LogiCORE™ IP Generic Framing Procedure v2.1

Getting Started Guide

UG151 April 25, 2008

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/30/04	1.1	Initial Xilinx release.
11/11/04	1.2	Updated 64-bit information
4/28/05	1.3	Updates to Chapters 3 and 4; ISE™ tools updated to 7.1i SP1.
1/18/06	1.4	Updated to ISE 8.1i.
4/25/08	2.1	Updated to ISE v10.1 and added support for Virtex®-5.

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

Table of Contents

Chapter 4: Detailed Example Design

Schedule of Figures

Chapter 2: Licensing the Core

Chapter 3: Quick Start Example Design

Chapter 4: Detailed Example Design

Schedule of Tables

Chapter 4: Detailed Example Design

About This Guide

The *Generic Framing Procedure v2.1 Getting Started Guide* provides information about generating a Generic Framing Procedure (GFP) core, customizing and simulating the core utilizing the provided example design, and running the design files through implementation using the Xilinx tools.

Guide Contents

The following chapters are included in this guide:

- Preface, "About This Guide" introduces the organization and purpose of the Getting Started Guide, including the conventions used in the guide.
- 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, "Quick Start Example Design" provides instructions to quickly generate the core and run the example design through implementation and simulation.
- Chapter 4, "Detailed Example Design" describes the demonstration test bench in detail and provides instructions for how to customize the demonstration test bench for use in an application.

Additional Resources

To find additional documentation, see the Xilinx website at:

www.xilinx.com/literature.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

www.xilinx.com/support.

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 that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Italic font	References to other manuals	See the *Development System Reference Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
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	Omitted repetitive material	`IOB #1: Name = QOUT'` `IOB #2: Name = CLKIN'`
Horizontal ellipsis `...`	Repetitive material that has been omitted	**`allow block`** *block_name loc1 loc2 ... locn;*

Online Document

The following 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. Refer to "Title Formats" in Chapter 1 for details.
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the *Virtex-II Platform FPGA User Guide.*
Blue, underlined text	Hyperlink to a website (URL)	Go towww.xilinx.com for the latest speed files.

Chapter 1

Introduction

The LogiCORE™ IP GFP core is a fully verified protocol encapsulation/de-encapsulation engine designed to support both Verilog and VHDL design environments. In addition, the example design in this guide is provided in both Verilog and VHDL.

This chapter introduces the GFP core and provides related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.

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® v10.1

About the Core

The GFP 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 www.xilinx.com/products/ipcenter/DO-DI-GFP.htm. For information about system requirements, installation, and licensing options, see Chapter 2, "Licensing the Core."

Recommended Design Experience

Although the GFP core is a fully verified solution, the challenge associated with implementing a complete GFP design varies depending on the application requirements. 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 assistance in evaluating your specific requirements.

Additional Core Resources

For detailed information and updates related to the GFP core, see the GFP product page.

Technical Support

For technical support, visit www.xilinx.com/support. Questions are routed to a team of engineers with expertise using the GFP core.

Xilinx will provide technical support for use of this product as described in the *GFP User Guide* and the *GFP Getting Started User 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 GFP core and the documentation supplied with the core.

GFP Core

For comments or suggestions about the GFP 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 and obtaining a license for the GFP core, which you must do before using the core in your designs. The GFP 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.

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 GFP product page.

Verifying your Installation

1. Start the CORE Generator.
2. After creating a new project or opening an existing one, the IP core functional categories appear at the left side of the window.

