Open borders for system-on-a-chip buses: A wire format for connecting large physics controls

M. Kreider,[1,2] R. Bär,[1] D. Beck,[1] W. Terpstra,[1] J. Davies,[2] V. Grout,[2] J. Lewis,[3] J. Serrano,[3] and T. Wlostowski[3]
[1]*GSI Helmholtz Centre for Heavy Ion Research, Darmstadt, Germany*
[2]*Glyndŵr University, Wrexham, United Kingdom*
[3]*CERN, Geneva, Switzerland*
(Received 27 January 2012; published 23 August 2012)

System-on-a-chip (SoC) bus systems are typically confined on-chip and rely on higher level components to communicate with the outside world. The idea behind the EtherBone (EB) protocol is to extend the reach of the SoC bus to remote field-programmable gate arrays or processors. The EtherBone core implementation 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. EB was developed in the scope of the WhiteRabbit Timing Project at CERN and GSI/FAIR. WhiteRabbit will make use of EB as a means to issue commands to its timing nodes and control connected accelerator hardware.

DOI: 10.1103/PhysRevSTAB.15.082801 PACS numbers: 84.40.Ua, 29.20.−c, 07.05.−t

I. PURPOSE AND ENVIRONMENT

This article builds on the paper by the title "EtherBone—A network Layer for the Wishbone SoC Bus" in the ICALEPCS 2011 conference proceedings and aims to provide details on design choices and performance analysis for implementation. EtherBone (EB) functionality has been successfully demonstrated at the 2012 WhiteRabbit workshop.

EtherBone is a fast, low-level network protocol layer intended for either software to hardware or hardware to hardware communication. It connects a client to a distant Wishbone (WB) system-on-a-Chip (SoC) bus and is capable of direct memory access to attached devices. This article builds on the paper by the same title in the ICALEPCS 2011 conference proceedings. EB will be used in the timing nodes at the GSI/FAIR and CERN accelerator facilities.

At this point, a description of the intended environments for the deployment of EB is in order. In synchrotron machines, the particle beam can only be kept on its desired path by a tight interplay between a vast number of smaller parts. There are beam guide components, like magnets and radio frequency units, diagnostic equipment, and large sensor arrays at the targets. All of these machines need accurately timed commands in the form of synchronous triggers, timestamps, and control signals in order to play their part at exactly the right time. When GSI is expanded with the planned Facility for Antiproton and Ion Research (FAIR), this means the coordination of more than 2000 timing end points. WhiteRabbit [1] (WR), the system chosen for time synchronization, is based on Ethernet technology and locally employs WB based systems hosted inside a field-programmable gate array (FPGA). EtherBone was named after these underlying technologies, Ethernet and Wishbone. However, EB resides in the Open Systems Interconnection session layer (OSI layer 5) and does not depend on a specific choice of lower layer protocols in implementation.

II. REQUIREMENTS

A. Timing challenge

Control of the accelerator's machines is time critical, varying from several n sec, over few μ sec , up to thousands of milliseconds. When compared with a delivery time of approximately 100 μsecs for an EB packet, this leads to several conclusions: (i) all actions need to be known in advance; (ii) all actions must be precisely timed; (iii) there is no time for acknowledgment or retransmission; (iv) a deterministic, low-latency command and parameter distribution system is necessary.

For the efficient running of the accelerator it is clearly necessary that the timing system must produce timing signals that are deterministic, i.e., the time at which an action is carried out must be precisely known and reproducible. To facilitate this, it is necessary that there is minimum delay in the path from the production of the timing signal to its use.

B. Deterministic command distribution

EB was designed to have very low latency and high determinism, giving secondary consideration to throughput. Because of this specialization, EB application focuses on commands rather than the transport of raw data. Separation between the data and the commands working with this data is possible in most cases. This means machines are fed with data via a standard network infrastructure and also receive EB packets over the timing network,

containing a command to carry out a preset action and a time of execution. EB is also able to address all specialized control blocks inside the WB hardware description language (HDL) designs directly, a fact that is very useful for this implementation. For example, consider remotely programming a function generator with a parameter set for output level, gate length, and a sequence of trigger pulses plus a time stamp to execute all this. The function generator would be a WB memory mapped device, so EB will provide direct and easy access to all its registers and functions.

C. Compatibility and expandability

Particle accelerator facilities like GSI and CERN host large heterogeneous pools of equipment, which have evolved for more than 40, or in CERN's case almost 60, years. Most equipment at the Large Hadron Collider (LHC) and on the FAIR project is quite new, but there are still many legacy systems to cover. FPGA based technology enables all sorts of adapters and converter logic with little or no extra hardware effort, while Ethernet based infrastructure makes the distances in growing facilities easily covered. This shall also be reflected by the next generation control systems.

III. FURTHER APPLICATIONS

There are many places on site where well-known hardware tools like debug modules, in-system programming adapters, logic analyzers, and similar are needed to deploy, maintain, and upgrade hardware. Because of the distances between nodes, which are about 2 km at FAIR and 10 km at CERN, and the quantity of nodes involved, routing was another requirement of EtherBone. It will not only reduce the time it takes engineers to travel on site, but also makes automated testing easier. Personnel can also collaborate more easily, since access to the hardware tools can be shared over the network. Last but not least, it is possible that the electronics are not accessible during beam time and for some time afterwards due to radiation concerns. All in all, this feature will reduce time requirements for maintenance and deployment.

