AXI Verification IP

Description

📖 This guideline helps you to **validate the operation of AXI4 Transaction Master Controller using AXI Verification IP** in Vivado.

AXI Verification IP verifies connectivity and basic functionality of AXI masters and AXI slaves with the custom RTL design flow. **Setting AXI VIP as AXI slave** will validate the operation of the custom **AXI master- AXI master controller**- by checking if it is working properly without any violation of AXI4 Transaction.

In this guideline, the AXI4 Transaction Master Controller is assumed to be operated with **256 bit data, and 64 bit address**. Adjust the controller and VIP data width as you needed.

Environment Setting and Requirements

- **Vivado (ver. 2021.2)**
- **M_AXI_256 master controller : DUT (Design Under Test)**
    - `M_AXI_256_READ`
    - `M_AXI_256_WRITE`
- **basic packages**
    - `base_pkg`
    - `axi_pkg / aximm_pkg`

  The `aximm_pkg` should define the AXI transaction between the **virtual AXI master-slave ports** imitating the actual AXI transaction between the IPs.
- **AXI Memory Mapped(aximm) interface**
    - `axi_interface`

  The `axi_interface` should define `aximm` signals as a grouped block.

📖 **SyetemVerilog Interface** : All related signals are grouped together to form an interface block so that the same `interface` can be re-used for other projects.

Step-By-Step Implementation

1. Create Vivado Project with required sources

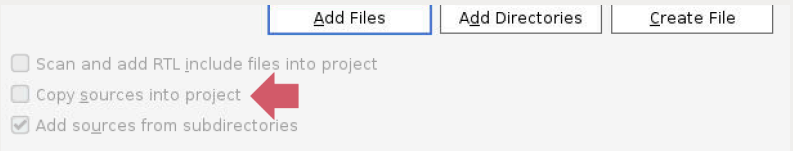
- **Default**
  - Include or create all the requirement files in the project.

▼ **If you are using the Git repository**

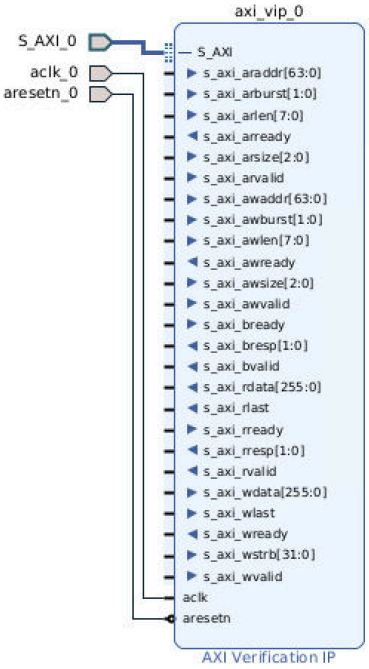
- Clone the existing git repository in your local environment.
- Create the Vivado project within the cloned repository.
- Include the following Github repository path into the project:
  - `AXI/M_AXI_256/rtl`
  - `interface`
  - `pkg`

💡 **Make sure to UNCHECK the 'Copy sources into project' when using Git.**

This action will sync the modifications of the local repository files in Visual Studio Code with the Vivado project.



2. Create Block Design



example block design

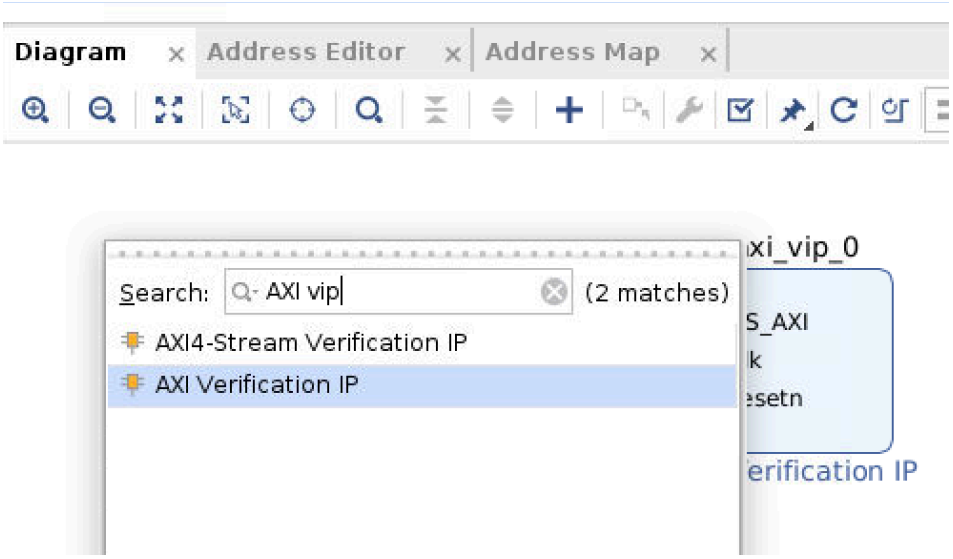
AXI Verification IP has 3 different interface mode: **Master, Pass-Through, and Slave**

To verify the AXI design, **set the AXI VIP as a pair of your custom design.** For example, if your design is AXI master, set AXI VIP as slave. If the DUT is AXI slave, set AXI VIP as master. This will generate the AXI master-slave pair that enables the AXI transaction.

The example block design above shows an AXI VIP set as AXI slave to verify the custom AXI Master. AXI slave VIP consists three ports: **S_AXI, the axi slave port, clock, and reset**. To connect the AXI VIP block design with your DUT, you should interconnect them through **an interface port,** which will be combined with the **aximm port** in your design via testbench. **S_AXI_0** in the above diagram is an example - it is **a slave interface port** that is connected to the **aximm port** in the master controller.

