Platform Specification Format Reference Manual

Embedded Development Kit EDK

EDK 10.1, Service Pack 3

XILINX®

Platform Specification Format Reference Manual
EDK 10.1, Service Pack 3

The following table shows the revision history for this document.

Date	Revision
08/20/04	Initial release for EDK 6.3i.
02/15/05	EDK 7.1i release.
04/28/05	EDK 7.1i Service Pack 1 release.
07/05/05	EDK 7.1i Service Pack 2 release.
10/24/05	EDK 8.1i release.
01/16/06	EDK 8.1 Service Pack 1 release. Updated to note obsolete cores.
06/23/06	EDK 8.2i release.
01/08/07	EDK 9.1i release.
09/05/07	EDK 9.2i release.
01/14/07	EDK 10.1 release.
09/19/08	EDK 10.1 Service Pack 3 release.

About This Guide

Guide Contents

This manual contains the following chapters:

- Chapter 1, "Introduction"
- Chapter 2, "Microprocessor Hardware Specification (MHS)"
- Chapter 3, "Microprocessor Peripheral Definition (MPD)"
- Chapter 4, "Peripheral Analyze Order (PAO)"
- Chapter 5, "Black-Box Definition (BBD)"
- Chapter 6, "Microprocessor Software Specification (MSS)"
- Chapter 7, "Microprocessor Library Definition (MLD)"
- Chapter 8, "Microprocessor Driver Definition (MDD)"
- Chapter 9, "Xilinx Board Description (XBD) Format"
- Appendix A, "Glossary"

Additional Resources

To find additional documentation, see the Xilinx website:

http://www.xilinx.com/support/documentation/index.htm.

The following table lists some of the resources you can access from this website. You can also directly access these resources using the provided URLs.

Resource	Description/URL
EDK Home	Embedded Development Kit home page, FAQ, and tips. http://www.xilinx.com/ise/embedded_design_prod/platform_studio.htm
EDK Examples	A set of complete EDK examples. http://www.xilinx.com/ise/embedded/edk_examples.htm
Tutorials	Tutorials covering Xilinx design flows from design entry to verification and debugging http://www.xilinx.com/support/techsup/tutorials/index.htm

Resource	Description/URL
Answer Browser	To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at: http://www.xilinx.com/support/mysupport.htm
Application Notes	For descriptions of device-specific design techniques and approaches, click the **Doc Type** tab on the following web page: http://www.xilinx.com/support/documentation/index.htm
Data Sheets	For device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging, click the **Doc Type** tab on the following web page: http://www.xilinx.com/support/documentation/index.htm
Problem Solvers	Interactive tools that allow you to troubleshoot your design issues: http://www.xilinx.com/support/troubleshoot/psolvers.htm
GNU Manuals	The entire set of GNU manuals may be found at: http://www.gnu.org/manual

Conventions

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

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *`design_name`*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**

Convention	Meaning or Use	Example
Italic font	Variables in a syntax statement for which you must supply values	**ngdbuild** *design_name*
	References to other manuals	See the *Development System Reference Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **bus[7:0]**, they are required.	**ngdbuild** [*option_name*] *design_name*
Braces { }	A list of items from which you must choose one or more	**lowpwr** ={**on**|**off**}
Vertical bar |	Separates items in a list of choices	**lowpwr** ={**on**|**off**}
Vertical ellipsis . . .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...	Repetitive material that has been omitted	**allow block** *block_name loc1 loc2 ... locn;*

Online Documents

The following conventions are used in this document:

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

Table of Contents

Chapter 3: Microprocessor Peripheral Definition (MPD)

Chapter 4: Peripheral Analyze Order (PAO)

Chapter 5: Black-Box Definition (BBD)

Chapter 6: Microprocessor Software Specification (MSS)

Chapter 7: Microprocessor Library Definition (MLD)

Chapter 8: Microprocessor Driver Definition (MDD)

XILINX®

Chapter 1

Introduction

EDK tools are designed to operate in a data-driven manner. There are various meta-data files that capture information, for example, about various IPs, drivers, and software libraries being used in the EDK tools. Files are also used to capture both hardware and software aspects of your design information. These are ASCII files. The set of all these meta-data formats is referred to as the Platform Specification Format or PSF.

This chapter contains the following sections:

Files

BBD - Black Box Definition

The Black Box Definition (BBD) file manages the file locations of optimized hardware netlists for the black-box sections of your peripheral design.

Refer to Chapter 5, "Black-Box Definition (BBD)," for more information.

MDD - Microprocessor Driver Definition

An MDD file contains directives for customizing software drivers.

Refer to Chapter 8, "Microprocessor Driver Definition (MDD)," for more information.

MHS - Microprocessor Hardware Specification

The Microprocessor Hardware Specification (MHS) file defines the hardware component. You supply an MHS file as an input to the Platform Generator (Platgen) tool.

Refer to Chapter 2, "Microprocessor Hardware Specification (MHS)," for more information.

MPD - Microprocessor Peripheral Definition

The Microprocessor Peripheral Definition (MPD) file defines the interface of the peripheral.

Refer to Chapter 3, "Microprocessor Peripheral Definition (MPD)," for more information.

MSS - Microprocessor Software Specification

You supply an MSS file as an input to the Library Generator (Libgen). The MSS file contains directives for customizing libraries, drivers, and file systems.

Refer to Chapter 6, "Microprocessor Software Specification (MSS)," for more information.

MLD - Microprocessor Library Definition

An MLD file contains directives for customizing software libraries and operating systems.

Refer to Chapter 7, "Microprocessor Library Definition (MLD)" for more information.

PAO - Peripheral Analyze Order

A PAO (Peripheral Analyze Order) file contains a list of HDL files that are needed for synthesis and defines the analyze order for compilation.

Refer to Chapter 4, "Peripheral Analyze Order (PAO)," for more information.

XBD - Xilinx Board Definition

An XBD file contains a definition of logical interfaces present on a board and how they are connected to the FPGA. Refer to Chapter 9, "Xilinx Board Description (XBD) Format," for more information.

File and IP Naming Rules

File and IP names must be all lower-case to ensure consistency across the following:

- OS: UNIX (case-sensitive) vs. Win (case-insensitive)
- HDL: Verilog (case-sensitive) vs. VHDL (case-insensitive)

A lower-case naming convention is used to deal with the above combinations. For example: `MYCORE_v2_1_0` and `mycore_v2_1_0` would mean two different files in UNIX, whereas in Windows, they would be the same.

Assembly of lower-level cores into the top-level are merged by name reference. Therefore, it is important that names match.

Version Scheme

Form of the version level is `X.Y.Z`

- `X` - major revision
- `Y` - minor revision
- `Z` - patch level

Version Setting for MHS and MSS

In the body of the MHS and MSS file, add the following statement:

```
PARAMETER VERSION = 2.1.0
```

The version is specified as a literal of the form 2.1.0.

Version Setting for BBD, MPD, and PAO

The version level is concatenated to the base name of the data files. The literal form of the version level is `vX_Y_Z`.

- `<ipname>_vX_Y_Z.mpd`
- `<ipname>_vX_Y_Z.bbd`
- `<ipname>_vX_Y_Z.pao`
- `<ipname>_vX_Y_Z.mdd`

Load Path

Peripheral and pcore Directory Structures

To specify additional directories, use one of the following options:

- Current directory
- Set the EDK tool **-lp** option.

EDK tools use a search priority mechanism to locate peripherals, as follows:

1. Search the pcores directory in the project directory.
2. Search *<library_path>*/*<Library Name>*/pcores as specified by the **-lp** option. The following figure shows the peripheral directory structure.

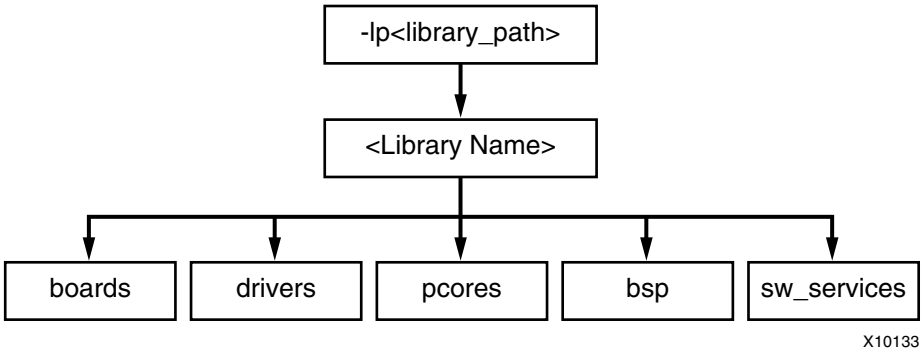

Figure 1-1: **Peripheral Directory Structure**

3. Search *XILINX_EDK*/hw/*<Library Name>*/pcores. The following figure shows the pcore directory structure.

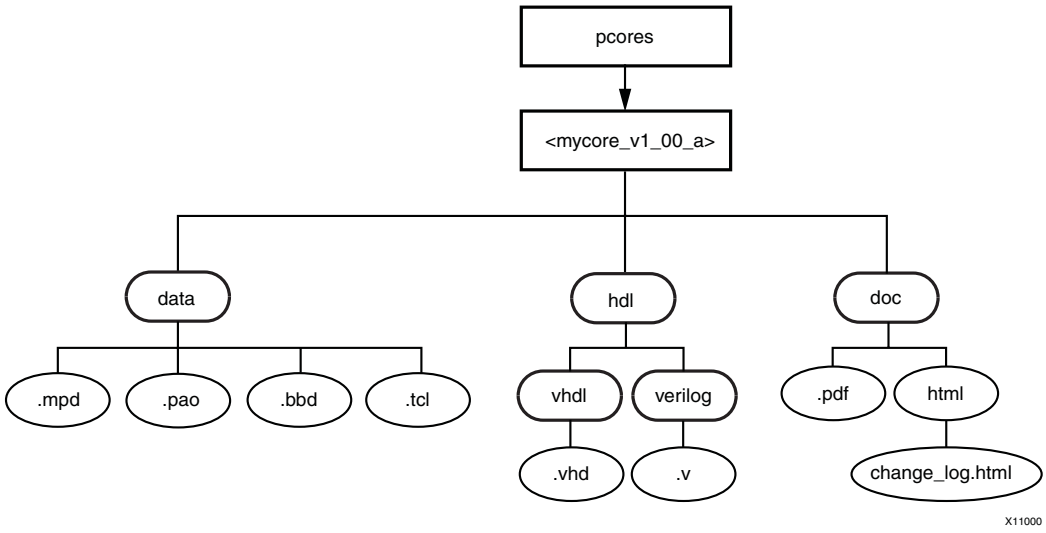

Figure 1-2: **pcore Directory Structure**

Using Versions

You can create multiple versions of your peripheral. The version is specified as a literal of the form 1.00.a. The version is always specified in lower case.

At the MHS level, use the HW_VER parameter to set the hardware version. The Platform Generator concatenates a _v and translates periods to underscores. The peripheral name and HW_VER are joined together to form a name for a search level in the load path. For example, if your peripheral is version 1.00.a, then the MPD, BBD, and PAO files are found in the following location:

<*repository_dir*>/pcores/<*ipname*>_v1_00_a/data (UNIX)

<*repository_dir*>\pcores\<*ipname*>_v1_00_a\data (PC)

Creating Your IP

How you build your own reference depends on the characteristics of your design.

Is Your IP Pure HDL?

Read about MPD and PAO files in the following chapters:

- Chapter 3, "Microprocessor Peripheral Definition (MPD)"
- Chapter 4, "Peripheral Analyze Order (PAO)"

Is Your IP Only a Black-Box Netlist?

Read about MPD and BBD files in the following chapters:

- Chapter 3, "Microprocessor Peripheral Definition (MPD)"
- Chapter 5, "Black-Box Definition (BBD)"

Is Your IP a Mixture of Black-Box Netlists and VHDL or Verilog?

Read about the MPD, BBD, and PAO files in the following chapters:

- Chapter 3, "Microprocessor Peripheral Definition (MPD)"
- Chapter 5, "Black-Box Definition (BBD)"
- Chapter 4, "Peripheral Analyze Order (PAO)"

Creating HDL Libraries for Your IP

There are two classes of design libraries: *primary* libraries and *resource* libraries.

Primary Library

The library into which the library unit resulting from the analysis of an IP is placed. A primary library contains all the primary HDL files for the IP, and is referenced in the PAO as <*ipname*>_v1_00_a.

Only primary libraries contain an MPD file; resource libraries do not.

Resource Library

A resource library contains library units that are referenced within the IP being analyzed. A resource library contains all the resource HDL files for the IP and is referenced in the PAO file as `<resource_name>_v1_00_a`.

Resource libraries must contain a PAO file to enable primary libraries access to the PAO file set from the resource library. To accomplish this from the primary library PAO, use the **`all`** keyword. For example:

```
# primary_core PAO
lib reference_lib_v1_00_a all
```

Resource Libraries and PAO Files

If a resource library defines a PAO, the language field must be present. Please refer to Chapter 4, "Peripheral Analyze Order (PAO)" for complete details.

For example:

```
# reference_lib PAO
lib reference_lib_v1_00_a file2.vhd vhdl
lib reference_lib_v1_00_a file1.vhd vhdl
```

Library File Locations

Primary and resource libraries are physically located in `<repository_dir>/pcores`.

Resource HDL File Locations

For VHDL:

`<repository_dir>/pcores/<resource_name>_v1_00_a/hdl/vhdl`

For Verilog:

`<repository_dir>/pcores/<resource_name>_v1_00_a/hdl/verilog`

Primary HDL File Locations

For VHDL:

`<repository_dir>/pcores/<ipname>_v1_00_a/hdl/vhdl`

For Verilog:

`<repository_dir>/pcores/<ipname>_v1_00_a/hdl/verilog`

Verilog Include Directories

You must use relative paths to allow project maneuverability from development platform to development platform. Use the `include compiler directive in your Verilog HDL files to insert the contents of an entire file.

The following is an example Verilog HDL file:

```
`include "global_consts.v"
`include "pcore_v1_00_a/hdl/verilog/consts.v"
```

By default, all known EDK repositories are automatically included to the calls that process Verilog:

```
<proj_dir>/pcores
$XILINX_EDK/hw/XilinxBFMinterface/pcores
$XILINX_EDK/hw/XilinxProcessorIPLib/pcores$XILINX_EDK/hw/XilinxReferen
ceDesigns/pcores
```

You need only specify include paths that are not default. User-specified paths have a higher precedence over the default paths.

Format

Use the following format:

```
vlgincdir <library> <relative path from library>
```

Restrictions

If you intend to define a library file of text macros, you must give each text macro a unique name. The IEEE document *1364-2006 Section 3.12* defines the Verilog name space.

The text macro name space is global. The text macro names are defined in the linear order of appearance in the set of input files that make up the description of the design unit.

Subsequent definitions of the same name override the previous definitions for the balance of the input files.

Use unique names, as shown below:

```
`define PCORE_V1_00_A_MASKVAL 2'b10
```

(Do *not* use `define MASKVAL 2'b10.)

When multiple vlgincdir options are in use, it is possible for the compiler to read an unwanted included file. The preferred use of text inclusion within Verilog files is to include the relative path of the pcore library in use, as shown in the example below:

```
`include "pcore_v1_00_a/hdl/verilog/consts.v"
```

XILINX®

Chapter 2

Microprocessor Hardware Specification (MHS)

The Microprocessor Hardware Specification (MHS) file defines the hardware component. An MHS file defines the configuration of the embedded processor system and includes the following:

- Bus architecture
- Peripherals
- Processor
- Connectivity
- Address space

This chapter contains the following sections:

MHS Syntax

About the Syntax

MHS file syntax is case insensitive. The current version is 2.1.0.

MHS parameter, component, instance, and signal names must be HDL, VHDL, and Verilog compliant. VHDL and Verilog have certain naming rules and conventions that must be followed.

Because the MHS stands as a neutral format on top of HDL, VHDL, and Verilog, it is possible to generate illegal VHDL or Verilog, even if the MHS is syntactically correct. You might, therefore, be violating syntax rules in either VHDL or Verilog in the downstream HDL compliant synthesis and simulation tools.

For example, it is illegal in VHDL to use an instance name that already exists as a component name. Consider the following example:

```
microblaze : microblaze
port map ( <snip> );
```

However, Verilog allows such a declaration:

```
microblaze microblaze ( <snip> );
```

It is also illegal in VHDL to declare an object (parameter, component, instance, or signal) name that already exists as the name of another object. For example, it is illegal to declare in VHDL a signal name, MYTESTNAME, and also declare an instance name of MYTESTNAME.

```
signal MYTESTNAME : std_logic;
MYTESTNAME : microblaze
port map ( <snip> );
```

However, this is legal in Verilog.

It is your responsibility to recognize the output format and comply with the rules of the HDL language.

Comments

You can insert comments in the MHS file without disrupting processing. The following are guidelines for inserting comments:

- Precede comments with the pound sign (#).
- Comments can continue to the end of the line.
- Comments can be anywhere on the line.

Format

Use the following format at the beginning of a component definition:

```
BEGIN peripheral_name
```

The BEGIN keyword signifies the beginning of a new peripheral.

Use the following format for assignment commands:

```
command name = value
```

Use the following format to end a peripheral definition:

```
END
```

There are three assignment commands:

- BUS_INTERFACE
- PARAMETER
- PORT

MHS Example

The following is an example MHS file:

```
# Parameters
PARAMETER VERSION = 2.1.0

# Global Ports

# Assign power signals
PORT vcc_out = net_vcc, DIR=OUTPUT
PORT gnd_out = net_gnd, DIR=OUT
PORT gnd_out6 = net_gnd, DIR=OUTPUT, VEC=[0:5]

PORT intr1 = intr_1, DIR=IN, SENSITIVITY=EDGE_RISING, SIGIS=INTERRUPT
PORT intr2 = intr2, DIR=INPUT, SENSITIVITY=LEVEL_HIGH, SIGIS=INTERRUPT

# Assign constant signals
PORT const1 = 0b1010, DIR=OUTPUT, VEC=[0:3]
PORT const2 = 0xC, DIR=OUTPUT, VEC=[0:3]

PORT sys_rst = sys_rst, DIR=IN
PORT sys_clk = sys_clk, DIR=IN, SIGIS=CLK, CLK_FREQ=100000000
PORT gpio_io = gpio_io, DIR=INOUT, VEC=[0:31]

# Sub Components

##############################################################################
BEGIN lmb_v10
PARAMETER INSTANCE = ilmb_v10
PARAMETER HW_VER = 1.00.a
PORT LMB_Clk = sys_clk
PORT SYS_Rst = sys_rst
END
##############################################################################
BEGIN lmb_v10
PARAMETER INSTANCE = dlmb_v10
PARAMETER HW_VER = 1.00.a
PORT LMB_Clk = sys_clk
PORT SYS_Rst = sys_rst
END
##############################################################################
BEGIN opb_v20
PARAMETER INSTANCE = myopb_bus
PARAMETER HW_VER = 1.10.c
PARAMETER C_PROC_INTRFCE = 0
PORT OPB_Clk = sys_clk
PORT SYS_Rst = sys_rst
END
##############################################################################
```

```
BEGIN opb_gpio
PARAMETER INSTANCE = mygpio
PARAMETER HW_VER = 1.00.a
PARAMETER C_GPIO_WIDTH = 32
PARAMETER C_BASEADDR = 0xffff0100
PARAMETER C_HIGHADDR = 0xffff01ff
PORT GPIO_IO = gpio_io
BUS_INTERFACE SOPB = myopb_bus
END
###############################################################################
BEGIN bram_block
PARAMETER INSTANCE = bram1
PARAMETER HW_VER = 1.00.a
BUS_INTERFACE PORTA = ilmb1_porta
BUS_INTERFACE PORTB = dlmb1_portb
END
###############################################################################
BEGIN lmb_bram_if_cntlr
PARAMETER INSTANCE = my_ilmb_cntlr1
PARAMETER HW_VER = 1.00.b
PARAMETER C_BASEADDR = 0x00000000
PARAMETER C_HIGHADDR = 0x00000fff
BUS_INTERFACE SLMB = ilmb_v10
BUS_INTERFACE BRAM_PORT = ilmb1_porta
END
###############################################################################
BEGIN lmb_bram_if_cntlr
PARAMETER INSTANCE = my_dlmb_cntlr1
PARAMETER HW_VER = 1.00.b
PARAMETER C_BASEADDR = 0x00000000
PARAMETER C_HIGHADDR = 0x00000fff
BUS_INTERFACE SLMB = dlmb_v10
BUS_INTERFACE BRAM_PORT = dlmb1_portb
END
###############################################################################
BEGIN microblaze
PARAMETER INSTANCE = mblaze
PARAMETER HW_VER = 4.00.a
BUS_INTERFACE DLMB = dlmb_v10
BUS_INTERFACE ILMB = ilmb_v10
BUS_INTERFACE DOPB = myopb_bus
PORT Interrupt = mblaze_intr
END
###############################################################################
```

```
# Priorities are numbered N downto 1, where 1 is the highest priority
BEGIN opb_intc
PARAMETER INSTANCE = opb_intc_1
PARAMETER HW_VER = 1.00.c
PARAMETER C_HIGHADDR = 0xC800001F
PARAMETER C_BASEADDR = 0xC8000000
PARAMETER C_HAS_IPR = 1 # Interrupt Pending Register present
PARAMETER C_HAS_SIE = 0 # Set Interrupt Enable bits not present
PARAMETER C_HAS_CIE = 0 # Clear Interrupt Enable bits not present
PARAMETER C_HAS_IVR = 0 # Interrupt Vector Register not present
BUS_INTERFACE SOPB = myopb_bus
PORT Intr = intr2 & intr_1 # intr_1 has highest priority
PORT Irq = mblaze_intr
END
```

Bus Interface

Definition

A bus interface is a grouping of interconnecting signals that are related.

Several components often have many of the same ports, requiring redundant port declarations for each component. Every component connected to an OPB bus, for example, must have the same ports defined and connected together.

A bus interface provides a high level of abstraction for the component connectivity of a common interface. Components can use a bus interface as if it were a single port. In its simplest form, a bus interface can be considered a bundle of signals.

The table below lists recommendations for bus labels.

Table 2-1: **Bus Labels**

Bus Name	Description
SDCR	Slave DCR interface
SLMB	Slave LMB interface
MOPB	Master OPB interface
MSOPB	Master-slave OPB interface
SOPB	Slave OPB interface
MPLB	Master PLB interface
MSPLB	Master-slave PLB interface
SPLB	Slave PLB interface

For components that have more than one bus interface, refer to the MPD file for a definition of listed bus interface labels. For example, the data-side OPB and instruction-side OPB are named DOPB and IOPB, respectively.

A bus interface is assigned by name to an instance of the bus in your system.

Example

For example, the OPB bus instance is named “myopb”, and a connection to the OPB slave interface of the OPB Uart Lite is made with the bus_interface command.

```
BEGIN opb_uartlite
PARAMETER HW_VER = 1.00.b
PARAMETER INSTANCE = myuartlite
PARAMETER C_HIGHADDR = 0xFFFF80FF
PARAMETER C_BASEADDR = 0xFFFF8000
BUS_INTERFACE SOPB = myopb
PORT RX = rx1
PORT TX = tx1
PORT Interrupt = uart_intr
END
```

Local Bus Interface

Local Bus Interface Keyword(s)

POSITION

Use the POSITION keyword to set the position of the bus interface on the bus. For example, use to define master request priority, or DCR slave rank, in the following format:

```
BUS_INTERFACE MOPB=opb_bus_inst, POSITION=N
```

Where *N* is a positive integer. Order is defined from 1 to *N*.

The order of assignment is retained as listed in the MHS in top-to-bottom order.

Note: When specifying bus interfaces of master-slave like MSPLB or MSOPB, there is a possibility that Platgen will error out when you have more masters than slaves on the bus. The reason is that the MSPLB or MSOPB is assigned a position. This means the master interface and the slave interface must reside at the same position. There is a possibility that the assigned position of the slave interface is out of range to the number of slaves on the bus.

Global Parameter

Definition

A global parameter is defined outside of an instance BEGIN-END block.

Global Parameter Keyword(s)

VERSION

Use the VERSION keyword to set the MHS version in the following format:

```
PARAMETER VERSION = 2.1.0
```

The version is specified as a literal of the form 2.1.0.

Local Parameter

Definition

A local parameter is defined between an instance BEGIN-END block.

Local Parameter Keyword(s)

A local parameter can have the keywords listed below:

HW_VER

Use the HW_VER keyword to set the hardware version in the following format:

```
PARAMETER HW_VER = 1.00.a
```

The version is specified as a literal of the form 1.00.a.

INSTANCE

Use the INSTANCE keyword to set the instance name of the peripheral. This keyword is mandatory, and the instance name must be specified in lower-case, in the following format:

```
PARAMETER INSTANCE = my_uart0
```

Global Port

Global Port Keyword Summary

A global port outside of a BEGIN-END block can have the keywords listed below:

BUFFER_TYPE
CLK_FREQ
DIR
RST_POLARITY
SENSITIVITY
SIGIS
VEC

Global Port Keyword Definitions

BUFFER_TYPE

Selects the type of buffer to be inserted on the input port using the BUFFER_TYPE keyword in the following format:

```
PORT CLK = "", DIR=I, BUFFER_TYPE=IBUF
```

The available values are the following: bufgdll, ibufg, bufgp, ibuf, bufr, and none.

If the BUFFER_TYPE exists on the MPD port, Platgen raises the property up to the top-level port that is directly connected. If the BUFFER_TYPE exists on the top-level MHS port, it overrides any MPD port definition of BUFFER_TYPE.

The BUFFER_TYPE is translated into an XST pragma resident in the top-level <*system*> HDL.

Note: This constraint selects the type of buffer to be inserted on the input port or internal net. In general, it is best to avoid using this constraint and allow XST to infer the proper buffer. Avoidance of this constraint allows for flexibility across device migration or synthesis tool selection.

For an EDK submodule flow, XST does not infer the buffer as defined by the BUFFER_TYPE. This is correct behavior since it is not expected that IO buffers will be present for a submodule flow.

CLK_FREQ

The frequency of the top-level clock port of the system can be specified in the MHS using the CLK_FREQ keyword, as in the following format:

```
PORT sys_clk = sys_clk, DIR = IN, SIGIS = CLK, CLK_FREQ = 100000000
```

This keyword should only be used with the top-level clock ports. For the tools to read the clock frequency specified using the CLK_FREQ keyword, the port *must* have the SIGIS=CLK sub-property.

DIR

The driver direction of a signal is specified by the DIR keyword in the following format:

```
PORT mysignal = "", DIR=direction
```

In this example, *direction* is either I, O, or IO.

