DW_div
Combinational Divider

Features and Benefits

- Parameterized word lengths
- Unsigned and signed (two’s complement) data operation
- Remainder or modulus as second output
- Inferable using a function call

Description

DW_div is a combinational integer divider with both quotient and remainder outputs. This component divides the dividend \( a \) by the divisor \( b \) to produce the quotient and remainder. Optionally, the remainder output computes the modulus.

Table 1-1 Pin Description

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Width</th>
<th>Direction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>( a_width ) bit(s)</td>
<td>Input</td>
<td>Dividend</td>
</tr>
<tr>
<td>b</td>
<td>( b_width ) bit(s)</td>
<td>Input</td>
<td>Divisor</td>
</tr>
<tr>
<td>quotient</td>
<td>( a_width ) bit(s)</td>
<td>Output</td>
<td>Quotient</td>
</tr>
<tr>
<td>remainder</td>
<td>( b_width ) bit(s)</td>
<td>Output</td>
<td>Remainder / modulus</td>
</tr>
<tr>
<td>divide_by_0</td>
<td>1 bit</td>
<td>Output</td>
<td>Indicates if ( b ) equals 0</td>
</tr>
</tbody>
</table>

Table 1-2 Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_width</td>
<td>( \geq 1 )</td>
<td>Word length of ( a )</td>
</tr>
<tr>
<td></td>
<td>Default: None</td>
<td></td>
</tr>
<tr>
<td>b_width</td>
<td>( \geq 1 )</td>
<td>Word length of ( b )</td>
</tr>
<tr>
<td></td>
<td>Default: None</td>
<td></td>
</tr>
<tr>
<td>tc_mode</td>
<td>0 or 1</td>
<td>Two'- complement control.</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
<td>0 = unsigned</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = signed</td>
</tr>
<tr>
<td>rem_mode</td>
<td>0 or 1</td>
<td>Remainder output control.</td>
</tr>
<tr>
<td></td>
<td>Default: 1</td>
<td>0 = a mod ( b )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = a rem ( b ), a % ( b )</td>
</tr>
</tbody>
</table>
The parameter \( tc\_mode \) determines whether the data of the inputs \( a \) and \( b \) and the outputs \( quotient \) and \( remainder \) are interpreted as unsigned (\( tc\_mode=0 \)) or two’s-complement (\( tc\_mode=1 \)) numbers.

The parameter \( rem\_mode \) determines whether the output \( remainder \) corresponds to the division remainder (\( rem\_mode=1 \)) or to the modulus (\( rem\_mode=0 \)) as defined in the IEEE Standard 1076-1993 VHDL Language Reference Manual.

The modulus is different from the remainder only for two’s complement division in VHDL. There, the modulus (“mod”) has the sign of the divisor while the remainder (“rem”) has the sign of the dividend. In Verilog, the modulus (“\%”) is actually the division remainder, which has the sign of the dividend.

The \( quotient \) and \( remainder \) outputs are defined as follows:

\[
quotient = \text{int} \left( \frac{a}{b} \right)
\]

The table shows the synthesis implementations and simulation models for the combinational divider.
For \( \text{rem\_mode} = 1 \) (remainder):
\[
\text{remainder} = a \mod b = a - \text{int}\,(a/b) \times b
\]

If \( a \) and \( b \) have the same sign for \( \text{rem\_mode} = 0 \) (modulus), then the following is true:
\[
\text{remainder} = a \mod b = a - \text{int}\,(a/b) \times b
\]

Otherwise, the following is true:
\[
\text{remainder} = a \mod b = a + \text{ceil}(|a/b|) \times b
\]

**Example**

\[
\begin{align*}
7 \mod 4 &= 3 \\
7 \mod -4 &= -1 \\
-7 \mod 4 &= 1 \\
-7 \mod -4 &= -3
\end{align*}
\]

**Out-of-Range Results**

In the case of dividing by zero, the \text{divide\_by\_0} output is set to 1 and the \text{quotient} is saturated to the maximum positive value if \( a \) is positive and to the minimum negative value if \( a \) is negative. The \text{remainder} is equal to \( a \).

**Example (two's complement):**

```
"0010" / "0000" = "0111"
"0010" rem "0000" = "0010"
"1110" / "0000" = "1000"
"1110" rem "0000" = "1110"
```

In the case of dividing the minimum negative value by -1, the quotient would be one larger than the maximum representable positive value (overflow). The \text{quotient} is then set to the minimum negative value (corresponds to correct value in unsigned representation) and the \text{remainder} is set to 0. Note that this is only case in which division of a negative number by a negative number does not result in a positive number.

