LULEÅ UNIVERSITY OF TECHNOLOGY

2004:075 CIV

MASTER'S THESIS

Evaluating Xilinx MicroBlaze for Network SoC Applications

PETER MAGNUSSON

MASTER OF SCIENCE PROGRAMME

Department of Computer Science and Electrical Engineering
EISLAB - Embedded Internet Systems Laboratory

2004:075 CIV • ISSN: 1402 - 1617 • ISRN: LTU - EX - - 04/75 - - SE

Evaluating Xilinx MicroBlaze for Network SoC solutions

Master's Thesis in Computer Engineering

Peter Magnusson
petmag-8@sm.luth.se

10th January 2004

Abstract

This thesis aims to create a System on Chip (SoC) solution for various network devices. A solution with network peripherals, processor core and network software in a single chip is designed and evaluated.

Typical applications of a network System on Chip include Ethernet switches, Internet-enabled embedded systems, small Internet Protocol clients for handheld devices and simple Internet gateways.

The the solution utilizes the Xilinx MicroBlaze soft processor core, MicroBlaze Development Kit, IBM CoreConnect On-Chip Peripheral Bus (OPB) peripherals and Xilinx Virtex Field Programmable Gate Array (FPGA).

Acknowledgment

The evaluation of the Xilinx MicroBlaze has been performed as a Master Thesis work in Computer Science and Engineering. The work was performed at the Department of Computer Science and Electrical Engineering (CSEE) and Embedded Internet Systems Laboratory (EISLAB) at Luleå University of Technology.

I wish to thank

- Per Lindgren (PhD) for supervising my thesis.
- Jonas Thor (PhD student) for feedback on various computer engineering topics.
- Jens Eliasson (MScE) for various MicroBlaze discussions.
- Students Frederik Schmid, Jan Dahlberg, Johan Mattsson, Stefan Nilsson and Jimmie Wiklander for reusing and verifying my Ethernet MAC.
- Students Stefan Nilsson, Frederik Schmid and Jimmie Wiklander for MicroBlaze lwIP implementation.
- Jens Eliasson (MScE), Tim Johansson (MScE) and Sara Lidqvist (MScE) for proof-reading my thesis.
- Xilinx Inc for permitting reprint of figures originally published by Xilinx Inc.
- IBM for permitting reprint of figures originally published by International Business Machines Corporation (IBM).

During the thesis work, I have taught MicroBlaze based System on Chip (SoC) development to MScE students in the Project in Digital Synthesis course at CSEE. Additional credits goes to these students for valuable input.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Objectives

The overall goal of this thesis is to evaluate the Xilinx MicroBlaze soft core processor for single chip network solutions, i.e. create a MicroBlaze based System On Chip (SoC). The SoC should be based on a well verified and evaluated platform. Possible future improvement should be well documented. To reach these overall goals, a number of objectives were identified:

- The MicroBlaze Development Kit should be evaluated. The evaluation should make clear distinction between peripherals, software development tools and tools for platform tailoring.
- MicroBlaze's bus architecture (CoreConnect On-Chip Peripheral Bus) should be evaluated.
- Network connectivity should be provided through customized Ethernet peripherals. Both MII and RMII connectivity should be provided.
- The XilNet TCP/IP stack for MicroBlaze should be evaluated.
- The lwIP TCP/IP stack should be ported to MicroBlaze.
- Availability of other network software for MicroBlaze should be investigated.

1.2 Method

Simple MicroBlaze platforms were created using MicroBlaze Development Kit. MicroBlaze development tutorials [XMB3, XMB4] were used to learn developing MicroBlaze platforms.

Ethernet peripherals were developed network enable the MicroBlaze platforms. These peripherals were developed for two purposes:

- to network enable MicroBlaze (MDK does not include network peripherals)
- to further verify that custom designed peripherals and platforms function properly (network applications are easy to monitor remotely for long periods of time)

Verification through re-use was utilized. Five students used, and in one case modified, the Ethernet peripherals in the Project in Digital Synthesis course at Luleå University of Technology.

Advanced platform designs were investigated. The investigated issues included:

- Custom FPGA buffer-pad insertion
- Custom OPB bus bindings (e.g. dedicated buses, bus bridges)

Re-use of other students projects. Fredrik Schmid, Jimmie Wiklander and Stefan Nilsson ported lwIP to MicroBlaze for their Digital Synthesis projects. In their project they used the Ethernet peripherals and a OPB memory controller (developed by Jens Eliasson) for the XESS XSV prototyping boards. Their results were incorporated in this thesis.

Publications were used to find facts and support statements. A problem which occurred was limited availability of MicroBlaze related publications. Official publications by Xilinx Inc, researchers and other credible sources have been used whenever possible. Sometimes open forums such as *Xilinx Embedded Processor Forum* have been used in lack of more credible sources.

1.3 Limitations

1.3.1 Time

This Master Thesis work has been performed during a period of 6 months.

1.3.2 Software

MicroBlaze Development Kit (MDK) 2.2 (Service Pack 2) has been used for this evaluation. It is the most recent release to date.

1.3.3 Hardware

XESS XSV-100 / XSV-800 Prototyping Board These are Xilinx Virtex based prototyping board. XSV-100 is equipped with a Virtex 100 FPGA, XSV-800 with a Virtex 800 FPGA. The Virtex chip is connected to a large set of interfaces, including:

- Programmable 100 MHz oscillator, $f = 100/n$ MHz, $n = 1, 2, 3, 4, ...$
- An MII PHY connected to a RJ45 twisted pair cable socket.
- Two 512K$*$16-Bit SRAM memory banks.

Platinum Virtex-E Prototyping Board This is a prototyping board developed at Luleå University of Technology and intended to be available to this thesis work. Unfortunately, it was not verified in time to be used in the thesis.

Platinum connects the Virtex-E chip with the following interfaces:

- Four RMII PHYs, connected to twisted pair cable sockets.
- One Motorola Microcontroller.

1.4 Thesis outline

Chapter 2 covers the technologies which are used and referred to in this thesis.

Chapter 3 introduces Xilinx MicroBlaze and the MDK software.

Chapter 4 investigates the possibility of using Xilinx MicroBlaze in a network SoC.

Chapter 5 outlines the design of a set of Ethernet peripherals for Xilinx MicroBlaze.

Chapter 6 compares the Ethernet peripherals designed to a number of other Ethernet peripherals.

Chapter 7 covers network software, such as TCP/IP stacks, available to Xilinx MicroBlaze.

Chapter 8 goes back to the objectives states in chapter 1 and how those objectives were met.

Chapter 9 discusses the results, suggest some topics for future work and gives a number of conclusions drawn from the thesis.

Chapter 2

Technology & Background

2.1 System on Chip (SoC) solutions

System on Chip (SoC) refers to devices where all essential parts of a computing systems have been integrated in a single circuit. Common SoC design goals include:

- reduced power dissipation
- reduced chip interconnects
- reduced device size

A typical SoC includes one (or many) processor core(s), an arbitrary number of peripherals, some on-chip memory and a bus architecture which interconnects all these devices. The System on Chip design goal is that only one circuit should required for an application. In practice a SoC may also contain a large set of I/O interfaces to other circuits, for instance:

- memory modules
- off-chip peripherals
- radio transceivers
- network interfaces

As SoCs usually are designed with a limited set of applications in mind, they tend to need less processing power than a general purpose computer. While a modern workstation operates at clock frequencies in the range of 500 MHz - 3 GHz, the SoC CPU might operate at just a few megahertz. An ideal SoC processor core is operating at the minimum clock frequency needed to properly perform the desired task. By utilizing a low clock frequency the power consumption and chip temperature is reduced. This allows SoCs to operate with less cooling devices and better battery/power utilization.

2.1.1 System on Chip or Microcontroller?

[FOLDOC] define Microcontroller as:

> A microprocessor on a single integrated circuit intended to operate as an embedded system. As well as a CPU, a microcontroller typically includes small amounts of RAM and PROM and timers and I/O ports. An example is the Intel 8751.

Distinguishing between microcontrollers from SoCs is hard; the words are synonyms. The word SoC is frequently used in reference to designs which are customized for a limited set of applications, while microcontroller tend to be used in reference to more general purpose embedded designs.

2.2 Application Specific Integrated Circuit (ASIC)

Application Specific Integrated Circuit (ASIC) is one of the most common chip types. An ASIC may implement simple designs ("application" in ASIC terms) but also large designs such as SoCs.

An ASIC is designed for a specific application. An ASIC can therefor be customized for reduced power dissipation, less chip area or greater clock frequencies. ASICs usually have low mass production costs.

ASIC drawbacks are low reconfigurability, long design phases and high startup costs.

This makes ASIC well suited for large scale manufacturing of well verified designs, but not well suited for prototypes.

2.3 Programmable logic device (PLD)

Programmable logic devices are chips which can be programmed to behave as an arbitrary design. A PLD may be programmed to implement something as simple as a small net of combinatorial logic, but it may also implement large designs such as a SoC.

2.3.1 Field Programmable Gate Array (FPGA)

Field Programmable Gate Array (FPGA) is a type of programmable logic devices. FPGA is a generic architecture consisting of configurable logic blocks and programmable interconnections. Several FPGAs contain enough logic to implement SoCs and other large designs.

FPGAs are not optimized for a specific application, and therefore they may consume more power or implement a design less efficient than an ASIC. Price per chip is high. An FPGA is however easy to reprogram, which shortens design cycles and allows early real world tests.

This makes FPGAs well suited for prototypes and small production volumes. FPGA may also be used for applications which are not of ASIC production quality. An example of this is first generation manufacturing where standards and applications are subject to change.

2.4 Hardware Description Language (HDL)

Hardware Description Languages (HDL) provide ways to describe hardware. Hardware Description Languages can be used for difference purposes.

Synthesis. A common usage of HDL is to describe how hardware should be implemented. Such descriptions can be translated into technology-dependent netlists[1] by software. The process of translating HDL descriptions to netlists is known as synthesis.

VHSIC Hardware Description Language (VHDL) or "Very High Speed Integrated Circuit Hardware Description Language" is a commonly used HDL. It is designed for military and space applications, but is also commonly used in academic and commercial hardware project. VHDL is similar to software languages such as Ada and Pascal.

2.5 Processor cores

A processor core refers to a processor, *excluding*[2] any peripherals it is used with. An traditional processor core resides in a dedicated processor chip. In SoC designs, one or more processor cores are integrated with peripherals on a single chip.

2.5.1 Soft, firm and hard cores

The terms soft, firm and hard cores are originally ASIC manufacturing jargon.

- "Soft core" refer to cores delivered as a technology dependent gate-level netlist or HDL source code.
- "Firm core" refer to cores delivered as a library element.
- "Hard core" refer to cores which has a fixed physical layout and is incorporated into the design as a standard cell.

Firm and hard cores mainly apply to ASIC design. Soft cores are commonly used with programmable logic as well.

2.5.2 Instruction Set Architectures

An Instruction Set Architecture is a definition of how processor should perform an instruction. An instruction is a very short and basic command to the processor, such as "add X with Y and store in Z" or "load X from memory Z".

Reduced Instruction Set Computer (RISC) refers to instruction set architectures with all or most of the following properties:

- rapid execution of a small instruction set with simple instructions
- uniform instruction length
- all processor registers are general purpose
- simple addressing modes.

RISC architectures are commonly used in microcontrollers and System on Chip cores.

[1] Netlists describe how ASIC or FPGA building blocks are interconnected and configured.
[2] As opposed to a microcontroller core which may include peripherals.

2.5.3 Soft Processors

Soft Processors are "soft core" processors - delivered as technology dependent netlists or HDL source code for synthesis. Soft processors have recently gained a lot of popularity. The popularity appears to be especially strong among FPGA developers. This thanks to several factors:

- Performance increases (soft cores are now utilizing FPGA/ASICs better).
- Increased performance/price ratio on FPGAs.
- Increased availability of both commercial and academic cores.
- Free Soft Processors have been released by teams consisting of professionals, academics and enthusiasts [FPGACPU, OPENCORES].

2.5.4 Why are processor cores and software used?

Almost any application may be implemented, without a processor or software, in application specific logic (e.g. ASIC) or programmable logic (e.g. FPGA). There two major reasons to utilize a processor core and run software on it:

Simplicity. Software often shorten design times and tend to be easier to develop. Software can be partially tested and verified in other environments before they are implemented in a particular processor architecture.

