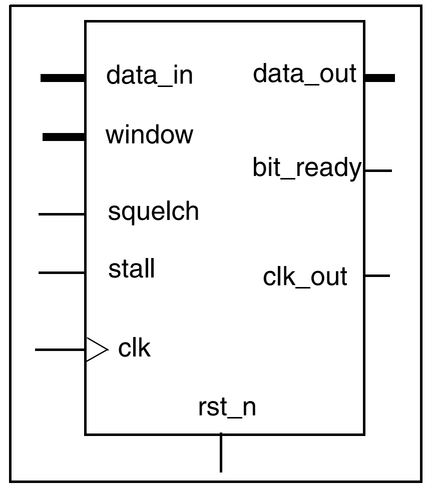
DW_dpll_sd

Digital Phase Locked Loop

Version, STAR and Download Information: IP Directory

Features and Benefits



- Parameterizable divisor (ratio of reference clock to baud rate)
- Multichannel data recovery (recovery of channels that accompany the locked channel)
- Stall input for power saving mode and/or prescaler (allowing one DW_dpll_sd to recover data at multiple rates)
- Squelch input for ignoring phase information when channel data is unknown or unconnected
- Sampling window control to aid data recovery under harsh conditions
- Parameterizable gain to meet a variety of application needs
- Parameterizable filter (controls phase correction reactiveness from minor phase errors)
- Provides minPower benefits with the DesignWare-LP license (Get the minPower version of this datasheet.)

Applications

- Networking
- Digital communication

Description

DW_dpll_sd is a digital phase-locked loop (DPLL) designed for data recovery.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Reference clock
rst_n	1 bit	Input	Asynchronous reset, active low
stall	1 bit	Input	Stalls everything except synchronizer, active high
squelch	1 bit	Input	Turns off phase detection. When high no phase correction is carried out leaving DPLL free running, active high

Table 1-1 Pin Description (Continued)

Pin Name	Width	Direction	Function
window	ceil(log2(*windows*))	Input	Sampling window selector[a]
data_in	*width* bit(s)	Input	Serial input data stream
clk_out	1 bit	Output	Recovered clock
bit_ready	1 bit	Output	Output data ready flag
data_out	*width* bit(s)	Output	Recovered output data stream

a. The minimum value must be 1.

Table 1-2 Parameter Description

Parameter	Values	Description
width	1 to 16 Default: 1	Number of input serial channels
divisor	4 to 256 Default: 4	Determines the number of samples per input clock cycle
gain	1 to 2 Default: 1	Phase correction factor for the absolute value of clock phase error greater than |1| 1 = 50% phase correction 2 = 100% phase correction
filter	0 to 8 Default: 2	Phase correction control for +/- 1 clock phase error region. 0 = no correction 1 = always correct For integer N > 1, correct after N samples at a current phase (such as, N consecutive samples at +1 or N consecutive samples at -1)
windows	1 to (divisor+1)/2 Default: 1	Number of sampling windows for the input serial data stream

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
str	Synthesis model	DesignWare

The DPLL functionality is achieved through the use of a state machine design that runs on a reference clock that is at least four times the bit rate of the data channel. After synchronization of the `data_in` input signal to the reference clock (using a three stage synchronizer) the oversampled signal is monitored for transitions. When transitions are detected, the state machine evaluates (based on what state it's currently in) whether or not an adjustment is needed to align the state machine's natural cycle to the incoming data stream. The amount of the adjustment depends on how far out of alignment the state machine is and the related parameter values used.

