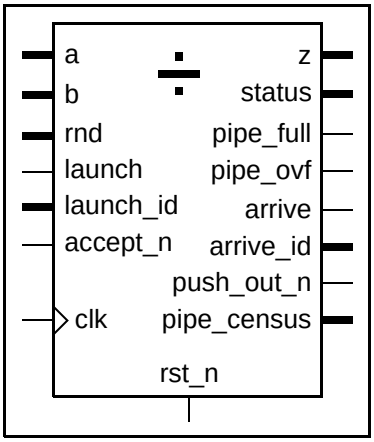
DW_lp_piped_fp_div

Low Power Pipelined Floating Point Divide

Version, STAR, and myDesignWare Subscriptions: IP Directory

Features and Benefits

Revision History



- The precision format is parameterizable for either IEEE single, double precision, or a user-defined custom format
- Fully compatible with the IEEE 754 standard with proper set of parameters
- Fully compatible with the IEEE 754 standard with proper set of parameters
- DesignWare datapath generator is employed for better timing and area
- Saves power by only enabling stages as needed
- Saves power based on input data patterns
- Parameter controlled pipeline stages
- Flow control to interface directly to FIFO
- Flow control interfaces to another managed pipe
- Bubble removal extends depth of subsequent FIFO by pipe depth
- Parameter sized identifier tracks data operations

Description

DW_lp_piped_fp_div is a floating point operator that divides two floating point operands, `a` divided by `b`, to produce a floating point quotient, `z`. The input `rnd` is a 3-bit rounding mode and the output `status` is an 8-bit status flag. For details about rounding modes and status flags, see the *Datapath Floating Point Overview*.

The operation is conditionally managed by the DW_lp_pipe_mgr, a pipeline controller. When *no_pm* = 1 (no pipeline management), the component is configured with a user-selectable number of *stages* of logic, allowing Design Compiler to optimize the logic between register stages to reduce power and area. Design Compiler is able to take advantage of the enable signals provided in the design to optimize the combinational logic throughout the divide operation. When *no_pm* = 0, the pipeline manager (DW_lp_pipe_mgr) manages the enabling of register stages of a pipeline based on launch requests that track the progress of a pipelined operation. Enabling stages of the pipeline only when they need to be clocked reduces dynamic power and is further enhanced through clock gate insertion (which requires Power Compiler). Launch requests can be accompanied by a launch ID, which can be used as a tag to identify operation results as they pass out of the pipe. A `pipe_census` output monitors the number of operations in the pipe.

The DW_lp_piped_fp_div component provides the capability to insert clock gating elements and insert datapath gating cells to minimize power consumption; it is only available for Design Compiler versions C-2009.06 and later.

Component pins are described in Table 1-1 and configuration parameters are described in Table 1-2.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock source
rst_n	1 bit	Input	Asynchronous reset
a	*exp_width* + *sig_width* + 1 bits	Input	Dividend
b	*exp_width* + *sig_width* + 1 bits	Input	Divisor
rnd	3 bits	Input	Rounding mode; supports all rounding modes described in the *Datapath Floating-Point Overview*; The `rnd` port takes effect only when *faithful_round* = 0.
z	*exp_width* + *sig_width* + 1 bits	Output	Input `a` divided by `b`
status	8 bits	Output	Status flags for the result `z` For details, see STATUS Flags in the *Datapath Floating-Point Overview.*
launch	1 bit	Input	Active high control input to launch data into pipe
launch_id	*id_width* bits	Input	ID associated with a launch
pipe_full	1 bit	Output	Status indication of no available slot in pipe
pipe_ovf	1 bit	Output	Error indicating pipe overflow (data lost)
accept_n	1 bit	Input	Flow control input, active low
arrive	1 bit	Output	Quotient result is valid on `z`
arrive_id	*id_width* bits	Output	`launch_id` from the originating `launch` that produced the quotient result
push_out_n	1 bit	Output	Used with FIFO, active low
pipe_census	ceiling(log_2(maximum(1,*in_reg* + (stages - 1) + *out_reg*) + 1)) bits	Output	Output bus indicating the number of pipe stages currently occupied

Table 1-2 Parameter Description

Parameter	Values	Description
sig_width	2 to 253 bits Default: 23	Word length of fraction field of floating point numbers `a`, `b`, and `z`

Table 1-2 Parameter Description (Continued)

Parameter	Values	Description
exp_width	3 to 31 bits Default: 8	Word length of biased exponent of floating point numbers `a`, `b`, and `z`
ieee_compliance	0 or 1 Default: 0	Level of support for IEEE 754: ■ 0: No support for IEEE 754 NaNs and denormals; NaNs are considered infinities and denormals are considered zeros ■ 1: Fully compliant with the IEEE 754 standard, including support for NaNs and denormals For more, see IEEE 754 Compatibility in the *Datapath Floating-Point Overview*.
faithful_round	0 or 1 Default: 0	Choose either a specific rounding mode (set by `rnd`) or a general rounding mode that allows maximum 1 ulp error ■ 0: Rounding mode is specific, as set by the `rnd` port; this choice increases the size of the resulting implementation. ■ 1: Rounding mode is general and, for *sig_width* ≤ 23, allows a maximum of 1 ulp error[a]; this choice decreases the size of the resulting implementation. When *faithful_round* = 1, note the following: - The inexact status flag in the output is not meaningful. - The other status flags will match one of the possible outputs for the calculation when *faithful_round* = 0.
op_iso_mode	0 to 4 Default: 0	Operand isolation mode (controls datapath gating for minPower flow) Allows you to set the style of minPower datapath gating for this module ■ 0: Use the DW_lp_op_iso_mode[b] synthesis variable ■ 1: 'none' ■ 2: 'and' ■ 3: 'or' ■ 4: Preferred gating style: ‘and’ Datapath gating is inserted only when there are no input registers on the operands at the component boundary. When inserted, datapath gating circuits are placed immediately after the input ports of the component (see Figure 1-2 on page 6).
id_width	1 to 1024 Default: 8	Width of input `launch_id` and output `arrive_id`
in_reg	0 or 1 Default: 0	Input register control ■ 0: No input register ■ 1: Include input register
stages	1 to 1022 Default: 4	Number of logic stages
out_reg	0 or 1 Default: 0	Output register control ■ 0: No output register ■ 1: Include output register

Table 1-2 Parameter Description (Continued)

Parameter	Values	Description
no_pm	0 or 1 Default: 1	No pipeline management used ■ 0: Use pipeline management ■ 1: Do not use pipeline management
rst_mode	0 or 1 Default: 0	Reset mode ■ 0: Asynchronous reset on `rst_n` ■ 1: Synchronous reset on `rst_n`

a. When *faithful_round* = 1 and *sig_width* > 23, the result can exceed 1 ulp for some corner cases. Configurations tested for this parameter range do not show errors larger than 2 ulps.
b. The DW_lp_op_iso_mode synthesis variable is available only in Design Compiler. DW_lp_op_iso_mode sets a global style of datapath gating. To use the global style, set *op_iso_mode* to '0', Note that If the *op_iso_mode* parameter is set to '0' and DW_lp_op_iso_mode is either not set or set to 0', then no datapath gating is inserted for this component.