Simple, fast and remote upgrades. Software can often be upgraded during operation without system downtime[3]. Operating system software often support remote administration and can terminate and load a new application in a few microseconds. Programmable logic and application specific logic have different upgrade characteristics:

- Programmable logic (e.g. FPGA) usually take some time, ranging from a few seconds to a couple of minutes, to update. Programmable logic is typically not upgraded by remote, although it is possible [XAPP632].
- ASICs are very hard to update. Support for changes must have been anticipated in advance (several configuration registers or programmable logic blocks included). If changes have not been anticipated, a new ASIC may have to be manufactured.

Software and processor cores are preferred to pure hardware (PLD/ASIC) solutions. This is because software adds flexibility, shortens development cycles, allows faster reconfigurations and simplifies debugging.

2.6 IBM CoreConnect Bus architecture

IBM CoreConnect [IBM1] architecture is a small set of buses intended for SoC designs. The design goal of CoreConnect is to provide different buses for different types

[3] Downtime is time which a system is off-line, unresponsive or otherwise useless.

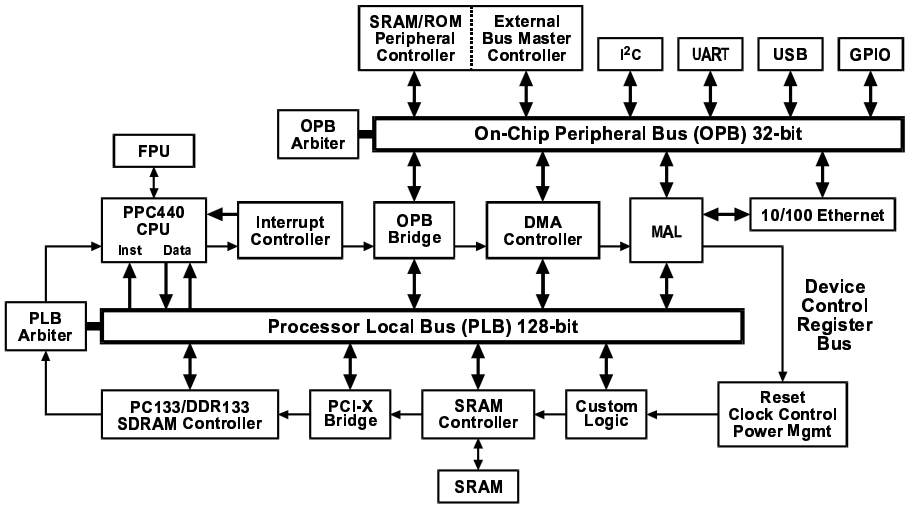
of cores. By providing different buses, the overall design may be optimized for performance while simple peripherals may be optimized for simplicity.

IBM CoreConnect features three buses:

- Processor Local Bus (PLB): a high performance bus for interconnecting fast processor cores and high performance peripherals (PCI interfaces, memory interfaces etc)
- On-Chip Peripheral Bus (OPB): a simple bus intended for use with peripherals which are slow or otherwise unsuited for PLB.
- Device Control Register Bus (DCR): a simple bus intended to distribute register values in an efficient manner.

An IBM CoreConnect system may utilize one, two or all three buses in a single SoC. Figure 2.1 shows a system utilizing all three buses.

Figure 2.1: CoreConnect based SoC



(from [IBM1] page 1)

A system utilizing two or three CoreConnect buses may use different system clocks - often OPB uses a slower clock than PLB. PLB and OPB may be interconnected using OPB-to-PLB and PLB-to-OPB bridges. The CoreConnect buses share important characteristics:

- CoreConnect is fully synchronous
- CoreConnect does not require tri-state[4] drivers

[4]Tri-state drivers are logic which may output boolean values ('0', '1') or output no signal. Tri-state drivers are notorious for causing design problems, but allows several drivers to share a single wire. To use or not to use tri-state drivers in bus design is a much debated subject.

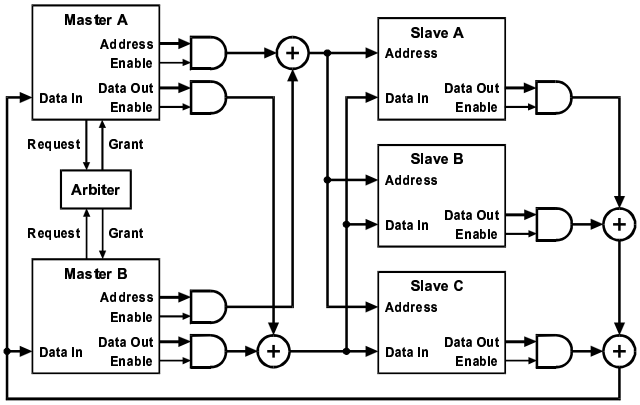
IBM CoreConnect is heavily customizable and several CoreConnect architecture parameters may differ between different implementations. Therefor IBM CoreConnect cores may be incompatible unless designed for compliance with a specific CoreConnect implementation. Important CoreConnect implementations include:

- IBM Blue Logic^{TM} Core library for ASIC SoC design.
- Xilinx OPB for Xilinx FPGA System on Chip design.

2.6.1 On-Chip Peripheral Bus (OPB)

The IBM CoreConnect On-Chip Peripheral Bus[5] is an easy to use bus. OPB allows an arbitrary number of masters to read from / write to an arbitrary number of slaves. Figure 2.2 shows a small IBM CoreConnect OPB System.

Figure 2.2: IBM CoreConnect OPB Physical Implementation



(from [IBM1] page 6)

The bus includes a data bus and an address bus. These buses are usually 32-Bit, 64-Bit or 128-Bit. A typical OPB device use the most significant bits of the address bus to determine weather it was selected or not, and the least significant bits to determine which register (internal address) was accessed.

A normal OPB access can be performed in one cycle. Slow OPB devices may respond in an arbitrary number of cycles if issuing a “Timeout Suppress” signal.

When several OPB masters share a bus, an OPB Arbiter is used to grant exclusive bus access. In these systems, a master may have to wait an arbitrary number of cycles until the bus is idle. The OPB Arbiter itself may also introduce a short mandatory delay before it grant access. An OPB master may utilize “Bus Lock” (also referred to as sequential access or bursts) to make several slave accesses per arbitrated bus grant. Utilization of sequential access keeps arbitration overhead to a minimum.

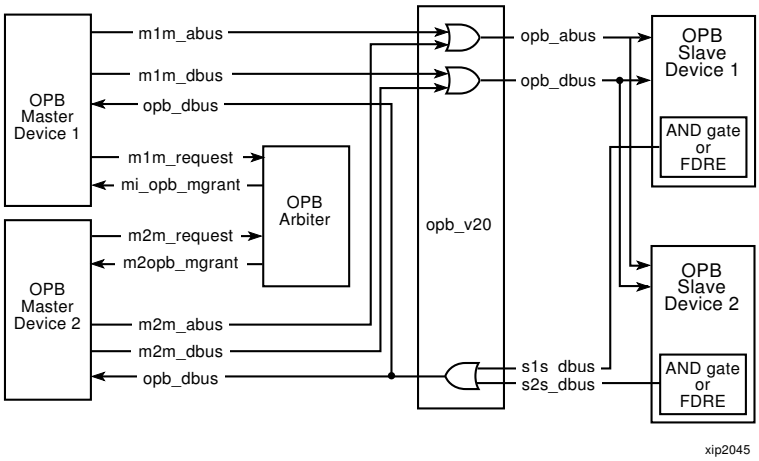
[5] Referred to as “OPB” or “OPB Bus” in this paper.

2.6.2 Xilinx OPB

Xilinx implements the IBM CoreConnect On-Chip Peripheral Bus with most configurable OPB parameters set to specific values. Both address and data bus is 32-Bit. Byte enable signals allow 1, 2 and 4 byte accesses. Xilinx OPB Arbiter supports up to 16 masters and allow slow slaves to respond in up to 16 cycles without issuing a "Timeout suppress" signal.

Figure 2.3 shows a small Xilinx OPB system. It is important to note that Xilinx OPB is based on OR-gates and does not implement "enable"-outputs[6].

Figure 2.3: Xilinx OPB physical implementation



Because Xilinx OPB devices share the same OPB parameters they are compatible. Xilinx OPB devices include Xilinx MicroBlaze soft core processor, Xilinx PicoBlaze soft core processor and a large set of OPB peripherals. Xilinx also provides a bridge which allows the IBM PPC 440 processor core, built into recent Xilinx Virtex II Pro FPGAs, to be interconnected with the OPB devices.

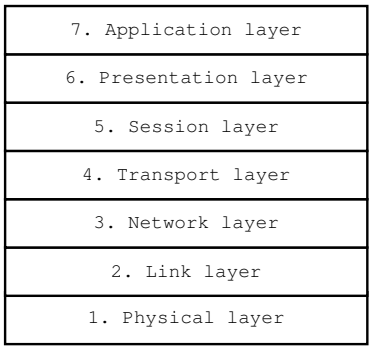
Xilinx provide "OPB Usage Notes" [XMB1] for mixed systems where Xilinx OPB and other OPB implementations are connected. With aid of these usage notes, a broad range of non-Xilinx OPB devices may be interconnected with some effort.

2.7 Networks and the OSI reference model

The Open Systems Interconnect (OSI) reference model is an abstracted approach to computer networks. It divides network hardware and software into seven layers. The layers are shown in figure 2.4. Layer 1, the physical layer, defines how signal is transmitted in cables, optical fibers, air or space. Layer 2, the Link layer, defines how

[6]Xilinx OPB requires that all slave/master peripherals set all output signals to logic zero when not selected.

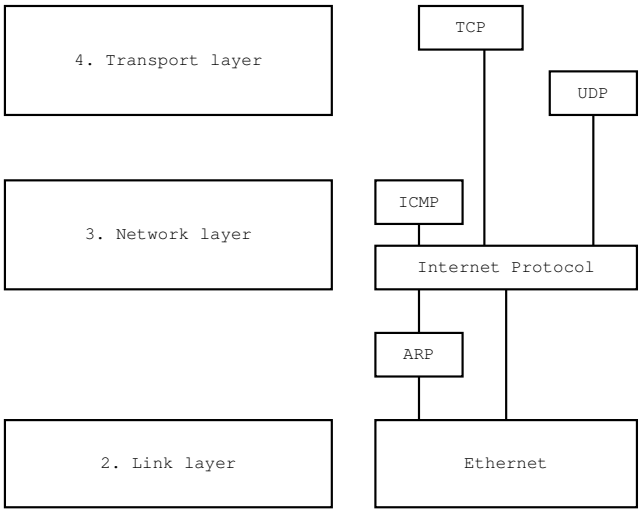
Figure 2.4: OSI layers



data should be transmitted between two devices. Layer 3, the network layer, defines how data should be transmitted between a network. This thesis deals mainly with layer 2 and 3.

The OSI model applied to Ethernet and Internet. Most network protocols can be consider in from an OSI point of view. Figure 2.5 visualizes the protocols used in thesis in an OSI view.

Figure 2.5: OSI view of Ethernet - IP network stacks



Ethernet is the link layer protocol.

IP (Internet Protocol) is the network layer protocol.

ICMP (Internet Control Message Protocol) is used to send control messages. An example of such messages is “No such network”, signaled when IP packets with non-existent destinations are detected.

ARP (Address Resolution Protocol) translates network layer addresses into link layer addresses. ARP is sometimes considered to be a part of the link layer protocols, but is better considered to be a glue which keeps link and network layer together.

TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) are the transport layer protocols.

2.8 About Ethernet 10/100 MBit/s

Ethernet[7] [ETHERNET] is a network link (OSI layer 2) protocol. Ethernet is by far the most common layer 2 protocol for LANs (Local Area Networks). An Ethernet Packet (figure 2.6) includes a 14 byte header (DA, SA, TYPE) and a 46 to 1500 byte DATA section[8]. In a typical implementation, hardware manages access control, synchroniza-

Figure 2.6: Ethernet Packet Format

PREAMBLE	SYNCH	DA	SA	TYPE	DATA	FCS
62 bits	2 bits	6 bytes	6 bytes	2 bytes	46-1500 bytes	4 bytes

tion (PREAMBLE, SYNC) and error detection (FCS).