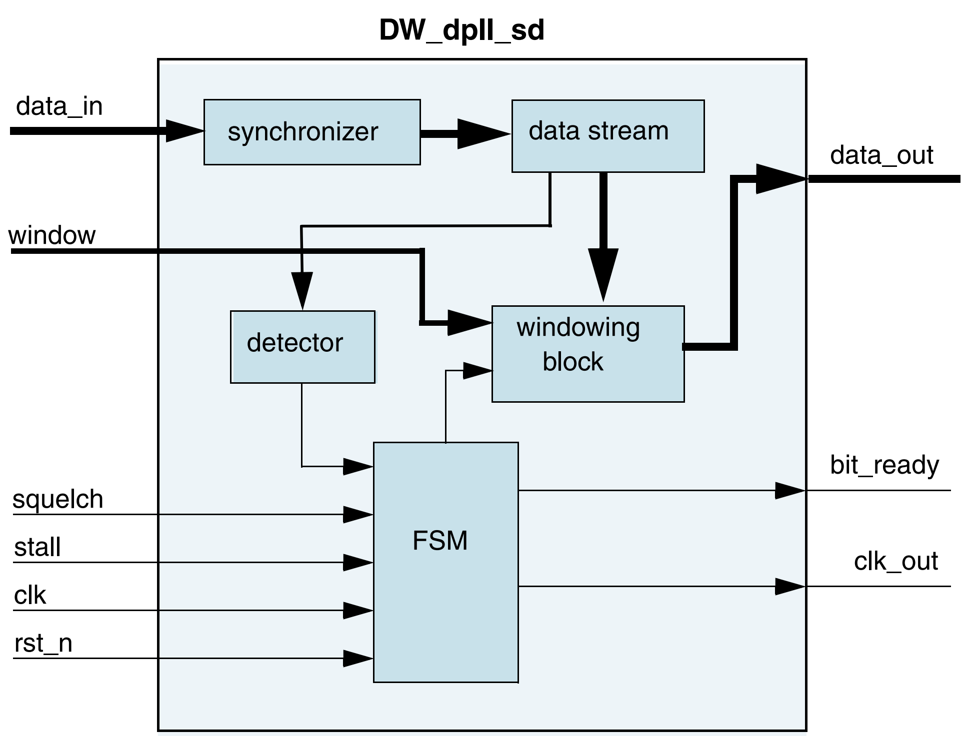
Recovered data is output on the `data_out` bus and the `bit_ready` output indicates the arrival of each data bit at the `data_out` bus. The `clk_out` port can provide a derived clock which is phase-aligned to cleanly clock recovered data from the `data_out` bus on its rising edge.

A `stall` input port is provided for power savings and for easily implementing multiple data rates through prescaling. A `squelch` input is provided to command the DPLL to ignore phase information during times when the link is not receiving valid data.

A windowing control bus is provided to allow control of where to sample data within a bit time. This may be useful in recovering noisy data especially if the data rate is significantly different from the DPLLs nominal rate.

The DW_dpll_sd can recover multiple channels of data as long as the channels are well phase-aligned with each other. See the *width* parameter functional description for more detail.

Figure 1-1 Functional Block



Functional Description

width -
The *width* parameter sets the number of serial input channels processed by the DW_dpll_sd component. Although as many as 16 channels can be recovered, phase locking operates only on the `data_in[0]` channel. Thus, all other channels received must be well phase-aligned with `data_in[0]` in order to cleanly recover data from them. Accompanying channels may be used to bring link status or other non-data information through the DPLL so as to keep this information aligned to the data stream.

divisor -
The static *divisor* parameter determines the number of divisions or phase states per bit time. This controls the sample rate, therefore, the granularity or coarseness of the phase corrections.

A *divisor* of 4, not only gives the coarsest granularity, but is the minimum number of samples per bit time to ensure an accurate sampling of the input signal. A large (*divisor*) value increases the number of correction (phase) states or divisions that a bit time is sub-divided into resulting in finer phase corrections.

Since each phase state is one reference clock period long, the number of phase states per bit time also determines the free running or center frequency of the DPLL. Increasing the *divisor* value increases the effective output clock period. For example, using a *divisor* value of 10 for a reference clock of 50 Mhz will divide the bit time into 10 states and set the center frequency to 5 Mhz.

The free running or nominal frequency should be chosen as close as possible to the input frequency or baud rate to ensure the best phase correction conditions.

stall -
The `stall` input port enhances the dynamics of the divisor to center frequency relationship. When using only the *divisor* parameter, compared with the addition of using the `stall` pin, each frequency is associated with only one *divisor* value to any one reference clock input. This means there is a set (inflexible) one-to-one correspondence between the *divisor* and center frequency.

