Integer Dividers

January 1996, ver. 1 — Functional Specification 3

Features

- `divide` and `dividex` reference designs implementing high-speed, parallel dividers
- Parameterized dividend and divisor bit widths
- Optimized for the FLEX 10K and FLEX 8000 device families
- High-speed operation
- Two's complement arithmetic for all inputs and outputs
- Supported by schematic and text design entry methods, including VHDL, Verilog HDL, and the Altera Hardware Description Language (AHDL)
- Useful for a variety of applications, including scaling data and computations

Division Algorithms

Altera FLEX 10K and FLEX 8000 devices provide an ideal architecture for implementing arithmetic division. The FLEX architecture includes an efficient and straightforward addition/subtraction function in each FLEX logic element (LE).

Two algorithms are commonly used to perform exact arithmetic division: restoring and non-restoring. The divider functions described in this functional specification are implemented using a non-restoring division algorithm.

Restoring Division

In restoring division, the divisor is shift-positioned and subtracted from the dividend. If subtraction of the divisor produces a negative result at any bit position relative to the dividend, the operation at that bit position is unsuccessful, and a 0 is placed in the corresponding location of the quotient. The divisor is added back (restored) to the result of the division operation, then the next highest bit of the dividend is shifted into the left bit position of the result. As each bit of the dividend is shifted from right to left, the quotient is built up from left to right. After n shifts, where n represents the number of bits in the dividend, the division operation is complete. The result after the last restore operation is the remainder. This algorithm is very similar to manually performing long division.

Non-Restoring Division

Non-restoring division is developed from the restoring algorithm as shown below. The operation in each step depends on the result of the previous step.

1. Subtract the divisor from the most significant bit (MSB) of the dividend.

2. "Bring down" the next MSB of the divisor and append it to the result of step 1.

3. Check the sign for the result of step 2. If the result of step 2 is positive:

   a. Set the next MSB of the quotient to 1.

   b. Subtract the divisor from the result to produce a new result.

   If the result from step 2 is negative:

   a. Set the next MSB of the quotient to 0.

   b. Add the divisor to the result to produce a new result.

4. Repeat steps 2 and 3 until all bits of the quotient are determined.

For example, to calculate the following equation, you can use longhand division as shown in Figure 1.

$2\overline{)41}$ decimal = $010\overline{)0101001}$ binary

Figure 1. Non-Restoring Division

In this example, 2 decimal equals 010 binary, and –2 decimal equals 110 binary. The bits shown in blue are “brought down” from the dividend.

010 ⟌ 0101001	Decimal Equivalent:	Comment:	Quotient:
110	–2	Subtract divisor (by adding two’s complement).	???????
1101	–3	Result is negative, MSB of quotient is 0.	0??????
010	2	Add divisor.	
1110	–2	Result is negative, MSB – 1 of quotient is 0.	00?????
010	2	Add divisor.	
0001	1	Result ≥ 0, MSB – 2 of quotient is 1.	001????
110	–2	Subtract divisor.	
1110	–1	Result < 0, MSB – 3 of quotient is 0.	0010???
010	2	Add divisor.	
0000	0	Result ≥ 0, MSB – 4 of quotient is 1.	00101??
110	–2	Subtract divisor.	
1101	–3	Result < 0, MSB – 5 of quotient is 0.	001010?
010	–2	Add divisor.	
111	–1	Result < 0, MSB – 6 of quotient is 0.	0010100
010	2	Add divisor for remainder.	
001	1	Remainder = 1.	

This long division produces a quotient of 0010100 binary, i.e., 20 decimal, with a remainder of 1. The result is intuitive because any number divided by 2 is simply that number right-shifted by 1 bit.

Extra logic is required to accommodate the two’s complement dividend and divisors to ensure that the signs are correct. This logic is implemented in the AHDL Text Design File (**.tdf**).

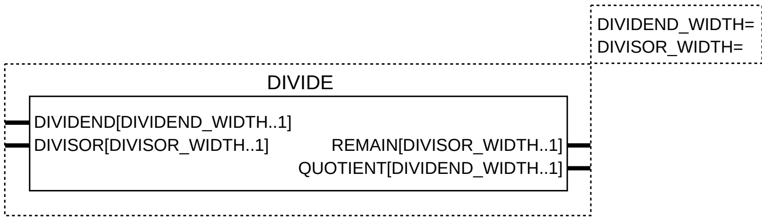
Go to MAX+PLUS II Help for more information about AHDL.

