Key Design Features

- Synthesizable, technology independent VHDL Core
- Function y = a / b
- Input values as signed or unsigned integers
- Output values as signed or unsigned integers
- Configurable numerator and denominator data width
- High-speed fully pipelined architecture with configurable number of register stages for area/speed trade off
- Quotient, remainder and div_zero outputs
- One output result per clock-cycle (i.e. pipelined operation)
- Capable of clock speeds of 400MHz+ on even basic FPGA platforms

Applications

- Fundamental building block in digital processing functions
- Division of integers and fixed-point numbers[1]
- Implementation of the reciprocal function f(x) = 1/x

Pin-out Description

Pin name	*I/O*	*Description*	*Active state*
clk	in	Synchronous clock	rising edge
en	in	Clock enable	high
a_in [dw-1:0]	in	Input numerator	data
b_in [dw-1:0]	in	Input denominator	data
quotient [dw -1:0]	out	Output quotient	data
remainder [dw-1:0]	out	Output remainder	data
div_zero	out	Divide by zero flag	high

Generic Parameters

Generic name	*Description*	*Type*	*Valid range*
dw	Input data width	integer	≥ 2 (dw | 2 = 0)
use_signed	Use signed or unsigned arithmetic	boolean	TRUE/FALSE
reg_stages	Number of pipeline register stages	integer	≥ 1 ≤ dw

1 For fixed-point numbers then inputs must be pre-scaled by a power of 2. E.g. the division 0.2/0.3 could be done as 51/77 in 8-bit arithmetic.

Block Diagram

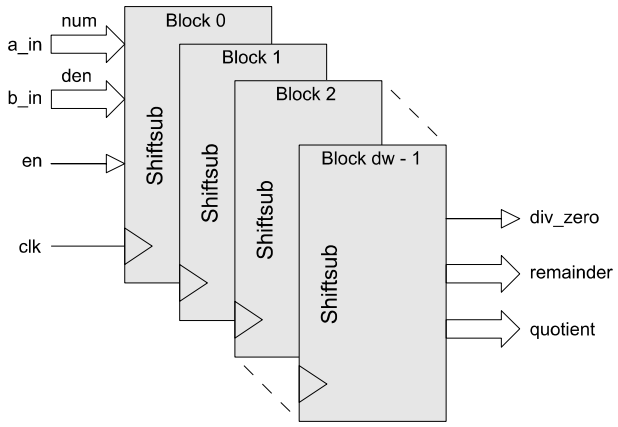

Figure 1: Pipelined Divider Architecture

General Description

PIPE_DIV (Figure 1) is a pipelined divider with configurable data width. The design is fully scalable and modular permitting the user to specify large dividers without compromising maximum attainable clock-speed.

The divider accepts input values as either signed or unsigned integers depending on the generic setting *use_signed*. An n-bit numerator and denominator will generate an n-bit result for the quotient and an n-bit result for the remainder. The output remainder always takes the sign of the numerator and is determined by the formula:

$$Rmd = Num - Quo * Den$$

Where *Rmd*, *Num*, *Quo* and *Den* represent the remainder, numerator, quotient and denominator respectively.

In the case of a divide by zero, then the *div_zero* flag is asserted at the divider output and the maximum value possible is returned in the quotient. The remainder takes the value of the numerator. For example, if *dw* = 8, the division -3/0 will return a result of -128 for the quotient and -3 for the remainder. The division 3/0 would return a remainder of 3 and a quotient of 127 for signed arithmetic or 255 for unsigned.

Values are sampled on the rising clock-edge of *clk* when *en* is high. The number of register stages in the pipeline may be modified in order to trade off maximum speed against the total resource used. The overall pipeline latency of the divider is given by the formula:

$$Latency = dw \,/\, reg\_stages$$

For example, a 24-bit divider with the number of register stages set to 3 will result in a circuit with 8 clock cycles of latency. In other words, the result of a division will take 8 clock-cycles to appear at the output. Note that while the latency may change depending on the implementation, the throughput is always maintained at one output result per clock.

Functional Timing

Figure 2 demonstrates two sequential calculations of 10/-3 and -5/0. In this example, the parameters have been set to dw = 4, use_signed = true, reg_stages = 1. The result has a latency of 4 clock cycles.