Therefore, a large *divisor* value for a comparatively high reference clock to expected input frequency may be inconvenient especially if there is no desire to increase the granularity.

By using the `stall` pin to prescale the reference clock input, `clk`, a more flexible choice of sample rate can be chosen for a frequency.

Naturally, this allows for maintaining the same granularity over different frequencies or a coarser phase correction at a desired frequency balanced by a lower power system.

The resulting baud time will be the product of the *divisor* value and prescale factor. So, a *divisor* value of 5 for a reference clock of 50 Mhz, with the `stall` pin driven by a divide-by-two circuit translates to a center frequency of 5 Mhz.

Care should be taken in the use of pre-scaling to affect frequency and sampling rate combinations. Stalling the DPLL freezes all outputs at their current states. This could lead to output data mis-sampling if not considered. For example, stalling the DPLL right at the time the `bit_ready` output signal goes high will freeze it at that state (see Figure 1-6 on page 10). So, for such a situation not gating the `bit_ready` signal with the `stall` signal, for instance, to generate a secondary data arrival flag will lead to incorrectly validating the `data_out` bus values if just reading the `bit_ready` output flag while in the stall state.

Input data passing through the synchronizer stage of the DPLL is unaffected by the `stall` pin.

filter -
The *filter* parameter controls the degree of adjustment in the plus or minus one phase error region, which

represents the edge detection of input data transition one division off in either direction of the center or zero error reference for correction. This helps alleviate possible jitter propagation due to any dithering that occurs in this region.

gain -
The *gain* parameter determines the amount of phase correction for each detected phase error. A *gain* value of 1 results in a 50% phase correction of any detected error and a *gain* value of 2 in a 100% correction. A 50% phase correction (*gain* = 1) means a phase error of 6 will be corrected by 3 states compared with a 6 state correction for the case of a 100% phase correction (*gain* = 2).

windows -
The *windows* parameter feature allows for a number of sample points along the input stream to be used. A zero reference point at the approximate center of the bit time is chosen as the default sample point for output data. The reference point is the center of the bit time for even integer values of *divisor* and slightly off center for odd values. Additional sample points, if defined, are accessed through the `window` pin by use of indices.

The index pattern for the window follows an alternating left and right sequence with the default zeroth index as the center pivot point, `window[0]`.

This windowing sequence is analogous to a spiral search pattern but in one dimension, where the goal is to maximize the probability of finding a good sample point assuming the center as the optimal sample point. See Figure 1-3 on page 7.

This windowing option, provides the ability to compensate for noisy conditions such as asymmetric noise along the input stream and/or frequency difference or static error between the transmit and receive clock domains.

Application Notes

Note that with each phase correction that is carried out, the `clk_out` signal will either be shortened or lengthened by an amount equal to the phase correction.

Therefore it is generally desired to use the `bit_ready` output signal as a flag to indicate when `data_out` is ready to be read during normal operations.

Note that the `squelch` input disables the phase detection mechanism leaving the DPLL free running

Examples of Application Configurations

Three following common configurations can be used to cover most applications:

- For the non-coherent phase DPLL case and where there is more jitter with respect to the reference clock, *gain=1*, *filter=2*.
- For the coherent phase DPLL case, where the transmission and reception clock phase relation do not change with respect to each other, *gain=2*, *filter=0*.
- For the non-coherent phase DPLL condition and where there is less jitter and smoother corrections are desired, *gain=1*, *filter>2*.

Timing Diagrams

The following figures show various timing conditions for the DW_dpll_sd.