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
rtl	Synthesis model	■ DesignWare (P-2019.03 and later) ■ DesignWare-LP[a] (before P-2019.03)

a. For Design Compiler versions before P-2019.03, see "Enabling minPower" on page 13.

Table 1-4 Simulation Models

Model	Function
DW03.DW_LP_PIPED_FP_DIV_CFG_SIM	Design unit name for VHDL simulation
dw/dw03/src/DW_lp_piped_fp_div_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_lp_piped_fp_div.v	Verilog simulation model source code

Block Diagram

Figure 1-1 shows the block diagram for DW_lp_piped_fp_div with *no_pm* = 1.

Figure 1-1 Block Diagram of DW_lp_piped_fp_div with No Pipeline Manager (*no_pm* = 1)

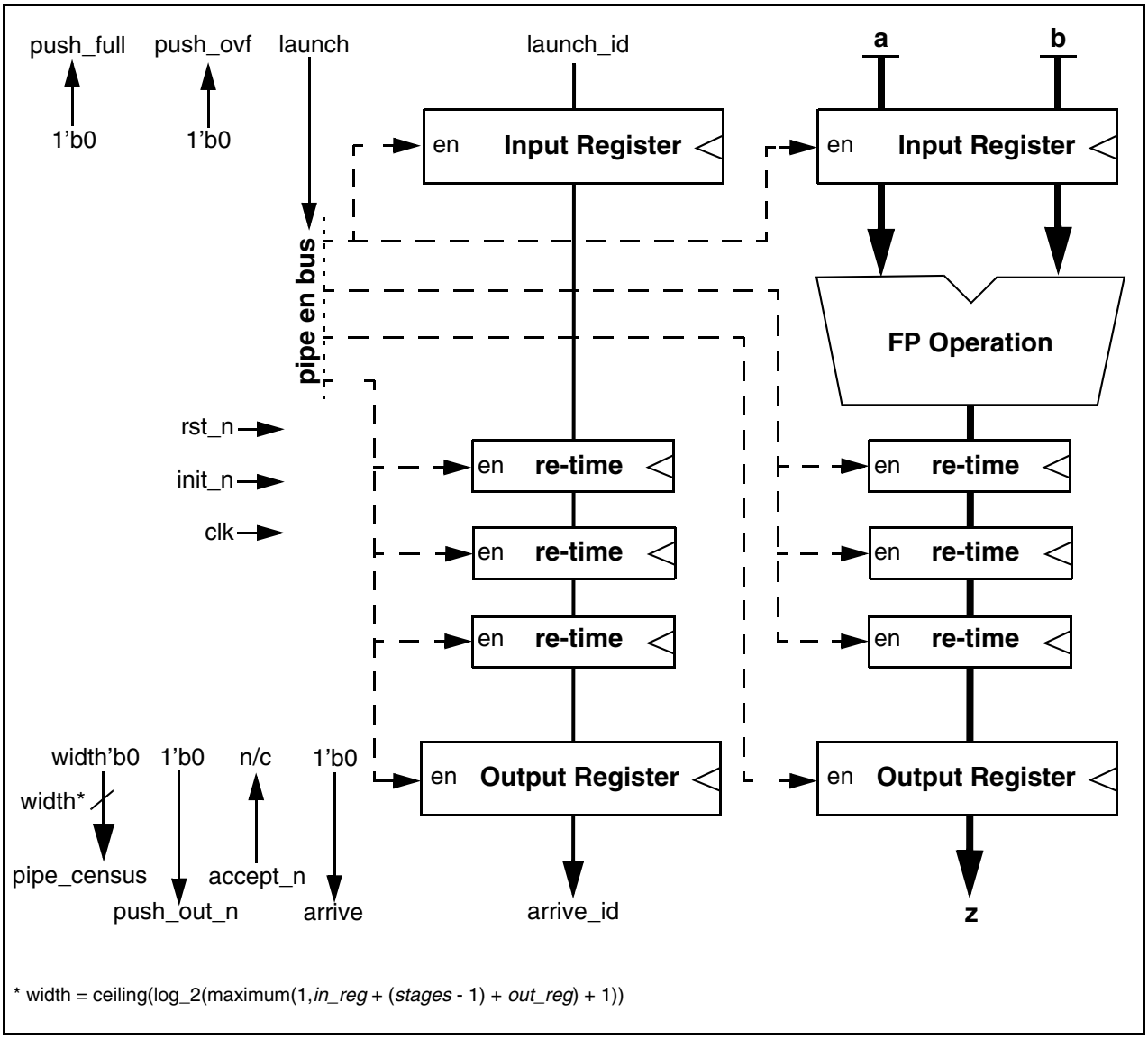


Figure 1-2 on page 6 shows the block diagram for DW_lp_piped_fp_div with *no_pm* = 0.

