

Processor IP User Guide

June 2003





"Xilinx" and the Xilinx logo shown above are registered trademarks of Xilinx, Inc. Any rights not expressly granted herein are reserved. CoolRunner, RocketChips, Rocket IP, Spartan, StateBENCH, StateCAD, Virtex, XACT, XC2064, XC3090, XC4005, and XC5210 are registered trademarks of Xilinx, Inc.



The shadow X shown above is a trademark of Xilinx, Inc.

ACE Controller, ACE Flash, A.K.A. Speed, Alliance Series, AllianceCORE, Bencher, ChipScope, Configurable Logic Cell, CORE Generator, CoreLINX, Dual Block, EZTag, Fast CLK, Fast CONNECT, Fast FLASH, FastMap, Fast Zero Power, Foundation, Gigabit Speeds...and Beyond!, HardWire, HDL Bencher, IRL, J Drive, JBits, LCA, LogiBLOX, Logic Cell, LogiCORE, LogicProfessor, MicroBlaze, MicroVia, MultiLINX, NanoBlaze, PicoBlaze, PLUSASM, PowerGuide, PowerMaze, QPro, Real-PCI, RocketIO, SelectIO, SelectRAM, SelectRAM+, Silicon Xpresso, Smartguide, Smart-IP, SmartSearch, SMARTswitch, System ACE, Testbench In A Minute, TrueMap, UIM, VectorMaze, VersaBlock, VersaRing, Virtex-II Pro, Virtex-II EasyPath, Wave Table, WebFITTER, WebPACK, WebPOWERED, XABEL, XACT-Floorplanner, XACT-Performance, XACTstep Advanced, XACTstep Foundry, XAM, XAPP, X-BLOX +, XC designated products, XChecker, XDM, XEPLD, Xilinx Foundation Series, Xilinx XDTV, Xinfo, XSI, XtremeDSP and ZERO+ are trademarks of Xilinx, Inc.

The Programmable Logic Company is a service mark of Xilinx, Inc.

All other trademarks are the property of their respective owners.

Xilinx, Inc. does not assume any liability arising out of the application or use of any product described or shown herein; nor does it convey any license under its patents, copyrights, or maskwork rights or any rights of others. Xilinx, Inc. reserves the right to make changes, at any time, in order to improve reliability, function or design and to supply the best product possible. Xilinx, Inc. will not assume responsibility for the use of any circuitry described herein other than circuitry entirely embodied in its products. Xilinx provides any design, code, or information shown or described herein "as is." By providing the design, code, or information as one possible implementation of a feature, application, or standard, Xilinx makes no representation that such implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of any such implementation, including but not limited to any warranties or representations that the implementation is free from claims of infringement, as well as any implied warranties of merchantability or fitness for a particular purpose. Xilinx, Inc. devices and products are protected under U.S. Patents. Other U.S. and foreign patents pending. Xilinx, Inc. does not represent that devices shown or products described herein are free from patent infringement or from any other third party right. Xilinx, Inc. assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made. Xilinx, Inc. will not assume any liability for the accuracy or correctness of any engineering or software support or assistance provided to a user.

Xilinx products are not intended for use in life support appliances, devices, or systems. Use of a Xilinx product in such applications without the written consent of the appropriate Xilinx officer is prohibited.

The contents of this manual are owned and copyrighted by Xilinx. Copyright 1994-2003 Xilinx, Inc. All Rights Reserved. Except as stated herein, none of the material may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Any unauthorized use of any material contained in this manual may violate copyright laws, trademark laws, the laws of privacy and publicity, and communications regulations and statutes.

Processor IP User Guide

June 2003

The following table shows the revision history for this document..

	Version	Revision
August 2002	1.0	Initial Xilinx release for EDK 3.1
October 2002	1.1	Add memory and peripheral cores
November 2002	1.2	Release for EDK 3.1 Service Pack 2
January 2003	1.3	Release for EDK 3.1 Service Pack 3
March 2003	1.4	Release for EDK 3.2
June 2003	1.5	Release for EDK 3.2 Service Pack 2

About This Manual

The Processor IP Reference Guide supports the Embedded systems Design Kit (EDK) for MicroBlaze™ and Virtex-II Pro™.

Note: For more information, refer to the *Embedded Software Tools Reference Guide* and *PowerPC 405 Processor Reference Guide*.

Manual Contents

This manual contains the following sections:

“Part I: Embedded Processor IP”

- Chapter 1: “OPB Usage in FPGAs”
- Chapter 2: “PLB Usage in Xilinx FPGAs”
- Chapter 3: “Bus Infrastructure Cores”
 - ◆ “On-Chip Peripheral Bus v2.0 with OPB Arbiter (v1.10a)”
 - ◆ “On-Chip Peripheral Bus v2.0 with OPB Arbiter (v1.10b)”
 - ◆ “OPB to PLB Bridge (v1.00a)”
 - ◆ “OPB to PLB Bridge (v1.00b)”
 - ◆ “OPB to OPB Bridge (Lite Version)”
 - ◆ “OPB to DCR Bridge Specification”
 - ◆ “Processor Local Bus (PLB) v3.4”
 - ◆ “PLB to OPB Bridge (v1.00a)”
 - ◆ “PLB to OPB Bridge (v1.00b)”
 - ◆ “Device Control Register Bus (DCR) v2.9”
 - ◆ “Processor System Reset Module”
 - ◆ “Local Memory Bus (LMB) v1.0”
 - ◆ “OPB Arbiter” (v1.02c)”
 - ◆ “Fast Simplex Link Channel (FSL) v1.0”
- Chapter 4: “IPIF”
 - ◆ “OPB IPIF Architecture”
 - “OPB IPIF Slave Attachment”
 - “OPB IPIF Master Attachment”
 - “OPB IPIF Address Decode”
 - “OPB IPIF Interrupt”

- "OPB IP Interface Packet FIFO"
- "Direct Memory Access and Scatter Gather"
- Chapter 5: "Memory Interface Cores"
 - ◆ "LMB Block RAM (BRAM) Interface Controller"
 - ◆ "Dual LMB Block RAM (BRAM) Interface Controller"
 - ◆ "OPB External Memory Controller (EMC) (v1.00d)"
 - ◆ "OPB External Memory Controller (EMC) (v1.10a)"
 - ◆ "OPB Synchronous DRAM (SDRAM) Controller"
 - ◆ "OPB Block RAM (BRAM) Interface Controller"
 - ◆ "OPB Block RAM Interface Controller (OPB_BRAM_IF_CNTL)"
 - ◆ "OPB Double Data Rate (DDR) Synchronous DRAM (SDRAM) Controller"
 - ◆ "OPB SYSACE (System ACE) Interface Controller"
 - ◆ "PLB External Memory Controller (EMC) Design Specification (v1.00d)"
 - ◆ "PLB External Memory Controller (EMC) Design Specification (v1.10a)"
 - ◆ "PLB Synchronous DRAM (SDRAM) Controller"
 - ◆ "PLB Block RAM (BRAM) Interface Controller"
 - ◆ "PLB Double Data Rate (DDR) Synchronous DRAM (SDRAM) Controller"
 - ◆ "DDR Clock Module Reference Core"
 - ◆ "Instruction Side OCM Block RAM (ISBRAM) Interface Controller"
 - ◆ "Data Side OCM Block RAM (DSBRAM) Interface Controller"
 - ◆ "Block RAM (BRAM) Block"
 - ◆ "OPB ZBT Controller Design Specification"
- Chapter 6: "Peripheral Cores"
 - ◆ "OPB Interrupt Controller (v1.00b)"
 - ◆ "OPB Interrupt Controller (v1.00c)"
 - ◆ "OPB 16550 UART"
 - ◆ "OPB 16450 UART"
 - ◆ "OPB UART Lite"
 - ◆ "OPB JTAG_UART"
 - ◆ "OPB IIC Bus Interface"
 - ◆ "OPB Serial Peripheral Interface (SPI)"
 - ◆ "OPB IPIF/LogiCore v3 PCI Core Bridge"
 - ◆ "OPB Ethernet Media Access Controller (EMAC) (v1.00j)"
 - ◆ "OPB Ethernet Media Access Controller (EMAC) (v1.00k)"
 - ◆ "OPB Ethernet Media Access Controller (EMAC) (v1.00m)"
 - ◆ "OPB Ethernet Lite Media Access Controller"
 - ◆ "OPB Asynchronous Transfer Mode Controller (OPB_ATMC) (v1.00b)"
 - ◆ "OPB Asynchronous Transfer Mode Controller (OPB_ATMC) (v2.00a)"
 - ◆ "OPB HDLC Interface" (single channel v1.00b)
 - ◆ "OPB Timebase WDT"

- ◆ "OPB Timer/Counter"
- ◆ "OPB General Purpose Input/Output (GPIO)"
- ◆ "Microprocessor Debug Module (MDM)"
- ◆ "OPB Central DMA Controller"
- ◆ "Channel FIFO"
- ◆ "Fixed Interval Timer (FIT)"
- ◆ "MII to RMII Design Specification"
- ◆ "PLB 1-Gigabit Ethernet Media Access Controller (MAC) With DMA"
- ◆ "PLB 1-Gigabit Ethernet Media Access Controller (MAC)"
- ◆ "PLB 16550 UART (v1.00b)"
- ◆ "PLB 16550 UART (v1.00c)"
- ◆ "PLB 16450 UART (v1.00b)"
- ◆ "PLB 16450 UART (v1.00c)"
- ◆ "PLB RapidIO LVDS Design"
- ◆ "PLB Asynchronous Transfer Mode Controller (PLB_ATMC)"
- ◆ "DCR Interrupt Controller Specification (v1.00a)"
- ◆ "DCR Interrupt Controller Specification (v1.00b)"

"Part II: Software"

- Chapter 7: "Device Driver Programmer Guide"
- Chapter 8: "Tornado 2.0 BSP User Guide"
- Chapter 9: "Device Driver Summary"
- Chapter 10: "Automatic Generation of Tornado 2.x (VxWorks 5.x) Board Support Packages"

This page left intentionally blank

Part I: Embedded Processor IP

This section contains information on the following:

Chapter 1, “[OPB Usage in FPGAs](#)”

Chapter 2, “[PLB Usage in Xilinx FPGAs](#)”

Chapter 3, “[Bus Infrastructure Cores](#)”

Chapter 4, “[IPIF](#)”

Chapter 5, “[Memory Interface Cores](#)”

Chapter 6, “[Peripheral Cores](#)”

OPB Usage in FPGAs

Overview

This chapter includes the following sections:

[Xilinx OPB Usage](#)

[Legacy OPB Devices](#)

[OPB Usage Notes](#)

[OPB Comparison](#)

[Revision History](#)

For detailed information on the IBM OPB, refer to IBM's *On-Chip Peripheral Bus, Architecture Specifications, Version 2.1*: [OpbBus.pdf](#)

The OPB is one element of IBM's CoreConnect architecture, and is a general-purpose synchronous bus designed for easy connection of on-chip peripheral devices. The OPB includes the following features:

- 32-bit or 64-bit data bus
- Up to 64-bit address
- Supports 8-bit, 16-bit, 32-bit, and 64-bit slaves
- Supports 32-bit and 64-bit masters
- Dynamic bus sizing with byte, halfword, fullword, and doubleword transfers
- Optional Byte Enable support
- Distributed multiplexer bus instead of 3-state drivers
- Single cycle transfers between OPB master and OPB slaves (not including arbitration)
- Support for sequential address protocol
- 16-cycle bus time-out (provided by arbiter)
- Slave time-out suppress capability
- Support for multiple OPB bus masters
- Support for bus parking
- Support for bus locking
- Support for slave-requested retry
- Bus arbitration overlapped with last cycle of bus transfers

The OPB is a full-featured bus architecture with many features that increase bus performance. You can use most of these features effectively in the FPGA architecture. However, some features can result in the inefficient use of FPGA resources or can lower system clock rates. Consequently, Xilinx uses an efficient *subset* of the OPB for Xilinx-developed OPB devices. However, because of the flexible nature of FPGAs, you can also implement systems utilizing OPB devices that are fully OPB V2.1 compliant.

Xilinx OPB Usage

OPB Options

Legacy Devices

Previous to OPB v2.0, there was a single signaling protocol for OPB data transfers. This protocol (which is also present in OPB v2.0 and later specifications) supports dynamic bus sizing through the use of transfer qualifiers and acknowledge signals. The transfer qualifiers denote the size of the transfer initiated by the master, and the acknowledge signals indicate the size of the transfer from the slave. Devices that support this type of dynamic bus sizing are called *legacy devices*.

Byte-enable Devices

Starting with OPB v2.0, IBM introduced an optional, alternate transfer protocol based on Byte Enables. In the byte-enable architecture, each byte lane of the data bus has an associated byte enable signal. For each transfer, the byte enable signals indicate which byte lanes have valid data. This eliminates the need for separate transfer qualifiers that indicate the transfer size since all size information is contained in the byte enable signals. The byte-enable architecture does not permit dynamic bus sizing, since there is only one acknowledge signal for each transfer. The OPB v2.0 specification (and later) allows you to build systems that are legacy-only, byte-enable only, or mixed. Devices that only support the byte-enable signaling are called *byte-enable devices*.

OPB V2.0 Devices

Devices that support both byte-enable signaling and legacy signaling are called *OPB v2.0 devices*. Systems that have both legacy signaling and byte-enable signaling can perform dynamic bus sizing. Note that legacy devices do not support byte-enable transfers.

Xilinx OPB Devices

These various transfer protocols have several implications for Xilinx OPB device implementations.

Conversion Cycles

Dynamic bus sizing (as supported by legacy devices) results in *conversion cycles*, which are extra transfer cycles that re-transfer data when the master-initiated transfer is larger than the slave response. For example, in a legacy system, if a master writes a 32-bit word to a slave, and the 8-bit device slave responds that it only accepted 8-bits of the transfer, then the master must perform three additional conversion cycles to transfer all of the data to the slave. Generating conversion cycles requires more logic, increases the complexity of the master, and is not an efficient use of FPGA resources. The byte-enable architecture provides a simple alternative to this problem, and is easier to implement in an FPGA.

Write Mirroring and Read Steering

Another consequence of supporting devices smaller than the bus size is *write mirroring* and *read steering*. In the OPB specification, devices smaller than the bus size are always left-justified (aligned toward the most significant side of the bus) so that the byte lanes associated with the smaller devices are easily determined. For example, a byte-wide peripheral is always located on the most-significant byte of the bus. The peripheral writes and reads data using this byte-lane. You can simplify the design of OPB masters by using a byte-enable only, no-write-mirroring architecture. A small degree of added complexity is required for peripherals that are smaller than the bus size if OPB masters do not mirror data.

Ideal FPGA Implementation of OPB-based System

The ideal FPGA implementation of an OPB-based system has the following features:

- Requires no conversion cycles
- Uses only the byte-enable architecture as specified in the OPB specification
- Does not require masters to mirror write data

These characteristics help determine how Xilinx-developed OPB devices are implemented. The detailed specifications that describe how the OPB is used in Xilinx intellectual property are provided in the next section.

Specifications for OPB Usage in Xilinx-developed OPB Devices

Xilinx-developed OPB devices adhere to the following OPB usage rules:

- The width of the OPB data buses and address buses is 32 bits. Some peripherals may parameterize these widths, but currently only 32-bit buses are supported. Peripherals smaller than 32-bits can be attached to the OPB with a corresponding restriction in addressing. For example, an 8-bit peripheral at base address A can be attached to byte lane 0, but can only be addressed at A, A+4, A+8, and so on.
- All OPB devices (masters and slaves) are byte-enable devices. These devices do not support the legacy data transfer signals and therefore do not support dynamic bus sizing. OPB masters do not mirror data to unused byte lanes. See [Figure 1-1](#) for the byte lane usage for aligned transfers.
- All OPB devices (masters and slaves) are required to output logic zero when inactive. This eliminates the need for the Mn_DBusEn and Sln_DBusEn signals external to the master or slave. The enable function is still implemented within the device.
- To obtain better timing in the FPGA implementation of the OPB, the OPB_timeout signal is registered. This means that all slaves must assert Sl_xferAck or Sl_retry on or before the rising edge of the 16th clock cycle after the assertion of OPB_select. If an OPB slave wishes to assert Sl_toutSup, Sl_toutSup must be asserted on or before the rising edge of the 15th clock after the assertion of OPB_select.
- The byte-enables and the least-significant address bits are driven by all masters and contain consistent information. Examples of byte lane usage for aligned transfers are shown in the following figure:

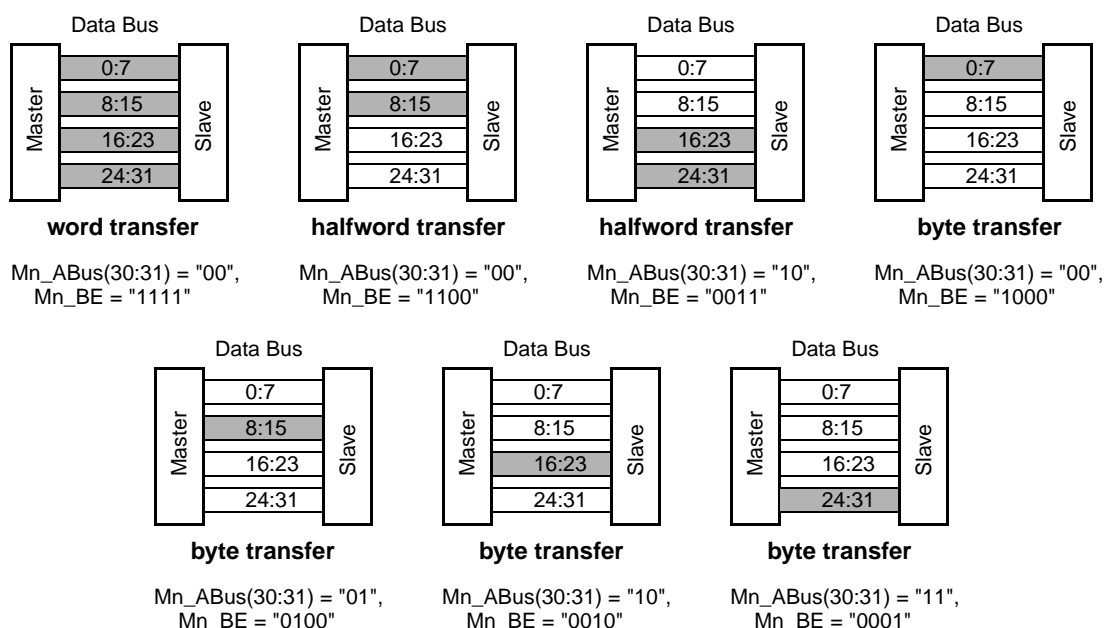


Figure 1-1: Byte lane usage for aligned transfers

- All OPB slave devices that require a continuous address space (use of all byte lanes) will implement an attachment to the OPB bus that is as wide as the OPB data width, regardless of device width. This eliminates the need for left justification on the OPB bus and eliminates the need for masters to mirror write data.

As an example, consider an 8-bit memory device that must be addressed at consecutive byte addresses being attached to a 32-bit OPB. The 8-bit memory device must implement a 32-bit wide attachment to the OPB; in the bus attachment, data is steered from the proper byte lane into the 8-bit device for writes, and from the 8-bit device onto the proper byte lane for reads.

The simplest way to accomplish this is with a multiplexer for steering the writes, and a connection from the 8-bit device to all byte lanes (essentially mirroring to all byte lanes) for reads.

- By convention, registers in all OPB slave devices are aligned to word boundaries (lowest two address bits are "00"), regardless of the size of the data in the register or the size of the peripheral.
- Master and Slave I/O: OPB masters adhere to the signal set shown in [Table 1-1](#). OPB slaves adhere to the signal set shown in [Table 1-2](#). Devices that are both master and slave adhere to the signal set shown in [Table 1-3](#). Page numbers referenced in the tables apply to both the OPB V2.0 specification and the OPB V2.1 specification, both from IBM. All signals shown must be present, except for the one signal shown as optional (<Master>_DBus[0:31] for devices that are both master and slave). No additional signals for OPB interconnection may be added. The naming convention is as follows: <Master> represents a master name or acronym that starts with an upper-case letter, <Slave> represents a slave name or acronym that starts with an upper-case letter. <nOPB> represents an OPB identifier (for masters or slaves with more than OPB attachment) and must start with an uppercase letter and end with upper-case "OPB". For devices with a single OPB attachment, the <nOPB> identifier should default to "OPB" (for example, OPB_ABus). All other parts of the signal name must be referenced exactly as shown (including case).

Table 1-1: Summary of OPB master-only I/O

Signal	I/O	Description	Page (in Ref. 1)
<nOPB>_Clk	I	OPB Clock	
<nOPB>_Rst	I	OPB Reset	
<Master>_ABus[0:31]	O	Master address bus	OPB-11
<Master>_BE[0:3]	O	Master byte enables	OPB-16
<Master>_busLock	O	Master buslock	OPB-9
<Master>_DBus[0:31]	O	Master write data bus	OPB-13
<Master>_request	O	Master bus request	OPB-8
<Master>_RNW	O	Master read, not write	OPB-12
<Master>_select	O	Master select	OPB-12
<Master>_seqAddr	O	Master sequential address	OPB-13
<nOPB>_DBus[0:31]	I	OPB read data bus	OPB-13
<nOPB>_errAck	I	OPB error acknowledge	OPB-15
<nOPB>_MGrant	I	OPB bus grant	OPB-9

Table 1-1: Summary of OPB master-only I/O <Italic>(Continued)

Signal	I/O	Description	Page (in Ref. 1)
<nOPB>_retry	I	OPB bus cycle retry	OPB-10
<nOPB>_timeout	I	OPB timeout error	OPB-10
<nOPB>_xferAck	I	OPB transfer acknowledge	OPB-14

Table 1-2: Summary of OPB Slave-only I/O

Signal	I/O	Description	Page (in Ref. 1)
<nOPB>_Clk	I	OPB Clock	
<nOPB>_Rst	I	OPB Reset	
<Slave>_DBus[0:31]	O	Slave data bus	OPB-11
<Slave>_errAck	O	Slave error acknowledge	OPB-15
<Slave>_retry	O	Slave retry	OPB-10
<Slave>_toutSup	O	Slave timeout suppress	OPB-15
<Slave>_xferAck	O	Slave transfer acknowledge	OPB-14
<nOPB>_ABus[0:31]	I	OPB address bus	OPB-11
<nOPB>_BE	I	OPB byte enable	OPB-16
<nOPB>_DBus[0:31]	I	OPB data bus	OPB-13
<nOPB>_RNW	I	OPB read/not write	OPB-12
<nOPB>_select	I	OPB select	OPB-12
<nOPB>_seqAddr	I	OPB sequential address	OPB-13

Table 1-3: Summary of OPB Master/Slave Device I/O

Signal	I/O	Description	Page (in Ref. 1)
<nOPB>_Clk	I	OPB Clock	
<nOPB>_Rst	I	OPB Reset	
<Master>_ABus[0:31]	O	Master address bus	OPB-11
<Master>_BE[0:3]	O	Master byte enables	OPB-16
<Master>_busLock	O	Master buslock	OPB-9
<Master>_DBus[0:31]	O	Master write data bus (optional)	OPB-13
<Master>_request	O	Master bus request	OPB-8
<Master>_RNW	O	Master read, not write	OPB-12
<Master>_select	O	Master select	OPB-12
<Master>_seqAddr	O	Master sequential address	OPB-13
<nOPB>_DBus[0:31]	I	OPB read data bus	OPB-13

Table 1-3: Summary of OPB Master/Slave Device I/O <Italic>(Continued)

Signal	I/O	Description	Page (in Ref. 1)
<nOPB>_errAck	I	OPB error acknowledge	OPB-15
<nOPB>_MGrant	I	OPB bus grant	OPB-9
<nOPB>_retry	I	OPB bus cycle retry	OPB-10
<nOPB>_timeout	I	OPB timeout error	OPB-10
<nOPB>_xferAck	I	OPB transfer acknowledge	OPB-14
<Slave>_DBus[0:31]	O	Slave data bus (may optionally function as master write data bus if <Master>_DBus not present)	OPB-11
<Slave>_errAck	O	Slave error acknowledge	OPB-15
<Slave>_retry	O	Slave retry	OPB-10
<Slave>_toutSup	O	Slave timeout suppress	OPB-15
<Slave>_xferAck	O	Slave transfer acknowledge	OPB-14
<nOPB>_ABus[0:31]	I	OPB address bus	OPB-11
<nOPB>_BE	I	OPB byte enable	OPB-16
<nOPB>_RNW	I	OPB read/not write	OPB-12
<nOPB>_select	I	OPB select	OPB-12
<nOPB>_seqAddr	I	OPB sequential address	OPB-13

Additional Notes on Signal Sets

- Xilinx-developed OPB devices do not support dynamic bus sizing and consequently *do not* use the following legacy signals: Mn_dwXfer, Mn_fwXfer, Mn_hwXfer, Sln_dwAck, Sln_fwAck, and Sln_hwAck.
- Since Xilinx-developed OPB devices are byte-enable only, the Mn_beXfer and Sln_beAck signals are not required and so are not used.
- The signals required for masters and slaves are separate from the signals present in the OPB interconnect. The OPB interconnect (the OR gates and other logic required to connect OPB devices) supports the full OPB V2.1 specification (i.e. all signals are present). Thus the OPB interconnect does not limit a design to byte-enable devices and supports designs in which a mix of byte-enable, legacy, and OPB V2.0 devices are present. The bus interconnect does not limit the use of any feature of the V2.1 specification.

Legacy OPB Devices

Although byte-enable devices are the preferred and most efficient OPB devices in Xilinx devices, some designs may also use legacy OPB devices or fully V2.0 compliant devices. However, a legacy device cannot communicate directly with a byte-enable device because they use different signal sets. An interface layer between the byte-enable device and the legacy device is required. This interface is called the Byte Enable Interface (BEIF) device.

Mixed Systems

The system shown in the following figure represents a design with a mix of byte-enable, legacy, and OPB V2.0 devices. The BEIF device converts the legacy-type signals to byte-enable-type signals and vice versa.

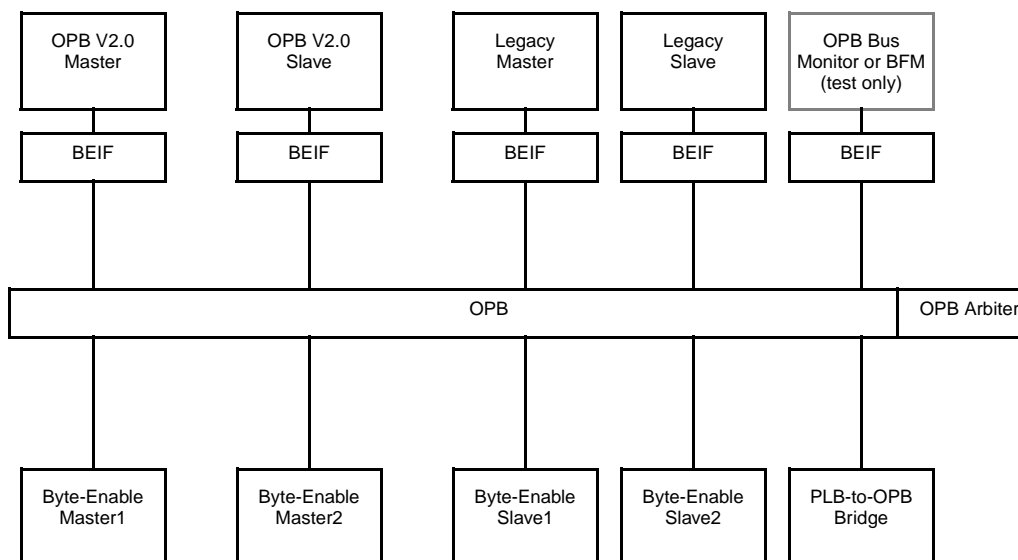


Figure 1-2: OPB Interconnect with Mixed Device Types

The BEIF device contains the following logic, not all of which must be used in all situations:

- Signal translation for byte-enable device to legacy device transfers: $\langle \text{Master} \rangle_BE$ is translated to the appropriate $\langle \text{Master} \rangle_hwXfer$, $\langle \text{Master} \rangle_fwXfer$, and $\langle \text{Master} \rangle_dwXfer$. $\langle nOPB \rangle_BE$ is translated to the appropriate $\langle nOPB \rangle_hwXfer$, $\langle nOPB \rangle_fwXfer$, and $\langle nOPB \rangle_dwXfer$. $\langle \text{Slave} \rangle_hwXfer$, $\langle \text{Slave} \rangle_fwXfer$, and $\langle \text{Slave} \rangle_dwXfer$ are translated to $\langle \text{Slave} \rangle_xferAck$. $\langle nOPB \rangle_hwXfer$, $\langle nOPB \rangle_fwXfer$, and $\langle nOPB \rangle_dwXfer$ are translated to $\langle nOPB \rangle_xferAck$. The correct lower address bits are also generated.
- Signal translation for legacy device to byte-enable device transfers: $\langle \text{Master} \rangle_hwXfer$, $\langle \text{Master} \rangle_fwXfer$, and $\langle \text{Master} \rangle_dwXfer$ are translated to $\langle \text{Master} \rangle_BE$. $\langle nOPB \rangle_hwXfer$, $\langle nOPB \rangle_fwXfer$, and $\langle nOPB \rangle_dwXfer$ are translated to $\langle nOPB \rangle_BE$. $\langle \text{Slave} \rangle_xferAck$ is translated to $\langle \text{Slave} \rangle_hwXfer$, $\langle \text{Slave} \rangle_fwXfer$, and $\langle \text{Slave} \rangle_dwXfer$. $\langle nOPB \rangle_xferAck$ is translated to $\langle nOPB \rangle_hwXfer$, $\langle nOPB \rangle_fwXfer$, and $\langle nOPB \rangle_dwXfer$.
- Mirroring and steering logic.
- Conversion cycle generator for byte-enable device to legacy device transfers.

With this architecture, systems that do not require full V2.1 features (for example, systems that contain only Xilinx IP) do not need to instantiate the BEIF and hence optimally use the available FPGA resources. Systems that require legacy or OPB V2.0 devices must instantiate the BEIF, although the most costly part of the BEIF (the conversion cycle generator) only needs to be instantiated if conversion cycles are possible (not all slaves will cause generation of conversion cycles).

OPB Usage Notes

The following are general notes on OPB usage that apply primarily to mixed systems:

- Conversion cycles are only required when a master generates a transfer request to a slave that is larger than the slave's width *and* the slave is capable of indicating that it accepted a smaller transfer than the master requested hence requiring with a conversion cycle.
- Byte-enable masters cannot directly generate conversion cycles. They require a conversion cycle generator in the Byte Enable Interface (BEIF) device. This is because byte-enable masters do not receive any size information in the acknowledge from the slave.
- Byte-enable slaves cannot cause generation of conversion cycles. A consequence of this is that any master accessing a byte-enable slave can only transfer data up to the size of the slave. Transfers larger than the slave size will result in either 1) no response from the slave (time-out), 2) an errAck from the slave, or 3) lost data; the actual result depends on how the decode and acknowledge logic is implemented in the slave.
- Conversion cycle generator logic in the BEIF is required only for byte-enable device to legacy/OPB V2.0 device transfers.
- Write mirroring and read steering in the V2.1 specification is based on left-justified peripherals. A more complex slave attachment can be used instead of left justification.

OPB Comparison

Table 1-4 illustrates the major embedded processor bus architectures used in Xilinx FPGAs and lists some of their characteristics. Each bus has different capabilities in terms of data transfer rates, multi-master capability, and data bursting. The use of a particular bus is dictated by the processor used, the data bandwidth required in the application, and availability of peripherals. The OPB is a general-purpose peripheral bus that can be effectively used in many design situations.

PLB - Processor Local Bus (IBM). [PLB Reference](#)

OPB - On-chip Peripheral Bus (IBM). [OPB Reference](#)

OCM - On-chip Memory interface (IBM). [OCM Reference](#)

LMB - Local Memory Bus (Xilinx). *MicroBlaze Processor Reference Guide*

DCR - Device Control Register bus (IBM). [DCR Reference](#)

Table 1-4: Comparison of buses used in Xilinx Embedded Processor Systems

Feature	CoreConnect Buses			Other Buses	
	PLB	OPB	DCR	OCM	LMB
Processor family	PPC405	PPC405, MicroBlaze	PPC405	PPC405	MicroBlaze
Data bus width	64	32	32	32	32
Address bus width	32	32	10	32	32
Clock rate, MHz (max) ¹	100	125	125	375	125
Masters (max)	16	16	1	1	1
Masters (typical)	2-8	2-8	1	1	1
Slaves (max)	limited only by hardware resources			1	1
Slaves (typical)	2-6	2-8	1-8	1	1
Data rate (peak) ²	1600 MB/s	500 MB/s	500 MB/s	500 MB/s	500 MB/s

Table 1-4: Comparison of buses used in Xilinx Embedded Processor Systems <Italic>(Continued)

Feature	CoreConnect Buses			Other Buses	
	PLB	OPB	DCR	OCM	LMB
Data rate (typical) ³	533 MB/s ⁴	167 MB/s ⁵	100 MB/s ⁸	333 MB/s ⁶	333 MB/s ⁷
Concurrent read/write	Yes	No	No	No	No
Address pipelining	Yes	No	No	No	No
Bus locking	Yes	Yes	No	No	No
Retry	Yes	Yes	No	No	No
Timeout	Yes	Yes	No	No	No
Fixed burst	Yes	No	No	No	No
Variable burst	Yes	No	No	No	No
Cache fill	Yes	No	No	No	No
Target word first	Yes	No	No	No	No
FPGA resource usage	High	Medium	Low	Low	Low
Compiler support for load/store	Yes	Yes	No	Yes	Yes

Notes:

1. Maximum clock rates are estimates and are presented for comparison only. The actual maximum clock rate for each bus is dependent on device family, device speed grade, design complexity, and other factors.
2. Peak data rate is the maximum theoretical data transfer rate at the clock rate shown for each bus.
3. The typical data rates are intended to illustrate data rates that are representative of actual system configurations. The typical data is highly dependent on the application software and system hardware configuration.
4. Assumes primarily cache-line fills, minimal read/write concurrency (66.7% bus utilization).
5. Assumes minimal use of sequential address capabilities and 3 clock cycles per OPB transfer.
6. The OCM controller operates at the PPC405 core clock rate, but its data transfer rate is limited by the access time of the on-chip memory. The typical data rate assumes 66.7% bus utilization.
7. Assumes 66.7% bus utilization.
8. Assumes DCR operates at same clock rate as PLB and each DCR access requires 5 clock cycles. The number of clock cycles per DCR transfer is dependent on how many DCR devices are present in the system. Each additional DCR device adds latency to all DCR transfers.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/17/01	1.0	Initial Xilinx version.
10/19/01	1.1	Minor editorial changes. Added links to bus references.
12/10/01	1.2	Changed Figure 2 and other minor edits.
3/20/02	1.3	Updated for MDK 2.2

PLB Usage in Xilinx FPGAs

Summary

This chapter describes how to use the IBM Processor Local Bus (PLB) in Xilinx FPGAs, and provides guidelines and simplifications for efficient FPGA implementations, and the set of signals used in Xilinx-developed PLB devices.

This chapter includes the following sections:

[Xilinx PLB Usage](#)

[PLB Comparison](#)

[Revision History](#)

Overview

For detailed information on the IBM PLB, refer to IBM's *64-bit Processor Local Bus, Architecture Specifications, Version 3.5*.

The PLB is one element of IBM's CoreConnect architecture, and is a high-performance synchronous bus designed for connection of processors to high-performance peripheral devices. The PLB includes the following features (from *64-bit Processor Local Bus, Architecture Specifications*):

- Overlapping of read and write transfers allows two data transfers per clock cycle for maximum bus utilization.
- Decoupled address and data buses support split-bus transaction capability for improved bandwidth.
- Address pipelining reduces overall bus latency by allowing the latency associated with a new request to be overlapped with an ongoing data transfer in the same direction.
- Late master request abort capability reduces latency associated with aborted requests.
- Hidden (overlapped) bus request/grant protocol reduces arbitration latency.
- Bus architecture supports sixteen masters and any number of slave devices.
- Four levels of request priority for each master allow PLB implementations with various arbitration schemes.
- Bus arbitration-locking mechanism allows for master-driven atomic operations.
- Support for 16-, 32-, and 64-byte line data transfers.
- Read word address capability allows slave devices to fetch line data in any order (that is, target word-first or sequential).
- Sequential burst protocol allows byte, halfword, and word burst data transfers in either direction.
- Guarded and unguarded memory transfers allow a slave device to enable or disable the pre-fetching of instructions or data.

The PLB is a full-featured bus architecture with many features that increase bus performance. Most of these features map well to the FPGA architecture, however, some can result in the inefficient use of FPGA resources or can lower system clock rates. Consequently, Xilinx uses an efficient *subset* of the PLB for Xilinx-developed PLB devices. However, because of the flexible nature of FPGAs, you can also implement systems utilizing PLB devices that are fully PLB V3.5 compliant.

Xilinx PLB Usage

Dynamic Bus Sizing

Dynamic bus sizing is a PLB architectural feature that allows a designer to mix 32 and 64-bit devices on the same 64-bit PLB. A master provides a master size signal, `<Master>_MSize[0:1]`, that describes the data width of the master initiating a transaction. Slaves provide a similar signal, `Sl_Mn_SSize(0:1)`, with the address acknowledge that describes the data width of the slave that is responding to the transaction. While dynamic bus sizing is a useful architectural feature, its use in FPGAs can result in inefficient implementations of PLB masters.

Conversion Cycles

Dynamic bus sizing results in *conversion cycles*, which are extra transfer cycles that re-transfer data when the master-initiated transfer is larger than the slave response. For example, if a master writes a 64-bit word to a slave, and the 32-bit device slave responds with a slave size of 32-bits, then the master must perform an additional conversion cycle to transfer all of the data to the slave. Generating conversion cycles requires more logic, increases the complexity of the master, and is typically not an efficient use of FPGA resources.

Write Mirroring and Read Steering

Another consequence of supporting devices smaller than the bus size is *write mirroring* and *read steering*. In the PLB specification, devices smaller than the bus size are always left-justified (aligned toward the most significant side of the bus) so that the byte lanes associated with the smaller devices are easily determined. For example, a word-wide peripheral is always located on the most-significant word of the 64-bit bus. The peripheral writes and reads data using only the four most significant byte lanes. You can simplify the design of PLB masters by using an architecture that requires no write mirroring and transfers data based on which byte enables are active. A small degree of added complexity is required in the bus attachment for peripherals that are smaller than the bus size if PLB masters do not mirror data. This additional logic is built into the parameterizable slave attachment in each Xilinx peripheral.

Xilinx PLB Devices

Ideal FPGA Implementation of PLB-based System

The ideal FPGA implementation of a PLB-based system has the following features:

- Requires no conversion cycles
- Does not require masters to mirror write data

These characteristics help determine how Xilinx-developed PLB devices are implemented. The detailed specifications that describe how the PLB is used in Xilinx intellectual property are provided in the next section.

Specifications for PLB Usage in Xilinx-developed PLB Devices

Xilinx-developed PLB devices adhere to the following PLB usage rules:

- The width of the PLB data buses is 64 bits and the width of address buses is 32 bits. Note that some peripherals may parameterize these widths, but currently only 64-bit data buses are supported. Peripherals that are smaller than 64-bits can be attached to the PLB with a corresponding restriction in addressing. For example, a 32-bit peripheral at base address A can be attached to byte lanes 0 – 4, but word-wide accesses can only be addressed at A, A+8, A+16, etc.
- PLB masters are not required to support dynamic bus sizing. PLB masters are not required to mirror data to unused byte lanes. See [Figure 2-1](#) and [Figure 2-2](#) for the byte lane usage for aligned transfers. PLB Masters are required to correctly drive the <Master>_MSize[0:1] signals. PLB slaves are required to correctly drive the <Slave>_SSize[0:1] signals for PLB masters that do provide conversion cycles (such as the PowerPC 405).
- All PLB slaves are required to output logic zero when they are inactive.
- The byte-enables and the least-significant address bits are driven by all masters and contain consistent information. Examples of byte lane usage for aligned transfers are shown in [Figure 2-1](#) and [Figure 2-2](#).