Figure 1-2 Functional Operation: Initialize reset followed by phase correction

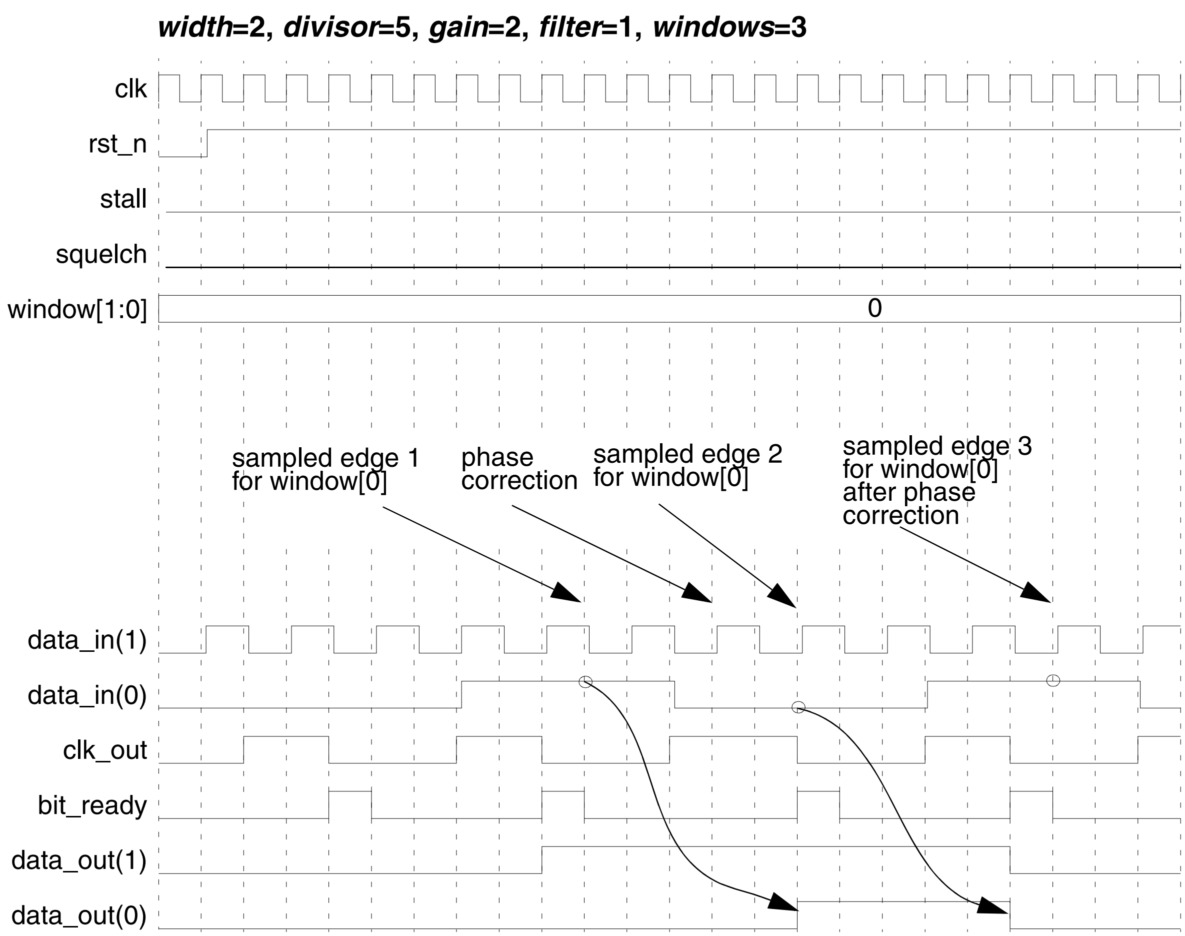