IV. RELATED WORK

There are many different examples of available protocols for direct data exchange. 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) [2,3] implementations. While there are pure software implementations of RDMA, their latencies cannot compete with hardware implementations like Infiniband [4] or iWARP [5], which can achieve latencies below 7 μ sec. However, these are mostly optimized for maximizing throughput, while short message latency is a secondary factor. PCI Express falls into the same category and will be addressed later in more detail.

There are also high level protocols available like CORBA [6] and SOAP [7], 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 software to hardware or hardware to hardware. All of the above have in common that they are not tied to a specific underlying bus protocol at their end points. While they keep data content, they will not preserve syntax during transport.

A comparison to widely used field buses, like USB, PCI, and PCI Express (PCIe) showed PCIe as most fitting for this scenario but it still does not quite achieve the requirements, mainly because of difficulties associated with routing over WANs. It has nevertheless many features that are desirable for EB [8].

Like the GbE Interface, EB uses a 125 MHz clock rate. While the network end point uses an 8 Bit interface, the Wishbone interface connected to EB is 32 Bit wide, giving it 4 times the bandwidth. Regardless of delays for processing the packet structure, the difference in bandwidth ensures EtherBone to be fully streaming capable. EB has several design traits in common with PCI Express. Both are serial field bus protocols, they feature error detection in the form of a cyclic redundancy check (CRC) to ensure packet integrity, carry routing information, and provide quality of service (QoS). Also, both protocols go all the way down to the physical layer. PCIe features autodiscovery of bus devices, which is also present in WB and (and therefore EB) since March 2012 under the name self-describing wishbone [9].

However, there are also differences. While PCIe is packet-based from the bottom up, EB's underlying WB bus is cycle based. PCIe was also designed for higher throughput than WB, by bundling several lanes into one connection, and is meant to run at higher frequencies. This said, there are also differences to EB on the upper layer. The first lies in the routing capabilities. While PCIe can be switched much like the MAC layer of Ethernet, it cannot do complex routing and most importantly is not native to Wide Area Networks (WANs). This makes long distance connections over common network architectures impossible. PCIe could of course be encapsulated in IP packets to do just that, but given the similarity between PCIe and Ethernet/IP packet headers, this would be almost completely redundant and therefore double the overhead. PCIe is more powerful than WB, but it also has its downsides. The controllers are vastly more complex, most of this is due to backward compatibility for PCI. It is much harder to implement in an FPGA than WB and therefore takes up many times more resources. Also, while PCIe controller chips are readily available, high quality HDL cores

providing similar functionality are all commercial and closed source. Superior routing capabilities, easy connection to HDL blocks, the possibilities for expansion, and the focus on latency over throughput make EB a worthwhile project to investigate. To conclude the evaluation, all differences are based on the trade-off between flexibility and extensibility on the one hand versus latency and overhead requirements.

V. ARCHITECTURE

General considerations

Since bus protocols can differ greatly in their operation and packet layout, conversion between them can severely reduce fidelity. For EB, therefore Wishbone B4 has been chosen as the bus implementation, while leaving the underlying transport protocol open. There are two categories of EB devices under development: buffered, nondeterministic software modules and low-latency, deterministic streaming hardware cores (see 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. Examples would be a developer's remote computer, running a serial console on one of the timing end points or debugging software via JTAG module on an embedded system elsewhere on the site.

Figure 2 shows an example block diagram of such a setup. The top box contains the EB library running on a PC, while the lower is an FPGA board hosting an EB slave. It is attached to multiple HDL blocks via a WB interconnect.

The EtherBone software library provides a generic interface for the driver, and it is not visible to the application how the EB device is connected. Most packet-based protocol could be applied to carry EB information, so an EB device locally connected via USB would behave exactly the same as if it were connected remotely over Ethernet. Hardware nodes operate in full streaming mode, they are fully deterministic and designed to minimize latency. An application example for a hardware node would be an end point of a timing system, receiving

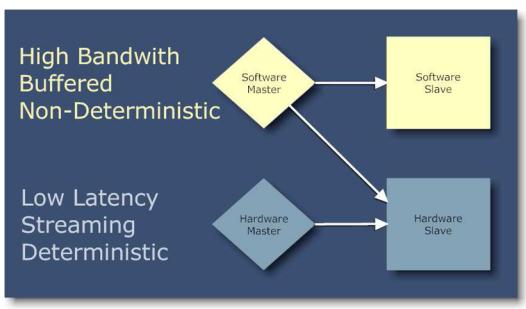

FIG. 1. Compatibility between EB node types [18].

