- DISCONTINUED PRODUCT -

LogiCORE™ IP SPI-3 Physical Layer v5.2

Getting Started Guide

UG094 April 24, 2009

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
8/31/05	1.1	Initial Xilinx release.
1/18/06	1.2	Updated version, date, and ISE tool version number
8/08/07	1.3	Updated version to 5.1, update to ISE v9.2i. Added support for Spartan-3A /3AN/3A DSP
4/24/09	1.4	Updated version to 5.2 and update to ISE v11.1. Removed support for Spartan-3A DSP.

09/29/09 - This is the final publication. No content was changed.

Table of Contents

About This Guide

The *SPI-3 Physical Layer v5.2 Getting Started Guide* provides information about generating a Xilinx LogiCORE™ IP SPI-3 Physical (PHY) Layer core, customizing and simulating the core using the provided example design, and running the design files through implementation using Xilinx tools.

Contents

This guide contains the following chapters:

- Preface, "About this Guide" introduces the organization and purpose of this guide and the conventions used in this document.
- Chapter 1, "Introduction," describes the SPI-3 PHY Layer core and related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- Chapter 2, "Licensing the Core," defines the installation methods and license types available for the core.
- Chapter 3, "Quick Start Example Design," describes how to generate a SPI-3 PHY Layer core and run the example design through implementation and simulation.
- Chapter 4, "Detailed Example Design," describes the demonstration test bench in detail and provides directions for customizing the demonstration test bench for use in an application.

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands you enter in a syntactical statement	**`ngdbuild`** `design_name`

Convention	Meaning or Use	Example
Italic font	Variables in a syntax statement for which you must supply values	See the *Development System Reference Guide* for more information.
	References to other manuals	See the *Product Specification* for details.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Dark Shading	Items that are not supported or reserved	This feature is not supported
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **bus[7:0]**, they are required.	**ngdbuild** [option_name] design_name
Braces { }	A list of items from which you must choose one or more	**lowpwr ={on|off}**
Vertical bar |	Separates items in a list of choices	**lowpwr ={on|off}**
Vertical ellipsis . . .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...	Omitted repetitive material	**allow block** block_name loc1 loc2 ... locn;
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returned 45524943h.
	An '_n' means the signal is active low	usr_teof_n is active low.

Online Documents

The following linking conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "Additional Resources" for details. See "Title Formats" in Chapter 1 for details.
Blue, underlined text	Hyperlink to a website (URL)	Go to www.xilinx.com for the latest speed files.

Introduction

The SPI-3 Physical (PHY) Layer core is a fully verified interconnection solution designed to support both Verilog and VHDL design environments. In addition, the example design documented in this guide is provided in both Verilog and VHDL.

This chapter introduces the SPI-3 PHY Layer core and provides related information, including recommended design experience, additional resources, technical support, and methods for submitting feedback to Xilinx.

About the Core

The SPI-3 PHY Layer core is a Xilinx CORE Generator™ IP core, included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, see the SPI-3 PHY Layer product page:

www.xilinx.com/products/ipcenter/DO-DI-POSL3PHY.htm

For information about system requirements, installation, and licensing options, see Chapter 2, "Licensing the Core."

Recommended Design Experience

Although the SPI-3 PHY Layer core is a fully verified solution, the challenge associated with implementing a complete design varies depending on the configuration and functionality of the application. For best results, previous experience building high performance, pipelined FPGA designs using Xilinx implementation software and user constraints files (.ucf) is recommended.

Additional Core Resources

For detailed information and updates about the SPI-3 PHY Layer core, see the following documents, located on the SPI-3 PHY Layer product page:

www.xilinx.com/products/ipcenter/DO-DI-POSL3PHY.htm

- *SPI-3 Physical Layer Product Specification*
- *SPI-3 Physical Layer Release Notes*

Technical Support

For technical support, go to www.xilinx.com/support. Questions are routed to a team of engineers with expertise using the SPI-3 PHY Layer core.

