A data readout approach for physics experiments*

HUANG Xi-Ru(黄锡汝)[1,2;1)] CAO Ping(曹平)[1,2;2)] GAO Li-Wei(高力为)[1,2] ZHENG Jia-Jun(郑佳俊)[1,2]

[1] State Key Laboratory of Particle Detection and Electronics, University of Science and Technology of China, Hefei 230026, China
[2] Anhui Key Laboratory of Physical Electronics, Department of Modern Physics, University of Science and Technology of China, Hefei 230026, China

Abstract: With increasing physical event rates and the number of electronic channels, traditional readout schemes meet the challenge of improving readout speed caused by the limited bandwidth of the crate backplane. In this paper, a high-speed data readout method based on the Ethernet is presented to make each readout module capable of transmitting data to the DAQ. Features of explicitly parallel data transmitting and distributed network architecture give the readout system the advantage of adapting varying requirements of particle physics experiments. Furthermore, to guarantee the readout performance and flexibility, a standalone embedded CPU system is utilized for network protocol stack processing. To receive the customized data format and protocol from front-end electronics, a field programmable gate array (FPGA) is used for logic reconfiguration. To optimize the interface and to improve the data throughput between CPU and FPGA, a sophisticated method based on SRAM is presented in this paper. For the purpose of evaluating this high-speed readout method, a simplified readout module is designed and implemented. Test results show that this module can support up to 70 Mbps data throughput from the readout module to DAQ.

Key words: data readout, physics experiments, readout system, data acquisition

PACS: 29.85.Ca **DOI:** 10.1088/1674-1137/39/7/076102

1 Introduction

In a typical particle physics experiment, the readout system is generally implemented in a standard crate (e.g. VME). The crate contains a variety of electronic modules, generally including: readout modules used for receiving data from the front-end electronics (FEE) and transmitting event data to the crate controller via the crate backplane bus, and a crate controller used for collecting event data and sending event data through Gigabit networks to the data acquisition (DAQ) system in real time [1]. Fig. 1 shows a simplified version of the architecture of a typical readout system.

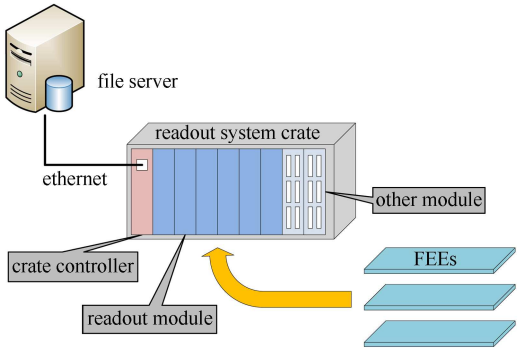

Fig. 1. (color online) Simplified architecture of a typical readout system.

An "event" describes the result of a single reaction between two colliding particles in nuclear and particle physics. Among the large amounts of events, only a rare event is associated with the physics objects (electrons, muons, photon, etc.) of interest to physicists. In order to take out most of the irrelevant events and reduce the challenges of storage, it is essential to realize an efficient trigger system to filter events. In a conventional data acquisition electronics system, the level-1 trigger that makes the first level of event selection is implemented by hardware and provides trigger signals for readout modules to select event data. As a result, the data rate to be transmitted can be reduced by several orders of magnitude. However, research on various particle physics experiments shows that the physical event rate and number of electronic channels are increasing. ATLAS, which is a particle physics experiment at the Large Hadron Collider at CERN, has over 10^7 electronic channels and the raw data rate is up to hundreds of GBytes/s [2, 3]. In

Received 24 October 2014

∗ Supported by National Natural Science Foundation of China (11005107) and Independent Projects of State Key Laboratory of Particle Detection and Electronics (201301)

1) E-mail: xiru@mail.ustc.edu.cn

2) E-mail: cping@ustc.edu.cn

addition, some collider physics experiments, for example GREAT [4] at Jyväskylä and CBM [5] at GSI/FAIR, have begun to use a triggerless data collection method called Total Data Readout [6] to solve some problems (e.g. dead time) caused by the traditional trigger structure. Here, the "triggerless" is not entirely without triggering, but the hardware trigger is replaced by a software trigger in the DAQ system. Therefore, the readout scheme meets the challenge of improving readout speed caused by the limited bandwidth of the crate backplane.

