- DISCONTINUED PRODUCT -

LogiCORE™ Generic Framing Procedure v2.1

User Guide

UG152 April 25, 2008

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
9/30/04	1.1	Initial Xilinx release.
11/11/04	1.2	Updated 64-bit information, added area constraints section.
4/28/05	1.3	Updated to Xilinx tools 7.1i SP1.
1/18/06	1.4	Update to ISE™ tools 8.1i.
4/25/08	2.1	Updated to ISE® v10.1 and added support for Virtex®-5 devices.
09/29/09 - This is the final publication. No content was changed.		

Table of Contents

Schedule of Figures 7

Schedule of Tables 9

Preface: About This Guide

Guide Contents 11
Conventions 12
Typographical 12
Online Document 13

Chapter 1: Introduction

About the Core 15
Recommended Design Experience 15
Additional Core Resources 15
Design Resources 16
Technical Support 16
Feedback 16
GFP Core 16
Document 16

Chapter 2: Installing and Licensing the Core

System Requirements 17
Before you Begin 17
Installing the Core 17
CORE Generator IP Updates Installer 18
Manual Installation 18
Verifying your Installation 19
License Options 19
Simulation Only Evaluation 20
Full System Hardware Evaluation 20
Full 20
Obtaining Your License 20
Installing Your License File 21

Chapter 3: Core Architecture

System Overview 23
MAP Core 24
System Interface 24
Line Interface 25
Host Interface 25
UNMAP Core 25
System Interface 25
Line Interface 25

Host Interface 26
MAP Core Interfaces 26
Common Interface 27
System Interface (M_SYS) 27
Line Interface (M_LINE) 30
Host Interface (M_HOST) 31
MAP Configuration Space 32
UNMAP Core Interfaces 34
Common Interface 35
System Interface (U_SYS) 35
Line Interface (U_LINE) 39
Host Interface (U_HOST) 40
UNMAP Configuration Space 41

Chapter 4: Generating the Core

CORE Generator Graphical User Interface 43
Main Screen 44
Customizing Core Features 46
MAP Core Register Configuration: General Registers 47
MAP Core Register Configuration: Per-Channel Configuration 50
UNMAP Core Register Configuration: General Registers 52

Chapter 5: Customizing the Core

Customizing the GFP Core 55
MAP General Registers 55
MAP Channel Specific Registers 58
UNMAP General Registers 59

Chapter 6: Designing with the Core

General Design Guidelines 63
Know the Degree of Difficulty 63
Understand Signal Pipelining 63
Keep it Registered 64
Use Supported Design Flows 64
Make Only Allowed Modifications 64
Initializing the GFP Core 64
MAP Core 65
Basic Operation 65
Operating the MAP System Interface with Frame-Mapped Frames 65
Operating the MAP System Interface with Transparent Frames 69
Operating the MAP System Interface in Mixed Mode 74
Operating the MAP Line Interface 75
Operating the MAP Core with Multiple-Channel Support 76
UNMAP Core 77
Basic Operation 77
Operating the UNMAP System Interface with Frame-Mapped Frames 77
Operating the UNMAP System Interface with Transparent Frames 79
Operating the UNMAP System Interface in Mixed Mode 83
Operating the UNMAP Line Interface with Streaming Data 84
Operating the UNMAP Line Interface with LocalLink Data 85

Accessing Control and Status Registers 86
Register Space 86
Address Space 86
Operating the Host Interface 88
Interrupts 90

Chapter 7: Constraining the Core

Required Constraints 91
Timing Constraints 91
Area Constraints 92
Optional User Constraints 92

Chapter 8: Special Design Considerations

I/O Pin Placement 93
Clocking 93
Common Use Cases 93

Chapter 9: Simulating and Implementing Your Design

Functional Simulation 97
Timing Simulation 97
Synthesis 98
Synthesis of Example Design 98
Xilinx Tool Flow 98
Example Design Script 99
NGDBuild 99
Mapping the Design 99
Place and Route 99
Static Timing Analysis 99
Timing Simulation 100
Generating a Bitstream 100

Appendix A: Core Directory Structure

Appendix B: Core Verification

Common Conditions 105
MAP Core Specific Verification 105
UNMAP Core Specific Verification 106

Appendix C: Packet and Control Symbol Format

Appendix D: Status and Error Reporting

MAP Core 111
UNMAP Core 117

Appendix E: Sample GFP Frames

Schedule of Figures

Chapter 2: Installing and Licensing the Core

Figure 2-1: **CORE Generator Window** 19

Chapter 3: Core Architecture

Figure 3-1: **GFP Core Block Diagram** 24
Figure 3-2: **MAP Core Interfaces** 26
Figure 3-3: **UNMAP Core Interfaces** 34

Chapter 4: Generating the Core

Figure 4-1: **Main GFP Screen** 44
Figure 4-2: **MAP/UNMAP Customization Features** 46
Figure 4-3: **MAP General Registers Configuration** 48
Figure 4-4: **Map Core Base Address** 49
Figure 4-5: **Map Per Channel Configuration** 51
Figure 4-6: **UNMAP General Registers Configuration** 53
Figure 4-7: **UNMAP Core Base Address** 53

Chapter 6: Designing with the Core

Figure 6-1: **MAP Core Frame-Mapped Transfer** 69
Figure 6-2: **MAP Core Transparent Transfer** 74
Figure 6-3: **MAP Core Transparent Transfer** 76
Figure 6-4: **UNMAP Core Frame-Mapped Transfer** 79
Figure 6-5: **UNMAP Core Transparent Transfer** 83
Figure 6-6: **UNMAP Streaming Interface** 85
Figure 6-7: **UNMAP Line Locallink Interface** 86
Figure 6-8: **MAP Register Space** 87
Figure 6-9: **UNMAP Register Space** 88
Figure 6-10: **Host Interface Waveform** 89

Chapter 8: Special Design Considerations

Figure 8-1: **Transmitting Ethernet over SONET/SDH** 94
Figure 8-2: **Transmitting Fibre Channel over SONET/SDH** 95

Appendix A: Core Directory Structure

Figure A-1: **GFP Directory Structure** 101

Appendix C: Packet and Control Symbol Format

Figure C-1: **GFP Frame Format** 107

Schedule of Tables

Chapter 3: Core Architecture

Table 3-1: **MAP Core Bus Widths** . . . 27
Table 3-2: **MAP Common Interface** . . . 27
Table 3-3: **MAP System Interface** . . . 28
Table 3-4: **MAP Line Interface** . . . 30
Table 3-5: **MAP Host Interface** . . . 31
Table 3-6: **MAP Core General Registers** . . . 32
Table 3-7: **MAP Core Channel-Specific Registers** . . . 33
Table 3-8: **UNMAP Core Bus Widths** . . . 35
Table 3-9: **UNMAP Common Interface** . . . 35
Table 3-10: **UNMAP System Interface** . . . 36
Table 3-11: **UNMAP Line Interface** . . . 39
Table 3-12: **UNMAP Host Interface** . . . 40
Table 3-13: **UNMAP Core General Registers** . . . 41

Chapter 4: Generating the Core

Table 4-1: **MAP GUI Customization: GFP_CTRL Register** . . . 49
Table 4-2: **MAP GUI Customization: GFP_ERR Register** . . . 49
Table 4-3: **MAP GUI Customization: GFP_INTMASK Register** . . . 50
Table 4-4: **MAP GUI Customization: CHANx_GFP_REGISTER_A** . . . 51
Table 4-5: **MAP GUI Customization: CHANx_GFP_REGISTER_B** . . . 52
Table 4-6: **UNMAP GUI Customization: GFP_CTRL Register** . . . 54
Table 4-7: **UNMAP GUI Customization: GFP_INTMASK Register** . . . 54

Chapter 5: Customizing the Core

Table 5-1: **MAP Core General Registers** . . . 55
Table 5-2: **MAP Core Channel-Specific Registers** . . . 58
Table 5-3: **UNMAP Core General Registers** . . . 59

Chapter 6: Designing with the Core

Table 6-1: **Maximum Length Values** . . . 66
Table 6-2: **M_SYS_DATA Byte to 8b/10b Control Code Mapping** . . . 70
Table 6-3: **Transparent Mode Maximum Length Frames** . . . 72
Table 6-4: **U_SYS_DATA Byte to 8b/10b Control Code Mapping** . . . 80
Table 6-5: **MAP Register Space Example for 10 Channels and a Base Address of 5** . . . 87

Appendix A: Core Directory Structure

Table A-1: **Directory Files** 101

Appendix C: Packet and Control Symbol Format

Table C-1: **GFP Frame Field Descriptions** 107
Table C-2: **UPI for Client Data Frames (PTI = 000)** 108
Table C-3: **UPI for Client Management Frames (PTI = 100)** 109
Table C-4: **GFP Host Interface to DCR Bus Mapping** 109

Appendix D: Status and Error Reporting

Table D-1: **M_SYS_STATUS_N Description** 111
Table D-2: **M_SYS_STATUS_N[1] Behavior** 112
Table D-3: **MAP Core Response to Varying Frame Sizes and M_SYS_SRC_DSC_N** . . 113
Table D-4: **MAP GFP_INT Description** 115
Table D-5: **: MAP GFP_ERR Description** 116
Table D-6: **U_SYS_STATUS_N Description** 117
Table D-7: **U_SYS_ERRBUS_N Description** 118
Table D-8: **UNMAP GFP_INT Description** 120
Table D-9: **UNMAP GFP_FIXERR Description** 121

Appendix E: Sample GFP Frames

Table E-1: **First Constructed GFP Frame for Various Core Configurations** 123

About This Guide

The *Generic Framing Procedure User Guide* describes the function and operation of the LogiCORE™ IP Generic Framing Procedure (GFP) core, as well as information about designing, customizing, and implementing the core.

Guide Contents

The following chapters are included:

- "Preface, About this Guide" describes how the User Guide is organized, and 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, "Installing and Licensing the Core" provides information about installing and licensing the core.
- Chapter 3, "Core Architecture" describes the GFP architecture and discusses the user signal interface.
- Chapter 4, "Generating the Core" describes how to generate using the Xilinx CORE Generator GUI.
- Chapter 5, "Customizing the Core" describes how to customize the core for specific applications.
- Chapter 6, "Designing with the Core" provides guidelines for initializing the GFP link, generating and consuming GFP packets, operating the management interface, and handling discontinues. It also discusses the error insertion and detection capabilities of the core.
- Chapter 7, "Constraining the Core" discusses the specification of timing constraints to meet the performance requirements, which are provided in a user constraints file (.ucf).
- Chapter 8, "Special Design Considerations" discusses specific design considerations that must be considered when designing for the core.
- Chapter 9, "Simulating and Implementing Your Design" provides instructions for setting up a synthesis, simulation, and implementation environment. It also instructs the user on generating a bitstream using the design flow.
- Appendix A, "Core Directory Structure" defines the core directory structure and associated file contents.
- Appendix B, "Core Verification" describes the verification methods used in an internal Xilinx test environment utilizing a a Xilinx-developed bus functional model (BFM).

- Appendix C, "Packet and Control Symbol Format" describes the GFP packet format.
- Appendix D, "Status and Error Reporting" describes the status and error reporting behaviors of the MAP and UNMAP cores.
- Appendix E, "Sample GFP Frames" illustrates a sample data frame and the resulting GFP encapsulation for various core configurations.

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`
Italic font	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.
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 `45524943`h.
	An '_n' means the signal is active low	`usr_teof_n is active low.`

Online Document

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.

Chapter 1

Introduction

The 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 delivered with the core 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.

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 the GFP product page. VHDL and Verilog functional simulation models are provided for immediate customer evaluation.

For information about system requirements, installation, and licensing options, see Chapter 2, "Installing and 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 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 following documents, located on the GFP product page.

Design Resources

The following resource information may be helpful when designing a GFP core.

Resource	URL/Description
ITU-T GFP Specification	www.itu.int/home/
IBM Device Control Register Specification	The IBM DCR specification is described in *SA-14-2525-00.*
SPI-4.2 IP Core	www.xilinx.com/products/ipcenter/DO-DI-POSL4MC.htm
Ethernet IP Cores	www.xilinx.com/products/ipcenter/DO-DI-GMIITO1GBSXPCS.htm
XAPP759, *Configurable Physical Coding Sublayer*	www.xilinx.com/support/documentation/application_notes/xapp759.pdf
Xilinx LocalLink Specification	The LocalLink interface specification can be accessed after registering for the Aurora Reference Design.

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. 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. 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

Installing and 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.

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

Before you Begin

Before installing the core, you must have a Xilinx.com account and the ISE v10.1 software installed on your system. If you have already completed these steps, go to "Installing the Core."

1. Click "Sign in to access account" at the top of the Xilinx home page; then follow the onscreen instructions to create a support account.
2. Install the ISE software and the applicable Service Pack software. ISE Service Packs can be downloaded from www.xilinx.com/support/download.

Installing the Core

You can install the core in two ways—using the CORE Generator IP Updates Installer, which lets you select from a list of updates, or by performing a manual installation after downloading the core from the web.

CORE Generator IP Updates Installer

Note: To use this installation method behind a firewall, you must know your proxy settings. Contact your administrator to determine the proxy host address and port number before you begin, if necessary.

1. Start the CORE Generator; then open an existing project or create a new one.
2. From the main CORE Generator window, choose Tools > Updates Installer to start the Updates Installer. If you are behind a firewall, you will be prompted to enter your proxy host and port settings.
3. If necessary, enter your proxy settings; then click Set. The IP Updates installer appears.
4. Click the checkbox next to 10.1_IP_Update1 to select it; then click Install Selected. Informational messages may appear indicating that additional installations are required.
5. Click OK to accept any messages and continue. The User Login dialog box appears.
6. Enter your login name and password; then click OK. The Updates Installer Generator downloads and installs the selected products, and then exits.
7. To confirm the installation, check the following file: `C:\Xilinx\coregen\install\install_history`.

   Note that this step assumes your Xilinx software is installed in C:\Xilinx.

Manual Installation

1. Close the CORE Generator application if it is running.
2. Download the IP Update ZIP file from the following location and save it to a temporary directory: www.xilinx.com/support/download.htm
3. Unpack the ZIP files using either WinZip (Windows) or Unzip (Linux).
4. Extract the ZIP file (ise_101_ip_update1.zip) archive to the root directory of your Xilinx software installation. (Allow the extractor utility you use to overwrite all existing files and maintain the directory structure defined in the archive.)
5. If you do not have a zip utility, do one of the following:
   - **Windows**. From a command window, type the following:

     ```
     %XILINX%/bin/nt/unzip -d %XILINX% ise_101_ip_update1.zip
     ```

   - **Linux**. From a command shell, type the following:

     ```
     $XILINX/bin/lin/unzip -d $XILINX ise_101_ip_update1.zip
     ```

6. To verify the root directory of your Xilinx installation, do one of the following:
   - **Windows**. Type `echo %XILINX%` from a DOS prompt.
   - **Linux**. If you have already installed the Xilinx ISE software, the Xilinx variable defined by your set-up script identifies the location of the Xilinx installation directory. After sourcing the Xilinx set-up script, type `echo $XILINX` to determine the location of the Xilinx installation.

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.

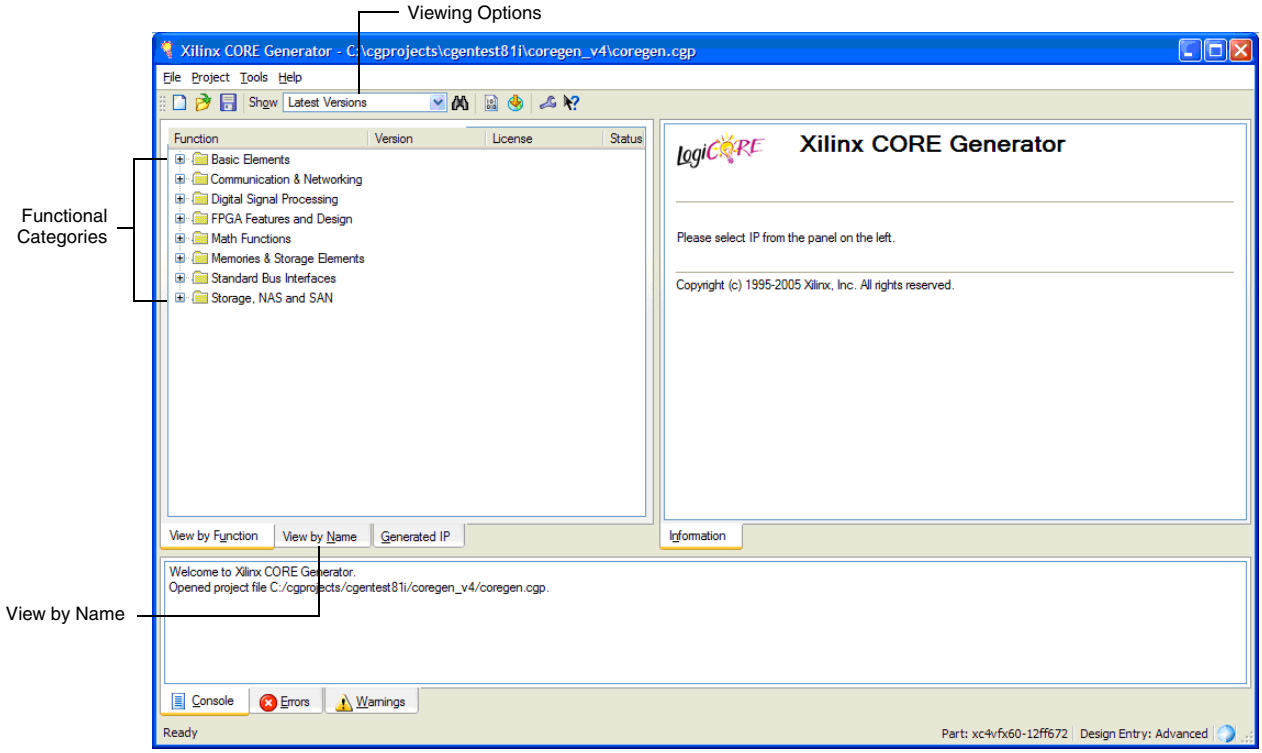

Figure 2-1: **CORE Generator Window**

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: http://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

Core Architecture

This chapter describes the GFP core architecture and interface signals.

System Overview

The GFP core is comprised of two separate cores that enable both the transmission (MAP core) and reception (UNMAP core) of data.

- The MAP core receives client network protocol data (such as Ethernet), encapsulates this data into GFP frames, and transmits the resulting frames to a SONET/SDH network.
- The UNMAP receives GFP encapsulated frames from a SONET/SDH network, de-maps the GFP frames into client network protocol data, and passes the data on to the client interface.

Figure 3-1 illustrates the MAP and UNMAP cores. Each core is comprised of three interfaces:

- **System interface**. Transmits data between the user's client interface and the GFP core. The data on the system interface is the client network protocol data, such as Ethernet or Fibre Channel.
- **Line interface**. Transmits data between the GFP core and the user's line interface. The data on the line interface is GFP encapsulated data, such as frame-mapped Ethernet, or transparent mode Fibre Channel.
- **Host interface**. Provides access to the GFP control and status registers and allows you to update the core behavior and error handling in-situ.

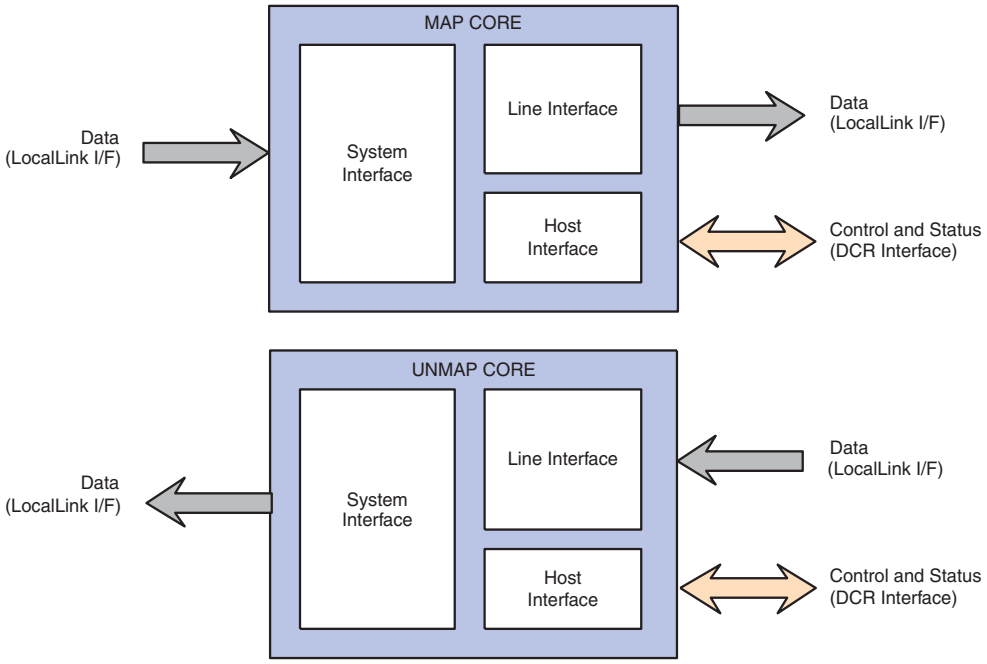

Figure 3-1: **GFP Core Block Diagram**

A brief overview of the MAP and UNMAP cores is provided in the next section, followed by a detailed description of their interface signals.

MAP Core

The MAP core maps client network protocol data into GFP frames. Depending on the core configuration, the MAP core performs the following operations:

- Header scrambling
- Insertion of core, type, and extension header error check (cHEC, tHEC, and eHEC)
- Payload scrambling
- Construction of superblocks (in transparent mode)
- FCS generation
- Transmission of idle and management frames

For information about these features and core customization, see "Customizing Core Features," page 46.

System Interface

The MAP system interface receives data that is ready to be encoded into GFP frames. In frame-mapped mode, the system interface requires complete frames of data, because there is a 1:1 correlation between client data frames and frame-mapped GFP frames. In transparent mode, the system interface expects a stream of 8b/10b decoded data words. Complete frames of data are not required because a fixed number of blocks of data are combined to create the transparent frame. For detailed information about using the MAP core system interface, see "MAP Core," page 65.

Line Interface

The MAP line interface passes the GFP encapsulated frame onto the line for transmission across a SONET/SDH network. The complete GFP frames are encapsulated within the data signal, and additional control signals are provided for easy connection to system packet interface (SPI) cores. For detailed information about using the MAP core line interface, see "Operating the MAP Line Interface," page 75.

Host Interface

The MAP host interface is an optional interface that provides in-situ access to configuration and status registers. The MAP core default operation is set via the CORE Generator GUI along with the initial values of the host interface. When the host interface is present, it can be used to change the default operation of the core. When the host interface is not present, the core operates using the initial values set during core generation. For detailed information about the MAP core configuration space, see "MAP Core Register Configuration: General Registers," page 47.

UNMAP Core

The UNMAP core de-maps GFP frames into client network protocol data. Depending on the core configuration the UNMAP core performs the following operations:

- Synchronization
- Header descrambling
- Detection/correction of core, type, and extension header error check (cHEC, tHEC, and eHEC)
- Payload descrambling
- Disassembly of superblocks (in transparent mode)
- FCS verification
- Reception and removal of idle frames

For information about these features and customizing the core, see "Customizing Core Features," page 46.

System Interface

The UNMAP system interface provides GFP decoded data to the user for transmission to their client network. In frame-mapped mode, the system interface provides the client protocol (such as Ethernet) extracted from the GFP frame. In transparent mode, the system interface provides a stream of 8b/10b data words (such as Fibre Channel), which are extracted from the GFP frame. For detailed information about using the UNMAP core system interface, see "UNMAP Core," page 77.

Line Interface

The UNMAP line interface receives GFP encoded data from the SONET/SDH network. The line interface supports either LocalLink signaling, or a streaming data interface, dependent on the configuration of the UNMAP core. For detailed information about using the UNMAP core line interface, see "Operating the MAP Line Interface," page 75.

Host Interface

The UNMAP host interface is an optional interface that provides in-situ access to configuration and status registers. The UNMAP core default operation is set via the CORE Generator GUI along with the initial values of the host interface. When the host interface is present, it can be used to change the default operation of the core. When the host interface is not present, the core operates using the initial values set during core generation. For detailed information about the UNMAP core configuration space, see "UNMAP Core General Registers," page 59.

MAP Core Interfaces

Figure 3-2 illustrates the MAP core interfaces. All signals are defined in their respective sections below the illustration. The MAP core utilizes the following top-level signal interfaces.

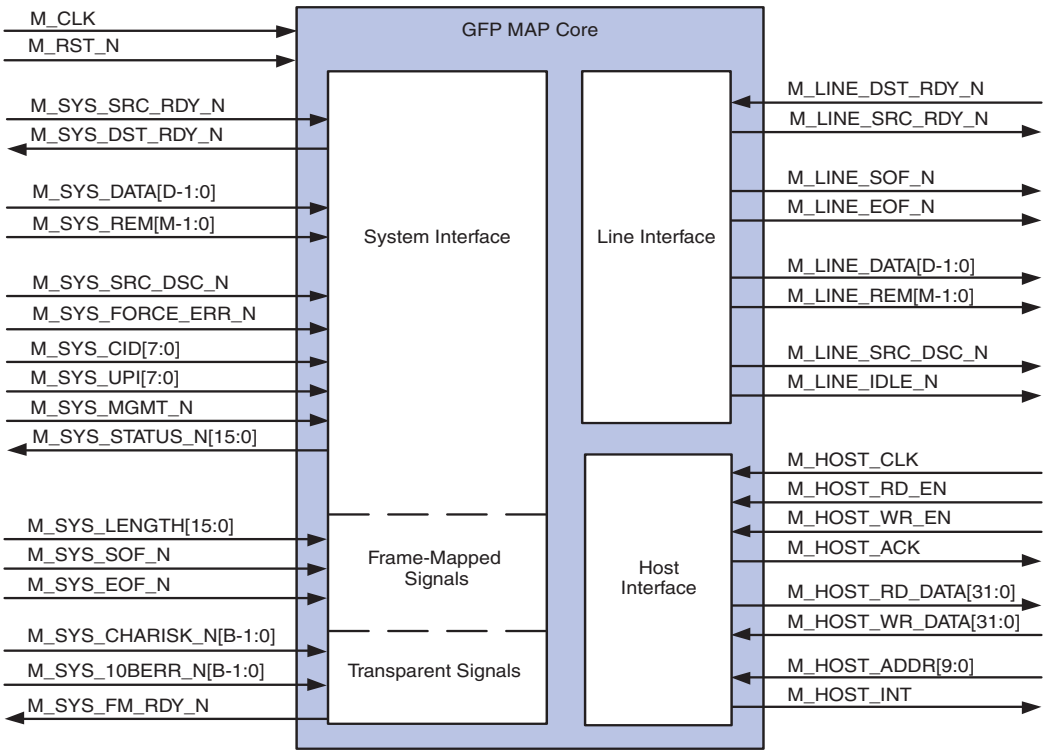

Figure 3-2: **MAP Core Interfaces**

Table 3-1 describes the relationship between the data bus width and additional signals. The MAP core supports both a 32-bit and a 64-bit interface. The width of many bus signals depends on the interface width selected (32 or 64 bits). Note that a signal ending in _N is active low, otherwise, the signal is active high. All signals apply to both frame-mapped and transparent mode unless otherwise noted.

Table 3-1: **MAP Core Bus Widths**

Data Bus Width (D) (*_DATA)	Remainder Width (M) (*_REM)	Data Byte Width (B) (*_10BERR_N, *_CHARISK_N)
32	2	4
64	3	8

Common Interface

Table 3-2 defines the signals common to the entire MAP core. The MAP reset signal (M_RST_N) causes a hard reset of the entire core (core logic and host interface). This signal is an asynchronous input which is synchronized internally in the core before being used. The initial hardware reset should be generated by the user. Subsequent resets may be asserted by using the M_RST_N pin (for a complete core reset), or by driving the specific reset register in the MAP host interface (for core logic reset only). For detailed information about reset requirements and operation, see "Initializing the GFP Core," page 64.

Table 3-2: **MAP Common Interface**

Name	Direction	Description
M_CLK	Input	**MAP Clock:** All system and line interface operations are synchronous to this clock.
M_RST_N	Input	**MAP Reset:** Asynchronous reset that resets both the MAP core logic and MAP host interface.