2.8.1 Link Speed

Currently most Ethernet devices in use and available for sale are 10/100 MBit/s; 100 MBit/s but may auto negotiate to 10 MBit/s when interfacing 10 MBit/s devices. Many old devices are 10 MBit/s only.

Auto Negotiation allows Ethernet devices to sense the link speed capabilities of the receiving device and adapt to them. The greatest link speed both parties are capable of will be selected.

Link speed in switched full duplex networks may vary for each link. Figure 2.7 shows a small network where end hosts Alice and Bob communicate through switches Smith and Summers. All are 100 MBit/s capable except Summers, thus the Smith⇔Summers and the Summers⇔Bob link speed is negotiated to 10 MBit/s. The Alice⇔Smith link is negotiated to 100 MBit/s.

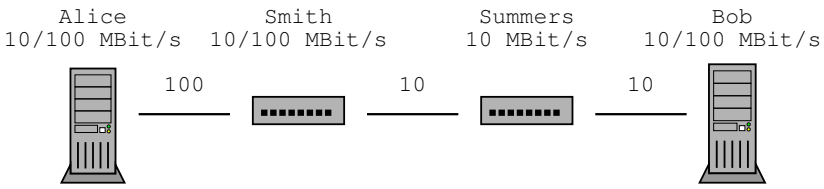
2.8.2 Duplex Mode

An Ethernet device is either operating in Full Duplex or Half Duplex mode. Most Ethernet devices can be configured to Auto Negotiate duplex or to use a statically selected duplex mode. If possible, hosts will negotiate to Full Duplex.

[7] Any reference to "Ethernet" in this paper refers to the 10/100 MBit/s Ethernet standards. Gigabit Ethernet, 10 Gigabit Ethernet, Experimental Ethernet and other none 10/100 MBit/s versions of Ethernet are not in the scope of this thesis.

[8] Short packets must be padded with unused "padding" bytes. The value of the padding is arbitrary, often the value 0x00 is used. More humorous paddings such as 0xBADDCAFE and 0xDEADBEEF are also in use.

Figure 2.7: Link speeds in a small Ethernet network



2.8.2.1 Half Duplex

In Half Duplex, an arbitrary number of Ethernet devices share a single Ethernet medium, using the CSMA/CD access protocol. Half Duplex does not offer any concurrency; only one device may transmit data at a time. Half Duplex is an old and inefficient Ethernet mode, although still in use in a decreasing number of legacy networks.

CSMA/CD is an abbreviation for the Carrier Sense Multiple Access with Collision Detect access protocol. In CSMA/CD no host may initiate a packet while carrier (transmission) is detected. If two devices should initiate transmission at the same time, a collision has occurred. All hosts should detect collisions and randomly set a "back off" timer and remain silent until it has timed out. If repeated collisions occur, greater "back off" values should be randomly selected.

2.8.2.2 Full Duplex

In Full Duplex mode each Ethernet device has a dedicated send medium and a dedicated receive medium. Interconnections are handled by network switches. A special Ethernet PAUSE frame is used to notify a sender that an Ethernet device is congested and may not receive more packets for a period of time. Depending on network design and switch capacities, Full Duplex may allow much more throughput than Half Duplex. This because several parallel mediums may be utilized concurrently instead of a single shared medium.

2.8.3 Ethernet destinations and promiscuous mode

Hardware may or may not verify destination address (DA). Packets can be divided in three classes based upon destination address:

- Broadcast packets; sent to all hosts
- Multicast packets; sent to a group of hosts
- Unicast packets; sent to a specific host

An end host may consider a packet to be a stray if it is:

- a multicast packet sent to a group which the end host does not participate in
- a unicast packet sent to a host other than the receiver

Stray packets are common in Ethernet. Switches in full duplex networks will broadcast multicast/unicast packets, if it does not know where the packet should be sent[9]. Therefor switches will generate stray packets in full duplex Ethernet networks. In half duplex, everyone receives everything. Each packet sent will reach the receiver host, but all other hosts will receive a stray copy.

Many end host Ethernet devices drop stray packets instead of passing them to software. To do the opposite, not drop stray packets, is commonly referred to as "promiscuous mode". Several end host Ethernet peripherals can enable/disable promiscuous mode by software. Interconnected devices, such as network switches, can be considered to always operate in promiscuous mode.

2.8.4 Physical Layer Device (PHY)

Several different mediums can be used for Ethernet, including coaxial cables, optical fibers and twisted pair cables. To simplify development, a PHY[10] handle the OSI Layer 2 (Link) $\Leftrightarrow$ OSI Layer 1 (Physical) interface.

A PHY is a transceiver which might be able to Auto Negotiate between different Ethernet standards. There are a number of different PHY standards, which defines how the PHY pins should be used by the Ethernet device. Since PHYs conform to standards, an Ethernet device does not need to be aware of what kind of physical medium is used. An Ethernet device may be used with all PHYs which conform to the same standard, making the device independent of PHY vendor.

There are three common PHYs: AUI, MII and RMII. Each of them have their own characteristics (as shown in table 2.1).

AUI is an old 10 MBit/s only interface.

MII was defined as the 10/100 MBit/s Ethernet was introduced. An MII design goal appears to have been to provide a high level of parallelism (data width), offering 100 MBit/s at a 25 MHz interface clock.

Reduced MII (RMII) was invented to reduce the interface pin count[11]. A high pin count raises production cost and power dissipation. Reducing pin count is a major design concern when designing switches and other multi-interface network devices. RMII provides 100 MBit/s at a 50 MHz interface clock.

Table 2.1: Common MAU and PHY protocols

MAU / PHY protocol	Pins/Interface	Width	MBit/s
Attachment Unit Interface (AUI)	6 pins	1 bit	10
Medium Independent Interface (MII)	16 pins	4 bit	10/100
Reduced Medium Independent Interface (RMII)	8 pins	2 bit	10/100

[9] This can be compared to a postman who has received mail for which the recipient's address is unknown. Instead of discarding the mail, the postman gives a copy of the mail to everyone he can find. The postman does so in hope that someone else will be able to deliver the mail correctly.

[10] Physical Layer Device (PHY) is used in Ethernet 10/100 MBit/s references. Medium Attachment Unit (MAU) used in Ethernet 10-only MBit/s references. In this paper, the term PHY is loosely used in reference to both MAU and PHY.

[11] The number of chip pins utilized.

Chapter 3

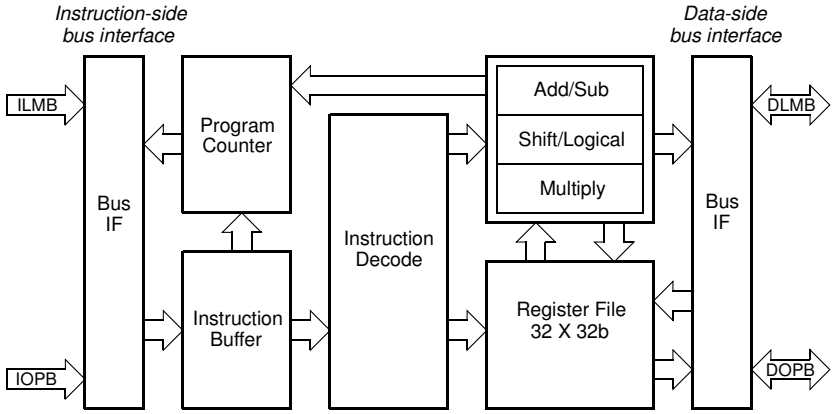
Xilinx MicroBlaze

3.1 The Xilinx MicroBlaze core

Xilinx MicroBlaze is a small processor core geared for embedded applications implemented in Xilinx FPGAs. A MicroBlaze platform consists of one or several MicroBlaze cores and a set of peripherals and an interconnecting OPB bus architecture.

The MicroBlaze core is a high performance, among the fastest available [XMB5] to Xilinx FPGAs, 32-bit RISC soft core processor. The core consists of a three stage pipeline with dedicated instruction and data paths (Harvard-style). The core is illustrated in figure 3.1.

Figure 3.1: MicroBlaze Core Block Diagram



3.1.1 Xilinx MicroBlaze bus interfaces

The MicroBlaze core contains two OPB master interfaces, IOPB which provides an instruction path and DOPB which provides a data path. The core also contains two

Local Memory Buses (LMB) interfaces, ILMB which provides an instruction path and DLMB which provides a data path.

A MicroBlaze platform must include a data path and an instruction path connected to the MicroBlaze core, therefor it will utilize 2, 3 or 4 bus interfaces.

3.1.1.1 Local Memory Bus (LMB)

The LMB bus is a highly optimized architecture, exploiting the Xilinx Dual Port Block RAM support for two concurrent single-cycle memory access. This enables single-cycle concurrent ILMB and DLMB access, making a MicroBlaze platform with only LMB access extremely efficient.

3.1.1.2 On-Chip Peripheral Bus (OPB)

Xilinx MicroBlaze is compatible with several other OPB devices, as stated in section 2.6.2. MicroBlaze's use of a dedicated data path (DOPB) and a dedicated instruction path (IOPB) allows several different bus configurations. The MicroBlaze platform may be configured to utilize the MicroBlaze architecture in a manner optimized for the application.

Examples of customized bus configurations are:

- Dual Port RAM configuration (figure 3.2). IOPB and DOPB both interface a single Dual Port RAM. This provides a shared, high speed instruction / data memory space. IOPB and DOPB may act concurrently and independently. This is similar to the LMB architecture which also utilize a Dual Port RAM. The downside is that Dual Port RAM tend to be more expensive than single port RAM and utilize more I/O pins.

- DOPB to IOPB Bridge configuration (figure 3.3). IOPB and DOPB are bridged, and the bridge will act as a DOPB slave and an IOPB master.The bridge allows IOPB and DOPB to act concurrently and independently except when DOPB access IOPB memory. This is less expensive than a Dual Port RAM while almost as fast in applications which does not read/write to memory often.

- IOPB ROM configuration (figure 3.4). This configuration utilizes that many application never need to write to instruction memory, and thus may store it in a Read-Only Memory (ROM). IOPB and DOPB may act concurrently independently.

- IOPB and DOPB interconnect configuration (figure 3.5). This is a low speed configuration. DOPB and IOPB are connected to the same bus, and they may not operate concurrently. It is a simple, slow and cheap solution.

3.1.1.3 Memory considerations

Many FPGAs contain only a few kilobytes of Block RAM. An expensive FPGA (such as a Virtex-II Pro XC2VP125) may contain more than a megabyte of Block RAM.

Applications with large buffers requirements such as network switches or routers are hard to fit even in expensive FPGA.

LMB-only memory solutions are therefor suited only for a limited range of applications. Many applications require external memory, accessed through the OPB bus.

