LogiCORE™ IP MOST® Network Interface Controller v1.4

Getting Started Guide

UG337 September 19, 2008

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
05/17/07	1.1	Initial Xilinx release.
08/08/07	1.2	Support for Spartan-3A DSP added.
04/25/08	1.3	No functional change.
09/19/08	1.4	Automatic lock detection enhancement.

Table of Contents

About This Guide

The *MOST® Network Interface Controller Getting Started Guide* provides information about generating a LogiCORE™ IP Media Oriented Systems Transport Network Interface Controller (MOST NIC) core, customizing and simulating the core using the provided example design, and running the design files through implementation using the Xilinx tools.

Contents

This guide contains the following chapters:

- Preface, "About this Guide" introduces the organization and purpose of this guide, a list of additional resources, and the conventions used in this document.
- Chapter 1, "Introduction" describes the core and related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- Chapter 2, "Licensing the Core" provides information about licensing the core.
- Chapter 3, "Quickstart Example Design," provides the steps required to quickly implement and simulate the MOST NIC core and the example design using the default parameters.
- Chapter 4, "Detailed Example Design" describes the example design in detail and describes the core directory structure, including all directories and files.

Additional Resources

For additional information and resources, see www.xilinx.com/support. To go directly to a specific area of the support site, click a link in the table below.

Resource	Description/URL
Tutorials	Tutorials covering Xilinx design flows, from design entry to verification and debugging www.xilinx.com/support/techsup/tutorials/index.htm
Answer Browser	Database of Xilinx solution records, latest news, design tips, and patch information for the Xilinx design environment. www.xilinx.com/xlnx/xil_ans_browser.jsp
Application Notes	Descriptions of device-specific design techniques and approaches www.xilinx.com/xlnx/xweb/xil_publications_index.jsp?category=Application+Notes

Resource	Description/URL
Data Sheets	Device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging www.xilinx.com/xlnx/xweb/xil_publications_index.jsp
Problem Solvers	Interactive tools that allow you to troubleshoot your design issues www.xilinx.com/support/troubleshoot/psolvers.htm

Conventions

This document uses the following conventions.

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`
Italics	References to other manuals	See the *User Guide* 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.
<text in brackets>	User-defined variable for directory name.	<component_name>
Shading	Unsupported or reserved items	This feature is not supported
Square brackets [ ]	Optional entry or parameter, with the exception of bus specifications. For bus specifications, brackets are required, for example **`bus[7:0]`**.	**`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}`**

Convention	Meaning or Use	Example

Vertical ellipsis
.
.
.
	Omitted repetitive material	`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` returns `45524943`h
	An '_n' means the signal is active low	`usr_teof_n` is active low

Online Document

The following conventions are used in this document for cross-references and links to URLs.

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

XILINX®

Chapter 1

Introduction

This chapter introduces the MOST NIC core and provides related information, including system requirements, recommended design experience, additional resources, technical support, and submitting feedback to Xilinx. The MOST NIC core is a fully verified solution that supports Verilog and VHDL. In addition, the example design in this guide is provided in both Verilog and VHDL.

System Requirements

Windows

- Windows® 2000 Professional with Service Pack 2-4
- Windows XP Professional with Service Pack 1-2

Solaris/Linux

- Sun Solaris® 9/10
- Red Hat® Enterprise Linux 3.0 (32-bit and 64-bit)

Software

- ISE® 10.1 Update 3

Check the release notes for the required Service Pack; ISE Service Packs can be downloaded from www.xilinx.com/xlnx/xil_sw_updates_home.jsp?update=sp.

About the Core

The MOST NIC 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 MOST NIC product page. For information about choosing a license option and requirements to obtaining a license for the core, see Chapter 2, "Licensing the Core."

Recommended Design Experience

Although the MOST NIC 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. Contact your local Xilinx representative for a closer review and estimation for your specific requirements.

Additional Core Resources

For detailed information and updates about the MOST NIC core, including the MOST NIC data sheet, see the MOST NIC product page.

After generating the core using the CORE Generator, the following documents are available in the project document directory.

- *MOST NIC Data Sheet*
- *MOST NIC User Guide*
- *MOST NIC Getting Started Guide*

Technical Support

For technical support, go to www.xilinx.com/support. Questions are routed to a team with expertise using the MOST NIC core.

Xilinx provides technical support for use of this product as described in the *MOST Network Interface Controller User Guide* and the *MOST Network Interface Controller Getting Started Guide*. Xilinx does not guarantee timing, functionality, or support of this product for designs that do not follow these guidelines.

Feedback

Xilinx welcomes comments and suggestions about the MOST NIC core and the accompanying documentation.

