# ETHERBONE - A NETWORK LAYER FOR THE WISHBONE SoC BUS

M. Kreider, W. Terpstra, GSI Helmholtz Centre for Heavy Ion Research, Darmstadt, Germany
J. Lewis, J. Serrano, T. Włostowski, CERN, Geneva, Switzerland

## *Abstract*

Today, there are several System on a Chip (SoC) bus systems. Typically, these buses are confined on-chip and rely on higher level components to communicate with the outside world. Taking these systems a step further, we see the possibility of extending the reach of the SoC bus to remote FPGAs or processors. This leads to the idea of the EtherBone (EB) core, which connects a Wishbone (WB) Ver. 4 Bus via a Gigabit Ethernet based network link to remote peripheral devices.

EB acts as a transparent interconnect module towards attached WB Bus devices. Address information and data from one or more WB bus cycles is preceded with a descriptive header and encapsulated in a UDP/IP packet. Because of this standard compliance, EB is able to traverse Wide Area Networks and is therefore not bound to a geographic location.

Due to the low level nature of the WB bus, EB provides a sound basis for remote hardware tools like a JTAG debugger, In-System-Programmer (ISP), boundary scan interface or logic analyser module. EB was developed in the scope of the WhiteRabbit Timing Project [1] (WR) at CERN and GSI/FAIR, which employs GigaBit Ethernet technology to communicate with memory mapped slave devices. WR will make use of EB as means to issue commands to its timing nodes and control connected accelerator hardware.

## PURPOSE

EB is a network protocol meant for fast, low level software to hardware or hardware to hardware communication. It connects two distant WB SoC buses and is capable of direct memory access to attached devices. EB shall be used in GSI/FAIR and CERNs timing nodes as well as for remote debugging and programming, making hard to reach embedded systems easier to deploy and maintain.

## RELATED WORK

There are many different examples of protocols for direct data exchange available. Among the most commonly used low-level were Myrinet in the supercomputing sector (almost completely replaced now by Ethernet based equipment) and different Remote Direct Memory Access (RDMA) [2] implementations. While there are pure software implementations of RDMA, their latencies cannot compete with hardware implementations like Infiniband [3] or iWARP [4], which can can achieve latencies below 7$\mu$s. However, these are mostly optimised for maximising throughput, while latency is still a secondary factor. There are also high level protocols available like CORBA [5] and SOAP [6], which aim for abstract software to software communication in heterogeneous environments. While being very versatile, due to their higher logistics overhead and generic nature, they are not well suited for fast communication between hardware and hardware or hardware - software. All of the above have in common that they are not tied to a specific underlying bus protocol of their endpoints. While they of course keep data content, they will not preserve syntax during transport.

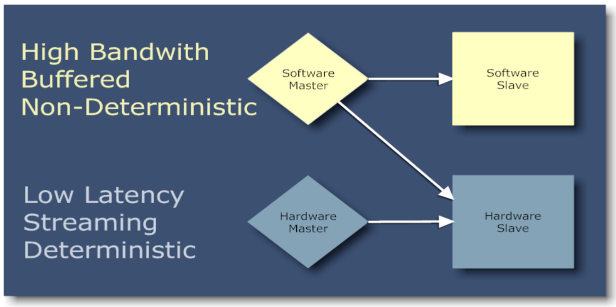

Figure 1: Compatibility between EB node types.

## ARCHITECTURE

### *General Considerations*

Since bus protocols can differ quite strongly in their workings and packet layout, conversion between them can severely reduce fidelity. For EB, we therefore chose Wishbone V4 as a concrete bus implementation, while leaving the underlying transport protocol open. There are two categories of EB devices under development: Buffered, non deterministic software modules and low-latency, deterministic streaming hardware cores (Fig. 1).

Software nodes are used for all applications where determinism and latency are not the main issue, but interoperability and fidelity of bus signals are. One example would be a developer's computer, remotely connected to the JTAG module of an embedded system elsewhere on the premises. Figure 2 shows an example block diagram of such a setup.

