

AXI VIP Tutorial 1

Introduction

The goal of this tutorial is to demonstrate how to use the Xilinx AXI Verification IP (VIP) to test and verify your custom AXI lite peripherals using the Xilinx Vivado Simulator. We will show you how to connect the VIP to your design and use it to write to and read from registers of the AXI GPIO IP.

Step 1: Create a new project and block diagram

1. Create a new project for the **Nexys 4** board.
2. Create a new block design and name it **design_1**.
3. Add and wire the **AXI Verification IP**, **AXI GPIO**, and **Constant** IPs to the diagram as shown in Figure 1 below. Make sure that the port names match the figure.

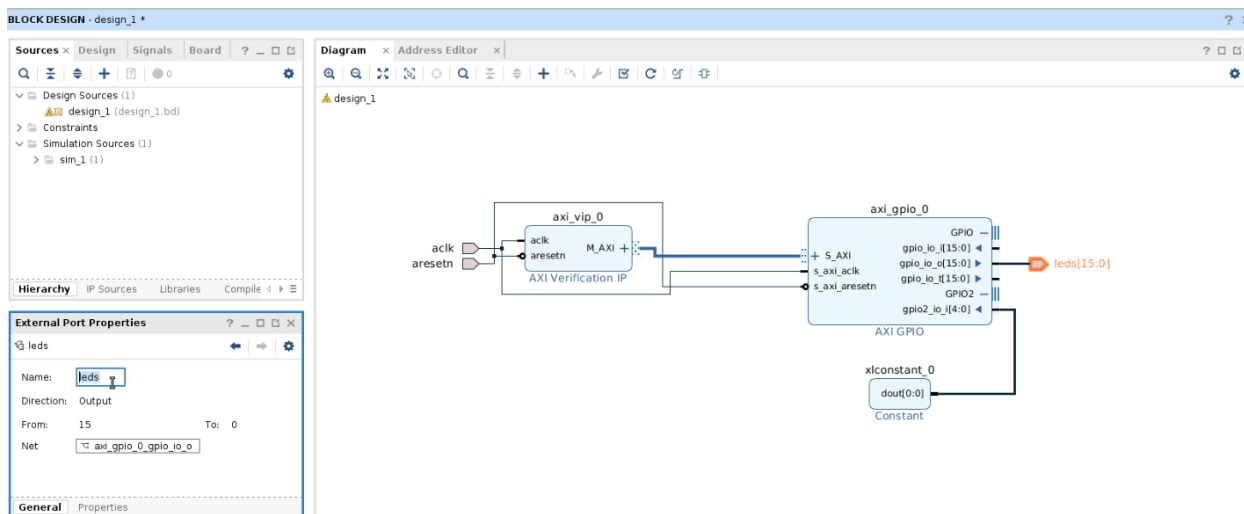


Figure 1: Block design.

4. Configure the **basic settings** tab of the **AXI VIP IP** as shown in the following figure. Leave the advance settings at their default values. This will set the VIP into AXI lite

mode.

The image shows a configuration window for an AXI VIP component. At the top, the 'Component Name' is 'axi_vip_0'. Below this are two tabs: 'Basic Settings' (selected) and 'Advanced Settings'. The 'Basic Settings' tab contains several configuration options, each with an 'Auto' button and a value field. The options are: 'INTERFACE MODE' (MASTER), 'PROTOCOL' (AXI4LITE), 'READ_WRITE MODE' (READ WRITE), 'ADDRESS WIDTH' (32, with a range of [12 - 64]), 'DATA WIDTH' (32), and 'ID WIDTH' (0). Below these is a section titled 'User signal widths' which contains eight more options, each with an 'Auto' button and a value field: 'AWUSER WIDTH' (0, range [0 - 0]), 'ARUSER WIDTH' (0, range [0 - 0]), 'HAS_USER_BITS_PER_BYTE' (NO), 'WUSER BITS PER BYTE' (0, range [0 - 0]), 'RUSER BITS PER BYTE' (0, range [0 - 0]), 'WUSER WIDTH' (0, range [0 - 0]), 'RUSER WIDTH' (0, range [0 - 0]), and 'BUSER WIDTH' (0, range [0 - 0]).