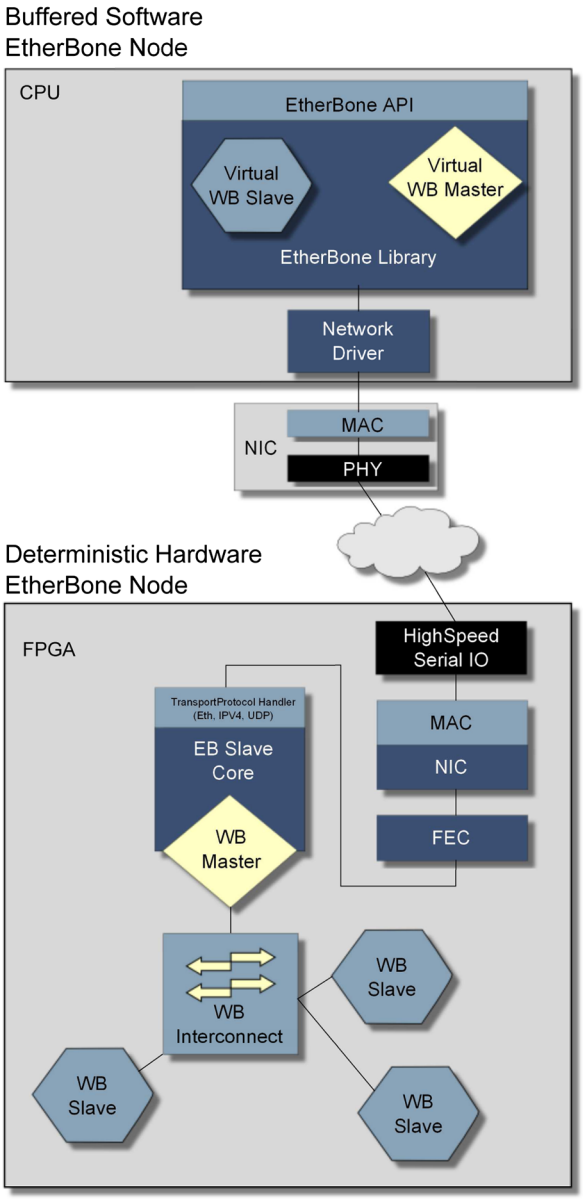

FIG. 2. Software EB master (e.g. developer's computer) connecting to a remote FPGA based hardware EB slave [18].

commands to generate a pulse at a specific execution time. The deterministic characteristics of EB ensure that the available time frame for delivery does not vary. Streaming provides low latencies, reducing the reaction time the control system has to an event occurring elsewhere on the timing network. Hardware implementations are of course not as flexible as software. Our streaming hardware slave implementation uses an HDL block as a deterministic EB node. On the network interface side, it features streaming WB channels to GbE block, on the SoC side, it has a WB master which will usually be connected to an interconnect. The RX and TX cores are directly linked in order to already prepare the TX reply header while the incoming header is processed.

VI. ETHERBONE DESIGN CHOICES

A. Underlying transport protocols

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. So a widely supported protocol with very low overhead was needed and the decision to use IPv4 at the network layer and User Datagram Protocol (UDP) at the transport layer was made [10].

1. Ethernet

Since WR utilizes 802.3 Gigabit Ethernet technology, making EB interoperable 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. Delays are reduced by the use of QoS functionality for command messages to ensure that they are not held up behind other frames. For lowest latencies, WhiteRabbit switches support cut-through mode, a packet is not stored and forwarded but directly passed through. Neither cut-trough nor link aggregation are a standard technique available in off-the-shelf network switches, but this is not an issue in our scenario since WR already employs custom switches.

2. IP

The choice for IP was based on the simple fact that it is the most widespread general purpose protocol. In order to send EB datagrams over WAN, there is no alternative supported by an off-the-shelf network equipment.

3. UDP

When comparing the structure of UDP (Fig. 3) to the more powerful TCP shown in Fig. 4, the main difference lies in UDP being a stateless protocol. It dispenses with a handshake, sequence numbers, and acknowledgments in favor of simplicity and low overhead. Contrary to UDP, TCP is capable of retransmitting of lost packets [11]. Unfortunately, this is not an option for the control system scenario, since there is not enough time for retransmission. These properties make UDP a better fit for the goals of EtherBone.

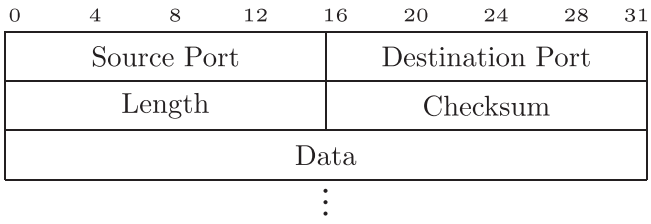

FIG. 3. UDP header. The simplistic structure goes well with the EB low overhead and latency goals.

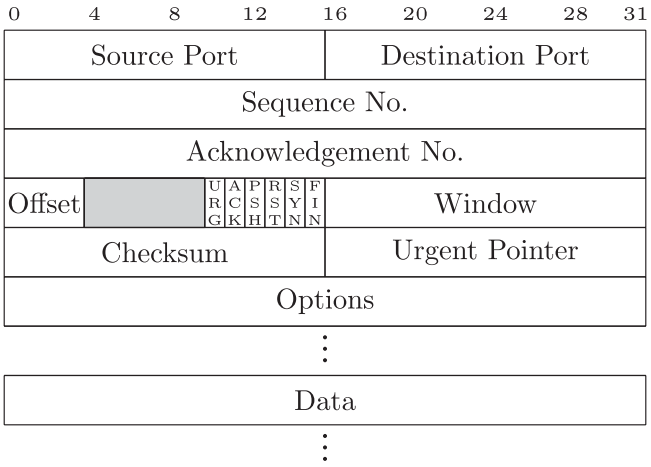

FIG. 4. TCP header. TCP is more powerful than UDP, but most of its features are not useful to a timing system. It also causes more overhead.