Advanced Telecommunications Computing Architecture (ATCA) [7] developed by the PCI Industrial Computer Manufacturers Group (PICMG) is a set of industry standard specifications for the next generation of telecommunication network and data center equipment. The ATCA backplane provides point-to-point connections between the boards and supports the dual star, dual-dual star, and mesh topologies. The dual star topology based on 4X InfiniBand links supports 10 Gbit/s of raw throughput between hub boards and node boards [8]. The ATCA high performance backplane has been proposed to develop the next generation of electronics standards for physics [9]. Despite all of these features, however, ATCA has not been widely applied and is popularizing only slowly in particle physics experiments.

The main task of the crate controller in a typical readout system is to gather data from the crate backplane and then transmit them to the DAQ system through networks. This is a centralized architecture that needs to improve backplane performance for applications with higher readout speed requirement. Actually, a decentralized architecture can be used for readout systems as it has the advantage of improved readout speed, compared with the centralized architecture.

In this paper, a high-speed data readout method based on the Ethernet is introduced. A simplified readout prototype module is designed and implemented. This module has a 100 M Ethernet port on it. The test result shows that the raw data throughput from the readout module to DAQ can reach up to 70 Mbps.

2 High-speed data readout method

Instead of improving backplane performance, each readout module is designed to have the capability of transmitting data to the DAQ, as shown in Fig. 2.

To make each readout module support network communication, a dedicated CPU is utilized to establish an embedded system. Fig. 3 shows the system architecture of the method. It consists of two main blocks: embedded CPU and field programmable gate array (FPGA). There are three important transmission channels: a high-speed data transmission channel, a low-speed command transmission channel, and an Ethernet interface. Considering the fact that the throughput of data upload from FEE to DAQ is much larger than that of data download to FEE in particle physics experiments, the high-speed data

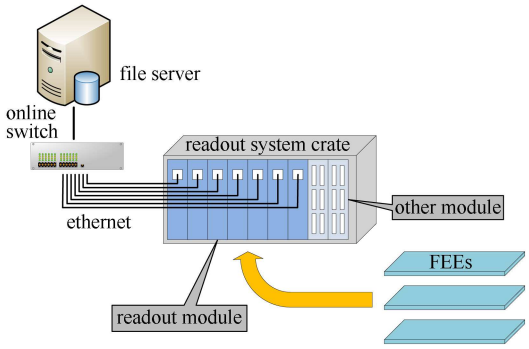

Fig. 2. (color online) Simplified architecture of a new-type of readout system.

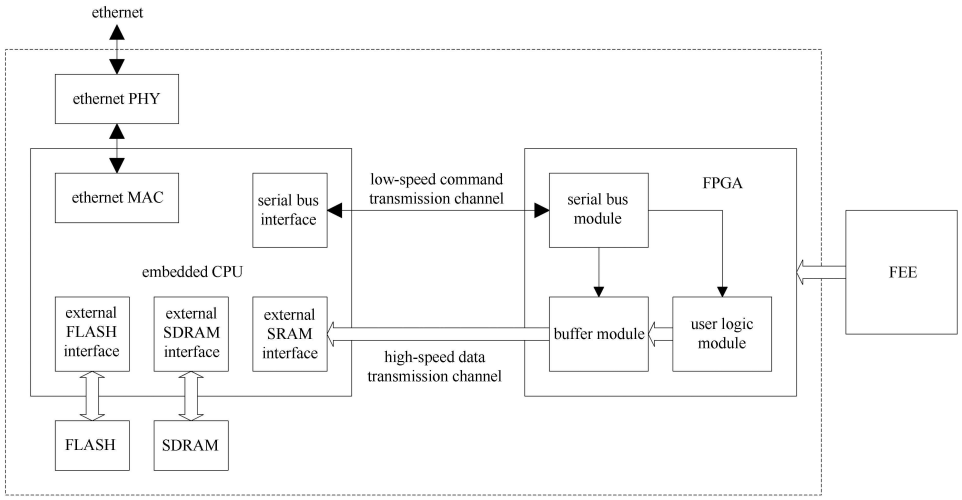

Fig. 3. System architecture of the high-speed data readout method.

transmission channel is designed as a simplex mode and only used for transmitting data from FPGA to CPU, while the low-speed command transmission channel is also committed as the simplex mode for sending commands or configurations from CPU to FPGA. Based on these two dedicated channels, a full-duplex data communication mode between CPU and FPGA can be realized.