Xilinx provides technical support for use of this product as defined in the *SPI-3 Physical Layer Product Specification* and the *SPI-3 Physical Layer Getting Started Guide*. Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow these guidelines.

Feedback

Xilinx welcomes comments and suggestions about the SPI-3 PHY Layer core and the documentation supplied with the core.

Core

For comments or suggestions about the SPI-3 PHY Layer core, please submit a WebCase from www.xilinx.com/support/clearexpress/websupport.htm. Be sure to include the following information:

- Product name
- Core version number
- Explanation of your comments

Document

For comments or suggestions about this document, please submit a WebCase from www.xilinx.com/support/clearexpress/websupport.htm. Be sure to include the following information:

- Document title
- Document number
- Page number(s) to which your comments refer
- Explanation of your comments

Licensing the Core

This chapter provides instructions for installing the SPI-3 PHY Layer core and obtaining a license for the core, which you must do before using the core in your designs. The SPI-3 PHY Layer core is provided under the terms of the Xilinx LogiCORE Site License Agreement, which conforms to the terms of the SignOnce IP License standard defined by the Common License Consortium. Purchase of the core entitles you to technical support and access to updates for one year.

System Requirements

Windows

- Windows XP® Professional 32-bit/64-bit
- Windows Vista® Business 32-bit/64-bit

Linux

- Red Hat® Enterprise Linux WS v4.0 32-bit/64-bit
- Red Hat® Enterprise Desktop v5.0 32-bit/64-bit (with Workstation Option)
- SUSE Linux Enterprise (SLE) v10.1 32-bit/64-bit

Software

- ISE® 11.1 with applicable service pack

Check the release notes for the required service pack; ISE Service Packs can be downloaded from

www.xilinx.com/support/download/index.htm

Before you Begin

This chapter assumes that you have installed the core using either the CORE Generator™ IP Update installer or by performing a manual installation after downloading the core from the web. For information about installing the core, see the SPI-3 PHY product page.

Before installing the core, you must have a Xilinx.com account and the ISE v11.1 software installed on your system.

To set up an account and install the ISE software:

1. Click Sign in to Access Account at the top of the Xilinx home page; then follow the instructions to create a support account.
2. Install the ISE 11.1 software with the applicable service pack.

License Options

The SPI-3 PHY Layer core provides three licensing options. After installing the required Xilinx ISE software and IP Service Packs, choose a license option.

Simulation Only

The Simulation Only Evaluation license key is provided with the Xilinx CORE Generator tool. This key lets you assess core functionality with either the example design provided with the SPI-3 PHY Layer core, or alongside your own design and demonstrates the various interfaces to the core in simulation. (Functional simulation is supported by a dynamically generated HDL structural model.)

Full System Hardware Evaluation

The Full System Hardware Evaluation license is available at no cost and lets you fully integrate the core into an FPGA design, place-and-route the design, evaluate timing, and perform functional simulation of the SPI-3 PHY Layer core using the example design and demonstration test bench provided with the core.

In addition, the license key lets you generate a bitstream from the placed and routed design, which can then be downloaded to a supported device and tested in hardware. The core can be tested in the target device for a limited time before timing out (ceasing to function), at which time it can be reactivated by reconfiguring the device.

Full

The Full license key is available when you purchase the core and provides full access to all core functionality both in simulation and in hardware, including:

- Functional simulation support
- Full implementation support including place and route and bitstream generation
- Full functionality in the programmed device with no time outs

Obtaining Your License Key

This section contains information about obtaining a simulation, full system hardware, and full license keys.

Simulation License

No action is required to obtain the Simulation Only Evaluation license key; it is provided by default with the Xilinx CORE Generator software.

Full System Hardware Evaluation License

To obtain a Full System Hardware Evaluation license, do the following:

1. Navigate to the product page for this core: www.xilinx.com/products/ipcenter/DO-DI-POSL3PHY.htm
2. Click Evaluate.
3. Follow the instructions to install the required Xilinx ISE software and IP Service Packs.

Full License

To obtain a Full license key, you must purchase a license for the core. After doing so, click the "Access Core" link on the Xilinx.com IP core product page for further instructions.