1) Import AXI verification IP in block design

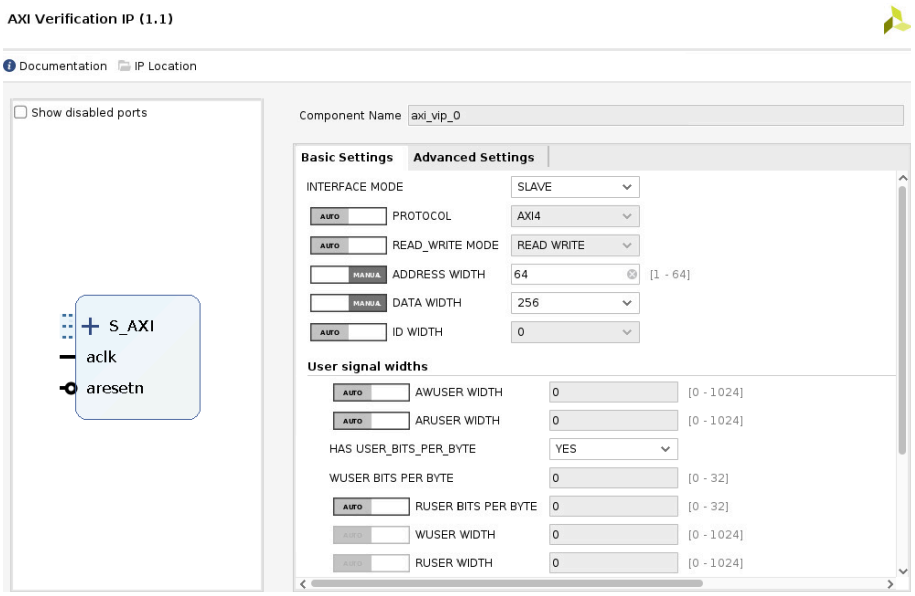
- Create block design → Specify block design name → Open block design → press '**+**' (Add IP) → search 'AXI vip' → select **AXI Verification IP**



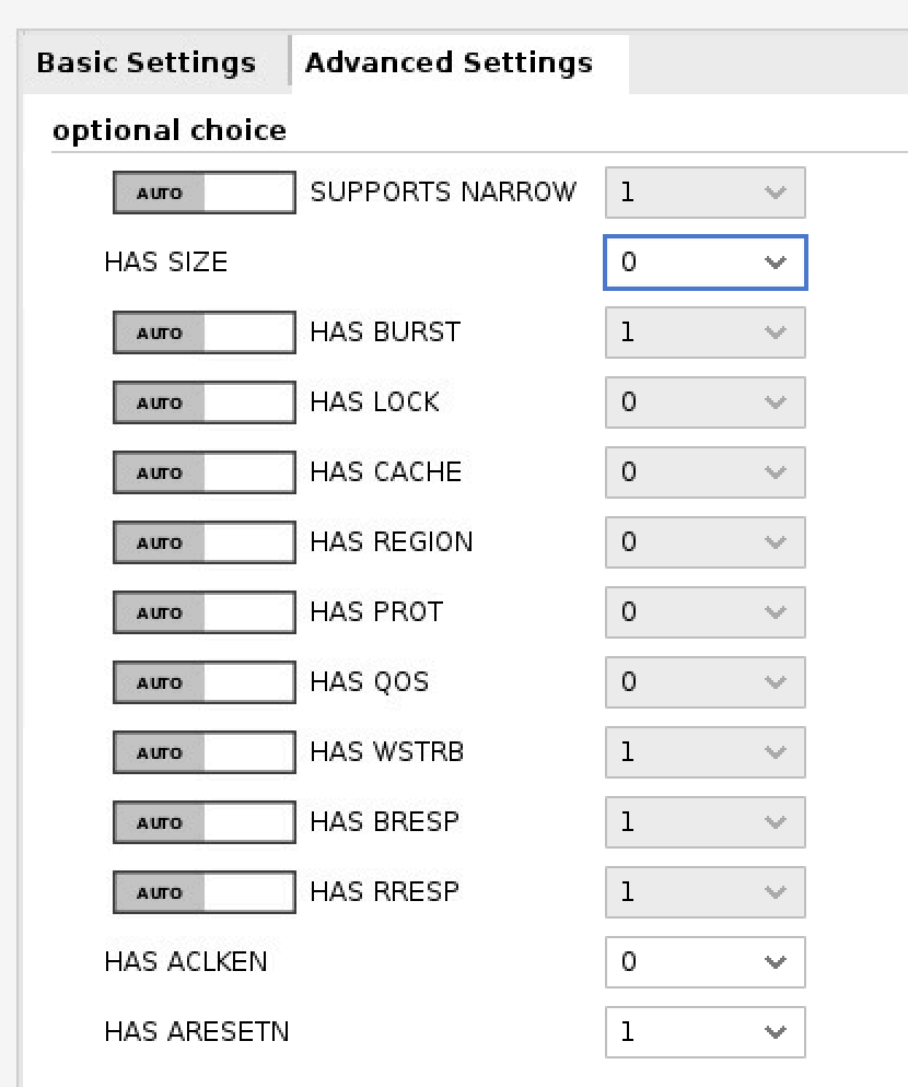
Import AXI Verification IP in the block design

2) Set the VIP properties

- double click the added VIP → move on to the setting tab
- set the interface mode as **SLAVE**
- set address width / data width. In this example, **address width = 64 bit / data width = 256 bit**
- adjust other ports according to your design