B. EtherBone protocol

EtherBone itself is designed for low overhead and latency as well. The packet header consists of a flag block, containing information on master and slave WB bus type, protocol version and a mechanism for negotiating a common bus mode. It is followed by records containing bus operations. Each record will add a header and one or two base address fields to the overhead. In our implementation, this adds 8 or 12 Bytes overhead per record. Packet header and record format have a smaller footprint than those of RDMA, the records are similar to PCIe.

The implementation also uses a 125 MHz bus clock for WB. At 32 Bit per cycle, WB has 4 times the data rate of the GbE interface. This makes packet processing without additional delay possible, the only limit stems from the reply time of targeted WB devices.

VII. METHODS AND TEST IMPLEMENTATION

The current hardware/software test implementation utilizes UDP/IP protocol. As a requirement, EB needs duplicate free transmission, which UDP alone cannot guarantee. It will therefore be assisted by a forward error correction 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 the passing data. Waiting for replies to EB requests may block the EB master and its network interface for an unknown time, which is unacceptable. In order to ensure determinism, a hard timeout is enforced at this point. If an EB slave has to wait too long for a connected WB device to answer, a buffer underrun will occur, corrupting the streaming reply.

The main problem is that the IP and UDP packet headers contain length information and checksums on the payload.

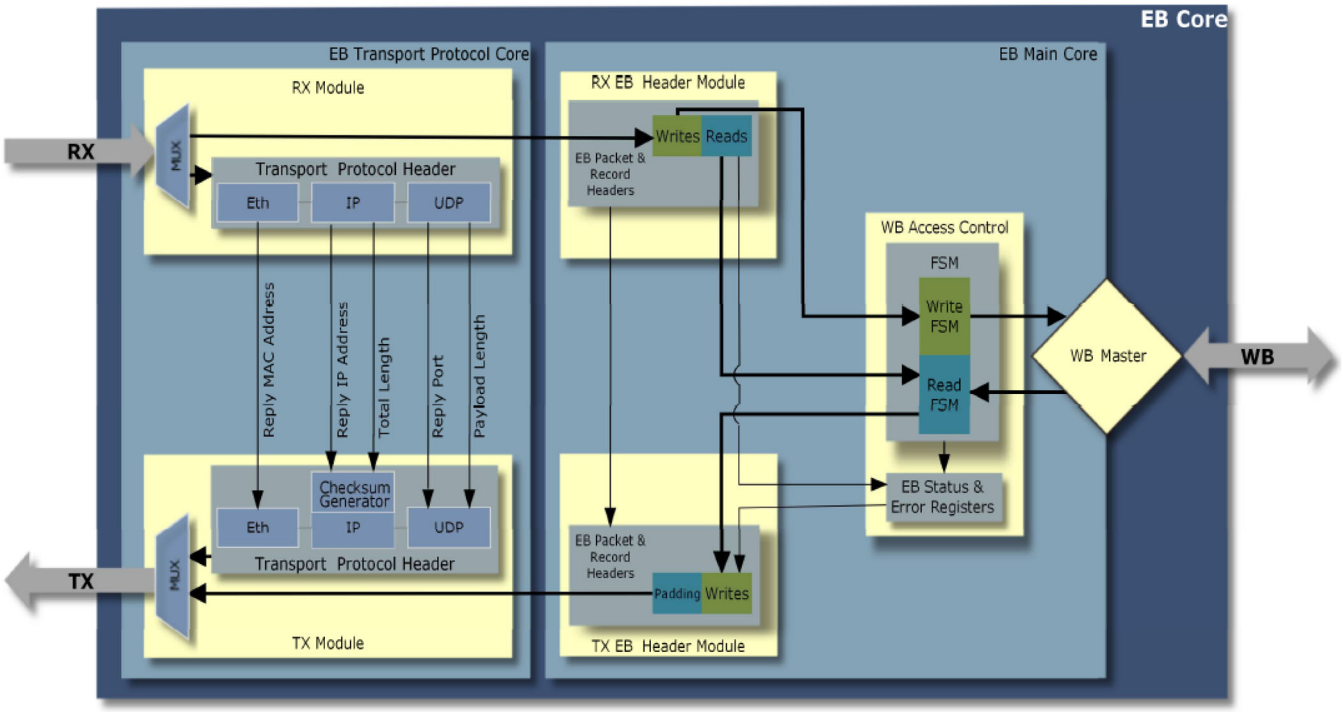

FIG. 5. EB streaming hardware slave. Network streaming interface on the left, on the right the IF to the WB interconnect. To speed up header processing, RX and TX cores are directly connected [18].