Core

For comments or suggestions about the MOST NIC core, 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 the MOST NIC core, 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

Chapter 2

Licensing the Core

This chapter provides instructions for choosing a license option and obtaining a license key for the MOST NIC core, which you must do before using it in your designs. The MOST NIC 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.

Before you Begin

This chapter assumes you have installed the core using either the CORE Generator IP Software Update installer or by performing a manual installation after downloading the core from the web. For information about installing the core, see the MOST NIC product page.

Additional Requirements for MOST NIC Licensing

The MOST NIC core provides two license options to evaluate the core: the Simulation Only license and the Full System Hardware Evaluation license. Before acquiring a key for the Full System Hardware Evaluation license, the following requirements must be satisfied:

- You must be a member of the MOST Cooperation www.mostcooperation.com/membership/index.php
- You must accept the Xilinx MOST NIC license agreement

For information about registering and acquiring a license key for the Full System Hardware Evaluation, see "Obtaining Your License," page 12.

License Options

Simulation Only License

The Simulation Only license, provided with the CORE Generator, lets you assess the core functionality with either the provided example design or alongside your own design, and demonstrates the various interfaces on the core in simulation. (Functional simulation is supported by a structural model provided by the CORE Generator.) To use the Simulation Only license, membership in the MOST Cooperation is not required.

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 back-annotated gate-level simulation of the core using the provided demonstration test bench.

In addition, this license 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 is provided after purchasing the core and provides full access to all core functionality in both simulation and in hardware, including

- Functional simulation support
- Back annotated gate-level 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

Obtaining a Full System Hardware Evaluation License

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

- Navigate to the MOST NIC product page.
- Click Evaluate; then click Full System Hardware Evaluation.
- Follow the onscreen instructions to both download the CORE Generator files (delivered as an IP Update) and satisfy any additional requirements associated with the license.

Obtaining a Full License

To obtain a Full license, you must purchase the core. After purchase, you will receive a letter containing a serial number, which is used to register for access to the *lounge*, a secured area of the MOST NIC product page.

- From the product page, click Register to register and request access to the lounge.
- Xilinx will review your access request and typically grants access to the lounge in 48 hours. (Contact Xilinx Customer Service if you need faster turnaround.)
- After you receive confirmation of lounge access, click Access Lounge on the product page and log in.
- Follow the instructions in the lounge to fill out the license request form; then click Submit to automatically generate the license. An e-mail containing the license and installation instructions will be sent to you immediately.

Installing Your License File

The Simulation Only Evaluation license is provided with the CORE Generator and does not require a license file. For a Full System Hardware Evaluation license and Full license, you will receive an email containing instructions for installing your license file, as well as information about advanced licensing options and technical support.

Chapter 3

Quickstart Example Design

The instructions in this chapter let you quickly generate a MOST NIC core, run the design through implementation using the Xilinx tools, and simulate the example design utilizing the provided demonstration test bench. For detailed information about the example design, see Chapter 4, "Detailed Example Design."

Overview

The MOST NIC example files consist of three parts, as illustrated in Figure 3-1:

- A top-level file instantiating the MOST NIC core with all required clock generators and drivers
- The MOST NIC OPB example design, a simple state machine that converts stored OPB transactions to signals to drive the core.
- The MOST NIC demonstration test bench that loads the OPB transactions and provides an environment to simulate the core.

In addition to the example design components, some helper tools are delivered:

- Scripts to implement the design, including sample constraints
- Low-level software drivers for use in the user application

The MOST NIC example design has been tested with XST synthesis tools, ISE 10.1 SP3 implementation tools, and ModelSim® v6.3c and Cadence® IUS 6.1 simulation tools.

Simulation Design Overview

As illustrated in Figure 3-1, two main circuit components are instantiated inside the test bench (module names assume the component name is `most_nic`). Both the MOST NIC with associated clocking blocks, and the MOST NIC OPB example design are fully synthesizeable. The top-level test bench loads the example design with OPB transactions and triggers events.

The MOST NIC has an internal loopback mode, and the provided example design contains the steps required to set this mode. However, the MOST ring output of the core is still available to the user and can be viewed to examine traffic that would be representative on the ring. For more information about the loopback debug mode, see "Programming the Core," in the *MOST NIC User Guide*.

