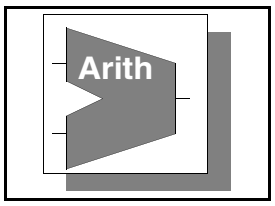
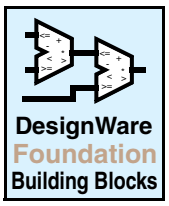




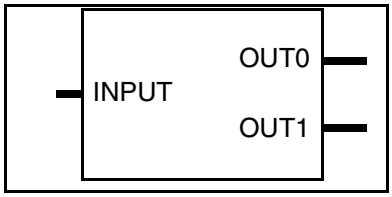
DW02_tree

Wallace Tree Compressor

Version, STAR and Download Information: IP Directory

Features and Benefits

- Parameterized word length



Applications

- Matrix arithmetic
- Digital filtering
- Vector addition
- Parameter control over carry-save (CS) design verification method

Description

DW02_tree is a Wallace-tree compressor. This component is used in building the Wallace-tree adder, DW02_sum. DW02_tree is included as a separate component for designing your own hierarchical summation blocks or Wallace-tree-based multipliers.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
INPUT	*num_inputs* ¥ *input_width* bit(s)	Input	Input vector
OUT0	*input_width* bit(s)	Output	Partial sum
OUT1	*input_width* bit(s)	Output	Partial sum

Table 1-2 Parameter Description

Parameter	Values	Description
num_inputs	≥ 1	Number of inputs
input_width	≥ 1	Word length of outputs `out0` and `out1`
verif_en[a]	0 or 1 Default: 1	Verification Enable Control 0: `out0` and `out1` are always the same for a given input value 1: `out0` and `out1` change with time for a given input value

a. Although the *verif_en* value can be set for all simulators, CS randomization is only implemented when using Synopsys simulators (VCS, VCS-MX). For more information about *verif_en*, refer to "Simulation Using Random Carry-save Representation (VCS/VCS-MX only)" on page 3

Table 1-3 Synthesis Implementations[a]

Implementation Name	Function	License Feature Required
pparch	Delay-optimized flexible parallel-prefix	DesignWare
apparch	Area-optimized flexible architecture that can be optimized for area, for speed, or for area, speed	DesignWare

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use any architectures described in this table. For more, see *DesignWare Building Block IP User Guide*

Table 1-4 Obsolete Synthesis Implementations[a]

Implementation Name	Function	License Feature Required
wallace	Wallace tree synthesis model	DesignWare

a. DC versions prior to 2007.03 will still include these implementations.

Table 1-5 Simulation Models

Model	Function
DW02.DW02_TREE_CFG_SIM	Design unit name for VHDL simulation
dw/dw02/src/DW02_tree_sim.vhd[a]	VHDL simulation model source code
dw/sim_ver/DW02_tree.v	Verilog simulation model source code

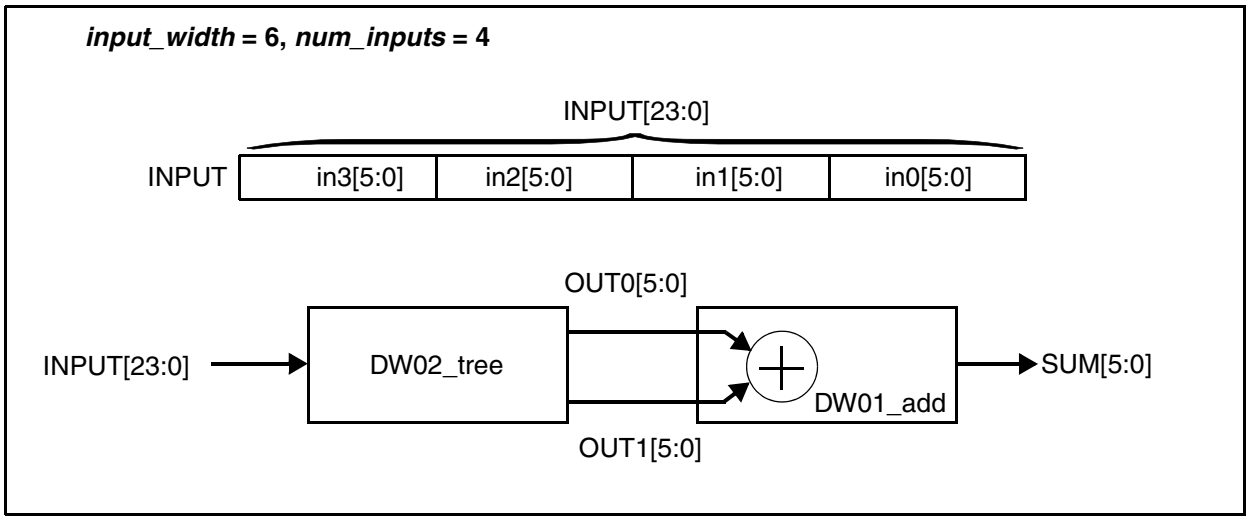
a. This is a plain-text simulation model file for use with 3rd-party VHDL simulators, and parenthetically does not support the *verif_en* control of CS random simulation.

