ECE 194BB Lab Assignment 4

Nitin Kataria, Yu Wang (TA)
ECI Lab (HFH 1140, Inner room)

Creating and Testing Custom SPI Master IP with AXI-Lite Interface in Vivado

1. Introduction

In complex digital systems, module communication often adheres to specific protocols. The creation of Intellectual Property (IP) capable of decoding and generating protocol-compliant signals is a pivotal aspect of digital system design. Although there exists a myriad of design and protocol combinations, Lab 4 is tailored to acquaint you with this process. We will narrow our focus to the development of a custom Serial Peripheral Interface (SPI) master module, subsequently integrating it with an AXI-Lite interface to form your own custom IP.

Lab 4 is divided into two principal sections. In Part A, you will engage with a Vivado example AXI peripheral design to learn the procedure of creating and modifying an AXI peripheral. This will include hands-on experience with reading and writing to memory-mapped registers via JTAG-AXI master. Part B will challenge you to devise a simplified SPI master module. This module will interface with a provided slave module and be incorporated as a custom IP. The functionality and integration of your IP will be verified and refined using JTAG-AXI master and the Integrated Logic Analyzer (ILA), tools with which you have been used in Lab 3.

This lab spans two weeks, with a final deadline for both the lab checkout and report submission set for **March 1st** and a total points of **200**.

2. Part A: Example AXI-Lite Peripheral in Vivado (20 pt)

Before starting this section, you are strongly recommended to review the basic AXI-Lite protocol and understand the control and data signal in the read and write channel. Although in this lab, we will adopt the pre-generated AXI-Lite logic from the example design, high performance design usually requires you to write you own decoding and encoding logic.

Step 1. Create Vivado Project:

1) Create a new Vivado project with board selected as Nexys A7-100T.

Step 2. Create Example AXI-Lite Peripheral IP (referred to "myspi" in later sections):

1) As in Figure 1, under "Tools", select "Create and Package New IP". Click "Next". In the next popped window, choose "Create a new AXI4 peripheral" and click "Next". Set a proper name for your IP, "myspi" is recommended.
2) Since myspi will be connected to a JTAG-AXI master, it needs a slave port. Configure the IP as shown in Figure 2. Click "Next".
3) In the next step, select "Edit IP" and click "Finish". A new Vivado window should appear as in Figure 3.

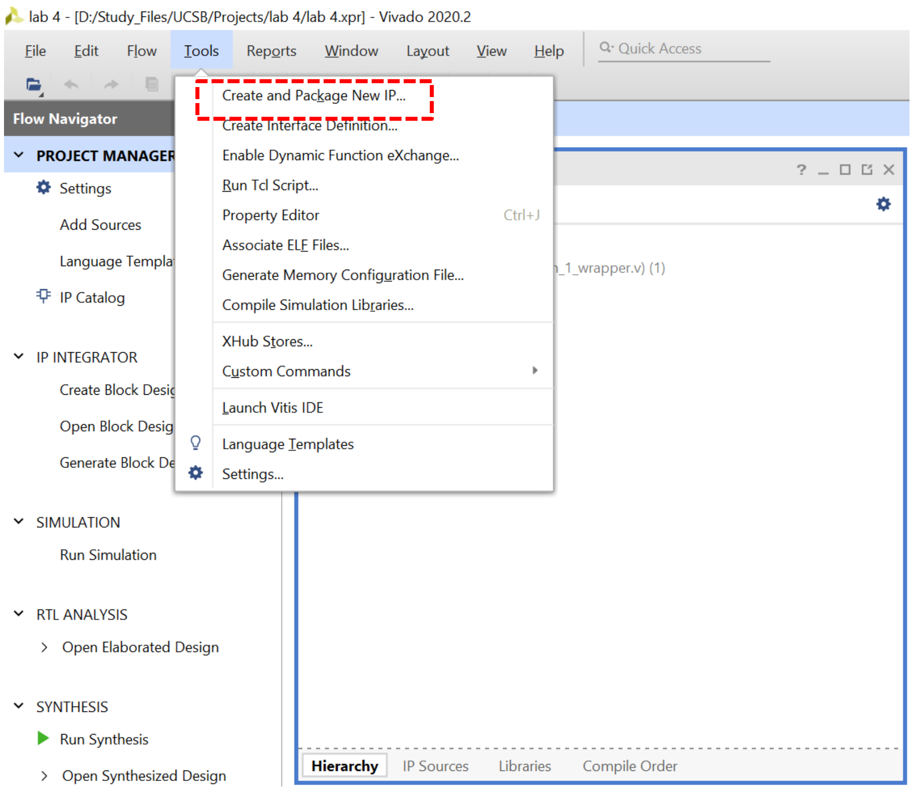
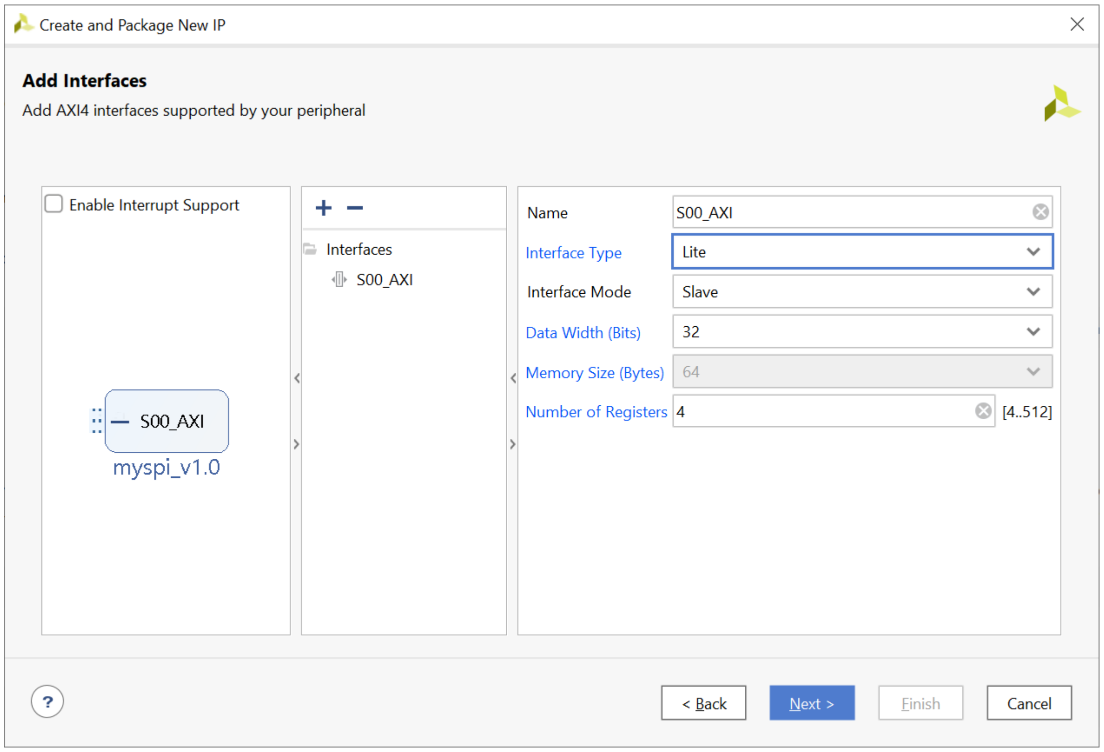
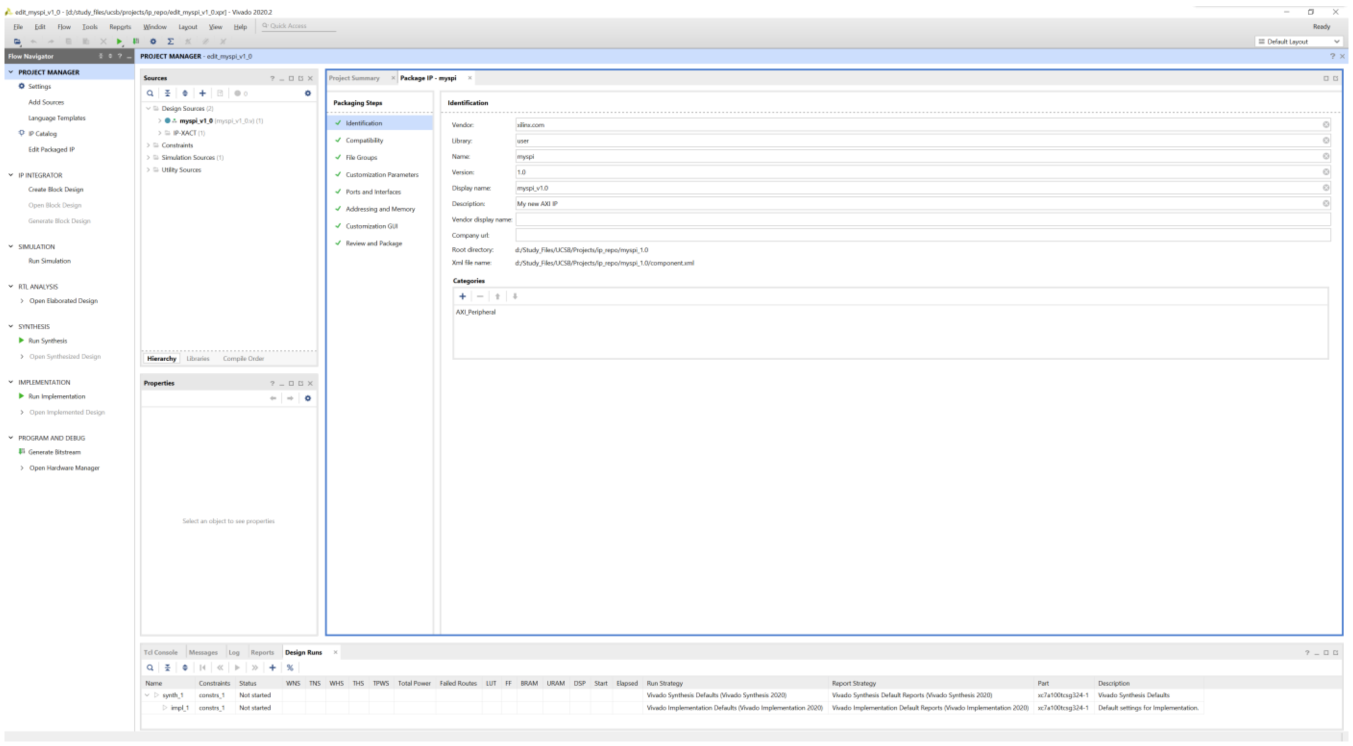


Figure 1: Create a new AXI peripheral.