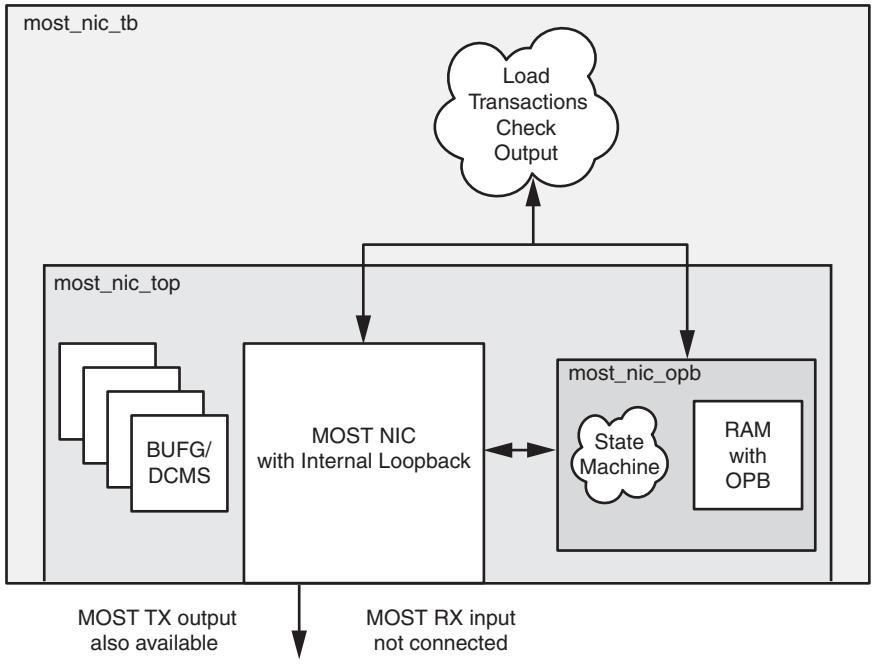

Figure 3-1: **Example Design**

Generating the Core

To generate a MOST NIC core with default values using the CORE Generator GUI, do the following:

1. Start the CORE Generator.

   For help starting and using the CORE Generator, see the Xilinx CORE Generator Guide, available from ISE documentation.
2. Choose File > New Project.
3. Type a directory name. In this example, the name `coregen` is used.
4. Set the project options:
   - From Target Architecture, select an FPGA family that supports the MOST NIC core, for example, Automotive Spartan-3E.

   Note: If an unsupported silicon family is selected, the MOST NIC core does not appear in the taxonomy tree. For a list of supported architectures, see the *MOST NIC User Guide*.

   - For Design Entry, select VHDL or Verilog; for Vendor, select Other.
5. After creating the project, locate the MOST NIC core in the taxonomy tree under Automotive & Industrial/Automotive.
6. Double-click the core. The MOST NIC customization screen appears.

If the license file is not properly configured, the CORE Generator reports an error. For information about core licensing, see Chapter 2, "Licensing the Core."