Installing Your License File

The Simulation Only Evaluation license key is provided with the ISE CORE Generator system and does not require installation of an additional license file. For the Full System Hardware Evaluation license and the Full license, an email will be sent to you containing instructions for installing your license file. Additional details about IP license key installation can be found in the ISE Design Suite Installation, Licensing and Release Notes document.

Chapter 3

Quick Start Example Design

The quick start instructions provide a way to quickly generate a SPI-3 PHY Layer core, run the design through implementation using the Xilinx tools, and simulate the example design using the provided demonstration test bench. For detailed information about the example design, see Chapter 4, "Detailed Example Design."

Overview

The SPI-3 PHY Layer example design consists of the following:

- Transmit (TX) and Receive (RX) core netlists
- TX and RX core simulation models
- Example HDL wrapper (which instantiates the cores and example design)
- Demonstration test bench to simulate the example design

The SPI-3 PHY Layer example design has been tested with Xilinx ISE 11.1, Mentor Graphics® ModelSim® 6.4b and Cadence™ IUS v8.1-s009.

Generating the Core

Use the following steps to generate a SPI-3 PHY Layer core with default values using the Xilinx CORE Generator tool.

1. Start the CORE Generator.

   For help starting and using the CORE Generator tool, see Xilinx CORE Generator Help, available from the ISE Software Manuals.
2. Choose File > New Project.
3. Type a directory name. In this example, the directory name *proj* is used.
4. Do the following to set project options:
   - Part Options
     - From Target Architecture, select the desired family. For a list of supported families, see the *SPI-3 Physical Layer Data Sheet*.

       If an unsupported silicon family is selected, the SPI-3 PHY Layer core does not appear in the taxonomy tree.
   - Generation Options
     - For Design Entry, select either VHDL or Verilog.
     - For Vendor, select Synplicity or Other (XST).

5. After creating the project, locate the SPI-3 PHY Layer core in the taxonomy tree under Communication & Networking > Telecommunication > SPI-3 Physical Layer Interface.
6. Double-click the core to display the SPI-3 PHY Layer Interface main screen (Figure 3-1).

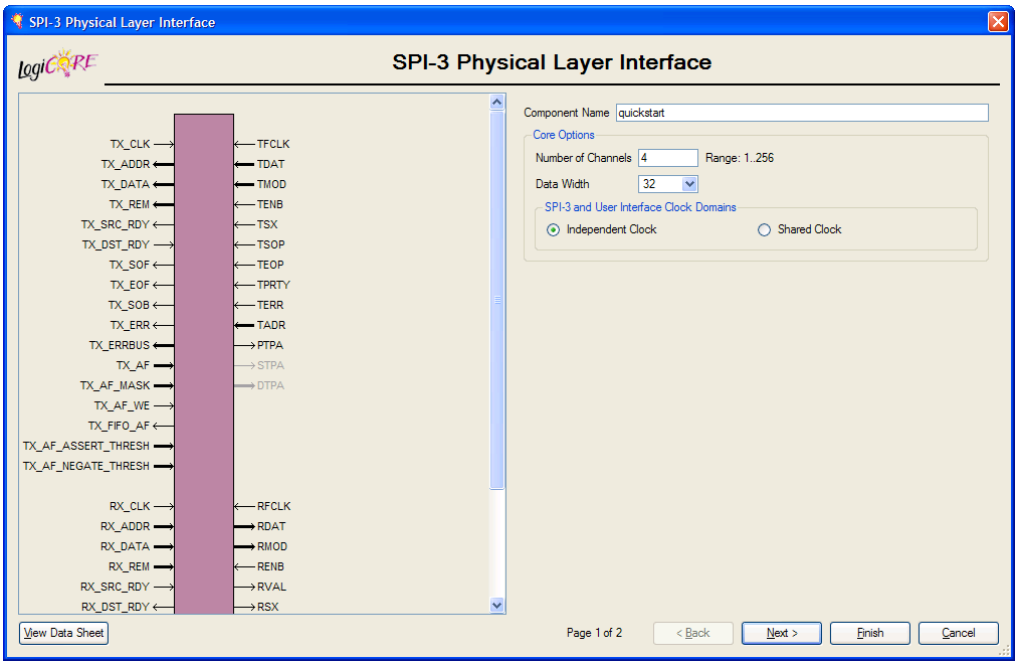

Figure 3-1: **SPI-3 PHY Layer Main Screen**

7. In the Component Name field, enter a name for the core instance. For this example, quickstart is used.
8. After selecting the desired features and parameters from the GUI screens, click Generate.