However, at the Ethernet layer the CRC field follows the payload so it is possible to insert information on the fly and allow the Ethernet CRC field to be recalculated hence solving the problem. This is not the case with either the UDP or IP fields, where prior knowledge about the payload is necessary. For low-latency streaming, it is necessary to resolve the dependencies between packet header and payload.

Like the underlying WB bus [12], EB has master and slave nodes, 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. A sample hardware implementation is shown in Fig. 5. EB slaves therefore have a WB master interface and form the remote representation of the local WB master.

A. Packet length

The UDP/IP header requires the packet length fields before the payload [10,13]. To avoid waiting for the complete packet to be received, streaming EB replies are the same length as the corresponding request, allowing the full header to be determined in advance. EB responds to every incoming read operation with an outgoing write operation, while incoming writes are answered with zero padding. There are no exceptions to the rules since these are treated as empty EB records. The result is similar to a no-operation instruction in a CPU.

Figures 6 and 7 show the structure of an outgoing UDP/IP header. The outgoing header does not need prior

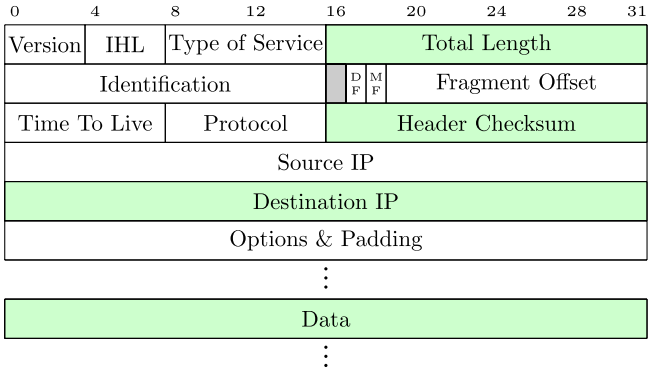

FIG. 6. IP header. All fields causing dependencies for the packet header of the reply are marked in green.

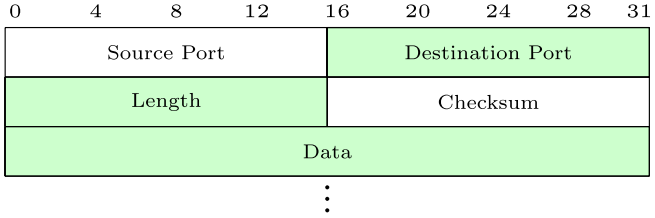

FIG. 7. UDP header. All fields causing dependencies for the packet header of the reply are marked in green.

knowledge of the contents of the payload in the incoming packet. All header fields can either be taken from the header of the incoming request (reply address, port, length) or deduced from it (IP checksum) or use data already locally available (source address and port). All processed fields are colored in light green.

B. Checksums

The Ethernet frame checksum field follows the frame's payload, so it can always be calculated on the fly. This is not the case with either the UDP or IP fields, where prior knowledge about the payload is necessary. Figure 8 gives a brief overview of the areas protected by different checksums in a UDP/IP packet. When replying to a request, almost all information required can be taken from the incoming packet header, except source IP address, IP checksum, and source UDP port. The IP checksum is only dependent on fields of the IP header (see Fig. 6). This includes source and destination address, IP packet options, and the length field. Source address and port are already known to the node, which leaves the payload length, on which the checksum depends, and the checksum itself [14]. Because the IP checksum algorithm is basically a sum, all known fields can already be added in advance. This leads to a prefabricated checksum that can be kept for future reference. Since only the length and new destination information needs to be included, this process is very fast (see Fig. 5, TX block). Because of the introduced symmetry, the length is also known in advance and so the IP checksum of the reply can be calculated in advance directly after header reception.

The UDP checksum is calculated from a pseudoheader, which is not transmitted. It contains the source IP, destination IP, protocol (see Fig. 6), the UDP length field, and the following data (see Fig. 7). The UDP checksum therefore depends on the payload, but UDP protocol specifications allow the checksum to be set to zero. This signals the recipient "not used" [10]. Since the more powerful forward error correction is used in addition to the CRC, the UDP checksum can be omitted without risking data integrity.

With this, all reply header information is available at the beginning of the incoming payload. This eliminates all wait times for payload reception, trading bandwidth for latency. In effect, the slave will already start sending the reply at the moment that the request header is processed.

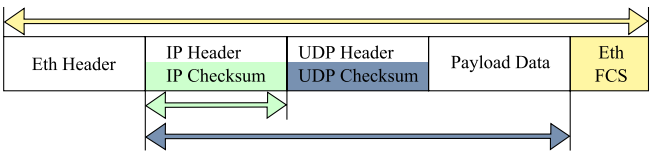

FIG. 8. Checksum coverage in a UDP/IP packet.