RST_POLARITY

The reset polarity of the top-level reset port in the system can be specified in the MHS using the RST_POLARITY keyword, as in the following format:

```
PORT sys_rst = sys_rst, DIR = IN, SIGIS = RST, RST_POLARITY=1
```

SENSITIVITY

The sensitivity of an interrupt signal is specified by the SENSITIVITY keyword. This keyword supersedes the EDGE and LEVEL keywords in the following format:

```
PORT interrupt = "", DIR=O, SENSITIVITY=value, SIGIS=INTERRUPT
```

In this example, the *value* is either EDGE_FALLING, EDGE_RISING, LEVEL_HIGH or LEVEL_LOW.

SIGIS

The class of a signal is specified by the SIGIS keyword in the following format:

```
PORT mysig = "", DIR=O, SIGIS=value
```

In this example, the *value* is either CLK, INTERRUPT, or RST. The table below describes SIGIS usage.

Table 2-2: **SIGIS Usage**

SIGIS	Usage
CLK	• XPS Displays all clock signals. • Platgen For all bus components, the clock signals are automatically connected to the clock input of the peripherals on the bus.
INTERRUPT	• XPS Displays all interrupt signals. • Platgen Encodes the priority interrupt vector.
RST	• XPS Displays all reset signals.

VEC

The vector width of a signal is specified by the VEC keyword in the following format:

```
PORT mysignal = "", DIR=I, VEC=[A:B]
```

In this example, A and B are positive integer expressions.

Local Port

A local port is a port defined between an instance BEGIN-END block. A local port does not have keywords.

Design Considerations

This section identifies general design considerations.

Defining Memory Size

Memory sizes are based on C_BASEADDR and C_HIGHADDR settings. Use the following format when defining memory size:

```
PARAMETER C_HIGHADDR= 0xFFFF00FF
PARAMETER C_BASEADDR= 0xFFFF0000
```

All memory sizes must be 2^N, where N is a positive integer and 2^N boundary overlaps are not allowed.

The range specified by C_BASEADDR and C_HIGHADDR must comprise a complete, contiguous power-of-two range, such that range = 2^N and the N least significant bits of C_BASEADDR must be zero.

Power Signals (net_gnd/net_vcc)

Power signals are constantly driven with either GND (net_gnd) or VCC (net_vcc) in the following format:

```
PORT mysignal = power_signal
```

In this example, *power_signal* is either net_vcc or net_gnd. Platgen expands net_vcc or net_gnd to the appropriate vector size. Therefore, it is not legal to use net_vcc or net_gnd in a concatenation construct because the number of bits that are consumed is unknown.

Unconnected Ports

Unconnected output ports are assigned open, and unconnected input ports are either set to GND (net_gnd) or VCC (net_vcc).

An unconnected output port is identified as an empty double-quote ("") string.

Platgen resolves the driver value on unconnected input ports with the INITIALVAL keyword, as defined in the MPD in the following format:

```
PORT mysignal = "", INITIALVAL=VCC
```

Constant Assignments

Use the 0b denotation to define a binary constant, or 0x for a hex constant. An underscore (_) can be used for readability in the following format:

```
PORT mysignal = 0b1010_0101 # mysignal is 8-bits
```

Or

```
PORT mysignal = 0xA5 # mysignal is 8-bits
```

In general, use the 0b syntax for bitwidths that are not evenly divided by 4. Use the 0x syntax for bitwidths that are multiples of 4.

Concatenation

Concatenation is performed with the ampersand (&) operator and allows you to group signals together. It is not legal to use net_vcc or net_gnd in a concatenation construct because the number of bits that are consumed is unknown.

Concatenation combines signals in their bit order. Note, for example, the following top-level port declarations:

```
PORT A = A, DIR=INPUT
PORT B = B, DIR=INPUT, VEC[1:0]
PORT C = C, DIR=INPUT
PORT D = D, DIR=INPUT, VEC[0:3]
PORT Y = A & B & C & D, DIR=OUTPUT, VEC=[7:0]
```

Concatenation is accomplished on A, B, C, and D connecting to port Y of [7:0]. This maps to the following: Y[7]=A, Y[6]=B[1], Y[5]=B[0], Y[4]=C, Y[3]=D[0], Y[2]=D[1], Y[1]=D[2], and Y[0]=D[3].

Concatenation is also useful for extending the length of a vector. Use the 0b denotation to define a binary constant, or the 0x for a hex constant. An underscore (_) can be used for readability. Note, for example, the following top-level port:

```
PORT E = E, DIR=INPUT, VEC=[1:0]
PORT Z = 0b00 & E, DIR=OUTPUT, VEC=[0:3]
```

In this example, the ampersand (&) operator is used to extend the signal `E` to 4 bits. This maps to the following: `Z[0]=0b0`, `Z[1]=0b0`, `Z[2]=E[1]`, and `Z[3]=E[0]`.

Note: MHS syntax does not support vector indexing. For example, the following syntax is unsupported. The usage of `B[1]` and `B[0]` is unsupported.

```
PORT Y = A & B[1] & B[0] & C & D, DIR=OUTPUT, VEC=[7:0]
```

Internal vs. External Signals

By default, all signals defined between a `BEGIN-END` block are internal signals.

External signals are available through the port-declaration of the top-level module. Use the `PORT` command outside of a `BEGIN-END` block to declare the external signal.

External Interrupt Signals

For internal interrupts, each interruptible, peripheral instance defines an interrupt signal locally.

For external interrupts, use the PORT command outside of a BEGIN-END block to declare the external signal and define the interrupt sensitivity in the following format:

```
PORT my_int1 = my_int1, LEVEL=HIGH, DIR=I
```

XILINX®

Chapter 3

Microprocessor Peripheral Definition (MPD)

The Microprocessor Peripheral Definition (MPD) file defines the interface of the peripheral.

An MPD file has the following characteristics:

- Lists ports and default connectivity for bus interfaces
- Lists parameters and default values
- Any MPD parameter is overwritten by the equivalent MHS assignment. Refer to Chapter 2, "Microprocessor Hardware Specification (MHS)" for additional information.

Individual peripheral documentation contains information on all MPD file keywords.

This chapter contains the following sections:

MPD Syntax

Definition

MPD file syntax is case insensitive. The current version is 2.1.0.

The MPD parameter or signal name must be Hardware Description Language (HDL) compliant. VHDL and Verilog have certain naming rules and conventions that must be followed.

The MPD file is supplied by the IP provider and provides peripheral information. This file lists ports and default connectivity to the bus interface. Parameters that you set in this file

are mapped to generics for VHDL, or to parameters for Verilog (with the exception of NON_HDL parameters, which you should specify with the TYPE=NON_HDL keyword).

Comments

You can insert comments in the MPD file without disrupting processing. The following are guidelines for inserting comments:

- Precede comments with the pound sign (#).
- Comments continue to the end of the line.
- Comments can be anywhere on the line.

Format

Use the following format at the beginning of a component definition:

```
BEGIN peripheral_name
```

The BEGIN keyword signifies the beginning of a new peripheral.

Use the following format for assignment commands:

```
command name = value
```

Use the following format to end a peripheral definition:

```
END
```

Assignment Commands

There are five assignment commands:

1. BUS_INTERFACE
2. IO_INTERFACE
3. OPTION
4. PARAMETER
5. PORT

Signal Direction

Signals have three modes. The three modes and their accepted values are as follows:

- input - [I]
- output - [O]
- inout - [IO]

Signal mode indicates its driver direction, and whether the port can be read from within the peripheral.

MPD Example

The following is an example MPD file:

```
BEGIN xps_gpio

## Peripheral Options
OPTION IPTYPE = PERIPHERAL
OPTION IMP_NETLIST = TRUE
OPTION HDL = VHDL
OPTION LAST_UPDATED = 9.2
OPTION USAGE_LEVEL = BASE_USER
OPTION DESC = XPS General Purpose IO
OPTION LONG_DESC = General Purpose Input/Output (GPIO) core for the
  PLBV46 bus.
OPTION IP_GROUP = General Purpose IO:MICROBLAZE:PPC
OPTION ARCH_SUPPORT_MAP = (spartan3=PREFERRED, virtex4lx=PREFERRED,
  virtex4sx=PREFERRED, virtex4fx=PREFERRED, spartan3e=PREFERRED,
  virtex5lx=PREFERRED, virtex5sx=PREFERRED, spartan3a=PREFERRED,
  spartan3adsp=PREFERRED)

IO_INTERFACE IO_IF = gpio_0, IO_TYPE = XIL_GPIO_V1

## Bus Interfaces
BUS_INTERFACE BUS = SPLB, BUS_STD = PLBV46, BUS_TYPE = SLAVE

## Generics for VHDL or Parameters for Verilog
PARAMETER C_BASEADDR = 0xffffffff, DT = std_logic_vector(0 to 31),
  BUS = SPLB, ADDRESS = BASE, PAIR = C_HIGHADDR, MIN_SIZE = 0x200,
  ASSIGNMENT = REQUIRE
PARAMETER C_HIGHADDR = 0x00000000, DT = std_logic_vector(0 to 31),
  BUS = SPLB, ADDRESS = HIGH, PAIR = C_BASEADDR, ASSIGNMENT = REQUIRE
PARAMETER C_SPLB_AWIDTH = 32, DT = INTEGER, BUS = SPLB,
  ASSIGNMENT = CONSTANT
PARAMETER C_SPLB_DWIDTH = 32, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_P2P = 0, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_MID_WIDTH = 1, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_NUM_MASTERS = 1, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_NATIVE_DWIDTH = 32, DT = INTEGER, BUS = SPLB,
  ASSIGNMENT = CONSTANT
PARAMETER C_SPLB_SUPPORT_BURSTS = 0, DT = INTEGER, BUS = SPLB,
  ASSIGNMENT = CONSTANT
PARAMETER C_FAMILY = virtex5, DT = STRING
PARAMETER C_GPIO_WIDTH = 32, DT = INTEGER, RANGE = (1:32),
  PERMIT = BASE_USER, DESC = GPIO Data Width, IO_IF = gpio_0,
  IO_IS = num_bits
PARAMETER C_ALL_INPUTS = 0, DT = INTEGER, RANGE = (0,1),
  PERMIT = BASE_USER, DESC = Data pins are all inputs, IO_IF = gpio_0,
  IO_IS = all_inputs, VALUES = (0= FALSE , 1= TRUE )
PARAMETER C_INTERRUPT_PRESENT = 0, DT = INTEGER, RANGE = (0,1)
PARAMETER C_IS_BIDIR = 1, DT = INTEGER, RANGE = (0,1),
  PERMIT = BASE_USER, DESC = Data pins are bi-directional,
  IO_IF = gpio_0, IO_IS = is_bidir, VALUES = (0= FALSE , 1= TRUE )
PARAMETER C_DOUT_DEFAULT = 0x00000000, DT = std_logic_vector
PARAMETER C_TRI_DEFAULT = 0xffffffff, DT = std_logic_vector
PARAMETER C_IS_DUAL = 0, DT = INTEGER, RANGE = (0,1),
  DESC = Use Dual GPIO, IO_IF = gpio_0, IO_IS = is_dual
PARAMETER C_ALL_INPUTS_2 = 0, DT = INTEGER, RANGE = (0,1), DESC = GPIO2
  Data All Inputs, IO_IF = gpio_0, IO_IS = all_inputs_2,
```

```
  VALUES = (0=FALSE, 1=TRUE)
PARAMETER C_IS_BIDIR_2 = 1, DT = INTEGER, RANGE = (0,1),
  DESC = Use GPIO2 Bidir IO Pin, IO_IF = gpio_0, IO_IS = is_bidir_2,
  VALUES = (0=FALSE, 1=TRUE)
PARAMETER C_DOUT_DEFAULT_2 = 0x00000000, DT = std_logic_vector
PARAMETER C_TRI_DEFAULT_2 = 0xffffffff, DT = std_logic_vector

## Ports
PORT SPLB_Clk = "", DIR = I, SIGIS = Clk, BUS = SPLB
PORT SPLB_Rst = SPLB_Rst, DIR = I, SIGIS = Rst, BUS = SPLB
PORT PLB_ABus = PLB_ABus, DIR = I, VEC = [0:31], BUS = SPLB
PORT PLB_UABus = PLB_UABus, DIR = I, VEC = [0:31], BUS = SPLB
PORT PLB_PAValid = PLB_PAValid, DIR = I, BUS = SPLB
PORT PLB_SAValid = PLB_SAValid, DIR = I, BUS = SPLB
PORT PLB_rdPrim = PLB_rdPrim, DIR = I, BUS = SPLB
PORT PLB_wrPrim = PLB_wrPrim, DIR = I, BUS = SPLB
PORT PLB_masterID = PLB_masterID, DIR = I,
  VEC = [0:(C_SPLB_MID_WIDTH-1)], BUS = SPLB
PORT PLB_abort = PLB_abort, DIR = I, BUS = SPLB
PORT PLB_busLock = PLB_busLock, DIR = I, BUS = SPLB
PORT PLB_RNW = PLB_RNW, DIR = I, BUS = SPLB
PORT PLB_BE = PLB_BE, DIR = I, VEC = [0:((C_SPLB_DWIDTH/8)-1)],
  BUS = SPLB
PORT PLB_MSize = PLB_MSize, DIR = I, VEC = [0:1], BUS = SPLB
PORT PLB_size = PLB_size, DIR = I, VEC = [0:3], BUS = SPLB
PORT PLB_type = PLB_type, DIR = I, VEC = [0:2], BUS = SPLB
PORT PLB_lockErr = PLB_lockErr, DIR = I, BUS = SPLB
PORT PLB_wrDBus = PLB_wrDBus, DIR = I, VEC = [0:(C_SPLB_DWIDTH-1)],
  BUS = SPLB
PORT PLB_wrBurst = PLB_wrBurst, DIR = I, BUS = SPLB
PORT PLB_rdBurst = PLB_rdBurst, DIR = I, BUS = SPLB
PORT PLB_wrPendReq = PLB_wrPendReq, DIR = I, BUS = SPLB
PORT PLB_rdPendReq = PLB_rdPendReq, DIR = I, BUS = SPLB
PORT PLB_wrPendPri = PLB_wrPendPri, DIR = I, VEC = [0:1], BUS = SPLB
PORT PLB_rdPendPri = PLB_rdPendPri, DIR = I, VEC = [0:1], BUS = SPLB
PORT PLB_reqPri = PLB_reqPri, DIR = I, VEC = [0:1], BUS = SPLB
PORT PLB_TAttribute = PLB_TAttribute, DIR = I, VEC = [0:15], BUS = SPLB
PORT Sl_addrAck = Sl_addrAck, DIR = O, BUS = SPLB
PORT Sl_SSize = Sl_SSize, DIR = O, VEC = [0:1], BUS = SPLB
PORT Sl_wait = Sl_wait, DIR = O, BUS = SPLB
PORT Sl_rearbitrate = Sl_rearbitrate, DIR = O, BUS = SPLB
PORT Sl_wrDAck = Sl_wrDAck, DIR = O, BUS = SPLB
PORT Sl_wrComp = Sl_wrComp, DIR = O, BUS = SPLB
PORT Sl_wrBTerm = Sl_wrBTerm, DIR = O, BUS = SPLB
PORT Sl_rdDBus = Sl_rdDBus, DIR = O, VEC = [0:(C_SPLB_DWIDTH-1)],
  BUS = SPLB
PORT Sl_rdWdAddr = Sl_rdWdAddr, DIR = O, VEC = [0:3], BUS = SPLB
PORT Sl_rdDAck = Sl_rdDAck, DIR = O, BUS = SPLB
PORT Sl_rdComp = Sl_rdComp, DIR = O, BUS = SPLB
PORT Sl_rdBTerm = Sl_rdBTerm, DIR = O, BUS = SPLB
PORT Sl_MBusy = Sl_MBusy, DIR = O, VEC = [0:(C_SPLB_NUM_MASTERS-1)],
  BUS = SPLB
PORT Sl_MWrErr = Sl_MWrErr, DIR = O, VEC = [0:(C_SPLB_NUM_MASTERS-1)],
  BUS = SPLB
PORT Sl_MRdErr = Sl_MRdErr, DIR = O, VEC = [0:(C_SPLB_NUM_MASTERS-1)],
  BUS = SPLB
PORT Sl_MIRQ = Sl_MIRQ, DIR = O, VEC = [0:(C_SPLB_NUM_MASTERS-1)],
  BUS = SPLB
PORT IP2INTC_Irpt = "", DIR = O, SIGIS = INTERRUPT,
```

```
  SENSITIVITY = LEVEL_HIGH, INTERRUPT_PRIORITY = MEDIUM
PORT GPIO_IO = "", DIR = IO, VEC = [0:(C_GPIO_WIDTH-1)],
  THREE_STATE = TRUE, TRI_I = GPIO_IO_I, TRI_O = GPIO_IO_O,
  TRI_T = GPIO_IO_T, ENABLE = MULTI, PERMIT = BASE_USER,
  DESC = 'GPIO1 Data IO', IO_IF = gpio_0, IO_IS = gpio_io
PORT GPIO_IO_I = "", DIR = I, VEC = [0:(C_GPIO_WIDTH-1)]
PORT GPIO_IO_O = "", DIR = O, VEC = [0:(C_GPIO_WIDTH-1)]
PORT GPIO_IO_T = "", DIR = O, VEC = [0:(C_GPIO_WIDTH-1)]
PORT GPIO_in = "", DIR = I, VEC = [0:(C_GPIO_WIDTH-1)],
  PERMIT = BASE_USER, DESC = 'GPIO1 Data In', IO_IF = gpio_0,
  IO_IS = gpio_data_in
PORT GPIO_d_out = "", DIR = O, VEC = [0:(C_GPIO_WIDTH-1)],
  PERMIT = BASE_USER, DESC = 'GPIO1 Data Out', IO_IF = gpio_0,
  IO_IS = gpio_data_out
PORT GPIO_t_out = "", DIR = O, VEC = [0:(C_GPIO_WIDTH-1)],
  PERMIT = BASE_USER, DESC = 'GPIO1 TriState Out', IO_IF = gpio_0,
  IO_IS = gpio_tri_out
PORT GPIO2_IO = "", DIR = IO, VEC = [0:(C_GPIO_WIDTH-1)],
  THREE_STATE = TRUE, TRI_I = GPIO2_IO_I, TRI_O = GPIO2_IO_O,
  TRI_T = GPIO2_IO_T, ENABLE = MULTI, PERMIT = BASE_USER,
  DESC = 'GPIO2 Data IO', IO_IF = gpio_0, IO_IS = gpio_io_2
PORT GPIO2_IO_I = "", DIR = I, VEC = [0:(C_GPIO_WIDTH-1)]
PORT GPIO2_IO_O = "", DIR = O, VEC = [0:(C_GPIO_WIDTH-1)]
PORT GPIO2_IO_T = "", DIR = O, VEC = [0:(C_GPIO_WIDTH-1)]
PORT GPIO2_in = "", DIR = I, VEC = [0:(C_GPIO_WIDTH-1)],
  PERMIT = BASE_USER, DESC = 'GPIO2 Data In', IO_IF = gpio_0,
  IO_IS = gpio_data_in_2
PORT GPIO2_d_out = "", DIR = O, VEC = [0:(C_GPIO_WIDTH-1)],
  PERMIT = BASE_USER, DESC = 'GPIO2 Data Out', IO_IF = gpio_0,
  IO_IS = gpio_data_out_2
PORT GPIO2_t_out = "", DIR = O, VEC = [0:(C_GPIO_WIDTH-1)],
  PERMIT = BASE_USER, DESC = 'GPIO2 TriState Out', IO_IF = gpio_0,
  IO_IS = gpio_tri_out_2

END
```

Bus Interface

Definition

A bus interface is a grouping of interface ports that are related.

Several components often have many of the same ports, requiring redundant port declarations for each component. Every component connected to a PLB v4.6 bus, for example, must have the same ports defined and connected.

A bus interface provides a high level of abstraction for the component connectivity of a common interface. Components can use a bus interface as if it were a single port. In its simplest form, a bus interface can be considered a bundle of signals.

Bus Interface Keyword Summary

A bus interface can have the following keywords:

BUS
BUS_STD
BUS_TYPE
EXCLUDE_BUSIF
GENERATE_BURSTS
ISVALID
SHARES_ADDR

Bus Interface Keyword Definitions

BUS

The label of a bus interface is specified by the BUS keyword. It is expressed in the following format in which bus_label is a string:

```
BUS_INTERFACE BUS=bus_label, BUS_STD=bus_std, BUS_TYPE=bus_type
```

BUS_STD

The bus standard for a bus interface is specified by the BUS_STD keyword. It is expressed in the following format:

```
BUS_INTERFACE BUS=bus_label, BUS_STD=bus_std, BUS_TYPE=bus_type
```

The Xilinx-known BUS_STD values are DCR, LMB, OPB, PLB, DSOCM, ISOCM, FSL. You can also define your own bus standards.

Note: BUS_STD = TRANSPARENT is deprecated. It should be replaced with a user-defined "compatibility" string for any point-to-point connections.

BUS_TYPE

The bus type for a bus interface is specified by the BUS_TYPE keyword. It represents the relationship of the interface to the connection. The BUS_TYPE keyword is expressed in the following format:

```
BUS_INTERFACE BUS=bus_label, BUS_STD=bus_std, BUS_TYPE=bus_type
```

For centrally connected buses (for example, PLB v4.6), valid values of BUS_TYPE are MASTER, MASTER_SLAVE, SLAVE, and MONITOR.

For point-to-point connected bus_interfaces, valid values are INITIATOR, TARGET.

EXCLUDE_BUSIF

The EXCLUDE_BUSIF keyword defines all BUS_INTERFACE connections when other BUS_INTERFACE connections are present. Supports a colon-separated list of elements, but can also take a single element.

For example, if two interfaces are defined for an IP as master-slave, *or* as a slave interface, then only one of them can be used to connect the IP. This keyword is expressed in the following format:

```
BUS_INTERFACE BUS=MSPLB, BUS_STD=PLB, BUS_TYPE=MASTER_SLAVE,
EXCLUDE_BUSIF=SPLB
BUS_INTERFACE BUS=SPLB, BUS_STD=PLB, BUS_TYPE=SLAVE,
EXCLUDE_BUSIF=MSPLB
```

GENERATE_BURSTS

GENERATE_BURSTS specifies that a non-bridge master generates bursts. This is only valid for BUS_INTERFACEs of BUS_TYPE=MASTER. The keyword is expressed in the following format:

```
BUS_INTERFACE BUS = MPLB, BUS_STD = PLBV46, BUS_TYPE = MASTER,
GENERATE_BURSTS = [FALSE|TRUE]
```

EDK tools observe all masters and slaves that could communicate through a given bridge. If there is at least one master that has the GENERATE_BURSTS=TRUE, and if there is at least one slave that can support bursts, the bridge is configured to support bursts. To determine whether a slave can support bursts, the EDK tools use the following method:

- If slave does not have the C_<busif>_SUPPORT_BURSTS parameter, the slave BUS_INTERFACE does not support bursts. This means any slave that always supports bursts must also have the parameter.
- If a slave has the C_<busif>_SUPPORT_BURSTS parameter, it is assumed that it can support bursts.

ISVALID

The ISVALID keyword defines the validity of a BUS_INTERFACE to an expression. If the expression evaluates true, the BUS_INTERFACE is included in the list of valid BUS_INTERFACEs for the defined system. If false, the BUS_INTERFACE is not included. It is expressed in the following format:

```
BUS_INTERFACE BUS = SPLB0, BUS_STD = PLBV46, BUS_TYPE = SLAVE, ISVALID
= (C_NUM_PORTS > 0   && C_PIM0_BASETYPE == 2)
```

SHARES_ADDR

The SHARES_ADDR keyword defines all BUS_INTERFACE address space regions that need to be checked against one another. The default is ALL. This keyword supports a colon-separated list of elements, but can also take a single element.

For example, the LMB and OPB memory mapped peripherals of a MicroBlaze™ processor must not conflict. Also, the PLB and OCM address spaces for a PowerPC® 405 processor must not conflict. This keyword is expressed in the following format:

```
BUS_INTERFACE BUS=DOPB, BUS_STD=OPB, BUS_TYPE=MASTER, SHARES_ADDR=DLMB
BUS_INTERFACE BUS=IOPB, BUS_STD=OPB, BUS_TYPE=MASTER, SHARES_ADDR=ILMB
BUS_INTERFACE BUS=DLMB, BUS_STD=LMB, BUS_TYPE=MASTER, SHARES_ADDR=DOPB
BUS_INTERFACE BUS=ILMB, BUS_STD=LMB, BUS_TYPE=MASTER, SHARES_ADDR=IOPB
```

Bus Interface Naming Conventions

The table below lists recommendations for bus labels.

Table 3-1: **Recommended Bus Labels**

Bus Label	Description
SDCR	Slave DCR interface
SLMB	Slave LMB interface
MOPB	Master OPB interface
MSOPB	Master-slave OPB interface
SOPB	Slave OPB interface
MPLB	Master PLB interface
MSPLB	Master-slave PLB interface
SPLB	Slave PLB interface

For the MSPLB bus interface, you should separate the master interface and slave interface as MPLB and SPLB, respectively, because the MSPLB is assigned its own position. This means that the master interface and the slave interface must reside at the same position. If given as separate interfaces for MPLB and SPLB, then each interface can have its own position assignment.

IO Interface

Definition

An IO_INTERFACE defines an interface between the IP and some external off-chip device. Associated with each IO Interface are sets of ports that the Base System Builder (BSB) Wizard defines as the systems top-level ports, and a set of parameters that BSB characterizes from the off-chip device. In an MPD file, you can define an IO_INTERFACE by assigning it a unique IO_IF name and an IO type (IO_TYPE). You can then specify which ports and parameters are associated with this interface by adding the IO_IF and IO_IS tags to those ports and parameters.

The tools in conjunction use the IO_INTERFACE with information from the Xilinx® Board Description (XBD) files in order to make intelligent decisions for the user regarding system connectivity and parameterization. The IO_INTERFACE declaration and associated tags are *not* required for core functionality.

IO Interface Keywords

An IO interface can have the following keywords:

IO_IF

A unique, user-defined label assigned an IO_INTERFACE. All ports and parameters declared in an MPD file that are associated with the name IO_INTERFACE have the same IO_IF value assigned to them. This keyword is expressed in the following format, in which io_label is the name of the IO_INTERFACE. This is a user-defined string:

```
IO_INTERFACE IO_IF=io_label, IO_TYPE=io_type
```

IO_TYPE

The IO type of an IO interface is specified by the IO_TYPE keyword, which is expressed in the following format:

```
IO_INTERFACE IO_IF=io_label, IO_TYPE=io_type
```

Here, io_type is one of the following values: XIL_DDR_V1, XIL_EMC_V1, XIL_Ethernet_V1, XIL_GPIO_V1, XIL_IIC_V1, XIL_PCI_ARBITER_V1, XIL_PCI32_V1, XIL_SDRAM_V1, XIL_SPI_V1, XIL_SYSACE_V1, or XIL_UART_V1.

Option

Definition

An option defines a tool directive.