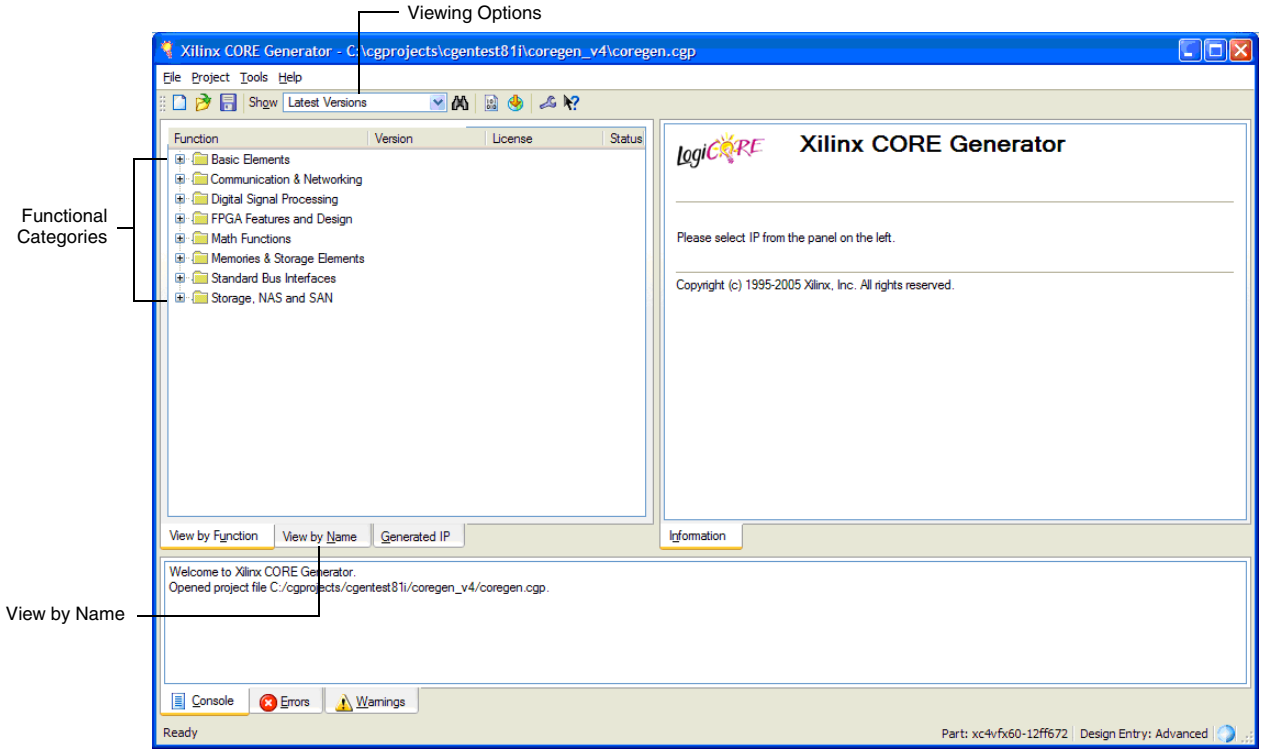

Figure 2-1: **CORE Generator Window**

3. Click to expand or collapse the view of individual functional categories, or click the View by Name tab at the bottom of the list to see an alphabetical list of all cores in all categories.
4. To view specific versions of the cores, choose an option from the Show drop-down list at the top of the window:
   - **Latest Versions**. Display the latest versions of all cores.
   - **All Versions**. Display all versions of cores, including new cores and new versions of cores.
   - **All Versions including Obsolete**. Display all cores, including those scheduled to become obsolete.
5. Determine if the installation was successful by verifying that the new core or cores appear in the CORE Generator GUI.

   For additional assistance installing the IP Update, contact www.xilinx.com/support.

License Options

The GFP core provides three licensing options. After installing the core, choose a license option.

Simulation Only Evaluation

The Simulation Only Evaluation license is the default license provided with the GFP core. This license lets you generate a simulation model and simulate it using the demonstration test bench provided. Functional simulation is supported by a structural model provided by the CORE Generator.

The Simulation Only Evaluation license 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.

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 demonstration test bench provided.

In addition, the 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.

You can obtain the Full System Evaluation license in one of the following ways, depending on the core:

- By registering on the Xilinx IP Evaluation page and filling out a form to request an automatically generated evaluation license
- By contacting your local Xilinx FAE to request a Full System Hardware Evaluation license key

Click Evaluate on the core's product page for information about how to obtain a Full System Hardware Evaluation license.

Full

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

- Gate-level 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 GFP product page: www.xilinx.com/products/ipcenter/DO-DI-GFP.htm.
- 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 GFP product page.

- From the GFP 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 receiving confirmation of lounge access, click Access Lounge on the GFP 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

After selecting either the Full System Hardware Evaluation or Full license option, you will receive an email containing instructions for installing your license. In addition, the email provides information about advanced licensing options and technical support.

Chapter 3

Quick Start Example Design

The quick start instructions let you quickly generate a GFP 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

Figure 3-1 illustrates the GFP example design.

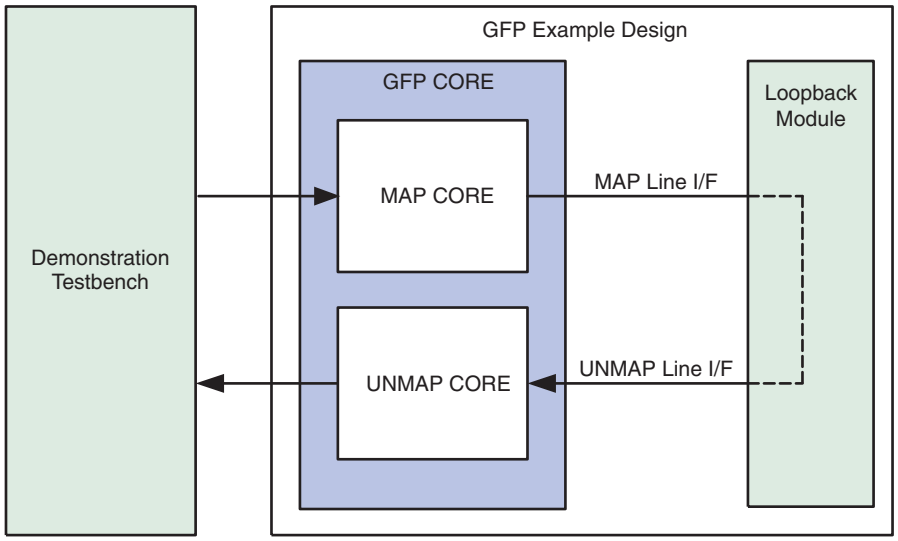

Figure 3-1: **Example Design**

The GFP example design consists of the following:

