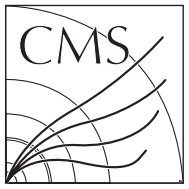
Available on CMS information server

CMS CR -2014/334



The Compact Muon Solenoid Experiment

Conference Report

Mailing address: CMS CERN, CH-1211 GENEVA 23, Switzerland

22 October 2014

IPbus A flexible Ethernet-based control system for xTCA hardware

Tom Williams for the CMS Collaboration

Abstract

The ATCA and uTCA standards include industry-standard data pathway technologies such as Gigabit Ethernet which can be used for control communication, but no specific hardware control protocol is defined. The IPbus suite of software and firmware implements a reliable high-performance control link for particle physics electronics, and has successfully replaced VME control in several large projects. In this paper, we outline the IPbus system architecture, and describe recent developments in the reliability, scalability and performance of IPbus systems, carried out in preparation for deployment of uTCA-based CMS upgrades before the LHC 2015 run. We also discuss plans for future development of the IPbus suite.SUMMARY IPbus will be used for controlling the uTCA electronics in the CMS HCAL, TCDS, Pixel and Level-1 trigger upgrades. IPbus control has already been extensively used in the work of these upgrade projects so far, and final uTCA systems will be deployed in the experiment starting from Autumn 2014. IPbus is also being evaluated for use in the ATLAS and ALICE upgrades, as well as other particle physics experiments. A tightly-integrated suite of software and firmware components has been developed to implement the IPbus protocol the firmware core, a reference VHDL implementation of an IPbus server over UDP, decoding IPbus read/write requests within end-user hardware; uHAL, the C++/Python library providing an end-user API for IPbus reads and writes; and the ControlHub, a software application which abitrates hardware access to each board from multiple clients. Over the past two years we have developed a new reliable, higher-throughput version of the IPbus protocol, firmware and software. We have set up an IPbus test system with realistic network topology in the CMS electronics integration centre, in order to validate the reliability and performance of the IPbus control system. The software has been optimised to increase the block write/read throughput towards the Gigabit Ethernet bandwidth, and to improve the scalability with the number of targets handled by each ControlHub instance. For 1 client and 1 target, the latency is about 250us for sequences of up to tens of transactions and the maximum block read/write throughput is 0.54Gbit/s; the throughput increases to 0.8Gbit/s for 3 or more targets. We have accumulated weeks of continuous high-throughput random writes and reads over IPbus, without any errors. We also investigated scenarios with network congestion in the MCH Ethernet switch for a full uTCA crate, and found that with appropriate configuration this congestion only has a small effect on the IPbus throughput (12pct reduction). Plans for future work include improving the monitoring of IPbus dataflows in large systems of hundreds of targets, and investigating further ideas for usability and performance improvements.

Presented at *TWEPP 2014 TWEPP 2014 - Topical Workshop on Electronics for Particle Physics*

IPbus: A flexible Ethernet-based control system for xTCA hardware

C. Ghabrous Larrea[a], K. Harder[b], D. Newbold[c], D. Sankey[b], A. Rose[d], A. Thea[b], and T. Williams[b*]

[a] *Dept. of Physics, University of Wisconsin-Madison, 1150 University Ave., Madison, WI 53706, U.S.A.*
[b] *Rutherford Appleton Laboratory, STFC, Harwell, OX11 0QX, U.K.*
[c] *H.H. Wills Physics Laboratory, University of Bristol, Tyndall Avenue, Bristol, BS8 1TL, U.K.*
[d] *Blackett Laboratory, Imperial College, Prince Consort Road, London, SW7 2BW, U.K.*

E-mail: t.willams@cern.ch

ABSTRACT: The ATCA and μTCA standards include industry-standard data pathway technologies such as Gigabit Ethernet which can be used for control communication, but no specific hardware control protocol is defined. The IPbus suite of software and firmware implements a reliable high-performance control link for particle physics electronics, and has successfully replaced VME control in several large projects. In this paper, we outline the IPbus control system architecture, and describe recent developments in the reliability, scalability and performance of IPbus systems, carried out in preparation for deployment of μTCA-based CMS upgrades before the LHC 2015 run. We also discuss plans for future development of the IPbus suite.

