

ELD Lab 6

Exploring AXI Interface

Objective

- Understand the basics of AXI Interface
- Design floating point arithmetic using logarithmic and square root IPs available in Vivado

$$y = \frac{1}{\ln(x)}$$

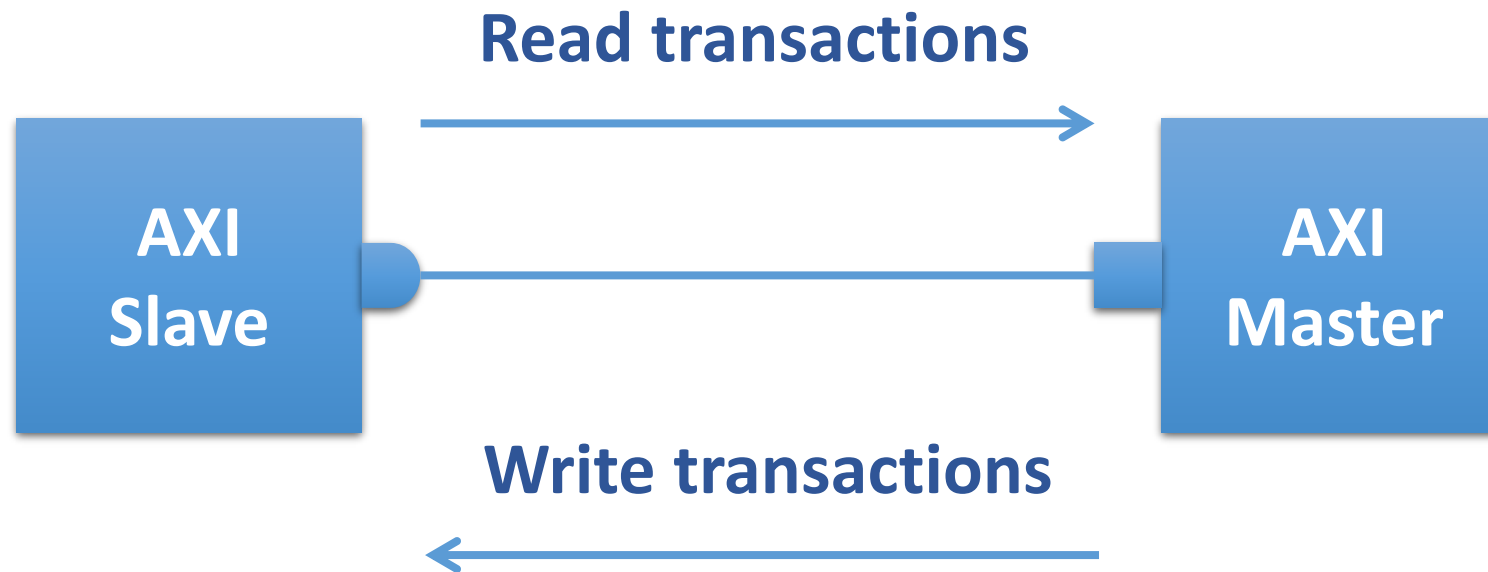
- **Lab Homework:** Design the floating point arithmetic circuit to implement following equation