Name	S00_AXI
Interface Type	Lite
Interface Mode	Slave
Data Width (Bits)	32
Memory Size (Bytes)	64
Number of Registers	4 [4..512]

Figure 2: Create a new AXI peripheral (cont).



Identification

Vendor: xilinx.com
Library: user
Name: myspi
Version: 1.0
Display name: myspi_v1.0
Description: My new AXI IP
Vendor display name:
Company url:
Root directory: d:/Study_Files/UCSB/Projects/ip_repo/myspi_1.0
Xml file name: d:/Study_Files/UCSB/Projects/ip_repo/myspi_1.0/component.xml

Categories

AXI_Peripheral

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy	Report Strategy	Part	Description
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivado Synthesis 2020)	Vivado Synthesis Default Reports (Vivado Synthesis 2020)	xc7a100tcsg324-1	Vivado Synthesis Defaults
impl_1	constrs_1	Not started															Vivado Implementation Defaults (Vivado Implementation 2020)	Vivado Implementation Default Reports (Vivado Implementation 2020)	xc7a100tcsg324-1	Default settings for Implementation.

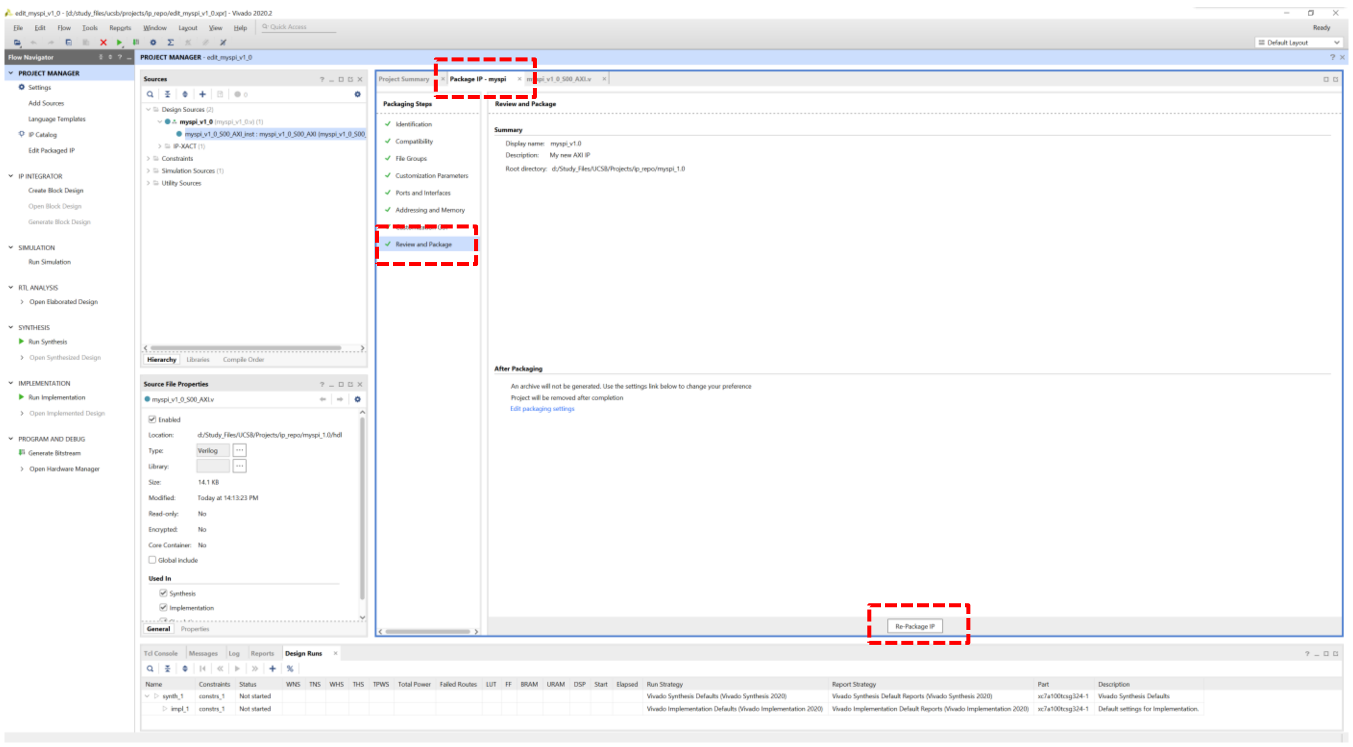
Figure 3: New project window for editing IP.

Step 3. Read and Understand the Template Logic:

1) In the “Design Source”, expand “myspi_v1_0” and open “myspi_v1_0_S00_AXI.v”. This is the template file generate by Xilinx with predefined encoding and decoding logic for AXI-Lite.
2) Read this Verilog file and understand how the four slave registers are selected with a given address.
3) Do not modify any part of the code for now.

Step 4. Create Block Design and HDL Wrapper:

1) As in Figure 4, click “Package IP - myspi”, go to “Review and Package” and click “Re-Package IP” (Note, this does nothing here since you haven’t modified the IP). Click “close” in the popped window and return back to your original project window.



Review and Package

Summary

Display name: myspi_v1.0
Description: My new AXI IP
Root directory: d:/Study_Files/UCSB/Projects/ip_repo/myspi_1.0

After Packaging

An archive will not be generated. Use the settings link below to change your preference
Project will be removed after completion
Edit packaging settings

Source File Properties

Location: d:/Study_Files/UCSB/Projects/ip_repo/myspi_1.0/hdl
Type: Verilog
Size: 14.1 KB
Modified: Today at 14:13:23 PM
Read-only: No
Encrypted: No
Core Container: No

Name	Constraints	Status	Run Strategy	Report Strategy	Part	Description
synth_1	constrs_1	Not started	Vivado Synthesis Defaults (Vivado Synthesis 2020)	Vivado Synthesis Default Reports (Vivado Synthesis 2020)	xc7a100tcsg324-1	Vivado Synthesis Defaults
impl_1	constrs_1	Not started	Vivado Implementation Defaults (Vivado Implementation 2020)	Vivado Implementation Default Reports (Vivado Implementation 2020)	xc7a100tcsg324-1	Default settings for Implementation.

Figure 4: Repackaging IP.

Step 5. Test on Board:

1) Create a new block design with a proper name. Add “JTAG to AXI Master” and “myspi_v1.0”. Double click on “JTAG to AXI Master” and configure the “AXI Protocol” to be “AXI4LITE”.
2) Click on “Run Connection Automation”, on the left panel, check “myspi_0” if it has not been checked, then click “OK”.
3) As in Lab 3, since we have only one AXI peripheral, select and delete “AXI Interconnect”. Next, on the “Processor System Reset” block, connect the “ext_reset_in” input to “locked” output of Clocking Wizard.
4) Click on “Run Connection Automation”, on the left panel, check all categories and click “OK”.
5) Right click on blank area and click “Regenerate Layout”, you block diagram should look like Figure 5.
6) Click on “Address Editor” and take a look on the base address.