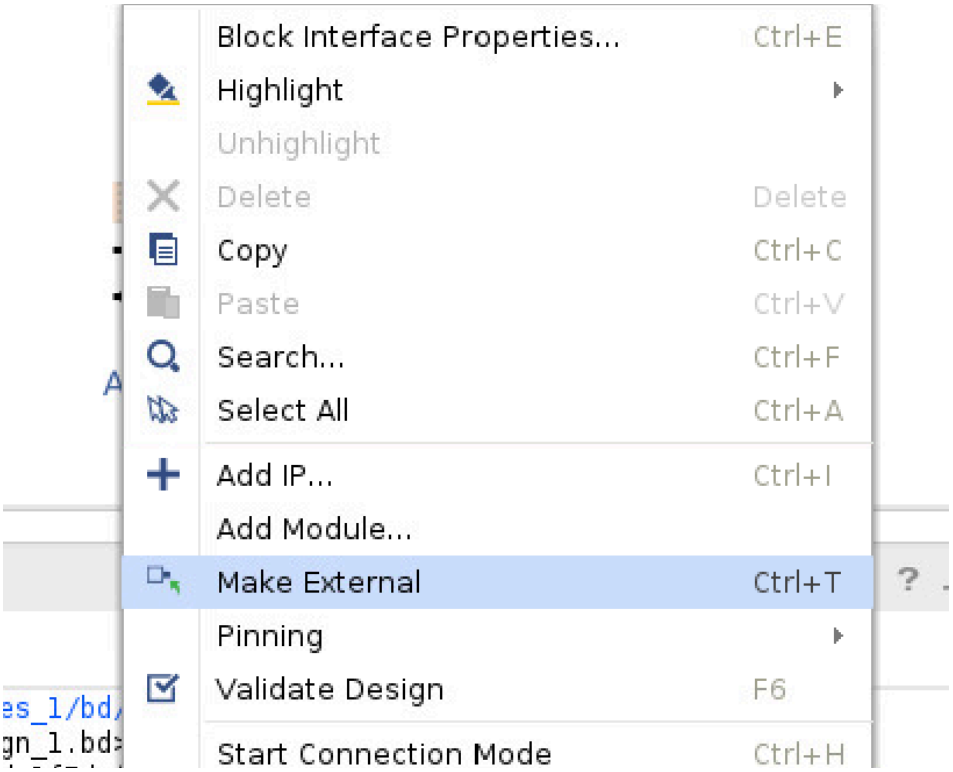
set vip interface mode as slave



disable unused ports. Adjust them according to your master controller design.

3) Create external connection

- Select the `S_AXI`, `aclk`, and `aresetn` port in VIP → click on right button → select **'Make External'**



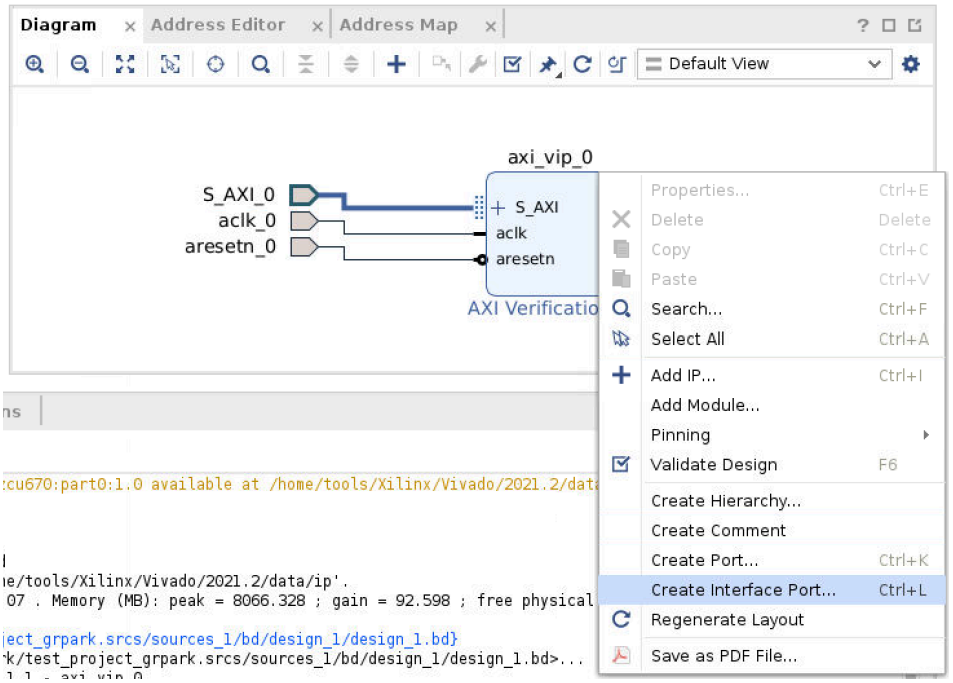
Make External menu will automatically create the external ports used to interconnect the VIP and the controller

- check the each created port and set properties according to your controller.
  - S_AXI
    - **address width: 64 bit**
    - **data width: 256 bit**
    - **max burst length: 128 ((256 bit / 8) * 128 = 4096 byte, AXI4 burst boundary)**
  - aclk
    - **Clock frequency: 100 MHz**
    - set '**Associated Busif**' as **interface port in 3)**, and set '**Associated Reset**' as **reset** port
  - aresetn
    - set as **'Active Low'**

▼ If you wish to create the ports individually

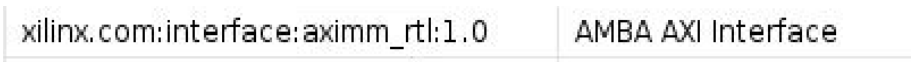
3) Create and connect interface port with the VIP

- press right button in the opened block design → **Create Interface Port**



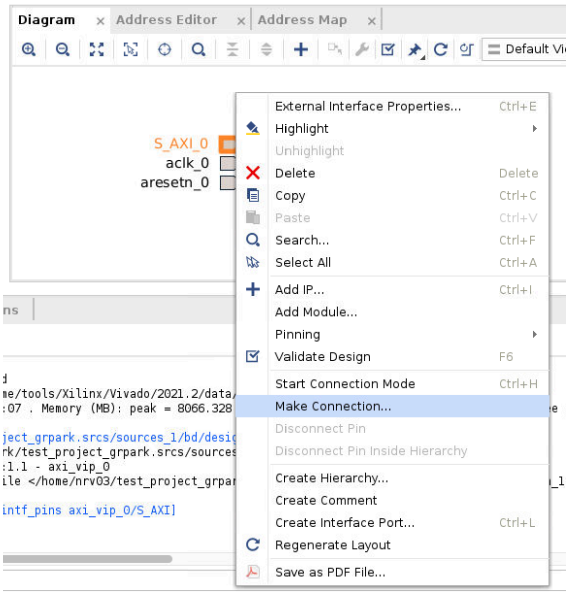
select 'Create Interface Port' in the block design

- set the interface port as **slave**, and select its type as **AMBA AXI interface**

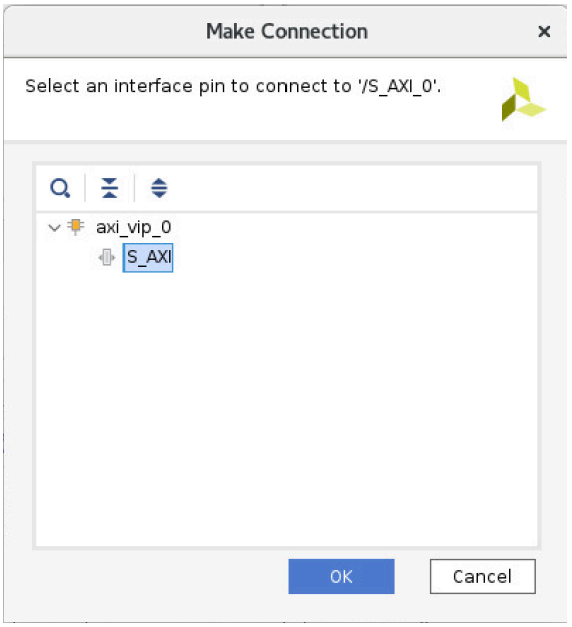


interface port type: AMBA AXI interface

- set **address width / data width / max burst length** of the interface port according to your design. In this example, details are as following:
  - **address width = 64 bit**
  - **data width = 256 bit**
  - **max burst length = 128 ((256 bit / 8) * 128 = 4096 byte, AXI4 burst boundary)**
- connect the interface port with S_AXI port in AXI VIP. You can just simply drag it, or use 'Make Connection' menu after selecting the created interface port.