Figure 1-2 Block Diagram of DW_lp_piped_fp_div with Pipeline Manager (*no_pm* = 0)

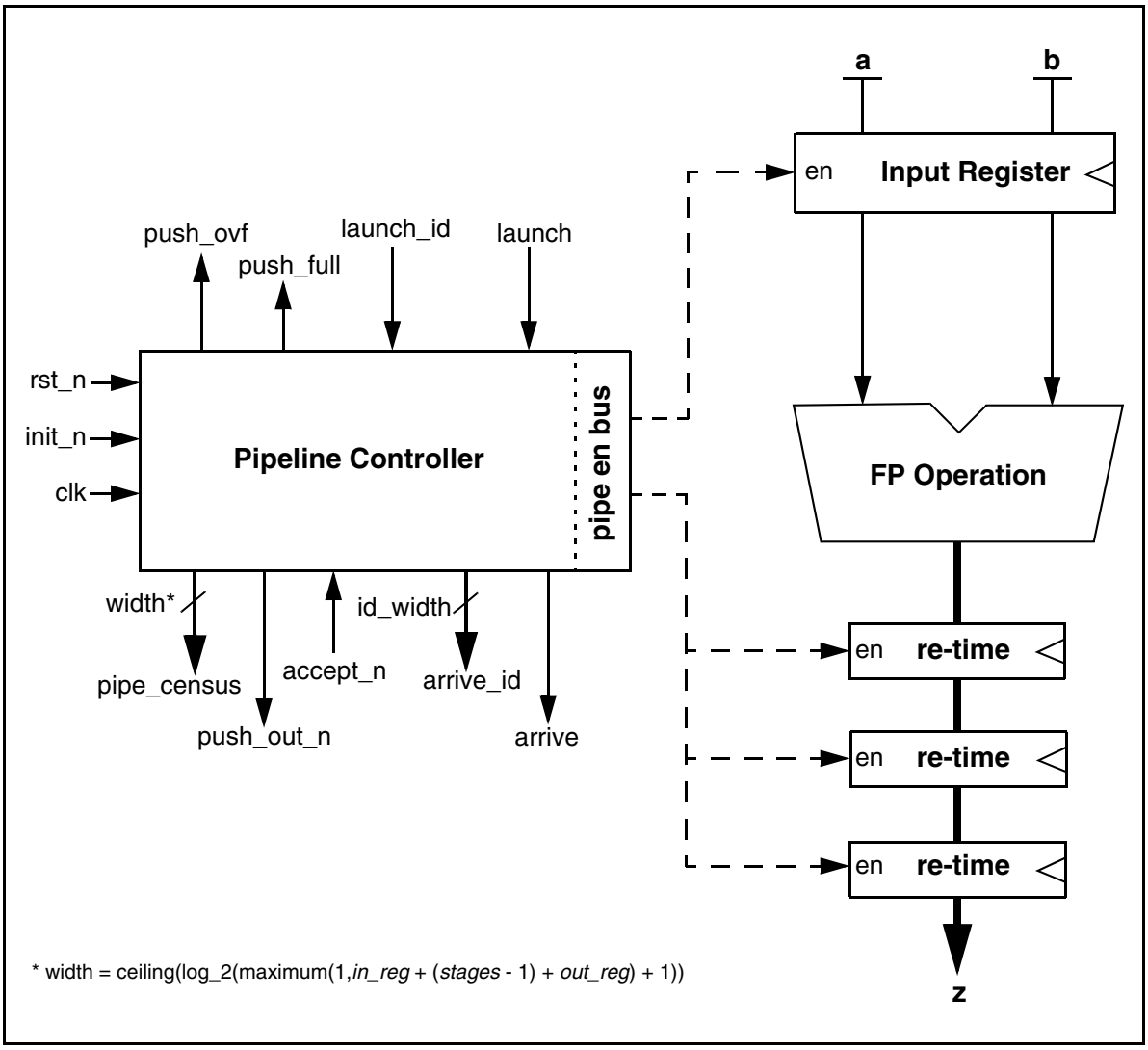
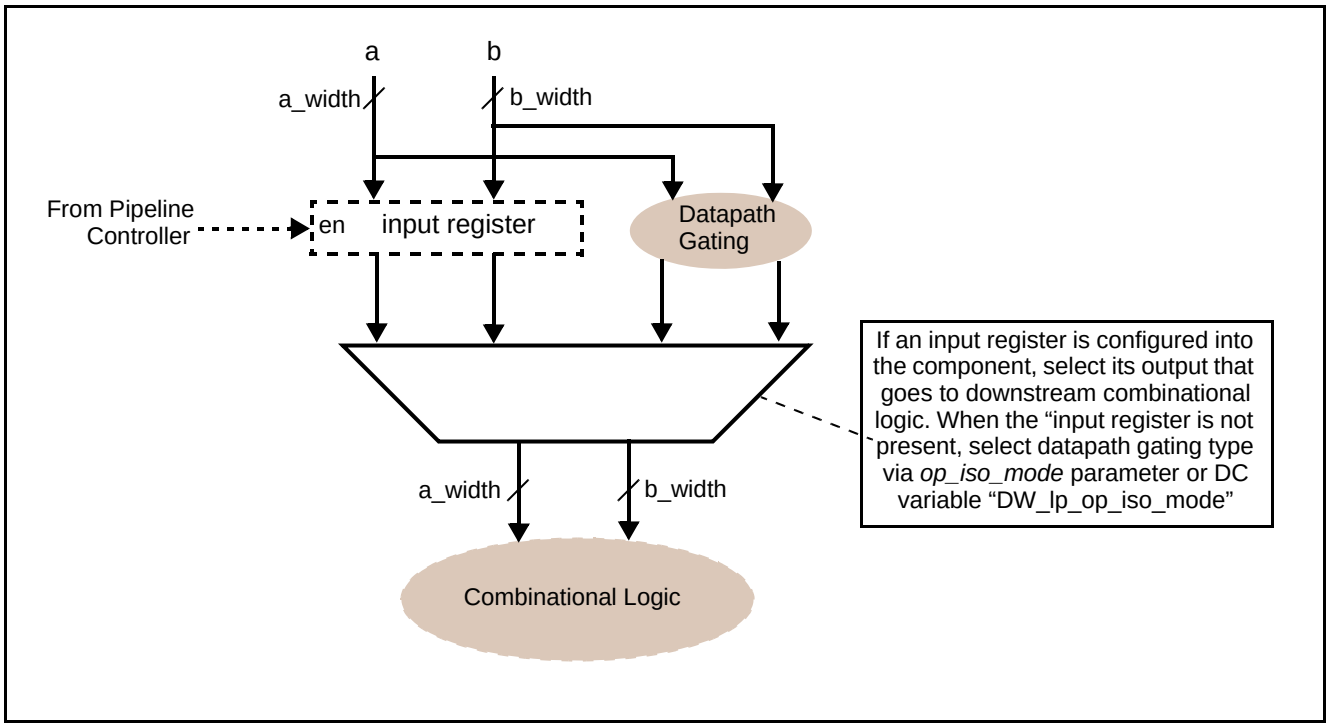


Figure 1-3 shows where datapath gating is inserted when the *op_iso_mode* parameter enables it.

