ECE532S Digital Systems Design

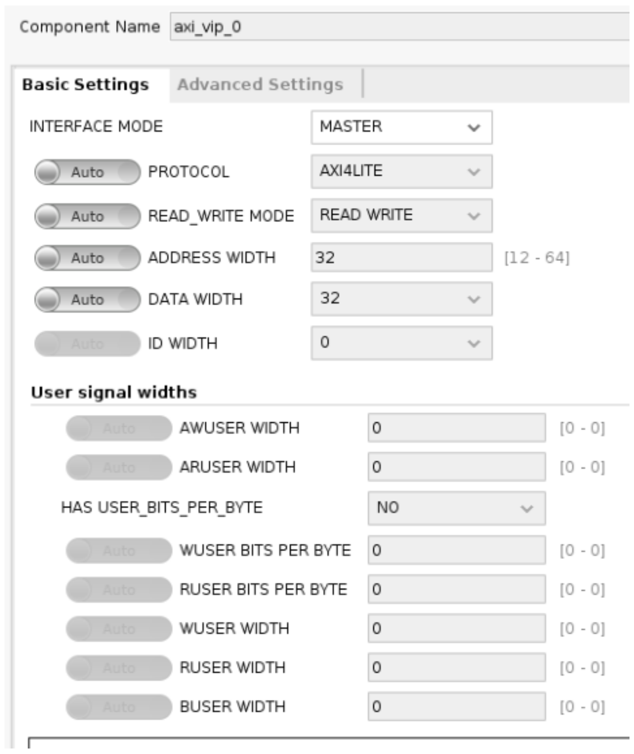
Tutorial 7 - Simulating AXI Interfaces Using the AXI VIP

Last Updated: July, 2020

The goal of this tutorial is to demonstrate how to use the Xilinx AXI Verification IP (VIP) to test and verify an AXI lite peripheral using the Xilinx Vivado Simulator. This tutorial will demonstrate how to connect the VIP to your design and use it to write to and read from registers of an AXI lite IP Core, in particular the AXI GPIO core.

1 Create Project with an AXI VIP

1. Create a new project, with no sources or constraints for now, targeting the FPGA on the *Nexsys* board (**xc7a100tcsg324-1**), as has been demonstrated in previous tutorials.
2. Create a new block design and name it **design_1**. In the block diagram, add an *AXI Verification IP* (the AXI VIP) and an *AXI GPIO* Core.
3. **Double click** the VIP Core to bring up the configuration window and ensure the *INTERFACE MODE* is set to **MASTER** (see Figure 1). This will set the AXI VIP into master mode, such that it can initiate reads and writes. Note, the *PROTOCOL* field is set to auto, as Vivado can deduce from the port it is connected to which protocol to use.


Component Name axi_vip_0

Basic Settings | Advanced Settings

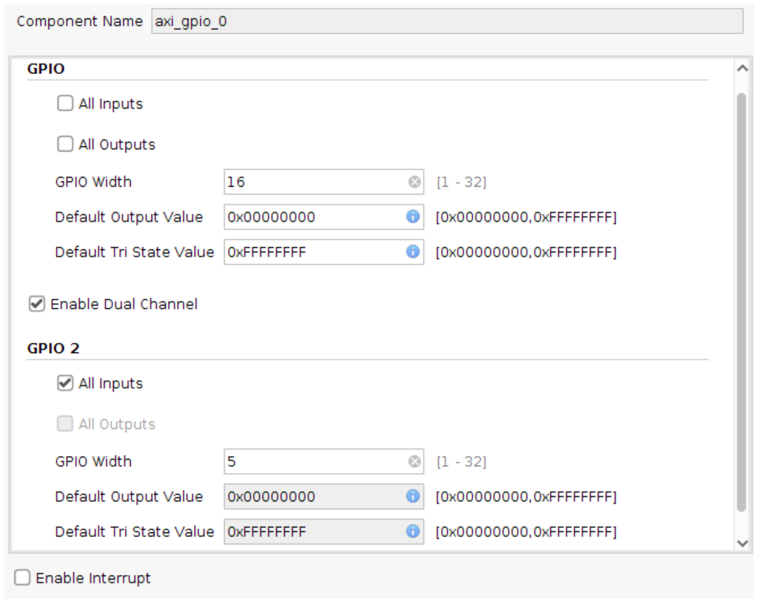
INTERFACE MODE	MASTER	
Auto PROTOCOL	AXI4LITE	
Auto READ_WRITE MODE	READ WRITE	
Auto ADDRESS WIDTH	32	[12 - 64]
Auto DATA WIDTH	32	
Auto ID WIDTH	0	