Figure 1-3 Window Shifting Sequence Concept

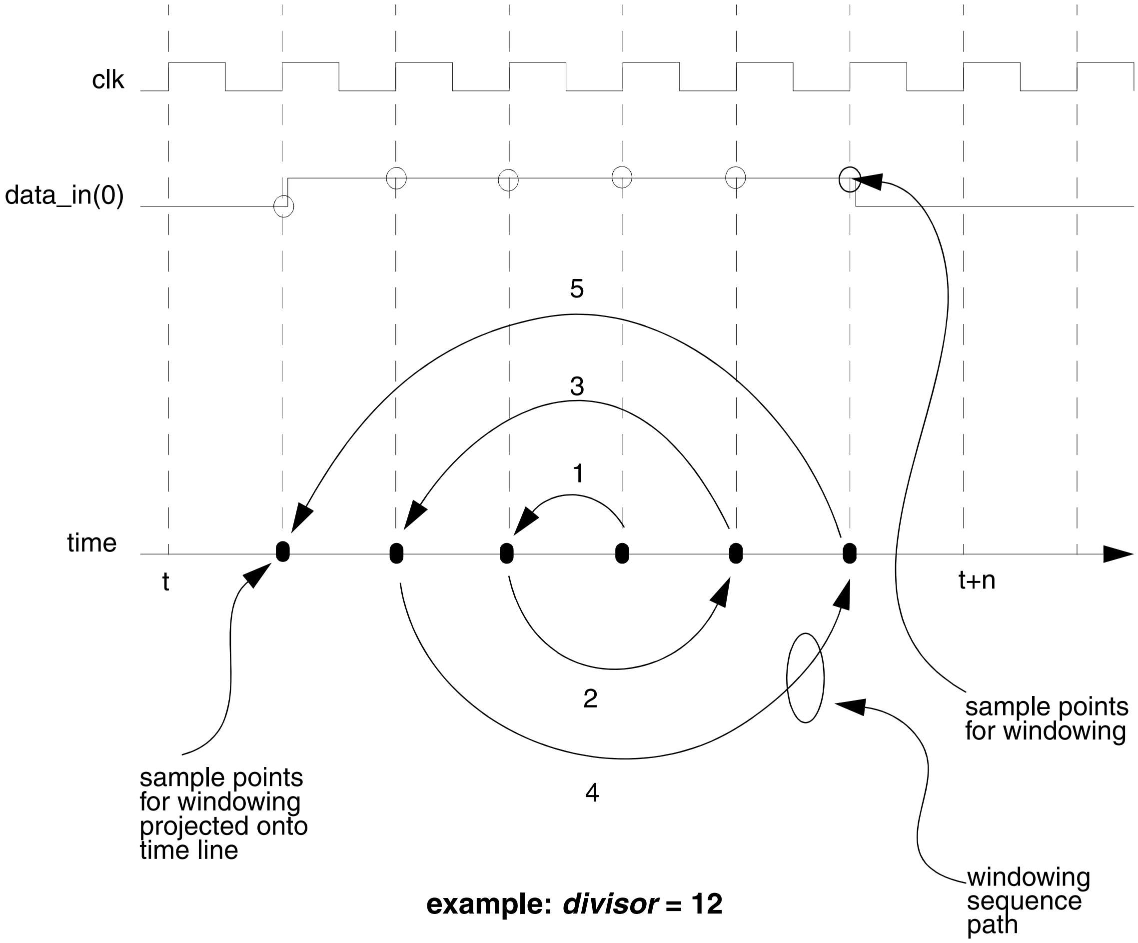