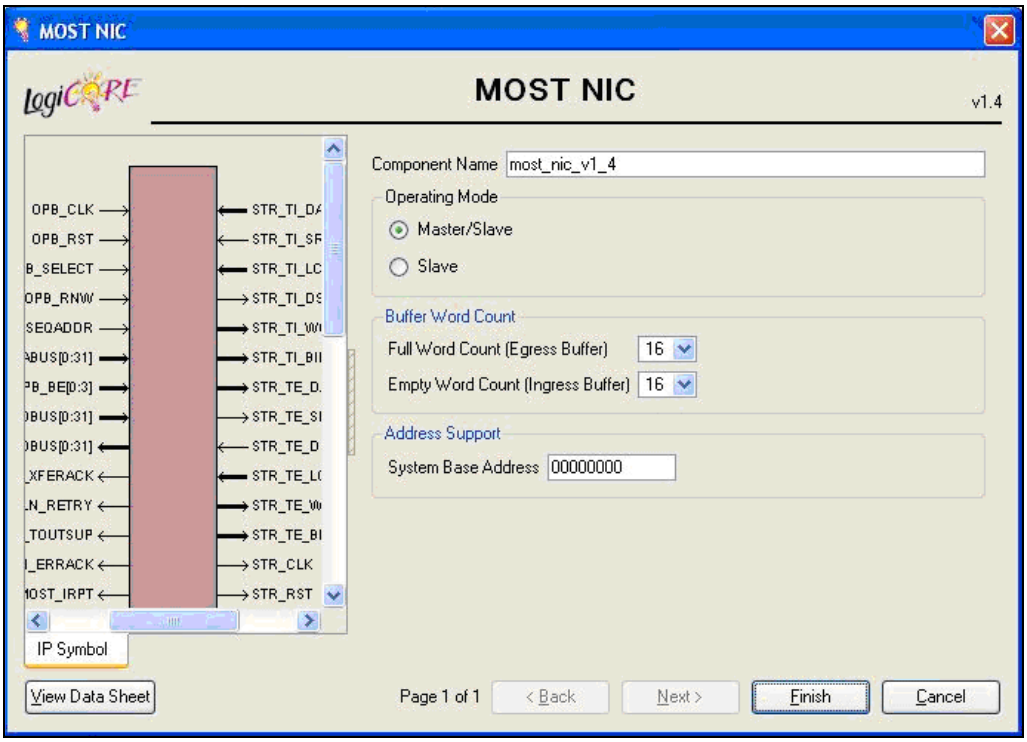

Figure 3-2: **CORE Generator Customization Screen**

7. In the Component Name field, enter a name for the core instance. In this example, the name most_nic_v1_4 is used.
8. Accept the remaining default parameters; then click Generate.

   The core and its supporting files, including the example design, are generated in the coregen project directory. For detailed information about the example design files and directories see "Directory and File Contents" in Chapter 4.

Implementing the Example Design

Note: Available only with a Full System Hardware Evaluation or Full license.

After the core is generated, the netlists and example design can be processed by the Xilinx implementation tools. The generated output files include several scripts to assist the user in running the Xilinx software.

From the CORE Generator project directory, do the following to implement the MOST NIC example design:

For Windows

```
> cd <component_name>/implement
> implement.bat
```

For Linux

```
% cd <component_name>/implement
% ./implement.sh
```

These commands execute a script that synthesizes, builds, maps, place-and-routes, and creates a gate-level simulation model for the example design. The resulting files are placed in the `implement/results` directory.

Simulating the Example Design

The MOST NIC core provides a quick way to simulate and observe the behavior of the core utilizing the provided example design. Before simulating the core, the functional (gate-level) simulation models are generated by the implementation script.

To run a ModelSim or IUS simulation of the MOST NIC core, use either the VHDL or Verilog instructions that follow.

ModelSim Simulations

To run a simulation of the example design using Mentor ModelSim:

1. Change the working directory to
   <component_name>/simulation/<simulation_type>

   where type is either `functional`, for a gate-based simulation, or `timing`, for a gate-based simulation with back-annotated SDF added.

2. Map the UNISIM and SIMPRIM libraries (if not already mapped globally):

   For VHDL

   ```
   % vmap unisim <path to complied libraries>/unisim
   % vmap simprim <path to complied libraries>/simprim
   ```

   For Verilog

   ```
   % vmap unisims_ver <path to compiled libraries>/unisims_ver
   % vmap simprims_ver <path to complied libraries>/simprims_ver
   ```

3. Launch the simulation script:

   ```
   > vsim -do simulate_mti.do
   % vsim -do simulate_mti.do
   ```

The ModelSim script compiles the simulation model and the demonstration test bench, adds relevant signals to the wave window, and runs the simulation, after which the simulation transcript and waveform can be inspected to observe the operation of the core.

IUS Simulations

To run a simulation of the example design using Cadence IUS:

1. Change the working directory to **<component_name>/simulation/<simulation_type>** where type is either `functional`, for a gate-based simulation, or `timing`, for a gate-based simulation with back-annotated SDF added.
2. Map the UNISIM and SIMPRIM libraries (if not already mapped globally), by adding the following lines to the appropriate `cds.lib`:

   For VHDL

   ```
   DEFINE work work
   DEFINE unisim <path to complied libraries>/unisim
   DEFINE simprim <path to complied libraries>/simprim
   ```

   For Verilog

   ```
   DEFINE work work
   DEFINE unisims_ver <path to compiled libraries>/unisims_ver
   DEFINE simprims_ver <path to complied libraries>/simprims_ver
   ```

3. Launch the simulation script:

   ```
   > simulate_ncsim.bat
   % ./simulate_ncsim.sh
   ```

The IUS script compiles the simulation model and the demonstration test bench, adds relevant signals to the wave window, and runs the simulation, after which the simulation transcript and waveform can be inspected to observe the operation of the core.

XILINX®

Chapter 4

Detailed Example Design

This chapter provides detailed information about the MOST NIC 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; user-defined name.
  - <project directory>/<component name>
    Core readme file.
    - <component name>/doc
      Product documentation.
    - <component name>/example_design
      Verilog and 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 | timing}
        Functional simulation files.
    - <component_name>/sw/device_driver_v1_1/build
      Files for compiling the low-level drivers data provided with the core.
      - sw/device_driver_v1_1/build/vxworks5_4
        Build files specific to VxWorks integration of low-level drivers data.
      - sw/device_driver_v1_1/data
        Data files for automatic generation of parameter-specific files when integrated into Xilinx Platform Studio.
      - sw/device_driver_v1_1/doc
        Placeholders for generation of documentation within Xilinx Platform Studio.
      - sw/device_driver_v1_1/examples
        Application examples using the low-level driver files and the MOST NIC.
      - sw/device_driver_v1_1/src
        Low-level driver source C files.

