White Paper

Accelerating Nios II Ethernet Applications

Ethernet is a popular communications media chosen for its combination of high throughput, the well-known TCP/IP sockets application programming interface (API), low-cost hardware, widely available infrastructure support, and the ability to communicate over long distances. However, sometimes it can be a challenge to achieve the desired Ethernet throughput along with the other system requirements. Designers are continually looking for solutions to increase throughput for their Ethernet applications.

This paper describes methods for accelerating Nios® II embedded processor TCP/IP Ethernet applications. The inherent flexibility of the Nios II embedded processor, SOPC Builder system integration tools, and Altera® FPGAs means there are many options for accelerating the Ethernet performance. Besides the normal FPGA optimizations and system architecture choices, software and hardware acceleration solutions are available that can have a huge impact on system throughput. This paper focuses on some of the highest performing techniques and solutions available for Nios II TCP/IP Ethernet applications.

Ethernet Performance

Before discussing how to improve Ethernet performance, this white paper discusses typical embedded Ethernet systems and the hinderances to achieving higher performance. This discussion does not exhaust all the areas that could be improved. The topics discussed here are some of the areas that have the most impact on performance.

One key area that dramatically impacts performance is the memory and CPU overhead of protocol data manipulation. In basic implementations of the TCP/IP protocol stack, a significant amount of CPU time and memory bandwidth is spent copying portions of the packet (payload, headers, checksums, etc.) until the packet is completely assembled. A similar process occurs for parsing a packet after it is received. At the physical level, while Ethernet packets are sent and received, the contents are copied between memory and the Ethernet media access controller (MAC). As a packet is received, it is copied from the Ethernet MAC into memory. As a packet is transmitted, it is copied from memory to the Ethernet MAC. Also, at higher layers of basic TCP/IP protocol stack implementations, while the frames are assembled and parsed, the contents are copied from one memory buffer to another. Considering that the payload of an Ethernet packet can be 1,500 bytes, these data copies consume a significant number of CPU cycles.

Another operation that consumes a fair amount of CPU cycles is calculating the checksum. A checksum is calculated for the IP header and TCP header and payload (or user datagram protocol (UDP) header and payload for UDP packets). This checksum is calculated by the transmitter and included with each packet sent. The receiver must perform the checksum calculation as well to ensure that the contents of the packet have not been corrupted upon transmission.

One factor that inhibits higher performance for some embedded Ethernet systems is the clock frequency. Typically, embedded systems do not run nearly as fast as PCs. Usually, they run at least one order of magnitude (10x) slower. This is one of the reasons why PCs have higher Ethernet performance. While the CPU clock frequency and Ethernet performance are not necessarily linearly related, this difference in clock frequency makes a huge difference in Ethernet performance. At the frequencies that most embedded systems run, increasing the clock frequency has a large impact on the Ethernet throughput.

Improving Performance in Nios II Systems

Before focusing on optimization techniques directly related to the TCP/IP Ethernet portion of the design, it is important to investigate general performance improvement techniques. This section discusses techniques for improving performance for Nios II systems independent from the TCP/IP Ethernet portion of the design.

Use the Fastest Nios II Embedded Processor Core

Of the three Nios II processor cores available, the highest performing is the Nios II/f core. This core's features increase the overall efficiency of the processor: features such as instruction cache, data cache, dynamic branch prediction, hardware multiply, hardware divide, and a barrel shifter. Also, the Nios II/f processor core is designed to run at a faster clock frequency. Therefore, when extra system performance is necessary, the Nios II/f processor is the best choice.

Use Memory With Fast Access Times & Low Latency

Memory access time and latency can have a large impact on overall system performance. In many systems, numerous cycles are spent retrieving instructions from memory and accessing data in memory. With the flexibility of Altera FPGAs, the Nios II embedded processor, and SOPC Builder, the designer can choose the optimal memory for their system needs. These tools make it easy to partition instruction code and data memory into multiple memory devices. The more critical code and data sections can be placed into faster memories. For faster memory access, it is recommended to use the memory blocks inside the FPGA or a fast off-chip memory such as SRAM.