Figure 1-4 Functional Operation: Window shifting

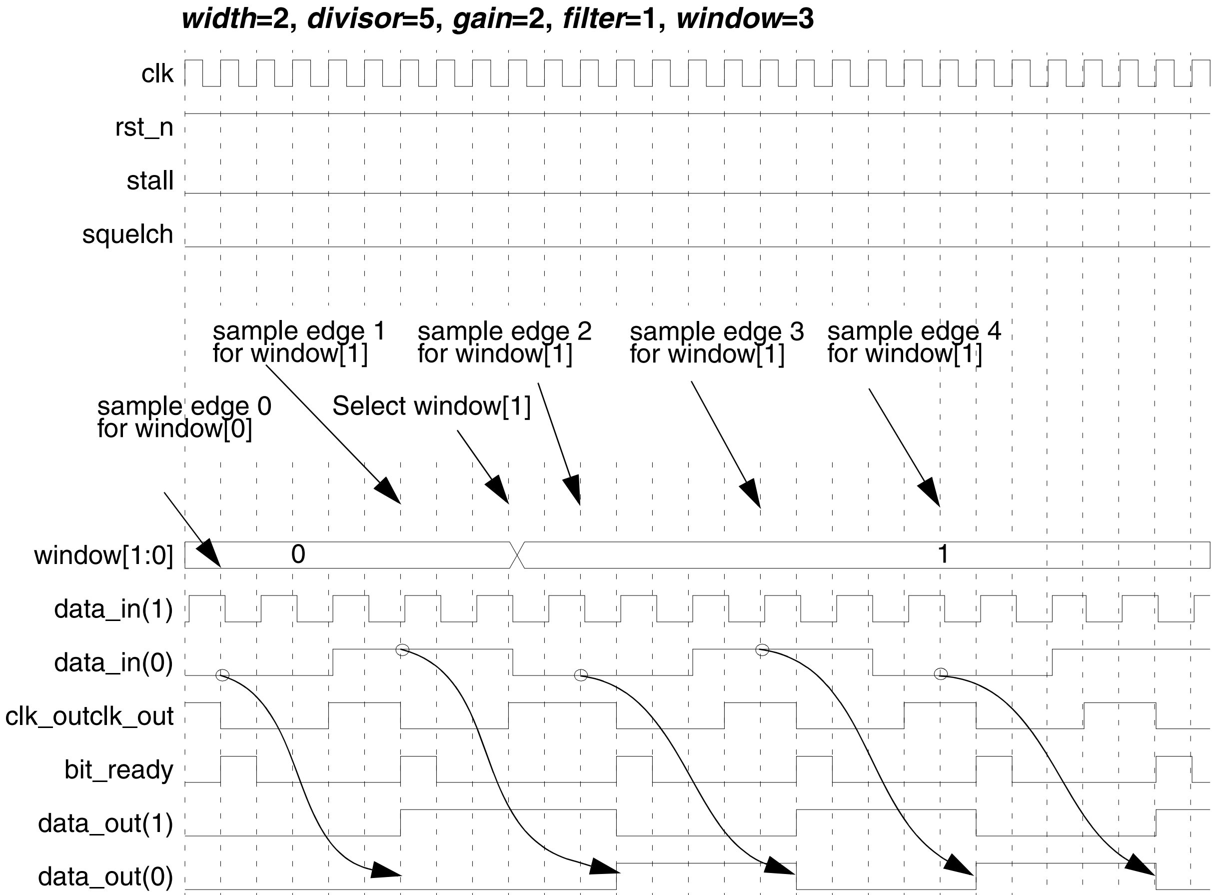


Figure 1-5 Functional