Option Keyword Summary

An option can have the following keywords:

ALERT
ARCH_SUPPORT
ARCH_SUPPORT_MAP
BUS_STD
CLK_FREQ_RATIOS
CORE_STATE
DESC
EARLY_ACCESS_ARCH_SUPPORT
HDL
IMP_NETLIST
IP_GROUP
IPLEVEL_DRC_PROC
IPTYPE
LAST_UPDATED
LONG_DESC
MAX_MASTERS
MAX_SLAVES
PAY_CORE
PLATGEN_SYSLEVEL_UPDATE_PROC
RUN_NGCBUILD
SPECIAL
STYLE
SYSLEVEL_DRC_PROC
SYSLEVEL_UPDATE_PROC
TCL_FILE
USAGE_LEVEL

Option Keyword Definitions

ALERT

A message alert for the IP core is specified with the ALERT keyword in the following format:

```
OPTION ALERT = "This belongs to Xilinx"
```

ARCH_SUPPORT

Deprecated. Subsumed by ARCH_SUPPORT_MAP.

ARCH_SUPPORT is a list of supported FPGA architectures. Valid values are the various FPGA architectures supported by EDK including (but not limited to): all, spartan2, spartan2e, spartan3, spartan3e, virtex, virtexe, virtex2, virtex2p, virtex4, virtex5. The default is ALL. This keyword supports a colon-separated list of elements, but can also take a single element, as in the following format:

```
OPTION ARCH_SUPPORT = virtex2:spartan2e
```

ARCH_SUPPORT_MAP

This is a list of supported FPGA architectures. Valid values are the various FPGA architectures supported by EDK.

The keys of the map are the architecture names. The architecture names are composed of a concatenation of C_FAMILY and C_SUBFAMILY. The parameter C_FAMILY contains VIRTEX4 and VIRTEX5 as valid values. The C_SUBFAMILY has an FX, LX, or SX designation for VIRTEX4, an LX or SX designation for VIRTEX5, and an empty "" for other architectures.

The table below lists ARCH_SUPPORT_MAP values.

Table 3-2: **ARCH_SUPPORT_MAP Values**

ARCH_SUPPORT_MAP	Definition
PREFERRED	Core is active (full uninhibited use) by EDK.
AVAILABLE	Core is available (full uninhibited use) by EDK and info message is given to that user.
BETA	Core is in beta stage and a warning is issued to the user
DEPRECATED	Core is deprecated. EDK tools allow use of core, but issues a warning that the core is deprecated.
DEVELOPMENT	Core is in development and will be synthesized each time Platgen is executed (no cache of synthesis results).
EARLY_ACCESS	Core is in early access stage and a warning is issued to the user.
OBSOLETE	Core is obsolete. EDK tools issue an error that this core is no longer valid.

The key is the primary family name, and value is the core state. It is followed by a comma-separated list of devices. The format is <key name>=<key value>, <key name>=<key value>, as in the following format:

```
OPTION ARCH_SUPPORT_MAP = (virtex2p=PREFERRED, virtex4=PREFERRED,
spartan3a=AVAILABLE, spartan3e=AVAILABLE, virtex5fx=EARLY_ACCESS,
spartan2=OBSOLETE)
```

You can use OTHERS as a key name that expands to all undefined architectures with a common core state, as illustrated in the following examples:

```
OPTION ARCH_SUPPORT_MAP = (virtex2p = PREFERRED, others = AVAILABLE)
OPTION ARCH_SUPPORT_MAP = (others = DEVELOPMENT)
```

Note: Parentheses enclose the beginning and ending of the mapping section. Due to current MPD parse rules, this must be done within one line of the MPD file. You do *not* have the option to cross multiple lines. However, for descriptive purposes, examples are shown to cross multiple lines.

BUS_STD

This keyword defines the bus standard of BUS or BUS_ARBITER cores, as in the following format:

```
OPTION BUS_STD = value
```

In this example, value is one of the following Xilinx-supported bus standards: DCR, DSOCM, FSL, ISOCM, LMB, OPB, PLBV46, or PLB. You can also define your own bus standard. There is no default.

CLK_FREQ_RATIOS

This option specifies the allowed ratios between clocks in an IP. Its value is a list of comma-separated, name-value pairs. Each such pair has a name that describes:

- The ports whose ratio must be computed
- A value that is the actual ratio allowed

The value can include multiple ratios and is therefore expressed as a comma-separated list. The syntax to specify the value for this option is as follows :

```
OPTION   CLK_FREQ_RATIOS  = (  P1/P2 = ( N1/D2, [N2:N3]/D2, N4/[D3:D4],
N5 ),  (P1,P2)/P3 = ( N6, [N7:N8]/[D5:D6] ) )
```

In the expression above, P1, P2, and P3 are the clock ports in the IP. N is the numerator in the ratio, and D is the denominator. The syntax rules to specify the ratio are given below.

- The whole value of the CLK_FREQ_RATIOS option *must* be specified in parentheses.
- The value of the option is a comma-separated list of name-value pairs, where name is the ratio of ports and value is the list of supported ratios with the syntax `Name = (list of values).`
- The list of values *must* be enclosed in parentheses, even if the list has only one element.
- A ratio must be specified as N/D, where N is the numerator and D is the denominator.
- A range (in the numerator or denominator) is only allowed in the list of values section, and it must be specified by using square brackets and a colon. Otherwise, it is considered illegal syntax.
- A range, if specified, cannot contain fractions. Both bounds must be positive integers.

If this option is specified in the MPD of the IP, the tools perform a design rule check (DRC) to verify that the clocks satisfy the ratio requirements in this option. The DRC passes at the first ratio in the list that matches the computed ratio.

CORE_STATE

Deprecated. Subsumed by ARCH_SUPPORT_MAP.

The CORE_STATE keyword specifies the state of the IP core, as in the following format:

```
OPTION CORE_STATE = ACTIVE
```

The table below lists CORE_STATE values.

Table 3-3: **CORE_STATE Values**

CORE_STATE	Definition
ACTIVE	Core is active (full uninhibited use) by EDK (default).
DEPRECATED	Core is deprecated. EDK tools allow use of core, but issues a warning that the core is deprecated.
DEVELOPMENT	Core is in development and will be synthesized each time Platgen is executed (no cache of synthesis results).
OBSOLETE	Core is obsolete. EDK tools issue an error that this core is no longer valid.

DESC

This keyword allows a short description of the core to be displayed by the GUI tools. The short description replaces the core name in the display field of the core, as in the following format:

```
OPTION DESC = "XPS GPIO"
```

EARLY_ACCESS_ARCH_SUPPORT

Deprecated. Subsumed by ARCH_SUPPORT_MAP.

This keyword is a list of FPGA architectures that the core supports in an early access form. There is no default value. It supports a colon-separated list of elements, but can also take a single element, as in the following formats:

```
OPTION EARLY_ACCESS_ARCH_SUPPORT = virtex5:spartan3e
```

And:

```
OPTION EARLY_ACCESS_ARCH_SUPPORT = virtex5
```

HDL

Deprecated. The HDL availability of the IP is specified with the HDL keyword, as in the following format:

```
OPTION HDL = VERILOG
```

The table below lists HDL values.

Table 3-4: **HDL Values**

HDL	Definition
BOTH	The design is completely written both in Verilog and VHDL. This means that the HDL code is available as Verilog and VHDL files. When you select Verilog as the top-level HDL language, the EDK tools release only the Verilog files for use. When you select VHDL, the EDK tools release only the VHDL files. The use of both is deprecated.
MIXED	The design is written as a mixture of Verilog and VHDL.
VERILOG	The design is completely written in Verilog.
VHDL	The design is completely written in VHDL.

IMP_NETLIST

The IMP_NETLIST keyword directs Platgen to write an implementation netlist file for the peripheral, as in the following format:

```
OPTION IMP_NETLIST = TRUE
```

The default is FALSE. If FALSE, Platgen does not generate an implementation netlist (NGC) file. You must provide a mechanism to synthesize your IP.

IP_GROUP

The IP_GROUP keyword defines the IP group classification. The keyword is expressed in the following format, in which ipgroup_label is a string:

```
OPTION IP_GROUP = ipgroup_label
```

If you have more than one IP group sharing the parameter, then use a colon to separate each IP group in the list.

IPLEVEL_DRC_PROC

The IPLEVEL_DRC_PROC keyword defines the Tcl entry point for the IP-level DRC routine. DRCs are based only on IP-level settings. The IPLEVEL_DRC_PROC keyword is expressed following format:

```
OPTION IPLEVEL_DRC_PROC = proc_name
```

IPTYPE

The IPTYPE keyword defines the type of the component, as in the following format:

```
OPTION IPTYPE = PERIPHERAL
```

The table below lists IPTYPE values.

Table 3-5: IPTYPE **Values**

IPTYPE	Definition
BUS	Bus component
PERIPHERAL	Component that is address-mapped to a bus
PROCESSOR	Processor component

Note: The values BRIDGE, IP, and BUS_ARBITER for IPTYPE have been deprecated. The following conventions are used to model them:

- IPTYPE=IP is now modeled as IPTYPE=PERIPHERAL, and the IP has no address parameters.
- IPTYPE=BUS_ARBITER is now modeled as IPTYPE=BUS, and the IP has address parameters.
- IPTYPE=Bridge is now modeled at the ADDRESS parameter level using ADDR_TYPE=BRIDGE and BRIDGE_TO tags.

LAST_UPDATED

This keyword indicates the release for which an IP was last updated. It includes the HDL, MPDs, and Tcl files for each IP. The syntax is as follows:

```
OPTION LAST_UPDATED = <edk_release_number>
```

The edk_release_number is expressed as a version number, for example, 9.2.

LONG_DESC

This keyword allows a long description of the core to be displayed by the GUI tools. The long description allows the GUI tools to display a floating text box that contains additional help information. There is no default, as in the following format:

```
OPTION LONG_DESC = "XPS GPIO - IO only, GPIO"
```

MAX_MASTERS

Define maximum number of masters allowed for cores marked as IPTYPE=BUS or IPTYPE=BUS_ARBITER. No default, as in the following format:

```
OPTION MAX_MASTERS = 8
```

MAX_SLAVES

This keyword defines the maximum number of slaves allowed for cores marked as IPTYPE=BUS or IPTYPE=BUS_ARBITER. There is no default, as in the following format:

```
OPTION MAX_SLAVES = 8
```

PAY_CORE

This keyword identifies a core as being free or purchased. PAY_CORE is Xilinx IP-specific and should be used only by Xilinx IPs. PAY_CORE allows a sub-property, ISVALID, which the tools use internally to identify the license status of the core.

PLATGEN_SYSLEVEL_UPDATE_PROC

The PLATGEN_SYSLEVEL_UPDATE_PROC keyword defines the Tcl entry point for the system-level update routine. This procedure is run after Platgen completes the merge of MPD and MHS descriptions, and it is suitable for use in Tcl to generate UCF entries. The updates are based only on system-level settings and are expressed in the following format:

```
OPTION PLATGEN_SYSLEVEL_UPDATE_PROC = proc_name
```

RUN_NGCBUILD

The RUN_NGCBUILD keyword directs Platgen to execute NGCBUILD to merge multiple hardware netlists into a single deliverable hardware netlist. It is also required to include UCF constraints in the generated netlist.

The flow, as implemented in Platgen, is as follows:

- Run TCL to generate IP level UCF file (referred to as NCF).
- Run XST to generate netlist from the HDL referred to in the Peripheral Analyze Order (PAO) file.

Run NCGBUILD to create a new IP-level netlist that combines the output from the two previous bullets.

RUN_NGCBUILD is required when the TCL generates constraints, or when edk_generatecore is used. It is intuited when a Black Box Definition (BBD) file is present. The RUN_NGCBUILD keyword is expressed in the following format:

```
OPTION RUN_NGCBUILD = TRUE
```

The default is FALSE.

SPECIAL

This keyword is reserved for internal use only.

The SPECIAL keyword defines a class of components that require special handling, as in the following format:

```
OPTION SPECIAL = BRAM_CNTLR
```

STYLE

The STYLE keyword defines the design composition of the peripheral.

If you have only optimized hardware netlists, you must specify the BLACKBOX value within the MPD file. In this case, only the BBD file is read by the EDK tools.

```
OPTION STYLE = BLACKBOX
```

If you have a mix of optimized hardware netlists and HDL files, you must specify the MIX value within the MPD file. In this case, the PAO and BBD files are read by the EDK tools.

```
OPTION STYLE = MIX
```

If you have only HDL files, you must specify the HDL value within the MPD file. In this case, only the PAO file is read by the EDK tools.

```
OPTION STYLE = HDL
```

The table below lists STYLE values.

Table 3-6: **STYLE Values**

STYLE	Definition
BLACKBOX	Only optimized hardware netlists
HDL	Only HDL files (default)
MIX	Mix of optimized hardware netlists and HDL files

SYSLEVEL_DRC_PROC

The SYSLEVEL_DRC_PROC keyword defines the Tcl entry point for the system-level DRC routine. DRCs are based only on system-level settings. The SYSLEVEL_DRC_PROC keyword is expressed in the following format:

```
OPTION SYSLEVEL_DRC_PROC = proc_name
```

SYSLEVEL_UPDATE_PROC

The SYSLEVEL_UPDATE_PROC keyword defines the Tcl entry point for the system-level update routine. The updates are based only on system-level settings. The keyword is expressed in the following format:

```
OPTION SYSLEVEL_UPDATE_PROC = proc_name
```

TCL_FILE

Deprecated. The TCL_FILE keyword defines the Tcl file name. The keyword is expressed in the following format:

```
OPTION TCL_FILE = opb_gpio_v2_1_0.tcl
```

USAGE_LEVEL

The USAGE_LEVEL keyword defines a tool option BSB uses to determine whether or not BSB should configure this IP module. It is expressed in the following format:

```
OPTION USAGE_LEVEL = BASE_USER
```

The table below lists USAGE_LEVEL values.

Table 3-7: **USAGE_LEVEL Values**

USAGE_LEVEL	Definition
ADVANCED_USER	IP *cannot* be configured by BSB.
BASE_USER	IP *can* be configured by BSB.

Parameter

Definition

A parameter defines a constant that is passed into the entity (VHDL) or module (Verilog) declaration.

Parameter Keyword Summary

A parameter can have the following keywords:

ADDRESS	DESC	MIN_SIZE
ADDR_TYPE	DT	PAIR
ASSIGNMENT	IO_IF	PERMIT
BRIDGE_TO	IO_IS	RANGE
BUS	IPLEVEL_DRC_PROC	SYSLEVEL_DRC_PROC
CACHEABLE	IPLEVEL_UPDATE_VALUE_PROC	SYSLEVEL_UPDATE_VALUE_PROC
CLK_PORT	ISVALID	TYPE
CLK_UNIT	LONG_DESC	VALUES

Parameter Keyword Definitions

ADDRESS

The ADDRESS keyword identifies a named parameter as a valid address parameter. The keyword is expressed in the following format:

```
PARAMETER C_BASEADDR=0xFFFFFFFF, MIN_SIZE=0x2000, ADDRESS=BASE
```

The table below lists ADDRESS values.

Table 3-8: **ADDRESS Values**

ADDRESS	Definition
BASE	Identify base address (default for C_BASEADDR)
HIGH	Identify high address (default for C_HIGHADDR)
SIZE	Deprecated. Identify size of address (paired with ADDRESS=HIGH or ADDRESS=BASE)
NONE	Disable identification of address parameter

ADDR_TYPE

The ADDR_TYPE keyword identifies an address parameter of a defined memory class. The keyword is expressed in the following format:

```
PARAMETER C_BASEADDR=0xFFFFFFFF, MIN_SIZE=0x2000, ADDR_TYPE=REGISTER
```

The table below lists ADDR_TYPE values.

Table 3-9: **ADDR_TYPE Values**

ADDR_TYPE	Definition
BRIDGE	Address window on the bridge. An address of this type is forwarded to the bus which is slave to the bridge.
MEMORY	Address window on the memory controller. An address of this type points to a storage memory, such as SDRAM, DDR, FLASH, or BRAM.
REGISTER	Address of its own registers. An address of this type points to registers in the peripheral. These could be status, control, data registers, or some FIFO registers in the peripheral. This is the default.

ASSIGNMENT

The ASSIGNMENT keyword defines the assignment usage level. The keyword is expressed in the following format:

```
PARAMETER C_HAS_EXTERNAL_XIN=0, DT=integer, ASSIGNMENT=OPTIONAL
```

The table below lists ASSIGNMENT values.

Table 3-10: **ASSIGNMENT Values**

ASSIGNMENT	Definition
CONSTANT	This value is a constant. You cannot modify it.
OPTIONAL	If you do not specify a value, the EDK batch tools use the default.
OPTIONAL_UPDATE	You can specify an MHS value for any parameter associated with a Tcl procedure. The MHS value has precedence over the Tcl-calculated value, as called through IPLEVEL_UPDATE_VALUE_PROC or SYSLEVEL_UPDATE_VALUE_PROC.
REQUIRE	You must specify a value.
UPDATE	You cannot specify a value. It is computed by the EDK batch tools.

Examples:

- MPD example of ASSIGNMENT=OPTIONAL_UPDATE:

  ```
  SYSLEVEL_UPDATE_VALUE_PROC
  PARAMETER C_MASK=0x00800000, SYSLEVEL_UPDATE_VALUE_PROC =
  update_syslevel_mask, ASSIGNMENT = OPTIONAL_UPDATE
  ```

  It is recommended that the IP developer also provide a DRC procedure to validate the user-specified MHS value. Use the SYSLEVEL_DRC_PROC to define the name of the DRC entry point.

- MPD example pairing SYSLEVEL_UPDATE_VALUE_PROC and SYSLEVEL_DRC_PROC (ASSIGNMENT=OPTIONAL_UPDATE inferred):

```
PARAMETER C_MASK=0x00800000, SYSLEVEL_UPDATE_VALUE_PROC =
update_syslevel_mask, SYSLEVEL_DRC_PROC = check_syslevel_mask
```

In the example above, the ASSIGNMENT=OPTIONAL_UPDATE is not listed because it is inferred from the pairing of the SYSLEVEL_UPDATE_VALUE_PROC and SYSLEVEL_DRC_PROC subproperties.

The pairing of IPLEVEL_UPDATE_VALUE_PROC and IPLEVEL_DRC_PROC or SYSLEVEL_UPDATE_VALUE_PROC and SYSLEVEL_DRC_PROC infers the ASSIGNMENT=OPTIONAL_UPDATE.

BRIDGE_TO

The BRIDGE_TO keyword allows an address to be visible through the bridge. The keyword is expressed in the following format:

```
PARAMETER C_BASEADDR=0xFFFFFFFF, BRIDGE_TO=SOPB
```

BUS

The bus interface of a parameter is specified by the BUS keyword. It is expressed in the following format in which bus_label is a string:

```
PARAMETER C_SPLB_AWIDTH = 32, DT=datatype, BUS=bus_label
```

If you have more than one bus interface sharing the parameter, then use the colon to separate each bus interface in the list. The first item in the list is the default setting.

CACHEABLE

The CACHECABLE keyword identifies a cacheable address. The keyword is expressed in the following format:

```
PARAMETER C_BASEADDR=0xFFFFFFFF, CACHEABLE=TRUE
```

CLK_PORT

The CLK_PORT keyword can be used to specify the clock port to which this parameter is related. It is specified in the following format:

```
PARAMETER C_CLK_IN = 20, CLK_PORT = CLK_IN, CLK_UNIT=NS
```

CLK_UNIT

The CLK_UNIT keyword can be used to specify the units in which the parameter value is specified. Allowed values for units are: Hz, KHz, MHz, S, MS, US, NS and PS. If the CLK_PORT keyword is specified for a parameter, then the CLK_UNIT keyword *must* be specified.

DESC

The DESC keyword allows a short description of a parameter to be displayed by the GUI tools. The short description replaces the parameter name in the display field. The DESC keyword is expressed in the following format:

```
PARAMETER C_HAS_EXTERNAL_XIN=0, DT=integer, DESC="HAS XIN"
```

DT

The data type for a parameter is specified by the DT keyword, which is expressed in the following format:

```
PARAMETER C_SPLB_AWIDTH = 32, DT=datatype, BUS=bus_label
```

In this example, datatype can be assigned the values shown in the table below. The table also describes how the DT value is translated in the appropriate language.

Table 3-11: **DT Values**

DT Value	VHDL Type	Verilog Type
bit	bit	bit
bit_vector	bit_vector	bit vector
integer	integer	integer
real	real	real
string	string	string
std_logic	std_logic	bit
std_logic_vector	std_logic_vector	bit vector

IO_IF

The IO interface association name is expressed in the following format:

```
PARAMETER C_HAS_EXTERNAL_RCLK=0, IO_IF=uart_0, IO_IS=has_ext_rclk
```

IO_IS

The IO_IS keyword defines an XBD relationship marker. IO_IS implies that the parameter value can be dictated by a feature on the external hardware to which this IP is connected. Without a corresponding XBD value the tag has no effect.

A parameter that has an IO_IS must also have an IO_IF association with a particular IO_INTERFACE in the MPD. There are special values of IO_IS that do not have an IO_IF association. This occurs when those values are not specific to a particular IO_INTERFACE. These special values are clk_freq and polarity.

An MPD could have more than one IO_INTERFACE. Therefore, a parameter with an IO_IS must be associated with only one of them. This keyword is expressed in the following format:

```
PARAMETER C_FAMILY=virtex, IO_IF=uart_0, IO_IS=C_FAMILY
```

IPLEVEL_DRC_PROC

The IPLEVEL_DRC_PROC keyword defines the Tcl entry point for the IP-level DRC routine. A DRC based only on IP-level settings is done. The IPLEVEL_DRC_PROC keyword is expressed in the following format:

```
PARAMETER C_SPLB_AWIDTH = 32, IPLEVEL_DRC_PROC = proc_name
```

IPLEVEL_UPDATE_VALUE_PROC

The IPLEVEL_UPDATE_VALUE_PROC keyword defines the Tcl entry point for the IP-level update routine on parameters. An update based on only IP-level settings is done. The IPLEVEL_UPDATE_VALUE_PROC keyword is expressed in the following format:

```
PARAMETER C_SPLB_AWIDTH = 32, IPLEVEL_UPDATE_VALUE_PROC = proc_name
```

ISVALID

The ISVALID keyword defines the validity of a parameter to an expression. If the expression evaluates true, the PARAMETER is included in the list of valid PARAMETERs of the defined system for DRC processing. If false, the PARAMETER is not included, and no DRC is performed. However, the PARAMETER remains listed in the HDL. The ISVALID keyword is expressed in the following format:

```
PARAMETER C_BASEADDR = 0xFFFFFFFF, ISVALID = (C_PROC_INTRFCE==1)
```

If the ISVALID expression includes a string comparison, it cannot be specified using the == operator. Instead, the following Tcl procedure and syntax should be used.

```
PARAMETER C_MEM_PART_DATA_DEPTH = 0, DT = INTEGER,
  ISVALID = ([xstrncmp C_MEM_PARTNO CUSTOM])
```

In this example, the expression is valid when the value of the parameter C_MEM_PARTNO is CUSTOM.

LONG_DESC

This keyword allows a long description of the parameter to be displayed by the GUI tools. The long description allows the GUI tools to display a hover help. There is no default, and LONG_DESC is expressed in the following format:

```
PARAMETER C_HAS_EXTERNAL_XIN=0, DT=integer, LONG_DESC="XIN? What XIN?"
```

MIN_SIZE

The minimum size for an address window is specified by the MIN_SIZE keyword, which is expressed in the following format:

```
PARAMETER C_BASEADDR = 0xFFFFFFFF, DT=std_logic_vector, MIN_SIZE=0x100
```

PAIR

The PAIR keyword tags unidentified BASEADDR-HIGHADDR pairs. If non-standard names are used instead of *_BASEADDR and *_HIGHADDR, address parameters must identify pairs that define the BASE and HIGH. You must use the ADDRESS keyword to identify the parameter as BASE address or HIGH address. It is expressed in the following format:

```
PARAMETER C_HIGH=0x00000000, PAIR=C_BASE, ADDRESS=HIGH
PARAMETER C_BASE=0xFFFFFFFF, PAIR=C_HIGH, ADDRESS=BASE
```

PERMIT

This keyword specifies whether BSB should display and lets users edit the parameter. Allowed values are ADVANCED_USER (default) and BASE_USER. Only parameters that are tagged as BASE_USER are shown in BSB. PERMIT is expressed in the following format:

```
PARAMETER C_HAS_EXTERNAL_XIN=0, DT=integer, PERMIT = BASE_USER, ...
```

RANGE

RANGE defines a range of allowed valid values. It covers sequences like 8,16,24,32, or breaks in ranges, for example: RANGE=(1:4,8,16). The RANGE keyword is expressed in the following format:

```
PARAMETER C_HAS_EXTERNAL_XIN=0, DT=integer, RANGE=(0:1)
```

SYSLEVEL_DRC_PROC

The SYSLEVEL_DRC_PROC keyword defines the Tcl entry point for the system level DRC routine. A DRC based on only system-level settings is done. The SYSLEVEL_DRC_PROC keyword is expressed in the following format:

```
PARAMETER C_SPLB_AWIDTH = 32, SYSLEVEL_DRC_PROC = proc_name
```

SYSLEVEL_UPDATE_VALUE_PROC

The SYSLEVEL_UPDATE_VALUE_PROC keyword defines the Tcl entry point for the system-level update routine on parameters. The updates are based only on system-level settings and are expressed in the following format:

```
PARAMETER C_SPLB_AWIDTH = 32, SYSLEVEL_UPDATE_VALUE_PROC = proc_name
```

TYPE

The value of the TYPE keyword defines the kind of parameter. Allowed values are:

- HDL
  correlates to a generic for VHDL, or to a parameter for Verilog.
- NON_HDL
  this parameter is not present in the HDL of the IP, and it is tools specific.

If the TYPE keyword is not specified, the default is assumed to be HDL.

Note: The TYPE keyword was introduced to support NON_HDL parameters. Previously, there was a requirement that all parameters of an IP in the MHS and MPD files must be present in the HDL for the IP. However, there were cases in which the tools need user input (specified using parameters) for configuring the IP. Dummy parameters were therefore introduced in the HDL. You can now avoid this workaround by specifying those parameters as NON_HDL using the TYPE keyword.

VALUES

This keyword is a list of name-value pairs. The name is a parameter value in the range of allowed valid values. The value is a GUI-representable display string. The VALUES keyword is expressed in the following format:

```
PARAMETER C_ODD_PARITY=1, RANGE=(0:1), VALUES=(0=Even, 1=Odd)
```

Parameter Naming Conventions

An MPD parameter correlates to a generic for VHDL, or to a parameter for Verilog. The parameter name must be HDL (VHDL, Verilog) compliant. VHDL and Verilog have certain naming rules and conventions that must be followed.