Unrelated with FPGA, the CPU is a standalone chip for maintaining an embedded Linux operation system. The TCP/IP protocol stack is supported in the operating system. This embedded system can be considered as a small computer based on a particular microprocessor core. This chip generally integrates various necessary features and peripherals such as internal memory (e.g., ROM, RAM), input/output control unit, serial bus interface (e.g., serial peripheral interface (SPI), universal asynchronous receiver transceiver (UART), inter-integrated circuit (I^2C)), and external memory controller etc. The external memory controller is capable of handling several types of external memories and peripheral devices, such as SRAM, Flash, and SDRAM. There are many embedded CPUs that can be considered as a candidate for implementing the readout module, such as ARM, Power PC, MIPS, Am186/88, and so on.

With the help of the embedded Linux system, data can be packaged with the TCP/IP protocol and sent to the DAQ, but there lies a problem in how to transfer data from the FEE to this CPU system efficiently and smoothly in real-time. Considering simplicity and universality, in this readout architecture, the high-speed data transmission channel is implemented by using an external SRAM interface which connects CPU and FPGA together. On one side, the FPGA receives data from the FEE, re-packages and feeds the data to this SRAM interface. On the other side, the CPU receives data from this SRAM interface and relays to the DAQ through the Ethernet port. The use of the SRAM interface makes it easier to design or implement readout modules, whether for hardware circuit or FPGA logic. There is no need to use special IP (intellectual property) cores, which degrades the cost or requirement for FPGA. The SRAM interface has advantages of simplicity, low cost and high performance as well. In fact, the maximum data throughput of a synchronous SRAM interface can reach up to 1600 Mbps with 100 MHz running clock and 16 bit data width. The maximum data throughput of an asynchronous SRAM interface can be up to 800 Mbps with 20 ns read/write cycle period and 16 bit data width.

Differing from the data channel, the low-speed command transmission channel is implemented by using a serial data interface between CPU and FPGA. Besides, the CPU system must also support a high-speed Ethernet MAC and physical signal interface, which is used for transmitting/receiving data to/from Ethernet transmission lines.

Nowadays, FPGAs have large resources of logic gates and RAM blocks which make them good for implementing complex digital logic design. In this system architecture, the key task of the FPGA is to process data sent from the FEE with customized format and protocol, and to send valid data to the CPU through the high-speed data channel for Ethernet transmission.

3 Implementation of the module inside FPGA

During transferring, the CPU reads data from the FPGA via the SRAM interface. However, FPGA is not a standard SRAM for CPU. It should always be ready and respond correctly whenever there is a data receiving or transmitting transaction. A data synchronization mechanism should be guaranteed between the CPU and FPGA. Here a hand-shake protocol is provided. Table 1 lists all of the hand-shake signals.

Table 1. Additional hand-shake signals.

signal name	description
read ready	The read ready signal is used for initiating a data transfer by CPU.
IRQ	The interrupt request signal is used for interrupting CPU by FPGA. When FPGA detects that the read ready signal is valid, it will judge whether the data is ready. If it is ready, FPGA makes the interrupt request signal valid. Then, when FPGA detects that the read ready signal is invalid, it will make the interrupt request signal invalid.
CLK	The clock signal is used for synchronizing signal timing of the SRAM interface. If the SRAM interface is an asynchronous SRAM interface, CPU needs to provide a clock signal for FPGA.

In the FPGA, the main logic module is called the buffer module. It is in charge of caching data from the user logic module and sending them to the CPU through the SRAM bus. Fig. 4 shows the detailed structure of this buffer module. It consists of three parts: FIFO, FIFO controller, and state machine. FIFO is used for caching data. It is designed as an asynchronous FIFO. It provides a general FIFO interface (including signals of datain, wrclk, wrreq, wrfull) to the user logic module from the input side. From the output side, there is a data bus connecting it with the SRAM interface. To guarantee synchronous operation, the clock of the FIFO output side is provided by the CPU.

The FIFO controller generates the read request signal for FIFO according to the control signal timing of the SRAM bus, and does not need to use the address signal of the SRAM bus. When the read ready signal is valid, the state machine will judge whether the data in FIFO is ready by comparing the transmission length with the

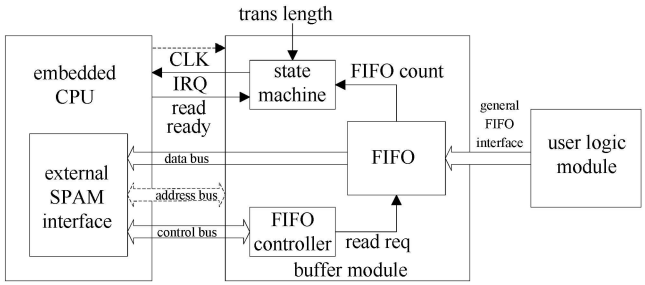

Fig. 4. Block diagram of the buffer module.

