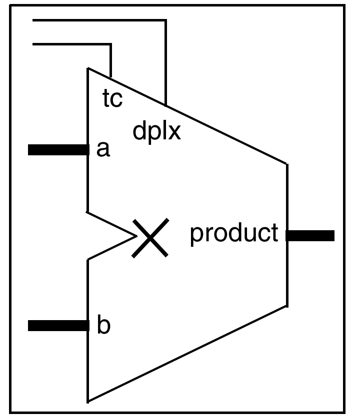
# DW_mult_dx

## Duplex Multiplier

Version, STAR and Download Information: IP Directory

## Features and Benefits




- Selectable single full-width multiplier (simplex) or two parallel smaller-width multiplier (duplex) operations
- Area and delay are similar to those of the DW02_mult Wallace architecture
- Selectable number system (unsigned or two's complement)
- Parameterized full word width
- Parameterized partial word width (allowing for asymmetric partial width operations)

## Description

DW_mult_dx performs multiplication of the operands `a` and `b` as either:

- A single product of *width* bits, or
- Two separated products (one of *p1_width* by *p1_width*, and one of *p2_width* by *p2_width*, where *p2_width* = *width* – *p1_width*).

**Table 1-1 Pin Description**

| Pin Name | Width | Direction | Function |
|---|---|---|---|
| a | *width* bit(s) | Input | Input data |
| b | *width* bit(s) | Input | Input data |
| tc | 1 bit | Input | Two's complement control |
| dplx | 1 bit | Input | Duplex mode select, active high |
| product | *width* × 2 bit(s) | Output | Product(s) |

**Table 1-2 Parameter Description**

| Parameter | Values | Description |
|---|---|---|
| width | $\geq 4$[a] | Word width of `a` and `b` |
| p1_width | 2 to *width* – 2[b] | Word width of Part1 of duplex multiplier |

a. Due to the limitation of memory addressing ranges of the computer operating system, there is an upper limit for parameter *width*. See "Memory Usage for Elaborating and Compiling DW_mult_dx" on page 4.
b. For the best performance of DW_mult_dx, *p1_width* should be set in the range [*width*/2, *width*–2]. For detailed information, see "Asymmetric Behavior of Parameter p1_width Setting for DW_mult_dx" on page 4.

**Table 1-3 Synthesis Implementations**

| Implementation Name | Function | License Feature Required |
|---|---|---|
| wall | Booth-recoded Wallace-tree synthesis model | DesignWare |

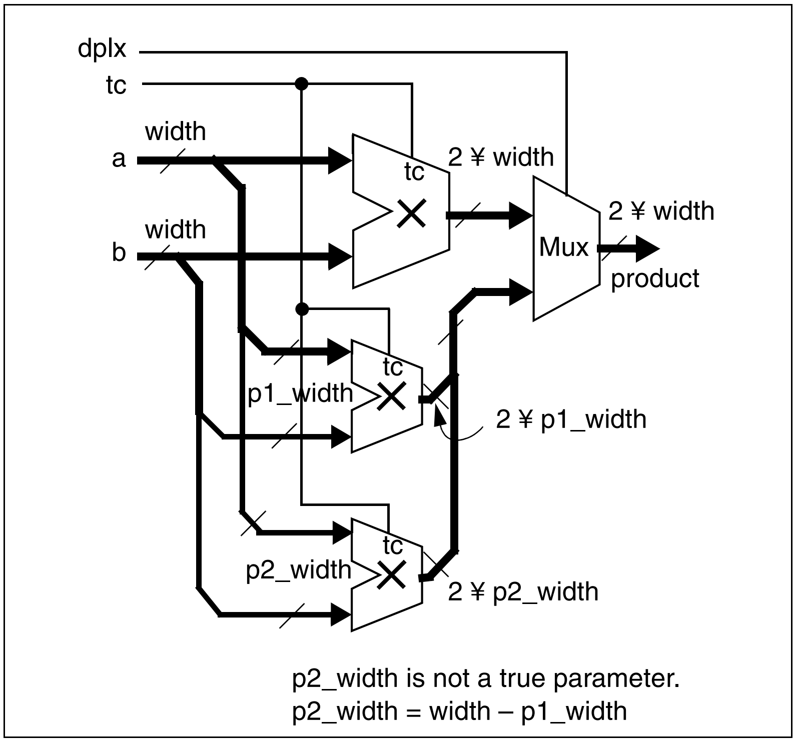
**Table 1-4 Simulation Models**

| Model | Function |
|---|---|
| DW02.DW_MULT_DX_CFG_SIM[a] | Design unit name for VHDL simulation |
| dw/dw02/src/DW_mult_dx_sim.vhd | VHDL simulation model source code |
| dw/sim_ver/DW_mult_dx.v | Verilog simulation model source code |

a. For reliable simulation in VHDL, always use a configuration in the design specifying the design unit name from the DesignWare Building Block IP (for example, DW02.DW_MULT_DX_CFG_SIM). Refer to "VHDL Simulation and Configuration" on page 5.