System Interface (M_SYS)

Table 3-3 describes the MAP system interface signals (M_SYS). The MAP system interface connects to the client side of the system and implements the Xilinx LocalLink standard, providing a simple, flexible way to receive frames. The system interface consists of a unidirectional data bus with control signals that allow the user application to stall or discontinue data. The system interface signals are divided into three categories: signals common to both frame-mapped and transparent mode, signals specific to frame-mapped mode, and signals specific to transparent mode.

Table 3-3: **MAP System Interface**

Name	Direction	Description
Common to all modes		
M_SYS_SRC_RDY_N	Input	**Write Enable:** Indicates a word presented by the client is valid (not accepted until M_SYS_DST_RDY_N is also asserted).
M_SYS_DST_RDY_N	Output	**Write Accepted:** Indicates a word presented by the client will be accepted (if M_SYS_SRC_RDY_N is also asserted).
M_SYS_DATA[D-1:0]	Input	**Data bus:** Client-side data to be encapsulated into a GFP frame. If Ethernet is being transferred (frame-mapped mode), then the first word sent should be the destination address. In transparent mode, the data will be raw 8b data.
M_SYS_REM[M-1:0]	Input	**Data Remainder:** The number of valid bytes in M_SYS_DATA is M_SYS_REM + 1, and is MSB justified. In frame-mapped mode, this is only valid when M_SYS_EOF_N is asserted. In transparent mode, this can be asserted at any time (core will insert 65B_PAD words to fill gaps if necessary). Example: 32-bit data bus: REM = “00” => DATA[31:24] valid REM = “01” => DATA[31:16] valid REM = “10” => DATA[31:8] valid REM = “11” => DATA[31:0] valid
M_SYS_SRC_DSC_N	Input	**Discontinue Frame:** Causes the frame currently in transit to terminate (once the specified length is reached) and invert its frame check sequence (FCS). In frame-mapped mode, this signal is only valid when M_SYS_EOF_N is asserted.
M_SYS_FORCE_ERR_N	Input	**Force Error:** Inserts errors as specified by the MAP host register GFP_ERR. This signal should be asserted for the entire frame in which an error is to be inserted.
M_SYS_CID[7:0]	Input	**Channel Identifier:** Indicates the channel id for linear frames. This signal is only present if linear extension headers are enabled.

Table 3-3: **MAP System Interface *(Continued)***

Name	Direction	Description
M_SYS_UPI[7:0]	Input	**User Payload Identifier:** Indicates the payload type for the current data frame. When M_SYS_MGMT_N is asserted, this signal indicates the type of management frame to send. See Appendix C, "Packet and Control Symbol Format" for UPI definitions.
M_SYS_MGMT_N	Input	**Management Frame Indicator:** Generates a client management frame to be transmitted on the line interface. When asserted, M_SYS_UPI will be used to indicate the type of client signal fail (CSF). See "Management Frames," page 68 for detailed use of this signal.
M_SYS_STATUS_N[15:0]	Output	**Status Bus:** Provides real-time status and errors. [15:6] = reserved [5] = Loss of client signal management frame transmitted due to CSF timer expiration [4] = Loss of character synchronization management frame transmitted due to CSF timer expiration [3:2] = reserved [1] = Invalid K character written in the system interface (mapped to 10B_ERR) [0] = PLI mismatch
Frame-mapped signals		
M_SYS_SOF_N	Input	**Start of Frame:** Indicates the beginning of either a data or management frame.
M_SYS_EOF_N	Input	**End of Frame:** Indicates the end of either a data or management frame.
M_SYS_LENGTH[15:0]	Input	**Payload Length:** Indicates the length of the current frame in bytes. This is only active when M_SYS_SOF_N is asserted.
Transparent signals		
M_SYS_FM_RDY_N	Output	**Frame Ready:** Look-ahead signal which indicates that the next transparent frame will begin in no fewer than 4 cycles (in 32-bit) or 2 cycles (in 64-bit). See "Operating the MAP System Interface with Transparent Frames," page 69.

Table 3-3: **MAP System Interface *(Continued)***

Name	Direction	Description
M_SYS_CHARISK_N[B-1:0]	Input	**K Character Indicator:** Indicates that the corresponding byte of M_SYS_DATA is a 8b/10b K character, not a data character. This signal will be ignored for a given byte lane if M_SYS_REM indicates that the byte is not valid.
M_SYS_10BERR_N[B-1:0]	Input	**Insert 10B_ERR:** Indicates that the corresponding byte of M_SYS_DATA should be replaced with the 10B_ERR code. This signal is ignored for a given byte lane if M_SYS_REM indicates that the byte is not valid. This signal overrides M_SYS_CHARISK_N if both signals are simultaneously asserted.

Line Interface (M_LINE)

Table 3-4 describes the MAP line interface signals (M_LINE). The MAP line interface utilizes the Xilinx LocalLink interface for a simple yet flexible medium for transmitting frames. It consists of a unidirectional data bus with additional control signals.

Table 3-4: **MAP Line Interface**

Name	Direction	Description
M_LINE_DST_RDY_N	Input	**Read Enable** (Destination Ready): Indicates a word is accepted this cycle (not accepted until M_LINE_SRC_RDY_N is also asserted).
M_LINE_SRC_RDY_N	Output	**Read Valid** (Source Ready): Indicates a word presented by the line interface is valid (not read until M_LINE_DST_RDY_N is also asserted).
M_LINE_SOF_N	Output	**Start of Frame:** Indicates the beginning of a frame.
M_LINE_EOF_N	Output	**End of Frame:** Indicates the end of a frame.
M_LINE_DATA[D-1:0]	Output	**Data bus:** Transmits GFP encapsulated data transmitted on the line interface.
M_LINE_REM[M-1:0]	Output	**Data Remainder:** The number of valid bytes in M_LINE_DATA is M_LINE_REM + 1, and is MSB justified. This signal is only valid when M_LINE_EOF_N is asserted. See M_SYS_REM for valid byte mappings.

Table 3-4: **MAP Line Interface *(Continued)***

Name	Direction	Description
M_LINE_SRC_DSC_N	Output	**Frame Discontinue:** Indicates the current GFP frame contains an error.
M_LINE_IDLE_N	Output	**Idle Indicator:** Indicates the current GFP frame is an idle frame, and can be dropped by the user if necessary. This signal is asserted for the entire idle frame.

Host Interface (M_HOST)

Table 3-5 describes the Map host interface signals (M_HOST). The MAP host interface to the user application utilizes the DCR bus for direct connection to the PowerPC, Microblaze, or other microprocessors. It consists of separate read and write data buses with a shared address bus. Note that the host interface is optional, and may be removed to conserve resources if in-situ access to control registers is not required.

Table 3-5: **MAP Host Interface**

Name	Direction	Description
M_HOST_CLK	Input	**Host Clock:** All host interface signals are synchronous to this clock. Note that this signal is optional, as the host interface can be configured to be synchronous to M_CLK.
M_HOST_RD_EN	Input	**Read Enable:** Indicates a read access to the register addressed on M_HOST_ADDR.
M_HOST_WR_EN	Input	**Write Enable:** Indicates a write access to the register addressed on M_HOST_ADDR.
M_HOST_ACK	Output	**Acknowledge:** Both read and write requests to the host interface are acknowledged through this signal. Following a read request, this signal indicates that the data on M_HOST_RD_DATA is valid.
M_HOST_RD_DATA[31:0]	Output	**Read Data:** Data read from the address M_HOST_ADDR, which is valid when a read request (M_HOST_RD_EN) followed by an acknowledge (M_HOST_ACK) is asserted.
M_HOST_WR_DATA[31:0]	Input	**Write Data:** Data to be written into the host interface at the address specified by M_HOST_ADDR. The write is acknowledged on M_HOST_ACK when the write succeeded.

Table 3-5: **MAP Host Interface *(Continued)***

Name	Direction	Description
M_HOST_ADDR[9:0]	Input	**Register Address:** Bus used to specify the address being accessed either for a read or write.
M_HOST_INT	Output	**Interrupt:** Indicates that an interrupt condition has been detected, and will continue to be driven until the interrupt is cleared (by writing to MAP GFP_INT).

MAP Configuration Space

The MAP core supports a host interface for access to control and status registers in-situ. The MAP core provides both general and channel-specific registers.

- Table 3-6 describes the MAP core general registers.
- Table 3-7 describes the MAP core channel-specific registers, which enable the user to customize specific parameters on a per channel basis.

Table 3-6: **MAP Core General Registers**

Register Name	Type	Description	DCR Offset
GFP_VERSION	R/W	[31] = Core reset	0x0
	R	[30:0] = Core version number	
GFP_CTRL		[31:18] = Reserved	0x1
	R/W	[17] = Disable scrambling of header	
	R/W	[16] = Disable scrambling of payload	
		[15:4] = Reserved	
	R/W	[3:0] = Upper 4 bits of CSF counter	
GFP_ERR		[31:8] = Reserved	0x2
	R/W	[7] = Payload scrambling error insertion	
	R/W	[6] = Location of superblock CRC error • (1) insert CRC error in all superblocks • (0) insert CRC error in first superblock	
	R/W	[5] = Superblock CRC error insertion	
	R/W	[4] = FCS error insertion	
	R/W	[3] = Core header scrambling error insertion	
	R/W	[2] = cHEC error insertion	
	R/W	[1] = tHEC error insertion	
	R/W	[0] = eHEC error insertion	

Table 3-6: **MAP Core General Registers *(Continued)***

Register Name	Type	Description	DCR Offset
GFP_INTMASK		[31:2] = Reserved	0x3
	R/W	[1] = Enable interrupt for invalid K characters received	
	R/W	[0] = Enable interrupt for PLI length mismatch	
GFP_INT[1]		[31:2] = Reserved	0x4
	R/W	[1] = Invalid K character received	
	R/W	[0] = PLI length mismatch	

1. All interrupts are cleared by performing a write operation to this register.

The MAP core also contains channel-specific registers, which enable the core to customize specific parameters on a per-channel basis (see Table 3-7). Note that the channel notation refers to channel-specific configurations, where the number of channels is specified in the GUI (x is 0 to number of channels -1). For detailed information about the MAP core configuration space, see "MAP Core Register Configuration: Per-Channel Configuration," page 50.

Table 3-7: **MAP Core Channel-Specific Registers**

Register Name	Type	Description	DCR Offset
CHANx_GFP_REGISTER_A		[31:28] = Reserved	0x0
	R/W	[27] = Core Configuration • Transparent mode(1) • Frame-mapped mode (0)	
	R/W	[26] = Enable FCS	
	R/W	[25] = Use UPI from REGISTER_A[23:16]	
	R/W	[24] = Use length from REGISTER_A[15:0]	
	R/W	[23:16] = Data UPI	
	R/W	[15:0] = Length	

Table 3-7: **MAP Core Channel-Specific Registers *(Continued)***

Register Name	Type	Description	DCR Offset
CHANx_GFP_REGISTER_B		[31:18] = Reserved	0x1
	R/W	[17] = Send loss of client signal management frame when CSF timer expires	
	R/W	[16] = Send loss of character synchronization management frame when CSF timer expires	
	R/W	[15:8] = Spare field	
	R/W	[7:0] = Channel id (CID) for this channel (alias)	

UNMAP Core Interfaces

Figure 3-3 displays the UNMAP core interface. All signals are defined in their respective sections below the illustration.

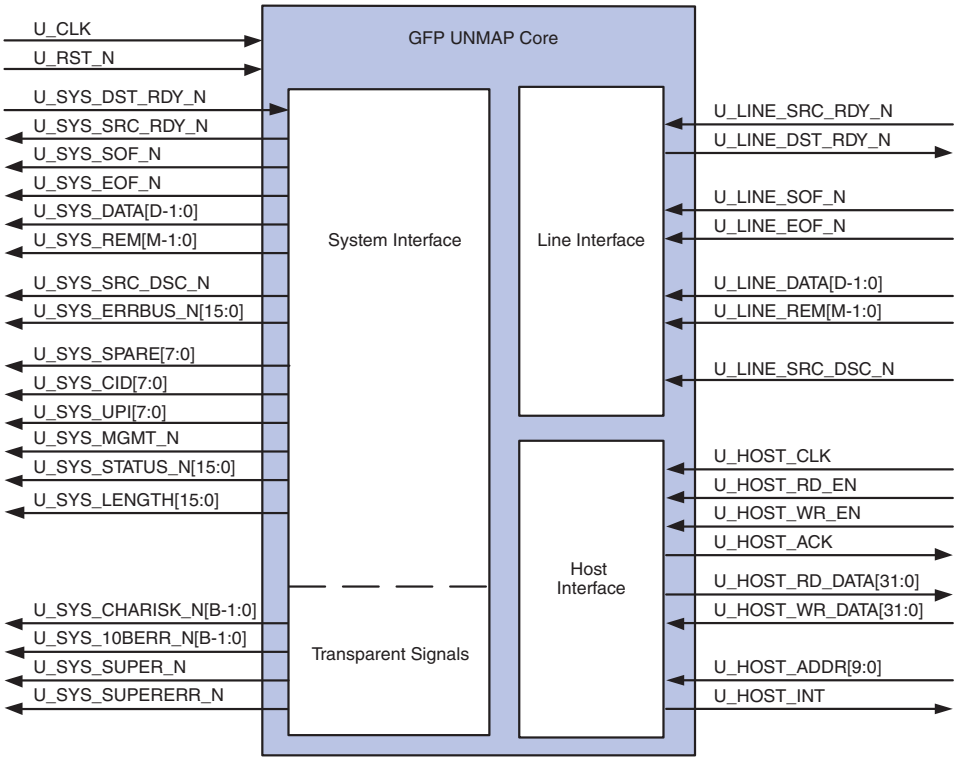

Figure 3-3: **UNMAP Core Interfaces**

Table 3-8 defines the relationship between the data bus width and additional signals. The UNMAP core supports both a 32-bit and a 64-bit interface. The bus widths of several signals, detailed below, depend on the interface width selected.

Table 3-8: **UNMAP Core Bus Widths**

Data Bus Width (D) (*_DATA)	Remainder Width (M) (*_REM)	Data Byte Width (B) (*_10BERR_N, *_CHARISK_N)
32	2	4
64	3	8

Common Interface

Table 3-9 describes the common interface. The UNMAP reset signal (U_RST_N) causes a hard reset of the entire core (core logic and host interface). This signal is an asynchronous input which is synchronized internally in the core before being used. The initial hardware reset should be generated by the user. Subsequent resets may be asserted by using the U_RST_N pin (for complete core resets), or by driving the appropriate register in the UNMAP host interface (for core logic reset only). For detailed information about reset requirements and operation, see "Initializing the GFP Core," page 64.

Table 3-9: **UNMAP Common Interface**

Name	Direction	Description
U_CLK	Input	**UNMAP Clock:** All system and line interface operations are synchronous to this clock.
U_RST_N	Input	**UNMAP Reset:** Asynchronous reset which resets both the UNMAP core logic and UNMAP host interface.

System Interface (U_SYS)

Table 3-10 describes the UNMAP system interface signals (U_SYS). The UNMAP system interface connects to the client side of the system and implements the Xilinx LocalLink system standard, providing a simple, flexible way to transmit frames. The system interface consists of a unidirectional data bus with control signals that provide the user application with real-time status about the current frame. The system interface signals are divided into three categories: signals common to both frame-mapped and transparent mode, signals specific to frame-mapped mode, and signals specific to transparent mode.

Table 3-10: **UNMAP System Interface**

Name	Direction	Description
Common to all modes		
U_SYS_DST_RDY_N	Input	**Read Enable** (Destination Ready)**:** Indicates a word is accepted this cycle (not accepted until U_SYS_SRC_RDY_N is also asserted).
U_SYS_SRC_RDY_N	Output	**Read Valid** (Source Ready)**:** Indicates a word presented by the system interface is valid (not read until U_SYS_DST_RDY_N is also asserted).
U_SYS_SOF_N	Output	**Start of Frame:** Indicates the beginning of a frame.
U_SYS_EOF_N	Output	**End of Frame:** Indicates the end of a frame.
U_SYS_DATA[D-1:0]	Output	**Data Bus:** Client network protocol data that has been extracted from the GFP frame. If Ethernet is being transferred (frame-mapped mode), the first word sent will be the destination address. In transparent mode, the data will be 8b/10b decoded data, which is either data or control/error (depending on the values of U_SYS_CHARISK_N and U_SYS_10BERR_N).
U_SYS_REM[M-1:0]	Output	**Data Remainder:** The number of valid bytes in U_SYS_DATA is U_SYS_REM + 1, and is MSB justified. This signal is only valid when U_SYS_EOF_N is asserted. Example: 32-bit data bus: REM = “00” => DATA[31:24] valid REM = “01” => DATA[31:16] valid REM = “10” => DATA[31:8] valid REM = “11” => DATA[31:0] valid
U_SYS_SRC_DSC_N	Output	**Frame Discontinue:** Indicates the current frame contains an error. This type of error will be asserted on U_SYS_ERRBUS_N.

Table 3-10: **UNMAP System Interface *(Continued)***

Name	Direction	Description
U_SYS_ERRBUS_N[15:0]	Output	**Error Condition:** Provides real-time errors and status for the current frame. The following errors are reported (bits 6, 2:0) if they are enabled in the host interface (UNMAP GFP_CTRL[3:0]). The remaining bits (7, 5:3) are always reported. [15:8] = reserved [7] = Transparent frame did not end on a superblock boundary [6] = FCS error [5] = cHEC corrected [4] = tHEC corrected [3] = eHEC corrected [2] = cHEC error [1] = tHEC error [0] = eHEC error
U_SYS_SPARE[7:0]	Output	**Spare Field:** Indicates the value of the spare field of a linear frame. This signal is present when linear extension headers are enabled.
U_SYS_CID[7:0]	Output	**Channel Identifier:** Indicates the channel ID for linear frames. This signal is present when linear extension headers are enabled.
U_SYS_UPI[7:0]	Output	**User Payload Identifier:** Indicates the payload type for the current data frame. When U_SYS_MGMT_N is asserted, this signal indicates the type of management frame received. See Appendix C, "Packet and Control Symbol Format" for UPI definitions.
U_SYS_MGMT_N	Output	**Management Frame Indicator:** Indicates a client management frame was received. When asserted, U_SYS_UPI indicates the type of client signal fail (CSF). See "Management Frames," page 68 for detailed use of this signal.

Table 3-10: **UNMAP System Interface *(Continued)***

Name	Direction	Description
U_SYS_STATUS_N[15:0]	Output	**Status Bus:** Provides real-time status and errors. [15:5] = reserved [4] = System FIFO almost full [3] = reserved Synchronization Status: Indicates the current status of the frame delineation state machine (one-cold encoding). See "Operating the UNMAP Line Interface with Streaming Data," page 84. [2] = SYNC [1] = PRESYNC [0] = HUNT
U_SYS_LENGTH[15:0]	Output	**Payload Length:** Indicates the length of the current frame in bytes.
Transparent signals		
U_SYS_CHARISK_N[B-1:0]	Output	**K Character Indicator:** Indicates that the corresponding byte of U_SYS_DATA is a 8b/10b K character, not a data character. This signal should be ignored for a given byte lane if U_SYS_REM indicates that the byte is not valid.
U_SYS_10BERR_N[B-1:0]	Output	**Insert 10B_ERR:** Indicates that the corresponding byte of U_SYS_DATA is an illegal 8b/10b word. This signal should be ignored for a given byte lane if U_SYS_REM indicates that the byte is not valid.
U_SYS_SUPER_N	Output	**Superblock Indication:** Asserted at the start of every new superblock for one clock cycle.
U_SYS_SUPERERR_N	Output	**Superblock CRC Error Indication:** Asserted at the end of a superblock which contains a CRC-16 error for one clock cycle.

Line Interface (U_LINE)

Table 3-11 describes the UNMAP line interface signals (U_LINE). The UNMAP line interface utilizes the Xilinx LocalLink standard, providing a simple, flexible way to receive frames. It consists of a unidirectional data bus with control signals.

Table 3-11: **UNMAP Line Interface**

Name	Direction	Description
U_LINE_SRC_RDY_N	Input	**Write Enable:** Indicates a word presented by the client is valid (not accepted until U_SYS_DST_RDY_N is also asserted).
U_LINE_DST_RDY_N	Output	**Write Accepted:** Indicates a word presented by the client will be accepted (if U_SYS_SRC_RDY_N is also asserted).
U_LINE_SOF_N	Input	**Start of Frame:** Indicates the beginning of a new GFP frame. This signal is only valid if No Hunting is selected for frame delineation. "Operating the UNMAP Line Interface with Streaming Data," page 84.
U_LINE_EOF_N	Input	**End of Frame:** Indicates the end of the current GFP frame. This signal is only valid if No Hunting is selected for frame delineation. See "Operating the UNMAP Line Interface with Streaming Data," page 84.
U_LINE_DATA[D-1:0]	Input	**Data bus:** Receives GFP encapsulated frames.
U_LINE_REM[M-1:0]	Input	**Data Remainder:** The number of valid bytes in U_LINE_DATA is U_LINE_REM + 1, and is MSB justified. This signal is only valid when U_LINE_EOF_N is asserted. See U_SYS_REM for valid byte mappings. This signal is only valid if No Hunting is selected for frame delineation. See "Operating the UNMAP Line Interface with Streaming Data," page 84.
U_LINE_SRC_DSC_N	Input	**Discontinue Frame:** Indicates the current GFP frame contains an error. This signal is only valid if No Hunting is selected for frame delineation. See "Operating the UNMAP Line Interface with Streaming Data," page 84.

Host Interface (U_HOST)

Table 3-12 describes the host interface signals (U_HOST). The UNMAP host interface to the user application utilizes the DCR bus for direct connection to the PowerPC, Microblaze, or other microprocessor. It consists of separate read and write data buses with a shared address bus. Note that the host interface is optional, and may be removed to conserve resources or if in-situ access to control registers is not required.

Table 3-12: **UNMAP Host Interface**

Name	Direction	Description
U_HOST_CLK	Input	**Host Clock:** All host interface signals are synchronous to this clock. Note that this signal is optional, as the host interface can be configured to be synchronous to U_CLK.
U_HOST_RD_EN	Input	**Read Enable:** Indicates a read access to the register addressed on U_HOST_ADDR.
U_HOST_WR_EN	Input	**Write Enable:** Indicates a write access to the register addressed on U_HOST_ADDR.
U_HOST_ACK	Output	**Acknowledge:** Both read and write requests to the host interface are acknowledged through this signal. Following a read request, this signal indicates that the data on U_HOST_RD_DATA is valid. Following a write request, this signal indicates that the data on U_HOST_WR_DATA was accepted.
U_HOST_RD_DATA[31:0]	Output	**Read Data:** Data read from the address U_HOST_ADDR, which is valid when a read request (U_HOST_RD_EN) followed by an acknowledge (U_HOST_ACK) is asserted.
U_HOST_WR_DATA[31:0]	Input	**Write Data:** Data to be written into the host interface at the address specified by U_HOST_ADDR. The write is acknowledged on U_HOST_ACK when the write succeeded.
U_HOST_ADDR[9:0]	Input	**Register Address:** Bus used to specify the address being accessed either for a read or write.
U_HOST_INT	Output	**Interrupt:** Indicates that an interrupt condition has been detected. This signal will be driven until the interrupt is cleared (by writing to GFP_INT).

UNMAP Configuration Space

The UNMAP core supports a host interface for access to control and status registers in-situ. The UNMAP core support only general registers. Table 3-13 describes the UNMAP core general registers.

Table 3-13: **UNMAP Core General Registers**

Register Name	Type	Description	DCR Offset
GFP_VERSION	R/W	[31] = Core reset	0x0
	R	[30:0] = Core version number	
GFP_CTRL		[31:28] = Reserved	0x1
	R/W	[27:24] = Number of cHEC matches required during the PRESYNC state to synchronize core (delta, minimum=1)	
		[23] = Reserved	
	R/W	[22] = Disable extension header error check (eHEC) correction	
	R/W	[21] = Disable type field error check (tHEC) correction	
	R/W	[20] = Disable core header error check (cHEC) correction	
	R/W	[19] = Disable descrambling of header	
	R/W	[18] = Disable descrambling of payload	
		[17:5] = Reserved	
	R/W	[4] = Disable reporting of superblock CRC error	
	R/W	[3] = Disable reporting of FCS error	
	R/W	[2] = Disable reporting of cHEC error	
	R/W	[1] = Disable reporting of tHEC error	
	R/W	[0] = Disable reporting of eHEC error	
GFP_FIXERR[1]		[31:27] = Reserved	0x2
	R/W	[26] = cHEC error corrected	
	R/W	[25] = tHEC error corrected	
	R/W	[24] = eHEC error corrected	
	R/W	[23:16] = CID of corrected cHEC frame	
	R/W	[15:8] = CID of corrected tHEC frame	
	R/W	[7:0] = CID of corrected eHEC frame	

Table 3-13: **UNMAP Core General Registers *(Continued)***

Register Name	Type	Description	DCR Offset
GFP_INTMASK		[31:5] = Reserved	0x3
	R/W	[4] = Enable interrupt for superblock CRC error	
	R/W	[3] = Enable interrupt for FCS error	
	R/W	[2] = Enable interrupt for cHEC error	
	R/W	[1] = Enable interrupt for tHEC error	
	R/W	[0] = Enable interrupt for eHEC error	
GFP_INT[2]		[31:5] = Reserved	0x4
	R/W	[4] = Superblock CRC error	
	R/W	[3] = FCS error	
	R/W	[2] = cHEC error	
	R/W	[1] = tHEC error	
	R/W	[0] = eHEC error	

1. This register holds the first value detected until cleared. All errors are cleared by performing a write operation to this register.
2. All interrupts are cleared by performing a write operation to this register.

Chapter 4

Generating the Core

The GFP core is a fully configurable implementation of the *ITU-T GFP Specification*. The core is highly customizable to achieve a resource-efficient implementation depending on the requirements of your system. The core is delivered through a GUI, which enables the specific features of the *ITU-T GFP Specification* to be optionally built-in logic gates, based on the system requirements.

CORE Generator Graphical User Interface

The GFP CORE Generator GUI consists of five screens:

- The first two screens let you generate specific hardware components (that is, use dedicated logic resources) to implement the *ITU-T GFP Specification.*
- The remaining three screens let you set default values for the core operation. If the host interface is present, you can also change these default values in-situ.

Main Screen

The main GFP screen defines the component name, operating mode, and system configuration options for the core.

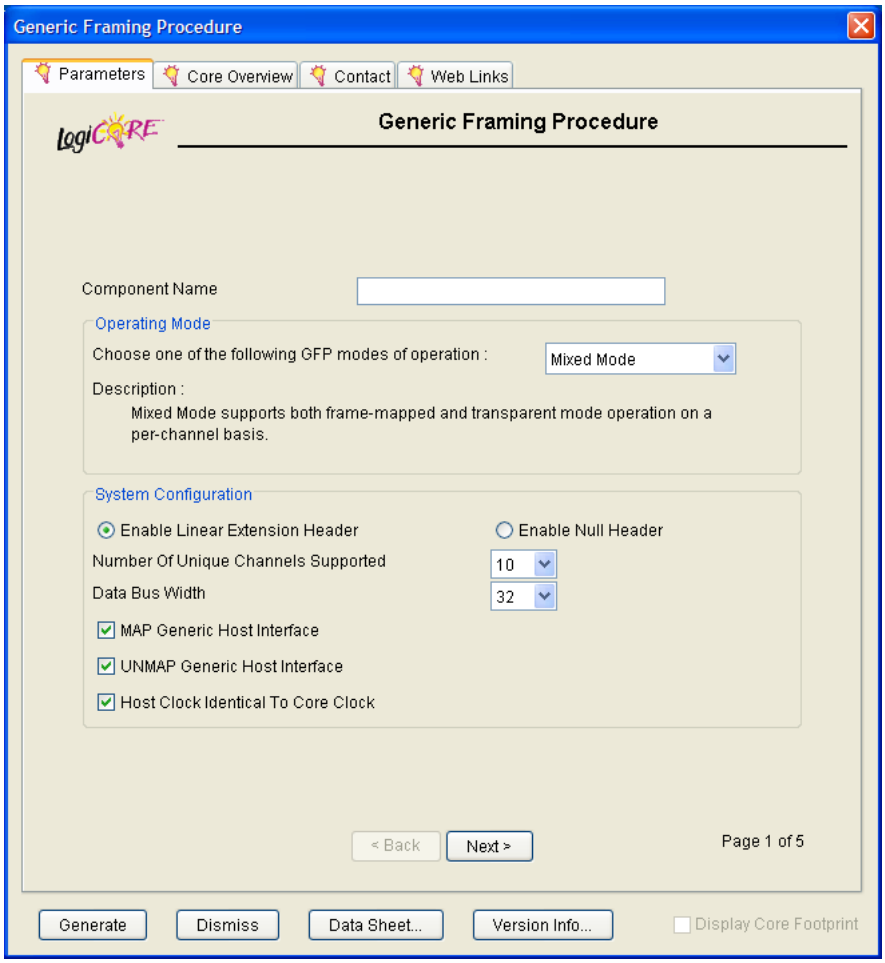

Figure 4-1: **Main GFP Screen**

Component Name

Base name of the output files generated for the core. The name must begin with a letter and be composed of the following characters: a to z, 0 to 9, and "_".