Hardware nodes operate in full streaming mode, They are fully deterministic and made to cut latency down to the minimum. An application example would be an endpoint of a timing system, receiving commands to generate a pulse at a specific execution time. The deterministic characteristics of EB ensure that available time frame for delivery does not vary, while streaming provides low latencies, reducing the time the control system needs to buffer. Hardware implementations are of course not as flexible as software.

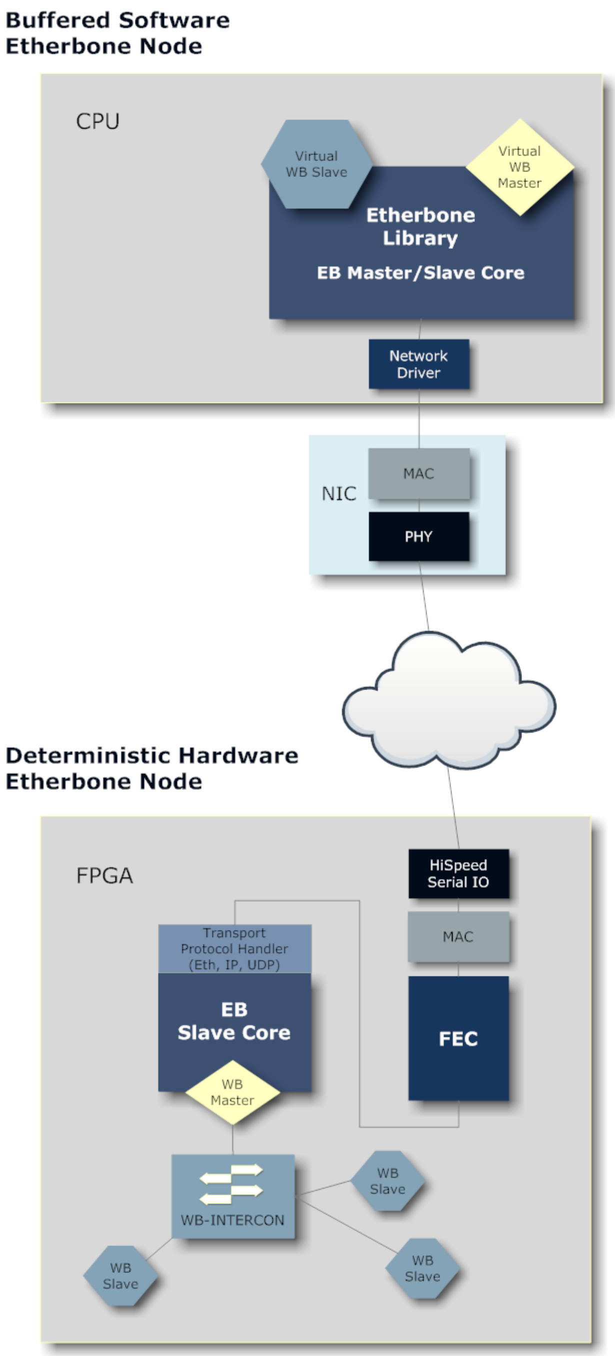

Figure 2: EB Slave node.

### Methods and Test Implementation

Our current SW/HW test implementation utilises UDP/IP as a transport protocol. As a requirement, EB needs duplicate free transmission, which UDP alone cannot guarantee. It is therefore assisted by a Forward Error Correction (FEC) Scheme on OSI layer II. In order to be fully deterministic and to achieve lowest latency possible, EB needed to be fully streaming capable, ideally introducing no additional delay to passing data. A hard timeout is enforced when waiting for replies to EB requests, depending on the given time frame. The main problem is in the IP and UDP packet headers, which contain length information and checksums on the payload. On the Ethernet layer for example, the Cyclic Redundancy Check (CRC) follows the payload, so on the fly processing or insertion is not a problem. This is not the case with UDP/IP, where prior knowledge about the payload is necessary. For low latency streaming, we had to resolve the dependencies between packet header and payload. An example for a concrete implementation can be found in Figure 3, showing the internal layout of the Hardware Description Language (HDL) of a deterministic EB slave node.