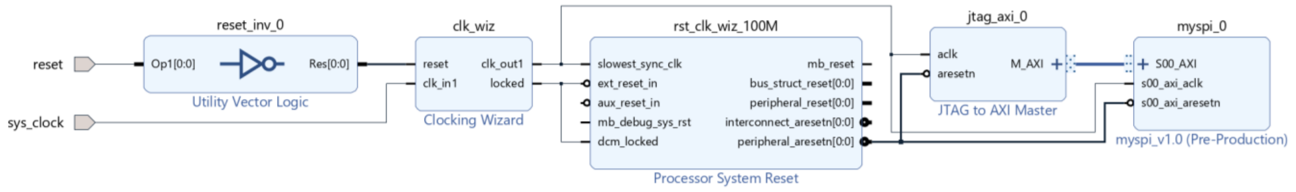
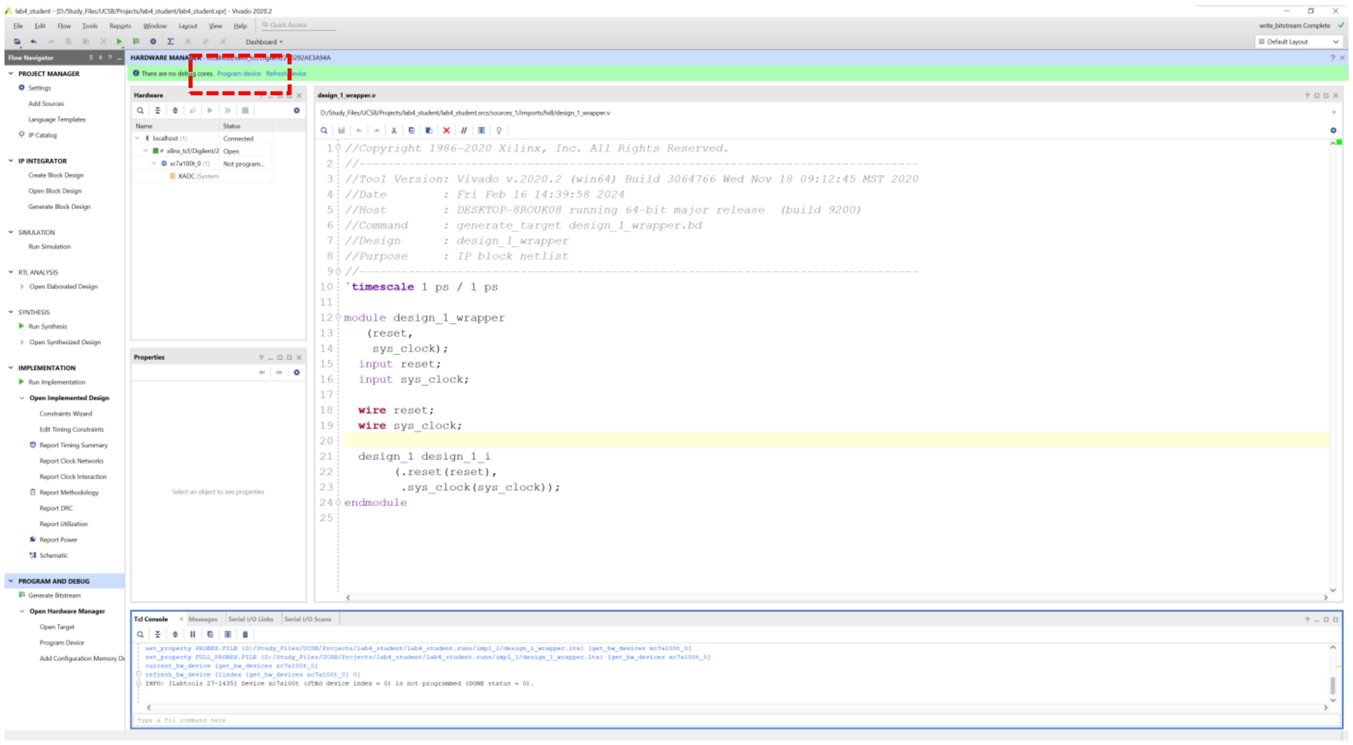


Figure 5: Final diagram for part A.

7) Follow Lab 3 to generate the HDL wrapper for the block design and run implementation.
8) With board files loaded, we don’t explicitly assign the pins. Once implementation is complete, click “Generate Bitstream”.
9) Plug in your board and power on. Click “Open Hardware Manager” and “Open Target”. Click “Auto Connect”. In the popped window, find and “program device”, as in Figure 6.



```
//Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.
//--------------------------------------------------------------------------------
//Tool Version: Vivado v.2020.2 (win64) Build 3064766 Wed Nov 18 09:12:45 MST 2020
//Date        : Fri Feb 16 14:39:58 2024
//Host        : DESKTOP-8ROUK08 running 64-bit major release  (build 9200)
//Command     : generate_target design_1_wrapper.bd
//Design      : design_1_wrapper
//Purpose     : IP block netlist
//--------------------------------------------------------------------------------
`timescale 1 ps / 1 ps

module design_1_wrapper
   (reset,
    sys_clock);
  input reset;
  input sys_clock;

  wire reset;
  wire sys_clock;

  design_1 design_1_i
       (.reset(reset),
        .sys_clock(sys_clock));
endmodule
```

```
set_property PROBES.FILE {D:/Study_Files/UCSB/Projects/lab4_student/lab4_student.runs/impl_1/design_1_wrapper.ltx} [get_hw_devices xc7a100t_0]
set_property FULL_PROBES.FILE {D:/Study_Files/UCSB/Projects/lab4_student/lab4_student.runs/impl_1/design_1_wrapper.ltx} [get_hw_devices xc7a100t_0]
current_hw_device [get_hw_devices xc7a100t_0]
refresh_hw_device [lindex [get_hw_devices xc7a100t_0] 0]
INFO: [Labtools 27-1435] Device xc7a100t (JTAG device index = 0) is not programmed (DONE status = 0).
```

Figure 6: Connect and program device

10) Answer the following questions: what are the addresses for slv_reg0, slv_reg1, slv_reg2 and slv_reg3?
11) Execute the TCL commands in order provided in part_A.txt. Explain the purpose of each command and demonstrate the result to your TA. What do you find after executing Command 7?

The example AXI peripheral design provide us a convenient way to read and write the memory mapped registers via the JTAG-AXI master. These registers can serve as the "bridge" to communicate with our custom logic, which you will be exercised in part B.

3. Part B.1: Specification and Design of the Custom SPI Module (80 pt)

Before you start this section, you are **required** to go through SPI Wikipedia and Quad AXI SPI and understand how SPI protocol transmit data between the SPI master and the SPI slave.

Step 1. IP Diagram Overview:

The high-level system diagram is shown in Figure 7. We will adopt the setting in part A with four 32-bit slave registers. The spi_slave module is a simple loopback shift register and the Verilog code is provided to you.

The module you need to design is spi_master. The slv_reg0 will be used as the control signal and connect to ctrl port. The slv_reg1 will be used as the data transition register which will set the shift register values inside spi-master. The slv_reg2 will be used as the data read register, which allow you to use TCL command to monitor the transition results.

Note: you may notice that slv_reg2 is drawn as a read only register (for AXI interface). Keep this in mind and we will discuss later.

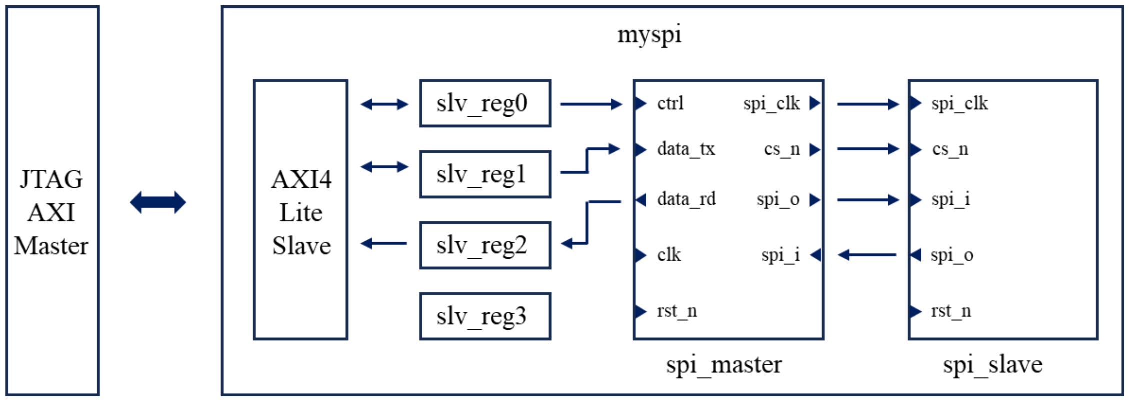


Figure 7: Design system diagram.

Step 2. Specification for SPI Transition:

You are asked to design the spi_master with the following specification:

- Frequency Ratio: 16
- Transition Width: 32
- CPOL: 0, CPHA: 0

Step 3. Control Signal and Logic:

- ctrl = 0x00000001: configure the local shift register with the data on data_tx.
- ctrl = 0x00000002: enables the SPI transition. However, the transition should not start unless the local shift register has been written at least once. After each complete transition, the spi_master is not allowed to initiate a new transition unless the shift register has been written again.
- The data on data_rd should be updated only outside the transition stage.

Note: a complete transition here denotes by transmitting all 32 bits sequentially, not just a single bit.

Step 4. Example Transition Timing Diagram:

To help you start, we provide a sample timing diagram in Figure 8 for the transition stage for the following specification:

- Frequency Ratio: 2
- Transition Width: 4
- CPOL: 0, CPHA: 0

The m_sr and s_sr denote by the shift register in the spi_master and spi_slave respectively.

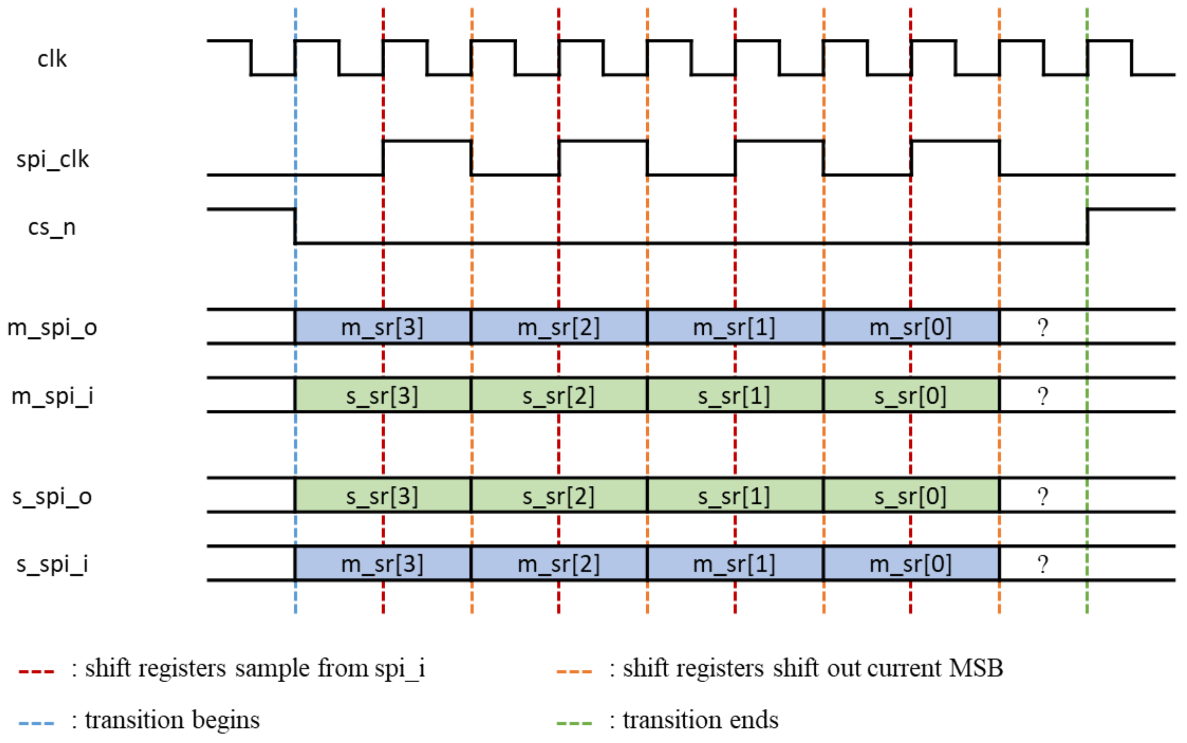


Figure 8: Example timing diagram with frequency ratio = 2, transition width= 4, CPOL = 0 and CPHA = 0.