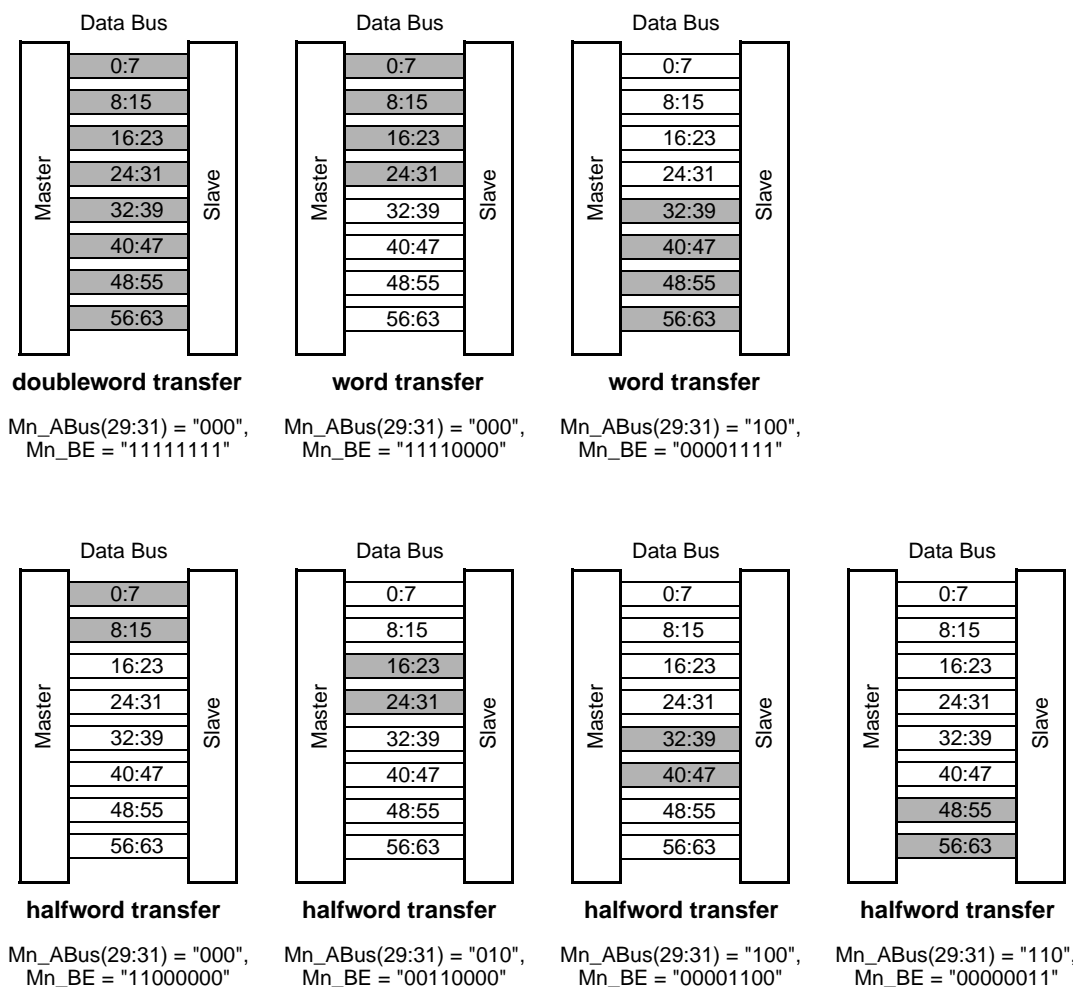


Figure 2-1: Byte lane usage for aligned doubleword, word, and halfword transfers

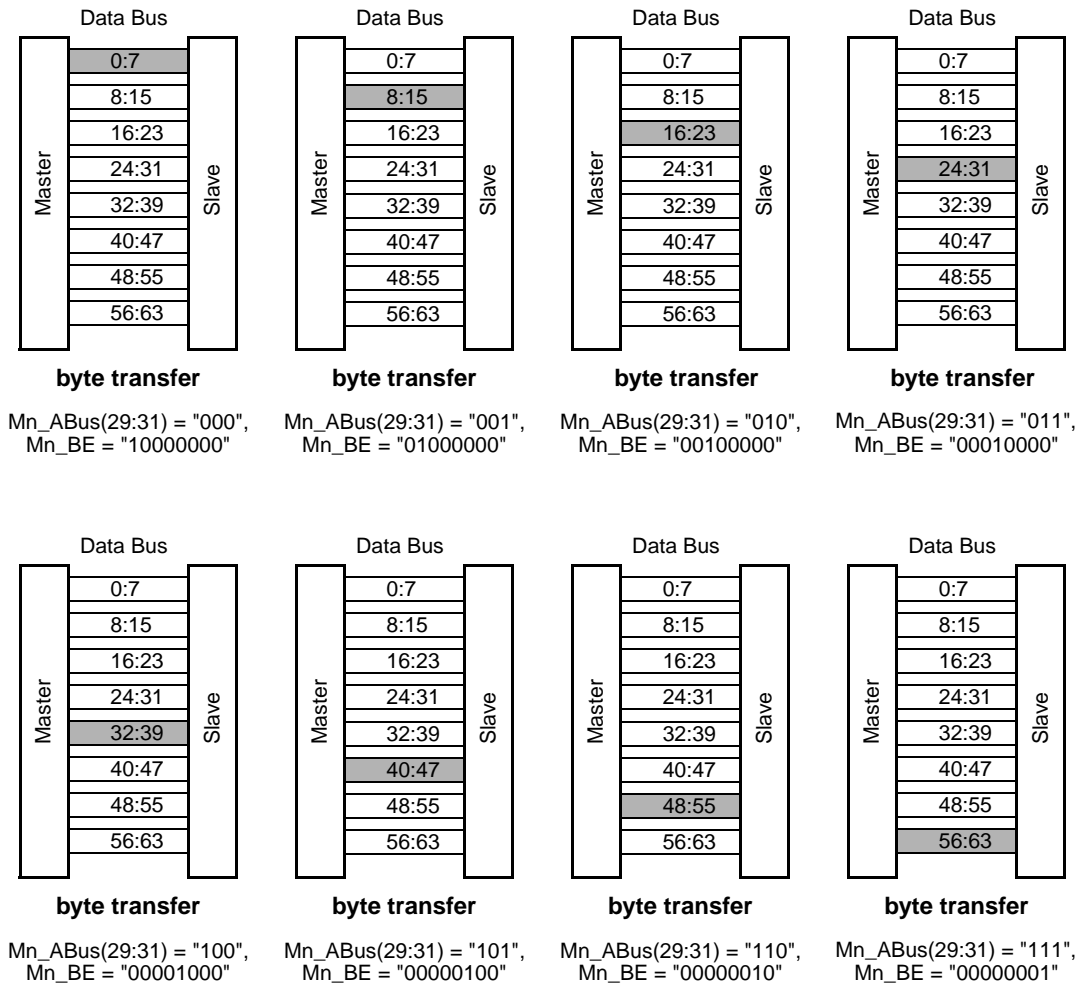


Figure 2-2: Byte lane usage for byte transfers

- All PLB slave devices that require a continuous address space (i.e. use of all byte lanes) will implement an attachment to the PLB bus that is as wide as the PLB data width, regardless of device width. This eliminates the need for left justification on the PLB bus and eliminates the need for masters to mirror write data. As an example, consider a 32-bit memory device that must be addressed at consecutive byte addresses being attached to a 64-bit PLB. The 32-bit memory device must implement a 64-bit wide attachment to the PLB; in the bus attachment, data is steered from the proper byte lanes into the 32-bit device for writes, and from the 32-bit device onto the proper byte lanes for reads.
- By convention, registers in all PLB slave devices are aligned to word boundaries (lowest two address bits are "00"), regardless of the size of the data in the register or the size of the peripheral.
- Master and Slave I/O: PLB masters adhere to the signal set shown in Table 2-1. PLB slaves adhere to the signal set shown in Table 2-2. Page numbers referenced in the tables apply to the PLB V3.5 specification from IBM. All signals shown must be present. No additional signals for PLB interconnection may be added. The naming convention is as follows: <Master> represents a master name or acronym that starts with an upper-case letter, <Slave> represents a slave name or acronym that starts with an upper-case letter. <nPLB> represents an PLB identifier (for masters or slaves with more than one PLB attachment) and must start with an uppercase letter and end with upper-case "PLB". For devices with a single PLB attachment, the <nPLB> identifier

should default to "PLB" (for example, PLB_ABus). All other parts of the signal name must be referenced exactly as shown (including case).

Table 2-1: Summary of PLB Master-only I/O

Signal	I/O	Description	Page (in Ref. 1)
<nPLB>_Clk	I	PLB Clock (SYS_plbClk)	PLB-11
<nPLB>_Rst	I	PLB Reset (SYS_plbReset)	PLB-11
<Master>_abort	O	Master abort bus request indicator	PLB-19
<Master>_ABus[0:31]	O	Master address bus	PLB-27
<Master>_BE[0:7]	O	Master byte enables	PLB-21
<Master>_busLock	O	Master buslock	PLB-13
<Master>_compress	O	Master compressed data transfer indicator	PLB-25
<Master>_guarded	O	Master guarded transfer indicator	PLB-26
<Master>_lockErr	O	Master lock error indicator	PLB-27
<Master>_MSize[0:1]	O	Master data bus size	PLB-40
<Master>_ordered	O	Master synchronize transfer indicator	PLB-26
<Master>_priority[0:1]	O	Master request priority	PLB-12
<Master>_rdBurst	O	Master burst read transfer indicator	PLB-34
<Master>_request	O	Master request	PLB-12
<Master>_RNW	O	Master read/not write	PLB-21
<Master>_size[0:3]	O	Master transfer size	PLB-24
<Master>_type[0:2]	O	Master transfer type	PLB-25
<Master>_wrBurst	O	Master burst write transfer indicator	PLB-29
<Master>_wrDBus[0:63]	O	Master write data bus	PLB-28
<nPLB>_<Master>_Busy	I	PLB master slave busy indicator	PLB-36
<nPLB>_<Master>_Err	I	PLB master slave error indicator	PLB-37
<nPLB>_<Master>_WrBTerm	I	PLB master terminate write burst indicator	PLB-30
<nPLB>_<Master>_WrDAck	I	PLB master write data acknowledge	PLB-29
<nPLB>_<Master>_AddrAck	I	PLB master address acknowledge	PLB-18
<nPLB>_<Master>_RdBTerm	I	PLB master terminate read burst indicator	PLB-36
<nPLB>_<Master>_RdDAck	I	PLB master read data acknowledge	PLB-33

Table 2-1: Summary of PLB Master-only I/O <Italic>(Continued)

Signal	I/O	Description	Page (in Ref. 1)
<nPLB>_<Master>RdDBus[0:63]	I	PLB master read data bus	PLB-31
<nPLB>_<Master>RdWdAddr[0:3]	I	PLB master read word address	PLB-32
<nPLB>_<Master>Rearbitrate	I	PLB master bus re-arbitrate indicator	PLB-19
<nPLB>_<Master>SSize[0:1]	I	PLB slave data bus size	PLB-40

Table 2-2: Summary of PLB Slave-only I/O

Signal	I/O	Description	Page (in Ref. 1)
<nPLB>_Clk	I	PLB Clock (SYS_plbClk)	PLB-11
<nPLB>_Reset	I	PLB Reset (SYS_plbReset)	PLB-11
<Slave>_addrAck	O	Slave address acknowledge	PLB-18
<Slave>_MBusy[0:3]	O	Slave busy indicator	PLB-36
<Slave>_MErr[0:3]	O	Slave error indicator	PLB-37
<Slave>_rdBTerm	O	Slave terminate read burst transfer	PLB-36
<Slave>_rdComp	O	Slave read transfer complete indicator	PLB-34
<Slave>_rdDAck	O	Slave read data acknowledge	PLB-33
<Slave>_rdDBus[0:63]	O	Slave read data bus	PLB-31
<Slave>_rdWdAddr[0:3]	O	Slave read word address	PLB-32
<Slave>_rearbitrate	O	Slave re-arbitrate bus indicator	PLB-19
<Slave>_SSize[0:1]	O	Slave data bus size	PLB-40
<Slave>_wait	O	Slave wait indicator	PLB-18
<Slave>_wrBTerm	O	Slave terminate write burst transfer	PLB-30
<Slave>_wrComp	O	Slave write transfer complete indicator	PLB-29
<Slave>_wrDAck	O	Slave write data acknowledge	PLB-29
<nPLB>_abort	I	PLB abort request indicator	PLB-19
<nPLB>_ABus[0:31]	I	PLB address bus	PLB-27
<nPLB>_BE[0:7]	I	PLB byte enables	PLB-21
<nPLB>_busLock	I	PLB bus lock	PLB-13
<nPLB>_compress	I	PLB compressed data transfer indicator	PLB-25
<nPLB>_guarded	I	PLB guarded transfer indicator	PLB-26
<nPLB>_lockErr	I	PLB lock error indicator	PLB-27
<nPLB>_masterID[0:1]	I	PLB current master identifier	PLB-20

Table 2-2: Summary of PLB Slave-only I/O <Italic>(Continued)

Signal	I/O	Description	Page (in Ref. 1)
<nPLB>_MSize[0:1]	I	PLB master data bus size	PLB-40
<nPLB>_ordered	I	PLB synchronize transfer indicator	PLB-26
<nPLB>_PAValid	I	PLB primary address valid indicator	PLB-13
<nPLB>_pendPri[0:1]	I	PLB pending request priority	PLB-20
<nPLB>_pendReq	I	PLB pending bus request indicator	PLB-20
<nPLB>_rdBurst	I	PLB burst read transfer indicator	PLB-34
<nPLB>_rdPrim	I	PLB secondary to primary read request indicator	PLB-36
<nPLB>_reqPri[0:1]	I	PLB current request priority	PLB-20
<nPLB>_RNW	I	PLB read/not write	PLB-21
<nPLB>_SAValid	I	PLB secondary address valid indicator	PLB-16
<nPLB>_size[0:3]	I	PLB transfer size	PLB-24
<nPLB>_type[0:2]	I	PLB transfer type	PLB-25
<nPLB>_wrBurst	I	PLB burst write transfer indicator	PLB-29
<nPLB>_wrDBus[0:63]	I	PLB write data bus	PLB-28
<nPLB>_wrPrim	I	PLB secondary to primary write request indicator	PLB-31

PLB Comparison

Table 2-3 illustrates the major embedded processor bus architectures used in Xilinx FPGAs and lists some of their characteristics. Each bus has different capabilities in terms of data transfer rates, multi-master capability, and data bursting. The use of a particular bus is dictated by the processor used, the data bandwidth required in the application, and availability of peripherals. The PLB is a high-performance local bus that can be effectively used in many design situations.

PLB - Processor Local Bus (IBM). [PLB Reference](#)

OPB - On-chip Peripheral Bus (IBM). [OPB Reference](#)

OCM - On-chip Memory interface (IBM). [OCM Reference](#)

DCR - Device Control Register bus (IBM). [DCR Reference](#)

Table 2-3: Comparison of buses used in Xilinx embedded processor systems

Feature	CoreConnect Buses			Other Buses	
	PLB	OPB	DCR	OCM	LMB
Processor family	PPC405	PPC405, MicroBlaze	PPC405	PPC405	MicroBlaze
Data bus width	64	32	32	32	32
Address bus width	32	32	10	32	32
Clock rate, MHz (max) ¹	100	125	125	375	125
Masters (max)	16	16	1	1	1
Masters (typical)	2-8	2-8	1	1	1
Slaves (max)	limited only by hardware resources			1	1
Slaves (typical)	2-6	2-8	1-8	1	1
Data rate (MB/s, peak) ²	1600	500	500	500	500
Data rate (MB/s, typical) ³	533 ⁴	167 ⁵	100 ⁸	333 ⁶	333 ⁷
Concurrent read/write	Yes	No	No	No	No
Address pipelining	Yes	No	No	No	No
Bus locking	Yes	Yes	No	No	No
Retry	Yes	Yes	No	No	No
Timeout	Yes	Yes	No	No	No
Fixed burst	Yes	No	No	No	No
Variable burst	Yes	No	No	No	No
Cache fill	Yes	No	No	No	No
Target word first	Yes	No	No	No	No
FPGA resource usage	High	Medium	Low	Low	Low
Compiler support for load/store	Yes	Yes	No	Yes	Yes

Notes:

1. Maximum clock rates are estimates and are presented for comparison only. The actual maximum clock rate for each bus is dependent on device family, device speed grade, design complexity, and other factors.
2. Peak data rate is the maximum theoretical data transfer rate at the clock rate shown for each bus.
3. The typical data rates are intended to illustrate data rates that are representative of actual system configurations. The typical data is highly dependent on the application software and system hardware configuration.
4. Assumes primarily cache-line fills, minimal read/write concurrency (66.7% bus utilization).
5. Assumes minimal use of sequential address capabilities and 3 clock cycles per OPB transfer.
6. The OCM controller operates at the PPC405 core clock rate, but its data transfer rate is limited by the access time of the on-chip memory. The typical data rate assumes 66.7% bus utilization.
7. Assumes 66.7% bus utilization.
8. Assumes DCR operates at same clock rate as PLB and each DCR access requires 5 clock cycles. The number of clock cycles per DCR transfer is dependent on how many DCR devices are present in the system. Each additional DCR device adds latency to all DCR transfers.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
5/8/02	1.0	Initial Xilinx version.

Bus Infrastructure Cores

This section of the User Guide contains information on the following bus infrastructure cores:

[On-Chip Peripheral Bus v2.0 with OPB Arbiter \(v1.10a\)](#)

[On-Chip Peripheral Bus v2.0 with OPB Arbiter \(v1.10b\)](#)

[OPB to PLB Bridge \(v1.00a\)](#)

[OPB to PLB Bridge \(v1.00b\)](#)

[OPB to OPB Bridge \(Lite Version\)](#)

[OPB to DCR Bridge Specification](#)

[Processor Local Bus \(PLB\) v3.4](#)

[PLB to OPB Bridge \(v1.00a\)](#)

[PLB to OPB Bridge \(v1.00b\)](#)

[Device Control Register Bus \(DCR\) v2.9](#)

[Processor System Reset Module](#)

[Local Memory Bus \(LMB\) v1.0](#)

[OPB Arbiter \(v1.02c\)](#)

[Fast Simplex Link Channel \(FSL\) v1.0](#)

Click [here](#) to view this data sheet

Product Overview

Introduction

The OPB_V20 module is used as the OPB interconnect for Xilinx FPGA based embedded processor systems. The bus interconnect in the OPB v2.0 specification is a distributed multiplexer implemented as an “and” function in the master or slave driving the bus and an OR to combine the drivers into a single bus.

The OPB_V20 module assumes the AND, or enable function is within the master or slave and provides the OR function to combine the various bus signals.

Features

The features of the OPB_V20 are:

- Includes parameterized OPB Arbiter
- Includes parameterized I/O signals to support and number of masters or slaves
- Includes all signals present in the OPB v2.0 Specification except the DMA handshake signals
- The OR structure can be implemented using only LUTs or a combination of LUTs and fast carry to reduce the number of LUTs in the OR interconnect
- Includes a 16-clock Power-on OPB Bus Reset and parameter for high or low external bus reset
- Includes input for reset from Watchdog Timer
- Option to split the read and write OPB data busses for optimal FPGA routing and resource utilization.

The OPB_V20 includes an OPB Arbiter that incorporates the features contained in the IBM On-chip Peripheral Bus Arbiter Core manual (version 1.5) for 32-bit implementation. This manual is referenced throughout this document and is considered the authoritative specification. Any differences between the IBM OPB Arbiter implementation and the Xilinx OPB Arbiter implementation are explained in the Specification Exceptions section of this [data sheet](#).

The Xilinx OPB Arbiter design allows you to tailor the OPB Arbiter to suit your application by setting certain parameters to enable/disable features.

In some cases, setting these parameters may cause the Xilinx OPB Arbiter design to deviate slightly from the IBM OPB Arbiter specification. These parameters are described in the OPB_V20 Design Parameters section of this [data sheet](#).

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	opb_v20	v1.10a
Resources Used		
	Min	Max
I/O	46	436
LUTs	80	666
FFs	5	145
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

Click [here](#) to view this data sheet

Product Overview

Introduction

The OPB_V20 module is used as the OPB interconnect for Xilinx FPGA based embedded processor systems. The bus interconnect in the OPB v2.0 specification is a distributed multiplexer implemented as an “and” function in the master or slave driving the bus and an OR to combine the drivers into a single bus.

The OPB_V20 module assumes the AND, or enable function is within the master or slave and provides the OR function to combine the various bus signals.

Features

- Includes parameterized OPB Arbiter
- Includes parameterized I/O signals to support and number of masters or slaves
- Includes all signals present in the OPB v2.0 Specification except the DMA handshake signals
- The OR structure can be implemented using only LUTs or a combination of LUTs and fast carry to reduce the number of LUTs in the OR interconnect
- Includes a 16-clock Power-on OPB Bus Reset and parameter for high or low external bus reset
- Includes input for reset from Watchdog Timer
- Option to split the read and write OPB data busses for optimal FPGA routing and resource utilization.

The OPB_V20 includes an OPB Arbiter that incorporates the features contained in the IBM On-chip Peripheral Bus Arbiter Core manual (version 1.5) for 32-bit implementation. This manual is referenced throughout this document and is considered the authoritative specification. Any differences between the IBM OPB Arbiter implementation and the Xilinx OPB Arbiter implementation are explained in the Specification Exceptions section of this [data sheet](#).

The Xilinx OPB Arbiter design allows you to tailor the OPB Arbiter to suit your application by setting certain parameters to enable/disable features.

In some cases, setting these parameters may cause the Xilinx OPB Arbiter design to deviate slightly from the IBM OPB Arbiter specification. These parameters are described in the OPB_V20 Design Parameters section of this [data sheet](#).

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	opb_v20	v1.10b
Resources Used		
	Min	Max
Slices	46	436
LUTs	81	668
FFs	5	145
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

Click [here](#) to view this data sheet

Product Overview

Introduction

The On-Chip Peripheral Bus (OPB) to Processor Local Bus (PLB) Bridge module translates OPB transactions into PLB transactions. It functions as a slave on the OPB side and a master on the PLB side. Access to the control register and bus error status registers is user selectable from either the OPB or an optional DCR interface. The OPB to PLB Bridge is necessary in systems where an OPB master device, such as a DMA engine or an OPB based coprocessor, requires access to PLB devices (i.e. high speed memory devices).

The Xilinx OPB to PLB Bridge design allows customers to tailor the bridge to suit their application by setting certain parameters to enable / disable features. The parameterizable features of the design are discussed in this [data sheet](#). Differences between the IBM OPB to PLB Bridge implementation and the Xilinx OPB to PLB Bridge implementation are also highlighted and explained.

The OPB to PLB Bridge, when convenient, is referred to as the Bridge In (BGI), and the PLB to OPB Bridge is referred to as the Bridge Out (BGO). This terminology reflects a PLB centric convention of data flowing “in from” and “out to” the peripheral bus, respectively. This is only a naming convention and in no way restricts the use of these bridges in alternative processor / bus configurations.

Features

- 64-bit PLB Master interface
 - Communicates with 32- or 64-bit PLB slaves
 - Non-burst transfers of 1 to 8 bytes
- 32-bit OPB Slave interface with byte enable transfers

Note Does not support dynamic bus sizing or non-byte enable transactions

 - Decodes up to 4 separate address ranges
 - PLB and OPB clocks can have a 1:1, 2:1, 3:1, or 4:1 synchronous relationship (OPB clock frequency must be less than or equal to the PLB clock frequency)
 - Asserts BGI_opbRetry if bridge is busy with a PLB transaction and a new OPB request is received.
- Bus Error Status Register (BESR) and Bus Error Address Register (BEAR) provide bus error status, and Bridge Control Register (BCR) provides bridge control functions

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro, Virtex II, Virtex, Virtex E, Spartan II	
Version of Core	opb2plb_bridge	v1.00a
Resources Used		
	Min	Max
I/O	373	373
LUTs	456	533
FFs	664	669
Block RAMs	2	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim 5.6e or later	
Synthesis	XST & Synplify (state machines)	
Support		
Support provided by Xilinx, Inc.		

- Parameterizable selection between DCR or OPB Slave interface, which provides access to BESR, BEAR, and BCR
- Edge-type interrupt generated when a bus error is detected

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The On-Chip Peripheral Bus (OPB) to Processor Local Bus (PLB) Bridge module translates OPB transactions into PLB transactions. It functions as a slave on the OPB side and a master on the PLB side. Access to the control register and bus error status registers is user selectable from either the OPB or an optional DCR interface. The OPB to PLB Bridge is necessary in systems where an OPB master device, such as a DMA engine or an OPB based coprocessor, that requires access to PLB devices (i.e. high speed memory devices, etc.).

The Xilinx OPB to PLB Bridge design allows customers to tailor the bridge to suit their application by setting certain parameters to enable / disable features. The parameterizable features of the design are discussed in this [data sheet](#). Differences between the IBM OPB to PLB Bridge implementation and the Xilinx OPB to PLB Bridge implementation are also highlighted and explained.

In subsequent sections, the OPB to PLB Bridge, when convenient, is referred to as the Bridge In (BGI), and the PLB to OPB Bridge is referred to as the Bridge Out (BGO). This terminology reflects a PLB centric convention of data flowing "in from" and "out to" the peripheral bus, respectively. However, this is only a naming convention and in no way restricts the use of these bridges in alternative processor / bus configurations.

Features

The Xilinx OPB to PLB Bridge is a soft IP core with the following features:

- 64-bit PLB Master interface
 - Communicates with 32- or 64-bit PLB slaves
 - Non-burst transfers of 1 to 8 bytes
 - Burst transfers, including word and double-word bursts of fixed lengths, up to 16 words of data
 - Cacheline transactions of 4, 8, and 16 words
- Translates OPB sequential accesses (bursts) to either cacheline or fixed length PLB burst transfers
 - Performing only single beat and cacheline transactions reduces system logic utilization and improves timing through PLB slave IP simplification
 - Target word first order supported when cacheline transactions selected

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, VirtexE, Virtex, Spartan™ II	
Version of Core	opb2plb_bridge	v1.00b
Resources Used		
	Min	Max
I/O	390	390
LUTs	821	875
FFs	887	949
Block RAMs	2	2
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST & Synplify (state machines)	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

- PLB burst transfers yield better bus cycle efficiency but may increase logic utilization and degrade timing in the system
- 32-bit OPB Slave interface with byte enable transfer
 - Note:** Does not support dynamic bus sizing or non-byte enable transactions
 - Decodes up to four separate address ranges
 - PLB and OPB clocks can have a 1:1, 2:1, 3:1, or 4:1 synchronous relationship (OPB clock frequency must be less than or equal to the PLB clock frequency)
 - Asserts BGI_opbRetry if bridge is busy with a PLB transaction and a new OPB request is received.
- Bus Error Status Register (BESR) and Bus Error Address Register (BEAR) provide bus error status, and Bridge Control Register (BCR) provides bridge control functions
 - Parameterizable selection between DCR or OPB Slave interface, which provides access to BESR, BEAR, and BCR
- Posted write buffer and read prefetch buffer 16 words deep
- Edge-type interrupt generated when bus error detected

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the design specification for the OPB to OPB Lite Bridge. The OPB to OPB Lite Bridge is used to connect two OPB buses. The bridge has one master port and one slave port. Two bridges may be used together to support full bus mastership in both directions.

Features

- Provides a bridge between two OPB V2.0 buses
- Connections for one master-side bus and one slave-side bus
- Parameterized data bus widths
- Simple transaction forwarding reduces LUT count
- Requires the two OPB buses to be on the same clock and the same size
- No support for data buffering or posted writes.

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II	
Version of Core	opb_opb_lite	v1.00a
Resources Used		
	Min	Max
Slices	21	26
LUTs	22	30
FFs	27	27
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The OPB to DCR Bridge translates transactions received on its OPB slave interface into DCR master operations. Its design utilizes an Intellectual Property InterFace (IPIF) module to abstract OPB transactions into a simple SRAM style protocol that is easier to design with.

The main advantage of using the bridge instead of the CPU to control the DCR bus is that it provides a memory mapped interface that may be preferable to the use of special move to/move from DCR instructions.

Since the bridge typically runs at a slower clock frequency than the CPU, its timing requirements are also less stringent. The OPB to DCR Bridge implements a simple and flexible method for communicating with DCR devices.

Features

- 32-bit DCR master with a 10-bit DCR address bus
- Memory-mapped interface from OPB to DCR, no special instructions required
- Increased timing flexibility in typical systems where the OPB clock is slower than the CPU clock
- Allows master devices other than the CPU to access the DCR bus
- Provides a mechanism where CoreConnect systems without a CPU can support DCR devices

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II	
Version of Core	opb2dcr_bridge	v1.01a
Resources Used		
	Min	Max
Slices	87	89
LUTs	42	44
FFs	131	131
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The Xilinx 64-bit Processor Local Bus (PLB) consists of a bus control unit, a watchdog timer, and separate address, write, and read data path units with a three-cycle only arbitration feature. It contains a DCR slave interface to provide access to its bus error status registers. It also contains a power-up reset circuit to ensure a PLB reset is generated if no external reset has been provided.

The IBM Processor Local Bus (PLB) 64-Bit Architecture Specification and the IBM Processor Local Bus (PLB) 64-Bit Arbiter Core User's Manual are referenced throughout this document. Differences between the IBM PLB Arbiter and the Xilinx PLB are highlighted and explained in Specification Exceptions.

Features

- PLB arbitration support for up to 16 masters
 - Number of PLB masters is configurable via a design parameter
- PLB address and data steering support for up to 16 masters
- 64-bit and/or 32-bit support for masters and slaves
- PLB address pipelining
- Three-cycle arbitration
- Four levels of dynamic master request priority
- PLB watchdog timer
- PLB architecture compliant
- Complete PLB Bus structure provided
 - Up to 16 slaves supported
 - Number of PLB slaves configurable via a design parameter
 - No external or gates required for PLB slave input signals
- PLB Reset circuit
 - PLB Reset generated synchronously to the PLB clock upon power up if no external reset is provided
 - PLB Reset generated synchronously from external reset when external reset provided
 - Active state of external reset selectable via a design parameter

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	plb_v34	v1.01a
Resources Used		
	Min	Max
Slice	194	1616
LUTs	263	2533
FFs	57	482
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The Processor Local Bus (PLB) to On-chip Peripheral Bus (OPB) Bridge translates PLB transactions into OPB transactions. It functions as a slave on the PLB side and a master on the OPB side. It contains a DCR slave interface to provide access to its bus error status registers. The PLB to OPB bridge is necessary in systems where a PLB master device requires access to OPB peripherals.

The Xilinx PLB to OPB Bridge design allows customers to tailor the bridge to suit their application by setting certain parameters to enable/disable features. The parameterizable features of the design are discussed in this [data sheet](#). Differences between the IBM PLB to OPB Bridge implementation and the Xilinx PLB to OPB Bridge implementation are also highlighted and explained.

Features

- PLB Slave interface
 - 32-bit or 64-bit PLB (configurable via the C_PLB_DWIDTH design parameter)
 - PLB slave width same as PLB bus width
 - Decodes up to four separate address ranges
 - Programmable lower and upper address boundaries for each range
 - Communicates with 32- or 64-bit PLB masters
 - Non-burst transfers of 1-8 bytes
 - Burst transfers, including word and double-word bursts of fixed or variable lengths, up to depth of burst buffer (16)
 - Limited support for byte, half-word, quad-word and octal-word bursts to maintain PLB compliance
 - Cacheline transactions of 4, 8, and 16 words
 - Support for burst transactions can be eliminated via a design parameter
 - save device resources by only supporting single beat, 4, 8, or 16 word line transfers
 - Supports up to 16 PLB masters (number of PLB masters configurable via a design parameter)
- OPB Master interface with byte enable transfers

Note: Does not support dynamic bus sizing without additional glue logic

 - Data width configurable via a design parameter

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	plb2opb_bridge	v1.00a
Resources Used		
	Min	Max
Slices	515	835
LUTs	531	826
FFs	384	630
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

- PLB and OPB clocks can have a 1:1, 1:2, 1:3, 1:4 synchronous relationship
- Bus Error Address Registers (BEAR) and Bus Error Status Registers (BESR) to report errors
 - DCR Slave interface provides access to BEAR/BESR
 - BEAR, BESR, and DCR interface can be removed from the design via a design parameter
- 16-deep posted write buffer and read pre-fetch buffer
- Edge-type interrupt generated when bus error detected

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The Processor Local Bus (PLB) to On-chip Peripheral Bus (OPB) Bridge translates PLB transactions into OPB transactions. It functions as a slave on the PLB side and a master on the OPB side. It contains a DCR slave interface to provide access to its bus error status registers. The PLB to OPB bridge is necessary in systems where a PLB master device requires access to OPB peripherals.

The Xilinx PLB to OPB Bridge design allows customers to tailor the bridge to suit their application by setting certain parameters to enable/disable features. The parameterizable features of the design are discussed in this [data sheet](#). Differences between the IBM PLB to OPB Bridge implementation and the Xilinx PLB to OPB Bridge implementation are also highlighted and explained.

Features

- PLB Slave interface
 - 32-bit or 64-bit PLB (configurable via the C_PLB_DWIDTH design parameter)
 - PLB slave width same as PLB bus width
 - Decodes up to four separate address ranges
 - Programmable lower and upper address boundaries for each range
 - Communicates with 32- or 64-bit PLB masters
 - Non-burst transfers of 1-8 bytes
 - Burst transfers, including word and double-word bursts of fixed or variable lengths, up to depth of burst buffer (16)
 - Limited support for byte, half-word, quad-word and octal-word bursts to maintain PLB compliance
 - Cacheline transactions of 4, 8, and 16 words
 - Support for burst transactions can be eliminated via a design parameter
 - save device resources by only supporting single beat, 4, 8, or 16 word line transfers
 - Supports up to 16 PLB masters (number of PLB masters configurable via a design parameter)
- OPB Master interface with byte enable transfers

Note: Does not support dynamic bus sizing without additional glue logic

 - Data width configurable via a design parameter

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	plb2opb_bridge	v1.00b
Resources Used		
	Min	Max
Slices	412	662
LUTs	499	787
FFs	355	620
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

- PLB and OPB clocks can have a 1:1, 1:2, 1:3, 1:4 synchronous relationship
- Bus Error Address Registers (BEAR) and Bus Error Status Registers (BESR) to report errors
 - DCR Slave interface provides access to BEAR/BESR
 - BEAR, BESR, and DCR interface can be removed from the design via a design parameter
- 16-deep posted write buffer and read pre-fetch buffer
- Edge-type interrupt generated when bus error detected

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The Xilinx 32-Bit Device Control Register Bus (DCR) provides the DCR bus structure as described in the IBM 32-Bit Device Control Register Bus (DCR) Architecture Specification to allow easy connection of the DCR Master to the DCR slaves. It provides the daisy-chain for the DCR data bus and the OR gate for the DCR acknowledge signals from the DCR slaves.

Features

The Xilinx DCR is a soft IP core designed for Xilinx FPGAs. These are the features:

- DCR connections for one DCR master and a variable number of DCR slaves, which are configurable via design parameter
- Daisy-chain connections for the DCR data bus
- Required OR function of the DCR slaves' acknowledge signal

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	dcr_v29	v1.00a
Resources Used		
	Min	Max
Slices	0	4
LUTs	0	5
FFs	0	0
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This specification defines the architecture and interface requirements for this module.

The Xilinx Processor System Reset Module design allows the customer to tailor the design to suit their application by setting certain parameters to enable/disable features. The parameterizable features of the design are discussed in this [data sheet](#).

Features

- Asynchronous external reset input is synchronized with clock
- Asynchronous auxiliary external reset input is synchronized with clock
- Both the external and auxiliary reset inputs are selectable active high or active low
- Selectable minimum pulse width for reset inputs to be recognized
- Selectable load equalizing
- DCM Locked input
- Power On Reset generation
- Sequencing of reset signals coming out of reset:
 - First - bus structures come out of reset
 - PLB and OPB Arbiter and bridges for example
 - Second - Peripheral(s) come out of reset 16 clocks later
 - UART, SPI, IIC for example
 - Third - the CPU(s) come out of reset 16 clocks after the peripherals

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex E, Virtex, Spartan™ II, Spartan IIE	
Version of Core	proc_sys_reset	v1.00a
Resources Used		
	Min	Max
I/O	1	2
LUTs	37	57
FFs	52	82
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	Alliance Tool Suite	
Verification	N/A	
Simulation	N/A	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The LMB_V10 module is used as the LMB interconnect for Xilinx FPGA based embedded processor systems. The LMB is a fast, local bus for connecting MicroBlaze instruction and data ports to high-speed peripherals, primarily on-chip block RAM (BRAM).

Features

- Efficient, single master bus (requires no arbiter)
- Separate read and write data buses
- Low FPGA resource utilization
- 125 MHz operation

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	lmb_v10	v1.00a
Resources Used		
	Min	Max
I/O		
LUTs		
FFs		
Block RAMs		
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The On-Chip Peripheral Bus (OPB) Arbiter design described in this document incorporates the features contained in the IBM On-chip Peripheral Bus Arbiter Core manual (version 1.5) for 32-bit implementation, which is referenced throughout this document and is considered the authoritative specification. Any differences between the IBM OPB Arbiter implementation and the Xilinx OPB Arbiter implementation are explained in the Specification Exceptions section of this [data sheet](#).

The Xilinx OPB Arbiter design allows you to tailor the OPB Arbiter to suit your application by setting certain parameters to enable/disable features. In some cases, setting these parameters may cause the Xilinx OPB Arbiter design to deviate slightly from the IBM OPB Arbiter specification. These parameters are described in the OPB Arbiter Design Parameters section.

Features

The OPB Arbiter is a soft IP core designed for Xilinx FPGAs and contains the following features:

- Optional OPB slave interface (included in design via a design parameter)
- OPB Arbitration
 - arbitrates between 1–16 OPB Masters (the number of masters is parameterizable)
 - arbitration priorities among masters programmable via register write
 - priority arbitration mode configurable via a design parameter
 - Fixed priority arbitration with processor access to read/write Priority Registers
 - Dynamic priority arbitration implementing a true least recent used (LRU) algorithm
- Two bus parking modes selectable via Control Register write:
 - park on selected OPB master (specified in Control Register)
 - park on last OPB master granted OPB access
- Watchdog timer asserts the OPB time-out signal if a slave response is not detected within 16 clock cycles

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex™-II, Virtex-II Pro™	
Version of Core	opb_arbiter	v1.02c
Resources Used		
	Min	Max
I/O	4	904
LUTs	6	252
FFs	4	1477
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

- Registered or combinational Grant outputs configurable via a design parameter

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the design specification for the Fast Simplex Link Channel v1.0. The FSL_V20 module is used as the FSL interconnect for Xilinx FPGA based embedded processor systems. The FSL is a fast uni-directional point-to-point communication channel.

Features

- Uni-directional point-to-point communication
- Unshared non-arbitrated communication mechanism
- Control and Data communication support
- FIFO based communication
- Configurable depth FIFO
- Configurable data path size
- 600 MHz standalone operation
- The FSL bus is driven by one master and drives one slave.

Core Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Spartan™ II, Spartan II E	
Version of Core	fsl_v20	v1.00b
Resources Used		
	Min	Max
Slices	26	26
LUTs	43	43
FFs	6	6
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	N/A	
Design Tool Requirements		
Xilinx Implementation Tools	ISE 5.2i or higher	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or higher	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

IPIF

This section of the User Guide contains information on the following:

[OPB IPIF Architecture](#)

[OPB IPIF Slave Attachment](#)

[OPB IPIF Master Attachment](#)

[OPB IPIF Address Decode](#)

[OPB IPIF Interrupt](#)

[OPB IP Interface Packet FIFO](#)

[Direct Memory Access and Scatter Gather](#)

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the architecture for the OPB IPIF, a module that facilitates the connection of Xilinx or customer IP modules to the IBM On-Chip Peripheral Bus (OPB). The OPB is part of IBM's CoreConnect™ family of data buses and associated infrastructure. CoreConnect is intended for use in system-on-a-chip environments, including Xilinx Virtex™-II Pro FPGAs with embedded PowerPC hard processors and FPGAs using the MicroBlaze soft processor.

An Intellectual Property solution, referred to herein as an *IP*, is a function targeted for implementation in a Xilinx FPGA. Figure 1 of this [data sheet](#) shows the OPB IPIF positioned between the OPB and the IP. The interface seen by the IP is called the *IP Interconnect* (IPIC for short). The combination of the IPIF and the IP is called a *device* (or in some circles, a *peripheral*).

In addition to facilitating OPB attachment, the IPIF provides additional optional services. These services, FIFOs, DMA, Scatter Gather (automated DMA), software reset, interrupt support and OPB bus-master access, are placed in the IPIF to standardize functionality that is common to many IPs and to reduce IP development effort. Figure 1 is a device which uses all IPIF protocols and services.

In most of the IPIF modules, the Read and Write FIFOs, DMA/SG, Interrupt Control and the Reset block, attach to the IPIC inside the IPIF and utilize the register and/or SRAM interfaces and take advantage of the centralized address decoding.

These modules are essentially on the same footing as the IP in terms of how they are interfaced to the OPB. The "glue" in the figure represents a small number of non-IPIC connections between modules.

At the other end of the spectrum, Figure 2 shows a device using a near minimal set of IPIF features. In this case, the IPIF does nothing more than provide OPB access to some IP registers. Between these extremes are various other possibilities for devices that use other combinations of IPIF features.

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	N/A	
Version of Core	opb_ipif_arch	v1.23e
Resources Used		
	Min	Max
I/O	N/A	N/A
LUTs	N/A	N/A
FFs	N/A	N/A
Block RAMs	N/A	N/A
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	N/A	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	N/A	
Design Tool Requirements		
Xilinx Implementation Tools	N/A	
Verification	N/A	
Simulation	N/A	
Synthesis	N/A	
Support		
Support provided by Xilinx, Inc.		

Features

The IPIF is a parametric soft IP core designed for Xilinx FPGAs and contains the capabilities and features summarized below.

- Synchronous operation
- Hardware and optional software reset
- Freeze signal (requests graceful stop of IP and IPIF to facilitate debug under serious system failure; keeps interrupts from “piling up”)
- Slave Interface:
 - FPGA friendly protocol meeting requirements of most new and legacy IP
 - Separate Address, Data-In, and Data-Out Buses
 - Transaction Qualification: Read Req, Write Req, Byte Enable, Burst
 - Transaction Response: Read Ack, Write Ack, Error, Retry, Timeout Suppression
 - Register Interface (optional)
 - Per register address decodes
 - Separate read and write enables for decoded register addresses
 - Doesn't need the address
 - SRAM Interface—for IP with SRAM-like interface (optional)
 - Block-address decode
 - Single enable (read or write) for the decoded block address
 - Uses the address
 - Support for burst transactions (optional)
- Interrupt Support (optional):
 - Parameterizeable: Select the required features
 - Captures up to 32 interrupt events from the IP
 - Device Interrupt Source Controller (ISC) in the IPIF is at top of a hierarchy of ISCs
 - Optional
 - Hierarchical structure can be eliminated by user parameter when all interrupt events are from the IP, leaving only the device-global interrupt enable function of the device ISC
 - IP ISC
 - IP may pass in one or more interrupt events which are latched, enabled, and cleared in the IP ISC, which passes its interrupt-active condition to the device ISC
 - IP Interrupt condition option
 - When the IPIF is configured without interrupt support, the IP may latch, enable and clear interrupts itself and pass the interrupt condition directly to the system interrupt controller
 - IPIF ISCs, as needed, feed additional interrupt-active conditions to the device ISC
 - E.g., one ISC for each DMA channel
- Master Interface (optional):
 - Parameterizeable: Select the required features
 - Bus Address
 - Local Address (allows master general access to local resources)
 - In lieu of dedicated in and out master data paths, the local address allows the slave-mode data paths and address decoding to be leveraged for master operations
 - Advantageous for devices with addresses that are accessed by remote OPB masters *and* by a local master
 - Cooperation between the Slave Attachment and the Master Attachment used to complete master operations
 - Single and burst transactions
 - Transaction Qualification: Read Req, Write Req, Byte Enable, Burst, Bus Lock
 - Transaction Response: separate Read and Write Acks, Transaction ack, Error, Retry, Timeout
- Write Packet FIFO (optional)
 - Parameterizeable: Select the required features
 - BRAM based

- Packet support: Data reads by IP can be provisional until explicitly committed:
 - Mark command sets a reference point for uncommitted reads
 - Restore command discards uncommitted reads; data is reread starting at the mark
 - Release command commits uncommitted reads
 - Useful, for example, in protocols that require retransmission
 - Interrupt when uncommitted reads empty the entire contents of the FIFO (indicates that the FIFO is too small for the application)
- Written from the OPB via IPIF-internal connection to the IPIC
- Status can be read from the OPB
- Read by IP via request/acknowledge protocol
- Dedicated Data Path to the IP
- IP Status flags: Empty, Almost Empty (one occupied), Occupancy Count
- Optional Module Identification Register
- Read FIFO (optional):
 - Parameterizeable: Select the required features
 - BRAM based
 - Packet support: Data writes by IP can be provisional until explicitly committed:
 - Mark command sets a reference point for uncommitted writes
 - Restore command discards uncommitted writes; data is re-written starting at the mark
 - Release command commits uncommitted writes
 - Useful, for example, in protocols that discard data on error conditions or address misses
 - Interrupt when uncommitted writes fill the entire contents of the FIFO (indicates that the FIFO is too small for the application)
 - Read from the OPB via IPIF-internal connection to the IPIC
 - Status can be read from the OPB
 - Written by IP via request/acknowledge protocol
 - Dedicated Data Path from the IP
 - IP Status flags: Full, Almost Full (one vacant), Vacancy Count
 - Mark command sets a reference point for uncommitted writes
 - Restore command discards uncommitted writes; data is re-written starting at the mark
 - Release command commits uncommitted writes
 - Useful, for example, in protocols that discard data on error conditions or address misses
 - Interrupt when uncommitted writes fill the entire contents of the FIFO (indicates that the FIFO is too small for the application)
 - Optional Module Identification Register
- DMA/Scatter Gather (optional):
 - Parameterizeable: Select the required features
 - Up to Two DMA channels may be included
 - Optional scatter gather capability for channels
 - Optional packet capability for SG channels
 - Optional interrupt coalescing for packet SG channels (number of packets per interrupt is software selectable; time wait bound gives packets guaranteed timely visibility)
 - Optional Module Identification Register
 - Attaches to IPIC internal to the IPIF
 - Uses an IPIF-internal master interface

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the specification for the OPB IPIF Slave Attachment. The OPB IPIF Slave Attachment is a sub-module of the OPB IPIF. The OPB IPIF is a module for attaching an Intellectual Property, or IP Core, to the IBM-defined On-Chip Peripheral Bus.

