A Versatile, Low Latency HyperTransport Core

David Slogsnat Alexander Giese Ulrich Brüning

University of Mannheim
Computer Architecture Group
B6 26
68131 Mannheim
Germany
{slogsnat,agiese,bruening}@uni-mannheim.de

ABSTRACT

This paper presents the design of a generic HyperTransport (HT) core. It is specially optimized to achieve a very low latency. The core has been verified in system using the rapid prototyping methodology with FPGAs. This exhaustive verification and the generic design allows the mapping to both ASICs and FPGAs. The implementation described in this paper supports a link width of 16bit, as is used in Opteron based systems. On a Xilinx Virtex4FX60, the core supports a link frequency of 400MHz DDR and offers a maximum bidirectional bandwidth of 3.6 GB/s. The in-system verification has been performed using a custom FPGA board that has been plugged into a HyperTransport Extension Connector (HTX) of a standard Opteron based mainboard. HTX slots in Opteron based mainboards allow a very high-bandwidth, low latency communication, as the HTX device is directly connected to one of the HyperTransport links of the processor. HyperTransport is a packet-based interconnect technology for low-latency, high-bandwidth point-to-point connections. The HT core in combination with the HTX board is an ideal base for prototyping systems and FPGA coprocessors. The HT core is available as open source.

Categories and Subject Descriptors

B.4.3 [**Hardware**]: Interconnections (Subsytems) - *Interfaces*

General Terms

Design, Verification

Keywords

FPGA, HyperTransport, HTX, Prototyping, RTL

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
FPGA'07, February 18–20, 2007, Monterey, California, USA.
Copyright 2007 ACM 978-1-59593-600-4/07/0002...$5.00.

1. INTRODUCTION

Cluster computing is becoming an increasingly important and popular trend in high performance computing. Current state of the art in cluster computing is to use commodity computing nodes and adding communication devices like Myrinet, Infiniband or Quadrics. This approach replaces the standard interconnect, which is the least performing component in those systems, by a high performance System Area Network (SAN). The result is a significant speedup of the parallel system. At the same time, it enables low system costs through the use of COTS (Components off the shelf) for most parts of the system. Within the last decade, much effort has been spent in the development and improvement of SANs. With success: routing mechanisms and chip fall-through latencies have improved steadily. Additional functional features like embedded processors, user-level communication, memory management units and support for frequent parallel communication patters have been added. As a result, today's SANs are very mature and highly optimized devices.

Nevertheless, such systems have a serious weakness: computing and communication resources are only loosely coupled over a hierarchy of buses, with the bottleneck being a high-latency, non-cache coherent I/O bus. Currently, PCI-X and PCI Express are commonly used for that. One of the implications is that a large fraction of the end-to-end message latency in modern SANs is introduced by the I/O bus. Latency is however one of the most critical performance criteria of SANs. Thus, the efficient fusion of computing and communication into a scalable network of computers constructed from basic building blocks is still an unreached goal. To overcome this limitation, computation and communication resources have to move much closer to each other.

AMD's Opteron processors offer a unique opportunity to do so, as they do not use a proprietary front side bus, but 3 HyperTransport interfaces per processor. The adoption of the HyperTransport Expansion Connector (HTX) specification [3] and mainboards that are equipped with this connector now make it possible to directly connect devices to an Opteron processor. This results in a major increase of performance [4], which also can be seen in practice with

Pathscale's InfiniPath, currently the only HyperTransport-enabled NIC. Of course, not only NICs benefit from the direct HyperTransport connection. Any device with high bandwidth or latency requirements, as FPGA coprocessors for example, will gain from the increased performance.

A very challenging task appears when FPGAs are used to interface the HT link of the Opteron processor, no matter if it is used for prototyping or in a production system. Current Opteron processors use Hypertransport link speeds of 1GHz DDR, a value which will very likely increase steadily over time. [22] found out that the gap in critical path delay from FPGA to a standard-cell ASIC is roughly a factor of 3 to 4. The link speed in Hypertransport is negotiated between both devices sharing that link, and is usually set to the highest common frequency, so that the HT link between processor and FPGA will work without problems, even if the FPGA supports only lower link frequencies. However, performance suffers from this, which of course is not desirable. It is thus a very important issue to implement HyperTransport links in FPGAs with an HT link speed that is not significantly lower than the original processor link speed.

Another issue is the limited size of FPGAs. [22] found out that FPGA designs require a silicon area that is 21 to 40 times higher than the respective ASIC implementation. Consequently, our experience is that even the largest FPGAs available do not privide enough space for complete prototypes of complex systems. Rather, only parts of the system can be implemented on the FPGA at the same time. As the HT core has to be present in almost any case of scenarios, it is of particular importance to keep the required area consumption as low as possible.

In this paper, we will describe such an FPGA implementation of a fully functional HyperTransport core. It will be used to prototype new SAN interfaces currently under development in our group, which are based on ideas of and experience with the Atoll SAN [16]. The remainder of this paper is organized as follows. Chapter 2 gives the background about the HyperTransport protocol and how it is used in Opteron based systems. Related work is summarized in Chapter 3. Details of the implementation are described in Chapter 4. Results of the FPGA implementation are given in Chapter 5, while Chapter 6 describes how the core has been verified in-system. Chapter 7 concludes this paper.