Figure 3.2: MicroBlaze OPB Configuration - Dual Port RAM

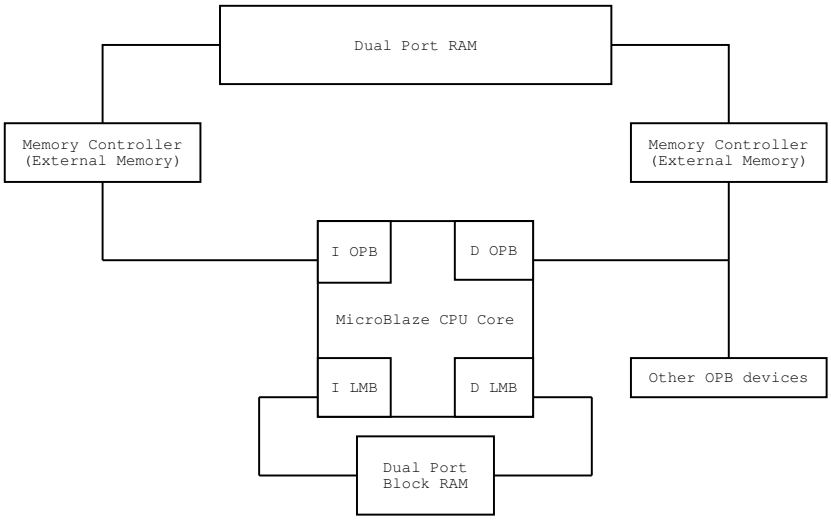


Figure 3.3: MicroBlaze OPB Configuration - OPB Bridge

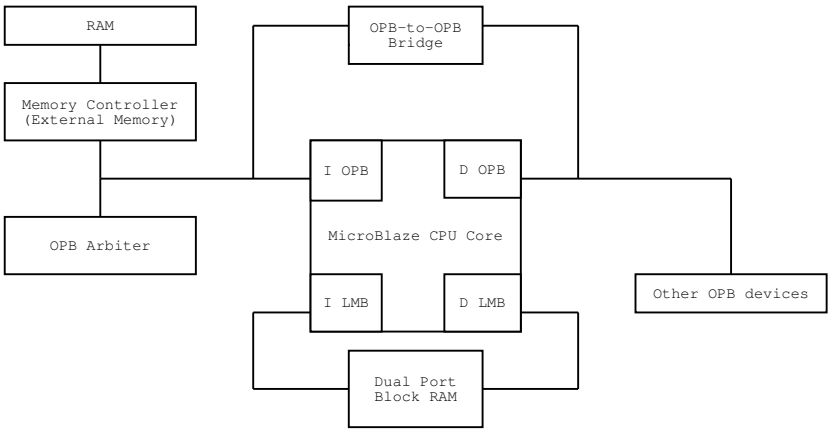


Figure 3.4: MicroBlaze OPB Configuration - ROM

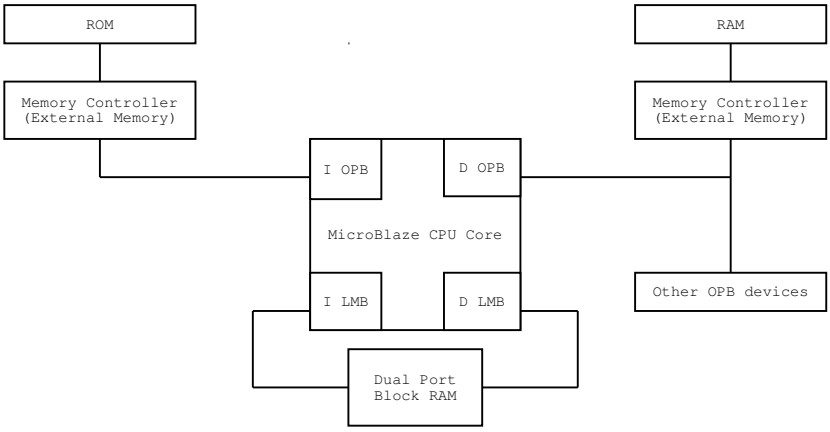
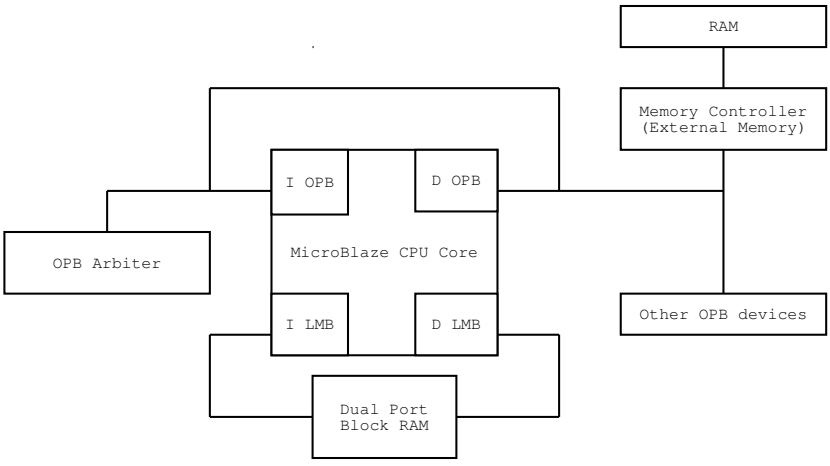


Figure 3.5: MicroBlaze OPB Configuration - IOPB & DOPB interconnected



3.2 Embedded Development Kit (EDK)

Xilinx main development kit for embedded applications and SoCs is the Embedded Development Kit (EDK). It contains a large set of OPB peripherals, Xilinx MicroBlaze soft processor, Xilinx PicoBlaze soft processor and a powerful up to date development kit.

3.3 MicroBlaze Development Kit (MDK)

MDK is a stripped down version of EDK containing less peripherals. The software in MDK is based upon an old version of EDK.

3.3.1 MicroBlaze peripherals included in MDK

MDK includes ten standard OPB peripherals, such as Memory Controller, UART, Watchdog, JTAG_UART, and Interrupt controller. One peripheral is an OPB Arbitrator which allows up to 16 masters to share the OPB Bus. Xilinx also provides documentation and tutorials on the subject of designing custom OPB Slaves for MicroBlaze platforms.

3.3.2 MDK platform tailoring utilities

A MicroBlaze platform is specified in a MicroBlaze Hardware Specification (MHS) configuration file. Typical MHS options include which bus configuration to use and which peripherals to interface.

The Platform tailoring utility is named Platform Generator (usually referred to by the acronym "platgen"). Platgen builds MicroBlaze platforms through an automated process. The process includes creating top modules and launching project synthesis. Platgen input is the MHS file and a small set of command line parameters.

MDK platgen is moderately simple and able to cope with most beginners' design issues.

3.3.3 MDK software development tools

MDK includes most common development tools such as an assembler, a compiler, a linker, a debugger, a makefile interpreter and some other utilities. All these tools are based on famous and well verified GNU tools. This is a major benefit as many developers have previous experience with these or similar tools[1].

The debugger is interfaced by JTAG[2] or RS-232 serial line interface. Any debugger supporting either the GNU Debugger Remote Protocol or the Unified Debugging Interface standard may be used to debug MicroBlaze platforms. This allows a broad range of debuggers and debugger GUIs to be used with a MicroBlaze platform.

Library Generator ("libgen") is a small script which prepares a MicroBlaze software system. Libgen utilizes a small set of standard libraries (libc, libm etc) and source code from peripherals' driver directories. Libgen reads a MicroBlaze Software Specification (MSS) configuration file and the MicroBlaze Hardware Specification (MHS)

[1] Several GNU tools, such as the C compiler, behaves similar to common UNIX tools.
[2] JTAG is a chip test interface. Often is used to verify successful ASIC manufacturing.

file. MSS specifies e.g. drivers to include. When executed, libgen compiles all drivers configured to be used and adds the compiled drivers to the libc library. C header files from the drivers are copied to the project's include path. Finally libgen creates *mbio.h*, a C header file which defines each base address of any peripheral included in the MicroBlaze platform.

Final compilation and linking of the project's main source code performed using the associated C header files and the modified libc library.

3.3.4 Problems with MDK

There are some problems with MDK which have been encountered. These problems include lack of in depth documentation for certain issues and limitations in the MDK platform tailoring utility.

3.3.4.1 MDK Documentation

The documentation [XMB1, XMB2, XMB3, XMB4] included in MDK is sufficient for beginner's applications. In some issues, in depth information is hard to find. When searching at Xilinx (or common Internet search engines such as Google or Altavista) information regarding EDK is usually found rather than MDK.

3.3.4.2 MDK platform tailoring limitations - OPB bus configuration

MDK can be configured to use no OPB bus, an instruction bus (IOPB) only, a data bus (DOPB) only or both OPB buses. MicroBlaze may utilize IOPB and DOPB in several different ways, allowing application specific customization as described in section 3.1.1.2.

However, MDK only allow the designer to specify which OPB/LMB interfaces to implement. MDK does not offer any configuration options for controlling if - or how - buses should be interconnected.

If both DOPB and IOPB are used, they will be connected to the same bus (shown in figure 3.5). When executing software from the shared bus it will be heavily utilized, bus delays will increase and data throughput will decrease. A shared bus has a significant negative performance impact.

Additionally, MDK does not include any OPB-to-OPB bridge.

EDK offers a number of measures to remedy these problems through a set of MHS directives which are not available in MDK. To customize OPB in MDK, a designer must venture beyond platgen and modify automatically generated VHDL files. This means that when designing advanced MicroBlaze platform, a designer loose a lot of time and effort saving automation.

This was discussed in [XEPF1] where Xilinx staff replied that it would be fixed before the end of summer 2002. Apparently Xilinx did not to update MDK. Possibly Xilinx put their efforts into EDK instead.

3.3.4.3 MDK platform tailoring limitations - buffers and pads configuration

Buffers. A Virtex FPGA input/output pin is connected to a "pad". Each Virtex FPGA contains four global clock pads. These pads are specifically used for clock signals. Each pad is in turn connected to a buffer. The buffer-pad combination decides how a

pin may be interfaced by the design. There are a number of different buffers which each has its own purpose and characteristics.

MDK platgen buffer options. MDK platgen can be configured to insert buffers or not insert buffers.

When inserting buffers, MDK platgen will auto-insert buffers as stated in table 3.1. These are the common buffers usually required by a design.

When configured to not insert buffers, MDK platgen will generate a netlist without pads (often referred to as black box or soft core). The netlist may be utilized as a component in a larger design. The netlist can also be used in a small wrapper top entity which defines which buffer should be used for each I/O signal.

The buffer problem. There are cases when the design require specific buffers. An example of this is when a designer wish to implement a synchronous Ethernet peripheral on a prototype board with RX_CLK, TX_CLK or REF_CLK connected to a global clock pad. An input which is not used as a clock will be buffered with an Input Buffers (IBUF). But an IBUF cannot be connected to global clock pad. In this case a Dedicated Input Buffer (IBUFG) must be used.

Solutions. This problem can be solved by not inserting buffers and utilizing a small wrapper as the top entity. Scripted modifications of generated top entities can also be used.

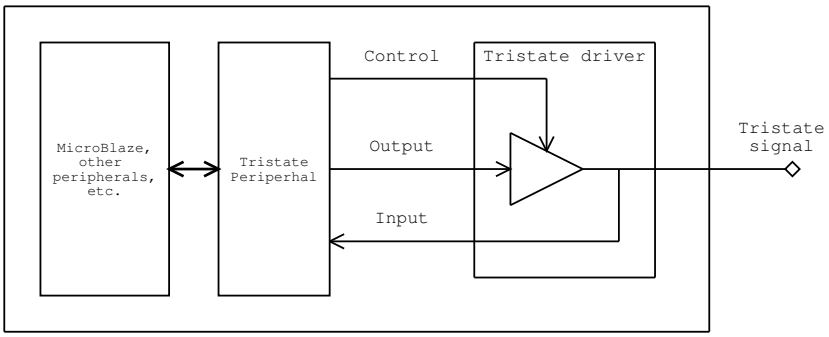
Table 3.1: MDK synthesis auto-inserted buffers

Buffer	Inserted at I/O signals used as
BUFGP	Clock signals
IBUF	General input
OBUF	General output
IOBUF	Tristate input/output

3.3.4.4 MDK platform tailoring limitations - Tristate design style

Another issue is that tristate buffers in peripherals must be written as a set of three I/O signals (input, output and tristate control) as shown in figure 3.6. Although this is not a major drawback when designing custom components, it is an unnecessary design restriction. Specifically it may limit which third party OPB components (distributed as netlists or source code) which are easy to utilize in a MicroBlaze peripheral design.

Figure 3.6: MicroBlaze platform with tristate peripheral



Chapter 4

Xilinx MicroBlaze as a Network SoC

4.1 MicroBlaze as a SoC system

The MicroBlaze Development Kit is geared at developing System on Chip solutions. All standard peripherals are implemented on the same chip (FPGA) as the MicroBlaze core. It also contains typical I/O peripherals for microcontrollers and SoCs, such as RS-232 serial line interface, memory controller (intended for SRAM and FlashRAM), General Purpose I/O and others.

4.2 MicroBlaze Network Support

MicroBlaze Development Kit does not include any network peripherals. This is obviously a drawback when designing network SoCs. However there are two MicroBlaze compatible OPB Ethernet peripherals available for sale at Xilinx. One of these is a small and simple Ethernet peripheral. The other is a professional Ethernet peripheral which is highly configurable and support most Ethernet modes. In a commercial SoC project, buying either of these peripherals is simple way to get a well verified peripheral.