KEYWORDS: Control system; IPbus; μTCA; MicroTCA; ATCA.

*Corresponding author.

Contents

1. Introduction

New electronics systems within many particle physics experiments are based on the ATCA and μTCA standards (henceforth collectively referred to as xTCA). The xTCA specifications incorporate industry-standard serial communication technologies such as Gigabit Ethernet; however, unlike the VMEbus standard, they do not specify a hardware access protocol for controlling xTCA boards from external software applications.

Several important requirements must be considered when designing the architecture and implementation of a hardware control system. Control systems must have reliable and predictable behaviour under all conditions, since they form the main link by which hardware is configured, monitored, and debugged in case of problems. The control system architecture for large experiments should be highly scalable, ideally with the same ease of setup and use from the simple 'board on benchtop' scenario to the final system with hundreds of boards. In modern particle physics experiments, the same electronics setup is often used for decades before being replaced, and the associated control infrastructure must have the same maintainable lifetime. Hence, it is typically beneficial to use widespread industry-standard technologies, in order to avoid the risk of reliance on a single vendor. Experience from the CMS experiment's online systems in LHC Run 1 also shows that for monitoring and debugging issues in complex scenarios, in general it is helpful to move complexity away from hardware/firmware into software running on commercial PC hardware.

The IPbus protocol — first developed by J. Mans et al. in 2009 — is a simple control protocol for reading and modifying registers within IP-aware hardware. A tightly-integrated suite of IPbus software and firmware components which can be used to construct reliable, scalable, high-performance control systems has previously been presented in Ref. [1]. This IPbus suite will be used to control the xTCA off-detector electronics in the Phase-1 upgrades to the CMS experiment [2], as well as the ATLAS experiment's Phase-0 and Phase-1 upgrades [3]. In this paper, we present recent improvements in the reliability, scalability and performance of the IPbus suite, based on a new version of the protocol.

2. IPbus protocol

The IPbus protocol is a simple protocol for controlling IP-aware hardware devices which have a virtual A32/D32 bus. It defines the following operations:

Read A read of user-definable depth. Two types of read are defined: incrementing (for multiple continuous registers in the IPbus address space) and non-incrementing (for a port or FIFO).

Write A write of user-definable depth. As with reads, two types of write are defined: incrementing and non-incrementing.

Read-Modify-Write bits (RMWbits) An atomic bit-masked write, defined as $X := (X \,\&\, A) \,|\, B$. This allows one to efficiently set/clear a subset of bits within a 32-bit register.

Read-Modify-Write sum (RMWsum) An atomic increment operation, defined as $X := X + A$, which is useful for adding values to a register (or subtracting, using two's complement).

The IPbus protocol lies in the application layer of the networking model and is transport protocol agnostic. Each IPbus host device (typically hardware in a remote electronics crate) has an IP address and port number on which it accepts IPbus control packets. The protocol is transactional — for each read, write or RMW operation, the IPbus client (typically software) sends a request to the IPbus device; the device then sends back a response message containing an error code (equal to 0 for a successful transaction), followed by return data in case of reads. In order to minimise latency, multiple transactions can be concatenated into a single IPbus packet.

Version 2.0 of the IPbus protocol [4] (finalised in early 2013) includes a reliability mechanism, through which the IPbus client can correct for any packet loss, duplication or re-ordering, if using an unreliable transport such as UDP. This mechanism is based on the client setting sequential packet ID values. In systems with multiple control applications, IPbus traffic must be routed via a network element that understands the IPbus protocol and thus can buffer the incoming request packets and reset their IDs (in practice, this is the role of the ControlHub).

3. Firmware and software suite

The IPbus software and firmware suite consists of the following components:

IPbus firmware A module that implements the IPbus protocol within end-user hardware

Table 1. Resource usage of IPbus firmware core.