Operating Mode

The operating mode enables the user to configure the GFP core to support one of three configurations: frame-mapped mode, transparent mode, or mixed mode. This selection controls the types of data protocols that the GFP core will support.

Frame-Mapped Mode

Frame-mapped mode enables the GFP core to process frames for all frame-mapped protocols, including Ethernet, PPP, and RPR. Frame-mapped mode requires complete frames of data to be transmitted; see Chapter 6, "Designing with the Core" for details.

Transparent Mode

Transparent mode enables the GFP core to process frames for all transparent protocols, including Fibre Channel, Gigabit Ethernet, ESCON and DVB-ASI. Transparent mode does not require complete frames of data to be transmitted, but rather operates on a continuous stream of data. See Chapter 6, "Designing with the Core" for details.

Mixed Mode

Mixed mode enables the GFP core to process both frame-mapped and transparent mode frames, and supports all protocols defined in the *ITU-T GFP Specification*. Supporting mixed mode operation will require more resources than generating either a frame-mapped or transparent mode only core, but provides additional flexibility regarding the data protocols that can be processed.

System Configuration

The system configuration provides customization of the GFP core to be built, including number of channels, data bus width, and host interfaces.

Enable Linear Extension Header

The GFP core supports two types of extension headers (defined by the *ITU-T GFP Specification* as extension header identifier, EXI). Null extension header (null header) enables only a single channel of data to be transmitted. Extension header for a linear frame (linear frames) enables one to 256 channels of data to be transmitted by using a channel id (CID).

Number of Unique Channels Supported

When linear frames are enabled, the extension header is present, and embeds the CID into the GFP frame. Select the number of channels from the drop-down menu to enable customization on a per-channel basis for the MAP core. Up to 10 channels can be configured for different protocols, and channel-specific register settings (see "MAP Core Register Configuration: Per-Channel Configuration," page 50). Additional channels beyond 10 (up to 256) are supported and use the same configuration as channel 0. When linear frames are not enabled, the number of channels is set to one.

Data Bus Width

The data bus width supported by the GFP core includes a 32-bit data path for up to OC-48 (2.5 Gbps) operation, and a 64-bit data path for up to OC-192 (10 Gbps) operation.

MAP/UNMAP Generic Host Interface

The MAP and UNMAP cores each independently support an optional host interface. When the host interface is present, the M_HOST_* or U_HOST_* signals are present, and the register space can be accessed for core configuration and error handling. If the host interface default values do not need to be changed in circuit, the host interface can be removed to save logic resources.

Host Clock Identical to Core Clock

When selected, this option connects the host clock to the core clock. When this option is not selected, then the host clock is present, and is asynchronous to the core clock.

Customizing Core Features

The Core Customization screen includes optional core features that perform specific operations—when selected, these result in the generation of dedicated logic that will be implemented in the FPGA.

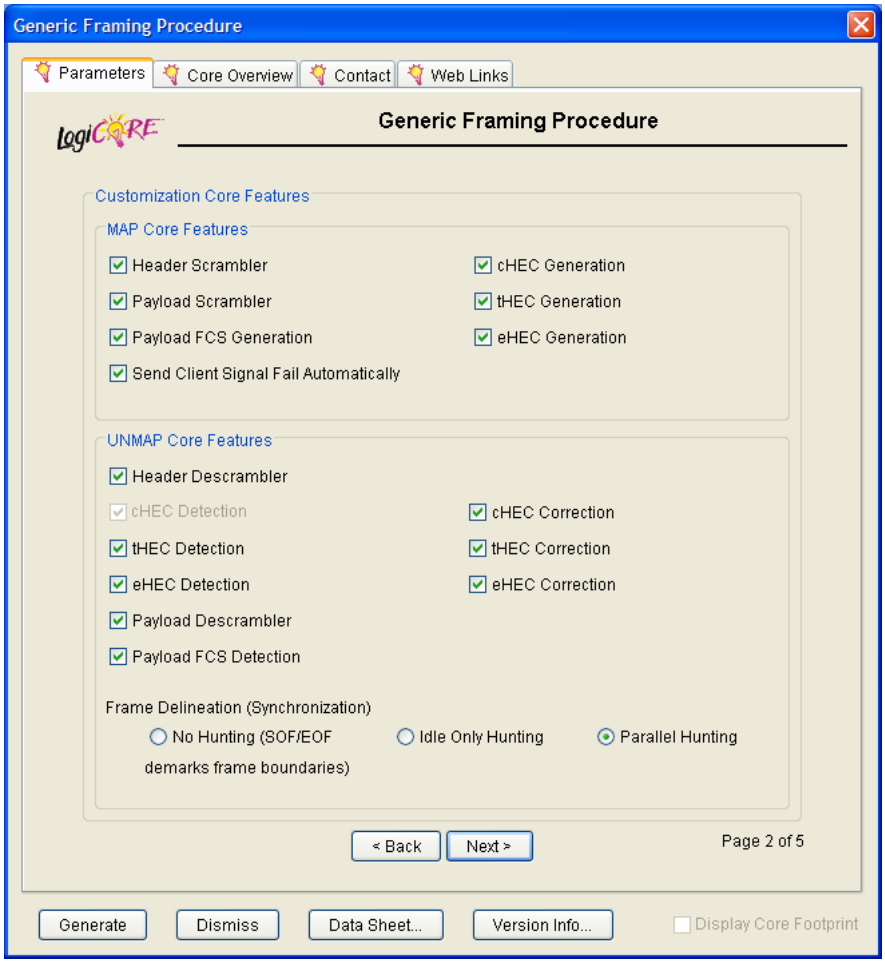

Figure 4-2: **MAP/UNMAP Customization Features**

MAP Core Features

The following options build hardware to perform the described operations:

- Header Scrambler: Scrambles the core header with the Barker-like sequence as described in the *ITU-T GFP Specification*.
- Payload Scrambler: Scrambles the GFP payload as described in the *ITU-T GFP Specification*.
- Payload FCS Generation: Generates the FCS over the payload area as described in the *ITU-T GFP Specification*.
- Send Client Signal Fail Automatically: Enables the core to generate CSF management frames automatically. The period in which the management frames are sent is controlled by the CSF timer in the MAP GFP_CTRL[3:0] register. See "MAP Core," page 65 for details.

- cHEC, tHEC, eHEC Generation: Generates the header error check for the core header (cHEC), type header (tHEC), and the extension header (eHEC - only available if linear extension header is enabled). Uses the HEC generating polynomial as described in the *ITU-T GFP Specification.*

UNMAP Core Features

The following options build hardware to perform the described operations when selected.

- Header Descrambler: Descrambles the core header with the Barker-like sequence as described in the *ITU-T GFP Specification.*
- cHEC, tHEC, eHEC Detection: Performs the header error check for the core header (cHEC), type header (tHEC), and the extension header (eHEC - only available if linear extension header is enabled). Uses the HEC generating polynomial as described in the *ITU-T GFP Specification.*
- cHEC, tHEC, eHEC Correction: Performs the header error check correction when single-bit errors are detected for the core header (cHEC), type header (tHEC), and the extension header (eHEC - only available if linear extension header is enabled). Uses the HEC generating polynomial as described in the *ITU-T GFP Specification.*
- Payload Descrambler: Descrambles the GFP payload as described in the *ITU-T GFP Specification*.
- Payload FCS Detection: Verifies the FCS over the payload area as described in the *ITU-T GFP Specification.*
- Frame Delineation (Synchronization): Determines the method the UNMAP core uses to synchronize with the far end GFP mapper.
  - No Hunting uses SOF_N and EOF_N to indicate frame boundaries.
  - Idle Only Hunting requires streaming data of idle frames to synchronize.
  - Parallel Hunting requires streaming data of any arbitrary data or idle packets. See "Operating the UNMAP Line Interface with Streaming Data," page 84 for details about frame delineation behavior.

MAP Core Register Configuration: General Registers

The GFP core provides a separate MAP host interface to configure the operation and error reporting of the core. When the host interface is present, initial default values are set by the GUI and can be changed in-situ. If the host interface default values do not need to be changed following core generation, the host interface can be removed to save logic resources.

The following sections detail the procedure for utilizing the CORE Generator GUI to provide default values to the host interface. Each option in the CORE Generator GUI relates to a specific register entry, as described below. For details about the register space operation, see "MAP General Registers," page 55.

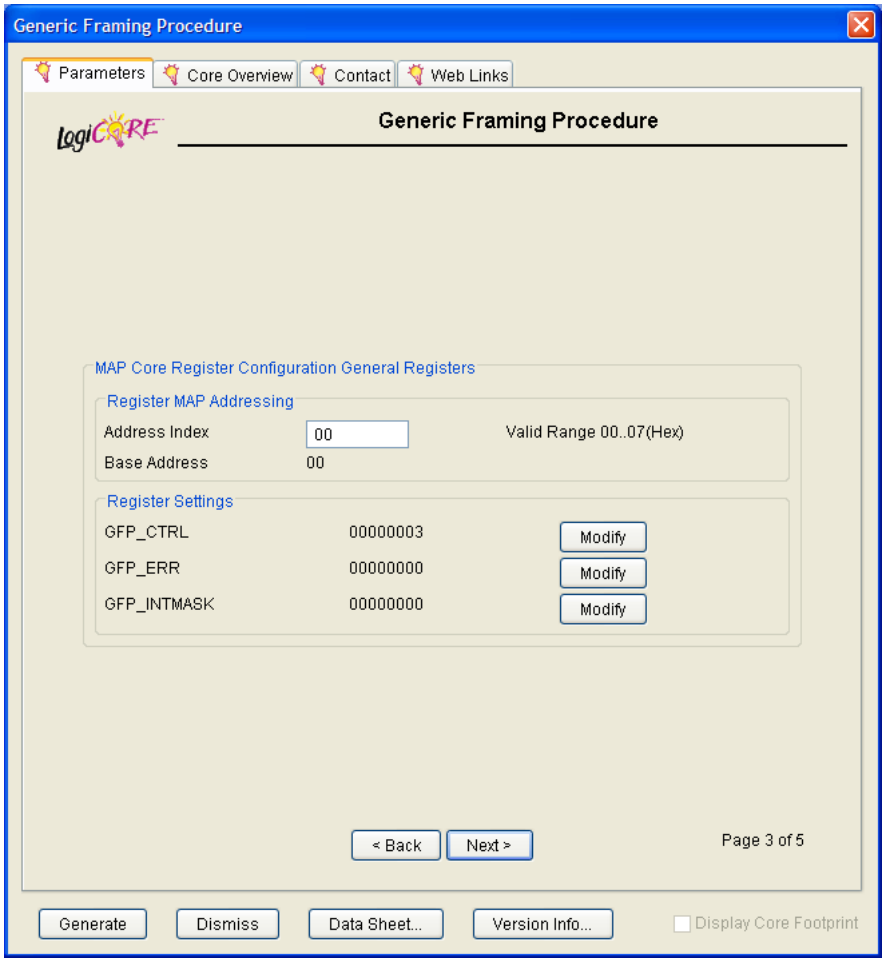

Figure 4-3: **MAP General Registers Configuration**

Register MAP Addressing

The MAP address index allows the user to partition the DCR register space, such that the MAP core can share the DCR bus with other DCR devices. The valid range of the address index is dependent upon the number of channels selected. See Figure 4-4 for the address index range relative to the entire address space (base address). The base address is composed of an address index and the number of channels (as set in the GUI) and an offset. The offset allows the user access to the specific registers. For a detailed description of the register space, see "Accessing Control and Status Registers," page 86. The MAP address index is only applicable if the MAP host interface is present.

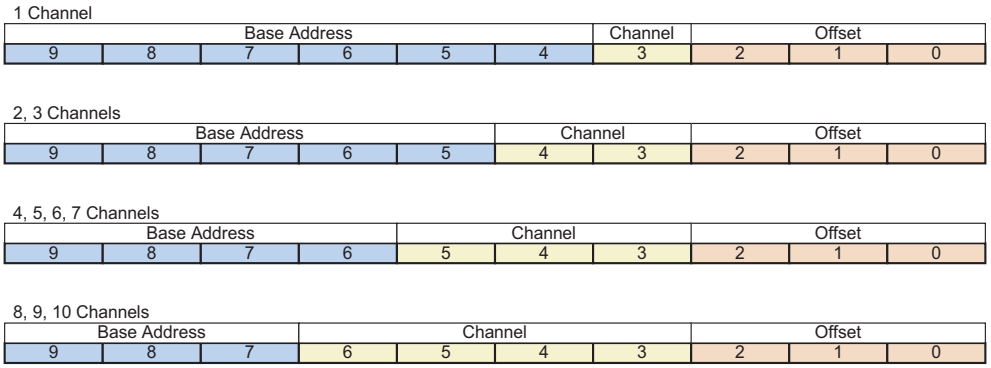

Figure 4-4: **Map Core Base Address**

MAP Registers

The following MAP registers can be configured by the user. Many of these registers require that the hardware be present. For example, header scrambling must be present in hardware to disable it in the configuration space. If the hardware is not present for a given register, the register is inactive. All registers are initially customized by clicking Modify in the GUI to set default values for the respective register option.

- GFP_CTRL: Control register that controls operations and behaviors of the MAP core (Table 4-1).

Table 4-1: **MAP GUI Customization: GFP_CTRL Register**

CORE Generator GUI Option	Register Space
Disable Header Scrambler	MAP GFP_CTRL[17]
Disable Payload Scrambling	MAP GFP_CTRL[16]
CSF Timer	MAP GFP_CTRL[3:0]

- GFP_ERR: Error register option that enables specific errors to be inserted by the MAP core (Table 4-2). The enabled errors are inserted each time the M_SYS_FORCE_ERR_N signal is asserted, as described in "MAP Core," page 65.

Table 4-2: **MAP GUI Customization: GFP_ERR Register**

CORE Generator GUI Option	Register Space
Payload Scrambling Error Insertion	MAP GFP_ERR[7]
First CRC in Superblock/All CRCs in Superblock	MAP GFP_ERR[6]
Enable Superblock CRC Error Insertion	MAP GFP_ERR[5]
Payload FCS Error Insertion	MAP GFP_ERR[4]
Core Header Scrambling Error Insertion	MAP GFP_ERR[3]
cHEC Error Insertion	MAP GFP_ERR[2]

Table 4-2: **MAP GUI Customization: GFP_ERR Register**

CORE Generator GUI Option	Register Space
tHEC Error Insertion	MAP GFP_ERR[1]
eHEC Error Insertion	MAP GFP_ERR[0]

- GFP_INTMASK: Interrupt register mask option, which controls the conditions in which the interrupt signal is asserted on the host interface (Table 4-3).

Table 4-3: **MAP GUI Customization: GFP_INTMASK Register**

CORE Generator GUI Option	Register Space
Enable interrupt for invalid K characters received	MAP GFP_INTMASK[1]
Enable interrupt for PLI length mismatch	MAP GFP_INTMASK[0]

MAP Core Register Configuration: Per-Channel Configuration

The MAP host interface provides configuration options for each channel in the MAP core. When linear extension header is selected, the user can select from one to ten unique channels, and each channel can be customized for independent operation. Note that up to 256 channels are supported, and channels beyond 10 use the same configuration as channel 0. When linear extension header is not enabled, the number of channels is automatically set to 1. See "Enable Linear Extension Header," page 45 for details.

When the host interface is present, initial default values of the per-channel registers are set in the GUI and can be changed in-situ. If these default values do not need to be changed following core generation, the host interface can be removed to save logic resources. The following details the procedure for utilizing the CORE Generator GUI to specify default values. Each option in the CORE Generator GUI relates to a specific register. For detailed

information about the operation of the register space, see "MAP General Registers," page 55.

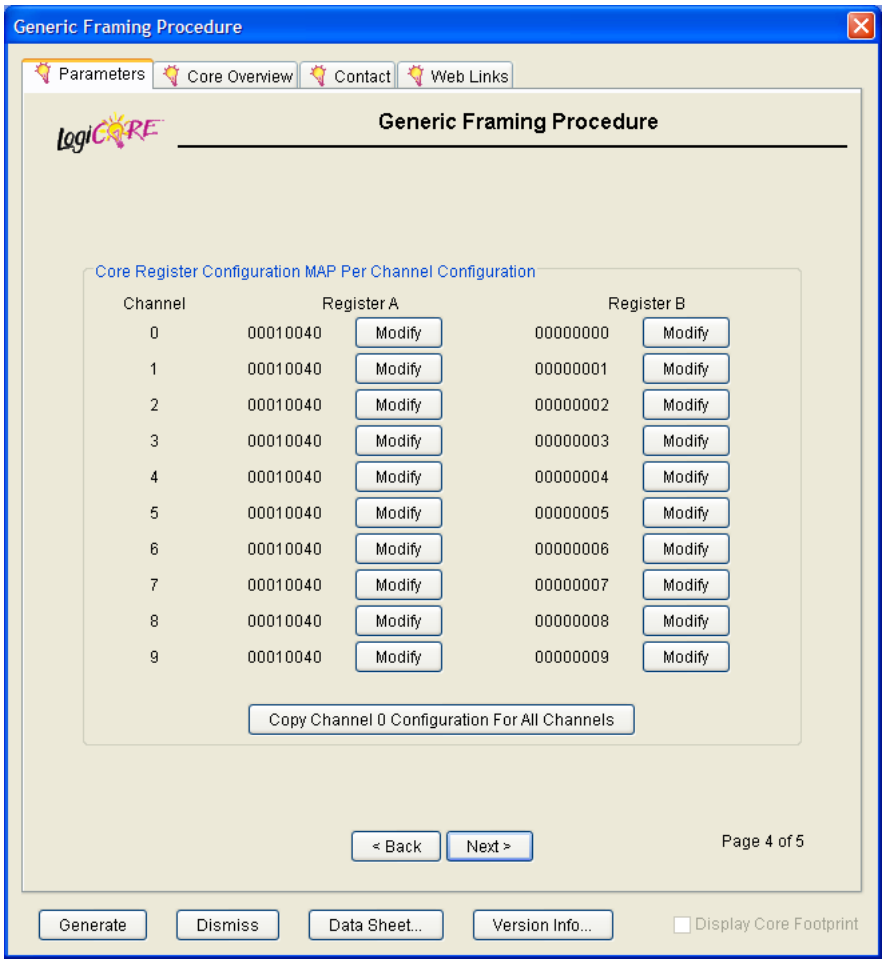

Figure 4-5: **Map Per Channel Configuration**

CHANx_GFP_REGISTER_A

- The MAP CHANx_GFP_REGISTER_A (where x represents channels 0 through 9) Controls the operation (frame-mapped or transparent) of CHANx, insertion of FCS, and setting the UPI and length values (Table 4-4).

Table 4-4: **MAP GUI Customization: CHANx_GFP_REGISTER_A**

CORE Generator GUI Option	Register Space
Frame-Mapped / Transparent	CHANx_GFP_REGISTER_A[27]
Insert FCS	CHANx_GFP_REGISTER_A[26]
Use UPI from REGISTER_A	CHANx_GFP_REGISTER_A[25]
Use Length from REGISTER_A	CHANx_GFP_REGISTER_A[24]
UPI Value	CHANx_GFP_REGISTER_A[23:16]
Length	CHANx_GFP_REGISTER_A[15:0]

CHANx_GFP_REGISTER_B

- The MAP CHANx_GFP_REGISTER_B configuration space register that controls the automatic insertion of CSF management frames, as well as defining the spare and alias fields for a given channel (Table 4-5).

Table 4-5: **MAP GUI Customization: CHANx_GFP_REGISTER_B**

CORE Generator GUI Option	Register Space
Send Loss of Client Signal when CSF Timer expires	CHANx_GFP_REGISTER_B[17]
Send Loss of Character Synchronization when CSF Timer expires	CHANx_GFP_REGISTER_B[16]
Spare Field	CHANx_GFP_REGISTER_B[15:8]
Alias Field	CHANx_GFP_REGISTER_B[7:0]

UNMAP Core Register Configuration: General Registers

The GFP core provides a separate UNMAP host interface to configure the operation and error reporting of the core. When the host interface is present, initial default values are set by the GUI and can be changed in-situ. If the host interface default values do not need to be changed following core generation, the host interface can be removed to save logic resources. The following sections detail the procedure for utilizing the CORE Generator GUI to provide default values to the host interface. Each option in the CORE Generator GUI relates to a specific register entry, as described below. For details about the operation of the register space, see "UNMAP General Registers," page 59.

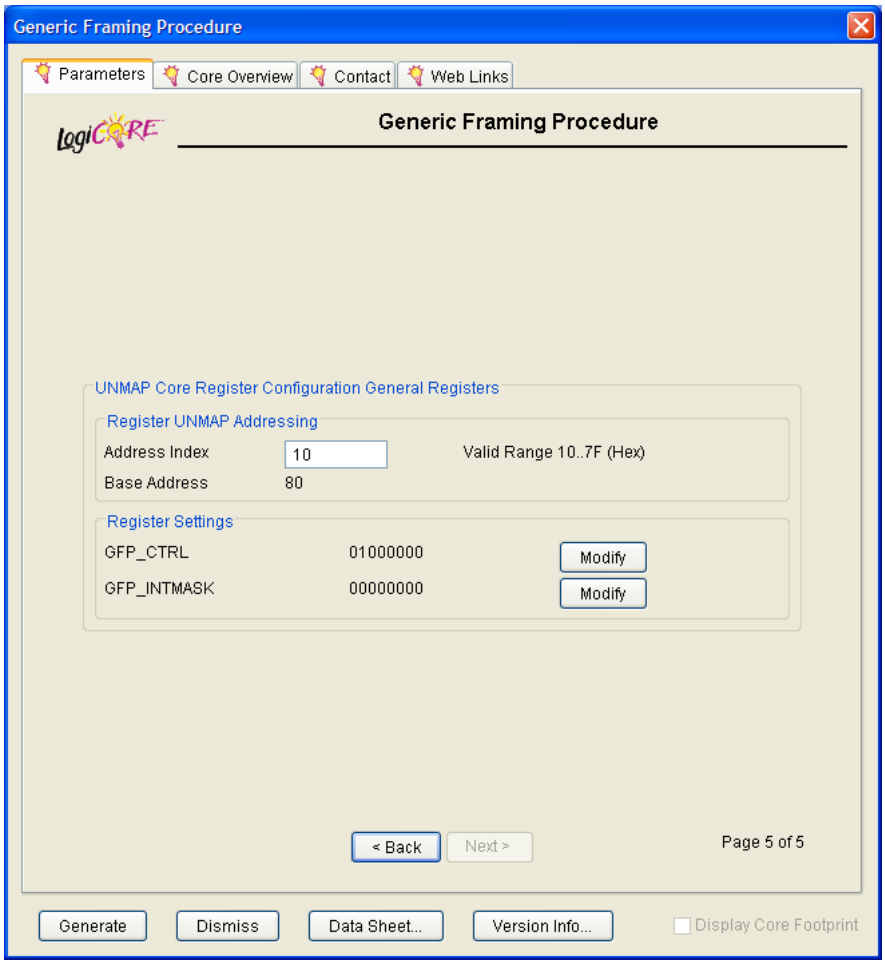

Figure 4-6: **UNMAP General Registers Configuration**

UNMAP Address Index

The UNMAP address index allows the user to partition the DCR register space, such that the UNMAP core can share the DCR bus with other DCR devices. Figure 4-7 shows the valid range of the address index relative to the entire address space (base address). The base address is composed of an address index and an offset. The offset allows the user access to the specific registers.

For a detailed description of the register space, see "Accessing Control and Status Registers," page 86. The UNMAP address index is only applicable if the UNMAP host interface is present.

Base Address							Offset		
9	8	7	6	5	4	3	2	1	0

Figure 4-7: **UNMAP Core Base Address**

UNMAP Registers

The following UNMAP core registers can be configured by the user. Many of these registers require that the hardware be present. For example, header descrambling must be

present in hardware in order to disable it in the configuration space. If the hardware is not present for a given register, the register is inactive.

- GFP_CTRL: Control register that controls operations and behaviors of the UNMAP core (Table 4-6).

Table 4-6: **UNMAP GUI Customization: GFP_CTRL Register**

CORE Generator GUI Option	Register Space
Number of cHEC matches to synchronize core	UNMAP GFP_CTRL[27:24]
Disable eHEC Correction	UNMAP GFP_CTRL[22]
Disable tHEC Correction	UNMAP GFP_CTRL[21]
Disable cHEC Correction	UNMAP GFP_CTRL[20]
Disable Header Descrambling	UNMAP GFP_CTRL[19]
Disable Payload Descrambling	UNMAP GFP_CTRL[18]
Ignore Superblock CRC Error	UNMAP GFP_CTRL[4]
Ignore FCS Error	UNMAP GFP_CTRL[3]
Ignore cHEC Error	UNMAP GFP_CTRL[2]
Ignore tHEC Error	UNMAP GFP_CTRL[1]
Ignore eHEC Error	UNMAP GFP_CTRL[0]

- GFP_INTMASK: Interrupt register mask that controls the conditions in which the interrupt signal is asserted on the host interface (Table 4-7).

Table 4-7: **UNMAP GUI Customization: GFP_INTMASK Register**

CORE Generator GUI Option	Register Space
Enable interrupt for superblock CRC error	UNMAP GFP_INTMASK[4]
Enable interrupt for FCS error	UNMAP GFP_INTMASK[3]
Enable interrupt for cHEC error	UNMAP GFP_INTMASK[2]
Enable interrupt for tHEC error	UNMAP GFP_INTMASK[1]
Enable interrupt for eHEC error	UNMAP GFP_INTMASK[0]

Chapter 5

Customizing the Core

This chapter describes how to customize the GFP core for specific applications. After specifying the initial values of the register space in the CORE Generator GUI to define the default operation of the core, the user can change these values and the corresponding operation of the core in circuit using the host interface.

Customizing the GFP Core

The Xilinx CORE Generator GUI sets the default operation of the GFP core, as described in Chapter 4, "Generating the Core." After generating the core, the user can change the register settings to control the behavior of the core if the host interface is present. This section details how the host interface can be used to access and use the control and status registers to modify the behavior of the core in-situ.

MAP General Registers

The MAP general registers apply to all channels of the MAP core, and control the overall operation of the core (Table 5-1).

Table 5-1: **MAP Core General Registers**

Register Name	Description	Details
GFP_VERSION	[31] = Core reset	When set to 1, resets the MAP core logic, but does not reset the host interface. When reconfiguring the core, the user should write a 1 to this register, perform all required register accesses, and then write a 0 to this register. For details about initializing the core, see "Initializing the GFP Core," page 64.
	[30:0] = Core version number	Provides the current version of the GFP MAP core.