Like the underlying WB bus, EB has master and slave nodes [7], which form complementary pairs. This leads to a bridge architecture, where an EB master accepts bus operations from local WB masters for transfer to a remote node. EB slaves therefore have a WB master interface and form the remote representation of the local WB master.

## TRANSPORT PROTOCOL

The EB protocol has been designed to be deterministic with a focus on minimal latency. It also needed to be able to use standard transport layer architectures. Since WR utilises Ethernet technology, making EB interoparable with GigaBit Ethernet standard was an obvious choice in the development. Raw Ethernet frames however were not an option, because they cannot pass routers and firewalls without special configuration. So a widely supported protocol with very low overhead was needed, and the choice fell to UDP [8].

### Packet Length

The UDP/IP Header requires packet length fields before the payload. To avoid waiting for packet completion, streaming EB replies are the same length as the corresponding request, making the full header knowable in advance. EB counters every incoming read operation by an outgoing write operation, while incoming Writes are answered with zero padding. Without any need for exceptions to the rules, these are treated as empty EB records, the result is similar to a no-operation instruction in a CPU.

### Checksums

The IP checksum is only dependent on fields of the IP header. This includes source and destination address, IP packet options and the packet length field. When replying to a request, almost all information required can be taken from the incoming packet header, except for length fields, source IP address, IP checksum and UDP port. Source address and port are already known to the node, which leaves the payload length and the checksum itself [9]. Due to the symmetry we introduced, length is also known in advance and the IP checksum of the reply can be already be calculated after header reception. With this, all reply header information is available at the beginning of the incoming

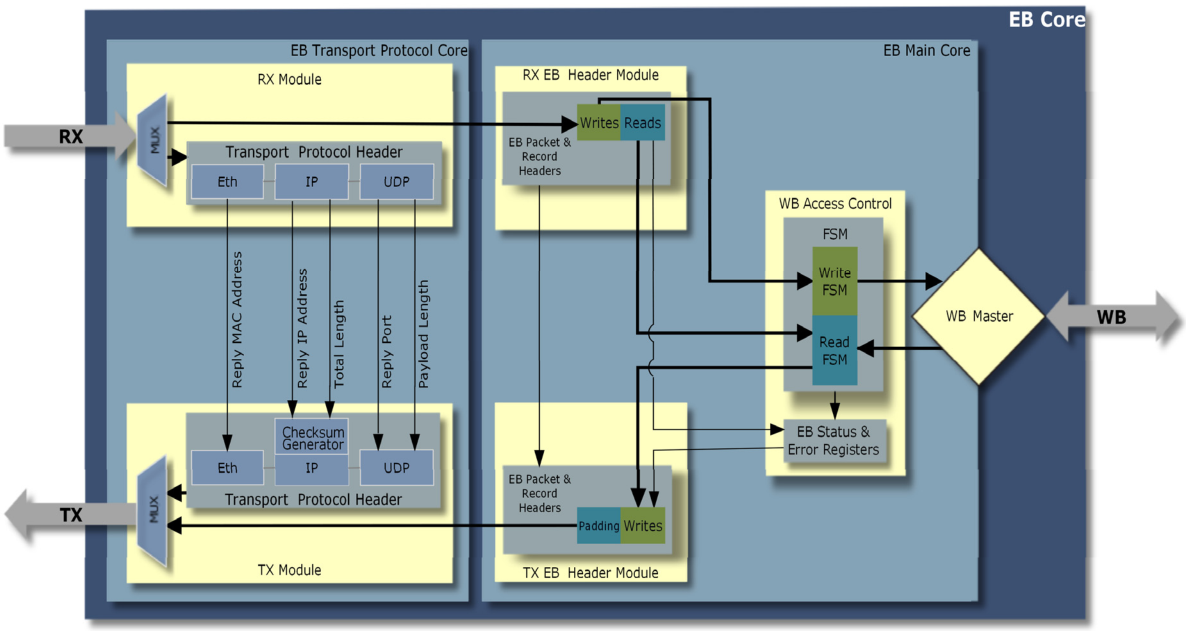

Figure 3: EB Slave node.