The tools perform automatic clock DRCs. That is, they check to see if the frequency (or period) values specified for clock-related parameters in the MHS and MPD files are equal to the values computed by the tools. For this purpose, the tools rely on certain

sub-properties (CLK_PORT and CLK_UNIT) to be present for that parameter. If the tools do not find those sub-properties, they try to infer them from the name of the parameter.

If the parameter name follows the convention C_<clock_port>_FREQ_<clock_units> or C_<clock_port>_PERIOD_<clock_units>

The tools then infer the clock name and clock units for that parameter.

Examples:

(a) `PARAMETER C_PLB_Clk_FREQ_HZ = 50000000`

(b) `PARAMETER C_PLB_Clk_PERIOD_NS = 20`

In case (a), the tools infer that the parameter is related to OPB_Clk clock port and the clock unit is Hz, that is, frequency = 50,000,000 Hz. In case (b), the tools infer that the parameter is related to the PLB_Clk clock port and the clock unit is ns, that is, period = 20 ns.

Port

Definition

A port defines a data flow path that is passed into the entity (VHDL) or module (Verilog) declaration.

Port Keyword Summary

A port can have the following keywords:

ASSIGNMENT	ENABLE	LONG_DESC
BUFFER_TYPE	ENDIAN	PERMIT
BUS	INITIALVAL	SENSITIVITY
CLK_FACTOR	INTERRUPT_PRIORITY	SIGIS
CLK_INPORT	IOB_STATE	THREE_STATE
CLK_PHASE	IO_IF	TRI_I, TRI_O, and TRI_T
DESC	IO_IS	VEC
DIR	ISVALID	

Port Keyword Definitions

ASSIGNMENT

The ASSIGNMENT keyword defines assignment usage level. The keyword is expressed in the following format:

```
PORT SPLB_Clk="", DT=integer, ASSIGNMENT=REQUIRE
```

The table below lists ASSIGNMENT values.

Table 3-12: **ASSIGNMENT Values**

ASSIGNMENT	Definition
CONSTANT	The value is a constant. You cannot modify this value.
OPTIONAL	If you do not specify a value, the EDK batch tools use the default.
REQUIRE	You must specify a value.
UPDATE	You are not allowed to specify a value. The EDK batch tools compute the value.

BUFFER_TYPE

This keyword is expressed in the following format:

```
PORT CLK = "", DIR=I, BUFFER_TYPE=IBUF
```

If the BUFFER_TYPE exists on the MPD port, Platgen moves the property to the top-level port that is directly connected. If the BUFFER_TYPE exists on the top-level MHS port, it overrides any MPD port definition of BUFFER_TYPE.

The BUFFER_TYPE is translated into an XST pragma resident in the top-level <system> HDL.

Note: This constraint selects the type of buffer to be inserted on the input port or internal net. In general, you should avoid using this constraint and allow XST to infer the proper buffer. Avoiding this constraint allows for flexibility across device migration or synthesis tool selection.

For an EDK submodule flow, XST does not infer the buffer as defined by the BUFFER_TYPE. This is correct behavior since it is not expected that IO buffers be present for a submodule flow.

BUS

The bus interface association name is expressed in the following format, in which bus_label is a string:

```
PARAMETER C_PLB_NUM_MASTERS = 8, DT = INTEGER, BUS = MSPLB
```

If you have more than one bus interface sharing the parameter, then use the colon to separate each bus interface in the list. The first item in the list is the default setting, which is expressed in the following format:

```
PARAMETER C_PLB_NUM_MASTERS = 8, DT = INTEGER, BUS = MSPLB:SPLB
```

CLK_FACTOR

The factor by which the output clock varies with respect to the input clock can be specified with the keyword CLK_FACTOR, as follows:

```
PORT CLK2X = clk_out, SIGIS=CLK, DIR=O, CLK_FACTOR=2
```

The clock factor can also be specified in terms of other parameters, as follows:

```
PORT CLKFX = "", SIGIS=CLK, DIR=O, CLK_FACTOR = "1.0 * C_CLKFX_MULTIPLY
  / C_CLKFX_DIVIDE"
```

Note: If the CLK_FACTOR expression includes a division, it is *required* that the clock factor be multiplied by 1.0 to ensure that Tcl does not trim off the decimals.

CLK_INPORT

This keyword should only be specified on clock ports when DIR=O (that is, output clock ports). The input clock port, on which the output clock ports value is dependent, is specified using the keyword CLK_INPORT, as follows:

```
PORT CLK_OUT = clk_out, CLK_INPORT=CLK_IN
```

If there is only one input clock in an IP, then the tools automatically infer it to be the input clock. If multiple input clocks are present, this keyword *must* be used so the tools can determine clock connectivity.

CLK_PHASE

This keyword specifies the phase information for the clock. Allowed values: 0 to 360.

```
PORT CLK_IN = sys_clk_s, CLK_FREQ=100000000, CLK_PHASE=180
```

DESC

This keyword allows a short description of the port to be displayed by the GUI tools. The short description replaces the port name in the display field. The DESC keyword is expressed in the following format:

```
PORT SPLB_Clk="", DIR=IN, SIGIS=CLK, BUS=SPLB, DESC="SPLB clock"
```

DIR

The driver direction of a signal is specified by the DIR keyword. The keyword is expressed in the following format:

```
PORT mysignal = "", DIR=direction
```

In this example, direction is either I, O, or IO.

ENABLE

The ENABLE keyword allows you to specify whether tri-state signals have multi-bit enable control or single-bit enable control on the bus. The keyword is expressed in the following format:

```
PORT mysignal = "", DIR=IO, VEC=[0:31], ENABLE=enable_value
```

In this example, *enable_value* is either SINGLE or MULTI. If there is no specification, then SINGLE is the default value.

Refer to the "Design Considerations" section of this chapter for information about designing tri-state signals at the HDL level.

ENDIAN

The endianess of a signal is specified by the ENDIAN keyword, which is expressed in the following format:

```
PORT mysignal = "", DIR=I, VEC=[A:B], ENDIAN=endian_value
```

In this example, endian_value is either BIG or LITTLE. If there is no specification, BIG is the default value. A and B are positive integer expressions.

Note: ENDIAN is not used to define the endianess of a signal; it is used as a hint to Platgen to correctly construct the port vector. Since we use [A:B] style syntax for VEC definitions, it is not known when writing VHDL that it should be `std_logic_vector(A to B)`. This is used in a situation in which A=B. For example, VEC=[0:0] in the port declaration of components allows Platgen to properly define the datatype of the port.

INITIALVAL

The signal driver value on unconnected input signals is specified by the INITIALVAL keyword, which is expressed in the following format:

```
PORT mysignal = "", DIR=INPUT, INITIALVAL=init_value
```

In this example, the init_value is either VCC or GND. If there is no specification, then GND is the default value.

INTERRUPT_PRIORITY

The INTERRUPT_PRIORITY keyword defines the relative priority of interrupt signals. The keyword is expressed in the following format:

```
PORT Intr="", DIR=O, SENSITIVITY=EDGE_RISING, SIGIS=INTERRUPT,
INTERRUPT_PRIORITY=LOW
```

The level is dependent on the speed of the interface that the IP controls. For example, a UART runs at default 19200 baud, which gives a byte-rate of around 2000 bytes/s. An Ethernet 100 runs at 100 MHz, which gives a byte-rate of 12 000 000 bytes/s. Therefore, UART is LOW and Ethernet is HIGH.

CANBus runs at 1 MHz and gives a byte-rate of 120 000 bytes/s, which would be MEDIUM. It is also dependent on whether or not the IP has FIFO, which is a judgment that the designer must make.

IOB_STATE

Deprecated. If necessary, Platgen allows XST to infer all IOB primitives.

Currently, used for DRC purposes to guard against specific IO connectivity issues:

- INOUT ports must have the same port name and connector name if the pcore embeds an IOBUF. Use THREE_STATE=FALSE to check for condition (remains as a Platgen DRC).
- An INPUT port must have a connector if the pcore embeds an IBUF. Use Tcl SYSLEVEL_DRC_PROC to check the condition.

The IOB_STATE keyword identifies ports that instantiate or infer IOB primitives. The keyword is expressed in the following format:

```
PORT DDR_Addr = "", DIR=OUT, VEC=[0:C_DDR_AWIDTH-1], IOB_STATE=REG
```

The values are BUF, INFER, or REG. The default is INFER.

When a port requires an IOB primitive (IOB_STATE=INFER), Platgen instantiates an IOB buffer. When a port has an IOB buffer (IOB_STATE=BUF) or IOB register (IOB_STATE=REG), Platgen does not instantiate an IOB primitive.

IO_IF

The IO interface association name is formatted as follows:

```
PORT C405TRCCYCLE="", DIR=O, IO_IF=trace_0, IO_IS=trace_clk
```

IO_IS

The IO_IS keyword defines an XBD relationship marker. IO_IS implies that the port can be pulled out to the FPGA boundary.

Without a corresponding XBD value, the tag has no effect.

A port that has an IO_IS must also have an IO_IF association with a particular IO_INTERFACE in the MPD. An MPD could have more than one IO_INTERFACE. Therefore, a port with an IO_IS must be associated with only one of them. The IO_IS keyword is expressed in the following format:

```
PORT C405TRCCYCLE="", DIR=O, IO_IF=trace_0, IO_IS=trace_clk
```

ISVALID

The ISVALID keyword defines the validity of a PORT to an expression. If the expression evaluates true, the PORT is included in the list of valid PORTs of the defined system for DRC processing. If false, the PORT is not included and no DRC is performed. However, the PORT remains listed in the HDL. It is expressed in the following format:

```
PORT SDRAM_RAS_n = "", DIR = O, ISVALID = (C_MEM_TYPE == 2)
```

LONG_DESC

Allows a long description of the port to be displayed by the GUI tools. The long description allows the GUI tools to display a hover help. There is no default and the use of this keyword is formatted as follows:

```
PORT OPB_Clk="", DIR=I, SIGIS=CLK, BUS=SOPB, LONG_DESC="Clock from OPB"
```

PERMIT

The PERMIT keyword determines whether or not BSB should specify a particular port as a top-level port in the MHS. Only those ports that have a PERMIT value of BASE_USER are connected by BSB to top-level ports. The default value is ADVANCED_USER. The PERMIT keyword is expressed in the following format:

```
PORT JTGC405TCK=JTGC405TCK, DIR=I, PERMIT = BASE_USER
```

SENSITIVITY

The interrupt sensitivity of an interrupt signal is specified by the SENSITIVITY keyword. This supersedes the EDGE and LEVEL keywords. The SENSITIVITY keyword is expressed in the following format:

```
PORT interrupt="", DIR=O, SENSITIVITY=value, SIGIS=INTERRUPT
```

In this example, the value is either EDGE_FALLING, EDGE_RISING, LEVEL_HIGH or LEVEL_LOW.

SIGIS

The class of a signal is specified by the SIGIS keyword, which is expressed in the following format:

```
PORT mysig="", DIR=O, SIGIS=value
```

In this example, the value is CLK, INTERRUPT, or RST. The table below describes SIGIS usage.

Table 3-13: **SIGIS Usage**

SIGIS	Usage
CLK	• XPS Displays all clock signals. • Platgen For all bus components, the clock signals are automatically connected to the clock input of the peripherals on the bus.
INTERRUPT	• XPS Displays all interrupt signals. • Platgen Encodes the priority interrupt vector
RST	• XPS Displays all reset signals.

THREE_STATE

The THREE_STATE keyword enables or disables tri-state expansion. This supersedes the deprecated 3STATE keyword. The THREE_STATE keyword is expressed in the following format:

```
PORT PAR="", DIR=INOUT, THREE_STATE=FALSE, IOB_STATE=BUF
```

For output ports, the default value is FALSE. For inout ports, the default value is TRUE.

Refer to the "Tri-state (InOut and Output) Signals" section of this chapter for information about designing tri-state signals at the HDL level.

TRI_I, TRI_O, and TRI_T

The TRI_I, TRI_O, and TRI_T keywords define the associative ports of the tri-state. This directly correlates to the IOBUF UNISIM primitive where the I port is represented by the TRI_O keyword, the T port by TRI_T, and the O port by the TRI_I. This also directly correlates to the OBUFT UNISIM primitive where the I port is represented by the TRI_O keyword, the T port by TRI_T, and the TRI_I keyword is not defined. These properties are only valid when THREE_STATE=TRUE. The TRI_I, TRI_O, and TRI_T keywords are expressed in the following format:

```
PORT IPIO = "", DIR=INOUT, TRI_T=x, TRI_O=y, TRI_T=z, THREE_STATE=TRUE
PORT x = "", DIR=OUT
PORT y = "", DIR=OUT
PORT z = "", DIR=IN
```

Refer to the "Tri-state (InOut and Output) Signals" section of this chapter for information about designing tri-state signals at the HDL level.

VEC

The vector width of a signal is specified by the VEC keyword, which is expressed in the following format:

```
PORT mysignal = "", DIR=INPUT, VEC=[A:B]
```

A and B are positive integer expressions.

Port Naming Conventions

This section provides naming conventions for bus interface signal names. These conventions are flexible to accommodate embedded processor systems that have more than one bus interface and more than one bus interface port per component.

The names must be HDL (VHDL or Verilog) compliant. As with any language, VHDL and Verilog have certain naming rules and conventions that you must follow.

Global Ports

The names for the global ports of a peripheral (such as clock and reset signals) are standardized. You can use any name for other global ports (such as the interrupt signal).

LMB - Clock and Reset

```
LMB_Clk
LMB_Rst
```

PLB - Clock and Reset

```
PLB_Clk
PLB_Rst
```

Slave DCR Ports

Naming conventions should be followed for that part of the identifier following the last underscore in the name.

DCR Slave Outputs

For interconnection to the DCR, all slaves must provide the following outputs in which <Sln> is a meaningful name or acronym for the slave output:

```
<Sln>_dcrDBus
<Sln>_dcrAck
```

An additional requirement on <Sln> is that it must not contain the string DCR (upper, lower, or mixed case), so that slave outputs are not confused with bus outputs.

```
uart_dcrAck
intc_dcrAck
memcon_dcrAck
```

DCR Slave Inputs

For interconnection to the DCR, all slaves must provide the following inputs:

```
<nDCR>_ABus
<nDCR>_Sl_DBus
<nDCR>_Read
<nDCR>_Write
```

In this example, <nDCR> is a meaningful name or acronym for the slave input. An additional requirement on <nDCR> is that the last three characters must contain the string DCR (upper, lower, or mixed case).

```
DCR_Sl_DBus
bus1_DCR_Sl_DBus
```

Slave LMB Ports

Naming conventions should be followed for that part of the identifier following the last underscore in the name.

LMB Slave Outputs

For interconnection to the LMB, all slaves must provide the following outputs:

```
<Sln>_DBus
<Sln>_Ready
```

In this example, <*Sln*> is a meaningful name or acronym for the slave output. An additional requirement on <*Sln*> is that it must not contain the string "LMB" (upper, lower, or mixed case), so that slave outputs are not confused with bus outputs.

```
d_Ready
i_Ready
```

LMB Slave Inputs

For interconnection to the LMB, all slaves must provide the following inputs:

```
<nLMB>_ABus
<nLMB>_ReadStrobe
<nLMB>_AddrStrobe
<nLMB>_WriteStrobe
<nLMB>_WriteDBus
<nLMB>_BE
```

In this example, <nLMB> is a meaningful name or acronym for the slave input. An additional requirement on <nLMB> is that the last three characters must contain the string LMB (upper, lower, or mixed case).

```
LMB_ABus
bus1_LMB_ABus
```

Master PLB Ports

Naming conventions should be followed for that part of the identifier following the last underscore in the name.

PLB Master Outputs

For interconnection to the PLB, all masters must provide the following outputs:

```
<Mn>_ABus
<Mn>_BE
<Mn>_RNW
<Mn>_abort
<Mn>_busLock
<Mn>_compress
<Mn>_guarded
<Mn>_lockErr
<Mn>_MSize
```

<Mn>_ordered
<Mn>_priority
<Mn>_rdBurst
<*Mn*>_request
<*Mn*>_size
<*Mn*>_type
<Mn>_wrBurst
<Mn>_wrDBus

In this example, <Mn> is a meaningful name or acronym for the master output. An additional requirement for <Mn> is that it must not contain the string PLB (upper, lower, or mixed case), so that master outputs are not confused with bus outputs.

iM_request
bridge_request
o2ob_request

PLB Master Inputs

For interconnection to the PLB, all masters must provide the following inputs:

<*nPLB*>_MAddrAck
<*nPLB*>_MBusy
<*nPLB*>_MErr
<*nPLB*>_MRdBTerm
<*nPLB*>_MRdDAck
<*nPLB*>_MRdDBus
<*nPLB*>_MRdWdAddr
<*nPLB*>_MRearbitrate
<*nPLB*>_MWrBTerm
<*nPLB*>_MWrDAck
<*nPLB*>_MSSize

In this example, <nPLB> is a meaningful name or acronym for the master input. An additional requirement on <nPLB> is that the last three characters must contain the string PLB (upper, lower, or mixed case).

iPLB_MBusy
PLB_MBusy
bus1_PLB_MBusy

Slave PLB Ports

Naming conventions should be followed for that part of the identifier following the last underscore in the name.

PLB Slave Outputs

For interconnection to the PLB, all slaves must provide the following outputs:

<*Sln*>_addrAck
<*Sln*>_MErr
<*Sln*>_MBusy
<*Sln*>_rdBTerm
<*Sln*>_rdComp
<*Sln*>_rdDAck
<*Sln*>_rdDBus
<*Sln*>_rdWdAddr
<*Sln*>_rearbitrate
<*Sln*>_SSize
<*Sln*>_wait

```
<Sln>_wrBTerm
<Sln>_wrComp
<Sln>_wrDAck
```

In this example, `<Sln>` is a meaningful name or acronym for the slave output. An additional requirement on `<Sln>` is that it must not contain the string `PLB` (upper, lower, or mixed case), so that slave outputs are not confused with bus outputs.

```
tmr_addrAck
uart_addrAck
intc_addrAck
```

PLB Slave Inputs

For interconnection to the PLB, all slaves must provide the following inputs:

```
<nPLB>_ABus
<nPLB>_BE
<nPLB>_PAValid
<nPLB>_RNW
<nPLB>_abort
<nPLB>_busLock
<nPLB>_compress
<nPLB>_guarded
<nPLB>_lockErr
<nPLB>_masterID
<nPLB>_MSize
<nPLB>_ordered
<nPLB>_pendPri
<nPLB>_pendReq
<nPLB>_reqPri
<nPLB>_size
<nPLB>_type
<nPLB>_rdPrim
<nPLB>_SAValid
<nPLB>_wrPrim
<nPLB>_wrBurst
<nPLB>_wrDBus
<nPLB>_rdBurst
```

In this example, <*nPLB*> is a meaningful name or acronym for the slave input. An additional requirement on <*nPLB*> is that the last three characters must contain the string `PLB` (upper, lower, or mixed case).

```
PLB_size
iPLB_size
dPLB_size
```

Reserved Parameters

Reserved Parameter Names Summary

The EDK tools automatically expand and populate a defined set of reserved parameters. This can help prevent errors when your peripheral requires information on the platform that is generated. The following list contains the reserved parameter names.

- C_DEVICE
- C_PACKAGE
- C_SPEEDGRADE
- C_FAMILY
- C_INSTANCE
- C_SUBFAMILY
- C_<BUS NAME>_AWIDTH
- C_<BUS NAME>_DWIDTH
- C_<BUS NAME>_NUM_SLAVES
- C_<BUS NAME>_MASTERS_SLAVES
- C_<BUS NAME>_MID_WIDTH

Reserved Parameter Descriptions

C_DEVICE

The C_DEVICE parameter defines the FPGA device. This parameter is automatically populated by the EDK tools, and is formatted as follows:

```
PARAMETER C_DEVICE = "4vlx100", DT=string
```

C_PACKAGE

The C_PACKAGE parameter defines the FPGA device package. This parameter is automatically populated by the EDK tools, and is formatted as follows:

```
PARAMETER C_PACKAGE = "ff1016", DT=string
```

C_SPEEDGRADE

The C_SPEEDGRADE parameter defines the FPGA device speed grade. This parameter is automatically populated by the EDK tools, and is formatted as follows:

```
PARAMETER C_SPEEDGRADE = "-11", DT=string
```

C_FAMILY

The C_FAMILY parameter defines the FPGA device family. This parameter is automatically populated by the EDK tools, and is formatted as follows:

```
PARAMETER C_FAMILY = "virtex4", DT=string
```

C_INSTANCE

The C_INSTANCE parameter defines the instance name of the component. This parameter is automatically populated by the EDK tools, and is formatted as follows:

```
PARAMETER C_INSTANCE = instance_name, DT=string
```

C_SUBFAMILY

The C_SUBFAMILY parameter defines the FPGA subfamily specification. C_SUBFAMILY will have an FX, LX, or SX designation for VIRTEX4, an LX or SX designation for VIRTEX5, and will be empty "" for other architectures.

This parameter is automatically populated by the EDK tools, and is formatted as follows:

```
PARAMETER C_SUBFAMILY = "fx", DT=string
```

C_<*BUS NAME*>_AWIDTH

The C_<BUS NAME>_AWIDTH parameter is automatically populated by the EDK tools, and is formatted as follows:

```
PARAMETER C_<BUS NAME>_AWIDTH = integer, DT=integer
```

The <BUS NAME> is replaced with the name of the BUS_STD in use or the name of the BUS_INTERFACE in use.

C_<*BUS NAME*>_DWIDTH

The C_<BUS NAME>_DWIDTH parameter is automatically populated by the EDK tools, and is formatted as follows:

```
PARAMETER C_<BUS NAME>_DWIDTH = integer, DT=integer
```

The <BUS NAME> is replaced with the name of the BUS_STD in use, or the name of the BUS_INTERFACE in use.

C_<*BUS NAME*>_NUM_SLAVES

The C_<BUS NAME>_NUM_SLAVES parameter is automatically populated by the EDK tools, and is formatted as follows:

```
PARAMETER C_<BUS NAME>_NUM_SLAVES = integer, DT=integer
```

The <BUS NAME> is replaced with the name of the BUS_STD in use, or the name of the BUS_INTERFACE in use.

C_<*BUS NAME*>_MASTERS_SLAVES

The C_<BUS NAME>_MASTERS_SLAVES parameter is automatically populated by the EDK tools, and is formatted as follows:

```
PARAMETER C_<BUS NAME>_NUM_MASTERS = integer, DT=integer
```

The <BUS NAME> is replaced with the name of the BUS_STD in use or the name of the BUS_INTERFACE in use.

C_<BUS NAME>_MID_WIDTH

The C_<BUS NAME>_MID_WIDTH parameter defines the PLB master ID width in bits. This is determined by the number of PLB masters, as shown in the table below.

Table 3-14: **C_<BUS NAME>_MID_WIDTH Calculation**

C_<BUS NAME>_NUM_MASTERS (Number of PLB Masters)	C_<BUS NAME>_MID_WIDTH
0 to 2	1
3 to 4	2
5 to 8	3
9 to 16	4

This parameter is automatically populated by the EDK tools and is formatted as follows:

```
PARAMETER C_<BUS NAME>_MID_WIDTH = <num>, DT=integer
```

In this example, <num> is an integer value. The <BUS NAME> is replaced with the name of the BUS_STD or BUS_INTERFACE in use.

Reserved Port Connections

Connectivity of the DCR, LMB, OPB, and PLB busses to peripherals is done through a common set of signal connections.

Clock and Reset Ports

For interconnection to the clock and reset ports:

```
PORT <BUS NAME>_Clk = "", DIR=I, SIGIS=CLK
PORT <BUS NAME>_Rst = <BUS NAME>_Rst, DIR=I
```

The <BUS NAME> is replaced with the name of the BUS_STD in use or the name of the BUS_INTERFACE in use. Notice that the clock port has no default value. The clock port is an input to the bus that you assign in the MHS. Therefore, all peripherals on the bus must also be treated as a user input port. If a default value were given to <BUS NAME>_Clk, this would not match the clock you defined in the MHS, and the EDK tools would assume a short in the system and tie off the sourceless ports.

The reset port is an output from the bus and has a default value. All peripherals on the bus share the same default: <BUS NAME>_Rst. Your input to the bus is SYS_Rst, which has no default value.