Setting	Value	Range
INTERFACE MODE	MASTER	
PROTOCOL	AXI4LITE	
READ_WRITE MODE	READ WRITE	
ADDRESS WIDTH	32	[12 - 64]
DATA WIDTH	32	
ID WIDTH	0	
User signal widths		
AWUSER WIDTH	0	[0 - 0]
ARUSER WIDTH	0	[0 - 0]
HAS_USER_BITS_PER_BYTE	NO	
WUSER BITS PER BYTE	0	[0 - 0]
RUSER BITS PER BYTE	0	[0 - 0]
WUSER WIDTH	0	[0 - 0]
RUSER WIDTH	0	[0 - 0]
BUSER WIDTH	0	[0 - 0]

Figure 2: AXI VIP configuration window.

5. Configure the **AXI GPIO IP** as shown in the following figure. We will be using the VIP to set the LEDs and read from the push buttons.

Component Name

Board | **IP Configuration**

Associate IP interface with board interface

IP Interface	Board Interface
GPIO	led 16bits
GPIO2	push buttons 5bits

☐ Enable Interrupt

Figure 3: AXI GPIO configuration window.

- Configure the **Constant** IP as shown in the following figure. The constant value of 5 will simulate pushing buttons 0 and 2.

Component Name

Const Width [1 - 4096]

Const Val

Figure 4: Constant configuration window.

- Go the **address editor** tab and set the offset address to **0x40000000**.

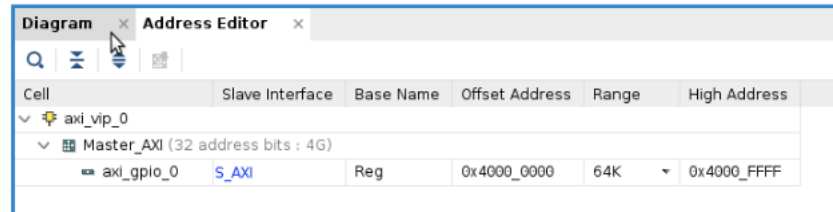


Figure 5: Address Editor

- Validate the block design, create the HDL wrapper, and generate the block design.

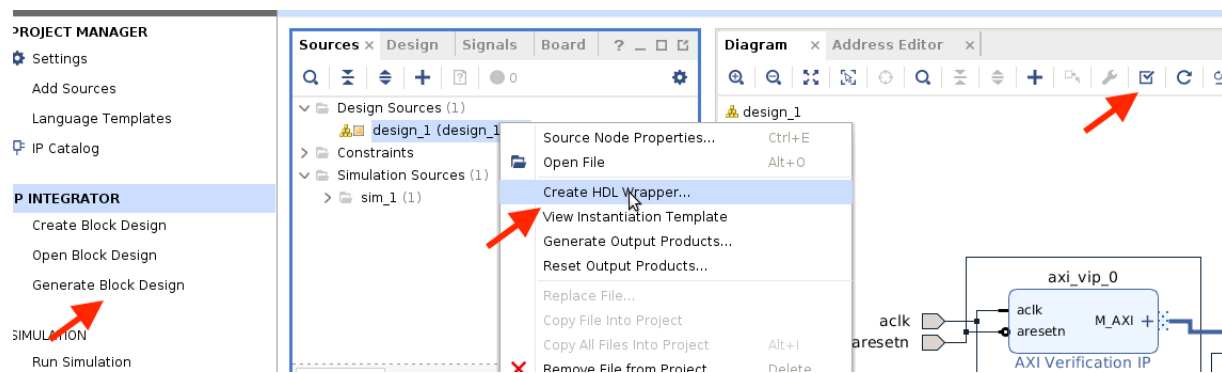


Figure 6: Generating the block design files.

Step 2: Creating the testbench

- Import **tb.sv** as a **simulation source**.