Step 5. FSM Design and Verilog Implementation:

1) **Hints:**

- Your FSM should at least contain three states: IDLE, SET_SR and TRANSITION (specify the names as you like)
- Look into Figure 8, how can you further divide TRANSITION state? How many total states do you have now?
- How to implement the frequency ratio specification? How to implement the “cs_n” changing logic? What are the additional registers you have to define?

2) After you have finished your FSM design, open your project and add the starter code “spi_master.v” and the provided “spi_slave.v” into your “Design Resources”. Add the provided “spi_tb.v” into the “Simulation Resources”, under “sim_1”.
3) Finish coding “spi_master.v”.

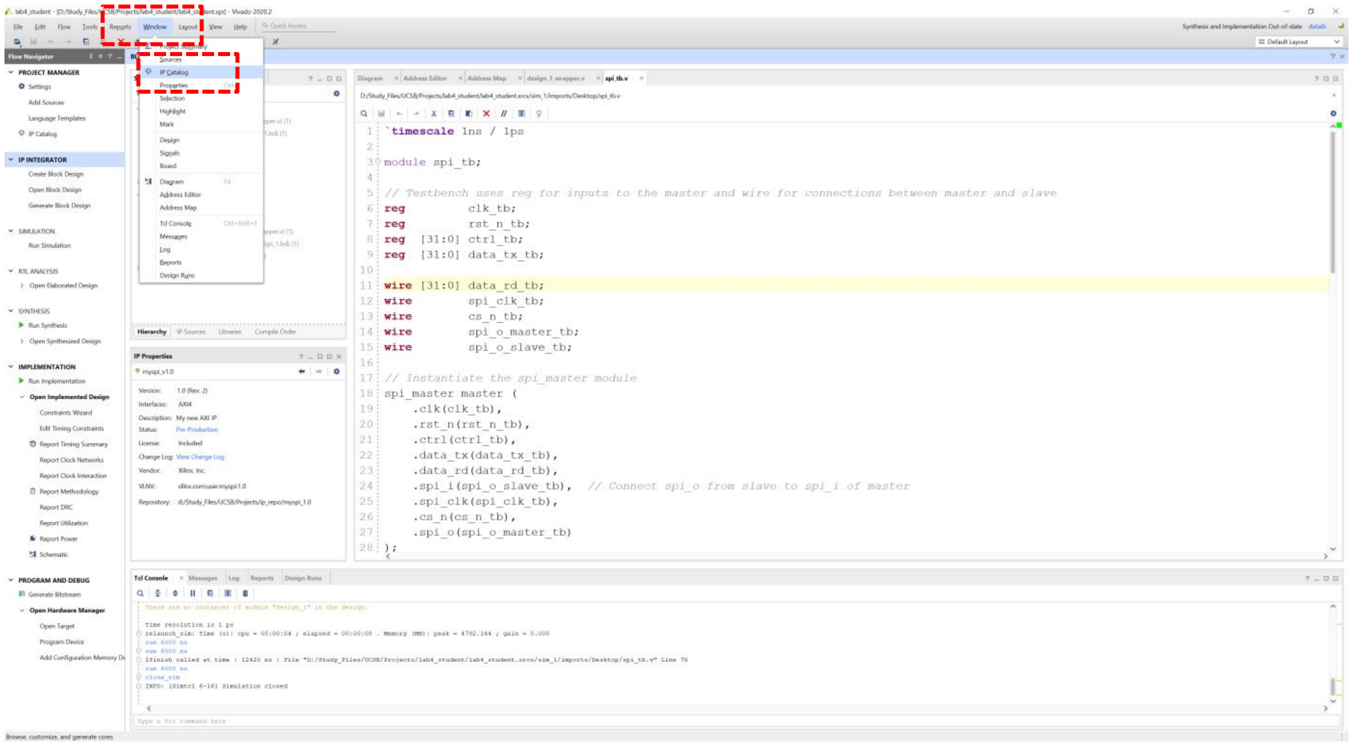
Step 6. Evaluate the Design with Provided Testbench:

1) In “Simulation Resources”, make “spi_tb.v” the top module and run the behavior simulation. Demonstrate your correct results to your TA.
2) Optional: run synthesis of your “spi_master.v”, make sure you don’t get critical problems, e.g., inferring a latch, multi-driven, etc; (Don’t forget to set the correct top module in “Design Resources”).

4. Part B.2: Test the Custom AXI-Lite SPI IP (40 pt)

Step 1. Integrate Custom SPI Module into myspi IP:

1) As in Figure 9, open “IP Catalog”. Search for myspi, right click on the IP and select “Edit in IP Packager”, as in Figure 10. This will again open a new window as in part A.


lab4_student - [D:/Study_Files/UCSB/Projects/lab4_student/lab4_student.xpr] - Vivado 2020.2
File Edit Flow Tools Reports Window Layout View Help
Project Summary
Sources
IP Catalog
Properties
Selection
Highlight
Mark
Design
Signals
Board
Diagram
Address Editor
Address Map
Tcl Console
Messages
Log
Reports
Design Runs
Hierarchy IP Sources Libraries Compile Order
Diagram Address Editor Address Map design_1_wrapper.v spi_tb.v
Tcl Console Messages Log Reports Design Runs

IP Properties

myspi_v1.0

Version: 1.0 (Rev. 2)
Interfaces: AXI4
Description: My new AXI IP
Status: Pre-Production
License: Included
Change Log: View Change Log
Vendor: Xilinx, Inc.
VLNV: xilinx.com:user:myspi:1.0
Repository: d:/Study_Files/UCSB/Projects/ip_repo/myspi_1.0

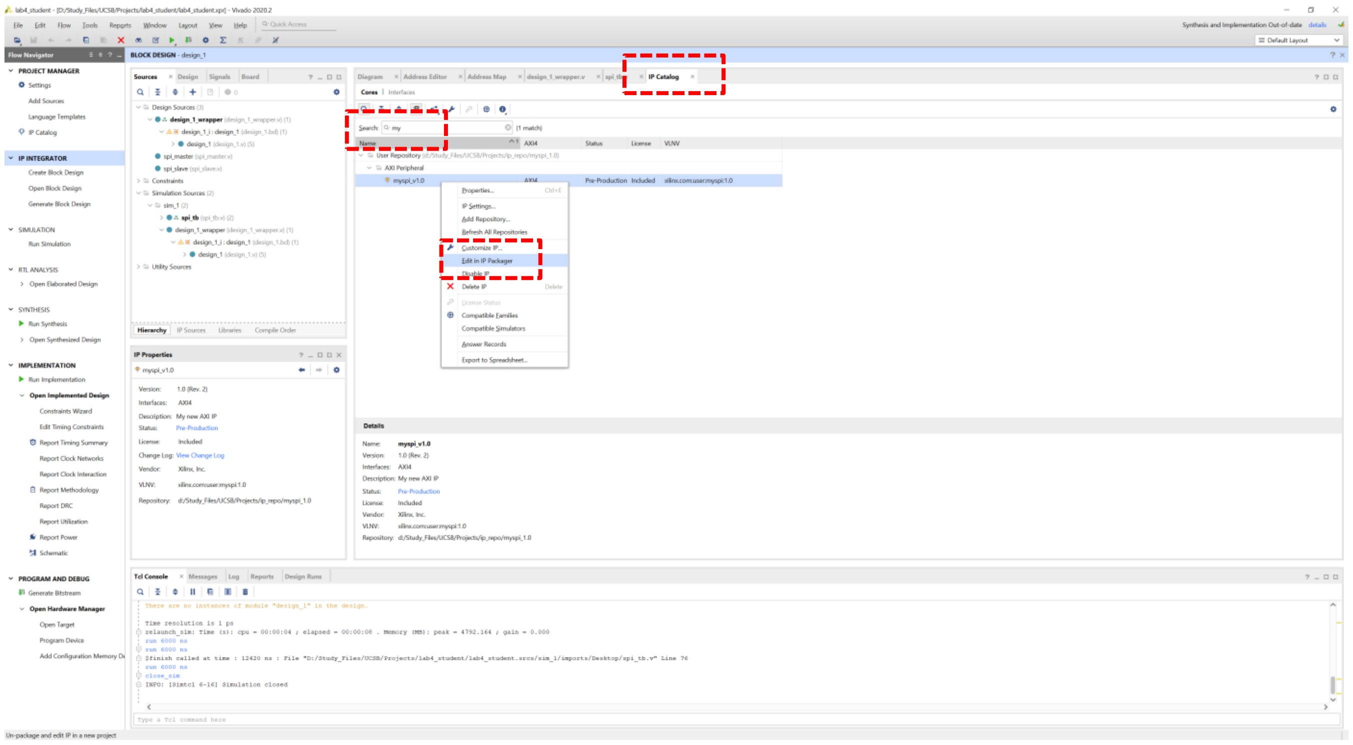
D:/Study_Files/UCSB/Projects/lab4_student/lab4_student.srcs/sim_1/imports/Desktop/spi_tb.v

```
`timescale 1ns / 1ps

module spi_tb;

// Testbench uses reg for inputs to the master and wire for connections between master and slave
reg          clk_tb;
reg          rst_n_tb;
reg  [31:0]  ctrl_tb;
reg  [31:0]  data_tx_tb;

wire [31:0]  data_rd_tb;
wire         spi_clk_tb;
wire         cs_n_tb;
wire         spi_o_master_tb;
wire         spi_o_slave_tb;

// Instantiate the spi_master module
spi_master master (
    .clk(clk_tb),
    .rst_n(rst_n_tb),
    .ctrl(ctrl_tb),
    .data_tx(data_tx_tb),
    .data_rd(data_rd_tb),
    .spi_i(spi_o_slave_tb),    // Connect spi_o from slave to spi_i of master
    .spi_clk(spi_clk_tb),
    .cs_n(cs_n_tb),
    .spi_o(spi_o_master_tb)
);
```

```
There are no instances of module "design_1" in the design.

Time resolution is 1 ps
relaunch_sim: Time (s): cpu = 00:00:04 ; elapsed = 00:00:08 . Memory (MB): peak = 4792.164 ; gain = 0.000
run 6000 ns
run 6000 ns
$finish called at time : 12420 ns : File "D:/Study_Files/UCSB/Projects/lab4_student/lab4_student.srcs/sim_1/imports/Desktop/spi_tb.v" Line 76
run 6000 ns
close_sim
INFO: [Simtcl 6-16] Simulation closed
```