$$z = \sqrt{x} + \frac{1}{\ln y} + 1.5$$

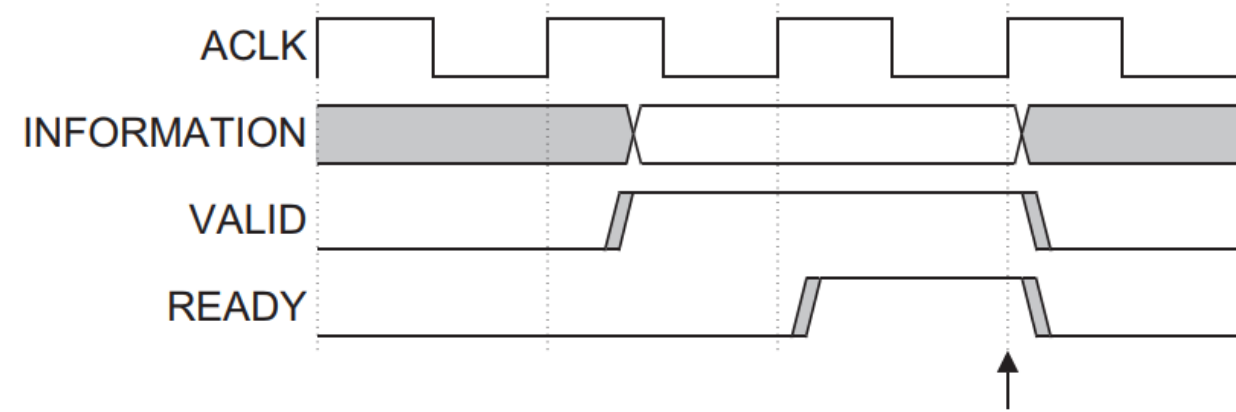
Theory

AXI

- ❖ Every AXI link contains two part: **AXI master and AXI slave**.
- ❖ **AXI master initializes the transactions** such as read and write. **AXI slave is the one who responds to AXI master transactions.**
- ❖ **Transaction:** Transfer of data from one point to another point