Table 5-1: **MAP Core General Registers *(Continued)***

Register Name	Description	Details
GFP_CTRL	[17] = Disable scrambling of header	When set to 1, bypasses the header scrambling logic when header scrambling is built in hardware.
	[16] = Disable scrambling of payload	When set to 1, bypasses the payload scrambling logic when payload scrambling is built in hardware.
	[3:0] = Upper 4 bits of CSF counter	Determines the period in which the core will automatically send management frames (if enabled in hardware). When set to 0x0, this feature is disabled. See "Management Frames," page 68 for details.
GFP_ERR	[7] = Payload scrambling error insertion	When set to 1, inserts an error during payload scrambling when payload scrambling is built in hardware.
	[6] = Location of superblock CRC error	When in transparent mode, enables insertion of CRC errors in the superblock. When set to 1, errors are inserted on all superblocks within the given frame. When set to 0, an error is inserted only on the first superblock in the frame. Bit 5 of this register enables the insertion of the error in the specified location.
	[5] = Superblock CRC error insertion	When in transparent mode, allows an error to be inserted in the superblock CRC (the location of inserted errors in the frame set by bit 6).
	[4] = FCS error insertion	When set to 1, inserts an error into the computed FCS when FCS is built in hardware.
	[3] = Core header scrambling error insertion	When set to 1, inserts an error into the core header scrambling when core header scrambling is built in hardware.
	[2] = cHEC error insertion	When set to 1, inserts an error into the cHEC when cHEC generation is built in hardware.
	[1] = tHEC error insertion	When set to 1, inserts an error into the tHEC when tHEC generation is built in hardware.
	[0] = eHEC error insertion	When set to 1, inserts an error into the eHEC when eHEC generation is built in hardware.

Table 5-1: **MAP Core General Registers *(Continued)***

Register Name	Description	Details
GFP_INTMASK	[1] = Enable interrupt for invalid K characters received	When in transparent mode, and set to 1, enables an interrupt to be asserted when an invalid K character is written into the system interface.
	[0] = Enable interrupt for PLI length mismatch	When in frame-mapped mode, and set to 1, enables an interrupt to be asserted when the pre-defined length of the current frame does not match the actual length of the frame written into the system interface.
GFP_INT[1]	[1] = Invalid K character received	When in transparent mode, is asserted when an invalid K character is written into the system interface, and the mask bit is set.
	[0] = PLI length mismatch	When in frame-mapped mode, is asserted when the predefined length of the current frame does not match the actual length of the frame written into the system interface, and the mask bit is set. For details on setting the length field, see "Operating the MAP System Interface with Frame-Mapped Frames," page 65.

1. Reports that an interrupt occurred. The interrupt is held with the first value detected until it is cleared by performing any write operation to the register.

MAP Channel Specific Registers

The MAP channel-specific registers provide customization of each channel as described below in "MAP Channel Specific Registers." (Table 5-2)

Table 5-2: **MAP Core Channel-Specific Registers**

Register Name	Description	Details
CHANx_GFP_ REGISTER_A	[27] = Core encapsulation • (1) Transparent mode • (0) Frame-mapped mode	When the MAP core supports transparent mode, and this register is set to 1, this register configures the core to perform transparent mode encapsulation. When the MAP core supports frame-mapped mode, and this register is set to 0, this register configures the core to perform frame-mapped mode encapsulation.
	[26] = Enable FCS	When set to 1, enables FCS insertion for the given channel when FCS is built in hardware.
	[25] = Use UPI from REGISTER_A[23:16]	Frame-Mapped Mode: When set to 1, the UPI value from REGISTER_A[23:16] is used. When set to 0, the UPI value from M_SYS_UPI is used. Transparent Mode: Not used. The UPI for all data frames is used from REGISTER_A[23:16]. See "Operating the MAP System Interface with Transparent Frames" for details.
	[24] = Use length from REGISTER_A[15:0]	Frame-Mapped Mode: When this register is set to 1, the length value from REGISTER_A[15:0] is used. When set to 0, the length value from M_SYS_LENGTH is used. Transparent Mode: Not used. The length for all data frames is used from REGISTER_A[15:0].
	[23:16] = Data UPI	Sets the data UPI to be transmitted. See Appendix C, "Packet and Control Symbol Format" for a complete list of UPI values; see REGISTER_A[25] for details about this register.

Table 5-2: **MAP Core Channel-Specific Registers *(Continued)***

Register Name	Description	Details
CHANx_GFP_ REGISTER_A (cont.)	[15:0] = Length	Frame-Mapped Mode: Sets the length of the frame to be transmitted (in bytes). See REGISTER_A[24] for details on when this register is used. Transparent Mode: Sets the length of the frame to be transmitted (in superblocks). This register is always used in transparent mode.
CHANx_GFP_ REGISTER_B	[17] = Insert loss of client signal	When set to 1, enables the MAP core to insert loss of client signal fail when the CSF timer expires, when send CSF automatically is built in hardware.
	[16] = Insert loss of character synchronization	When set to 1, enables the MAP core to insert loss of character synchronization when the CSF timer expires, when send CSF is automatically is built in hardware.
	[15:8] = Spare field	When linear frame support is built in hardware, this register contains the spare field to be inserted into the linear extension header.
	[7:0] = Channel id (CID) for this channel (alias)	When linear frame support is built in hardware, this register contains the CID field to be inserted into the linear extension header.

UNMAP General Registers

The UNMAP general registers control the overall operation of the core (Table 5-3).

Table 5-3: **UNMAP Core General Registers**

Register Name	Description	Details
GFP_VERSION	[31] = Core reset	When set to 1, resets the UNMAP core logic, but does not reset the host interface. When reconfiguring the core, the user should write a 1 to this register, perform all required register accesses, and then write a 0 to this register. For details on initializing the core, see "Initializing the GFP Core," page 64.
	[30:0] = Core version number	Provides the current version of the GFP UNMAP core.

Table 5-3: **UNMAP Core General Registers *(Continued)***

Register Name	Description	Details
GFP_CTRL	[27:24] = Number of cHEC matches during PRESYNC to synchronize core	Indicates the number of cHEC matches (minimum value is 1) required before the core will synchronize. This register is only used if the synchronization mode is set to “Idle Only Hunting” or “Parallel Hunting”. The synchronization status is indicated on U_SYS_STATUS_N[2:0].
	[22] = Disable extension header error check (eHEC) correction	When set to 1, turns off eHEC correction when eHEC correction is built in hardware.
	[21] = Disable type field error check (tHEC) correction	When set to 1, turns off tHEC correction when tHEC correction is built in hardware.
	[20] = Disable core header error check (cHEC) correction	When set to 1, turns off cHEC correction when cHEC correction is built in hardware.
	[19] = Disable descrambling of header	When set to 1, turns off header descrambling when header descrambling is built in hardware.
	[18] = Disable descrambling of payload	When set to 1, turns off payload descrambling when payload descrambling is built in hardware.
	[4] = Disable reporting of superblock CRC error	When set to 1, turns off reporting of superblock CRC errors (on U_SYS_SUPERERR_N).
	[3] = Disable reporting of FCS error	When set to 1, turns off reporting of FCS error (on U_SYS_ERRBUS[6]).
	[2] = Disable reporting of cHEC error	When set to 1, turns off reporting of cHEC error (on U_SYS_ERRBUS[2]).
	[1] = Disable reporting of tHEC error	When set to 1, turns off reporting of tHEC error (on U_SYS_ERRBUS[1]).
	[0] = Disable reporting of eHEC error	When set to 1, turns off reporting of eHEC error (on U_SYS_ERRBUS[0]).

Table 5-3: **UNMAP Core General Registers *(Continued)***

Register Name	Description	Details
GFP_FIXERR[1]	[26] = cHEC error corrected	When cHEC correction is built in hardware, this register indicates that the UNMAP core corrected a cHEC error. The channel number (if linear frames are enabled) is indicated on bits 23:16.
	[25] = tHEC error corrected	When tHEC correction is built in hardware, this register indicates that the UNMAP core corrected a tHEC error. The channel number (if linear frames are enabled) is indicated on bits 15:8.
	[24] = eHEC error corrected	When eHEC correction is built in hardware, this register indicates that the UNMAP core corrected a eHEC error. The channel number (if linear frames are enabled) is indicated on bits 7:0.
	[23:16] = CID of corrected cHEC frame	When linear frames are enabled, this indicates the channel number of the corrected cHEC frame.
	[15:8] = CID of corrected tHEC frame	When linear frames are enabled, this indicates the channel number of the corrected tHEC frame.
	[7:0] = CID of corrected eHEC frame	When linear frames are enabled, this indicates the channel number of the corrected eHEC frame.
GFP_INTMASK	[4] = Enable interrupt for Superblock CRC error	When set to 1, this register enables an interrupt to be asserted when a superblock CRC error is detected.
	[3] = Enable interrupt for FCS error	When set to 1, this register enables an interrupt to be asserted when an FCS error is detected.
	[2] = Enable interrupt for cHEC error	When set to 1, this register enables an interrupt to be asserted when a cHEC error is detected.
	[1] = Enable interrupt for tHEC error	When set to 1, this register enables an interrupt to be asserted when a tHEC error is detected.
	[0] = Enable interrupt for eHEC error	When set to 1, this register enables an interrupt to be asserted when an eHEC error is detected.

Table 5-3: **UNMAP Core General Registers *(Continued)***

Register Name	Description	Details
GFP_INT[2]	[4] = Superblock CRC	When in transparent mode, this register indicates that a superblock CRC error was detected.
	[3] = FCS	When FCS is built in hardware, this register indicates that an FCS error was detected.
	[2] = cHEC	When cHEC is built in hardware, this register indicates that a cHEC error was detected.
	[1] = tHEC	When tHEC is built in hardware, this register indicates that a tHEC error was detected.
	[0] = eHEC	When eHEC is built in hardware, this register indicates that an eHEC error was detected.

1. Reports that an error was corrected in one of the following fields. The error is held with the first value detected until it is cleared by performing any write operation to this register.
2. Reports that an interrupt occurred. The interrupt is held with the first value detected until it is cleared by performing any write operation to the register.

Designing with the Core

This chapter discusses how to use the Xilinx GFP core in a user application, including frame-mapped, transparent, and mixed mode operation and accessing control and status registers for the MAP and UNMAP cores.

General Design Guidelines

This section describes the steps required to turn a GFP core into a fully-functioning design integrated with user application logic. It is important to recognize that not all designs will require all steps listed in this chapter. The following sections discuss the design steps required for each feature's implementation. It is recommended that the design guidelines in this guide be carefully followed.

Know the Degree of Difficulty

A fully compliant GFP core is challenging to implement in any technology. In addition, the degree of difficulty is significantly influenced by the following:

- Maximum system clock frequency
- Targeted device architecture
- Specific user application

All implementations require careful attention to system performance requirements. Pipelining, placement constraints, and logic duplication are all methods the user can use to improve system performance.

Understand Signal Pipelining

Due to the nature of packet and frame based protocols, it is important to understand that the GFP MAP and UNMAP cores have been pipelined to maximize performance. Data that is delivered to the MAP core takes several clock cycles before the completed frame appears on the line interface due to the pipelining required to insert the header fields, encode transparent superblocks, and create FCS and HEC fields.

Similarly, frames delivered to the UNMAP core take several clock cycles before the data content appears on the system interface due to the following: the pipelining required to convert the streaming input bus to an aligned output with header fields parsed out; transparent superblocks decoded; and FCS and HEC fields checked. The exact latency of the MAP and UNMAP cores varies based on the configuration of the core, and can be accurately determined through simulation.

Keep it Registered

The best method to simplify timing and increase system performance in an FPGA design is to keep everything registered, meaning 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 helps the user achieve timing closure.

Use Supported Design Flows

The GFP core has been tested with a variety of design flows. While other design tools can be used to simulate and synthesize the user design with the core, they have not been tested and functionality can not be guaranteed. See "Simulating and Implementing Your Design," page 97 for information about supported design tools.

Make Only Allowed Modifications

All modifications to the GFP core must be done via the Xilinx CORE Generator. Do not make other modifications as they may have adverse effects on system timing and GFP protocol compliance.

Initializing the GFP Core

The GFP cores (both MAP and UNMAP) must be reset after the FPGA is configured or before operation begins. A reset pin is provided for the MAP core (`M_RST_N`) and the UNMAP core (`U_RST_N`). This external reset is synchronized internal to the cores and will reset all core logic and the register space to the default state.

A reset through the host interface (using the register space, which will be referred to as a software reset) is also provided, which will cause all core logic other than the host interface and register space to reset. This reset is turned on by writing a 1 to location `GFP_VERSION[31]` in either core, and it is turned off by writing a 0 to the same location.

For most settings, it is a requirement that a software reset be issued and held to the core if a register map setting is to be modified; failure to do so could result in incorrect operation. Exceptions include writing the interrupt `GFP_INT` or changing interrupt mask `GFP_INTMASK`, changing the timer timeout value in `GFP_CTRL[3:0]`, changing the error insertion settings in `GFP_ERR`, or clearing the UNMAP register `GFP_FIXERR`. After the registers have been updated, the software reset may be released and operation can resume. A read to any register can be performed while the core is operating. Note that if the reset pin is asserted (`M_RST_N` or `U_RST_N`), the register space, in addition to the core logic, will be reset to its default values defined in the CORE Generator GUI.

When the MAP core is in reset (either reset pin or software reset), the line interface will output a continuous stream of GFP idle frames. `M_LINE_SRC_RDY_N` will be asserted indicating valid data, and `M_LINE_IDLE_N` will be asserted indicating that the data on the line interface is an idle frame. If the user application requires continuous data to be transmitted (even during reset), then the user can pass the idle frames onto the SONET/SDH network. If this is not a requirement of the system, then the user can discard these frames based on the idle indication `M_LINE_IDLE_N`. When the UNMAP core is in reset, the system interface will indicate that no data is available by deasserting `U_SYS_SRC_RDY_N` until the core is not in reset, and valid data has been received. The MAP system interface and UNMAP line interface will also report to the user that they are not ready to receive data until reset is released.

MAP Core

Basic Operation

The MAP core receives client network protocol data (such as Ethernet) from the system interface, encapsulates this data into GFP frames (either frame-mapped or transparent, depending on the MAP core configuration), and transmits the resulting GFP frames to a SONET/SDH network. The MAP core requires frames to be presented to the system interface using the LocalLink protocol.

Operating the MAP System Interface with Frame-Mapped Frames

When configured for frame-mapped mode operation, the MAP core receives client network protocol data (such as Ethernet frames) from the system interface, encapsulates the client frame into a GFP frame, and inserts the GFP headers and FCS (if enabled). This client data, as required by the *ITU-T GFP Specification*, must be an entire client frame, such as Ethernet or PPP.

The user must also supply all necessary information to compose the GFP headers, unless the core is configured to retrieve the information from the host interface register space. The MAP system interface uses LocalLink signaling, as described below.

Frame Demarcation

When the user asserts `M_SYS_SOF_N` the frame write has begun, and when it asserts `M_SYS_EOF_N` the frame write has ended. It is possible that both signals could assert at the same time, if the GFP frame contains one or fewer words of data, or if it is a management frame.

All valid frame cycles (including the cycles beginning with `M_SYS_SOF_N` and ending with `M_SYS_EOF_N`) must be qualified by the assertion of both `M_SYS_DST_RDY_N` from the core, and `M_SYS_SRC_RDY_N` from the user. If either of those signals are deasserted, both the MAP core and the system side user logic must consider that cycle to be a stall, and the user must hold the value on all control and data signals until both `M_SYS_DST_RDY_N` and `M_SYS_SRC_RDY_N` are asserted.

All valid frame cycles must have data on `M_SYS_DATA`, unless the transmitted frame is a management frame. The data is the "Client Payload Information Field" (CPIF) defined in the *ITU-T GFP Specification*, and it must not include the core or payload headers, or the FCS. The MAP core will insert those fields as appropriate. See "Sideband Fields," page 65 for further CPIF requirements.

When the end of the frame is reached, the user must assert the `M_SYS_EOF_N` signal with the last data word. The `M_SYS_REM` bus is used to indicate how many bytes of the last data word are valid; the number of valid bytes in the word is `M_SYS_REM` + 1, and is MSB justified. The `M_SYS_REM` bus is only valid when `M_SYS_EOF_N` is asserted and will be invalid otherwise. For example, if the interface is 64 bits (8 bytes) wide and `M_SYS_EOF_N` is asserted with a `M_SYS_REM` of 6, then 7 bytes are valid and `M_SYS_DATA[63:8]` contains the final word.

Sideband Fields

In addition to the data bus `M_SYS_DATA`, there are sideband signals that are required to be valid when the start of frame indication `M_SYS_SOF_N` is asserted. The MAP core captures most of the sideband signals at the beginning of a new frame, so it is not necessary

for the user logic to hold the value throughout the frame. The following is a list of these fields:

- M_SYS_LENGTH
- M_SYS_UPI
- M_SYS_MGMT_N

With all frames, the byte length of the data field (CPIF) must be indicated on either the M_SYS_LENGTH bus or the CHANx_GFP_REGISTER_A[15:0] register. The user selects the register map instead of the interface signal as the source of the length by setting CHANx_GFP_REGISTER_A[24]to 1. This must be done for each channel (CHAN0, CHAN1, and so forth) if linear frames are supported. If only null headers are supported, then CHAN0 is the only one necessary to configure.

The MAP core calculates the PLI for the GFP frame by adding four bytes to the length value input for the existence of each of the *type* field, *extension* header, and the FCS. The type field is always required, and is 4 bytes in length. The extension header and the FCS are optional, and if present are 4 bytes each. The user must ensure that the core will never transmit a frame which would exceed the maximum PLI allowed by the *ITU-T GFP Specification*. The equation for calculating the maximum size of the CPIF follows:

$$\text{max length CPIF (in bytes)} = 0\text{xFFFF} - \text{type field} - \text{extension header} - \text{FCS}$$

Table 6-1 lists the maximum length (as defined by the *ITU-T GFP Specification*) for each possible configuration. If a length larger than the maximum value is used, the operation of the core cannot be guaranteed, and should be reset for proper operation.

Table 6-1: **Maximum Length Values**

Core Configuration	Maximum Length CPIF (in bytes) (M_SYS_LENGTH or REGISTER_A[15:0])
No extension header, no FCS	0xFFFB
Either an extension header or an FCS is present, not both	0xFFF7
Both an extension header and an FCS are present	0xFFF3

All frames must have the user payload identifier (UPI) indicated on either the M_SYS_UPI bus or the CHANx_GFP_REGISTER_A[23:16] register. The user may select the register map instead of the interface signal as the source of the length by setting CHANx_GFP_REGISTER_A[25]to 1. This must be done for each channel (CHAN0, CHAN1, and so forth) if linear frames are supported. If only null headers are supported, then only CHAN0 needs to be configured.

See "Management Frames," page 68 for detailed information about the use of management frames (M_SYS_MGMT_N).

If the transmitted GFP frame has an extension field, the channel identifier (CID) must be indicated on M_SYS_CID. Unlike the other sideband signals, M_SYS_CID must be valid for the entire data frame, from the assertion of M_SYS_SOF_N to M_SYS_EOF_N if linear frames are enabled.

If linear frames are to be used, the spare field must be programmed in CHANx_GFP_REGISTER_B[15:8]. If the transmitted GFP frame is to have null headers, then the spare field is not used, and M_SYS_CID will not be used on the interface (CHAN0 is the only register that will be used).

If a channel id not previously aliased to a CHANx configuration is presented on M_SYS_CID, the MAP core defaults to the CHAN0 configuration, enabling the user to support up to 256 channels. See "Operating the MAP Core with Multiple-Channel Support," page 76 for details.

Errors and Discontinues

When the user determines there is an error condition with the frame data, the frame may be discontinued with an immediate assertion of M_SYS_SRC_DSC_N and M_SYS_EOF_N. M_SYS_SOF_N and M_SYS_EOF_N must still demarcate the frame data, but the MAP core will now regard the entire frame as corrupt. As with all frame cycles, M_SYS_SRC_RDY_N and M_SYS_DST_RDY_N both must be asserted when M_SYS_SRC_DSC_N is asserted; otherwise the MAP core will consider the cycle a stall.

The result of the discontinue is that the frame will be padded with 1's until the proper length is reached, the FCS will be inverted (if it exists), and M_LINE_SRC_DSC_N will be asserted at the end of the discontinued frame when it is read out of the line interface. The user should beware that prematurely terminating the current frame may result in a stall on the system interface (the deassertion of M_SYS_DST_RDY_N), as the core may need to pad the current frame until the required length is met. The MAP core will indicate to the user if M_SYS_LENGTH does not match the number of data bytes sent by asserting M_SYS_STATUS_N[0].

The MAP core responds to errored data frames in the following ways:

- Missing SOF - If a new frame is started (immediately after reset, or following an EOF) without an SOF, the data will be ignored and discarded until an SOF initiates a new frame.
- Early, late or missing EOF - The MAP core will always generate proper-length GFP frames, as specified by the length (either on M_SYS_LENGTH or CHANx_GFP_REGISTER_A[15:0]). If an EOF is received too early, then the core will pad the frame with 1's until the correct length is reached, invert the FCS (if it exists), and assert M_SYS_SRC_DSC_N. If an EOF is received too late, or is missing, the core will ignore all data words after the correct length is reached, invert the FCS (if it exists), and assert M_SYS_SRC_DSC_N. The user should beware that prematurely terminating the current frame may result in a stall on the system interface (the deassertion of M_SYS_DST_RDY_N), as the core will need to pad the current frame until the required length is met.
- Early or late EOF with DSC - If M_SYS_SRC_DSC_N is asserted with an early or late EOF, then the core will respond in the same manner as with a early or late EOF. Additionally, M_LINE_SRC_DSC_N will be asserted on the line interface with the end of frame.
- CID change in middle of a frame - if M_SYS_CID is changed in the middle of the frame, the MAP core will pad the current frame with 1s until the proper length is reached, and insert an EOF and DSC, causing the FCS to be inverted on the line interface, as well as asserting M_LINE_SRC_DSC_N with the end of the frame. The user should beware that prematurely terminating the current frame may result in a stall on the system interface (the deassertion of M_SYS_DST_RDY_N), as the core will need to pad the current frame until the required length is met.

For additional information regarding the error handling of the core, see Appendix D, "Status and Error Reporting."

Error Insertion

For testing purposes, the user may induce errors into the GFP frame by programming the register GFP_ERR[7:0] and by asserting M_SYS_FORCE_ERR_N for the duration of a frame. Each bit of the register will induce a different kind of corruption (FCS, tHEC, and so forth), and the signal will permit that corruption to be applied on a frame-by-frame basis.

Management Frames

The MAP core supports generation of management frames by asserting M_SYS_MGMT_N. Note that the MAP core does not support management frames with data, so the values on M_SYS_DATA, M_SYS_REM, and M_SYS_LENGTH are not used (and FCS will not be inserted, even if FCS is enabled in the host interface). The user is expected to present a management frame with M_SYS_SOF_N, M_SYS_EOF_N, M_SYS_SRC_RDY_N, and M_SYS_MGMT_N asserted simultaneously, and hold those values if M_SYS_DST_RDY_N is not asserted. When M_SYS_MGMT_N is asserted, M_SYS_UPI determines the type of management frame to send (see Appendix C, "GFP Frame Format"). If linear frames are enabled, the management frame will contain the spare field as set in the host interface, and channel id as set by M_SYS_CID.

Alternatively, the user can cause management frames to be sent automatically by the core by setting a timer in the register map. To do this, program GFP_CTRL[3:0] to a non-zero value (zero will turn this feature off). Each increment of this value represents 167.8ms at 100MHz. For faster clock frequencies, the increment value may be determined by: 167.8ms * (100MHZ/actual_clock_freq.)

If the timer is used, a management frame will be sent immediately following the expiration of the timer (after any current frames in transit), and the timer automatically restarts. Use of the timer does not preclude using the interface to send management frames. Although a detailed discussion of end-to-end flow control is outside of the scope of this document, the user may use this feature to implement the client signal fail indication as specified by the *ITU-T GFP Specification*.

Sample Operation

Figure 6-1 illustrates a sample operation of the MAP core system interface transmitting frame-mapped frames. Three frames are transferred: a normal data frame, an errored data frame, and a management frame. Note that for data frames, the length and UPI are only required to be asserted during the start of frame, but the channel id must be asserted for the entire frame.

The first frame is a data frame to channel 0x03, and the frame is initiated by asserting M_SYS_SOF_N, M_SYS_DATA, and M_SYS_SRC_RDY_N. The length of the frame, channel id, and UPI are also driven to active values. After two successful writes, the user requests a stall by deasserting M_SYS_SRC_RDY_N. The user then completes the frame by asserting M_SYS_EOF_N and M_SYS_REM with the last valid data word.

The second frame is a data frame to channel 0xFA. Note that in this case the core is requesting a stall by deasserting M_SYS_DST_RDY_N, and the user must hold the values on the system interface until it is asserted. The errored data frame is terminated by asserting M_SYS_EOF_N and M_SYS_SRC_DSC_N with the last data word.

The third frame is a management frame to channel 0x09, and the frame is initiated by asserting M_SYS_SOF_N, M_SYS_EOF_N, and M_SYS_MGMT_N. The UPI value, M_SYS_UPI, determines the type of management frame to be sent.

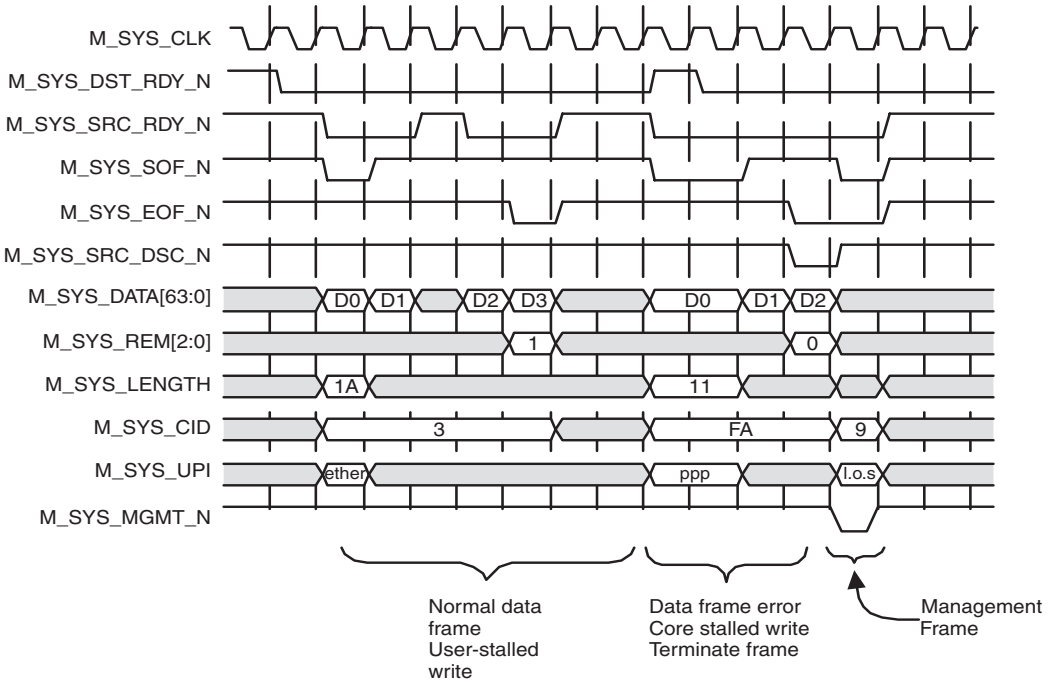

Figure 6-1: **MAP Core Frame-Mapped Transfer**

Operating the MAP System Interface with Transparent Frames

When configured for transparent mode operation, the MAP core will receive a raw stream of 8b/10b decoded data characters (such as Fibre Channel) from the system interface to be encapsulated into fixed-length GFP frames. Additionally, the user must ensure that the register space is configured for the correct frame settings (length, UPI, and so forth). The MAP core will encapsulate the raw data stream into a GFP frame, inserting the GFP headers and FCS (if it is enabled).

Frame Demarcation

The MAP core generates fixed-length GFP frames when configured in transparent mode. The frame starts when the user initiates a write to the core (asserts M_SYS_SRC_RDY_N and the core asserts M_SYS_DST_RDY_N). The MAP core will then complete the current frame, by inserting either the stream of data characters from the user, or pad words (65B_PAD as defined by the *ITU-T GFP Specification*) every cycle until the end of the frame is reached. Note that unlike frame-mapped mode, a stall on the system interface will result in pad words being inserted if the MAP core is in the middle of transmitting a frame. When the end of the frame is reached, the MAP core will automatically begin transmission of another GFP frame if data exists, else it will insert idle frames.