Slave LMB Ports

For interconnection to the LMB, all slaves must provide the following connections:

```
PORT <Sln>_DBus = Sl_DBus, DIR=O, VEC=[0:C_LMB_DWIDTH-1], BUS=SLMB
PORT <Sln>_Ready = Sl_Ready, DIR=O, BUS=SLMB
PORT <nLMB>_ABus = LMB_ABus, DIR=I, VEC=[0:C_LMB_AWIDTH-1], BUS=SLMB
PORT <nLMB>_ReadStrobe = LMB_ReadStrobe, DIR=I, BUS=SLMB
PORT <nLMB>_AddrStrobe = LMB_AddrStrobe, DIR=I, BUS=SLMB
PORT <nLMB>_WriteStrobe = LMB_WriteStrobe, DIR=I, BUS=SLMB
PORT <nLMB>_WriteDBus = LMB_WriteDBus, DIR=I, VEC=[0:C_LMB_DWIDTH-1],
BUS=SLMB
PORT <nLMB>_BE = LMB_BE, DIR=I, VEC=[0:C_LMB_DWIDTH/8-1], BUS=SLMB
```

Master PLB Ports

For interconnection to the PLB, all masters must provide the following connections:

```
PORT <Mn>_ABus = M_ABus, DIR=O, VEC=[0:C_PLB_AWIDTH-1], BUS=MPLB
PORT <Mn>_BE = M_BE, DIR=O, VEC=[0:C_PLB_DWIDTH/8-1], BUS=MPLB
PORT <Mn>_RNW = M_RNW, DIR=O, BUS=MPLB
PORT <Mn>_abort = M_abort, DIR=O, BUS=MPLB
PORT <Mn>_busLock = M_busLock, DIR=O, BUS=MPLB
PORT <Mn>_compress = M_compress, DIR=O, BUS=MPLB
PORT <Mn>_guarded = M_guarded, DIR=O, BUS=MPLB
PORT <Mn>_lockErr = M_lockErr, DIR=O, BUS=MPLB
PORT <Mn>_MSize = M_MSize, DIR=O, VEC=[0:1], BUS=MPLB
PORT <Mn>_ordered = M_ordered, DIR=O, BUS=MPLB
PORT <Mn>_priority = M_priority, DIR=O, VEC=[0:1], BUS=MPLB
PORT <Mn>_rdBurst = M_rdBurst, DIR=O, BUS=MPLB
PORT <Mn>_request = M_request, DIR=O, BUS=MPLB
PORT <Mn>_size = M_size, DIR=O, VEC=[0:3], BUS=MPLB
PORT <Mn>_type = M_type, DIR=O, VEC=[0:2], BUS=MPLB
PORT <Mn>_wrBurst = M_wrBurst, DIR=O, BUS=MPLB
PORT <Mn>_wrDBus = M_wrDBus, DIR=O, VEC=[0:C_PLB_DWIDTH-1], BUS=MPLB
PORT <nPLB>_MAddrAck = PLB_MAddrAck, DIR=I, BUS=MPLB
PORT <nPLB>_MBusy = PLB_MBusy, DIR=I, BUS=MPLB
PORT <nPLB>_MErr = PLB_MErr, DIR=I, BUS=MPLB
PORT <nPLB>_MRdBTerm = PLB_MRdBTerm, DIR=I, BUS=MPLB
PORT <nPLB>_MRdDAck = PLB_MRdDAck, DIR=I, BUS=MPLB
PORT <nPLB>_MRdDBus = PLB_MRdDBus, DIR=I, VEC=[0:C_PLB_DWIDTH-1],
BUS=MPLB
PORT <nPLB>_MRdWdAddr = PLB_MRdWdAddr, DIR=I, VEC=[0:3], BUS=MPLB
PORT <nPLB>_MRearbitrate = PLB_MRearbitrate, DIR=I, BUS=MPLB
PORT <nPLB>_MWrBTerm = PLB_MWrBTerm, DIR=I, BUS=MPLB
PORT <nPLB>_MWrDAck = PLB_MWrDAck, DIR=I, BUS=MPLB
PORT <nPLB>_MSSize = PLB_MSSize, DIR=I, VEC=[0:1], BUS=MPLB
```

Slave PLB Ports

For interconnection to the PLB, all slaves must provide the following connections:

```
PORT <Sln>_addrAck = Sl_addrAck, DIR=O, BUS=SPLB
PORT <Sln>_MErr = Sl_MErr, DIR=O, VEC=[0:C_NUM_MASTERS-1], BUS=SPLB
PORT <Sln>_MBusy = Sl_MBusy, DIR=O, VEC=[0:C_NUM_MASTERS-1], BUS=SPLB
PORT <Sln>_rdBTerm = Sl_rdBTerm, DIR=O, BUS=SPLB
PORT <Sln>_rdComp = Sl_rdComp, DIR=O, BUS=SPLB
PORT <Sln>_rdDAck = Sl_rdDAck, DIR=O, BUS=SPLB
PORT <Sln>_rdDBus = Sl_rdDBus, DIR=O, VEC=[0:C_PLB_DWIDTH-1],BUS=SPLB
PORT <Sln>_rdWdAddr = Sl_rdWdAddr, DIR=O, VEC=[0:3], BUS=SPLB
PORT <Sln>_rearbitrate = Sl_rearbitrate, DIR=O, BUS=SPLB
PORT <Sln>_SSize = Sl_SSize, DIR=O, VEC=[0:1], BUS=SPLB
PORT <Sln>_wait = Sl_wait, DIR=O, BUS=SPLB
PORT <Sln>_wrBTerm = Sl_wrBTerm, DIR=O, BUS=SPLB
PORT <Sln>_wrComp = Sl_wrComp, DIR=O, BUS=SPLB
PORT <Sln>_wrDAck = Sl_wrDAck, DIR=O, BUS=SPLB
PORT <nPLB>_ABus = PLB_ABus, DIR=I, VEC=[0:C_PLB_AWIDTH-1], BUS=SPLB
PORT <nPLB>_BE = PLB_BE, DIR=I, VEC=[0:(C_PLB_DWIDTH/8)-1], BUS=SPLB
PORT <nPLB>_PAValid = PLB_PAValid, DIR=I, BUS=SPLB
PORT <nPLB>_RNW = PLB_RNW, DIR=I, BUS=SPLB
PORT <nPLB>_abort = PLB_abort, DIR=I, BUS=SPLB
PORT <nPLB>_busLock = PLB_busLock, DIR=I, BUS=SPLB
PORT <nPLB>_compress = PLB_compress, DIR=I, BUS=SPLB
PORT <nPLB>_guarded = PLB_guarded, DIR=I, BUS=SPLB
PORT <nPLB>_lockErr = PLB_lockErr, DIR=I, BUS=SPLB
PORT <nPLB>_masterID = PLB_masterID, DIR=I,VEC=[0:C_PLB_MID_WIDTH-1],
BUS=SPLB
PORT <nPLB>_MSize = PLB_MSize, DIR=I, VEC=[0:1], BUS=SPLB
PORT <nPLB>_ordered = PLB_ordered, DIR=I, BUS=SPLB
PORT <nPLB>_pendPri = PLB_pendPri, DIR=I, VEC=[0:1], BUS=SPLB
PORT <nPLB>_pendReq = PLB_pendReq, DIR=I, BUS=SPLB
PORT <nPLB>_reqPri = PLB_reqPri, DIR=I, VEC=[0:1], BUS=SPLB
PORT <nPLB>_size = PLB_size, DIR=I, VEC=[0:3], BUS=SPLB
PORT <nPLB>_type = PLB_type, DIR=I, VEC=[0:2], BUS=SPLB
PORT <nPLB>_rdPrim = PLB_rdPrim, DIR=I, BUS=SPLB
PORT <nPLB>_SAValid = PLB_SAValid, DIR=I, BUS=SPLB
PORT <nPLB>_wrPrim = PLB_wrPrim, DIR=I, BUS=SPLB
PORT <nPLB>_wrBurst = PLB_wrBurst, DIR=I, BUS=SPLB
PORT <nPLB>_wrDBus = PLB_wrDBus, DIR=I, VEC=[0:C_PLB_DWIDTH-1],BUS=SPLB
PORT <nPLB>_rdBurst = PLB_rdBurst, DIR=I, BUS=SPLB
```

Design Considerations

Unconnected Ports

Unconnected output ports are assigned open, and unconnected input ports are either set to GND or VCC.

An unconnected port is identified as an empty double-quote (“”) string.

The EDK tools resolve the driver value on unconnected input ports by the `INITIALVAL` keyword, using this format:

```
PORT mysignal = “”, DIR=OUTPUT
```

Scalable Data Path

Using an MPD keyword declaration, you can automatically scale the data path width. Bus expressions are evaluated as arithmetic equation, and are formatted as follows.

```
PORT name = default_connection, VEC=[A:B]
```

In this example, A and B are positive integer expressions.

MPD Example

The following is an example MPD file:

```
BEGIN my_peripheral
# Generics for vhdl or parameters for verilog
PARAMETER C_BASEADDR = 0xB00000, DT=std_logic_vector(0 to 31)
PARAMETER C_MY_PERIPH_AWIDTH = 17, DT=integer
# Global ports
PORT OPB_Clk = "", DIR=I
PORT OPB_Rst = "", DIR=I
# My peripheral signals
PORT MY_ADDR = "", DIR=O, VEC=[0:C_MY_PERIPH_AWIDTH-1]
# OPB signals
.
.
END
```

By default, if the vectors are larger than one bit, EDK tools determine the range specification on buses as either big-endian or little-endian. However, if the vector has a one-bit width, then the range cannot be determined, and the EDK tools default to big-endian style notation.

To change this default behavior, use the `ENDIAN` keyword, which should be formatted as follows:

```
PORT mysignal = "", DIR=I, VEC=[0:0], ENDIAN=LITTLE
```

This builds the VHDL equivalent:

```
mysignal: in std_logic_vector(0 downto 0);
```

Interrupt Signals

Interrupt signals are identified by the `SIGIS=INTERRUPT` name-value.

Tri-state (InOut and Output) Signals

Note: Read this section if you want the EDK tools to infer a tri-state port for your output or in/out port.

A system on a programmable chip design methodology follows these general rules of thumb:

- Submodule port driver directions (modes) should be either `IN` or `OUT`
- Top level module/entity is allowed to have ports of mode `INOUT`

The drive direction (mode) of a port impacts the partitioning of a design. The mode of a port must propagate through all levels of hierarchy, with the result that if the top-level requests an inout port, then a low-level module must provide an inout port for connectivity. Alternatively, at the top-level hierarchy, the user must describe the inout

drive direction to connect the lower-level, unidirectional ports of the submodule to the top-level bi-directional inout port.

This methodology fits well into an FGPA architecture since tri-state buffers are only available as an IOBUF primitive in the IOB cell. A reduced implementation is a tri-output, which Platgen maps to the OBUFT primitive. Tri-states in the CLB do not exist; the synthesis tool translates the tri-state logic to MUXes.

An abstract inout port on the MPD is defined for connectivity purposes. The abstract port allows a user to connect the top-level inout port to the lower-level abstract inout port without changing the partition or interface of the submodule in hardware.

At the MHS/MPD level, there is an abstract inout port in the MPD file that allows a connection through the IOBUF to the top-level inout port declaration in the MHS file. This corresponds to the usage of defining an inout port at the top level and preserving unidirectional ports at the lower level.

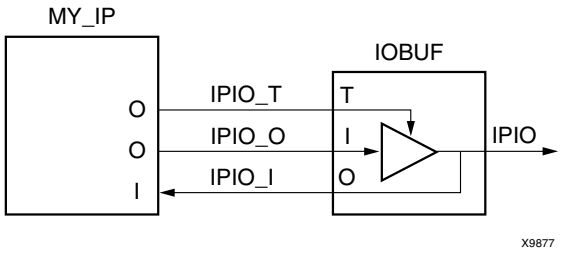

Figure 3-1: **OBUF Implementation**

Note: The tri-state enable is active-low. This allows a direct connection to the OBUFT or the IOBUF without an inversion of the tri-state enable port.

The IPIO port in Figure 3-1 is described as an abstract port of drive direction inout. This port is not listed on the port interface of the hardware module or entity, as demonstrated in the following HDL code examples.

In the MPD file, an abstract inout port is identified by the inout direction mode and THREE_STATE=TRUE without defined TRI_I, TRI_O, and TRI_T keywords. In this case, the abstract inout port name must share a common basename across the basename_I, basename_O, and basename_T ports on the port interface of the hardware module or entity. In Figure 3-1, the basename is IPIO. Platgen expands the inout port in the MPD file to _I, _O, and _T ports in the port interface declaration of the HDL file. This method does not allow the individual ports that construct the abstract port to be listed in the MPD.

In the MPD file, an abstract inout port is identified by the inout or output (tri-output) direction mode and THREE_STATE=TRUE with defined TRI_I, TRI_O, and TRI_T keywords. In this case, the abstract inout port name allows free connection to the individual ports that construct the abstract port. The abstract inout port or output (tri-output) is freely named. This method does allow the individual ports that construct the abstract port to be listed in the MPD.

Tri-state (InOut) With Single-Bit Enable

The following examples include a tri-state port with a single-bit enable.

VHDL Example

```
entity tri_state_single is
generic (C_WIDTH: integer:= 9);
port (
     -- tri-state signal
    IPIO_I: in std_logic_vector(0 to C_WIDTH-1);
    IPIO_O: out std_logic_vector(0 to C_WIDTH-1);
    IPIO_T: out std_logic);
end entity tri_state_single;
```

MPD Example

```
BEGIN tri_state_single
OPTION IPTYPE=IP
PARAMETER C_WIDTH=9, DT=integer
PORT IPIO = "", DIR=INOUT, VEC=[0:C_WIDTH-1], ENABLE=SINGLE,
THREE_STATE=TRUE
END
```

Tri-state (InOut) With Multi-Bit Enable

The following examples include a tri-state port with a multi-bit enable.

VHDL Example

```
entity tri_state_multi is
generic (C_WIDTH: integer:= 9);
port (
     -- tri-state signal
    IPIO_I: in std_logic_vector(0 to C_WIDTH-1);
    IPIO_O: out std_logic_vector(0 to C_WIDTH-1);
    IPIO_T: out std_logic_vector(0 to C_WIDTH-1));
end entity tri_state_multi;
```

MPD Example

```
BEGIN tri_state_multi
OPTION IPTYPE=IP
PARAMETER C_WIDTH = 9, DT=integer
PORT IPIO = "", DIR=INOUT, VEC=[0:C_WIDTH-1], ENABLE=MULTI,
THREE_STATE=TRUE
END
```

Tri-state (In/Out) With Single-Bit Enable With Freely Named Ports

The following examples include a tri-state port with a single-bit enable with freely named ports.

VHDL Example

```
entity tri_state_single is
generic (C_WIDTH: integer:= 9);
port (
     -- tri-state signal
    ITRI: in std_logic_vector(0 to C_WIDTH-1);
    OTRI: out std_logic_vector(0 to C_WIDTH-1);
    TTRI: out std_logic);
end entity tri_state_single;
```

MPD Example

```
BEGIN tri_state_single
OPTION IPTYPE=IP
PARAMETER C_WIDTH=9, DT=integer
PORT IPIO="", DIR=IO,VEC=[0:C_WIDTH-1],THREE_STATE=TRUE, TRI_I=ITRI,
TRI_O=OTRI, TRI_T=TTRI
PORT ITRI="", DIR=I, VEC=[0:C_WIDTH-1]
PORT OTRI="", DIR=O, VEC=[0:C_WIDTH-1]
PORT TTRI="", DIR=I
END
```

Tri-state (InOut) With Multi-Bit Enable With Freely Named Ports

These examples show a tri-state port with a multi-bit enable with freely-named ports.

VHDL Example

```
entity tri_state_single is
generic (C_WIDTH: integer:= 9);
port (
     -- tri-state signal
    ITRI: in std_logic_vector(0 to C_WIDTH-1);
    OTRI: out std_logic_vector(0 to C_WIDTH-1);
    TTRI: out std_logic(0 to C_WIDTH-1));
end entity tri_state_single;
```

MPD Example

```
BEGIN tri_state_single
OPTION IPTYPE=IP
PARAMETER C_W=9, DT=integer
PORT IPIO="", DIR=IO, VEC=[0:C_WIDTH-1], ENABLE=MULTI, TRI_I=ITRI,
TRI_O=OTRI, TRI_T=TTRI
PORT ITRI="", DIR=I, VEC=[0:C_WIDTH-1]
PORT OTRI="", DIR=O, VEC=[0:C_WIDTH-1]
PORT TTRI="", DIR=I, VEC=[0:C_WIDTH-1]
END
```

Tri-state (Output) With Single-Bit Enable

These examples include a tri-state output port with a single-bit enable.

VHDL Example

```
entity tri_state_output_single is
generic (C_WIDTH: integer:= 9);
port (
-- tri-state signal
IPO_O: out std_logic_vector(0 to C_WIDTH-1);
IPO_T: out std_logic);
end entity tri_state_output_single;
```

MPD Example

```
BEGIN tri_state_output_single
OPTION IPTYPE=IP
PARAMETER C_WIDTH=9, DT=integer
PORT IPO = "", DIR=O, VEC=[0:C_WIDTH-1], ENABLE=SINGLE,
THREE_STATE=TRUE
END
```

Tri-state (Output) With Multi-Bit Enable

These examples include a tri-state output port with a multi-bit enable.

VHDL Example

```
entity tri_state_output_multi is
generic (C_WIDTH: integer:= 9);
port (
-- tri-state signal
IPO_O: out std_logic_vector(0 to C_WIDTH-1);
IPO_T: out std_logic_vector(0 to C_WIDTH-1));
end entity tri_state_output_multi;
```

MPD Example

```
BEGIN tri_state_output_multi
OPTION IPTYPE=IP
PARAMETER C_WIDTH = 9, DT=integer
PORT IPO = "", DIR=O, VEC=[0:C_WIDTH-1], ENABLE=MULTI, THREE_STATE=TRUE
END
```

Tri-state (Output) With Single-Bit Enable With Freely Named Ports

These examples include a tri-state output port with a single-bit enable and with freely named ports.

VHDL Example

```
entity tri_state_output_single is
generic (C_WIDTH: integer:= 9);
port (
-- tri-state signal
OTRI: out std_logic_vector(0 to C_WIDTH-1);
TTRI: out std_logic);
end entity tri_state_output_single;
```

MPD Example

```
BEGIN tri_state_output_single
OPTION IPTYPE=IP
PARAMETER C_WIDTH=9, DT=integer
PORT IPO="", DIR=O,VEC=[0:C_WIDTH-1],THREE_STATE=TRUE, TRI_O=OTRI,
TRI_T=TTRI
PORT OTRI="", DIR=O, VEC=[0:C_WIDTH-1]
PORT TTRI="", DIR=I
END
```

Tri-state (Output) With Multi-Bit Enable With Freely Named Ports

These examples show a tri-state output port with a multi-bit enable and with freely named ports.

VHDL Example

```
entity tri_state_output_single is
generic (C_WIDTH: integer:= 9);
port (
-- tri-state signal
OTRI: out std_logic_vector(0 to C_WIDTH-1);
TTRI: out std_logic(0 to C_WIDTH-1));
end entity tri_state_output_single;
```

MPD Example

```
BEGIN tri_state_output_single
OPTION IPTYPE=IP
PARAMETER C_W=9, DT=integer
PORT IPO="", DIR=O, VEC=[0:C_WIDTH-1], ENABLE=MULTI, TRI_O=OTRI,
TRI_T=TTRI
PORT OTRI="", DIR=O, VEC=[0:C_WIDTH-1]
PORT TTRI="", DIR=I, VEC=[0:C_WIDTH-1]
END
```

XILINX®

Chapter 4

Peripheral Analyze Order (PAO)

A PAO (Peripheral Analyze Order) file contains a list of HDL files that are needed for synthesis and defines the analyze order for compilation.

If the STYLE option in the MPD file has a value of MIX or HDL, the core has a PAO file.

This chapter contains the following sections:

PAO Format

Format

Use the following format:

```
tooltarget libraryname filename hdllang
```

- The *tooltarget* specifies the tool target. Valid values are *lib*, *simlib*, *synlib*, and *vlgincdir*. Files specified with *lib* are used for both synthesis and simulation. Files specified with *simlib* are only for simulation. The *vlgincdir* defines the relative path of the Verilog Include directories. Files specified with *synlib* are only for synthesis.
- The *libraryname* specifies the library that contains the file. All of the files for the IP should use the IP as the library name. If no version is specified, the latest version of the library is used. If a specific version is required, then the library name is given with the version appended. For example, for version 1.00.a the library name is *libraryname_v1_00_a*. The name is in lower case.
- The *filename* specifies the file name. The filename optionally can have a file extension. If the file extension is omitted, then for VHDL, the `.vhd` extension is added; for Verilog, the `.v` extension is added. If the MPD file specifies `OPTION HDL=BOTH`, then extensions may not be specified.

  The *filename* can specify the keyword `all` in place of a file name. This causes all the files from the given library to be included. Do not use the keyword `all` if you are referring to the same library to which the PAO belongs. Any sub-library that is referenced using the `all` keyword must have a valid PAO file associated with it. The name is in lower-case.
- The *hdllang* specifies the language of the file name. Valid values are `verilog` and `vhdl`. This field is required when `OPTION HDL=MIXED` is used. If the language is not specified, the `OPTION HDL` value determines the value of this field. This field cannot

be specified for `OPTION HDL=BOTH`. This field is ignored if the keyword `all` is used. In that particular case, the PAO of the sub-library determines the language of each of the files included from that library.

Comments

You can insert comments without disrupting processing. The following are guidelines for inserting comments:

- Precede comments with the pound sign (#)
- Comments can continue to the end of the line
- Comments can be anywhere on the line

Verilog Include Directories

You must use relative paths to allow project maneuverability from development platform to development platform. Use the `` `include `` compiler directive in your Verilog HDL files to insert the contents of an entire file.

The following is an example Verilog HDL file:

```
`include “global_consts.v”
`include “pcore_v1_00_a/hdl/verilog/consts.v"
```

By default, all known EDK repositories are automatically included to the calls that process Verilog:

```
<proj_dir>/pcores
$XILINX_EDK/hw/XilinxBFMinterface/pcores
$XILINX_EDK/hw/XilinxProcessorIPLib/pcores$XILINX_EDK/hw/XilinxReferen
ceDesigns/pcores
```

You need only specify include paths that are not default. User-specified paths have a higher precedence over the default paths.

Format

Use the following format:

```
vlgincdir <library> <relative path from library>
```

Restrictions

If you intend to define a library file of text macros, you must give each text macro a unique name. The document IEEE 1364-2006 Section 3.12 defines the Verilog name space.

The text macro name space is global. The text macro names are defined in the linear order of appearance in the set of input files that make up the description of the design unit. Subsequent definitions of the same name override the previous definitions for the balance of the input files.

Use unique names, as shown below:

```
`define MASKVAL 4’b1010=> NO
`define PCORE_V1_00_A_MASKVAL 4’b1010=> YES
```

When multiple `vlgincdir` options are in use, it is possible for the compiler to read an unwanted included file. The preferred use of text inclusion within Verilog files is to include the relative path of the pcore library in use, as shown in the example below.

```
`include "pcore_v1_00_a/hdl/verilog/consts.v"
```

PAO Example

The following is an example of a VHDL PAO file:

```
lib common_v1_00_a common_types_pkg.vhd
lib common_v1_00_a pselect.vhd
lib opb_gpio_v1_00_a gpio_core
lib opb_gpio_v1_00_a opb_gpio
```

The following is an example of a MIXED PAO file:

```
lib libname_v2_00_a file1.vhd vhdl
lib ipname_v1_00_a file2.v verilog
lib ipname_v1_00_a file3.vhd vhdl
simlib ipname_v1_00_a simfile.v verilog
synlib ipname_v1_00_a synfile.vhd vhdl
```

Chapter 5

Black-Box Definition (BBD)

The Black Box Definition (BBD) file manages the file locations of optimized hardware netlists for the black-box sections of your peripheral design.

The `STYLE` option in the MPD with the values of `MIX`, or `BLACKBOX`, identify the core as having a BBD file.

This chapter contains the following sections:

BBD Format

The BBD format is a look-up table that lists netlist files. The first line is the header of the look-up table. There can be as many entries as necessary in the header to make a selection. Header entries are available only if defined as MPD parameters. The last column of the table must be the `FILES` column.

The netlist directory in the IP directory can have its own underlying directory structure because the BBD file manages the relative file locations.

Each file is listed with the file extension of the hardware implementation netlist. Since implementation netlists have multiple file extensions (such as `.edn`, `.edf`, `.edo`, `.ngo`), it is important to identify the format.

Comments

You can insert comments without disrupting processing. The following are guidelines for inserting comments:

- Precede comments with the pound sign (#).
- Comments can continue to the end of the line.
- Comments can be anywhere on the line.

Lists

If you have multiple hardware implementation netlists, then use a comma to separate each individual netlist in the list.

Common Repository Library

Support for relative paths to a common repository library is specified with the (:) syntax. For example, `ddr_common` is used to release netlists for the DDR for both `opb_ddr` and `plb_ddr`.

BBD Examples

File Selection Without Options

The following is an example of a file selection *without* options. The NGC netlist is copied into your implementation directory, regardless of specific options set on the core.

```
FILES
blackbox.ngc
```

Multiple File Selections Without Options

The following is an example of multiple file selections *without* options. The set of NGC netlists are copied into the your implementation directory regardless of specific options set on the core.

```
FILES
blackbox1.ngc, blackbox2.ngc, blackbox3.edn
```

File Selection With Options

The following is an example of a file selection *with* options. The specific EDIF netlist is copied into your implementation directory dependent on the `C_FAMILY` and `C_BUS_CONFIG` parameters set on the core.

```
C_FAMILY C_BUS_CONFIG FILES
virtex          1       virtex/ip1.edf
virtex          2       virtex/ip2.edf
spartan2        1       virtex/ip1.edf
spartan2        2       virtex/ip2.edf
virtexe         1       virtex/ip1.edf
virtexe         2       virtex/ip2.edf
spartan2e       1       virtex/ip1.edf
spartan2e       2       virtex/ip2.edf
virtex2         1       virtex2/ip1.edf
virtex2         2       virtex2/ip2.edf
virtex2p        1       virtex2/ip1.edf
virtex2p        2       virtex2/ip2.edf
```

File Selection With Common Repository Library

The following is an example of a file selection with a common repository library. The following example illustrates that the netlist `ddr_v1_00_b_virtex2_async_fifo`.edn is delivered from `ddr_v1_00_b` repository library.

```
C_FAMILY FILES
virtex2 ddr_v1_00_b:ddr_v1_00_b_virtex2_async_fifo.edn
virtex2p ddr_v1_00_b:ddr_v1_00_b_virtex2_async_fifo.edn
```

XILINX®

Chapter 6

Microprocessor Software Specification (MSS)

This chapter describes the Microprocessor Software Specification (MSS) format.

This chapter contains the following sections:

Overview

You supply an MSS file as an input to the Library Generator (Libgen). The MSS file contains directives for customizing operating systems (OSs), libraries, and drivers.

Note: RevUp tool provides a way to convert the old MSS format to the new one used in this version of the EDK tools. For more information see Chapter 9, "Format Revision Tool," in the *Embedded System Tools Reference Manual*.

Additional Resources

Embedded System Tools Reference Manual:
http://www.xilinx.com/ise/embedded/edk_docs.htm

TMSS Format

You supply an MSS file as an input to the Library Generator (Libgen). An MSS file is case insensitive and any reference to a file name or instance name in the MSS file is also case sensitive.

Comments can be specified anywhere in the file. A pound (#) character denotes the beginning of a comment, and all characters after it, right up to the end of the line, are ignored. All white spaces are also ignored and carriage returns act as sentence delimiters.

MSS Keywords

The keywords that are used in an MSS file are as follows:

`BEGIN`

The keyword begins a driver, processor, or file system definition block. `BEGIN` should be followed by the `driver`, `processor` or `filesys` keywords.

`END`

This keyword signifies the end of a definition block.

`PARAMETER`

The MSS file has a simple `name = value` format for most statements. The `PARAMETER` keyword is required before all such `NAME`, `VALUE` pairs. The format for assigning a value to a parameter is `parameter name = value`. If the parameter is within a `BEGIN-END` block, it is a local assignment; otherwise it is a global (system level) assignment.

Requirements

The MSS file has a dependency on the MHS file. This dependency has to be specified as a command line option to Libgen using the `-mhs` option. Refer to the "Library Generator" chapter in the *Embedded System Tools Reference Manual* for more information. (For a link to the manual, see the "Additional Resources" section of this chapter.) There is a resulting dependency on hardware for the software flow. Refer to Chapter 2, "Microprocessor Hardware Specification (MHS)" for more information on hardware configuration.

Prior to the EDK 6.1 release, this dependency was specified in the MSS file as `parameter HW_SPEC_FILE = file_name.mhs`. This parameter will be deprecated for the EDK6.1 release, since the MHS file is given as a command line option to the Libgen tool, and will eventually be removed for future releases.

The syntax of various files that the embedded development tools use is described by the Platform Specification Format (PSF). The current PSF version is 2.1.0. The MSS file should also contain version information in the form of `parameter Version = 2.1.0`, which represents the PSF version 2.1.0.