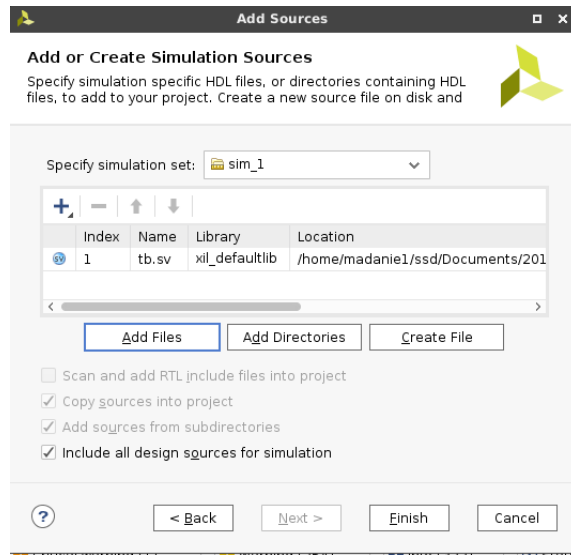


Figure 7: Importing testbench files.

2. Open the tb.sv file in Vivado text editor. Replace the `<vip_component_name>` in line 4 and 15 with the vip component name. Refer to Figure 10 to determine what the component name is for your design.

Figure 8: Replace lines 4 and 15 with the appropriate component name for your VIP.

```

1 timescale 1ns / 1ps
2
3 import axi_vip_pkg::*;
4 import design_1_axi_vip_0_0_pkg::*;
5
6 //test module to drive the AXI VIP
7 module axi_lite_stimulus();
8     /**
9      * <component_name>_mst_t for master agent
10     * <component_name> can be easily found in vivado bd design: click on the instance,
11     * Then click CONFIG under Properties window and Component_Name will be shown
12     * More details please refer P6267 section about "Useful Coding Guidelines and Examples"
13     * for more details.
14     */
15     design_1_axi_vip_0_0_mst_t agent;

```

Figure 9: Example of what lines 4 and 15 should look like.

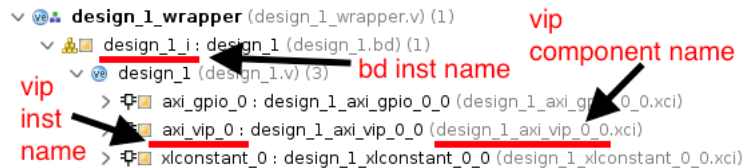


Figure 10: How to find the component and instance name.

- Go to line 60, and replace **<bd_instance_name>** and **<vip_instance_name>** with the bd instance name and vip instance name. Refer to Figure 10 to determine what the instance names are for your design.

```

52 localparam GPIO2_TRI_OFFSET = 'hC;
53 initial begin
54     /**
55     * Before agent is newed, user has to run simulation with an empty testbench to find the hierarchy
56     * path of the AXI VIP's instance.Message like
57     * "Xilinx AXI VIP Found at Path: my_ip_exdes_tb.DUT.ex_design.axi_vip_mst.inst" will be printed
58     * out. Pass this path to the new function.
59     */
60     agent = new("master vip agent",DUT.<bd_instance_name>.<vip_instance_name>.inst.IF);
61     agent.start_master();

```

Figure 11: Replace line 60 with the appropriate instance names for your BD and VIP.

```

Project Summary x tb.v x
/home/madaniel/ssd/Documents/2017.4/8v3/vip_lite_tutorial/vip_lite_tutorial.srscs/sources

46 //Constants
47 //AXI GPIO base address and register offsets
48 localparam GPIO_BASE_ADDRESS = 'h40000000;
49 localparam GPIO_DATA_OFFSET = 'h0;
50 localparam GPIO_TRI_OFFSET = 'h4;
51 localparam GPIO2_DATA_OFFSET = 'h8;
52 localparam GPIO2_TRI_OFFSET = 'hC;
53 initial begin
54 //*****
55 * Before agent is newed, user has to run simulation with an empty tes
56 * path of the AXI VIP's instance.Message like
57 * "Xilinx AXI VIP Found at Path: my_ip_exdes_tb.DUT.ex_design.axi_vip
58 * out. Pass this path to the new function.
59 //*****
60 agent = new('master vip agent', DUT.design_1_i.axi_vip_0.inst.IF);
61 agent.start_master(); // agent start to run
62

```

Figure 12: Example of what line 60 should look like.