4.2.1 Customized Ethernet peripheral

Custom design of an Ethernet peripheral may be preferred in SoC design, especially in application specific SoCs. A custom made peripheral may yield a number of interesting properties:

- Cheap
- Area, performance and features may be customized for a specific application

The level of application specific customization is of interest in a large number of projects. A few examples of interesting customizations are:

- Speed or area optimized design

- DMA or register access
- Promiscuous mode or not
- Hardware support for Ethernet multicast
- Classification, validation, or special handling of certain Ethernet frames

Classification, validation, and special handling of certain Ethernet frames is performed by most Ethernet peripherals, as it is a requirement for IEEE Ethernet standard compliance[1]. A typical Ethernet peripheral will verify Ethernet frames and take special actions upon receipt of Ethernet PAUSE frames. It would not be difficult to extend such a classification module to also handle layer 3 frames (such as IPv4) to reduce MicroBlaze CPU utilization. By decreasing CPU utilization the software application can be simplified or power dissipation may be reduced.

[1] It may also be performed in software.

Chapter 5

Development of MicroBlaze Ethernet peripherals

5.1 Design requirements and limitations

5.1.1 Full Duplex only

In principal, supporting the entire range of Full/Half/Auto Duplex combinations is easy, but it takes time and effort to develop and verify. For simplicity, all Ethernet peripherals were developed for Full Duplex only, which is by far the most common Ethernet Duplex mode today.

5.1.2 Ethernet Interfaces (PHY): MII and RMII

The Ethernet peripherals were designed to be used on two different prototype boards, one based on an MII Ethernet Interface and one based on a Reduced MII (RMII) Ethernet Interface. The designs therefor needed to support both interfaces.

5.2 Implementation

5.2.1 Eth Version 1.00, Revision A

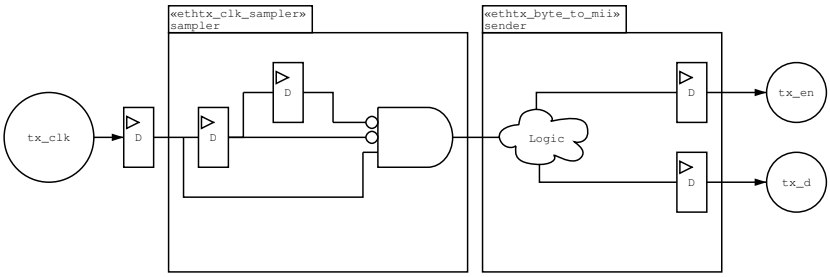
Eth Version 1.00 Revision A is designed to be a very simple MAC, portable to most FPGA and prototype boards.

5.2.1.1 Fully synchronous, single edge triggered

To simplify design and verification, no asynchronous logic is used in the design. Additionally, only flip flops triggered by system clock rising edge are used. Figure 5.1 illustrates how an input clock rising edge (a “001” pattern) is detected and output is triggered by rising edge.

It should be noted that due to this design choice, Ethernet Interface clocks must be sampled fast enough to detect rising edge and change values prior to falling edge.

Figure 5.1: Eth Version 1.00 Revision A, Input to Output illustration



The system clock must therefor be approximately 5 times faster than Ethernet interface clocks. Table 5.1 shows the interface clocks and resulting minimal system clock frequency.

Table 5.1: Minimum System Clock for reliable synchronous Ethernet sampling

Interface	Interface Clock	Minimum System Clock
MII 10 MBit	2.5 MHz	12.5 MHz
MII 100 MBit	25 MHz	125 MHz
RMII 10 MBit	5 MHz	25 MHz
RMII 100 MBit	50 MHz	250 MHz

Why synchronous and single edge triggered flipflops? One reason is that asynchronous logic and systems triggered both edges are harder to verify. Asynchronous operations may introduce hard to find race conditions and errors which rarely occur in simulations.

Another reason is that asynchronous logic in Xilinx Virtex FPGA implementations require a global clock net dedicated each interface clock (MII uses two clocks, RMII uses one). Xilinx Virtex only provide four dedicated clock pads [VIRTEX]. Synchronous solutions therefor enables multi-interface designs in Xilinx Virtex FPGA.

5.2.1.2 System Clock and Performance Impact

A MicroBlaze platform consisting of Eth Version 1.00 Revision A, the OPB bus and the MicroBlaze core was created. The platform was synthesized and implemented for a Virtex 800 speedgrade 4 FPGA.

Maximum system frequency for the platform proved to be 50 - 60 MHz[1]. As shown in table 5.1, this only enables 10 MBit/s implementations.

Eth Version 1.00 Revision A bus is operating at 50 MHz, 32-bit data-width and three cycle access time. This yields a maximum switching capacity of

$$\frac{32\,\text{bit} * 50\,\text{Mcycles/s}}{3\,\text{cycles}} = 533\,\text{MBit/s}.$$

[1] Utilizing a Place And Route overall effort of 5.

This should be enough for a very large number of 10 MBit/s devices.

5.2.1.3 Modular design

To provide a simple, expendable and reusable interface, each component is designed with modularity and re-usability in mind.

All send-logic in a transmit (TX) core (figure A.1) and all receive-logic is placed in a receive (RX) core (figure A.2). The cores are mapped into a small top-module which handles bus interconnections. As shown in table 5.2, all parts of the design have been kept at a small number of short and easily read VHDL files.

Table 5.2: Eth Version 1.00 Revision A VHDL files

Modules	VHDL Files/Entities	Average number of lines per file
General	4	93
RX Core	7	112
TX Core	6	133
OPB Slave	1	250

5.2.2 Eth Version 1.00 Revision C

This is a redesign of Eth V.100 Revision A. It is designed to reach 100 MBit at target technology MII, Virtex 800 speedgrade -4 with system clock set to 50 MHz.

5.2.2.1 Asynchronous

This peripheral utilizes three clocks: the OPB system clock, the MII RX_CLK and the MII TX_CLK. The MII clocks are significantly slower (25 MHz at 100 MBit/s) than the system clock. This is utilized to reduce most of the peripheral's clock frequency. Only the register files and the top module is clocked by system clock, most of the design is clocked by MII clocks.

Clock pad utilization. This peripheral utilizes three out of four Xilinx Virtex [VIRTEX] clock pads. The clock pad utilization restrict the design to platforms with a single Ethernet peripheral.

Reduced power dissipation. Thanks to its asynchronous design, a large part of the design may operate at low clock frequencies. These low, system clock independent, clock frequencies allows reduced power dissipation.

Open Loop and minimum system clock frequency. The design utilizes open loop [XTE1] solutions to move from Ethernet clock domains to system clock domain. The open loop requires system clock to always be faster than interface clocks. Utilizing the variables in table 5.3 the timing requirements may be expressed as:

$$t_{sysclk} + \left|d_{sysclk}\right| \leq t_{phyclk} - \left|d_{phyclk}\right|$$

This formula may be rewritten as:

$$f_{sysclk} \geq \frac{1}{t_{phyclk} - |d_{phyclk}| - |d_{sysclk}|}$$

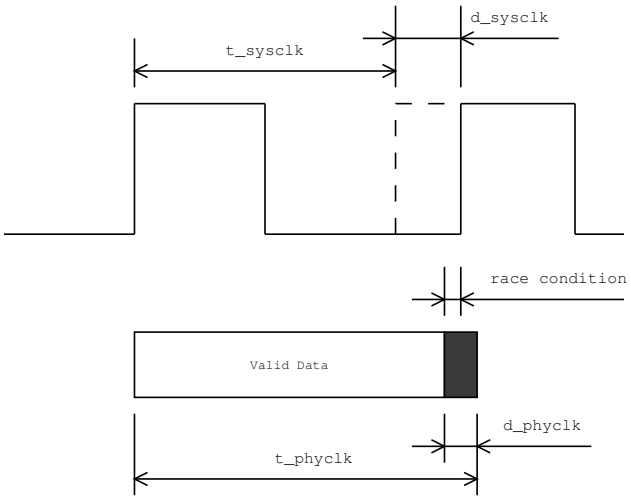
Table 5.3: Open Loop - Minimum system clock frequency

Variable	Denotes
f_{sysclk}	System clock frequency (Hz)
t_{phyclk}	Ethernet PHY clock period (s)
t_{sysclk}	System clock period (s)
d_{phyclk}	Ethernet PHY clock drift, worst case (s)
$d_{sysdrift}$	System clock drift, worst case (s)

It is reasonable to assume $|d_{sysclk}|$ to be small because a system clock oscillator is usually very exact. $|d_{phyclk}|$may however be large; Ethernet receiver clocks must synchronize against Ethernet transmitter clocks which could possibly be skewed. During synchronization, Ethernet clocks will drift. Assuming clock drift worst case of $\pm 19\%$ [2] yields that system clock must be 23% faster than interface clock.

Figure 5.2 illustrates the race condition which eventually will occur if the timing requirements requirement is not met; the system clock domain fails to sample valid data.

Figure 5.2: Open Loop Race Condition



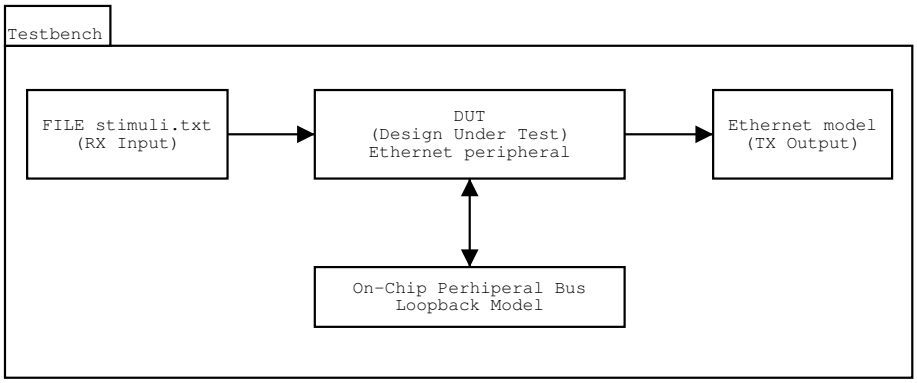
[2] A simulation testbench for verification of academic Ethernet peripherals (supplied by Robert Wikander of Switchcore AB) have a $\frac{|d_{phyclk}|}{t_{phyclk}} < 19\%$ worst case clock drift. The testbench simulate interface clock drift but assume system clock drift to be zero.

5.3 Verification

5.3.1 Testbench

The testbench was designed to be modular and to be able to simulated different input. This was archived by abstracting input/output verification into two submodules, and by reading Ethernet input from file. The testbench design is shown in . The testbench

Figure 5.3: Modular Testbench



were also designed to allow commonly performed quick tests (regression test).

5.3.1.1 Regression tests

Testbenches were automated and generate logfiles containing assertion notes, warnings and errors. Human inspection of waveforms were therefor rarely needed to verify that recently added code does not break previous functionality. Regression tests proved to be a highly useful tool and shortened verification cycles considerably by automatically finding errors in designs.

5.3.1.2 Modular testbench: Ethernet Model

To simplify the main testbench, verification of Ethernet correctness is performed in a stand alone model. The model identifies short packets, long packets, short interframe gaps, incorrect FCS and a number of other Ethernet violations. Ethernet violations will cause warning assertions. Correct packets will cause assertion notes, representing the packet in hexadecimal notation.

5.3.1.3 Modular testbench: OPB Model / Loopback

This model models the OPB bus. The model serves two main purposes: OPB Verification and Loopback.

OPB Verification

Verification of OPB behavior correctness is performed in the OPB Loopback Model. The model is able to identify several possible violations of the OPB bus. This part of

the model is not directly Eth Version 1.00 specific. Improvements must however be made before it is reusable as a general purpose Xilinx OPB verification model.

Loopback

The model also acts as a loopback device: Any packet received by the model, will also be transmitted by the model.