The environment for the Slave Attachment is shown in Figure 1. The IP Core and service blocks inside the IPIF use a set of signals and protocols called the IP Interconnect (or IPIC). The Slave Attachment serves a bridging function between the OPB and the bus-transaction part IPIC.

The Slave Attachment responds to OPB transactions when the device is addressed as a slave. The Slave Attachment translates the OPB transaction into a corresponding IPIC transaction and then responds to the OPB master.

Figure 1 shows additional detail of the Slave and Master Attachments, including the signal groups attached to each. The essential role of the Slave Attachment as a converter from the OPB bus protocol to the IPIC protocol is apparent. (See Figure 7 for additional detail.)

A role in execution of locally initiated master transactions is also indicated by the "cooperation" signals between the Master Attachment and the Slave Attachment. The motivation for this role is that a master/slave device commonly has some registers that are accessed in both slave and master modes.

Since a data path must exist from the registers to the OPB for slave access, this data path (and other infrastructure) can be reused for master operation. The IPIF-resident DMA and Scatter Gather services operate as masters and take ample advantage of this reuse of infrastructure.

Features

- Protocol translation from OPB transactions to IPIC transactions.
- Optional support for slave-mode OPB sequential address (burst) transactions.
- Optional cooperation with the Master Attachment for master operation.

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	N/A	
Version of Core	opb_ipif_slave_attachment	v1.23e
Resources Used		
	Min	Max
I/O	N/A	N/A
LUTs	N/A	N/A
FFs	N/A	N/A
Block RAMs	N/A	N/A
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	N/A	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	N/A	
Design Tool Requirements		
Xilinx Implementation Tools	N/A	
Verification	N/A	
Simulation	N/A	
Synthesis	N/A	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the specification for the OPB IPIF Master Attachment, which is a sub-module of the OPB IPIF.

Figure 1 of this [data sheet](#) shows the Master Attachment in context and Figure 2 shows more detail of the Master Attachment and its closely affiliated module, the Slave Attachment. Inclusion of the Master Attachment in an IPIF system is optional, needed only if the device operates as an OPB master.

There are two points from which master operations may be initiated within a device. One is the attached IP core. The other is the DMA engine inside the IPIF. Either of these masters may be present or absent. If both are absent, the Master Attachment is not included in the device's IPIF.

The Master Attachment handles the address, transaction qualifier and response signals but relies upon the cooperation of the Slave Attachment to move data. For systems that have addresses that are sources and sinks for both slave and master transactions, this results in reuse of the slave-mode infrastructure and a resource savings.

The IP Core and service blocks inside the IPIF use a set of signals and protocols called the IP Interconnect (or IPIC). The Master Attachment translates the IP Interconnect master transaction into a corresponding OPB master transaction. As OPB transfers complete, the Master Attachment generates the IPIC response signals.

Features

- Protocol translation from IPIC master transactions to OPB master transactions.
- Reuses existing slave-mode data paths and other infrastructure through cooperation with the Slave Attachment.
- Single and burst master transactions supported; singles are a limiting case of bursts.
- Optimized retry mode.

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	N/A	
Version of Core	opb_ipif_master_attachment	v1.23a
Resources Used		
	Min	Max
I/O	N/A	N/A
LUTs	N/A	N/A
FFs	N/A	N/A
Block RAMs	N/A	N/A
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	N/A	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	N/A	
Design Tool Requirements		
Xilinx Implementation Tools	N/A	
Verification	N/A	
Simulation	N/A	
Synthesis	N/A	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

Address decoding is a service performed by the IPIF. The service supplies to the IP core a set of individual register decodes, simplifying register interfacing in the core, and/or an address-block decode, simplifying interfacing to an SRAM-like device. The service is also utilized by IPIF-resident blocks such as the Read and Write FIFOs, the DMA[SG] controller, the Interrupt Source Controller, and the Reset Controller.

Features

- Supports the following superset of optional address ranges:
 - IP core register block
 - IP core address block (SRAM)
 - Write FIFO
 - Register block for CSRs
 - Address block for data
 - Read FIFO
 - Register block for CSRs
 - Address block for data
 - DMA[SG] address block
 - Device Interrupt Source Controller register block
 - Reset/MIR register block
- Address ranges not needed in a system as not implemented
- Device-select signal allows the part of the address common to all address ranges to be decoded by external logic

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	N/A	
Version of Core	opb_ipif_address_decode	v1.23e
Resources Used		
	Min	Max
I/O	N/A	N/A
LUTs	N/A	N/A
FFs	N/A	N/A
Block RAMs	N/A	N/A
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	N/A	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	N/A	
Design Tool Requirements		
Xilinx Implementation Tools	N/A	
Verification	N/A	
Simulation	N/A	
Synthesis	N/A	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The OPB IPIF function has a requirement to collect all internal interrupts within a peripheral device and coalesce those interrupts through masking and 'ORing' into a single interrupt output that is sent to the microprocessor system Interrupt Controller (ITNC).

The IPIF Interrupt module, therefore, provides interrupt support logic for a user IP (connected to the IPIF) and internal IPIF interrupt source functions (DMA/SG, PFIFOs, etc.). The interrupt hierarchy of an OPB device is shown in Figure 1 of this [data sheet](#).

The IPIF Interrupt module incorporates two main functions: the Device Interrupt Source Controller (ISC) and the IP ISC. Each ISC function collects multiple interrupt inputs and outputs a single interrupt.

Features

The IPIF Interrupt module is incorporated in the standard OPB IPIF block designed for Xilinx FPGAs and contains these features:

- IP Interrupt Source Controller Function
 - Parameterized number of interrupts needed by IP
 - Provides both Interrupt Status Register (ISR) and Interrupt Enable Register (IER) functions for the user IP connecting to the IPIF
 - Registers are OPB accessible via the IPIF Local Bus Interface
- Device Interrupt Source Controller Function
 - Device ISC omission through input parameter programming
 - Parameterized number of local IPIF generated interrupt sources
 - Provides both Interrupt Status Register (ISR) and Interrupt Enable Register (IER) functions for the Device level interrupts
 - Registers are OPB accessible via the IPIF Local Bus Interface.
 - Selectable Priority Encoder function on asserted and enabled interrupts
 - Global Enable/Disable for final interrupt output to the System Interrupt Controller.

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	N/A	
Version of Core	opb_ipif_interrupt	v1.00b
Resources Used		
	Min	Max
I/O	N/A	N/A
LUTs	N/A	N/A
FFs	N/A	N/A
Block RAMs	N/A	N/A
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	N/A	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	N/A	
Design Tool Requirements		
Xilinx Implementation Tools	N/A	
Verification	N/A	
Simulation	N/A	
Synthesis	N/A	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This specification defines the First In First Out memory design incorporating packet data support (PFIFO). The PFIFO resides within the Xilinx Intellectual Property Interface (IPIF) module which interfaces to a host bus structure such as the IBM OPB. The FIFOs primarily utilize Virtex BRAM elements as the basic data storage medium.

FIFO depths of 16 words or less may optionally utilize SRL16 elements as the memory medium. The memory is coupled with surrounding counters and logic that are necessary for the functional requirements.

These requirements include interfaces to the IP, interfaces to the Direct Memory Access/ Scatter Gather (DMA/SG) interface, and to the IPIF Local Bus protocol.

The PFIFO is used as a data buffering agent between the IP function and a Bus Master such as a DMA/SG Engine (see Figure 1). The basic operation of the module is a FIFO buffer. Data can be shuttled in and out of the module without the need for the accessing agent to provide successive memory addresses. Data is stored and read in a First In First Out (FIFO) sequence. However, the PFIFO to IP interface has been enhanced with additional packet management controls that facilitate packet retransmission or receive packet discard functionality necessary for efficient operation of some IP protocols (such as Ethernet).

The IPIF accesses the PFIFO module as a simple memory mapped "FIFO" interface and memory mapped registers.

A PFIFO module is instantiated within the IPIF framework and consists of two different types: a Write PFIFO (WrPFIFO) and a Read PFIFO (RdPFIFO). The WrPFIFO is the intermediate storage medium for data from the Host Bus to the IP.

The RdPFIFO is used to buffer data from the IP that needs to be sent to the Host Bus. Both Packet FIFO designs utilize a single synchronous clock domain for the input and output sides of the modules. This simplifies the design and reduces LUT count as compared to an asynchronous dual clock implementation.

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	N/A	
Version of Core	opb_pfifo	v1.23e
Resources Used		
	Min	Max
I/O	N/A	N/A
LUTs	N/A	N/A
FFs	N/A	N/A
Block RAMs	N/A	N/A
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	N/A	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	N/A	
Design Tool Requirements		
Xilinx Implementation Tools	N/A	
Verification	N/A	
Simulation	N/A	
Synthesis	N/A	
Support		
Support provided by Xilinx, Inc.		

Features

- Two independent functions are provided: the Read PFIFO (for host bus receive data buffering) and the Write PFIFO (for host bus transmit data buffering).
- The PFIFO design adheres to IPSPEC035 OPB IPIF Architectural Specification.
- User controlled features that include parameters for:
 - Setting FIFO data width
 - Setting FIFO data depth (words)
 - Inclusion/Omission of Packet mode support
 - Setting IPIF Data Bus width
 - Inclusion/Omission of Vacancy calculation for write port
 - Selecting Target FPGA family type
 - Inclusion/Omission of Module Identification Register
- "FIFO like" status outputs of AlmostFull, Full, AlmostEmpty, and Empty in addition to true 'Occupancy' and 'Vacancy' outputs. Host bus interface provides applicable status as a read accessible Status Register
- Write and Read Ports synchronized to a common clock source (synchronous operation)
- IPIF Local Bus Read access to Occupancy count on the Read PFIFO and the Vacancy count of the Write FIFO

The PFIFO modules can be reset from either an external reset input signal or a software initiated write to the Reset Register port.

Click [here](#) to view this data sheet

Product Overview

Introduction

Many soft IP input/output peripheral devices in Xilinx products with embedded or attached processors require the automation facilities of Direct Memory Access and Scatter Gather. This is a specification for such facilities.

DMA[SG] Controller Overview

Definitions

Direct memory access (DMA) allows for a bounded number of sequential data transfers to take place between regions in the address space (typically between memory and an I/O device) without processor management of individual transfers. The processor sets up the *DMA operation* by specifying the number accesses and the source and destination addresses.

Scatter gather (SG) allows a *sequence* of DMA operations to be pre-specified by software and performed automatically without further processor intervention. The processor prepares the DMA operations in a system of buffers and their associated *Buffer Descriptors*. The SG automation hardware processes the Buffer Descriptors and performs the DMA operations specified therein through activation of the DMA hardware.

Often it is useful to consider that one or more DMA operations combine to compose a higher-level unit of data. An example of such a unit is a packet or frame¹ of data in a communications protocol. This specification addresses issues associated with the handling of packets and allows packets to be distributed across one or more buffers.

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	N/A	
Version of Core	opb_ipif_dma_sg	v1.23e
Resources Used		
	Min	Max
I/O	N/A	N/A
LUTs	N/A	N/A
FFs	N/A	N/A
Block RAMs	N/A	N/A
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	N/A	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	N/A	
Design Tool Requirements		
Xilinx Implementation Tools	N/A	
Verification	N/A	
Simulation	N/A	
Synthesis	N/A	
Support		
Support provided by Xilinx, Inc.		

Memory Interface Cores

This section of the reference guide contains information on the following memory interface cores:

- LMB Block RAM (BRAM) Interface Controller
- Dual LMB Block RAM (BRAM) Interface Controller
- OPB External Memory Controller (EMC) (v1.00d)
- OPB External Memory Controller (EMC) (v1.10a)
- OPB Synchronous DRAM (SDRAM) Controller**
- OPB Block RAM (BRAM) Interface Controller
- OPB Block RAM Interface Controller (OPB_BRAM_IF_CNTL)
- OPB Double Data Rate (DDR) Synchronous DRAM (SDRAM) Controller**
- OPB SYSACE (System ACE) Interface Controller
- PLB External Memory Controller (EMC) Design Specification (v1.00d)
- PLB External Memory Controller (EMC) Design Specification (v1.10a)
- PLB Synchronous DRAM (SDRAM) Controller**
- PLB Block RAM (BRAM) Interface Controller
- PLB Double Data Rate (DDR) Synchronous DRAM (SDRAM) Controller
- DDR Clock Module Reference Core
- Instruction Side OCM Block RAM (ISBRAM) Interface Controller
- Data Side OCM Block RAM (DSBRAM) Interface Controller
- Block RAM (BRAM) Block
- OPB ZBT Controller Design Specification

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the design specification for the LMB (Local Memory Bus) Block Ram (BRAM) Interface Controller.

The LMB BRAM Interface Controller is a module that attaches to one LMB.

This controller supports the LMB v1.0 bus protocol and byte-enable architecture. Any access size up to the width of the LMB data bus is permitted. The LMB BRAM Interface Controller is the interface between the LMB and the bram_block peripheral. A BRAM memory subsystem consists of the controller along with the actual BRAM components that are included in the bram_block peripheral. System Generator for Processors automatically adds the bram_block when a bram interface controller is instantiated. If the text-based Microprocessor Hardware Specification (MHS) file is used for design entry, then the bram controller and bram_block must both be explicitly instantiated.

Features

- LMB v1.0 bus interfaces with byte enable support
- Used in conjunction with bram_block peripheral to provide fast BRAM memory solution for MicroBlaze™ ILMB and DLMB ports.
- Supports byte, half-word, and word transfers.
- Supports Virtex™, Virtex-E, Spartan™-II, Virtex-II and Virtex-II Pro™ BRAM

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro, Virtex II	
Version of Core	lmb_bram_if_cntlr	v1.00a
Resources Used		
	Min	Max
Slices	3	3
LUTs	6	6
FFs	2	2
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the design specification for the Dual LMB (Local Memory Bus) Block Ram (BRAM) Interface Controller. The Dual LMB BRAM Interface Controller is a module that attaches to two LMBs (Local Memory Buses).

This controller supports the LMB V1.0 bus protocol and byte-enable architecture. Any access size up to the width of the LMB data bus is permitted. The LMB BRAM Interface Controller is the interface between the LMB and the bram_block peripheral. A BRAM memory subsystem consists of the controller along with the actual BRAM components that are included in the bram_block peripheral. System Generator for Processors automatically adds the bram_block when a bram interface controller is instantiated. If the text-based Microprocessor Hardware Specification (MHS) file is used for design entry, then the bram controller and bram_block must both be explicitly instantiated.

Features

- Two LMB V1.0 bus interfaces with byte enable support
- Used in conjunction with bram_block peripheral to provide fast BRAM memory solution for MicroBlaze ILMB and DLMB ports
- Utilizes dual port features of BRAM
- Supports byte, half-word, and word transfers
- Supports Virtex™, Virtex-E, Spartan™-II, Virtex-II, and Virtex-II Pro™ BRAM

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro, Virtex II	
Version of Core	lmb_lmb_bram_if_cntlr	v1.00a
Resources Used		
	Min	Max
Slices	3	3
LUTs	6	6
FFs	2	2
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This specification defines the architecture and interface requirements for the EMC. This module supports data transfers between the On-Chip Peripheral Bus (OPB) and external synchronous and asynchronous memory devices.

Example synchronous devices for use with this controller are the synchronous Integrated Device Technology, Inc. IDT71V546 SRAM with ZBT™ Feature. Example asynchronous devices include the IDT71V416S SRAM and Intel 28F128J3A StrataFlash Memory.

The Xilinx EMC design allows the customer to tailor the EMC to suit their application by setting certain parameters to enable/disable features.

Features

The EMC is a soft IP core designed for Xilinx FPGAs:

- Parameterized for up to a total of eight memory (Synchronous/Asynchronous) banks
 - Separate base addresses and address range for each bank of memory
- Separate Control Register for each bank of memory to control memory mode
- OPB V2.0 bus interface with byte-enable support
- Memory width is independent of OPB bus width (memory width must be less than or equal to OPB bus width)
 - Supports memory widths of 32 bits, 16 bits, or 8 bits
 - Memory width can vary by bank
- Parameterizable memory data-width/bus data-width matching
 - Multiple memory cycles will be performed when the memory width is less than the OPB bus width to provide full utilization of the OPB bus
 - Data-width matching can be enabled separately for each memory bank
- Configurable wait states for read, write, read in page, read recovery before write, and write recovery before read
 - Optional faster access for in-page read accesses (page size 8 bytes)

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	opb_emc	v1.10a
Resources Used		
	Min	Max
Slices	193	385
LUTs	216	362
FFs	239	534
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This specification defines the architecture and interface requirements for the EMC. This module supports data transfers between the On-Chip Peripheral Bus (OPB) and external synchronous and asynchronous memory devices.

Example synchronous devices for use with this controller are the synchronous Integrated Device Technology, Inc. IDT71V546 SRAM with ZBT™ Feature. Example asynchronous devices include the IDT71V416S SRAM and Intel 28F128J3A StrataFlash Memory.

The Xilinx EMC design allows the customer to tailor the EMC to suit their application by setting certain parameters to enable/disable features.

Features

The EMC is a soft IP core designed for Xilinx FPGAs:

- Parameterized for up to a total of eight memory (Synchronous/Asynchronous) banks
 - Separate base addresses and address range for each bank of memory
- Separate Control Register for each bank of memory to control memory mode
- OPB v2.0 bus interface with byte-enable support
- Supports 32-bit, 16-bit, and 8-bit bus interfaces
- Supports memory width of 32-bits, 16 bits, or 8 bits
- Memory width is independent of OPB bus width (memory width must be less than or equal to OPB bus width)
- Configurable wait states for read, write, read in page, read recovery before write, and write recovery before read
- Optional faster access for in-page read accesses (page size 8 bytes)
- System clock frequency of up to 133 MHz

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	opb_emc	v1.00d
Resources Used		
	Min	Max
Slice	163	194
LUTs	188	219
FFs	186	217
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

Click [here](#) to view this data sheet

Product Overview

Introduction

The Xilinx OPB SDRAM controller provides a SDRAM controller that connects to the OPB bus and provides the control interface for SDRAMs. It is assumed that the reader is familiar with SDRAMs and the IBM PowerPC.

Features

The Xilinx SDRAM Controller is a soft IP core designed for Xilinx FPGAs and contains the following features:

- OPB interface
- Performs device initialization sequence upon power-up and reset conditions
- Performs auto-refresh cycles
- Supports single-beat and burst transactions
- Supports target-word first cache-line transactions
- Supports cacheline latencies of 2 or 3 set by a design parameter
- Supports various SDRAM data widths set by a design parameter

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	SDRAM	v1.00c
Resources Used		
	Min	Max
Slices	222	246
LUTs	239	252
FFs	212	245
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The OPB BRAM Interface Controller is a module that attaches to the OPB (On-chip Peripheral Bus).

This controller supports the OPB v2.0 byte enable architecture. Any access size up to the width of the OPB data bus is permitted. The OPB BRAM Interface Controller is the interface between the OPB and the bram_block peripheral. A BRAM memory subsystem consists of the controller along with the actual BRAM components that are included in the bram_block peripheral. System Generator for Processors automatically adds the bram_block when a bram interface controller is instantiated. If the text-based Microprocessor Hardware Specification (MHS) file is used for design entry, then the bram controller and bram_block must both be explicitly instantiated.

Features

- OPB v2.0 bus interface with byte-enable support
- Used in conjunction with bram_block peripheral to provide total BRAM memory solution
- Supports a wide range of memory sizes
- Handles byte, half-word, word and double word transfers
 - Single cacheline bursts
- Handles Virtex™, Virtex-E, Spartan™-II, Virtex-II and Virtex-II Pro™ BRAM

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex, Virtex-E, Spartan-II, Virtex-II and Virtex-II Pro	
Version of Core	opb_bram_if_cntlr	v1.00a
Resources Used		
	Min	Max
Slices	42	42
LUTs	61	61
FFs	55	55
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the design specification for the Block Ram (BRAM) Controller Interface Controller. The OPB_BRAM Interface Controller is a module that attaches to the OPB (On-chip Peripheral Bus).

Features

- OPB v2.0 bus interface with byte-enable support
- Used in conjunction with bram_block peripheral to provide total BRAM memory solution.
- Supports a wide range of memory sizes.
- Handles byte, half-word, word and double word transfers
 - Single cacheline bursts
- Handles Virtex™, Virtex-E, Spartan™-II, Virtex-II and Virtex-II Pro™ BRAM

This controller supports the OPB v2.0 byte enable architecture. Any access size up to the width of the OPB data bus is permitted. The OPB BRAM Interface Controller is the interface between the OPB and the bram_block peripheral. A BRAM memory subsystem consists of the controller along with the actual BRAM components that are included in the bram_block peripheral. System Generator for Processors automatically adds the bram_block when a bram interface controller is instantiated. If the text-based Microprocessor Hardware Specification (MHS) file is used for design entry, then the bram controller and bram_block must both be explicitly instantiated.

Core Facts		
Core Specifics		
Supported Device Family	Virtex, Virtex-E, Spartan-II, Virtex-II and Virtex-II Pro	
Version of Core	opb_bram_if_cntlr	v2.00a
Resources Used		
	Min	Max
Slices	21	22
LUTs	9	39
FFs	37	2
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	N/A	
Design Tool Requirements		
Xilinx Implementation Tools	ISE 5.2i or higher	
Verification	N/A	
Simulation	ModelSim SE/EE 5.5e or higher	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

Introduction

The Xilinx OPB DDR SDRAM controller for Virtex™-II and Virtex™-II Pro FPGAs provides a DDR SDRAM controller that connects to the OPB bus and provides the control interface for DDR SDRAMs. It is assumed that the reader is familiar with DDR SDRAMs and the IBM PowerPC.

Features

The Xilinx DDR SDRAM Controller is a soft IP core designed for Xilinx FPGAs and contains the following features:

- OPB interface
- Performs device initialization sequence upon power-up and reset conditions
- Performs auto-refresh cycles
- Supports single-beat and burst transactions
- Supports target-word first cache-line transactions
- Supports cacheline latencies of 2 or 3 set by a design parameter
- Supports various DDR data widths set by a design parameter

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	DDR	v1.00b
Resources Used		
	Min	Max
Slices	278	314
LUTs	352	371
FFs	250	307
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

Click [here](#) to view this data sheet

Product Overview

Introduction

The OPB System ACE Interface Controller is a module that attaches to the OPB (On-chip Peripheral Bus).

This controller supports the OPB v2.0 byte enable architecture. Any access size up to the width of the OPB data bus is permitted. The OPB System ACE Interface Controller is the interface between the OPB and the System ACE CompactFlash solution peripheral.

Features

- OPB v2.0 bus interface with byte-enable support
- Used in conjunction with System ACE CompactFlash Solution to provide a System ACE memory solution
- System ACE Microprocessor Interface (MPU)
 - Read/Write from or to a CompactFlash device
 - MPU provides a clock for proper synchronization
 - Must comply with System ACE timing
- ACE Flash (Xilinx-supplied Flash Cards)
 - Densities of 128 Mbits and 256 Mbits
 - CompactFlash Type 1 form factor
 - Supports any standard CompactFlash module, or IBM microdrives up to 8 Gbits, all with the same form factor.
- Handles byte, half-word, and word transfers
- Supported in Virtex™, Virtex-E, Spartan™-II, Virtex-II and Virtex-II Pro™

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro, Virtex II, Virtex, Virtex E, Spartan II	
Version of Core	opb_sysace	v1.00a
Resources Used		
	Min	Max
Slices	154	171
LUTs	112	102
FFs	240	280
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This specification defines the architecture and interface requirements for the EMC. This module supports data transfers between the Processor Local Bus (PLB) and external synchronous and asynchronous memory devices. Example synchronous devices for use with this controller are the synchronous Integrated Device Technology, Inc. IDT71V546 SRAM with ZBT™ Feature.

Example asynchronous devices include the IDT71V416S SRAM, Intel 28F128J3A StrataFlash Memory and Xilinx's System ACE™ Devices. The EMC module is organized to be a PLB slave-only device, which differs from the IBM EBC specification.

The Xilinx EMC design allows the customer to tailor the EMC to suit their application by setting certain parameters to enable/disable features.

Features

The EMC is a soft IP core designed for Xilinx FPGAs:

- Parameterized for up to a total of eight memory (Synchronous/Asynchronous) banks
 - Separate base addresses and address range for each bank of memory
- Separate Control Register for each bank of memory to control memory mode
- Supports the following PLB transactions:
 - Single beat read/write transfers
 - In-line burst for 4,8,16 word cacheline read/write transfers
- Memory width independent of PLB bus width (memory width must be less than or equal to PLB bus width)
- Supports memory widths of 64 bits, 32 bits, 16 bits, or 8 bits
- Configurable wait states for read, write, read in page, read recovery before write, and write recovery before read
- Optional faster access for in-page read accesses (page size 8 bytes)
- System clock frequency of up to 133 MHz

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	plb_emc	v1.00d
Resources Used		
	Min	Max
Slices	342	376
LUTs	425	455
FFs	329	360
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This specification defines the architecture and interface requirements for the EMC. This module supports data transfers between the Processor Local Bus (PLB) and external synchronous and asynchronous memory devices.

Example synchronous devices for use with this controller are the synchronous Integrated Device Technology, Inc. IDT71V546 SRAM with ZBT™ Feature. Example asynchronous devices include the IDT71V416S SRAM and Intel 28F128J3A StrataFlash Memory. The Xilinx EMC design allows the customer to tailor the EMC to suit their application by setting certain parameters to enable/disable features.

Features

The EMC is a soft IP core designed for Xilinx FPGAs:

- Parameterized for up to a total of eight memory (Synchronous/Asynchronous) banks
 - Separate base addresses and address range for each bank of memory
- Separate Control Register for each bank of memory to control memory mode
- Supports the following PLB transactions:
 - Single beat read/write transfers
 - In-line burst for 4,8,16 word cacheline read/write transfers
- Memory width independent of PLB bus width (memory width must be less than or equal to PLB bus width)
 - Supports memory widths of 64 bits, 32 bits, 16 bits, or 8 bits, which can vary by bank
- Parameterizable memory data-width/bus data-width matching
 - Multiple memory cycles are performed when the memory width is less than the PLB bus width to provide full utilization of the PLB bus
 - Data-width matching can be enabled separately for each memory bank
- Configurable wait states for read, write, read in page, read recovery before write, and write recovery before read
 - Optional faster access for in-page read accesses (page size 8 bytes)

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	plb_emc	v1.10a
Resources Used		
	Min	Max
Slices	348	863
LUTs	366	920
FFs	465	1112
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The Xilinx PLB SDRAM controller provides a SDRAM controller which connects to the PLB bus and provides the control interface for SDRAMs. It is assumed that the reader is familiar with SDRAMs and the IBM PowerPC™.

Features

The Xilinx SDRAM Controller is a soft IP core designed for Xilinx FPGAs and contains the following features:

- PLB interface
- Performs device initialization sequence upon power-up and reset conditions
- Performs auto-refresh cycles
- Supports single-beat and burst transactions
- Supports target-word first cache-line transactions
- Supports cacheline latencies of 2 or 3 set by a design parameter
- Supports various SDRAM data widths set by a design parameter

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	SDRAM	v1.00c
Resources Used		
	Min	Max
Slices	368	622
LUTs	340	665
FFs	408	660
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/a	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The PLB BRAM Interface Controller is a module that attaches to the PLB (Processor Local Bus).

This controller supports the PLB v3.4 byte enable architecture. Any access size up to the width of the PLB data bus is permitted. The PLB BRAM Interface Controller is the interface between the PLB and the bram_block peripheral. A BRAM memory subsystem consists of the controller along with the actual BRAM components that are included in the bram_block peripheral. System Generator for Processors automatically adds the bram_block when a bram interface controller is instantiated. If the text-based Microprocessor Hardware Specification (MHS) file is used for design entry, then the bram controller and bram_block must both be explicitly instantiated.

Features

- PLB v3.4 bus interface with byte_enable support
- Used in conjunction with bram_block peripheral to provide total BRAM memory solution
- Supports a wide range of memory sizes
- Handles byte, half-word, word and double word transfers
 - Single cacheline bursts
- Handles Virtex™, Virtex-E, Spartan™-II, Virtex-II and Virtex-II Pro™ BRAM

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro, Virtex II, Virtex, Virtex E, Spartan II	
Version of Core	plb_bram_if_cntlr	v1.00a
Resources Used		
	Min	Max
Slices	181	181
LUTs	204	204
FFs	223	223
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The Xilinx PLB DDR SDRAM controller for Virtex™-II and Virtex-II Pro™ FPGAs provides a DDR SDRAM controller that connects to the PLB bus and provides the control interface for DDR SDRAMs. It is assumed that the reader is familiar with DDR SDRAMs and the IBM PowerPC.

Features

The Xilinx DDR SDRAM Controller is a soft IP core designed for Xilinx FPGAs and contains the following features:

- PLB interface
- Performs device initialization sequence upon power-up and reset conditions
- Performs auto-refresh cycles
- Supports single-beat and burst transactions
- Supports target-word first cache-line transactions
- Supports cacheline latencies of 2 or 3 set by a design parameter

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro, Virtex II, Virtex, Virtex E, Spartan™ II	
Version of Core	DDR	v1.00b
Resources Used		
	Min	Max
Slices	509	861
LUTs	602	977
FFs	477	919
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The DDR Clock Module Reference core, targeted for the Virtex™-II Pro FPGAs, provides a clocking solution for embedded processor designs that make use of a Dual Data Rate (DDR) memory controller and external DDR memory. The reference core provides the clocks required to support the DDR memory as well as basic system clocks for the PPC405 CPU, the PLB bus, and the OPB bus.

This DDR Clock Module is also suitable for use with the ML300 Development Board. The default configuration provides a 300 MHz CPU clock, a 100 MHz PLB clock, and a 100 MHz OPB clock. The clock module uses three Digital Clock Managers (DCMs) to provide the correct clock frequency multiplication and phase shifts for the CPU, bus, and DDR clocks.

Features

- Provides clock multiplication for PPC405
- Provides clock multiplication and phase shifting for DDR clocks
- Provides clock multiplication for PLB bus and OPB bus
- Provides clocking solution for systems using DDR on the ML300 Development Board
- Supports PLB and OPB clock rate of 100 MHz, and PPC405 clock rate of 300 MHz

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex™-II, Virtex-II Pro™	
Version of Core	ddr_clock_module_ref	v1.00a
Resources Used		
	Min	Max
Slices		
LUTs	2	2
FFs		
Block RAMs		
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	MHS, VHDL	
Constraints File	None	
Verification	None	
Instantiation Template	None	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	EDK 3.1 or later ISE 4.2.03i or later	
Verification	ModelSim PE 5.4e	
Simulation	ModelSim PE 5.4e	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the design specification for the Instruction Side OCM Block RAM (ISBRAM) Interface Controller. The ISBRAM Interface Controller is a module that connects the PowerPC™ 405 to BRAM blocks.

This controller supports the ISOCM port of PowerPC405. PowerPC 405 fetches two instructions per cycle from a 64-bit ISOCM read-only port. The ISBRAM controller connects this instruction fetch port to one port of the BRAM memory (Port B). PowerPC 405 can also write to the ISBRAM using the DCR registers ISINIT and ISFILL through a 32-bit write-only port. The ISBRAM controller connects the DCR write port to the other port of the BRAM memory (Port A). The Instruction Side OCM BRAM Interface Controller is the interface between the ISOCM and the bram_block peripheral. The ISBRAM memory subsystem consists of the controller along with the actual BRAM components that are included in the bram_block peripheral.

Features

- Used in conjunction with bram_block peripheral to provide a deterministic ISBRAM memory solution for PowerPC405
- Utilizes dual port features of BRAM
- Supports byte, half-word, and word transfers
- Supports Virtex-II Pro BRAM

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II	
Version of Core	isbram_if_cntlr	v1.00a
Resources Used		
	Min	Max
Slices	0	0
LUTs	0	0
FFs	0	0
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The DSBRAM Interface Controller is a module that connects the PowerPC™ 405 to BRAM blocks.

This controller supports the DSOCM port of PowerPC405. The swidth of the data bus is 32bits. The Data Side OCM BRAM Interface Controller is the interface between the DSOCM and the bram_block peripheral. The DSBRAM memory subsystem consists of the controller along with the actual BRAM components that are included in the bram_block peripheral.

Features

- Used in conjunction with bram_block peripheral to provide a deterministic DSBRAM memory solution for PowerPC405.
- Utilizes dual port features of BRAM
- Supports byte, half-word, and word transfers
- Supports Virtex-II Pro™ BRAM

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro, Virtex II	
Version of Core	dsbram_if_cntlr	v1.00a
Resources Used		
	Min	Max
Slices	0	0
LUTs	0	0
FFs	0	0
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The BRAM Block is a parameterizable memory module that attaches to a variety of BRAM Interface Controllers.

Features

- Uses from 4 to 64 dual-port BRAMs to provide memory sizes from 2KB to 128KB
- Both Port A and Port B of the memory block can be connected to independent BRAM Interface Controllers.
- When used in conjunction with BRAM Interface Controllers, provides fast internal memory for LMB (Local Memory Bus), OPB (On-chip Peripheral Bus), PLB (Processor Local Bus), and OCM (On-Chip Memory).
- Supports PowerPC™ and MicroBlaze™ systems
- Supports byte, half-word, word, and doubleword transfers
- Supports Virtex™, Virtex-E, Spartan™-II, Virtex-II and Virtex-II Pro™ BRAM

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro, Virtex II	
Version of Core	bram_block	v1.00a
Resources Used		
	Min	Max
Slices	N/A	N/A
LUTs	N/A	N/A
FFs	N/A	N/A
Block RAMs	4	64
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Peripheral Cores

This section of the User Guide contains information on the following peripheral cores:

- OPB Interrupt Controller (v1.00b)
- OPB Interrupt Controller (v1.00c)
- OPB 16550 UART
- OPB 16450 UART
- OPB UART Lite
- OPB JTAG_UART
- OPB IIC Bus Interface
- OPB Serial Peripheral Interface (SPI)
- OPB IPIF/LogiCore v3 PCI Core Bridge
- OPB Ethernet Media Access Controller (EMAC) (v1.00j)
- OPB Ethernet Media Access Controller (EMAC) (v1.00k)
- OPB Ethernet Media Access Controller (EMAC) (v1.00m)
- OPB Ethernet Lite Media Access Controller
- OPB Asynchronous Transfer Mode Controller (OPB_ATMC) (v1.00b)
- OPB Asynchronous Transfer Mode Controller (OPB_ATMC) (v2.00a)
- OPB HDLC Interface
- OPB Timebase WDT
- OPB Timer/Counter
- OPB General Purpose Input/Output (GPIO)
- Microprocessor Debug Module (MDM)
- OPB Central DMA Controller
- Channel FIFO
- Fixed Interval Timer (FIT)
- MII to RMI Design Specification
- PLB 1-Gigabit Ethernet Media Access Controller (MAC) With DMA
- PLB 1-Gigabit Ethernet Media Access Controller (MAC)
- PLB 16550 UART (v1.00b)
- PLB 16550 UART (v1.00c)
- PLB 16450 UART (v1.00b)

[PLB 16450 UART \(v1.00c\)](#)

[PLB RapidIO LVDS Design](#)

[PLB Asynchronous Transfer Mode Controller \(PLB_ATMC\)](#)

[DCR Interrupt Controller Specification \(v1.00a\)](#)

[DCR Interrupt Controller Specification \(v1.00b\)](#)

Click [here](#) to view this data sheet

Product Overview

Introduction

This document describes the specifications for a ZBT controller core for the On-chip Peripheral Bus (OPB).

The ZBT Memory Controller is a 32-bit peripheral that attaches to the OPB.

Features

- OPB v2.0 bus interface with byte-enable support
- Supports 32-bit bus interfaces
- Supports memory width of 32-bits

Operation

The OPB ZBT Controller provides an interface between the OPB and external ZBT memories. The controller supports OPB data bus widths of 32bits, and memory subsystem widths of 32 bits. This controller supports the OPB V2.0 byte enable architecture.

LogiCORE™ Facts		
Core Specifics		
Supported Device Family		
Version of Core	opb_zbt_controller	v1.00a
Resources Used		
	Min	Max
I/O		
LUTs		
FFs		
Block RAMs		
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats		
Constraints File		
Verification		
Instantiation Template		
Reference Designs		
Design Tool Requirements		
Xilinx Implementation Tools		
Verification		
Simulation		
Synthesis		
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

An Interrupt Controller is composed of a bus-centric wrapper containing the IntC core and a bus interface. The IntC core is a simple, parameterized interrupt controller that, along with the appropriate bus interface, attaches to either the OPB (On-chip Peripheral Bus) or DCR (Device Control Register) Bus. It can be used in embedded PowerPC systems (Virtex-II Pro™ devices), and in MicroBlaze™ soft processor systems. There are two versions of the Simple Interrupt Controller:

- OPB IntC (OPB interface)
- DCR IntC (DCR interface)

In this document, IntC and Simple IntC are used interchangeably to refer to functionality or interface signals common to all variations of the Simple Interrupt Controller. However, when it is necessary to make a distinction, the interrupt controller is referred to as OPB IntC or DCR IntC.

Features

- Modular design provides a core interrupt controller functionality instantiated within a bus interface design (currently OPB and DCR buses are supported)
- OPB v2.0 bus interface with byte-enable support (IBM SA-14-2528-01 64-bit On-chip Peripheral Bus Architecture Specifications, v2.0)
- Supports data bus widths of 8-bits, 16-bits, or 32-bits for OPB interface
- Number of interrupt inputs configurable up to the width of data bus
- Easily cascaded to provide additional interrupt inputs
- Interrupt Enable Register for selectively disabling individual interrupt inputs
- Master Enable Register for disabling interrupt request output
- Each input is configurable for edge or level sensitivity; edge sensitivity can be configured for rising or falling; level sensitivity can be active-high or -low
- Automatic edge synchronization when inputs are configured for edge sensitivity
- Output interrupt request pin is configurable for edge or level generation — edge generation configurable for rising or falling; level generation configurable for active-high or -low

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	opb_intc	v1.00b
Resources Used		
	Min	Max
I/O	55	116
LUTs	42	395
FFs	63	342
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

An Interrupt Controller is composed of a bus-centric wrapper containing the IntC core and a bus interface. The IntC core is a simple, parameterized interrupt controller that, along with the appropriate bus interface, attaches to either the OPB (On-chip Peripheral Bus) or DCR (Device Control Register) Bus. It can be used in embedded PowerPC systems (Virtex-II Pro™ devices), and in MicroBlaze™ soft processor systems. There are two versions of the Simple Interrupt Controller:

- OPB IntC (OPB interface)
- DCR IntC (DCR interface)

In this document, IntC and Simple IntC are used interchangeably to refer to functionality or interface signals common to all variations of the Simple Interrupt Controller. However, when it is necessary to make a distinction, the interrupt controller is referred to as OPB IntC or DCR IntC.

Features

- Modular design provides a core interrupt controller functionality instantiated within a bus interface design (currently OPB and DCR buses are supported)
- OPB v2.0 bus interface with byte-enable support (IBM SA-14-2528-01 64-bit On-chip Peripheral Bus Architecture Specifications, Version 2.0)
- Supports data bus widths of 8-bits, 16-bits, or 32-bits for OPB interface
- Number of interrupt inputs configurable up to the width of data bus
- Easily cascaded to provide additional interrupt inputs
- Interrupt Enable Register for selectively disabling individual interrupt inputs
- Master Enable Register for disabling interrupt request output
- Each input is configurable for edge or level sensitivity; edge sensitivity can be configured for rising or falling; level sensitivity can be active-high or -low
- Automatic edge synchronization when inputs are configured for edge sensitivity
- Output interrupt request pin is configurable for edge or level generation — edge generation configurable for rising or falling; level generation configurable for active-high or -low

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	opb_intc	v1.00c
Resources Used		
	Min	Max
I/O	55	116
LUTs	42	395
FFs	63	342
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the specification for the OPB Universal Asynchronous Receiver/Transmitter (UART) Intellectual Property (IP).

The UART described in this document has been designed incorporating features described in National Semiconductor PC16550D UART with FIFOs data sheet (June, 1995), (<http://www.national.com/pf/PC/PC16550D.html>).

The National Semiconductor PC16550D data sheet is referenced throughout this document and should be used as the authoritative specification. Differences between the National Semiconductor implementation and the OPB UART Point Design implementation are highlighted and explained in this [data sheet](#).