Figure 1-1 illustrates the behavior of DW_mult_dx.

**Figure 1-1 DW_mult_dx Functional Block Diagram**




However, Figure 1-1 differs from the actual synthesis implementation. In comparison to the structure shown in Figure 1-1, the synthesis model has significant area and delay savings.

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

The `dplx` input determines whether to perform a single full-width operation (`dplx` is 0) or two smaller-width operations (`dplx` is 1).

Table 1-5 shows the two separated products (one of *p1_width* by p1_width, and one of *p2_width* by *p2_width*, where *p2_width* = *width*–*p1_width*).

**Table 1-5 Operating Modes**

| dplx | tc | Function |
|---|---|---|
| 0 | 0 | Simplex unsigned multiply operation: product = a $\times$ b |
| 0 | 1 | Simplex signed multiply operation: product = a $\times$ b |
| 1 | 0 | Duplex unsigned multiply operation:[a]<br>p1_product = p1_a $\times$ p1_b<br>p2_product = p2_a $\times$ p2_b |
| 1 | 1 | Duplex signed multiply operation:<br>p1_product = p1_a $\times$ p1_b<br>p2_product = p2_a $\times$ p2_b |

a. For bit locations:
p1_a = a[p1_width–1:0], p2_a = a[width–1:p1_width]
p1_b = b[p1_width–1:0], p2_b = b[width–1:p1_width]
p1_product = product[2 $\times$ p1_width–1:0]
p2_product = product[2 $\times$ width–1:2 $\times$ p1_width]

# Application Note

## Asymmetric Behavior of Parameter p1_width Setting for DW_mult_dx

With a fixed parameter *width*, different values of *p1_width* may result in a different timing behavior of DW_mult_dx. Figure 1-2 shows the QOR data for different *p1_width* values, compiled with a 0.25 micron technology library using Design Compiler version 1998.08. This data changes with different technology libraries and Design Compiler versions.

Figure 1-2 on page 5 shows that DW_mult_dx is faster when *p1_width* is set in the range of [*width/2*, *width*–2] rather than [2, *width*/2], which allows you to improve circuit speed. For example, a 2 + 14 = 16 duplex multiplier can be set as *p1_width* = 14 (*p2_width* = 2) to get a delay of 4.06 ns, instead of setting *p1_width* = 2 to get a delay of 4.49 ns. This assumes that the circuit is compiled with the same technology library used in Figure 1-2.