FIFO count. Then the state machine makes the IRQ signal valid to respond to the read data request of the CPU. In addition, the transmission length can be set to a default value or configured by a command.

4 Implementation of network transmission

An embedded Linux operating system will be installed on this microcomputer system. Linux OS provides complete, powerful network functions for users, with performance in real-time applications improving with the development of new scheduling algorithms. The Linux kernel is released under an open source license, so anyone can read and modify its code.

4.1 Design of device-driver

A device driver (commonly called a driver) provides a software interface to hardware devices and tells the operating system and other software how to access hardware without needing to know precise details of the hardware being used. In Linux environments, drivers are built as modules that can be easily loaded into or unloaded from the kernel during running time.

For the Linux OS, the buffer module can be viewed as a device which cannot be driven by the generic SRAM driver. Therefore, a special driver needs to be prepared for controlling and managing the buffer module. The driver can be classified as a character driver. Fig. 5 shows the structure of the driver in kernel space.

There are several major modules in this structure: initialization module, exit module, open module, release module, read module, memory mapping (mmap) module, and interrupt handler.

First, the initialization module has to configure the device resources, for example, set IRQ pin, set read ready pin, and configure the SRAM interface. Then the module allocates a major and a minor number for this char device. Finally, the initialization is finished after registration of the char device.

Once the module has been installed successfully, the driver is ready for work. The transmission software in user space uses the open method to request the I/O memory resources according to the memory mapping of CPU. Then the open method completes an important process: requesting an interrupt number from the system and installing an interrupt handler. Finally, if the open method is called successfully, it will return a file descriptor that establishes an access path to the device.

In our design scheme, the transmission software does not use the read method to acquire data, but accesses the address pointer returned from the mmap method to fetch data. Procedures for obtaining data are described as follows: first, the transmission software tries to read the device and waits for its return. On the kernel side, the read module first sets read ready pin valid to inform FPGA that it is ready to receive data. Next a macro

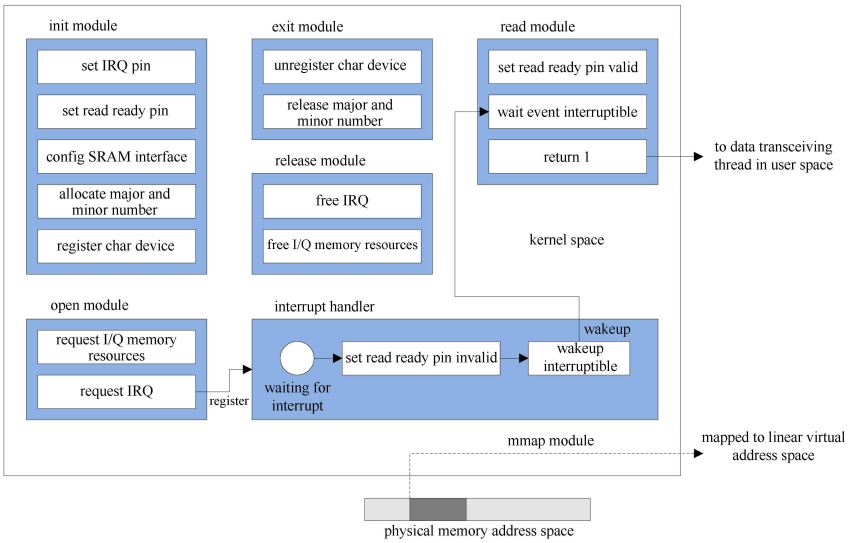

Fig. 5. (color online) Driver in kernel space.

called wait_event_interruptible is used to put the process into an interruptible sleep. Once the data in the buffer module is ready, an interrupt is generated and the corresponding handler is immediately evoked. Then the interrupt handler sets read ready pin invalid and wakes the read module up by the function called wake_up_interruptible. Finally the read module returns a status flag to the transmission software. At this point, the read method is complete, and then the transmission software can access the address pointer to read data.