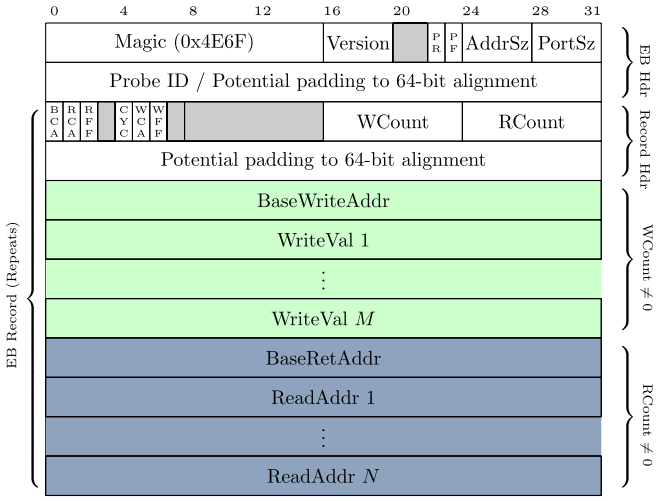

FIG. 9. EB packet structure, featuring an 8B header followed by EB records containing WB bus operations.

VIII. EB DATA FORMAT

A full EB datagram is shown in Fig. 9. It consists of a header block and one or more record headers with matching write or read operations. 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 bits in this application 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. 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 should be written to on the master.

A. COMMUNICATION

1. Negotiation

The developer has a free choice in both bus width and address width of a WB bus to best suit the requirements and can choose to also support smaller configurations. This leads to the question which mode is supported by both a master and the targeted slave device.

Initially, the master sends a probe packet to the slave, consisting solely of an EB header without a payload. This header has a set probe flag and shows all modes supported by the master, followed by probe ID. This ID is necessary in case the destination IP of the sender does not match the source IP of the recipient. The slave then sets the probe

response flag and answers with all modes it supports, followed by the received probe ID code. The EB master then knows about all possible bus and address widths it can choose from for communication with this particular slave device, completing the negotiation.

The next step is to send a normal EB packet 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. In the FAIR and CERN control systems, the most common bus width will be 32 bits.

2. Atomics

EB supports atomic WB bus operations. By holding the cycle line on the WB interconnect, the connection to the target slave is kept, allowing the next record to be transferred without another slave being able to use the bus. Each EB record comes with the option of ending the current bus cycle on completion or keeping it for the next record. With this mechanism, bus ownership can be held over several EB records, avoiding interference with the operation by other bus devices.

3. Status Information

To determine the success of any EB operation, the status register in the EB configuration space can be read at the end of a stream of bus operations. It shows the flags for all previous operations, either ACK or ERROR.

4. Symmetry

Command messages mostly go from a master to a slave, but there are also cases where the master urgently needs to read information from a slave device. Equal packet length of ingoing and outgoing traffic is essential for EB streaming mode, because it significantly reduces round-trip time. In order to keep equal length between request and reply, results from reads are converted to write operations while writes are turned into empty records, i.e., padding. This makes the length field of UDP/IP headers known in advance and by that removes all wait times stemming from header/payload dependencies.

5. Addressing

EB write operations are values to be written, while reads are addresses to be read. Write addresses therefore need to be generated by incrementing the start address, reads can be random access. The increment can either be zero, in which case the target is treated as a FIFO, or sequential. An EB master must keep track of read addresses it sent in order to correctly interpret the answering write.

6. 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. It can both be accessed via EB and a local WB slave interface used for configuration of the EB node. This configuration space has several purposes. For one, the EB node's own MAC and IP address are set via one of its interfaces and kept here. The configuration space is also used to map incoming EB packets to the originating queries. Last but not least, it contains a feedback register of operations on the bus. If feedback for success of operations is required, this shift register can be read to supply the ACK or ERR bit for the last 64 operations on the WB bus interface.

7. Security

In the development of the EtherBone protocol, the decision about the OSI transport-, internet- and link-layer protocol has been intentionally left open to provide maximum flexibility. Likewise, EtherBone does not contain any specific authentication or encryption mechanism. If access control and cryptography are required, it can instead be wrapped in any proven secure protocol of choice (e.g. L2TP, IPsec, TLS/SSL). EtherBone itself is always assumed to communicate over a trusted channel, just like the WB SoC bus.

In a timing context at GSI/FAIR and CERN, EtherBone will run inside a separate timing LAN with special switch hardware. Since deterministic communication in the microsecond range is not possible over WANs, all necessary connections from the outside world into the timing network can be assumed to have loose timing constraints. They will therefore be handled by gateway computers connected both to the campus and the timing network. These computers will host a secure socket layer (SSL) connection to the campus network and will tunnel all the client's EtherBone traffic. While not in the scope of EB security, it would be easy to provide barriers against accidental misconfiguration or misuse at the WB and network level.

IX. PERFORMANCE ANALYSIS

GbE uses an 8b/10b channel encoding in order to encode the clock into the data stream and make the data stream DC free. This results in 20% extra overhead. Whenever we talk about overhead or latency, we refer not to the raw bit stream, but to the higher layer data rates. The results remain comparable because the main contestants, RDMA and PCIe ($<$ v3.0), also employ 8b/10b.