User signal widths

Auto AWUSER WIDTH	0	[0 - 0]
Auto ARUSER WIDTH	0	[0 - 0]
HAS USER_BITS_PER_BYTE	NO	
Auto WUSER BITS PER BYTE	0	[0 - 0]
Auto RUSER BITS PER BYTE	0	[0 - 0]
Auto WUSER WIDTH	0	[0 - 0]
Auto RUSER WIDTH	0	[0 - 0]
Auto BUSER WIDTH	0	[0 - 0]

Figure 1: AXI VIP configuration window, basic settings

4. Next, we need to configure the AXI GPIO Core. **Double click** the GPIO Core to bring up the configuration window. Set *GPIO Width* under *GPIO* to **16**, and place a checkmark by *Enable Dual Channel*. This will allow us to enable and modify the settings of GPIO2. Place a checkmark by *All inputs*, and set *GPIO Width* under *GPIO2* to **5**. Your settings should look like the settings in Figure 2. Press **OK** to save.


Component Name axi_gpio_0

GPIO

☐ All Inputs

☐ All Outputs

GPIO Width	16	[1 - 32]
Default Output Value	0x00000000	[0x00000000,0xFFFFFFFF]
Default Tri State Value	0xFFFFFFFF	[0x00000000,0xFFFFFFFF]

☑ Enable Dual Channel

GPIO 2

☑ All Inputs

☐ All Outputs

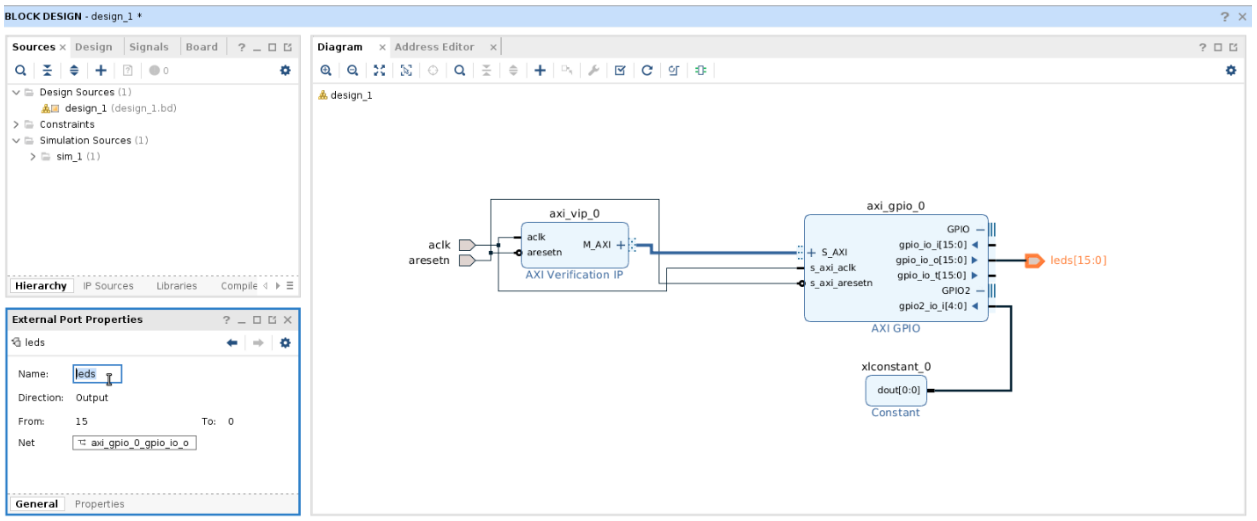
GPIO Width	5	[1 - 32]
Default Output Value	0x00000000	[0x00000000,0xFFFFFFFF]
Default Tri State Value	0xFFFFFFFF	[0x00000000,0xFFFFFFFF]

☐ Enable Interrupt

Figure 2: GPIO Core configuration window, basic settings

5. Connect the *M_ AXI* signal of the VIP to the *S_ AXI* signal of the GPIO, an AXI lite protocol signal. If you find that the signals will not connect, re-open the VIP Core's configuration window and verify that *PROTOCOL* is set to **AXI4LITE** (manually change the setting if necessary).

6. Add and wire the **AXI Verification IP**, **AXI GPIO**, and **Constant** IPs to the diagram as shown in Figure 3 below. Make sure that the port names match the figure. The **aclk** port should be set as an **Input Clock** port with frequency **100MHz**. The **aresetn** port should be set as an **Input Reset** port set to **Active Low**. The *leds* port should be set as an **Output Other** port which is a **vector** from **15 to 0**.