MSS Example

An example MSS file is given below:

```
parameter VERSION = 2.1.0

BEGIN OS
parameter PROC_INSTANCE = my_microblaze
parameter OS_NAME = standalone
parameter OS_VER = 1.00.a
parameter STDIN = my_uartlite_1
parameter STDOUT = my_uartlite_1
END
```

```
BEGIN PROCESSOR
parameter HW_INSTANCE = my_microblaze
parameter DRIVER_NAME = cpu
parameter DRIVER_VER = 1.00.a
parameter XMDSTUB_PERIPHERAL = my_jtag
END

BEGIN OS
parameter PROC_INSTANCE = my_ppc
parameter OS_NAME = standalone
parameter OS_VER = 1.00.a
parameter STDIN = my_uartlite_2
parameter STDOUT = my_uartlite_2
END

BEGIN PROCESSOR
parameter HW_INSTANCE = my_ppc
parameter DRIVER_NAME = cpu_ppc405
parameter DRIVER_VER = 1.00.a
END

BEGIN DRIVER
parameter HW_INSTANCE = my_intc
parameter DRIVER_NAME = intc
parameter DRIVER_VER = 1.00.a
END

BEGIN DRIVER
parameter HW_INSTANCE = my_uartlite_1
parameter DRIVER_VER = 1.00.a
parameter DRIVER_NAME = uartlite
parameter INT_HANDLER = uart_1_handler, INT_PORT = Interrupt
END

BEGIN DRIVER
parameter HW_INSTANCE = my_uartlite_2
parameter DRIVER_VER = 1.00.a
parameter DRIVER_NAME = uartlite
parameter INT_HANDLER = uart_2_handler, INT_PORT = Interrupt
END

BEGIN DRIVER
parameter HW_INSTANCE = my_timebase_wdt
parameter DRIVER_VER = 1.00.a
parameter DRIVER_NAME = timebase_wdt
parameter INT_HANDLER=my_timebase_hndl, INT_PORT = Timebase_Interrupt
parameter INT_HANDLER=my_timebase_hndl, INT_PORT = WDT_Interrupt
END

BEGIN LIBRARY
parameter LIBRARY_NAME = XilMfs
parameter LIBRARY_VER = 1.00.a
parameter NUMBYTES = 100000
parameter BASE_ADDRESS = 0x80f00000
END

BEGIN DRIVER
parameter HW_INSTANCE = my_jtag
parameter DRIVER_NAME = uartlite
```

```
parameter DRIVER_VER = 1.00.a
parameter INT_HANDLER = jtag_uart_handler, INT_PORT = Interrupt
END
```

Global Parameters

These parameters are system-specific parameters and do not relate to a particular driver, file system, or library.

PSF Version

This option specifies the PSF version of the MSS file. This option is mandatory for versions 2.1.0 and above, and is formatted as:

```
parameter VERSION = 2.1.0
```

Parameter INT_HANDLER

This option defines the interrupt handler software routine for an external interrupt port given in the MHS file, and is formatted as:

```
parameter INT_HANDLER = my_int_handl, INT_PORT = Interrupt
```

The external interrupt port that raises the interrupt is specified after the attribute as shown above with the `INT_PORT` keyword. This port should match the port name (not the signal name) specified in the MHS file as a global external port.

Instance-Specific Parameters

These parameters are OS-, processor-, driver-, or library-specific. The parameters must be within a `BEGIN` and `END` block.

OS, Driver, Library, and Processor Block Parameters Summary

The following list shows the parameters that can be used in OS, driver, library and processor blocks.

PROC_INSTANCE
HW_INSTANCE
OS_NAME
OS_VER
DRIVER_NAME
DRIVER_VER
INT_HANDLER
LIBRARY_NAME
LIBRARY_VER

OS, Driver, Library, and Processor Block Parameters Definitions

PROC_INSTANCE

This option is required for the OS associated with a processor instances specified in the MHS file, and is formatted as:

```
parameter PROC_INSTANCE = instance_name
```

All OSs in EDK require processor instances to be associated with the OSs. The instance name that is given must match the name specified in the MHS file.

HW_INSTANCE

This option is required for drivers associated with peripheral instances specified in the MHS file and is formatted as:

```
parameter HW_INSTANCE = instance_name
```

All drivers in EDK require instances to be associated with the drivers. Even a processor definition block should refer to the processor instance. The instance name that is given must match the name specified in the MHS file.

OS_NAME

This option is needed for processor instances that have OSs associated with them and is formatted as:

```
parameter OS_NAME = standalone
```

Library Generator copies the OS directory specified to `OUTPUT_DIR/processor_instance_name/libsrc` directory and compiles the OS sources using makefiles provided. See the "Library Generator" chapter in the *Embedded System Tools Reference Manual* for more information. For a link to the manual, see the "Additional Resources" section of this chapter.

OS_VER

The OS version is set using the `OSVER` option and is formatted as:

```
parameter OS_VER = 1.00.a
```

This version is specified in the following format: `x.yz.a`, where `x`, `y` and `z` are digits, and `a` is a character. This is translated to the OS directory searched by Libgen as follows:

`USER_PROJECT/bsp/OS_NAME_vx_yz_a`

`XILINX_EDK/sw/lib/bsp/OS_NAME_vx_yz_a`

The MLD (Microprocessor Library Definition) files Libgen needs for each OS should be named `OS_NAME_v2_1_0.mld` and should be present in a subdirectory `data/` within the driver directory. Refer to Chapter 7, "Microprocessor Library Definition (MLD)" for more information.

DRIVER_NAME

This option is needed for peripherals that have drivers associated with them and is formatted as:

```
parameter DRIVER_NAME = uartlite
```

Library Generator copies the driver directory specified to `OUTPUT_DIR/processor_instance_name/libsrc` directory and compiles the drivers using makefiles provided. Refer to the "Library Generator" chapter in the *Embedded System Tools Reference Manual*, for more information.

DRIVER_VER

The driver version is set using the `DRIVER_VER` option, and is formatted as:

```
parameter DRIVER_VER = 1.00.a
```

This version is specified in the following format: `x.yz.a`, where `x`, `y` and `z` are digits, and `a` is a character. This is translated to the driver directory searched by Libgen as follows:

```
USER_PROJECT/drivers/DRIVER_NAME_vx_yz_a
USER_PROJECT/pcores/DRIVER_NAME_vx_yz_a
XILINX_EDK/sw/XilinxProcessorIPLib/drivers/DRIVER_NAME_vx_yz_a
```

The MDD (Microprocessor Driver Definition) files needed by Libgen for each driver should be named `DRIVER_NAME_v2_1_0.mdd` and should be present in a subdirectory `data/` within the driver directory. Refer to Chapter 8, "Microprocessor Driver Definition (MDD)," for more information.

INT_HANDLER

This option defines the interrupt handler software routine for an interrupt port of the peripheral and is formatted as:

```
parameter INT_HANDLER = my_int_handl, INT_PORT = Interrupt
```

The interrupt port of the peripheral instance that raises the interrupt is specified after the attribute as shown above with the `INT_PORT` keyword. This port should match the port name (and not the signal name) specified in the MHS file for that peripheral instance.

LIBRARY_NAME

This option is needed for libraries, and is formatted as:

```
parameter LIBRARY_NAME = xilmfs
```

Library Generator copies the library directory specified in the `OUTPUT_DIR/processor_instance_name/libsrc` directory and compiles the libraries using makefiles provided. See the "Library Generator" chapter in the *Embedded System Tools Reference Manual*, for more information.

LIBRARY_VER

The library version is set using the `LIBRARY_VER` option and is formatted as:

```
parameter LIBRARY_VER = 1.00.a
```

This version is specified in the following format: `x.yz.a`, where `x`, `y` and `z` are digits, and `a` is a character. This is translated to the library directory searched by Libgen as follows:

```
USER_PROJECT/sw_services/LIBRARY_NAME_vx_yz_a
XILINX_EDK/sw/lib/sw_services/LIBRARY_NAME_vx_yz_a
```

The MLD (Microprocessor Library Definition) files needed by Libgen for each library should be named `LIBRARY_NAME_v_2_1_0.mld` and should be present in a subdirectory `data/` within the library directory. See Chapter 7, "Microprocessor Library Definition (MLD)" for more information.

MDD/MLD Specific Parameters

Parameters specified in the MDD/MLD file can be overwritten in the MSS file and formatted as

```
parameter PARAM_NAME = PARAM_VALUE
```

See Chapter 7, "Microprocessor Library Definition (MLD)" and Chapter 8, "Microprocessor Driver Definition (MDD)" for more information.

OS-Specific Parameters Summary

The following list identifies all the parameters that can be specified only in an OS definition block.

STDIN

Identify the standard input device with the STDIN option, which is formatted as:.

```
parameter STDIN = instance_name
```

STDOUT

Identify the standard output device with the STDOUT option, which is formatted as:

```
parameter STDOUT = instance_name
```

Example: MSS Snippet Showing OS Options

```
BEGIN OS
parameter PROC_INSTANCE = my_microblaze
parameter OS_NAME = standalone
parameter OS_VER = 1.00.a
parameter STDIN = my_uartlite_1
parameter STDOUT = my_uartlite_1
END
```

Processor-Specific Parameter Summary

Following is a list of all of the parameters that can be specified only in a processor definition block.

XMDSTUB_PERIPHERAL
COMPILER
ARCHIVER
COMPILER_FLAGS
EXTRA_COMPILER_FLAGS

Processor-Specific Parameter Definitions

XMDSTUB_PERIPHERAL

The peripheral that is used to handle the XMDStub should be specified in the XMDSTUB_PERIPHERAL option. This is useful for the MicroBlaze™ processor only, and is formatted as follows:

```
parameter XMDSTUB_PERIPHERAL = instance_name
```

COMPILER

This option specifies the compiler used for compiling drivers and libraries. The compiler defaults to mb-gcc or powerpc-eabi-gcc depending on whether the drivers are part of the MicroBlaze processor or PowerPC® processor instance. Any other compatible compiler can be specified as an option, and should be formatted as follows:

```
parameter COMPILER = dcc
```

This example denotes the Diab compiler as the compiler to be used for drivers and libraries.

ARCHIVER

This option specifies the utility to be used for archiving object files into libraries. The archiver defaults to mb-ar or powerpc-eabi-ar depending on whether or not the drivers are part of the MicroBlaze or PowerPC processor instance. Any other compatible archiver can be specified as an option, and should be formatted as follows:

```
parameter ARCHIVER = ar
```

This example denotes the archiver ar to be used for drivers and libraries.

COMPILER_FLAGS

This option specifies compiler flags to be used for compiling drivers and libraries. If the option is not specified, Libgen automatically uses platform and processor-specific options. This option should *not* be specified in the MSS file if the standard compilers and archivers in EDK are used. The COMPILER_FLAGS option can be defined in the MSS if there is a need for custom compiler flags that override Libgen-generated flags. The EXTRA_COMPILER_FLAGS option is recommended if compiler flags must be appended to the ones Libgen already generates. Format this option as follows:

```
parameter COMPILER_FLAGS = ""
```

EXTRA_COMPILER_FLAGS

This option can be used whenever custom compiler flags need to be used in addition to the automatically generated compiler flags, and should be formatted as follows:

```
parameter EXTRA_COMPILER_FLAGS = -g
```

This example specifies that the drivers and libraries must be compiled with debugging symbols in addition to the Libgen generated COMPILER_FLAGS.

Example MSS Snippet Showing Processor Options

```
BEGIN PROCESSOR
parameter HW_INSTANCE = my_microblaze
parameter DRIVER_NAME = cpu
parameter DRIVER_VER = 1.00.a
parameter DEFAULT_INIT = xmdstub
parameter XMDSTUB_PERIPHERAL = my_jtag
parameter STDIN = my_uartlite_1
parameter STDOUT = my_uartlite_1
parameter COMPILER = mb-gcc
parameter ARCHIVER = mb-ar
parameter EXTRA_COMPILER_FLAGS = -g -O0
parameter OS = standalone
END
```

Chapter 7

Microprocessor Library Definition (MLD)

This chapter describes the Microprocessor Library Definition (MLD) format, Platform Specification Format 2.1.0.

This chapter contains the following sections:

Overview

An MLD file contains directives for customizing software libraries and generating Board Support Packages (BSP) for Operating Systems (OS). This document describes the MLD format and the parameters that can be used to customize libraries and OSs. It is recommended that you read this document to become familiar with user-written libraries and OSs that must be configured by the Libgen tool.

Requirements

Each OS and library has an MLD file and a Tcl (Tool Command Language) file associated with it. The MLD file is used by the Tcl file to customize the OS or library, depending on different options in the MSS file. For more information on the MSS file format, see Chapter 6, "Microprocessor Software Specification (MSS)."

The OS and library source files and the MLD file for each OS and library must be located at specific directories if Libgen is to find the files and libraries. Refer to the "Library Generator" chapter in the *Embedded System Tools Reference Manual*, for a list of directories to be searched for OSs and libraries. A link to the *Embedded System Tools Reference Manual* is provided in the following section.

Additional Resources

Embedded System Tools Reference Manual:
http://www.xilinx.com/ise/embedded/edk_docs.htm

Library Definition Files

Library Definition involves defining Data Definition (MLD) and a Data Generation (Tcl) files.

Data Definition File

The MLD file (named as `<library_name>_v2_1_0.mld` or `<os_name>_v2_1_0.mld`) contains the configurable parameters. A detailed description of the various parameters and the MLD format is described in "MLD Parameter Description Section," page 100.

Data Generation File

The second file (named as `<library_name>_v2_1_0.tcl` or `<os_name>_v2_1_0.tcl`, with the filename being the same as the MLD filename) uses the parameters configured in the MSS file for the OS or library to generate data. Data generated includes, but is not limited to, generation of header files, C files, running DRCs for the OS or library and generating executables. The Tcl file includes procedures that are called by the Libgen tool at various stages of its execution. Various procedures in a Tcl file include: `DRC` (the name of the DRC given in the MLD file); `generate` (Libgen defined procedure) called after OS and library files are copied; `post_generate` (Libgen defined procedure) called after `generate` has been called on all OSs, drivers, and libraries; and `execs_generate` (a Libgen-defined procedure) called after the BSPs, libraries, and drivers have been generated. For more information on the workings of the Libgen tool refer to the "Library Generator" chapter in the *Embedded System Tools Reference Manual*. A link to this book is provided in the "Additional Resources" section, above.

Note: An OS/library need not have the data generation file (Tcl file).

MLD Format Specification

The MLD format specification involves the MLD file Format specification and the Tcl file Format specification. These are described below.

MLD File Format Specification

The MLD file format specification involves the description of parameters defined in the Parameter Description section.

Parameter Description Section

This data section describes configurable parameters in an OS/library. The format used to describe this section is discussed in "MLD Parameter Description Section," page 100.

Tcl File Format Specification

Each OS and library has a Tcl file associated with the MLD file. This Tcl file has the following sections:

DRC Section

This section contains Tcl routines that validate your OS and library parameters for consistency.

Generation Section

This section contains Tcl routines that generate the configuration header and C files based on the library parameters.

Examples

This section explains the MLD format through an example MLD file and its corresponding Tcl file.

Example: MLD File for a Library

An example of an MLD file for the `xilmfs` library is given below:

```
OPTION psf_version = 2.1.0 ;
```

`OPTION` is a keyword identified by the Libgen tool. The option name following the `OPTION` keyword is a directive to the Libgen tool to do a specific action. Here `psf_version` of the `MLD` file is defined to be 2.1. This is the only option that can occur before a `BEGIN LIBRARY` construct now.

```
BEGIN LIBRARY xilmfs
```

The `BEGIN LIBRARY` construct defines the start of a library named `xilmfs`.

```
OPTION DRC = mfs_drc ;
OPTION COPYFILES = all;
```

The `COPYFILES` option indicates the files to be copied for the library. The `DRC` option specifies the name of the Tcl procedure that the tool invokes while processing this library. Here `mfs_drc` is the Tcl procedure in the `xilmfs_v2_1_0.tcl` file that would be invoked by Libgen while processing the `xilmfs` library.

```
PARAM NAME = numbytes, DESC = "Number of Bytes", TYPE = int, DEFAULT =
100000, DRC = drc_numbytes ;
PARAM NAME = base_address, DESC = "Base Address", TYPE = int, DEFAULT =
0x10000, DRC = drc_base_address ;
PARAM NAME = init_type, DESC = "Init Type", TYPE = enum, VALUES = ("New
file system"=MFSINIT_NEW, "MFS Image"=MFSINIT_IMAGE, "ROM
Image"=MFSINIT_ROM_IMAGE), DEFAULT = MFSINIT_NEW ;
PARAM NAME = need_utils, DESC = "Need additional Utilities?", TYPE =
bool, DEFAULT =  false ;
```

`PARAM` defines a library parameter that can be configured. Each `PARAM` has the following properties associated with it, whose meaning is self-explanatory: `NAME`, `DESC`, `TYPE`, `DEFAULT`, `RANGE`, `DRC`. The property `VALUES` defines the list of possible values associated with an `ENUM` type.

```
BEGIN INTERFACE file
 PROPERTY HEADER="xilmfs.h" ;
```

```
 FUNCTION NAME=open, VALUE=mfs_file_open ;
 FUNCTION NAME=close, VALUE=mfs_file_close ;
 FUNCTION NAME=read, VALUE=mfs_file_read ;
 FUNCTION NAME=write, VALUE=mfs_file_write ;
 FUNCTION NAME=lseek, VALUE=mfs_file_lseek ;
END INTERFACE
```

An Interface contains a list of standard functions. A library defining an interface should have values for the list of standard functions. It must also specify a header file where all the function prototypes are defined.

PROPERTY defines the properties associated with the construct defined in the BEGIN construct. Here HEADER is a property with value "xilmfs.h", defined by the file interface. FUNCTION defines a function supported by the interface. The open, close, read, write, and lseek functions of the file interface have the values mfs_file_open, mfs_file_close, mfs_file_read, mfs_file_write, and mfs_file_lseek. These functions are defined in the header file xilmfs.h.

```
BEGIN INTERFACE filesystem
```

BEGIN INTERFACE defines an interface the library supports. Here file is the name of the interface.

```
 PROPERTY HEADER="xilmfs.h" ;
 FUNCTION NAME=cd, VALUE=mfs_change_dir ;
 FUNCTION NAME=opendir, VALUE=mfs_dir_open ;
 FUNCTION NAME=closedir, VALUE=mfs_dir_close ;
 FUNCTION NAME=readdir, VALUE=mfs_dir_read ;
 FUNCTION NAME=deletedir, VALUE=mfs_delete_dir ;
 FUNCTION NAME=pwd, VALUE=mfs_get_current_dir_name ;
 FUNCTION NAME=rename, VALUE=mfs_rename_file ;
 FUNCTION NAME=exists, VALUE=mfs_exists_file ;
 FUNCTION NAME=delete, VALUE=mfs_delete_file ;
END INTERFACE

END LIBRARY
```

END is used with the construct name that was used in the BEGIN statement. Here END is used with INTERFACE and LIBRARY constructs to indicate the end of each of INTERFACE and LIBRARY constructs.

Example: Tcl File of a Library

The following is the xilmfs_v2_1_0.tcl file corresponding the xilmfs_v2_1_0.mld file described in the previous section. The mfs_drc procedure would be invoked by Libgen for the xilmfs library while running DRCs for libraries. The generate routine generates constants in a header file and a c file for xilmfs library based on the library definition segment in the MSS file.

```
proc mfs_drc {lib_handle} {
    puts "MFS DRC ..."
}
proc mfs_open_include_file {file_name} {
    set filename [file join "../../include/" $file_name]
    if {[file exists $filename]} {
      set config_inc [open $filename a]
    } else {
       set config_inc [open $filename a]
```

Example: MLD File for an OS

An example of an MLD file for the standalone OS is given below:

```
OPTION psf_version = 2.1.0 ;
```

OPTION is a keyword identified by the Libgen tool. The option name following the OPTION keyword is a directive to the Libgen tool to do a specific action. Here the psf_version of the MLD file is defined to be 2.1. This is the only option that can occur before a BEGIN OS construct at this time.

```
BEGIN OS standalone
```

The BEGIN OS construct defines the start of an OS named standalone.

```
OPTION DESC = "Generate standalone BSP";
OPTION COPYFILES = all;
```

The DESC option gives a description of the MLD. The COPYFILES option indicates the files to be copied for the OS.

```
PARAM NAME = stdin, DESC = "stdin peripheral ", TYPE =
peripheral_instance, REQUIRES_INTERFACE = stdin, DEFAULT = none;
PARAM NAME = stdout, DESC = "stdout peripheral ", TYPE =
peripheral_instance, REQUIRES_INTERFACE = stdout, DEFAULT = none ;
PARAM NAME = need_xilmalloc, DESC = "Need xil_malloc?", TYPE = bool,
DEFAULT =  false ;
```

PARAM defines an OS parameter that can be configured. Each PARAM has the following, associated properties: NAME, DESC, TYPE, DEFAULT, RANGE, DRC. The property VALUES defines the list of possible values associated with an ENUM type.

```
END OS
```

END is used with the construct name that was used in the BEGIN statement. Here END is used with OS to indicate the end of OS construct.

Example: Tcl File of an OS

The following is the standalone_v2_1_0.tcl file corresponding to the standalone_v2_1_0.mld file described in the previous section.The generate routine generates constants in a header file and a c file for xilmfs library based on the library definition segment in the MSS file.

```
proc generate {os_handle} {
  global env

  set need_config_file "false"

  #Copy over the right set of files as src based on processor type
  set prochandle [xget_processor]
  set proctype [xget_value $prochandle "OPTION" "IPNAME"]
  set mbsrcdir "./src/microblaze"
  set ppcsrcdir "./src/ppc405"
  switch $proctype {
  "microblaze" {
    foreach entry [glob -nocomplain [file join $mbsrcdir *]] {
      file copy -force $entry "./src/"
    }
    set need_config_file "true"
```

```
  }
  "ppc405"  {
    foreach entry [glob -nocomplain [file join $ppcsrcdir *]] {
      file copy -force $entry "./src/"
    }
  }
  "default" {puts "unknown processor type\n"}
  }

  # Remove microblaze and ppc405 directories...
  file delete -force $mbsrcdir
  file delete -force $ppcsrcdir

  # Handle stdin and stdout
  xhandle_stdin $os_handle
  xhandle_stdout $os_handle

  # Create config file for microblaze interrupt handling
  if {[string compare -nocase $need_config_file "true"] == 0} {
    xhandle_mb_interrupts
  }

  # Generate xil_malloc.h if required
  set xil_malloc [xget_value $os_handle "PARAMETER" "need_xil_malloc"]
  if {[string compare -nocase $xil_malloc "true"] == 0} {
    xcreate_xil_malloc_config_file
  }
}
```

MLD Parameter Description Section

This section gives a detailed description of the constructs used in the MLD file.

Conventions

`[]` Denotes optional values.

<> Value substituted by the MLD writer.

Comments

Comments can be specified anywhere in the file. A "#" character denotes the beginning of a comment and all characters after the "#" right up to the end of the line are ignored. All white spaces are also ignored and semi-colons with carriage returns act as sentence delimiters.

OS or Library Definition

The OS or library section includes the OS or library name, options, dependencies, and other global parameters, using the following syntax:

```
OPTION psf_version = <psf version number>
BEGIN LIBRARY/OS <library/os name>
  [OPTION drc = <global drc name>]
  [OPTION depends  = <list of directories>]
  [OPTION help = <help file>]
  [OPTION requires_interface = <list of interface names>]
```

```
  PARAM <parameter description>
  [BEGIN CATEGORY <name of category>
    <category description>
  END CATEGORY]
  BEGIN INTERFACE <interface name>
        .......
  END INTERFACE]
END LIBRARY/OS
```

MLD or MDD Keyword Summary

The keywords that are used in an MLD or MDD file are as follows:

BEGIN	OS_STATE	PARAM
END	REQUIRES_INTERFACE	PROPERTY
PSF_VERSION	HELP	NAME
DRC	DEP	DESC
OPTION	INTERFACE	TYPE
COPYFILES	HEADER	DEFAULT
DEPENDS	FUNCTION	GUI_PERMIT
SUPPORTED_PERIPHERALS	CATEGORY	ARRAY
LIBRARY_STATE		

MLD or MDD Keyword Definitions

The keywords that are used in an MLD or MDD file are as follows:

BEGIN

The BEGIN keyword begins one of the following: `os`, `library`, `driver`, `block`, `category`, `interface`, `array`.

END

The `END` keyword signifies the end of a definition block.

PSF_VERSION

Specifies the PSF version of the library.

DRC

Specifies the DRC function name. This is the global DRC function, which is called by the GUI configuration tool or the command-line Libgen tool. This DRC function is called once you enter all the parameters and MLD or MDD writers can verify that a valid OS, library, or driver can be generated with the given parameters.

OPTION

Specifies that the name following the keyword `option` is an option to the tool Libgen.

COPYFILES

Specifies the files to be copied for the OS, library, or driver. If `ALL` is used, then Libgen copies all the OS, library, or driver files.

DEPENDS

Specifies the list of directories that needs to be compiled before the OS or library is built.

SUPPORTED_PERIPHERALS

Specifies the list of peripherals supported by the OS. The values of this option can be specified as a list, or as a regular expression. For example,

```
option supported_peripherals = (ppc405)
```

Indicates that the OS supports all versions of `ppc_405`. Regular expressions can be used in specifying the peripherals and versions. The regular expression (RE) is constructed as follows:

Single-character REs

- Any character that is not a special character (to be defined) matches itself.
- A backslash (followed by any special character) matches the literal character itself. That is, this “escapes” the special character.
- The special characters are: `+ * ? . [ ] ^ $`
- The period (.) matches any character except the new line. For example, `.umpty` matches either "Humpty" or "Dumpty."
- A set of characters enclosed in brackets `([])` is a one-character RE that matches any of the characters in that set. For example, `[akm]` matches either an "`a`", "`k`", or "`m`". A range of characters can be indicated with a dash. For example, `[a-z]` matches any lower-case letter. However, if the first character of the set is the caret (^), then the RE matches any character except those in the set. It does not match the empty string. Example: `[^akm]` matches any character except "`a`", "`k`", or "`m`". The caret loses its special meaning if it is not the first character of the set.

Multi-character REs

- A single-character RE followed by an asterisk (*) matches zero or more occurrences of the RE. Thus, `[a-z]*` matches zero or more lower-case characters.
- A single-character RE followed by a plus (+) matches one or more occurrences of the RE. Thus, `[a-z]+` matches one or more lower-case characters.
- A question mark (?) is an optional element. The preceeding RE can occur zero or once in the string -- no more. Thus, `xy?z` matches either xyz or xz.
- The concatenation of REs is a RE that matches the corresponding concatenation of strings. For example, `[A-Z][a-z]*` matches any capitalized word.
- For example, the following matches an version of the `ppc405` and `ppc405_virtex4`:

```
OPTION supported_peripherals = (ppc405_v[0-9]+_[1-9][0-9]_[a-z]
ppc405_virtex4);
```

LIBRARY_STATE

Specifies the state of the library. Following is the list of values that can be assigned to LIBRARY_STATE:

ACTIVE

An active library. By default the value of LIBRARY_STATE is ACTIVE.