A. Overhead details

1. EtherBone

Overhead in EB contains the network header, meaning all protocol headers below EB (802.3 GbE, IP, UDP). Inside EB, overhead consists of EB record headers, base write addresses, read back addresses, and read addresses. The amount of overhead in an EB packet depends on the type, the order of WB bus operations, and their addressing. EtherBone supports block and random access operations in a single network packet. Minimum packet overhead in our

current implementation with 802.3 Ethernet, IPV4, UDP, and EB is $18 + 20 + 8 + 8 + 8 = 62$ Bytes. Efficiency is calculated as payload over packet data (payload + overhead). Assuming a maximum packet size of 1500 Bytes, the best case efficiency for EB is in block write operations $1432/(66 + 1432) \cdot 100 = 95.6\%$, the worst comes with random access writes $(480/(1010 + 480) \cdot 100 = 32.2\%$. Block reads do not differ in format from random access reads, all read addresses must be provided. This takes up a great deal more bandwidth than block writes. Because of streaming, however, block read operations have no impact on latency.

2. RDMA

In case of RDMA, packet overhead is somewhat bigger in general. Let us have a closer look at sample read request and write operations of captured iWarp RDMA packets. With 802.3 Ethernet, IP, TCP, iWarp, DDP, and RDMA, it has more protocol layers than EB. The overhead of all the packet headers add up to $18 + 20 + 32 + 6 + 14 + 1 = 91$ Bytes [3].

3. PCIe

PCI Express is the most lightweight of the three protocols in terms of overhead. Physical layer, DLLP, and TLP add up to a minimum overhead of $2 + 6 + 16 = 24$ Bytes [8]. The small footprint is mostly due to the lack of routing capabilities, though PCIe still carries many legacy features meant to keep compatibility to PCI.

B. Overhead comparison

Figure 10 shows a comparison between the overhead ratios for block and random access operations of EB, PCIe, RDMA. It becomes clear that EB fills a specialist role for control applications when it comes to bandwidth efficiency. For block writes of 256 Bytes, EB can well compete the other protocols. When it comes to block reads, EB has the worst efficiency of the three, because all addresses are contained inside the packet. However, due to the streaming functionality, this does not have an impact on latency. The more random access operations are present, the better EB fares in comparison. When handling many small bus operations, EB beats RDMA as well as PCIe, because it can transport all of them in the same network frame. Infiniband RDMA is of course vastly superior to EB in terms of bandwidth, as is PCIe, if more lanes were added. The outcome fits well with EB's intended role as a slim protocol for control applications. For FAIR's planned control system, the high efficiency for block writes also comes in handy for occasional distribution of set values.

FIG. 10. Overhead of EB, PCIe, and RDMA when transmitting 256 Bytes read and write operations, 32 Bit word width. EtherBone distinctly excels in random access operations and block writes.

C. Latency

EB was designed to introduce very low latency. In our implementation, the WB data rate is 4 times that of the GbE network, therefore all processing can be handled while gathering the next data word. Ideally, EB therefore does not cause any additional latency. The introduced delay when processing or answering an EB packet equals the time it takes to receive and process the packet headers, determining if a valid EB packet was received and if a reply was requested.

However, there is a threshold to the latency caused by targeted WB slaves above which delays will accumulate. While pipelined operations are supported, a bus cycle is not complete before all acknowledgments and data blocks have been received by the WB master. The EB core has to wait for all answers at the end of a WB cycle. If the maximum latency of the ACKs on the WB bus is never greater than 3 (4 cycles per word on IF side—1 cycle on EB/WB side) cycles, EB is guaranteed not to need prebuffering. Since the TX header is complete in the buffer by the time the first WB operations take place, there is still a reserve of 42 cycles if individual operations should take longer to complete.

The measurements for EB were taken using the WhiteRabbit time stamping unit, normally used for a special version of precision time protocol. Figures for all other systems were taken from literature. We define the round-trip latency as first byte sent to the first byte of the reply arriving at the sender, connected end to end, in our case over a 10 m fiber cable. This shows EtherBone to be well for short message latency with 1.2 μs round-trip time. The prototype hardware implementation still encompassed packet buffers that could not be removed in this version due to their importance for the WR timing framework. Without this buffering, simulations show EB latency to be around 550 ns for our setup and will also be independent of message size again. This result fits in well with the 1 μs estimates made by Rumble *et al.* [15].

Direct quantitative comparison with other systems is difficult when the latency measurements have to be taken from third parties. In most papers, the test beds are not completely described and the understanding of latency is not exactly defined. For example, when talking about round-trip latency, it is very important at which OSI layer the turnaround takes place. On top of this, the underlying hardware differs widely. Bearing this in mind, we will stick to a qualitative view on latency.

The lowest figures found for RDMA over Infiniband found comes from a performance benchmark found online [16] and is stated as <1 μs. Since it is given as an end to end figure, it would have to be doubled to ≈ 2 μs.

The results for PCIe [17] were taken from a comparison to the Hypertransport protocol. The read latency of an 8 Byte packet came down to 232 ns from first byte sent to the first byte of the reply packet. Considering the PCIe bandwidth of 2.5 Gb/s and the bus controller running at 333 MHz, EB did not fare badly in comparison. Furthermore, there is still room for improvement in EB down to ≈ 550 ns when the WR store-and-forward buffers are removed. In the context of these figures, EB results can be said as being on the edge of the technically feasible and being a good fit for a timing/control system application.

