

Single-Stage Pipeline Register with Valid/Ready Handshake

A synthesizable SystemVerilog implementation of a pipeline register using standard valid/ready handshake protocol for correct backpressure handling.

Overview

This module implements a single-stage pipeline buffer that:

- Accepts data when both `in_valid` and `in_ready` are asserted
- Stores data internally and presents it with `out_valid` asserted
- Handles backpressure correctly when `out_ready` is deasserted
- Guarantees no data loss or duplication
- Resets to a clean, empty state

Interface

Parameters

- `DATA_WIDTH`: Width of the data path (default: 32 bits)

Ports

Port	Direction	Width	Description
<code>clk</code>	input	1	Clock signal
<code>rst_n</code>	input	1	Active-low asynchronous reset
<code>in_valid</code>	input	1	Input data valid signal
<code>in_ready</code>	output	1	Ready to accept input data
<code>in_data</code>	input	<code>DATA_WIDTH</code>	Input data
<code>out_valid</code>	output	1	Output data valid signal
<code>out_ready</code>	input	1	Downstream ready to accept data

Port	Direction	Width	Description
out_data	output	DATA_WIDTH	Output data

Protocol

The module implements the standard valid/ready handshake:

1. **Data Transfer:** Occurs when `valid` and `ready` are both high on a clock edge
2. **Input Side:** `in_ready` is high when the register can accept new data (empty or being read)
3. **Output Side:** `out_valid` is high when the register contains valid data
4. **Backpressure:** When `out_ready` is low, data is held without loss

Design Details

State Machine

The design uses a simple valid bit to track whether the register contains data:

- **Empty:** `valid_reg = 0`, ready to accept new data
- **Full:** `valid_reg = 1`, presenting data to output

Handshake Logic

```
systemverilog

input_transfer = in_valid && in_ready
output_transfer = out_valid && out_ready
in_ready      = !valid_reg || output_transfer
```

Key Features

1. **No Data Loss:** Data is held stable when `out_valid` is high and `out_ready` is low
2. **No Duplication:** Each input transfer produces exactly one output transfer
3. **Full Throughput:** Can accept new data on the same cycle old data is consumed
4. **Clean Reset:** Asynchronous reset clears valid flag and data register

Verification

The testbench (`(pipeline_register_tb.sv)`) includes:

1. **Basic Transfer Test:** Simple data through the pipeline
2. **Backpressure Test:** Verifies data holds when output not ready
3. **Continuous Stream Test:** 10 consecutive transfers at full throughput
4. **Reset Test:** Verifies clean reset behavior
5. **Random Handshake Test:** Randomized valid/ready patterns

Running the Testbench

With any SystemVerilog simulator (Questa, VCS, Verilator, etc.):

```
bash

# Example with Questa
vlog pipeline_register.sv pipeline_register_tb.sv
vsim -c pipeline_register_tb -do "run -all; quit"

# Example with Verilator (requires C++ wrapper)
verilator --binary --timing -Wall pipeline_register.sv pipeline_register_tb.sv
./obj_dir/Vpipeline_register_tb
```

Synthesis Considerations

- **Fully Synthesizable:** No simulation-only constructs in RTL
- **Single Flip-Flop Stage:** Minimal area and power
- **Combinational Ready:** `[in_ready]` is combinational for zero-cycle acceptance
- **Asynchronous Reset:** Compatible with most ASIC and FPGA flows
- **No Timing Loops:** Clean combinational and sequential separation

Timing

- **Latency:** 1 cycle from input transfer to output valid
- **Throughput:** 1 transfer per cycle when not backpressured

- **Ready Path:** Combinational from `out_ready` to `in_ready`

Use Cases

This pipeline register is useful for:

- Interface buffering between pipeline stages
- Clock domain crossing preparation (add synchronizers)
- Timing closure (breaking long combinational paths)
- Rate matching between producer and consumer

Formal Verification

The design includes SVA assertions (enabled with `FORMAL` define):

- `p_no_data_loss`: Data preserved during backpressure
- `p_valid_until_ready`: Valid stays high until acknowledged
- `p_reset_clears_valid`: Reset properly clears state

License

This is a reference implementation for educational and practical use.

Author

Created as a practical RTL design exercise demonstrating:

- Standard valid/ready handshake protocol
- Proper backpressure handling
- Synthesizable SystemVerilog coding style
- Comprehensive verification methodology