4. Set tb instance under Simulation Sources as the top instance if it isn't already. The simulation source hierarchy should look like the figure below.

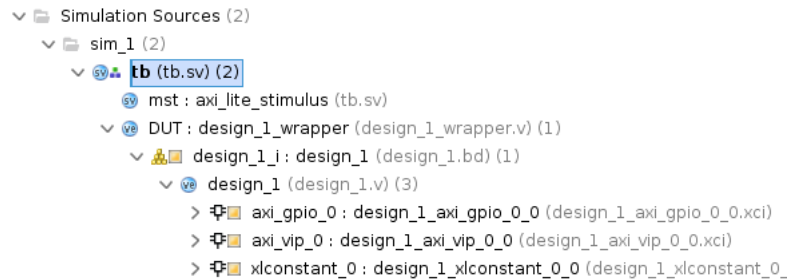


Figure 13: Simulation source hierarchy.

Step 3: Simulate the testbench

1. Before simulation, go to tb.v and analyze line 66 to 75. This is where our test vectors are located. We will be writing to the output register of the GPIO bank that is connected to the LEDs. Then we'll set the push button GPIOs to be inputs and then wait for all writes to finish. We will then read the GPIO data register for the push buttons, and print the value of that register to the tcl console.

```

66 //Turn on LED 0,1,3,5,7
67 writeRegister(GPIO_BASE_ADDRESS + GPIO_DATA_OFFSET, 'hAB);
68 //Set pushbutton GPIOs to input
69 writeRegister(GPIO_BASE_ADDRESS + GPIO2_TRI_OFFSET, 'hFFFFFFF);
70 //Wait for all writes to complete
71 agent.wait_drivers_idle();
72 //Read push button values
73 readRegister(GPIO2_DATA_OFFSET, Rdatabeat);
74 //Print read data to tcl console (similar to printf)
75 $write("GPIO2 Data register value: 0x%08x\n", Rdatabeat[0]);

```

Figure 14: TB test vectors.

2. Start the simulation by clicking "Run Simulation" on the navigation pane. Add the AXI bus signals to the simulation window and click relaunch simulation. Make sure that you select the VIP instance in the scope window.

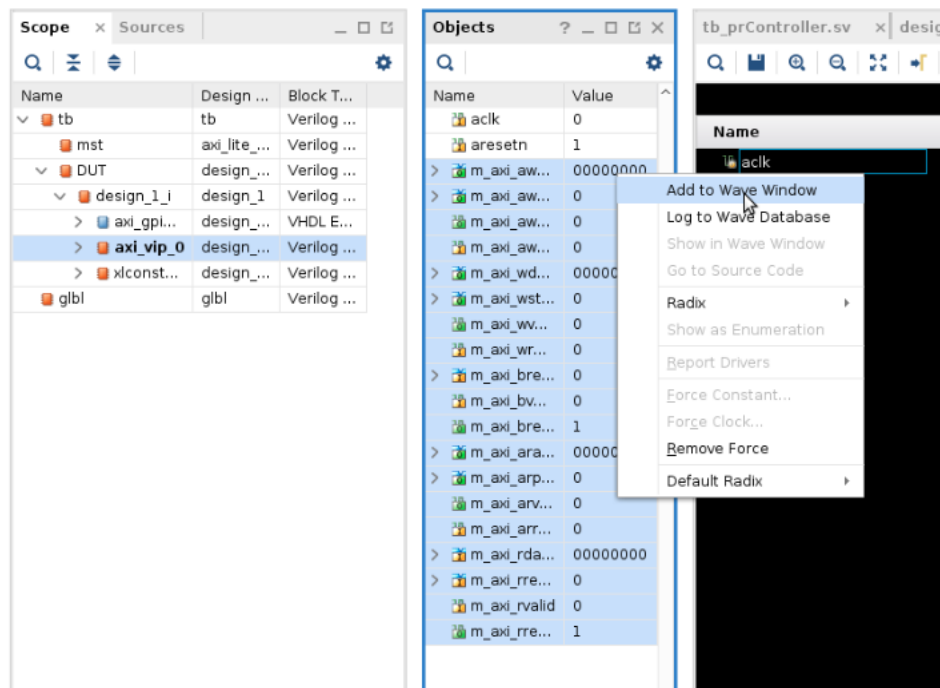


Figure 15: Adding AXI signals to the simulation.