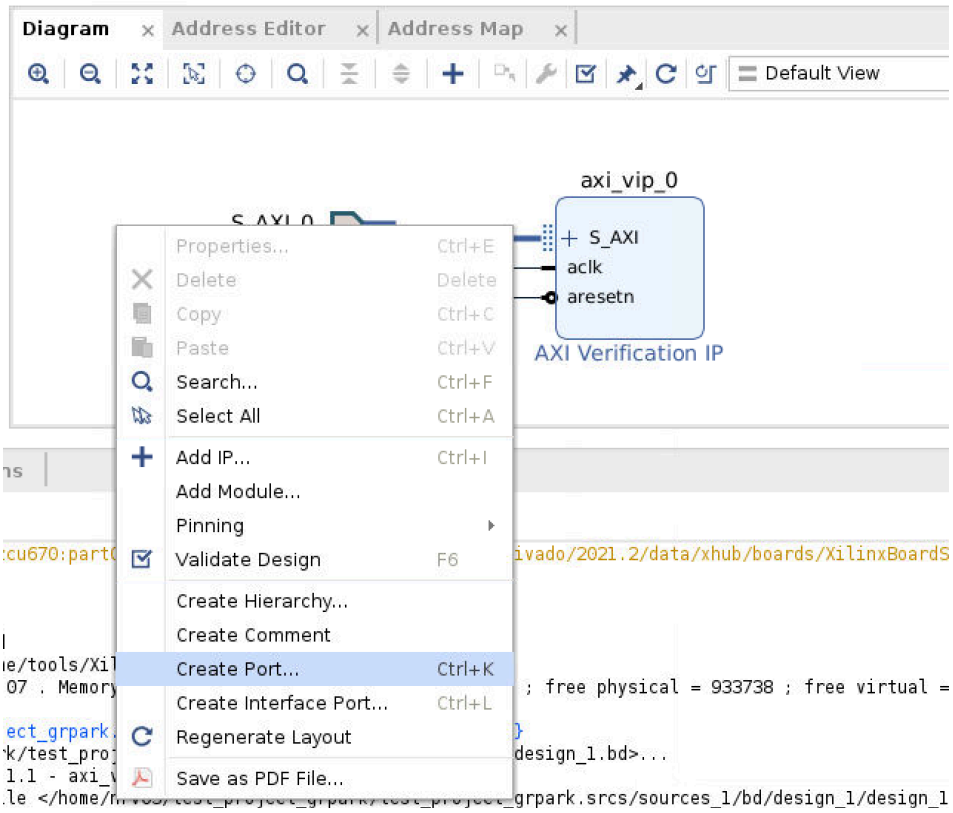
Select interface port → Press 'Make Connection'



Select the S_AXI port in AXI VIP

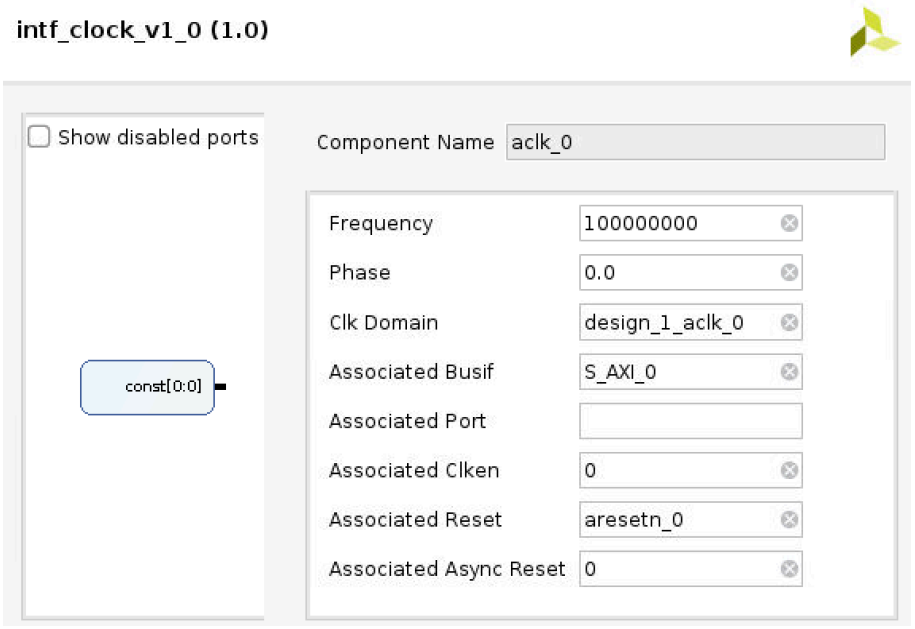
4) Create and connect clock and reset port with the VIP

- press right button in the block design → **create port**



clk

- set port type as **clk**
- **Clock frequency: 100 MHz**
- set '**Associated Busif'** as **interface port in 3)**, and set '**Associated Reset**' as **reset** port



clock port setting

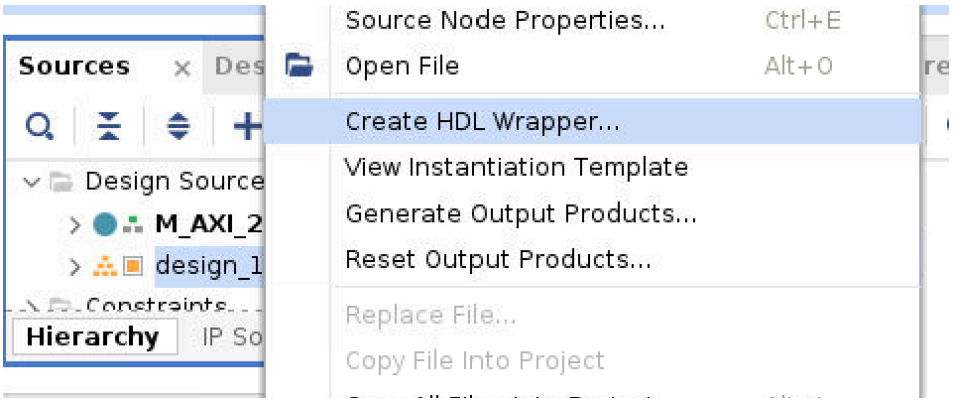
reset

- set port type as **reset**
- set reset type as **active low**

- **connect generated clock port with aclk, and reset with aresetn port in AXI VIP**