Directory and File Contents

The MOST NIC directories and their associated files are defined in the sections that follow.

<project directory>

The project directory contains all the CORE Generator project files.

Table 4-1: **Project Directory**

Name	Description
<project_dir>	
<component_name>.ngc	Top-level netlist.
<component_name>.v[hd]	Verilog or VHDL simulation model.
<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.
<component_name>_xmdf.tcl	Project Navigator integration file for the core.
<component_name>.{veo|vho}	VHDL or Verilog instantiation template.

Back to Top

<project directory>/<component name>

The <component name> directory contains the readme 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>	
most_nic_readme.txt	MOST NIC readme 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	
most_nic_ds543.pdf	*MOST Network Interface Controller Data Sheet*
most_nic_gsg347.pdf	*MOST Network Interface Controller Getting Started Guide*

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

Name	Description
most_nic_ug336.pdf	*MOST Network Interface Controller User 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.v[hd]	Verilog or VHDL instantiation of the core, example design, and clocking circuitry.
<component_name>_opb.v[hd] <component_name>_opb_pkg.vhd	Verilog or VHDL example design containing the state machine and transaction memory.
<component_name>_top.ucf	MOST NIC clocking and path UCF.

Back to Top

<component name>/implement

The implement directory contains the core implementation script files.

Table 4-5: **Implement Directory**

Name	Description
<project_dir>/<component_name>/implement	
implement.bat implement.sh	Implementation scripts for the top-level example design for Windows and Linux, respectively.
xst.prj xst.scr	Configuration files for XST.

Back to Top

implement/results

The results directory is created by the implement script, after which the implement script results are placed in the results directory.

Table 4-6: **Results Directory**

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

Back to Top

<component name>/simulation

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

Table 4-7: **Simulation Directory**

Name	Description
<project_dir>/<component_name>/simulation	
<component_name>_tb.v[hd] <component_name>_tb_pkg.vhd	Verilog or VHDL test bench, which instantiates the top level including the example design and the core, and loads the example design with transactions.

Back to Top

simulation/{functional | timing}

The functional and timing directories contain simulation scripts provided with the core.

Table 4-8: **Functional Directory**

Name	Description
<project_dir>/<component_name>/simulation/{functional | timing}	
simulate_mti.do	The do file used by ModelSim to compile the test bench and simulate the core.
simulate_ncsim.bat simulate_ncsim.sh	Windows or Linux script used with IUS to compile the test bench and simulate the core.
wave.do	The .do file used by ModelSim to setup the display with useful signals from the simulation.
wave.sv	The simvision file used by IUS to setup the display with useful signals from the simulation.

Back to Top

<component_name>/sw/device_driver_v1_1/build

The driver build directory contains the files necessary to compile the low-level drivers data provided with the core.

Table 4-9: **Driver Build Directory**

Name	Description
<project_dir>/<component_name>/sw/device_driver_v1_1/build	
makefile	Makefile to compile the drivers.

Back to Top

sw/device_driver_v1_1/build/vxworks5_4

The driver VxWorks directory contains build files specific for VxWorks integration of the low-level drivers data provided with the core.

Table 4-10: **Driver Build Directory**

Name	Description
<project_dir>/<component_name>/sw/device_driver_v1_1/build	
most_nic_v1_4.file	A placeholder file for future use.

Back to Top

sw/device_driver_v1_1/data

The driver data directory contains the data files for automatic generation of parameter specific files when integrated into Xilinx Platform Studio.

Table 4-11: **Driver Data Directory**

Name	Description
<project_dir>/<component_name>/sw/device_driver_v1_1/data	
most_v2_1_0.mdd	Current MDD file used, including the version of the tools interface.
most_v2_1_0.tcl	Used to provide design rule checks within Xilinx Platform Studio.

Back to Top

sw/device_driver_v1_1/doc

The driver doc directory contains placeholders for generation of documentation within Xilinx Platform Studio

Table 4-12: **Driver Doc Directory**

Name	Description
<project_dir>/<component_name>/sw/device_driver_v1_1/doc	
most_nic_v1_4.file	A placeholder file for future use.

Back to Top

sw/device_driver_v1_1/examples

The driver example directory contains application examples using the low-level driver files and the MOST NIC.