Figure 1-3 Location of Datapath Gating (If Inserted)



Number of Pipeline Register Levels

Setting the value for the parameters *in_reg*, *stages*, and *out_reg* (see Table 1-5) determines the number of pipeline register level(s) that are inserted and, therefore, the number of clock cycles for the output (z) and `status` results to propagate out varies.

Table 1-5 Number of Pipeline Register Levels

in_reg	*out_reg*	Number of Pipeline Register Levels
0	0	*stages* - 1
0	1	stages
1	0	stages
1	1	*stages* + 1

This DW_lp_piped_fp_div is designed to make it easy to pipeline floating point divide logic using the register retiming features of Design Compiler (DC). It also contains parameter controlled input and output registers which will stay in place at their respective block boundary, that is, they are not allowed to be moved by DC's register retiming features. The input and output registers are not available when using DC versions earlier than A-2007.12.

The parameter *stages* refers to the number of logic stages desired after register retiming is performed. The number of register levels is not necessarily the same as the number of logic stages. If no input or output registers are used (when *in_reg* = 0 or *out_reg* = 0), then there is one fewer register level than logic stages. If either an input register or output register is specified, then the number of register levels is the same as the number of logic stages. If both input and output registers are specified, then the number of register levels is the number of logic *stages* + 1 (refer to Table 1-5). The number of pipeline register levels that can be retimed is always *stages* - 1.

Pipeline Control and Power Savings (*no_pm* = 0)

When the pipeline manager is used (*no_pm* = 0), as seen in Figure 1-2 on page 6, pipeline control logic monitors the activity for power-saving opportunities. When there is inactivity on a particular register level, the pipeline control disables that level to promote power savings. Furthermore, if using the Synopsys Power Compiler tool, the presence of the pipeline control and its wiring to the pipeline register levels provides an opportunity for increased power reduction in the form of clock gating.

Along with the potential power savings that the pipeline control provides, the pipeline manager can be used to improve performance in cases where intermittent launch operations are present and there contains first-in first-out (FIFO) structures upstream and downstream of the DW_lp_piped_fp_div. The handshake is made between the DW_lp_piped_fp_div and the external FIFOs via the `accept_n` and `pipe_full` ports. Effectively, the DW_lp_piped_fp_div can be considered part of the external FIFO structures.

The performance gain comes when inactive stages (bubbles) are detected. These pipeline 'bubbles' are removed to produce a contiguous set of active pipeline stages. The result is empty pipeline slots at the head of (or entering) the DW_lp_piped_fp_div pipeline for new operations to be launched. The `accept_n` input controls advancing the shifting of operations through the pipeline when a valid result is available (`arrive` = 1).

When the DW_lp_piped_fp_div pipeline is full of active entries, the `pipe_full` output is driven to 1. To disable this feature in cases where no external FIFOs are present, set the `accept_n` input to 0, which effectively eliminates any flow control. At the same time, the `pipe_full` output would always be 0.

When there is no pipelining (*in_reg* = 0, *stages* = 1, and *out_reg* = 0), the pipeline control signals remain active, with one exception: `pipe_census` is always set to 0.

Data Tracking

To assist in tracking of 'launched' operands, the pipeline control logic provides interface ports called `launch_id` and `arrive_id`. The `launch_id` input is assigned a value during an active launch operation. Given that `launch_id` values are unique in successive launch operations, the results can be distinguished from one another with the assertion of `arrive` and the associated `arrive_id`. The `arrive_id` is the `launch_id` from the originating `launch` that produced the valid `z` and `status` outputs.

When the pipeline manager is not used (*no_pm* = 1), the launch input is connected directly to the enable line of the pipeline registers. The pipeline stalls when `launch` = 0.

System Resets (Synchronous or Asynchronous)

Two system reset modes are available from the `rst_n` input: asynchronous or synchronous. Asynchronous system reset is implemented when *rst_mode* = 0 and synchronous system reset is applied when *rst_mode* = 1.

During reset conditions, all the output ports are set to 0.

Suppressing Warning Messages During Verilog Simulation

The Verilog simulation model includes macros that allow you to suppress warning messages during simulation.

To suppress all warning messages for all DWBB components, define the DW_SUPPRESS_WARN macro in either of the following ways:

- Specify the Verilog preprocessing macro in Verilog code:

  ```
  `define DW_SUPPRESS_WARN
  ```

- Or, include a command line option to the simulator, such as:

  `+define+DW_SUPPRESS_WARN` (which is used for the Synopsys VCS simulator)

The warning messages for this model include the following:

- If values other than 1 or 0 are present on a clock port, the following message is displayed:

  ```
  WARNING: <instance_path>.<clock_name>_monitor:
      at time = <timestamp>, Detected unknown value, x, on <clock_name> input.
  ```

  To suppress only this warning message for all DWBB components, use the following macro:

  - Define the DW_DISABLE_CLK_MONITOR macro. You can define this macro in the following ways:
    - Specify the Verilog preprocessing macro in Verilog code:

      ```
      `define DW_DISABLE_CLK_MONITOR
      ```

    - Or, include a command line option to the simulator, such as:

      `+define+DW_DISABLE_CLK_MONITOR` (which is used for the Synopsys VCS simulator)

  This message is also suppressed using the DW_SUPPRESS_WARN macro explained earlier.