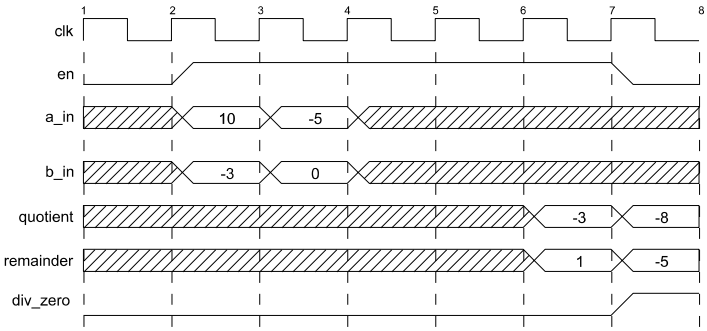

Figure 2: Calculation of a/b

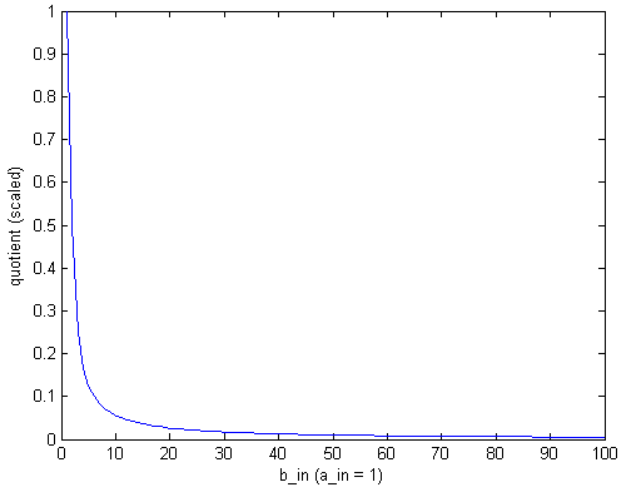

Figure 3: Plot of test results for function: f(x) = 1/x

Source File Description

All source files are provided as text files coded in VHDL. The following table gives a brief description of each file.

Source file	***Description***
pipe_div_shiftsub.vhd	Shift-subtract block
pipe_div.vhd	Top-level block
pipe_div_bench.vhd	Top-level test bench

Functional Testing

An example VHDL testbench is provided for use in a suitable VHDL simulator. The compilation order of the source code is as follows:

1. pipe_div_shiftsub.vhd
2. pipe_div.vhd
3. pipe_div_bench.vhd

The VHDL testbench instantiates the divider component and the user may modify the generic parameters as required. The simulation must be run for at least 2 ms during which time the divider will be driven with a randomized sequence input values. The test terminates automatically.

The simulation generates two text files called: *pipe_div_in.txt* and *pipe_div_out.txt.* These files respectively contain the input and output data samples captured at the interfaces during the test.

Figure 3 shows the results of the divider used to implement the function f(x) = 1/x with the generic parameter dw = 16. Results are shown for the first 100 samples.

Synthesis

The source files required for synthesis and the design hierarchy is shown below:

- pipe_div.vhd
  - pipe_div_shiftsub.vhd

The VHDL core is designed to be technology independent. However, as a benchmark, synthesis results have been provided for the Xilinx Virtex 5 and the Altera Stratix III series of FPGA devices. The lowest and highest speed grade devices have been chosen in both cases for comparison.

Note that the generic parameter *reg_stages* will have a significant effect on the speed and area of the synthesized design. For the fastest possible design, the generic parameter *reg_stages* should be set to 1. For the smallest design, then *reg_stages* should be set to equal the data width, *dw*. In addition, choice of unsigned logic will result in a design with a slightly smaller area.

Trial synthesis results are shown with the generic parameters set to: dw = 16, use_signed = true, reg_stages = 1.

Resource usage is specified after Place and Route.

VIRTEX 5

Resource type	***Quantity used***
Slice register	801
Slice LUT	704
Block RAM	0
DSP48	0
Clock frequency (worst case)	385 MHz
Clock frequency (best case)	495 MHz

STRATIX III

Resource type	***Quantity used***
Register	947
ALUT	751
Block Memory bit	225
DSP block 18	0
Clock frequency (worse case)	350 MHz
Clock frequency (best case)	400 MHz

Revision History

Revision	***Change description***	***Date***
1.0	Initial revision	23/07/2008
1.1	Added new *opt_mode* generic parameter	02/10/2008
1.2	Renamed *opt_mode* parameter to reg_stages. Design now supports any number of register stages up to the maximum *dw*	09/09/2011