   The core and its supporting files, including the example design, are generated in the project directory. For detailed information about the example design files and directories see "Directory and File Contents," page 18.

Implementing the Example Design

After generating a core with a Full System Hardware Evaluation or Full license, the netlists and the example design can be processed by the Xilinx implementation tools. The generated output files include scripts to assist the user in running the Xilinx software.

To implement the SPI-3 PHY Layer example design, open a command prompt or terminal window and type the following commands:

For Windows

```
ms-dos> cd <proj>\quickstart\implement
ms-dos> implement.bat
```

For UNIX

```
unix-shell% cd <proj>/quickstart/implement
unix-shell% ./implement.sh
```

These commands execute a script that synthesizes, builds, maps, and place-and-routes the example design. The script then generates a post-par simulation model for use in timing simulation. The resulting files are placed in the `results` directory.

Simulating the Example Design

The SPI-3 PHY Layer core provides a quick way to simulate and observe the behavior of the core utilizing the provided example design. There are two different simulation types: functional and timing. The simulation models provided are either VHDL or Verilog, depending on the CORE Generator Design Entry project options.

Setting up for Simulation

The Xilinx UniSim and SimPrim libraries must be mapped into the simulator. If the UniSim or SimPrim libraries are not set for your environment, see the *Synthesis and Simulation Design Guide* available from the ISE documentation page.

Functional Simulation

Instructions for running a functional simulation of the SPI-3 PHY Layer core using either VHDL or Verilog are provided below. Functional simulation models are provided when the core is generated. Note that implementing the core before simulating the functional models is not required.

To run a VHDL or Verilog functional simulation of the example design using ModelSim:

1. Set the current directory to:
   `<proj>/quickstart/simulation/functional`
2. Launch the ModelSim simulation script:
   `vsim -do simulate_mti.do`

To run a VHDL or Verilog functional simulation of the example design using NCSIM:

1. Set the current directory to:
   `<proj>/quickstart/simulation/functional`
2. Execute the simulation script:
   `unix-shell% simulate_ncsim.sh`

The functional simulation script compiles the simulation model and the demonstration test bench, adds relevant signals to the wave window, and runs the simulation. To observe the operation of the core, inspect the simulation transcript and the waveform.

Timing Simulation

Timing simulation is available only with purchase of the core (Full license) or with access to the Full System Hardware Evaluation license. With a Simulation Only Evaluation license the core cannot be run through the implementation tools, which is required for timing based simulation.

Instructions for running a timing simulation of the SPI-3 Link core using either VHDL or Verilog are given below. A timing simulation model is generated when the core is run through the Xilinx tools using the implement script.

To run a VHDL or Verilog timing simulation of the example design using ModelSim:

1. Set the current directory to:
   `<proj>/quickstart/simulation/timing`
2. Launch the ModelSim simulator.
   `vsim -do simulate_mti.do`

To run a VHDL or Verilog timing simulation of the example design using NCSIM:

1. Set the current directory to:
   `<proj>/quickstart/simulation/timing`
2. Execute the simulation script:
   `unix-shell% simulate_ncsim.sh`

The simulation script compiles the timing simulation model and the demonstration test bench, adds relevant signals to the wave window, and runs the simulation. To observe the operation of the core, inspect the simulation transcript and the waveform.

Chapter 4

Detailed Example Design

This chapter provides detailed information about the example design, including a description of files and the directory structure generated by the Xilinx CORE Generator, the purpose and contents of the provided scripts, the contents of the example HDL wrappers, and the operation of the demonstration test bench.

