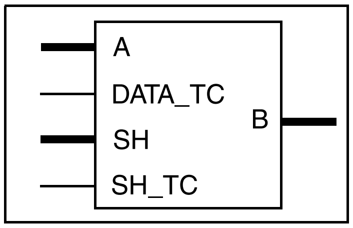
DW_rash

Arithmetic Shifter with Preferred Right Direction

Version, STAR and Download Information: IP Directory

Features and Benefits

- Parameterized word length
- Parameterized shift coefficient width
- Inferable using a function call



Description

DW_rash is a general-purpose arithmetic shifter similar to DW01_ash, but with preferred right direction for shifting. The input data A is shifted right or left by the number of bits specified by the control input SH.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
A	*A_width*	Input	Input data
DATA_TC	1 bit	Input	Data two's complement control 0 = unsigned 1 = signed
SH	*SH_width*	Input	Shift control
SH_TC	1 bit	Input	Shift two's complement control 0 = unsigned 1 = signed
B	*A_width*	Output	Output data.

Table 1-2 Parameter Description

Parameter	Values	Description
A_width	≥ 2	Word length of A and B
SH_width	≥ 1	Word length of SH

Table 1-3 Synthesis Implementations[a]

Implementation Name	Function	License Feature Required
str	Synthesis model target for speed	DesignWare
astr	Synthesis model target for area	DesignWare
mx2	Implement using 2:1 multiplexers only. The mx2 implementation is valid only for SH_width values up to and including 31.	none

a. During synthesis, Design Compiler selects the appropriate architecture for your constraints. However, you can force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

Table 1-4 Simulation Models

Model	Function
dw/dw01/src/DW_rash_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_rash.v	Verilog simulation model source code

When the control signal `SH_TC=0`, the coefficient `SH` is interpreted as an unsigned positive number and DW_rash performs only right shift operations.

When `SH_TC=1`, `SH` is interpreted as a two's complement number. A negative SH value performs a left shift and a positive SH value performs a right shift.

The input data A is interpreted as an unsigned number when DATA_TC=0 or a two's complement number when `DATA_TC=1`.

The type of A is only significant for right shift operations (SH is positive), in which case a zero padding is done on the most significant bits for unsigned data and sign extension is done for signed two's complement data.

Table 1-5 Truth Table (SH_width = 3, A_width = 8)

SH(2:0)	SH_TC	DATA_TC	B(7)	B(6)	B(5)	B(4)	B(3)	B(2)	B(1)	B(0)
000	X	0	A(7)	A(6)	A(5)	A(4)	A(3)	A(2)	A(1)	A(0)
001	X	0	0	A(7)	A(6)	A(5)	A(4)	A(3)	A(2)	A(1)
010	X	0	0	0	A(7)	A(6)	A(5)	A(4)	A(3)	A(2)
011	X	0	0	0	0	A(7)	A(6)	A(5)	A(4)	A(3)
100	0	0	0	0	0	0	A(7)	A(6)	A(5)	A(4)
101	0	0	0	0	0	0	0	A(7)	A(6)	A(5)
110	0	0	0	0	0	0	0	0	A(7)	A(6)
111	0	0	0	0	0	0	0	0	0	A(7)
000	X	1	A(7)	A(6)	A(5)	A(4)	A(3)	A(2)	A(1)	A(0)
001	X	1	A(7)	A(7)	A(6)	A(5)	A(4)	A(3)	A(2)	A(1)
010	X	1	A(7)	A(7)	A(7)	A(6)	A(5)	A(4)	A(3)	A(2)
011	X	1	A(7)	A(7)	A(7)	A(7)	A(6)	A(5)	A(4)	A(3)
100	0	1	A(7)	A(7)	A(7)	A(7)	A(7)	A(6)	A(5)	A(4)
101	0	1	A(7)	A(7)	A(7)	A(7)	A(7)	A(7)	A(6)	A(5)
110	0	1	A(7)	A(7)	A(7)	A(7)	A(7)	A(7)	A(7)	A(6)
111	0	1	A(7)	A(7)	A(7)	A(7)	A(7)	A(7)	A(7)	A(7)
100	1	X	A(3)	A(2)	A(1)	A(0)	0	0	0	0
101	1	X	A(4)	A(3)	A(2)	A(1)	A(0)	0	0	0
110	1	X	A(5)	A(4)	A(3)	A(2)	A(1)	A(0)	0	0
111	1	X	A(6)	A(5)	A(4)	A(3)	A(2)	A(1)	A(0)	0