Resource	Usage	
	Minimal configuration	Fully-featured
Flip flops	2000	3500
Slices	1000	2900
Block RAMs	5	17

ControlHub Software application that mediates simultaneous hardware access from multiple μHAL clients, and implements the IPbus reliability mechanism over UDP

μHAL C++ and Python end-user programming interface for writes, reads and RMW operations

End-user instructions and source code for these components are available through the CERN CACTUS (Code Archive for CMS Trigger UpgradeS) website and SVN repository [5]. The software is packaged as RPMs for Scientific Linux versions 5 and 6, and available through a YUM repository.

3.1 IPbus firmware

The IPbus 2.0 firmware module is a reference system-on-a-chip implementation of an IPbus 2.0 UDP server in VHDL; it interprets IPbus transactions on an FPGA. It has been designed as a common module to run alongside a device's main processing logic (e.g. trigger algorithms) on the same FPGA, only using resources from within the FPGA. As a result of this, the IPbus firmware core must have a low resource usage, which is an important consideration in the choice of transport protocol. The TCP protocol exhibits various highly-desirable features of a transport protocol, such as reliable, ordered data transmission and congestion avoidance; however, the underlying algorithm is significantly more complex than for the other ubiquitous transport layer protocol, UDP. Hence, UDP has been chosen as the transport protocol; any loss, re-ordering or duplication of the IPbus UDP packets is automatically corrected by the ControlHub using the IPbus reliability mechanism.

The IPbus firmware module has been designed to be simple to integrate into variety of platforms, and there are example designs for several development boards and standard platforms. The source code is currently Xilinx-specific, but has been successfully adapted for Altera devices. In addition to UDP, the IPbus firmware module also implements: the echo request/reply semantics from ICMP (RFC 792, used in the Unix `ping` command); ARP (RFC 826, used for resolving IP addresses into MAC addresses); and RARP (RFC 903, used for requesting an IP address on startup). Several parameters are configurable at build time, including: the Ethernet frame MTU; the number of buffers for incoming/outgoing IPbus packets which determines the maximum possible control throughput; and the method used for IP address assignment — RARP, IPMI, or fixed IP address. The resource usage of the IPbus firmware core under 'minimal' and 'fully-featured' configurations is shown in table 1.

3.2 ControlHub

The ControlHub is a software application that forms a single point of access for IPbus control of each device; specifically, it arbitrates simultaneous access from multiple control applications to one or more devices, and it implements the IPbus reliability mechanism for the ControlHub–device

UDP packets. Since the ControlHub is a software application, the μHAL–ControlHub communication uses TCP, which has sophisticated congestion mitigation and flow-control algorithms.

Design requirements and implementation. The ControlHub must be at least as reliable and transparent as a VME crate controller, since failure or crash within the ControlHub could disrupt the communications of several upstream control or monitoring applications. Additionally its design must allow multiple clients to communicate with multiple targets reliably, efficiently and independently.

Erlang is a general-purpose, concurrent programming language, designed by Ericsson to build high-availability, fault-tolerant applications. The main structural unit in Erlang is the process. Erlang processes are lightweight compared to operating system processes; they share no state, instead communicating by message passing. These features are well-suited to the ControlHub's requirements for high reliability, performance, and scalability in routing IPbus transactions, and therefore the ControlHub is implemented in Erlang. The ControlHub uses a separate Erlang process for each connected μHAL client and each IPbus device, ensuring workload can be spread across multiple CPU cores; its internal structure is described in more detail in Ref. [1].

3.3 μHAL library

μHAL is the Hardware Access Library (HAL) providing an end-user C++/Python API for IPbus reads, writes and RMW transactions. It is based on a delayed dispatch model in which multiple transactions are queued and concatenated within the transport layer payload buffers until the `dispatch` method is called.

In μHAL each device's register layout is specified by XML files. Each node of the XML tree represents either a single register, block RAM, FIFO, or a collection of these; the nodes in one file can reference other address files, such that the interfaces to repeated instances of a firmware module can be generated with minimal copy-paste of address file contents. This enables the user to write control software in a manner that intuitively mirrors the modular, hierarchical structure of large firmware designs.