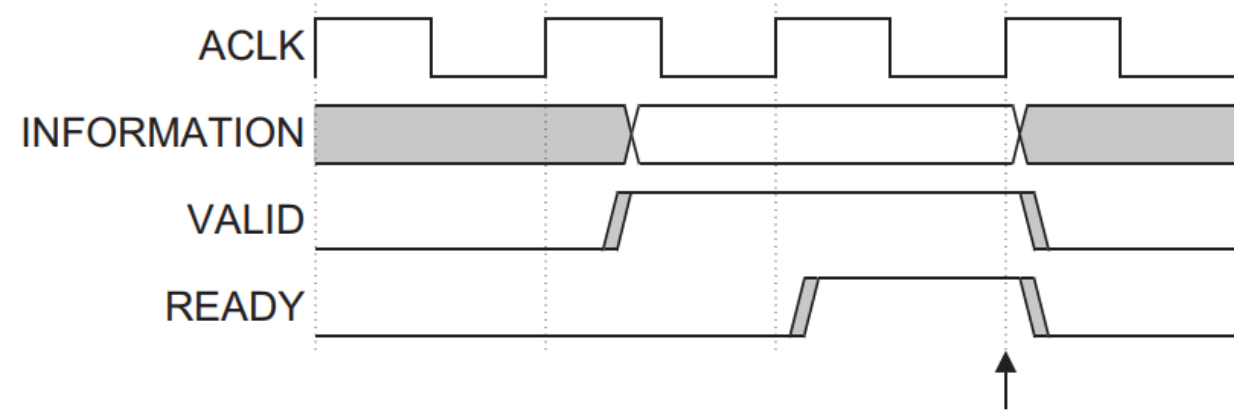


AXI Memory Mapped



VALID before READY handshake

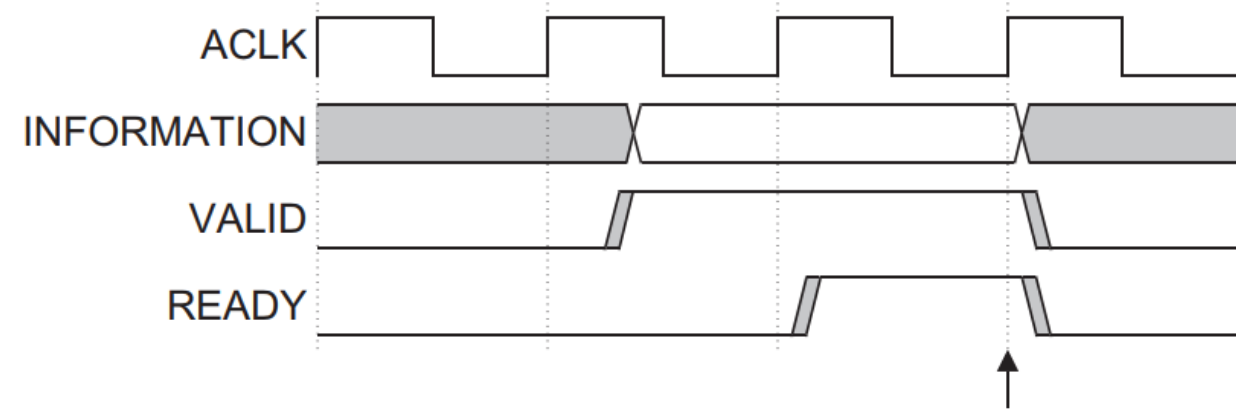
AXI Memory Mapped



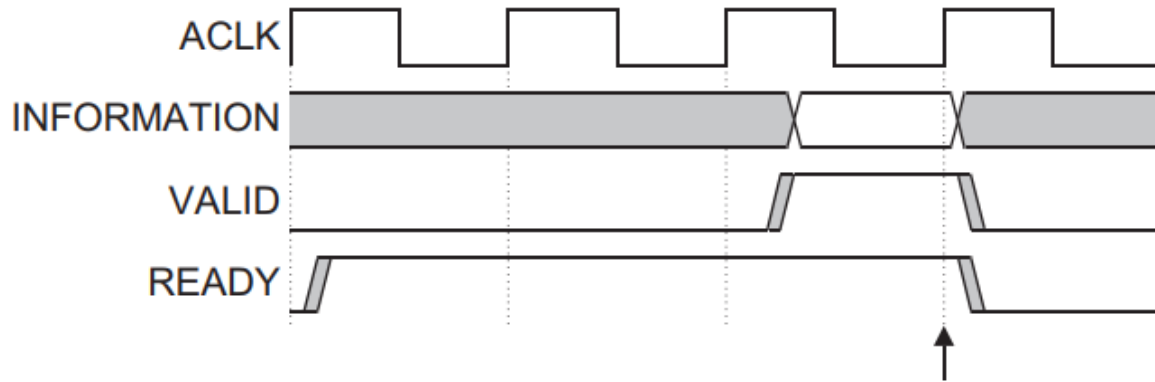
VALID before READY handshake

- ❖ The *source* generates the **VALID** signal to indicate when the address, data or control information is available.
- ❖ The *destination* generates the **READY** signal to indicate that it can accept the information.
- ❖ Transfer occurs only when *both* the **VALID** and **READY** signals are HIGH