Browse, customize, and generate cores

Figure 9: Open IP catalog tab.


lab4_student - [D:/Study_Files/UCSB/Projects/lab4_student/lab4_student.xpr] - Vivado 2020.2
File Edit Flow Tools Reports Window Layout View Help
BLOCK DESIGN - design_1
Sources Design Signals Board
Diagram Address Editor Address Map design_1_wrapper.v spi_tb.v IP Catalog
Cores Interfaces
Search: my (1 match)
Name AXI4 Status License VLNV
User Repository
AXI Peripheral
myspi_v1.0 AXI4 Pre-Production Included xilinx.com:user:myspi:1.0
Properties...
IP Settings...
Add Repository...
Refresh All Repositories
Customize IP...
Edit in IP Packager
Disable IP
Delete IP
License Status
Compatible Families
Compatible Simulators
Answer Records
Export to Spreadsheet...

Design Sources (3)
design_1_wrapper (design_1_wrapper.v) (1)
design_1_i : design_1 (design_1.bd) (1)
design_1 (design_1.v) (5)
spi_master (spi_master.v)
spi_slave (spi_slave.v)
Constraints
Simulation Sources (2)
sim_1 (2)
spi_tb (spi_tb.v) (2)
design_1_wrapper (design_1_wrapper.v) (1)
design_1_i : design_1 (design_1.bd) (1)
design_1 (design_1.v) (5)
Utility Sources

Details

Name: myspi_v1.0
Version: 1.0 (Rev. 2)
Interfaces: AXI4
Description: My new AXI IP
Status: Pre-Production
License: Included
Vendor: Xilinx, Inc.
VLNV: xilinx.com:user:myspi:1.0
Repository: d:/Study_Files/UCSB/Projects/ip_repo/myspi_1.0

Un-package and edit IP in a new project

Figure 10: Search and edit IP in IP packager.

2) Add your completed “spi_master.v” and the provided “spi_slave.v” into the “Design Resources”, as in Figure 11.

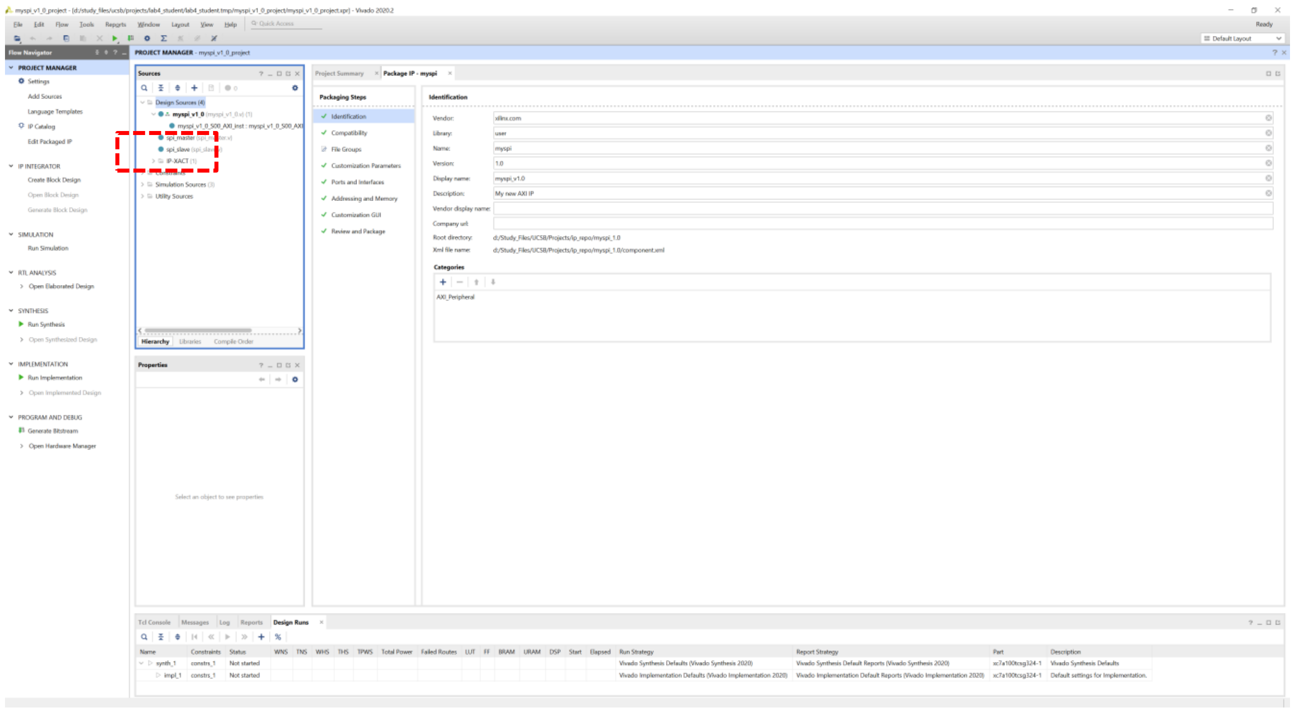
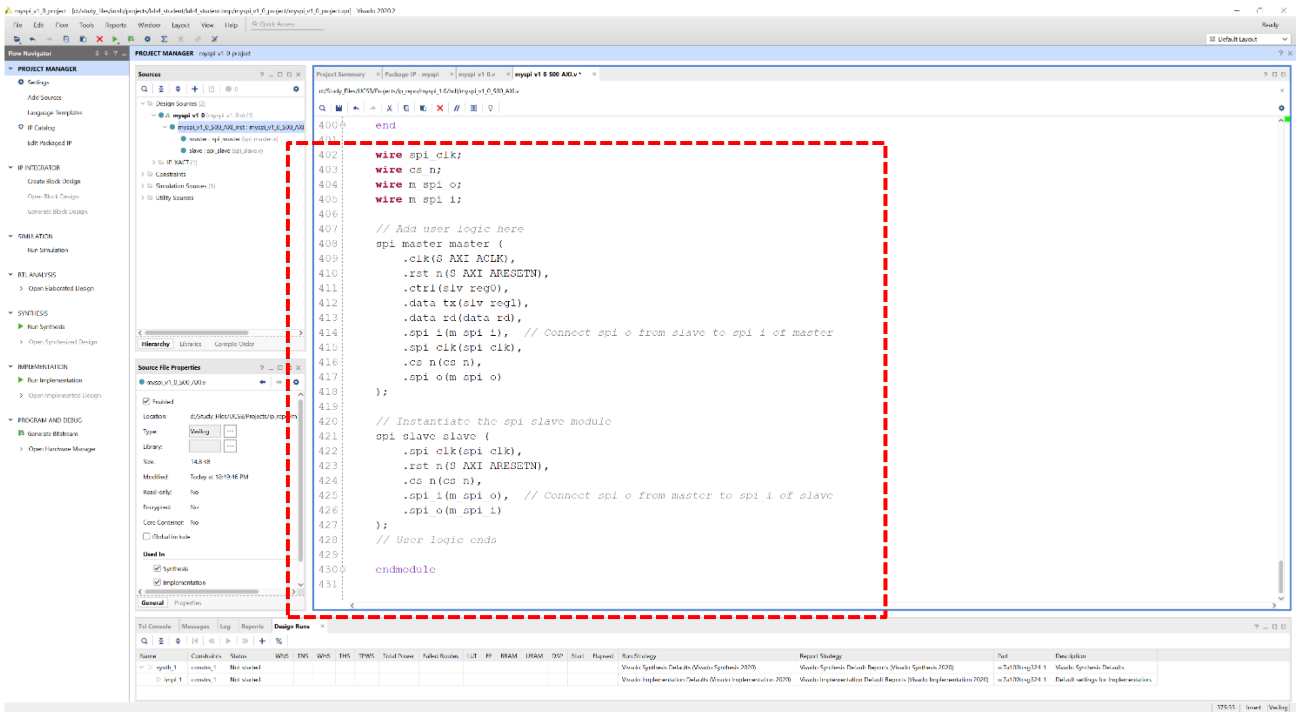


Figure 11: Add “spi_master.v” and “spi_slave.v” to the design.

