An IP bus protocol for the ATLAS Tile calorimeter

Carlos Solans Sánchez[1] and Kehinde Gbenga Tomiwa[2]

[1]CERN
[2]University of the Witwatersrand, South Africa

October 21, 2015

Abstract

This document describes the implementation of a software client of the IP bus protocol for the ATLAS Tile calorimeter based on the one designed by CMS to satisfy the growing need to standardize the communication procedure with FPGA enabled devices inside a closed network in a platform independent way. This protocol is being used for the current electronics prototypes for the Phase-II upgrades of the Tile calorimeter. The specifications described are taken from version 2.0 of the original protocol.

1 Introduction

The ATLAS Tile calorimeter Phase-II upgrade program foresees the complete redesign of the front-end and back-end electronics in order to cope with the luminosity upgrade of the LHC. The back-end element of the detector read-out chain is a network enabled hardware device with an FPGA core that needs to be accessed simultaneously by the Detector Control System (DCS) and the read-out of the data for debugging and monitoring purposes.

The IP bus is a UDP/IP based protocol designed to communicate with ethernet aware FPGA devices. The protocol was designed by CMS Level-1 Trigger group and is recommended by the VME replacement group for Phase-II upgrades[1]. This protocol has a firmware module for hardware design and a software client to communicate with the hardware.

This document is organized like the following: Section 2 briefly describes the protocol, Section 3 describes the software implementation of a multi-platform client, Section 4 describes the application needed to connect more than one client simultaneously to any device, followed by a discussion of the performance measurements of the current implementation Section 5, and conclusions are discussed in Section 6.

2 The IP bus protocol

The IP bus protocol makes use of UDP payload of an IP packet with a maximum packet size of 1472 bytes, unless MTU is increased. The protocol is endian independent and the default port used by it is 50001. The IP bus packet starts with a 32-bit header that specifies the protocol version (2), contains a packet identifier, a byte order qualifier for the endianess declaration and the packet type. There are three types of packets in the protocol. A transaction packet, type 0x0, that describes read and write operations, a status packet, type 0x1, to retrieve operational information from the hardware, and the resend packet, type 0x2, to request a packet in case of packet loss. It includes a reliability mechanism to avoid packet loss, due to the fact that it relies on the UDP specification without packet acknowledge which is actually based on an acknowledge procedure. For each packet request there is a reply. The contents of the reply are different for each type of request. More information about the packet types can be found in [2].

3 The IP bus client

An IP bus software client has been implemented following the version 2.0 of the IP bus specification in Java [3], C++ [4], and Python. By definition the Java client is portable to any system running Java, and the C++ client is coded using C++11 standard. The Python client is a simple module extended from the C++ client. The client is based on the concept of direct memory access. Any 32-bit memory address can be accessed via the client for a read and write transaction and for one or many memory positions. This makes the client ideal for debugging purposes where the table of registers is constantly changing. One of the characteristics of the client is that it is single threaded. Only one transaction can be performed at a time in order to guarantee reliable communication. This allows to keep the code footprint low and portability across different operating systems leveraging on the performance of the client.

Upon creation, the client synchronizes the expected packet ID with the hardware and the is ready to use. If the reply for any request times out, the client will try to re-synchronize with the server and resend the request only once. The client implements a read and a write method to perform the corresponding operations. One address can be accessed at a time, with one or more values read or written at a time in contiguous addresses or on the same address. The next expected packet ID is increased after the reply of the server for each transaction which is used by the constructor of the packet to allocate the ID for the next request. If a reply is not received for a given request, the packet is considered as not sent.

4 The IP bus hub

In order to handle more than one client communication at a time, a packet manager is needed between the clients and the hardware, in order to properly queue the requests. The IP bus hub is a software instance of this packet manager that acts as a mediator between a single hardware server and multiple clients. It allows simultaneous access to one device from one or more client applications, as foreseen in the phase-II upgrade, where the back-end element should be accessed by DCS and monitoring processes simultaneously. The clients communicate with the IP bus hub using the IP bus client, and the hub itself uses an instance of the client to have one single connection with the hardware.

