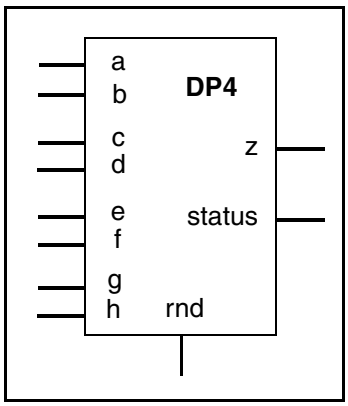
DW_fp_dp4

4-Term Floating-Point Dot-product

Version, STAR and Download Information: IP Directory

Features and Benefits



- The precision is controlled by parameters, and covers formats in the IEEE standard
- Exponents can range from 3 to 31 bits
- Fractional part of the floating-point number can range from 2 to 253 bits
- A parameter controls the use of denormal values
- More accurate than using a combination of basic FP operators.
- Provides a variety of rounding modes.
- DesignWare datapath generator is employed for reduced delay and area.

Description

DW_fp_dp4 is a floating-point component that computes the dot-product of eight floating-point inputs (`a`, `b`, `c`, `d`, `e`, `f`, `g` and `h`) to produce a floating-point result `z=a * b + c * d + e * f + g * h`, where the symbols "*" and "+" represent floating-point multiplication and floating-point addition, respectively. Therefore, it incorporates four FP multiplications and three FP additions. The accuracy of this component is better than the accuracy of an implementation using DW_fp_mult components and DW_fp_add components.

The input `rnd` defines a 3-bit rounding mode value (see Rounding Modes in the *Datapath Floating-Point Overview*) and the output `status` is an 8-bit status flags.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
a	*sig_width*+*exp_width*+1 bits	Input	FP input data
b	*sig_width*+*exp_width*+1 bits	Input	FP input data
c	*sig_width*+*exp_width*+1 bits	Input	FP input data
d	*sig_width*+*exp_width*+1 bits	Input	FP input data
e	*sig_width*+*exp_width*+1 bits	Input	FP input data
f	*sig_width*+*exp_width*+1 bits	Input	FP input data
g	*sig_width*+*exp_width*+1 bits	Input	FP input data
h	*sig_width*+*exp_width*+1 bits	Input	FP input data

Table 1-1 Pin Description (Continued)

Pin Name	Width	Direction	Function
rnd	3 bits	Input	Rounding mode
z	*sig_width*+*exp_width*+1 bits	Output	(a * b) + (c * d) + (e * f) + (g * h)
status	8 bits	Output	See STATUS Flags in the *Datapath Floating-Point Overview*

Table 1-2 Parameter Description

Parameter	Values	Description
sig_width	2 to 253 bits	Word length of fraction field of floating-point numbers a, b, c, d, e, f, g, h, and z
exp_width	3 to 31 bits	Word length of biased exponent of floating-point numbers a, b, c, d, e, f, g, h, and z
ieee_compliance	0 or 1	When 1, the generated architecture is fully compliant with IEEE 754 standard, including the use of denormals and NaNs.
arch_type	0 or 1	Controls the use of an alternative architecture. Default value is 0 (previous architecture).

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
rtl	Synthesis model	DesignWare

Table 1-4 Simulation Models

Model	Function
DW02.DW_FP_DP4_CFG_SIM	Design unit name for VHDL simulation
dw/dw02/src/DW_fp_dp4_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_fp_dp4.v	Verilog simulation model source code

Table 1-5 Functional Description

a	b	c	d	e	f	g	h	status[a]	z[b]
a (FP)	b (FP)	c (FP)	d (FP)	e (FP)	f (FP)	g (FP)	h (FP)	*	a*b+c*d+e*f+g*h (FP)

a. The details of status flags, if used, can be found in Table 9 of the *Datapath Floating-Point Overview*.
b. The actual value of the result is defined by the rounding mode.

The parameters *ieee_compliance* and *arch_type* control the functionality of this component. Different values of *arch_type* result in slightly different numeric behaviors, but in any case, the component is more accurate than the implementation of the same function using independent floating-point multipliers and adders.

When the parameter *arch_type* is set to 0 the component calculates the dot-product function as if the operation was done using infinite precision followed by a single rounding step at the end to generate the final result. The numerical behavior of this configuration is the most accurate as possible.