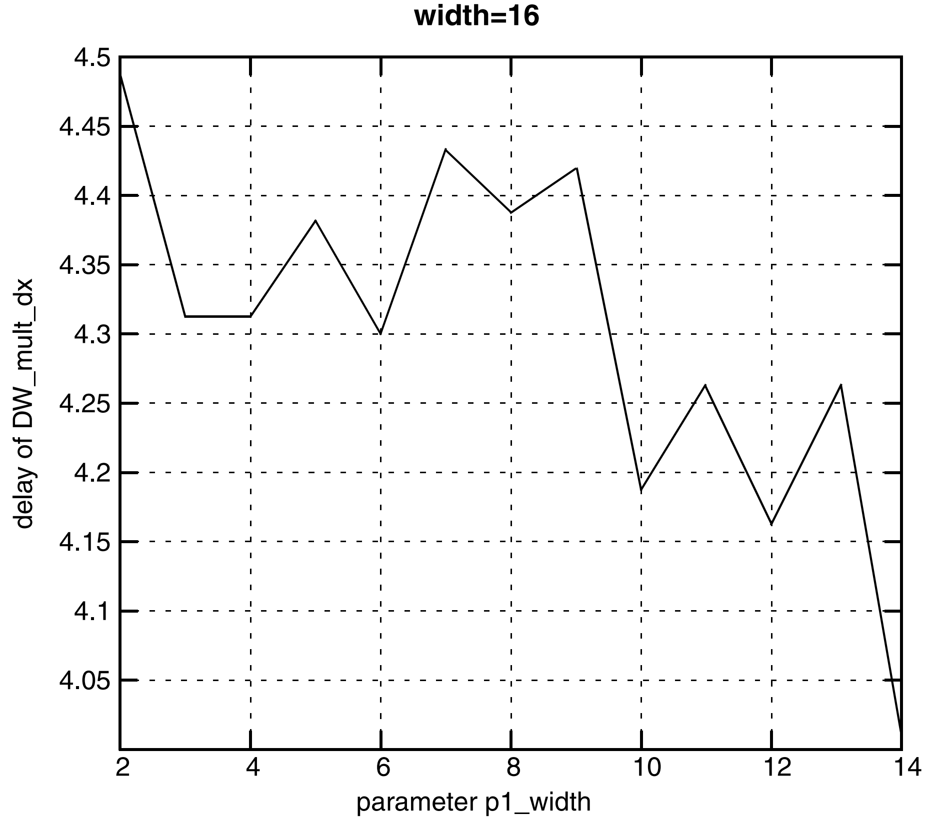
However, changing *p1_width* to (*width*–*p1_width*) may not be free. Connection switching circuits may be required to swap input and output data between [*p1_width*–1:0] and [*width*–1:*p1_width*], especially if the system also uses other duplex parts such as DW_addsub_dx or DW_cmp_dx.

You must find out whether the switching circuits cause area and delay increases that offset the benefit of using a larger *p1_width*. You must decide how to set a proper *p1_width* parameter, based on your design.

## Memory Usage for Elaborating and Compiling DW_mult_dx

Although the size and timing of gate-level DW_mult_dx are very close to those of DW02_mult, DW_mult_dx is more complex than DW02_mult, and requires more memory when elaborating and compiling. Figure 1-2 lists the CPU and memory usage when elaborating and compiling DW_mult_dx with different *width* and *p1_width*.

**Figure 1-2 DW_mult_dx p1_width vs. delay curve, compiled for speed**




### VHDL Simulation and Configuration

For reliable and consistent simulation in VHDL, users should always use a configuration in the design specifying the design unit name from the DesignWare Building Block IP (for example, DW02.DW_MULT_DX_CFG_SIM).

For better reusability and product quality, the simulation model of DW_mult_dx is coded hierarchically, and uses other DesignWare Foundation parts as sub-components. Therefore, you must have proper configuration for VHDL simulation.

For an example of configuration in VHDL simulation, refer to the VHDL instantiation code below.

## Related Topics

- Math – Arithmetic 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_mult_dx_inst is
  generic ( inst_width : NATURAL := 16;
            inst_p1_width : NATURAL := 8 );
  port ( inst_a    : in std_logic_vector(inst_width-1 downto 0);
         inst_b    : in std_logic_vector(inst_width-1 downto 0);
         inst_tc   : in std_logic;
         inst_dplx : in std_logic;
         product_inst : out std_logic_vector(2*inst_width-1 downto 0) );
end DW_mult_dx_inst;

architecture inst of DW_mult_dx_inst is
begin

  -- Instance of DW_mult_dx
  U1 : DW_mult_dx
  generic map ( width => inst_width,
                p1_width => inst_p1_width )
  port map ( a => inst_a,   b => inst_b,   tc => inst_tc,
             dplx => inst_dplx,   product => product_inst );
end inst;

-- pragma translate_off
configuration DW_mult_dx_inst_cfg_inst of DW_mult_dx_inst is
  for inst
  end for; -- inst
end DW_mult_dx_inst_cfg_inst;
-- pragma translate_on
```

## HDL Usage Through Component Instantiation - Verilog

```
module DW_mult_dx_inst(inst_a, inst_b, inst_tc, inst_dplx, product_inst);

  parameter width = 16;
  parameter p1_width = 8;

  input [width-1 : 0] inst_a;
  input [width-1 : 0] inst_b;
  input inst_tc;
  input inst_dplx;
  output [2*width-1 : 0] product_inst;

  // Instance of DW_mult_dx
  DW_mult_dx #(width, p1_width)
    U1 ( .a(inst_a),   .b(inst_b),   .tc(inst_tc),   .dplx(inst_dplx),
         .product(product_inst));

endmodule
```

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043

www.synopsys.com