The μHAL interface to each device (based on the methods of the `HwInterface` and `Node` classes) can run in one of two modes of operation. In the local-client mode, the μHAL library communicates directly with device over UDP. In the remote-client mode, the μHAL library communicates with hardware exclusively via a ControlHub. These differing modes of operation are implemented through the inheritance of of common interface, such that users can switch between the modes of operation by simply changing the prefix of a single string when creating a `HwInterface` instance.

μHAL is also packaged with an example GUI that is useful for monitoring the values of a subset of registers on a device during hardware development.

4. Control system topology

The topologies of an IPbus control system in some common scenarios are shown in figure 1. The simplest system (*upper left*) is a single target running the IPbus firmware, directly connected by a

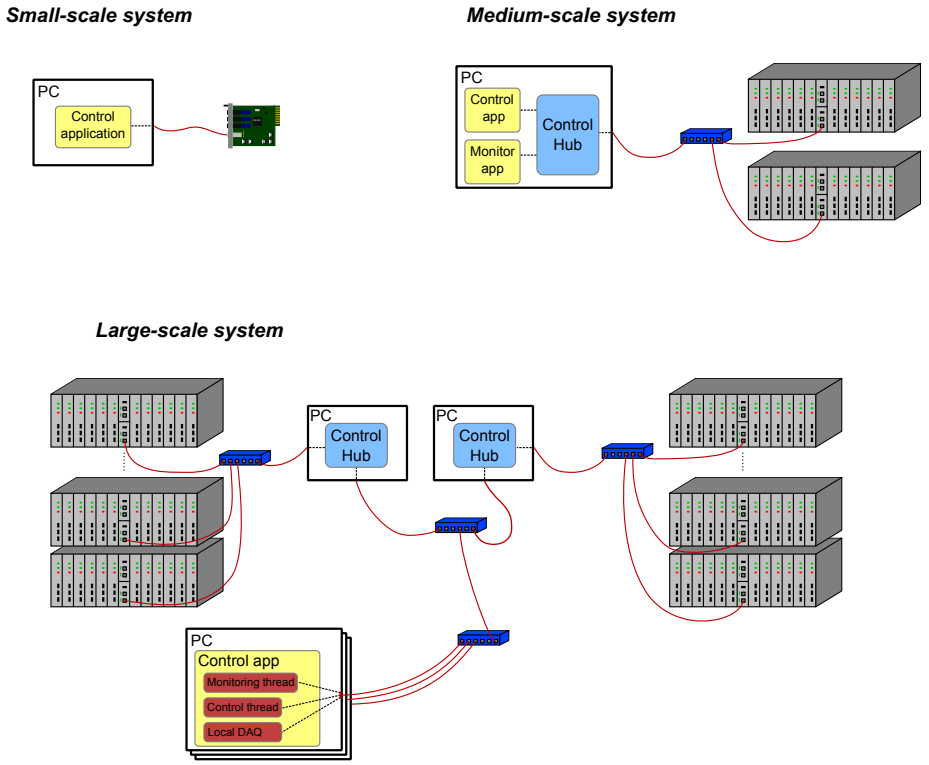


Figure 1. Example topologies of IPbus control systems involving μTCA hardware, from small to large scale.

single Ethernet cable to a computer running a C++/Python control application based on the μHAL library. This is the typical layout during early hardware development.

In a more complex scenario such as a beam test or integration tests, there will typically be several devices, with multiple control, monitoring and DAQ applications, as shown in figure 1 (*upper right*). Due to multiple applications simultaneously communicating with the devices, the IPbus traffic would be routed via a ControlHub, which would also recover any lost packets making the IPbus communication 100 % reliable.

For a full-scale IPbus system at a large experiment (such as ATLAS or CMS) there would be hundreds of IPbus devices spread across many crates, and the control/monitoring applications would be spread across many computers, as shown in figure 1 (*lower*). In this case the use of an Ethernet network naturally allows scalability with the ease of extending the network using multiple switches and routers. Additionally the recovery from computer failure is simplified with the possibility of having spare computers already connected to the network. Notably, the network could be divided into a separate subnet for each subdetector so that the network's logical segmentation matches the typical IPbus dataflow. The exact number of devices per ControlHub would be adapted based on performance requirements.