When *arch_type* is set to 1, a special architecture is used to reduce hardware and the component behaves more like a network of independent FP operators (network of multipliers and adders), without intermediate rounding. Only one rounding step is performed to compute the output value. In this case, the component produces logic that is not as accurate as when *arch_type* = 0, but it is still more accurate than the network of multipliers and adders. When *arch_type* = 1, the logic created by the component is smaller and faster than when *arch_type* = 0. The QoR of this configuration is competitive with the network of FP multipliers and adders for tight time constraint, with superior accuracy.

When the parameter *ieee_compliance* is set to 0 the component considers denormal values as zeros and NaN values as infinity. When the *ieee_compliance* parameter is set to 1 the component operates with denormals and NaNs as described in the IEEE Standard 754.

For more information about the floating-point system defined for all the DW_fp components, including `status` flag bits and floating-point formats, refer to the *Datapath Floating-Point Overview*.

Related Topics

- Datapath Floating-Point 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_arith.all;

entity DW_fp_dp4_inst is
      generic (
        inst_sig_width : POSITIVE := 23;
        inst_exp_width : POSITIVE := 8;
        inst_ieee_compliance : INTEGER := 0;
        inst_arch_type : INTEGER := 0
        );
      port (
        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_c : in std_logic_vector(inst_sig_width+inst_exp_width downto 0);
        inst_d : in std_logic_vector(inst_sig_width+inst_exp_width downto 0);
        inst_e : in std_logic_vector(inst_sig_width+inst_exp_width downto 0);
        inst_f : in std_logic_vector(inst_sig_width+inst_exp_width downto 0);
        inst_g : in std_logic_vector(inst_sig_width+inst_exp_width downto 0);
        inst_h : 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)
        );
    end DW_fp_dp4_inst;


architecture inst of DW_fp_dp4_inst is

begin

    -- Instance of DW_fp_dp4
    U1 : DW_fp_dp4
    generic map (
          sig_width => inst_sig_width,
          exp_width => inst_exp_width,
          ieee_compliance => inst_ieee_compliance,
          arch_type => inst_arch_type
          )
    port map (
          a => inst_a,
          b => inst_b,
          c => inst_c,
          d => inst_d,
          e => inst_e,
          f => inst_f,
          g => inst_g,
          h => inst_h,
```

```
            rnd => inst_rnd,
            z => z_inst,
            status => status_inst
            );


  end inst;

  -- pragma translate_off
   configuration DW_fp_dp4_inst_cfg_inst of DW_fp_dp4_inst is
   for inst
   end for; -- inst
   end DW_fp_dp4_inst_cfg_inst;
  -- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_fp_dp4_inst( inst_a, inst_b, inst_c, inst_d, inst_e,
          inst_f, inst_g, inst_h, inst_rnd, z_inst,
          status_inst );

parameter inst_sig_width = 23;
parameter inst_exp_width = 8;
parameter inst_ieee_compliance = 0;
parameter inst_arch_type = 0;


input [inst_sig_width+inst_exp_width : 0] inst_a;
input [inst_sig_width+inst_exp_width : 0] inst_b;
input [inst_sig_width+inst_exp_width : 0] inst_c;
input [inst_sig_width+inst_exp_width : 0] inst_d;
input [inst_sig_width+inst_exp_width : 0] inst_e;
input [inst_sig_width+inst_exp_width : 0] inst_f;
input [inst_sig_width+inst_exp_width : 0] inst_g;
input [inst_sig_width+inst_exp_width : 0] inst_h;
input [2 : 0] inst_rnd;
output [inst_sig_width+inst_exp_width : 0] z_inst;
output [7 : 0] status_inst;

    // Instance of DW_fp_dp4
    DW_fp_dp4 #(inst_sig_width, inst_exp_width, inst_ieee_compliance, inst_arch_type)
U1 (
                .a(inst_a),
                .b(inst_b),
                .c(inst_c),
                .d(inst_d),
                .e(inst_e),
                .f(inst_f),
                .g(inst_g),
                .h(inst_h),
                .rnd(inst_rnd),
                .z(z_inst),
                .status(status_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