- If an invalid rounding mode has been detected on `rnd`, the following message is displayed:

  ```
  WARNING: <instance_path>:
      at time = <timestamp>: Illegal rounding mode.
  ```

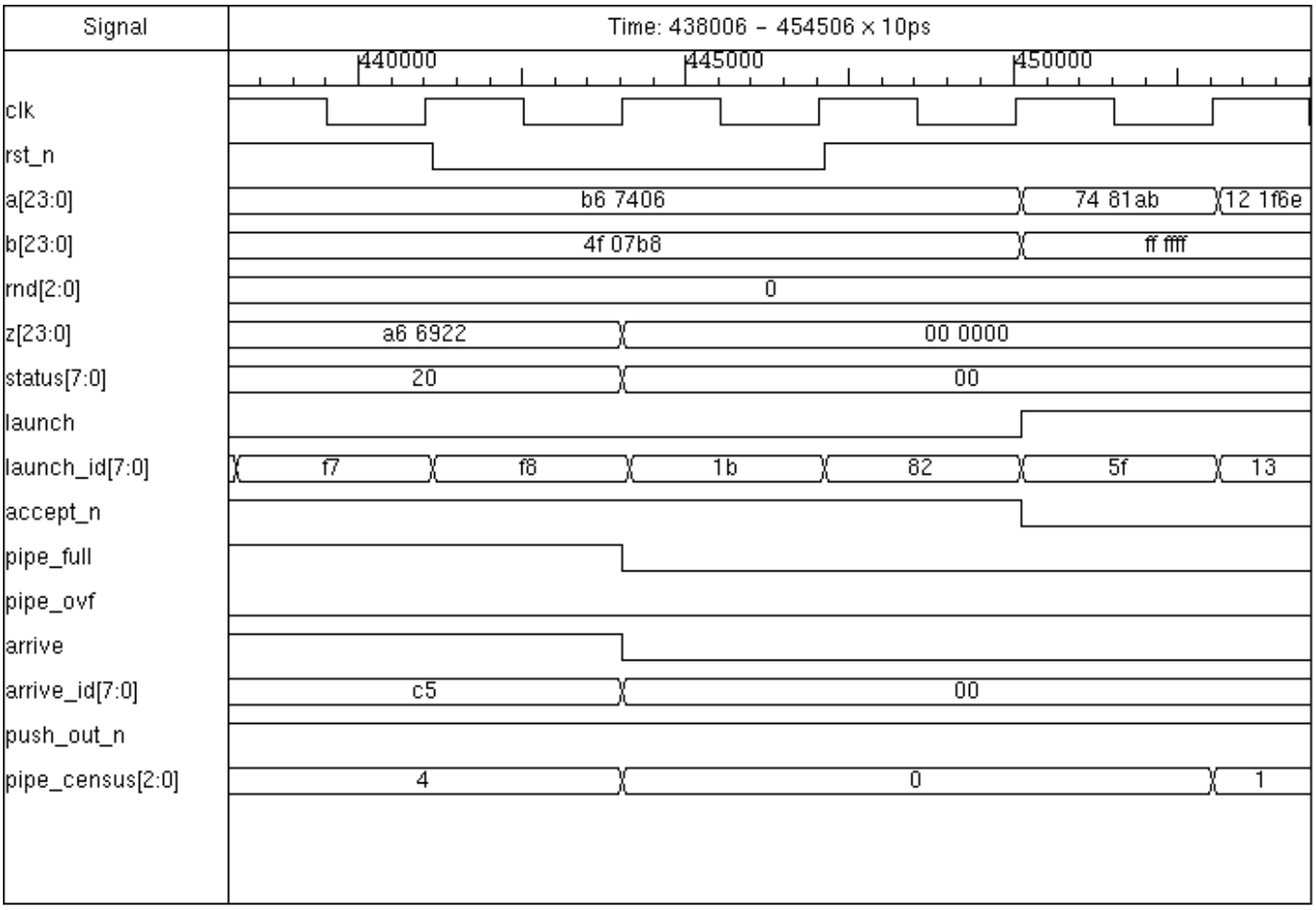
  To suppress this message, use the DW_SUPPRESS_WARN macro explained earlier.

Timing Waveforms

Synchronous Reset

Figure 1-4 shows the result of *reset_mode* = 1. In this mode, the outputs are reset after the `rst_n` signal is active (low), and the positive edge of the `clk`.