- **<project directory>**
  Top-level project directory; name is user-defined
  - <project directory>/<component name>
    Core release notes file
    - <component name>/doc
      Product documentation
    - <component name>/example design
      Verilog or VHDL design files
    - <component name>/implement
      Implementation script files
      - implement/results
        Results directory, created after implementation scripts are run, and contains implement script results
    - <component name>/simulation
      Simulation scripts
      - simulation/functional
        Functional simulation files
      - simulation/timing
        Timing simulation files

Directory and File Contents

The SPI-3 PHY core directories and their associated files are defined below.

<project directory>

The CORE Generator provides the resulting TX and RX core files in the *<project_dir>* directory.

Table 4-1: **Project Directory**

Name	Description
<project_dir>	
<component_name>_spi3_phy_tx.ngc	Core netlist
<component_name>_spi3_phy_rx.ngc	Core netlist
<component_name>_spi3_phy_tx.v[hd]	Functional simulation model
<component_name>_spi3_phy_rx.v[hd]	Functional simulation model
<component_name>_spi3_phy_tx.v{ho|eo}	Instantiation template
<component_name>_spi3_phy_rx.v{ho|eo}	Instantiation template
<component_name>.xco	CORE Generator project-specific option file; can be used as an input to the CORE Generator.
<component_name>_flist.txt	List of files delivered with the core.

Back to Top

<project directory>/<component name>

The <component name> directory contains the release notes file provided with the core, which may include last-minute changes and updates.

Table 4-2: **Component Name Directory**

Name	Description
<project_dir>/<component_name>	
spi3_phy_readme.txt	The *SPI-3 Physical Layer Release Notes* text file.

Back to Top

<component name>/doc

The doc directory contains the PDF documentation provided with the core.

Table 4-3: **Doc Directory**

Name	Description
<project_dir>/<component_name>/doc	
spi3_phy_ds339.pdf	The *SPI-3 Physical Layer Data Sheet*.

Table 4-3: **Doc Directory *(Continued)***

Name	Description
spi3_phy_gsg94.pdf	The *SPI-3 Physical Layer Getting Started Guide*.

Back to Top

<component name>/example design

The example design directory contains the example design files provided with the core.

Table 4-4: **Example Design Directory**

Name	Description
<project_dir>/<component_name>/example_design	
<component_name>_top.ucf	The user constraints file (UCF) for both the core and the example design provides example constraints necessary for processing the SPI-3 PHY Layer core using the Xilinx implementation tools. The UCF can be modified by the user to meet individual system requirements. The UCF contains a timing constraint on both the TX and RX clocks.
<component_name>_top.v[hd]	The VHDL or Verilog top-level file for the example design; instantiates the TX and RX cores.

Back to Top

<component name>/implement

The implement directory contains the core implementation script files. Note that the implement directory is only generated for Full-System Hardware Evaluation and Full license types. See Chapter 2, "Licensing the Core," for licensing information.

Table 4-5: **Implement Directory**

Name	Description
<project_dir>/<component_name>/implement	
implement.{sh | bat}	A Windows (.bat) or UNIX (.sh) script that processes the example design through the Xilinx tool flow. For more information, see "Implementation Script Details," page 22.
xst.prj	The XST project file for the example design that lists all of the source files to be synthesized. Only available when the CORE Generator Vendor project option is set to ISE or Other.
xst.scr	The XST script file for the example design used to synthesize the core. Only available when the CORE Generator Vendor project option is set to ISE or Other.
synplify.prj	The Synplify synthesis script for the example design. Only available when the CORE Generator Vendor project option is set to Synplicity.

Back to Top

implement/results

The results directory is created by the implement script. Implement script results are placed in this directory.

Table 4-6: **Results Directory**

Name	Description
<project_dir>/<component_name>/implement/results	
Implement script result files.	

Back to Top

<component name>/simulation

A directory containing the necessary files to test a VHDL or Verilog example design is provided with the demonstration test bench.