Transparent mode signaling on the MAP system interface does not use M_SYS_SOF_N or M_SYS_EOF_N for data frames because it is assumed that the user does not know anything about the frame boundaries of the underlying protocol. All valid line interface write cycles must be qualified by the assertion of both M_SYS_DST_RDY_N from the core, and M_SYS_SRC_RDY_N from the user. If either of those signals are deasserted, both the MAP

core and the system side user logic must consider that cycle to be a stall, and the user must hold the value on all control and data signals until both M_SYS_DST_RDY_N and M_SYS_SRC_RDY_N are asserted. The MAP core will continue sending data pad characters during a stall; a stall on the system interface will not cause a stall on the line interface.

The length of the frames (on a per-channel basis) is configured through the host interface, register CHANx_GFP_REGISTER_A[15:0]. This length is the number of superblocks per GFP frame, not the number of bytes per frame as in frame-mapped mode. The maximum length for a transparent frame when linear frames and FCS are both disabled is 0x3D2, and will vary depending on the core configuration. See "Sideband Fields," page 71 for additional requirements and calculations of the maximum number of superblocks per frame. See the *ITU-T GFP Specification* for the recommended number of superblocks per frame.

To provide the user some warning that the current frame is ending, the MAP core will assert M_SYS_FM_RDY_N at least 4 cycles (in 32-bit) or 2 cycles (in 64-bit) prior to the end of the current frame. The user may therefore choose to stop sending data in the next few cycles causing minimal pad characters to be inserted into the frame, change the channel on the next cycle to begin a new frame on a new channel, or continue sending data on the same channel, which will be packaged into a new frame. Note that M_SYS_FM_RDY_N is only meaningful when the number of superblocks per frame is greater than one. This implementation provides a simple interface enabling the user to minimize the number of pad words inserted per transparent mode frame.

The user may induce an early end to the data in the current transparent mode frame by prematurely changing M_SYS_CID (changing the value without waiting for M_SYS_FM_RDY_N). If this occurs, the current GFP frame transmission will automatically have pad words inserted to fill out to the required frame length. The user should beware of prematurely changing the channel if the current frame length is long (several superblocks); it can take many clock cycles for the current frame to be padded out to its end, and until the core is finished, M_SYS_DST_RDY_N may be forced to deassert.

Data inside transparent frames are mapped to *superblocks*, which reorder the 8b/10b control and data characters into a 64b/65b coding scheme. There may be one or several superblocks in a transparent GFP frame. The data is the Client Payload Information Field (CPIF) defined in the *ITU-T GFP Specification*, and it must not include the core or payload headers, or the FCS. The MAP core will insert those fields as appropriate. The 64b/65b encoding that is used with transparent mode is performed by the MAP core; the user should present the unencoded 8b data characters on the data bus. If raw 10b characters are received by the user logic, it is the user's responsibility to perform 8b/10b decoding to get the original 8b characters prior to writing into the system interface.

The user must know the position of the 8b/10b control characters in the data word, and indicate their position(s) on M_SYS_CHARISK_N, which is 4 or 8 bits wide. Each bit corresponds to a byte on M_SYS_DATA, and if the bit is low, the corresponding byte is a control character. For instance, if M_SYS_CHARISK_N is 01111101, then M_SYS_DATA[63:56] and M_SYS_DATA[15:8] are control codes. The control code mappings are shown in Table 6-2, and the meaning of the 8b/10b control codes are specific to the client protocol.

Table 6-2: **M_SYS_DATA Byte to 8b/10b Control Code Mapping**

8b/10b Control Character	M_SYS_DATA
K28.0	0x1C
K28.1	0x3C

Table 6-2: **M_SYS_DATA Byte to 8b/10b Control Code Mapping**

8b/10b Control Character	M_SYS_DATA
K28.2	0x5C
K28.3	0x7C
K28.4	0x9C
K28.5	0xBC
K28.6	0xDC
K28.7	0xFC
K23.7	0xF7
K27.7	0xFB
K29.7	0xFD
K30.7	0xFE

All non-valid K characters (valid K characters shown in Table 6-2) inserted on the system interface will result in 10B_ERR words inserted in the GFP frame. If pad words need to be inserted into the transparent frame, the user must use the M_SYS_REM bus, as described below. If the user attempts to insert pad words by using the 65B_PAD K character, the 65B_PAD K character will result in a 10B_ERR inserted into the GFP frame as this not a valid K character.

Unlike in frame-mapped mode, the M_SYS_REM signal is valid on all non-stalled frame cycles. This enables the user to send in partial words throughout the transfer; the remaining bytes of the word will be filled with 65B_PAD characters (a special filler character defined by the *ITU-T GFP Specification*). The number of valid bytes is M_SYS_REM + 1. For example, if the interface is 8 bytes wide and M_SYS_REM of 6, then 7 bytes are valid and M_SYS_DATA[63:8] contains the data characters, and one pad character will be inserted into the GFP frame.

Sideband Fields

The only system interface sideband field required when in transparent mode is the channel id M_SYS_CID (only valid if linear frames are enabled). The M_SYS_CID signal must be valid for the entire frame, as prematurely changing the channel id will cause the current frame to be padded until the correct length is met. The user should beware of prematurely changing the channel if the current frame length is long (several superblocks); it may take a long time for the current frame to be padded out to its end, and until the core is finished, M_SYS_DST_RDY_N may be forced to deassert. If a channel id is presented on M_SYS_CID that was not previously aliased to a CHANx configuration, then the MAP core defaults to the CHAN0 configuration. See "Operating the MAP Core with Multiple-Channel Support," page 76 for details. When linear frames are enabled, the spare field must also be programmed in CHANx_GFP_REGISTER_B[15:8]. If the transmitted GFP frame uses null headers, M_SYS_CID will not be used on the interface (CHAN0 is the only register that will be used), and the spare field is not used.

For all data frames, the length of the data field (CPIF) must be indicated on the CHANx_GFP_REGISTER_A[15:0] register. In transparent mode, the length of the data field is specified in number of superblocks per GFP frame. This must be configured for

each channel (CHAN0, CHAN1, etc.). If only null headers are supported, then CHAN0 is the only channel necessary to configure.

The length value (the product of CHANx_GFP_REGISTER_A[15:0] and 67) becomes the payload length indicator (PLI) field of the GFP frame by adding 4 bytes to the value for the existence of each of the *type* field, the *extension* field, and the FCS. The type field is always required, and is 4 bytes in length. The extension header and the FCS are optional, and if present are 4 bytes each. The number o f bytes per superblock is 67 (64 bytes of data, 1 byte 64b/65b flags, 2 bytes for CRC). The user must ensure that the core will never transmit a frame which would exceed the maximum PLI allowed by the *ITU-T GFP Specification*. The equation for calculating the maximum number of superblocks follows:

$$\text{max length CPIF (in superblocks)} = \frac{\text{0xFFFF} - \text{type field} - \text{extension header} - \text{FCS}}{67}$$

Table 6-3 lists the maximum number of superblocks for each possible configuration. If a length larger than the maximum value is used, the operation of the core can not be guaranteed, and should be reset before proper operation will continue.

Table 6-3: **Transparent Mode Maximum Length Frames**

Core Configuration	Maximum Length CPIF (in superblocks) (REGISTER_A[15:0])
No extension header, no FCS	0x03D2
Either an extension header or FCS is present, not both	0x03D2
Both an extension header and FCS are present	0x03D1

All frames must have the user payload identifier (UPI) indicated on the CHANx_GFP_REGISTER_A[23:16] register. This must be done for each channel (CHAN0, CHAN1, etc.). If only null headers are supported, then CHAN0 is the only channel necessary to configure. The system interface UPI signal, M_SYS_UPI, is only used for management frames, as described below.

Errors and Discontinues

When the user determines there is an error condition with the frame data, the frame may be discontinued with an immediate assertion of M_SYS_SRC_DSC_N. The MAP core will now regard the entire frame as corrupt, and will pad out the rest of the frame and invert the FCS, if it exists. The length of the frame will not be shorter as a result of the discontinue. As with all frame cycles, M_SYS_SRC_RDY_N and M_SYS_DST_RDY_N both must be asserted when M_SYS_SRC_DSC_N is asserted; otherwise the MAP core will consider the cycle a stall. In addition to padding, the result of the discontinue is that M_LINE_SRC_DSC_N will be asserted at the end of the discontinued frame when it is read out of the line interface.

The user should note that performing a discontinue at some point other than the intended end of the frame (defined by length CHANx_GFP_REGISTER_A[15:0]) may force a stall on the system interface (the core deasserts M_SYS_DST_RDY_N), as the MAP core will pad

out the current frame before starting a new frame. M_SYS_FM_RDY_N will still be asserted at least 4 cycles (in 32-bit) or 2 cycles (in 64-bit) before the end of the current frame.

There are error conditions associated with transparent frames that do not necessitate discarding the entire frame. Sometimes the 8b/10b decode logic is not able to recognize a 10b character. If this happens, the *ITU-T GFP Specification* defines a special character called 10B_ERR to be sent in its place. If the user encounters this situation and wants to send a 10B_ERR character, the user can indicate this on M_SYS_10BERR_N. Each bit corresponds to a byte on M_SYS_DATA, and when active, the current data word is replaced with the 10B_ERR code word. For instance, if M_SYS_10BERR_N is 11011110, then the byte positions occupied by M_SYS_DATA[47:40] and M_SYS_DATA[7:0] are to be sent as 10B_ERR codes. Note that unlike M_SYS_CHARISK_N, the value on the corresponding byte positions of M_SYS_DATA is not interpreted and is always replaced with 10B_ERR characters. If the user inputs an invalid K character (see Table 6-2 for the list of valid K characters), the MAP core automatically replaces the word with the 10B_ERR word and reports this on M_SYS_STATUS_N[1] and MAP register GFP_INT[1] if enabled. M_SYS_10BERR_N will override M_SYS_CHARISK_N if both signals are asserted simultaneously.

For additional information regarding the error handling of the core, see Appendix D, "Status and Error Reporting."

Error Insertion

For testing purposes, the user may induce errors into the GFP frame by programming the register GFP_ERR[7:0] and by asserting M_SYS_FORCE_ERR_N for the duration of a frame. Each bit of the register will induce a different kind of corruption (FCS, tHEC, and so forth), and the signal will permit that corruption to be applied on a frame-by-frame basis.

Management Frames

The MAP core supports the generation of management frames. Note that the MAP core does not support management frames with data, so the values on M_SYS_DATA and M_SYS_REM are ignored when M_SYS_MGMT_N is asserted. The user is expected to present a frame with M_SYS_SRC_RDY_N and M_SYS_MGMT_N asserted simultaneously, and hold those values if M_SYS_DST_RDY_N is not asserted. When M_SYS_MGMT_N is asserted, M_SYS_UPI determines the type of management frame to send. See Appendix C, "Packet and Control Symbol Format." Asserting M_SYS_MGMT_N in the middle of a data frame causes the current frame to be terminated with pad words, after which the management frame is transmitted.

Alternatively, the user can cause management frames to be sent automatically by the core by setting a timer in the register map. To do this, program GFP_CTRL[3:0] to a non-zero value (zero will turn this feature off). Each increment of this value represents 167.8ms at 100MHz. For faster clock frequencies, the increment value may be determined by: 167.8ms * (100MHZ/actual_clock_freq.)

If the timer is used, a management frame will be sent immediately following the expiration of the timer (after any current frames in transit), and the timer will automatically restart. Use of the timer does not preclude using the interface to send management frames. Although a detailed discussion of end-to-end flow control is outside of the scope of this document, the user may use this feature to implement the client signal fail indication as specified by the *ITU-T GFP Specification*.

Sample Operation

Figure 6-2 illustrates a sample operation of the MAP core system interface transmitting transparent mode frames. Three frames are transferred: two normal data frames and a management frame. Note that the channel id must be asserted for the entire frame.

The first frame is a data frame to channel 0x03, and the frame is initiated by asserting M_SYS_DATA, M_SYS_REM, and M_SYS_SRC_RDY_N. Note that M_SYS_REM, M_SYS_CHARISK_N, and M_SYS_10BERR_N are valid on every write cycle, and indicate the type of data word that will be transmitted. After four successful writes, the core is requesting a stall by deasserting M_SYS_DST_RDY_N, and the user must hold the values on the system interface until it is asserted. M_SYS_FM_RDY_N is asserted at least 4 cycles (in 32-bit) or 2 cycles (in 64-bit) before the end of the current frame, providing a look-ahead to the user indicating when the start of the next frame transmission will begin.

The second frame is a data frame to channel 0x09. Three words are written for this frame, and then the user swaps channels to send a management frame. Note that this will cause a frame to be generated with three valid words, and enough pad words to complete the frame as set by the length field (CHANx_GFP_REGISTER_A[15:0])

The third frame is a management frame to channel 0x05, and the frame is initiated by asserting M_SYS_MGMT_N. The UPI value, M_SYS_UPI, determines the type of management frame to be sent.

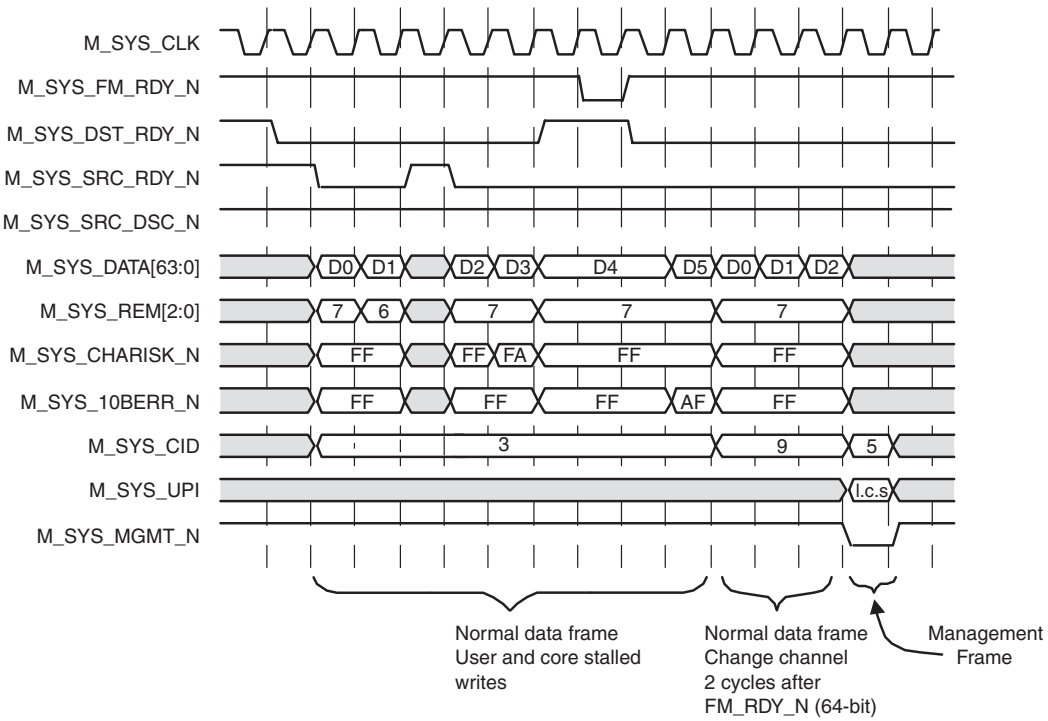

Figure 6-2: **MAP Core Transparent Transfer**

Operating the MAP System Interface in Mixed Mode

If the MAP core is configured to handle both frame-mapped and transparent frames, then it is typical that the user will have a multi-channel configuration (linear frames enabled), and that each channel is configured as being in either frame-mapped or transparent mode.

When a frame is to be transmitted, the mode of the frame is specified by register CHANx_GFP_REGISTER_A[27], and the channel is indicated by M_SYS_CID.

This scheme is not required; it is possible to have mixed mode traffic on a given channel (both frame-mapped and transparent), and it is possible to not use channels at all (null headers) and have mixed mode traffic. If either is the case, the user logic simply reprograms the channel(s) to be frame-mapped or transparent while applying a software reset to the core.

The system interface in mixed mode operates identically to the individual frame-mapped and transparent modes, except that during a frame-mapped operation the transparent mode-specific signals will not be used (M_SYS_CHARISK_N, M_SYS_10BERR_N, etc.), and in transparent mode the frame-mapped signals will not be used (M_SYS_SOF_N, M_SYS_EOF_N, M_SYS_LENGTH). The user may still rely on M_SYS_FM_RDY_N to indicate when to start a frame-mapped frame when making the transition from transparent to frame-mapped mode.

Management frames may be sent using the transparent mode method to the current channel only if the channel is configured to be a transparent channel. Since M_SYS_SOF_N and M_SYS_EOF_N are ignored in transparent mode, it is recommended to send management frames using the frame-mapped method (M_SYS_SOF_N, M_SYS_EOF_N, M_SYS_SRC_RDY_N, and M_SYS_MGMT_N) for both frame-mapped and transparent mode.

Operating the MAP Line Interface

The line interface of the MAP core transmits GFP encapsulated frames to the line side user. The MAP core line interface utilizes the LocalLink protocol and provides data with indicators as to where the start and end of frames are occurring. The line interface operation is the same regardless of the core configuration (frame-mapped or transparent mode).

When the core logic asserts M_LINE_SOF_N the frame has begun, and when it asserts M_LINE_EOF_N the frame has ended.

All valid frame cycles (including the cycles beginning with M_LINE_SOF_N and ending with M_LINE_EOF_N) must be qualified by the assertion of both M_LINE_SRC_RDY_N from the core, and M_LINE_DST_RDY_N from the user. If either of those signals are deasserted, both the MAP core and the line side user logic must consider that cycle to be a stall, and the core will hold the value on all line control and data signals until both M_LINE_DST_RDY_N and M_LINE_SRC_RDY_N are asserted.

All non-stalled frame cycles will have valid data on M_LINE_DATA. The data transmitted on the line interface is the entire GFP frame, including the core header, payload headers, client payload information field (CPIF), and the FCS.

When the end of a frame is reached, the core will assert the M_LINE_EOF_N signal with the last frame word. The M_LINE_REM bus is used to indicate how many bytes of the last frame word are valid; the number of valid bytes is M_LINE_REM + 1. The M_LINE_REM bus is only valid when M_LINE_EOF_N is asserted and must ignored otherwise. For example, if the interface is 8 bytes wide and M_LINE_EOF_N is asserted with a M_LINE_REM of 6, then 7 bytes are valid and M_LINE_DATA[63:8] contains the final word.

If an error was reported by the system-side user while creating the GFP frame (via M_SYS_SRC_DSC_N), the M_LINE_SRC_DSC_N signal will assert with M_LINE_EOF_N on the errored frame.

If there are no GFP frames pending or the core is in reset, the line interface will transmit GFP idle frames. The idles are transmitted the same way other frame types are; the user

will see a simultaneous assertion of M_LINE_SOF_N, M_LINE_EOF_N, and M_LINE_SRC_RDY_N. The signal M_LINE_IDLE_N will assert to identify them as idle frames in case the user wants to discard them.

Sample Operation

Figure 6-3 illustrates a sample operation of the MAP core line interface. When the line interface has no data to send, GFP idle frames are automatically generated. In this example, the first two words sent are idle frames, as indicated by the assertion of M_LINE_SOF_N, M_LINE_EOF_N, and M_LINE_IDLE_N. The idle word contents will vary depending on the setting for core header scrambling (either unscrambled - 0x00000000, or scrambled 0xB6AB31E0). Following the two idle frames, a GFP data frame is transmitted. The first word, indicated by the assertion of M_LINE_SOF_N, contains the core header. The second word contains the type field, followed by three data payload words. The frame is terminated with the assertion of M_LINE_EOF_N and M_LINE_REM. Note that this example is for a 32-bit data bus. In the 64-bit case, both the core header and the type field would be transmitted with the first data word.

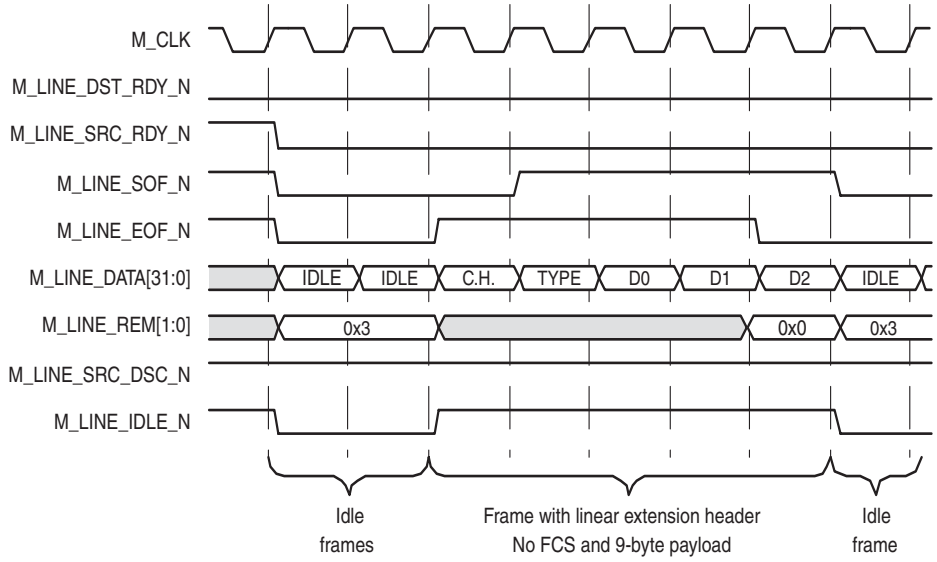

Figure 6-3: **MAP Core Transparent Transfer**

Operating the MAP Core with Multiple-Channel Support

The MAP core supports extension headers for linear frames (enabling multi-channel support). Using the CORE Generator GUI, the MAP core can be configured to support either linear frames (enabling one to ten channels) or null extension headers (enabling one channel). For details about generating the MAP core with the CORE Generator, see Chapter 4, "Generating the Core."

Each channel has a unique register space (CHANx_GFP_REGISTER_A/B). CHANx_GFP_REGISTER_B[7:0] is the alias field, and maps the logical channel number (CHAN0 - CHAN9) to a user-specified channel number (channel identification, CID). This enables the user to configure up to ten unique channels, but supports CID values from 0-255. Each logical channel corresponds to only a single user-specified CID value. The M_SYS_CID value the user writes into the MAP interface must be a user-specified value from 0-255, as this CID is mapped in the host interface to the logical channel number for the register access. If a CID is received by the core that is not pre-defined in the alias

register, the logical channel 0 (CHAN0) register space is accessed. An example of a MAP core with two channels follows:

`CHAN0_GFP_REGISTER_B[7:0]` = 0x4D
`CHAN1_GFP_REGISTER_B[7:0]` = 0x2F

A `M_SYS_CID` value of 0x4D accesses the logical channel 0 (CHAN0), and a `M_SYS_CID` value of 0x2F accesses the logical channel 1 (CHAN1). If a `M_SYS_CID` value of any other value is written into the system interface, the `M_SYS_CID` presented by the user is inserted into the linear frame header, and the frame is generated using the channel 0 (CHAN0) configuration.

UNMAP Core

Basic Operation

The UNMAP core receives GFP encapsulated frames from a SONET/SDH network on the line interface, de-maps the GFP frames into client network protocol data, and passes the data onto the client via the system interface. The UNMAP core permits frames to be presented to the line interface in either a streaming format or using the LocalLink protocol. In the case of streaming data, the boundaries of GFP frames have not yet been demarcated, and the GFP core will locate the frame boundaries according to the frame delineation algorithm detailed in the *ITU-T GFP Specification*.

Operating the UNMAP System Interface with Frame-Mapped Frames

When configured for frame-mapped mode operation, the UNMAP core relays to the system interface the client network protocol data that has been extracted from the GFP frame. This client data, as required by the *ITU-T GFP Specification*, will be an entire client frame, such as Ethernet or PPP. The UNMAP core will strip out all GFP headers and FCS (if it exists), and present the entire client data frame to the user. The fields and information from the GFP headers are presented alongside the payload data. If an error exists in the GFP frame, whether in the FCS or another field, it will be indicated. The UNMAP system interface uses LocalLink signaling as described below.

LocalLink Frame Demarcation

When the UNMAP core asserts `U_SYS_SOF_N` the frame read has begun, and when it asserts `U_SYS_EOF_N` the frame read has ended. It is possible that both signals could assert at the same time, if the GFP frame contained only one word or less of data.

All valid frame cycles (including the cycles beginning with `U_SYS_SOF_N` and ending with `U_SYS_EOF_N`) must be qualified by the assertion of both `U_SYS_SRC_RDY_N` from the core, and `U_SYS_DST_RDY_N` from the user. If either of those signals are deasserted, both the UNMAP core and the system side user logic must consider that cycle to be a stall, and the UNMAP core will hold the value on all control and data signals until both `U_SYS_DST_RDY_N` and `U_SYS_SRC_RDY_N` are asserted.

All valid non-stalled frame cycles will have data on `U_SYS_DATA`. The data is the "Client Payload Information Field" (CPIF) defined in the *ITU-T GFP Specification*, and it will not include the core or payload headers, or the FCS. If a GFP data frame is received with no CPIF, that frame will be dropped and it will not be presented on the system interface.

When the end of a frame is reached, the core asserts the `U_SYS_EOF_N` signal with the last data word. The `U_SYS_REM` bus is used to indicate how many bytes of the last data word

are valid; the number of valid bytes is U_SYS_REM + 1. The U_SYS_REM bus is only valid when U_SYS_EOF_N is asserted and will be invalid otherwise. For example, if the interface is 64 bits (8 bytes) wide and U_SYS_EOF_N is asserted with a U_SYS_REM of 6, then 7 bytes are valid and U_SYS_DATA[63:8] contains the final word.

Sideband Fields

There are fields associated with each GFP frame, and those fields will be valid starting with the assertion of U_SYS_SOF_N, and remain constant throughout the frame read for a given frame. The list of fields includes:

- U_SYS_LENGTH
- U_SYS_UPI
- U_SYS_CID
- U_SYS_SPARE
- U_SYS_MGMT_N

With all frames, the byte length of the data field (CPIF) is indicated on U_SYS_LENGTH. The UNMAP core calculates the byte length of the CPIF by subtracting four bytes for the existence of each of the *type* field, *extension* header, and the FCS from the payload length identifier (PLI).

All frames indicate the user payload identifier (UPI) field on U_SYS_UPI.

If the received GFP frame has an extension field, the channel identifier (CID) and spare field will be indicated on U_SYS_CID and U_SYS_SPARE, respectively. Otherwise, these signals should be considered invalid (and may not be present, depending on the core configuration).

For detailed information about management frames, see (U_SYS_MGMT_N) "Management Frames."

Errors and Discontinues

When there is an error condition with the frame (such as an FCS error), the signal U_SYS_SRC_DSC_N will assert on the last cycle of the frame with U_SYS_EOF_N. U_SYS_SOF_N and U_SYS_EOF_N will still demarcate the frame, but the client must now regard the entire frame as corrupt. The cause of the error condition will be signaled on U_SYS_ERRBUS_N. As with all frame cycles, U_SYS_SRC_RDY_N and U_SYS_DST_RDY_N both must be asserted when U_SYS_SRC_DSC_N asserts; otherwise the UNMAP core will hold the values on all signals until both U_SYS_DST_RDY_N and U_SYS_SRC_RDY_N are asserted.

For additional information regarding the error handling of the core, see Appendix E, "Sample GFP Frames."

Management Frames

When management frames are received, the frame is indicated as a management frame by asserting U_SYS_MGMT_N. The management frame will demark the frame boundary with the use of U_SYS_SOF_N, U_SYS_EOF_N, U_SYS_SRC_RDY_N, and U_SYS_MGMT_N. The type of management frame is indicated on U_SYS_UPI. See , "Packet and Control Symbol Format," for details.

Sample Operation

Figure 6-4 illustrates a sample operation of the UNMAP core system interface transmitting frame-mapped frames. Three frames are transferred: a normal data frame, errored data frame, and a management frame. Note that for the data frames, the length, channel id and UPI are asserted for the entire frame.

The first frame is a data frame to channel 0x03, and is initiated by asserting U_SYS_SOF_N, U_SYS_DATA, and U_SYS_SRC_RDY_N. After two successful reads, the user requests a stall by deasserting U_SYS_DST_RDY_N. The frame is then completed by asserting U_SYS_EOF_N and U_SYS_REM with the last valid data word.