**Example (two's complement):**

```
"1000" / "1111" = "1000"
"1000" rem "1111" = "0000"
```

**Application Examples**

**Fixed-Point Arithmetic**

For operands that are in integer.fraction format, the fraction width of the quotient is the fraction width of the dividend minus the fraction width of the divisor.
Example

\[110.01011 / 00100.101 = 000001.01\]

**Rounding**

\(\text{DW}_{\text{div}}\) truncates fractions to produce an integer quotient. To round to the nearest integer value, concatenate a fraction bit of zero to the least significant end of the dividend. The resulting quotient has a one-bit fraction. Add one-half to round the quotient to the nearest integer.

**VHDL Example**

```vhdl
signal quot_temp : unsigned(width downto 0);
quot_temp <= (a & '0') / b + 1;
quot <= quot_temp(width downto 1);
```

**Verilog Example**

```verilog
Wire frac;
assign {quot, frac} = DWF_div_uns ({a, 1'b0}, b) + 1;
```

**Reciprocal**

To compute the reciprocal of a number, simply divide \(1.00...0\) (represented by a one and \(b_{width-1}\) zeroes) by the number.

**Division by a Normalized Number**

When the divisor is normalized (its MSB is always one in unsigned numbers), \(\text{DW}_{\text{div}}\) will be better optimized if the MSB of the divisor is explicitly connected to a one (it is hard wired at the divider input) instead of being implicitly always one (it comes out of logic).

In two’s complement normalized numbers, the second-most significant bit is always the inverse of the sign bit (MSB). There, best optimization results for \(\text{DW}_{\text{div}}\) are achieved by explicitly connecting the inverted sign bit to the second-most significant bit of the divisor.

**VHDL Example**

```vhdl
quot_uns <= a / ('1' & b(width-2 downto 0);
quot_tc <= a / (b(width-1) & not b(width-1) & b(width-3 downto 0));
```

**Verilog Example**

```verilog
assign quot_uns = DWF_div_uns (a, {1'b1, b[width-2 : 0]});
assign quot_tc = DWF_div_tc a, {b[width-1], ~b[width-1], b[width-3 : 0]});
```

**Unsigned-Signed Divider**

\(\text{DW}_{\text{div}}\) can only perform either unsigned or signed division (specified by parameter \(\text{tc_mode}\)). If a divider is required that can do both (selectable by a \(\text{tc}\) pin), extend both inputs by one bit (zero-extend for unsigned
numbers and sign-extend for two’s complement numbers depending on the tc pin) and perform a signed division.

**VHDL Example**

```vhdl
signal quot_ext : signed(width downto 0);
quot_ext <= ((tc and a(width-1)) & a) / ((tc and b(width-1)) & b);
quot <= quot_ext(width-1 downto 0);
```

**Verilog Example**

```verilog
wire ext_bit;
assign {ext_bit, quot} = DWF_div_tc ({tc & a[width-1], a},
{tc & b[width-1], b});
```

**Related Topics**

- Math - Arithmetic Overview
- DesignWare Building Block IP Documentation Overview
HDL Usage Through Operator Inferencing - VHDL

library IEEE, DWARE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use DWARE.DW_Foundation_arith.all;

entity DW_div_oper is
  generic ( width : positive := 8);
  port ( a : in std_logic_vector(width-1 downto 0);
        b : in std_logic_vector(width-1 downto 0);
        quotient_uns : out std_logic_vector(width-1 downto 0);
        quotient_tc : out std_logic_vector(width-1 downto 0);
        remainder_uns : out std_logic_vector(width-1 downto 0);
        remainder_tc : out std_logic_vector(width-1 downto 0);
        modulus_uns : out std_logic_vector(width-1 downto 0);
        modulus_tc : out std_logic_vector(width-1 downto 0));
end DW_div_oper;

architecture oper of DW_div_oper is
begin

  -- operators for unsigned/signed quotient, remainder and modulus
  quotient_uns  <= unsigned(a) / unsigned(b);
  quotient_tc   <= signed(a) / signed(b);
  remainder_uns <= unsigned(a) rem unsigned(b);
  remainder_tc  <= signed(a) rem signed(b);
  modulus_uns   <= unsigned(a) mod unsigned(b);
  modulus_tc    <= signed(a) mod signed(b);

end oper;
HDL Usage Through Operator Inferencing - Verilog

module DW_div_oper (a, b, quotient_uns, quotient_tc, remainder_uns, remainder_tc);
    parameter width = 8;

    input [width-1 : 0] a, b;
    output [width-1 : 0] quotient_uns, remainder_uns;
    output signed [width-1 : 0] quotient_tc, remainder_tc;
    // operators for unsigned/signed quotient and remainder
    assign quotient_uns = a / b;
    assign quotient_tc = $signed(a) / $signed(b);
    assign remainder_uns = a % b;
    assign remainder_tc = $signed(a) % $signed(b);
endmodule
HDL Usage Through Function Inferencing - VHDL