- MAP and UNMAP core netlists
- MAP and UNMAP core simulation models
- Example HDL wrapper (which instantiates the cores and example design)
- Customizable demonstration test bench to simulate the example design

The GFP example design has been tested with Xilinx ISE 10.1i and the following simulators:

- Mentor Graphics® ModelSim® v6.3c
- Cadence® IUS v6.1

Generating the Core

To generate a GFP core with default values using the Xilinx CORE Generator tool, do the following:

1. Start the CORE Generator tool.

   For help starting and using the CORE Generator tool, see the Xilinx CORE Generator Guide, available from the ISE documentation web page.
2. Choose File > New Project.
3. Type a directory name. In this example, the directory name *design* 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 *GFP Data Sheet*.

     Note: If an unsupported silicon family is selected, the GFP core will not appear in the taxonomy tree.
   - Generation Options
     - For Design Entry, select either VHDL or Verilog.
     - For Vendor, select Synplicity or Other (for XST).
5. After creating the project, locate the GFP core in the taxonomy tree under Communications & Networking > Telecommunications > Generic Framing Procedure.
6. Double-click the core to display the main GFP screen.

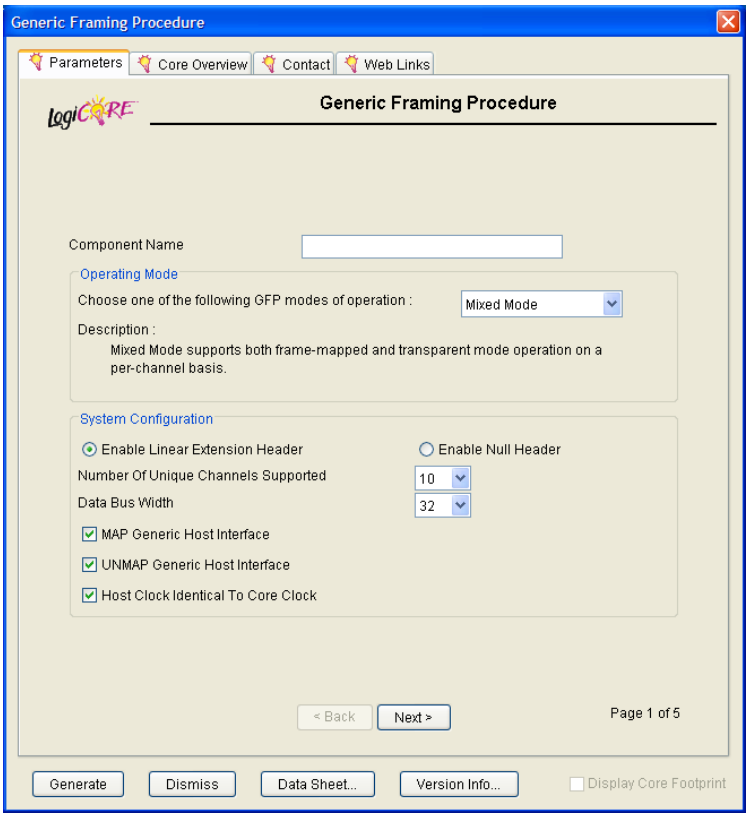

Figure 3-2: **GFP Main Screen**

7. In the Component Name field, enter a name for the core instance. In this chapter, the name *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 Structure," on page 21.

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 GFP example design, open a command prompt or terminal window and type the following commands:

For Windows

```
> cd <project_dir>\<component_name>\implement
> implement.bat
```

For Linux

```
% cd <project_dir>/<component_name>/implement
% ./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 GFP 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 will either be in VHDL or Verilog, depending on the CORE Generator Design Entry project option.

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, go to Answer Record 15338 on www.xilinx.com/support for assistance compiling Xilinx simulation models.

Simulation scripts are provided for ModelSim and Cadence IUS.

Functional Simulation

Instructions for running a functional simulation of the GFP core using either VHDL or Verilog are 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:

1. Set the current directory to:

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

2. Launch the simulation script:

```
modelsim: vsim -do simulate_mti.do
ius: ./simulate_ncsim.sh
```

The simulation script compiles the functional simulation models, the loopback 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 only supported for Full-System Hardware Evaluation and Full license types, as the core cannot be implemented using a Simulation Only Evaluation license. Instructions for running a timing simulation of the GFP core using either VHDL or Verilog are below. A timing simulation model is generated when the core is run through the Xilinx tools using the implement script. It is a requirement that the core is implemented before attempting to run timing simulation.

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

1. Set the current directory to:

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

2. Launch the simulation script:

```
modelsim: vsim -do dimulate_mti.do
nc-sim: ./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 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.