2. BACKGROUND

Today, devices within a COTS-system are connected over PCI, PCI-X, or PCI-Express I/O buses. PCI-Express offers the highest bandwidth and thus is currently prevailing among these I/O buses. [5] shows that the HyperTransport protocol offers significantly lower latencies than PCI-Express. This is due to the fact that PCI-Express uses a small number of high speed serial lines, instead of a larger number of lines with reduced frequencies as HT does. This high-speed serialization and de-serialization process generates latency itself, and additionally requires to recode all transmitted data into a DC-free code. Any latency-tolerant application will benefit from the reduced bus latency of HT. Nevertheless, the main reduction in latency comes from the fact that the HTX slot is directly connected to the Opteron processor, instead of having to go through one or more I/O bridges, as depicted in Figure 1. This avoids time-consuming protocol conversions in these bridges and synchronization FIFOs between the different clock domains.

Opteron processors can use up to three HT links for cache-coherent communication between different processors in multiprocessor systems. In this case, they use the cache-coherent HT protocol, which is not part of the public HT specification, as it is AMD proprietary and confidential. To connect devices or other bridges to the processor, the respective HT links are configured to be non-coherent and thus use the open HyperTransport specification. In this case, the HT link within the processor has to translate accesses from the noncoherent domain into the coherent domain and vice versa. By doing so, every such noncoherent HT link has the functionality of an I/O bridge. Due to the similarity of non-coherent and coherent HT protocols, protocol conversions and synchronizations are less expensive than in other I/O bridges.

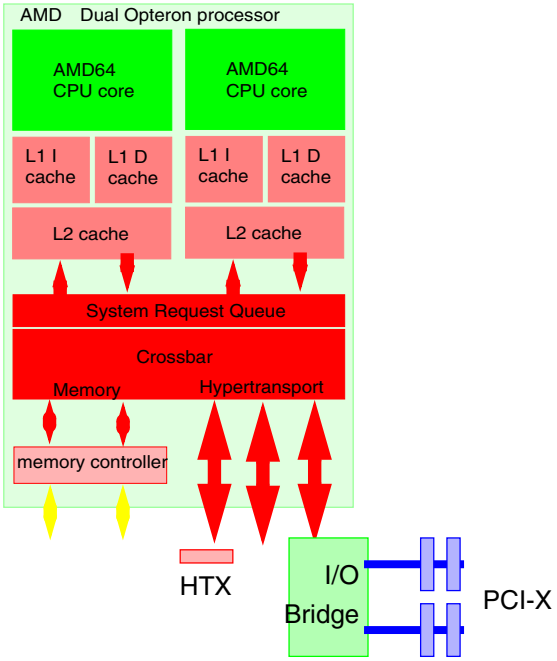

Figure 1:Block diagram of an Opteron dual core processor in a system with HTX and PCI-X slots

2.1 The HyperTransport Protocol

HyperTransport (HT) is a packet-based communication protocol for data transfer. There are three versions of HyperTransport: HT 1.05 has been developed in 2001, and was updated by HT 2.0 in 2004. In April 2006, HT 3.0 [2] has been defined as the next successor. Current Opteron processors adhere the HT 2.0b specification [1], there are no HT 3.0 devices or systems available yet. Therefore this work focuses on the implementation of an HT 2.0b device. Additionally to the HyperTransport specification, [6] defines the precise behaviour, and in particular the initialization, of HT devices that are used in Opteron based systems.

A HyperTransport link consists of two sets of unidirectional signals. Each set can be distinguished into three signal types: CAD (command, address, data), CTL (control) and CLK (clock signals). The CAD lines are used to transport command and data packets, while the CTL line distinguishes between command or data packets on the CAD lines. The HT protocol supports CAD buses with a width of 2,4,8,16 or 32 bit, as depicted in Table 1. The width of the CAD bus is usually called the width of the HT link. If more than 8 CAD lines are used per link and direction, every group of 8 signals has its own CLK signal. These groups of signals are synchronously transmitted with the source associated CLK signal. This means that one CLK and its associated group of CAD signals must be routed with equal length traces in order to minimize skew. The data transferred on the CAD bus is 32bit aligned, independently of the bus width. All transferred packets have at least a size of one doubleword, i.e. 32bit. HT allows frequencies from 200MHz to 1.4GHz in HT 2.0 and up to 2.6GHz in HT 3.0. Current Opteron processors use link widths of 16 bit and frequencies up to 1GHz. In Opteron systems, all devices start at powerup of the system with 200MHz and 8bit wide links. The BIOS checks the capabilites of all devices by accessing the device's HT register space, and sets new values for frequency and width for every link according to the capabilites of the two devices that share the link. After that, it forces a reinitialization of all HT devices to establish the new parameters.