Table 4-13: **Driver Example Directory**

Name	Description
<project_dir>/<component_name>/sw/device_driver_v1_1/examples	
xmost_async_channel_intr_example.c	Contains an example design application to test the MOST NIC in Loopback Mode over the asynchronous channel.
xmost_control_channel_intr_example.c	Contains an example design application to test the MOST NIC in loopback Mode over the control channel.
xmost_selftest_example.c	Contains a very basic example design using the MOST NIC driver.
xmost_sync_channel_intr_example.c	Contains an example design application to test the MOST NIC in loopback mode over the synchronous channel.

Back to Top

sw/device_driver_v1_1/src

The driver source directory contains the low-level driver source C files.

Table 4-14: **Driver Source Directory**

Name	Description
<project_dir>/<component_name>/sw/device_driver_v1_1/src	
README.txt	Contains a description and explanation of all files in this directory.
xmost.c	Contains the bare minimum driver functions to initialize and run the Most device.

Table 4-14: **Driver Source Directory *(Continued)***

Name	Description
xmost.h	Main header file for the Xmost driver. The file header gives the complete details of the device driver.
xmost_config.c	Contains the functions needed to configure the Most device for various configurations/features/options.
xmost_g.c	Contains a structure which has all the configuration values selected by the user during the hardware design for the Most device.
xmost_intr.c	Contains the interrupt related functions.
xmost_l.c	Contains all the constant definitions and bare minimum functions for register read/write access.
xmost_selftest.c	Contains the basic test function for the sanity check of the most device.
xmost_sinit.c	Contains the function for static initialization of the Most device/driver.

Back to Top

Implementation Scripts

Note: Present only with a Full or a Hardware Evaluation license.

The implementation script is either a shell script or batch file that processes the example design through the Xilinx tool flow, and is located in the following directories:

Linux

```
<project_dir>/<component_name>/implement/implement.sh
```

Windows

```
<project_dir>/<component_name>/implement/implement.bat
```

The implement script performs the following steps:

- Synthesizes the HDL example design files using XST
- Runs Ngdbuild to consolidate the core netlist and the top level of the example design netlist into the NGD file containing the entire design
- Maps the design to the target technology
- Place-and-routes the design on the target device
- Performs static timing analysis on the routed design using Timing Analyzer (TRCE)
- Generates a bitstream
- Enables Netgen to run on the routed design to generate a VHDL or Verilog netlist (as appropriate for the Design Entry project setting) and timing information in the form of SDF files

The Xilinx tool flow generates several output and report files. These are saved in the following directory, which is created by the implement script:

```
<project_dir>/<component_name>/implement/results
```

Simulation Scripts

Functional Simulation

The test scripts are a ModelSim macro or a shell script for IUS that automate the simulation of the test bench. They are available from the following location:

```
<project_dir>/<component_name>/simulation/functional/
```

The test script performs the following tasks:

- Compiles the structural UNISIM simulation model
- Compiles HDL Example Design source code
- Compiles the demonstration test bench
- Starts a simulation of the test bench
- Opens a Wave window and adds signals of interest (wave.do/wave.sv)
- Runs the simulation to completion

Timing Simulation

Note: Present only with a Full or a Hardware Evaluation license.

The test scripts are a ModelSim macro or a shell script for IUS macro that automates the simulation of the test bench. They are located in:

```
<project_dir>/<component_name>/simulation/timing/
```

The test script performs the following tasks:

- Compiles the SimPrim based gate level netlist simulation model
- Compiles the demonstration test bench, including back annotation of SDF generated in the implement step
- Starts a simulation of the test bench
- Opens a Wave window and adds signals of interest (wave.do/wave.sv)
- Runs the simulation to completion

Example Design

OPB Example Design

The following files describe the OPB example design for the MOST NIC core:

VHDL

```
<project_dir>/<component_name>/example_design/<component_name>_opb.vhd
<project_dir>/<component_name>/example_design/<component_name>_opb_pkg
.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/<component_name>_opb.v
```

Figure 4-1 shows an example design for a component named `most_nic` containing a state machine and a memory that contains OPB transactions (which communicate with the MOST NIC via the OPB interface) without requiring all the infrastructure generally required for OPB interfaces. It is designed to be synthesized and can be implemented in a target device to provide post place-and-route gate-level simulation.

The outputs from the instruction space memory can be viewed in the simulation, and if desired, the memory can be modified to have pre-loaded instructions and treat the memory like a ROM, allowing the design to also run in hardware.