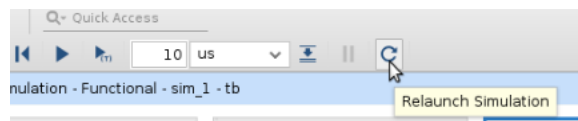


Figure 16: Relaunching the simulation.

3. Analyze the tb results...

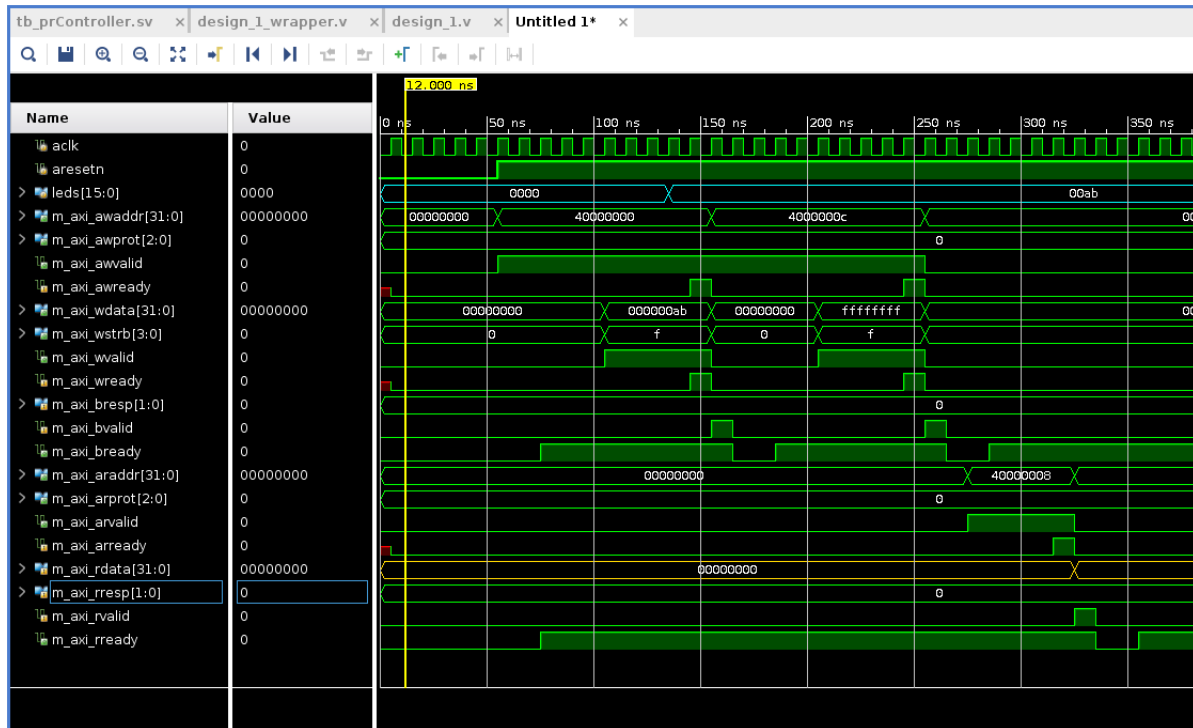


Figure 17: Simulation waveform.