Table 4-7: **Simulation Directory**

Name	Description
<project_dir>/<component_name>/simulation	
spi3_phy_clk_gen.v[hd] spi3_phy_driver.v[hd] spi3_phy_monitor.v[hd] spi3_phy_procedures.v[hd] spi3_phy_tb.v[hd] spi3_phy_testcase_pkg.v[hd]	Two files, spi3_phy_driver and spi3_phy_testcase_pkg control the operation of the demonstration test bench and can be modified by the user. For information about modifying these files, see "Customizing the Demonstration Test Bench," page 25.

Back to Top

simulation/functional

The functional directory contains functional simulation scripts provided with the core.

Table 4-8: **Functional Directory**

Name	Description
<project_dir>/<component_name>/simulation/functional	
simulate_mti.do	A ModelSim macro file that compiles the HDL sources and runs the simulation.
wave.do	A ModelSim macro file that opens a wave window and adds key signals to the wave viewer. The wave.do file is called by the simulate_mti.do file, and is displayed after the simulation is loaded.
simulate_ncsim.sh	Shell script that compiles the HDL sources and runs the simulation.
wave.sv	NCSIM macro file that opens a wave window and adds key signals to the wave viewer. The wave.sv file is called by the simulate_ncsim.sh script.

Back to Top

simulation/timing

Note that the timing simulation scripts are only generated for Full-System Hardware Evaluation and Full license types.

Table 4-9: **Functional Directory**

Name	Description
<project_dir>/<component_name>/simulation/timing	
simulate_mti.do	A ModelSim macro file that compiles the post-par timing netlist and demonstration test bench files and runs the simulation. Note that the implement script must be run to generate the post-par timing simulation model. Simulation can only be run after the timing simulation model is generated.
wave.do	A ModelSim macro file that opens a wave window and adds key signals to the wave viewer. The wave.do file is called by the simulate_mti.do file, and is displayed after the simulation is loaded.
simulate_ncsim.sh	Shell script that compiles the post-par timing netlist and demonstration test bench, and runs the simulation.
wave.sv	NCSIM macro file that opens a wave window and adds key signals to the wave viewer. The wave.sv file is called by the simulate_ncsim.sh script.

Back to Top

Implementation and Simulation Scripts

Implementation Script Details

The implementation script is either a shell script (.sh) or batch file (.bat) that processes the example design through the Xilinx tool flow. The scripts are located in the following directory:

<proj>/<component_name>/implement/

If the CORE Generator is run with the Full System Hardware license or the Full license, the implementation script is present and performs the following steps:

- Synthesizes the example design using the selected synthesis tool (XST or Synplify)
- Runs ngdbuild to consolidate the core netlists, wrapper netlist, and constraints file into the common database
- Runs map to perform technology specific mapping of the design
- Runs par to perform place and route of the design
- Runs trce to perform static timing analysis of the routed design
- Runs bitgen to generate a bitstream for download to the target FPGA
- Runs netgen to generate a post-par simulation model for use in timing simulation

Simulation Script Details

A simulation script is provided for ModelSim, which compiles the simulation model and demonstration test bench, and runs the simulation. The simulation files are located in the following directory:

```
<proj>/<component_name>/simulation/{functional | timing}/
```

For functional simulation, the simulation script performs the following tasks:

- Compiles the simulation models provided with the core
- Compiles the wrapper file, which instantiates the cores
- Compiles the demonstration test bench
- Starts a simulation of the demonstration test bench
- Opens the waveform viewer and adds key signals
- Runs the simulation

For timing simulation, the simulation script performs the following tasks:

- Compiles the post-par example design, which includes the cores
- Compiles the demonstration test bench
- Starts a simulation of the demonstration test bench
- Opens the waveform viewer and adds key signals
- Runs the simulation

Demonstration Test Bench

The SPI-3 PHY Layer example design (Figure 4-1) consists of two components:

- **Example Design Wrapper**

  Instantiates both the TX and RX netlists (UniSim and SimPrim simulation models). A direct loopback from the TX to the RX is used; the TX LocalLink interface is connected directly to the RX LocalLink interface.