IPbus test system. A test system was set up in the CMS electronics integration centre at CERN, in order to investigate the reliability and performance of the IPbus suite using very similar network layout and hardware to that planned for final deployment in the CMS experiment. The test sys-

tem consists of network infrastructure, two computers, and one μTCA shelf containing 12 μTCA boards (AMCs), each running the IPbus 2.0 firmware core. The computers are Dell PowerEdge R300 rack PCs; three of the AMCs are GLIBs [6] and the other nine are Mini-T5s [7].

5. System reliability

The reliability and robustness of the IPbus suite has been ensured by extensive testing of both the software and firmware in a range of scenarios.

The software is tested by itself (independent of the hardware) each night using a *dummy hardware* executable which emulates the response of an IPbus device. A suite of unit test executables are run in order to test μHAL and the ControlHub with basic read/write/RMW operations to the dummy hardware running on the same machine. By configuring the operating system to randomly drop IP packets, these executables are also used to test the ControlHub's reliability mechanism.

The full IPbus control link (μHAL–ControlHub–firmware) has been tested with a variety of μTCA boards, using a μHAL-based C++ executable. This executable issues random sequences of reads, writes and RMW transactions to a device using random addresses, random depths for the reads and writes, and random values for the data written and the RMW parameters. The executable checks that all of the returned error codes indicate success, and checks that the values returned by the reads and RMW transactions are always correct. The released version of the firmware core was validated by running the executable for over 20 hours (corresponding to over 10 billion transactions) against the IPbus firmware core loaded on each of the Mini-T5, GLIB and MP7 boards. No errors were observed during this final testing.

6. Performance

The latency and block transfer throughput are two important parameters of a control system:

Latency is defined as the total round-trip time taken to perform an IPbus transaction, as measured in the μHAL client application.

Throughput is defined as the amount of user data transferred or received per unit of time.

In order to predict the performance of the future CMS IPbus control system, and verify the design of the IPbus components and their planned layout, the system performance was measured in several benchmark scenarios. These measurements were carried out in the IPbus test system, with the μHAL clients running on one computer and the ControlHub on the other computer.

1-to-1 block transfers. The block read/write latency and throughput for one μHAL client controlling one device via the ControlHub, is shown in figure 2. The median single-word write/read latency is approximately 250 μs. Although this single-word latency is significantly larger than with VME/PCIe-based control, for multiple transactions or large block transfers this is compensated by concatenating multiple transactions into each packet, and by having multiple packets in flight around the system at any given time. Hence, the block read/write throughput for payloads larger than 1 MByte is above 0.5 Gbit/s.

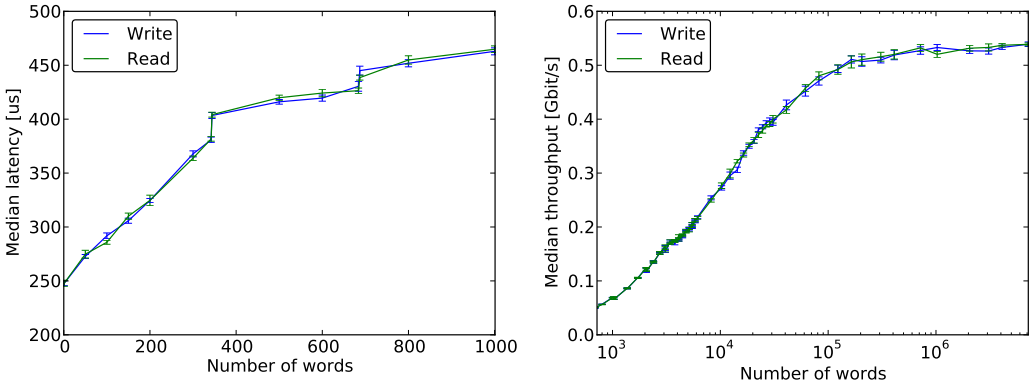


Figure 2. The median write/read and throughput as a function of depth, for one software client controlling one IPbus device, via the ControlHub.

