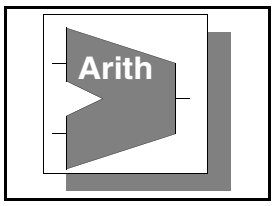
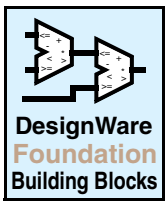


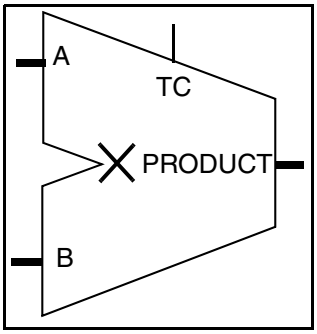


DW02_mult

Multiplier

Version, STAR and Download Information: IP Directory

Features and Benefits



- Parameterized word length
- Unsigned and signed (two's-complement) data operation

Description

DW02_mult is a multiplier that multiplies the operand A by B to produce the output, PRODUCT.

The control signal TC determines whether the input and output data is interpreted as unsigned (TC= 0) or signed (TC=1) numbers.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
A	*A_width* bit(s)	Input	Multiplier
B	*B_width* bit(s)	Input	Multiplicand
TC	1 bit	Input	Two's complement control 0 = unsigned 1 = signed
PRODUCT	*A_width* + *B_width* bit(s)	Output	Product A × B

Table 1-2 Parameter Description

Parameter	Values	Description
A_width	≥ 1	Word length of A
B_width	≥ 1	Word length of B

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
csa[a]	Carry-save array synthesis model	none
pparch[b]	Delay-optimized flexible Booth Wallace	DesignWare
apparch[b]	Area-optimized flexible architecture that can be optimized for area, for speed, or for area, speed	DesignWare

a. The ‘csa’ implementation does not require a DesignWare license and is therefore the only implementation available to DesignCompiler Expert with no DesignWare license. When a DesignWare license is available (which is required for Design Compiler Ultra), Design Compiler will not select the ‘csa’ implementation.

b. The ‘pparch’ (optimized for delay) and ‘apparch’ (optimized for area) implementations are dynamically generated to best meet your constraints. The ‘pparch’ and ‘apparch’ implementations can generate a variety of multiplier architectures including Radix-2 non-Booth, Radix-4 non-Booth, Radix-4 Booth recoded and Radix-8 Booth recoded. Generation of ‘pparch’ and ‘apparch’ implementations automatically detects which input would be best suited to be considered the multiplier as opposed to the multiplicand (for Booth recoding). The ‘pparch’ and ‘apparch’ implementations are generated making use of any special arithmetic technology cells that are found to be available in your target technology library. The `dc_shell` command, `set_dp_smartgen_options`, can be used to force specific multiplier architectures. For more information on forcing generated arithmetic architectures, use 'man set_dp_smartgen_options' (in dc_shell) to get a listing of the command options.

Table 1-4 Simulation Models

Model	Function
DW02.DW02_MULT_CFG_SIM	Design unit name for VHDL simulation
dw/dw02/src/DW02_mult_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW02_mult.v	Verilog simulation model source code

Table 1-5 Functional Description

TC	A	B	PRODUCT
0	A (unsigned)	B (unsigned)	A × B (unsigned)
1	A (two’s complement)	B (two’s complement)	A × B (two’s complement)

Related Topics

- Math – Arithmetic Overview
- DesignWare Building Block IP Documentation Overview

HDL Usage Through Operator Inferencing - VHDL

```
library IEEE, DWARE;
use DWARE.DW_foundation_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity DW02_mult_oper is
  generic(wordlength1, wordlength2: integer := 8);
  port(in1     : in std_logic_vector(wordlength1-1 downto 0);
       in2     : in std_logic_vector(wordlength2-1 downto 0);
       control : in std_logic;
       mult    : out std_logic_vector(wordlength1+wordlength2-1 downto 0) );
end DW02_mult_oper;

architecture func of DW02_mult_oper is
  signal mult_sig : SIGNED(wordlength1+wordlength2-1 downto 0) ;
  signal mult_usig : UNSIGNED(wordlength1+wordlength2-1 downto 0) ;
begin
  mult_sig <= SIGNED(in1) * SIGNED(in2);
  mult_usig <= UNSIGNED(in1) * UNSIGNED(in2);

  process (mult_sig,mult_usig, control)
  begin
    if control = '1' then
      mult <= std_logic_vector(mult_sig);
    else
      mult <= std_logic_vector(mult_usig);
    end if;
  end process;
end func;
```

HDL Usage Through Operator Inferencing - Verilog

```
module DW02_mult_oper (in1, in2, control, product);
  parameter wordlength1 = 8, wordlength2 = 8;
  input [wordlength1-1:0] in1;
  input [wordlength2-1:0] in2;
  input control;

  output [wordlength1+wordlength2-1:0] product;
  wire signed [wordlength1+wordlength2-1:0] product_sig;
  wire [wordlength1+wordlength2-1:0] product_usig;

  assign product_sig = $signed(in1) * $signed(in2);
  assign product_usig = in1 * in2;
  assign product = (control == 1'b1) ? $unsigned(product_sig) : product_usig;
endmodule
```

HDL Usage Through Function Inferencing - Verilog

```
module DW02_mult_func (in1, in2, prod1, prod2);
  parameter func_A_width = 8, func_B_width = 8;

  // Pass the widths to the DWF_mult functions
  parameter A_width = func_A_width, B_width = func_B_width;

  // Please add search_path = search_path + {synopsys_root + "/dw/sim_ver"}
  // to your .synopsys_dc.setup file (for synthesis) and add
  // +incdir+$SYNOPSYS/dw/sim_ver+ to your verilog simulator command line
  // (for simulation).
  `include "DW02_mult_function.inc"

  input [func_A_width-1 : 0] in1;
  input [func_B_width-1 : 0] in2;
  output [func_A_width-1 : 0] prod1, prod2;

  assign prod1 =  DWF_mult_tc(in1, in2);
  assign prod2 =  DWF_mult_uns(in1, in2);

endmodule
```

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 DW02_mult_inst is
  generic ( inst_A_width : NATURAL := 8;
            inst_B_width : NATURAL := 8 );
  port ( inst_A  : in std_logic_vector(inst_A_width-1 downto 0);
         inst_B  : in std_logic_vector(inst_B_width-1 downto 0);
         inst_TC : in std_logic;
    PRODUCT_inst : out std_logic_vector(inst_A_width+inst_B_width-1 downto 0)
        );
end DW02_mult_inst;

architecture inst of DW02_mult_inst is
begin

  -- Instance of DW02_mult
  U1 : DW02_mult
    generic map ( A_width => inst_A_width, B_width => inst_B_width )
    port map ( A => inst_A, B => inst_B, TC => inst_TC,
               PRODUCT => PRODUCT_inst );
end inst;

-- pragma translate_off
configuration DW02_mult_inst_cfg_inst of DW02_mult_inst is
  for inst
  end for; -- inst
end DW02_mult_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW02_mult_inst( inst_A, inst_B, inst_TC, PRODUCT_inst );

  parameter A_width = 8;
  parameter B_width = 8;

  input [A_width-1 : 0] inst_A;
  input [B_width-1 : 0] inst_B;
  input inst_TC;
  output [A_width+B_width-1 : 0] PRODUCT_inst;

  // Instance of DW02_mult
  DW02_mult #(A_width, B_width)
    U1 ( .A(inst_A), .B(inst_B), .TC(inst_TC), .PRODUCT(PRODUCT_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