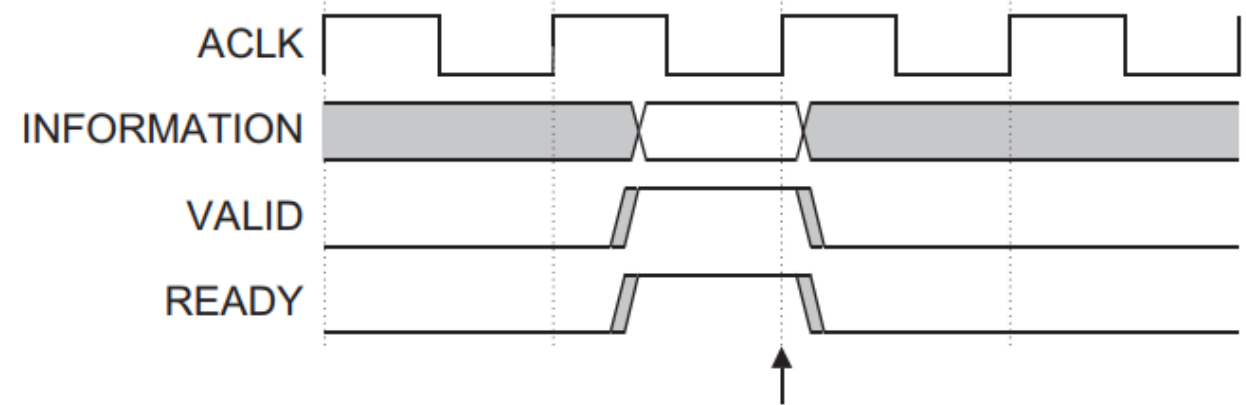
AXI Memory Mapped



VALID before READY handshake

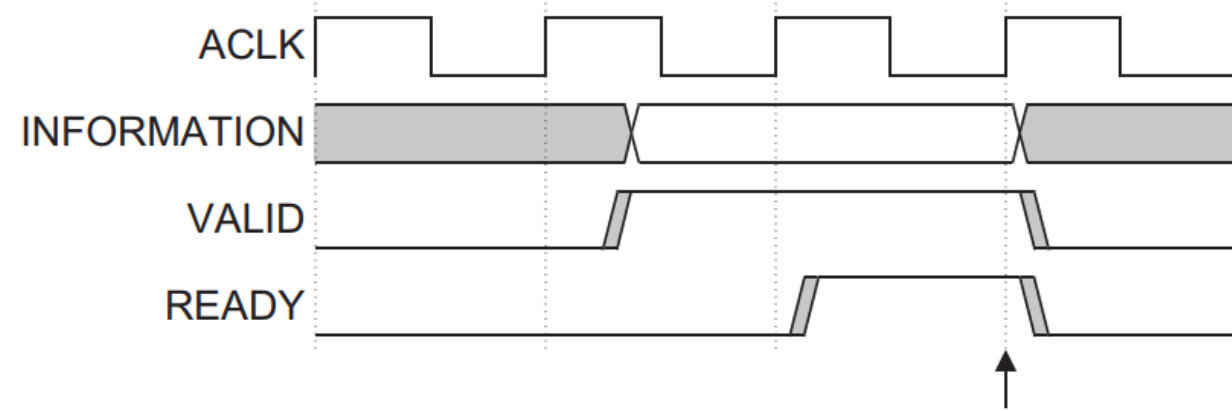


READY before VALID handshake



VALID with READY handshake

AXI Memory Mapped

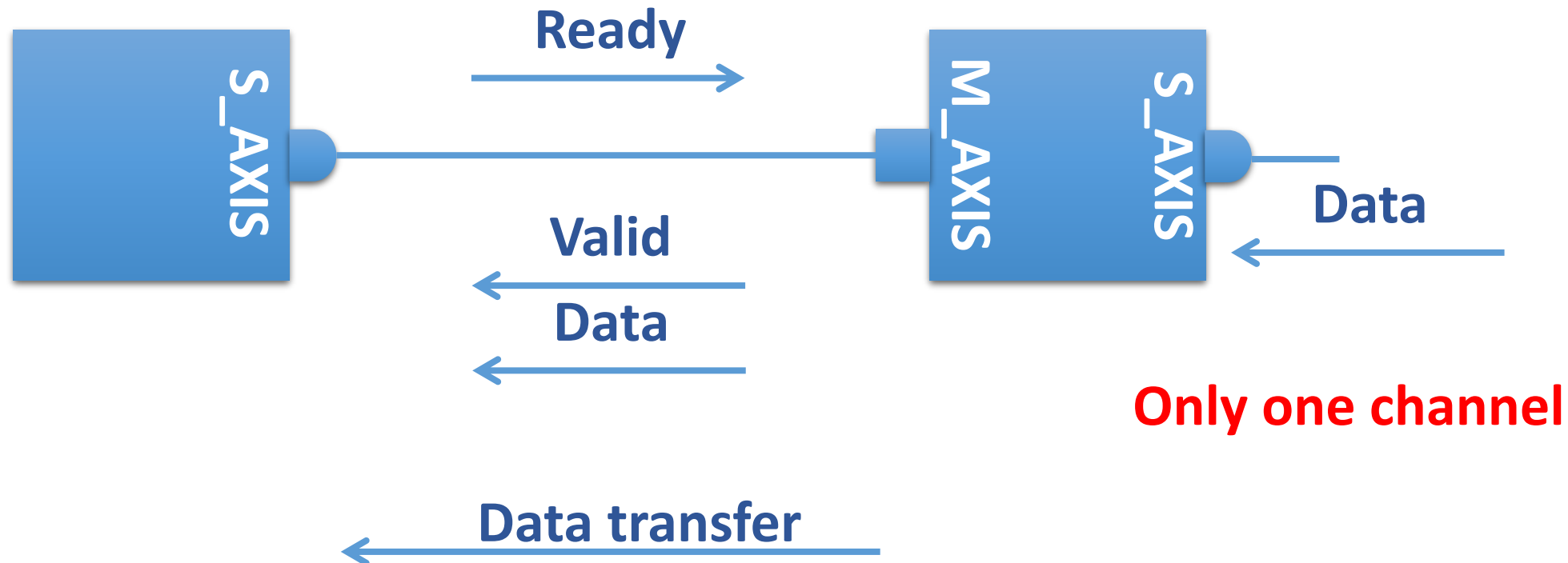


VALID before READY handshake

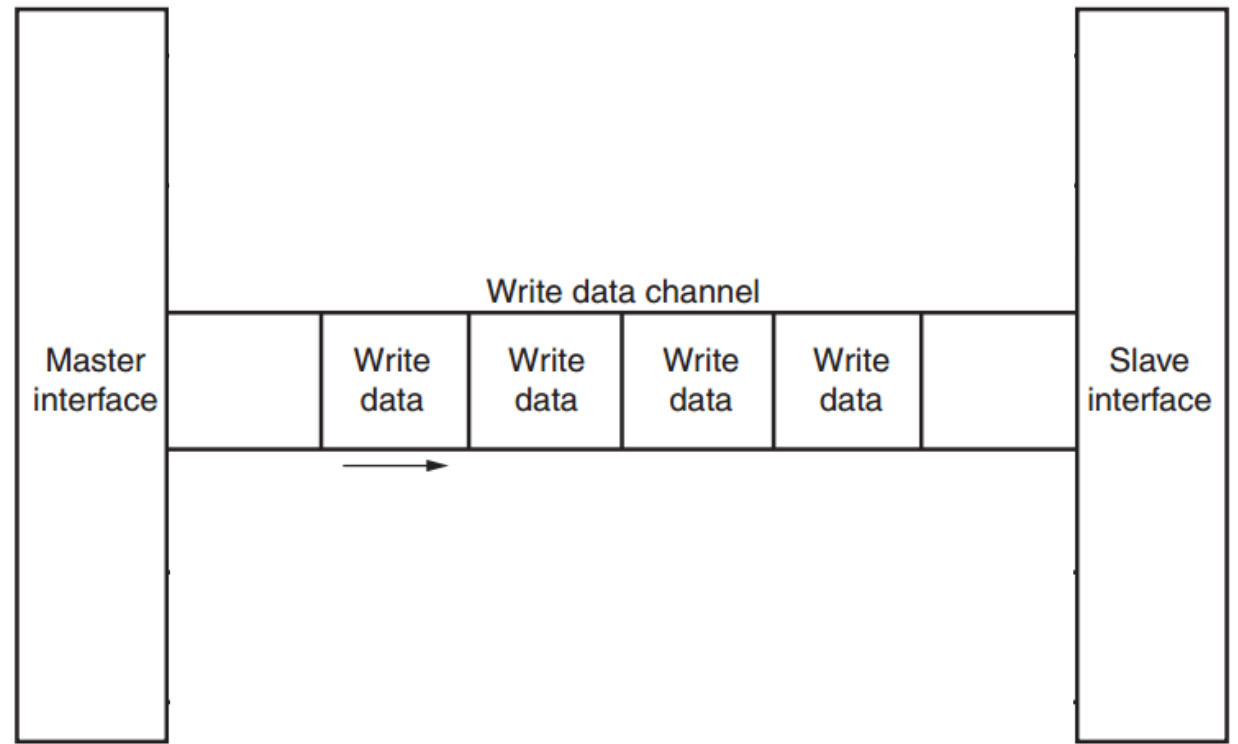
- ❖ A source is **NOT** permitted to wait until **READY** is asserted before asserting **VALID**
- ❖ Once **VALID** is asserted it must remain asserted until the handshake occurs, at a rising clock edge at which **VALID** and **READY** are both asserted.
- ❖ A destination is permitted to wait for **VALID** to be asserted before asserting the corresponding **READY**.
- ❖ If **READY** is asserted, it is permitted to deassert **READY** before **VALID** is asserted.

AXI Stream

- ❖ The AXI4-Stream protocol defines a single channel for transmission of streaming data (unlimited burst).