Table 1: HT link widths

Signal	Narrow	HTX	Wide	Description
CLK	1	2	4	Clock signals from HT Device 0
CTL	1	1	1	CTL signal from HT Device 0
CAD	2,4,8	16	32	Command, Address, Data from HT Device 0

HyperTransport topologies consists of three different device types, which are distinguished by their connection to other HT devices. Generally, HyperTransport devices are connected in chains. There can be up to 32 devices in one single chain. Different chains can be connected with each other by HyperTransport bridges. The top of a chain is always a bridge. Caves have a single link and are connected to one other device, thus they form the lower end of a HT chain. Tunnels have two links and are connected at least with the upstream link with one device, or with both links to different devices. HTX currently only supports cave devices.

The transfer with HyperTransport is packet based. In order to decouple the transmission response from the request, the packets are transferred in a split phase transaction. Split phase transactions work in a way that the initiator of a transaction sends a request and can afterwards continue with other tasks, so that it does not have to wait for an immediate response. Then, the receiver of the request computes the response and sends it back at a later time. This basic function is shown in Figure 2 with the example of a read and write operation. A transfer always starts with a control packet. Three types of control packets can be distinguished: information, request, and response packets. Information packets are used for flow control and synchronization. Request packets are sent to write data to a receiver, and are also used to initiate requests. Response packets contain the answer to a corresponding request. Control packets have a size of 4 or 8 bytes or, if they use addresses of 64bits instead of 40bit addresses, the extended format with a size of 12 bytes. If a transfer contains payload data, the next data packet which is sent on the link belong to this packet. A data packet can have a maximum size of up to 64 bytes. Sending other control packets during a stream of data packet at every 32 bit boundary is allowed, but only if this control packet would not be followed by data. Otherwise it could not be possible to determine which control packet the data belongs to. This mechanism makes it possible to send urgent control packets with priority.

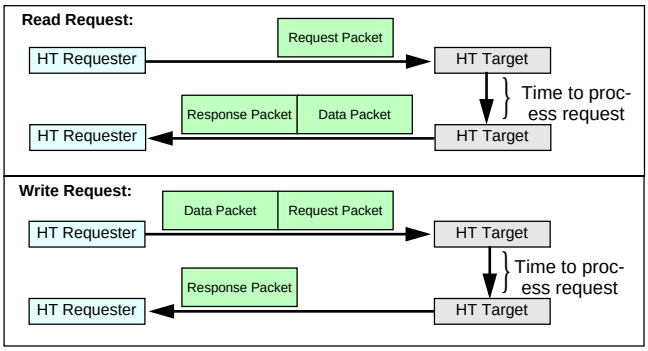

Figure 2:Read and Write examples

Packets travel in different virtual channels in order to avoid deadlocks. Within these channels, all data packets move along with the control packets. The virtual channels are classified into three sets: Posted Requests, Nonposted Requests, and Responses. Posted requests do not get a response packet from the receiver. Non-Posted requests always need a response to complete the outstanding transaction. However, these sets are not totally independed of each other, as there is the option to order Nonposted Requests and Responses in relation to Posted Packets on a packet by packet basis.

3. RELATED WORK

In the past, many projects have successfully directly interfaced processors for communication [14]. The number of recent projects in which FPGAs interface directly to a high-performance processor is very limited. One major reason for this is that processor interfaces are usually proprietary and thus information about the protocol is not available. Recent projects that access the FSB of an processor are [17] [18]. In both projects, the FPGA is connected to one slot of a dual-processor Pentium-mainboard. The front side bus of the Pentium processor runs with 66MHz, which does not pose technological challenges. MemoNet [15] goes a different approach to overcome the performance gap of I/O buses. The NIC is connected to the SO-DIMM slots of the memory controller. While this approach offers higher bandwidths and lower latencies than classical I/O buses, the NIC can not issue DMA accesses or interrupts. It can only act as a slave device, thus, this approach is very limited compared to the HTX interface.

