**Implementation of SKID BUFFER**

**using AXI4-Stream protocol**

**Why we need skid-buffer?**

In most modern bus protocols like **AXI4-Stream**, data transfer between Master(sender) and Slave(receiver) happens using a **handshake**:

* **Valid**: Asserted by the sender, which indicates it has**valid data**.
* **Ready**: Asserted by the receiver, which indicates it is ready to **accept** the data.
* Data is transferred only when **Valid = 1** and **Ready = 1** in the same cycle.

But when Ready signal suddenlygoes low while the sender has already asserted Valid signal, there might be a chance of losing data in this case. This scenario is called **“Back-Pressure”** where master is not ready(ready=0) to accept the data(valid=1).

**Skid Buffer:**

It is like a temporary storage device(a one deep fifo), which stores the valid data in back-pressure condition and placed between master and slave device in a handshake-based protocol. It allows one data word to “skid”through it when Ready suddenly deasserts.

**Block diagram:**

Valid

Data

Ready

**Slave**

**Master**

**AXI4-STREAM**

clk

o\_data

i\_data

**Skid-buffer**

i\_valid

o\_valid

**Slave**

**Master**

resetn

i\_ready

o\_ready

**Skid Buffer in AXI4-STREAM**

**Uses of skid-buffer:**

* AXI-Stream interconnects
* Pipeline stages in high-frequency designs
* Breaking long combinational Ready paths
* Preventing throughput loss in backpressure situations

**Working:**

**Case 1: Receiver is Ready**

* Data just flows through the buffer (transparent).
* No delay, no storage needed.

**Case 2: Receiver deasserts Ready suddenly**

* Sender may still drive one more valid word.
* Skid buffer captures this word in its register.
* Output to receiver is stalled until Ready is reasserted.

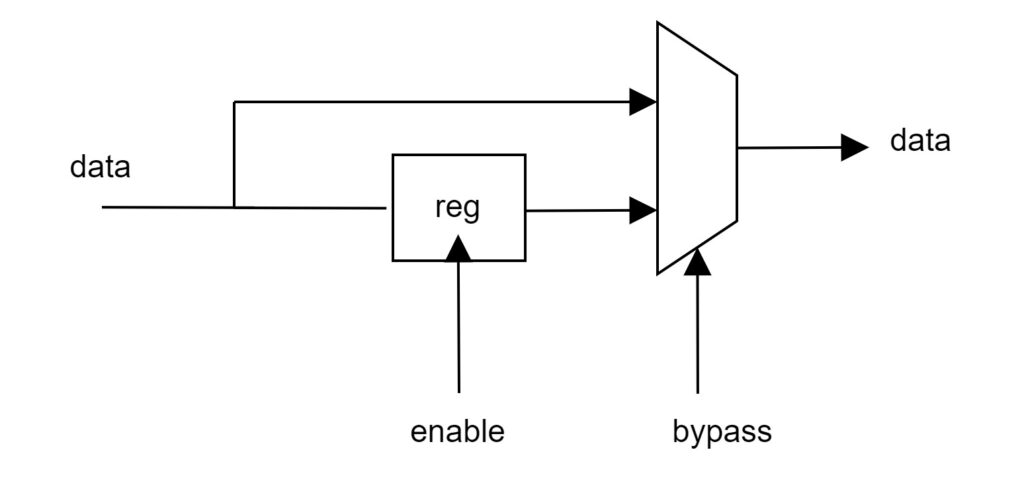
**Case 3: Receiver becomes Ready again**

* Skid buffer forwards the stored word.
* Then continues with direct pass-through.

**Implementation:**

A typical skid buffer is made of:

* **Register** to hold 1 data word
* **Valid signal** for stored word
* Simple control logic



**Design code :**

`timescale 1ns / 1ps

module skid\_buffer\_axi # (parameter WIDTH=32)

( // Global signals

input aclk,aresetn,

//input interface

input i\_valid,

input [WIDTH-1:0] i\_data,

output reg i\_ready,

//output interface

output reg o\_valid,

output reg [WIDTH-1:0] o\_data,

input o\_ready);

//skid\_buffer internal signals

reg [WIDTH-1:0] s\_data =0;

reg s\_valid =0;

always @(posedge aclk or negedge aresetn)

begin

i\_ready = (~s\_valid) & ~(s\_valid & o\_ready) ;

if(~aresetn)

begin

o\_data <= 0;

o\_valid <= 0;

end

else begin

// Default value

o\_valid <= 0;

// Case-1: Flushing skid-buffer data to output interface

if (s\_valid && o\_ready) begin

o\_data <= s\_data; // sending skid data

o\_valid <= 1;

s\_valid <= 0;

i\_ready <= 0;

end

// Case-2: Direct data transfer from input to output interface

else if (i\_valid && o\_ready && ~s\_valid) begin

o\_data <= i\_data;

o\_valid <= 1;

end

// Case-3: If output is not ready to accept data then capture it into skid

else if (i\_valid && ~o\_ready && ~s\_valid) begin

s\_data <= i\_data;

s\_valid <= 1;

end

end

end

endmodule

**Testbench code :**

`timescale 1ns / 1ps

module skid\_buffer\_tb();

parameter WIDTH=32;

reg aclk,aresetn;

reg i\_valid;

reg [WIDTH-1:0] i\_data;

wire i\_ready;

wire o\_valid;

wire [WIDTH-1:0] o\_data;

reg o\_ready;

skid\_buffer\_axi drt (aclk,aresetn,i\_valid,i\_data,i\_ready,o\_valid,o\_data,o\_ready); //dut

initial aclk=0;

always #5 aclk=~aclk;

initial begin

aresetn =0;

#20;

aresetn=1;

end

initial begin

i\_valid =0; //initial values

i\_data=0;

o\_ready=0;

@(posedge aresetn);

o\_ready=1;

//Direct data transfer

@(posedge aclk);

i\_data = 32'haadd\_1234;

i\_valid=1;

@(posedge aclk);

i\_valid=0;

//Skid capture

@(posedge aclk);

i\_data = 32'h3333\_1234;

i\_valid=1;

o\_ready=0;

// Flushing the skid

@(posedge aclk);

o\_ready=1;

i\_valid=0;

//Direct data transfer

@(posedge aclk);

i\_valid=1;

i\_data = 32'h7777\_cccc;

@(posedge aclk);

i\_data = 32'h0000\_cccc;

//Skid capture

@(posedge aclk);

i\_data = 32'h1111\_2222;

o\_ready =0;

//Flushing the skid + new data

@(posedge aclk);

o\_ready=1;

i\_data = 32'h1236\_9870;

i\_valid=1;

@(posedge aclk);

@(posedge aclk);

i\_valid=1;

i\_data = 32'hffff\_ffff;

//Skid capture

@(posedge aclk);

o\_ready=0;

i\_data = 32'h0123\_4567;

//Flushing the skid

@(posedge aclk);

o\_ready=1;

repeat(2) @(posedge aclk);

$finish;

end

endmodule

**Simulation waveform:**