External Port Properties

leds

Name: leds

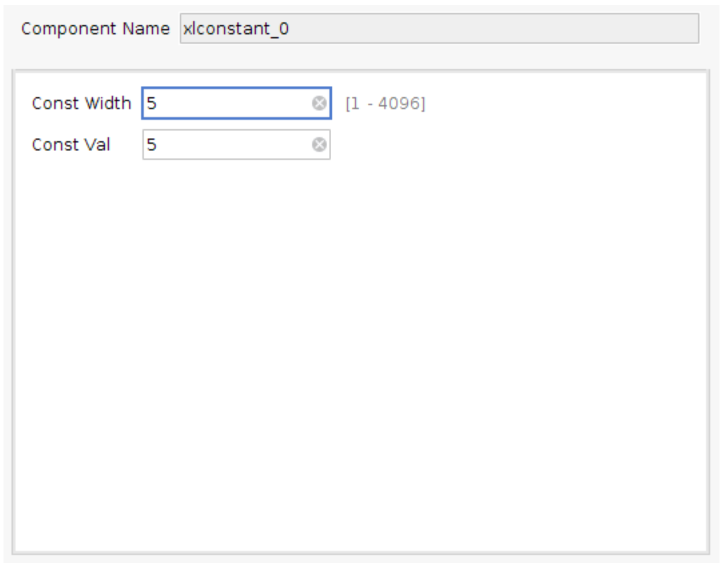
Direction: Output

From: 15 To: 0

Net axi_gpio_0_gpio_io_o

Figure 3: Block design.

7. Configure the **Constant** IP as shown in Figure 4. The constant value of 5 will simulate pushing buttons 0 and 2.



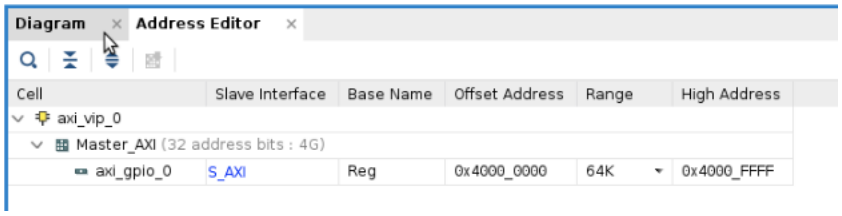
Component Name: xlconstant_0

Const Width: 5 [1 - 4096]

Const Val: 5

Figure 4: Constant configuration window.

8. Go the **address editor** tab and set the offset address to **0x40000000** (see Figure 5). If the GPIO Core (*axi_ gpio_ 0*) is *unmapped*, click and drag the entire row into **MASTER_AXI** to map it.



Cell	Slave Interface	Base Name	Offset Address	Range	High Address
axi_vip_0					
Master_AXI (32 address bits : 4G)					
axi_gpio_0	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF

Figure 5: Address Editor

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

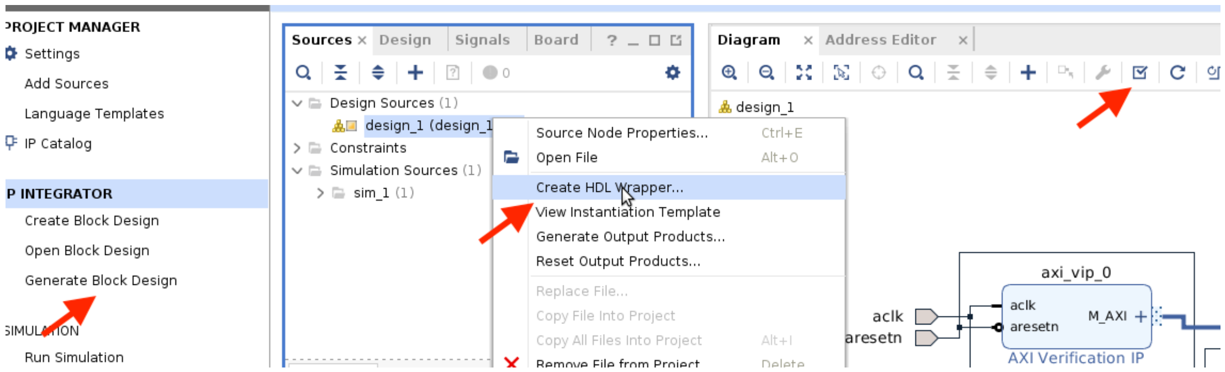
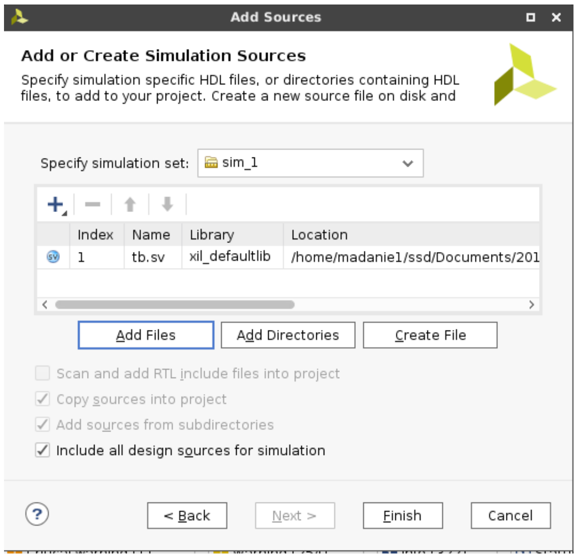


Figure 6: Generating the block design files.