Attention

The simulation architecture does not produce the same values on `OUT0` and `OUT1` as produced by the synthetic architecture (see Figure 1-1), but once added together via a component like the DW01_add, the resulting `SUM` will be the same for the synthetic and simulation architectures. More specifically, (out0+out1)mod 2^{input_width} is the same in both cases.

Functional Operation

Figure 1-1 Functional Operation



The compressor tree is built from compressor cells. The top of the tree accepts a vector of length *num_inputs* × *input_width*. The tree compresses this vector to form two outputs, `OUT0` and `OUT1`. These two outputs may be added together to form the final sum of the vector elements, or may in turn be fed into another Wallace tree compressor along with other sum terms to form a hierarchical summation tree.

It is important to note that the outputs, `OUT0` and `OUT1`, are not regular signed or unsigned numbers until they are summed together, but rather they represent an intermediate result in redundant carry-save representation. Therefore, you cannot perform sign extension on `OUT0` and `OUT1` directly, but must do this at the input or after final summation.

Simulation Using Random Carry-save Representation (VCS/VCS-MX only)

The carry-save representation for the output of DW02_tree is “redundant” and therefore there are many possible ways to represent the same output value. The simulation model of DW02_tree most likely does not match the behavior of the synthesized circuit, however, although they may deliver completely different outputs, they are numerically equivalent (the value (`out0+out1`) mod $2^{\text{input\_width}}$ is the same in both cases).

Instead of having only a “static” behavior, the simulation models of DW02_tree provide a parameter *verif_en* to let the user control the randomness in the CS representation of their outputs. This parameter applies only to simulation models. The term “static” is used here to denote when the CS representation delivered for a given set of input values is always the same, independent of time. Such a behavior is not ideal for verification of designs that manipulate the CS values produced by DW02_tree because the design that instantiates this component should work independently of any particular “static” implementation of DW02_tree. A “random” CS representation of the output is one that still represents the summation result but changes every time a set of input values is applied. The random behavior has a better chance of exposing issues in the use of CS representation, but it is not a full proof that the design works properly for any implementation of DW02_tree. It provides a better coverage than the static simulation model.

The parameter *verif_en* controls if the CS output behavior is static (*verif_en*=0) or random (*verif_en*=1).

Using this mechanism, the designer has a better simulation environment to discover design problems related to incorrect CS manipulation. These problems could be masked by a static behavior of the simulation model, and could manifest later after synthesis of DW02_tree. It is recommended to use the more aggressive simulation behavior (*verif_en*=1).

Table 1-6 shows the behavior of DW02_tree for a sequence of inputs, using the parameter values *num_inputs*=4 and *input_width*=8 (the table has only hexadecimal values). The sequence repeats the input set to demonstrate the component behavior. For the case *verif_en*=0, the output is always the same when the same input value is applied. When *verif_en*=1, the output values change for the same input values. Notice that sign-extension of the DW02_tree output should not be applied in any case.

Table 1-6 DW02_tree Behavior with *num_inputs*=4 and *input_width*=8

Four 8-bit inputs	(out0,out1) verif_en=0	(out0,out1) verif_en=1
F4, A4, 56, 0F	F8, 05	35, C8
12, 34, 54, 32	28, A4	E8, E4
F4, A4, 56, 0F	F8, 05	1B, E2
12, 34, 54, 32	28, A4	8A, 42
F4, A4, 56, 0F	F8, 05	E8, 15
12, 34, 54, 32	28, A4	3D, 8F
F4, A4, 56, 0F	F8, 05	6E, 8F
12, 34, 54, 32	28, A4	9A, 32
F4, A4, 56, 0F	F8, 05	AA, 53

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.DWpackages.all;
use DWARE.DW_foundation_comp.all;

entity DW02_tree_inst is
      generic (
        inst_num_inputs : POSITIVE := 8;
        inst_input_width : POSITIVE := 8;
        inst_verif_en : INTEGER := 1); -- level 1 is the most aggressive
                                       -- verification mode for simulation
      port (
        inst_INPUT : in std_logic_vector(inst_num_inputs*
                                         inst_input_width-1 downto 0);
        OUT0_inst : out std_logic_vector(inst_input_width-1 downto 0);
        OUT1_inst : out std_logic_vector(inst_input_width-1 downto 0));
    end DW02_tree_inst;


architecture inst of DW02_tree_inst is
begin
    -- Instance of DW02_tree
    U1 : DW02_tree
    generic map ( num_inputs => inst_num_inputs,
                  input_width => inst_input_width,
                  verif_en => inst_verif_en )
    port map ( INPUT => inst_INPUT, OUT0 => OUT0_inst, OUT1 => OUT1_inst );
end inst;

-- pragma translate_off
configuration DW02_tree_inst_cfg_inst of DW02_tree_inst is
for inst
end for; -- inst
end DW02_tree_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW02_tree_inst( inst_INPUT, OUT0_inst, OUT1_inst );

parameter num_inputs = 8;
parameter input_width = 8;
parameter verif_en = 1; // value 1 is a more aggressive verification
                        // mode

input [num_inputs*input_width-1 : 0] inst_INPUT;
output [input_width-1 : 0] OUT0_inst;
output [input_width-1 : 0] OUT1_inst;

    // Instance of DW02_tree
    DW02_tree #(num_inputs, input_width, verif_en)
      U1 ( .INPUT(inst_INPUT), .OUT0(OUT0_inst), .OUT1(OUT1_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