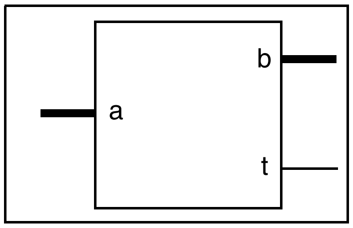
DW_inv_sqrt

Reciprocal of Square-Root

Version, STAR and Download Information: IP Directory

Features and Benefits

- Parameterized word length
- Combinational implementation to maximize speed
- Easy to pipeline for increased throughput



Description

The DW_inv_sqrt component provides a fixed-precision implementation of the reciprocal square-root of the input value, or $b = (\sqrt{a})^{-1}$.

Input a is a positive fraction in the following fixed-point format $a = (0.a_1a_2a_3a_4a_5...a_{a\_width})$, where a_i represents a bit. In order to have a valid output the input must be in the range [1/4, 1], which means that at least one bit in the pair of bits (a_1, a_2) must have a value 1. When the input is outside these bounds, the output is unpredictable.

This component expects to receive as input only the fractional bits of the input a. The MS 0 bit used in the format above was included to indicate that input a represents a value smaller than 1.

With the given input range, the output values satisfy the relation $1 < b < 2$ and are delivered in the format $b = (1.b_1b_2b_3b_4b_5...b_{a\_width-1})$. The output contains all the bits indicated in this vector (including the fixed 1 bit at the MSB position).

An important property of this component is that all bits provided at the output are correct. This property is important to simplify rounding operations. In particular, it is consistent with the requirement for the rounding module provided in the DesignWare Library (see DW_norm_rnd). Based on this observation, the output has an error $0 < e < 2^{a\_width-1}$. The error is always positive and less than the weight of the unit in the least position (ulp).

A *prec_control* parameter enables the designer to reduce or increase the internal precision used in the component. When *prec_control* = 0, all the bits required for precise computation are used. This means there are no bits discarded in intermediate steps. A *prec_control* = *n* > 0 instructs the IP block to remove *n* LS bits of some internal variables in order to save area. Of course, when this is done, errors are introduced, and the component may not satisfy the conditions for exact rounding. This parameter provides a trade-off between accuracy and cost.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
a	*a_width* bit(s)	Input	Input fractional bits only

Table 1-1 Pin Description (Continued)

Pin Name	Width	Direction	Function
b	*a_width* bit(s)	Output	Output data
t	1 bit	Output	sticky bit

Table 1-2 Parameter Description

Parameter	Values	Description
a_width	≥ 2	Word length of *a* and *b*
prec_control[a]	$\leq$ (*a_width* - 2) / 2	Controls the number of LS bits that may be removed or added to the internal precision, where 0 implies to keep all the bits.

a. Starting in the Z2007.03 release, the DW_inv_sqrt component has a tighter upper bound for the *prec_control* parameter (from *a_width*-2 to (*a_width*-2)/2). If you are using this parameter on an earlier released version of this component, there are two possible consequences when the most recent version is used: (1) the component will not elaborate for the *prec_control* value previously used if the value is now out of range, and (2) the component will not have the same numerical behavior as before for the same *prec_control* value.
As a rule of thumb to get the same numerical behavior: a previous value x should be mapped to a value y=floor(x-(*a_width*-2)/2), if the mapped value y is positive, or 0 otherwise.

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
rtl	Implement using the Datapath Generator technology combined with static DesignWare components.	DesignWare

Table 1-4 Simulation Models

Model	Function
DW01.DW_INV_SQRT_SIM	Design unit name for VHDL simulation
dw/dw01/src/DW_inv_sqrt_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_inv_sqrt.v	Verilog simulation model source code

The output *t* (sticky bit) is 1 when there are non-zero bits following the LS bit of the b output. This information is useful for rounding.

A numerical example of the component behavior is given as follows, for *a_width* = *10* bits:

$a = 0.0110111001 = 0.4306640625$

$b = 1.100001100 = 1.5234375$

Using infinite precision, the result would be

b_∞ = ***1.100001100****0011000011000011000…* where the bolded digits represent the digits generated by the component. All of them match the infinite precision result. The output of the DW_inv_sqrt is always less than the infinite precision value, and its error is computed in this example as:

$$Error = b_\infty - b \approx 0.000372 < ulp = 2^{-9} \approx 0.001953$$

This component provides a very fast implementation of the inverse square-root function. When exact rounding is not a requirement, you can use the *prec_control* parameter to obtain design tradeoffs between area/delay and error. Increasing *prec_control* reduces the area and delay in the design.