3. Create Design wrapper

- Create the wrapper that instantiates the block design.
- **Sources → select the block design → press right button → create HDL wrapper → select 'let Vivado manage wrapper and auto-update' → wrapper will automatically created by Vivado**



creating the HDL Wrapper for the design

- This wrapper will be used to connect the block design and the testbench.
- **BE AWARE of the data width of the each port! Carefully check the generated wrapper and modify if there are some mismatch with your design.**
- **Make sure you disable unused ports in AXI VIP and S_AXI_0 interface**

example wrapper

▼ design_1_wrapper

```
//Copyright 1986-2021 Xilinx, Inc. All Rights Reserved.
//----------------------------------------------------------------------------------
//Tool Version: Vivado v.2021.2 (lin64) Build 3367213 Tue Oct 19 02:47:39 MDT 2021
//Date        : Thu Dec 26 16:55:01 2024
//Host        : nrv-eda01 running 64-bit unknown
//Command     : generate_target design_1_wrapper.bd
//Design      : design_1_wrapper
//Purpose     : IP block netlist
//----------------------------------------------------------------------------------
`timescale 1 ps / 1 ps

module design_1_wrapper
   (S_AXI_0_araddr,
    S_AXI_0_arburst,
    S_AXI_0_arlen,
    S_AXI_0_arready,
    S_AXI_0_arsize,
    S_AXI_0_arvalid,
    S_AXI_0_awaddr,
    S_AXI_0_awburst,
    S_AXI_0_awlen,
    S_AXI_0_awready,
    S_AXI_0_awsize,
    S_AXI_0_awvalid,
    S_AXI_0_bready,
    S_AXI_0_bresp,
    S_AXI_0_bvalid,
    S_AXI_0_rdata,
    S_AXI_0_rlast,
    S_AXI_0_rready,
    S_AXI_0_rresp,
    S_AXI_0_rvalid,
    S_AXI_0_wdata,
    S_AXI_0_wlast,
    S_AXI_0_wready,
    S_AXI_0_wstrb,
    S_AXI_0_wvalid,
    aclk_0,
    aresetn_0);
  input [63:0]S_AXI_0_araddr;
  input [1:0]S_AXI_0_arburst;
  input [7:0]S_AXI_0_arlen;
  output S_AXI_0_arready;
  input [2:0]S_AXI_0_arsize;
  input S_AXI_0_arvalid;
  input [63:0]S_AXI_0_awaddr;
  input [1:0]S_AXI_0_awburst;
  input [7:0]S_AXI_0_awlen;
  output S_AXI_0_awready;
```

```
input [2:0]S_AXI_0_awsize;
input S_AXI_0_awvalid;
input S_AXI_0_bready;
output [1:0]S_AXI_0_bresp;
output S_AXI_0_bvalid;
output [255:0]S_AXI_0_rdata;
output S_AXI_0_rlast;
input S_AXI_0_rready;
output [1:0]S_AXI_0_rresp;
output S_AXI_0_rvalid;
input [255:0]S_AXI_0_wdata;
input S_AXI_0_wlast;
output S_AXI_0_wready;
input [31:0]S_AXI_0_wstrb;
input S_AXI_0_wvalid;
input aclk_0;
input aresetn_0;

wire [63:0]S_AXI_0_araddr;
wire [1:0]S_AXI_0_arburst;
wire [7:0]S_AXI_0_arlen;
wire S_AXI_0_arready;
wire [2:0]S_AXI_0_arsize;
wire S_AXI_0_arvalid;
wire [63:0]S_AXI_0_awaddr;
wire [1:0]S_AXI_0_awburst;
wire [7:0]S_AXI_0_awlen;
wire S_AXI_0_awready;
wire [2:0]S_AXI_0_awsize;
wire S_AXI_0_awvalid;
wire S_AXI_0_bready;
wire [1:0]S_AXI_0_bresp;
wire S_AXI_0_bvalid;
wire [255:0]S_AXI_0_rdata;
wire S_AXI_0_rlast;
wire S_AXI_0_rready;
wire [1:0]S_AXI_0_rresp;
wire S_AXI_0_rvalid;
wire [255:0]S_AXI_0_wdata;
wire S_AXI_0_wlast;
wire S_AXI_0_wready;
wire [31:0]S_AXI_0_wstrb;
wire S_AXI_0_wvalid;
wire aclk_0;
wire aresetn_0;

design_1 design_1_i
    (.S_AXI_0_araddr(S_AXI_0_araddr),
     .S_AXI_0_arburst(S_AXI_0_arburst),
     .S_AXI_0_arlen(S_AXI_0_arlen),
     .S_AXI_0_arready(S_AXI_0_arready),
     .S_AXI_0_arsize(S_AXI_0_arsize),
     .S_AXI_0_arvalid(S_AXI_0_arvalid),
     .S_AXI_0_awaddr(S_AXI_0_awaddr),
     .S_AXI_0_awburst(S_AXI_0_awburst),
     .S_AXI_0_awlen(S_AXI_0_awlen),
     .S_AXI_0_awready(S_AXI_0_awready),
     .S_AXI_0_awsize(S_AXI_0_awsize),
```

```
        .S_AXI_0_awvalid(S_AXI_0_awvalid),
        .S_AXI_0_bready(S_AXI_0_bready),
        .S_AXI_0_bresp(S_AXI_0_bresp),
        .S_AXI_0_bvalid(S_AXI_0_bvalid),
        .S_AXI_0_rdata(S_AXI_0_rdata),
        .S_AXI_0_rlast(S_AXI_0_rlast),
        .S_AXI_0_rready(S_AXI_0_rready),
        .S_AXI_0_rresp(S_AXI_0_rresp),
        .S_AXI_0_rvalid(S_AXI_0_rvalid),
        .S_AXI_0_wdata(S_AXI_0_wdata),
        .S_AXI_0_wlast(S_AXI_0_wlast),
        .S_AXI_0_wready(S_AXI_0_wready),
        .S_AXI_0_wstrb(S_AXI_0_wstrb),
        .S_AXI_0_wvalid(S_AXI_0_wvalid),
        .aclk_0(aclk_0),
        .aresetn_0(aresetn_0));
endmodule
```