4.2 Design of software

One of the main tasks of the readout module designed by the high-speed data readout method is to transfer data to the concentration center or PC farms through the Ethernet. To optimize transfer performance, the transmission software residing in the embedded Linux system must be designed in a simple and parallel structure, as shown in Fig. 6.

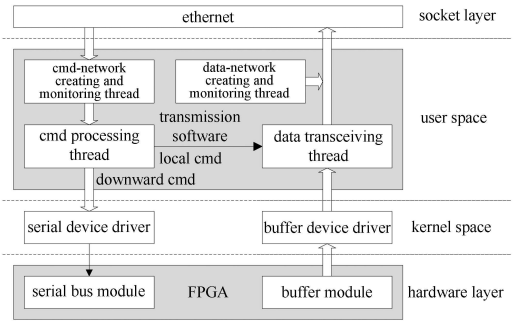

Fig. 6. (color online) Structure of the transmission software in user space.

There are four kinds of threads in this transmission software: a command-network creating and monitoring thread, a command processing thread, a data-network creating and monitoring thread, and a data transceiving thread. When the transmission software starts running, these threads will automatically be created with detached attributes and be run in parallel in the user space. The command-network creating and monitoring thread creates a server socket and then builds a command channel by accepting connections from the PC client. This command channel is only used for receiving commands. After the channel is built, the thread monitors its connected state. Once the channel is disconnected, the thread will close the original channel and rebuild a new one. The data-network creating and monitoring thread also works in a similar way.

The task of the command processing thread is to deal with commands. If the command is a local command used for configuring and controlling the software, the thread will analyze the command and then make an appropriate response. If the command is a downward command used for configuring and controlling the hardware, the thread will send the command to FPGA via the write method related to the serial device.

The task of the data transceiving thread is to receive data from the FPGA and then transmit them to the PC via the Ethernet. During the initialization of the thread, it creates a file descriptor related to the buffer module by using the open method and gets an address pointer related to the physical address space of the buffer module by using the mmap method. Then in the loop process, it first needs to ensure that the data in the buffer module is ready using the read method with the file descriptor, and next transfers this piece of data to the PC using the write method with the data server socket and address pointer, and then returns to the beginning of the loop.

5 Experiments and evaluation

For the purpose of evaluating this high-speed readout method, a simplified readout module is designed and implemented with an AT91RM9200 and EP3C40F780C8 chip.

The AT91RM9200 chip [10] is a microcontroller produced by the Atmel Corporation, which is based on the ARM920T 32 bit RISC processor with 16 KB instruction and 16 KB data cache memories and memory management unit (MMU). Its speed can achieve 200 MIPS when working at 180 MHz frequency. The chip supports a 16 bit asynchronous SRAM interface, a 10/100 Base-T Ethernet MAC, and four universal synchronous/asynchronous receiver transceivers (USART). Therefore, this chip is able to satisfy the application requirements of the readout method. The EP3C40F780C8 chip [11] is an FPGA of the Cyclone III device family, which is a high functionality, low-power and low-cost FPGA family of the Altera Corporation.

Here, the high-speed data transmission channel is implemented by using the static memory controller, an external interrupt source, a programmable external clock signal and a programmable I/O line of the AT91RM9200 chip. The low-speed command transmission channel is implemented by using a USART of the AT91RM9200 chip. The prototype of the readout module is shown in Fig. 7.

To measure the network transfer performance of the readout module, the Ethernet port of the readout module is directly connected to a PC using a good wire. Then an embedded Linux OS with kernel version 2.6 runs on the readout module. As shown in Fig. 8, route 1 is used for measuring the throughput of the SRAM bus from EP3C40F780C8 to AT91RM9200, route 2 is used for measuring the throughput of the 100M Ethernet from AT91RM9200 to the PC, and route 3 is used for measuring the throughput of the 100M Ethernet from EP3C40F780C8 to PC. Moreover, the data in the buffer

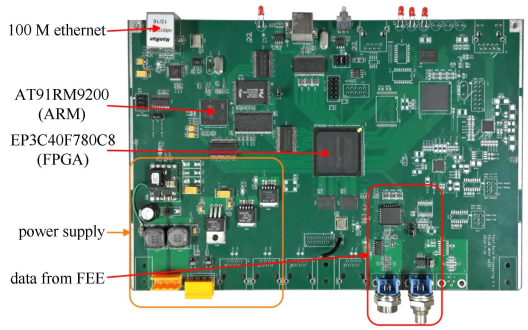

Fig. 7. (color online) Prototype PCB of the readout module.