3) Open “myspi_v1_0_S00_AXI.v” instantiate the “spi_master” and “spi_slave” modules, then connect them correctly to corresponding port and registers following the system diagram in Figure 7. An incomplete example of the instantiation is in Figure 12.

- You need to figure out where to specify the wire “data_rd” and modify the register write logic in the template code to allow “slv_reg2” to read from the “data_rd”.
- Note: You MUST solve this issue before you can proceed.



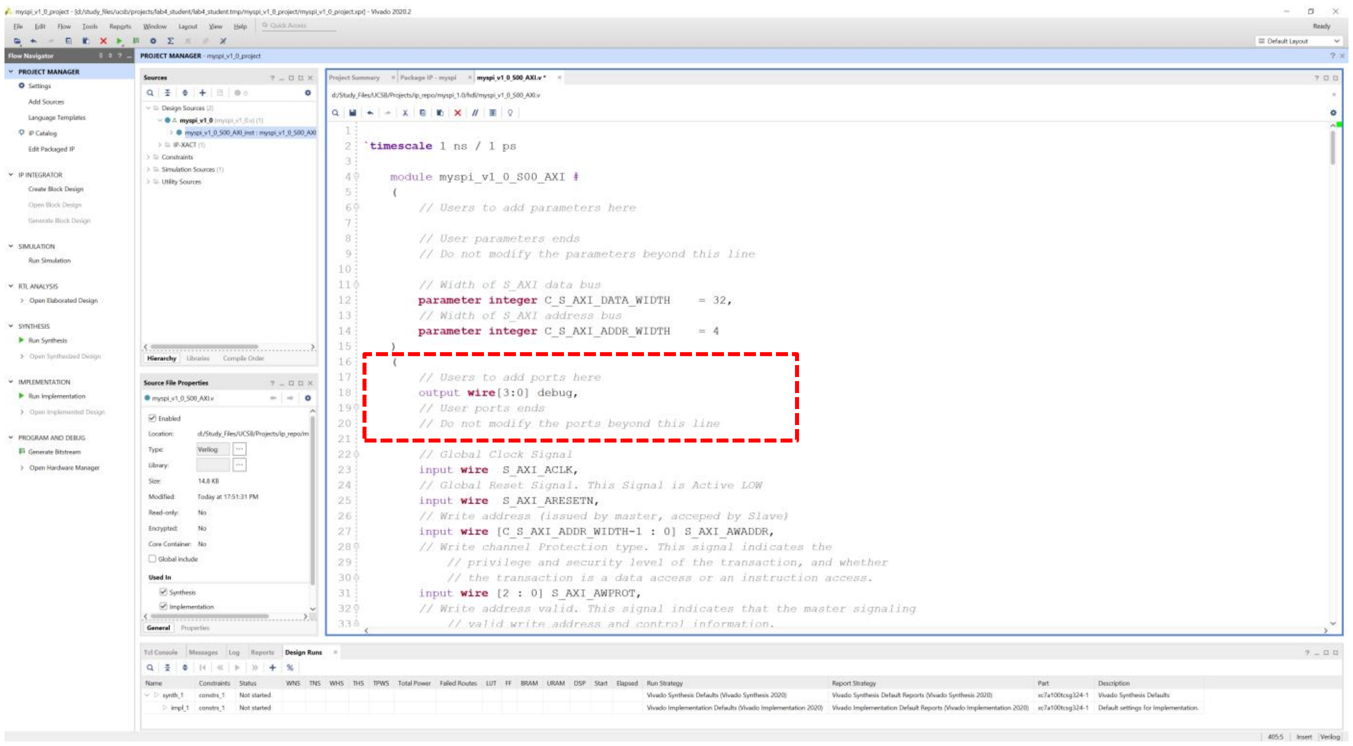
```
    end

    wire spi_clk;
    wire cs_n;
    wire m_spi_o;
    wire m_spi_i;

    // Add user logic here
    spi_master master (
        .clk(S_AXI_ACLK),
        .rst_n(S_AXI_ARESETN),
        .ctrl(slv_reg0),
        .data_tx(slv_reg1),
        .data_rd(data_rd),
        .spi_i(m_spi_i),  // Connect spi_o from slave to spi_i of master
        .spi_clk(spi_clk),
        .cs_n(cs_n),
        .spi_o(m_spi_o)
    );

    // Instantiate the spi slave module
    spi_slave slave (
        .spi_clk(spi_clk),
        .rst_n(S_AXI_ARESETN),
        .cs_n(cs_n),
        .spi_i(m_spi_o),  // Connect spi_o from master to spi_i of slave
        .spi_o(m_spi_i)
    );
    // User logic ends

    endmodule
```

Figure 12: Example, incomplete instantiation of modules in the template.

Step 2. Create Debug Port for ILA:

Sometimes, we want to use ILA to monitor the exact waveforms of certain signals to help debugging, instead of just read register values via JTAG-AXI. To do so, we need to manually create a debug port and connect the signals that you are interested in. Similar in Lab 3, let's monitor the "spi_clk", "cs_n", "m_spi_o" and "m_spi_i" signals.

1) As in Figure 13, declare a four-bit output wire "debug".
2) Add the line assign debug = {spi_clk, cs_n, m_spi_o, m_spi_i}; at line 406 in Figure 12.



```
`timescale 1 ns / 1 ps

    module myspi_v1_0_S00_AXI #
    (
        // Users to add parameters here

        // User parameters ends
        // Do not modify the parameters beyond this line

        // Width of S_AXI data bus
        parameter integer C_S_AXI_DATA_WIDTH    = 32,
        // Width of S_AXI address bus
        parameter integer C_S_AXI_ADDR_WIDTH    = 4
    )
    (
        // Users to add ports here
        output wire[3:0] debug,
        // User ports ends
        // Do not modify the ports beyond this line

        // Global Clock Signal
        input wire  S_AXI_ACLK,
        // Global Reset Signal. This Signal is Active LOW
        input wire  S_AXI_ARESETN,
        // Write address (issued by master, acceped by Slave)
        input wire [C_S_AXI_ADDR_WIDTH-1 : 0] S_AXI_AWADDR,
        // Write channel Protection type. This signal indicates the
            // privilege and security level of the transaction, and whether
            // the transaction is a data access or an instruction access.
        input wire [2 : 0] S_AXI_AWPROT,
        // Write address valid. This signal indicates that the master signaling
            // valid write address and control information.
```

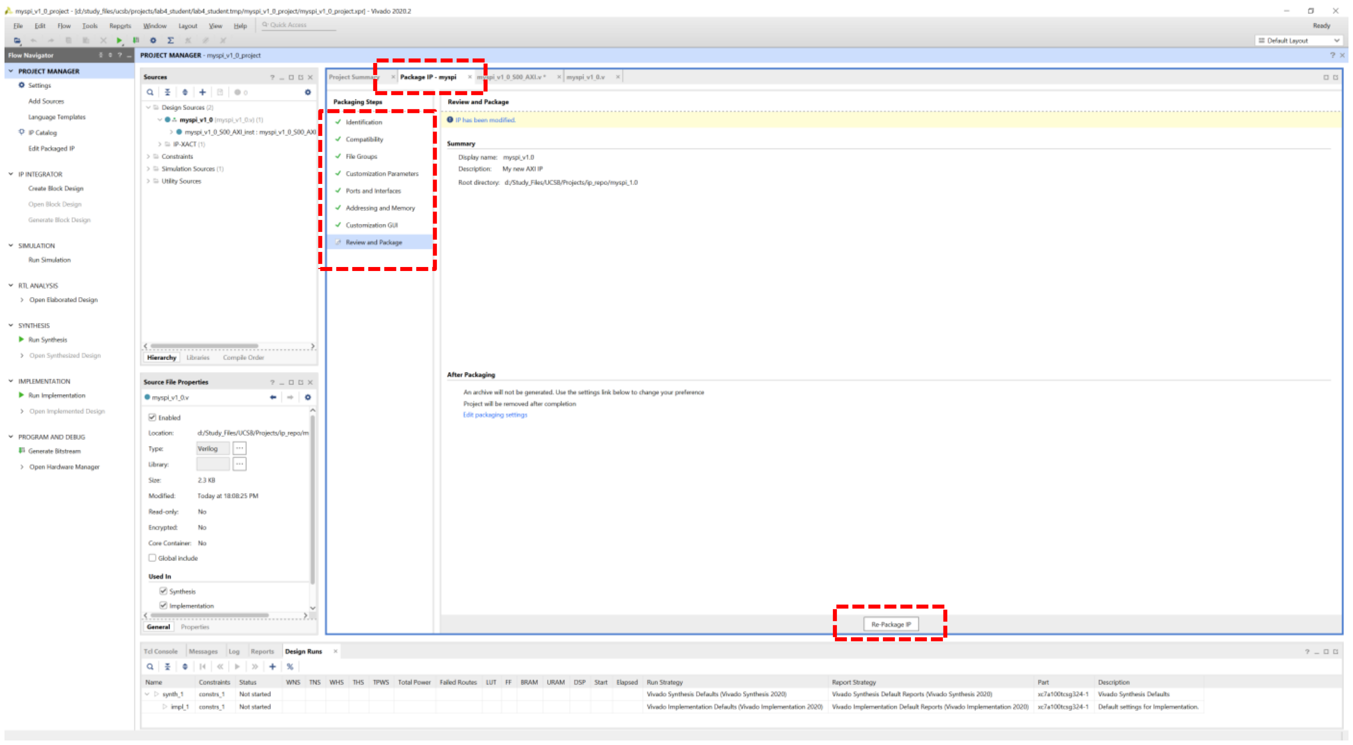
Figure 13: Add debug port to the design.