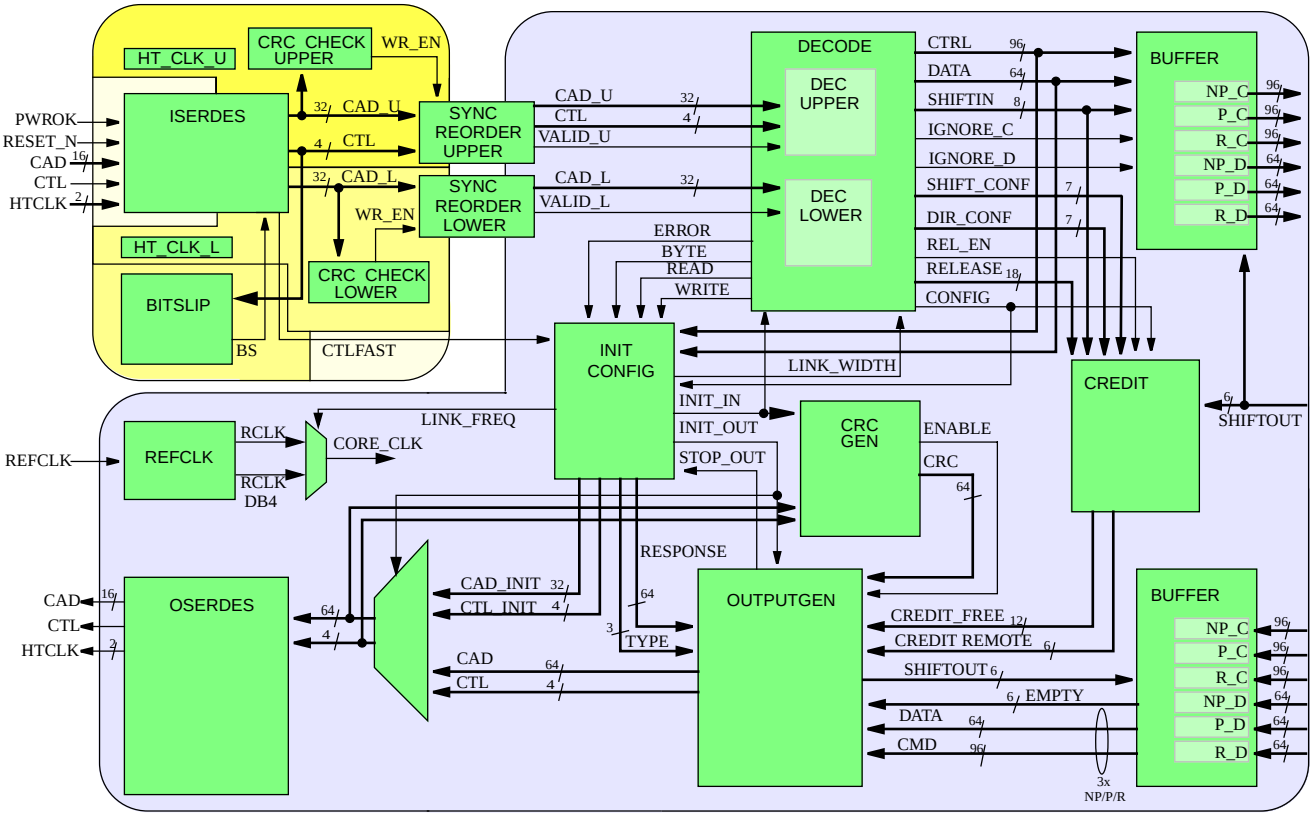

Figure 3:Block Diagram of the Hypertransport Core

To our knowlege, the only paper about an HT implementation on an FPGA is [23]. SystemC code of an HT bridge has been synthesized to a Xilinx Virtex II Pro FPGA, yielding in a post synthesis HT link frequency of 81 MHz. The following place and route steps, which usually lead to a significant decrease of the clock frequency, have not been performed. The core could not be used or even verified in any system, as the minimum HT clock frequency is 200 MHz. The HyperTransport Lite Core [21] from Xilinx is a very rudimentary implementation of HyperTransport for FPGA, supporting 8bit links with 400MHz link clock. It is not compliant to the HyperTransport specification. Just one package type is supported, the other packages are just ignored. The core is targeted towards Xilinx Virtex-II FPGAs, which do not have the serializer blocks that have been used to implemented our core. They solve the problem by sampling the incoming signals with a number of phase-sifted, slower clocks.

Several companies have announced HT400 FPGA solutions: Celoxica [11] has a HTX based solution, whereas XtremeData's XD1000 [12] and DRC's RPU-100 [10] connect to the socket 970. However, there is no or only few information about the HT cores used in these products. Another product, Pathscale's InfiniPath [20], which supports HT400 with 16bit wide links, is implemented as an ASIC.

4. THE HYPERTRANSPORT CORE

The HT core, as depicted in Figure 3, has two different interfaces: one are the HT send and receive links, as specified in the HT protocol, which can be seen on the left. On the other side, there is the application interface, which allows FPGA designs to access the HT core. This interface consists of three queues in each direction, one for every virtual channel. Applications can access these using a valid-stop synchronization mechanism. Control and attached data packets can be delivered simultaniously over the 160bit wide interface (96bit to handle extendet control packets, 64bit for data packets).

Data on an HT link is transmitted at a high clock rate and needs the exact alignment with the according clock. Therefore, every 8 bit of the CAD lines on the link have their own clock signal, which has to be used to sample the CAD lines. As a result, there are two digital clock manager (DCM) necessary for 16 bit wide links. However, these link clocks cannot be used for the internal core, since clock stability is not guaranteed for all the times it is needed, in particular during system startup. Therefore, the core clock is generated from the HTX reference clock, which is a 200Mhz clock that is stable at all times after power up. In this core the number of clock domains has been kept at a minimum, thus it needs two clock domains with a link width of 8 bits, and three domains with a link width of 16 bit.