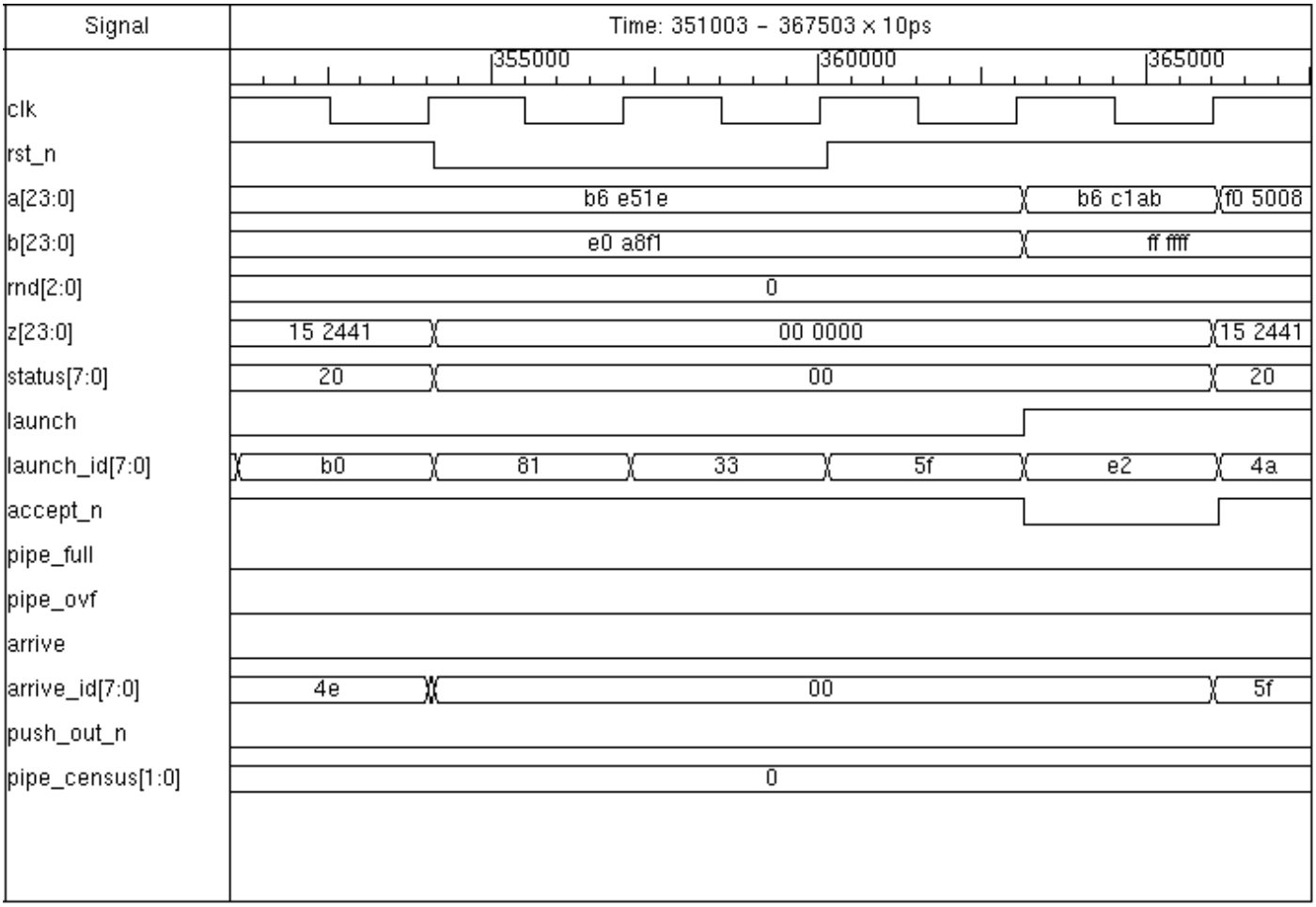
Figure 1-4 Synchronous Reset Timing



Asynchronous Reset

Figure 1-5 illustrates the action of reset when *reset_mode* = 0. In this case, the reset is asynchronous to the clock, and the outputs and internal registers clear upon the falling edge of the `rst_n` signal.

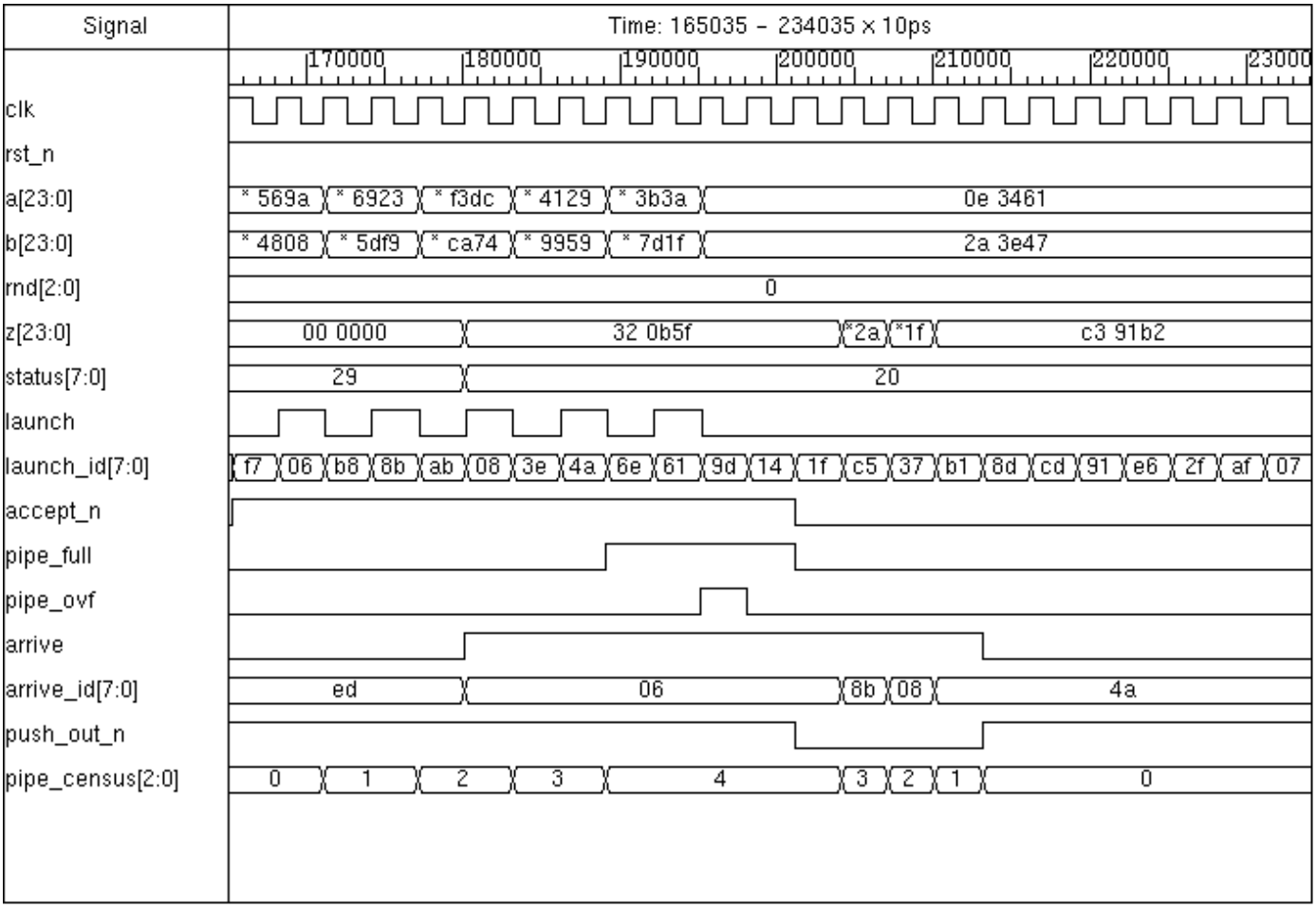
Figure 1-5 Asynchronous Reset Timing



Push Until Pipe Overflow

Figure 1-6 shows the result of push until the pipeline is full, and then, pushing once more to create a pipeline overflow. The `pipe_full` output asserts, and, at the next `launch`, the `pipe_ovf` asserts.