Increase Instruction Cache & Data Cache Sizes

As memory bandwidth is often critical to maximize system performance, the size of the data and instruction caches can have a significant effect on the performance. As mentioned earlier, typically, TCP/IP applications involve numerous data copies and manipulation of the same data. Increasing the data cache size minimizes latency due to the cache misses and memory access. Increasing the instruction cache improves performance since it improves the latency for fetching the instructions from memory. The Nios II/f processor has configurable instruction and data cache which allows designers to choose the size needed for each application. If the main program and data memories have fast access times, then increasing the cache sizes may not significantly improve the overall performance. Benchmark the application for various sizes of the instruction cache and data cache to find the optimal sizes.

Use Optimal Period for System Clock Timer

When system performance is a critical requirement, it is important to minimize unecessary activity. For example, the system clock timer interrupt occurs frequently in most embedded systems. Every time the system clock timer interrupt occurs, CPU cycles are spent switching the context. The context must be switched twice: once before executing the timer interrupt status register (ISR), and again to switch back to the interrupted code. If the period of the system clock timer is shorter than it needs to be, the CPU spends an unnecessary percentage of cycles servicing the timer interrupt. These cycles could be spent on other operations that would increase the overall performance of the system. Make sure to choose the longest period for the system clock timer while still meeting all the system requirements (e.g., OS clock tick requirements and system interrupt response requirements).

Increase Clock Frequency

Certain architectural decisions can have a significant effect on the clock frequency. For example, by isolating the data path from the FPGA to the Ethernet MAC/PHY device, a faster clock frequency occurs. On the other hand, if the board is designed such that the MAC/PHY device shares data, address, and control lines with other devices, then there will be a wide multiplexer inside the FPGA. This wide multiplexer makes it difficult to place-and-route the design while maintaining a fast clock frequency. Therefore, to get better performance, design the system such that the Ethernet interface has dedicated I/O signals between the FPGA and the MAC/PHY device.

SOPC Builder version 5.0 supports multiple clock domains that allows for selecting a clock domain for each peripheral. Partitioning the design into multiple clock domains may allow a portion of the design to run at a faster clock frequency. For example, the Nios II processor, Ethernet interface, and associated memory can be placed on the fast clock domain with the rest of the system on a slower clock domain. For more information on multiple clock domain support in SOPC Builder, refer to the *Building Systems with Multiple Clock Domains* chapter of the Quartus® II Development Software Handbook (www.altera.com/literature/hb/qts/qts_qii54008.pdf).

There are other optimization techniques related to general FPGA optimization. For information on using the Quartus II software to optimize designs, refer to the Quartus II Development Software Handbook (www.altera.com/literature/lit-qts.jsp).

Improving Ethernet Performance for Nios II Systems

This section discusses methods for improving the performance of Nios II Ethernet applications. Techniques for addressing the bottlenecks mentioned previously are discussed.

Implement Hardware Acceleration

This section discusses ways to improve Ethernet performance using hardware acceleration with the Nios II embedded processor. There are several operations performed in Ethernet applications that are much faster when implemented in hardware versus software.

Typical networking applications include numerous data copies that can consume many CPU instruction cycles. Using dynamic memory access (DMA) for some of the data transfers can produce a significant improvement in throughput. After setting up the DMA transfer, the Nios II processor can perform other tasks while the DMA is transferring data. One area of the system that can take advantage of a DMA is the MAC in conjunction with the TCP/IP stack. Thus, the MAC driver must be written to use a DMA instead of software for copying the data. Using a MAC, which utilizes a DMA, frees up the processor to execute other operations while data is being transferred. For a MAC without DMA support, the processor would have to transfer the data via software, which is typically much slower than a DMA.

As discussed earlier, another operation that makes an impact on the performance is the calculation of the checksum for the IP header and TCP/UDP header and payload. As with virtually all mathematical operations, the hardware implementation is much faster than the same function implemented in software. Offloading the checksum calculation to hardware as a Nios II custom instruction or custom peripheral increases the checksum performance by a significant amount when compared to a software implementation. The exact improvement in Ethernet performance depends on the application and the size of the packets (as well as the original checksum software algorithm).