4. Implement testbench

1) Import packages

```
import aximm_pkg::*;
import axi_vip_pkg::*;
import design_1_axi_vip_0_0_pkg::*;
```

- `aximm_pkg` : custom axi memory mapped package
- `axi_vip_pkg` : package for AXI VIP usage
- `design_1_axi_vip_0_0_pkg` : package for the created block design

2) Declare slave agent

```
design_1_axi_vip_0_0_slv_mem_t slv_mem_agent;
```

- declare the created block design as AXI slave agent

3) Instantiate master axi controller

```
aximm __m_256_axi ();

// DUT
M_AXI_256_M_AXI DUT (
   .axi_read_start_addr(axi_read_start_addr),
   .axi_write_start_addr(axi_write_start_addr),
   .axi_read_length(axi_read_length),
   .axi_write_length(axi_write_length),
   .init_read(init_read),
   .init_write(init_write),
   .writes_done(writes_done),
   .reads_done(reads_done),
   .USER_RDATA(USER_RDATA),
   .USER_RDATA_VLD(USER_RDATA_VLD),
   .USER_WDATA(USER_WDATA),
   .USER_WDATA_VLD(USER_WDATA_VLD),
   .USER_FIFO_EMPTY(USER_FIFO_EMPTY),
```

```
    .M_AXI_ACLK(clk),
    .M_AXI_ARESETN(rstn),
    .__m_256_axi(__m_256_axi)
);
```

- make sure to create aximm port and connect them in `__m_256_axi()` port in master controller

4) Instantiate design wrapper

```
design_1_wrapper design_1_wrapper_0 (
        .aclk_0(clk),
        .aresetn_0(rstn),
        .S_AXI_0_awaddr(__m_256_axi.AWADDR),
        .S_AXI_0_awlen(__m_256_axi.AWLEN),
        .S_AXI_0_awsize(__m_256_axi.AWSIZE),
        .S_AXI_0_awburst(__m_256_axi.AWBURST),
        .S_AXI_0_awvalid(__m_256_axi.AWVALID),
        .S_AXI_0_awready(__m_256_axi.AWREADY),
        .S_AXI_0_wdata(__m_256_axi.WDATA),
        .S_AXI_0_wstrb(__m_256_axi.WSTRB),
        .S_AXI_0_wlast(__m_256_axi.WLAST),
        .S_AXI_0_wvalid(__m_256_axi.WVALID),
        .S_AXI_0_wready(__m_256_axi.WREADY),
        // .S_AXI_0_bid(__m_256_axi.BID),
        .S_AXI_0_bresp(__m_256_axi.BRESP),
        .S_AXI_0_bvalid(__m_256_axi.BVALID),
        .S_AXI_0_bready(__m_256_axi.BREADY),
        .S_AXI_0_araddr(__m_256_axi.ARADDR),
        .S_AXI_0_arlen(__m_256_axi.ARLEN),
        .S_AXI_0_arsize(__m_256_axi.ARSIZE),
        .S_AXI_0_arburst(__m_256_axi.ARBURST),
        .S_AXI_0_arvalid(__m_256_axi.ARVALID),
        .S_AXI_0_arready(__m_256_axi.ARREADY),
        // .S_AXI_0_rid(__m_256_axi.RID),
        .S_AXI_0_rdata(__m_256_axi.RDATA),
        .S_AXI_0_rresp(__m_256_axi.RRESP),
        .S_AXI_0_rlast(__m_256_axi.RLAST),
        .S_AXI_0_rvalid(__m_256_axi.RVALID),
        .S_AXI_0_rready(__m_256_axi.RREADY)
    );
```

- connect the wrapper with the ports in __m_256_axi() interface port that was defined in **3)**

5) Start the VIP agent

```
initial begin
    slv_mem_agent = new(
      "my VIP agent",
      M_AXI_tb.design_1_wrapper_0.design_1_i.axi_vip_0.inst.IF
  );
  slv_mem_agent.set_agent_tag("Slave VIP");
  slv_mem_agent.set_verbosity(400);
  slv_mem_agent.start_slave();
end
```

- Create a new agent and pass the hierarchy path of IF correctly into the new function
    - Instance Name: `<tb_name>.<wrapper_name>.<block_design_name_in_wrapper>.<vip_name>.inst.IF`

- Set a tag for agents for easy debug (optional)
- Set print out verbosity level for the agent
- Start the agent

6) Run task to start AXI transaction

- Create the task to run the AXI master controller and start the AXI transaction.
- **example task (1): test_write writes the given length of a data in the given length**

```
// example writing task
task automatic test_write(
    ref logic clk, ref logic [ADDR_WIDTH-1:0] axi_write_start_addr,
    ref logic [31:0] axi_write_length, ref logic init_write,
    ref logic writes_done, input logic [ADDR_WIDTH-1:0] addr,
    input logic [31:0] length);

    @(posedge clk);
    axi_write_start_addr = addr;
    axi_write_length = length;

    init_write = 0;
    #(10 * CLK_PERIOD);
    @(posedge clk);
    init_write = 1; // start the AXI transaction by turning on the master controller
    USER_WDATA = 0;
    @(posedge clk);
    init_write = 0;

    wait (writes_done);
    $display("writes_done!!!!!!!!!!!!!!!!!!! = %0d", writes_done);
endtask

// Data define
always @(posedge clk) begin
    if (!rstn) begin
        USER_WDATA <= 0;
    end else begin
        if (USER_WDATA_VLD) begin
            USER_WDATA <= USER_WDATA + 1;
        end
    end
end
```

- Additional) If read / write tasks are already defined in `aximm_pkg`, you can also create task by using them instead of defining the data manually in the testbench.

