_sync_type => inst_f_sync_type,
                r_sync_type => inst_r_sync_type,
                tst_mode => inst_tst_mode,
                verif_en => inst_verif_en,
```

```
                send_mode => inst_send_mode )
    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,
              empty_s => empty_s_inst,
              full_s => full_s_inst,
              done_s => done_s_inst,
              data_avail_d => data_avail_d_inst,
              data_d => data_d_inst );


end inst;
-- pragma translate_off
library DW03;
configuration DW_data_sync_inst_cfg_inst of DW_data_sync_inst is
  for inst
  end for; -- inst
end DW_data_sync_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_data_sync_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, empty_s_inst,
          full_s_inst, done_s_inst, data_avail_d_inst, data_d_inst );

parameter width = 8;
parameter pend_mode = 0;
parameter ack_delay = 1;
parameter f_sync_type = 2;
parameter r_sync_type = 2;
parameter tst_mode = 0;
parameter verif_en = 1;
parameter send_mode = 0;

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 empty_s_inst;
output full_s_inst;
output done_s_inst;
output data_avail_d_inst;
output [width-1 : 0] data_d_inst;

    // Instance of DW_data_sync
    DW_data_sync #(width, pend_mode, ack_delay, f_sync_type, r_sync_type, tst_mode,
verif_en, send_mode)
      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), .empty_s(empty_s_inst), .full_s(full_s_inst),
.done_s(done_s_inst), .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