Features

- Hardware and software register compatible with all standard 16450 and 16550 UARTs
- Implements all standard serial interface protocols
 - 5, 6, 7, or 8 bits per character
 - Odd, Even, or no parity detection and generation
 - 1, 1.5, or 2 stop bit detection and generation
 - Internal baud rate generator and separate receiver clock input
 - Modem control functions
 - False start bit detection and recovery
 - Prioritized transmit, receive, line status, and modem control interrupts
 - Line break detection and generation
 - Internal loop back diagnostic functionality
 - Independent 16 word transmit and receive FIFOs
- Registers
 - Receiver Buffer Register (Read Only)
 - Transmitter Holding Register (Write Only)
 - Interrupt Enable Register
 - Interrupt Identification Register (Read Only)
 - FIFO Control Register (Read/Write)
 - Line Control and Line Status Registers
 - Modem Control and Modem Status Registers
 - Scratch Register

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II	
Version of Core	opb_uart16550	v1.00c
Resources Used		
	Min	Max
Slices	442	442
LUTs	534	534
FFs	401	401
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

- Divisor Latch (least and more significant byte)
- System clock frequency of 100 MHz

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the specification for the OPB Universal Asynchronous Receiver/Transmitter (UART) Intellectual Property (IP).

The UART described in this document has been designed incorporating the features described in *National Semiconductor PC16550D UART with FIFOs* data sheet (June, 1995), (<http://www.national.com/pf/PC/PC16550D.html>).

The National Semiconductor PC16550D data sheet is referenced throughout this document and should be used as the authoritative specification. Differences between the National Semiconductor implementation and the OPB UART Point Design implementation are highlighted and explained in this [data sheet](#).

Features

- Hardware and software register compatible with all standard 16450 UARTs
- Implements all standard serial interface protocols
 - 5, 6, 7, or 8 bits per character
 - Odd, Even, or no parity detection and generation
 - 1, 1.5, or 2 stop bit detection and generation
 - Internal baud rate generator and separate receiver clock input
 - Modem control functions
 - False start bit detection and recovery
 - Prioritized transmit, receive, line status, and modem control interrupts
 - Line break detection and generation
 - Internal loop back diagnostic functionality
- Registers
 - Receiver Buffer Register (Read Only)
 - Transmitter Holding Register (Write Only)
 - Interrupt Enable Register
 - Interrupt Identification Register (Read Only)
 - Line Control and Line Status Registers
 - Modem Control and Modem Status Registers
 - Scratch Register
 - Divisor Latch (least and more significant byte)
- System clock frequency of 100 MHz

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II	
Version of Core	opb_uart16450	v1.00c
Resources Used		
	Min	Max
Slices	341	341
LUTs	357	357
FFs	347	347
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview
Introduction

This document describes the specifications for a UART core for the On-Chip Peripheral Bus (OPB). The UART Lite is a module that attaches to the OPB.

Features

- OPB v2.0 bus interface with byte-enable support
- Supports 8-bit bus interfaces
- One transmit and one receive channel (full duplex)
- 16-character transmit FIFO and 16-character receive FIFO
- Number of databits in a character is configurable (5-8)
- Parity; can be configured for odd or even
- Configurable baud rate

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II	
Version of Core	opb_uartlite	v1.00b
Resources Used		
	Min	Max
Slices	N/A	N/A
LUTs	88	108
FFs	48	57
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document describes the specifications for a JTAG_UART core for the On-Chip Peripheral Bus (OPB). The JTAG_UART is a module that attaches to the OPB.

Features

- Mimics UART functionality to MicroBlaze but sends data over JTAG
- OPB v2.0 bus interface with byte-enable support
- Supports 8-bit bus interfaces
- One transmit and one receive channel (full duplex)
- 16-character transmit FIFO and 16-character receive FIFO
- Requires xmd or xmdterm to run on host for JTAG communication
- Able to reset system and MicroBlaze™
- Able to assert break signals to MicroBlaze

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II	
Version of Core	opb_jtag_uart	v1.00b
Resources Used		
	Min	Max
Slices	57	57
LUTs	86	86
FFs	70	70
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This specification defines the architecture and interface requirements for this module. This includes registers that must be initialized for proper operation. This module supports all features, except high speed mode, of the Philips I²C bus, v2.1, release January 2000. See the Specification Exceptions of this [data sheet](#) for more details.

The Xilinx IIC design allows the customer to tailor the IIC to suit their application by setting certain parameters to enable/disable features. The parameterizable features of the design are discussed in IIC Design Parameters section of this data sheet.

Features

The IIC bus interface is a soft IP core designed for Xilinx FPGAs and contains these features:

- Master or Slave operation
- Multi-master operation
- Software selectable acknowledge bit
- Arbitration lost interrupt with automatic mode switching from Master to Slave
- Calling address identification interrupt with automatic mode switching from Master to Slave
- START and STOP signal generation/detection
- Repeated START signal generation
- Acknowledge bit generation/detection
- Bus busy detection
- Fast Mode 400 KHz operation or Standard Mode 100 KHz
- 7 Bit or 10 Bit addressing
- General Call Enable or Disable
- Transmit and Receive FIFOs - 16 bytes deep
- Throttling

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, VirtexE, Virtex, Spartan™ II, Spartan IIE	
Version of Core	opb_iic	v1.01a
Resources Used		
	Min	Max
I/O	2	2
LUTs	404	425
FFs	272	280
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	N/A	
Design Tool Requirements		
Xilinx Implementation Tools	Alliance	
Verification	N/A	
Simulation	N/A	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document presents specifications for the VHDL implementation of the Motorola Serial Peripheral Interface (SPI) in a Xilinx FPGA. The original specifications closely followed the Motorola M68HC11-Rev. 4.0 Reference Manual, and this document emphasizes the M68HC-11 specifications. The design, however, was enhanced with a number of exceptions and enhancements as described in this document. The default mode of operation has been changed to a manual slave select operation (not included in the M68HC11 specification).

The Serial Peripheral Interface (SPI) is a full-duplex, synchronous channel that supports a four-wire interface (receive, transmit, clock, and slave select) between one master and one slave. The original specifications followed closely Motorola's M68HC11-Rev. 4.0 Reference Manual.

There are differences from the 68HC11 specification that should be reviewed when utilizing this SPI Assembly, see the Specification Exceptions section of this [data sheet](#).

The Version B specification has extended functionality, including a manual slave select mode. This mode allows you to manually control the slave select line directly by the data written to the slave select register.

This allows transfers of an arbitrary number of bytes without toggling the slave select line until all bytes are transferred. In this mode, you must toggle the slave select by writing the appropriate data to the slave select register. The manual slave select mode is the default mode of operation.

This parameterized module permits additional slaves with automatic generation of the required decoding of the individual slave select outputs by the master. Additional masters can be added as well; however, means to detect all possible conflicts are not implemented with this interface standard, but rather require the software to arbitrate bus control in order to eliminate conflicts.

At this time only SPI slave devices are allowed off-chip. This is an artifact of software master control arbitration which can not be guaranteed if off-chip masters were allowed and is due to issues with asynchronous external clocks as well. Essentially any number of internal slave and master SPI devices is allowed. The actual number is limited by the performance that is desired.

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, VirtexE, Virtex, Spartan™ II, Spartan IIE	
Version of Core	opb_spi	v1.00b
Resources Used		
	Min	Max
I/O	N/A	N/A
LUTs	N/A	N/A
FFs	N/A	N/A
Block RAMs	N/A	N/A
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	None	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

Features

- Four signal interface (MOSI, MISO, SCK, and \overline{SS})-- \overline{SS} bit for each slave on the SPI bus
- Three signal in/out (in, out, 3-state) for implementing 3-state SPI device in/outs to support multi-master configuration within the FPGA
- Full-duplex operation
- Works with N times 8-bit data characters in default configuration. The default mode implements manual control of the \overline{SS} output via data written to the slave select register which appears directly on the \overline{SS} output when the master is enabled. This mode can be used only with external slave devices. In addition, an optional operation where the \overline{SS} output is toggled automatically with each 8-bit character transfer by the master device internal state machine can be selected via a bit in the command register for SPI master devices.
- Supports back-to-back character transmission and reception
- Master and slave SPI modes supported
- Multi-master environment supported (implemented with 3-state drivers and requires software arbitration for possible conflict)
- Multi-slave environment supported (automatic generation of additional master slave select signals)
- Continuous transfer mode for automatic scanning of a peripheral
- Supports maximum clock rates of up to one-half the OPB clock rate in both master and slave modes when both SPI devices are in the same FPGA part (routing constraints of SPI bus signals must be incorporated in map/par process). In anticipation of remote master operation, slaves operation supports one-fourth the OPB clock rate (artifact of asynchronous SCK clock relative to the OPB clock which requires clock synchronization).
- Parameterizable baud rate generator
- Programmable clock phase and polarity
- External ports (selected via a parameter) for off-chip slave interconnects (off-chip masters not supported)
- Optional transmit and receive FIFOs (implemented as a pair only)
- Local loopback capability for testing

The Xilinx SPI design allows you to tailor the SPI Assembly to suit your application by setting certain parameters to enable or disable features. The parameterizable features of the design are discussed in the SPI Configuration Parameters section of this [data sheet](#).

The basic SPI device consists of a register module and the SPI module. Optional FIFOs and support are discussed in a later section. The register block includes all memory mapped registers (as shown in Figure 1) and resides on the Xilinx OPB.

As shown in Figure 3, the SPI module consists of transmitter and receiver sections, a parameterized baud rate generator (BRG) and a control unit. The registers are an 8-bit status register, an 8-bit control register, an N-bit slave select register and a pair of 8-bit transmit/receiver registers.

In the 68HC11 implementation, the transmit register is transparent to the shift register and the receive register is double buffered with the shift register. In this implementation without FIFOs, both the transmit and receive register are double buffered.

Hardware prevents data transfer from the transmit buffer to the shift register while an SPI transfer is in progress, consequently, the write collision error described in the MC68HC11 Reference Manual can not occur and the WCOL flag is not supported.

All registers are accessed directly from the Xilinx OPB which is a subset of IBM's 64-bit OPB utilizing byte enables (see IBM's 64-Bit On-Chip Peripheral Bus document for details). As shown in Figure 1, optional FIFOs can be implemented on both receive and transmit paths.

Click [here](#) to view this data sheet

Product Overview

Introduction

The OPB IPIF/LogiCORE PCI64 v3.0 bridge design described in this document bridges between the OPB IPIF (On-Chip Peripheral Bus Intellectual Property InterFace) and the Xilinx LogiCORE PCI64 Interface v3.0 core to provide full bridge functionality between the Xilinx 32-bit OPB and a 32-bit V2.2 compliant PCI (Peripheral Component Interconnect) bus. This bridge is referred to as the IPIF/V3 bridge in this document.

The Xilinx OPB is a 32-bit bus that is a subset of the IBM OPB that is described in 64-Bit On-Chip Peripheral Bus Architecture Specifications v2.0. Details of bridging between a PCI bus and the v3 protocol that interfaces with the OPB IPIF/V3 bridge described herein is described in detail in the LogiCORE PCI64 Interface v3.0 Interface Data Sheet and Xilinx The Real-PCI Design Guide v3.0.

Host bridge functionality (often called North bridge) is implemented in this release. Configuration read and write PCI commands can be performed from the OPB-side of the bridge. the OPB IPIF/V3PCI core bridge will only support the 32-bit PCI bus.

Not all commands supported by the v3 core are supported. Details of exceptions are explained in the following section.

The Xilinx IPIF/V3 PCI core bridge design has parameters that allow customers to configure the IPIF/V3 PCI core bridge to suit their application. The parameterizable features of the design are discussed in the OPB PCI Bus Interface Parameters section of this [data sheet](#).

Features

- OPB and PCI clocks are required to be a global buffer
- 33/66 MHz, 32-bit PCI buses
- Utilizes the SRAM interface of the OPB IPIF for PCI data transfers
- Includes a master IP module for PCI initiator transactions, which follows the protocol for interfacing with the master attachment
- Full bridge functionality
 - OPB Master read and write of a remote PCI target (both single and burst))
 - PCI Initiator read and write to a remote OPB slave (both single and multiple)

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, VirtexE, Virtex, Spartan™ II, Spartan IIE	
Version of Core	opb_pci	v1.00b
Resources Used		
	Min	Max
I/O	48	49
LUTs	3050	3950
FFs	1470	1850
Block RAMs	2	20
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VDHL	
Constraints File	None	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST & Synplify (state machines)	
Support		
Support provided by Xilinx, Inc.		

- Supports all PCI commands supported by the v3 core with the following exceptions:
 - The interrupt acknowledge command will be supported in future releases of the core and will be only for OPB masters to execute the command while being ignored when executed by a remote PCI agent.

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

- The special cycle command will be supported in future releases of the core.
- I/O read and I/O write commands are supported only for OPB master read and writes of PCI I/O space as designated by its associated memory designator bits of the generics. All memory space on the OPB-side is designated as memory space in the PCI sense, hence, I/O commands can not be used to access memory on the OPB-side.
- Configuration read and writes are supported (including self-configuration transactions) only when upper word address lines are utilized for IDSel lines. The configuration read and write commands are automatically executed by writing to the configuration data port register. Data in the configuration address port register and the configuration bus number/subordinate bus number register is used in execution of the configuration transaction per PCI 2.2 specification.
- Memory read line command is not supported mainly because the OPB does not have direct support of this type of line wrapping.
- Memory write invalidate is not supported where the v3 core is a target because the low utility of having memory on the OPB side with a cached counterpart and memory controller on the PCI side of the bridge. The v3 core does not support this command when it is an initiator.
- Programmable enabling of the bridge feature to inhibit one or any number of the four transfer types when an error is detected.
- Supports up to 6 OPB devices with unique memory OPB memory space
 - Each device has the following generics: OPB BAR, length, prefetchability, Big Endian to Little Endian translate, and offset for mapping OPB address space to PCI address space.
- Supports up to 3 PCI devices with unique memory PCI memory space. The v3 core supports up to 3 PCI BAR.
 - Each device has the following generics: PCI BAR, length, prefetchability, memory designator, Little Endian to Big Endian translate, and offset for mapping PCI address space to OPB address space
- Registers include
 - Interrupt and interrupt enable registers at different hierarchal levels
 - Reset
 - Prefetch override
 - Bridge and v3 Core Transaction Status
 - Inhibit Transfers on Error
 - OPB Mst Address Definition
 - OPB Mst Read and Write Addresses
- Address range decode for supported BAR, length, and prefetch operation
- OPB-side Interrupts include
 - OPB Master Read SERR and PERR
 - OPB Master Read Target Abort
 - OPB Master Write SERR and PERR
 - OPB Master Write Target Abort
 - OPB Master Abort Write
 - OPB Master Write Retry and Retry Disconnect
 - OPB Master Write Retry Timeout
 - OPB Master Write Range
 - PCI Initiator Read and Write SERR
- Asynchronous FIFOs with backup capability
- Synchronization circuits for signals that cross time-domain boundaries
- PCI and OPB clocks can be totally independent
- Responding to the PCI latency timer
- Completing posted operations prior to initiating new operations

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the design specification for the 10/100 Mbs Ethernet Media Access Controller (EMAC). The EMAC incorporates the applicable features described in IEEE Std. 802.3 MII interface specification. The IEEE Std. 802.3 MII interface specification is referenced throughout this document and should be used as the authoritative specification. Differences between the IEEE Std. 802.3 MII interface specification and the Xilinx EMAC implementation are highlighted and explained in the Specification Exceptions section of this [data sheet](#).

The EMAC Interface design is a soft intellectual property (IP) core designed for implementation in a Virtex-E, Virtex-II, Spartan-II, Spartan-IIIE, or Virtex-II Pro FPGA. It supports the IEEE Std. 802.3 Media Independent Interface (MII) to industry standard Physical Layer (PHY) devices and communicates to a processor via an IBM On-Chip Peripheral Bus (OPB) interface. The design provides a 10 Megabits per second (Mbps) and 100 Mbps (also known as Fast Ethernet) EMAC Interface. This design includes many of the functions and the flexibility found in dedicated Ethernet controller devices currently on the market.

The Xilinx EMAC design allows the customer to tailor the EMAC to suit their application by setting certain parameters to enable/disable features. The parameterizable features of the design are discussed in EMAC Design Parameters.

The EMAC is comprised of two IP blocks as shown in Figure 1: The IP Interface (IPIF) block is a subset of OPB bus interface features chosen from the full set of IPIF features to most efficiently couple the second block, the EMAC core, to the OPB processor bus for this packet¹ based interface (this combined entity is referred to as a device). Although there are separate specifications for the IPIF design, this specification addresses the specific implementation required for the EMAC design.

EMAC Endianness

Please note that the EMAC is designed as a big endian device (bit 0 is the most significant bit and is shown on the left of a group of bits).

1. IEEE Std. 802.3 uses the terms Frame and Packet interchangeably when referring to the Ethernet unit of transmission; this specification does likewise

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II, Spartan IIE	
Version of Core	opb_ethernet	v1.00j
Resources Used		
	Min	Max
I/O	179	179
LUTs	1998	3642
FFs	1528	2215
Block RAMs	2	8
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

The 4-bit transmit and receive data interface to the external PHY is little endian (bit 3 is the most significant bit and appears on the left of the bus). The MII management interface to the PHY is serial with the most significant bit of a field being transmitted first.

Features

- 32-bit OPB master and slave interfaces
- Memory mapped direct I/O interface to registers and FIFOs as well as DMA and Scatter/Gather DMA capabilities for low processor and bus utilization
- Media Independent Interface (MII) for connection to external 10/100 Mbps PHY transceivers
 - IEEE 802.3-compliant MII
 - Supports auto-negotiable and non auto-negotiable PHYs
 - Supports 10BASE-T and 100BASE-TX/FX IEEE 802.3 compliant MII PHYs at full or half duplex
- Independent internal 2K byte TX and RX FIFOs for holding data for more than one packet
- 16 entry deep FIFOs for the Transmit Length, Receive Length, and Transmit Status registers to support multiple packet operation
- CSMA/CD compliant operation at 10 Mbps and 100 Mbps in half duplex mode
- Programmable PHY reset signal
- Internal loop-back capability
- Supports unicast, multicast, and broadcast transmit and receive modes as well as promiscuous address receive mode
- Supports a "Freeze" (graceful halt) mode based on input signal assertion to assist with emulator based software development
- Provides auto or manual source address field insertion or overwrite for transmission
- Provides auto or manual pad and Frame Check Sequence (FCS) field insertion
- Provides auto pad and FCS field stripping on receive
- Processes received pause packets
- Supports reception of longer VLAN type frames
- Supports MII management control writes and reads with MII PHYs
- Programmable interframe gap
- Provides counters and interrupts for many error conditions

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the design specification for the 10/100 Mbs Ethernet Media Access Controller (EMAC). The EMAC incorporates the applicable features described in IEEE Std. 802.3 MII interface specification. The IEEE Std. 802.3 MII interface specification is referenced throughout this document and should be used as the authoritative specification. Differences between the IEEE Std. 802.3 MII interface specification and the Xilinx EMAC implementation are highlighted and explained in the Specification Exceptions section of this [data sheet](#).

The EMAC Interface design is a soft intellectual property (IP) core designed for implementation in a Virtex-E, Virtex-II, Spartan-II, Spartan-IIe, or Virtex-II Pro FPGA. It supports the IEEE Std. 802.3 Media Independent Interface (MII) to industry standard Physical Layer (PHY) devices and communicates to a processor via an IBM On-Chip Peripheral Bus (OPB) interface. The design provides a 10 Megabits per second (Mbps) and 100 Mbps (also known as Fast Ethernet) EMAC Interface. This design includes many of the functions and the flexibility found in dedicated Ethernet controller devices currently on the market.

The Xilinx EMAC design allows the customer to tailor the EMAC to suit their application by setting certain parameters to enable/disable features. The parameterizable features of the design are discussed in EMAC Design Parameters.

The EMAC is comprised of two IP blocks as shown in Figure 1: The IP Interface (IPIF) block is a subset of OPB bus interface features chosen from the full set of IPIF features to most efficiently couple the second block, the EMAC core, to the OPB processor bus for this packet¹ based interface (this combined entity is referred to as a device). Although there are separate specifications for the IPIF design, this specification addresses the specific implementation required for the EMAC design.

EMAC Endianness

Please note that the EMAC is designed as a big endian device (bit 0 is the most significant bit and is shown on the left of a group of bits).

1. IEEE Std. 802.3 uses the terms Frame and Packet interchangeably when referring to the Ethernet unit of transmission; this specification does likewise

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II, Spartan IIE	
Version of Core	opb_ethernet	v1.00k
Resources Used		
	Min	Max
I/O	179	179
LUTs	2018	3688
FFs	1557	2228
Block RAMs	2	16
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

The 4-bit transmit and receive data interface to the external PHY is little endian (bit 3 is the most significant bit and appears on the left of the bus). The MII management interface to the PHY is serial with the most significant bit of a field being transmitted first.

Features

- 32-bit OPB master and slave interfaces
- Memory mapped direct I/O interface to registers and FIFOs as well as DMA and Scatter/Gather DMA capabilities for low processor and bus utilization
- Media Independent Interface (MII) for connection to external 10/100 Mbps PHY transceivers
 - IEEE 802.3-compliant MII
 - Supports auto-negotiable and non auto-negotiable PHYs
 - Supports 10BASE-T and 100BASE-TX/FX IEEE 802.3 compliant MII PHYs at full or half duplex
- Independent internal 2K byte TX and RX FIFOs for holding data for more than one packet
- 16 entry deep FIFOs for the Transmit Length, Receive Length, and Transmit Status registers to support multiple packet operation
- CSMA/CD compliant operation at 10 Mbps and 100 Mbps in half duplex mode
- Programmable PHY reset signal
- Internal loop-back capability
- Supports unicast, multicast, and broadcast transmit and receive modes as well as promiscuous address receive mode
- Supports a "Freeze" (graceful halt) mode based on input signal assertion to assist with emulator based software development
- Provides auto or manual source address field insertion or overwrite for transmission
- Provides auto or manual pad and Frame Check Sequence (FCS) field insertion
- Provides auto pad and FCS field stripping on receive
- Processes received pause packets
- Supports reception of longer VLAN type frames
- Supports MII management control writes and reads with MII PHYs
- Programmable interframe gap
- Provides counters and interrupts for many error conditions

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the design specification for the 10/100 Mbs Ethernet Media Access Controller (EMAC). The EMAC incorporates the applicable features described in IEEE Std. 802.3 MII interface specification. The IEEE Std. 802.3 MII interface specification is referenced throughout this document and should be used as the authoritative specification. Differences between the IEEE Std. 802.3 MII interface specification and the Xilinx EMAC implementation are highlighted and explained in the Specification Exceptions section of this [data sheet](#).

The EMAC Interface design is a soft intellectual property (IP) core designed for implementation in a Virtex-E, Virtex-II, Spartan-II, Spartan-IIe, or Virtex-II Pro FPGA. It supports the IEEE Std. 802.3 Media Independent Interface (MII) to industry standard Physical Layer (PHY) devices and communicates to a processor via an IBM On-Chip Peripheral Bus (OPB) interface. The design provides a 10 Megabits per second (Mbps) and 100 Mbps (also known as Fast Ethernet) EMAC Interface. This design includes many of the functions and the flexibility found in dedicated Ethernet controller devices currently on the market.

The Xilinx EMAC design allows the customer to tailor the EMAC to suit their application by setting certain parameters to enable/disable features. The parameterizable features of the design are discussed in EMAC Design Parameters.

The EMAC is comprised of two IP blocks as shown in Figure 1: The IP Interface (IPIF) block is a subset of OPB bus interface features chosen from the full set of IPIF features to most efficiently couple the second block, the EMAC core, to the OPB processor bus for this packet¹ based interface (this combined entity is referred to as a device). Although there are separate specifications for the IPIF design, this specification addresses the specific implementation required for the EMAC design.

EMAC Endianness

Please note that the EMAC is designed as a big endian device (bit 0 is the most significant bit and is shown on the left of a group of bits).

1. IEEE Std. 802.3 uses the terms Frame and Packet interchangeably when referring to the Ethernet unit of transmission; this specification does likewise

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II, Spartan IIE	
Version of Core	opb_ethernet	v1.00m
Resources Used		
	Min	Max
I/O	179	179
LUTs	2018	3688
FFs	1557	2228
Block RAMs	2	16
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

The 4-bit transmit and receive data interface to the external PHY is little endian (bit 3 is the most significant bit and appears on the left of the bus). The MII management interface to the PHY is serial with the most significant bit of a field being transmitted first.

Features

The EMAC is a soft IP core designed for Xilinx FPGAs and contains the following features:

- 32-bit OPB master and slave interfaces
- Memory mapped direct I/O interface to registers and FIFOs as well as simple DMA and Scatter/Gather DMA capabilities for low processor and bus utilization
- Media Independent Interface (MII) for connection to external 10/100 Mbps PHY transceivers
 - IEEE 802.3-compliant MII
 - Supports auto-negotiable and non auto-negotiable PHYs
 - Supports 10BASE-T and 100BASE-TX/FX IEEE 802.3 compliant MII PHYs at full or half duplex
- Independent internal 2K, 4K, 8K, 16K, or 32K byte TX and RX FIFOs for holding data for more than one packet. 2K byte depth is sufficient for normal 1518 maximum byte packets but 4K byte depth provides better throughput.
- 16, 32, or 64 entry deep FIFOs for the Transmit Length, Receive Length, and Transmit Status registers to support multiple packet operation.
- CSMA/CD compliant operation at 10 Mbps and 100 Mbps in half duplex mode
- Programmable PHY reset signal
- Internal loop-back capability
- Supports unicast, multicast, and broadcast transmit and receive modes as well as promiscuous address receive mode
- Supports a "Freeze" (graceful halt) mode based on input signal assertion to assist with emulator based software development
- Provides auto or manual source address field insertion or overwrite for transmission
- Provides auto or manual pad and Frame Check Sequence (FCS) field insertion
- Provides auto pad and FCS field stripping on receive
- Processes received pause packets
- Supports reception of longer VLAN type frames
- Supports MII management control writes and reads with MII PHYs
- Programmable interframe gap
- Provides counters and interrupts for many error conditions

Click [here](#) to view this data sheet

Product Overview

Introduction

The Ethernet Lite MAC described in this document designed incorporating the applicable features described in IEEE Std. 802.3 MII interface specification, which should be used as the authoritative specification. Differences between the IEEE Std. 802.3 MII interface specification and the Xilinx EMAC Lite implementation are highlighted and explained in the Specification Exceptions section.

The EMAC Interface design is a soft intellectual property (IP) core designed for implementation in a Virtex™-E, Virtex-II, Spartan™-II, Spartan-IIE or Virtex™-II Pro FPGA.

The EMAC Lite supports the IEEE Std. 802.3 Media Independent Interface (MII) to industry standard Physical Layer (PHY) devices and communicates to a processor via an IBM On-Chip Peripheral Bus (OPB) interface. The design provides a 10 Megabits per second (Mbps) and 100 Mbps (also known as Fast Ethernet) Interface. The goal is to provide the minimal functions necessary to provide an Ethernet interface with the least resources used.

The Ethernet Lite MAC is comprised of two IP blocks: The IP Interface (IPIF) block is a subset of OPB bus interface features chosen from the full set of IPIF features to most efficiently couple the second block, the Ethernet Lite MAC core, to the OPB processor bus for this packet based interface (this combined entity is referred to as a device).

Although there are separate specifications for the IPIF design, this specification addresses the implementation required for the Ethernet Lite MAC design.

Features

- 32-bit OPB slave interface
- Memory mapped direct I/O interface to the transmit and receive data dual port memory
- Media Independent Interface (MII) for connection to external 10/100 Mbps PHY transceivers
 - IEEE 802.3-compliant MII
 - Supports auto-negotiable and non auto-negotiable PHYs
 - Supports 10BASE-T and 100BASE-TX/FX IEEE 802.3 compliant MII PHYs at full or half duplex
- Independent internal 2K byte TX and RX dual port memory for holding data for one packet each

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro, Virtex II, Virtex, Virtex E, Spartan II, Spartan IIE	
Version of Core	opb_ethernetlite	v1.00a
Resources Used		
	Min	Max
I/O	124	124
LUTs	494	610
FFs	293	362
Block RAMs	2	8
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	N/A	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	N/A	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

- CSMA/CD compliant operation at 10 Mbps and 100 Mbps in half duplex mode
- Supports unicast, and broadcast transmit and receive modes
- Provides auto Frame Check Sequence (FCS) field insertion on transmit and validation on receive

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The OPB_ATMC Design described in this document is designed to incorporate the features defined in *UTOPIA Level 2, v1.0, af-phy-0039.000*, written by the ATM Forum Technical Committee, June, 1995.

The UTOPIA Level 2, v1.0 document is referenced throughout this document and is the authoritative specification. Differences between the UTOPIA Level 2, v1.0 document and the Xilinx OPB_ATMC Design implementation are highlighted and explained in the Specification Exceptions section of this [data sheet](#).

Features

- UTOPIA Level 2 master or slave interface
- UTOPIA interface data path of 8 or 16 bits
- Interface throughput up to 622 Mbps (OC12)
- Single channel VPI/VCI service and checking in received cells
- Header error check (HEC) generation and checking
- Parity generation and checking
- IP interface frequency of 10 MHz to 40 MHz
- System operating frequency up to 125 MHz through OPB interface
- OPB interface including register, FIFO, DMA, and scatter gather capabilities
- Statistics gathering of short cells, long cells, unknown VPI/VCI, parity errors, and HEC errors
- Selectively prepend headers to transmit cells
- Selectively pass entire received cells or payloads only
- Selectively transfer 48 byte ATM payloads only
- Loop back test mode
- Auto processing or discard of short received cells, parity errored cells, unknown VPI/VCI, or HEC errored cells

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II	
Version of Core	opb_atmc	v1.00b
Resources Used		
	Min	Max
I/O	36	52
LUTs	1700	3300
FFs	1350	2150
Block RAMs	2	2
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The OPB_ATMC Design described in this document is designed to incorporate the features defined in *UTOPIA Level 2, v1.0, af-phy-0039.000*, written by the ATM Forum Technical Committee, June, 1995.

The UTOPIA Level 2, v1.0 document is referenced throughout this document and is the authoritative specification. Differences between the UTOPIA Level 2, v1.0 document and the Xilinx OPB_ATMC Design implementation are highlighted and explained in the Specification Exceptions section of this [data sheet](#).

Features

- UTOPIA Level 2 master or slave interface
- UTOPIA interface data path of 8 or 16 bits
- Interface throughput up to 622 Mbps (OC12)
- Single channel VPI/VCI service and checking in received cells
- Header error check (HEC) generation and checking
- Parity generation and checking
- IP interface frequency of 10 MHz to 40 MHz
- System operating frequency up to 125 MHz through OPB interface
- OPB interface including register, FIFO, DMA, and scatter gather capabilities
- Statistics gathering of short cells, long cells, unknown VPI/VCI, parity errors, and HEC errors
- Selectively prepend headers to transmit cells
- Selectively pass entire received cells or payloads only
- Selectively transfer 48 byte ATM payloads only
- Loop back test mode
- Auto processing or discard of short received cells, parity errored cells, unknown VPI/VCI, or HEC errored cells

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II	
Version of Core	opb_atmc	v2.00a
Resources Used		
	Min	Max
I/O	36	52
LUTs	1500	3000
FFs	1300	2000
Block RAMs	2	2
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the design specification for the OPB High Level Data Link Control (HDLC) Interface Intellectual Property (IP) solution. It defines the architecture and interface requirements to this module. This includes registers the user must initialize for proper operation. This module is compatible with {ITU Q.921}. See the Specification Exceptions of this [data sheet](#).

The Xilinx HDLC design allows the customer to tailor the HDLC to suit their application by setting certain parameters to enable/disable features. The parameterizable features of the design are discussed in HDLC Design Parameters.

Features

- Support for a single independent full duplex HDLC channel
- Receive memory buffer of selectable depth
- Transmit FIFO of selectable depth
- Selectable 8/16 bit address receive address detection
- Selectable receive frame address discard
- Selectable receive broadcast address detection. Broadcast address = 0xFF
- Selectable 16 bit (CRC-CCITT) or 32 bit (CRC-32) frame check sequence
- 16 bit CRC error counter
- 16 bit Aborted frame counter
- Multiple Interrupts including:
 - Rx FCS error interrupt
 - Rx frame alignment error interrupt
 - FIFO overrun/underrun interrupts
 - Interrupt generated when either error counter rolls over
- Tx frame abort control
- Memory mapped direct I/O interface to registers and FIFOs as well as DMA and Scatter/Gather DMA capabilities for low processor and bus utilization.
- 16 entry deep FIFOs for the Transmit Length, Receive Length, Transmit Status and Receive Status registers to support multiple packet operation.
- Flag sharing between back to back frames

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II™ Pro, Virtex™ II, VirtexE, Virtex, Spartan™ II, Spartan IIE	
Version of Core	opb_hdlc	v1.00b
Resources Used		
	Min	Max
I/O	6	6
LUTs	472	2388
FFs	445	1413
Block RAMs	0	2
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	N/A	
Design Tool Requirements		
Xilinx Implementation Tools	Design Manager	
Verification	N/A	
Simulation	N/A	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document describes the specifications for a 32-bit free-running timebase and watchdog timer core for the On-Chip Peripheral Bus (OPB). The TimeBase WatchDog Timer (TBWDT) is a 32-bit peripheral that attaches to the OPB

Features

- OPB V2.0 bus interface with byte-enable support
- Supports 32-bit, 16-bit, and 8-bit bus interfaces
- Watchdog timer (WDT) with selectable timeout period and interrupt
- Configurable WDT enable: enable-once or enable-disable
- One 32-bit free-running timebase counter with rollover interrupt

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II	
Version of Core	opb_timebase_wdt	v1.00a
Resources Used		
	Min	Max
Slices	N/A	N/A
LUTs	63	63
FFs	111	111
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document describes the specifications for the General Purpose Input/Output (GPIO) core for the On-Chip Peripheral Bus (OPB) bus. The GPIO is a 32-bit peripheral that attaches to the OPB.

Features

- OPB v2.0 bus interface with byte-enable support
- Supports 32-bit, 16-bit, and 8-bit bus interfaces
- Each GPIO bit dynamically programmable as input or output
- Number of GPIO bits configurable up to size of data bus interface
- Can be configured as inputs-only to reduce resource utilization

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II	
Version of Core	opb_gpio	v1.00a
Resources Used		
	Min	Max
Slices	22	104
LUTs	8	49
FFs	31	193
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document describes the specifications for a timer/counter core for the On-Chip Peripheral Bus (OPB).

The TC (Timer/Counter) is a 32-bit timer module that attaches to the OPB.

Features

- OPB v2.0 bus interface with byte-enable support
- Supports 32-bit bus interface
- Two programmable interval timers with interrupt, event generation, and event capture capabilities
- Configurable counter width
- One Pulse Width Modulation (PWM) output
- Freeze input for halting counters during software debug

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II	
Version of Core	opb_timer	v1.00b
Resources Used		
	Min	Max
Slices	99	200
LUTs	99	275
FFs	105	266
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the design specification for the MicroBlaze™ Debug Module (MDM). The MDM core enables JTAG based debugging of one or more MicroBlaze processors.

Features

- Support for JTAG based software debug tools
- Support for debugging a configurable number of MicroBlaze processors
- Support for synchronized control of multiple processors - stop and single step
- Support for a JTAG based UART with an OPB interface
- Based on BSCAN logic in Xilinx FPGAs.

Core Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II, Spartan II E	
Version of Core	opb_mdm	v1.00c
Resources Used		
	Min	Max
Slices	67	163
LUTs	45	283
FFs	79	167
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	N/A	
Design Tool Requirements		
Xilinx Implementation Tools	ISE 5.2i or higher	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or higher	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The OPB Central DMA Controller provides simple Direct Memory Access (DMA) services for peripherals and memory devices on the OPB bus. The controller moves a programmable quantity of data from a source address to a destination address without processor intervention.

Features

- Provides a single physical channel of Direct Memory Access between a source address and a destination address.
- Provides programmable registers for transfer length, source address, destination address, and dataword size.
- Addresses may be set up as incrementing or non-incrementing (for supporting keyhole type memory devices))
- Byte, halfword, and word data sizes supported.
- Provides fast internal data buffer to support OPB burst transfers

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	opb_central_dma	v1.00a
Resources Used		
	Min	Max
Slices	180	208
LUTs	296	346
FFs	271	282
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats		
Constraints File		
Verification		
Instantiation Template		
Reference Designs		
Design Tool Requirements		
Xilinx Implementation Tools		
Verification		
Simulation		
Synthesis		
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

CFIFO contains separate write (transmit) and read (receive) FIFO designs called WFIFO and RFIFO, respectively. WFIFO and RFIFO can be used together or separately, and both are built from common functional elements such as special low level counter circuitry and compare functions. A CFIFO is intended to reside within other cores which require a multichannel FIFO capability, such as an HDLC transmitter / receiver. CFIFO does not contain a host bus interface, instead relying on core instantiation to provide interface. FIFOs utilize Virtex BRAM elements as data storage medium. CFIFO design incorporates special purpose counters, state machines, and logic necessary to implement functional requirements of a channelized FIFO.

Features

- Two independent channel FIFO components provided: Read CFIFO (for host bus receive data buffering) and Write CFIFO (for host bus transmit data buffering).
- User controlled features include parameters for:
 - Setting the number of channels
 - Setting FIFO data depth.
 - Setting FIFO data width.
 - Setting independent fixed length burst sizes for each side of a CFIFO. Setting size to zero removes burst transfer support logic from side and disables bursts for side of CFIFO.
 - Selecting target FPGA family type.
- "FIFO like" status outputs on communications interface side of HalfFull, AlmostFull, and Full for Read Channel FIFO, and HalfEmpty, AlmostEmpty, and Empty for Write Channel FIFO, in addition to true 'Occupancy' (Write Channel FIFO) and 'Vacancy' (Read Channel FIFO) outputs.
- A Tag field input to Write CFIFO is stored in memory array on every write data transfer. This field is user definable and is intended to provide a mechanism for indicating which byte within a word is last valid byte in a packet.
- Write and Read Ports on each CFIFO are synchronized to a common clock source (synchronous operation).

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, VirtexE, Virtex, Spartan™ 3, Spartan IIE	
Version of Core	channel_fifo	v1.00a
Resources Used		
	Min	Max
I/O		
LUTs	199	345
FFs	61	96
Block RAMs	18	36
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	UCF	
Verification	VHDL Testbench	
Instantiation Template	VHDL Wrapper	
Reference Designs	N/A	
Design Tool Requirements		
Xilinx Implementation Tools	ISE 5.1 or later	
Verification	ModelSim PE 5.6d	
Simulation	ModelSim PE 5.6d	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document describes the specifications for a core for fixed interval interrupts. The fit_timer core is a peripheral that generates a strobe signal at fixed intervals and is not attached to any bus.

Features

- Configurable number of clock cycles between strobes
- Configurable inaccuracy in clock intervals between strobes

Core Facts		
Core Specifics		
Supported Device Family	Virtex II Pro, Virtex II	
Version of Core	fit_timer	v1.00a
Resources Used		
	Min	Max
Slices	6	11
LUTs	6	19
FFs	10	19
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	N/A	
Design Tool Requirements		
Xilinx Implementation Tools	ISE 5.2i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or higher	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

The MII_to_RMII design described in this document provides the Reduced Media Independent Interface between ethernet PHYs and Xilinx ethernet cores such as the OPB_Ethernet. The OPB_Ethernet provides the traditional Media Independent Interface (MII) that requires sixteen signals to communicate with an Ethernet PHY. The MII_to_RMII accepts the sixteen signal MII interface and provides a six signal interface to an RMII compliant PHY. Additionally, a fixed 50 MHz reference clock synchronizes the MII_to_RMII with both interfaces. This MII_to_RMII follows the specification defined by the RMII Consortium found on the internet site http://broadband.spirentcom.com/technology/chipsolutions/rmii_1_2.pdf.

The Xilinx MII_to_RMII design allows the customer to tailor their application by setting certain parameters to enable or disable features.

Features

The MII_to_RMII is a soft IP core designed for Xilinx FPGAs and contains the following features:

- MII Interface
- RMII Interface
- Parameters to select fixed 10 or 100 Mbit per second throughput
- Parameter to allow auto detection of receive throughput (transmit side always fixed)
- A fixed clock frequency of 50 MHz.

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Spartan-3, Virtex-II, Virtex-II Pro	
Version of Core	mii_to_rmii	v1.00a
Resources Used		
	Min	Max
I/O	25	25
LUTs	16	145
FFs	20	146
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	UCF	
Verification	VHDL Testbench	
Instantiation Template	VHDL Wrapper	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.2i or later	
Verification	ModelSim PE 5.5e or later	
Simulation	ModelSim PE 5.5e or later	
Synthesis	Synplify Pro 7.1	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the design specification for the 1 Gbs Ethernet Media Access Controller (GEMAC) with DMA. The GEMAC described in this document has been designed incorporating the applicable features described in IEEE Std. 802.3-2000. Differences between that specification and the Xilinx GEMAC implementation are highlighted and explained in the Specification Exceptions section.

The GEMAC Interface design is a soft intellectual property (IP) core designed for implementation in a Virtex™-II, or Virtex-II Pro™ FPGA. The GEMAC supports the IEEE Std. 802.3 Gigabit Media Independent Interface (GMII) to industry standard Physical Layer (PHY) devices for full and half duplex applications.

For full duplex designs in Virtex-II or Virtex-II Pro devices, including the optional Physical Coding Sublayer (PCS) function allows the GEMAC to support the standard Ten Bit Interface (TBI) to external PHY devices. For full duplex designs in Virtex-II Pro devices, including the optional Physical Media Attachment (PMA) function with the PCS function allows the GEMAC to take advantage of the built-in Multi-Gigabit Transceivers (MGT) for a greatly reduced signal count SerDes interface to external transceivers. This option greatly reduces routing complexity in the Printed Wiring Board (PWB).

The GEMAC communicates to a processor via a 64 bit IBM Processor Local Bus (PLB) interface. The design provides a 1 Gigabit per second (Gbps) full or half duplex Ethernet Interface. The Xilinx GEMAC design allows the customer to tailor the GEMAC to suit their application by setting certain parameters to enable/disable features. The parameterizable features of the design are discussed in GEMAC Design Parameters.