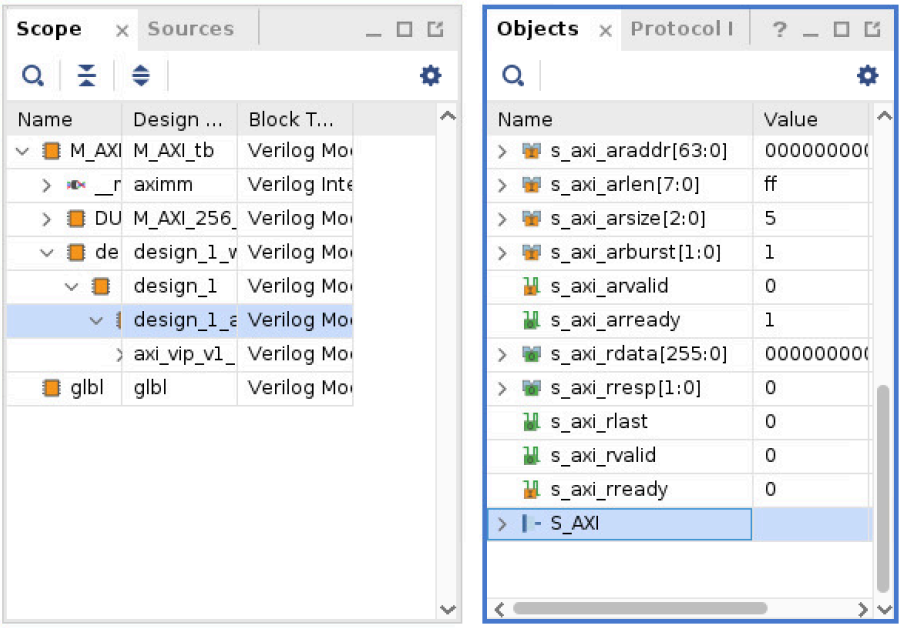
  example task(2): test_write using the task 'run()' in aximm_pkg

```
// virtual instance define
virtual_axi_mm axi_mm_inst;

aximm slave_if ();

// instantiate the virtual instance
initial begin
    axi_mm_inst = new();
```

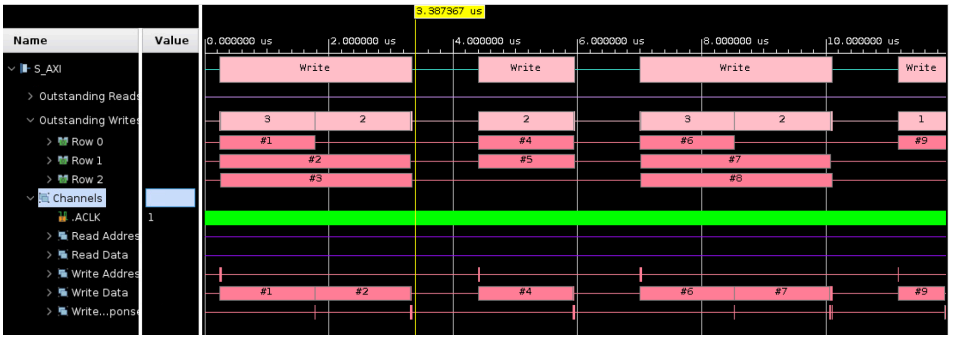
```
    axi_mm_inst.__aximm = slave_if;
end

// example task using the pre-defined tasks in the aximm_pkg
task automatic test_write(
    ref logic clk, ref logic [ADDR_WIDTH-1:0] axi_write_start_addr,
    ref logic [31:0] axi_write_length, ref logic init_write,
    ref logic writes_done, input logic [ADDR_WIDTH-1:0] addr,
    input logic [31:0] length);

    @(posedge clk);
    axi_write_start_addr = addr;
    axi_write_length = length;

    axi_mm_inst.init_arr_random();

    #(10 * CLK_PERIOD);
    @(posedge clk);
    init_write = 1; // start the AXI transaction by turning on the master controller
    @(posedge clk);
    init_write = 0;

    for (i = 0; i < length; i = i + 1) begin
        axi_mm_inst.get_data(USER_WDATA, i);
    end

    initial begin
        axi_mm_inst.run(); //run AXI transaction with the tasks in aximm_pkg
    end

    wait (writes_done);
    $display("writes_done!!!!!!!!!!!!!!!!!!! = %0d", writes_done);
    $finish;
endtask
```

7) Run Simulation

- check **S_AXI port** of block design in waveform for proper display.
- Find the port in the scope, select it and drag it to the waveform.



S_AXI port is an interface port in <design_name>_axi_vip_0_0



example waveform of S_AXI port

- The **Read / Write channels** shows the details of the AXI transaction. For example, the example waveform above shows the write channels containing the following:
  1. The transaction starts by setting the address on the **Write Address Channel**
  2. Then a burst of data is sent on the **Write Data channel**
  3. Finally, the slave responds if the write was successful on the **Write Response Channel**
     - You can also check the specific data flow by clicking on each channel wave.
- check tcl console if there's no fatal error occurred. If your DUT (in this case, AXI master controller) **violates the AXI4 transaction**, error message and the detail will appear as **'FATAL: '** in the tcl console.
- This way, you can check whether the controller runs properly. **If the read / write tasks are done without errors, your design is validated!**

8) Create and run tcl file for future usage

create tcl file

- put command in tcl console

```
write_bd_tcl <file path>/<file name>.tcl  // save block design as tcl file
```

run existing tcl file

- put command in tcl console

```
cd <tcl_file_saved_path>      // move to tcl file path
source <tcl_file_name>.tcl    // run tcl file - this will load the block design as saved
```

Example Testbench Code

▼ M_AXI_tb.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2023/08/25 12:51:33
// Design Name:
// Module Name:
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module M_AXI_tb
    import aximm_pkg::*;
    import axi_vip_pkg::*;
    import design_1_axi_vip_0_0_pkg::*;
;

    design_1_axi_vip_0_0_slv_mem_t slv_mem_agent;

    logic clk;
    logic rstn;

    // Write - Input
    logic [ADDR_WIDTH-1:0] axi_write_start_addr;
    logic [31:0] axi_write_length;
    logic init_write;
    logic [DATA_WIDTH-1 : 0] USER_WDATA;
    logic USER_WDATA_VLD;
    // Write - Output
    logic writes_done;

    // Read - Input
```

```
logic [ADDR_WIDTH-1:0] axi_read_start_addr;
logic [31:0] axi_read_length;
logic init_read;
// Read - Output
wire [DATA_WIDTH-1 : 0] USER_RDATA;
wire USER_RDATA_VLD;
wire reads_done;