DEPRECATED

This library is deprecated and will be removed from the release soon.

OBSOLETE

This library is obsolete and will not be recognized by any tools. Tools error out on an obsolete library and a new library should be used instead.

OS_STATE

Specifies the state of the operating system (OS). Following is the list of values that can be assigned to OS_STATE:

ACTIVE

This is an active OS. By default the value of OS_STATE is ACTIVE.

DEPRECATED

This OS is deprecated and would be removed from the release soon.

OBSOLETE

This OS is obsolete and will not be recognized by any tools. Tools error out on an obsolete OS and a new OS should be used instead.

REQUIRES_INTERFACE

Specifies the interfaces that must be provided by other OSs, libraries, or drivers in the system.

HELP

Specifies the HELP file that describes the OS, library, or driver.

DEP

Specifies the condition that must be satisfied before processing an entity. For example to include a parameter that is dependent on another parameter (defined as a DEP, for dependent, condition), the DEP condition should be satisfied. Conditions of the form (operand1 OP operand2) are only supported currently. In the future any expression can be given as condition.

INTERFACE

Specifies the interfaces implemented by this OS, library, or driver. It describes the interface functions and header files used by the library/driver.

```
BEGIN INTERFACE <interface name>
  OPTION DEP=<list of dependencies>;
  PROPERTY HEADER=<name of header file where the function is declared>;
  FUNCTION NAME=<name of interface function>, VALUE=<function name of
library/driver implementation> ;
END INTERFACE
```

HEADER

Specifies the HEADER file in which the interface functions would be defined.

FUNCTION

Specifies the FUNCTION implemented by the interface. This is a name-value pair in which name is the interface function name and value is the name of the function implemented by the OS, library, or driver.

CATEGORY

The CATEGORY block defines an unconditional block. This block gets included based on the default value of the category or if included in the MSS file.

```
BEGIN CATEGORY <category name>
  PARAM name = <category name>, DESC=<param description>,
TYPE=<category type>, DEFAULT=<default>, GUI_PERMIT=<value>, DEP =
<condition>
  OPTION DEPENDS=<list of dependencies>, DRC=<drc name>, HELP=<help
file>;
  < parameters or categories description>
END CATEGORY
```

Currently, nested categories are *not* supported though the syntax that specifies them. A category is selected in a MSS file by specifying the category name as a parameter with a boolean value TRUE. A category must have a PARAM with category name.

PARAM

The MLD file has a simple *name = value* format for most statements. The PARAM keyword is required before every such NAME, VALUE pairs. The format for assigning a value to a parameter is param *name = <name>*, default = *value*. The PARAM keyword specifies that the parameter can be overwritten in the MSS file.

PROPERTY

Specifies the various properties of the entity defined with a BEGIN statement

NAME

Specifies the name of the entity in which it was defined. (Examples: param and property.)

DESC

Describes the entity in which it was defined. (Examples: param and property.)

TYPE

Specifies the type for the entity in which it was defined. (Example: `param`**.)** The following types are supported:

bool

Boolean (true or false).

int

Integer

string

String value within " ".

enum

List of possible values that a parameter can take.

library

`Specify other library that is needed for building the library/driver.`

peripheral_instance

Specify other hardware drivers that is needed for building the library.

DEFAULT

Specifies the default value for the entity in which it was defined.

GUI_PERMIT

Specifies the permissions for modification of values. The following permissions exist:

NONE

The value cannot be modified at all.

ADVANCED_USER

The value can be modified by all. The XPS GUI does not display this value by default. This is displayed only for the advanced option in the GUI.

ALL_USERS

The value can be modified by all. The XPS GUI displays this value by default. This is the default value for all the values.

If `GUI_PERMIT = NONE`, the category is always active.

ARRAY

```
BEGIN ARRAY <array name>
  PROPERTY desc = <array description> ;
  PROPERTY size = <size of the array>;
  PROPERTY default = <List of Values for each element based on the size
of the array>
  # array field description as parameters
  PARAM name = <name of parameter>, desc = "description of param", type
= <type of param>, default = <default value>
.....
```

```
END ARRAY
```

ARRAY can have any number of PARAMs, and only PARAMs. It cannot have CATEGORY as one of the fields of an array element. The size of the array can be defined as one of the properties of the array. An array with default values specified in the default property leads to its size property being initialized to the number of values. If there is no size property defined, a size property is created before initializing it with the default number of elements. Each parameter in the array can have a default value. In cases in which size is defined with an integer value, an array of size elements would be created wherein the value of each element would be the default value of each of the parameters.

Design Rule Check (DRC) Section

```
proc mydrc { handle } {

}
```

The DRC function could be any Tcl code that checks your parameters for correctness. The DRC procedures can access (read-only) the Platform Specification Format database (which Libgen builds using the MHS and the MSS files) to read the parameter values that you set. The *handle* is associated with the current library in the database. The DRC procedure can get the OS and library parameters from this handle. It can also get any other parameter from the database by first requesting a handle and using the handle to get the parameters.

For errors, DRC procedures call the Tcl error command error "error msg" that displays in an error dialog box.

For warnings, DRC procedures return a string value that can be printed on the console.

On success, DRC procedures return without any value.

Library Generation (Generate) Section

```
proc mygenerate { handle } {

}
```

Generate could be any Tcl code that reads your parameters and generates configuration files for the OS or library. The configuration files can be C files, Header files, Makefiles, etc. The generate procedures can access (read-only) the Platform Specification Format database (which Libgen builds using the MHS and the MSS files) to read the parameter values of the OS or library that you set. The *handle* is a handle to the current OS or library in the database. The generate procedure can get the OS or library parameters from this handle. It can also get any other parameter from the database by first requesting a handle and using the handle to get the parameter.

XILINX®

Chapter 8

Microprocessor Driver Definition (MDD)

This chapter describes the Microprocessor Driver Definition (MDD) format, Platform Specification Format 2.1.0.

This chapter contains the following sections:

Overview

An MDD file contains directives for customizing software drivers. This document describes the MDD format and the parameters that can be used to customize drivers. For more information on drivers, refer to the *Driver Reference Guide* (xilinx_drivers.htm) and/or the *Device Driver Programmer Guide* (xilinx_drivers_guide.pdf), both contained in your EDK installation directory under `docs/usenglish`. It is recommended that you refer to these documents to gain an understanding of user-written drivers that must be configured by the Libgen tool.

Requirements

Each device driver has an MDD file and a Tcl (Tool Command Language) file associated with it. The MDD file is used by the Tcl file to customize the driver, depending on different options configured in the MSS file. For more information on the MSS file format, refer to Chapter 6, "Microprocessor Software Specification (MSS)."

The driver source files and the MDD file for each driver must be located at specific directories in order for Libgen to find the files and the drivers. Refer to the "Library Generator" chapter in the *Embedded System Tools Reference Manual* for a list of directories that is searched for drivers. A link to the manual is provided in the section below.

Additional Resources

- Embedded System Tools Reference Manual: http://www.xilinx.com/ise/embedded/edk_docs.htm
- *Driver Reference Guide* (xilinx_drivers.htm): contained in your EDK installation directory under `docs/usenglish`
- *Device Driver Programmer Guide* (xilinx_drivers_guide.pdf): contained in your EDK installation directory under `docs/usenglish`

Driver Definition Files

Driver Definition involves defining a Data Definition file (MDD) and a Data Generation file (Tcl file).

- *Data Definition File* - The MDD file (`<driver_name>_v2_1_0.mdd`) contains the configurable parameters. A detailed description of the parameters and the MDD format is described in "MDD Parameter Description," section of this chapter.
- *Data Generation File* - The second file (`<driver_name>_v2_1_0.tcl`, with the filename being the same as the MDD filename) uses the parameters configured in the MSS file for the driver to generate data. Data generated includes but not limited to generation of header files, C files, running DRCs for the driver and generating executables. The Tcl file includes procedures that are called by the Libgen tool at various stages of its execution. Various procedures in a Tcl file includes: the `DRC` (name of the DRC given in the MDD file), `generate` (Libgen defined procedure) called after driver files are copied, `post_generate` (Libgen defined procedure) called after `generate` has been called on all drivers and libraries, and `execs_generate` (Libgen defined procedure) called after the libraries and drivers have been generated. For more information on the working of the Libgen tool, refer to the "Library Generator" chapter in the *Embedded System Tools Reference Manual.* (A link to the document is provided in the "Additional Resources" section, above.)

Note: A driver need not have the data generation file (Tcl file).

MDD Format Specification

The MDD format specification involves the MDD file Format specification and the Tcl file Format specification. These are described below.

MDD File Format Specification

The MDD file format specification describes the parameters defined in the Parameter Description section. This data section describes configurable parameters in a driver. The format used to describe these parameters is discussed in the "MDD Parameter Description," section of this chapter.

Tcl File Format Specification

Each driver has a Tcl file associated with the MDD file. This Tcl file has the following sections:

DRC Section

This section contains Tcl routines that validate your driver parameters for consistency.

Generation Section

This section contains Tcl routines that generate the configuration header and C files based on the driver parameters

Example

This section explains the MDD format through an example of an MDD file and its corresponding Tcl file.

MDD: File Example

An example of an MDD file for the uartlite driver is given below:

```
OPTION psf_version = 2.1;
```

OPTION is a keyword identified by the Libgen tool. The option name following the OPTION keyword is a directive to the Libgen tool to do a specific action. Here the psf_version of the MDD file is defined as 2.1. This is the only option that can occur before a BEGIN DRIVER construct.

```
BEGIN DRIVER uartlite
```

The BEGIN DRIVER construct defines the start of a driver named uartlite.

```
  PARAM NAME = level, DESC = "Driver Level", TYPE = int, DEFAULT = 0,
RANGE = (0, 1);
```

PARAM defines a driver parameter that can be configured. Each PARAM has the following properties associated with it: NAME, DESC, TYPE, DEFAULT, RANGE.

```
BEGIN BLOCK, DEP = (level = 0)
```

BEGIN BLOCK, DEP allows conditional inclusion of a set of parameters subject to a condition fulfillmen. The condition is given by the DEP construct. Here the set of parameters defined inside the BLOCK would be processed by Libgen tool only when "level" parameter has a value 0.

```
  OPTION DEPENDS = (common_v1_00_a);
  OPTION COPYFILES = (xuartlite_l.c xuartlite_l.h Makefile);
OPTION DRC = uartlite_drc;
```

The DEPENDS option specifies that the driver depends on the sources of a directory named common_v1_00_a. The area for searching the dependent directory is decided by the Libgen tool. The COPYFILES option indicates the files to be copied for a "level" 0 uartlite driver. The DRC option specifies the name of the Tcl procedure that the tool invokes while processing this driver. The uartlite_drc is the Tcl procedure in the uartlite_v2_1_0.tcl file that would be invoked by Libgen while processing the uartlite driver.

```
BEGIN INTERFACE stdin
```

BEGIN INTERFACE defines an interface the driver supports. The interface name is stdin.

```
    PROPERTY header = xuartlite_l.h;
    FUNCTION name = inbyte, value = XUartLite_RecvByte;
END INTERFACE
```

An Interface contains a list of standard functions. A driver defining an interface should have values for the list of standard functions. It must also specify a header file in which all the function prototypes are defined.

PROPERTY defines the properties associated with the construct defined in the BEGIN construct. The header is a property with the value xuartlite_l.h, defined by the stdin interface. FUNCTION defines a function supported by the interface. The inbyte function of the stdin interface has the value XUartLite_RecvByte. This function is defined in the header file xuartlite_l.h.

```
BEGIN INTERFACE stdout
  PROPERTY header = xuartlite_l.h;
  FUNCTION name = outbyte, value = XUartLite_SendByte;
END INTERFACE

BEGIN INTERFACE stdio
  PROPERTY header = xuartlite_l.h;
  FUNCTION name = inbyte, value = XUartLite_RecvByte;
  FUNCTION name = outbyte, value = XUartLite_SendByte;
END INTERFACE

BEGIN ARRAY interrupt_handler
  PROPERTY desc = "Interrupt Handler Information";
  PROPERTY size = 1, permit = none;
  PARAM name = int_handler, default = XIntc_DefaultHandler, desc =
"Name of Interrupt Handler", type = string;
  PARAM name = int_port, default = Interrupt, desc = "Interrupt pin
associated with the interrupt handler", permit = none;
END ARRAY
```

The ARRAY construct defines an array of parameters. The interrupt_handler is the name of the array. The description (DESC) of the array and the size (SIZE) are defined as properties of the array interrupt_handler. The construct GUI_PERMIT is a directive to the tool that you cannot change the size of the array. The array defines int_handler and int_port as parameters of an element of the array.

```
  END BLOCK
BEGIN BLOCK, dep = (level = 1)
    OPTION depends = (common_v1_00_a uartlite_vxworks5_4_v1_00_a);
    OPTION copyfiles = all;
BEGIN ARRAY interrupt_handler
      PROPERTY desc = "Interrupt Handler Information";
      PROPERTY size = 1, permit = none;
      PARAM name = int_handler, default = XUartLite_InterruptHandler,
desc = "Name of Interrupt Handler", type = string;
     PARAM name = int_port, default = Interrupt, desc = "Interrupt pin
associated with the interrupt handler", permit = none;
    END ARRAY
PARAM name = connect_to, desc = "Connect to operationg system", type =
enum, values = {"VxWorks5_4" = VxWorks5_4, "None" = none}, default =
none;
  END BLOCK
END DRIVER
```

END is used with the construct name that was used in the BEGIN statement. Here END is used with BLOCK and DRIVER constructs to indicate the end of each BLOCK and DRIVER construct.

Example: Tcl File

The following is the uartlite_v2_1_0.tcl file corresponding to the uartlite_v2_1_0.mdd file described in the previous section. The "uartlite_drc" procedure would be invoked by Libgen for the uartlite driver while running DRCs for drivers. The **generate** routine generates constants in a header file and a c file for uartlite driver, based on the driver definition segment in the MSS file.

```
proc uartlite_drc {drv_handle} {
  puts "UartLite DRC"
}

proc generate {drv_handle} {
  set level [xget_value $drv_handle "PARAMETER" "level"]
  if {$level == 0} {
    xdefine_include_file $drv_handle "xparameters.h" "XUartLite"
"NUM_INSTANCES" "C_BASEADDR" "C_HIGHADDR"
  }
  if {$level == 1} {
    xdefine_include_file $drv_handle "xparameters.h" "XUartLite"
"NUM_INSTANCES" "C_BASEADDR" "C_HIGHADDR" "DEVICE_ID" "C_BAUDRATE"
"C_USE_PARITY" "C_ODD_PARITY"
    xdefine_config_file $drv_handle "xuartlite_g.c" "XUartLite"
"DEVICE_ID" "C_BASEADDR" "C_BAUDRATE" "C_USE_PARITY" "C_ODD_PARITY"
  }
}
```

MDD Parameter Description

This section gives a detailed description of the constructs used in the MDD file.

Conventions

[] - Denotes optional values.

<> - Value substituted by the MDD writer.

Comments

Comments can be specified anywhere in the file. A pound (#) character denotes the beginning of a comment, and all characters after it, right up to the end of the line, are ignored. All white spaces are also ignored and semicolons with carriage returns act as sentence delimiters.

Driver Definition

The driver section includes the driver name, options, dependencies, and other global parameters, using the following syntax:

```
OPTION psf_version = <psf version number>
BEGIN DRIVER <driver name>
  [OPTION drc = <global drc name>]
  [OPTION depends  = <list of directories>]
  [OPTION help = <help file>]
  [OPTION requires_interface = <list of interface names>]
  PARAM <parameter description>
  [BEGIN BLOCK,dep = <condition>
        .......
  END BLOCK]
  [BEGIN INTERFACE <interface name>
        .......
  END INTERFACE]
END DRIVER
```

MDD Keyword Summary

BEGIN
END
PSF_VERSION
DRC
OPTION
COPYFILES
DEPENDS
SUPPORTED_PERIPHERALS
DRIVER_STATE
REQUIRES_INTERFACE
HELP
DEP
BLOCK
INTERFACE
HEADER
FUNCTION
PARAM
PROPERTY
NAME
DESC
TYPE
DEFAULT
GUI_PERMIT
ARRAY

MDD Keyword Definitions

BEGIN

The BEGIN keyword begins with one of the following: `library`, `drive`, `block`, `category`, `interface`, or `array`.

END

The END keyword signifies the end of a definition block.

PSF_VERSION

Specifies the PSF version of the library.

DRC

Specifies the DRC function name. This is the global DRC function, which is called by the GUI configuration tool or the command line Libgen tool. This DRC function will be called once you enter all the parameters and MLD or MDD writers can verify that a valid library or driver can be generated with the given parameters.

OPTION

Specifies the name following the keyword OPTION is an option to the tool Libgen. The following five Libgen options are supported: `COPYFILES`, `DEPENDS`, `SUPPORTED_PERIPHERALS`, and `DRIVER_STATE`. These are described below.

COPYFILES

Specifies the list of files to be copied for the driver. If `ALL` is specified as the value, Libgen copies all the driver files.

DEPENDS

Specifies the list of directories on which a driver depends for compilation.

SUPPORTED_PERIPHERALS

Specifies the list of peripherals supported by the driver. The values of this option can be specified as a list or as a regular expression. The following example indicates that the driver supports all versions of `opb_jtag_uart` and the `opb_uartlte_v1_00_b` version:

```
option supported_peripherals = (xps_uartlite_v1_00_a, xps_uart16550)
```

Regular expressions can be used in specifying the peripherals and versions. The regular expression (RE) is constructed as follows:

Single-character REs

Any character that is not a special character (to be defined) matches itself.

A backslash (followed by any special character) matches the literal character itself. That is, it escapes the special character.

The special characters are: `+ * ? . [ ] ^ $`

The period matches any character except the newline. For example, `.umpty` matches either `Humpty` or `Dumpty`.

A set of characters enclosed in brackets `([])` is a one-character RE that matches any of the characters in that set. For example, `[akm]` matches an `a`, `k`, or `m`. A range of characters can be indicated with a dash. For example, `[a-z]` matches any lower-case letter. However, if the first character of the set is the caret `(^)`, then the RE matches any character except those in the set. It does not match the empty string. Example: `[^akm]` matches any character except `a`, `k`, or `m`. The caret loses its special meaning if it is not the first character of the set.

Multi-character REs

A single-character RE followed by an asterisk `(*)` matches zero or more occurrences of the RE. Therefore, `[a-z]*` matches zero or more lower-case characters.

A single-character RE followed by a plus (+) matches one or more occurrences of the RE. Therefore, `[a-z]+` matches one or more lower-case characters.

A question mark `(?)` is an optional element. The preceding RE can occur zero or once in the string -- no more. For example, xy?z matches either xyz or xz.

The concatenation of REs is an RE that matches the corresponding concatenation of strings. For example, `[A-Z][a-z]*` matches any capitalized word.

The following example matches any version of xps_uartlite, xps_uart16550 and mdm.

```
OPTION supported_peripherals = (xps_uartlite_v[0-9]+_[1-9][0-9]_[a-z]
  xps_uart16550 mdm);
```

DRIVER_STATE

Specifies the state of the driver. The following are the list of values that can be assigned to DRIVER_STATE:

ACTIVE

This is an active driver. By default the value of DRIVER_STATE is ACTIVE.

DEPRECATED

This driver is deprecated and would be removed from the release soon.

OBSOLETE

This driver is obsolete and is not recognized by any tools. Tools error out on an obsolete driver, and a new driver should be used instead.

REQUIRES_INTERFACE

Specifies the interfaces that must be provided by other libraries or drivers in the system.

HELP

Specifies the help file that describes the library or driver.

DEP

Specifies the condition that needs to be satisfied before processing an entity. For example to enter into a BLOCK, the DEP condition should be satisfied. Conditions of the form (operand1 OP operand2) are supported.

BLOCK

Specifies the block is to be entered into when the DEP condition is satisfied. Nested blocks are not supported.

INTERFACE

Specifies the interfaces implemented by this library or driver and describes the interface functions and header files used by the library or driver.

```
BEGIN INTERFACE <interface name>
  OPTION DEP=<list of dependencies>;
  PROPERTY HEADER=<name of header file where the function is declared>;
  FUNCTION NAME=<name of interface function>, VALUE=<function name of
library/driver implementation> ;
END INTERFACE
```

HEADER

Specifies the header file in which the interface functions would be defined.

FUNCTION

Specifies the function implemented by the interface. This is a name-value pair where *name* is the interface function name and *value* is the name of the function implemented by the library or driver.

PARAM

The MLD/MDD file has a simple *name = value* format for most statements. The PARAM keyword is required before every such NAME, VALUE pair. The format for assigning a value to a parameter is param *name = <name>*, default= *value*. The PARAM keyword specifies that the parameter can be overwritten in the MSS file.

PROPERTY

Specifies the various properties of the entity defined with a BEGIN statement

NAME

Specifies the name of the entity in which it was defined (example: PARAM, PROPERTY).

DESC

Describes the entity in which it was defined (example: PARAM, PROPERTY).

TYPE

Specifies the type for the entity in which it was defined (example: PARAM**)**. The following are the supported types:

bool

Boolean (true or false)

int

Integer

string

String value within " "

enum

List of possible values, that this parameter can take

library

Specify other library that is needed for building the library or driver.

peripheral_instance

Specify other hardware drivers needed for building the library or driver. Regular expressions can be used to specify the peripheral instance. Refer to the section "SUPPORTED_PERIPHERALS," page 113 for more details on regular expressions.

DEFAULT

Specifies the default value for the entity in which it was defined.

GUI_PERMIT

Specifies the permissions for modification of values. The following permissions exist:

NONE

The value cannot be modified at all

ADVANCED_USER

The value can be modified by all. The XPS GUI does not display this value by default. It is displayed only as an advanced option in the GUI.

ALL_USERS

The value can be modified by all. The XPS GUI displays this value by default. This is the default value for all the values.

If `GUI_PERMIT = NONE`, the category is always active.

ARRAY

```
BEGIN ARRAY <array name>
  PROPERTY desc = <array description> ;
  PROPERTY size = <size of the array>;
  PROPERTY default = <List of Values for each element based on the size
of the array>
  # array field description as parameters
  PARAM name = <name of parameter>, desc = "description of param", type
= <type of param>, default = <default value>
.....
END ARRAY
```

`ARRAY` can have any number of `PARAM`s and only `PARAM`s. It cannot have `CATEGORY` as one of the field of an array element. Size of the array can be defined as one of the `PROPERTY` of the `ARRAY`. An array with default values specified in `default` property, leads to its `size` property being initialized to the number of values. If there is no `size` property defined, a `size` property is created before initializing it with the default number of elements. Each parameter in the array can have a default value. If `size` is defined with an integer value, an array of `size` elements is created wherein the value of each element being the default value of each of the parameter.

Design Rule Check (DRC) Section

```
proc mydrc { handle } {

}
```

The DRC function can be any Tcl code that checks your parameters for correctness. The DRC procedures can access (read-only) the Platform Specification Format database (built by the Libgen tool using the MHS and the MSS files) to read the parameter values you set. The "handle" is a handle to the current driver in the database. The DRC procedure can get the driver parameters from this handle. It can also get any other parameter from the database, by first requesting a handle and using the handle to get the parameters.

For errors, DRC procedures would call the Tcl error command `error "error msg"` that displays in an error dialog box.

For warnings, DRC procedures return a string value that can be printed on the console.

On success, DRC procedures just return without any value.

Driver Generation (Generate) Section

```
proc mygenerate { handle } {

}
```

`Generate` could be any Tcl code that reads your parameters and generates configuration files for the driver. The configuration files can be C files, Header files, or Makefiles. The `generate` procedures can access (read-only) the Platform Specification Format database (built by the Libgen tool using the MHS and the MSS files) to read the parameter values of the driver that you set. The `handle` is a handle to the current driver in the database. The `generate` procedure can get the driver parameters from this handle. It can also get any other parameters from the database by requesting a handle and then using the handle to get the parameter.

XILINX®

Chapter 9

Xilinx Board Description (XBD) Format

Overview

The Xilinx® Board Description (XBD) file defines the contents of a particular board and how it interfaces with the FPGA devices on the board.

An XBD file has the following characteristics:

- Blocks that define the FPGA interfaces supported by the board
- Each block has list of attributes, parameters and ports
- Connectivity information between different ports or modules
- UCF Constraints information for each FPGA pin

This chapter includes the following sections:

XBD Syntax

XBD file syntax is case insensitive.

Note: The current XBD version is 2.2.0. The version number is reflected in the names of the XBD files read by the EDK tools.

Comments in XBD

You can insert comments in the XBD file without disrupting processing. Use the following guidelines:

- Precede comments with the pound sign (#).
- Comments continue to the end of the line.
- Comments can be anywhere on the line.

Format

Module Definitions

Use the following format at the beginning of a module definition:

```
BEGIN <block_type_keyword>
```

The `BEGIN` keyword signifies the beginning of a new module. There are three block types currently identified in XBD files:

IO_INTERFACE

An `IO_INTERFACE` specifies a physical module on the board. This does not include the FPGA itself. Each `IO_INTERFACE` also has a reference to soft IPs that you can use on the FPGA to interface with that module on the board.

IO_ADAPTER

An `IO_ADAPTER` specifies any soft glue-logic that might be needed to bridge any `IO_INTERFACE` pins with the ports of the soft-IP used for that `IO_INTERFACE`.

FPGA

An FPGA block represents the FPGA itself.

Use the following format to end a module definition:

```
END
```

Assignment Commands

Each BEGIN-END block contains multiple assignment commands. An assignment command is a name-value pair and can have one of more subproperty name-value pairs associated with it.

Use the following format for assignment commands:

```
<command> <name> = <value> {, <subproperty_name> = <subproperty_value>}
```

There are three assignment commands:

ATTRIBUTE

Names of all the ATTRIBUTEs are keywords. EDK tools perform certain actions or use the value of the attribute in a particular manner. You can use the ATTRIBUTE assignment command both inside or outside a BEGIN-END block.

PARAMETER

You can use any name for a PARAMETER. PARAMETER names specify values of PARAMETERs on the IPs connected to the IO_INTERFACE. A PARAMETER can be specified inside IO_INTERFACE blocks only.

PORT

Any name can be used for a PORT. PORT names specify connectivity between modules (including the FPGA) on the board. A PORT can be specified only inside IO_INTERFACE and IO_ADAPTER blocks.

Both PARAMETERs and PORTs can have subproperties associated with them. Each subproperty is a name-value pair. You must specify subproperties on the same line as the PARAMETER or the PORT. Subproperties must be comma-separated.

XBD Example

Your EDK installation directory contains XBD files shipped with EDK.

The board files can be found at:

$XILINX_EDK/board/Xilinx/boards/<board_name>/data/<board_name>_<version>.xbd

In the example path above, <board name> might be Xilinx_ML505, for example. The current XBD version is 2.2.0, therefore, <version> would be 2_2_0.