The GEMAC is comprised of two, three, or four IP blocks as shown in Figure 1: The IP Interface (IPIF) block is a subset of PLB bus interface features chosen from the full set of IPIF features to most efficiently couple the second block, the GEMAC core, to the PLB processor bus for this packet based interface. The optional third (PCS) and fourth (PMA) blocks provide flexibility for connection to external Ethernet physical layer devices. This combined entity is referred to as a device. Although there are separate specifications for the IPIF design, this specification addresses the specific implementation required for the GEMAC design.

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex	
Version of Core	plb_gemac	v1.00a
Resources Used		
	Min	Max
I/O	457	474
LUTs	3879	5356
FFs	3494	3438
Block RAMs	8	38
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Features

The GEMAC is a soft IP core designed for Xilinx FPGAs and contains the following features:

- 64-bit PLB master and slave interfaces.
- Memory mapped direct I/O interface to registers and FIFOs as well as simple DMA and Scatter/Gather DMA capabilities for low processor and bus utilization.
- Optional Media Independent Interface Management (MIIM) for access to PHY transceiver registers
- GMII interface to external PHY devices
- Optional full duplex PCS function with Ten Bit Interface (TBI) to external PHY devices.
- Option full duplex PCS/PMA functions with SerDes interface to external transceiver devices for reduced signal count
- Independent internal 2K, 4K, 8K, 16K, or 32K byte TX and RX FIFOs for holding data for more than one packet (2K byte depth is sufficient for normal 1518 maximum byte packets but 4K byte depth provides better throughput. 16K or 32K byte depth is required for Jumbo frames up to 9K bytes long)
- 16 entry deep FIFOs for the Transmit Length, Receive Length, and Transmit Status registers to support multiple packet operation
- CSMA/CD compliant operation in half duplex mode with automatic storage and retransmission of packet data for transmit collisions
- Filtering of "bad" receive packets to reduce processor bus utilization
- Programmable PHY reset signal
- Auto pad and Frame Check Sequence (FCS) field insertion or pass through on transmit
- Auto source address field insertion, overwrite, or pass through on transmit
- Auto pad and FCS field stripping or pass through on receive
- Processes transmission and reception of Pause frames for flow control
- Supports receive and transmit of longer VLAN type frames
- Programmable interframe gap
- Provides interrupts for many error and status conditions
- Optional support of jumbo frames up to 9K bytes in length
- Optional statistics gathering
- Unicast, Broadcast, Multicast (up to 16 addresses stored), and Promiscuous address recognition modes for receive packets for flexibility and to reduce processor bus utilization
- Supports carrier extension and frame bursting as needed for half-duplex mode

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the design specification for the 1 Gbs Ethernet Media Access Controller (GEMAC) with DMA. The GEMAC described in this document has been designed incorporating the applicable features described in IEEE Std. 802.3-2000. Differences between that specification and the Xilinx GEMAC implementation are highlighted and explained in the Specification Exceptions section.

The GEMAC Interface design is a soft intellectual property (IP) core designed for implementation in a Virtex-II or Virtex-II Pro FPGA. The GEMAC supports the IEEE Std. 802.3 Gigabit Media Independent Interface (GMII) to industry standard Physical Layer (PHY) devices for full duplex only applications.

For designs in Virtex-II or Virtex-II Pro devices, including the optional Physical Coding Sublayer (PCS) function allows the GEMAC to support the standard Ten Bit Interface (TBI) to external PHY devices. For designs in Virtex-II Pro devices, including the optional Physical Media Attachment (PMA) function with the PCS function allows the GEMAC to take advantage of the built-in Multi-Gigabit Transceivers (MGT) for a greatly reduced signal count SerDes interface to external transceivers. This option greatly reduces routing complexity in the Printed Wiring Board (PWB).

The GEMAC communicates to a processor via a 64-bit IBM Processor Local Bus (PLB) interface, which provides a 1 Gigabit per second full duplex only Ethernet Interface.

The Xilinx GEMAC design allows the customer to tailor the GEMAC to suit their application by setting certain parameters to enable/disable features. The parameterizable features of the design are discussed in this data sheet.

The GEMAC is comprised of two, three, or four IP blocks as shown in **Figure 1**: The IP Interface (IPIF) block is a subset of PLB bus interface features chosen from the full set of IPIF features to most efficiently couple the second block, the GEMAC core, to the PLB processor bus for this packet based interface. The optional third (PCS) and fourth (PMA) blocks provide flexibility for connection to external Ethernet physical layer devices. This combined entity is referred to as a device. Although there are separate specifications for the IPIF design, this specification addresses the specific implementation required for the GEMAC design.

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II	
Version of Core	plb_gemac	v1.00b
Resources Used		
	Min	Max
I/O		
LUTs		
FFs		
Block RAMs		
Provided with Core		
Documentation	Click here to view this data sheet	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

GEMAC Endianness

Please note that the GEMAC is designed as a big endian device (bit 0 is the most significant bit and is shown on the left of a group of bits).

The 8-bit GMII transmit and receive data interface to the external PHY is little endian (bit 7 is the most significant bit and appears on the left of the bus). The MII management interface to the PHY is serial with the most significant bit of a field being transmitted first.

Features

The GEMAC is a soft IP core designed for Xilinx FPGAs and contains the following features:

- 64-bit PLB master and slave interfaces.
- Memory mapped direct I/O interface to registers and FIFOs as well as Simple DMA and Scatter/Gather DMA capabilities for low processor and bus utilization..
- Optional Media Independent Interface Management (MIIM) for access to PHY transceiver registers
- GMII interface to external PHY devices
- Optional PCS function with Ten Bit Interface (TBI) to external PHY devices.
- Option PCS/PMA functions with SerDes interface to external transceiver devices for reduced signal count
- Independent internal 2K, 4K, 8K, 16K, or 32K byte TX and RX FIFOs for holding data for more than one packet (2K byte depth is sufficient for normal 1518 maximum byte packets but 4K byte depth provides better throughput. 16K or 32K byte depth is required for Jumbo frames up to 9K bytes long)
- 16 entry deep FIFOs for the Transmit Length, Receive Length, and Transmit Status registers to support multiple packet operation
- Filtering of "bad" receive packets to reduce processor bus utilization
- Programmable PHY reset signal
- Auto pad and Frame Check Sequence (FCS) field insertion or pass through on transmit
- Auto pad and FCS field stripping or pass through on receive
- Processes transmission and reception of Pause frames for flow control
- Supports receive and transmit of longer VLAN type frames
- Programmable interframe gap
- Provides interrupts for many error and status conditions
- Optional support of jumbo frames up to 9K bytes in length
- No receive destination address validation. All properly formed packets are accepted.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the specification for the PLB Universal Asynchronous Receiver/Transmitter (UART) Intellectual Property (IP).

The UART described in this document has been designed incorporating the features described in *National Semiconductor PC16550D UART with FIFOs* data sheet (June, 1995), (<http://www.national.com/pf/PC/PC16550D.html>).

The National Semiconductor PC16550D data sheet is referenced throughout this document and should be used as the authoritative specification. Differences between the National Semiconductor implementation and the OPB UART Point Design implementation are highlighted and explained in this [data sheet](#).

Features

- Hardware and software register compatible with all standard 16450 and 16550 UARTs
- Implements all standard serial interface protocols
 - 5, 6, 7, or 8 bits per character
 - Odd, Even, or no parity detection and generation
 - 1, 1.5, or 2 stop bit detection and generation
 - Internal baud rate generator and separate receiver clock input
 - Modem control functions
 - False start bit detection and recovery
 - Prioritized transmit, receive, line status, and modem control interrupts
 - Line break detection and generation
 - Internal loop back diagnostic functionality
 - Independent 16 word transmit and receive FIFOs
- Registers
 - Receiver Buffer Register (Read Only)
 - Transmitter Holding Register (Write Only)
 - Interrupt Enable Register
 - Interrupt Identification Register (Read Only)
 - FIFO Control Register (Read/Write)
 - Line Control and Line Status Registers
 - Modem Control and Modem Status Registers
 - Scratch Register

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II	
Version of Core	plb_uart16550	v1.00b
Resources Used		
	Min	Max
I/O	538	538
LUTs	650	650
FFs	463	463
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

- Divisor Latch (least and more significant byte)
- System clock frequency of 100 MHz

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the specification for the PLB Universal Asynchronous Receiver/Transmitter (UART) Intellectual Property (IP).

The UART described in this document has been designed incorporating the features described in *National Semiconductor PC16550D UART with FIFOs* data sheet (June, 1995), (<http://www.national.com/pf/PC/PC16550D.html>).

The National Semiconductor PC16550D data sheet is referenced throughout this document and should be used as the authoritative specification. Differences between the National Semiconductor implementation and the OPB UART Point Design implementation are highlighted and explained in this [data sheet](#).

Features

- Hardware and software register compatible with all standard 16450 and 16550 UARTs
- Implements all standard serial interface protocols
 - 5, 6, 7, or 8 bits per character
 - Odd, Even, or no parity detection and generation
 - 1, 1.5, or 2 stop bit detection and generation
 - Internal baud rate generator and separate receiver clock input
 - Modem control functions
 - False start bit detection and recovery
 - Prioritized transmit, receive, line status, and modem control interrupts
 - Line break detection and generation
 - Internal loop back diagnostic functionality
 - Independent 16 word transmit and receive FIFOs
- Registers
 - Receiver Buffer Register (Read Only)
 - Transmitter Holding Register (Write Only)
 - Interrupt Enable Register
 - Interrupt Identification Register (Read Only)
 - FIFO Control Register (Read/Write)
 - Line Control and Line Status Registers
 - Modem Control and Modem Status Registers
 - Scratch Register

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II	
Version of Core	plb_uart16550	v1.00c
Resources Used		
	Min	Max
Slices	538	538
LUTs	650	650
FFs	463	463
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

- Divisor Latch (least and more significant byte)
- System clock frequency of 100 MHz

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the specification for the PLB Universal Asynchronous Receiver/Transmitter (UART) Intellectual Property (IP).

The UART described in this document has been designed incorporating the features described in *National Semiconductor PC16550D UART with FIFOs* data sheet (June, 1995), (<http://www.national.com/pf/PC/PC16550D.html>).

The National Semiconductor PC16550D data sheet is referenced throughout this document and should be used as the authoritative specification. Differences between the National Semiconductor implementation and the OPB UART Point Design implementation are highlighted and explained in this [data sheet](#).

Features

- Hardware and software register compatible with all standard 16450 UARTs
- Implements all standard serial interface protocols
 - 5, 6, 7, or 8 bits per character
 - Odd, Even, or no parity detection and generation
 - 1, 1.5, or 2 stop bit detection and generation
 - Internal baud rate generator and separate receiver clock input
 - Modem control functions
 - False start bit detection and recovery
 - Prioritized transmit, receive, line status, and modem control interrupts
 - Line break detection and generation
 - Internal loop back diagnostic functionality
- Registers
 - Receiver Buffer Register (Read Only)
 - Transmitter Holding Register (Write Only)
 - Interrupt Enable Register
 - Interrupt Identification Register (Read Only)
 - Line Control and Line Status Registers
 - Modem Control and Modem Status Registers
 - Scratch Register
 - Divisor Latch (least and more significant byte)
- System clock frequency of 100 MHz

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II	
Version of Core	plb_uart16450	v1.00b
Resources Used		
	Min	Max
Slices	432	432
LUTs	487	487
FFs	410	410
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the specification for the PLB Universal Asynchronous Receiver/Transmitter (UART) Intellectual Property (IP).

The UART described in this document has been designed incorporating the features described in *National Semiconductor PC16550D UART with FIFOs* data sheet (June, 1995), (<http://www.national.com/pf/PC/PC16550D.html>).

The National Semiconductor PC16550D data sheet is referenced throughout this document and should be used as the authoritative specification. Differences between the National Semiconductor implementation and the OPB UART Point Design implementation are highlighted and explained in this [data sheet](#).

Features

- Hardware and software register compatible with all standard 16450 UARTs
- Implements all standard serial interface protocols
 - 5, 6, 7, or 8 bits per character
 - Odd, Even, or no parity detection and generation
 - 1, 1.5, or 2 stop bit detection and generation
 - Internal baud rate generator and separate receiver clock input
 - Modem control functions
 - False start bit detection and recovery
 - Prioritized transmit, receive, line status, and modem control interrupts
 - Line break detection and generation
 - Internal loop back diagnostic functionality
- Registers
 - Receiver Buffer Register (Read Only)
 - Transmitter Holding Register (Write Only)
 - Interrupt Enable Register
 - Interrupt Identification Register (Read Only)
 - Line Control and Line Status Registers
 - Modem Control and Modem Status Registers
 - Scratch Register
 - Divisor Latch (least and more significant byte)
- System clock frequency of 100 MHz

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II	
Version of Core	plb_uart16450	v1.00c
Resources Used		
	Min	Max
Slices	432	432
LUTs	487	487
FFs	410	410
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document presents the design specification for the Xilinx PLB RapidIO™ LVDS Intellectual Property (IP) solution. This LogiCORE™ module provides an interface between the IBM® CoreConnect™ Processor Local Bus (PLB) and an LVDS based RapidIO interface standard.

The PLB RapidIO LVDS design provides an interface between the PPC405 (via PLB CoreConnect Bus) and a RapidIO protocol network. The physical interface to the RapidIO bus uses the 8 bit LVDS standard.

Features

The PLB RapidIO LVDS is a soft IP core designed for Xilinx FPGAs incorporating PPC405 and MicroBlaze processing elements. The design provides the following features:

- Front end Interface to the IBM CoreConnect PLB Bus
 - Supports PLB signaling per the IBM *64-Bit Processor Local Bus, Architectural Specification*
 - Integrates easily with the Xilinx Platform Studio for PPC405 System Development.
 - 64 bit wide data transfers
 - 512x64 Tx and Rx Packet Buffers (Up to 8 maximally sized packets can be queued for Tx and Rx)
 - PLB Cacheline and Burst Transfer Interface Support with Packet Buffers.
 - Parameterized System Address Block.
- Back end Interface to RapidIO Bus.
 - Incorporates Xilinx RapidIO Physical Layer
 - Supports *RapidIO Physical Layer 8/16 LP-LVDS Interconnect Specification v1.1*.
 - 8-bit LVDS PHY (TX and Rx functions)
 - 500 MBytes/sec Peak Transfer Rate at the PHY Tx and Rx ports.
- Processor Accessible Registers
 - Interrupt Enable and Status Registers
 - S/W Reset/MIR Register
 - RapidIO PHY Link Status Register
 - RapidIO PHY Management Register Set
- System Interrupt Support
 - Tx and Rx Flow Control Interrupts

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II,	
Version of Core	plb_rapidio_lvds	v1.00a
Resources Used		
	Min	Max
I/O	40	40
LUTs	5849	6138
FFs	2960	3089
Block RAMs	4	4
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	Example UCF	
Verification	N/A	
Instantiation Template	None	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

- Programmable Enables/Disables.
- Interrupt Status Registers can support S/W Polled Mode control flow in place of Interrupt Control Flow
- PLB System clock frequency up to 100 MHz

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

This document provides the specification for the PLB_ATMC IP, which includes an asynchronous transfer mode controller with a UTOPIA Level 2 or UTOPIA Level 3 interface.

The PLB_ATMC Design described in this document is designed to incorporate the features defined in *UTOPIA Level 2, Version 1.0, af-phy-0039.000*, written by the ATM Forum Technical Committee, June, 1995 or in *UTOPIA Level 3 Physical Layer Interface, af-phy-0136.000*, written by the ATM Forum Technical Committee, November, 1999.

The UTOPIA Level 2 and 3 documents are referenced throughout this document and are the authoritative specifications. Differences between these documents and the Xilinx PLB_ATMC Design implementation are highlighted and explained in Specification Exceptions.

Features

The PLB_ATMC Design is a soft IP core designed for Xilinx FPGAs and contains the following features:

- UTOPIA Level 2 or UTOPIA Level 3
- UTOPIA master or slave interface for either level
- UTOPIA interface data path of 8 or 16 bits for level 2; and 8, 16 or 32 bits for level 3
- Interface throughput up to 622 Mbps (OC12) for 16 bit UTOPIA Level 2; and up to 2.4 Gbps (OC48) for 32 bit UTOPIA Level 3
- Single channel VPI/VCI service and checking in received cells
- Header error check (HEC) generation and checking
- Parity generation and checking
- IP interface frequency of 10 MHz to 100 MHz
- System operating frequency up to 100 MHz through PLB interface
- PLB interface including register and FIFO capabilities
- Statistics gathering of errored cells
- Selectively prepend headers to transmit cells
- Selectively pass entire received cells or payloads only
- Selectively transfer 48 byte ATM payloads only
- Loop back test mode
- Auto processing or discard of errored cells

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II	
Version of Core	plb_atmc	v1.00a
Resources Used		
	Min	Max
I/O	36	84
LUTs	1500	4000
FFs	1300	2600
Block RAMs	2	4
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

A DCR Interrupt Controller is composed of a bus-centric wrapper containing the IntC core and a bus interface. The IntC core is a simple, parameterized interrupt controller that, along with the appropriate bus interface, attaches to either the OPB (On-chip Peripheral Bus) or the DCR (Device Control Register) Bus.

It can be used in either embedded PowerPC systems (Virtex-II Pro devices), or in MicroBlaze™ soft processor systems. There are two versions of the DCR Interrupt Controller, one with an OPB interface, called OPB IntC, and another with a DCR interface called DCR IntC.

In this document, IntC and DCR IntC are used interchangeably to refer to functionality or interface signals common to all variations of the DCR Interrupt Controller. When the discussion switches to a bus centric version, the interrupt controller is referred to as OPB IntC or DCR IntC.

Features

- Modular design provides a core interrupt controller functionality instantiated within a bus interface design (currently the OPB and DCR buses are supported)
- OPB v2.0 bus interface with byte-enable support (IBM SA-14-2528-01 64-bit On-Chip Peripheral Bus Architecture Specifications, v2.0)
- DCR v2.0 bus interface (IBM SA-14-2525-00 32-bit Device Control Register Bus Architecture Specifications, v2.9)
- Supports data bus widths of 8-bits, 16-bits, or 32-bits for OPB interface, and 32-bits for DCR interface
- Number of interrupt inputs is configurable up to the width of the data bus
- Interrupt controllers can be easily cascaded to provide additional interrupt inputs
- Interrupt Enable Register for selectively disabling individual interrupt inputs
- Master Enable Register for disabling the interrupt request output
- Each input is configurable for edge or level sensitivity—edge sensitivity can be configured for rising or falling; level sensitivity can be active-high or -low
- Automatic edge synchronization when inputs are

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	dcr_intc	v1.00a
Resources Used		
	Min	Max
I/O	70	70
LUTs	41	73
FFs	18	198
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

- configured for edge sensitivity
- Output interrupt request pin configurable for edge or level generation—edge generation configurable for rising or falling; level generation configurable for active-high or -low

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Click [here](#) to view this data sheet

Product Overview

Introduction

A DCR Interrupt Controller is composed of a bus-centric wrapper containing the IntC core and a bus interface. The IntC core is a simple, parameterized interrupt controller that, along with the appropriate bus interface, attaches to either the OPB (On-chip Peripheral Bus) or the DCR (Device Control Register) Bus.

It can be used in either embedded PowerPC systems (Virtex-II Pro devices), or in MicroBlaze™ soft processor systems. There are two versions of the DCR Interrupt Controller, one with an OPB interface, called OPB IntC, and another with a DCR interface called DCR IntC.

In this document IntC and DCR IntC are used interchangeably to refer to functionality or interface signals that are common to all variations of the DCR Interrupt Controller. When the discussion switches to a bus centric version, then the interrupt controller will be referred to as OPB IntC or DCR IntC.

Features

- Modular design provides core interrupt controller functionality instantiated within a bus interface design (currently OPB and DCR buses supported)
- DCR v2.0 bus interface (IBM SA-14-2525-00 32-bit DCR Bus Architecture Specifications, v2.9)
- Supports data bus width of 32-bits for DCR interface
- Number of interrupt inputs is configurable up to the width of the data bus
- Interrupt controllers can be easily cascaded to provide additional interrupt inputs
- Interrupt Enable Register for selectively disabling individual interrupt inputs
- Master Enable Register for disabling the interrupt request output
- Each input is configurable for edge or level sensitivity; edge sensitivity can be configured for rising or falling; level sensitivity can be active-high or -low
- Automatic edge synchronization when inputs are configured for edge sensitivity
- Output interrupt request pin is configurable for edge or level generation — edge generation configurable for rising or falling; level generation configurable for active-high or -low

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex II Pro™, Virtex™ II, Virtex, Virtex E, Spartan™ II	
Version of Core	dcr_intc	v1.00b
Resources Used		
	Min	Max
I/O	76	107
LUTs	71	424
FFs	55	334
Block RAMs	0	0
Provided with Core		
Documentation	Click here to view this data sheet.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Part II: Software

This section contains information on the following:

Chapter 7 , [“Device Driver Programmer Guide”](#)

Chapter 8, [“Tornado 2.0 BSP User Guide”](#)

Chapter 9, [“Device Driver Summary”](#)

Chapter 10, [“Automatic Generation of Tornado 2.x \(VxWorks 5.x\) Board Support Packages”](#)

Device Driver Programmer Guide

Overview

This document describes the Xilinx device driver environment, and includes information on the following:

- Design and implementation details for using the drivers
- Device driver architecture
- Application Programmer Interface (API) conventions
- Scheme for configuring the drivers to work with reconfigurable hardware devices
- Infrastructure that is common to all device drivers.

Goals and Objectives

The Xilinx device drivers are designed to meet the following goals and objectives:

- Provide maximum portability

The device drivers are provided as ANSI C source code. ANSI C was chosen to maximize portability across processors and development tools. Source code is provided both to aid customers in debugging their applications as well as allow customers to modify or optimize the device driver if necessary.

A layered device driver architecture additionally separates device communication from processor and Real Time Operating System (RTOS) dependencies, thus providing portability of core device driver functionality across processors and operating systems.

- Support FPGA configurability

Since FPGA-based devices can be parameterized to provide varying functionality, the device drivers must support this varying functionality. The configurability of device drivers should be supported at compile-time and at run-time. Run-time configurability provides the flexibility needed for future dynamic system reconfiguration.

In addition, a device driver supports multiple instances of the device without code duplication for each instance, while at the same time managing unique characteristics on a per instance basis.

- Support simple and complex use cases

Device drivers are needed for simple tasks such as board bring-up and testing, as well as complex embedded system applications. A layered device driver architecture provides both simple device drivers with minimal memory footprints and more robust, full-featured device drivers with larger memory footprints.

- Ease of use and maintenance

Xilinx makes use of coding standards and provides well-documented source code in order to give developers (i.e., customers and internal development) a consistent view

of source code that is easy to understand and maintain. In addition, the API for all device drivers is consistent to provide customers a similar look and feel between drivers.

Device Driver Architecture

The architecture of the device drivers is designed as a layered architecture as shown in the following figure. The layered architecture accommodates the many use cases of device drivers while at the same time providing portability across operating systems, toolsets, and processors. The layered architecture provides seamless integration with an RTOS (Layer 2), high-level device drivers that are full-featured and portable across operating systems and processors (Layer 1), and low-level drivers for simple use cases (Layer 0). The following paragraphs describe each of the layers. The user can choose to use any and all layers.

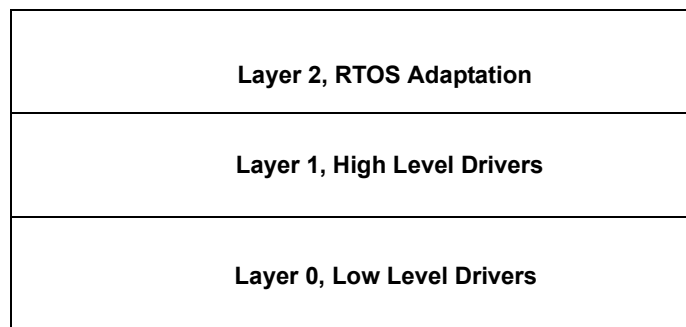


Figure 7-1: Layered Architecture

Layer 2, RTOS Adaptation

This layer consists of adapters for device drivers. An adapter converts a Layer 1 device driver interface to an interface that matches the requirements of the device driver scheme for an RTOS. Unique adapters may be necessary for each RTOS. Adapters typically have the following characteristics.

- Communicates directly to the RTOS and the Layer 1, high-level driver.
- References functions and identifiers specific to the RTOS. This layer is therefore not portable across operating systems.
- Can use memory management
- Can use RTOS services such as threading, inter-task communication, etc.
- Can be simple or complex depending on the RTOS interface and requirements for the device driver

Layer 1, High Level Drivers

This layer consists of high level device drivers . They are implemented as macros and functions and are designed to allow a developer to utilize all features of a device. These high-level drivers are independent of operating system and processor, making them highly portable. They typically have the following characteristics.

- Consistent and high-level (abstract) API that gives the user an "out-of-the-box" solution
- No RTOS or processor dependencies, making them highly portable
- Run-time error checking such as assertion of input arguments. Also provides the ability to compile away asserts.

- Comprehensive support of device features
- Abstract API that isolates the API from hardware device changes
- Supports device configuration parameters to handle FPGA-based parameterization of hardware devices.
- Supports multiple instances of a device while managing unique characteristics on a per instance basis.
- Polled and interrupt driven I/O
- Non-blocking function calls to aid complex applications
- May have a large memory footprint
- Typically provides buffer interfaces for data transfers as opposed to byte interfaces. This makes the API easier to use for complex applications.
- Does not communicate directly to Layer 2 adapters or application software. Utilizes asynchronous callbacks for upward communication.

Layer 0, Low Level Drivers

This layer consists of low level device drivers. They are implemented as macros and functions and are designed to allow a developer to create a small system, typically for internal memory of an FPGA. They typically have the following characteristics.

- Simple, low-level API
- Small memory footprint
- Little to no error checking is performed
- Supports primary device features only
- Minimal abstraction such that the API typically matches the device registers. The API is therefore less isolated from hardware device changes.
- No support of device configuration parameters
- Supports multiple instances of a device with base address input to the API
- None or minimal state is maintained
- Polled I/O only
- Blocking functions for simple use cases
- Typically provides byte interfaces but can provide buffer interfaces for packet-based devices.

Object-Oriented Device Drivers

In addition to the layered architecture, it is important that the user understand the underlying design of the device drivers. The device drivers are designed using an object-oriented methodology. The methodology is based upon components and is described in the following paragraphs. This approach pertains particularly to the Layer 1, high-level device drivers.

Component Definition

A component is a logical partition of the software which provides a functionality similar to one or more classes in C++. Each component provides a set of functions that operate on the internal data of the component. In general, components are not allowed access to the data of other components. A device driver is typically designed as a single component. A component may consist of one or more files.

Component Implementation

The component contains data variables which define the set of values that instances of that type can hold and a set of functions that operate on those data variables. Components must utilize the functions of other components in order to access the data of other components,

rather than accessing component data directly. Components provide data abstraction and encapsulation by gathering the state of an object and the functions that operate on that object into a single unit and by denying direct access to its data members.

Component Data Variables

The primary mechanism for implementing a component in C is the structure. The data variables for a component are grouped in a single structure such that instances of the component each have their own data. The structure and the prototypes for all component functions are declared in the header file which is shared between the implementing component and other components which utilize it. A pointer to this structure, referred to as the instance pointer, is passed into each function of the component which operates on the instance data.

Component Interface

Each component has a set of functions which are collectively referred to as the component interface. Every function of a component which operates on the instance data utilizes a pointer, named InstancePtr, to an instance of a component as the first argument. This argument emulates the *this* pointer in C++ and allows the component function to manipulate the instance data.

Component Instance

An instance of a component is created when a variable is created using the component data type. An instance of a component maps to each physical hardware device. Each instance may have unique characteristics such as its memory mapped address and specific device capabilities.

Component Example

The following code example illustrates a device driver component.

```
/* the device component data type */

typedef struct
{
    Xuint32 BaseAddress; /* component data variables */
    Xuint32 IsReady;
    Xuint32 IsStarted;
} XDevice;

/* create an instance of a device */

XDevice DeviceInstance;

/* device component interfaces */

XStatus XDevice_Initialize(XDevice *InstancePtr, Xuint16 DeviceId);
XStatus XDevice_Start(XDevice *InstancePtr);
```

API and Naming Conventions

External Identifiers

External identifiers are defined as those items that are accessible to all other components in the system (global) and include functions, constants, typedefs, and variables.

An 'X' is prepended to each Xilinx external so it does not pollute the global name space, thus reducing the risk of a name conflict with application code. The names of externals are based upon the component in which they exist. The component name is prepended to each

external name. An underscore character always separates the component name from the variable or function name.

External Name Pattern:

```
X<component name>_VariableName;
X<component name>_FunctionName(ArgumentType Argument)
X<component name>_TypeName;
```

Constants are typically defined as all uppercase and prefixed with an abbreviation of the component name. For example, a component named XUartLite (for the UART Lite device driver) would have constants that begin with XUL_, and a component named XEmac (for the Ethernet 10/100 device driver) would have constants that begin with XEM_. The abbreviation utilizes the first three uppercase letters of the component name, or the first three letters if there are only two uppercase letters in the component name.

File Naming Conventions

The file naming convention utilizes long file names and is not limited to 8 characters as imposed by the older versions of the DOS operating system.

Component Based Source File Names

Source file names are based upon the name of the component implemented within the source files such that the contents of the source file are obvious from the file name. All file names must begin with the lowercase letter "x" to differentiate Xilinx source files. File extensions .h and .c are utilized to distinguish between header source files and implementation source files.

Implementation Source Files (*.c)

The C source files contain the implementation of a component. A component is typically contained in multiple source files to allow parts of the component to be user selectable.

Source File Naming Pattern:

```
x<component name>.c                main source file
x<component name>_functionality.c  secondary source file
```

Header Source Files (*.h)

The header files contain the interfaces for a component. There will always be external interfaces which is what an application that utilizes the component invokes.

- The external interfaces for the high level drivers (Layer 1) are contained in a header file with the file name format *x<component name>.h*.
- The external interfaces for the low level drivers (Layer 0) are contained in a header file with the file name format *x<component name>_l.h*.

In the case of multiple C source files which implement the class, there may also be a header file which contains internal interfaces for the class. The internal interfaces allow the functions within each source file to access functions in the another source file.

- The internal interfaces are contained in a header file with the file name format *x<component name>_i.h*.

Device Driver Layers

Layer 1 and Layer 0 device drivers (i.e., high-level and low-level drivers) are typically bundled together in a directory. The Layer 0 device driver files are named *x<component name>_l.h* and *x<component name>_l.c*. The "_l" indicates low-level driver. Layer 2 RTOS adapter files include the word "adapter" in the file name, such as *x<component name>_adapter.h* and *x<component name>_adapter.c*. These are typically stored in a different directory name (e.g., one specific to the RTOS) than the device driver files.

Example File Names

The following source file names illustrates an example which is complex enough to utilize multiple C source files.

xuartns550.c	Main implementation file
xuartns550_intr.c	Secondary implementation file for interrupt handling
xuartns550.h	High level external interfaces header file
xuartns550_i.h	Internal identifiers header file
xuartns550_l.h	Low level external interfaces header file
xuartns550_l.c	Low level implementation file
xuartns550_g.c	Generated file controlling parameterized instances

and,

xuartns550_sio_adapter.c VxWorks Serial I/O (SIO) adapter

High Level Device Driver API

High level device drivers are designed to have an API which includes a standard API together with functions that may be unique to that device. The standard API provides a consistent interface for Xilinx drivers such that the effort to use multiple device drivers is minimized. An example API follows.

Standard Device Driver API

Initialize

This function initializes an instance of a device driver. Initialization must be performed before the instance is used. Initialization includes mapping a device to a memory-mapped address and initialization of data structures. It maps the instance of the device driver to a physical hardware device. The user is responsible for allocating an instance variable using the driver's data type, and passing a pointer to this variable to this and all other API functions.

Reset

This function resets the device driver and device with which it is associated. This function is provided to allow recovery from exception conditions. This function resets the device and device driver to a state equivalent to after the Initialize() function has been called.

SelfTest

This function performs a self-test on the device driver and device with which it is associated. The self-test verifies that the device and device driver are functional.

Optional Functions

Each of the following functions may be provided by device drivers.

Start

This function is provided to start the device driver. Starting a device driver typically enables the device and enables interrupts. This function, when provided, must be called prior to other data or event processing functions.

Stop

This function is provided to stop the device driver. Stopping a device driver typically disables the device and disables interrupts.

GetStats

This function gets the statistics for the device and/or device driver.

ClearStats

This function clears the statistics for the device and/or device driver.

InterruptHandler

This function is provided for interrupt processing when the device must handle interrupts. It does not save or restore context. The user is expected to connect this interrupt handler to their system interrupt controller. Most drivers will also provide hooks, or callbacks, for the user to be notified of asynchronous events during interrupt processing (e.g., received data or device errors).

Configuration Parameters

Standard device driver API functions (of Layer 1, high-level drivers) such as Initialize() and Start() require basic information about the device such as where it exists in the system memory map or how many instances of the device there are. In addition, the hardware features of the device may change because of the ability to reconfigure the hardware within the FPGA. Other parts of the system such as the operating system or application may need to know which interrupt vector the device is attached to. For each device driver, this type of information is distributed across two files: *xparameters.h* and *x<component name>.g.c*.

Typically, these files are automatically generated by a system generation tool based on what the user has included in their system. However, these files can be hand coded to support internal development and integration activities. Note that the low-level drivers of Layer 0 do not require or make use of the configuration information defined in these two files. Other than the memory-mapped location of the device, the low-level drivers are typically fixed in the hardware features they support.

xparameters.h

This source file centralizes basic configuration constants for all drivers within the system. Browsing this file gives the user an overall view of the system architecture. The device drivers and Board Support Package (BSP) utilize the information contained here to configure the system at runtime. The amount of configuration information varies by device, but at a minimum the following items should be defined for each device:

- Number of device instances
- Device ID for each instance

A Device ID uniquely identifies each hardware device which maps to a device driver. A Device ID is used during initialization to perform the mapping of a device driver to a hardware device. Device IDs are typically assigned either by the user or by a system generation tool. It is currently defined as a 16-bit unsigned integer.

- Device base address for each instance
- Device interrupt assignment for each instance if interrupts can be generated.

File Format and Naming Conventions

Every device must have the following constant defined indicating how many instances of that device are present in the system (note that `<component name>` does not include the preceding "X"):

```
XPAR_X<component name>_NUM_INSTANCES
```

Each device instance will then have multiple, unique constants defined. The names of the constants typically match the hardware configuration parameters, but can also include other constants. For example, each device instance has a unique device identifier

(DEVICE_ID), the base address of the device's registers (BASEADDR), and the end address of the device's registers (HIGHADDR).

```
XPAR_<component name>_<component instance>_DEVICE_ID
XPAR_<component name>_<component instance>_BASEADDR
XPAR_<component name>_<component instance>_HIGHADDR
```

<component instance> is typically a number between 0 and (XPAR_X<component name>_NUM_INSTANCES - 1). Note that the system generation tools may create these constants with a different convention than described here. Other device specific constants are defined as needed:

```
XPAR_<component name>_<component instance>_<item description>
```

When the device specific constant applies to all instances of the device:

```
XPAR_<component name>_<item description>
```

For devices that can generate interrupts, a separate section within *xparameters.h* is used to store interrupt vector information. While the device driver implementation files do not utilize this information, their RTOS adapters, BSP files, or user application code will require them to be defined in order to connect, enable, and disable interrupts from that device. The naming convention of these constants varies whether an interrupt controller is part of the system or the device hooks directly into the processor.

For the case where an interrupt controller is considered external and part of the system, the naming convention is as follows:

```
XPAR_INTC_<instance>_<component name>_<component instance>_VEC_ID
```

Where INTC is the name of the interrupt controller component, <instance> is the component instance of the INTC, <component name> and <component instance> is the name and instance number of the component connected to the controller. Of course XPAR_INTC must have the other required constants DEVICE_ID, BASEADDR, etc. This convention supports single and cascaded interrupt controller architectures.

For the case where an interrupt controller is considered internal to a processor, the naming convention changes:

```
XPAR_<proc name>_<component name>_<component instance>_VEC_ID
```

Where <proc name> is the name of the processor.

x<component name>_g.c

The header file *x<component name>.h* defines the type of a configuration structure. The type will contain all of the configuration information necessary for an instance of the device. The format of the data type is as follows:

```
typedef struct
{
    Xuint16 DeviceID;
    Xuint32 BaseAddress;

    /* Other device dependent data attributes */

} X<component name>_Config;
```

The implementation file *x<component name>_g.c* defines an array of structures of X<component name>_Config type. Each element of the array represents an instance of the device, and contains most of the per-instance XPAR constants from *xparameters.h*.

Example

To help illustrate the relationships between these configuration files, an example is presented that contains a single interrupt controller whose component name is INTC and

a single UART whose component name is (UART). Only xintc.h and xintc_g.c are illustrated, but xuart.h and xuart_g.c would be very similar.

xparameters.h

```

/* Constants for INTC */
XPAR_INTC_NUM_INSTANCES      1
XPAR_INTC_0_DEVICE_ID        21
XPAR_INTC_0_BASEADDR         0xA0000100

/* Interrupt vector assignments for this instance */
XPAR_INTC_0_UART_0_VEC_ID    0

/* Constants for UART */
XPAR_UART_NUM_INSTANCES      1
XPAR_UART_0_DEVICE_ID        2
XPAR_UART_0_BASEADDR         0xB0001000

```

xintc.h

```

typedef struct
{
    Xuint16 DeviceID;
    Xuint32 BaseAddress;
} XIntc_Config;

```

xintc_g.c

```

static XintcConfig[XPAR_INTC_NUM_INSTANCES] =
{
    {
        XPAR_INTC_0_DEVICE_ID,
        XPAR_INTC_0_BASEADDR,
    }
}

```

Common Driver Infrastructure

Source Code Documentation

The comments in the device driver source code contain *doxygen* tags for *javadoc*-style documentation. *Doxygen* is a *javadoc*-like tool that works on C language source code. These tags typically start with "@" and provide a means to automatically generate HTML-based documentation for the device drivers. The HTML documentation contains a detailed description of the API for each device driver.

Driver Versions

Some device drivers may have multiple versions. Device drivers are usually versioned when the API changes, either due to a significant hardware change or simply restructuring of the device driver code. The version of a device driver is only indicated within the comment block of a device driver file. A modification history exists at the top of each file and contains the version of the driver. An example of a device driver version is "1.00b", where 1 is the major revision, 00 is the minor revision, and b is a subminor revision. The hardware device and its device driver must match major and minor revisions in order to be compatible.

Currently, the user is not allowed to link two versions of the same device driver into their application. The versions of a device driver use the same function and file names, thereby preventing them from being linked into the same link image. As multiple versions of

drivers are supported, the version name will be included in the driver file names, as in *x<component>_v1_00_a.c*.

Primitive Data Types

The primitive data types provided by C are minimized by the device drivers because they are not guaranteed to be the same size across processor architectures. Data types which are size specific are utilized to provide portability and are contained in the header file *xbasic_types.h*.

Device I/O

The method by which I/O devices are accessed varies between processor architectures. In order for the device drivers to be portable, this difference is isolated such that the driver for a device will work for many microprocessor architectures with minimal changes. A device I/O component, XIo, in *xio.c* and *xio.h* source files, contains functions and/or macros which provide access to the device I/O and are utilized for portability.

Error Handling

Errors that occur within device drivers are propagated to the application. Errors can be divided into two classes, synchronous and asynchronous. Synchronous errors are those that are returned from function calls (either as return status or as a parameter), so propagation of the error occurs when the function returns. Asynchronous errors are those that occur during an asynchronous event, such as an interrupt and are handled through callback functions.

Return Status

In order to indicate an error condition, functions which include error processing return a status which indicates success or an error condition. Any other return values for such functions are returned as parameters. Error codes are standardized in a 32-bit word and the definitions are contained in the file *xstatus.h*.

Asserts

Asserts are utilized in the device drivers to allow better debugging capabilities. Asserts are used to test each input argument into a function. Asserts are also used to ensure that the component instance has been initialized.

Asserts may be turned off by defining the symbol NDEBUG before the inclusion of the header file *xbasic_types.h*.

The assert macro is defined in *xbasic_types.h* and calls the function XAssert when an assert condition fails. This function is designed to allow a debugger to set breakpoints to check for assert conditions when the assert macro is not connected to any form of I/O.

The XAssert function calls a user defined function and then enters an endless loop. A user may change the default behavior of asserts such that an assert condition which fails does return to the user by changing the initial value of the variable XWaitInAssert to XFALSE in *xbasic_types.c*. A user defined function may be defined by initializing the variable XAssertCallbackRoutine to the function in *xbasic_types.c*.

Communication with the Application

Communication from an application to a device driver is implemented utilizing standard function calls. Asynchronous communication from a device driver to an application is accomplished with callbacks using C function pointers. It should be noted that callback functions are called from an interrupt context in many drivers. The application function called by the asynchronous callback must minimize processing to communicate to the application thread of control.

Reentrancy and Thread Safety

The device drivers are designed to be reentrant, but may not be thread-safe due to shared resources.

Interrupt Management

The device drivers use device-specific interrupt management rather than processor-specific interrupt management.

Multi-threading & Dynamic Memory Management

The device drivers are designed without the use of multi-threading and dynamic memory management. This is expected to be accomplished by the application or by an RTOS adapter.

Cache & MMU Management

The device drivers are designed without the use of cache and MMU management. This is expected to be accomplished by the application or by an RTOS adapter.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
06/28/02	1.0	Xilinx initial release.
7/02/02	1.1	Made IP Spec # conditional text and removed ML reference.

Tornado 2.0 BSP User Guide

General Overview

The purpose of this document is to provide an introduction to the Tornado 2.0 BSP as implemented in BSPs generated by the EDK and on handcoded BSPs implemented on selected Virtex-II Pro reference boards. The first part of this document goes over general design principles and APIs utilized. The second part goes over specific BSPs details for the ML300 and Insight MDFG456 reference boards.

The addition of the Chip Support Package (CSP) into a Tornado 2.0 BSP is a unique challenge because of the nature of how easily hardware is added and removed from the FPGA using System Build Generator and how difficult it is to accommodate this feature into a Tornado 2.0 BSP. The CSP is a part of the BSP in that it provides the software drivers for hardware IP utilized by the BSP and application code. The CSP is designed to be primarily operating system independent so in many respects it is segregated and independently configured from the BSP.