The HyperTransport-core consists of 10 major submodules, which are briefly described in the following. At the interface to the HTX-connector, there are *deserializers* at the incoming link and the respective *serializers* at the outgoing link. Both work with a parallelization degree of 4, which means that at HT400, the data coming in with 800MT/s can be processed within the core with a clock frequency of 200MHz. An internal clock frequency of 800MHz is currently impossible in FPGAs. As a result, it is the best way to use special purpose SERDES FPGA blocks for this, as logic in the normal FPGA fabric could hardly do the serialization with this speed. Behind the deserializers, the data path has a width of 32bit for each of the 8bit CAD words. Closely coupled to the deserializers is the *bitslip* module. During initilisation, an alignment of the 32bit words coming out of the deserializers to the 32 bit packet boundaries has to be performed. This is provided by the inclusion of the bitslip module within in the device. It detects the present offset to the boundary and controls the bitslip input of the serializers, which shift the data stream bitwise.

Still within the link clock domains, the CRCs are checked. In HT, CRCs are periodically sent on each link. They contain the checksum of all packets on the bus within the respective timing window. CRCs are calculated seperately for every 8bit group in the CAD bus. With links that are 8bit wide or larger, the timing window is 512 bittimes. The CRC has to be inserted exactly 64 bittimes after the end of the window. It is not a packet, but only a plain 32bit checksum, so it can only be identified by sampling it at the right time. The CRC check module performs the check and removes the CRC from the data stream to the following FIFO buffer, which is used to synchronize the stream into the core clock domain. It is important to remove the CRC before the data stream enters the FIFO, as this allows the sender to transmit with a clock up to be 1/512 bittimes faster than the clock on the receiver side without pruducing an overflow in the FIFO.

The *decode* module distinguishes the different HyperTransport packets. When NOP packets arrive, credit information has to be forwarded to the credit module. Accesses to the HT registers are identified by an address range check and are forwarded to the init/config module. Other packets destined to the address range of the device are put into the adequate virtual channel queue. Packets that are not destined to the device are answered with a master abort. Broadcast, flushes and fences can be safely ignored within the core. However, credits have to be released when these packets are received.

On the packet interface from the HT core to the application, HT ordering requirements are currently satisfied by priorizing posted packets. This means, that nonposted and response packets are only marked valid for the application if the posted queue is empty. This is the most efficient solution in terms of hardware complexity. Nevertheless, a more sophisiticated implementation is currently beeing planned.

Packets from the application are placed in the incoming buffer. The *outputgenerator* has the task to arbitrate between the three virtual channels and any other packets that have to be sent by the core. It performs priority arbitration to ensure that packets to be sent immediately are not delayed. CRC packets have the highest priority and must be sent at a fixed time slot. They cannot be displaced without a link failure. The second highest priority is held by both the responses for configuration cycles and the responses with master abort error. Buffer space for these packets is limited and not flow controlled, so a loss of data could result if they are excessively delayed. The packets received from the virtual channel buffers have third highest priority. Among the different virtual channels, posted packets have the highest priority to fulfill HT ordering requirements in this direction. If one of these packets is in process, indicated by a set block signal, then it may not be disturbed by packets other than the CRC packets. NOP packets have the lowest priority. In order to avoid starvation of the sender on the other side of the link, the *credit* module can also force to send a NOP packet after the current packet. This is done when the remote sender has to few of the free local credits. The credit module also keeps track of the remote credits, i.e. the remote buffer status, to let the outputgenerator know on which VC packets may be sent.

The *init/config* module is used for the low level initialization, sending the correct initialization sequence at the beginning. Additionally, the HT configuration registers are located here. From an architectural perspective, it would be better to place these registers behind the HT core, together with the application's register space. This would simplify the arbiters within the core very much. We decided to implement them within the core as this solution is much easier to integrate together with application designs. Low lovel initialization basically has to set up the link after power up and every time a link width or frequency chance should take effect.

5. RESULTS

The Virtex-4 FX 60 FPGA [7] that has been used for implementation has high-speed in- and outputs that are called SelectIO. In combination with dedicated OSERDES and ISERDES de/serializers they support data rates up to 1GT/s in the FPGA. The DCMs of the FPGA support the generation of clocks up to 500MHz. As a result, these components are the limiting factor regarding the link frequency. They limit it to HT500, which has a 500MHz clock and data rates of 1GT/s. Within the FPGA, higher link frequencies could be handled with a higher parallization degree instead of scaling the frequency up. As the Opteron processors do not support HT500, we support only HT400 in the implementation. The successor of the Virtex4, the Virtex5, should allow HT600 given the information in the data sheets. However, a verification of this assumption is out of the scope of this paper.

Generally, the key to success is not to use any high-speed signals directly, but to use only the parallelized data stream. The only exception here is the usage of pwrok (power ok), reset and CTL signals during the initialization phase. The HT protocol defines that they hold their values a number of clock cycles before they change their values again, so that it is safe to sample them directly, even with a very low frequency.