Related Topics

- Logic – Combinational Overview
- DesignWare Building Block IP Documentation Overview

HDL Usage Through Function Inferencing - VHDL

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

entity DW_rash_func is
      generic (
        func_A_width : POSITIVE := 8;
        func_SH_width : POSITIVE := 3
        );
      port (
        func_A : in std_logic_vector(func_A_width-1 downto 0);
        func_DATA_TC : in std_logic;
        func_SH : in std_logic_vector(func_SH_width-1 downto 0);
        func_SH_TC : in std_logic;
        B_func : out std_logic_vector(func_A_width-1 downto 0)
        );
    end DW_rash_func;

architecture func of DW_rash_func is

begin

process (func_DATA_TC, func_SH_TC, func_A,func_SH)
begin

if func_DATA_TC = '0' and func_SH_TC = '0' then
    B_func <= std_logic_vector(DWF_rash(unsigned(func_A),unsigned(func_SH)));
elsif func_DATA_TC = '1' and func_SH_TC = '0' then
    B_func <= std_logic_vector(DWF_rash(signed(func_A),unsigned(func_SH)));
elsif func_DATA_TC = '1' and func_SH_TC = '1' then
    B_func <= std_logic_vector(DWF_rash(signed(func_A),signed(func_SH)));
else
    B_func <= std_logic_vector(DWF_rash(unsigned(func_A),signed(func_SH)));
end if;

end process;

end func;
```

HDL Usage Through Function Inferencing - Verilog

```
module DW_rash_func( func_A, func_DATA_TC, func_SH, func_SH_TC, B_func );

parameter func_A_width = 8;
parameter func_SH_width = 3;

// secondary parameters used to pass parameters to function
// when parameter names differ

parameter A_width = func_A_width;
parameter SH_width = func_SH_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 "DW_rash_function.inc"


input [func_A_width-1 : 0] func_A;
input func_DATA_TC;
input [func_SH_width-1 : 0] func_SH;
input func_SH_TC;
output [func_A_width-1 : 0] B_func;
reg    [func_A_width-1 : 0] B_func_i;

    // Infer DW_rash

   always @ (func_A or func_DATA_TC or func_SH or func_SH_TC)
     begin
     casex({func_DATA_TC,func_SH_TC}) // synopsys full_case
     2'b00: B_func_i = DWF_rash_uns_uns(func_A,func_SH);
     2'b10: B_func_i = DWF_rash_tc_uns(func_A,func_SH);
     2'b01: B_func_i = DWF_rash_uns_tc(func_A,func_SH);
     2'b11: B_func_i = DWF_rash_tc_tc(func_A,func_SH);
     endcase
     end

assign B_func = B_func_i;

endmodule
```

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 DW_rash_inst is
      generic (
        inst_A_width : POSITIVE := 8;
        inst_SH_width : POSITIVE := 3
        );
      port (
        inst_A : in std_logic_vector(inst_A_width-1 downto 0);
        inst_DATA_TC : in std_logic;
        inst_SH : in std_logic_vector(inst_SH_width-1 downto 0);
        inst_SH_TC : in std_logic;
        B_inst : out std_logic_vector(inst_A_width-1 downto 0)
        );
    end DW_rash_inst;


architecture inst of DW_rash_inst is

begin

    -- Instance of DW_rash
    U1 : DW_rash
    generic map (
          A_width => inst_A_width,
          SH_width => inst_SH_width
          )
    port map (
          A => inst_A,
          DATA_TC => inst_DATA_TC,
          SH => inst_SH,
          SH_TC => inst_SH_TC,
          B => B_inst
          );


end inst;
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_rash_inst( inst_A, inst_DATA_TC, inst_SH, inst_SH_TC, B_inst );

parameter A_width = 8;
parameter SH_width = 3;


input [A_width-1 : 0] inst_A;
input inst_DATA_TC;
input [SH_width-1 : 0] inst_SH;
input inst_SH_TC;
output [A_width-1 : 0] B_inst;

    // Instance of DW_rash
    DW_rash #(A_width, SH_width) U1 (
                .A(inst_A),
                .DATA_TC(inst_DATA_TC),
                .SH(inst_SH),
                .SH_TC(inst_SH_TC),
                .B(B_inst) );

endmodule
```

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

www.synopsys.com