The second frame is a data frame to channel 0x01. This data frame is an errored data frame, and is terminated by asserting U_SYS_EOF_N and U_SYS_SRC_DSC_N with the last data word. The signal U_SYS_ERRBUS_N will indicate the type of error that occurred for this frame (in this case an eHEC error was detected).

The third frame is a management frame to channel 0x09, and is initiated by asserting U_SYS_SOF_N, U_SYS_EOF_N, and U_SYS_MGMT_N. The UPI value, U_SYS_UPI, indicates the type of management frame received.

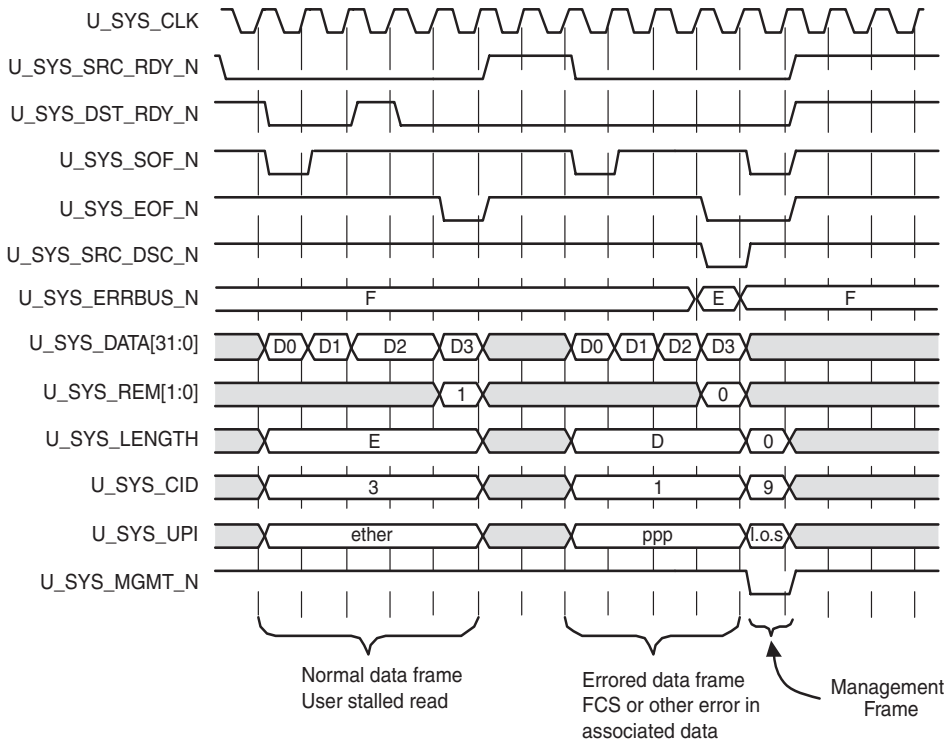

Figure 6-4: **UNMAP Core Frame-Mapped Transfer**

Operating the UNMAP System Interface with Transparent Frames

When configured for transparent mode operation, the UNMAP core will relay to the system interface the client network protocol data which has been extracted from the GFP frame. This client data, as required by the *ITU-T GFP Specification*, will be a raw stream of 8b/10b decoded data characters (such as Fibre Channel), which are packaged into fixed-length GFP frames. The UNMAP core will strip out all GFP headers and FCS (if it exists), and present the streaming client data frame to the user. The 64b/65b encoding that is used

with transparent mode is decoded back to the original data characters before passing the information to the user. Furthermore, it will remove all pad words (65B_PAD) from the data stream, as well as align the received data into complete data words. The fields and information from the GFP headers are presented alongside the payload data. If an error exists in the GFP frame, whether in the FCS or another field, it will be indicated. The UNMAP system interface uses LocalLink signaling, as described below.

LocalLink Frame Demarcation

When the UNMAP core asserts U_SYS_SOF_N the frame read has begun, and when it asserts U_SYS_EOF_N the frame read has ended. It is possible that both signals could assert at the same time, if the GFP frame contained only one word or less of data.

All valid frame cycles (including the cycles beginning with U_SYS_SOF_N and ending with U_SYS_EOF_N) must be qualified by the assertion of both U_SYS_SRC_RDY_N from the core, and U_SYS_DST_RDY_N from the user. If either of those signals are deasserted, both the UNMAP core and the system side user logic must consider that cycle to be a stall, and the UNMAP core will hold the value on all control and data signals until both U_SYS_DST_RDY_N and U_SYS_SRC_RDY_N are asserted.

All valid non-stalled frame cycles will have data on U_SYS_DATA. The data is the Client Payload Information Field (CPIF) defined in the *ITU-T GFP Specification*, and it will not include the core or payload headers, or the FCS. If a GFP data frame is received with no CPIF, that frame will be dropped and it will not be presented on the system interface.

Data inside transparent frames are mapped to *superblocks*, which reorder the 8b/10b control and data characters into a 64b/65b coding scheme. There may be one or several superblocks in a transparent GFP frame. The UNMAP core will decode these superblocks, and data will be presented in the original 8b format. The position of the control characters in the data word are indicated on U_SYS_CHARISK_N, which is 4 or 8 bits wide. Each bit corresponds to a byte on U_SYS_DATA, and if the bit is active, the corresponding byte is a control character. For instance, if U_SYS_CHARISK_N is "11101101", then U_SYS_DATA[39:32] and U_SYS_DATA[15:8] are control codes. The control code mappings are shown in Table 6-4, and the meaning of the control codes are specific to the client protocol.

Table 6-4: **U_SYS_DATA Byte to 8b/10b Control Code Mapping**

8b/10b Control Character	U_SYS_DATA
K28.0	0x1C
K28.1	0x3C
K28.2	0x5C
K28.3	0x7C
K28.4	0x9C
K28.5	0xBC
K28.6	0xDC
K28.7	0xFC
K23.7	0xF7
K27.7	0xFB

Table 6-4: **U_SYS_DATA Byte to 8b/10b Control Code Mapping**

8b/10b Control Character	U_SYS_DATA
K29.7	0xFD
K30.7	0xFE
10B_ERR	0x01
Spare	0x03
Spare	0x04

Furthermore, the *ITU-T GFP Specification* defines a special character called 65B_PAD, which is a filler character. The UNMAP core will remove these characters, and re-compact the data. All valid cycles, except possibly the final data cycle, will have complete words.

When the end of a frame is reached, the core asserts the U_SYS_EOF_N signal with the last data word. The U_SYS_REM bus is used to indicate how many bytes of the last data word are valid; the number of valid bytes is U_SYS_REM + 1. The U_SYS_REM bus is only valid when U_SYS_EOF_N is asserted and will be invalid otherwise. For example, if the interface is 8 bytes wide and U_SYS_EOF_N is asserted with a U_SYS_REM of 6, then 7 bytes are valid and U_SYS_DATA[63:8] contains the final word.

Sideband Fields

There are fields associated with each GFP frame, and those fields will be valid starting with the assertion of U_SYS_SOF_N, and will remain constant throughout the frame read for a given frame. The following is a list of these fields:

- U_SYS_LENGTH
- U_SYS_CID
- U_SYS_UPI
- U_SYS_SPARE
- U_SYS_MGMT_N

With all frames, the byte length of the data field (CPIF) is indicated on U_SYS_LENGTH. The UNMAP core calculates the byte length of the CPIF by subtracting four bytes for the existence of each *type* field, *extension* header, and the FCS from the payload length identifier (PLI). Note that the U_SYS_LENGTH for transparent frames will indicate the maximum possible CPIF value for the given frame. If pad words are inserted into the frame, the UNMAP core will drop these words, and U_SYS_LENGTH will be greater than the actual frame length presented to the user.

If the received GFP frame has an extension field, the channel identifier (CID) and spare field will be indicated on U_SYS_CID and U_SYS_SPARE, respectively. Otherwise, these signals should be considered invalid (and may not be present, depending on the core configuration).

All frames will have the user payload identifier (UPI) field indicated on U_SYS_UPI.

Errors and Discontinues

When there is an error condition with the GFP frame (such as an FCS error), the signal U_SYS_SRC_DSC_N will assert on the last cycle of the frame with U_SYS_EOF_N. U_SYS_SOF_N and U_SYS_EOF_N will still demarcate the frame, but the client must now regard the entire frame as corrupt. The cause of the error condition will be signaled on

U_SYS_ERRBUS_N. If error reporting is disabled for a particular error condition (specified in UNMAP GFP_CTRL), neither U_SYS_SRC_DSC_N nor U_SYS_ERRBUS_N will be asserted for that condition. See Table 5-3 for details. As with all frame cycles, U_SYS_SRC_RDY_N and U_SYS_DST_RDY_N both must be asserted when U_SYS_SRC_DSC_N asserts; otherwise the UNMAP core will hold the values on all signals until both U_SYS_DST_RDY_N and U_SYS_SRC_RDY_N are asserted.

The superblock boundaries will be indicated with the U_SYS_SUPER_N signal. Whenever this signal is asserted, the start of a superblock is indicated. Each superblock has its own CRC-16 check, and if a superblock has a CRC-16 error, the user side logic has the option of discarding only that superblock, but not discarding the remaining superblocks. A superblock CRC-16 error is indicated by the assertion of U_SYS_SUPERERR_N on the final word of the superblock. Because there may be 65B_PAD removal and compacting, there is a possibility that U_SYS_SUPER_N and U_SYS_SUPERERR_N assert the same cycle. There are two possible interpretations of this event:

- A superblock is received that is composed almost entirely of 65B_PAD characters, and the superblock was compacted to one word. That compacted superblock had a CRC-16 error.
- A superblock was compacted such that its ending byte is not on an aligned word boundary, and the first byte of the following superblock is therefore in the same word. The first superblock had a CRC-16 error.

The user side logic may do the selective removal of the bad superblock by discarding all data cycles starting with the previous assertion of U_SYS_SUPER_N to the cycle where U_SYS_SUPERERR_N is asserted. If they are simultaneously asserted, the previous assertion of U_SYS_SUPER_N should still be used, unless the error asserts on the first word of the GFP frame. Note that a CRC-16 error on a superblock will not cause an assertion of U_SYS_SRC_DSC_N. If the user wishes to discard the entire frame due to an individual superblock error, then U_SYS_SUPERERR_N should be captured, and that captured value can be applied during U_SYS_EOF_N like a discontinue.

There are additional error conditions associated with transparent frames that do not necessitate discarding the entire frame. When a GFP transparent frame is mapped, sometimes the mapping logic is not able to interpret the 8b/10b character. If this happens, the *ITU-T GFP Specification* defines a special character called 10B_ERR. If the UNMAP core receives a 10B_ERR, it does not remove the character, but it does indicate to the user logic that the byte is invalid. This condition is signaled on U_SYS_10BERR_N. Each bit corresponds to a byte on U_SYS_DATA, and if the bit is low, the corresponding byte is a 10B_ERR character. For instance, if U_SYS_10BERR_N is "11011110", then U_SYS_DATA[47:40] and U_SYS_DATA[7:0] are 10B_ERR codes. How this is handled by the user-side logic is specific to the client protocol being implemented.

For additional information regarding the error handling of the core, see Appendix D, "Status and Error Reporting."

Management Frames

When management frames are received, the frame is indicated as a management frame by asserting U_SYS_MGMT_N. The management frame will demark the frame boundary with the use of U_SYS_SOF_N, U_SYS_EOF_N, U_SYS_SRC_RDY_N, and U_SYS_MGMT_N. The type of management frame is indicated on U_SYS_UPI (see Appendix C, "Packet and Control Symbol Format").

Sample Operation

Figure 6-5 illustrates a sample operation of the UNMAP core system interface transmitting transparent mode frames. Two frames are transferred: an errored data frame, and a normal data frame. For all data frames, the channel id and UPI are asserted for the entire frame. Additionally, the U_SYS_REM, U_SYS_CHARISK_N, and U_SYS_10BERR_N signals are valid on every read cycle, and indicate the type of data word received.

The first frame is a data frame to channel 0x03, and is initiated by asserting U_SYS_SOF_N, U_SYS_DATA, U_SYS_REM and U_SYS_SRC_RDY_N. The U_SYS_SUPER_N signal indicates the start of a superblock. After two successful reads, the user requests a stall by deasserting U_SYS_DST_RDY_N. The first superblock is completed with an error (U_SYS_SUPERERR_N), indicating that a CRC error occurred during the superblock. The start of the second superblock is indicated with the assertion of U_SYS_SUPER_N, and the end of the frame is the completed by asserting U_SYS_EOF_N and U_SYS_SRC_DSC_N.

The second frame is a data frame to channel 0x09.

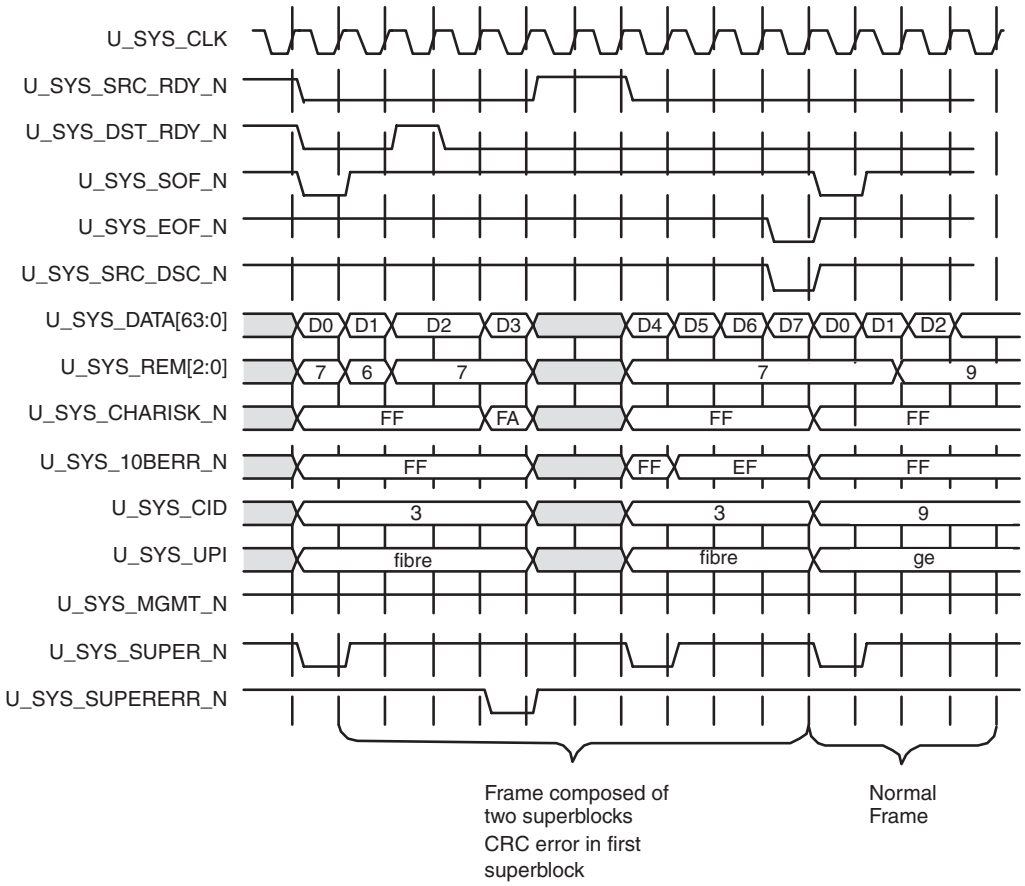

Figure 6-5: **UNMAP Core Transparent Transfer**

Operating the UNMAP System Interface in Mixed Mode

If the UNMAP core is configured to handle both frame-mapped and transparent frames, then it is typical that the user will have a multi-channel configuration (linear headers in use), and that each channel is configured as either frame-mapped or transparent mode.

When a frame is received, the mode of the frame is therefore implied by its channel id on U_SYS_CID.

This scheme is not required; it is possible to have mixed mode traffic on a given channel (both frame-mapped and transparent), and it is possible to not use channels at all (null headers) and have mixed mode traffic. If either is the case, then the user logic must determine the mode of a frame based on the U_SYS_UPI field.

The system interface in mixed mode operates identically to the individual frame-mapped and transparent modes, except that during a frame-mapped operation the transparent mode-specific signals should be ignored (U_SYS_CHARISK_N, U_SYS_10BERR_N, and so forth).

Operating the UNMAP Line Interface with Streaming Data

The line interface of the UNMAP core receives GFP encapsulated frames from the line side user. If the UNMAP core is configured for streaming mode, the line interface will receive data continuously with no indicators as to where the start or end of frames are occurring. Furthermore the start of a frame does not necessarily begin in any particular byte lane (unlike LocalLink, which requires it to start in the most-significant lane), but it is expected to follow immediately after the end of the previous GFP frame, with no intervening invalid bytes. Many of the signals used for LocalLink are therefore not used in streaming mode, such as U_LINE_SOF_N, U_LINE_EOF_N, U_LINE_REM (since the starting and ending position of a frame is unknown), and U_LINE_SRC_DSC_N. Data enters the line interface on U_LINE_DATA, but stalls can be induced by either the core or the user with the deassertion of U_LINE_DST_RDY_N or U_LINE_SRC_RDY_N, respectively. However, note that user-induced line stalls may result in stalls on the output system interface.

The streaming mode data is parsed by the UNMAP core using the frame delineation algorithm defined in the *ITU-T GFP Specification*. This method initially performs a CRC-CCITT on all alignment permutations of the line data to find the core header ("hunt"), and then uses the length field of the found frame indicated in the header to find all subsequent frame boundaries, linked-list style. The implementation of this feature uses significant logic resources, so there exists an option to initially hunt for idles only (not data), since idles always look the same (no CRC-CCITT is necessary).

The type of frame delineation process is selected through the Xilinx CORE Generator GUI. The two options which enable the UNMAP core to perform synchronization are Parallel Hunting and Idle Only Hunting. In both of these configurations, the UNMAP core will attempt to perform synchronization as described above. The status of the UNMAP core synchronization is presented to the user on U_SYS_STATUS[2:0]. For information about generating the UNMAP core, see Chapter 4, "Generating the Core." For details about the U_SYS_STATUS bus, see Chapter 3, "Core Architecture."

Sample Operation

Figure 6-6 illustrates the operation of the streaming interface. In this case, there is no relationship between the start of a frame and the byte lanes. As shown, data words are packed together, and data words within a frame may be spread across multiple input cycles. The UNMAP core will determine the start of frame based on the *ITU-T GFP*

Specification synchronization operation, and will align the data on the system interface automatically.

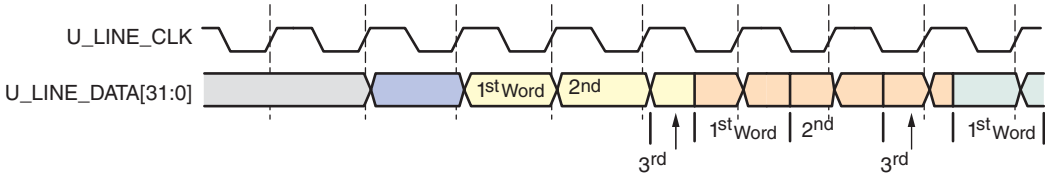

Figure 6-6: **UNMAP Streaming Interface**

Operating the UNMAP Line Interface with LocalLink Data

If the UNMAP core is configured for the LocalLink protocol on the line interface, the line expects to receive data with indicators as to where the start and end of frames are occurring, and the data must be aligned to the most significant byte lane of the input word.

When the user logic asserts `U_LINE_SOF_N` the frame has begun, and when it asserts `U_LINE_EOF_N` the frame has ended.

All valid frame cycles (including the cycles beginning with `U_LINE_SOF_N` and ending with `U_LINE_EOF_N`) must be qualified by the assertion of both `U_LINE_SRC_RDY_N` from the user, and `U_LINE_DST_RDY_N` from the core. If either of those signals are deasserted, both the UNMAP core and the line side user logic must consider that cycle to be a stall, and the user must hold the value on all line control and data signals until both `U_LINE_DST_RDY_N` and `U_LINE_SRC_RDY_N` are asserted.

All valid non-stalled frame cycles must have valid data on `U_LINE_DATA`. The data on the line interface is the entire GFP frame, including the core header, payload headers, client payload information field (CPIF), and the FCS (if present).

When the end of a frame is reached, the user must assert the `U_LINE_EOF_N` signal with the last frame word. The `U_LINE_REM` bus is used to indicate how many bytes of the last frame word are valid; the number of valid bytes is `U_LINE_REM` + 1. The `U_LINE_REM` bus is only valid when `U_LINE_EOF_N` is asserted and will be ignored otherwise. For example, if the interface is 8 bytes wide and `U_LINE_EOF_N` is asserted with a `U_LINE_REM` of 6, then 7 bytes are valid and `U_LINE_DATA[63:8]` contains the final word.

It is possible for the user to indicate the GFP frame has an error. If `U_LINE_SRC_DSC_N` is asserted with `U_LINE_EOF_N`, then the data associated with the errored GFP frame will have `U_SYS_SRC_DSC_N` asserted with it on the system interface.

When using LocalLink mode on the line interface, the frame delineation algorithm defined in the *ITU-T GFP Specification* is not used, since the user logic or upstream devices must have performed it to have known where the start and end of frames are at. The type of frame delineation process is selected through the CORE Generator GUI. The option which enables the UNMAP core to not perform synchronization is No Hunting. Because the frame delineation algorithm is not implemented in this mode, the UNMAP core will require significantly fewer resources. In this case, the UNMAP core is always synchronized (since it is always ready to accept data), and therefore `U_SYS_STATUS[2:0]` should be ignored. For details about generating the UNMAP core, see Chapter 4, "Generating the Core."

Sample Operation

Figure 6-7 illustrates the operation of the line interface using LocalLink signaling. In this case, there is a fixed relationship between the start of a frame and the byte lanes. As shown,

data words are not packed together, and instead each new frame will start on a new data word. The UNMAP core will not perform the *ITU-T GFP Specification* synchronization operation. Instead, the `U_LINE_SOF_N` and `U_LINE_EOF_N` signals are used to indicate frame boundaries, and `U_LINE_REM` indicates the number of valid bytes in the last data word. All invalid data words at the end of the frame are discarded.

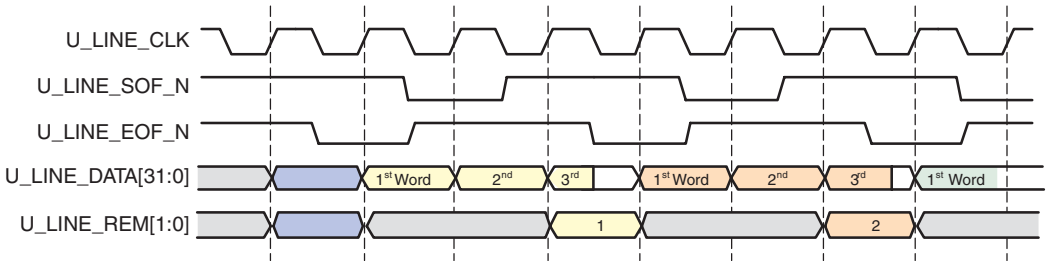

Figure 6-7: **UNMAP Line Locallink Interface**

Accessing Control and Status Registers

The host interface is an optional interface to access control and status registers. The MAP and UNMAP core each have unique host interfaces. If the user requires only default control and does not require access to these registers, this interface can be removed to reduce the resource usage of the core. The host interface conforms to the 32-Bit Device Control Register (DCR) Bus Slave described in the IBM specification *SA-14-2525-00*.

Register Space

A complete listing of the GFP register map is provided in Chapter 3, "Core Architecture." The MAP core registers are subdivided into two parts; general and channel-specific registers. The MAP core may have up to 10 unique channels, each corresponding to a particular channel id (CID) and each permitting the GFP frame to be sent with different fields and options. The UNMAP core supports only general registers.

Address Space

The MAP and UNMAP cores must have their own unique address index (if the two cores share a common DCR bus), which enables each core to be addressed individually on a shared DCR bus. The address index is configured in the CORE Generator GUI. In the case of the MAP core, the number of bits in the address index is determined by the number of channels implemented. If a one channel configuration is chosen, the address index is the upper 6 bits of `M_HOST_ADDR`; if 2 or 3 channels are chosen, the upper 5 bits are used; if 4 through 7 channels are chosen, the upper 4 bits are used; and if 8 or more are chosen, the upper 3 bits are used. Note that the CORE Generator GUI will report the base address to

the user based on the address index and number of channels selected. Figure 6-8 shows the four possible options.

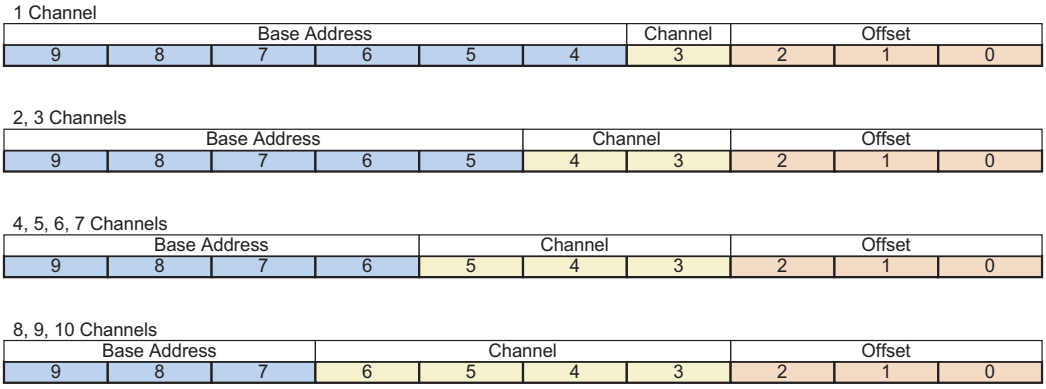

Figure 6-8: **MAP Register Space**

As shown, the MAP register space is divided into an address index, channel, and offset. The address index is set through the CORE Generator GUI. The channel address indicates either the general register space (0x0), or the channel number (CHAN0 is accessed by 0x1, CHAN1 is accessed by 0x2, and so forth). The offset accesses the individual registers within the channel, as described in Chapter 3, "Core Architecture." The user should only access valid registers as defined by the register map.

For example, if a one channel configuration is chosen, and the address index is 0x3C (6 bits; 111100), then the general registers are accessed by the address range 0x3C0 to 0x3C7, and the channel-specific registers for CHAN0 are accessed by address range 0x3C8 to 0x3CF. If ten channels are chosen, and the address index is 0x5 (3 bits; 101), then the general registers are accessed by the address range 0x280 (101 0000 000) to 0x287 (101 0000 111), and the channel-specific registers are accessed by address range 0x288 (101 0001 000) to 0x2D7 (101 1010 111). The following table partially lists the addresses of the individual registers, using this address index of 0x5 and a ten channel configuration.

Table 6-5: **MAP Register Space Example for 10 Channels and a Base Address of 5**

M_HOST_ADDR Address	Register Accessed
0x000-0x27F	Free to User
0x280	GFP_VERSION
0x281	GFP_CTRL
0x282	GFP_ERR
0x283	GFP_INTMASK
0x284	GFP_INT
0x285-0x287	Reserved
0x288	CHAN0_GFP_REGISTER_A
0x289	CHAN0_GFP_REGISTER_B
0x28A-0x28F	Reserved
0x290	CHAN1_GFP_REGISTER_A

M_HOST_ADDR Address	Register Accessed
0x291	CHAN1_GFP_REGISTER_B
0x292-0x297	Reserved
.....	.....
0x2D0	CHAN9_GFP_REGISTER_A
0x2D1	CHAN9_GFP_REGISTER_B
0x2D2-0x2D7	Reserved
0x2D8-0x2FF	Reserved
0x300-0x3FF	Free to User

The UNMAP address map has only general registers. The user should select a unique address in index if the MAP and UNMAP core share the same DCR bus; the upper bits of the UNMAP address index should not match the MAP address index. (The CORE Generator GUI will automatically ensure that a unique address index is selected for each core.) Figure 6-9 displays the UNMAP address map.

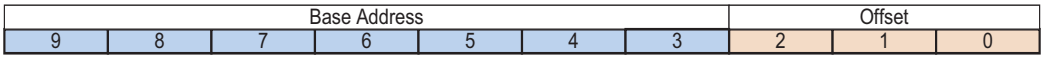

Figure 6-9: **UNMAP Register Space**

Operating the Host Interface

To write to the MAP control registers, the DCR master asserts `M_HOST_WR_EN` with the address on `M_HOST_ADDR` and the data on the `M_HOST_WR_DATA` bus. The DCR slave, in this case the GFP host interface, will then have 16 clock cycles to acknowledge the transaction by asserting `M_HOST_ACK` before the master times out.