X. EB FOR ACCELERATOR TIMING

The main purpose of the timing system is to provide two basic functions. The first is time tagging input/output (IO) or internal events. Timestamp latch units in the end points provide and buffer timestamps for all events they are programmed to listen for. The end point's buffer access is handled over EB. Their programming can be done over EB as well.

The second is sending timing messages that will trigger preprogrammed actions at a given time. EB can be used for both, the carrying messages as well as doing the preprogramming. The difference between a message and any other EB content lies only in the targeted WB slave and data content, not in format. Messages go to an event-condition-action (ECA) unit, which controls the end point's IO cores. A timing message contains an absolute time of execution and ID(s) of actions to be executed. It must be sent sufficiently in advance to arrive before the time of execution stated in the message is due. Low latency in EB and switching will help reduce transmission time. The architecture foreseen for the FAIR facility will have a master unit on top of a basically treelike network architecture. Command messages will be broadcast, making the difference in time it takes them to arrive at all the end points smaller than using unicast. This also means that contention for downward packets can never occur, since the bandwidth is determined by the bottleneck. Quality of service (QoS) and cut through mode make command message delivery faster and more deterministic. To detect relevant message content, ECAs feature preprogrammed message filters, receiving and possibly logging all messages but reacting only if the instructions concern their own end points.

XI. CONCLUSION

In March 2012, interplay between EtherBone HDL macrocores and software API has been successfully shown at the 6th WhiteRabbit Timing Workshop. Faithful transmission of bus operations and timely message delivery were demonstrated, as well as scanning the device chain on a remote WB bus. In addition, basic functionality for the GSI/FAIR's and CERN's timing system was shown in the form of timestamping pulses from a signal generator and producing a pulse on the end points IO based on the read timestamp plus a delay.

The desired small protocol footprint has been achieved. The measured round-trip latency at 1.2 μs was above the calculated values, but the difference stemmed from packet buffering in this prototype which is indispensable for the operation of the WhiteRabbit clock synchronization cores for now. Without the buffering time, the latency will be even lower. This performance proves EB to be a worthwhile protocol for the use in future timing and control systems. We hope to see further applications in the large physics community in the near future.

XII. FUTURE WORK

Immediate goals are the implementation of an EB hardware master and first tests of an SSL-EB gateway. The next task will be full integration with GSI and CERN's next generation timing system. For GSI, this will happen early in 2013 on the example of a proton linear accelerator, the first component to be deployed in GSI's FAIR extension. The next steps will be the completion of a remote toolbox, containing a firmware loader, in-system programming adapter, and debugger for use with our network-capable embedded systems.

[1] P. Moreira, J. Serrano, T. Wlostowski, P. Loschmidt, and G. Gaderer, *White Rabbit: Sub-nanosecond Timing Distribution Over Ethernet* (IEEE, Brescia, Italy,2009), pp. 1–5.

[2] A. Romanow and S. Bailey, in Proceedings of the First International Workshop on Protocols for Fast Long-Distance Networks PFLDnet (Geneva, Switzerland, 2003).

[3] An RDMA Protocol Specification, Technical Report (RDMA Consortium, 2003).

[4] J. Liu, J. Wu, and D. K. Panda, Int. J. Parallel Programming **32**, 167 (2004).

[5] M. J. Rashti and A. Afsahi, in *Proceedings of the International Parallel and Distributed Processing Symposium* (IEEE, Long Beach, 2007).

[6] A. S. Gokhale and D. C. Schmidt, IEEE Trans. Comput. **47**, 391 (1998).

[7] Ying Ying, Yan Huang, and David W. Walker, in Proceedings of the UK e-Science All Hands Meeting (Nottingham, UK, 2005), pp. 796–803.
[8] PCI Express Base Specification Revision 3.0, Technical Report (PCI-SIG, 2010).
[9] FPGA Configuration Space, Technical Report (Open Hardware Repository, 2011).
[10] J. B. Postel, User Datagram Protocol, RFC 768 (Internet Engineering Task Force, 1980).
[11] J. B. Postel, Transmission Control Protocol, RFC 793 (Internet Engineering Task Force, 1981).
[12] Wishbone B4 WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores, Technical Report (OpenCores, 2010).
[13] J. B. Postel, Internet Protocol, RFC 791 (Internet Engineering Task Force, 1981).
[14] N. Alachiotis, S. A. Berger, and A. Stamatakis, in *Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology, CIT '10* (IEEE, Bradford, UK, 2010), pp. 1727–1734.
[15] S. Rumble, D. Ongaro, R. Stutsman, M. Rosenblum, and J. Ousterhout, in Proceedings of USENIX Workshop on Hot Topics in Operating Systems (HOTOS) (2011).
[16] Mellanox Technologies, "Infiniband performance," http://www.mellanox.com/content/pages.php?pg=performance_infiniband (2012).
[17] B. Holden, "Latency comparison between hypertransport and pci-express in communications systems," http://www.caoni.info/pdf/Latency/7.pdf (2006).
[18] M. Kreider, W. Terpstra, J. Lewis, J. Serrano, and T. Wlostowski, in *Proceedings of the International Conference on Accelerator and Large Experimental Physics Control Systems ICALEPCS* (JaCoW, Grenoble, France2011), pp. 642–645.