3) Note, the file we edit right now (myspi_v1_0_S00_AXI.v) is instantiated in "myspi_v1_0.v". This means we also need to add the debug port in "myspi_v1_0.v".

Step 3. Repackage IP:

Since the IP kernel and the ports have been modified, we need to repackage the IP.

1) Go to "Package IP – myspi", under "File Groups", click "Merge Changes from File Groups Wizard".
2) Do the same thing for "Customization Parameters".
3) Under "Ports and Interfaces", you should see "debug" appears as a 4-bit output.
4) As in Figure 14, You should see all sub tabs are green checked. Go to "Review and Package", click "Re-package IP".
5) Click "Yes" and we return to the original project window.



Review and Package

IP has been modified.

Summary

Display name: myspi_v1.0
Description: My new AXI IP
Root directory: d:/Study_Files/UCSB/Projects/ip_repo/myspi_1.0

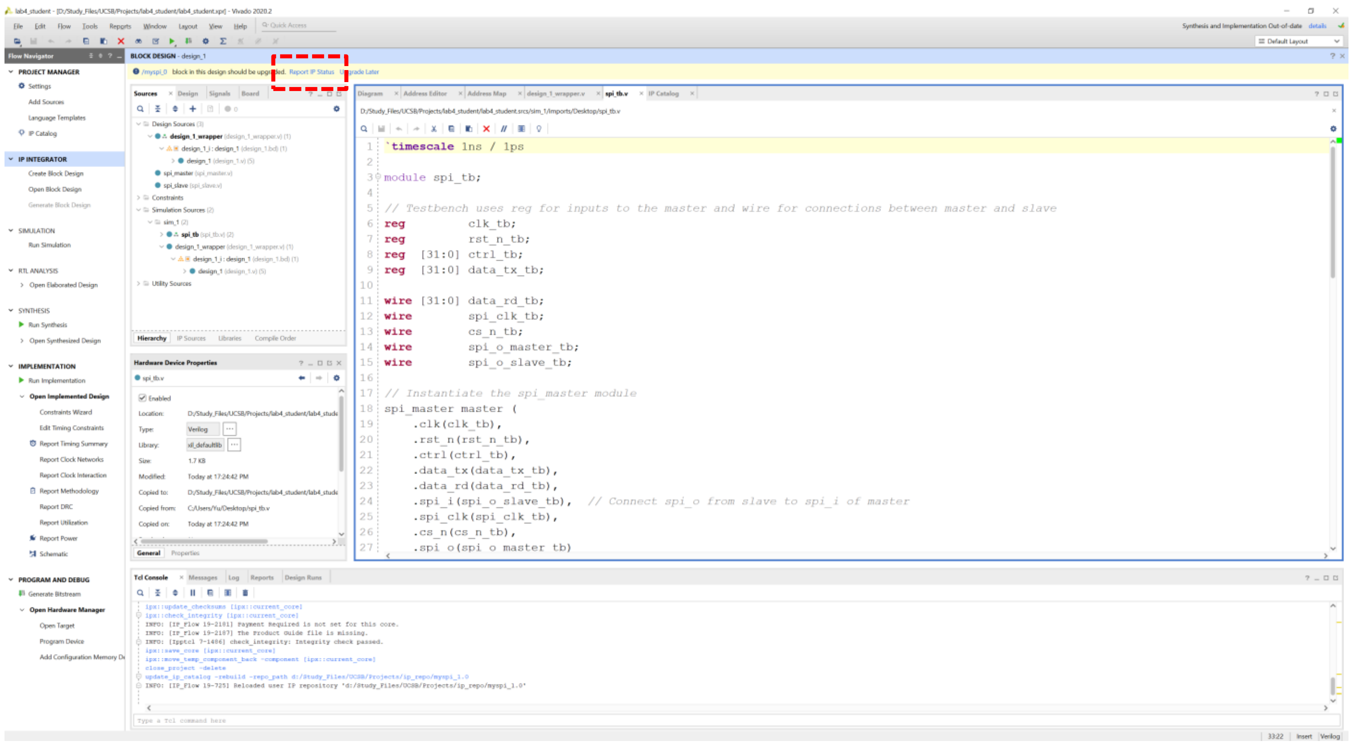
After Packaging

An archive will not be generated. Use the settings link below to change your preference
Project will be removed after completion
Edit packaging settings

Figure 14: Repackage IP.

Step 4. Report IP status

1) You will be notified that you need to report the IP status, as in Figure 15. Click on “Upgrade Selected”, as in Figure 16. In the popped window named “Generate Output Product”, click “Skip”.



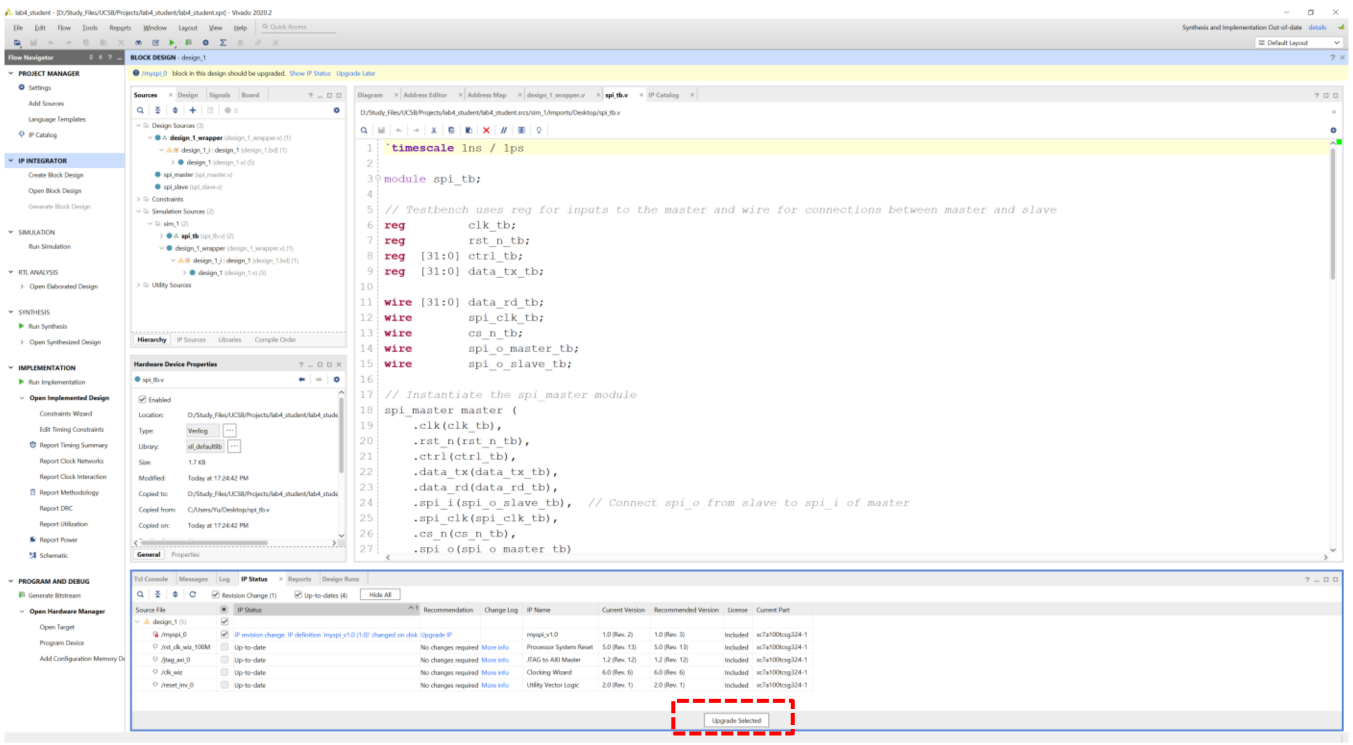
/myspi_0 block in this design should be upgraded. Report IP Status Upgrade Later

D:/Study_Files/UCSB/Projects/lab4_student/lab4_student.srcs/sim_1/imports/Desktop/spi_tb.v

```
`timescale 1ns / 1ps

module spi_tb;

// Testbench uses reg for inputs to the master and wire for connections between master and slave
reg         clk_tb;
reg         rst_n_tb;
reg  [31:0] ctrl_tb;
reg  [31:0] data_tx_tb;

wire [31:0] data_rd_tb;
wire        spi_clk_tb;
wire        cs_n_tb;
wire        spi_o_master_tb;
wire        spi_o_slave_tb;

// Instantiate the spi_master module
spi_master master (
    .clk(clk_tb),
    .rst_n(rst_n_tb),
    .ctrl(ctrl_tb),
    .data_tx(data_tx_tb),
    .data_rd(data_rd_tb),
    .spi_i(spi_o_slave_tb),  // Connect spi_o from slave to spi_i of master
    .spi_clk(spi_clk_tb),
    .cs_n(cs_n_tb),
    .spi_o(spi_o_master_tb)
```

```
ipx::update_checksums [ipx::current_core]
ipx::check_integrity [ipx::current_core]
INFO: [IP_Flow 19-2181] Payment Required is not set for this core.
INFO: [IP_Flow 19-2187] The Product Guide file is missing.
INFO: [Ipptcl 7-1486] check_integrity: Integrity check passed.
ipx::save_core [ipx::current_core]
ipx::move_temp_component_back -component [ipx::current_core]
close_project -delete
update_ip_catalog -rebuild -repo_path d:/Study_Files/UCSB/Projects/ip_repo/myspi_1.0
INFO: [IP_Flow 19-725] Reloaded user IP repository 'd:/Study_Files/UCSB/Projects/ip_repo/myspi_1.0'
```

Figure 15: Return to original design and report IP status.