To read from the MAP status registers, the DCR master asserts `M_HOST_RD_EN` high with the address on `M_HOST_ADDR`. The DCR slave, in this case the GFP host interface, will then have 16 clock cycles to acknowledge the transaction with `M_HOST_ACK` before the master times out. The addressed data will appear on `M_HOST_RD_DATA` coincident with the `M_HOST_ACK` assertion.

All register accesses to a valid address index will be acknowledged by the assertion of `M_HOST_ACK`. Any register access to an invalid address index will not be acknowledged (a time-out counter may be necessary to guard against commands that do not return a response).

The host interface has an (optional) independent clock, `M_HOST_CLK`. If the host clock and core clock are asynchronous to each other, the core automatically synchronizes the clock domains.

The GFP UNMAP host interface operates in an identical fashion to the GFP MAP host interface. Note that the DCR interface requires big-endian bit notation, whereas the GFP host interface uses little-endian bit notation. See Appendix C, "Packet and Control Symbol Format" for a mapping of DCR interface signals to the GFP host interface signals.

Sample Operation

Figure 6-10 shows and example of reading and writing data via the host interface. The core in the example is configured for three channels and an address index of 0x18. In the first transaction the DCR master drives the ADDR bus with the address of the GFP_VERSION register, which has an offset of 0x0 from the base address. The DCR master places the data (0x8000_0000) on the WR_DATA bus and asserts WR_EN. The data from the WR_DATA bus then gets placed on the RD_DATA bus asynchronously through the GFP core in order to allow for daisy chaining of DCR slave devices. This transaction places the core into a software reset to enable re-programming of the registers. According to the DCR specification, the DCR slave must respond with an ACK within one to sixteen cycles of the assertion of the WR_EN signal.

The DCR master then drives the WR_DATA, ADDR, and WR_EN to values that do not correspond to a valid transaction (this causes the host interface to deassert the ACK signal). Three clock-cycles after the WR_EN was deasserted, the DCR master drives the WR_DATA bus with data (0x0000_A342), the ADDR bus with the address (0x309), and asserts the WR_EN signal. This address corresponds to register B channel 0 (CHAN0_GFP_REGISTER_B). This write operation sets the spare field to 0xA3, the channel id (alias) to 0x42, and disables sending loss of client signal and loss of character synchronization on the CSF timer event.

The third transaction is a read of the GFP_VERSION register. The DCR master drives the ADDR bus and assert the RD_EN to initiate a read transaction. The data on the read data bus (RD_DATA) is valid when the GFP core acknowledges the read by asserting ACK. After the GFP core is no longer selected through the ADDR bus and the RD_EN signal, the RD_DATA bus removes the register value being read and places the value on the WR_DATA onto the RD_DATA bus. The value read from the MAP GFP_VERSION register shows the configuration of the core when it was created (the MAP version value will depend on the configuration of the GFP core).

In an actual system, this transaction would be followed by disabling the software reset by writing 0x0000_0000 to the GFP_VERSION register.

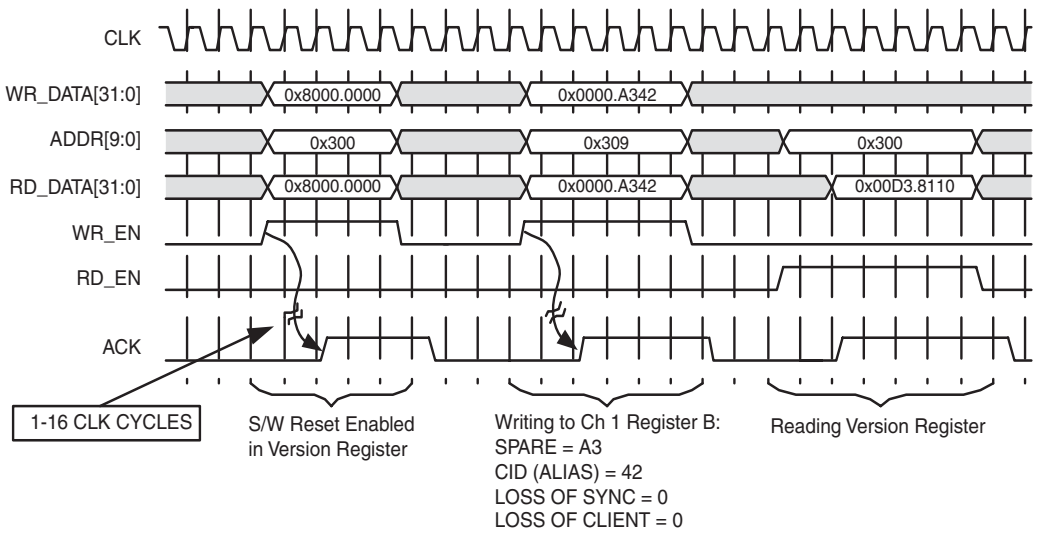

Figure 6-10: **Host Interface Waveform**

Interrupts

In addition to the DCR signals, the MAP core can also drive the interrupt signal M_HOST_INT, which may be used by the user to implement interrupt handling routines. The interrupt will occur if one of the error bits in the register GFP_INT is set and the corresponding mask bit in GFP_INTMASK is 1. The interrupt can be cleared by performing a write operation to GFP_INT. For example, if the MAP core must insert a 10B_ERR word in the superblock, then the bit GFP_INT[1] will be set to 1. If GFP_INTMASK[1] is 1, then M_HOST_INT will be asserted. If the user then writes 0x00000001 to address {base_addr, 000...0, 100}, the interrupt will be cleared and M_HOST_INT will be deasserted.

Interrupts for the UNMAP core operate identically to the MAP core.

Chapter 7

Constraining the Core

The GFP core requires the specification of timing constraints to meet the performance requirements. These constraints are provided in a user constraints file (.ucf).

For proper implementation results, a .ucf file containing these original, unmodified constraints must be used when a design is run through the Xilinx tools. For additional details on the definition and use of a .ucf file or specific constraints, see the *Xilinx Libraries Guide* and/or *Development System Reference Guide*.

Required Constraints

Timing Constraints

Timing constraints are crucial for proper operation. The following constraints are provided with the GFP core, and the user can modify these constraints to meet their system requirements. However, the user is responsible for ensuring that any modification to these constraints does not result in paths which are unconstrained.

Timenames for Clocks

The following constraints are for the MAP and UNMAP core clocks, and are always required.

NET “M_CLK” TNM_NET = “M_CLK”;

NET “U_CLK” TNM_NET = “U_CLK”;

The following constraints are for the MAP and UNMAP host clocks, and are only required if the host interface and host clock are present.

NET “M_HOST_CLK” TNM_NET = “M_HOST_CLK”;

NET “U_HOST_CLK” TNM_NET = “U_HOST_CLK”;

Timespecs for Clocks

The following constraints are for the MAP and UNMAP core clocks, and are always required. Note the generated GFP core may have different timing constraints than the examples provided below.

TIMESPEC “TS_M_CLK” = PERIOD “M_CLK” 10.0 ns HIGH 50 %;

TIMESPEC “TS_U_CLK” = PERIOD “U_CLK” 10.0 ns HIGH 50 %;

The following constraints are for the MAP and UNMAP host clocks, and are only required if the host interface is present.

TIMESPEC “TS_M_HOST_CLK” = PERIOD “M_HOST_CLK” 10.0 ns HIGH 50 %;

TIMESPEC “TS_U_HOST_CLK” = PERIOD “U_HOST_CLK” 10.0 ns HIGH 50 %;

These constraints specify the clock frequency and duty cycle of the clock signal.

Area Constraints

Area constraints provide a number of benefits to the user, including clock region management, increased performance, and/or increased resource utilization. Area constraints are only required in certain configurations, and will be appropriately generated by the CORE Generator system as required. The following are examples of the area constraints provided with the GFP core. Note that these constraints can be modified by the user to meet their requirements. For example, the number of clock regions can be adjusted depending on the target device, and the location of the clock regions can be adjusted to provide optimal placement for a given implementation.

MAP Core Area Constraints

INST “gfp_map/*” AREA_GROUP = AG_gfp_map;

AREA_GROUP “AG_gfp_map” RANGE = CLOCKREGION_X0Y1, CLOCKREGION_X0Y2, CLOCKREGION_X1Y1, CLOCKREGION_X1Y2;

UNMAP Core Area Constraints

INST “gfp_unmap/*” AREA_GROUP = AG_gfp_unmap;

AREA_GROUP “AG_gfp_unmap” RANGE = CLOCKREGION_X0Y4, CLOCKREGION_X0Y3, CLOCKREGION_X1Y1, CLOCKREGION_X1Y3;

Optional User Constraints

It is recommended to add additional constraints to cover other logic implemented by the user. While the .ucf file provided with the core is designed to completely constrain the Xilinx GFP core itself, it may not adequately constrain user-implemented logic interfaced to the core. This is the responsibility of the user.

Special Design Considerations

This chapter defines the GFP core design considerations.

I/O Pin Placement

Because the GFP core interfaces are typically not routed to external I/O pins, but instead interface to other logic blocks within the FPGA, there are no pinout restrictions or I/O timing constraints associated with the core.

Clocking

The only clocking requirement for both the MAP and UNMAP cores is that all clock pins (*_CLK) be sourced by dedicated clock buffers (there are no clock buffers instantiated in the cores). The user has the option to connect all core clocks together, or have all independent clocks. If the host interface is present, and the host clock is asynchronous from the core clock, then two global clocks are required per core. If the host interface can be shared with the core clock, or if the host interface is not present, then only one clock is required per core. Additionally, the MAP and UNMAP core clocks can be shared, depending on the user application.

The core does not instantiate any clock buffers internal to the core. Instead, the example design provided with the core instantiates a single global clock buffer, and connects all clocks together. This is only one possible configuration, and can be easily changed by the user. No DCMs or other clock management strategies are implemented in the provided example design, but this should not be ignored by the user. A carefully designed clocking strategy is crucial to the success of any high-speed design, and should receive careful attention. Please contact your Xilinx Field Application Engineer for assistance in this area. See the *GFP Getting Started Guide* for details about the example design.

Common Use Cases

The GFP core can be used in many applications, enabling efficient transport of LAN/SAN protocols over SONET/SDH networks. Two common applications utilizing Xilinx IP cores and reference designs are described below.

Frame-Mapped Mode (Ethernet)

Figure 8-1 displays a typical application of transmitting Ethernet over SONET/SDH. This example utilizes the Xilinx 1000BASE-X PCS/PMA plus the Xilinx 1-Gigabit Ethernet MAC for the client interface. The user logic interfaces between the MAC and the GFP core, accumulating frames of data on a per-channel basis. The GFP core is configured in frame-

mapped mode, with linear extension headers enabled, allowing the GFP core to time-multiplex many channels into a single GFP stream. The GFP core interfaces to the Xilinx SPI-4.2 core, providing an interface to an external framer.

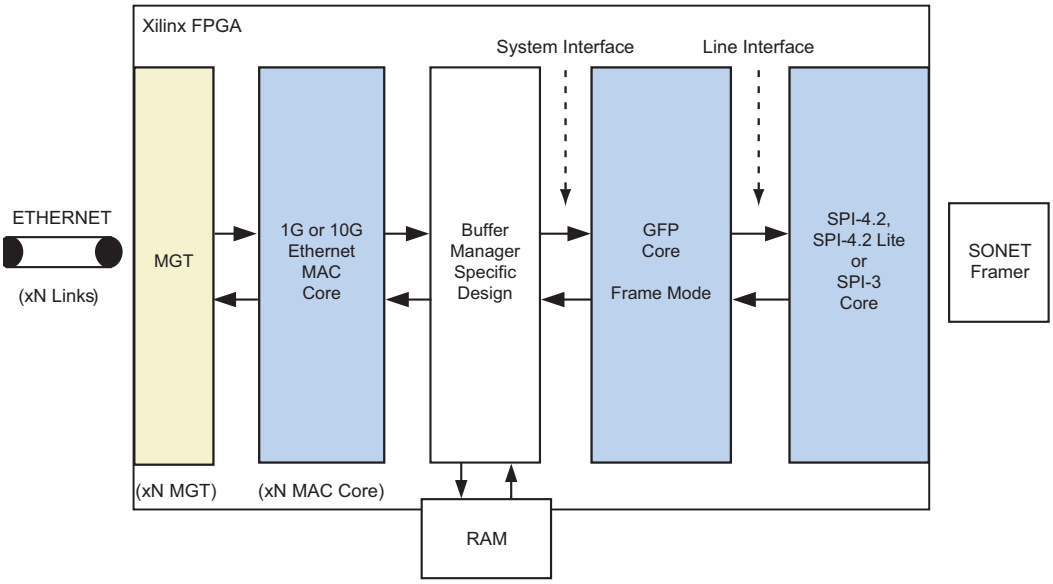

Figure 8-1: **Transmitting Ethernet over SONET/SDH**

Transparent Mode (Fibre Channel)

Figure 8-2 illustrates a typical application of transmitting Fibre Channel over SONET/SDH. This example utilizes Xilinx Application Note XAPP759 Configurable PCS (CPCS) for the client interface. The CPCS can be configured to support a variety of protocols, including Fibre Channel, ESCON, FICON, and *N* Gigabit Ethernet MAC cores. In this case, the CPCS design is configured for Fibre Channel. The user logic interfaces between the CPCS and the GFP core, performing Fibre Channel flow control (*spoofing*), as well as rate adaptation and minimal buffering on a per-channel basis.

The GFP core is configured in transparent mode, with linear extension headers enabled, allowing the GFP core to time-multiplex many channels into a single GFP stream. The GFP core interfaces to the Xilinx SPI-4.2 core, providing an interface to an external framer.

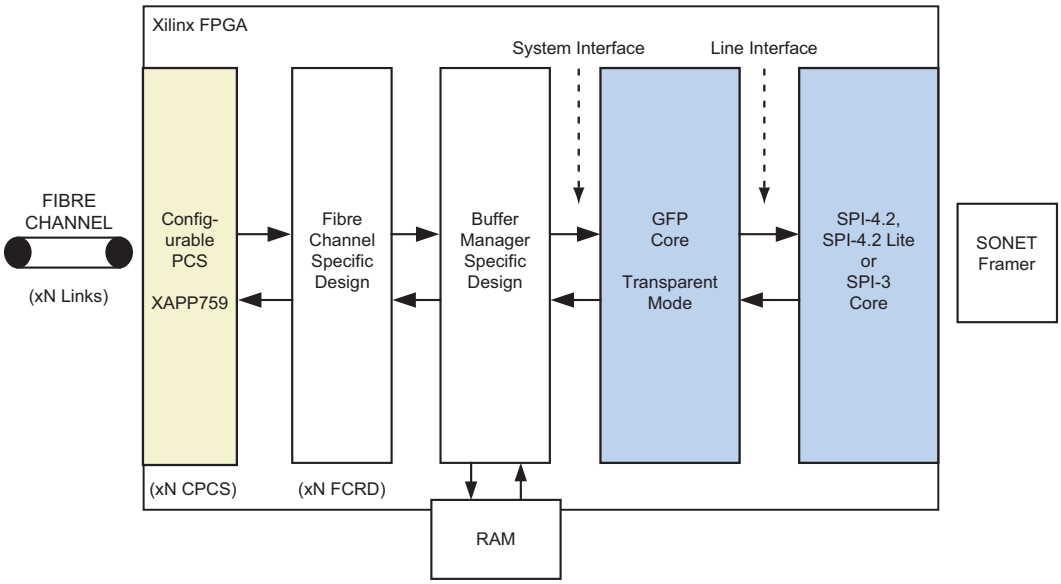

Figure 8-2: **Transmitting Fibre Channel over SONET/SDH**

Chapter 9

Simulating and Implementing Your Design

The GFP core is provided as a Xilinx technology-specific netlist and simulation model. This chapter provides instructions for simulating and implementing the GFP core in your design.

Functional Simulation

Functional simulation of the GFP core is performed with the provided simulation models (UniSim models). The simulation models provide cycle-accurate simulations for use in the verification of the user's application. The GFP core has been verified with Mentor Graphics ModelSim® PE/SE/EE and NC-Sim simulators. While other simulation tools can be used to simulate the core, they have not been tested and functionality cannot be guaranteed. Before attempting functional simulation, ensure that the simulator environment is properly configured.

1. Compile the Xilinx UniSim libraries (if not already compiled). For details, see Xilinx Answer Record 15338.
2. Compile the simulation model, user application, and user test environment. An example functional simulation script is provided with the example design, which compiles the example design and demonstration test bench. The user may use this script as an example for creating their test environment. For details about the functional simulation script, see the *GFP Getting Started Guide*.

Timing Simulation

Timing simulation of the GFP core is performed on the post-par simulation model after the core and user design are implemented through the Xilinx tools. This simulation will provide not only a cycle-accurate simulation, but also model how the design will operate in hardware. The GFP core has been verified with Mentor Graphics ModelSim PE/SE/EE and NC-Sim simulators. While other simulation tools can be used to simulate the core, they have not been tested and functionality cannot be guaranteed. Before attempting timing simulation, ensure that the simulator environment is properly configured.

1. Compile the Xilinx SIMPRIM libraries (if not already compiled). For details, see Xilinx Answer Record 15338.
2. Run the design through the Xilinx tool flow. An implement script is provided with the example design. The user may use this script as an example for creating their environment. For details about the implementation script, see the *GFP Getting Started Guide*.

3. Compile the post-par simulation model. An example timing simulation script is provided with the example design, and may be used as an example for creating the user test environment. For details about the timing simulation script, see the *GFP Getting Started Guide*.

Synthesis

Synthesis of Example Design

Synthesis of the example design is supported by XST and Synplify. While other synthesis tools may be used to synthesize the example design, they have not been tested and functionality can not guaranteed. For detailed use of the example design, see the *GFP Getting Started Guide*.

XST

Before synthesizing with XST, be sure that the Xilinx environment is properly configured for use. A sample synthesis script is provided in the implement directory and can be used as an example for synthesizing the user design.

1. Create an XST project file or open the ISE GUI.
2. Add the necessary user source files to the project file or ISE GUI. If creating a project file, also add the unisim_comp.v[hd] file located in the <*Xilinx Install Path*>/`{vhdl|verilog}/src/ise/directory`. This file is included automatically when using the ISE GUI.
3. Synthesize the user application.
   - If using the Project Navigator ISE environment, double-click Synthesize-XST in the Processes for Source window. Set the HDL language to VHDL or Verilog, the results directory and the part being used.
   - If the command line mode is being used, at the prompt, start an XST shell session by typing xst at the prompt and hitting enter. Synthesize the design by calling the XST run command to process the files in the project file.
   - For additional options that can be set to further customize synthesis of the user design, see the XST section of the Xilinx development tools manual, located at http://www.xilinx.com/documentation.

Synplify

Before synthesizing with Synplify, make sure that the Synplify environment is properly configured for use. A sample synthesis script is provided in the implement directory, which can be used as an example for the synthesizing the user design.

1. Create a Synplify project file.
2. Add the necessary user source files to the project file.
3. Select target device and speed grade.
4. Synthesize the user application.

Xilinx Tool Flow

This section provides an overview of the Xilinx tool flow and discusses how to implement the GFP core and the user design with the Xilinx implementation tools. Detailed

information about Xilinx tools can be found in the *Xilinx Development System Reference Guide*.

Before executing the Xilinx tool flow, a user design netlist must be generated where the GFP core is instantiated and all required constraints must be set in the user constraints file (.ucf). See Chapter 7, "Constraining the Core" for information about constraining the user design.

Example Design Script

An implementation script is provided with the example design to execute all the commands described below. This script can be used as an example of how to run the Xilinx tools with the GFP core. For details about the example design, see the *GFP Getting Started Guide*.

NGDBuild

Run ngdbuild to translate and merge the various source files of a design into a single NGD design database.

An example of the ngdbuild command is provided below:

ngdbuild -p `<part> <component_name>_top`

The output of ngdbuild will be `<component_name>_top.ngd`.

Mapping the Design

To map the logic gates of the user's design (previously written to an NGD file by ngdbuild) into the CLBs and IOBs of the physical device, the map command must be executed. The map command writes out this physical design to an NCD file. An example of the map command is provided below:

```
map -o mapped.ncd <component_name>_top.ngd
```

The map command outputs a `mapped.ncd` and `mapped.pcf`.

Place and Route

To place and route the user's design logic components (mapped physical logic cells) contained within a NCD file based on the layout and timing requirements specified within the physical constraints file (PCF), the par command must be executed. An example of the par command is provided below:

```
par mapped.ncd routed.ncd
```

The par command outputs `routed.ncd` file that contains the placed and routed design.

Static Timing Analysis

To evaluate timing closure on a design and create a timing report file (TWR) derived from static timing analysis of the physical design file (NCD), the `trce` command must be executed. The analysis is typically based on constraints included in the optional physical constraints file (PCF). An example of the `trce` command is provided below:

```
trce -e 10 routed.ncd mapped.pcf -o routed.twr
```

The `trce` command outputs a routed.twr file, which contains timing analysis of the placed and routed design based on the user constraints.

Timing Simulation

After the user design is functionally correct and meets all timing constraints, it is recommended to perform back-annotated timing simulation to verify that the entire user design will function correctly before the user tests their design in hardware. The `netgen` command is used to generate a post-par simulation model, which includes all timing information. An example of the netgen command is provided below:

```
netgen -sim -ofmt <vhdl | verilog> routed.ncd
```

The netgen command outputs routed.v[hd] and routed.sdf files, which allow the user to run timing simulation.

Generating a Bitstream

To create the configuration (BIT) file based on the contents of a physical implementation file (NCD), the bitgen command must be executed. The BIT file defines the behavior of the programmed FPGA. An example of the bitgen command is provided below:

bitgen -w routed.ncd

Note the user should take care in setting the required bitgen options, including selection of the startup clock. See the *Development System Reference Guide* for details.

XILINX®

Appendix A

Core Directory Structure

Appendix A defines the core directory structure and associated file contents. Figure A-1 illustrates the directory structure, and Table A-1 provides a description of each file in the core directories.

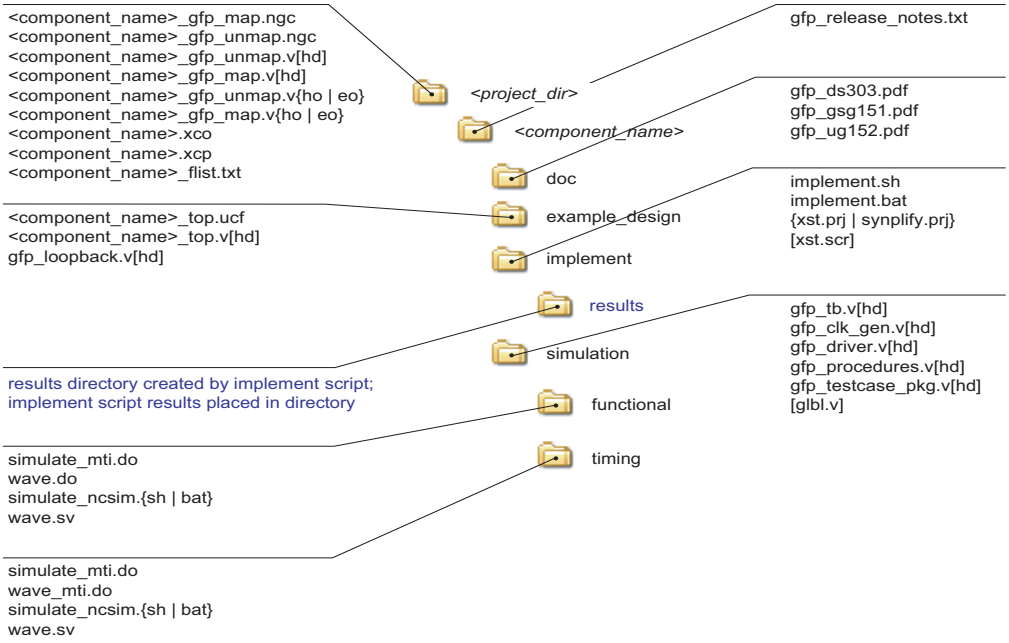

Figure A-1: **GFP Directory Structure**

Table A-1: **Directory Files**

Name	Description
CORE Generator Project Files (`<project_dir>`)	
<component_name>_gfp_map.ngc	MAP core top-level netlist
<component_name>_gfp_map.v[hd]	MAP core VHDL or Verilog simulation model
<component_name>_gfp_map.v{ho | eo}	MAP core VHDL or Verilog instantiation template
<component_name>_gfp_unmap.ngc	UNMAP core top-level netlist

Table A-1: **Directory Files *(Continued)***

Name	Description
<component_name>_gfp_unmap.v[hd]	UNMAP core VHDL or Verilog simulation model
<component_name>_gfp_unmap.v{ho | eo}	UNMAP core VHDL or Verilog instantiation template
<component_name>.xco	CORE Generator project-specific option file
<component_name>_flist.txt	List of files delivered with core
Release Notes (`<project_dir>/<component_name>`)	
gfp_release_notes.txt	*GFP Release Notes*
Documentation (`<project_dir>/<component_name>/doc`)	
gfp_ds303.pdf	*GFP Data Sheet*
gfp_gsg151.pdf	*GFP Getting Started Guide*
gfp_ug152.pdf	*GFP User Guide*
Implementation Script Files (`<project_dir>/<component_name>/implement`)	
implement.{sh | bat}	LINUX or DOS implementation script
synplify.prj	Synplify synthesis script
xst.scr	XST synthesis script
xst.prj	XST synthesis project file
Results Directory and Files (`<project_dir>/<component_name>/implement/results`)	
results directory created by implement script; results of implement script placed in results directory	
Example Design Files (`project_dir>/<component_name>/example_design`)	
<component_name>_top.v[hd]	GFP VHDL or Verilog example design top-level
gfp_loopback.vhd	GFP line interface loopback
<component_name>_top.ucf	GFP core user constraints file for the example design
Demonstration Test Bench Files (`<project_dir>/<component_name>/simulation`)	
gfp_clk_gen.v[hd]	GFP clock generator
gfp_driver.v[hd]	GFP driver module that generates data
gfp_procedures.v[hd]	GFP procedure calls used by the driver

Table A-1: **Directory Files *(Continued)***

Name	Description
gfp_tb.v[hd]	GFP demonstration test bench top-level module
gfp_testcase_pkg.v[hd]	GFP package file containing definitions of constants used by the simulation
Functional Simulation (`<project_dir>/<component_name>/simulation/functional`)	
simulate_mti.do	ModelSim simulation script for compiling, loading, and running the demonstration test bench
wave.do	ModelSim script for loading signals into the waveform viewer
simulate_ncsim.{sh | bat}	NC-Sim simulation script for compiling, loading, and running the demonstration test bench
wave.sh	NC-Sim script for loading signals into the waveform viewer
Timing Simulation (`<project_dir>/<component_name>/simulation/functional`)	
simulate_mti.do	ModelSim simulation script for compiling, loading, and running the back-annotated timing netlist and demonstration test bench files
wave.do	ModelSim script for loading signals into the waveform viewer
simulate_ncsim.{sh | bat}	NC-Sim simulation script for compiling, loading, and running the back-annotated timing netlist and demonstration test bench files
wave.sh	NC-Sim script for loading signals into the waveform viewer

XILINX®

Appendix B

Core Verification

The GFP core has been verified in a proprietary Xilinx test environment utilizing a Xilinx developed bus functional model (BFM). Using this detailed BFM, the GFP core has been tested with a variety of traffic that represents regular GFP traffic as well as erroneous traffic that stresses key protocol rules and error reporting/recovery capability.

While an exhaustive list of the test scenarios is beyond the scope of this document, the following list is an example of the test conditions generated against the GFP core.

