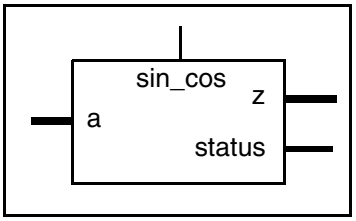
DW_fp_sincos

Floating-Point Sine and Cosine

Version, STAR and Download Information: IP Directory

Features and Benefits

- The precision format is parameterizable for either IEEE single, double precision, or a user-defined custom format.
- Hardware for denormal numbers of IEEE 754 standard is selectively provided.



Description

DW_fp_sincos is a floating-point sine and cosine unit that calculates z = sin(a), sin(πa), cos(a) or cos(πa) where a and z are floating-point values. Users can choose a function by controlling the input `sin_cos` or setting the parameter *pi_multiple*. There are two additional parameters that control the performance: *arch* is for the implementation selection between the area-optimized and the speed-optimized, and *err_range* allows users to choose the error range between 1 ulp error and 2 ulp error. DW_fp_sincos does not support rounding modes. The output `status` is an 8-bit optional status flag port.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
a	*exp_width* + *sig_width* + 1 bits	Input	Input data (radian)
sin_cos	1 bit	Input	Function select: 0 = sine , 1 = cosine
z	*exp_width* + *sig_width* + 1 bits	Output	Sine or cosine value
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 33 bits	Word length of fraction field of floating-point numbers `a` and `z`
exp_width	3 to 31 bits	Word length of biased exponent of floating-point numbers `a`, `z`
ieee_compliance	0 or 1 Default: 0	When 1, the generated architecture is fully compliant with IEEE 754 standard, including the use of denormals and NaNs.
pi_multiple	0 or 1 Default: 1	Angle is multiplied by π 0: `z` = sin(a) or cos(a) 1: `z` = sin(πa) or cos(πa)

Table 1-2 Parameter Description (Continued)

Parameter	Values	Description
arch	0 or 1 Default: 0	Selection of optimized implementation 0: Area-optimized implementation 1: Speed-optimized implementation
err_range	1 or 2 Default: 1	Error range selection when *pi_multiple* = 1 1: | true value - calculated | < 1 ulp 2: | true value - calculated | < 2 ulp This parameter is ignored when *pi_multiple* = 0 and the output is within 2 ulp range.

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_SINCOS_CFG_SIM	Design unit name for VHDL simulation
dw/dw02/src/DW_fp_sincos_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_fp_sincos.v	Verilog simulation model source code

As shown in Table 1-5, the operation of DW_fp_sincos is determined by the parameter *pi_multiple* and the input `sin_cos`. Since the sine and cosine are periodic functions with $\sin(2\pi k + \theta) = \sin(\theta)$ or $\cos(2\pi k + \theta) = \cos(\theta)$, where k is an integer, the floating-point sine/cosine operations cannot avoid the input precision error when the value of the floating-point input is larger than 2π.

Table 1-5 Operations of DW_fp_sincos

Parameter *pi_multiple*	Input `sin_cos`	Output `z`
0	0	sin(a)
	1	cos(a)
1	0	sin(πa)
	1	cos(πa)

That is, for example, if input `a` is a big number like 1.0×2^{50} for 32-bit floating-point number so that the value of the lsb bit of the significand is larger than 2π, then the input precision error becomes dominant because it is larger than the range of the angle θ. In this case, DW_fp_sincos(a) produces 0 when *pi_multiple* = 1 and `sin_cos` = 0, assuming $\theta \quad 0$. On the other hand, if *pi_multiple* = 0, `sin_cos` = 0, the

calculation of the angle θ from the input a where a = 2π*k* + θ brings the internal truncation error. In this case, the angle θ cannot be zero and DW_fp_sincos(a) generates a non-zero arbitrary number. However, if the input `a` is an infinite number, DW_fp_sincos produces a NaN. Table 1-6 shows examples of DW_fp_sincos operations.

Table 1-6 Result of special inputs when *sig_width* = 10, *exp_width* = 5, *ieee_compliance* = 0 and `sin_cos` = 0

pi_multiple	Input `a`	Output `z`
0	0000_0000_0000_0000 (Zero)	0000_0000_0000_0000
1		0000_0000_0000_0000
0	0111_1010_1010_1011 (Big Number)	0000_0000_0000_0000
1		0111_1010_0000_1110
0	0111_1100_0000_0000 (+ Infinity)	0111_1100_0000_0000
1		0111_1100_0000_0000
0	1111_1100_0000_0000 (- Infinity)	0111_1100_0000_0000
1		0111_1100_0000_0000

DW_fp_sincos provides the hardware for denormal numbers and NaNs of IEEE 754 standard. If the parameter *ieee_compliance* is turned off, denormal numbers are considered as zeros, and NaNs are considered as Infinity. Otherwise, denormal numbers and NaNs become effective and additional hardware to manipulate them is integrated.

Alternative Implementation of Floating-point Sine and Cosine with DW_lp_fp_multifunc

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

```
DW_lp_fp_multifunc #(sig_width, exp_width, ieee_compliance, 24, pi_multiple) U1 (
    .A(A),
    .FUNC(FUNC),
    .RND(3'h0),
    .Z(Z),
    .STATUS(STATUS)
);
```

For more information about the floating-point, including `status` flag bits, and integer 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.DWpackages.all;
use DWARE.DW_Foundation_comp_arith.all;

entity DW_fp_sincos_inst is
    generic (
      inst_sig_width : POSITIVE := 23;
      inst_exp_width : POSITIVE := 8;
      inst_ieee_compliance : INTEGER := 0;
      inst_pi_multiple : INTEGER := 1;
      inst_arch : INTEGER := 0;
      inst_err_range : INTEGER := 1
      );
    port (
      inst_a : in std_logic_vector(inst_sig_width+inst_exp_width downto 0);
      inst_sin_cos : in std_logic;
      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_sincos_inst;


architecture inst of DW_fp_sincos_inst is

begin

    -- Instance of DW_fp_sincos
    U1 : DW_fp_sincos
    generic map (
          sig_width => inst_sig_width,
          exp_width => inst_exp_width,
          ieee_compliance => inst_ieee_compliance,
          pi_multiple => inst_pi_multiple,
          arch => inst_arch,
          err_range => inst_err_range
          )
    port map (
          a => inst_a,
          sin_cos => inst_sin_cos,
          z => z_inst,
          status => status_inst
          );


end inst;
```

```
-- pragma translate_off
configuration DW_fp_sincos_inst_cfg_inst of DW_fp_sincos_inst is
for inst
end for;
end DW_fp_sincos_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_fp_sincos_inst( inst_a, inst_sin_cos, z_inst, status_inst );

parameter sig_width = 23;
parameter exp_width = 8;
parameter ieee_compliance = 0;
parameter pi_multiple = 1;
parameter arch = 0;
parameter err_range = 1;

input [sig_width+exp_width : 0] inst_a;
input inst_sin_cos;
output [sig_width+exp_width : 0] z_inst;
output [7 : 0] status_inst;

    // Instance of DW_fp_sincos
    DW_fp_sincos #(sig_width, exp_width, ieee_compliance, pi_multiple, arch, err_range)
      U1 ( .a(inst_a), .sin_cos(inst_sin_cos), .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