payload. This eliminates all wait times for payload reception, trading bandwidth for latency. The UDP Checksum is purely dependent on payload, but UDP protocol specifications allow the checksum to be set to zero. This signals the recipient “not used” [8]. Since we use a more powerful FEC in addition to the CRC, the UDP checksum can be omitted without risking data integrity.

## ETHERBONE STRUCTURES

### *Packet Layout*

A full EB Datagram is shown in figure 4. It consists of a header block and one or more record headers with matching Write or Read operations.

The header block starts with a magic word and two bit masks, used to signal the used WB bus width and address size. There is also the option to set a probe flag, which is used for negotiation of usable bus and address widths between two devices. A Probe-packet is always padded to the maximum alignment, 64 bit in our test case to ensure compatibility.

A record header contains a set of flag bits, stating optional information about source and destination, like the use of FIFO mode, etc. The flags are followed by the number of write and read operations in the record, this can be a number between zero and 255 if no feedback is necessary (assuming sufficient free space in the packet). If the bus is wider than the record header, it will be padded.

After the record header follow bus operations, Writes first, then Reads. Bus Operations are not mandatory, an EB record can therefore be empty, contain Writes, Reads, or both. Each of these blocks is preceded by an address field. For a Write, this field signifies the target start address on the slave; for a Read it is the address to which the read values shall be written to on the master.

### *Communication*

**Negotiation** A typical EB connection starts by sending a probe packet to a slave. It solely consists of the header block with all possible bus and address sizes the master supports and a set probe flag. The slave then sends back the intersection of the offered modes and the one it supports itself. The result shows all possible bus and address widths the EB master can choose from for communication with this particular slave device, completing the negotiation. In the next step, a normal EB packet is sent, containing one or more EB records. The slave will reply with the bus width and address size chosen by the master in the request header.

**Atomics** EB supports atomic WB bus operations. While the cycle line is held, so is the connection to the target slave through a WB interconnect. Each Etherbone record comes with the option of ending the current bus cycle on completion or keeping it to the next record. With this mechanism, bus ownership can be held over several EB records, avoiding interference to the operation by other bus devices.

**Symmetry** Equal packet length of in- and outgoing traffic is essential for EB streaming mode. In order to keep equal length between request and reply, results from Reads are converted to write operations while Writes are turned

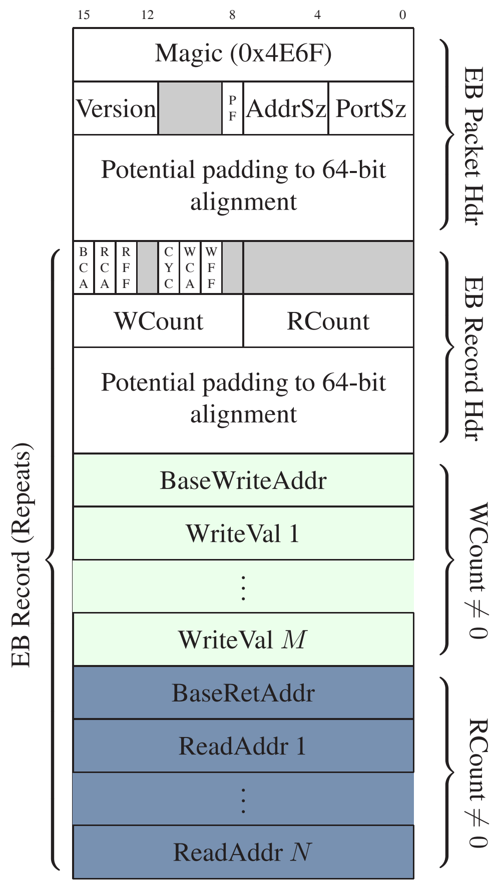

Figure 4: Etherbone v0.2 Message Format.

into empty Records, i.e. padding. This makes the length field of UDP/IP headers known in advance and resolves all header/payload dependencies.

