Key Design Features

- Synthesizable, technology independent VHDL IP Core
- Fully synchronous design
- Configurable data width
- Uses a simple valid-ready streaming protocol
- Compatible with other streaming protocols such as: AMBA® AXI4-stream, Altera® Avalon-ST and Xilinx® local-link
- Self flushing
- 1 cycle latency

Applications

- High-speed data streaming interfaces for DSP, video and data
- Interfacing between other pipeline elements
- Interfacing with other Zipcores IP Cores
- Registering the data path to improve timing
- Registering the data path on and off chip
- Simple buffering

Generic Parameters

Generic name	*Description*	*Type*	*Valid range*
dw	data width	integer	≥ 1

Pin-out Description

Pin name	*I/O*	*Description*	*Active state*
clk	in	Synchronous clock	rising edge
reset	in	Asynchronous reset	low
datain [dw-1:0]	in	Pipeline register input data	data
datain_val	in	Indicates valid data at the pipeline register input	high
datain_rdy	out	Indicates that the pipeline register is ready to accept data	high
dataout [dw-1:0]	out	Pipeline register output data	data
dataout_val	out	Indicates that the pipeline register contains valid data	high
dataout_rdy	in	Indicates that the output is ready to receive data	high

Block Diagram

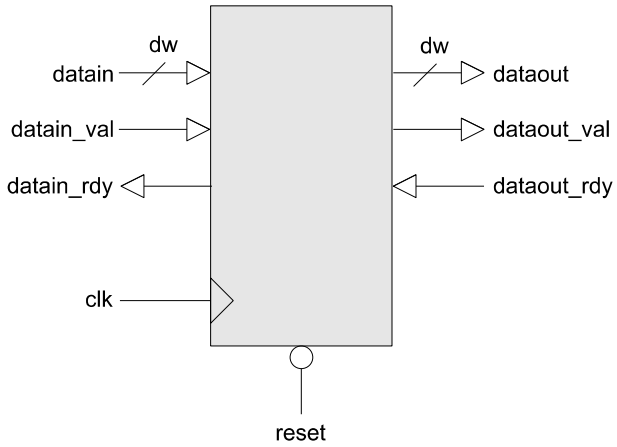

Figure 1: Pipeline register

General Description

PIPELINE_REG (Figure 1) is a self-flushing register element used as a fundamental building block in pipelined architectures. It's principal use is to register the data path between series elements in a pipeline and also to interface between blocks using a simple data-streaming protocol.

Data flows in and out of the pipeline register in accordance with the valid-ready protocol[1]. Data is written to the register on a rising clock-edge when both *datain_val* and *datain_rdy* are high. Data is read out of the register on a rising clock-edge when both *dataout_val* and *dataout_rdy* are high. The component has a latency of 1 clock cycle.

The pipeline register is self-flushing in that once data is written to the register then it will try and empty this data on the next clock cycle - irrespective of whether there is new input data or not. In this way, the register works somewhat like a one-deep FIFO.

If the register is full then the internal state-machine will maintain *dataout_val* asserted until the register is empty or no further valid input data is present. Conversely, if the register is empty, then the state-machine will assert *datain_rdy* high to actively request data from the upstream interface.

1 For more examples of the valid-ready streaming protocol and it's implementation see Zipcores application note: app_note_zc001.pdf

Functional Timing

Figure 2 shows a simple data transfer when the pipeline register is initially empty and the downstream side is stalling. The downstream side then asserts *dataout_rdy* high to complete the data transfer. Figure 3 demonstrates a series of sequential data transfers without stalling.

Note that data is only transferred at the pipeline register interfaces on a rising clock-edge when valid and ready are both active high.

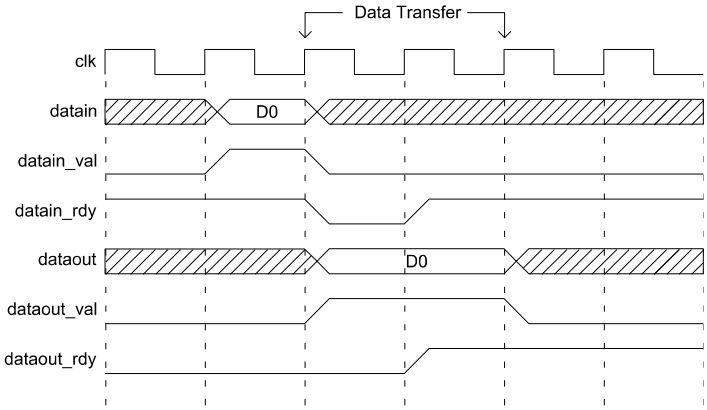

Figure 2: Pipeline stalling

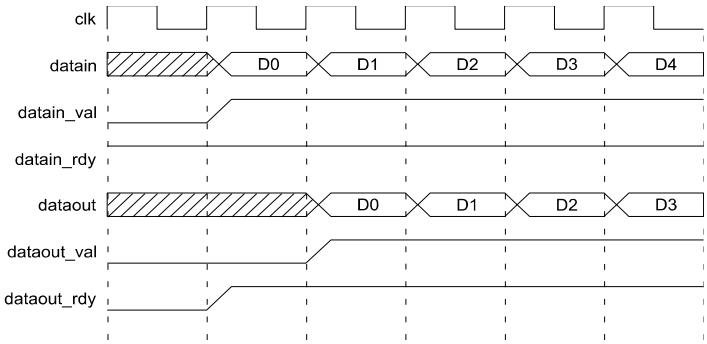

Figure 3: Sequential data transfers

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***
pipeline_reg.vhd	Top-level block
pipeline_reg_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. pipeline_reg.vhd
2. pipeline_reg_bench.vhd

The VHDL testbench instantiates the pipeline register component and the user may modify the generic parameters as required.

The simulation must be run for at least 3 ms during which time a randomized input data sequence will be written to the pipeline register. In addition to the random input data, the test bench also generates randomized valid and ready signals at the input and output of the pipeline register in order to verify that the the flow-control is working correctly.

The simulation generates two text files called: *pipeline_reg_in.txt* and *pipeline_reg_out.txt*. These files respectively contain the input and output data captured at the interfaces during the test. Matching input and output files indicate that the test has run successfully.

Synthesis

The source file 'pipeline_reg.vhd' is the only file required for synthesis. There are no sub-modules in the design.

The VHDL core is designed to be technology independent. However, as a benchmark, synthesis results have been provided for the Xilinx® Virtex 6 and Spartan-6 FPGA devices. Synthesis results for other FPGAs and technologies can be provided on request.

Trial synthesis results are shown with the generic parameters set to: dw = 32. Resource usage is specified after Place and Route.

VIRTEX 6

Resource type	***Quantity used***
Slice register	33
Slice LUT	34
Block RAM	0
DSP48	0
Occupied Slices	10 MHz
Clock frequency (approx)	600 MHz

SPARTAN 6

Resource type	***Quantity used***
Slice register	34
Slice LUT	33
Block RAM	0
DSP48	0
Occupied Slices	10
Clock frequency (approx)	600 MHz

Revision History

Revision	*Change description*	*Date*
1.0	Initial revision	19/04/2008
1.1	Minor source-code changes. Previous data held as default value when no further valid input present.	06/04/2014