Directory Structure

Figure 4-1 displays the files and directories created by CORE Generator. <*project_dir*> is the CORE Generator project directory, and <*component_name*> is the component name entered on the GFP core customization screen.

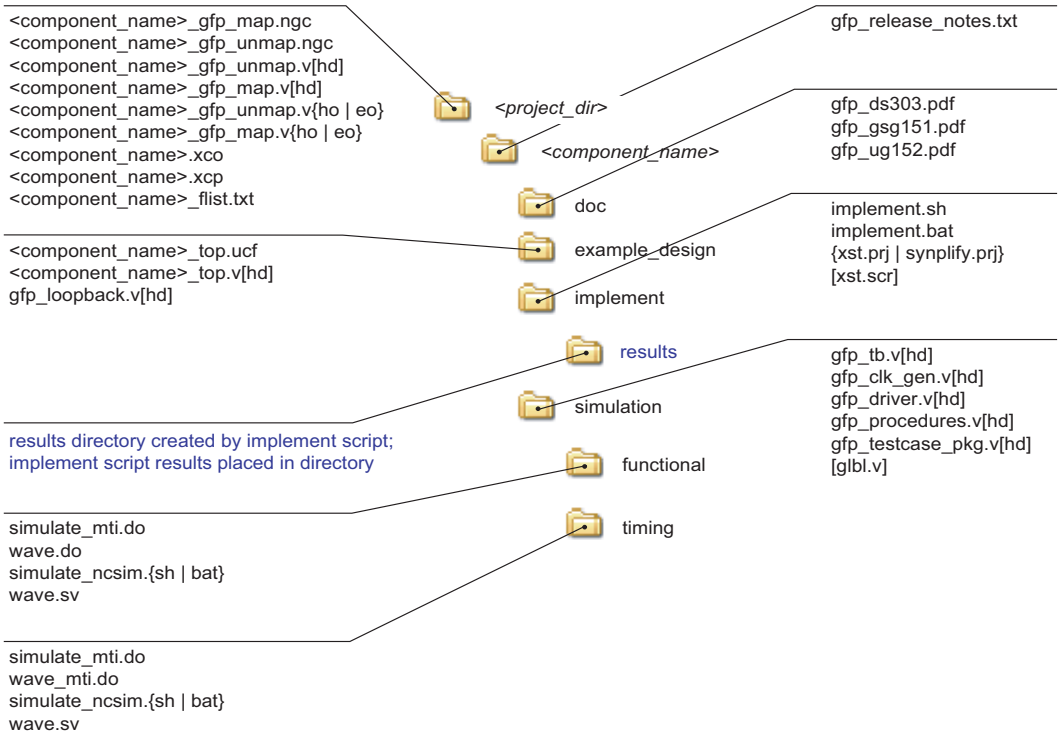

Figure 4-1: **GFP Directory Structure**

Project Directory (<project_dir>)

The CORE Generator provides the resulting MAP and UNMAP core files in the <project_dir> directory. The NGC files are the resulting core netlists, the VHDL and Verilog files are the functional simulation models, and the VHO and VEO files are instantiation templates. See the *GFP User Guide* for detailed information about each file.

```
<component_name>_[un]map.ngc
<component_name>_[un]map.v[hd]
<component_name>_[un]map.v{ho | eo}
<component_name>.xco
<component_name>_flist.txt
```

<project_dir>/<component_name>

`gfp_readme.txt`

The release notes text file.

<component_name>/doc

`gfp_ds303.pdf`

The *GFP Data Sheet*.

`gfp_gsg151.pdf`

The *GFP Getting Started Guide*.

`gfp_ug152.pdf`

The *GFP User Guide*.

<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 GFP 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 MAP and UNMAP clocks. Note that there are no IO placement constraints, as the GFP core is typically not routed to external pins. For additional information about the required GFP constraints, see the *GFP User Guide*.

Note that the example `.ucf` file is only generated for Full-System Hardware Evaluation and Full license types. See Chapter 2, "Licensing the Core" for information about licensing.

`<component_name>_top.v[hd]`

The VHDL or Verilog top-level file for the example design; instantiates the MAP and UNMAP cores and the loopback module.

`gfp_loopback.v[hd]`

The VHDL or Verilog line-side loopback that connects the MAP and UNMAP line interfaces.

<component_name>/implement

Note that the implement directory is only generated for Full-System Hardware Evaluation and Full license types. See Chapter 2, "Licensing the Core" for information about licensing.

`implement.{sh | bat}`

A Windows (.bat) or Linux (.sh) script that processes the example design through the Xilinx tool flow. For more information, see "Implementation Script Details," page 24.