Global Attribute Commands

Global Attribute Command Summary

The Global Attribute command has the following syntax:

```
ATTRIBUTE <name> = <value>
```

Global Attribute Command Definitions

VENDOR

Specifies the name of the vendor. Tools use this attribute to sort various board files based on vendor name, using the following format:

```
ATTRIBUTE VENDOR= Xilinx
```

NAME

The NAME attribute is a string representing the name of the board. This is the name the tools display for you when they select a board. It is expressed in the following format:

```
ATTRIBUTE NAME= AFX Virtex-II Pro fg456 Proto Board
```

REVISION

The REVISION attribute identifies the revision number of the board that the XBD file represents. You must associate every board revision with an XBD file that is dedicated to that board revision alone. Use the following format to specify the revision:

```
ATTRIBUTE REVISION = C
```

CONTACT_INFO_URL

Displays a web URL link that you can use to contact Xilinx if you need assistance. The CONTACT_INFO_URL attribute is expressed in the following format:

```
ATTRIBUTE CONTACT_INFO_URL =
http://www.xilinx.com/support/techsup/tappinfo.htm
```

SPEC_URL

Displays a URL that takes you to the Xilinx website. The SPEC_URL attribute is expressed in the following format:

```
ATTRIBUTE SPEC_URL = http://www.xilinx.com
```

DESC

Provides a short text description of the board. Base System Builder uses the DESC attribute value and displays it in the GUI. The DESC attribute is expressed in the following format:

```
ATTRIBUTE DESC = some text
```

LONG_DESC

Specifies a long text description for the board. Base System Builder uses the LONG_DESC attribute value and displays it in the GUI. If the description string contains embedded commas, it *must* be enclosed in single quotes because the comma is a name-value delimiter. The LONG_DESC attribute is expressed using the following format:

```
ATTRIBUTE TEXT= 'some long text which gives an idea to user about the
board'
```

Local Attribute Commands

Local Attribute Command Summary

A local attribute is defined between a BEGIN-END block and expressed in the following format:

```
ATTRIBUTE <name> = <value>
```


Local Attribute Command Definitions

INSTANCE

Distinguishes one module from another. The INSTANCE attribute is expressed in the following format:

```
ATTRIBUTE INSTANCE = clk_module
```

CORENAME

Identifies the pcore that is instantiated in the MHS to represent the IO_ADAPTER. Use the CORENAME attribute only with IO_ADAPTER blocks. The CORENAME attribute is expressed in the following format:

```
ATTRIBUTE CORENAME = mypcore
```

VERSION

Specifies the HW_VER of the pcore to be instantiated in the MHS file. Use the VERSION attribute only with IO_ADAPTER blocks. The VERSION attribute is expressed in the following format:

```
ATTRIBUTE VERSION= 1.00.a
```

IOTYPE

Specifies what type of IO_INTERFACE block is being used. Use the IOTYPE attribute to match the pcore instantiated in the MHS. There are no versions for the IO_INTERFACE type. Any version information must be embedded in the IOTYPE string itself. Use the IOTYPE attribute with IO_INTERFACE blocks only. The IOTYPE attribute is expressed in the following format:

```
ATTRIBUTE IOTYPE = XIL_GPIO_V1
```

EXCLUSIVE

Represents a group of IO_INTERFACEs that are exclusive to each other. If you use one IO_INTERFACE in this group, you cannot use others, mainly because they share the same ports with the FPGA on the board. The EXCLUSIVE attribute is expressed in the following format:

```
ATTRIBUTE EXCLUSIVE = excl_group
```

JTAG_POSITION

Determines the position of the FPGA in the JTAG chain. Base System Builder uses this information while creating the *etc/download.cmd* file for the project. The JTAG_POSITION attribute is expressed in the following format:

```
ATTRIBUTE JTAG_POSITION = 1
```

Local Parameter Commands

A module can have any number of parameters. Parameters have a subproperty called IO_IS. You can use the string value of the IO_IS subproperty to match the parameter whose value must be overwritten with the value of the XBD local parameter. This parameter is expressed in the following format:

```
PARAMETER <name> = <value> {, <subprop_name> = <subprop_value>}
```

Local Parameter Subproperties

A subproperty on a local parameter is a name-value pair. A local parameter can have any number of subproperty name-value pairs associated with it. All the subproperties have to be specified on the same line as the parameter itself. Each name-value pair is separated by a comma.

The value of the IO_IS subproperty matches parameters on the IP with the parameters on the hardware component on the board. The IO_IS subproperty is expressed in the following format:

```
PARAMETER MYPARAM = 3, IO_IS = myparam
```

MEMORY_TYPE

The only allowed value of the MEMORY_TYPE subproperty is FLASH. Use the MEMORY_TYPE subproperty only on the baseaddress parameter of a flash memory module. Use the values of IO_IS=C_BASEADDR and C_HIGHADDR parameters only to get the size of the memory. A different local memory address may be assigned to the memory controller slave to which the memory module is connected. The MEMORY_TYPE subproperty is expressed in the following format:

```
PARAMETER C_BASEADDR = 0x00000000, IO_IS=C_BASEADDR, MEMORY_TYPE=FLASH
PARAMETER C_HIGHADDR = 0x0003FFFF, IO_IS=C_HIGHADDR
```

RANGE

The value of the RANGE subproperty is a comma-separated list of integers, intended for use only on the clock module. These integers specify a list of possible clock frequencies on the board. The following example shows a board with two clock frequencies of 66 and 100 MHz. The default frequency that BSB will use is 100 Mhz.

```
PARAMETER CLK_FREQ = 100000000, IO_IS=clk_freq, RANGE=(66000000,
100000000) # 66 Mhz or 100 Mhz
```

VALUE_NOTE

The value of the VALUE_NOTE subproperty provides a short text description of values associated with this parameter. It is expressed in the following format:

```
PARAMETER RST_POLARITY = 0, IO_IS = polarity, VALUE_NOTE = Active LOW
```

Local Port Commands

A local port is defined between the BEGIN-END block of a module. XBD supports local ports only; global ports are not supported. There are no reserved PORT names. You can specify local ports in all three block types. Local port commands are formatted as follows:

```
PORT <name> = <connector_name> {,<subprop_name> = <subprop_value>}
```

Local Port Subproperties

Local Port Subproperty Summary

You can associate a local port with any number of subproperty name-value pairs. All the subproperties must be specified on the same line as the port itself. Each name-value pair is separated by a comma.

DIR
INTERRUPT_PRIORITY
IO_IS
SENSITIVITY
SIGIS
UCF_NET_STRINGS
INITIALVAL

Local Port Subproperty Definitions

DIR

Specifies the FPGA port direction. The allowed value is IO, which designates the port as an IO port.

INTERRUPT_PRIORITY

The value of the INTERRUPT_PRIORITY subproperty defines the priority of an interrupt source. This affects the order in which various interrupts are connected to the interrupt controller (and, consequently, their priority). Use the INTERRUPT_PRIORITY subproperty only for those signals that are marked as SIGIS=INTERRUPT. The INTERRUPT_PRIORITY subproperty is formatted as follows:

```
PORT Intr = CONN_Intr, IO_IS=intr, SIGIS=INTERRUPT,
INTERRUPT_PRIORITY=HIGH
```

IO_IS

The value of the IO_IS subproperty matches ports on the FPGA to ports of peripherals instantiated in the MHS file. If the value of IO_IS on a port matches that on a port in the MPD file of the IP which, in turn, matches the IO_IF, it is considered a match. If there is a match, that particular port of the instantiated IP is defined as a global port in the MHS file and connected to this port on the board. The IO_IS value is formatted as follows:

```
PORT LED1 = CONN_LED1, IO_IS=gpio_io<0>, VALUE=net_vcc
```

SENSITIVITY

For a signal of type SIGIS=INTERRUPT, the SENSITIVITY subproperty defines the signal type to which this interrupt is sensitive. The global port created in MHS that corresponds to this signal is marked with the SENSITIVITY property. This subproperty is formatted as follows:

```
PORT Intr = CONN_Intr, SIGIS=INTERRUPT, SENSITIVITY=LEVEL_HIGH
```

SIGIS

SIGIS is a subproperty of PORT and designates a port as an interrupt port.

UCF_NET_STRINGS

Specifies the constraints associated with the NET for this PORT. The UCF_NET_STRINGS subproperty takes as value a comma separated list of strings. Each string in the list creates a separate line in the UCF file related to that net. This subproperty is formatted as follows:

```
PORT SDRAM_BA0 = CONN_SDRAM_8Mx32BA1, UCF_NET_STRINGS=("LOC = L4",
"IOSTANDARD=LVDCI_25")
```

The above line in an XBD file would lead to the following lines in the UCF file:

```
NET “CONN_SDRAM_8Mx32BA1” LOC = L4
NET “CONN_SDRAM_8Mx32BA1” IOSTANDARD=LVDCI_25
```

INITIALVAL

A subproperty of VALUE, INITIALVAL specifies the value to which a port must be driven if there is no corresponding port on the IP core connected to the device. In this case, a

top-level output port for the system is created and driven with this constant value. This subproperty is formatted as follows:

```
PORT LED1 = CONN_LED1, IO_IS=gpio_io<0>, INITIALVAL = net_vcc
```

Associating IPs with IO_INTERFACE in XBD

As previously described, an XBD file contains a number of BEGIN-END blocks, each corresponding to a hardware module on the board. The type of the module is specified using the attribute IOTYPE, as in the following example:

```
BEGIN <iotype>
```

The IOTYPE string is used to match an IP that can communicate with this module. Refer to Chapter 3, "Microprocessor Peripheral Definition (MPD)" for more information. An MPD file describes the behavior of an IP. Each IP can have a number of IO_INTERFACEs. Each IO_INTERFACE has a subproperty called IOTYPE. The value of this subproperty determines whether or not an IP can communicate with a particular hardware module on the board.

For example, consider the following line in the MPD file for IP xps_ethernetlite:

```
IO_INTERFACE NAME = Ethernet_0, iotype = XIL_Ethernet_V1
```

This IO_INTERFACE indicates that this particular IP can communicate with a module of type Ethernet. Similarly, the XBD file for any board that has an Ethernet module should define a block as follows:

```
BEGIN IO_INTERFACE
  ATTRIBUTE INSTANCE = myEthernet
  ATTRIBUTE IOTYPE = XIL_Ethernet_V1
  PARAMETER ...
  PORT ...
END
```

When tools try to find an IP that can communicate with this module on the board, they search for MPDs that have an IO_INTERFACE with IOTYPE and that match the IOTYPE of the IO_INTERFACE module in the XBD file. If there are several such IPs, users can select any of them.

Once an IP has been selected for communicating with that particular module on the board, tools use the IO_IS subproperty to connect ports of the IP to the module on the board.

Generally, an IP is designed to be parametric (in terms of VHDL, the IP has generics). When used with a particular board, you can specify some parameter values based on board characteristics. The matching of parameters in the XBD module with that of parameters on the IP is also done using the IO_IS block.

For example, consider the MPD snippet for xps_gpio:

```
######## MPD Snippet ############

BEGIN xps_gpio

## Peripheral Options
OPTION ...

IO_INTERFACE NAME = gpio_0, iotype = XIL_GPIO_V1

## Bus Interfaces
BUS_INTERFACE BUS = SPLB, BUS_STD = PLBV46, BUS_TYPE = SLAVE
```

```
## Generics for VHDL or Parameters for Verilog
PARAMETER C_GPIO_WIDTH = 32, DT=integer, IO_IF=gpio_0, IO_IS=num_bits
PARAMETER C_ALL_INPUTS = 0, DT=integer, IO_IF=gpio_0, IO_IS=all_inputs
PARAMETER ...

## Ports
PORT GPIO_IO = "", DIR = INOUT, IO_IF = gpio_0, IO_IS = gpio_io, ...
PORT ...

END
```

The MPD file defines an IO interface, `gpio_0` of `IOTYPE XIL_GPIO_V1`. The `gpio_0` IO interface has two parameters associated with it, `C_GPIO_WIDTH` and `C_ALL_INPUTS`. This parameter-IO interface association is specified with the `IO_IF` subproperty on the parameter. Similarly, `PORT GPIO_IO` is also associated with the `gpio_0` IO interface, using the `IO_IF` subproperty.

Consider the following XBD snippet from a hypothetical board that includes LEDs. These LEDs are of `IOTYPE GPIO`:

```
###### XBD Snippet ############

BEGIN IO_INTERFACE
   ATTRIBUTE INSTANCE = LEDs_4Bit
ATTRIBUTE IOTYPE= XIL_GPIO_V1
   PARAMETER num_bits =4, IO_IS=num_bits
   PARAMETER all_inputs =0, IO_IS=all_inputs# All outputs

   PORT LED1 = CONN_LED1, IO_IS=gpio_io[0], VALUE = net_vcc,
UCF_NET_STRINGS = (“N1”)
   PORT LED2 = CONN_LED2, IO_IS=gpio_io[1], VALUE=net_vcc,
UCF_NET_STRINGS = (“N2”)
   PORT LED3 = CONN_LED3, IO_IS=gpio_io[2], VALUE=net_vcc,
UCF_NET_STRINGS = (“P1”)
   PORT LED4 = CONN_LED4, IO_IS=gpio_io[3], VALUE=net_vcc,
UCF_NET_STRINGS = (“P2”)
END
```

As explained above, the `IOTYPE XIL_GPIO_V1` of the `IO_INTERFACE` in the MPD file is matched with the `IOTYPE XIL_GPIO_V1` of the module `LEDs_4Bit` in the XBD file. As a result, the `IO_IS` subproperty on ports in the MPD file and ports in the XBD file determine which ports connect to which. The following MHS instantiation results:

```
### MHS File Snippet #####

## Global Ports
PORT LEDs_4Bit_GPIO_IO = LEDs_4Bit_GPIO_IO, VEC = [0:3], DIR = INOUT

## GPIO instance
BEGIN xps_gpio
  ATTRIBUTE INSTANCE = LEDs_4Bit
  PARAMETER HW_VER = 1.00.a
  PARAMETER C_GPIO_WIDTH = 4 ## IO_IS = num_bits
  PARAMETER C_ALL_INPUTS = 0 ## IO_IS = all_inputs
  PORT GPIO_IO = LEDs_4Bit_GPIO_IO ## IO_IS = gpio_io
  ...
END
```

The output UCF snippet is as follows:

```
############ UCF Snippet ########

Net LEDs_4Bit_GPIO_IO<0> LOC=N1;
Net LEDs_4Bit_GPIO_IO<1> LOC=N2;
Net LEDs_4Bit_GPIO_IO<2> LOC=P1;
Net LEDs_4Bit_GPIO_IO<3> LOC=P2;
```

Bridging IP with IO_INTERFACE

Each IP that communicates with modules outside the FPGA (on the board) has its ports connected to pins on the FPGA. The FPGA, in turn, is connected to the pins on the board module. For standard IPs and modules, there is usually a one-to-one correspondence between ports on the IP and the pins on the external modules. However, on some boards, the external modules might have slightly different requirements. In such cases, a small amount of logic might need to be used before the IP ports can be connected to the external module. XBD allows you to specify such glue logic in the board file.

When creating the XBD file, you must be aware of parameter and port connection requirements for the board module. You must also associate at least one compatible soft-IP with that external module. (This is done by using the specific `IOTYPE` in `IO_INTERFACE` block.) If there is any mismatch, users can specify `IO_ADAPTER` in the XBD file to map the `IO_INTERFACE` on the board with the desired soft IP on the FPGA.

XBD Load Path

Refer to Figure 9-1 for an illustration of the library directory structure. The XPS tools search the PSF files across libraries, and they search in the *<library_name>/boards* directory for boards.

Each of these libraries contains the */boards* directory, which identifies where various boards are located. This is equivalent to the */pcores* directory in the IP search mechanism.

The directory name for a board must be the same as the name of the board itself. Each board directory should contain a */data* directory. The XBD file must reside in this data directory and must be called `<board_name>_v2_2_0.xbd`.

The directory called `edk_user_repository` appears at the same level as the EDK installation (*$XILINX_EDK*) and is automatically searched by all EDK tools for libraries. This feature is deprecated in in EDK 10.1. It is recommended that you explicitly specify repositories that apply to projects. You can do this in XPS using **Edit ->Preferences->Application Preferences ->Global Peripheral Repository**.

For example, to make a hypothetical board called `myboard` "visible" to the EDK tools, create a library of boards, for example, `MyEDKBoards`, in a library search directory. Your directory structure must appear as follows:

<Peripheral Repository Directory>/MyEDKBoards/boards/myboard_rev1/data/myboard_rev1_v2_2_0.xbd

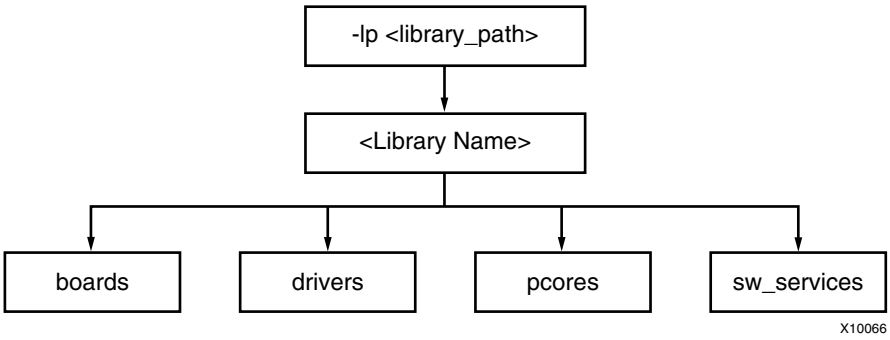

Figure 9-1: **Library Directory Structure**

The *$XILINX_EDK/board* directory is added as a default library search path for board files. Each library contains several boards. In XPS, you can specify a library search path that enables tools to locate additional board files.

BSB Restrictions

While most of the BSB is data-driven, there are some exceptions. Special processing is done inside the BSB for these. Some of the restrictions are listed below:

- BSB clock module generation is not data-driven. BSB can only handle certain input clock frequencies and can only produce certain multiples of the input clocks. BSB does not support multiple DDR and PCI™ interfaces because they require special clock generation.
- The parameter customization of instantiated pcores is not data-driven. For each known type of IO interface, BSB presents certain selectable parameters. If there is an `IO_INTERFACE` in the XBD file but no matching soft IP, BSB does not display the configuration of parameters on that IP.

Existing Xilinx IO Types

Establishing a match between an MPD IO_INTERFACE and an IOTYPE module in XBD is accomplished solely through string comparison. The table below shows a list of IO interfaces implemented by various Xilinx IPs. For a new board that includes a module that communicates with the FPGA using a Xilinx-provided IP, module and IO interface names must be the same. This is required to enable automatic connection between the board module and IP.

Table 9-1: **List of IO INTERFACES Supported by Xilinx IPs**

IOTYPE/ IO_INTERFACE	Supporting IPs
XIL_CPUDEBUG_V1	ppc405
XIL_CLOCK_V1	None. Used by tools internally.
XIL_MEMORY	mpmc (supports DDR and DDR2)
XIL_EMC_V1	xps_emc
XIL_EPC_V1	xps_epc
XIL_Ethernet_V1	xps_ethernetlite
XIL_GPIO_V1	xps_gpio
XIL_IIC_V1	xps_iic
XIL_MGT_PROTECTOR_V1	mgt_protector
XIL_PCI32_V1	opb_pci
XIL_PCI_ARBITER_V1	opb_pci_arbiter
XIL_RESET_V1	None. Used by tools internally.
XIL_SDRAM_V1	xps_sdram
XIL_SPI_V1	xps_spi
XIL_SYSACE_V1	xps_sysace
XIL_TEMAC_V1	xps_II_temac
XIL_TRACE_V1	ppc405
XIL_UART_V1	xps_uartlite xps_uart16550

XILINX®

Appendix A

Glossary

B

BBD file

Black Box Definition file. The BBD file lists the netlist files used by a peripheral.

BFL

Bus Functional Language.

BFM

Bus Functional Model.

BIT File

Xilinx® Integrated Software Environment (ISE™) Bitstream file.

BitInit

The Bitstream Initializer tool. It initializes the instruction memory of processors on the FPGA and stores the instruction memory in blockRAMs in the FPGA.

block RAM (BRAM)

A block of random access memory built into a device, as distinguished from distributed, LUT based random access memory.

BMM file

Block Memory Map file. A BMM file is a text file that has syntactic descriptions of how individual block RAMs constitute a contiguous logical data space. Data2MEM uses BMM files to direct the translation of data into the proper initialization form. Since a BMM file is a text file, it is directly editable.

BSB

Base System Builder. A wizard for creating a complete design in Xilinx Platform Studio (XPS). BSB is also the file type used in the Base System Builder.

BSP

See Standalone BSP.

C

CFI

Common Flash Interface

D

DCM

Digital Clock Manager

DCR

Device Control Register.

DLMB

Data-side Local Memory Bus. See also: LMB.

DMA

Direct Memory Access.

DOPB

Data-side On-chip Peripheral Bus. See also: OPB.

DRC

Design Rule Check.

DSPLB

Data-side Processor Local Bus. See also: ISPLB.

E

EDIF file

Electronic Data Interchange Format file. An industry standard file format for specifying a design netlist.

EDK

Xilinx Embedded Development Kit.

ELF file

Executable and Linkable Format file.

EMC

External Memory Controller.

EST

Embedded System Tools.

F

FATfs (XilFATfs)

LibXil FATFile System. The XilFATfs file system access library provides read/write access to files stored on a Xilinx SystemACE CompactFlash or IBM microdrive device.

FPGA

Field Programmable Gate Array.

FSL

MicroBlaze™ processor Fast Simplex Link. Unidirectional point-to-point data streaming interfaces ideal for hardware acceleration. The MicroBlaze processor has FSL interfaces directly to the processor.

G

GDB

GNU Debugger.

GPIO

General Purpose Input and Output. A 32-bit peripheral that attaches to the on-chip peripheral bus.

H

Hardware Platform

Xilinx FPGA technology allows you to customize the hardware logic in your processor subsystem. Such customization is not possible using standard off-the-shelf microprocessor or controller chips. Hardware platform is a term that describes the flexible, embedded processing

subsystem you are creating with Xilinx technology for your application needs.

HDL

Hardware Description Language.

I

IBA

Integrated Bus Analyzer.

IDE

Integrated Design Environment.

ILA

Integrated Logic Analyzer.

ILMB

Instruction-side Local Memory Bus. See also: LMB.

IOPB

Instruction-side On-chip Peripheral Bus. See also: OPB.

IPIC

Intellectual Property Interconnect.

IPIF

Intellectual Property Interface.

ISA

Instruction Set Architecture. The ISA describes how aspects of the processor (including the instruction set, registers, interrupts, exceptions, and addresses) are visible to the programmer.

ISC

Interrupt Source Controller.

ISE

Xilinx ISE Project Navigator project file.

ISPLB

Instruction-side Peripheral Logical Bus. See also: DSPLB.

ISS

Instruction Set Simulator.

J

JTAG

Joint Test Action Group.

L

Libgen

Library Generator sub-component of the Xilinx Platform Studio technology.

LibXil Standard C Libraries

EDK libraries and device drivers provide standard C library functions, as well as functions to access peripherals. Libgen automatically configures the EDK libraries for every project based on the MSS file.

LibXil File

A module that provides block access to files and devices. The LibXil File module provides standard routines such as open, close, read, and write.

LibXil Profile

A software intrusive profile library that generates call graph and histogram information of any program running on a board.

LMB

Local Memory Bus. A low latency synchronous bus primarily used to access on-chip block RAM. The MicroBlaze processor contains an instruction LMB bus and a data LMB bus.

M

MDD file

Microprocessor Driver Description file.

MDM

Microprocessor Debug Module.

MFS

LibXil Memory File System. The MFS provides user capability to manage program memory in the form of file handles.

MHS file

Microprocessor Hardware Specification file. The MHS file defines the configuration of the embedded processor system including buses,peripherals, processors, connectivity, and address space.

MLD file

Microprocessor Library Definition file.

MPD file

Microprocessor Peripheral Definition file. The MPD file contains all of the available ports and hardware parameters for a peripheral.

MSS file

Microprocessor Software Specification file.

N

NCF file

Netlist Constraints file.

NGC file

The NGC file is a netlist file that contains both logical design data and constraints. This file replaces both EDIF and NCF files.

NGD file

Native Generic Database file. The NGD file is a netlist file that represents the entire design.

NGO File

A Xilinx-specific format binary file containing a logical description of the design in terms of its original components and hierarchy.

NPI

Native Port Interface.

O

OCM

On Chip Memory.

OPB

On-chip Peripheral Bus.

P

PACE

Pinout and Area Constraints Editor.

PAO file

Peripheral Analyze Order file. The PAO file defines the ordered list of HDL files needed for synthesis and simulation.

PBD file

Processor Block Diagram file.

Platgen

Hardware Platform Generator sub-component of the Platform Studio technology.

PLB

Processor Local Bus.

PROM

Programmable ROM.

PSF

Platform Specification Format. The specification for the set of data files that drive the EDK tools.

S

SDF file

Standard Data Format file. A data format that uses fields of fixed length to transfer data between multiple programs.

SDK

Software Development Kit.

SDMA

Soft Direct Memory Access

Simgen

The Simulation Generator sub-component of the Platform Studio technology.

Software Platform

A software platform is a collection of software drivers and, optionally, the operating system on which to build your application. Because of

the fluid nature of the hardware platform and the rich Xilinx and Xilinx third-party partner support, you may create several software platforms for each of your hardware platforms.

SPI

Serial Peripheral Interface.

Standalone BSP

Standalone Board Support Package. A set of software modules that access processor-specific functions. The Standalone BSP is designed for use when an application accesses board or processor features directly (without an intervening OS layer).

SVF File

Serial Vector Format file.

U

UART

Universal Asynchronous Receiver-Transmitter.

UCF

User Constraints File.

V

VHDL

VHSIC Hardware Description Language.

VP

Virtual Platform.

VPgen

The Virtual Platform Generator sub-component of the Platform Studio technology.

X

XBD File

Xilinx Board Definition file.

XCL

Xilinx CacheLink. A high performance external memory cache interface available on the MicroBlaze processor.

Xilkernel

The Xilinx Embedded Kernel, shipped with EDK. A small, extremely modular and configurable RTOS for the Xilinx embedded software platform.

XMD

Xilinx Microprocessor Debugger.

XMK

Xilinx Microkernel. The entity representing the collective software system comprising the standard C libraries, Xilkernel, Standalone BSP, LibXil MFS, LibXil File, and LibXil Drivers.

XMP File

Xilinx Microprocessor Project file. This is the top-level project file for an EDK design.

XPS

Xilinx Platform Studio. The GUI environment in which you can develop your embedded design.

XST

Xilinx Synthesis Technology.

Z

ZBT

Zero Bus Turnaround™.