Operation: Squelch active

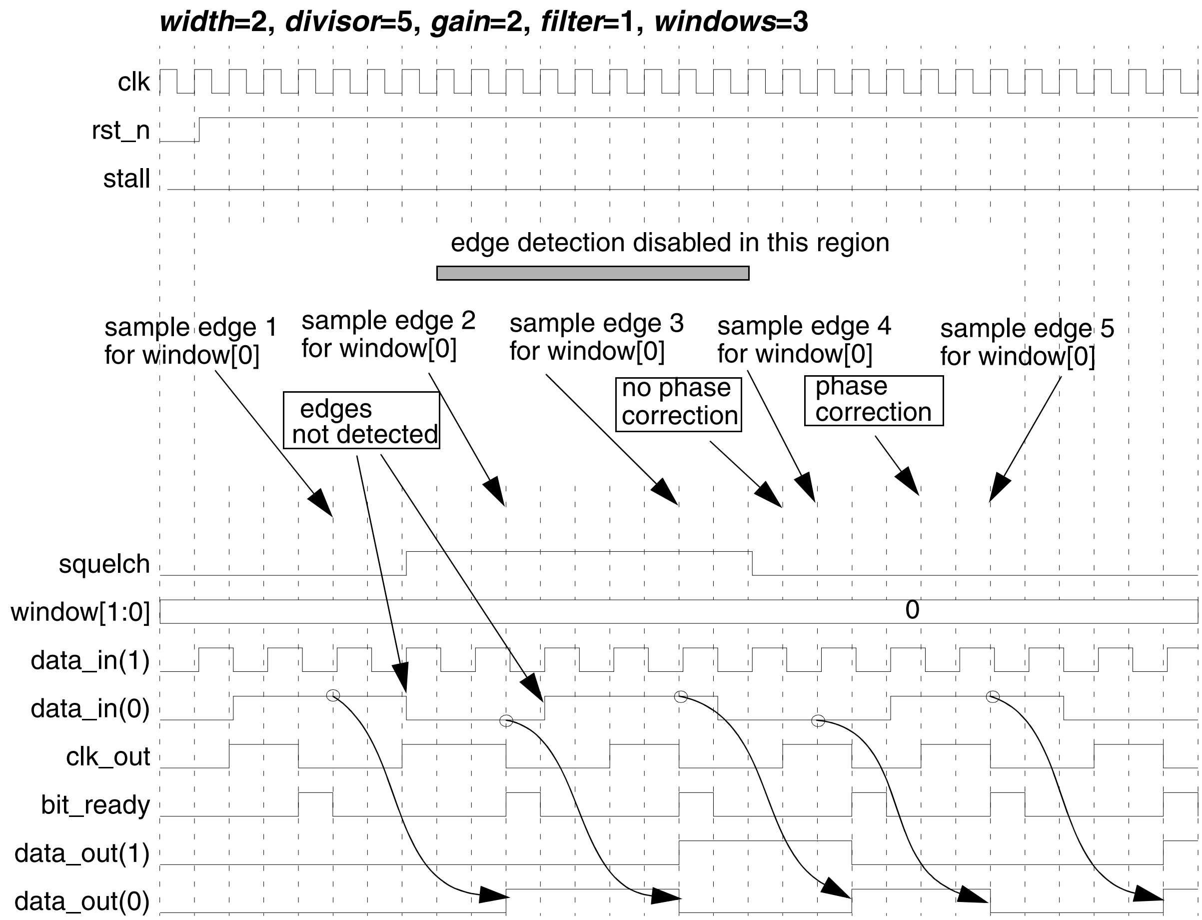
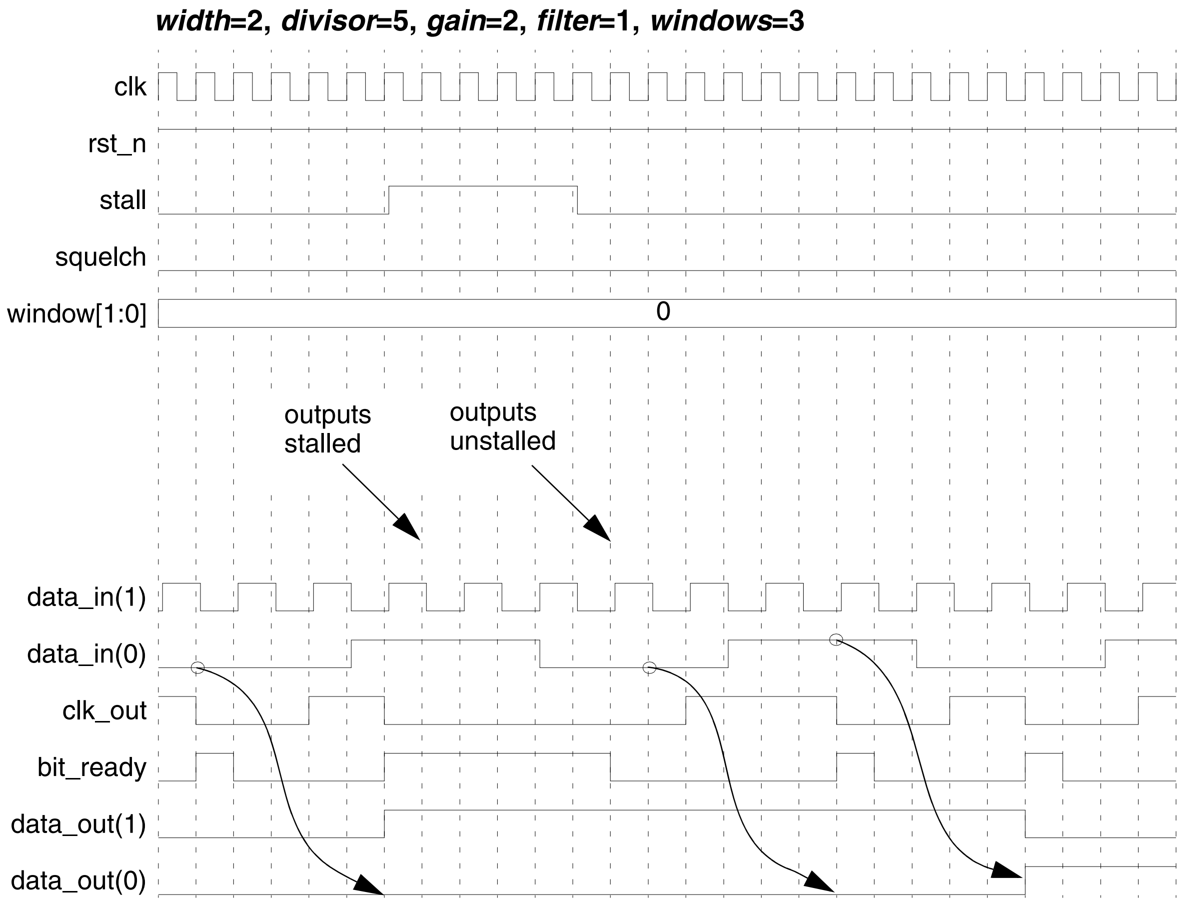