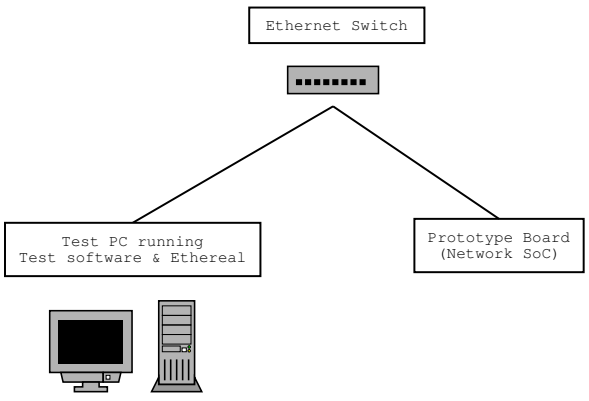
5.3.2 Multiple Compilation and Synthesis

Ethernet devices were compiled and simulated with two common simulation environments, Modeltech Modelsim and Cadence Logic Verification. Ethernet devices were compiled and synthesized with Xilinx Synthesis Technology (XST) and Synplicity Synplify Pro. By testing code in a large number of different tools, non-portable VHDL constructs have been identified and removed, as well as most code constructs which cause warnings in any of the tools used.

5.3.3 Packet monitoring with The Ethereal Network Analyzer

The Ethereal Network Analyzer ("Ethereal") was used to capture Ethernet packets, efficiently monitoring network conversations between the Test PC and the Ethernet peripheral. Ethereal tests were usually performed in a simple network consisting of a Test PC, a network switch and a prototype board (with an FPGA configured with a MicroBlaze platform). This setup is shown in figure 5.4.

Figure 5.4: Network Test Setup



Ethereal efficiently detected several types of bad layer 3 packets. This proved useful for tracking software errors.

The switch and the Ethernet peripheral on the Test PC drop bad Ethernet packets, without showing them to Ethereal. Peripheral errors usually cause bad Ethernet packets. Because bad Ethernet packets are dropped, Ethereal was not very useful for detecting errors in the peripheral.

5.3.4 Real World Tests

To verify hardware robustness, the Ethernet devices were used in a number of real world tests. Some of them, such as a simple ICMP Echo Request / ICMP Echo Reply test[3], were performed for several hours with no packet loss. All real world tests indicates that the final designs are robust.

[3]Commonly referred to as "ICMP ping" and performed by the command line tool "ping" which is included in most operating systems.

Chapter 6

Comparing Ethernet peripherals

Comparing peripherals is hard. Feature requirements, area requirements and system clock requirements may be different for each SoC project. A SoC is also likely to have high quality requirements, but quality is hard to measure.

Given all these factors, it is not feasible to name a single peripheral which is optimal for all applications. However it is possible to compare peripherals and determine which is best suited for certain applications.

A number of Ethernet peripherals have been compared: EMAC Lite - Xilinx OPB Ethernet Lite Media Access Controller, EMAC - Xilinx OPB Ethernet Media Access Controller, EthMac - Opencores.org 10/100 Ethernet MAC, Eth Version 1.00 Revision A and Eth Version 1.00 Revision C.

EthMac and WISHBONE. WISHBONE is a simple bus standard maintained by opencores.org. WISHBONE and OPB is similar, both are intended to be an easy to use System on Chip bus. EthMac is an WISHBONE peripheral. MicroBlaze cannot use it because MicroBlaze does not support the WISHBONE bus. EthMac has been included because it is a well verified MAC, has been used in real world applications, and is open source. EthMac could probably be converted into an OPB peripheral with some effort.

6.1 Feature comparison

The following set of features has been compared in table 6.1:

- Asynchronous - does the peripheral make use of MII/RMII clock signals to clock flipflops/registers? This is often used to reach 100 MBit/s while maintaining low system clock requirements. But it has a major drawback: In several FPGAs, you only have a few clock input nets. Utilizing MII/RMII clocks in a peripheral may limit the application to a single Ethernet peripheral.
- System bus. OPB is preferred since it is the only native peripheral bus in MicroBlaze.
- DMA. Is the peripheral able to do direct memory access? This is important to make high performance applications.

- MII Management. Is the peripheral able to read/write MII Management registers?
- PAUSE frame support. This is an important Ethernet standard which prevents network congestion. If not supported in hardware, it should be supported in the software/drivers.
- RMII support. RMII is a pin-reduced MII-similar interface, preferred in multi-interface/peripherals designs such as switches.

The only RMII peripherals are Eth Version 1.00 Revision A and C. This is a major advantage in multi-interface designs. EMAC, EMAC Lite and Eth Version 1.00 Revision A are synchronous designs. This gives them further advantage in multi-interface designs since they do not utilize additional clock nets (which are scarce in some FPGA such as Xilinx Virtex).

The peripherals can be divided into a simple peripherals and advanced peripherals. EMAC and EthMac support a large set of features (DMA, MII Management, PAUSE frames) while the others does not. Therefor peripherals such as EMAC and EthMac are preferable if an application require a full scale DMA-capable interface. Peripherals such as EMAC Lite and Eth Version 1.00 Revision A and C are preferable if only a simple peripheral is required.

Table 6.1: Ethernet peripherals feature comparison

Device	Async	Bus	DMA	Mgt	Pause	RMII
EMAC Lite	No	OPB	No	No	No	No
EMAC	No	OPB	Yes	Yes	Yes	No
EthMac	Yes	WB	Yes	Yes	Yes	No
Eth v1.00 A	No	OPB	No	No	No	Yes
Eth v1.00 C	Yes	OPB	No	No	No	Yes

Header/Note	Description / Explanation
Async	Asynchronous; utilize MII/RMII clocks to trigger flipflops
Mgt	MII Management interface support
Pause	Ethernet full duplex PAUSE frame support
RMII	Reduced MII interface support

6.2 Speed comparison

Speed comparison was based on estimated performance for Xilinx Virtex-E FPGA (chip xsv2000e-8-FG1156) implementations. One reason for using this particular FPGA is that the Platinum prototype board which was supposed to be available for the evaluation was Virtex-E based. Another reason is that speed/area statistics for this particular FPGA is included in EMAC / EMAC Lite documentation.

f_{max} is the estimated maximum system clock frequency. For configurable designs (such as EMAC / EMAC Lite), the greatest value for f_{max} is listed. For Eth v1.00 Revision A and C, MII configuration is used.

f_{max} was estimated by synthesizing the design with XST (optimization level 2 and optimization mode "speed"). Then the netlist is passed through the Xilinx Place & Route flow (overall effort set to 5, extra effort set to 2) utilizing a set of Ethernet timing constraints (table 6.2) used by Xilinx in EMAC documentation. The system clock period constraint was tweaked until a minimum period was found for which all constraints were met.

Then f_{max} was calculated as $f_{max} = \frac{1000}{system\,clock\,period\,(nanoseconds)}$ MHz.

Table 6.3 illustrates a 133 MHz system clock constraint.

For EMAC and EMAC Lite, the corresponding values are extracted from the specification.

The results are shown in table 6.4.

Table 6.2: Ethernet timing constraints

```
NET "rx_clk" TNM_NET = "RXCLK_GRP";
NET "tx_clk" TNM_NET = "TXCLK_GRP";
TIMESPEC "TSTXOUT" = FROM "TXCLK_GRP" TO "PADS" 10 ns;
TIMESPEC "TSRXIN" = FROM "PADS" TO "RXCLK_GRP" 6 ns;
NET "rx_clk" USELOWSKEWLINES;
NET "tx_clk" USELOWSKEWLINES;
NET "tx_clk" MAXSKEW= 2.0 ns;
NET "rx_clk" MAXSKEW= 2.0 ns;
NET "rx_clk" PERIOD = 40 ns HIGH 14 ns;
NET "tx_clk" PERIOD = 40 ns HIGH 14 ns;
NET "rx_d<3>" NODELAY;
NET "rx_d<2>" NODELAY;
NET "rx_d<1>" NODELAY;
NET "rx_d<0>" NODELAY;
NET "rx_dv" NODELAY;
```

Table 6.3: System clock and reset constraints

```
NET "opb_clk" TNM_NET = "opb_clk";
TIMESPEC "TS_opb_clk" = PERIOD "opb_clk" 7.5 ns HIGH 50 %;
NET "opb_rst" TIG;
```

6.3 Resource comparison

Resource comparison was performed for Xilinx Virtex-E FPGA (chip xsv2000e-8-FG1156) implementations. Virtex-E was selected for the same reasons as stated in section 6.2 (Speed Comparison).

Virtex-E has three major resources which is used by the Ethernet peripherals:

Slices are the main Virtex-E resource. These are blocks of programmable logic. Most things are implemented in slices. Table 6.5 shows that all simple MACs consume few

Table 6.4: Ethernet peripherals performance comparison

Device	f_{min} MII 10 MBit	f_{min} MII 100 MBit	f_{min} RMII 10 MBit	f_{min} RMII 100 MBit	f_{max}
EMAC Lite	5 MHz	50 MHz	N/A	N/A	80.8 MHz
EMAC	5 MHz	50 MHz	N/A	N/A	88.0 MHz
EthMac	?	?	N/A	N/A	76.9 MHz
Eth v1.00 A	12.5 MHz	125 MHz	25 MHz	N/A	137 MHz
Eth v1.00 C	3.1 MHz	31 MHz	6.2 MHz	62 MHz	137 MHz

Header/Note	Description / Explanation
f_{min}	Minimum system clock frequency
f_{max}	Maximum system clock frequency
?	f_{min} has not been researched by EthMac developers

slices (350 - 470) while advanced MACs consume a few thousand slices.

BRAM (Block Select RAM) are memory blocks in which data can be stored efficiently. Table 6.5 shows that EthMac utilize less BRAM resources than other MACs in test.

GCLK (Global Clock Nets) are routing nets which provide clock signals to logic (e.g. slices and BRAM). This is a scarce resource: Virtex-E have only four GCLKs. Table 6.5 shows that all synchronous MACs in the test utilize one GCLK (system clock). All asynchronous MACs used in test utilize three GCLKs (System, transmit and receive clocks).

Table 6.5: Ethernet peripherals resource comparison

Device	Slices	BRAM	GCLK
EMAC Lite	350 - 405	8 (4KB)	1
EMAC	1528 - 2570	8 - 16 (4 - 8KB)	1
EthMac	2439	2 (1KB)	3
Eth v1.00 A	370	8 (4KB)	1
Eth v1.00 C	476	8 (4KB)	3

6.4 Quality comparison

EMAC Lite and EMAC is considered production quality by the Xilinx corporation. EthMac is considered production quality by the opencores.org developers. These three MACs have been tested by developers as well as number of users/customers. Therefor, they are by far the most well verified peripherals.

Eth Version 1.00 Revision A and C are not as well verified.

The Eth Version 1.00 revisions has been reused by a small number of students. Eth has also passed a number of verification tests. The Eth 1.00 revisions are therefor considered to be of good academic quality but more tests are needed before they can be considered to be of production quality.

6.5 Details about the devices used in analysis

6.5.1 EMAC Lite - Xilinx OPB Ethernet Lite Media Access Controller

EMAC Lite is a simple Ethernet MAC provided by Xilinx. It is an OPB device and easily used in MicroBlaze applications. It is marketed as a simpler and smaller device than EMAC.

Features:

- 10/100 MBit/s, MII only.
- Minimum system clock frequency is 5 MHz (50 MHz for 100 MBit/s).
- Synchronous.
- Not runtime configurable, e.g. duplex mode is configured at build time.
- No promiscuous mode.
- Small 2*2KB packet FIFOs, one packet per FIFO only.

6.5.2 EMAC - Xilinx OPB Ethernet Media Access Controller

EMAC is an advanced and full-featured MAC provided by Xilinx. It is an OPB device and easily used in MicroBlaze applications. It has features which enables great performance (such as DMA and multiple packets per FIFO) and useful options (MII Management, auto-pad, promiscuous mode).

Features:

- 10/100 MBit/s, MII only.
- Minimum system clock frequency is 5 MHz (50 MHz for 100 MBit/s).
- Synchronous.
- MII Management module.
- Small 2*2KB or 2*4KB FIFOs, up to 16 packets per FIFO.
- Pause frame support.
- DMA or FIFO-registers for I/O.
- Auto-pad short packages.
- Promiscuous mode runtime configurable.

6.5.3 EthMac - Opencores.org 10/100 Ethernet MAC

EthMac is an advanced and full featured MAC developed in the Open Source community Opencores.org. EthMac uses a WISHBONE data bus for I/O.

Features:

- 10/100 MBit/s, MII only.
- Minimum system clock frequency unspecified[1].
- Asynchronous.
- Pause frame support.
- DMA or FIFO registers for I/O.
- Auto-pad short packages.
- Promiscuous mode runtime configurable.