- **Demonstration Test Bench**

  Instantiates the test bench files that exercise the cores. The demonstration test bench is comprised of a number of modules that work together to generate and drive data into

the SPI-3 Packet Interface of the TX core, and read data from the SPI-3 Packet Interface of the RX core. A brief description of each module follows.

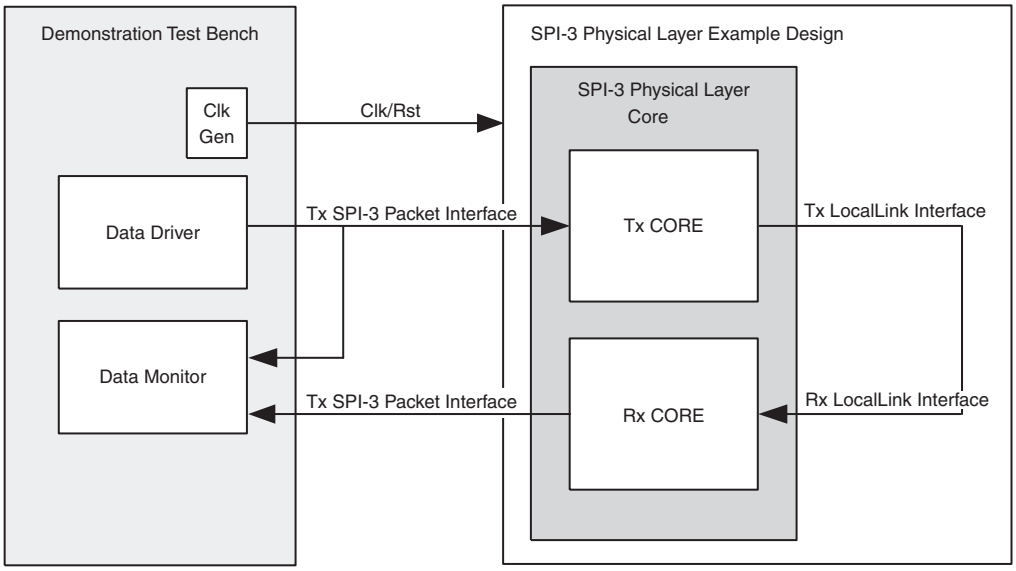

Figure 4-1: **Demonstration Test Bench**

Clock Generator Module

The clock generator module (spi3_phy_clk_gen.v[hd]) creates all clocks required for the demonstration test bench. The frequency of each core clock is defined in the testcase package, as defined below.

Procedures Module

The procedure module (spi3_phy_procedures.v[hd]) contains all functions and procedures used to create the bus functional model of the TX core's SPI-3 Packet Interface.

Testcase Package

The testcase package file (spi3_phy_testcase_pkg.v[hd]) contains all global parameters (constants).

Data Driver Module

The data driver module (spi3_phy_driver.v[hd]) is the customer interface for passing data to and from the SPI-3 PHY Layer TX core by transmitting data to the SPI-3 Packet Interface of the TX core. Data is transferred from the data driver to the TX core using a variety of procedure calls. This allows for customization of the traffic patterns being sent, and to observe the responses from the core.

The procedure calls support transmission of valid and errored packets, which enables the the observation of normal operation and the core response under various error conditions.

Data Monitor Module

The data monitor module (spi3_phy_monitor.v[hd]) records all transactions sent by the driver on the TX SPI-3 Packet Interface, and compares them to the transactions received on the RX SPI-3 Packet Interface. If a mismatch is observed, an error is flagged.

Customizing the Demonstration Test Bench

The demonstration test bench can be customized to send different data patterns than the default case. To achieve the desired test bench behavior, the following files can be customized:

```
<proj>/<component_name>/simulation/spi3_phy_testcase_pkg.v[hd]
<proj>/<component_name>/simulation/spi3_phy_driver.v[hd]
```

Customizing the Testcase Package

Several user-defined parameters (constants) appear at the top of the testcase package file, which can be changed to modify the behavior of the demonstration test bench. (Table 4-10.) Note that *only* the parameters in the testcase package listed below should be modified; the remaining testcase package parameters are internal to the test environment and should not be modified.