Another operation that occurs in Ethernet systems that use a little endian processor such as the Nios II embedded processor, is switching the byte ordering (i.e., endianness) of the data as the packets are being

assembled and parsed. TCP/IP stacks usually have a software macro to handle converting the byte order from host order to network order and network order to host order. Since this operation is performed frequently, the overhead adds up. Implementing this operation in hardware as a custom instruction of the processor would decrease this overhead. The Nios II processor supports custom instructions, which means that a byte ordering custom instruction can be implemented to offload this operation from the Nios II CPU.

These hardware acceleration methods are features supported in the MAC and TCP/IP stack used in the system. Upcoming sections in this document describe solutions for using accelerated MAC and TCP/IP stacks in Nios II Ethernet applications.

Optimize TCP/IP Stack Implementation

Typically, TCP/IP stack implementations give the designer some flexibility in configuring the stack. This allows the designer to modify the implementation to suit the requirements of the design. For example, the lightweight IP (lwIP) TCP/IP stack included in Nios II development kits has many parameters that can be modified to improve performance. Usually, the tradeoff for performance is a larger memory footprint. Some of the parameters that have the most impact are the TCP/IP heap size and the transmit and receive buffer sizes. For more information on optimizing lwIP throughput, refer to the *Using Lightweight IP with the Nios II Processor Tutorial* (www.altera.com/literature/tt/tt_nios2_lwip_tutorial.pdf).

Utilize Accelerated On-Chip MAC Cores

In embedded Ethernet systems, it is common for the Ethernet MAC to be integrated with the Ethernet physical device into a MAC/PHY device. For faster performance, consider using a MAC core implemented on the Altera FPGA (an on-chip MAC) which incorporates some of the improvements discussed in this document. For example, two of the most common hardware acceleration features of accelerated MAC cores are using a DMA to copy data, and offloading the checksum calculation to hardware. Some on-chip MAC cores implement even more operations that increase performance, such as performing the byte ordering operating in hardware.

Utilize Accelerated TCP/IP Stacks

Some specialized TCP/IP stack implementations are optimized to obtain higher Ethernet throughput while minimizing system overhead. These accelerated stacks are written to take advantage of the available hardware acceleration features of the MAC as well as to utilize a range of techniques to improve performance of the TCP/IP stack operations themselves. When extra system performance is needed, consider using an accelerated TCP/IP stack.

Accelerated Software & Hardware IP

Many options are available for the MAC and TCP/IP stack implementations for Nios II embedded Ethernet systems. This section discussed the advantages of a particular accelerated MAC core and TCP/IP stack implementation.

Accelerated On-Chip MAC

The MAC-NET core from MorethanIP implements the hardware acceleration features mentioned earlier along with a byte ordering and 32-bit boundary payload alignment. Below are some of the key performance features of the MorethanIP accelerated MAC (MAC-NET) core:

- Support for 32-bit DMA transactions to copy IP frames directly from and to FIFOs ensuring 32-bit alignment for more efficient access.
- Support for TCP, UDP, ICMP and IP layer (IPv4 and IPv6) hardware checksum (significantly faster than software algorithms for checksum calculation).
- On-the-fly byte order conversion (network-to-host and host-to-network) for IP and TCP/UDP/ICMP headers.
- Programmable Ethernet payload alignment on 32-bit boundaries for more efficient software access on protocol headers.
- Frame pre-filtering on receive (discard on checksum errors, IP errors, etc.) to reduce unnecessary processing of irrelevant packets.

The checksum and DMA features are configurable so that the designer can choose which of these features to include. This allows the designer to trade off size and performance to suit their design requirements.

The MorethanIP MAC cores work with various TCP/IP stacks including Interniche's TCP/IP stack implementation, NicheStack, and lwIP, the TCP/IP stack implementation included in Nios II development kits. For more information about MorethanIP and their products, refer to their web site (www.morethanip.com).