[1] EthMac developers have focused on achieving great clock frequencies. Since low clock frequency have not been a design concerned, this has not been researched by EthMac developers.

Chapter 7

Network software for MicroBlaze

7.1 Small TCP/IP stacks for embedded applications

In the context of small TCP/IP stacks, it is interesting to know "how small is a small TCP/IP stack?". The answer is simple - it depends on what you are comparing with regard to. Some application specific stacks may only contain 256 bytes of code, but a general stack is hard to make that small.

Therefore, we define a number of characteristics preferred in a small TCP/IP stack for MicroBlaze:

- The stack should be general and reusable, not designed for a single application.
- The TCP/IP stack should provide two buffers: one for receiving and one for sending packets. These buffers may be as small as a single packet. These buffers enable simple interfaces for handling datagram oriented stacks (ARP, ICMP, UDP and others). These buffers may be reused by all datagram functions to save memory.
- The stack should not be multi-user oriented[1].
- The stack should be modular. If an application only need ARP+UDP, or ARP+ICMP, it should not be required to implement memory expensive protocols such as TCP.
- The stack should be available as well commented/documented source code[2].

7.1.1 Xilinx XilNet

Xilinx XilNet is a small TCP/IP stack included in MDK. It is provided to demonstrate the networking capabilities of MicroBlaze. However XilNet suffers a number of drawbacks:

[1] There are usually no user (or a single user) in SoCs and embedded applications. Providing memory protection and multi-user handling is memory expensive and serves no purpose in this context.

[2] Because developers may wish to tweak the stack for a specific application.

- No network peripheral is included in MicroBlaze, so it is not possible to use XilNet with MDK peripherals only.
- The requirements on MAC send/receive functions are unspecified. No example of such functions is provided for reference.
- No example applications utilizing XilNet are provided.
- XilNet is unsupported (as opposed to all other MDK libraries).
- XilNet documentation is of low quality and only describes the function calls. No usage notes nor general design description is provided.

7.1.1.1 Several issues with XilNet

Attempts were made to make XilNet work. The XilNet implementation used is described in Appendix B. MAC functions were easily implemented.

ARP was successfully tested but IPv4/ICMP tests failed. XilNet was able to receive ICMP Echo Request packets, but the following errors were observed on the corresponding ICMP Echo Reply packets:

- IPv4 checksum was incorrect.
- IPv4 total length was incorrect.
- ICMP checksum was incorrect.
- ICMP data was incorrect.

These errors may be due to incorrect implementation / usage of XilNet or a flaw in XilNet itself. Xilinx Embedded Processor Forum was searched for a solution, but no relevant information was found. The forum indicates that there are issues [XPEF2, XPEF3] even with recent XilNet versions included in EDK.

7.1.1.2 Modularity

XilNet has a modular design at file level (*arp.c*, *ip*.c, *tcp.c* etc). However there are several interdependencies, especially in the file *ip.c*. A number of changes to ip.c (mainly preprocessor directives such as *#ifdef* and *#endif*) were applied and XilNet was successfully stripped of TCP support to save memory.

7.1.1.3 Functionality

XilNet's TCP support is limited. It supports only a single concurrent TCP connection.

7.1.1.4 Overall

XilNet is a very small TCP/IP stack. XilNet is 8 KB large, but a small test-application (appendix B) with XilNet but without TCP and UDP support is only 5 KB large. This makes XilNet well suited for applications with very little RAM. However, several users have experienced problems using XilNet.

7.1.2 lwIP

lwIP is described by [lwIP] as:

> The focus of the lwIP TCP/IP implementation is to reduce the RAM usage while still having a full scale TCP. This makes lwIP suitable for use in embedded systems with tens of kilobytes of free RAM and room for around 40 kilobytes of code ROM.

lwIP was ported to MicroBlaze by Nilsson, Schmid and Wiklander. The MicroBlaze port is known as MB-lwIP [MB-lwIP].

7.1.2.1 Modularity

lwIP is of a modular design. Its is made to be easy to port and to add new network interface drivers.

7.1.2.2 Functionality

lwIP is a full scale TCP/IP stack.

7.1.2.3 Overall

lwIP uses 40 KB RAM, which is a lot in (low cost) SoC applications. Either the SoC contains more than 40 KB of internal RAM, or external memory must be used.

7.2 Operating Systems for MicroBlaze

A number of real-time and micro controller operating systems are ported to MicroBlaze, e.g:

- Micrium μC/OS-II
- ATI Nucleus
- μCLinux

Notably is RealFast [REALFAST] Sierra 16 HW-RTOS (hardware real-time operating systems) support for MicroBlaze. Sierra 16 and MicroBlaze may be implemented in a single Xilinx FPGA. RealFast's vision is to relieve micro controllers of time consuming software tasks by providing dedicated hardware such as Sierra 16 HW-RTOS.

Chapter 8

Results

In chapter 1 a set of objectives were identified. This chapter states how they were met.

Xilinx MicroBlaze and the IBM CoreConnect On-Chip Peripheral Bus were evaluated. They were shown to provide a powerful RISC architecture with good bus performance. The bus enables a large number of network peripherals even at low system frequencies.

The MDK standard peripherals were evaluated. They are concluded to be a small set of simple peripherals which are useful in several applications. A greater set of peripherals, e.g. network peripherals and bus bridges, would have made MDK much more complete.

The MDK software development tools proved to be easy to use - partially thanks to their similarities with standard UNIX tools. MicroBlaze software development is usually simple and straight-forward programming in the C language. The GNU Debugger (GDB) allows remote on-chip debugging which is useful.

The MDK platform tailoring tools were evaluated. They are adequate for simple platforms. Some problems became apparent in more complex designs.

Network connectivity has been provided through the Eth Version 1.00 set of peripherals which were designed in this thesis work. A number of different customizations of Eth have been demonstrated. MII and RMII are supported in all customizations and MII has been verified on prototyping boards. RMII capabilities could only be verified in simulations.

Ethernet peripherals have been analyzed. Because each peripheral has a unique set of features and drawbacks, no peripheral can be selected to be considered the best. Eth Version 1.00 performed especially well in performance benchmarks, while other peripherals were concluded to support more features and to be better verified.

Xilinx XilNet TCP/IP stack was evaluated. A number of problems were identified. It was determined that several developers in the *Xilinx Embedded Processor Forum* various experienced problems with XilNet.

lwIP was evaluated for MicroBlaze. [MB-lwIP] demonstrated a lwIP based web-server running on MicroBlaze.

Chapter 9

Discussion

Xilinx MicroBlaze is one among several new soft core processors. Several soft core processors are emerging from commercial manufacturers, academic institutions and open source communities.

This thesis has not attempted to compare MicroBlaze to the broad range of soft core processors. Xilinx market MicroBlaze as "the industry's fastest soft processing solution" [XMB5], so MicroBlaze is likely to perform well in performance evaluations.

When this thesis work set out, the overall goal was to evaluate Xilinx MicroBlaze for Network applications. A number of key objectives for reaching this goal were identified in chapter 1. These objectives have all been met.

No multi-interface prototyping board. An important setback was that the Platinum prototyping board did not become available during the work on this thesis due to development and manufacturing problems. Because Platinum was not available, RMII capabilities of the network peripheral Eth could not be verified. A large set of applications which requires several network interfaces could not be demonstrated without the Platinum prototyping board.

This setback did not have a major impact to the evaluation, but a number of simple "proof of concept" multi-interface applications, such as network switches, could not be implemented.

9.1 Future work

Embedded Development Kit (EDK) was not used in this evaluation. All experiments have been performed using the MDK environment. A re-evaluation using EDK would be useful. EDK might remedy the problems found in the MDK platform tailoring utilities.

XilNet sent bad IP and ICMP packets in this evaluation. The cause of bad packets is not known. Making XilNet send correct packets would be useful. More recent versions of XilNet included in EDK could be evaluated.

Soft core processor comparison. An overview of the soft core processors available would be interesting. Without an independent review of a large set of processors, engi-

neers will have a hard time finding out which processor is best suited for their application.

The OpenCores.org 10/100 Ethernet MAC is well verified and has been used by a large set of developers. It has been used in professional projects. This MAC is therefor of production quality and is available for free (GPL "copyleft" open source license).

The MAC uses the WISHBONE as its system bus. It would be interesting to port this MAC to Xilinx OPB. A port may be of interest to developers with limited funding or with previous experience of this MAC.

There are at least two ways to port the WISHBONE MAC:

- Rewrite the top module (and some of the components) for Xilinx OPB.
- Create a WISHBONE $\Leftrightarrow$ OPB bridge for the MAC to use.

The later approach may be much more beneficial because then the port would not break upon changes in the OpenCores MAC source code. It may also be simpler, as it require only basic understanding of WISHBONE and OPB. The "OPB Usage Notes" [XMB1] provided by Xilinx, intended to aid bridging Xilinx OPB with other OPB implementations, might also be used in the development of such a bridge.

Adding features to Eth Version 1.00 revisions. The Eth 1.00 revisions are simple Ethernet peripherals and can be extended with several features. Some new features might include:

- Half Duplex (CSMA/CD) support, non-promiscuous mode, MII Management and PAUSE frame support

Utilizing more clock edges in Eth Version 1.00. Revision A is single edge triggered and synchronous. Revision C is asynchronous and all flipflops are triggered by rising edge. Minimum Clock frequency and power dissipation may be reduced by utilizing both edges. This is most likely performed in other peripherals (e.g. Xilinx EMAC) which synchronously detect MII rising edges at half the frequency of Eth Revision A.

Independent and professional verification of Eth Version 1.00. Eth would benefit from an independent and professional verification. As of current date it has been reused, modified and verified in a few student projects. But student project tend to keep loose quality standards. Professional engineers would not consider Eth to be of production quality.

More interesting proof-of-concept applications. Implementing a network switch or other multi-interface design would be nice. With access to a multi-interface FPGA prototyping board, this should be easy.

9.2 Conclusions

Processor core and architecture. Xilinx MicroBlaze is a powerful soft core processor well suited for Network SoC design. The IBM CoreConnect On-Chip Peripheral Bus provides a simple and fast interface to MicroBlaze peripherals.

Software. A large set of software is available or can be ported to Xilinx MicroBlaze. A small number of operating systems are already available to MicroBlaze. TCP/IP stacks for embedded applications, such as lwIP, are easily ported to MicroBlaze.

Development kits for MicroBlaze. The MicroBlaze Development Kit (MDK) gives a good overall impression and is easy to use. Some issues with MDK have been found, but the Embedded Development Kit (EDK) is believed to remedy at least one of them.

Bibliography

[ETHERNET] Jayant Kadambi, Ian Crayford, Mohan Kalkunte.
Gigabit Ethernet - Migrating to High-Bandwidth LANs.
Prentice Hall, 1998. ISBN 0-13-913286-4

[IBM1] *The CoreConnect™ Bus Architecture (White Paper)*
International Business Machines Corporation (IBM), 1999.
http://www.chips.ibm.com/products/coreconnect/

[VIRTEX] *Virtex™ 2.5 V Field Programmable Gate Arrays*
Xilinx Inc., 2002.

[XMB1] *MicroBlaze™ Hardware Reference Guide (version 2.2)*.
Xilinx Inc., 2002.

[XMB2] *MicroBlaze™ Software Reference Guide (version 2.2)*.
Xilinx Inc, 2002.

[XMB3] *MicroBlaze Development Kit Tutorial (version 2.2).*
Xilinx Inc, 2002.

[XMB4] *Designing Custom OPB Slave Peripherals for MicroBlaze (version 2.2).*
Xilinx Inc, 2002.

[XMB5] *MicroBlaze Soft Processor*.
Xilinx Inc, 2002.
4th November 2003. *http://www.xilinx.com/xlnx/xil_prodcat_product.jsp?title=microblaze*

[XAPP632] Marc Defossez.
Programming an FPGA via E-Mail.
In *Xilinx Application Note 632 (v 1.0)*, 2002.

[XTE1] Peter Alfke.
Moving Data Across Asynchronous Clock Boundaries
In *Xilinx techXclusives*, 2001.

[XEPF1] Mathew Oullette et al.
Platgen: OPB Limitation
In *Xilinx Embedded Processor Forum*, 2002.

[XPEF2] Brett Boren et al.
Xilnet: more meaningful examples please.
In *Xilinx Embedded Processor Forum*, 2003.