The largest part of the core has been implemented in the Verilog hardware description language, and thus can be mapped to any device, in particular it could easily be mapped to an ASIC or to FPGAs from other vendors. To reach the goal of a low-latency design with high link frequencies while at the same time holding FPGA resource utilization as small as possible, build-in FPGA

blocks have been used whenever possible. Besides the serializers and deserializers that have been mentioned above, DCMs have been used for clock generation purpuses, and embedded RAM modules have been used to implement queues and FIFOs. As a result, the ressource requirements of the core, as depicted in Table 2, are relatively moderate. In particular, more than 80% of the logic slices can be used to implement applications. If the core is implemented to support a link width of 8bit only, the ressource utilization is significantly lower. This is due to the fact that the datapaths in the core can be reduced to a width of 32bit, compared to 64bit that are required for 16bit wide links. As the HT core is still beeing optimized and improved, these numbers may change for future versions of the core.

Table 2: Resource requirements in a Virtex-4 FX 60 FPGA

Ressource	8 bit link		16 bit link	
Logic Slices	2,699	10%	4,123	16%
FIFO16/ RAMB16s	30	12%	30	12%
DCM_ADVs	3	25%	4	33%
ISERDESs	10	1%	19	2%
OSERDES	9	2%	17	2%

The hardware latency of the core is very low, as Table 3 shows. The output path has a latency of 7 cycles. The input path of the core has a higher latency of 10 clock cycles. This difference is only caused by the synchronization FIFO buffer. The Xilinx FIFO core alone used in our design consumes 5 of the 11 total clock cycles. The respective bidirectional bandwidths with a 16bit link for HT200, HT400 and HT500 are 1,6GB/s, 3,2GB/s and 4GB/s.

Table 3: Hardware Latencies of the HT core

Direction	Clock Cycles	Delay@ HT200	Delay@ HT400	Delay@ HT500
In	11	55ns	27.5ns	22ns
Out	7	35ns	17.5ns	14ns

Much more important than the pure hardware latency is the access latency and bandwidth between software and device. This is a vast area of exploration, as a high number of system parameters has an influence on the performance. As we just started to analyze and optimize the performance in detail, we can just give very first results here. Also, for stability reasons, measurements so far have been performed using the slowest HT core configuration with 200MHz and 8 bit wide links only. Measurements have been performend in a system with the Iwill DK8-HTX mainboard and two Opteron 246 processors.

We measured the latency of processor read accesses to the device. These accesses were issued by a primitive device driver under Linux 2.6.18. There have been problems to obtain reliable results when using the processor clock cycle counter, therefore latency was measured on the FPGA device by measuring the time that passes between two subsequent read request. This results in a meaningful figure, as we observed that read requests are only issued when the previous request has been answered by the device. However, it may be that this measurment will not return the software-device latency, but rather the host bridge-device latency. The measured latency is 30 HT core clock cycles, which corresponds to 300ns. For read accesses, we observed 32 bit read accesses only, and were not able to provoke larger reads. This of course implies that larger processor reads result in a fairly high latency, as e.g. an 128 bit read is executed using 4 individual 32 bit reads, resulting in a latency of 1200ns. The bandwidth for reads is thus quite low: 100Mb/s. Further investigations will have to show wether there is a way to improve read performance, and how these values compare to other I/O buses like PCIe. In any case, this is an issue of the Opteron system or it's configuration, not of the HT protocol or the HT core.

Writes, in contrast, may have a size of up to 64bytes when write-combining memory is used. Also, one write may follow the previous without a gap. As a result, we measured a write bandwidth of 2.8Gb/s, which is close to the theoretical maximum of the link of 3.2Gb/s.

6. VERIFICATION

The HT core has been verified by simulation using an HT bus functional model from AMD. Figure 4 shows a screenshot of the simulation of the low level initialization. To ensure the correct operation in a running system, it has been mapped to a newly developed FPGA HTX-Board [19]. It contains an HTX connector and a Virtex-4 FX 60 FPGA which can be programmed via JTAG or USB. The HTX-Board can be plugged into any motherboard providing an HTX slot, in our case this is the Iwill DK8-HTX dual processor Hypertransport-enabled server board. To be able to monitor the transmissions on the link, an HTX extender board is used. This board enables the user to connect direct probes of a logic state analyzer, which is a very efficient method for low-level debugging. Figure 5 shows the test setup with extender and FPGA boards. At the front the HTX-Board is shown. It is plugged into the extender board, which has direct probes of an Agilent logic state analyzer connected to it. The extender board is again plugged into the HTX slot of the Iwill board.

The process of verification can be distinguished into three different phases. In the first phase, the HT core has been verified by simulation both in the bus model environment and with low level testbenches. Here, the correctness of the basic behaviour of the core has been verified. This included the initialization sequence, CRC insertion, basic packet handling and credit generation. Also, some corner cases have been simulated. However, no long-running testbenches have been run in this phase. We observed that a simulation of the HT core with the bus functional model is in the order of 10,000 times slower than the FPGA system. Thus, longer running verification has all been performed in-system.