The reader is expected to have a working knowledge in these areas:

- Tornado 2.0 BSPs
- "C" programming language

Requirements

Tornado 2.0.2

The user should have Wind River Tornado 2.0.2 installed on their PC with the PPC405 libraries.

Patches required that can be found at Wind River's Windsurf technical support web site:

- SPR67953 Cumulative patch
- DosFs 2.0 DOS file system support

SingleStep (XE)

The XE stands for Xilinx Edition. This version of the SingleStep debugger is Virtex-II Pro aware. This debugger works in concert with the VisionProbe debugger pod.

Installation

Copy the entire BSP source tree to `$WIND_BASE\config\ and perform the following operations from the DOS command-line:`

```
C:\> make clean
C:\> make release
```

When this process finishes, a new project is placed at:

```
$WIND_BASE\proj\_vx.
```

As an alternative to this procedure, the Tornado project facility can be used to create a bootable application using the given BSP as the basis BSP. When creating a project in this way, the BSP can be located anywhere. See Project Facility documentation from Wind River.

Files and Directories

While the root directory of the BSP can be placed anywhere, it is typically located at `$WIND_BASE/target/config/<bspname>`. The Tornado Project component of the BSP is located at `$WIND_BASE/target/proj/<bspname>_vx`.

The project component of the BSP is required if it will be configured/compiled with the Tornado Project Facility IDE. Normally, the Project Facility is utilized during application development and trivial BSP tweaks. The non-project component (also referred to as the command-line Tornado 1.0.1 BSP) is utilized during BSP development. Note that the methods of configuring and building the BSP differ greatly between the Project and command-line methods. See Tornado documentation for more information.

The CSP adds a directory structure not usually seen with Tornado 2.0 BSPs. It has been added to segregate BSP files from the CSP.

The following directories make up BSPs:

config/<bspname>

The traditional directory for Tornado 2.0 BSPs. Contains BSP library source code and the command-line makefile.

config/<bspname>/net

Contains Tornado Project "configlette" network source code that overrides configlettes located at `$WIND_BASE/target/config/comps/src/net`.

config/<bspname>/ace

Contains the bitstream and compact flash image which in itself contains the bootrom and other sample VxWorks ace and elf images.

config/<bspname>/ip_csp

The base directory for the CSP.

config/<bspname>/ip_csp/xsrc

Contains source code for the CSP.

proj/<bspname>_vx

The base directory for a Tornado project. All files here are maintained by the Project Facility.

proj/<bspname>_vx/<build spec>

A build specification maintained by the Project Facility. There is typically a "default" build spec here unless removed by the developer. Other build specifications can be added by the developer.

CSP Driver Organization

This section briefly discusses how the CSP is compiled and linked and eventually used by Tornado makefiles to include into the VxWorks image.

CSP drivers are implemented in "C" and can be distributed among several source files unlike traditional VxWorks drivers which consist of single "C" header and implementation files.

There are up to three components for CSP drivers:

- Driver source inclusion.

- OS independent implementation
- OS dependent implementation (optional).

"Driver source inclusion" refers to how CSP drivers are compiled. For every CSP driver, there is a file named `ip_<dev>_<version>.c`. This file `#include's` each CSP driver source file(s) (`*.c`) for the given device.

This process is analogous to how VxWorks' `sysLib.c` `#include's` source for Wind River supplied drivers. The reason why CSP files are not simply `#include'd` in `sysLib.c` like the rest of the drivers is due to namespace conflicts and maintainability issues. If all CSP files were part of a single compilation unit, static functions and data are no longer private. This places restrictions on the CSP device drivers and would negate their operating system independence.

The OS independent part of the driver is designed for use with any operating system or any processor. It provides an API that utilizes the functionality of the underlying hardware. The OS dependent part of the driver adapts the driver for use with VxWorks. Such examples are SIO drivers for serial ports, or END drivers for ethernet adapters. Not all drivers require the OS dependent drivers, nor is it required to include the OS dependent portion of the driver in the CSP build.

Configuration

These BSPs are configured just like any other Tornado 2.0 BSP. There is not much configurability to CSP drivers since the IP hardware has been pre-configured in most cases by System Build Generator. The only configuration available generally is whether the driver is included in the CSP at all. How to go about including/excluding drivers depends on whether the Project facility or the command-line method is being used to perform the configuration activities.

Note that simply by including a CSP device driver does not mean that driver will be automatically utilized. Most CSP drivers with VxWorks adapters have initialization code. In some cases the user may be required to add the proper driver initialization function calls to the BSP.

Command-Line

A set of constants (one for each driver) are defined in `config/ML300/ip_config.h` and follow the format:

```
#define INCLUDE_<XDRIVER>
```

This file is included near the top of `config/ML300/config.h`. By default all drivers are included in the build. To exclude a driver, add the following line in `config.h` after the `#include "ip_config.h"` statement.

```
#undef INCLUDE_<XDRIVER>
```

This will prevent the driver from being compiled and linked into the build. To re-instate the driver, remove the `#undef` line from `config.h`. Some care is required for certain drivers. For example, Ethernet may require that a DMA driver be present. undefining the DMA driver will cause the build to fail.

Project Facility

The Project Facility is part of the Tornado IDE. It is a GUI driven environment. To add/delete CSP drivers, go to the VxWorks pane in the workspace window (see figure below). Then add/delete driver components under `IP_CSP` just as you would with any other VxWorks component.

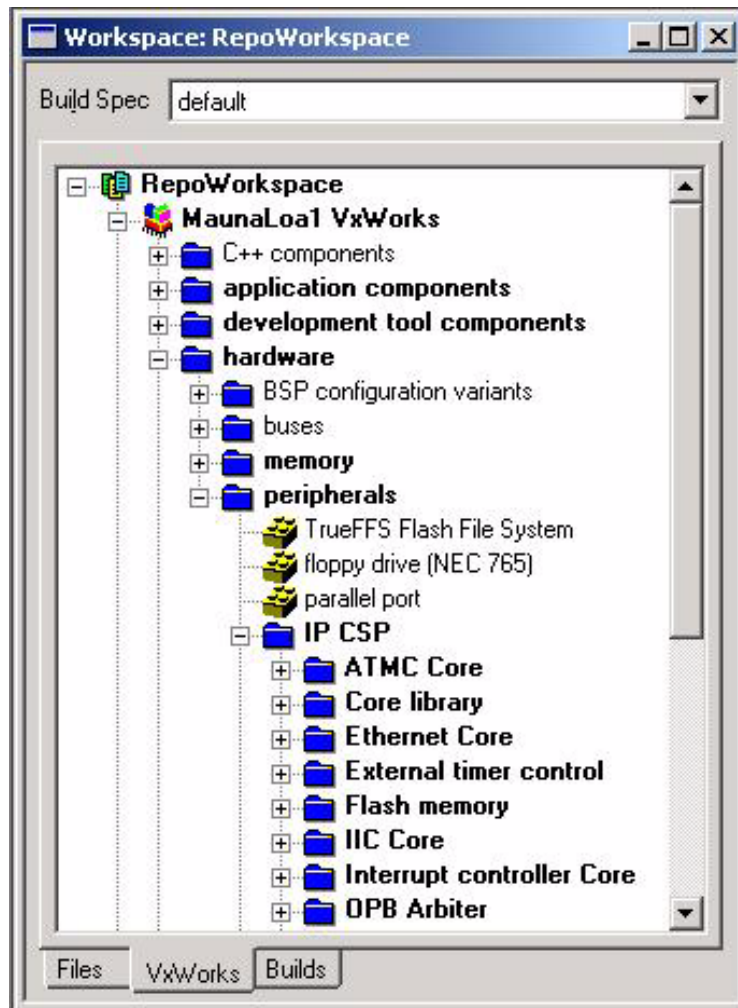


Figure 8-1: Project Facility GUI Configuration

Note that whatever configuration has been specified in `ip_config.h` and `config.h` will be overridden by the project facility.

These BSPs follow the standard Tornado conventions when it comes to creating VxWorks images. Refer to Tornado documentation on how to make a VxWorks image. The next section discusses extensions made to the build process.

Building VxWorks

Command-Line BSP Build Extensions

The CSP is compiled/linked with the same toolchain VxWorks is built with. Minor additions to the `Makefile` were required to help Tornado find the location of CSP source code files.

Project BSP Build Extensions

There are no extensions to the Project build.

Bootup Sequence

There are many variations of VxWorks images with some based in RAM, some in ROM. Depending on board design, not all these images are supported. The following list discusses various image types:

- ROM compressed images - These images begin execution in ROM and decompress the BSP image into RAM, then transfer control to the decompressed image in RAM. This image type is not compatible with SystemACE because SystemACE doesn't know the image is compressed and will dutifully place it in RAM at an address that will be overwritten by the decompression algorithm when it begins. It may be possible to get this type of image to work if modifications are made to the standard Tornado makefiles to handle this scenario.
- RAM based images - These images are loaded into RAM by a bootloader, SystemACE, or an emulator. These images are fully supported.
- ROM based images - These images begin execution in ROM, copy themselves to RAM then transfer execution to RAM. In designs with SystemACE as the bootloader, the image is automatically copied to RAM. The handcoded BSP examples short-circuit the VxWorks copy operation so that the copy does not occur again after control is transferred to RAM by SystemACE (see `romInit.s`).
- ROM resident images - These images begin execution in ROM, copy the data section to RAM, and execution remains in ROM. In systems with only a SystemACE, this image is not supported. Theoretically BRAM could be used as a ROM, however, the current Virtex-II Pro parts being used in evaluation boards do not have the capacity to store a VxWorks image which could range in size from 200KB to over 700KB.

"vxWorks" Boot Sequence

This image is meant to be downloaded to the target RAM space. Once downloaded, the processor is setup to begin execution at function `_sysInit` (implemented in `sysALib.s`). Most of the time, the device performing the download will do this automatically as it can extract the entry point from the image.

1. `_sysInit`: Low level initialization. Since this image is copied to RAM, the device that downloaded the image may have to perform manual system initialization to make RAM operational. When completed, this function will setup the initial stack and invoke the first "C" function `usrInit()`.
2. `usrInit()`: Performs pre-kernel initialization. Invokes `sysHwInit()` implemented in `sysLib.c` to place the HW in a quiescent state. When completed, this function will call `kernelInit()` to bring up the VxWorks kernel. This function will in turn invoke `usrRoot()` as the first task.
3. `usrRoot()`: Performs post-kernel initialization. Hooks up the system clock, initializes the TCP/IP stack, etc. Invokes `sysHwInit2()` implemented in `sysLib.c` to attach and enable HW interrupts. When complete, `usrRoot()` invokes user application startup code `usrAppInit()` if so configured in the BSP.

"bootrom_uncmp" Boot Sequence using SystemACE

This image is ROM based but in reality it is linked to execute out of RAM addresses. While executing from ROM, this image uses relative addressing to perform tasks before jumping to RAM. This image behaves differently than a traditional `bootrom_uncmp` due to the fact it is already in RAM when control is passed to it (via System ACE).

1. Power on. System ACE loads the bitstream into the FPGA then loads the `bootrom` image into RAM and passes control to assembly language function `_romInit` located in `romInit.s`.

2. `_romInit`: Traditionally this function would perform board level initialization then call `romInit()` which would copy the VxWorks image to RAM. Since the image is already in RAM, this function simply jumps to assembly function `_sysInit`.
3. Follows steps 1, 2 & 3 of the "vxWorks" bootup sequence.

Difference Between Command-Line & Project BSPs

Functions `usrInit()`, `usrRoot()`, and `romStart()` as explained in the boot sequence steps above are implemented by Tornado. In command line BSPs, these functions are defined in source code located at `$WIND_BASE/target/config/all`. In Project BSPs, the Project Facility generates this code in the user's project directory.

Functions `_sysInit`, `_romInit`, `sysHwInit()`, and `sysHwInit2()` are implemented by the BSP with various source code files in `config/<bspname>`. These functions are utilized on both the command-line and project BSPs.

System ACE

The System ACE controller is a device that provides a way to store multiple FPGA bitstream loads. These loads are stored in a DOS FAT formatted compact flash (CF) device and downloaded by the System ACE controller into the FPGA when the system is powered up. Additionally, these bitstreams can contain a software load that is downloaded to RAM after the FPGA's IP cores have been programmed. Since a DOS filesystem is used, regular files can be accessed from the CF as well. Such files include VxWorks ELF images, application code & data, and script files.

BSPs equipped with the EDK System ACE core and drivers utilize the controller in two ways. First as a boot device and second as an external storage device. Together these features provide a very flexible storage mechanism.

DOS File System

When being used as a file storage device, the BSP will mount the CF as a DOS FAT disk partition using Wind River's DosFs2.0 add-on. To get the required VxWorks libraries into the image, the following packages must be `#define'd` in `config.h` or by the Project Facility:

- `INCLUDE_DOSFS_MAIN`
- `INCLUDE_DOSFS_FAT`
- `INCLUDE_DISK_CACHE`
- `INCLUDE_DISK_PART`
- `INCLUDE_DOSFS_DIR_FIXED`
- `INCLUDE_DOSFS_DIR_VFAT`
- `INCLUDE_CBIO`

Mounting the DOS FAT File System

Programatically, an application can mount the file system using the following API calls:

```
FILE *fp;

sysSystemAceInitFS();
if (sysSystemAceMount("/cf0", 1) != OK)
{
    /* handle error */
}

fp = fopen("/cf0/myfile.dat", "r");
.
.
```

Automounting the DOS FAT File System

To automatically mount the System ACE as a file system at boot time, `INCLUDE_XSYSACE_AUTOMOUNT` must be defined. In the Project facility, this is defined by enabling the automount feature in the System ACE folder. When defined, two more constants are utilized to mount the compact flash device: `SYSACE_AUTOMOUNT_POINT` and `SYSACE_AUTOMOUNT_PARTITION`. In the Project facility, these constants can be set by editing the System ACE properties folder. This relieves the application from having to initialize and mount the DOS File system. Note that this works only for Project builds. Command line builds require that the application invoke `sysSystemAceInitFS()` and `sysSystemAceMount()`. These functions are described in the Board API section below.

Bootroms

The bootrom is a scaled down VxWorks image that operates in much the same way a PC BIOS does. Its primary job is to find and boot a full VxWorks image. The full VxWorks image may reside on disk, in flash memory, or on some host via the Ethernet. The bootrom must be compiled in such a way that it has the ability to retrieve the full image. If the image is retrieved on the Ethernet, then the bootrom must have the TCP/IP stack compiled in, if the image is on disk, then the bootrom must have disk access support compiled in, etc. The bootroms do little else than retrieve and start the full image and maintain a bootline. The bootline is a text string that set certain user characteristics such as the target's IP address if using Ethernet and the file path to the VxWorks image to boot.

Bootroms are not a requirement. They are typically used in a development environment.

Creating Bootroms

At a command shell in the `<bspname>` directory, issue the following command:

```
make bootrom_uncmp
```

to create an uncompressed bootrom image (required for SystemACE), or

```
make bootrom
```

to create a compressed image suitable for placing in a flash memory array.

Bootrom Display

Upon cycling power, if the bootroms are working correctly, output similar to the following should be seen on the console serial port:

```
VxWorks System Boot
```

```
Copyright 1984-1998 Wind River Systems, Inc.
```

```
CPU: ML300 VirtexII Pro PPC405 Rev D
```

```
Version: 5.4.2
```

```
BSP version: 1.2/0
```

```
Creation date: July 26 2002, 12:51:32
```

```
Press any key to stop auto-boot...
```

```
3
```

```
[VxWorks Boot]:
```

Typing the "help" at this prompt lists the available commands.

Bootline

The bootline is a text string that defines user servicable characteristics such as the IP address of the target board and how to find a vxWorks image to boot. The bootline is maintained at runtime by the bootrom and is typically kept in some non-volatile (NVRAM) storage area of the system such as an EEPROM or flash memory. If there is no NVRAM, or an error occurs reading it, then the bootline is hard-coded with `DEFAULT_BOOT_LINE` defined in the BSP's `config.h` source code file. In brand new systems where NVRAM has not been initialized, then the bootline may be gibberish.

The bootline can be changed if the auto-boot countdown sequence is interrupted by entering a character on the console serial port. The "c" command can then be used to interactively edit the bootline. Enter "p" to view the bootline. On a non-bootrom image, you can still change the bootline by entering the `bootChange` command at a host or target shell prompt.

The following list goes over the meanings of the bootline fields:

- `boot device` : Device to boot from. This could be Ethernet, or a local disk. Note that when changing the bootline, the unit number may be shown appended to this field ("`xemac0`" or "`sysace=10`") when prompting for the new boot device. This number can be ignored.
- `processor number` : Always 0 with single processor systems.
- `host name` : Name as needed.
- `file name` : The VxWorks image to boot.
- `inet on ethernet (e)` : The IP internet address of the target. If there is no network interface, then this field can be left blank.
- `host inet (h)` : The IP internet address of the host. If there is no network interface, then this field can be left blank.
- `user (u)` : Username for host file system access. Pick whatever name suites you. Your ftp server must be setup to allow this user access to the host file system.
- `ftp password (pw)` : Password for host file system access. Pick whatever name suites you. Your ftp server must be setup to allow this user access to the host file system.
- `flags (f)` : For a list of options, enter the "help" command at the [VxWorks Boot]: prompt.
- `target name (tn)` : Name as needed. Set per network requirements.
- `other (o)` : This field is useful when you have a non-Ethernet device as the boot device. When this is the case, VxWorks will not start the network when it boots. Specifying an Ethernet device here will enable that device at boot time with the network parameters specified in the other bootline fields.
- `inet on backplane (b)` : Typically left blank if the target system is not on a VME or PCI backplane.
- `gateway inet (g)` : Enter an IP address here if you have to go through a gateway to reach the host computer. Otherwise leave blank.
- `startup script (s)` : Path to a file on the host computer containing shell commands to execute once bootup is complete. Leave blank if not using a script.

Examples:

```
SystemACE resident script: /cf0/vxworks/scripts/myscript.txt
Host resident script:      c:/temp/myscript.txt
```

Booting from SystemACE

If the target system has a SystemACE and the Xilinx EDK drivers are being used, then SystemACE can be setup as the boot device.

The "boot device" field of the bootline is specified using the following syntax:

sysace=<partition number>

where <partition number> is the partition to boot from. Normally, this value is set to 1, but some CF devices do not have a partition table and are formatted as if they were a large floppy disk. In this case, specify 0 as the partition number. Failure to get the partition number correct will lead to errors being reported by VxWork's dosFS libraries when the drive is mounted.

The "file name" field of the bootline is set depending on how the System ACE is to boot the system. There are two boot methods:

1. Boot from a regular file. This is similar to network booting in that the vxWorks image resides in the SystemACE compact flash storage device instead of the host file system. The compact flash device is a DOS FAT file system partition. Simply build vxWorks using the Tornado tools then copy the resulting image file to the compact flash device using a USB card reader or similar tool. Then specify that file in the "file name" field of the boot rom.

The "file name" must have the following syntax:

```
/cf0/<path/to/vxWorks/image>
```

where cf0 is the mount point. <path/to/vxWorks/image> should provide the complete path to the VxWorks image to boot. When being specified in this way, the bootrom will mount the drive as a FAT formatted disk, load the file into memory and begin execution.

Boot from an ace file. The ace file can contain HW only, SW only, or HW + SW. When booting from an ace file with HW, the FPGA is reprogrammed. If the ace file contains SW, then it is loaded into the memory, the processor's PC is set to the entry point and released to begin fetching instructions. This boot method is flexible in that a totally different HW profile can be "booted" from a VxWorks bootrom. ace files are created with the Xilinx ISI tools and is beyond the scope of this manual.

The "file name" must have the following syntax:

```
cfgaddr [x]
```

where [X] is a number between 0 and 7 that corresponds to one of the configuration directories specified in the XILINX.SYS file resident in the root directory of the compact flash device. If [X] is omitted, then the default configuration is used. The default configuration is typically selected by a rotary switch mounted somewhere on the evaluation board. The bootrom will trigger a JTAG download of the ace file pointed to by the specified config address. There should be only a single file with an .ace extension in the selected configuration directory.

In either boot scenario, if you want an Ethernet device started when the downloaded VxWorks starts, then modify the "other" field of the bootline to contain the name of the network device.

Booting from an EMAC

If the target system has an EMAC core and Xilinx EDK drivers are being used, then use the following information to boot from this device.

In the "boot device" field, use "xemac". If there is a single EMAC, then set the "unit number" to 0.

Bootline Examples

The following example boots from the ethernet using the Xilinx "xemac" as the boot device. The image booted is on the host file system on drive C.

```

boot device           : xemac
unit number          : 0
processor number     : 0
host name            : host
file name            : c:/tornado/target/config/ML300/vxWorks
inet on ethernet (e) : 192.168.0.2
host inet (h)        : 192.168.0.1
user (u)             : xemhost
ftp password (pw)    : whatever
flags (f)            : 0x0
target name (tn)     : vxtarget
other (o)            :

```

The following example boots from a file resident on the first partition of the SystemACE's compact flash device. If the file booted from /cf0/vxworks/images/vxWorks utilizes the network, then the "xemac" device is initialized.

```

boot device           : sysace=1
unit number          : 0
processor number     : 0
host name            : host
file name            : /cf0/vxworks/images/vxWorks
inet on ethernet (e) : 192.168.0.2
host inet (h)        : 192.168.0.1
user (u)             : xemhost
ftp password (pw)    : whatever
flags (f)            : 0x0
target name (tn)     : vxtarget
other (o)            : xemac

```

The following example boots from an ace file resident on the first partition of the SystemACE's compact flash device. The location of the ace file is set by XILINX.SYS located in the root directory of the compact flash device. If the ace file contains a VxWorks SW image that utilizes the network, then the "xemac" device is initialized for that image.

```

boot device           : sysace=1
unit number          : 0
processor number     : 0
host name            : host
file name            : cfgaddr2
inet on ethernet (e) : 192.168.0.2
host inet (h)        : 192.168.0.1
user (u)             : xemhost
ftp password (pw)    : whatever
flags (f)            : 0x0
target name (tn)     : vxtarget
other (o)            : xemac

```

Caches

The instruction and data caches are managed by VxWorks proprietary libraries. They are enabled by modifying the following constants in `config.h` or by using the Tornado Project facility to change the constants of the same name:

- `INCLUDE_CACHE_SUPPORT` - If `#define'd`, the VxWorks cache libraries are linked into the image. If caching is not desired, then `#undef` this constant.
- `USER_I_CACHE_ENABLE` - If `#define'd`, VxWorks will enable the instruction cache at boottime. Requires `INCLUDE_CACHE_SUPPORT` be `#define'd` to have any effect.
- `USER_D_CACHE_ENABLE` - If `#define'd`, VxWorks will enable the data cache at boottime. Requires `INCLUDE_CACHE_SUPPORT` be `#define'd` to have any effect.

MMU

If the MMU is enabled, then the cache control discussed in the previous section may not have any effect. The MMU is managed by VxWorks proprietary libraries but the initial setup is defined in the BSP. To enable the MMU, the constant `INCLUDE_MMU_BASIC` should be `#define'd` in `config.h` or by using the Project Facility. The constant `USER_D_MMU_ENABLE` and `USER_I_MMU_ENABLE` control whether the instruction and/or data MMU is utilized.

VxWorks initializes the MMU based on data in the `sysPhysMemDesc` structure defined in `sysCache.c`. Amongst other things, this table configures memory areas with the following attributes:

- Whether instruction execution is allowed.
- Whether data writes are allowed
- Instruction & data cacheability attributes.
- Translation offsets used to form virtual addresses.

The PPC405 is capable of other attributes including zone protection, however, Wind River documentation is rather deficient in this area and it is unclear whether the basic MMU package supports them. An add-on is available from Wind River for advanced MMU operations.

When VxWorks initializes the MMU, it takes the definitions from `sysPhysMemDesc` and creates page table entries (PTEs) in RAM. Each PTE describes 4KB of memory area (even though the processor is capable of representing up to 16MB per PTE) Beware that specifying large areas of memory uses substantial amounts of RAM to store the PTEs. To map 4MB of contiguous memory space takes 8KB of RAM to store the PTEs.

To increase performance with the VxWorks basic MMU package for the PPC405 processor, it may be beneficial to not enable the instruction MMU and rely on the cache control settings in the ICCR register. This strategy can dramatically reduce the number of page faults while still keeping instructions in cache. The initial setting of the ICCR is defined in the `<bspname>.h` header file.

Without the MMU enabled, the following rules apply to configuring memory access attributes and caching:

- There is no address translation, all effective addresses are physical.
- Cache control granularity is 128MB.
- The guarded attribute applies only to speculative instruction fetches on the PPC405.

Exception Handling

There are two types of exceptions which are of importance to PPC405 BSPs. The first type are internal exceptions such as machine check, illegal instruction, etc.. By default, the BSP configures VxWorks to trap these types of exceptions. When one occurs, the offending task is suspended and a descriptive message is displayed on the console. If the exception occurs in interrupt context, VxWorks will warm-reboot itself and leave a message to be displayed during the reboot. Note that if SystemACE resets the system then this message may not be available. See [Limitations, page 143](#).

The other type of exception are external asynchronous. The BSP initializes and handles these exceptions which are the result of an active signal on the external or critical interrupt pins of the processor.

With the provided example BSPs, there are two INTC IP devices within the FPGA, one connected to the processor's external interrupt and the other on the critical interrupt. Functions in BSP source code file `sysInterrupt.c` are responsible for initializing these two devices with the `XIntc` component driver and hooking them into VxWorks.

External Interrupts

Most IP peripherals that can generate interrupts are attached to the INTC component responsible for asserting the external interrupt processor exception. BSP initialization code hooks control of this device into the VxWorks `intLib` library.

External interrupt vectors are defined in `xparameters.h`. `<bspname.h>` may translate these vectors into `SYS_<device>_VEC_ID` to limit changes to BSP source code when device names change. These constants are utilized when invoking the VxWorks `intLib` functions. Example:

```
#include <intLib.h>

void foo(void)
{
    intEnable(SOME_DEVICE_VEC_ID);
}
```

Critical Interrupts

Since VxWorks does not define a critical interrupt API as it does for external interrupts, the user must utilize the API defined in `sysLibExtra.h`. Functions `sysIntCritConnect`, `sysIntCritEnable`, and `sysIntCritDisable` are designed to work identically to those for the external interrupt defined by the VxWorks `intLib.h` library. Example:

```
#include "sysLibExtra.h"

void foo(void)
{
    sysIntCritEnable(SOME_CRITICAL_DEVICE_VEC_ID);
}
```

Deviations

This section sums up the differences between garden variety BSPs and the BSPs presented in this document. The differences between the two fall roughly into key areas: CSP and System ACE support.

1. The CSP contains drivers for Xilinx IP cores (see [CSP Driver Organization, page 132](#)). To keep the BSP buildable while maintaining compatibility with the Tornado Project facility, a set of files named `ip_<driver>_<version>.c` populate the BSP directory that simply `#include` the source code from the CSP.
2. The location of the CSP relative to the BSP directory causes problems because the command line and Project facility differ in how BSP files are found during compilation. To address this issue, a key Project macro (`BSP_DIR`) is defined in the BSP's Makefile. Of all deviations, this one is the most dangerous because future versions of Tornado may cause builds to fail. The Makefile contains more information about this deviation.
3. System ACE, being a boot device and a DOS file system, has required that two VxWorks source code files found in the Tornado distribution be changed. Wind River allows BSP developers to change some source code files provided they follow set guidelines. The two files that have been modified from their original version are `bootConfig.c` and `net/ usrNetBoot.c`. These files exist in the BSP directory structure and override the Tornado versions.

`usrNetBoot.c`, used only by Project Facility builds, required a 1 line of code change to tell VxWorks that the System ACE device is a disk based system like IDE, SCSI, or floppy drives. This change allows the BSP to properly process the "other" field of the bootline (see [Bootroms, page 137](#)) when System ACE is the boot device. The "other" field allows the selection of a network device when booting from a disk based system.

`bootConfig.c`, used only by bootroms builds, required extensive modifications to

support SystemACE as a boot device. These mods are bracketed by INCLUDE_XSYSACE preprocessor ifdefs. Another mod enables the data cache when ethernet frames are copied from fifos instead of DMA. This change greatly increases the bootup times for the system but could cause problems if another device required for booting utilizes DMA or requires some sort of special cache coherency in the first 128MB of address space.

Limitations

This section goes over what limitations these BSPs impose under certain circumstances and the reasons why.

No WARM boots with SystemACE

When SystemACE is setup to download VxWorks images into RAM via JTAG, all boots are cold. This is because the System ACE controller resets the processor whenever it performs an ace download.

An effect of this could cause exception messages generated by VxWorks to not be printed to the console when the system is rebooted due to an exception in an ISR or a kernel panic. See troubleshooting guide for tips to get at this exception message.

No Compressed Images with SystemACE

If a compressed image such as "bootrom" is converted to an ace file and placed in the SystemACE's compact flash for booting, results will be undetermined. This is because System ACE cannot decompress data as it writes it to RAM.

Command line builds cannot initialize the network when System ACE is the boot device

This requires that the application provide code to initialize the network. Project builds can get around this because a modified `net/usrNetBoot.c` is provided in the BSP directory (see [Deviations, page 142](#)). The equivalent file for command line builds is located at `$WIND_BASE/target/src/config/usrNetwork.c`. The architecture of the command line build prevents us from overriding this file with a copy in the BSP directory.

Fixing `usrNetwork.c` requires changing the following code in function `usrNetInit()`:

```

    if ((strcmp (params.bootDev, "scsi", 4) == 0) ||
        (strcmp (params.bootDev, "ide", 3) == 0) ||
        (strcmp (params.bootDev, "ata", 3) == 0) ||
        (strcmp (params.bootDev, "fd", 2) == 0) ||
        (strcmp (params.bootDev, "tffs", 4) == 0))
to
    if ((strcmp (params.bootDev, "scsi", 4) == 0) ||
        (strcmp (params.bootDev, "ide", 3) == 0) ||
        (strcmp (params.bootDev, "ata", 3) == 0) ||
        (strcmp (params.bootDev, "fd", 2) == 0) ||
        (strcmp (params.bootDev, "sysace", 6) == 0) ||
        (strcmp (params.bootDev, "tffs", 4) == 0))

```

Edit this code at your own risk.

Reset Vector and SystemACE

On the PPC405 processor, the reset vector is at physical address `0xFFFFFFF0`. There is a short time window where the processor will attempt to fetch and execute the instruction at this address while SystemACE processes the ace file. The processor needs to be given something to do during this time even if it is a spin loop:

```

FFFFFFFFC b .

```

If BRAM occupies this address range, then the designer who creates the bitstream should place instructions here with the elf to BRAM utility found in the Xilinx ISI tools.

ML300 Reference BSP

This handcoded BSP was created to complement the EDK example system for the ML300 reference board. It has been customized to work with hardware devices specific to the board. The BSP has enough support to allow application development using the Tornado tool chain.

This BSP directly supports the following hardware:

- 10/100 BaseT Ethernet
- RS-232 Serial ports (2)
- System ACE
- GPIO (LEDs & pushbutton switches)
- IIC (including EEPROM, temperature & power monitors)
- 128MB DDR RAM
- 32KB BRAM
- PPC405 built-in timers, instruction cache, data cache

The BSP utilizes drivers for the following EDK IP cores:

- EMAC Ethernet: END driver type
- 16550 UART on connector P107: SIO driver type on /tyco/0 (VxWorks console)
- 16550 UART on connector P106: SIO driver type on /tyco/1
- PLB to OPB bridge, PLB arbiter
- System ACE as a JTAG device
- System ACE as a block device for disk access (FAT32)
- INTC Interrupt controllers. 1 critical controller, 1 external controller

This BSP does not support the following hardware due to the lack of support in the IP bitstream or from the lack of drivers:

- LCD Display
- PCI
- Parallel port
- PS2 ports
- USB ports
- Audio ports
- Fiber ports
- SPI

Installation

See **Installation**, page 131 for instruction on how to install this BSP.

Compact Flash & SystemACE

A compressed zipfile is provided in the ace subdirectory. This is a complete image containing a bootrom and sample VxWorks images in ace and elf file formats. See the README in the ace directory for more information.

To install this image, do the following:

1. Make a backup of your microdrive then erase all files from it.
2. Uncompress the ace/compactFlash.zip file to the microdrive.
3. Insert the microdrive into the compact flash slot on the ML300.

4. Connect a serial port cable to the P106 connector on the evaluation board. Default comm settings are 115200, N, 8, 1.
5. Set the rotary switch on the ML300 to setting 6 and apply power. At this point, the VxWorks bootrom should be running and writing to the console serial port.
6. Set the bootrom's bootline per your requirements. See [Bootroms, page 137](#) for more information.

Setting Ethernet MAC Address

To verify your MAC address is correct perform the following steps:

1. Set the rotary switch associated with the VxWorks bootrom and reboot the ML300.
2. Interrupt the countdown sequence to get the `[VxWorks Boot] :` prompt.
3. Enter the "N" command (case sensitive). The current MAC will be displayed and you will be prompted to enter a new MAC. The first three bytes of the MAC should be 000A35.

```
Press any key to stop auto-boot...
1
[VxWorks Boot]: N
Current Ethernet Address is: 00:0a:35:00:03:20
Modify only the last 3 bytes (board unique portion) of Ethernet Address.
The first 3 bytes are fixed at manufacturer's default address block.
00- 00
0a- 0a
35- 35
00-
```

If the MAC is valid, then press return three times to accept the default. On new boards, the address may be all FFs. If this is the case, enter the last three bytes that are assigned to the board's serial number. If you are not sure of the numbers, then enter return three times. This will change the MAC to 00:0a:35:FF:FF:FF. This will provide you with a valid MAC until the correct number is obtained. Boards with a MAC of all FFs will not be capable of running the network stack. Multiple boards connected to the same network with the same MAC will not work either.

Bootstrap Information

The default bitstream contains a bootstrap program that consists of a single instruction at the processor's reset vector which is in effect an endless loop, or `while(1)` in "C" programming constructs. This loop keeps the processor from running amok until SystemACE completes its download or an emulator connects to the target board. See [Reset Vector and SystemACE, page 143](#).

Memory Maps

Due to the nature of this evaluation board a full memory map is not given in this document. The user is instead referenced to "C" source code header file `xparameters.h`. This source file provides a memory map for all CSP devices. A partial map is given here.

Table 8-1: System Memory Map

Device	Start (hex)	End (hex)	Size (bytes)
PLB DDR	00000000	07FFFFFF	128 MB
OPB Space	60000000	60010000	64 KB
BRAM	FFFF8000	FFFFFFF	32 KB

RAM Memory Map

RAM device contains the VxWorks runtime image and heap space. ML300 follows VxWorks conventions for RAM usage for PowerPC processors. Refer to Appendix F of the *VxWorks 5.4 Programmer's Guide*.

Table 8-2: RAM Memory Map

Physical Address Range (hex)	Usage
00000000..000000FF	(DDR) Unused & undefined
00000100..00002FFF	(DDR) Interrupt Vector table
00003000..00010000	(DDR) VxWorks usage. Exception reason message and other VxWorks constructs are at the bottom of this region. Initial stack is set at the top of this range and grows downward. Once VxWorks has switched to multi-tasking mode, this stack is no longer used.
00010000..00BFFFFFFF	(DDR) RAM_LOW_ADRS. VxWorks image, interrupt stack, host memory pool, and heap space.
00C00000..07DFFFFFFF	(DDR) RAM_HIGH_ADRS. Two possible uses. (1) VxWorks bootrom image and heap space. (2) VxWorks heap space.
07E00000..07EFFFFFFF	(DDR) USER_RESERVED_MEM. This 1MB is used for network data buffers and network DMA descriptor spaces.
07F00000..07FFFFFFF	(DDR) USER_RESERVED_MEM. This 1 MB is not used by BSP. Available for application use
FFFF8000..FFFFFFFF	(BRAM) Address FFFFFFFFC contains reset vector.

NVRAM Memory Map

NVRAM support is provided by a Microchip Technology 24LC32A EEPROM on the IIC bus. This device provides 4KB of storage space. BSP source code file `24LC32aNvRam.c` is the driver for this device and provides the API interface required by VxWorks. The primary BSP related objects stored in NVRAM are the bootline and the Ethernet MAC address.

When there is no IIC bus support, the BSP will replace the EEPROM driver with `$WIND_BASE/src/drv/mem/nullNvRam.c` which provides only function stubs so that VxWorks will link. When this is the case, the default bootline is used (see `config.h`) and the Ethernet MAC address defaults to: `00:0a:35:00:00:00`.

Table 8-3: NVRAM Memory Map

Part Offset Range (hex)	sysNvRamGet/Set Offset	Usage
0000..07FF	N/A*	Reserved for board level objects such as the Ethernet MAC address
0800..08FF	0000..00FF	Reserved for VxWorks bootline
0900..0FEF	0100..07EF	Unused
0FF0..0FFF	07F0..07FF	Reserved

* `sysNvRamGet` and `sysNvRamSet` are the VxWorks required NVRAM interface functions. The interface they provide uses offsets relative to the bootline offset. Accessing part offsets 0000..07FF requires an alternate interface.

Caches

The caches are configured by the following constants in `ML300.h`. These constants map to the PPC cache control registers of the same name. See PPC405 documentation for further information on these registers:

- `ML300_ICCR_VAL` - Initial contents of the ICCR register (instruction cacheability attribute).
- `ML300_DCCR_VAL` - Initial contents of the DCCR register (data cacheability attribute).
- `ML300_DCWR_VAL` - Initial contents of the DCWR register (write back/through attribute).
- `ML300_SGR_VAL` - Initial contents of the SGR register (guarded attribute).

Table 8-4: Cache Map

Physical Address Range (hex)	I Cache	D Cache	Write Back/Through	Guarded
00000000..07F FFFF	Y	Y	Back	N
F8000000..FFF FFFF	Y	Y	N/A	N
everything else	N	N	N/A	N

External Interrupts

There are two INTC interrupt controller IP cores in the design. One is wired to the external interrupt signal of the PPC405 and the other to the critical interrupt signal.

PLB/OPB bridges & arbiters are wired to the critical INTC instance. If these interrupt sources are enabled and the PPC machine check interrupt is enabled then VxWorks may reboot when an exception occurs. This is because the PLB/OPB bridge/arbiter will assert their interrupt signal when they cannot complete a transaction to the INTC. This signal will propagate to the PPC as a critical interrupt exception (vector 0x100). At the same time the PPC will detect a bad bus cycle and generate a machine check exception (vector 0x200). VxWorks will begin handling the first exception, but during this time the second exception arrives. VxWorks architecturally does not allow this and will reboot the system when it occurs.

It is not recommended to `sysIntCritEnable()` one of these interrupt sources. Instead, use the VxWorks `excHookAdd()` function to use your own function perform custom exception processing (after VxWorks finishes its own processing). Here, the hook function can examine the bridges/arbiters and perform whatever task is required for the event.

IIC

There are several devices connected to the IIC bus with hardwired addresses. These addresses are defined for the BSP in the `ML300.h` header file. The BSP provides a polled interface to the IIC bus to access these devices. The interface includes a mutual exclusion semaphore that can be used to prevent more than one task from accessing the bus at a time. Before the operating system is up, the semaphore is not available and it is up to boot code to sequence access to the bus. This should not be an issue since the system is single-threaded at boot time and the only device accessed should be the NVRAM.

BSP file `sysIic.c` provides initialization, read/write primitives, and resource allocation functions.

Board API

There are a handful of "board level" BSP functions not implemented by the CSP device drivers. Prototypes for these functions are located in `config/ML300/sysLibExtra.h`.

This section will not go over CSP device driver functions. Instead the user is directed to the appropriate `ip_csp/xsrc/<device>.c` file for documentation and usage.

Standard I/O

The BSP comes with `stdin`, `stdout`, and `stderr` directed through the UART on the P106 connector. The default UART baud rate is set to 115200, no parity, 8 data bits, and 1 stop bit. The secondary UART on P107 is enabled and ready for application usage. It defaults to 19200 baud, no parity, 8 data bits, and 1 stop bit.

GPIO

Two instances of GPIO can be included in the BSP. The first instance controls the momentary push button switches and their surrounding LEDs. The second controls the 32 GPIO lines on the J10 connector. Both instances require that `INCLUDE_XGPIO` constant be defined. Each instance can be enabled or disabled with constants `INCLUDE_GPIO_LED_SWITCHES` and `INCLUDE_GPIO_TEST_PORT`.

void sysLedOn(UINT32 mask)

Turns on LEDs in the mask. Bits set to one cause the associated LED to be illuminated. The mask is built using constants `GPIO_LED_DSxx` defined in `ML300.h` where `xx` is the LED number and `DSxx` is the LED label on the PCB. This function requires that both `INCLUDE_XGPIO` and `INCLUDE_GPIO_LED_SWITCHES` be defined.

void sysLedOff(UINT32 mask)

Turns off LEDs in the mask. Bits set to one cause the associated LED to be turned off. The mask is built using constants `GPIO_LED_DSxx` defined in `ML300.h` where `xx` is the LED number and `DSxx` is the LED label on the PCB. This function requires that both `INCLUDE_XGPIO` and `INCLUDE_GPIO_LED_SWITCHES` be defined.

UINT32 sysSwitchReadState(void)

Reads the state of all the push button switches. A mask is returned describing which switches are closed (i.e. being pushed). The mask is decoded using constants `GPIO_SWITCH_SWxx` defined in `ML300.h` where `xx` is the switch number and `SWxx` is the switch label on the PCB. This function requires that both `INCLUDE_XGPIO` and `INCLUDE_GPIO_LED_SWITCHES` be defined. Usage example:

```
UINT32 mask = sysSwitchReadState();

if (mask & GPIO_SWITCH_SW06)
{
    // handle switch 6 press
}
```

void sysGpioBankSetDataDirection(UINT32 mask)

Sets the output enable for the J10 32-bit GPIO header located adjacent to the LCD display. Bits in the mask set to "1" are inputs, "0" are outputs.

void sysGpioBankWriteDiscretets(UINT32 data)

Writes to the 32-bit GPIO J10 header.

UINT32 sysGpioBankReadDiscretets(void)

Reads the state of the pins of the 32-bit GPIO J10 header.

void sysLedBusErrClear(UINT32 ledMask)

Turns the PLB & OPB bus error LEDs from red (bus error occurred) to green. This function does not clear the error condition. Parameter `ledMask` is formed from or'ing together `GPIO_LED_BUSERR` constants defined in `ML300.h`.

System ACE

These routines require that the `INCLUDE_XSYSACE` constant be defined.

