IN5200 Lab Assignment 5

SPI Protocol Implementation and Verification

Lab goals

- Understand the Serial Peripheral Interface (SPI) protocol.
- Implement SPI protocol connection to external component in VHDL.
- Implement 4-wire SPI SystemVerilog/UVM Agent
- Verification of the implementation of the SPI protocol and the SPI 4-wire Agent with SoC processor read and write accesses.
- Whitebox verification of SPI VHDL module with bind statements**.**

Deliverables
For passing this lab, a zip file archive named in the format
lab5_UseName_FirstName_FamilyName
(e.g., lab5_hsimpson_Homer_Simpson.zip) has to be uploaded to Canvas.

The submission deadline is given on the course semester page.

The file has to contain report, files and screenshots as shown in the appendix.

Lab setup

The MAX31723PMB1 module has to be in the JA1 connector on the ZEDBOARD.

Download files from Labs/Lab5 on the course semester page.

Task 1: Simulation of DTI control module loop.

The DTI module consist of the modules spictrl, dti_reg, fifo_512x16bit and dti_spi.

The dti_reg module is in directory if/psif/hdl/pif/dti, the fifo_512x16bit module is in the managed IP project in directory ip/zedboard_xc7z020/fifo_512x16bit. The spictrl and dti_spi modules are in the directory core/dti/hdl.

The dti_spi module currently only has a dummy architecture.

The DTI module register addresses are in the package if/psif/packages/psif_dti_pck.vhd.

Study the DTI module, the simulation case case_psif_dti_spi_loop, and run the simulation case case_psif_dti_spi that for now only simulates the spictrl loop.

Task 2: Implementation of the SPI protocol in the dti_spi module.

Study the Silica_SPI_and MAX31723 paper and the MAX31723 data sheets.

Implement a 4-wire SPI protocol according to the MAX31723 data sheet in an architecture called rtl in the file *dti_spi_rtl.vhd* and use this module instead of the dmy architecture (i.e. dti_spi_dmy.vhd) in simulations (i.e. Questa compilation script) and in Vivado project mode.

Hint 2.0: Let the SPI output signals be 1/32 of the 100 MHz clock. This can be accomplished through counting clock cycles and changing the values of the sclk and other output signals every 32 100MHz clk cycle.

Hint 2.1: Study the MAX31722/MAX31723 data sheet with emphasis on SPI read in figure 3 and SPI write in figure 4.

Task 3: Implementation of SPI 4-wire SystemVerilog/UVM Agent.

Implement a SystemVerilog/UVM agent in a new *vip/spi_4wire_agent* directory for simulation of the 4-wire SPI interface protocol and integrate the agent in the SoC testbench environment.

The agent shall have a *spi_4wire_agent_item* class with the 8-bit logic vector transaction items *rw_address, write_data* and *read_data*.

Implement the *spi_4wire_agent_item* class and include a virtual function void *displayAlldata()* method in the *spi_4wire_agent_item* class that displays all item data as three uvm_info messages.

The agent shall be connected to the DUT in the testbench with an interface *spi_4wire_agent_if.* Implement the interface with the five logic variables *clk* as input parameters, and chip enable *ce,* serial clock *sclk*, serial data in *sdi* and serial data out *sdo* as *logic* variables.

The implemented *spi_4wire_agent* agent should use the given driver and monitor classes and be based on the SystemVerilog/UVM agents *reset_agent* and *oled_spi_agent*. The reset agent uses the sequence top_reset_seq in /top/tb_base directory. The *oled_spi_agent* is a passive agent without driver class, but with a monitor and hence do not have an associated sequence.

It is important that the spi_4wire_agent config class is set to be an active agent (i.e. UVM_ACTIVE) such as done in the reset_agent config class!

Connect the *spi_4wire_agent* to the DUT in *top/tb/tb_top_beh.sv* with the *spi_4wire_agent_if* interface. Create and connect the *spi_4wire_agent* to the scoreboard and coverage classes in the testbench environment in *top/tb/tb_env.sv*.

Study the given sequence *spi_4wire_seq* and include it in the directory *top/tb* and in the design to communicate with the 4-wire SPI agent.

Add the *spi_4wire_agent* to the *base_test.svh* and *base_test_pkg.sv*, and add compilation of the *spi_4wire_agent* package in the compilation script *comp_tb.tcl* (i.e. remove comment statements). Run compilation of the design with the *comp_all.tcl* script.

Rerun the simulation case *case_psif_dti_spi_loop* that for now only simulates the spictrl loop to check that the design still works for this simulation case (i.e. regression testing).

Hint 3.0: The videos/slides in “Architecting a UVM Testbench” and “The Proper Care and Feeding of Sequences” in the Advanced UVM course are about this topic.

Hint 3.1: The driver first receive the address and send the address to the receiving sequence as an item when all address bits have been received on positive *sclk* edge.

Hint 3.2: After the address have been received the driver check the address in the driver for SPI write or read transaction. For write it sends the received sdi data as write_data to the sequence and for read it receives the data from the sequence and transmits data as sdo data.

Hint 3.3: The driver initializes the *sdo* signal to zero in the driver before execution of the driver starts; i.e. before the forever-loop.

Explain the implementation of the agent with emphasis on structure and explain what statements like *@(posedge m_cfg.spi_4wire_if.ce), @(posedge m_cfg.spi_4wire_if.sclk), m_cfg.spi_4wire_if.sdi* and *m_cfg.spi_4wire_if.sdo* do in the driver.

Task 4: Verification of SPI 4-wire connection with MAX31723.

The sequence *spi_4wire_seq* have registers 0-6 to simulate the registers in the MAX31723 circuit with registers having write enable to simply testing (i.e. not as shown in the data sheet).

Check the coverage class coverage_dti_spi in top/tb directory that a requirement of read and write access to all registers 0-6 are met.

Run the full simulation case case_psif_dti_spi. Check the coverage of the dti_spi_cg cover group.

Task 5: Whitebox verification of SPI protocol module.

The simulation case *case_psif_dti_spi_module* simulates the dti_spi module through test sequence access to the internal signals from the spictrl module.

Run the whitebox module simulation for the simulation case *case_psif_dti_spi_module*.

Change the address data constraint from randc to rand the simulation case and observe simulation time. Why is the simulation time different?

Explain how the test case works.

Appendix

Report Template:

We do not demand a specific layout, but the report (to be delivered in PDF format within the ZIP-archive) shall include the following information:

Lab5: SPI Protocol Implementation and Verification

Name: Homer Simpson

Email (UIO): hsimpson@student.matnat.uio.no

Task 1:

Log file for spictrl loop simulation.

Task 2:

Commented source code for dti_spi_rtl.vhd.

Task 3:

Commented source code of the 4-wire SPI UVM agent.

Commented source code for the comp_tb.tcl script.

Answers to questions.

Task 4:

Commented source code for the coverage_dti_spi class.

Screenshot showing simulation coverage of the dti_spi_cg cover group.

Task 5:

Explain why the simulation time is different when the address data constraint is changed from randc to rand?

Explanation of how the test case works.