Step 2: Creating the testbench

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



Add Sources

Add or Create Simulation Sources

Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and

Specify simulation set: sim_1

	Index	Name	Library	Location
	1	tb.sv	xil_defaultlib	/home/madaniel/ssd/Documents/201

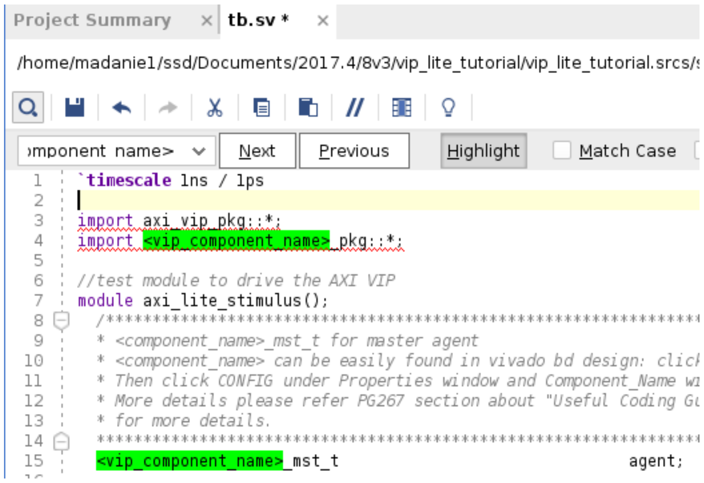
Add Files | Add Directories | Create File

- [ ] Scan and add RTL include files into project
- [✓] Copy sources into project
- [✓] Add sources from subdirectories
- [✓] Include all design sources for simulation

< Back | Next > | Finish | Cancel

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.

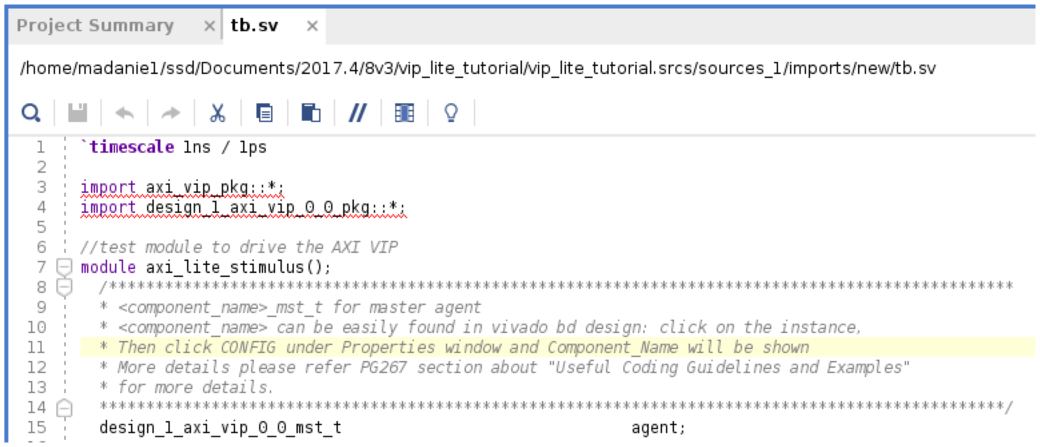


Project Summary × | tb.sv * ×

/home/madaniel/ssd/Documents/2017.4/8v3/vip_lite_tutorial/vip_lite_tutorial.srcs/s

```
1   `timescale 1ns / 1ps
2   
3   import axi_vip_pkg::*;
4   import <vip_component_name>_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
11     * Then click CONFIG under Properties window and Component_Name wi
12     * More details please refer PG267 section about "Useful Coding Gu
13     * for more details.
14    *************************************************************
15    <vip_component_name>_mst_t                              agent;
```