STATUS sysSystemAceSetRebootAddr(unsigned configAddr)

Sets the reboot JTAG configuration address. This address is mapped to `cfgaddr0..7` as defined in `XILINX.SYS` in the root directory of the CF device. If this function is never invoked, then the default address is used. The default address is the address selected by the rotary switch. The given address will be rebooted if `sysToMonitor()` or `reset()` is called.

The `configAddr` parameter range is 0..7 (i.e. `cfgaddr0..7`) or -1 to select the default address.

Returns `ERROR` if `configAddr` is out of range, `OK` otherwise.

void sysSystemAceInitFS(void)

Initializes the required Wind River DosFs 2.0 libraries. Application code is not required to call this function on a BSP built with the Project facility.

STATUS sysSystemAceMount(char* mountPoint, int partition)

Mount the compact flash as DOS file system volume. The `mountpoint` parameter is an arbitrary string labeling the device. Once mounted, refer to this mountpoint in all file accesses. The `partition` parameter specifies the partition to mount. If "0" is specified then the boot device is assumed to not contain a partition table (i.e. it is treated like a floppy disk).

Note: Before calling this routine, be sure to initialize the DOS file system with a call to `sysSystemAceInitFS()`.

Note: Application code is not required to call this function on a BSP built with the Project facility with `INCLUDE_XSYSACE_AUTOMOUNT` defined.

Power & Temperature Monitor Functions

These functions require that `INCLUDE_XIIC` constant be defined.

void sysPowerMonCpuGet(int *v1_8, int *v2_5, int *v3_3, int *v5, int *v12)

This routine reads the two power monitor devices on the IIC bus to determine the current voltage levels on the CPU board. All voltages are returned in units of milli-volts. If the voltage cannot be read for any reason, then that voltage level is returned as `SYS_MEASUREMENT_ERROR`. Parameters are interpreted as follows: `v1_8` = 1.8volt source, etc..

void sysPowerMonIoGet(int *v1_8, int *v2_5, int *v3_3, int *v5, int *v12)

This routine reads the two power monitor devices on the IIC bus to determine the current voltage levels on the IO board. All voltages are returned in units of milli-volts. If the voltage cannot be read for any reason, then that voltage level is returned as `SYS_MEASUREMENT_ERROR`. Parameters are interpreted as follows: `v1_8` = 1.8volt source, etc..

void sysPowerMonShow(void)

Print the voltages from all power monitor sources to the console. If errors are encountered while reading the voltage monitors, then "Err" is displayed next to the voltage. This function requires `INCLUDE_POWERMON_SHOW` be defined in `config.h` or in the project facility under *development tool components* -> *show routines*.

void sysTemperatureMonGet(int *cpu, int *ambient)

This routine reads the two temperature sensing devices on the IIC bus to determine the current temperature. If the temperature cannot be read for whatever reason, then that temperature is returned as `SYS_MEASUREMENT_ERROR`. Temperature is returned in units of deg C.

void sysPowerMonShow(void)

Print the temperature (in deg C) for the CPU and the ambient temperature. If errors are encountered while reading the temperature monitors, then "Err" is displayed next to the temperature. `INCLUDE_TEMPERATUREMON_SHOW` be defined in `config.h` or in the project facility under *development tool components -> show routines*.

Miscellaneous Functions

void sysMsDelay(UINT32 delay)

Delay the specified number of milliseconds. The delay is implemented as a busy loop that occupies the CPU. The delay can be pre-empted by a higher priority task or interrupts if tasking/interrupts are enabled causing loss of delay precision.

void sysUsDelay(UINT32 delay)

Delay the specified number of microseconds. The delay is implemented as a busy loop that occupies the CPU. The delay can be pre-empted by a higher priority task or interrupts if tasking/interrupts are enabled causing loss of delay precision.

This function not accurate for delay times below 20us due to system overhead. The overhead is more or less constant and can be negated by the use of `SYS_US_DELAY_BIAS` defined in `config.h`. Use this constant to calibrate to your system's needs. As delivered with a 300 MHz CPU clock and a bias of -2, this function is accurate within +/-15% for a 20us delay. As the delay time increases, the accuracy increases.

void sysEepromWriteEnable(void)

Enables writes to the IIC EEPROM.

void sysEepromWriteDisable(void)

Disable writes to the IIC EEPROM.

Board Specific Options

This section discusses ML300 specific configuration options that can be set either in `config.h` or in the Project GUI. Unless otherwise stated, these options can be set by `#define`'ing or `#undef`'ing them in `config.h` or by defining them in the Project GUI in the project workspace's macros settings in the build tab.

Table 8-5: Custom BSP Options

Option	Description
INCLUDE_XSYSACE_INSTALL_-RESET_VEC	Controls whether reset code is placed the processor's reset vector address. This reset code will trigger SystemACE to load the default configuration bitstream.
INCLUDE_XSYSACE_AUTOMOUNT	Controls whether the System ACE filesystem is mounted at boot time using the next two SYSACE_ constants defined in this table. This constant affects only Project builds.
SYSACE_AUTOMOUNT_POINT	Default mount point used when INCLUDE_-XSYSACE_AUTOMOUNT is defined in Project builds.
SYSACE_AUTOMOUNT_PARTITION	Default partition used when INCLUDE_XSYSACE_-AUTOMOUNT is defined in Project builds.
INCLUDE_GPIO_LED_SWITCHES	Controls whether GPIO support is present for the switches on top of the board and the LED in close proximity to those switches. If support is not included, then functions sysLedOn, sysLedOff, and sysSwitchReadState have no effect.
INCLUDE_GPIO_TEST_PORT	Controls whether GPIO support is present for the J10 I/O connector port. If support is not included, then sysGpioBank functions have no effect.
SYS_US_DELAY_BIAS	Adds the specified number of microseconds to the delay parameter in sysUsDelay(). This option can be used to cancel out overhead.
INCLUDE_EMAC_PHY_RESET_-AT_BOOT	Controls whether the Ethernet PHY is reset at boot time. In the Project GUI, this is a parameter under the emac component and can be found under <i>hardware->peripherals->IP CSP-> Ethernet Core</i> . Set to TRUE to enable, FALSE to disable.
INCLUDE_POWERMON_SHOW	Controls whether function sysPowerMonShow is compiled into the BSP.
INCLUDE_TEMPERATUREMON_-SHOW	Controls whether function sysTemperatureMonShow is compiled into the BSP.
SYS_GPIO_SWITCH_DEBOUNCE_TICKS	Sampling interval used by function sysSwitchReadState() when attempting to debounce switches. Units are in clock ticks.

Release History

Version 1.2/0 - September 26, 2002

First pre-release for Tornado 2.0. This is a beta release that does not include support for all HW. Note that testing has been done on the ML3 evaluation board as opposed to the ML300. In other words, this version of the BSP has never been run on the ML300 hardware. The SEG IP bitstream is used for this load.

HW Supported

- 16550 UART on outside edge connector (VxWorks console)

- EMAC Ethernet
- 16MB DDR RAM
- 32KB BRAM
- System ACE
- PPC Instruction cache

HW Not supported

- PPC Data cache
- PPC MMU
- PLB/OPB Bridge register access (no access to BEAR, BESR registers).
- LCD display
- PCI
- GPIO
- Parallel Port
- PS2 Ports

Usage Notes

1. Bootroms: The bootroms are integrated into the FPGA bitstream and downloaded by System ACE at powerup and reset. There are two different types of bootroms stored in the ace subdirectory. Each one uses serial port #1 as the console at 38400 baud, N,8,1.

`top_vxboot.ace`: This bootrom has a hardcoded bootline of `"sysace=1(0,0):/cf0/vxworks/vxWorks.st"`. It will mount the compact flash device using the MPU interface of the System ACE as an external DOS volume. The given VxWorks image will be loaded and started. If a `/vxworks` directory is not in your compact flash device then create one and place your `vxWorks.st` image there. This bootrom has no network support.

`top_vxbootnet.ace`: This bootrom has a hardcoded bootline of

- a. `"xemac(0,0)host:c:/tornado/target/config/ML300/vxWorks h=192.168.0.1 e=192.168.0.2 u=xemhost pw=slurm"`. The network is started and the `vxWorks` image is downloaded via ftp.

2. When using SystemACE as the boot device for bootroms, make sure macro `INCLUDE_NET_INIT` is undefined and any macro that causes it to be defined such as `WDB_COMM_TYPE=WDB_COMM_END`.
3. Reset vector issue. The PPC405 reset vector is at physical address `FFFFFFFC`. With SystemACE at powerup, the processor will be prevented from executing an instruction at this address in some cases. With a VxWorks bootrom in the bitstream, SystemACE will load the FPGA IP cores, then the bootrom, place the PC at the bootrom entry point `romInit` and release it to begin fetching instructions. So what happens when you press the reset button? The answer is the processor will vector to `FFFFFFFC` and something had better be there. The BSP can be configured to initialize this reset vector with code to cause a System ACE jtag reboot. Defining `INCLUDE_XSYSACE_INSTALL_RESET_VEC` will install this code but a better solution would be to place something at the reset vector with `Data2Bram` in the Xilinx design flow toolchain.

Errata

1. Since there is no NVRAM support, the bootline is hardcoded in the `DEFAULT_BOOT_LINE` macro defined in `config.h`. The Ethernet MAC address is hardcoded in `sysNet.c`.
2. Pressing the CPU reset button will not reset the system and reload the bootrom. Users must press the System ACE reset button or place code in the BRAM at the end of the memory map that triggers system ACE to reset itself.

3. Ethernet 100Base-T may have poor performance with some host computers. If this is the case, then switch your host computer's ethernet adapter to 10Base-T.
4. When creating a Tornado Project using this BSP as the "basis BSP", ensure the data cache is disabled. Go to the "enable caches" component of the memory folder and set `USER_D_CACHE_ENABLE` to no value.

Version 1.2/1 - November 13, 2002

The first release using an EDK/platgen generated bitstream.

HW Supported

- 16550 UART on P107 (VxWorks console)
- 16550 UART on P106
- EMAC Ethernet
- 128MB DDR RAM
- 32KB BRAM
- PPC MMU & Instruction & Data caches
- PLB/OPB Bridge
- GPIO (LEDs & Switches)
- IIC including NVRAM, temperature & power monitors.

HW Not supported

- System ACE
- LCD Display
- PCI
- Parallel Port
- PS2 Ports
- USB Ports
- Audio Ports
- SPI

Usage Notes

At the time this document was updated, this BSP is largely untested on a real EDK/platgen bitstream load.

Errata

1. System mode debugging through the END connection does not work.
2. Serial port usage as the WDB target connection does not work. Serial port polling mode does not seem to work.

Version 1.2/2 - January 10, 2003

Tested using the EDK/platgen reference design bitstream `ML300_ppc405_example`.

New Features & Enhancements

1. Added support for SystemACE HW using 16-bit bus mode.
2. Added boot error logging to note where CSP driver errors were encountered at boot time.
3. Added bus error LED clear functions.
4. Added EEPROM write protect functions.
5. BRAM is now data cached by default.
6. Changed push button switch debouncing to use `taskDelay` instead of busy loop.

Bug Fixes

1. Moved implementation of `sysDcrPlb` functions to `sysALib.s` from `sysPlbOpb.c` because these functions cannot be implemented in "C" due to the way the VxWorks exception library works.
2. Corrected GPIO bitmasks and XPAR to SYS constants mappings in `ML300.h`.
3. Corrected `NETWORK_MEMORY_SIZE` constant definition. Previously, it had worked correctly only with the default setting of `USER_RESERVE_MEMORY`.
4. Removed init code that by default enabled critical interrupts from the PLB arbiter and PLB/OPB bridge. In some situations, this setup could cause VxWorks to reboot during a bus error.
5. Fixed bug in `intConnect` that prevented the user from connecting an ISR to a source that already had a connected ISR.

Usage Notes

1. Using the push button switches as interrupt sources requires a debouncing algorithm. Otherwise an avalanche of interrupts will befall you due to the noisy signals present on this board.

Errata

1. System mode debugging through the END connection does not work.
2. Serial port usage as the WDB target connection does not work. Serial port polling mode does not seem to work.
3. Stubbed out LCD code. Functions left to stay compatible with ML300seg. At this time there is no LCD HW IP Core for this particular LCD display in the EDK.
4. MMU induced instability. When MMU is enabled, have noted system stability problems especially when using the host shell to download and run object code. The BSP is shipped with the MMU disabled.
5. Errors accessing the IIC bus. On some occasions, the IIC bus has come up in an inaccessible state. When this occurs, the BSP will use the default bootline and the default MAC address.

ML300Seg Reference BSP

This handcoded BSP was created to complement the V2PDK example system for the ML300 reference board. It has been customized to work with hardware devices specific to the board. The BSP has enough support to allow application development using the Tornado tool chain.

This BSP directly supports the following hardware:

- 10/100 BaseT Ethernet
- RS-232 Serial ports (2)
- System ACE
- LCD Display
- GPIO (LEDs & pushbutton switches)
- IIC (including EEPROM, temperature & power monitors)
- 128MB DDR RAM
- 32KB BRAM
- PPC405 built-in timers, instruction cache, data cache

The BSP utilizes drivers for the following IP cores:

- EMAC Ethernet: END driver type
- 16550 UART on connector P107: SIO driver type on /tyco/0 (VxWorks console)
- 16550 UART on connector P106: SIO driver type on /tyco/1
- PLB to OPB bridge, PLB arbiter
- System ACE as a JTAG device
- System ACE as a block device for disk access (FAT32)
- INTC Interrupt controllers. 1 critical controller, 1 external controller

This BSP does not support the following hardware due to the lack of driver support:

- PCI
- Parallel port
- PS2 ports
- USB ports
- Audio ports
- Fiber ports
- SPI

Installation

See [Installation, page 131](#) for instruction on how to install this BSP.

Compact Flash & SystemACE

A compressed zipfile is provided in the ace subdirectory. This is a complete image containing a bootrom and sample VxWorks images in ace and elf file formats. See the README in the ace directory for more information.

To install this image, do the following:

1. Make a backup of your microdrive then erase all files from it.
2. Uncompress the ace/compactFlash.zip file to the microdrive.
3. Insert the microdrive into the compact flash slot on the ML300.
4. Connect a serial port cable to the P106 connector on the evaluation board. Default comm settings are 115200, N, 8, 1.
5. Set the rotary switch on the ML300 to setting 6 and apply power. At this point, the VxWorks bootrom should be running and writing to the console serial port.
6. Set the bootrom's bootline per your requirements. See [Bootroms, page 137](#) for more information.

Setting Ethernet MAC Address

To verify your MAC address is correct perform the following steps:

1. Set the rotary switch associated with the VxWorks bootrom and reboot the ML300.
2. Interrupt the countdown sequence to get the [VxWorks Boot]: prompt.
3. Enter the "N" command (case sensitive). The current MAC will be displayed and you will be prompted to enter a new MAC. The first three bytes of the MAC should be 000A35.

```
Press any key to stop auto-boot...
1
[VxWorks Boot]: N
```

```

Current Ethernet Address is: 00:0a:35:00:03:20
Modify only the last 3 bytes (board unique portion) of Ethernet Address.
The first 3 bytes are fixed at manufacturer's default address block.
00- 00
0a- 0a
35- 35
00-

```

If the MAC is valid, then press return three times to accept the default. On new boards, the address may be all FFs. If this is the case, enter the last three bytes that are assigned to the board's serial number. If you are not sure of the numbers, then enter return three times. This will change the MAC to 00:0a:35:FF:FF:FF. This will provide you with a valid MAC until the correct number is obtained. Boards with a MAC of all FFs will not be capable of running the network stack. Multiple boards connected to the same network with the same MAC will not work either.

Bootstrap Information

The default bitstream contains a bootstrap program that consists of a single instruction at the processor's reset vector which is in effect an endless loop, or `while(1)` in "C" programming constructs. This loop keeps the processor from running amok until SystemACE completes its download or an emulator connects to the target board. See [Reset Vector and SystemACE](#), page 143.

Memory Maps

Due to the nature of this evaluation board a full memory map is not given in this document. The user is instead referenced to "C" source code header file `xparameters.h`. This source file provides a memory map for all CSP devices. A partial map is given here.

Table 8-6: System Memory Map

Device	Start (hex)	End (hex)	Size (bytes)
PLB DDR	00000000	07DFFFFFFF	126 MB
PLB LCD Frame Buffer	07E00000	07FFFFFFF	2 MB
OPB Space	60000000	DFFFFFFF	2 GB
BRAM	FFFF8000	FFFFFFF	32 KB

RAM Memory Map

RAM device contains the VxWorks runtime image and heap space. ML300Seg follows VxWorks conventions for RAM usage for PowerPC processors. Refer to Appendix F of the *VxWorks 5.4 Programmer's Guide*.

Table 8-7: RAM Memory Map

Physical Address Range (hex)	Usage
00000000..000000FF	(DDR) Unused & undefined
00000100..00002FFF	(DDR) Interrupt Vector table
00003000..00010000	(DDR) VxWorks usage. Exception reason message and other VxWorks constructs are at the bottom of this region. Initial stack is set at the top of this range and grows downward. Once VxWorks has switched to multi-tasking mode, this stack is no longer used.

Table 8-7: RAM Memory Map

Physical Address Range (hex)	Usage
00010000..00BFFFFFFF	(DDR) RAM_LOW_ADRS. VxWorks image, interrupt stack, host memory pool, and heap space.
00C00000..07BFFFFFFF	(DDR) RAM_HIGH_ADRS. Two possible uses. (1) VxWorks bootrom image and heap space. (2) VxWorks heap space.
07C00000..07CFFFFFFF	(DDR) USER_RESERVED_MEM. This 1MB is used for network data buffers and network DMA descriptor spaces.
07D00000..07DFFFFFFF	(DDR) USER_RESERVED_MEM. This 1 MB is not used by BSP. Available for application use
07E00000..07FFFFFFF	(DDR) LCD frame buffer
FFFF8000..FFFFFFFF	(BRAM) Address FFFFFFFFC contains reset vector.

NVRAM Memory Map

NVRAM support is provided by a Microchip Technology 24LC32A EEPROM on the IIC bus. This device provides 4KB of storage space. BSP source code file `24LC32aNvRam.c` is the driver for this device and provides the API interface required by VxWorks. The primary BSP related objects stored in NVRAM are the bootline and the Ethernet MAC address.

When there is no IIC bus support, the BSP will replace the EEPROM driver with `$WIND_BASE/src/drv/mem/nullNvRam.c` which provides only function stubs so that VxWorks will link. When this is the case, the default bootline is used (see `config.h`) and the Ethernet MAC address defaults to: `00:0a:35:00:00:00`.

Table 8-8: NVRAM Memory Map

Part Offset Range (hex)	sysNvRamGet/Set Offset	Usage
0000..07FF	N/A*	Reserved for board level objects such as the Ethernet MAC address
0800..08FF	0000..00FF	Reserved for VxWorks bootline
0900..0FEF	0100..07EF	Unused
0FF0..0FFF	07F0..07FF	Reserved

* `sysNvRamGet` and `sysNvRamSet` are the VxWorks required NVRAM interface functions. The interface they provide uses offsets relative to the bootline offset. Accessing part offsets `0000..07FF` requires an alternate interface.

Caches

The caches are configured by the following constants in `ML300.h`. These constants map to the PPC cache control registers of the same name. See PPC405 documentation for further information on these registers:

- `ML300_ICCR_VAL` - Initial contents of the ICCR register (instruction cacheability attribute).
- `ML300_DCCR_VAL` - Initial contents of the DCCR register (data cacheability attribute).
- `ML300_DCWR_VAL` - Initial contents of the DCWR register (write back/through attribute).

- ML300_SGR_VAL - Initial contents of the SGR register (guarded attribute).

Table 8-9: Cache Map

Physical Address Range (hex)	I Cache	D Cache	Write Back/Through	Guarded
00000000..07FFFFFF	Y	Y ¹	Back	N
F8000000..FFFFFFFF	Y	Y	N/A	N
everything else	N	N	N/A	N

¹ This region includes the LCD frame buffer on the ML300Seg. A data cache flush may be required to complete writes to this buffer.

External Interrupts

There are two INTC interrupt controller IP cores in the design. One is wired to the external interrupt signal of the PPC405 and the other to the critical interrupt signal.

PLB/OPB bridges & arbiters are wired to the critical INTC instance. If these interrupt sources are enabled and the PPC machine check interrupt is enabled then VxWorks may reboot when an exception occurs. This is because the PLB/OPB bridge/arbiters will assert their interrupt signal when they cannot complete a transaction to the INTC. This signal will propagate to the PPC as a critical interrupt exception (vector 0x100). At the same time the PPC will detect a bad bus cycle and generate a machine check exception (vector 0x200). VxWorks will begin handling the first exception, but during this time the second exception arrives. VxWorks architecturally does not allow this and will reboot the system when it occurs.

It is not recommended to `sysIntCritEnable()` one of these interrupt sources. Instead, use the VxWorks `excHookAdd()` function to use your own function perform custom exception processing (after VxWorks finishes its own processing). Here, the hook function can examine the bridges/arbiters and perform whatever task is required for the event.

IIC

There are several devices connected to the IIC bus with hardwired addresses. These addresses are defined for the BSP in the `ML300.h` header file. The BSP provides a polled interface to the IIC bus to access these devices. The interface includes a mutual exclusion semaphore that can be used to prevent more than one task from accessing the bus at a time. Before the operating system is up, the semaphore is not available and it is up to boot code to sequence access to the bus. This should not be an issue since the system is single-threaded at boot time and the only device accessed should be the NVRAM.

BSP file `sysIic.c` provides initialization, read/write primitives, and resource allocation functions.

Board API

There are a handful of "board level" BSP functions not implemented by the CSP device drivers. Prototypes for these functions are located in `config/ML300/sysLibExtra.h`.

This section will not go over CSP device driver functions. Instead the user is directed to the appropriate `ip_csp/xsrc/<device>.c` file for documentation and usage.

Standard I/O

The BSP comes with `stdin`, `stdout`, and `stderr` directed through the UART on the P106 connector. The default UART baud rate is set to 115200, no parity, 8 data bits, and 1 stop bit. The secondary UART on P107 is enabled and ready for application usage. It defaults to 19200 baud, no parity, 8 data bits, and 1 stop bit.

GPIO

Two instances of GPIO can be included in the BSP. The first instance controls the momentary push button switches and their surrounding LEDs. The second controls the 32 GPIO lines on the J10 connector. Both instances require that `INCLUDE_XGPIO` constant be defined. Each instance can be enabled or disabled with constants `INCLUDE_GPIO_LED_SWITCHES` and `INCLUDE_GPIO_TEST_PORT`.

void sysLedOn(UINT32 mask)

Turns on LEDs in the mask. Bits set to one cause the associated LED to be illuminated. The mask is built using constants `GPIO_LED_DSxx` defined in `ML300.h` where `xx` is the LED number and `DSxx` is the LED label on the PCB. This function requires that both `INCLUDE_XGPIO` and `INCLUDE_GPIO_LED_SWITCHES` be defined.

void sysLedOff(UINT32 mask)

Turns off LEDs in the mask. Bits set to one cause the associated LED to be turned off. The mask is built using constants `GPIO_LED_DSxx` defined in `ML300.h` where `xx` is the LED number and `DSxx` is the LED label on the PCB. This function requires that both `INCLUDE_XGPIO` and `INCLUDE_GPIO_LED_SWITCHES` be defined.

UINT32 sysSwitchReadState(void)

Reads the state of all the push button switches. A mask is returned describing which switches are closed (i.e. being pushed). The mask is decoded using constants `GPIO_SWITCH_SWxx` defined in `ML300.h` where `xx` is the switch number and `SWxx` is the switch label on the PCB. This function requires that both `INCLUDE_XGPIO` and `INCLUDE_GPIO_LED_SWITCHES` be defined. Usage example:

```
UINT32 mask = sysSwitchReadState();

if (mask & GPIO_SWITCH_SW06)
{
    // handle switch 6 press
}
```

void sysGpioBankSetDataDirection(UINT32 mask)

Sets the output enable for the J10 32-bit GPIO header located adjacent to the LCD display. Bits in the mask set to "1" are outputs, "0" are inputs.

void sysGpioBankWriteDiscrettes(UINT32 data)

Writes to the 32-bit GPIO J10 header.

UINT32 sysGpioBankReadDiscrettes(void)

Reads the state of the pins of the 32-bit GPIO J10 header.

System ACE

These routines require that the `INCLUDE_XSYSACE` constant be defined.

STATUS sysSystemAceSetRebootAddr(unsigned configAddr)

Sets the reboot JTAG configuration address. This address is mapped to `cfgaddr0..7` as defined in `XILINX.SYS` in the root directory of the CF device. If this function is never invoked, then the default address is used. The default address is the address selected by the rotary switch. The given address will be rebooted if `sysToMonitor()` or `reset()` is called.

The `configAddr` parameter range is 0..7 (i.e. `cfgaddr0..7`) or -1 to select the default address.

Returns `ERROR` if `configAddr` is out of range, `OK` otherwise.

void sysSystemAceInitFS(void)

Initializes the required Wind River DosFs 2.0 libraries. Application code is not required to call this function on a BSP built with the Project facility.

STATUS sysSystemAceMount(char* mountPoint, int partition)

Mount the compact flash as DOS file system volume. The `mountpoint` parameter is an arbitrary string labeling the device. Once mounted, refer to this mountpoint in all file accesses. The `partition` parameter specifies the partition to mount. If "0" is specified then the boot device is assumed to not contain a partition table (i.e. it is treated like a floppy disk).

Note: Before calling this routine, be sure to initialize the DOS file system with a call to `sysSystemAceInitFS()`.

Note: Application code is not required to call this function on a BSP built with the Project facility with `INCLUDE_XSYSACE_AUTOMOUNT` defined.

LCD

These functions perform very basic operations useful for verifying the LCD is working. A more substantial library will be required if, say, someone wanted to port Quake. Screen geometry is defined in `ML300.h` with the constants `LCD_COLS`, `LCD_ROWS`, and `LCD_ROW_ALIGNMENT`.

void sysLcdSetColor(UINT32 rgb)

Set the entire display to the color encoded by the `rgb` parameter. The `rgb` parameter is encoded as follows: `0x00RRGGB` where `RR` is the red component, `GG` is the green component, and `RR` is the red component. A value of `0x00000000` is black, `0x00FFFFFF` is white.

void sysLcdDisplayColorBars(void)

Writes a test display to the LCD screen that includes 8 bars in the top half of the display and a 256 grey-scale pattern in the lower half of the display.

void sysLcdSetPixels(int row, int col, unsigned numPixels, UINT32 rgb)

Starting at `row` and `column`, write the RGB encoded value to consecutive pixels as they exist in the LCD's frame buffer. This is basically a horizontal line draw function. If `numPixels` is large enough, then the RGB color continues onto the next row. See `sysLcdSetColor` described above for information on the `rgb` parameter.

STATUS sysLcdSetBrightness(unsigned char value)

This routine sets the brightness level of the LCD display. Valid range is 0..255 with 0 being the dimmest setting. This setting is persistent in that the LCD keeps this setting even through power cycles.

Returns `ERROR` if unable to communicate with device, `OK` otherwise.

STATUS sysLcdGetBrightness(unsigned char *value)

This routine retrieves the brightness level of the LCD display. Valid returned range is 0..255 with 0 being the dimmest setting.

Returns `ERROR` if unable to communicate with device, `OK` otherwise.

Power & Temperature Monitor Functions

void sysPowerMonCpuGet(int *v1_8, int *v2_5, int *v3_3, int *v5, int *v12)

This routine reads the two power monitor devices on the IIC bus to determine the current voltage levels on the CPU board. All voltages are returned in units of milli-volts. If the voltage cannot be read for any reason, then that voltage level is returned as

`SYS_MEASUREMENT-_ERROR`. Parameters are interpreted as follows: `v1_8` = 1.8volt source, etc..

void sysPowerMonIoGet(int *v1_8, int *v2_5, int *v3_3, int *v5, int *v12)

This routine reads the two power monitor devices on the IIC bus to determine the current voltage levels on the IO board. All voltages are returned in units of milli-volts. If the voltage cannot be read for any reason, then that voltage level is returned as `SYS_MEASUREMENT-_ERROR`. Parameters are interpreted as follows: `v1_8` = 1.8volt source, etc..

void sysPowerMonShow(void)

Print the voltages from all power monitor sources to the console. If errors are encountered while reading the voltage monitors, then "Err" is displayed next to the voltage. This function requires `INCLUDE_POWERMON_SHOW` be defined in `config.h` or in the project facility under *development tool components -> show routines*.

void sysTemperatureMonGet(int *cpu, int *ambient)

This routine reads the two temperature sensing devices on the IIC bus to determine the current temperature. If the temperature cannot be read for whatever reason, then that temperature is returned as `SYS_MEASUREMENT_ERROR`. Temperature is returned in units of deg C.

void sysPowerMonShow(void)

Print the temperature (in deg C) for the CPU and the ambient temperature. If errors are encountered while reading the temperature monitors, then "Err" is displayed next to the temperature. `INCLUDE_TEMPERATUREMON_SHOW` be defined in `config.h` or in the project facility under *development tool components -> show routines*.

Miscellaneous Functions

void sysMsDelay(UINT32 delay)

Delay the specified number of milliseconds. The delay is implemented as a busy loop that occupies the CPU. The delay can be pre-empted by a higher priority task or interrupts if tasking/interrupts are enabled causing loss of delay precision.

void sysUsDelay(UINT32 delay)

Delay the specified number of microseconds. The delay is implemented as a busy loop that occupies the CPU. The delay can be pre-empted by a higher priority task or interrupts if tasking/interrupts are enabled causing loss of delay precision.

This function not accurate for delay times below 20us due to system overhead. The overhead is more or less constant and can be negated by the use of `SYS_US_DELAY_BIAS` defined in `config.h`. Use this constant to calibrate to your system's needs. As delivered with a 300 MHz CPU clock and a bias of -2, this function is accurate within +/-15% for a 20us delay. As the delay time increases, the accuracy increases.

Board Specific Options

This section discusses ML300Seg specific configuration options that can be set either in `config.h` or in the Project GUI. Unless otherwise stated, these options can be set by `#define`'ing or `#undef`'ing them in `config.h` or by defining them in the Project GUI in the project workspace's macros settings in the build tab.

Table 8-10: Custom BSP Options

Option	Description
INCLUDE_XSYSACE_INSTALL_-RESET_VEC	Controls whether reset code is placed the processor's reset vector address. This reset code will trigger SystemACE to load the default configuration bitstream.
INCLUDE_XSYSACE_AUTOMOUNT	Controls whether the System ACE filesystem is mounted at boot time using the next two SYSACE_ constants defined in this table. This constant affects only Project builds.
SYSACE_AUTOMOUNT_POINT	Default mount point used when INCLUDE_-XSYSACE_AUTOMOUNT is defined in Project builds.
SYSACE_AUTOMOUNT_PARTITION	Default partition used when INCLUDE_XSYSACE_-AUTOMOUNT is defined in Project builds.
INCLUDE_GPIO_LED_SWITCHES	Controls whether GPIO support is present for the switches on top of the board and the LED in close proximity to those switches. If support is not included, then functions sysLedOn, sysLedOff, and sysSwitchReadState have no effect.
INCLUDE_GPIO_TEST_PORT	Controls whether GPIO support is present for the J10 I/O connector port. If support is not included, then sysGpioBank functions have no effect.
SYS_US_DELAY_BIAS	Adds the specified number of microseconds to the delay parameter in sysUsDelay(). This option can be used to cancel out overhead.
INCLUDE_LCD_CLEAR_AT_BOOT	Clears the LCD display at VxWorks boot time.
INCLUDE_LCD_BARS_AT_BOOT	Displays the color bar test screen on the LCD at VxWorks boot time. If both INCLUDE_LCD_BARS_-AT_BOOT and INCLUDE_LCD_CLEAR_AT_BOOT are defined, the LCD will first be cleared then the color bars will be drawn.
INCLUDE_EMAC_PHY_RESET_-AT_BOOT	Controls whether the Ethernet PHY is reset at boot time. In the Project GUI, this is a parameter under the emac component and can be found under <i>hardware->peripherals->IP CSP-> Ethernet Core</i> . Set to TRUE to enable, FALSE to disable.
INCLUDE_POWERMON_SHOW	Controls whether function sysPowerMonShow is compiled into the BSP.
INCLUDE_TEMPERATUREMON_-SHOW	Controls whether function sysTemperatureMonShow is compiled into the BSP.
SYS_GPIO_SWITCH_DEBOUNCE_TICKS	Sampling interval used by function sysSwitchReadState() when attempting to debounce switches. Units are in clock ticks.

Release History

Version 1.2/0 (seg 092402) - November 13, 2002

Written using the ML300 HW as a testbed and the 9/24/02 hand-coded version of the SEG IP load. This release supports more HW including the data cache, LCD, IIC, GPIO LEDs and switches.

This is in reality a renamed ML300 1.2/0 BSP with additional HW support and other refinements. The ML300 1.2/x BSP will be based on EDK/platgen bitstreams.

HW Supported

- 16550 UART on P107 (VxWorks console)
- 16550 UART on P106
- EMAC Ethernet
- 126MB DDR RAM
- 32KB BRAM
- LCD Display
- System ACE
- PPC MMU & Instruction & Data caches
- PLB/OPB Bridge BEAR & BESR register access (seg version)
- GPIO (LEDs & Switches)
- IIC including NVRAM, temperature & power monitors, and LCD brightness.

HW Not supported

The SEG IP load used by this BSP does map control registers for the following devices, however there is no BSP support for them.

- PCI
- Parallel Port
- PS2 Ports
- USB Ports
- Audio Ports
- SPI

Usage Notes

None.

Errata

1. System mode debugging through the END connection does not work.
2. Serial port usage as the WDB target connection does not work. Serial port polling mode does not seem to work.

Insight MDFG456 Reference BSP

This handcoded BSP was created to complement the EDK example system for the Insight MDFG456 reference board. It has been customized to work with hardware devices specific to the board. The BSP has enough support to allow application development using the Tornado tool chain.

This BSP directly supports the following hardware:

- 10/100 BaseT Ethernet on P160 module

- RS-232 Serial ports, one on main board, one on P160 module
- System ACE (requires daughter card)
- LCD Display
- GPIO (LEDs, pushbutton switches, DIP switches)
- 8MB SDRAM (Rev1 boards)
- 32KB BRAM
- 8MB Flash memory / 1MB SRAM on P160 module
- PPC405 built-in timers, instruction cache, data cache

The BSP utilizes drivers for the following IP cores:

- EMAC Ethernet: END driver type
- UART Lite on main board connector, SIO driver type on /tyco/0 (VxWorks console)
- UART Lite on P160 connectorP, SIO driver type on /tyco/1
- PLB to OPB bridge, PLB arbiter
- System ACE as a JTAG device
- System ACE as a block device for disk access (FAT32)
- INTC Interrupt controllers. 1 critical controller, 1 external controller

This BSP does not support the following hardware:

- IIC header on P160 module
- SPI header on P160 module
- PS/2 ports
- USB ports

Installation

See **Installation**, page 131 for instruction on how to install this BSP.

Compact Flash & SystemACE

A compress zipfile is provided in the ace subdirectory. This is a complete image containing a bootrom and sample VxWorks images in ace and elf file formats. See the README in the ace directory for more information.

To install this image, do the following:

1. Make a backup of your microdrive then erase all files from it.
2. Uncompress the ace/compactFlash.zip file to the microdrive.
3. Insert the microdrive into the compact flash slot on the MDFG456.
4. Connect a serial port cable to the mainboard connector. Default comm settings are 19200, N, 8, 1. Note that you shouldn't use a null modem cable with this board.
5. Set the rotary switch on the MDFG456 to setting 6 and apply power. At this point, the VxWorks bootrom should be running and writing to the console serial port.
6. Set the bootrom boot line per your requirements. See **Bootroms**, page 137 for more information.

Setting Ethernet MAC Address

This procedure assumes a working P160 module has been attached to the mainboard.

To verify your MAC address is correct perform the following steps:

1. Reboot the system.
2. Interrupt the countdown sequence to get the [VxWorks Boot]: prompt.
3. Enter the "N" command (case sensitive). The current MAC will be displayed and you will be prompted to enter a new MAC. The first three bytes of the MAC should be 000A35.

```

Press any key to stop auto-boot...
1
[VxWorks Boot]: N
Current Ethernet Address is: 00:0a:35:00:03:20
Modify only the last 3 bytes (board unique portion) of Ethernet Address.
The first 3 bytes are fixed at manufacturer's default address block.
00- 00
0a- 0a
35- 35
00-

```

If the MAC is valid, then press return three times to accept the default. On new boards, the address may be all FFs. If this is the case, enter the last three bytes that are assigned to the board's serial number. If you are not sure of the numbers, then enter return three times. This will change the MAC to 00:0a:35:FF:FF:FF. This will provide you with a valid MAC until the correct number is obtained. Boards with a MAC of all FFs will not be capable of running the network stack. Multiple boards connected to the same network with the same MAC will not work either.

Bootrom Flash Programming

This section will discuss steps involved with getting a VxWorks bootrom into flash memory storage using the SingleStep debugger. It is assumed the reader is familiar with the debugger and how to setup a connection to the target board.

Procedure

The following steps assume the target board is off, contains the FPGA bitstream associated with the example system, a VisionProbe is connected to the board, the debugger is not running, and "bootrom" or "bootrom_uncmp" has been created (see [Creating Bootroms, page 137](#)).

1. Ensure DIP switch #2 is set to the ON position and turn the target board on. LED1 should be flashing. If a different bitstream other than the one that came with the example system is being used, then the DIP switch and LED behaviour may be different.
2. Start the SingleStep debugger. Click "Debug without a file" then click "OK" to connect to the target board.
3. Go to the main menu and select "Tools" then "Vision Flash Utility".
4. Click the "Files" tab and press the "Convert" button. Select the file to program.
5. In the "File Conversion" dialog, change the "start address" to 0x10000 for "bootrom" or 0xC00000 for "bootrom_uncmp" image types. Click the "Add the newly converted file to the flash file list" check box, then click "Convert". The file should be converted from ELF to a binary representation with the resulting .bin added to the binary files list.
6. Click on the newly created file then click "Edit". Change the start address to 0xDE020000 then select "OK". This address corresponds to the first main block of flash memory. If the bitstream has changed the base address of flash then use the appropriate value (base + 0x20000).

7. Click the "toggle enable" button. The "Enabled" field in the file list should change from NOT ENABLED to ENABLED. Now click the "Configuration" tab to setup which flash programming algorithm to utilize.
8. In the folder window select Devices -> AMD -> 29DL32xxB -> 2 Devices. In the "Device Configuration" fields, enter the base address of the flash array at 0xDE000000. In the "Erase Range" fields select block #'s 8 through 31.
9. Click the "Program tab". Select the "Initialize Target" checkbox. Click "Erase/Program" to write the bootrom to flash memory.

Troubleshooting

Problem: SingleStep never finishes the programming operation.

Erasing and programming can take up to 3 minutes to complete. During this operation, SingleStep shows a progress bar. If this progress stops for more than a few minutes then emulator configuration settings may be incorrect. Verify "trap exception" is set to yes and the workspace is set to a valid writable RAM space. This check is done using the emulator command window. At the "SingleStep>" prompt, enter "vsh", then enter "cf". The emulator settings are listed. Change as required.

Problem: The bootline and MAC address have been overwritten.

The BSP uses the first block of flash to store the bootline and MAC address. The flash erase range may have been set to erase this block. You will need to adjust the erase range to leave this block untouched. If this occurs, then the bootline and MAC will have to be reset. See [Bootline, page 138](#) and [Setting Ethernet MAC Address, page 164](#).

Bootstrap Information

The default bitstream contains bootstrap code that performs the following algorithm when the processor is reset:

```

if (DIP switch #2 == "ON")
{
    Flash LED #1 forever;
}
else
{
    Turn on LEDs #1,#2,#3,#4;
    Jump to flash address 0xDE020000;
}

```

DIP #2, when in the "ON" position, keeps the processor from running amok until SystemACE completes its download or an emulator connects to the target board.

When "OFF", the processor will begin executing instruction in flash memory where a VxWorks bootrom or some other OS loader resides. If there is no flash, or nothing programmed into flash, or no GPIO with LED/DIP connections in the bitstream, then there will likely be an exception to address 0xFFFF0200, 0xFFFF0100, or 0xFFFF0700. If there is nothing at these address, then the processor will remain stuck there in an exception loop.

Memory Maps

Due to the nature of this evaluation board a full memory map is not given in this document. The user is instead referenced to "C" source code header file `xparameters.h`. This source file provides a memory map for all CSP devices. A partial map is given here that relates directly to BSP operation.

Table 8-11: System Memory Map

Device	Start (hex)	End (hex)	Size (bytes)
PLB SDRAM	00000000	007FFFFFFF	32 MB
OPB Space	40000000	DFFFFFFFF	2.5 GB
PLB BRAM	FFFF8000	FFFFFFFF	32 KB

RAM Memory Map

MDFG456 follows VxWorks conventions for RAM usage for PowerPC processors. Refer to Appendix F of the *VxWorks 5.4 Programmer's Guide*.

Table 8-12: RAM Memory Map

Physical Address Range (hex)	Usage
00000000..000000FF	(SDRAM) Unused & undefined
00000100..00002FFF	(SDRAM) Interrupt Vector table
00003000..00010000	(SDRAM) VxWorks usage. Exception reason message and other VxWorks constructs are at the bottom of this region. Initial stack is set at the top of this range and grows downward. Once VxWorks has switched to multi-tasking mode, this stack is no longer used.
00010000..00BFFFFFF	(SDRAM) RAM_LOW_ADRS. VxWorks image, interrupt stack, host memory pool, and heap space.
00C00000..01FFFFFF	(SDRAM) RAM_HIGH_ADRS. Two possible uses. (1) VxWorks bootrom image and heap space. (2) VxWorks heap space.
48000000..480FFFFFF	(SRAM) This memory is resident on the P160 communications add-on module and is part of the Toshiba flash memory device. This memory area is used for network buffers.
FFFF8000..FFFFFFFF	(BRAM) Address FFFFFFFC contains reset vector.

Flash Memory Map

The P160 communications module contains 8MB of flash memory implemented on two Toshiba parts. These parts are wired together to form a 32 bit memory width. The flash blocks are not of uniform size. Smaller "parameter" blocks reside at the beginning of the addressing range of these parts. The BSP uses the first one of these eight parameter blocks. The remaining parameter blocks are not used. The "main" blocks can be used to store a VxWorks image or any other user data. The "hidden" block is not utilized by this BSP.