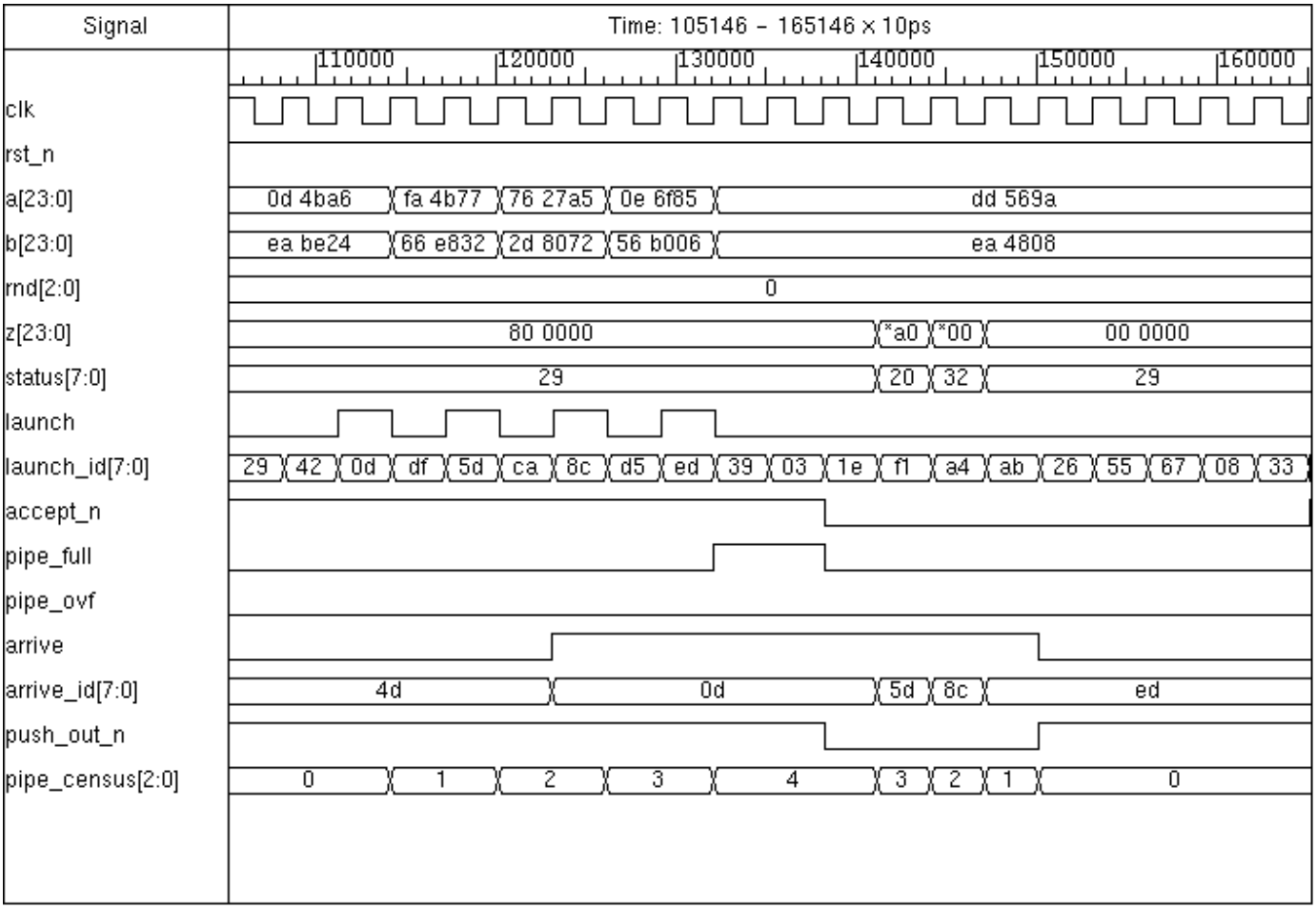
Figure 1-6 Push Until Pipe Overflow Timing



Push until Full, Pop until Empty

Figure 1-7 shows the result of push until the pipeline is full, and then, pop until empty. The `pipe_full` output asserts, and then with the assertion (active low) of `accept_n`, the pipeline empties.

Figure 1-7 Push Until Full, Pop until Empty Timing



Enabling minPower

In Design Compiler (version P-2019.03 and later) and Fusion Compiler, you can instantiate this component and use all its features without special settings.

For versions of Design Compiler before P-2019.03, enable minPower as follows:

```
set synthetic_library {dw_foundation.sldb dw_minpower.sldb}
set link_library {* $target_library $synthetic_library}
```

Related Topics

- Datapath Floating-Point Overview
- DesignWare Building Blocks User Guide

HDL Usage Through Component Instantiation - VHDL

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

entity DW_lp_piped_fp_div_inst is
      generic (
        inst_sig_width : POSITIVE := 23;
        inst_exp_width : POSITIVE := 8;
        inst_ieee_compliance : INTEGER := 0;
        inst_faithful_round : INTEGER := 0;
        inst_op_iso_mode : NATURAL := 0;
        inst_id_width : POSITIVE := 8;
        inst_in_reg : NATURAL := 0;
        inst_stages : POSITIVE := 4;
        inst_out_reg : NATURAL := 0;
        inst_no_pm : NATURAL := 1;
        inst_rst_mode : NATURAL := 0
        );
      port (
        inst_clk : in std_logic;
        inst_rst_n : in std_logic;
        inst_a : in std_logic_vector(inst_sig_width+inst_exp_width downto 0);
        inst_b : in std_logic_vector(inst_sig_width+inst_exp_width downto 0);
        inst_rnd : in std_logic_vector(2 downto 0);
        z_inst : out std_logic_vector(inst_sig_width+inst_exp_width downto 0);
        status_inst : out std_logic_vector(7 downto 0);
        inst_launch : in std_logic;
        inst_launch_id : in std_logic_vector(inst_id_width-1 downto 0);
        pipe_full_inst : out std_logic;
        pipe_ovf_inst : out std_logic;
        inst_accept_n : in std_logic;
        arrive_inst : out std_logic;
        arrive_id_inst : out std_logic_vector(inst_id_width-1 downto 0);
        push_out_n_inst : out std_logic;
        pipe_census_inst : out
std_logic_vector(bit_width(maximum(1,inst_in_reg+(inst_stages-1)+inst_out_reg)+1)-1
downto 0)
        );
    end DW_lp_piped_fp_div_inst;


architecture inst of DW_lp_piped_fp_div_inst is

    component DW_lp_piped_fp_div
      generic (
        sig_width : POSITIVE := 23;
        exp_width : POSITIVE := 8;
```