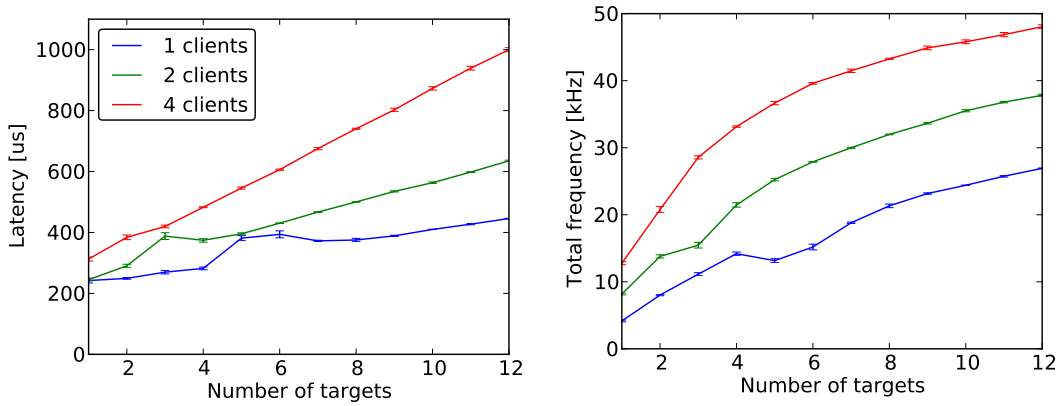


Figure 3. The latency and total system polling frequency for n clients each simultaneously polling a register in one of the m targets.

n-to-m polling. The system performance for multiple μHAL clients polling a single-word register in multiple devices is shown in via one ControlHub was also measured. The mean polling latency, and total system polling frequency, for 1, 2 or 4 clients per device are shown in figure 3 as a function of the number of devices. The latency experienced by each client gradually increases with the number of clients or devices, due to the the increasing load of network interrupts on the computers. However, the total polling frequency increases with the number of clients or devices in the system, as the ControlHub spreads its increasing workload over the four CPU cores.

n-to-n block transfers. The performance for continuous block reads and writes of all 12 boards in the μTCA crate was also measured. The Ethernet connection to a μTCA crate is via a Gigabit Ethernet socket on the front panel of the crate management module, the MCH (MicroTCA Carrier Hub). Each individual AMC in a μTCA crate is connected to the MCH's Ethernet switch by a separate bidirectional 1 Gbit/s link. In theory, this network topology could lead to congestion in the MCH switch during simultaneous block reads from multiple AMCs. For block reads, the reply packets are significantly larger than the request packets, and so the total instantaneous return bandwidth from the 12 AMCs into the MCH could exceed the 1 Gbit/s capacity of the link from

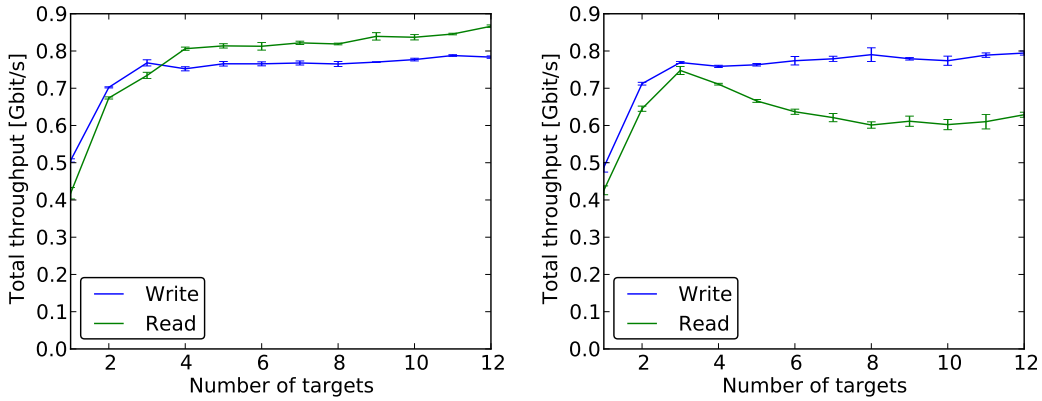


Figure 4. The total system throughput for n IPbus clients each simultaneously writing to / reading from one of n devices, via one ControlHub, using a NAT MCH (*left*) or a Vadatech MCH (*right*).