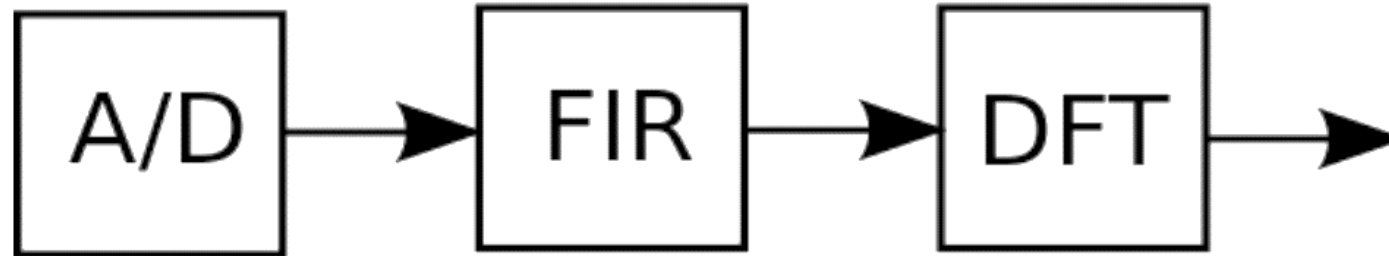
AXI Stream



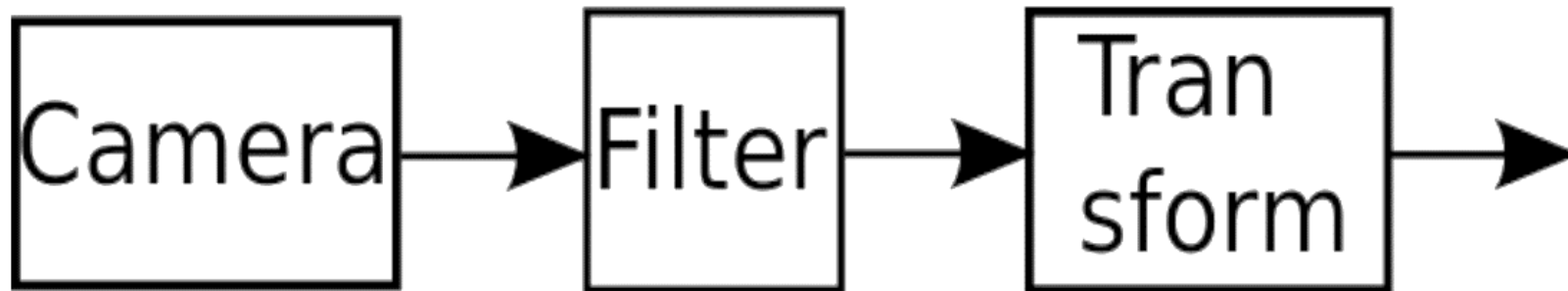
- ❖ The AXI4-Stream channel is modeled after the **Write Data channel** of the AXI4.
- ❖ Unlike AXI4, AXI4-Stream interfaces can burst an **unlimited** amount of data.