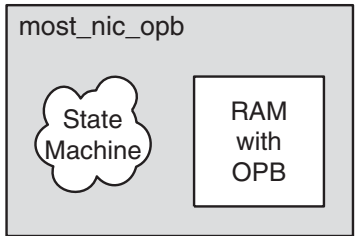

Figure 4-1: **OPB Example Design for MOST NIC**

MOST Clocking Top Level

The following files describe the top-level clocking for the MOST NIC core:

VHDL

```
<project_dir>/<component_name>/example_design/<component_name>_top.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/<component_name>_top.v
```

Constraints

```
<project_dir>/<component_name>/example_design/<component_name>_top.ucf
```

For a slave-only core, the wrapper automatically connects both clock inputs to the common clock. For a master/slave core, the wrapper file core shows all the required clocking, and is fully constrained with the delivered UCF, as illustrated in Figure 4-2, for a component named `most_nic`. This clocking mechanism has been validated in hardware, and meets all required MOST jitter specifications; do not modify it.

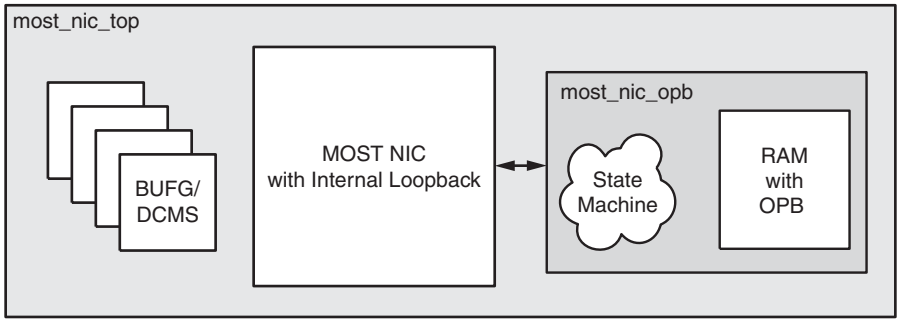

Figure 4-2: **Example Clocking Wrapper for MOST NIC**

Demonstration Test Bench

The following files describe the demonstration test bench:

VHDL

```
<project_dir>/<component_name>/simulation/<component_name>_tb.vhd
<project_dir>/<component_name>/simulation/<component_name>_tb_pkg.vhd
```

Verilog

```
<project_dir>/<component_name>/simulation/<component_name>_tb.v
```

Figure 4-3 illustrates the demonstration test bench (assuming a component name of `most_nic`) as a simple VHDL entity or Verilog module that instantiates and exercises the top level of the example design by programming and sending and receiving synchronous data while configured in Slave/Loopback mode (the device will be a pseudo-master).

The demonstration test bench contains the following helper components:

- Local constants mapping all the register offsets to the mnemonics described in the *MOST NIC Data Sheet* and *MOST NIC User Guide*
- Tasks that convert the register offsets to locations using `C_BASEADDR`
- Clock generation

The demonstration test bench performs the following tasks:

- Applies a hard reset to the core
- Reads the version register
- Applies a soft reset to the core
- Programs the PRCR(PLL Reset Count Register) register
- Configures the core for Bypass mode
- Programs the address registers
- Programs the retry counters
- Reads the status of the core
- Clears and enables MOST interrupts
- Places the core in loopback mode, and waits for *ring* synchronization
- Reads the Synchronous Boundary Descriptor and Delay and Position registers
- Primes the TX buffer with a series of synchronous data
- Programs the TX and RX routing tables
- Enables logical channel 1 for both TX and RX
- Compares the received synchronous data