`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.

<component_name>/implement/results

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

<component_name>/simulation

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

```
gfp_clk_gen.v[hd]
gfp_driver.v[hd]
gfp_procedure.v[hd]
gfp_tb.v[hd]
gfp_testcase_pkg.v[hd]
```

Two files, `gfp_driver` and `gfp_testcase_pkg` control the operation of the demonstration test bench and can be modified by the user. For information about modifying these files, see "Customization of the Demonstration Test Bench," page 28.

<component_name>/simulation/functional

`simulate_mti.do, simulate_ncsim.sh`

A macro file (ModelSim and Cadence IUS, respectively) that compiles the HDL sources and runs the simulation.

`wave.do, wave.sh`

A macro file (ModelSim and Cadence IUS, respectively) that opens a wave window and adds key signals to the wave viewer. The respective wave file is called by the corresponding simulate file, and is displayed after the simulation is loaded.

<component_name>/simulation/timing

Note that the timing simulation directory is only generated for Full-System Hardware Evaluation and Full license types. See Chapter 2, "Licensing the Core" for information about licensing.

`simulate_mti.do, simulate_ncsim.sh`

A macro file (ModelSim and Cadence IUS, respectively) 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, wave.sh`

A macro file (ModelSim and Cadence IUS, respectively) that opens a wave window and adds key signals to the wave viewer. The respective wave file is called by the corresponding simulate file, and is displayed after the simulation is loaded.

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:

`<project_dir>/<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

Simulation scripts are provided for ModelSim and Cadence IUS. The provided simulation scripts compile the simulation model and demonstration test bench, and run the simulation. The simulation files are located in the following directory:

`<project_dir>/<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 loopback example design
- Compiles the wrapper file, which instantiates the cores and the loopback
- 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 and the loopback
- Compiles the demonstration test bench
- Starts a simulation of the demonstration test bench

- Opens the waveform viewer and adds key signals
- Runs the simulation

Example Design Configuration

Figure 4-2 illustrates the configuration of the example design.

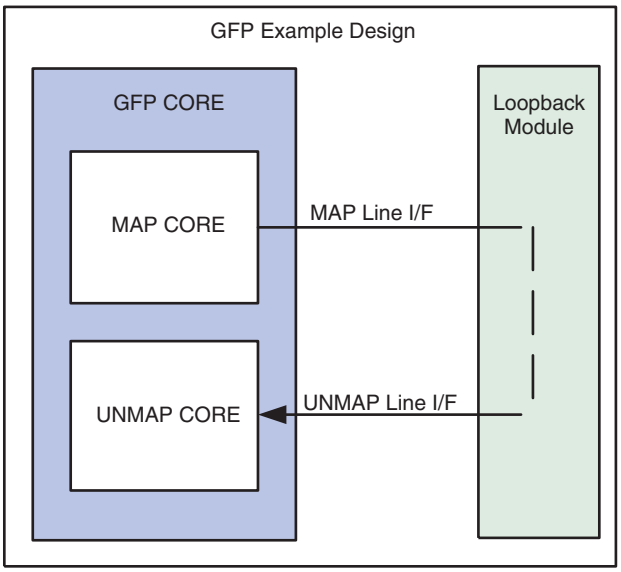

Figure 4-2: **Example Design Configuration**

Example design configuration:

- The example design instantiates a MAP core, UNMAP core, and loopback module.
- The MAP and UNMAP cores are instantiated as black-boxes, and replaced with the CORE Generator netlists during implementation, and the gate-level simulation model during simulation.
- The line interface loopback module is provided as source code, and is used in the demonstration test bench as well as implementation of the example design.
  In a system application, the loopback module may be used to help with system-level diagnostics by isolating the GFP core from the upstream application. Note that the operation of the loopback module is dependent on the configuration of the core, as described below.

- The configuration of the UNMAP core dictates the required data format the UNMAP core expects to receive.
  - If *No Hunting* is selected in the CORE Generator GUI, the UNMAP core requires word-aligned frames, using `SOF_N`, `EOF_N`, and `REM` to perform frame delineation. When the UNMAP core is configured for No Hunting, the loopback module wires the MAP and UNMAP line interfaces together.
  - If *Idle Only Hunting* or *Full Synchronization* is selected, the UNMAP core requires streaming data (`SOF_N`, `EOF_N`, `DSC_N`, and `REM` are not present). In this case, the loopback module is required to create a streaming interface out of aligned packets coming from the MAP core. The loopback module takes the the word-aligned data from the MAP interface and creates a packed stream of data for the UNMAP core. For detailed information about the UNMAP core behavior under these conditions, see the *GFP User Guide*.

Demonstration Test Bench

The GFP example design consists of three components:

- Gate-level simulation models of the MAP and UNMAP cores

  Generated from the netlists (UniSim simulation models) and uses the configurations selected in the GUI.