AXI Stream

Signal Processing

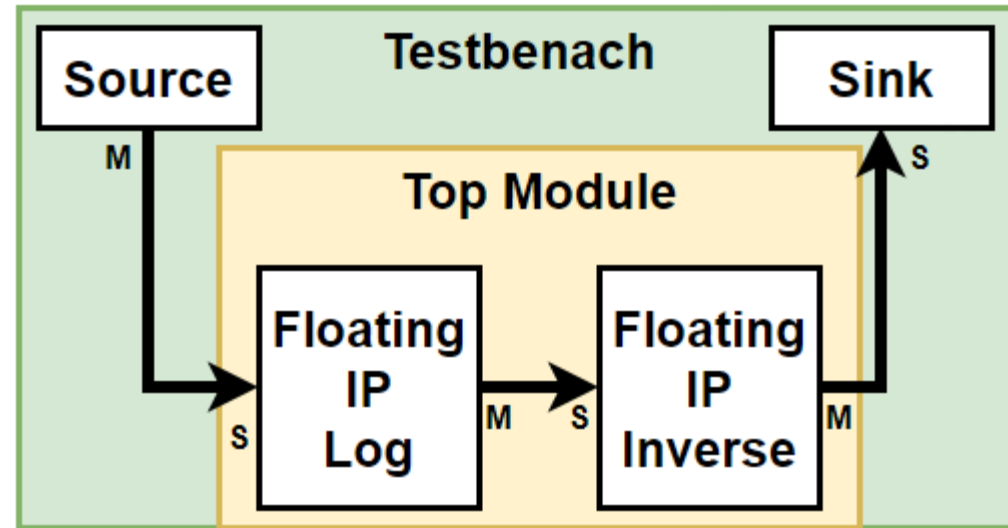


Video Processing



Lab

Proposed Approach



Locate the IP in Vivado

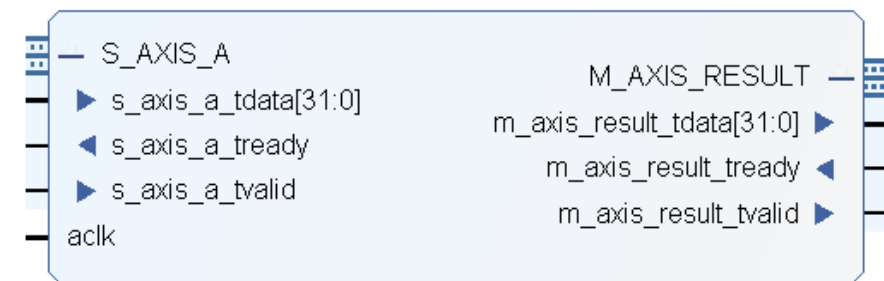
Project Summary x IP Catalog x

Cores | Interfaces

Search: (2 matches)

Name	AXI4	Status	License	VLNV
▼ Vivado Repository				
▼ Math Functions				
▼ Floating Point				
⚡ Floating-point	AXI4-Stream	Production	Included	xilinx.com:ip:floating_point:7.1

Log Operation



Customize IP

Floating-point (7.1)

[Documentation](#) [IP Location](#) [Switch to Defaults](#)

IP Symbol **Implementation Details**

☐ Show disabled ports

Component Name FP_log

Operation Selection **Precision of Inputs** **Optimizations** **Interface Options**

Please select from the following functions:

Operation Selection

- ☐ Absolute Value
- ☐ Accumulator
- ☐ Add/Subtract
- ☐ Compare
- ☐ Divide
- ☐ Exponential
- ☐ Fixed-to-float
- ☐ Float-to-fixed
- ☐ Float-to-float
- ☐ Fused Multiply-Add
- ☒ Logarithm
- ☐ Multiply
- ☐ Reciprocal
- ☐ Reciprocal Square Root
- ☐ Square-root

Logarithm operation selected. **RESULT = ln(A)**

Block diagram of the FP_log component. On the left, the input port S_AXIS_A has three data ports: s_axis_a_tdata[31:0] (output), s_axis_a_tready (input), and s_axis_a_tvalid (input). A clock port aclk is also on the left. On the right, the output port M_AXIS_RESULT has three data ports: m_axis_result_tdata[31:0] (input), m_axis_result_tready (output), and m_axis_result_tvalid (output).

Reciprocal operation

Customize IP

Floating-point (7.1)

[Documentation](#) [IP Location](#) [Switch to Defaults](#)

IP Symbol **Implementation Details**

☐ Show disabled ports

Component Name FP_recp

Operation Selection **Precision of Input**

Please select from the following functions:


Operation Selection

- ☐ Absolute Value
- ☐ Accumulator
- ☐ Add/Subtract
- ☐ Compare
- ☐ Divide
- ☐ Exponential
- ☐ Fixed-to-float
- ☐ Float-to-fixed
- ☐ Float-to-float
- ☐ Fused Multiply-Add
- ☐ Logarithm
- ☐ Multiply
- ☒ Reciprocal
- ☐ Reciprocal Square Root
- ☐ Square-root

Diagram:

Diagram showing a block with inputs **S_AXIS_A** and **ack**, and output **M_AXIS_RESULT**.