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

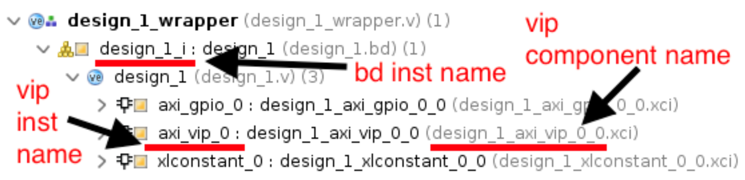


Project Summary × tb.sv ×

/home/madaniel/ssd/Documents/2017.4/8v3/vip_lite_tutorial/vip_lite_tutorial.srcs/sources_1/imports/new/tb.sv

```
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 PG267 section about "Useful Coding Guidelines and Examples"
13    * for more details.
14    *************************************************************************************************/
15    design_1_axi_vip_0_0_mst_t                              agent;
```

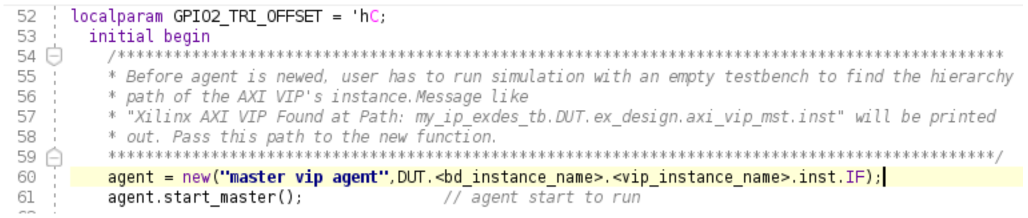
Figure 9: Example of what lines 4 and 15 should look like.



```
design_1_wrapper (design_1_wrapper.v) (1)
  design_1_i : design_1 (design_1.bd) (1)
    design_1 (design_1.v) (3)
      axi_gpio_0 : design_1_axi_gpio_0_0 (design_1_axi_gpio_0_0.xci)
      axi_vip_0 : design_1_axi_vip_0_0 (design_1_axi_vip_0_0.xci)
      xlconstant_0 : design_1_xlconstant_0_0 (design_1_xlconstant_0_0.xci)
```

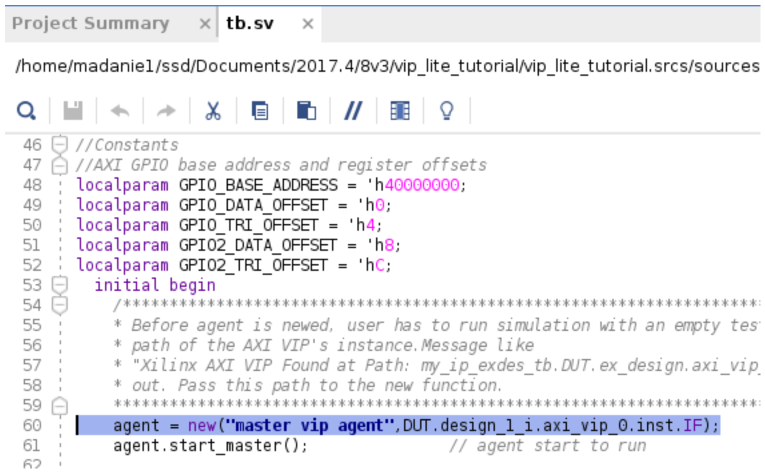
Figure 10: How to find the component and instance name.

3. 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();              // agent start to run
```

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

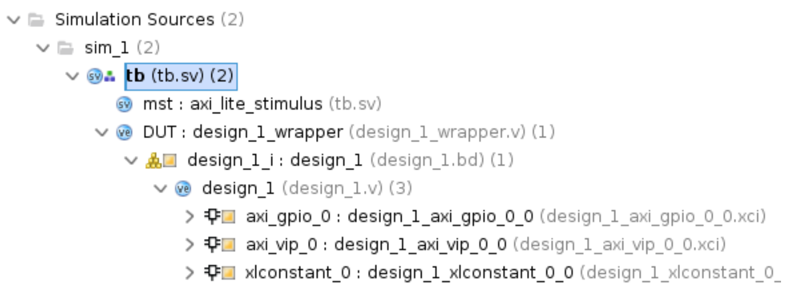


/home/madaniel/ssd/Documents/2017.4/8v3/vip_lite_tutorial/vip_lite_tutorial.srcs/sources

```
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.