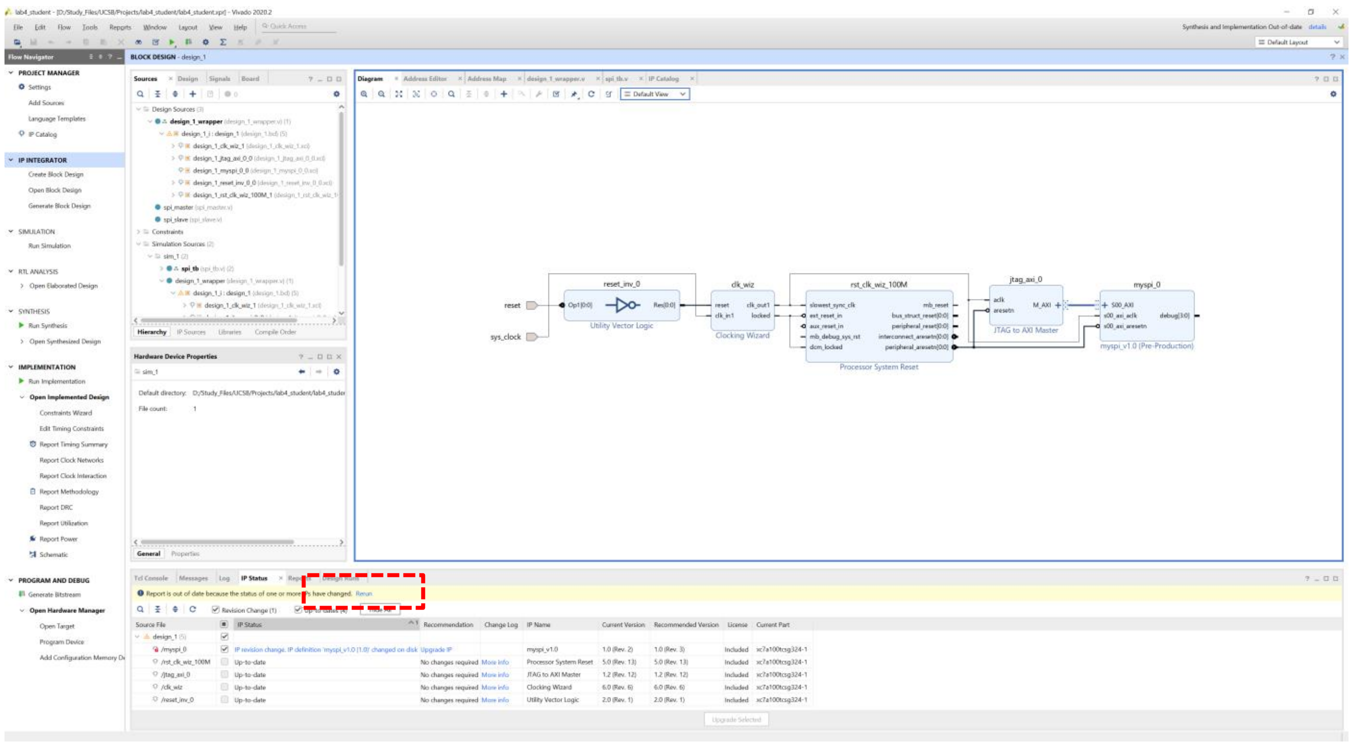
D:/Study_Files/UCSB/Projects/lab4_student/lab4_student.srcs/sim_1/imports/Desktop/spi_tb.v

```
`timescale 1ns / 1ps

module spi_tb;

// Testbench uses reg for inputs to the master and wire for connections between master and slave
reg         clk_tb;
reg         rst_n_tb;
reg  [31:0] ctrl_tb;
reg  [31:0] data_tx_tb;

wire [31:0] data_rd_tb;
wire        spi_clk_tb;
wire        cs_n_tb;
wire        spi_o_master_tb;
wire        spi_o_slave_tb;

// Instantiate the spi_master module
spi_master master (
    .clk(clk_tb),
    .rst_n(rst_n_tb),
    .ctrl(ctrl_tb),
    .data_tx(data_tx_tb),
    .data_rd(data_rd_tb),
    .spi_i(spi_o_slave_tb),  // Connect spi_o from slave to spi_i of master
    .spi_clk(spi_clk_tb),
    .cs_n(cs_n_tb),
    .spi_o(spi_o_master_tb)
```

Source File		IP Status	Recommendation	Change Log	IP Name	Current Version	Recommended Version	License	Current Part
design_1 (5)	✓								
/myspi_0	✓	IP revision change. IP definition 'myspi_v1.0 (1.0)' changed on disk	Upgrade IP		myspi_v1.0	1.0 (Rev. 2)	1.0 (Rev. 3)	Included	xc7a100tcsg324-1
/rst_clk_wiz_100M		Up-to-date	No changes required	More info	Processor System Reset	5.0 (Rev. 13)	5.0 (Rev. 13)	Included	xc7a100tcsg324-1
/jtag_axi_0		Up-to-date	No changes required	More info	JTAG to AXI Master	1.2 (Rev. 12)	1.2 (Rev. 12)	Included	xc7a100tcsg324-1
/clk_wiz		Up-to-date	No changes required	More info	Clocking Wizard	6.0 (Rev. 6)	6.0 (Rev. 6)	Included	xc7a100tcsg324-1
/reset_inv_0		Up-to-date	No changes required	More info	Utility Vector Logic	2.0 (Rev. 1)	2.0 (Rev. 1)	Included	xc7a100tcsg324-1

Figure 16: Return to original design and report IP status (cont).

2) Click "OK" if a critical warning message window appears.
3) As in Figure 17, click "rerun".



Report is out of date because the status of one or more IPs have changed. Rerun

Source File		IP Status	Recommendation	Change Log	IP Name	Current Version	Recommended Version	License	Current Part
design_1 (5)	✓								
/myspi_0	✓	IP revision change. IP definition 'myspi_v1.0 (1.0)' changed on disk	Upgrade IP		myspi_v1.0	1.0 (Rev. 2)	1.0 (Rev. 3)	Included	xc7a100tcsg324-1
/rst_clk_wiz_100M		Up-to-date	No changes required	More info	Processor System Reset	5.0 (Rev. 13)	5.0 (Rev. 13)	Included	xc7a100tcsg324-1
/jtag_axi_0		Up-to-date	No changes required	More info	JTAG to AXI Master	1.2 (Rev. 12)	1.2 (Rev. 12)	Included	xc7a100tcsg324-1
/clk_wiz		Up-to-date	No changes required	More info	Clocking Wizard	6.0 (Rev. 6)	6.0 (Rev. 6)	Included	xc7a100tcsg324-1
/reset_inv_0		Up-to-date	No changes required	More info	Utility Vector Logic	2.0 (Rev. 1)	2.0 (Rev. 1)	Included	xc7a100tcsg324-1

Figure 17: Return to original design and report IP status (cont).

Step 5. Recreate the Block Diagram Design and HDL Wrapper:

Since the IP has changed, we need to recreate all block design and add ILA.

1) Delete all components in the current block design.
2) Add “JTAG to AXI Master” and “myspi_v1.0”. Double click on “JTAG to AXI Master” and configure the “AXI Protocol” to be “AXI4LITE”.
3) Add “ILA”. Double click on the “ILA” IP. Set “Monitor Type” as “Native” and change the sample depth to 4096. Click on the “Probe_Ports(0…0)” tab and set the “Probe Width” to 4.
4) Click “Run Connection Automation”, check “myspi” only. Click “OK”.
5) Select and delete “AXI Interconnect”. Next, on the “Processor System Reset” block, connect the “ext_reset_in” input to “locked” output of Clocking Wizard.
6) Manually connect “debug[3:0]” of “myspi_v1.0” to “probe[3:0]” of ILA.
7) Click “Run Connection Automation”, check all items and click “OK”.
8) Right click on blank area and click “Regenerate Layout”, you block diagram should look like Figure 18.
9) Click on “Address Editor” and take a look on the base address.

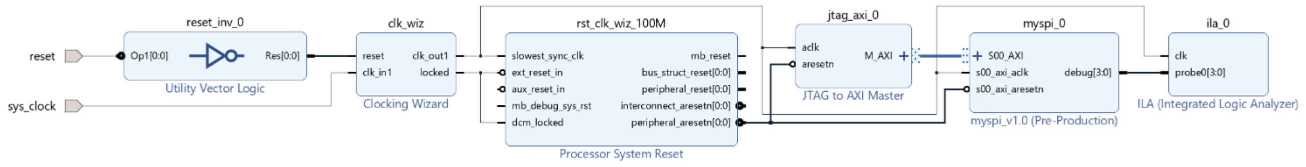


Figure 18: Final block diagram for part B.

10) Recreate the HDL wrapper, set it as the top if it is not. Then run implementation.

Step 6. Test on Board:

1) Generate the bitstream and program your device again.
2) Set the probe trigger condition as in Lab 3, but do not turn it on yet.
3) Follow the instruction in part_b.txt and execute the commands in order. Demonstrate to your TA with your correct output.
4) Turn on the ILA trigger, rerun commands in part_b.txt. Demonstrate your correct wave form to your TA.

5. Lab Report (60 pt)

Other than regular contents, your lab report should also contain the following parts:

1) Your illustrated design of the FSM. Explain clearly what are the states, what are the transition conditions and how output values change at different states.
2) Your correct behavior simulation results, the correct TCL output screenshot and the correct ILA detected waveform screenshot in part B2, step 6.
3) Summarize the lab and explain the procedure you expect when dealing with large system level design on FPGA in the future.