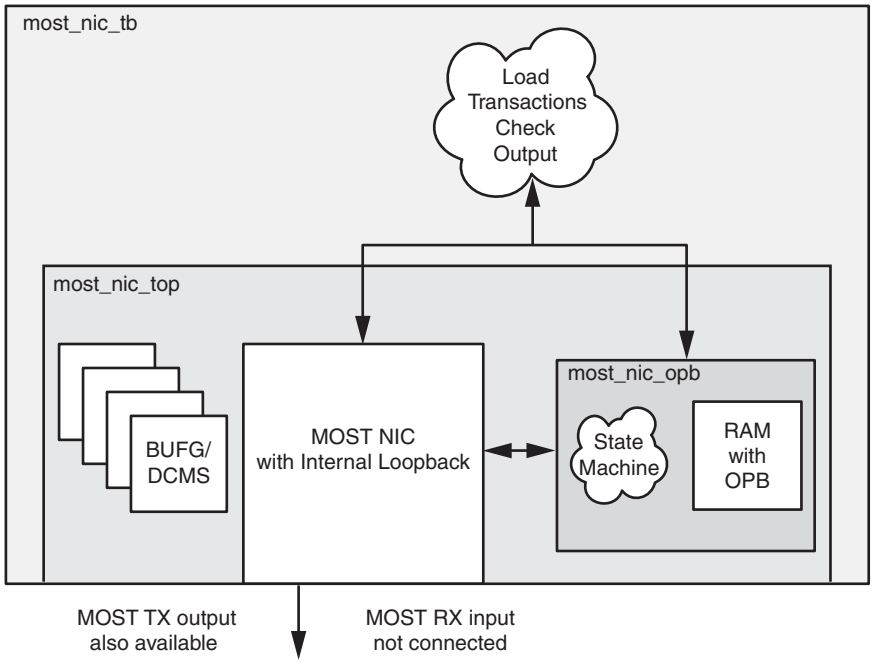

Figure 4-3: **Demonstration Test Bench for MOST NIC Core and Example Design**

Customizing the Test Bench

The core should be verified in a *complete* system with the desired application running. However, when starting with the design, follow the guidelines below to modify the demonstration test bench provided with the core:

- Note that not all instruction combinations are used. Using an invalid instruction results in that instruction being skipped and the assertion of `OPB_ERROR`.
- The user may issue multiple read requests. In this case, each read response returns in the order received. The data arrives on the `OPB_DATA` bus and `OPB_READ_RESP` asserts. The user may also issue a `READ COMPARE` instruction, at which time the example design automatically compares the received value with an expected value. On failure, the example design asserts `OPB_CMP_INVALID`.
- From time to time, data from the ring may arrive to the user on the `OPB_DATA` bus, the same bus used for read responses. When the ring data arrives, it is qualified with `OPB_BUFFER_RD`.
- The example design can be used regardless of the state of MOST NIC core. However, it is recommended that the user put the core in loopback mode.

Tips for Implementing the MOST NIC Core in a System

Keep it Registered

To simplify timing and increase system performance in an FPGA design, keep all inputs and outputs registered between the user application and the core. This means that all inputs and outputs from the user application should come *from*, or connect *to*, a flip-flop. While registering signals may not be possible for all paths, it simplifies timing analysis and makes it easier for the Xilinx tools to place-and-route the design. The majority of the MOST NIC input and outputs within the core are registered to make this procedure simple for the user. However, keep the following exceptions in mind:

- `OPB_SeqAddr`: This signal has some logic directly connected to it before being latched in the core to handle the immediate turnaround associated with burst transactions.
- `MOST_TX/MOST_RX`: In Bypass mode, the MOST NIC needs to tie the `MOST_RX` input directly to the `MOST_TX` output because there may be no clock active. The MOST NIC should be placed adjacent to these pins in the FPGA, where the pins themselves are close together.

Recognize Timing Critical Signals

The UCF provided with the example design identifies the critical signals and timing constraints that should be applied. For detailed information, see "Chapter 5, Constraining the Core," in the *MOST NIC User Guide*.

Note that while the example design uses the same frequency for `MOST_PLL_CLOCK` and `OPB_Clk`, this is not required in an actual system. However, there are very specific constraints applied to `MOST_PLL_CLOCK`, and these need to be adhered to, in order to meet the very tight jitter constraints in a MOST system.

The recovery PLL from Integrated Device Technology (IDT®) is designed to operate with the MOST NIC LogiCORE, and is the suggested external PLL implementation. For more information about the clock generator/recovery PLL, go to www.idt.com.

Use Supported Design Flows

The MOST NIC core is pre-synthesized and delivered as an NGC netlist. The example implementation scripts provided use XST 10.1 SP3 as the synthesis tool for the HDL example design delivered with the core. Other synthesis tools may be used for the user application logic; however, the core will be unknown to the synthesis tool and appears as a black box.

Note: After synthesis, only ISE 10.1 SP3 tools are supported.

Running the MOST NIC with Low-Level Drivers in Hardware

You can import the MOST NIC into an EDK project by following the steps to import a black box IP. Please see the Xilinx Platform Studio documentation for information about importing the MOST NIC into an EDK project. Note that for this process to work properly, VHDL should be chosen as language for the output files, because VHDL is required for all wrapper files for black box IP.

After importing the generated netlist into the project, the drivers can also be linked into the software files. After completing this process, a C program can be used to exercise the core. See "Programming the Core" in the *MOST NIC User Guide* for more information.

Running the MOST NIC with MOCEAN Network Services

A complete Network Services Programming Stack for the MOST NIC is also available from MOCEAN Laboratories AB. For more information about this stack, contact MOCEAN Laboratories at www.mocean-labs.com.