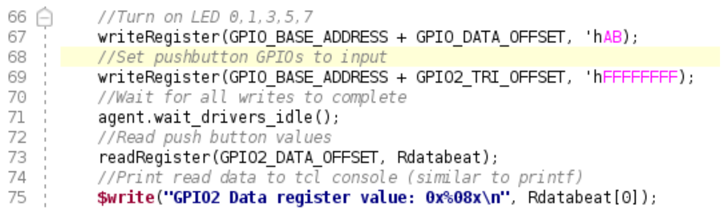


```
Simulation Sources (2)
  sim_1 (2)
    tb (tb.sv) (2)
      mst : axi_lite_stimulus (tb.sv)
      DUT : design_1_wrapper (design_1_wrapper.v) (1)
        design_1_i : design_1 (design_1.bd) (1)
          design_1 (design_1.v) (3)
            axi_gpio_0 : design_1_axi_gpio_0_0 (design_1_axi_gpio_0_0.xci)
            axi_vip_0 : design_1_axi_vip_0_0 (design_1_axi_vip_0_0.xci)
            xlconstant_0 : design_1_xlconstant_0_0 (design_1_xlconstant_0_
```

Figure 13: Simulation source hierarchy.

Step 3: Simulate the testbench

1. Before simulation, go to tb.sv 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.

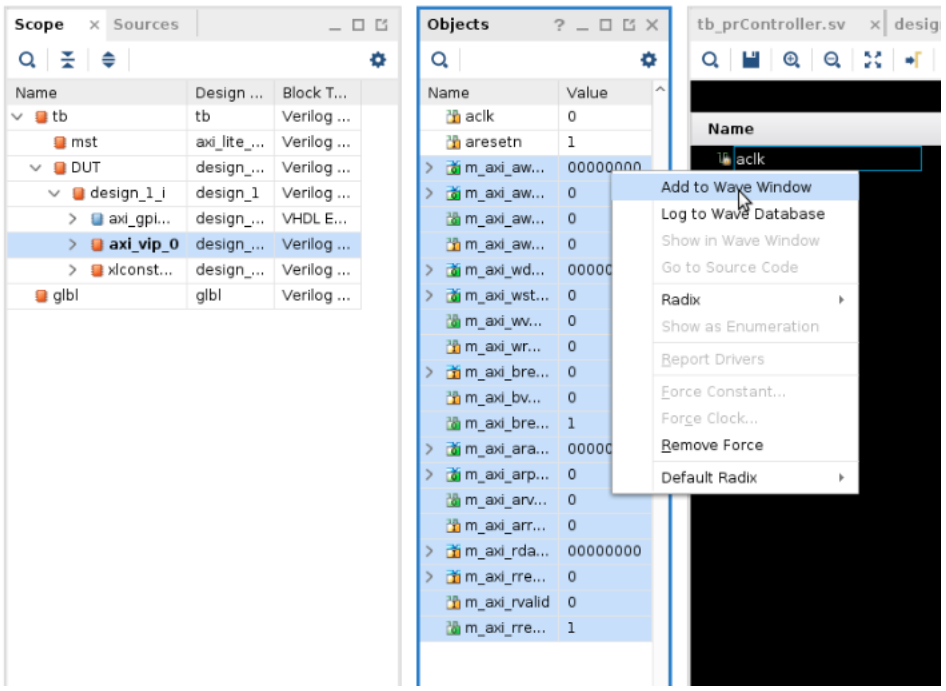


```
66      //Turn on LED 0,1,3,5,7
67      writeRegister(GPIO_BASE_ADDRESS + GPIO_DATA_OFFSET, 'hAB);
68      //Set pushbutton GPIOs to input
69      writeRegister(GPIO_BASE_ADDRESS + GPIO2_TRI_OFFSET, 'hFFFFFFFF);
70      //Wait for all writes to complete
71      agent.wait_drivers_idle();
72      //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** → **Run Behavioural Simulation** on