[XPEF3] Brett Boren, Trond Kortner et al.
XilNet: Incomplete tcp/ip stack.
In *Xilinx Embedded Processor Forum*, 2003.

[REALFAST] Tommy Klevin.
Get RealFast RTOS with Xilinx FGPAs.
In *Xilinx Journal*, Spring 2003.

[lwIP] Adam Dunkels.
lwIP - A Lightweight TCP/IP Stack.
4th November 2003. *http://www.sics.se/~adam/lwip/*

[MB-lwIP] Stefan Nilsson, Frederik Schmid, Jimmie Wiklander.
MB-lwIP: An lwIP implementation for MicroBlaze.
Luleå University of Technology, 2003.

[FOLDOC] Denis Howe et.al.
FOLDOC - Free On-Line Dictionary of Computing.
Imperial Collage Department of Computing.
4th November 2003. *http://foldoc.doc.ic.ac.uk/foldoc/*

[FPGACPU] FPGA CPU News.
4th November 2003. *http://www.fpgacpu.org/*

[OPENCORES] OPENCORES.ORG.
4th November 2003. *http://www.opencores.org/*

Copyrights & Trademarks

Appendix A

Eth Version 1.00 Revision A Schematics

Figure A.1: Eth Version 1.00 Revision A, TX Core

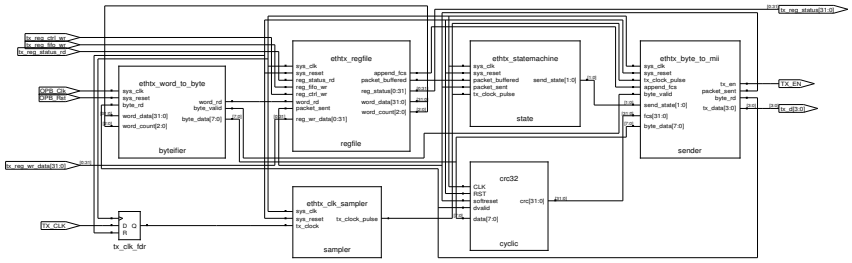
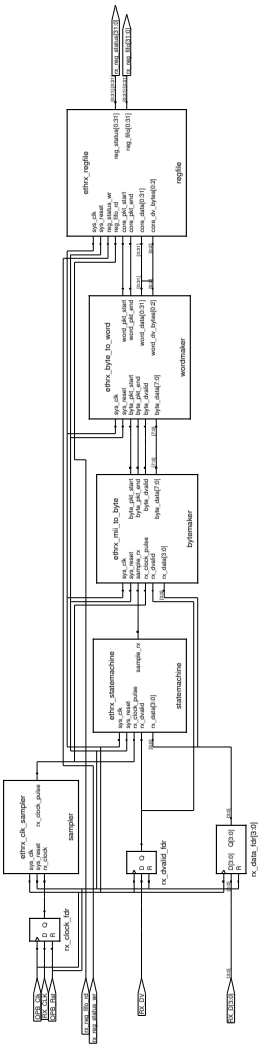


Figure A.2: Eth Version 1.00 Revision A, RX Core



Appendix B

XilNet implementation for Eth V 1.00 A

Figure B.1: XilNet test application

```
#define _CONFIG_ETH_
#include <eth.h>
#include <packet.h>
#include <eth_hwdr.h>
#include <ip.h>
#include <mbio.h>
unsigned char protype_board_mac_addr[6] = {0x00,0x00,0x3f,0xde,0xad,0xf0};
int main( void ) {
        int i;
        xilnet_eth_port_init( protype_board_mac_addr, ETH1_BASEADDR );
        mb_ip_addr[0] = 130;
        mb_ip_addr[1] = 240;
        mb_ip_addr[2] = 2;
        mb_ip_addr[3] = 24;
        while (1) {
                xilnet_eth_recv_frame( recvbuf, LINK_FRAME_LEN );
        }
}
```

Figure B.2: Implementation of xilnet_mac_send_frame()

```
void xilnet_mac_send_frame( unsigned char* frame, int len) {
        unsigned int i, fifo=0;
        while (! ETH_HWDR_TX_EMPTY(xilnet_base) ) ;

        for (i=0; i<len; i++) {
                switch(i&3) {
                case 0: fifo  = frame[i]<<24; break;
                case 1: fifo |= frame[i]<<16; break;
                case 2: fifo |= frame[i]<<8; break;
                case 3: fifo |= frame[i];
                         *ETH_HWDR_TX_FIFO(xilnet_base) = fifo;
                        break;
                }
        }
        if (i&3) *ETH_HWDR_TX_FIFO(xilnet_base) = fifo;
        ETH_HWDR_TX_SEND(xilnet_base, len);
        return ;
}
```

Figure B.3: Implementation of xilnet_mac_recv_frame()

```
int xilnet_mac_recv_frame ( unsigned char *frame, int len ) {
        unsigned int fifo=0;
        int i, length, lengthDiv4, lengthMod4;

        if (!xilnet_base) return 0;
        while ( ! ETH_HWDR_RX_BUFFERED(xilnet_base) ) ;
        length = ETH_HWDR_RX_PKT_LENGTH(xilnet_base);
        length = (length>len) ? len : length ;
        lengthDiv4 = length >> 2;
        lengthMod4 = length & 3;
        for (i=0; i<lengthDiv4; i++) {
                fifo = *ETH_HWDR_RX_FIFO(xilnet_base) ;
                (*frame++) = (fifo>>24) & 0xff;
                (*frame++) = (fifo>>16) & 0xff;
                (*frame++) = (fifo>>8) & 0xff;
                (*frame++) = fifo & 0xff;
        }
        if (lengthMod4) fifo = *ETH_HWDR_RX_FIFO(xilnet_base) ;
                switch(lengthMod4) {
                case 3:
                        (*frame++) = (fifo>>24) & 0xff;
                        (*frame++) = (fifo>>16) & 0xff;
                        (*frame++) = (fifo>>8) & 0xff;
                        break;
                case 2:
                        (*frame++) = (fifo>>24) & 0xff;
                        (*frame++) = (fifo>>16) & 0xff;
                        break;
                case 1:
                         (*frame++) = (fifo>>24) & 0xff;
                         break;
        }
        ETH_HWDR_RX_CLEAR_FIFO(xilnet_base);
        return (length>4) ? (length-4) : 0;
}
```

Figure B.4: Etherreal log, ICMP packet from Test PC to XilNet

```
Frame 43 (74 bytes on wire, 74 bytes captured)
    Arrival Time: Jun  2, 2003 12:39:36.893565000
    Time delta from previous packet: 0.000020000 seconds
    Time relative to first packet: 9.895098000 seconds
    Frame Number: 43
    Packet Length: 74 bytes
    Capture Length: 74 bytes
Ethernet II, Src: 00:d0:b7:b3:fc:f6, Dst: 00:00:3f:de:ad:f0
    Destination: 00:00:3f:de:ad:f0 (Syntrex_de:ad:f0)
    Source: 00:d0:b7:b3:fc:f6 (Intel_b3:fc:f6)
    Type: IP (0x0800)
Internet Protocol, Src Addr: 130.240.35.9 (130.240.35.9),
                   Dst Addr: 130.240.2.24 (130.240.2.24)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 60
    Identification: 0xa7aa
    Flags: 0x00
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 32
    Protocol: ICMP (0x01)
    Header checksum: 0xc815 (correct)
    Source: 130.240.35.9 (130.240.35.9)
    Destination: 130.240.2.24 (130.240.2.24)
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xc256 (correct)
    Identifier: 0x0100
    Sequence number: 8a:05
    Data (32 bytes)

0000  61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70   abcdefghijklmnop
0010  71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69   qrstuvwabcdefghi
```

Figure B.5: Etherreal log, ICMP packet from Xilnet to Test PC

```
Frame 44 (82 bytes on wire, 82 bytes captured)
    Arrival Time: Jun  2, 2003 12:39:36.894029000
    Time delta from previous packet: 0.000464000 seconds
    Time relative to first packet: 9.895562000 seconds
    Frame Number: 44
    Packet Length: 82 bytes
    Capture Length: 82 bytes
Ethernet II, Src: 00:00:3f:de:ad:f0, Dst: 00:d0:b7:b3:fc:f6
    Destination: 00:d0:b7:b3:fc:f6 (Intel_b3:fc:f6)
    Source: 00:00:3f:de:ad:f0 (Syntrex_de:ad:f0)
    Type: IP (0x0800)
    Trailer: 00000000000000000000000000000000...
Internet Protocol, Src Addr: 130.240.2.24 (130.240.2.24),
                   Dst Addr: 130.240.35.9 (130.240.35.9)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 28
    Identification: 0x0000
    Flags: 0x00
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 32
    Protocol: ICMP (0x01)
    Header checksum: 0xfde1 (incorrect, should be 0x6fe0)
    Source: 130.240.2.24 (130.240.2.24)
    Destination: 130.240.35.9 (130.240.35.9)
Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0xfffa (incorrect, should be 0x74fa)
    Identifier: 0x0100
    Sequence number: 8a:05
```

Appendix C

List of acronyms and abbreviations

- ASIC - Application Specific Integrated Circuit
- ARP - Address Resolution Protocol
- AUI - Attachment Unit Interface
- BUFGP - Primary Global Buffer for Driving Clocks or Longlines
- CPU - Central Processing Unit
- CSEE - Department of Computer Science and Electrical Engineering at Luleå University of Technology
- CSMA/CD - Carrier Sense Multiple Access / Collision Detect
- DA - Destination Address
- DCR - Device Control Register Bus (IBM CoreConnect)
- DMA - Direct Memory Access
- EDK - Embedded Development Kit
- EISLAB - Embedded Internet Systems Laboratory
- EMAC - Xilinx OPB Ethernet Media Access Controller
- EMAC Lite - Xilinx OPB Ethernet Lite Media Access Controller
- EthMac - Opencores.org 10/100 Ethernet MAC
- FCS - Frame Check Sequence
- FlashRAM - Flashable (Non-Volatile) Random Access Memory
- FPGA - Field Programmable Gate Array
- GHz - Giga Hertz

- GNU - GNU is Not Unix
- GPL - General Public License
- GUI - Graphical User Interface
- I/O - Input/Output
- IBUF - Input Buffer
- IBUFG - Dedicated Input Buffer
- IOBUF - Bi-Directional Buffer
- JTAG - Joint Test Action Group
- HDL - Hardware Description Language
- IBM - International Business Machines Corporation
- ICMP - Internet Control Message Protocol
- IP - Internet Protocol
- IPv4 - Internet Protocol Version 4
- LMB - Local Memory Bus
- lwIP - Light Weight Internet Protocol
- MAC - Media Access Controller
- MAU - Medium Attachment Unit
- MBit/s - Mega Bit per second
- MDK - MicroBlaze Development Kit
- MHz - Mega Hertz
- MII - Medium Independent Interface
- MScE - Master of Science and Engineering
- OBUF - Output Buffer
- OPB - On-Chip Peripheral Bus (IBM CoreConnect)
- OSI - Open Systems Interconnect
- PhD - Philosophiae Doctor / Doctor of Philosophy
- PHY - Physical Layer Device
- PLB - Processor Local Bus (IBM CoreConnect)
- PLD - Programmable Logic Device
- PROM - Programmable Read-Only Memory

- RAM - Random Access Memory
- RISC - Reduced Instruction Set Computer
- RJ45 - A socket interface for twisted pair cables
- RMII - Reduced Medium Independent Interface
- ROM - Read-Only Memory
- RS-232 - Recommended Standard 232 (asynchronous serial line standard)
- SA - Source Address
- SoC - System on Chip
- SRAM - Static Random Access Memory
- TCP - Transmission Control Protocol
- TCP/IP - The Internet Protocol Version 4 protocol suite (TCP, UDP, ICMP etc)
- TYPE - Type-field (Ethernet frames)
- SYNC - Synchronization bits (Ethernet frames)
- UART - Universal Asynchronous Receiver/Transmitter
- UDP - User Datagram Protocol
- VHDL - Very High Speed Integrated Circuit (VHSIC) Hardware Description Language.
- VHSIC -Very High Speed Integrated Circuit (VHDL)
- XST - Xilinx Synthesis Technology