Table 8-13: Flash Memory Map

Offset Byte Address Range (hex)	Usage
00000000..00003FFF	First parameter block used for NVRAM storage VxWorks boot line at offset [0..255] Ethernet mac address at offset [3FFA-3FFF]
00004000..0001FFFF	Remaining parameter blocks unused

Table 8-13: Flash Memory Map

Offset Byte Address Range (hex)	Usage
00020000..0031FFFF	VxWorks image ¹
00320000..007FFFFFFF	Unused

¹ This region can be enlarged past 0x00320000 to accommodate big VxWorks images. ROM_TEXT_SIZE in config.h and the BSP makefile must be changed to reflect the larger range.

OPB Memory Map

Table 8-14: OPB Memory Map (unused areas not shown)

Physical Address Range (hex)	Usage
48000000..480FFFFFFF	SRAM
5FFFFFF00..5FFFFFFF	EMC control
60000000..6000FFFF	OPB peripherals
DE000000..DE7FFFFFFF	Flash memory

NVRAM Memory Map

NVRAM support is provided by the Toshiba flash memory. The first parameter block of this flash array is reserved for NVRAM. A special NVRAM to flash driver is utilized by the BSP at \$WIND_BASE/src/drv/mem/nvRamToFlash.c. This driver uses functions in sysFlash.c to read/write parameters to NVRAM.

When there is no flash support, the BSP will replace the NVRAM driver with \$WIND_BASE/src/drv/mem/nullNvRam.c which provides only function stubs so that VxWorks will link. When this is the case, the default bootline is used (see config.h) and the Ethernet MAC address defaults to: 00:0a:35:00:00:00.

Table 8-15: NVRAM Memory Map

Part Offset Range (hex)	sysNvRamGet/Set Offset	Usage
0000..00FF	0000..00FF	Reserved for VxWorks bootline
0100..3FF9	0100..3FF9	Unused
3FFA..3FFF	3FFA..3FFF	Ethernet MAC address

Caches

The caches are configured by the following constants in MDFG456.h. These constants map to the PPC cache control registers of the same name. See PPC405 documentation for further information on these registers:

- MDFG456_ICCR_VAL - Initial contents of the ICCR register (instruction cacheability attribute).
- MDFG456_DCCR_VAL - Initial contents of the DCCR register (data cacheability attribute).
- MDFG456_DCWR_VAL - Initial contents of the DCWR register (write back/through attribute).
- MDFG456_SGR_VAL - Initial contents of the SGR register (guarded attribute).

Table 8-16: Cache Map

Physical Address Range (hex)	I Cache	D Cache	Write Back/Through	Guarded
00000000..01FFFFFF	Y	Y ¹	Back	N
F8000000..FFFFFFFF	Y	Y	N/A	N
everything else	N	N	N/A	N

¹ Rev1 boards require data caching enabled in this address range. This is due to a SDRAM design error that prevents byte and half-word access.

External Interrupts

There are two INTC interrupt controller IP cores in the design. One is wired to the external interrupt signal of the PPC405 and the other to the critical interrupt signal.

PLB/OPB bridges & arbiters are wired to the critical INTC instance. If these interrupt sources are enabled and the PPC machine check interrupt is enabled then VxWorks may reboot when an exception occurs. This is because the PLB/OPB bridge/arbiters will assert their interrupt signal when they cannot complete a transaction to the INTC. This signal will propagate to the PPC as a critical interrupt exception (vector 0x100). At the same time the PPC will detect a bad bus cycle and generate a machine check exception (vector 0x200). VxWorks will begin handling the first exception, but during this time the second exception arrives. VxWorks architecturally does not allow this and will reboot the system when it occurs.

It is not recommended to `sysIntCritEnable()` one of these interrupt sources. Instead, use the VxWorks `excHookAdd()` function to use your own function perform custom exception processing (after VxWorks finishes its own processing). Here, the hook function can examine the bridges/arbiters and perform whatever task is required for the event.

GPIO Switches & LEDs

LEDs 1, 2, 3, and 4 are available for application use. Pushbutton switches 1, 2, and 3 are available for application use. DIP switches 1-4 are reserved for HW and BSP use. DIP switches 5-8 are available for application use.

Table 8-17: DIP Switch Usage

Number	Use
1	Controls JTAG muxing between SystemACE and the CPU debug port. This feature is implemented by the bitstream.
2	Controls whether system boots from flash memory. When set to "ON", the processor spins in a loop flashing LED1 forever. When set to "OFF", all LEDs are illuminated and control is transferred to flash memory. This feature is implemented by the bootstrap program located in BRAM and incorporated into the bitstream.
3	Reserved for future use
4	Reserved for future use
5	Available for application use

Table 8-17: DIP Switch Usage

Number	Use
6	Available for application use
7	Available for application use
8	Available for application use

Board API

This section will not go over CSP device driver functions. Instead the user is directed to the appropriate `ip_csp/xsrc/<device>.c` file for documentation and usage.

There are a handful of "board level" BSP functions not implemented by the CSP device drivers. Prototypes for these functions are located in:

```
config/MDFG456/sysLibExtra.h.
```

Standard I/O

The BSP comes with `stdin`, `stdout`, and `stderr` directed through the UART on the main board. The default UART baud rate is set to 19200, no parity, 8 data bits, and 1 stop bit. The secondary UART on the P160 communications is enabled and ready for application usage. It defaults to 19200 baud, no parity, 8 data bits, and 1 stop bit.

GPIO

Two instances of GPIO can be included in the BSP. The first instance controls the momentary push button switches, their surrounding LEDs, and the bank of 8 DIP switches. The second GPIO instance controls the LCD display. Both instances require that `INCLUDE_XGPIO` constant be defined.

DIP switches 1-4 are reserved for the bitstream and for the BSP. Application code may use switches 5-8.

void sysLedOn(UINT32 mask)

Turns on LEDs in the mask. Bits set to one cause the associated LED to be illuminated. The mask is built using constants `GPIO_OUT_LEDx` defined in `MDFG456.h` where `x` is the LED number on the PCB. This function requires `INCLUDE_XGPIO` be defined.

void sysLedOff(UINT32 mask)

Turns off LEDs in the mask. Bits set to one cause the associated LED to be turned off. The mask is built using constants `GPIO_OUT_LEDx` defined in `MDFG456.h` where `x` is the LED number on the PCB. This function requires `INCLUDE_XGPIO` be defined.

UINT32 sysSwitchReadState(void)

Reads the state of all the push button switches. A mask is returned describing which switches are closed (i.e. being pushed). The mask is decoded using constants `GPIO_IN_PUSHx` defined in `MDFG456.h` where `x` is the switch on the PCB. This function requires `INCLUDE_XGPIO` be defined. Usage example:

```
UINT32 mask = sysSwitchReadState();

if (mask & GPIO_IN_PUSH3)
{
    // handle switch 3 press
}
```

UINT32 sysDipReadState(void)

Reads the state of all the DIP switches. A mask is returned describing which switches are in the "ON" position. The mask is decoded using constants `GPIO_IN_DIPx` defined in

MDFG456.h where x is the DIP switch position on the PCB. This function requires `INCLUDE_XGPIO` be defined. Usage example:

```
UINT32 mask = sysDipReadState();

if (mask & GPIO_IN_DIP5)
{
    // handle DIP #5 being "on"
}
```

System ACE

These routines require that the `INCLUDE_XSYSACE` constant be defined.

STATUS sysSystemAceSetRebootAddr(unsigned configAddr)

Sets the reboot JTAG configuration address. This address is mapped to `cfgaddr0..7` as defined in `XILINX.SYS` in the root directory of the CF device. If this function is never invoked, then the default address is used. The default address is the address selected by the rotary switch. The given address will be rebooted if `sysToMonitor()` or `reset()` is called.

The `configAddr` parameter range is 0..7 (i.e. `cfgaddr0..7`) or -1 to select the default address.

Returns `ERROR` if `configAddr` is out of range, `OK` otherwise.

void sysSystemAceInitFS(void)

Initializes the required Wind River DosFs 2.0 libraries. Application code is not required to call this function on a BSP built with the Project facility.

STATUS sysSystemAceMount(char* mountPoint, int partition)

Mount the compact flash as DOS file system volume. The `mountpoint` parameter is an arbitrary string labeling the device. Once mounted, refer to this mountpoint in all file accesses. The `partition` parameter specifies the partition to mount. If "0" is specified then the boot device is assumed to not contain a partition table (i.e. it is treated like a floppy disk).

Note: Before calling this routine, be sure to initialize the DOS file system with a call to `sysSystemAceInitFS()`.

Note: Application code is not required to call this function on a BSP built with the Project facility with `INCLUDE_XSYSACE_AUTOMOUNT` defined.

LCD

This board contains a LCD character display capable of displaying 1 or 2 lines of characters. Control of the display is handled using GPIO lines. The BSP sets up the LCD display for two lines using the 5x8 character matrix with no cursor. This setup provides a 2x16 character window.

Low level functions are available to write instructions and data to the device (see `sysLcdWriteInstruction()` and `sysLcdWriteData()` in section **GPIO**, page 170).

void sysLcdWriteString(char* string)

This function clears the display then writes the given string to it. The string parameter may truncate in the display if too long. A null string will clear the display. If string contains the newline character "\n", then characters following it will be placed in the 2nd line of the display. If more than one newline is present, then the text following the last newline will be written to the 2nd line of the display.

This function assumes LCD display characteristics have not been changed since `sysLcdInit()` was invoked at boot time. If display font or line mode have been changed, then the string may look scrambled in the display.

void sysLcdWriteInstruction(UINT8 data)

This function clocks in an instruction command to the LCD device. Application code can use this function to perform low level operations to the LCD display.

void sysLcdWriteData(UINT8 data)

This function clocks in data to the LCD device's internal RAM. This function is typically used to place a character somewhere in the device's memory. Application code can use this function to perform low level operations to the LCD display.

Miscellaneous Functions

void sysMsDelay(UINT32 delay)

Delay the specified number of milliseconds. The delay is implemented as a busy loop that occupies the CPU. The delay can be pre-empted by a higher priority task or interrupts if tasking/interrupts are enabled causing loss of delay precision.

void sysUsDelay(UINT32 delay)

Delay the specified number of microseconds. The delay is implemented as a busy loop that occupies the CPU. The delay can be pre-empted by a higher priority task or interrupts if tasking/interrupts are enabled causing loss of delay precision.

This function not accurate for delay times below 20us due to system overhead. The overhead is more or less constant and can be negated by the use of `SYS_US_DELAY_BIAS` defined in `config.h`. Use this constant to calibrate to your system's needs.

Board Specific Options

This section discusses MDFG456 specific configuration options that can be set either in `config.h` or in the Project GUI. Unless otherwise stated, these options can be set by `#define`'ing or `#undef`'ing them in `config.h` or by defining them in the Project GUI in the project workspace's macros settings in the build tab.

Table 8-18: Custom BSP Options

Option	Description
<code>INCLUDE_P160_COMM_MODULE</code>	Controls whether code used to control peripherals is eligible to be compiled into the BSP. P160 has an ethernet controller, flash/SRAM memories, and a UART amongst other things.
<code>INCLUDE_XSYSACE_INSTALL_-RESET_VEC</code>	Controls whether reset code is placed the processor's reset vector address. This reset code will trigger SystemACE to load the default configuration bitstream.
<code>INCLUDE_XSYSACE_AUTOMOUNT</code>	Controls whether the System ACE filesystem is mounted at boot time using the next two <code>SYSACE_</code> constants defined in this table. This constant affects only Project builds.
<code>SYSACE_AUTOMOUNT_POINT</code>	Default mount point used when <code>INCLUDE_-XSYSACE_AUTOMOUNT</code> is defined in Project builds.
<code>SYSACE_AUTOMOUNT_PARTITION</code>	Default partition used when <code>INCLUDE_XSYSACE_-AUTOMOUNT</code> is defined in Project builds.
<code>INCLUDE_BOOT_FLASH</code>	Sets the BSP up to boot from Flash memory. If not defined, then SystemACE is assumed to be the bootstrap device.

Table 8-18: Custom BSP Options <Italic>(Continued)

Option	Description
SYS_US_DELAY_BIAS	Adds the specified number of microseconds to the delay parameter in <code>sysUsDelay()</code> . This option can be used to cancel out overhead.
INCLUDE_EMAC_PHY_RESET_-AT_BOOT	Controls whether the Ethernet PHY is reset at boot time. In the Project GUI, this is a parameter under the emac component and can be found under <i>hardware->peripherals->IP CSP-> Ethernet Core</i> . Set to TRUE to enable, FALSE to disable.
SYS_GPIO_SWITCH_DEBOUNCE_TICKS	Sampling interval used by function <code>sysSwitchReadState()</code> when attempting to debounce switches. Units are in clock ticks.
SYS_LCD_INIT_DISPLAY	If this character constant is defined, then it's value will be written to the LCD display at boot time. Requires GPIO support (<code>INCLUDE_XGPIO</code>).

Release History

Version 1.2/0 - January 10, 2003

First pre-release for Tornado 2.0. This is a beta release that does not include support for all HW. Note that testing has been done on the ML3 evaluation board as opposed to the MDFG456. In other words, this version of the BSP has never been run on the MDFG456 hardware. The SEG IP bitstream is used for this load.

HW Supported

- Main board UART (console). IP core is UartLite.
- Main board SDRAM 32MB
- Main board LEDs (1-4)
- Main board push button switches (1-3)
- Main board DIP switch (1-8)
- Main board LCD display
- P160 daughter board UART. IP core is a UartLite.
- P160 daughter board Flash/SRAM.
- P160 daughter board Ethernet. IP core is a Emac.

HW Not supported

- System ACE daughter board.
- P160 daughter board IIC header
- P160 daughter board SPI header
- P160 daughter board USB port.
- P160 daughter board PS/2 port.

Errata

1. System mode debugging through the END connection does not work.
2. Serial port usage as the WDB target connection does not work. Serial port polling mode does not seem to work.

3. Rev1 evaluation boards have mis-wired SDRAM that requires all accesses occur in 32 bit divisible quantities. This problem is worked around by enabling the data cache at bootstrap time and leaving it on. Instruction fetches always occur in 32 bit divisible quantities.
4. Since System ACE is not supported in this revision (it will be in future revisions), the only way to get a VxWorks image downloaded into the target is via an emulator.
5. Ethernet links have been iffy on two sample Rev1 boards this BSP has been tested on. If the link LED on the P160 board does not illuminate after applying power to the target (assuming there is a cable connected between the RJ45 jack and another network device), then try reseating the P160, emulator connector, and RS232 connectors. Once the link is established, it seems to remain that way.
6. Use non null-modem cabling for the RS-232 serial ports.
7. System not tested with MMU enabled.

These BSP on Other Boards

The BSPs discussed here can be used directly on other development hardware such as the Xilinx AFX Evaluation board. Modifications will be required to the BSP to account for a differing list of CSP peripherals and the amount and type of RAM.

First step is to remove all unsupported peripheral drivers from the BSP. This can be done by following steps in the section titled **Configuration, page 133**.

Next step is to change the amount and type of RAM the BSP recognizes. This can be done by editing the constant `LOCAL_MEM_SIZE`. If not using the Tornado project facility, this constant can be modified by editing its definition in `config.h`. If using the Tornado Project facility, this constant can be modified by changing its property definition in the memory folder.

Other areas to watch out for are constants defined by `xparameters.h`. These constants must match the VirtexII Pro bitstream. If they do not then all kinds of problems can be expected such as bus errors. Key constants to watch out for:

- `XPAR_CPU_PPC405_CORE_CLOCK_FREQ_HZ`
- `XPAR_UARTNS550_0_CLOCK_HZ`
- `XPAR_<device>_BASEADDR`
- `XPAR_INTC_0_<device>_X_VEC_ID`

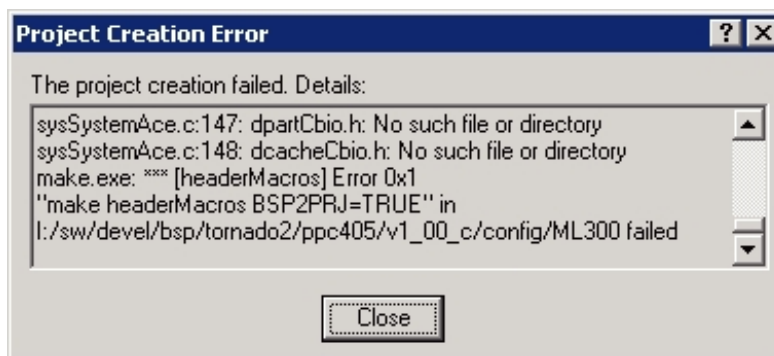
Trouble-Shooting

Project Creation

Issues seen when creating a Tornado Project based on the ML300 BSP.

"Project Creation Error" Dialog Pop-up

Scroll to the end of the box and if it contains error messages complaining about missing header files `dpartCbio.h` and `dcacheCbio.h`, then you don't have the DosFS 2.0 libraries installed in your system.



SingleStep

Issues seen when using SingleStep with this BSP.

Source browser not displaying source code at addresses where source code should be

Try to rebuild everything with the `-gdwarf` compiler option.

Tornado Crosswind debugger

Issues seen when using Tornado's IDE debugger with this BSP.

Source browser not displaying source code at addresses where source code should be

Is the `-gdwarf` option enabled in the compiler? Try to rebuild everything with the `-g` compiler option.

Target Shell Issues

Issues seen when using the built-in target shell.

Relocation value does not fit in 24 bits message from Loader

This is seen when a system contains more than 32MB of memory. Recompile your source code using the `-mlongcall` compiler option.

Ethernet Issues

Issues seen when integrating/using the XEmac Ethernet adapter

Network interface xemac unknown. Message from console at boot

There are multiple causes to this problem.

1. Did you compile the XEmac component into the BSP? In the Tornado Project facility, check that both `hardware->peripherals->IP CSP->Ethernet EMAC` and `EMAC END` components are included. On command line BSP builds, is `INCLUDE_XEMAC` and `INCLUDE_XEMAC_END` declared in `ip_config.h` and not `#undef'd` anywhere.
2. In the Tornado Project facility, if you remove then later restore network support, Project fails to restore "END" driver support. Check folder `network components->network devices` and verify that both `END attach interface` and `END interface support` components are included.

References

- VxWorks 5.4 Programmer's Guide

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
02/07/03	1.0	Combined IPSPEC124 and 138 into this document

Device Driver Summary

Summary

A summary of each device driver is provided with a link to its main header file. In addition, building block components are described. A hardware-to-software driver cross-reference table is also provided.

Device Driver Reference

ATM Controller

The Asynchronous Transfer Mode (ATM) Controller driver resides in the *atmc* subdirectory. Details of the driver can be found in the [xatmc.h](#) header file

Ethernet 10/100 MAC

The Ethernet 10/100 MAC driver resides in the *emac* subdirectory. Details of the driver can be found in the [xemac.h](#) header file.

Ethernet 10/100 MAC Lite

The Ethernet 10/100 MAC Lite driver resides in the *emac_lite* subdirectory. Details of the driver can be found in the [xemac_lite.h](#) header file.

External Memory Controller

The External Memory Controller driver resides in the *emc* subdirectory. Details of the driver can be found in the [xemc.h](#) header file.

General Purpose I/O

The General Purpose I/O driver resides in the *gpio* subdirectory. Details of the driver can be found in the [xgpio.h](#) header file.

HDLC

The HDLC driver resides in the *hdlc* subdirectory. Details of the driver can be found in the [xhdlc.h](#) header file.

Intel StrataFlash

The Intel StrataFlash driver resides in the *flash* subdirectory. Details of the driver can be found in the [xflash.h](#) header file.

Inter-Integrated Circuit (IIC)

The IIC driver resides in the `iic` subdirectory. Details of the driver can be found in the [xiic.h](#) header file.

Interrupt Controller

The Interrupt Controller driver resides in the `intc` subdirectory. Details of the driver can be found in the [xintc.h](#) header file.

OPB Arbiter

The OPB Arbiter driver resides in the `opb_arbiter` subdirectory. Details of the driver can be found in the [xopb_arbiter.h](#) header file.

OPB to PLB Bridge

The OPB to PLB bridge driver resides in the `opb2plb` subdirectory. Details of the driver can be found in the [xopb2plb.h](#) header file.

PLB Arbiter

The PLB arbiter driver resides in the `plbarb` subdirectory. Details of the driver can be found in the [xplbarb.h](#) header file.

PLB to OPB Bridge

The PLB to OPB bridge driver resides in the `plb2opb` subdirectory. Details of the driver can be found in the [xplb2opb.h](#) header file.

Rapid I/O

The Rapid I/O driver resides in the `rapidio` subdirectory. Details of the 0 low leve driver can be found in the [xrapidio_1.h](#) header file

Serial Peripheral Interface (SPI)

The SPI driver resides in the `spi` subdirectory. Details of the driver can be found in the [xspi.h](#) header file.

System ACE

The System ACE driver resides in the `sysace` subdirectory. Details of the driver can be found in the [xsysace.h](#) header file.

Timer/Counter

The Timer/Counter driver resides in the `tmrctr` subdirectory. Details of the driver can be found in the [xtmrctr.h](#) header file.

UART Lite

The UART Lite driver resides in the `uartlite` subdirectory. Details of the driver can be found in the UART Lite Driver Datasheet and in the [xuartlite.h](#) header file.

UART 16450/16550

The UART 16450/16550 driver resides in the `uartns550` subdirectory. Details of the driver can be found in the [xuartns550.h](#) header file.

Watchdog Timer/Timebase

The Watchdog Timer/Timebase driver resides in the `wdttb` subdirectory. Details of the driver can be found in the [xwdttb.h](#) header file.

Building Block Components

Common

Common components reside in the `common` subdirectory and comprise a collection of header files and ".c" files that are commonly used by all device drivers and application code. Included in this collection are: [xstatus.h](#), which contains the identifiers for Xilinx status codes; [xparameters.h](#), which contains the identifiers for the driver configurations and memory map; and [xbasic_types.h](#), which contains identifiers for primitive data types and commonly used constants.

CPU/CPU_PPC405

CPU components reside in the `cpu[_ppc405]` subdirectory and comprise I/O functions specific to a processor. These I/O functions are defined in [xio.h](#). These functions are used by drivers and are not intended for external use.

IPIF

IPIF components reside in the `ipif` subdirectory and comprise functions related to the IP Interface (IPIF) interrupt control logic. Since most devices are built with IPIF, drivers utilize this common source code to prevent duplication of code within the drivers. These functions are used by drivers and are not intended for external use.

DMA

DMA components reside in the `dma` subdirectory and comprise functions used for Direct Memory Access (DMA). Both simple DMA and scatter-gather DMA are supported.

Packet FIFO

Packet FIFO components reside in the `packet_fifo` subdirectory and comprise functions used for packet FIFO control. Packet FIFOs are typically used by devices that process and potentially retransmit packets, such as Ethernet and ATM. These functions are used by drivers and are not intended for external use.

Hardware/Software Cross Reference

Table 9-1: Hardware and Software Cross Reference

Hardware Device	Software Driver
DCR Bus Structure	XIo
DCR Interrupt Controller (INTC)	XIntc
OCM Packet Processing Engine	
OPB <-> PCI Full Bridge	XPci
OPB 10/100M Ethernet Controller	XEmac

Table 9-1: Hardware and Software Cross Reference <Italic>(Continued)

Hardware Device	Software Driver
OPB 10/100M Ethernet Controller - Lite	XEmacLite
OPB 16450 UART Controller	XUartNs550
OPB 16550 UART Controller	XUartNs550
OPB Arbiter and Bus Structure	XOpbArb
OPB ATM Utopia Level 2 Master	XAtmc
OPB ATM Utopia Level 2 Slave	XAtmc
OPB External Memory Controller (EMC)	XEmc
OPB GPIO Controller	XGpio
OPB IIC Master and Slave Bus Controller	XIic
OPB Interrupt Controller (INTC)	XIntc
OPB IPIF	XIpIf
OPB JTAG UART	XUartLite
OPB PS/2 Controller	
OPB Single Channel HDLC Controller	XHdlc
OPB SPI Master and Slave Bus Controller	XSpi
OPB TimeBase / WatchDog Timer	XWdtTb
OPB Timer / Counter	XTmrCtr
OPB Touchscreen Controller	
OPB UART - Lite	XUartLite
OPB2PLB Bridge	XOpb2Plb
PLB 1Gb Ethernet Controller	
PLB Arbiter and Bus Structure	XPlbArb
PLB External Memory Controller (EMC)	XEmc
PLB IPIF	XIpIf
PLB Packet Processing Engine	
PLB TFT VGA LCD Controller	
PLB UART-16450	XUartNs550
PLB UART-16550	XUartNs550
PLB2OPB Bridge	XPlb2Opb
RAPID IO	Xrapidio

Automatic Generation of Tornado 2.x (VxWorks 5.x) Board Support Packages

Overview

One of the key embedded system development activities is the development of the Board Support Package (BSP). Creation of a BSP can be a lengthy and tedious process that must be incurred every time the microprocessor complex (processor plus associated peripherals) changes. While managing these changes applies to any microprocessor-based project, the changes can come about more rapidly than ever with the advent of programmable System-on-Chip (SoC) hardware.

This document describes the BSP Generator (*BSPgen*), which automatically generates a customized BSP for various microprocessor, peripheral, and RTOS combinations. This tool enables embedded system designers to do the following:

- Substantially decrease development cycles (decrease time-to-market)
- Create a BSP that matches the application (customized BSP)
- Eliminate BSP design bugs (automatically created based on certified components)
- Enable application software developers (do not have to wait for BSP development)

BSPgen is currently used in conjunction with the Xilinx Embedded Development Kit (EDK). Using EDK, the user can choose to automatically create a BSP based on an embedded system just created. The BSP contains all the necessary support software for a system, including boot code, device drivers, and RTOS initialization. The BSP is customized based on the type of operating system, processor, and peripherals chosen by the user for the FPGA-based embedded system.

The only types of BSPs currently supported by *BSPgen* are for the WindRiver VxWorks 5.4/5.5 operating systems and Tornado 2.0.2/2.2 IDE, in conjunction with the IBM PowerPC 405 microprocessor core.

Tool/User Input

BSPgen requires a description of the embedded system in order to customize the BSP. The system description includes the type of processor(s) in the system, the types of peripherals and bus connections, interrupt connections, versions of peripherals and device drivers, and any other information needed to generate a functional BSP. This system description is typically provided by the tool that invokes *BSPgen*. But the user, of course, is ultimately responsible for input of the system through the tool.

In addition, information about the BSP to be generated is provided. This information includes the type of operating system, the directory location where the BSP will reside, and the name of the BSP.

The user may also have to decide which peripherals, or devices, are tightly integrated into the operating system. Tight integration typically means the device driver is connected

directly to the operating system through an OS interface, such as a network stack or a serial I/O interface. Other devices are loosely integrated into the OS, which means that although there is no standard interface to access the device, the user can access the device by using the device driver directly from the application.

Template-Based Approach

A set of BSP template files will be released with *BSPgen*. Every operating system supported will have a corresponding set of template files. These template files are used during creation of the BSP, making appropriate modifications based on the makeup of the FPGA-based embedded system.

If the user chooses not to automatically generate a BSP, these template files could be used as a reference for building a BSP from scratch.

Device Drivers

A set of device driver source files is released with the EDK tools and resides in an installation directory. During creation of a customized BSP, device driver source code is copied from this installation directory to the BSP directory. Only the source code pertaining to the devices built into the FPGA-based embedded system are copied. This copy provides the user with a self-contained, standalone BSP directory which can be modified by the user if necessary and/or relocated if necessary. If the user makes changes to the device driver source code for this BSP and sometime later wishes to back those changes out, the user can use the EDK tools to regenerate the BSP. Device driver source files are then recopied from the installation directory to the BSP.

Backups

If the directory location of the BSP contains existing files, these files are copied into a backup directory before being overwritten. This prevents the inadvertent loss of changes made by the user to BSP source files. The backup directory will reside within the BSP directory and will be named *backup<timestamp>*, where *<timestamp>* represents the current date and time.

Generating the Tornado BSP

Using the Embedded Development Kit (EDK)

Xilinx Platform Studio (XPS) is available in the EDK and is a graphical design entry and implementation tool for a PPC405- or MicroBlaze-based embedded system. This section describes the steps needed to invoke *BSPgen* and create a Tornado BSP using XPS.

1. Select the operating system and core clock frequency

In the software settings for the PPC instance, select the operating system using the Environment tab. In this case, we want to select a VxWorks5_x operating system. In addition, the Tornado BSP needs to know the frequency of the CPU. Enter it in the Core Clk Freq field in MHz.
2. Configure the VxWorks console device

If you intend to use a serial device, such as a Uart, as the VxWorks console, select or enter the instance name of the serial device as the STDIN/STDOUT peripheral. It is important to enter the same device for both STDIN and STDOUT. Currently, *BSPgen* supports only the Uart 16550/16450 and UartLite devices as VxWorks console devices.
3. Integrate the device drivers
 - a. Configure all drivers with Level 1 interface

In general, all drivers that you want accessible from your VxWorks application or tightly integrated into VxWorks need to be configured as Level 1 drivers. Only if there is no Level 1 interface for a driver will its Level 0 interface be made available in the Tornado BSP. Driver levels can be configured in the S/W Settings dialog box for each device

b. Connect to VxWorks

Some S/W Settings dialog boxes will provide a checkbox labeled “Connect to OS” or “Connect to VxWorks”. These are typically provided for devices that can be tightly integrated into the OS. If you want this device to be tightly integrated into the OS, select this checkbox. See the section **Device Integration** for more details on tight integration of devices.

c. Overwrite MDD Parameters

On the S/W Settings dialog box, there is a button labeled “MDD Params”. For certain devices there may be a device parameter that needs to be tailored in order for software to communicate correctly to that device. Be sure to click the “MDD Params” button and set any parameter values correctly based on your system. An example of this is the interrupt controller. If the interrupt controller device is attached to the PPC405’s DCR bus, then the USE_DCR parameter value in the intc MDD Params dialog box should be set to 1.

4. Generate the Tornado BSP

In the Tools menu of XPS, the menu item “Generate BSP for VxWorks” can be selected to invoke *BSPgen*. The output of this invocation is shown in the XPS output window. Warnings may be output for those devices that do not have Level 1 device driver interfaces selected. Once *BSPgen* is done, the resulting Tornado BSP should exist under the PPC405 instance subdirectory of the user’s EDK project. For example, if in XPS the user has named their PPC405 instance *myppc405*, then the Tornado BSP will reside at `<user project>/myppc405/bsp_myppc405`.

Using the Command Line and EDK Files

BSPgen supports a command-line interface in conjunction with files produced by the EDK. Although the user would typically invoke *BSPgen* using the EDK tools, there is nothing to prevent command-line invocation. The command-line interface requires system description files in the form of .mss/.mhs files that are output by the EDK tools. The command-line usage syntax is as follows:

```
Usage: bspgen -h <mhsfile> -s <mssfile> -p <project_path>
```

where:

```
-h <mhsfile>
```

Specifies the name of the .mhs file created by the EDK toolset. The .mhs file describes the hardware selected by the user for the embedded system.

```
-s <mssfile>
```

Specifies the name of the .mss file created by the EDK toolset. The .mss file describes the software, or device drivers, selected by the user and corresponding to the system hardware.

```
-p <project_path>
```

The absolute path of the user’s EDK project directory.

BSPgen makes use of the XILINX_EDK environment variable. It should be set to the installation directory of the EDK.

Note that the end user does not typically invoke *BSPgen*. Instead, the EDK tools invoke *BSPgen* using the appropriate interface.

The Tornado 2.x BSP

This section assumes the reader is familiar with WindRiver's Tornado 2.0.2 or 2.2 IDE. It describes the Tornado BSP output by *BSPgen*.

Integration with IDE

The automatically generated BSP is integrated into the Tornado IDE and Project facility. The BSP can be compiled from the command-line using the Tornado make tools, or from the Tornado Project facility (also referred to as the Tornado GUI). Once the BSP has been generated, the user can simply type *make vxWorks* from the command-line to compile a bootable RAM image. This assumes the Tornado environment has been previously set up, which can be done via the command-line using the *host/x86-win32/bin/torVars.bat* script if on a Windows platform. If using the Tornado Project facility, the user can create a project based on the newly generated BSP, then use the build environment provided through the GUI to compile the BSP.

In Tornado 2.2, the *diab* compiler is supported in addition to the *gnu* compiler. The Tornado BSP created by *BSPgen* has a Makefile that can be modified by the command-line user to use the *diab* compiler instead of the *gnu* compiler. Look for the make variable named *TOOLS* and set the value to "diab" instead of "gnu". If using the Tornado Project facility, the user can select the desired tool when the project is first created.

The file *50<csp_name>.cdf* resides in the BSP directory and is tailored during creation of the BSP. This file integrates the CSP device drivers into the Tornado GUI. CSPs hook themselves into the BSP at the **hardware/peripherals** sub-folder. Below this is a **Core library** folder and individual device driver folders. [Figure 1](#) shows the look of the GUI given the CSP name "IP".

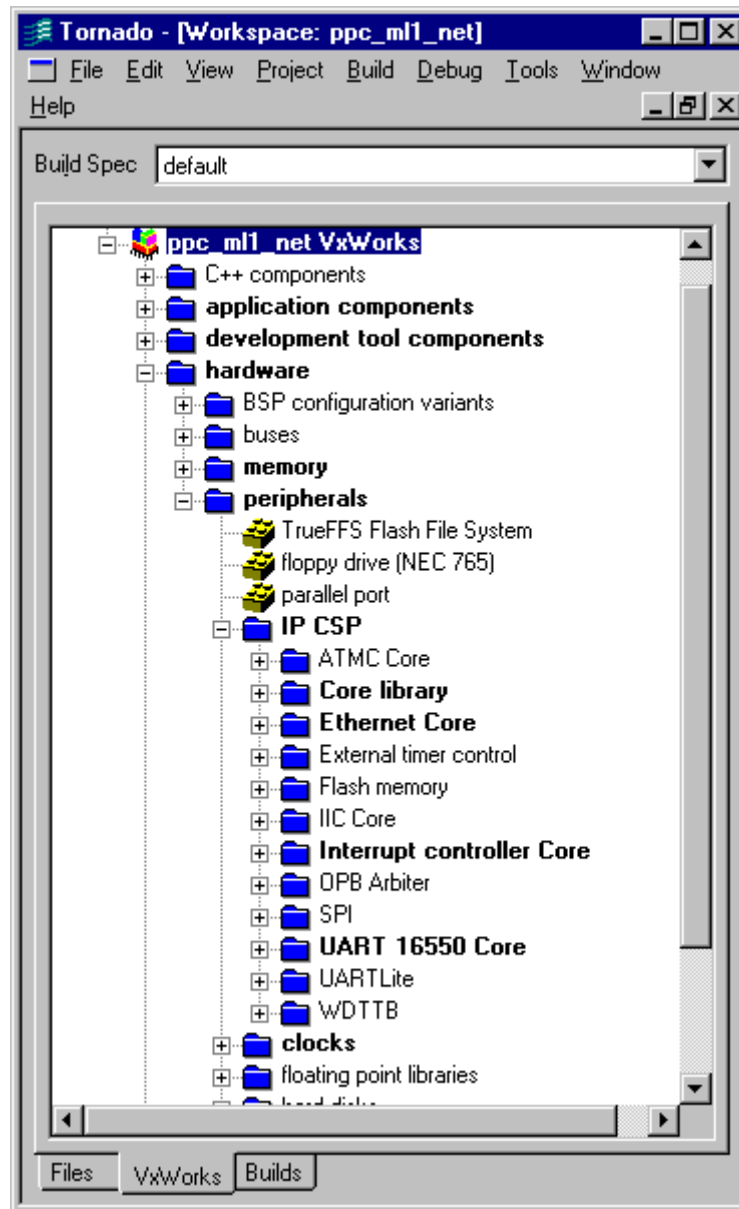


Figure 1: Tornado 2.x Project GUI - VxWorks

The “Files” tab of the Tornado Project GUI will also show a number of new files used to integrate the CSP device drivers into the Tornado build process. Once again, these files are automatically created by *BSPgen*. The user need only be aware of that the files exist.

These files are prefixed with the name of the CSP. [Figure 2](#) shows an example of the CSP build files.

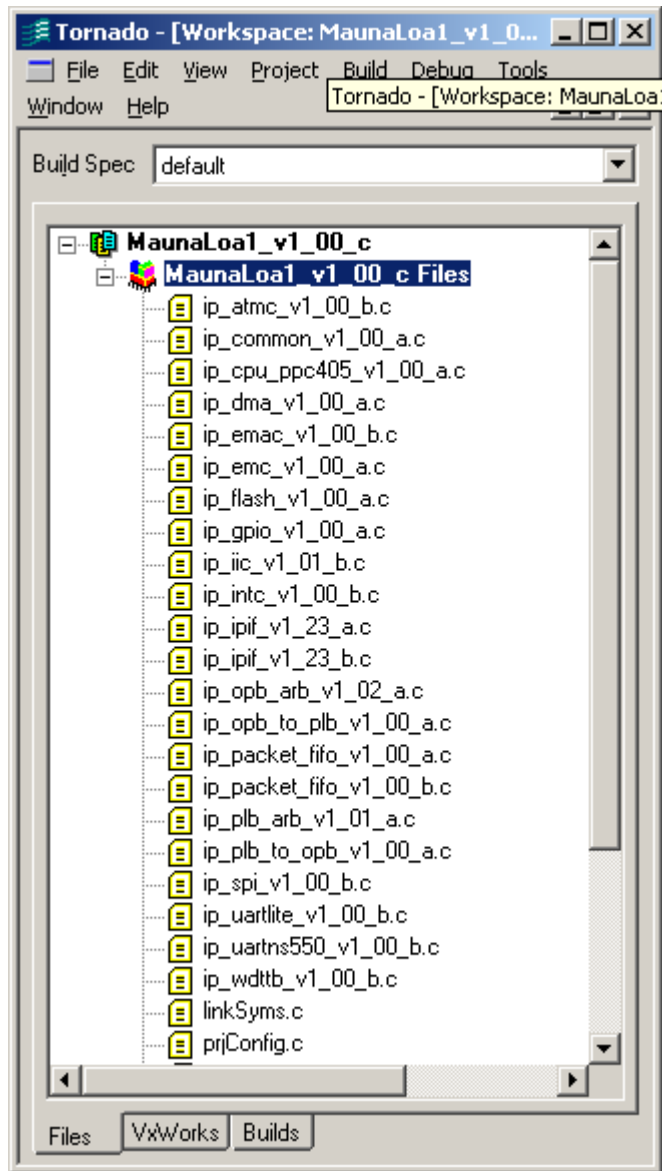


Figure 2: Tornado 2.x Project GUI - Files

Device Integration

Devices in the FPGA-based embedded system have varying degrees of integration with the VxWorks operating system. The degree of integration may be selectable by the user through the system generation tools. Below is a list of currently supported devices and their level of integration.

- A UART 16450/16550/Lite can be integrated into the VxWorks Serial I/O (SIO) interface. This makes the UART available for file I/O and printf/stdio. Only one UART device can be selected as the console, where standard I/O (stdin, stdout, and stderr) is directed. Reference the *sysSerial.c* file of the BSP to see details of this integration.
- An Ethernet 10/100 MAC can be integrated into the VxWorks Enhanced Network Driver (END) interface. This makes it available to the VxWorks network stack and thus socket-level applications. Reference the *configNet.h* and *sysNet.c* files of the BSP to

see details of this integration.

- An Interrupt controller can be connected to the VxWorks exception handling and the PowerPC 405 external non-critical interrupt pin. *BSPgen* does not currently handle interrupt controller integration for the critical interrupt pin of the PPC405. However, the user is always free to manually add this integration in the *sysInterrupt.c* file of the BSP.
- A System ACE controller can be connected to VxWorks as a block device, allowing the user to attach a filesystem to the CompactFlash device connected to the System ACE controller. The user must manually call BSP functions to initialize the System ACE/CompactFlash as a block device and attach it to the DOS operating system. The functions currently available to the user are: *sysSystemAceInitFS()* and *sysSystemAceMount()*. Reference the file *sysSystemAce.c* in the BSP for more details.
- All other devices and associated device drivers are not tightly integrated into a VxWorks interface. Instead, they are loosely integrated and access to these devices is available by directly accessing the associated device drivers from the user's application.

Device Driver Location and BSP Directory Tree

The automatically generated BSP contains boot code, device driver code, and initialization code. The BSP resembles most other Tornado BSPs except for the placement of device driver code. Off-the-shelf device driver code distributed with the Tornado IDE typically resides in the *target/src/drv* directory in the Tornado distribution directory. Device driver code for a BSP that is automatically generated resides in the BSP directory itself. This minor deviation is due to the dynamic nature of FPGA-based embedded system. Since the FPGA-based embedded system can be reprogrammed with new or changed IP, the device driver configuration can change, calling for a more dynamic placement of device driver source files.

The directory tree for the automatically generated BSP is shown below.

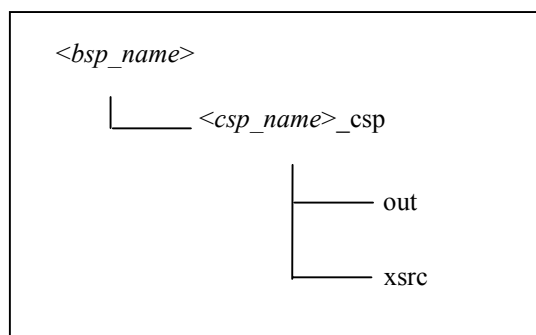


Figure 3: **BSP Directory Tree**

The top-level directory is named according to the name of the BSP the user provides. The customized BSP source files reside in this directory. There is a subdirectory within the BSP directory named according to the name of the CSP the user provides. The CSP directory contains two subdirectories. The *xsrc* subdirectory contains all the device driver related source files. The *out* subdirectory is created during the build process and only exists if building from the command-line. It contains files generated during the compilation or build process (e.g., the *.o* files for each driver source file). If building from the Project facility, the files generated during the build process reside at `$PRJ_DIR/$BUILD_SPEC/<csp_name>_csp`.

Limitations

The automatically generated BSP should be considered a good starting point for the user, but should not be expected to meet all the user's needs. Due to the potential complexities of a BSP, the variety of features that can be included in a BSP, and the support necessary for board devices external to the FPGA, the automatically generated BSP will likely require enhancements by the user. However, the generated BSP will be compilable and will contain all the necessary device drivers represented in the FPGA-based embedded system. Some of commonly used devices are also integrated with the operating system.