Top Module

 Define Module ✕

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Module name:

I/O Port Definitions

+

-

↑

↓

Port Name	Direction	Bus	MSB	LSB
Clk_100M	input	<input type="checkbox"/>	0	0
S_data	input	<input checked="" type="checkbox"/>	31	0
S_valid	input	<input type="checkbox"/>	0	0
S_ready	output	<input type="checkbox"/>	0	0
M_data	output	<input checked="" type="checkbox"/>	31	0
M_valid	output	<input type="checkbox"/>	0	0
M_ready	input	<input type="checkbox"/>	0	0

?

OK

Cancel

```
module Top_arith(  
    input Clk_100M,  
    input [31:0] S_data,  
    input S_valid,  
    output S_ready,  
    output [31:0] M_data,  
    output M_valid,  
    input M_ready  
);  
  
wire int_valid, int_ready;  
wire [31:0] int_data;  
FP_log l1 (  
    .aclk(Clk_100M),  
    .s_axis_a_tvalid(S_valid),  
    .s_axis_a_tready(S_ready),  
    .s_axis_a_tdata(S_data),  
    .m_axis_result_tvalid(int_valid),  
    .m_axis_result_tready(int_ready),  
    .m_axis_result_tdata(int_data)  
);  
  
FP_recpr1 (  
    .aclk(Clk_100M),  
    .s_axis_a_tvalid(int_valid),  
    .s_axis_a_tready(int_ready),  
    .s_axis_a_tdata(int_data),  
    .m_axis_result_tvalid(M_valid),  
    .m_axis_result_tready(M_ready),  
    .m_axis_result_tdata(M_data)  
);  
  
endmodule
```

Testbench

- Floating point converter: <https://www.h-schmidt.net/FloatConverter/IEEE754.html>

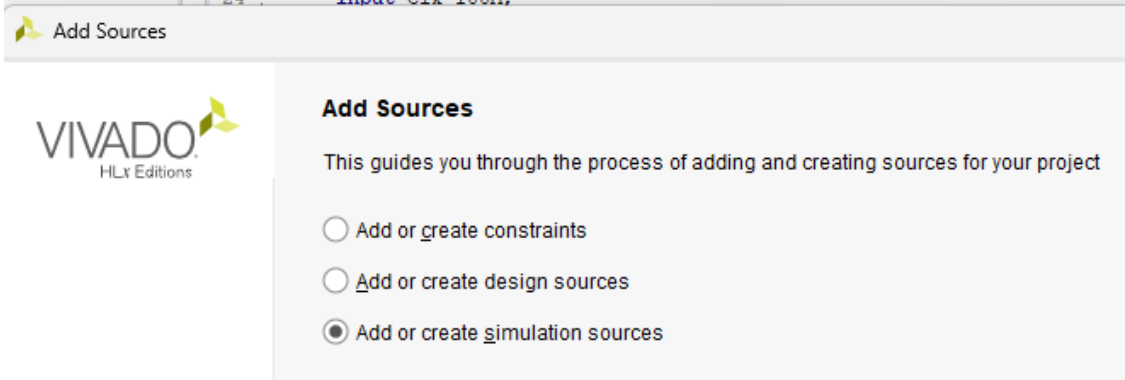
```
module arith_test(
);

reg Clk_100M, S_valid, M_ready;
reg [31:0] S_data;
wire M_valid, S_ready;
wire [31:0] M_data;
Top_arith t1 (.Clk_100M(Clk_100M), .S_data(S_data), .S_valid(S_valid), .S_ready(S_ready), .M_data(M_data), .M_valid(M_valid), .M_ready(M_ready));

initial begin
    Clk_100M = 0;
    S_valid = 0;
    S_data = 0;
    M_ready = 1;
end

always
    #5 Clk_100M = ~Clk_100M;

initial begin
    S_data = 32'b01000000101000000000000000000000;
    S_valid = 1;
    while (S_ready == 0)
        S_valid = 1;
    #5 S_valid = 0;
    @(posedge M_valid);
    #10 $stop;
end
```



Simulations