the navigation pane. Add the AXI bus signals in the *Objects* window (Make sure that you select the VIP instance in the *Scope* window) to the simulation window, shown in Figure 15. Relaunch the simulation by clicking the **Relaunch** button shown in Figure 16.



Name	Design ...	Block T...
tb	tb	Verilog ...
mst	axi_lite_...	Verilog ...
DUT	design_...	Verilog ...
design_1_i	design_1	Verilog ...
axi_gpi...	design_...	VHDL E...
axi_vip_0	design_...	Verilog ...
xlconst...	design_...	Verilog ...
glbl	glbl	Verilog ...

Name	Value
aclk	0
aresetn	1
m_axi_aw...	00000000
m_axi_aw...	0
m_axi_aw...	0
m_axi_aw...	0
m_axi_wd...	0000
m_axi_wst...	0
m_axi_wv...	0
m_axi_wr...	0
m_axi_bre...	0
m_axi_bv...	0
m_axi_bre...	1
m_axi_ara...	0000
m_axi_arp...	0
m_axi_arv...	0
m_axi_arr...	0
m_axi_rda...	00000000
m_axi_rre...	0
m_axi_rvalid	0
m_axi_rre...	1

Figure 15: Adding AXI signals to the simulation.

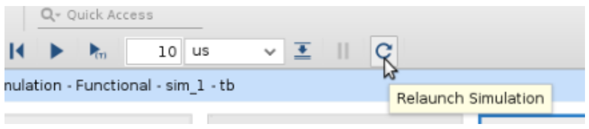


Figure 16: Relaunching the simulation.

3. Analyze the tb results. The waveform should look like Figure 17. In the *Tcl Console* you should see *GPIO2* returning *0x00000005*, as seen in Figure 18.

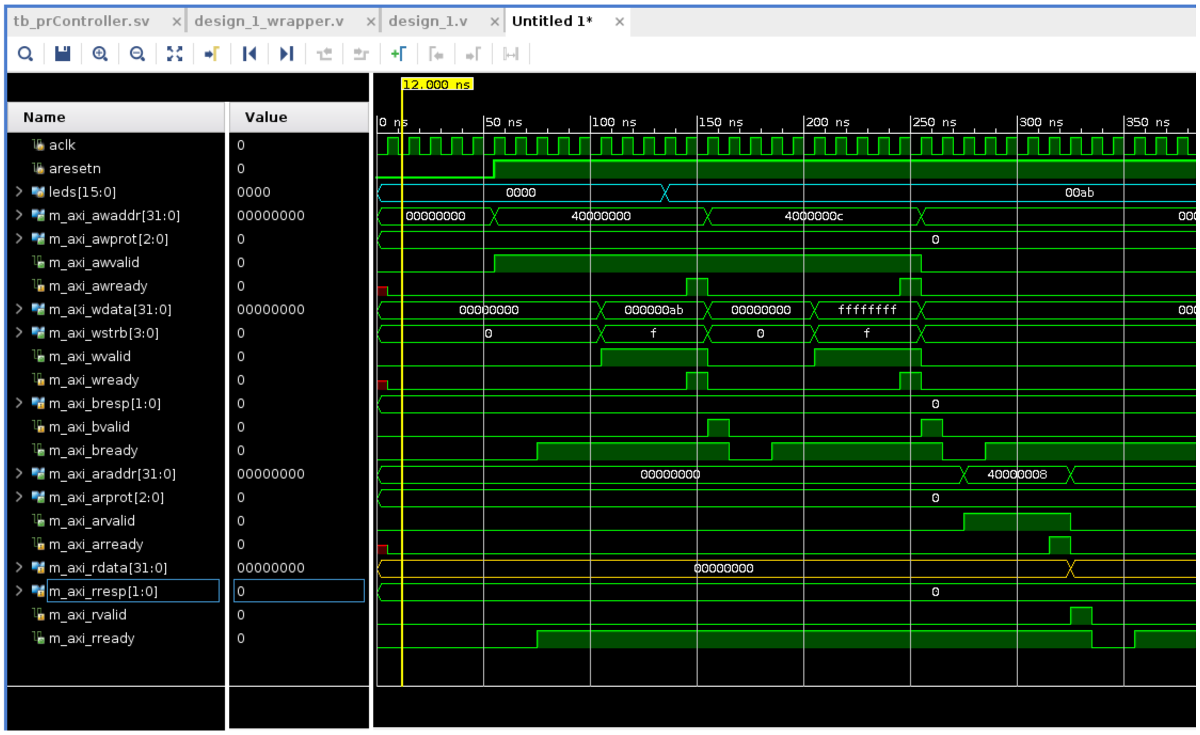
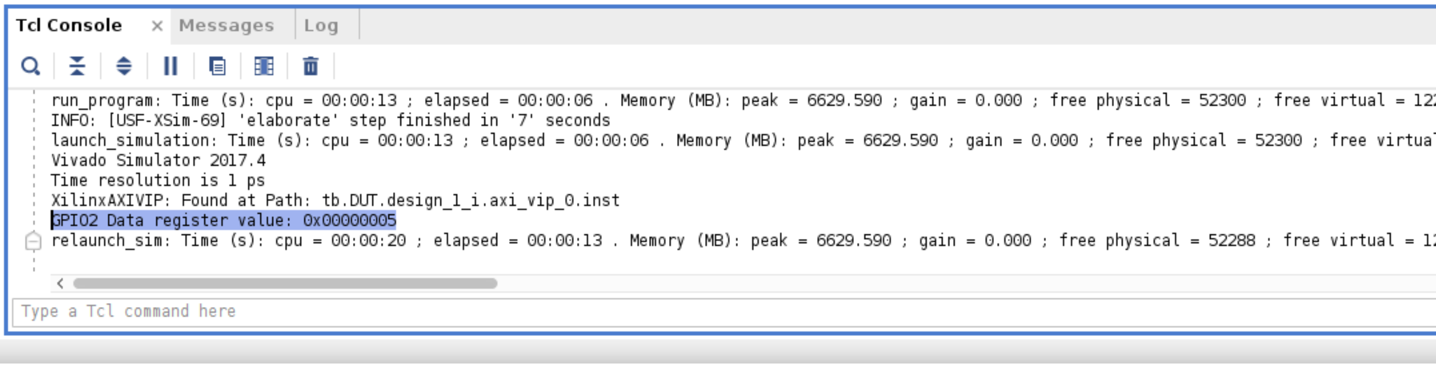


Figure 17: Simulation waveform.



