

## Considerations for High-Bandwidth TCP/IP PowerPC Applications

by Chris Borrelli, Embedded Networking Manager, Xilinx, Inc.

The TCP/IP protocol suite is the de facto worldwide standard for communications over the Internet and almost all intranets. Interconnecting embedded devices is becoming standard practice even in device classes that were previously stand-alone entities. By its very definition, an embedded architecture has constrained resources, which is often at odds with rising application requirements.

Achieving wire-speed TCP/IP performance continues to be a significant engineering challenge, even for high-powered Intel™ Pentium™-class PCs. In this article, we'll discuss the per-byte and per-packet overheads limiting TCP/IP performance and present the techniques utilized to maximize TCP/IP over Gigabit Ethernet performance in embedded processor-based applications.

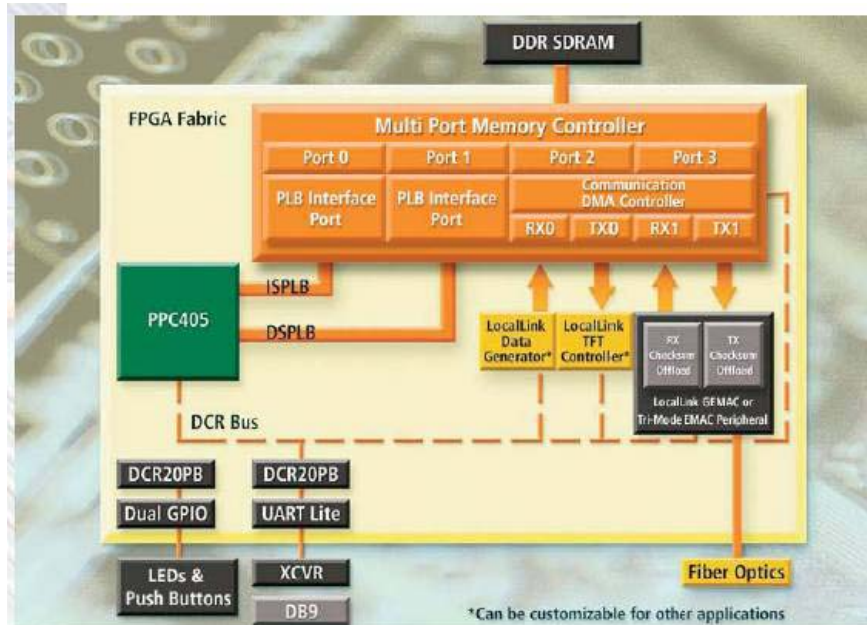
### Overview

Gigabit Ethernet performance is achieved by leveraging a multi-port DDR SDRAM memory controller to allocate memory bandwidth between embedded PowerPC™ processor local bus (PLB) interfaces and two data ports. Each data port is attached to a direct memory access (DMA) controller, allowing hardware peripherals high-bandwidth access to memory.

### System Architecture

Memory bandwidth is an important consideration for high-performance network attached applications. Typically, external DDR memory is shared between the processor and one or more high-bandwidth peripherals such as Ethernet. A multi-ported memory controller efficiently divides the available memory bandwidth between processor interfaces and streaming peripherals, including Ethernet. The streaming peripherals are linked to memory through a point-to-point streaming interface via Direct Memory Access (DMA)

controllers. The DMA controller implements a scatter-gather scheme whereby multiple buffers are converted to/from a contiguous stream on the Ethernet or other streaming peripheral. The Ethernet peripheral implements checksum offload on both the Transmit and Receive paths for optimal TCP performance. A block diagram of the system described above is shown in Figure 1.



## TCP/IP Per-Byte Overhead

Per-byte overhead occurs when the processor touches payload data 1. The two most common operations of this type are buffer copies and TCP checksum calculation. Buffer copies represent a significant overhead for two reasons:

1. Most of the copies are unnecessary.
2. The processor is not an efficient data mover.

TCP checksum calculation is expensive, as it is calculated over each payload data byte. Embedded TCP/IP-enabled applications such as medical imaging require near wire speed TCP bandwidth to reliably transfer image data over a Gigabit Ethernet network. The data is generated from a high-resolution image source, not the processor. In this case, introducing a zero-copy software API and offloading the checksum calculation into FPGA fabric completely removes the per-byte overheads. "Zero-copy" is a term that describes a TCP software

interface where no buffer copies occur. Linux and other operating systems have introduced software interfaces like `sendfile()` <sup>2</sup> that serve this purpose, and commercial standalone TCP/IP stack vendors like Treck TM offer similar zero-copy features. These software features allow the removal of buffer copies between the user application and the TCP/IP stack or operating system.

The scatter-gather and the checksum offload features of the system provide the hardware support necessary for zero-copy functionality. The scatter-gather feature is a flexibility of the DMA controller that allows software buffers to be located at any byte offset. This removes the need for the processor to copy unaligned or fragmented buffers.

Checksum offload is a feature of the Ethernet peripheral. It allows the TCP payload checksum to be calculated in FPGA fabric as Ethernet frames are transferred between main memory and the peripheral's hardware FIFOs. These system features remove the need for costly buffer copies and processor checksum operations, leaving the processor to perform protocol operations and user functions.

### **TCP/IP Per-Packet Overhead**

Per-packet overhead is associated with operations surrounding the transmission or reception of packets <sup>1</sup>. Packet interrupts, hardware interfacing, and header processing are examples of per-packet overheads. Interrupt overhead represents a considerable burden on the processor and memory subsystem, especially when small packets are transferred. Interrupt coalescing is a technique used in such a system to alleviate some of this pressure by amortizing the interrupt overhead across multiple packets. The DMA engine waits until there are  $n$  frames to process before interrupting the processor, where  $n$  is a software-tunable value.

Transferring larger sized packets (jumbo frames of 9,000 bytes) has a similar effect by reducing the number of frames transmitted, and therefore the number of interrupts generated. This amortizes the per-packet overhead over a larger data payload.

### **Implementation**

An example implementation of this architecture is the Gigabit System Reference Design from Xilinx (GSRD). It is geared toward high-performance

bridging between TCP/IP-based protocols and user data interfaces like high-resolution image capture or Fibre Channel. The components of GSRD contain features to address the per-byte and per-packet overheads of a TCP/IP system. For applications requiring an embedded operating system, a MontaVista™ Linux™ port is available while a commercial standalone TCP/IP stack from Treck™ is available to satisfy applications with the highest bandwidth requirements.

The GSRD can provide Transmit TCP performance up to 890Mbps using jumbo frames, and is implemented in the latest FPGA technology available today from Xilinx. The GSRD can be downloaded today from <http://www.xilinx.com/gsr/>.

*by Chris Borrelli, Embedded Networking Manager, Xilinx, Inc.*

*July 26, 2005*

#### References:

1. "End-System Optimizations for High-Speed TCP" ([www.cs.duke.edu/ari/publications/end-system.pdf](http://www.cs.duke.edu/ari/publications/end-system.pdf))
2. "Use sendfile to optimize data transfer" (<http://builder.com.com/5100-6372-1044112.html>)