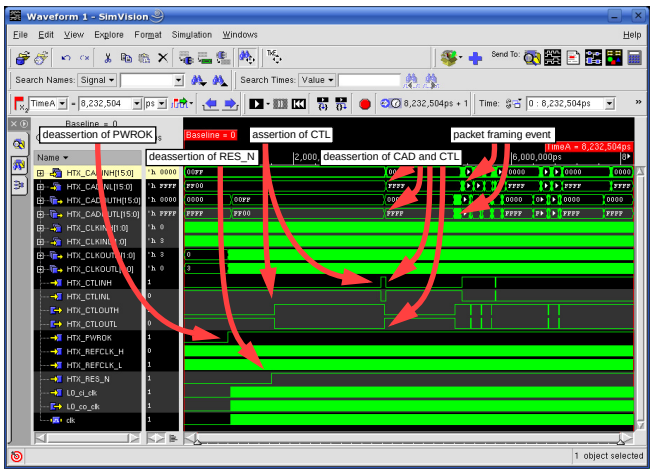

Figure 4:Low-level initialization in the simulator

In the second phase, the core has been mapped to the FPGA and verified in the system. Aim of this phase was to verify the low-level initialization sequence and the configuration phase, in which the BIOS configures the device using PCI compatible configuration cycles. During this phase, the biggest problem turned out to be the Iwill's BIOS. We observed that it does not seem to follow the HT specification strictly. Also, it was a problem that we did not have the BIOS source code, and thus had to work without the knowledge how exactly the BIOS initilizes the HT links and devices. To resolve this problem, we ported LinuxBIOS [8] to this mainboard and added HTX support for it. Without our comprehensive test environment, debugging LinuxBIOS and HT core at the same time would have been extremely difficult.

After the system was successfully booting Linux on top of LinuxBIOS, the third verification phase could start: the verification with user applications. We first implemented a simple register file which could be written and read by a basic HTX board Linux kernel driver, which has been developed simultaniously. The driver development has been very straightforward, as HT is PCI compatible on the software layers. In consequence of this, HT devices do not differ from PCI or PCI Express devices from the software perspective. As a more sophisticated user application, send- and receive units as used in NICs have been used. All problems and bugs that appeared during in-system verification have been re-enacted in the simulation environment. There, the problem has been traced, analyzed, fixed and verified by simulation before testing it in-system again.

Summarizing, it can be said that the strategy of very early in-system verification clearly proved itself. One reason for this is the increased speed of verification, and thus in the end a higher coverage and the earlier finding of problems. But the main benefit is the early elimination of uncertainties that are always present when the implementation can only follow a specification and a very generic functional model. These uncertainties can express themselves by a violation or different interpretation of the specification by other parties. But there is also the uncertainty of how frequent specific events occur. For example, we did not expect to receive a high number of FLUSH and FENCE broadcasts during the initialization phase, as well as hundreds of read and write packets that were not destined to our device. This observation influenced the way how these packets are treated within the core.

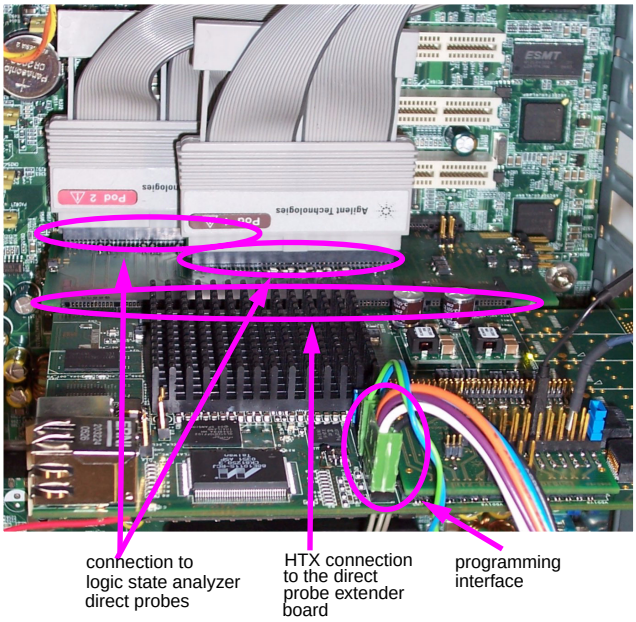

Figure 5:In-system verification setup

7. CONCLUSION AND FUTURE WORK

The work presented in this paper is an efficient implementation of an HyperTransport 2.0b link. It successfully exploits the potential of the used FPGA in terms of bandwidth, latency and resource utilization. Offering an HT400 connection, the HT core can, combined with an HTX FPGA board, be used for more than just prototyping. The performance is sufficiently good to serve as a production coprocessor board as well. Compared to the link speed between Opterons in a multiprocessor environment, which currently support HT1000, our HT link provides less than half the bandwidth. However, the link to our device is free of coherency traffic and remote memory accesses from other processors, so that not the same bandwidth is required as on the links between processors. The link clock frequency scales with the maximum clock speed of the FPGAs. Thus, faster links will be possible with higher speed grades of the same FPGA, or with faster FPGA families, as the Virtex5 family for example. The HT core can also be mapped to an ASIC implementation by replacing FPGA specifics as DCMs, SERDES and RAM blocks with the corresponding ASIC macros.

The HT core is still being optimized, in particular a more efficient enforcement of ordering between virtual channels is being worked on. Another important task is to analyze the performance of the HT core within the Opteron sytem in much more detail, and to tweak the sytem to get best performance.