Common Conditions

- Generation of all available core configurations, including:
  - 32-bit and 64-bit interfaces
  - Enabling of frame-mapped frames, transparent frames, or mixed mode frames
  - Host interface present or missing, including different host address indexes
  - Channel-specific programming (1 to 10 channels)
  - Linear extension header support enabled or disabled
  - Features such as scrambling and CRC generation present or missing
- Transmission and reception of back to back data flow including discontinue and deassertion of ready conditions induced by the user application interface
- Transmission and reception of varying sized packets, from minimum to maximum PLI values
- Transmission and reception of frame-mapped frames, transparent frames, and both frame types when the core is configured for mixed mode operation
- Read/write and verification of every address location within the register space
- Core operation with features such as scrambling and CRC generation enabled or disabled

MAP Core Specific Verification

- Automatic idle frame generation
- Proper handling of invalid K characters in a transparent data stream
- Transmission of management frames via the system interface as well as the host interface
- Transmission of null extension header frames
- Transmission of linear frames, testing all 256 valid CIDs
- Transmission of frames with single-bit errors injected by the host interface

UNMAP Core Specific Verification

- Reception of both streaming data and LocalLink signaling on the line interface
- Reception and removal of idle frames
- Reception and removal of 65B_PAD characters from transparent frames
- Reception of management frames.
- Reception of null extension header and linear extension header frames
- Reception of frames containing single-bit and multi-bit errors

XILINX®

Appendix C

Packet and Control Symbol Format

Figure C-1 illustrates the GFP frame format.

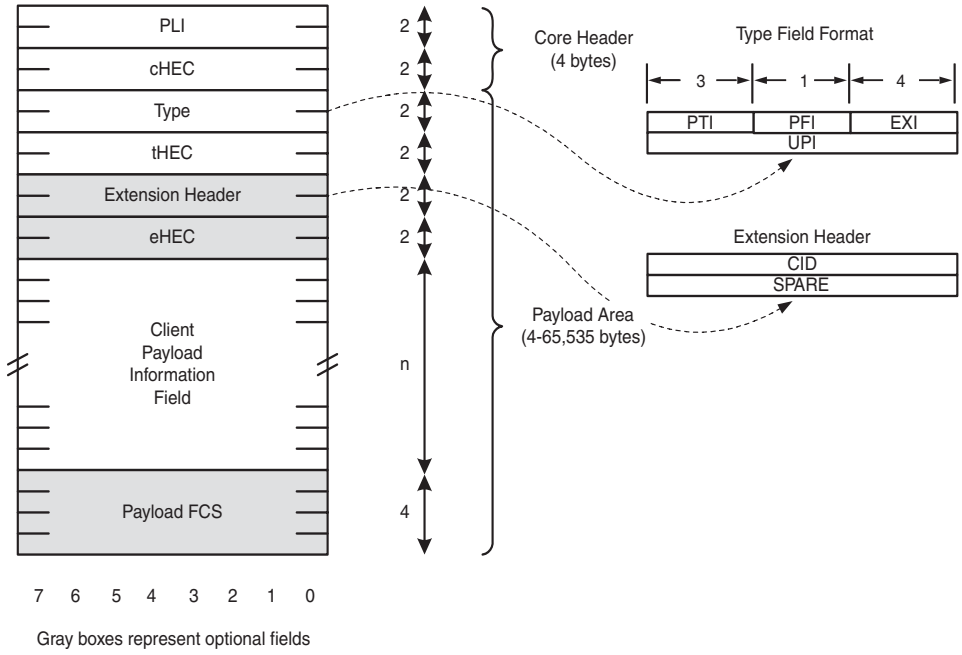

Figure C-1: **GFP Frame Format**

Table C-1 describes the frame fields.

Table C-1: **GFP Frame Field Descriptions**

Field	Name	Description
PLI	Payload Length Indicator	Indicates the number of bytes in the payload area
cHEC	Core Header Error Check	CRC-16 that protects the PLI
PTI	Payload Type Identifier	Indicates the type of GFP client frame
PFI	Payload FCS Identifier	Indicates if the optional FCS field is present

Table C-1: **GFP Frame Field Descriptions**

Field	Name	Description
EXI	Extension Header Identifier	Indicates the type of extension header
UPI	User Payload Identifier	Indicates the type of payload
tHEC	Type Header Error Check	CRC-16 that protects the PTI, PFI, EXI and UPI
Extension Header	Extension Header	Supports aggregation of several independent links onto a single transport path
eHEC	Extension Header Error Check	CRC-16 that protects the extension header
Payload	Payload	GFP payload
Payload FCS	Payload Frame Check Sequence	CRC-32 that protects the payload

Table C-2 defines the UPI for client data frames' client data frames have a PTI value of "000".

Table C-2: **UPI for Client Data Frames (PTI = 000)**

User Payload Identifier (UPI) Type Bits [7:0]	GFP Frame Payload Area
0000_0000 1111_1111	Reserved and not available
0000_0001	Frame-Mapped Ethernet
0000_0010	Frame-Mapped PPP
0000_0011	Transparent Fibre Channel
0000_0100	Transparent FICON
0000_0101	Transparent ESCON
0000_0110	Transparent Gigabit Ethernet
0000_0111	Reserved for future standardization
0000_1000	Frame-Mapped Multiple Access Protocol over SDH (MAPOS)
0000_1001	Transparent DVB ASI
0000_1010	Frame-Mapped IEEE 802.17 Resilient Packet Ring (RPR)
0000_1011	Frame-Mapped Fibre Channel FC-BBW
0000_1100	Asynchronous Transparent Fibre Channel

Table C-2: **UPI for Client Data Frames (PTI = 000)**

User Payload Identifier (UPI) Type Bits [7:0]	GFP Frame Payload Area
0000_1101 through 1110_1111	Reserved for future standardization
1111_0000 through 1111_1110	Reserved for proprietary use

Table C-3 defines the UPI for client management frames; client management frames have a UPI value of "100".

Table C-3: **UPI for Client Management Frames (PTI = 100)**

User Payload Identifier (UPI) Type Bits [7:0]	GFP Frame Payload Area
0000_0000 1111_1111	Reserved
0000_0001	Client signal fail (loss of client signal)
0000_0010	Client signal fail (loss of character synchronization)
0000_0011 through 1111_1110	Reserved for future standardization

Table C-4 shows the correlation between the IBM DCR bus interface and the GFP core interface. Note that the DCR specification utilizes big-endian notation, where the GFP host interface uses little-endian notation. Bit-swapping should be performed, if required, external to the GFP core.

Table C-4: **GFP Host Interface to DCR Bus Mapping**

GFP Host Interface	DCR Bus
HOST_RD_EN	CPU_dcrRead
HOST_WR_EN	CPU_dcrWrite
HOST_ACK	DCR_cpuAck
HOST_RD_DATA	DCR_cpuDBusIn
HOST_WR_DATA	CPU_dcrDBusOut
HOST_ADDR	CPU_dcrABus
HOST_INT	N/A

Status and Error Reporting

This section details the status and error reporting behaviors supported by the GFP MAP and UNMAP cores.

MAP Core

The MAP core provides status and error reporting using both real-time signals and the host interface. These signals and behaviors are detailed below.

M_SYS_STATUS_N

The signal `M_SYS_STATUS_N` provides real-time status and error information regarding the current state of the MAP core. This includes the insertion of management frames by the MAP core, as well as reporting of errored data written into the MAP core. See Table D-1 for details.

Table D-1: **M_SYS_STATUS_N Description**

Name	Description
M_SYS_STATUS_N[5]	**Loss of client signal management frame transmission:** This signal provides the user with an indication that the MAP core has sent a management frame indicating a loss of client signal due to the expiration of the CSF timer.
M_SYS_STATUS_N[4]	**Loss of character synchronization management frame transmission:** This signal provides the user with an indication that the MAP core has sent a management frame indicating a loss of character synchronization due to the expiration of the CSF timer.

Table D-1: **M_SYS_STATUS_N Description *(Continued)***

Name	Description
M_SYS_STATUS_N[1]	**Invalid K character received:** This signal is only valid when the MAP core is transmitting transparent mode frames. This signal provides the user with an indication that a byte presented on M_SYS_DATA (and indicated as a K character with the assertion of M_SYS_CHARISK_N) is not a valid K character (Table D-2) For a list of valid K characters, see Table 6-2, page 70.
M_SYS_STATUS_N[0]	**PLI length mismatch:** This signal is only valid when the MAP core has been configured for either framed-mapped or mixed mode operation. This signal provides the user with an indication that there is a mismatch between the specified length and the actual length written (caused by an early or late end of frame indication, or a channel ID change not at a frame boundary). The length of the frame is specified by M_SYS_LENGTH or CHANx_GFP_REGISTER_A[[15:0] (as set by CHANx_GFP_REGISTER_A[24]). If the user has discontinued the current frame by asserting M_SYS_SRC_DSC_N and M_SYS_EOF_N, this signal will not be flagged. See Table D-3 for details.

Table D-2: **M_SYS_STATUS_N[1] Behavior**

Input conditions			Response
DATA	M_SYS_CHARISK_N	M_SYS_10BERR_N	M_SYS_STATUS_N[1]
Valid K-Char	Asserted	Asserted	No Error
Valid K-Char	Asserted	De-asserted	No Error
Valid K-Char	De-asserted	Asserted	No Error
Valid K-Char	De-asserted	De-asserted	No Error
Invalid K-Char	Asserted	Asserted	No Error
Invalid K-Char	Asserted	De-asserted	Error
Invalid K-Char	De-asserted	Asserted	No Error
Invalid K-Char	De-asserted	De-asserted	No Error

M_SYS_SRC_DSC_N

This signal is used to indicate to the MAP core that the current frame contains an error and should be discontinued. The user can discontinue a frame at any time, and the MAP core

will ensure that a properly formed GFP frame is transmitted on the line interface. Asserting M_SYS_SRC_DSC_N to discontinue a frame will result in the discontinue indication M_LINE_SRC_DSC_N also being asserted at the end of the frame.

The user specifies the length of the frame to be transmitted on either M_SYS_LENGTH or CHANx_GFP_REGISTER_A. When the frame is terminated before the specified length, the frame will be padded by the MAP core. When the frame is terminated after the expected length, the frame will be truncated to the specified length and the remaining data will be discarded. When FCS insertion is built into hardware, the FCS will be inverted to further indicate that the frame has been discontinued by M_SYS_SRC_DSC_N. See Table 3 for addition conditions and responses.

M_LINE_SRC_DSC_N

This signal indicates that the encapsulated GFP frame contains an error. This signal is only asserted with M_LINE_EOF_N. M_LINE_SRC_DSC_N may be asserted under the following conditions (see Table 3 for details):

- M_SYS_SRC_DSC_N is asserted: If the user requests a discontinue on M_SYS_SRC_DSC_N, M_LINE_SRC_DSC_N will be asserted at the end of the frame.
- Incorrect length frame: If a frame is transmitted to the system interface that is not of the specified length (as set by M_SYS_LENGTH or CHANx_GFP_REGISTER_A), then the MAP core will pad or truncate the frame to the specified length. In this case, M_LINE_SRC_DSC_N will be asserted, and the FCS will be inverted (if it exists).

Table D-3: **MAP Core Response to Varying Frame Sizes and M_SYS_SRC_DSC_N**

Input Conditions		Response		
Frame Demarcation	M_SYS_SRC_DSC_N	MAP Core Behavior	M_SYS_STATUS_N[0]	M_LINE_SRC_DSC_N
Correct size frame	De-asserted	Frame passes through	De-asserted	De-asserted
Correct size frame	Asserted	FCS inverted	Asserted	Asserted
Shorter than expected frame	Asserted	Frame padded to specified length, FCS inverted	De-asserted	Asserted
Shorter than expected frame	De-asserted	Frame padded to specified length, FCS inverted	Asserted	Asserted
Longer than expected frame	Asserted	Frame truncated to specified length, FCS inverted	Asserted	Asserted
Longer than expected frame	De-asserted	Frame truncated to specified length, FCS inverted	Asserted	Asserted
CID Change before expected length	De-asserted	Frame padded to specified length, FCS inverted	De-asserted	Asserted
CID Change after expected length	De-asserted	Frame truncated to specified length, FCS inverted	Asserted	Asserted
Missing EOF, correct size frame	De-asserted	FCS inverted	Asserted	Asserted

Table D-3: **MAP Core Response to Varying Frame Sizes and M_SYS_SRC_DSC_N**

Input Conditions		Response		
Missing EOF, longer than expected frame	De-asserted	Frame truncated to specified length, FCS inverted	Asserted	Asserted
Missing EOF, shorter than expected frame	De-asserted	Frame padded to specified length, FCS inverted	De-asserted	Asserted

M_SYS_FORCE_ERR_N

This signal is used in conjunction with the MAP GFP_ERR register to insert errors into the GFP frame. The M_SYS_FORCE_ERR_N signal should be asserted for the entire frame that is to have errors introduced. When M_SYS_FORCE_ERR_N is asserted, the MAP core will insert errors as specified in the GFP_ERR register. When this signal is not asserted, no errors will be introduced into the output GFP frame. See MAP GFP_ERR and Table D-4 for details.

MAP GFP_INT

The MAP host interface interrupt signal, M_HOST_INT, indicates that the MAP core detected an interrupt. When M_HOST_INT is asserted, the user can poll the MAP GFP_INT register for the interrupts that were detected. These conditions can be selectively masked out by clearing the mask bit in GFP_INTMASK for the corresponding interrupt error condition. The user can clear the interrupts by writing to this register. See Table D-4 for details.

MAP GFP_ERR

This register is used with M_SYS_FORCE_ERR_N to selectively insert errors into the GFP frame. The user sets the corresponding bit in the register to control the type of error to introduce, and then asserts M_SYS_FORCE_ERR_N for the duration of the frame. Errors that can be inserted are listed in Table D-4.

Table D-4: **MAP GFP_INT Description**

Name	Description
GFP_INT[1]	**Invalid K character received:** This interrupt is only valid when the MAP core is transmitting transparent mode frames. It provides the user with an indication that a byte presented on M_SYS_DATA (and indicated as a K character with the assertion of M_SYS_CHARISK_N) is not a valid K character (see Table 2 for details). For a list of valid K characters, see Table 6-2, page 70.
GFP_INT[0]	**PLI length mismatch:** This interrupt is only valid when the MAP core has been configured for either framed-mapped or mixed mode operation. It provides the user with an indication that there is a mismatch between the specified length and the actual length written (caused by an early or late end of frame indication, or a channel ID change not at a frame boundary). The length of the frame is specified by M_SYS_LENGTH or CHANx_GFP_REGISTER_A[[15:0] (as set by CHANx_GFP_REGISTER_A[24]). If the user has discontinued the current frame by asserting M_SYS_SRC_DSC_N and M_SYS_EOF_N, this interrupt will not be flagged.

Table D-5: **: MAP GFP_ERR Description**

Name	Description
GFP_ERR[7]	**Payload scrambling error insertion:** To create a payload scrambling error, payload scrambling must be turned on and payload scrambling built in hardware. Depending on the data bus width, error insertion will invert the most significant bit (31 or 63) for every word inside the payload.
GFP_ERR[6]	Location of superblock CRC error: **(1) Corrupt all superblocks:** When corruption of all superblock CRCs is enabled, this will invert the CRC for every superblock in a transparent frame. The only exception to this is when a transparent frame has been padded to the specified frame length (with 65B_PAD words); superblocks that do not contain data will not have the CRC inverted. **(0) Corrupt only the first superblock:** When corruption of the first superblock CRC is enabled, this will invert the CRC for the first superblock in every frame.
GFP_ERR[5]	**Superblock CRC error:** To enable insertion of errors into the superblock CRC this register bit must be enabled. The error to be inserted is determined by the value of GFP_ERR[6]. Superblock CRC errors can only be inserted into transparent frames.
GFP_ERR[4]	**FCS error:** To create a FCS error, the FCS must be enabled and FCS generation must be built in hardware. In order to create an error, the MAP core initializes the FCS generator with 0x7FFFFFFF instead of 0xFFFFFFFF.
GFP_ERR[3]	**Core header scrambling error:** To create a core header scrambling error, header scrambling must be turned on and the header scrambler must be built in hardware. In order to create an error, the MAP core inverts the least significant bit of the PLI.
GFP_ERR[2]	**cHEC error:** To create a cHEC error, cHEC generation must be turned on and the cHEC generator must be built in hardware. In order to create an error, the MAP core inverts the least significant bit of the cHEC.

Table D-5: **: MAP GFP_ERR Description *(Continued)***

Name	Description
GFP_ERR[1]	**tHEC error:** To create a tHEC error, tHEC generation must be turned on and the tHEC generator must be built in hardware. In order to create an error, the MAP core inverts the least significant bit of the tHEC.
GFP_ERR[0]	**eHEC error:** To create a eHEC error, eHEC generation must be turned on and the eHEC generator must be built in hardware. In order to create an error, the MAP core inverts the least significant bit of the eHEC.

UNMAP Core

The UNMAP core provides status and error reporting using both real-time signals and the host interface. These signals and behaviors are detailed below.

U_SYS_STATUS_N

The U_SYS_STATUS_N bus provides real-time status information for the UNMAP core. It reports the FIFO fill level, as well as the synchronization status of the core (synchronization is only valid when the UNMAP core is configured in either idle-only hunting or parallel hunting, see "Customizing Core Features," page 46 for details.

Table D-6: **U_SYS_STATUS_N Description**

Name	Description
U_SYS_STATUS_N[4]	**System FIFO Almost Full:** Indicates that the FIFO in the system interface is almost full, and will cause the line interface to back-pressure if data is not read from the system interface.
U_SYS_STATUS_N[2]	**SYNC:** Indicates that the hardware is in a state where it has matched the specified number of cHEC matches as given in the GFP_CTRL register. Valid frames will appear on the system interface when in SYNC.
U_SYS_STATUS_N[1]	**PRESYNC:** Indicates that the hardware has found a cHEC that has matched core header and is accumulating matches until it meets the number of matches given in the GFP_CTRL register. When in PRESYNC, frames will not appear on the system interface.
U_SYS_STATUS_N[0]	**HUNT:** The UNMAP core has lost synchronization (or has come out of reset) and is currently searching for a matching cHEC. When in HUNT, frames will not appear on the system interface.

U_SYS_SRC_DSC_N

This signal is used to indicate that the current frame contains an error. When asserted, the U_SYS_ERRBUS_N signal will indicate the type of error that was detected. This signal is only valid with U_SYS_EOF_N.

U_LINE_SRC_DSC_N

When the UNMAP core is configured to be of synchronization type no hunting, this signal is used to indicate to the UNMAP core that the current frame contains an error. This signal is only valid with U_LINE_EOF_N.

U_SYS_ERRBUS_N

This signal provides real-time status and error reporting on the system interface. The following errors are reported (bits 6, 2:0) if they are enabled in the host interface (UNMAP GFP_CTRL[3:0]). The remaining bits (7, 5:3) are always reported.

Table D-7: **U_SYS_ERRBUS_N Description**

Name	Description
U_SYS_ERRBUS_N[7]	**Transparent frame did not end on a superblock boundary:** This signal indicates that the current transparent frame was not terminated on a superblock boundary. In this case, the UNMAP core will discontinue the current frame by asserting U_SYS_SRC_DSC_N and U_SYS_EOF_N.
U_SYS_ERRBUS_N[6]	**FCS error:** This signal indicates that the current frame contained an FCS error. In this case, the UNMAP core will discontinue the current frame by asserting U_SYS_SRC_DSC_N and U_SYS_EOF_N.
U_SYS_ERRBUS_N[5]	**cHEC corrected:** This signal will be asserted for one clock cycle each time a cHEC error is corrected. This signal is asserted with U_SYS_EOF_N.
U_SYS_ERRBUS_N[4]	**tHEC corrected:** This signal will be asserted for one clock cycle each time a tHEC error is corrected. This signal is asserted with U_SYS_EOF_N.
U_SYS_ERRBUS_N[3]	**eHEC corrected:** This signal will be asserted for one clock cycle each time a eHEC error is corrected. This signal is asserted with U_SYS_EOF_N.
U_SYS_ERRBUS_N[2]	**cHEC error:** This signal indicates that the current frame contained an uncorrectable cHEC error. In this case, the UNMAP core will discontinue the current frame by asserting U_SYS_SRC_DSC_N and U_SYS_EOF_N.

Table D-7: **U_SYS_ERRBUS_N Description**

Name	Description
U_SYS_ERRBUS_N[1]	**tHEC error:** This signal indicates that the current frame contained an uncorrectable tHEC error. In this case, the UNMAP core will discontinue the current frame by asserting U_SYS_SRC_DSC_N and U_SYS_EOF_N.
U_SYS_ERRBUS_N[0]	**eHEC error:** This signal indicates that the current frame contained an uncorrectable eHEC error. In this case, the UNMAP core will discontinue the current frame by asserting U_SYS_SRC_DSC_N and U_SYS_EOF_N.

U_SYS_CHARISK_N

This signal indicates the status of each byte of data on U_SYS_DATA. If U_SYS_CHARISK_N is asserted, then the corresponding byte on U_SYS_DATA is a 8b/10b K character. If U_SYS_CHARISK_N is not asserted, then the corresponding byte on U_SYS_DATA is not a K character. This signal is only valid if the byte(s) in question is valid as indicated by U_SYS_REM.

U_SYS_10BERR_N

This signal indicates the status of each byte of data on U_SYS_DATA. If U_SYS_10BERR_N is asserted, then the corresponding byte on U_SYS_DATA is an illegal 8b/10b K character (10B_ERR). If U_SYS_10BERR_N is not asserted, then the corresponding byte on U_SYS_DATA is not an illegal K character. This signal is only valid if the byte(s) in question is valid as indicated by U_SYS_REM.

U_SYS_SUPERERR_N

This signal indicates that a CRC-16 error has occurred in the current superblock. The U_SYS_SUPERERR_N signal is asserted at the end of the superblock, to enable the user to drop the errored superblock if desired.

UNMAP GFP_INT

The UNMAP host interface interrupt signal, U_HOST_INT, indicates that the UNMAP core detected an interrupt. When U_HOST_INT is asserted, the user can poll the UNMAP GFP_INT register for the interrupts that were detected. These conditions can be selectively masked out by clearing the mask bit in GFP_INTMASK for the corresponding interrupt

error condition. The user can clear the interrupts by writing to this register. See Table D-8 for details.

Table D-8: **UNMAP GFP_INT Description**

Name	Description
GFP_INT[5]	**PLI Mismatch Error:** This bit indicates that there was a mismatch in the indicated length of the frame given by the PLI and the demarcation of the frame with U_SYS_SOF_N and U_SYS_EOF_N.
GFP_INT[4]	**Superblock CRC Error:** This bit is only valid when the UNMAP core is configured in either transparent or mixed mode. This bit indicates that an error was detected in the CRC check over the current superblock. When superblock CRC error reporting is disabled in the UNMAP GFP_CTRL register, this bit does not flag an error.
GFP_INT[3]	**FCS Error:** When FCS detection is built in hardware, this bit is used to indicate that a FCS error was detected over the payload. When FCS error reporting is disabled in the UNMAP GFP_CTRL register, this bit does not flag an error.
GFP_INT[2]	**cHEC Error:** When cHEC detection is built in hardware, this bit is used to indicate that an error was detected in the core header. When cHEC detection is disabled in the UNMAP GFP_CTRL register, this bit does not flag an error. A bit error will not be reported if the cHEC bit error was corrected.
GFP_INT[1]	**THEC Error:** When tHEC detection is built in hardware, this bit is used to indicate that an error was detected in the type or tHEC fields. When tHEC detection is disabled in the UNMAP GFP_CTRL register, this bit does not flag an error. A bit error will not be reported if the tHEC bit error was corrected.
GFP_INT[0]	**eHEC Error:** When eHEC detection is built in hardware, this bit is used to indicate that an error was detected in the extension header or eHEC fields. When eHEC detection is disabled in the UNMAP GFP_CTRL register, this bit does not flag an error. A bit error will not be reported if the eHEC bit error was corrected.

UNMAP GFP_FIXERR

The UNMAP GFP_FIXERR register indicates that a detected error was corrected by the core, and also indicates the channel ID of the corrected error. This register holds the first value detected for each field until cleared by the user.

Table D-9: **UNMAP GFP_FIXERR Description**

Name	Description
GFP_FIXERR[26]	**cHEC error corrected:** When cHEC correction is built in hardware, and enabled in the UNMAP GFP_CTRL register, this bit indicates that a bit error over the core header was corrected. The corrected frame channel ID appears in GFP_FIXERR[23:16].
GFP_FIXERR[25]	**tHEC error corrected:** When tHEC correction is built in hardware, and enabled in the UNMAP GFP_CTRL register, this bit indicates that a bit error over the type or tHEC field was corrected. The corrected frame channel ID appears in GFP_FIXERR[15:8].
GFP_FIXERR[24]	**eHEC error corrected:** When eHEC correction is built in hardware, and enabled in the UNMAP GFP_CTRL register, this bit indicates that a bit error over the extension or eHEC fields was corrected. The corrected frame channel ID appears in GFP_FIXERR[7:0].
GFP_FIXERR[23:16]	**CID of corrected cHEC:** The channel ID of the first detected cHEC frame corrected.
GFP_FIXERR[15:8]	**CID of corrected tHEC:** The channel ID of the first detected tHEC frame corrected.
GFP_FIXERR[7:0]	**CID of corrected eHEC:** The channel ID of the first detected eHEC frame corrected.

XILINX®

Appendix E

Sample GFP Frames

The following table illustrates a sample data frame and the resulting GFP encapsulation for various core configurations. The sample data frame is:

0001 0203 0405 0607 0809 0A0B 0C0D 0E0F

As seen below, the resulting frame depends on the configuration options selected. In the following examples, frame-mapped frames were generated for Ethernet, and transparent mode frames were generated for Fibre Channel.

Table E-1: **First Constructed GFP Frame for Various Core Configurations**

Data Type	Ethernet	Ethernet	Ethernet	Ethernet	Ethernet	Ethernet	Fibre Channel	Fibre Channel	Fibre Channel	Fibre Channel
Extension Header[1]	Linear	Linear	Linear	Null	Null	Null	Linear	Linear	Null	Null
Payload FCS Generation	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	No
Header Scrambling	No	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes
Payload Scrambling	No	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes
Output data stream (in hex)	001C D3BD 1101 2063 1234 13C6 0001 0203 0405 0607 0809 0A0B 0C0D 0E0F 5685 00B2	B6B7 E25D 1101 2603 1236 33E2 0C63 44C5 7844 8A6F 90A6 029A 41FF 1ACF 05CD 3F51	B6B3 A2D9 0101 2310 1234 33E2 6203 4485 7849 466F 98A6 0323 C1FE 1ACF	0018 9339 1001 1352 0001 0203 0405 0607 0809 0A0B 0C0D 0E0F 5685 00B2	B6B3 A2D9 1001 1352 0003 0221 6E45 0667 4C24 C2AB C0E4 8A97 03FD 1C23	B6BF 6355 0001 1021 0001 0221 0025 0627 4C29 0EAB C8E4 8B2E	004F_B92B 1103_0021 1234_13C6 0001_0203 0405_0607 0809_0A0B 0C0D_0E0F 8D9D_ADBD CDDD_ED7D 8D9D_ADBD CDDD_ED7D 8D9D_ADBD CDDD_ED7D 8D9D_ADBD CDDD_ED7D 8D9D_ADBD CDDD_ED7D 3FD2_5B50 B2D6_B8--	B6E4_88CB 1103_0021 1236_33A6 0423_44C5 70C5_826F 90A7_12BB 41FF_1AED DAF5_925E 9066_B3CF C64F_A16B B425_2489 A0EB_2919 5CE9_F018 AEB6_3083 CEC8_3BBB 9DE4_74BA BAAE_51F3 1ACA_F877 F3BE_B422 312C_2C86 3690_9D--	0047_3823 0003_3063 0001_0203 0405_0607 0809_0A0B 0C0D_0E0F 8D9D_ADBD CDDD_ED7D 8D9D_ADBD CDDD_ED7D 8D9D_ADBD CDDD_ED7D 8D9D_ADBD CDDD_ED7D 8D9D_ADBD CDDD_ED7D 8D9D_ADBD CDDD_ED7D 3FD2_5B--	B6EC_09C3 0003_3063 0001_0265 0865_0627 44A8_06AB C8E5_9B0F 58E4_B10E AC36_F1EB AC48_2B63 F0A8_6478 E1E3_B8B1 42C1_D10A 9BB5_F587 EC8E_9BC3 3D60_3C6E B5BA_417A 004B_1AF5 E29D_E41E 616E_08--

1. All extension headers have CID = 0x12, and Spare = 0x34