Figure 1-6 Functional Operation: Stall active



Related Topics

- Logic – Sequential 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_dpll_sd_inst is
  generic (inst_width : INTEGER := 1;
           inst_divisor : INTEGER := 5;
           inst_gain : INTEGER := 1;
           inst_filter : INTEGER := 2;
           inst_windows : INTEGER := 1);
  port (inst_clk : in std_logic;
        inst_rst_n : in std_logic;
        inst_stall : in std_logic;
        inst_squelch : in std_logic;
        inst_window: in std_logic_vector(bit_width(inst_windows)-1 downto 0);
        inst_data_in : in std_logic_vector(inst_width-1 downto 0);
        clk_out_inst : out std_logic;
        bit_ready_inst : out std_logic;
        data_out_inst : out std_logic_vector(inst_width-1 downto 0) );
end DW_dpll_sd_inst;

architecture inst of DW_dpll_sd_inst is
begin

  -- Instance of DW_dpll_sd
  U1 : DW_dpll_sd
    generic map ( width => inst_width, divisor => inst_divisor,
                  gain => inst_gain, filter => inst_filter,
                  windows => inst_windows )
  port map ( clk => inst_clk, rst_n => inst_rst_n, stall => inst_stall,
             squelch => inst_squelch, window => inst_window,
             data_in => inst_data_in, clk_out => clk_out_inst,
             bit_ready => bit_ready_inst, data_out => data_out_inst );
end inst;

-- pragma translate_off
configuration DW_dpll_sd_inst_cfg_inst of DW_dpll_sd_inst is
  for inst
  end for; -- inst
end DW_dpll_sd_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_dpll_sd_inst( inst_clk, inst_rst_n, inst_stall,
                        inst_squelch, inst_window, inst_data_in,
                        clk_out_inst, bit_ready_inst, data_out_inst );

  parameter inst_width = 1;
  parameter inst_divisor = 5;
  parameter inst_gain = 1;
  parameter inst_filter = 2;
  parameter inst_windows = 3;
  `define bit_width_windows 2 // ceil(log2(inst_windows)

  input inst_clk;
  input inst_rst_n;
  input inst_stall;
  input inst_squelch;
  input [`bit_width_windows-1 : 0] inst_window;
  input [inst_width-1 : 0] inst_data_in;
  output clk_out_inst;
  output bit_ready_inst;
  output [inst_width-1 : 0] data_out_inst;

  // Instance of DW_dpll_sd
  DW_dpll_sd #(inst_width, inst_divisor, inst_gain,
               inst_filter, inst_windows)
    U1 ( .clk(inst_clk), .rst_n(inst_rst_n), .stall(inst_stall),
         .squelch(inst_squelch), .window(inst_window),
         .data_in(inst_data_in), .clk_out(clk_out_inst),
         .bit_ready(bit_ready_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