Non-restoring division can be easily extended to produce a fractional result. Simply assume a binary point to the right of the dividend's least significant bit (LSB), and right-extend the dividend with 0s as far as desired. The same algorithm is continued until the quotient reaches the desired precision.

General Description

The `divide` reference design, implemented with the AHDL file **divide.tdf**, is a high-speed integer parallel divider with a remainder. It provides parameterized dividend and divisor (and consequently quotient and remainder) bit widths. See Figure 2.

Figure 2. divide Symbol



Regardless of the sign of the dividend and divisor, this function always maintains the following relationship:

quotient + (remainder / divisor) = (dividend / divisor)

This number can be represented in multiple ways. For example, the decimal result for both –3 / 2 and 3 / –2 is –1.5. In quotient + remainder notation, –3 / 2 is represented as – 2 + 1 / 2, and 3 / –2 is represented as –2 + –1 / –2.

Function Prototype

The AHDL Function Prototype for the `divide` function is shown below:

```
FUNCTION divide (dividend[dividend_width..1],
  divisor[divisor_width..1])
    WITH (dividend_width, divisor_width)
    RETURNS (remain[divisor_width..1], quotient[dividend_width..1]);
```

Parameters

Parameters for the `divide` function are provided in Table 1.

Table 1. divide Parameters

Name	Default	Value	Description
`dividend_width`	7	Integer	Width of dividend (in bits)

Table 1. divide Parameters			
Name	**Default**	**Value**	**Description**
`divisor_width`	7	Integer	Width of divisor (in bits)

Ports

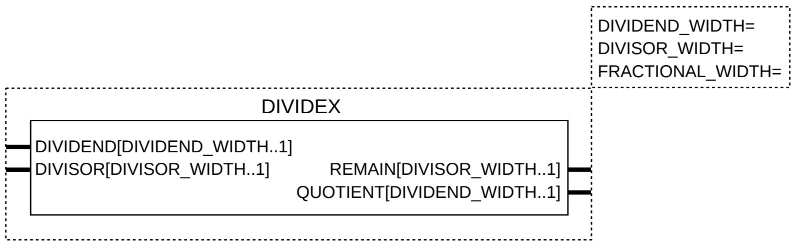
Input and output ports for the `divide` function are described in Table 2.

Table 2. Input & Output Ports		
Port Type	**Name**	**Description**
Input	`dividend[dividend_width..1]`	Dividend input
Input	`divisor[divisor_width..1]`	Divisor input
Output	`quotient[dividend_width..1]`	Integer portion of result; value is `dividend[]/divisor[]`
Output	`remain[divisor_width..1]`	Remainder; value is `dividend[]` mod `divisor[]`

General Description

The `dividex` reference design, implemented with the AHDL file **dividex.tdf**, is a high-speed integer parallel divider with a fractional result. See Figure 3. The algorithm used for the `dividex` function is identical to the algorithm used for the `divide` function, except it includes an assumed binary point to the right of the LSB of the dividend. All bits to the right of the binary point are assumed to be 0.

Figure 3. dividex Symbol



Function Prototype

The AHDL Function Prototype for the `dividex` function is shown below:

```
FUNCTION dividex (dividend[dividend_width..1],
  divisor[divisor_width..1])
    WITH (dividend_width, divisor_width, fractional_width)
    RETURNS (quotient[dividend_width..1],
      fractional[fractional_width..1]);
```

Parameters

Parameters for the `dividex` function are provided in Table 3.

Table 3. dividex Parameters

Name	Default	Value	Description
`dividend_width`	7	Integer	Width of dividend (in bits)
`divisor_width`	7	Integer	Width of divisor (in bits)
`fractional_width`	4	Integer	Width of fractional portion of output

Ports

Input and output ports for the `dividex` function are described in Table 4.

Table 4. Input & Output Ports		
Port Type	**Name**	**Description**
Input	`dividend[dividend_width..1]`	Dividend input.
Input	`divisor[divisor_width..1]`	Divisor input.
Output	`quotient[dividend_width..1]`	Integer portion of result. Value is `integer (dividend[]/divisor[])`.
Output	`fractional[divisor_width..1]`	Fractional portion of result. Value is `fractional (dividend[]/divisor[])`.

2610 Orchard Parkway
San Jose, CA 95134-2020
(408) 894-7000
Applications Hotline:
(800) 800-EPLD
Customer Marketing:
(408) 894-7104
Literature Services:
(408) 894-7144