The HT link presented here is also the foundation for future work in this area by our group. Currently, an HT crossbar switch is under development, which allows to connect multiple functional units or devices within the FPGA to the HT core. It may also be an option to connect several HT cores with this switch, thus creating an HT-to-HT bridge device. With support from AMD, other development efforts concentrate to add support for the cache-coherent HT protocol to the core, which will add the possibility to construct cache coherent devices within the FPGA. We will also continue to maintain the core, which is freely available for download [13].

8. ACKNOWLEDGEMENTS

Special thanks go to Jay Owens, Rich Oehler, Michael Goddard, Dan Mudgett and Doug O'Flaherty from AMD for their support, and in particular for the financial aid which allows us to continue to maintain the HT core.

This project has been strongly supported by the highly engaged team of the Computer Architecture Group and their students at the University of Mannheim.

9. REFERENCES

[1] Hypertransport Technology Consortium, *Hypertransport I/O Link Specification Revision 2.00b*, Document #HTC20031217-0036-0009, 2005

[2] Hypertransport Technology Consortium, *Hypertransport I/O Link Specification Revision 3.00*, Document #HTC20051222-0046-0008, 2006

[3] Hypertransport Consortium, *HyperTransport EATX Motherboard/Daughtercard Specification*, www.hypertransport.org, 2004

[4] Hypertransport Consortium, *The Future of High Performance Computing: Direct Low Latency Peripheral-to-CPU Connections*, www.hypertransport.org, November 2005.

[5] Duncan Bees, Brian Holden, *HyperTransport reduces delays in some applications*, EETimes 2004

[6] Advanced Micro Devices, *AMD BIOS and Kernel Developer's Guide for the AMD Athlon 64 and AMD Opteron Processors*, #26094, Rev 3.3, 2006

[7] Xilinx Corporation, *Virtex-4 User Guide*, Document ug070 v1.5, www.xilinx.com, 2006

[8] LinuxBIOS, http://www.linuxbios.org

[9] The HyperTransport Consortium, http://www.hypertransport.org/

[10] *DRC RPU100-L60 Datasheet*, www.drccomputer.com/pdfs/DRC_RPU100_datasheet.pdf

[11] *Celoxica RCHTX-XV4 Datasheet*, http://www.celoxica.com/techlib/files/CEL-W06112119BY-517.pdf

[12] *XtremeData XD1000 Product Brief*, http://www.xtremedatainc.com/pdf/XD1000_Brief.pdf

[13] Center of Excellence on research in HyperTransport technology. Website: http://www.ra.informatik.uni-mannheim.de/coeht/

[14] Kai Hwang, Zhiwei Xu, *Scalable Parallel Computing*, 1st Edition, McGraw-Hill, 1998.

[15] Noboru Tanabe, Junji Yamamoto, Hiroaki Nishi, Tomohiro Kudoh, Yoshihiro Hamada, Hironori Nakajo, Hideharu Amano, *MEMOnet : Network interface plugged into a memory slot*, IEEE International Conference on Cluster Computing , p. 17, 2000.

[16] Holger Fröning, Mondrian Nüssle, David Slogsnat, Patrick R. Haspel, Ulrich Brüning, *Performance Evaluation of the ATOLL Interconnect*, IASTED Conference: Parallel and Distributed Computing and Networks (PDCN) , Feb. 15 - 17, 2005, Innsbruck, Austria

[17] Jumnit Hong, Eriko Nurvitadhi, Shih-Lien L. Lu, *Design, implementation, and verification of active cache emulator (ACE)* , Proceedings of the internation symposium on Field programmable gate arrays, Monterey, California, USA, 63-72, 2006

[18] Taeweon Suh, Hsien-Hsin S. Lee, Shih-Lien Lu, John Shen, *Initial Observations of Hardware/Software Co-Simulation using FPGA in Architecture Research*, 2nd Workshop on Architecture Research using FPGA Platforms, Austin, 2006

[19] ,Holger Fröning, Mondrian Nüssle, David Slogsnat, Heiner Litz, Ulrich Brüning, *The HTX-Board: A Rapid Prototyping Station*, 3rd annual FPGAWorld Conference, Nov. 16, 2006, Stockholm, Sweden

[20] Dickman, L. Lindahl, G. Olson, D. Rubin, J. Broughton, J. , *Pathscale InfiniPath: a first look*, 13th Symposium on High Performance Interconnects, 2005. Proceedings.

[21] HyperTransport Lite Interface for Virtex-II FPGAs; XILINX; Document # 1-800-255-7778; 31.03.2004, http://www.xilinx.com

[22] Ian Kuon, Jonathan Rose, *Measuring the gap between FPGAs and ASICs*, Proceedings of the internation symposium on Field programmable gate arrays table of contents Monterey, California, USA, 2006

[23] Castonguay, A., Savaria, Y, *A Hypertransport Chip-to-Chip Interconnect Tunnel Developed Using SystemC*. 16th International Workshop on Rapid System Prototyping, Proceedings. Shortening the Path From Specification to Prototype, p. 264-266, 2005