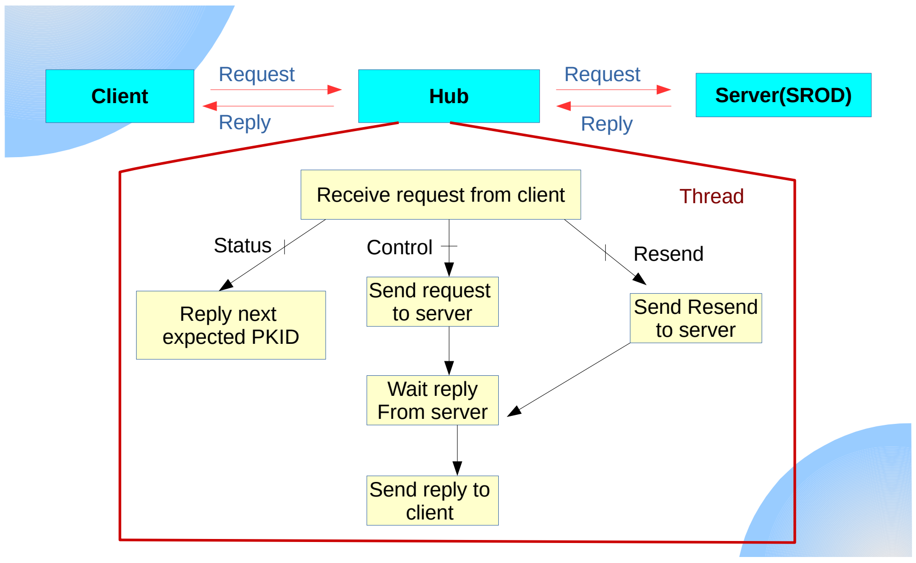
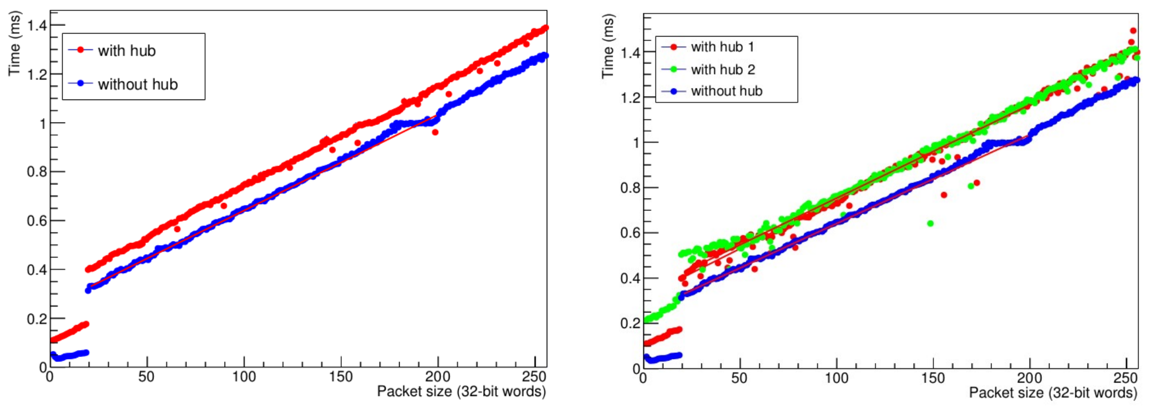


Figure 1: Block diagram of the IP bus hub program logic.

The IP bus hub is coded using C++11 standard and uses standard threads to listen on the different ports, one per client application to be connected. At initialization, it synchronizes with the server and creates one thread per client, so that each thread listens to IP bus requests independently with its own private packet ID sequence. Each time a new request is received, the listener locks the access to the server and performs the action requested. If a status request is received, the hub returns the expected packet ID of that client. When a control packet is received the packet ID sent by the client is translated to the expected packet ID of the server and the packet forwarded to it. Upon reply from the server, the packet ID is translated back to the expected packet ID by the client. The resend packet is used to recover a missing packet, when the client times out while waiting for a packet. Figure 1 shows a block diagram of the IP bus hub program flow.

5 Performance measurements

To measure the performance of our implementation of the IP bus protocol we use a Xilinx VC707 evaluation board that is equipped with a Virtex-7 FPGA to act as our hardware device (server), configured with a static IP address, and a software client running our version of the software. We consider two different configurations. First one using a direct connection to the server, with the IP address of the client being in the same net mask as the server, and a second one where the client is connected to the server through the IP bus hub which is executed on the same client host without any network communication delays.



(a) One client (b) Two clients

Figure 2: Average transaction time per number of 32-bit words read.

First we measure the maximum sample rate achievable by the protocol with and without the hub. Figure 2a shows the time it takes to read a packet as a function of the packet size in units of 32-bit words. There is a step in time at 20 32-bit words that is explained by the minimum size required for small packet sizes, where the non-dependence with time is observable in the test without the hub. Since these small packet sizes are not characteristic of the protocol, a linear fit is conducted on the rest of the domain and fit parameters are shown in Table 1.

The linear fit is inverted resulting in a bandwidth (rate) measurement of 63 ± 1 kB/s (254 ± 1 kwords/s) without the hub and 61± 1 kB/s (244 ± 1 kwords/s) with the hub.

In a more realistic approach, taking into account that the event payload is one 32-bit word that contains the 12 bit sample value plus other quantities per each of the two gains for each of the 48 channels, the event rate for 32 samples is 150 ± 1 Hz with the hub and 160 ± 1 Hz without the hub.

Parameter	Direct (μs)	With hub (μs)
y-intercept	248.7 ± 0.3	326.9 ± 0.2
slope	3.926 ± 0.003	4.100 ± 0.002

Table 1: Fit parameters

Secondly we measure the impact of a second client connected to the hub that is reading different packet sizes every 5 seconds. As shown in Figure 2b, the time increase due to a parallel read-out has a negligible impact on the primary one, and the event rate is 150 ± 1 Hz for both clients.

6 Conclusions

A custom implementation of the IP bus software clients is described in this document. The performance of which is not comparable to the one of the IP bus repository. Yet the purpose of a fully functional client is accomplished and enables development of multiple tools in three programming languages fulfilling the requirement of being portable across multiple operating systems. Furthermore, the IP bus hub software described in this document enables multiple clients to connect to the same host with a low impact on transaction speed. For these reasons this custom implementation of the IP bus software is recommended for configuration and control of ethernet enabled FPGA designs for Tile calorimeter upgrades.

References

[1] ATCA in ATLAS. EDMS document ATU-GE-ES-0001.

[2] https://svnweb.cern.ch/trac/cactus/browser/trunk/doc/ipbus_protocol_v2_0.pdf.

[3] https://svnweb.cern.ch/trac/atlasgroups/browser/detectors/tilecal/prometeo/jipbus.

[4] https://svnweb.cern.ch/trac/atlasgroups/browser/detectors/tilecal/prometeo/ipbus.