the MCH to the local network. However, within the IPbus protocol only a limited number of requests are in flight to each target at any given time, which imposes an upper limit on the total size of packets that would have to be buffered within the MCH switch. In practice whether or not such congestion leads to reduced performance depends on various factors, including the number of packets in flight to each AMC, and the design of the MCH switch. Within the CMS collaboration, MCH modules are currently being purchased from two vendors: NAT and Vadatech.

The IPbus system throughput for multi-client block reads and writes with multiple targets are shown in figure 4 for both the NAT and Vadatech MCHs. For the NAT MCH (V3.4), the read and write throughputs are similar; over 75 % of the Gigabit Ethernet bandwidth is utilised with three or more devices. However, using the Vadatech MCH (model UTC002-210-440-010), the system throughput degrades for simultaneous block reads from four or more devices due to congestion in the MCH switch, with read throughput approximately 20 % lower than write throughput for 8 or more targets. In order to reduce congestion, the system performance was re-measured with fewer packets in flight to each device; this can be achieved by editing one line in the ControlHub configuration file. With 11 packets in flight to each device (default value is 16), there is less congestion-induced packet loss, and so the simultaneous read throughput is above 0.75 Gbit/s for three or more devices; however, the maximum 1-client-to-1-target throughput decreases by approximately 12 %.

7. Conclusions

A new reliable, high-performance version of the IPbus protocol has been developed along with the associated suite of software and firmware, in order to control xTCA hardware via Gigabit Ethernet. An IPbus test system with realistic network topology was set up in the CMS electronics integration centre in order to verify the control system's reliability, and investigate its performance. For one software client controlling one device, the single-word read/write latency is approximately 250 μs and the block read/write throughput is above 0.5 Gbit/s for payloads larger than 1 MByte; the total block read/write throughput is above 0.75 Gbit/s for three or more boards in a single μTCA shelf.

The first large-scale IPbus system in the CMS experiment was deployed in August 2014, in preparation for the start of LHC Run 2 in 2015. Hence, development is now focused on simplifying

the monitoring of IPbus dataflows in large systems of hundreds of devices. The IPbus software and firmware suite will be optimised in order to improve performance with 10 Gigabit Ethernet. Additionally, an IPbus locking mechanism is being considered in order to provide exclusive access to IPbus devices from a single client for extended configuration sequences.

Acknowledgments

Acknowledgments.

References

[1] R. Frazier, G. Iles, D. Newbold, and A. Rose, *Software and firmware for controlling CMS trigger and readout hardware via gigabit Ethernet*, *Physics Procedia* **37** (2012) 1892

[2] The CMS collaboration, *Technical proposal for the upgrade of the CMS detector through 2020*, CERN, Geneva 2011. CERN-LHCC-2011-006.

[3] The ATLAS collaboration, *Letter of Intent for the Phase-I Upgrade of the ATLAS Experiment*, CERN, Geneva 2011. CERN-LHCC-2011-012.

[4] R. Frazier, G. Iles, M. Magrans de Abril, D. Newbold, A. Rose, D. Sankey, and T. Williams, *The IPbus protocol: version 2.0*, https://svnweb.cern.ch/trac/cactus/browser/trunk/doc/ipbus_protocol_v2_0.pdf

[5] The CMS Level-1 trigger project, *The CACTUS (Code Archive for CMS Trigger UpgradeS) SVN repository*, http://cactus.web.cern.ch/

[6] P. Vichoudis et al, *The Gigabit Link Interface Board (GLIB) ecosystem*, 2013 *JINST* **8** C03012.

[7] C. Foudas, R. Frazier, G. Hall, G. Iles, J. Jones, J. Marrouche, D. Newbold, and A. Rose, *A demonstrator for a level-1 trigger system based on MicroTCA technology and 5Gb/s optical links*, 2010 *JINST* **5** C11015.