- A loopback module

  Connects the MAP and UNMAP line interfaces together enabling the MAP core to transfer data to the UNMAP core.

- The demonstration test bench

  Instantiates both the MAP and UNMAP cores, the loopback module, and the test bench files, which exercise the cores. The demonstration test bench is comprised of a number of modules that work together to generate and drive data into the MAP core's system interface, and read data from the UNMAP core's system interface. A brief description of each module is provided below.

Note: Because the MAP line interface is connected to the UNMAP line interface via the loopback module, the MAP and UNMAP cores must be configured in the same operating modes for the demonstration test bench to properly simulate. For example, if header scrambling is enabled in the MAP core, then header descrambling must be enabled in the UNMAP core.

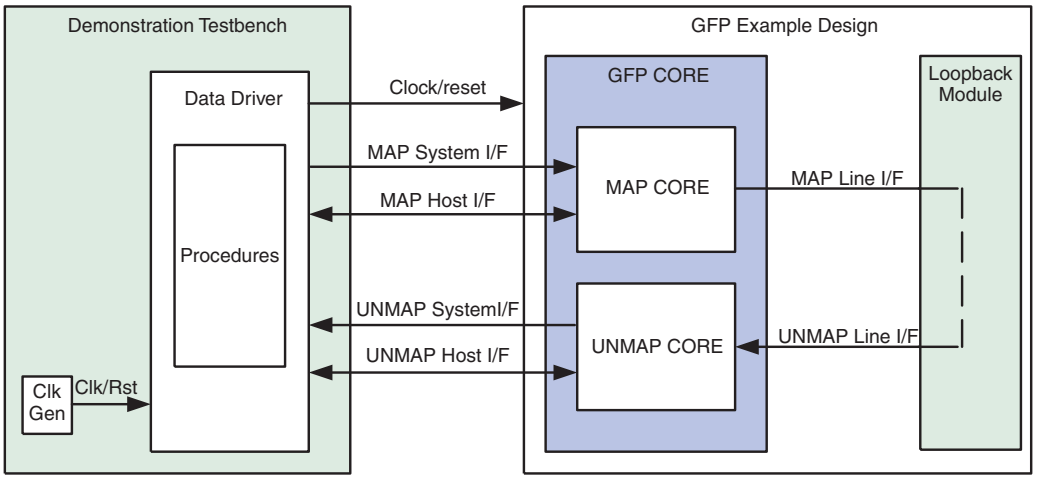

Figure 4-3: **Demonstration Test Bench**

Clock Generator Module (gfp_clk_gen.v[hd])

The clock generator module creates all clocks required for the demonstration test bench. The frequency of each core clock is defined in the testcase package, as defined below. By default, the example design connects all clocks to the MAP core clock `M_CLK`.

Procedure Module (gfp_procedure.v[hd])

The procedure module contains all functions and procedures used to create the bus functional model of the MAP core's system interface.

Testcase Package (gfp_testcase_pkg.v[hd])

The testcase package file contains all global parameters (constants). It also contains initialization values of the host interface registers as customized by the user through the GUI.

Data Driver Module (gfp_driver.v[hd])

The data driver module is the customer interface for passing data to/from the GFP core by transmitting data to the MAP core's system interface. Data is transferred from the data driver to the MAP core using a variety of procedure calls, which let the user customize the traffic patterns being sent and observe the core's responses.

The procedure calls support transmission of both valid and invalid frames, which enables the user to observe both normal operation and the core response under various error conditions. There are procedure calls specific to frame-mapped operation, transparent operation, and a common procedure call used for management frames.

Default Configurations of the Data Driver

Frame-Mapped Test Sequence

If only frame-mapped mode is selected, the default data driver transmits the following sequence:

- Twenty frames with incrementing data pattern
- One frame with pause cycles inserted in the middle of the frame
- One frame with source discontinue (`M_SYS_SRC_DSC_N`)
- Two management frames

This can be modified by the user by adding additional procedure calls as described below.

Transparent Test Sequence

If only transparent mode is selected, the default data driver transmits the following sequence:

- Twenty transparent frames (The actual size of the transparent mode GFP frame is determined by the number of superblocks per frame indicated in the host interface registers as set by the CORE Generator GUI.)
- One transparent frame with control characters (K-characters) and invalid control characters (10B errors) dispersed throughout the frame
- Two management frames

This can be modified by the user by adding additional procedure calls as described below.

Mixed Mode Test Sequence

If mixed mode is selected, the default data driver transmits the following sequence:

- Twenty frames (frames to all frame-mapped channels followed by frames to all transparent channels)
- Two management frames

Customization of 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:

```
quickstart/simulation/[vhdl | verilog]/gfp_testcase_pkg.v[hd]
quickstart/simulation/[vhdl | verilog]/gfp_driver.v[hd]
```

Customizing the Testcase Package

Seven 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. Note that *only* the parameters in the testcase package listed below should be changed by the user; the remaining testcase package parameters are internal to the test environment and should not be modified.