// Debug
logic USER_FIFO_EMPTY;

aximm __m_256_axi ();

// DUT
M_AXI_256_M_AXI DUT (
    .axi_read_start_addr(axi_read_start_addr),
    .axi_write_start_addr(axi_write_start_addr),
    .axi_read_length(axi_read_length),
    .axi_write_length(axi_write_length),
    .init_read(init_read),
    .init_write(init_write),
    .writes_done(writes_done),
    .reads_done(reads_done),
    .USER_RDATA(USER_RDATA),
    .USER_RDATA_VLD(USER_RDATA_VLD),
    .USER_WDATA(USER_WDATA),
    .USER_WDATA_VLD(USER_WDATA_VLD),
    .USER_FIFO_EMPTY(USER_FIFO_EMPTY),
    .M_AXI_ACLK(clk),
    .M_AXI_ARESETN(rstn),
    .__m_256_axi(__m_256_axi)
);

design_1_wrapper design_1_wrapper_0 (
    .aclk_0(clk),
    .aresetn_0(rstn),
    .S_AXI_0_awaddr(__m_256_axi.AWADDR),
    .S_AXI_0_awlen(__m_256_axi.AWLEN),
    .S_AXI_0_awsize(__m_256_axi.AWSIZE),
    .S_AXI_0_awburst(__m_256_axi.AWBURST),
    .S_AXI_0_awvalid(__m_256_axi.AWVALID),
    .S_AXI_0_awready(__m_256_axi.AWREADY),
    .S_AXI_0_wdata(__m_256_axi.WDATA),
    .S_AXI_0_wstrb(__m_256_axi.WSTRB),
    .S_AXI_0_wlast(__m_256_axi.WLAST),
    .S_AXI_0_wvalid(__m_256_axi.WVALID),
    .S_AXI_0_wready(__m_256_axi.WREADY),
    // .S_AXI_0_bid(__m_256_axi.BID),
    .S_AXI_0_bresp(__m_256_axi.BRESP),
    .S_AXI_0_bvalid(__m_256_axi.BVALID),
    .S_AXI_0_bready(__m_256_axi.BREADY),
    .S_AXI_0_araddr(__m_256_axi.ARADDR),
    .S_AXI_0_arlen(__m_256_axi.ARLEN),
    .S_AXI_0_arsize(__m_256_axi.ARSIZE),
    .S_AXI_0_arburst(__m_256_axi.ARBURST),
    .S_AXI_0_arvalid(__m_256_axi.ARVALID),
    .S_AXI_0_arready(__m_256_axi.ARREADY),
    // .S_AXI_0_rid(__m_256_axi.RID),
```

```
    .S_AXI_0_rdata(__m_256_axi.RDATA),
    .S_AXI_0_rresp(__m_256_axi.RRESP),
    .S_AXI_0_rlast(__m_256_axi.RLAST),
    .S_AXI_0_rvalid(__m_256_axi.RVALID),
    .S_AXI_0_rready(__m_256_axi.RREADY)
);

// Clock
parameter CLK_PERIOD = 5;  //100MHz
initial begin
    clk = 1'b0;
    #(4 * CLK_PERIOD) fork forever #(CLK_PERIOD / 2) clk = ~clk; join
end

task automatic test_read(
    ref logic clk, ref logic [ADDR_WIDTH-1:0] axi_read_start_addr,
    ref logic [31:0] axi_read_length, ref logic init_read,
    input logic [ADDR_WIDTH-1:0] addr, input logic [31:0] length);

    @(posedge clk);
    axi_read_start_addr = addr;
    axi_read_length = length;

    init_read = 0;
    #(10 * CLK_PERIOD);
    @(posedge clk);
    init_read = 1;
endtask

task automatic test_write(
    ref logic clk, ref logic [ADDR_WIDTH-1:0] axi_write_start_addr,
    ref logic [31:0] axi_write_length, ref logic init_write,
    ref logic writes_done, input logic [ADDR_WIDTH-1:0] addr,
    input logic [31:0] length);

    @(posedge clk);
    axi_write_start_addr = addr;
    axi_write_length = length;

    init_write = 0;
    #(10 * CLK_PERIOD);
    @(posedge clk);
    init_write = 1;
    USER_WDATA = 0;
    @(posedge clk);
    init_write = 0;

    wait (writes_done);
    $display("writes_done!!!!!!!!!!!!!!!!!!! = %0d", writes_done);
endtask

always @(posedge clk) begin
    if (!rstn) begin
        USER_WDATA <= 0;
    end else begin
        if (USER_WDATA_VLD) begin
            USER_WDATA <= USER_WDATA + 1;
        end
```

```
        end
    end

    integer i;
    // Test cases
    initial begin
        slv_mem_agent = new(
            "my VIP agent",
            M_AXI_tb.design_1_wrapper_0.design_1_i.axi_vip_0.inst.IF
        );
        slv_mem_agent.set_agent_tag("Slave VIP");
        slv_mem_agent.set_verbosity(400);
        slv_mem_agent.start_slave();

        rstn <= 0;
        axi_write_start_addr <= 0;
        axi_write_length <= 0;
        axi_read_start_addr <= 0;
        axi_read_length <= 0;
        init_write <= 0;
        init_read <= 0;
        USER_FIFO_EMPTY <= 0;

        #(32 * CLK_PERIOD);
        rstn <= 1;
        #(4 * CLK_PERIOD);


        test_write(clk, axi_write_start_addr, axi_write_length, init_write,
                writes_done, 0, 258);
        #1000;

        test_write(clk, axi_write_start_addr, axi_write_length, init_write,
                writes_done, 0, 129);
        #1000;

        test_write(clk, axi_write_start_addr, axi_write_length, init_write,
                writes_done, 32, 258);
        #1000;

        test_write(clk, axi_write_start_addr, axi_write_length, init_write,
                writes_done, 64, 63);

        $display("Test finished");
        $finish;
    end

endmodule
```

Reference

https://www.xilinx.com/products/intellectual-property/axi-vip.html

https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841614/Validating+a+master+AXI4+interface+using+the+Verification+IP+as+a

https://adaptivesupport.amd.com/s/article/1062002?language=en_US

https://github.com/chamchiking/npu/blob/main/M_AXI.tcl