```
run_program: Time (s): cpu = 00:00:13 ; elapsed = 00:00:06 . Memory (MB): peak = 6629.590 ; gain = 0.000 ; free physical = 52300 ; free virtual = 12
INFO: [USF-XSim-69] 'elaborate' step finished in '7' seconds
launch_simulation: Time (s): cpu = 00:00:13 ; elapsed = 00:00:06 . Memory (MB): peak = 6629.590 ; gain = 0.000 ; free physical = 52300 ; free virtua
Vivado Simulator 2017.4
Time resolution is 1 ps
XilinxAXIVIP: Found at Path: tb.DUT.design_1_i.axi_vip_0.inst
GPIO2 Data register value: 0x00000005
relaunch_sim: Time (s): cpu = 00:00:20 ; elapsed = 00:00:13 . Memory (MB): peak = 6629.590 ; gain = 0.000 ; free physical = 52288 ; free virtual = 1
```

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 reseting your behaviour simulation.

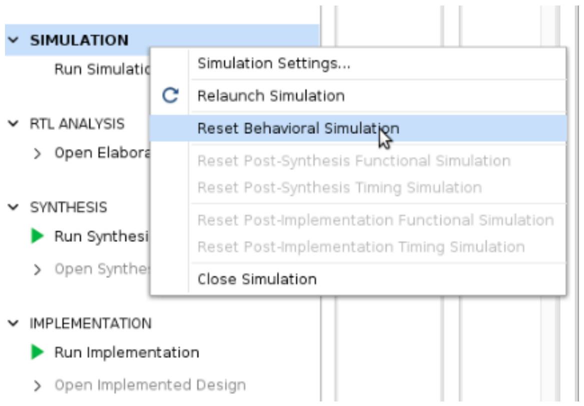

SIMULATION
Run Simulatio
RTL ANALYSIS
Open Elabora
SYNTHESIS
Run Synthesi
Open Synthe
IMPLEMENTATION
Run Implementation
Open Implemented Design
Simulation Settings...
Relaunch Simulation
Reset Behavioral Simulation
Reset Post-Synthesis Functional Simulation
Reset Post-Synthesis Timing Simulation
Reset Post-Implementation Functional Simulation
Reset Post-Implementation Timing Simulation
Close 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.

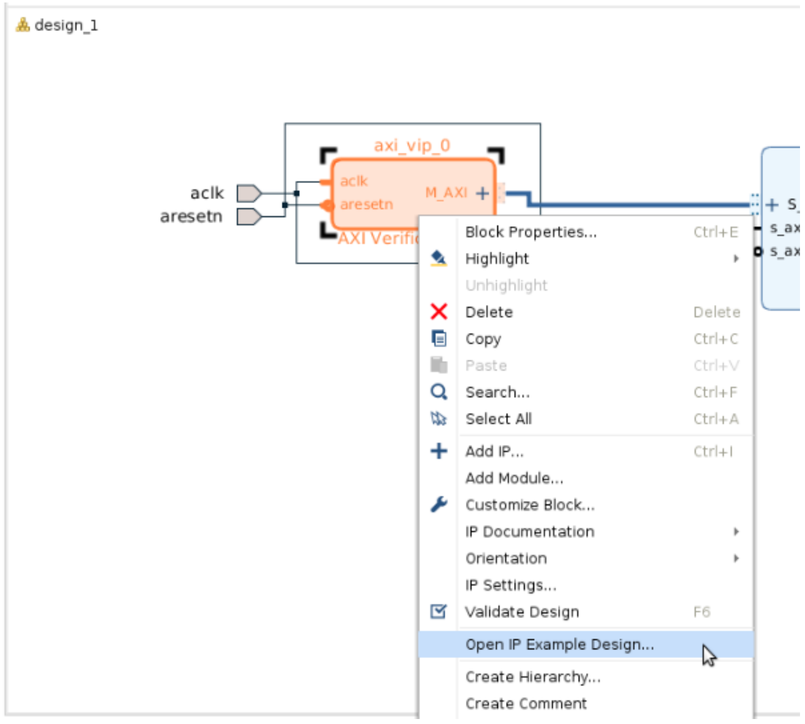


Figure 20: Advanced example design for the AXI VIP.