Accelerated TCP/IP Stack

Interniche offers a networking TCP/IP stack, NicheStack IPv4, that implements a range of architectural- and implementation-dependent optimizations while maintaining a low memory footprint and without compromising interoperability. Below is a list of the key features that increase the performance of the NicheStack IPv4 product:

- True zero-copy architecture: eliminates data copies from buffer to buffer as IP packets pass up and down the communication abstraction layers. This not only increases throughput, but also lowers the stress on memory bandwidth in higher performance systems.
- Optimized functional abstraction: the product is architected to eliminate as many layers of subroutine/function calls as possible by grouping operational features in optimal blocks of code. In contrast to many open source offerings, the code base is substantially reorganized and re-optimized on a regular basis as features/functions are added and issues are addressed (as opposed to patchwork enhancement).
- Compact code: The InterNiche focus on small memory footprint also improves performance since more optimized code flow equates to higher performance.
- Low multi-threading overhead: Process and context switching overhead is dramatically reduced by a very lightweight approach to multi-threading and an efficient integration layer with any underlying RTOS environment.
- Ease of integration with hardware features: Flexibility and portability are extended by offering effective mechanisms and integration points to incorporate acceleration code that exploits specific capabilities of the underlying hardware. Examples of this include:
  - Optimized interfaces to high-performance MAC implementations
  - Use of special instructions to accelerate protocol processing functions
  - Leverage of hardware encryption engines, where they exist, to reduce the overhead of secure protocols such as SSL

The NicheStack products work with various operating systems (including MicroC/OS-II) and MACs, and have been integrated with the MAC-NET core from MorethanIP. For more information on Interniche and their products, refer to their web site (www.iniche.com).

Ethernet Solutions for the Nios II Embedded Processor

Nios II development kits come with a driver for the MAC/PHY on the development board as well as the open source lwIP TCP/IP stack.

As with all design decisions, there are tradeoffs involved. For Nios II Ethernet solutions the main tradeoffs are cost and logic utilization versus performance. Table 1 summarizes these tradeoffs, comparing the solution included in the Nios II development kits with other high-performance solutions.

Table 1: Comparison of Nios II Kit Ethernet Solutions vs. Other High-Performance Solutions

MAC	TCP/IP stack	OS	TCP tx (Mbits/s)	TCP rx (Mbits/s)	Clock Frequency (MHz)	Logic Utilization (Logic Elements)	Cost
SMSC LAN91C111 MAC/PHY	lwIP	MicroC/OS-II	450 kbits/s	176 kbits/s	100	minimal	Free(1)
MorethanIP MAC-NET core	NicheStack	none	63	62	84	~4,500	Contact MorethanIP & Interniche
MorethanIP MAC-NET core	lwIP(2)	none	39	56	84	~4,500	Contact MorethanIP

Notes:
(1) Drivers for the lan91c111 chip and lwIP TCP/IP stack are free with Nios II development kits. Refer to the Altera web site for information on Nios II development kits (www.altera.com/nioskits).
(2) Modified version of lwIP stack. Contact IP vendor for more information.

The benchmark applications consist of a program running on a host PC which communicates with an application running on the Nios II target system. For testing TCP receive throughput, the PC sends TCP packets with a payload size of 1,024 bytes to the Nios II system. For testing TCP transmit throughput, the Nios II system sends TCP packets with a payload of 1,024 bytes to the host PC. These results are based on version 1.1 of the Nios II embedded processor. The example design for this solution is available in the Reference Design section of the Altera web site (www.altera.com/solutions/refdesigns/sys-sol/computing/ref-accel-network.html). For more information about the accelerated solutions, contact the corresponding vendors.

Summary

Using Ethernet as a communication media in high-performance embedded systems is increasingly popular. The ability to support various TCP/IP Ethernet implementations gives a designer many options when designing a Nios II processor system. Full Ethernet solutions are available with Nios II development kits. For applications that need higher Ethernet performance, use the hardware resources in the FPGA as well as Nios II hardware acceleration techniques along with third-party IP (hardware and software). With the inherent flexibility of the Nios II processor, SOPC Builder, and Altera FPGAs, there are many options for increasing the Ethernet performance for Nios II processor systems.

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com