Another alternative, when larger errors are allowed, is to use a combination of the DW_div and DW_sqrt components. This is a VHDL example:

```
architecture small of inv_sqrt is
       signal square_root : std_logic_vector(a_width-1 downto 0);
       signal extended_input : std_logic_vector(2*a_width-1 downto 0);
       signal numerator : std_logic_vector(2*a_width-1 downto 0);
       signal zero_vector1 : std_logic_vector(a_width-1 downto 0);
       signal zero_vector2 : std_logic_vector(2*a_width-2 downto 0);
       signal b_int : std_logic_vector(2*a_width-1 downto 0);
begin
-- initialize the vectors
       zero_vector1 <= (others => '0');
       zero_vector2 <= (others => '0');
-- extend the input since the square root is done for integers and
-- the output has half of the input precision
       extended_input <= a & zero_vector1;
-- compute the square root of the extended input
square_root <= _std_logic_vector_(DWF_sqrt (unsigned(extended_input)));
-- create the numerator of the integer division to compute inverse value
       numerator <= '1' & zero_vector2;
-- perform the division
       b_int <= unsigned(numerator) / unsigned(square_root);
-- throw away the MS zeros
       b <= b_int (a_width-1 downto 0);
end _archname;
```

This is a Verilog example:

```
module DW_inv_sqrt (a, b);
parameter a_width = 8;
input [a_width-1 : 0] a;
output [a_width-1 : 0] b;
reg [a_width-1 : 0] square_root;
reg [2*a_width-1 : 0] extended_input;
reg [2*a_width-1 : 0] numerator;
reg [a_width-1 : 0] zero_vector1;
reg [2*a_width-2 : 0] zero_vector2;
reg [2*a_width-1 : 0] b_int;
parameter width = 2*a_width;
`include "DW_sqrt_function.inc"
```

```
always @(a)
begin
// initialize the vectors
 zero_vector1 = 0;
 zero_vector2 = 0;
// create the numerator of the integer division to compute inverse value
 numerator = {1'b1, zero_vector2};
// extend the input since the square root is done for integers and
// the output has half of the input precision
 extended_input = {a, zero_vector1};
// compute the square root of the extended input
 square_root = DWF_sqrt_uns (extended_input);
// perform the division
 b_int = numerator / square_root;
end
// throw away the MS zeros
 assign b = b_int[a_width-1 : 0];

endmodule
```

This solution provides a small implementation of the inverse square-root function. However, it is not functionally equivalent to DW_inv_sqrt when *prec_control* = 0.

Alternative Implementation of Reciprocal Square Root with DW_lp_multifunc

The reciprocal square root operation can also be implemented by DW_lp_multifunc component (a member of the minPower Library, licensed separately), which evaluates the value of reciprocal square root with 1 ulp error bound. There will be 1 ulp difference between the value from DW_lp_multifunc and the value from DW_invsqrt. Performance and area of the synthesis results are different between the DW_invsqrt and reciprocal square root implementation of the DW_lp_multifunc, depending on synthesis constraints, library cells and synthesis environments. By comparing performance and area between the reciprocal square root implementation of DW_lp_multifunc and DW_invsqrt component, the DW_lp_multifunc provides more choices for the better synthesis results. Below is an example of the Verilog description for the reciprocal square root of the DW_lp_multifunc. For more detailed information, see the DW_lp_multifunc datasheet.

```
DW_lp_multifunc #(op_width, 4) U1 (
    .A(A),
    .FUNC(16'h0004),
    .Z(Z),
    .STATUS(STATUS)
);
```

Related Topics

- Logic – Combinational 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_inv_sqrt_inst is
      generic (
        inst_a_width : POSITIVE := 8
        );
      port (
        inst_a : in std_logic_vector(inst_a_width-1 downto 0);
        b_inst : out std_logic_vector(inst_a_width-1 downto 0);
        t_inst : out std_logic
        );
    end DW_inv_sqrt_inst;


architecture inst of DW_inv_sqrt_inst is

begin

    -- Instance of DW_inv_sqrt
    U1 : DW_inv_sqrt
    generic map ( a_width => inst_a_width )
    port map ( a => inst_a, b => b_inst, t => t_inst );


end inst;
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_inv_sqrt_inst( inst_a, b_inst, t_inst );

parameter a_width = 8;


input [a_width-1 : 0] inst_a;
output [a_width-1 : 0] b_inst;
output t_inst;

    // Instance of DW_inv_sqrt
    DW_inv_sqrt #(a_width)
      U1 ( .a(inst_a), .b(b_inst), .t(t_inst) );

endmodule
```