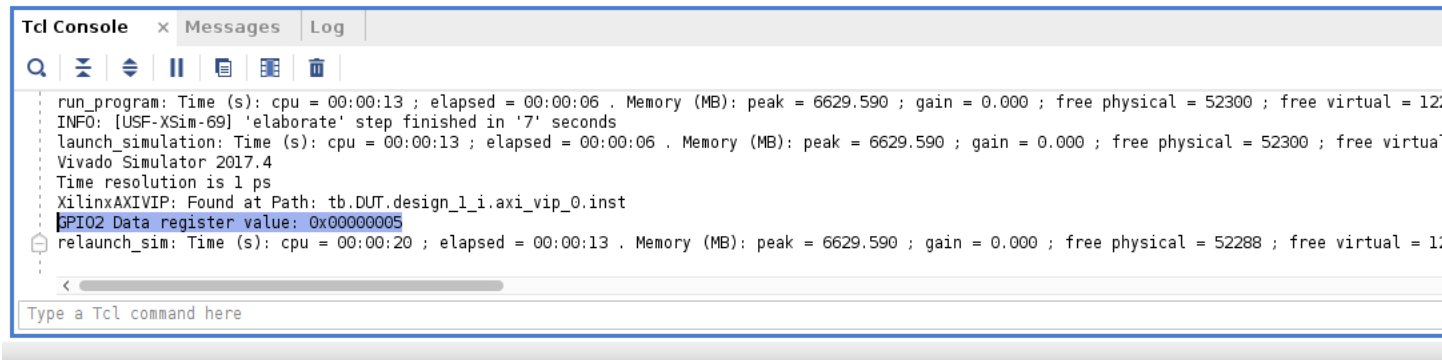


Figure 18: Tcl console print out of register.

Troubleshooting

1. If the Vivado simulator bugs out and starts throwing errors that are not related to your source files, you can try to fix it by resetting your behaviour simulation.

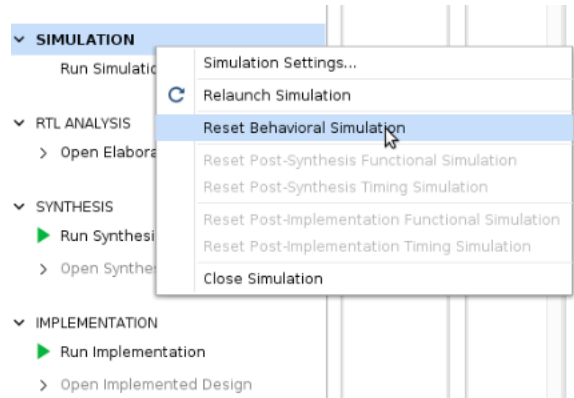


Figure 19: Resetting the simulation.

2. If you want to create more advanced AXI testbenches with the Vivado AXI VIP refer to their example design.

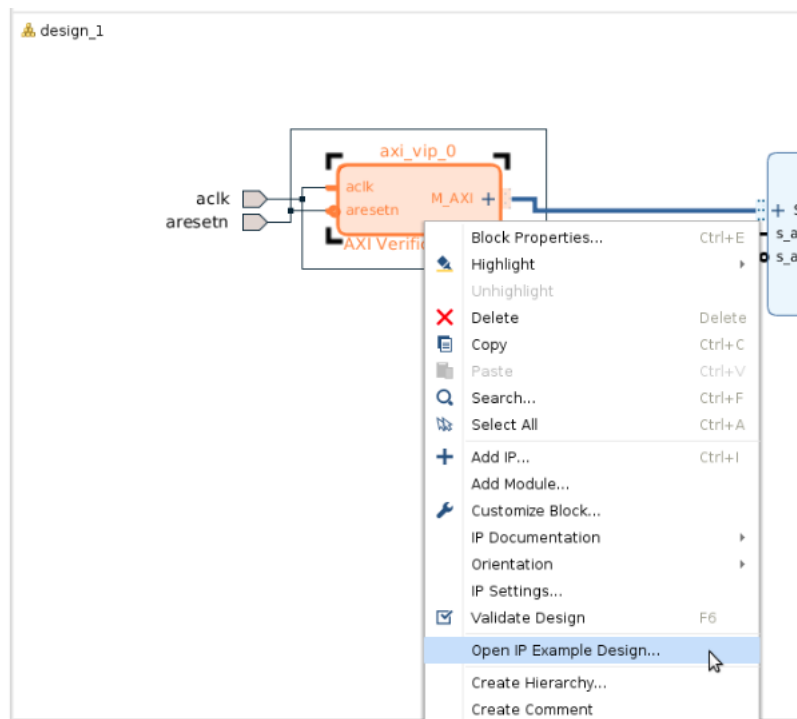


Figure 20: Advanced example design for the AXI VIP.