Table 4-10: **Customizeable Test Bench Parameters**

Parameter (Constant)	Valid Value	Description
DATA_TYPE	0	Incrementing data pattern
	1	Random data pattern
RANDOM_SEED	Positive integer	Seed used for random number generation
VERBOSE_MODE	0	Verbose mode off
	1	Verbose mode on
SPICLK_PERIOD	N/A	Sets the TFCLK and RFCLK clock periods (VHDL in ns, Verilog in ps)
LLCLK_PERIOD	N/A	Sets the TX_CLK and RX_CLK clock periods (VHDL in ns, Verilog in ps)

Note that the default example design connects both the TX and RX LocalLink clocks to `TX_CLK` because a loopback configuration is used.

Customizing the Data Driver Module

The data driver is the main interface used to control the behavior of the demonstration test bench. Each procedure call in the main process of the driver corresponds to a single SPI-3 transaction to the TX core. The data driver module can be customized by performing different procedure calls within the main process, as described in the following sections.

Packet Procedures

- `write_pkt` (addr, len, sop, eop, stut, err, and perr as shown in Table 4-11)

Transmits a SPI-3 packet or packet segment of a specified length to the address indicated. If a complete packet is to be sent, then both **sop** and **eop** should be set, which corresponds to sending TSOP and TEOP at the beginning and end of the packet data.

It is possible to leave **sop** and/or **eop** unset if a packet fragment is to be sent. This would be done if portions of several packets are being sent in an interwoven manner. If *stut* is set, then the packet will be transmitted with TENB randomly toggling on and off. If **err** is set, TERR is asserted with the packet. If **perr** is set, parity errors are generated.

Table 4-11: **write_pkt**

Input	Range	Description
addr	0-255	Address used in the packet transaction; the value will be sent on TDAT when TSX asserts.
len	1-256	Number of bytes sent to the TX core for a SPI-3 packet or packet segment.
sop	0,1	Indicates that a start-of-packet indicator should be sent. If set to 1, TSOP is asserted with the first byte of data. This should be set if a complete SPI-3 packet is to be sent with a single write_pkt procedure call.
eop	0,1	Indicates that an end-of-packet indicator should be sent. If set to 1, TEOP is asserted with the last byte of data. This should be set if a complete SPI-3 packet is to be sent with a single write_pkt procedure call.
stut	0,1	When set to 1, causes TENB to be stuttered, or randomly turned on and off throughout the packet transmission. Otherwise, TENB will assert constantly throughout the transfer.
err	0,1	When set to 1, causes the TERR signal to assert with the last data.
perr	0,1	When set to 1, causes parity errors to be generated on TPRTY for address and data. Otherwise, parity will be correct.

- check_stpa (addr and value as shown in Table 4-12)

  Transmits a 1-byte SPI-3 packet to the address indicated, and checks that the resulting STPA signal assertion corresponds to what is expected in the value field. This procedure is only valid if the TX core is configured for byte-level (direct) transfer control.

Table 4-12: **check_stpa**

Input	Range	Description
addr	0-255	Address used in the packet transaction; the value will be sent on TDAT when TSX asserts.
value	0,1	Expected signal level for STPA after packet is transmitted.

Verilog Procedure Calls

The following examples illustrate the use of the data driver procedure calls in Verilog:

```
procedures.write_pkt(0,128,1,1,0,0,0);
procedures.check_stpa(1,0);
```

VHDL Procedure Calls

All VHDL procedures have one more required port in addition to those described above. This port is a VHDL record which enables the procedures to read/write to the signals used in the data driver (note that Verilog does not require this). The following examples illustrate the use of the data driver procedures in VHDL.

```
write_packet(addr=>12,len=>256,sop=>1,eop=>1,stut=>0,err=>0,per
r=>0,pro_bus=>pro_bus);
check_stpa(addr=>4,value=>'1',pro_bus=>pro_bus);
```