Table 4-1: **Customizable 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
M_CLK_PERIOD	N/A	Sets the M_CLK clock period (VHDL in ns, Verilog in ps)
U_CLK_PERIOD	N/A	Sets the U_CLK clock period (VHDL in ns, Verilog in ps)
M_HOST_CLK_PERIOD	N/A	Sets the M_HOST_CLK clock period (VHDL in ns, Verilog in ps)
U_HOST_CLK_PERIOD	N/A	Sets the U_HOST_CLK clock period (VHDL in ns, Verilog in ps)

Note that the default example design connects all core clocks to M_CLK.

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 frame transaction to the MAP core. The data driver module can be customized by performing different procedure calls within the main process, which are described below.

Frame-Mapped Procedures

The following procedures are valid for a core configured in frame-mapped mode:

- send_frame (length, CID)

  Transmits a complete frame of a specified length to the channel indicated by the CID. It waits for `M_SYS_DST_RDY_N` to be asserted before asserting `SOF_N`; then sends *length* number of words before asserting the `EOF_N`.

Table 4-2: **Frame-Mapped: send_frame**

Input	Range	Description
length	1-65535*	Number of bytes written between SOF_N and SOF_N
CID (Channel ID)	0-255	Channel identification number
* See the *GFP User Guide* for maximum length value.		

- send_part_frame (expected, CID, actual)

  Transmits a frame that does not match the length indicated on the `M_SYS_LENGTH` port. The *expected* value is applied on the `M_SYS_LENGTH` port while the frame in transit consists of *actual* number of valid words between `SOF_N` and `EOF_N`.

Table 4-3: **Frame-Mapped: send_part_frame**

Input	Range	Description
expected	1-65535*	Expected number of bytes between `SOF_N` and `EOF_N` (indicated on `M_SYS_LENGTH`).
CID	0-255	Channel identification number
actual	0-65535	The actual number of bytes between the `SOF_N` and `EOF_N`. When set to 0, `EOF_N` will not be asserted.
* See the *GFP User Guide* for maximum length value.		

- send_no_sof (length, CID)

  Transmits a complete frame of a specified length to the channel indicated by the CID. It waits for M_SYS_DST_RDY_N to be asserted; then starts to send data without an SOF_N. When *length* number of words are transmitted it asserts the EOF_N. Note that this causes the MAP core to drop this frame, because the frame did not start with an SOF_N.

Table 4-4: **Frame-Mapped: send_no_sof**

Input	Range	Description
length	1-65535*	Number of bytes between SOF_N and EOF_N
CID	0-255	Channel identification number
* See the *GFP User Guide* for maximum length value.		

- send_frame_w_idles (length, CID, idles, repeat_cnt)

  Transmits a complete frame of a specified length to the channel indicated by the CID with idle cycles (deassertion of the M_SYS_SRC_RDY_N signal) dispersed throughout the frame transmission. It waits for M_SYS_DST_RDY_N to be asserted; then starts the frame transmission by asserting SOF_N. The procedure will deassert M_SYS_SRC_RDY_N signal for *idles* number of cycles and then reassert it. It repeats this *repeat_cnt* number of times, sending data between each set of idles. If the *idles* or *repeat_cnt* input value is zero, it is set to a random value by the procedure.

Table 4-5: **Frame-Mapped: send_frame_w_idles**

Input	Range	Description
length	1-65535*	Number of words between SOF_N and EOF_N
CID	0-255	Channel identification number
idles	0-255	Duration of each M_SYS_SRC_RDY_N signal deassertion during frame transmission in number of clock cycles
repeat_cnt	0-15	Number of times to repeat deasserting M_SYS_SRC_RDY_N
* See the *GFP User Guide* for maximum length value.		

- send_frame_change_cid (length, CID1, byte_cnt, CID2)

  Transmits a complete frame of specified length. The M_SYS_CID signal is initially set to CID1, but after *byte_cnt* number of bytes are transmitted, the M_SYS_CID signal is changed to CID2. If *byte_cnt* is greater than *length*, it will change the CID after (length - 1) number of bytes. Note that this causes the MAP core to terminate the current frame by padding the frame until the length is met, and then start a new frame on the CID2 channel.

Table 4-6: **Frame-Mapped: send_frame_change_cid**

Input	Range	Description
length	1-65535*	Number of bytes between SOF_N and EOF_N
CID1	0-255	First channel identification number

Table 4-6: **Frame-Mapped: send_frame_change_cid *(Continued)***

Input	Range	Description
byte_cnt	0-(length-1)	Number of bytes to wait before changing to the second channel. If value is zero, it is set to a random value.
CID2	0-255	Second channel identification number
* See the *GFP User Guide* for maximum length value.		