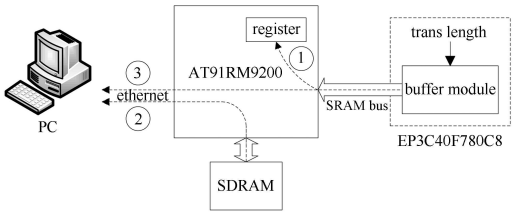

Fig. 8. Measurement scheme.

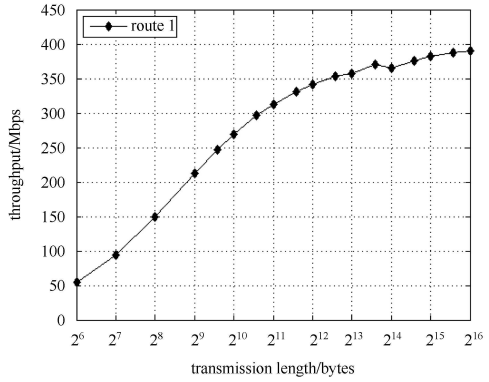

Fig. 9. Data throughput of the SRAM interface.

module is always ready.

The measurement of the three throughputs mentioned above is limited in a wide range of transmission lengths, as shown in Figs. 9 and 10. There are two main time costs in route 1 and route 3: data synchronization time and data transmission time. When the transmission length is relatively small, e.g. 256 bytes, most of the time is spent on data synchronization. With the increasing of transmission length, data synchronization time decreases. Meanwhile, the throughput of route 1 and 3 increases and finally approaches a stable value. For transmission length larger than 16384 bytes, the throughput of route 2 and 3 can reach up to 70 Mbps, while route 1 can reach more than 350 Mbps.

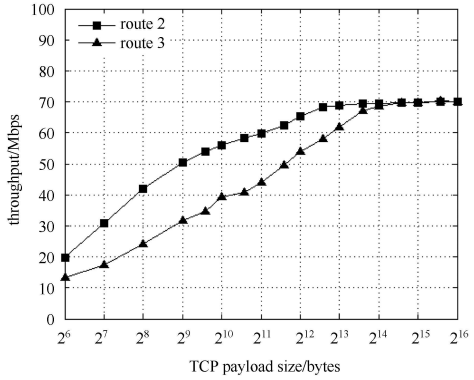

Fig. 10. Network throughput.

6 Conclusions and future work

A high-speed data readout method based on embedded CPU and FPGA is presented in this paper. This method makes each readout module capable of communicating with the DAQ system through the network. It has the advantages of simplicity, universality, expansibility and low cost. It is suitable for various applications of data readout in particle physics experiments. To verify and evaluate this method, a prototype readout module has been designed and implemented. Test results show that this module can support up to 70 Mbps valid data throughput from the readout module to the DAQ.

To further improve the data throughput (e.g. 1000 Mbps), a more powerful CPU should be adopted for data and protocol processing. To achieve higher performance of transmitting data to the CPU from the FPGA with this method, there is little modification except for improving the clock frequency and modifying the logical behavior of the SRAM interface based on the CPU.

References

1 BESIII Design Report. BESIII DAQ System (Online). http://bes.ihep.ac.cn/bes3/design05/design/design1.htm
2 ATLAS Collaboration. ATLAS Inner Detector: Technical Design Report. Volume 1. 1997, CERN-LHCC-97-016
3 ATLAS Collaboration. ATLAS High-Level Trigger, Data-Acquisition and Controls: Technical Design Report. 2003, CERN-LHCC-2003-022
4 Page R D, Andreyev A N, Appelbe D E et al. Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms, 2003, **204**: 634–637
5 Friese V. Nuclear Physics A, 2006, **774**: 377–386
6 Lazarus I H et al. IEEE Trans. on Nucl. Sci., 2001, **48**(3): 567
7 http://www.picmg.org/openstandards/advancedtca/
8 http://picmg.org//wp-content/uploads/PICMG_3_2_Shortform.pdf
9 Raymond S. Larsen. Advances in Developing Next-Generation Electronics Standards for Physics. Real Time Conference, 2009. RT'09. 16th IEEE-NPSS. 2009, 7–15
10 http://www.atmel.com/Images/doc1768.pdf
11 http://www.altera.com/literature/hb/cyc3/cyclone3_handbook.pdf