```
          ieee_compliance : INTEGER := 0;
          faithful_round : INTEGER := 0;
          op_iso_mode : NATURAL := 0;
          id_width : POSITIVE := 8;
          in_reg : NATURAL := 0;
          stages : POSITIVE := 4;
          out_reg : NATURAL := 0;
          no_pm : NATURAL := 0;
          rst_mode : NATURAL := 0
          );
        port (
          clk : in std_logic;
          rst_n : in std_logic;
          a : in std_logic_vector(sig_width+exp_width downto 0);
          b : in std_logic_vector(sig_width+exp_width downto 0);
          rnd : in std_logic_vector(2 downto 0);
          z : out std_logic_vector(sig_width+exp_width downto 0);
          status : out std_logic_vector(7 downto 0);
          launch : in std_logic;
          launch_id : in std_logic_vector(id_width-1 downto 0);
          pipe_full : out std_logic;
          pipe_ovf : out std_logic;
          accept_n : in std_logic;
          arrive : out std_logic;
          arrive_id : out std_logic_vector(id_width-1 downto 0);
          push_out_n : out std_logic;
          pipe_census : out std_logic_vector(bit_width(maximum(1,in_reg+(stages-
1)+out_reg)+1)-1 downto 0)
          );
      end component;

  begin

      -- Instance of DW_lp_piped_fp_div
      U1 : DW_lp_piped_fp_div
      generic map ( sig_width => inst_sig_width,
                    exp_width => inst_exp_width,
                    ieee_compliance => inst_ieee_compliance,
                    faithful_round => inst_faithful_round,
                    op_iso_mode => inst_op_iso_mode,
                    id_width => inst_id_width,
                    in_reg => inst_in_reg,
                    stages => inst_stages,
                    out_reg => inst_out_reg,
                    no_pm => inst_no_pm,
                    rst_mode => inst_rst_mode )
      port map ( clk => inst_clk,
                 rst_n => inst_rst_n,
                 a => inst_a,
```

```
                    b => inst_b,
                    rnd => inst_rnd,
                    z => z_inst,
                    status => status_inst,
                    launch => inst_launch,
                    launch_id => inst_launch_id,
                    pipe_full => pipe_full_inst,
                    pipe_ovf => pipe_ovf_inst,
                    accept_n => inst_accept_n,
                    arrive => arrive_inst,
                    arrive_id => arrive_id_inst,
                    push_out_n => push_out_n_inst,
                    pipe_census => pipe_census_inst );


  end inst;
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_lp_piped_fp_div_inst( inst_clk, inst_rst_n, inst_a, inst_b, inst_rnd,
          z_inst, status_inst, inst_launch, inst_launch_id, pipe_full_inst,
          pipe_ovf_inst, inst_accept_n, arrive_inst, arrive_id_inst, push_out_n_inst,
          pipe_census_inst );

parameter sig_width = 23;
parameter exp_width = 8;
parameter ieee_compliance = 0;
parameter faithful_round = 0;
parameter op_iso_mode = 0;
parameter id_width = 8;
parameter in_reg = 0;
parameter stages = 4;
parameter out_reg = 0;
parameter no_pm = 1;
parameter rst_mode = 0;


`define t1 4
`define bit_width_MX_1__in_reg_P_stages_M_1_P_out_reg_P_1 2

input inst_clk;
input inst_rst_n;
input [sig_width+exp_width : 0] inst_a;
input [sig_width+exp_width : 0] inst_b;
input [2 : 0] inst_rnd;
output [sig_width+exp_width : 0] z_inst;
output [7 : 0] status_inst;
input inst_launch;
input [id_width-1 : 0] inst_launch_id;
output pipe_full_inst;
output pipe_ovf_inst;
input inst_accept_n;
output arrive_inst;
output [id_width-1 : 0] arrive_id_inst;
output push_out_n_inst;
output [(`bit_width_MX_1__in_reg_P_stages_M_1_P_out_reg_P_1)-1 : 0] pipe_census_inst;

    // Instance of DW_lp_piped_fp_div
    DW_lp_piped_fp_div #(sig_width,
                         exp_width,
                          ieee_compliance,
                         faithful_round,
                         op_iso_mode,
                         id_width,
                         in_reg,
                         stages,
```

```
                        out_reg,
                        no_pm,
                        rst_mode)
      U1 ( .clk(inst_clk),
                .rst_n(inst_rst_n),
                .a(inst_a),
                .b(inst_b),
                .rnd(inst_rnd),
                .z(z_inst),
                .status(status_inst),
                .launch(inst_launch),
                .launch_id(inst_launch_id),
                .pipe_full(pipe_full_inst),
                .pipe_ovf(pipe_ovf_inst),
                .accept_n(inst_accept_n),
                .arrive(arrive_inst),
                .arrive_id(arrive_id_inst),
                .push_out_n(push_out_n_inst),
                .pipe_census(pipe_census_inst) );

endmodule
```

Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For lists of both known and fixed issues for this component, refer to the STAR report.

For a version of this datasheet with visible change bars, click here.

Date	Release	Updates
July 2020	DWBB_201912.5	■ Adjusted the description of the *ieee_compliance* parameter in Table 1-2 on page 2 ■ Adjusted content and title of "Suppressing Warning Messages During Verilog Simulation" on page 9 and added the DW_SUPPRESS_WARN macro and the illegal round mode message
April 2020	DWBB_201912.3	■ Updated the description of `rnd` in Table 1-1 on page 2 and *faithful_round* in Table 1-2 on page 2 ■ For STAR 3124396, added a footnote to Table 1-2 on page 2 to update the error range when *faithful_round* = 1 and *sig_width* > 23. This update is based on a limitation found during the investigation of STAR 3124396.
October 2019	DWBB_201903.5	■ Added "Disabling Clock Monitor Messages"
March 2019	DWBB_201903.0	■ Clarified the op_iso_mode parameter in Table 1-2 on page 2 ■ Clarified licensing requirements in Table 1-3 on page 4 ■ Added Figure 1-2 on page 6 to clarify datapath gating ■ Added "Enabling minPower" on page 13 ■ Added this Revision History table and the document links on this page
July 2017	DWBB_201612.5	■ For STAR 9001178646, added the ports `accept_n` and `push_out_n` in Table 1-1 on page 2, and clarified the width of port `pipe_census`. ■ Clarified heading titles, organization, and language for pipelining behavior ■ Added this Revision History table

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.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

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.
www.synopsys.com