- send_frame_dsc (length, CID, dsc_cnt)

  Transmits an incomplete frame that is terminated with the `M_SYS_SRC_DSC_N` assertion. It waits for `M_SYS_DST_RDY_N` to be asserted before asserting `SOF_N` and sending data. After *dsc_cnt* number of bytes are sent, `M_SYS_SRC_DSC_N` and `M_SYS_EOF_N` signals are asserted to prematurely terminate the frame.

Table 4-7: **Frame-Mapped: send_frame_dsc**

Input	Range	Description
length	1-65535*	Number of bytes between `SOF_N` and `EOF_N`
CID	0-255	Channel identification number
dsc_cnt	0-(length-1)	Number of bytes to wait before asserting `M_SYS_SRC_DSC_N` and `M_SYS_EOF_N`
* See the *GFP User Guide* for maximum length value.		

Transparent Procedures

- send_trans (CID)

  Transmits data to be encapsulated by the transparent mode GFP core to the indicated channel. It waits for the `M_SYS_FM_RDY_N` and `M_SYS_DST_RDY_N` to be asserted before asserting `M_SYS_SRC_RDY_N` to send data. The first and last byte of the frame are K-characters.

Table 4-8: **Transparent: send_trans**

Input	Range	Description
CID	0-255	Channel identification number

- send_user_trans (CID, num_k, num_err)

  Transmits data to be encapsulated by the transparent mode GFP core to the indicated channel. It waits for the `M_SYS_FM_RDY_N` and `M_SYS_DST_RDY_N` to be asserted before asserting `M_SYS_SRC_RDY_N` to send data. The `num_k` input indicates how many K-characters should be transmitted in a superblock. The `num_err` input indicates how many bytes of a superblock are flagged to be 10B_ERR bytes.

Table 4-9: **Transparent: send_user_trans**

Input	Range	Description
CID	0-255	Channel identification number
num_k	0-64	Number of K-characters in a superblock
num_err	0-64	Number of bytes with `M_SYS_10BERR_N` flag associated with them (can be k-characters)

- send_trans_change_cid (CID1, word_cnt, CID2)

  Transmits data to be encapsulated by the transparent mode GFP core. The data is initially transmitted with `M_SYS_CID` signal set to CID1, and after `word_cnt` number of bytes are transmitted, the `M_SYS_CID` signal is changed to CID2.

Table 4-10: **Transparent: send_trans_change_cid**

Input	Range	Description
CID1	0-255	First channel identification number
word_cnt	0-(NUM_SB*64)	Number of bytes to wait for before changing the CID. If the value is zero, it is set to a random value.
CID2	0-255	Second channel identification number

- send_trans_dsc (CID, dsc_cnt)

  Transmits data to be encapsulated by the transparent mode GFP core. The data is transmitted with `M_SYS_CID` signal set to CID. After `dsc_cnt` number of bytes are transmitted, the `M_SYS_SRC_DSC_N` signal is asserted. The first and last bytes of the transfer are K-characters.

Table 4-11: **Transparent: send_trans_dsc**

Input	Range	Description
CID	0-255	Channel identification number
dsc_cnt	0-(NUM_SB*64)	Number of bytes to wait for before asserting `M_SYS_SRC_DSC_N`. If the value is zero, it is set to a random value.

Common Procedures

- send_mgmt (CID, type)

  Used to send management frames to the core.

Table 4-12: **Common: send_mgmt**

Input	Range	Description
CID	0-255	Channel identification number
type	0,1	Indicates the type of management frame 0: Client Signal Fail (Loss of Client Signal) 1: Client Signal Fail (Loss of Character Synchronization)

Host Interface Procedures

- host_read_addr (data, address)

  Read out contents of the specified configuration register.

Table 4-13: **Host: host_read_addr**

Input/Output	Range	Description
data (output)	std_logic_vector (31 down to 0)	Data read from the register
address (input)	Constants defined in the gfp_testcase_pkg.vhd	Register address

- host_write_addr (data, address)

  Write to a specified configuration register.

Table 4-14: **Host: host_write_addr**

Input	Range	Description
data	std_logic_vector (31 down to 0)	Data to be written into the register
address	Constants defined in the gfp_testcase_pkg.vhd	Register address

Verilog Procedure Calls

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

```
procedures.send_frame(31, 0);
procedures.send_trans(0);
procedures.send_mgmt(7, 0);
procedures.host_read_addr(HOST_DATA_BUFFER,'MAP_GFP_VERSION)
procedures.host_write_addr(32'hFFFFFFFF,'MAP_GFP_CTRL);
```

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.

```
send_frame(pro_bus => pro_bus, length => 31,cid => 0);
send_trans(pro_bus => pro_bus, cid => 0);
send_mgmt(pro_bus => pro_bus, cid => 7, type => 0);
host_read_addr(host_bus => host_bus, data => host_data,address =>
MAP_GFP_VERSION);
host_write_addr(host_bus => host_bus, data => x”FFFFFFFF” address =>
MAP_GFP_CTRL);
```