**Addressing** EB Write operations are Values to be written, while Reads are addresses to be read. Write addresses therefore need to be generated by an increment to the start address. This increment can either be zero, in which case the target is treated as a FIFO, or match the byte alignment master and slave agreed on. This implies Writes to be sequential, while Reads can be random access. A WB master must keep track of read addresses in order to correctly interprete the answering Write.

**Management** EB also supports the use of a configuration space, an address space that is not associated with the local WB bus and only concerns the EB node itself. Here, settings like the MAC address or status information like the error log are kept. If the flag is set, all following operations will not be put on the bus but will point to the config space instead. If feedback for success of operations is required, a shift register in the config space can be read to supply the error bit for the last 32 operations on the WB bus interface.

## CONCLUSION

Time driven HDL simulation and minor tests on evaluation boards for GigaBit Ethernet links show possible latencies below 1 $\mu$s. Further tests on final WR hardware later this year have yet to confirm these values, but intermediate results show EB as a well suited candidate for control systems and remote tools alike.

## OUTLOOK

With open EB hardware and software implementations becoming available as open source projects, chances for it becoming a widely accepted WAN remote bus protocol within the timing and control systems community are increasing. Our next task will be full integration with GSI and CERN's next generation timing system. For GSI, this will happen on the example of a proton linear accelerator, the first component to be deployed in GSI's FAIR extension. Next steps will be the completion of a remote toolbox, containing an ISP and Debugger for use with our network capable embedded systems.

## REFERENCES

[1] P. Moreira, J. Serrano, T. Wlostowski, P. Loschmidt, G. Gaderer, "White Rabbit: Sub-Nanosecond Timing Distribution over Ethernet", IEEE Precision Clock Synchronization for Measurement, Control and Communication 2009, pp1-5, Brescia, October 2009

[2] A. Romanow, and S. Bailey, An Overview of RDMA over IP, Proceedings of the First International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet 2003), Feburary 2003

[3] J. Liu, J. Wu, D. K. Panda, "High Performance RDMA Based MPI Implementation over Infiniband", International Journal on parallel programming, Vol. 32, No. 3, pp167-198, 2003

[4] M. J. Rashti, A. Afsahi, "10-Gigabit iWarp Ethernet: Comparative Performance Analysis with Infiniband and Myrinet-10G", IEEE International Parallel and Distributed Processing Symposium, pp.290, 2007

[5] A. S. Gokhale, D. C. Schmidt, "Measuring and Optimizing CORBA Latency and Scalability over High-speed Networks", IEEE Transaction on Computers, Vol. 47, No. 4, 1998

[6] Y. Ying, Y. Huang, and D. Walker, "A performance evaluation of using SOAP with attachments for e-Science", Proceedings of UK eScience All Hands Meeting, pp. 796-803, 2005

[7] Opencores, "Wishbone B4 WISHBONE System-on-Chip (SoC)Interconnection Architecture for Portable IP Cores" (Standard), 2010 `http://cdn.opencores.org/downloads/wbspec_b4.pdf`, last visited 20.09.2011

[8] J. Postel, "User Datagram Protocol", RFC 768 (Standard), Internet Engineering Task Force, August 1980

[9] N. Alachiotis, S. A. Berger, and A. Stamatakis, Efficient PC-FPGA communication over Gigabit Ethernet, Proceedings of the International Conferences on Embedded Software and Systems (ICESS 10), pp. 1727-1734, Bradford, UK, 2010