library IEEE, DWARE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use DWARE.DW_Foundation_arith.all;

design DW_div_func

generic ( width : positive := 8);
port ( a : in std_logic_vector(width-1 downto 0);
b : in std_logic_vector(width-1 downto 0);
quotient_uns : out std_logic_vector(width-1 downto 0);
quotient_tc : out std_logic_vector(width-1 downto 0);
remainder_uns : out std_logic_vector(width-1 downto 0);
remainder_tc : out std_logic_vector(width-1 downto 0);
modulus_uns : out std_logic_vector(width-1 downto 0);
modulus_tc : out std_logic_vector(width-1 downto 0));
end DW_div_func;

architecture func of DW_div_func is
begin
-- function calls for unsigned/signed quotient, remainder and modulus
quotient_uns <= std_logic_vector(DWF_div (unsigned(a), unsigned(b)));
quotient_tc <= std_logic_vector(DWF_div (signed(a), signed(b)));
remainder_uns <= std_logic_vector(DWF_rem (unsigned(a), unsigned(b)));
remainder_tc <= std_logic_vector(DWF_rem (signed(a), signed(b)));
modulus_uns <= std_logic_vector(DWF_mod (unsigned(a), unsigned(b)));
modulus_tc <= std_logic_vector(DWF_mod (signed(a), signed(b)));
end func;
HDL Usage Through Function Inferencing - Verilog

module DW_div_func (a, b, quotient_uns, quotient_tc, remainder_uns, remainder_tc, modulus_uns, modulus_tc);

parameter width = 8;

input [width-1 : 0] a;
input [width-1 : 0] b;
output [width-1 : 0] quotient_uns;
output [width-1 : 0] quotient_tc;
output [width-1 : 0] remainder_uns;
output [width-1 : 0] remainder_tc;
output [width-1 : 0] modulus_uns;
output [width-1 : 0] modulus_tc;

// pass "a_width" and "b_width" parameters to the inference functions
parameter a_width = width;
parameter b_width = 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_div_function.inc"

// function calls for unsigned/signed quotient, remainder and modulus
assign quotient_uns = DWF_div_uns (a, b);
assign quotient_tc = DWF_div_tc (a, b);
assign remainder_uns = DWF_rem_uns (a, b); // corresponds to "%" in Verilog
assign remainder_tc = DWF_rem_tc (a, b); // corresponds to "%" in Verilog
assign modulus_uns = DWF_mod_uns (a, b); // corresponds to "mod" in VHDL
assign modulus_tc = DWF_mod_tc (a, b); // corresponds to "mod" in VHDL

endmodule
HDL Usage Through Component Instantiation - VHDL

library IEEE, DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DW_Foundation_comp_arith.all;

entity DW_div_inst is
  generic ( width : positive := 8;
            tc_mode : natural := 0;
            rem_mode : natural := 1);
  port ( a : in std_logic_vector(width-1 downto 0);
         b : in std_logic_vector(width-1 downto 0);
         quotient : out std_logic_vector(width-1 downto 0);
         remainder : out std_logic_vector(width-1 downto 0);
         divide_by_0 : out std_logic);
end DW_div_inst;

architecture inst of DW_div_inst is
begin
  -- instance of DW_div
  U1 : DW_div
    generic map ( a_width => width, b_width => width,
                  tc_mode => tc_mode, rem_mode => rem_mode)
    port map ( a => a, b => b,
               quotient => quotient, remainder => remainder,
               divide_by_0 => divide_by_0);
end inst;

-- pragma translate_off
configuration DW_div_inst_cfg_inst of DW_div_inst is
  for inst
  end for;
end DW_div_inst_cfg_inst;
-- pragma translate_on
HDL Usage Through Component Instantiation - Verilog

module DW_div_inst (a, b, quotient, remainder, divide_by_0);

parameter width = 8;
parameter tc_mode = 0;
parameter rem_mode = 1; // corresponds to "%" in Verilog

input [width-1 : 0] a;
input [width-1 : 0] b;
output [width-1 : 0] quotient;
output [width-1 : 0] remainder;
output divide_by_0;

// Please add +incdir+$SYNOPSYS/dw/sim_ver+ to your verilog simulator
// command line (for simulation).

// instance of DW_div
DW_div #(width, width, tc_mode, rem_mode)
   U1 (.a(a), .b(b),
       .quotient(quotient), .remainder(remainder),
       .divide_by_0(divide_by_0));
endmodule