Hardware/Software Co-design of a General-Purpose Computation Platform in Particle Physics

Ming Liu[1,2], Wolfgang Kuehn[2], Zhonghai Lu[1], Axel Jantsch[1], Shuo Yang[1,2], Tiago Perez[2], Zhenan Liu[3]

1. Dept. of Electronic, Computer and Software Systems, Royal Institute of Technology, Sweden

2. II. Physics Institute, Justus-Liebig-University Giessen, Germany

3. Experimental Physics Center, Institute of High Energy Physics, China

{mingliu, zhonghai, axel, shuo}@kth.se, {wolfgang.kuehn, tiago.perez}@exp2.physik.uni-giessen.de, liuza@ihep.ac.cn

Abstract

In this paper we present a hardware/software co-design based computation platform for online data processing in particle physical experiments. Our goal is to ease and accelerate the development and make it universal and scalable for multiple applications, on the premise of guaranteeing high communicating and processing capabilities. The entire computation network consists of quite a few interconnected compute nodes, each of which has multiple FPGAs to implement specific algorithms for data processing. High-speed communication features including RocketIO multi-gigabit transceiver and Gigabit Ethernet are supported by FPGAs to construct internal and external connections. An embedded Linux operating system is fitted on the PowerPC CPU core inside the Xilinx Virtex-4 FX FPGA. Thus programmers can access hardware resources via device drivers and write application programs to manage the system from the high level. Furthermore measurements have been executed using the development board to investigate both communicating and processing performances of the system. Results show that the computation platform is able to communicate at a UDP/IP data rate of around 400 Mbps per Ethernet link, and the event selection engine could process an event stream of 148.1 MBytes/s at an interesting event rate of 25%.

1. Introduction

Modern nuclear and particle physics experiments, for example HADES [1] and PANDA [2] at GSI, BESIII [3] at IHEP, are expected to run at a very high reaction rate (e.g. PANDA, 10-20 MHz) and able to deliver a data rate of up to hundred GBytes/s (PANDA, from 40 up to 200 GBytes/s). Among the huge amounts of data, only a rare proportion is of interest due to its particular physics contents and should be selected for in-depth physics analysis. In addition to the heavy offline computation on the PC farm, in-field processing and triggering are also desired to select candidates of interesting events and discard some obviously uninteresting ones. Feature extraction algorithms are relevant to the online selection process such as *Cherenkov ring recognition*, *shower recognition*, *Time-Of-Flight (TOF) processing*, *tracking in Multiwire Drift Chambers (MDC)*, high level *correlation* [4] and *event selection*. To implement those algorithms in field and achieve the high processing requirement, an extremely powerful computation network which consists of many nodes working in parallel is to be constructed and integrated into the experimental setup. [5] Our aim is to provide such a platform which could be easily networked and as well it is expected to be universal and highly parameterized for different experimental conditions.

Due to the particularity of our application in particle physics, the requirements on our design vary a lot from other common embedded ones, such as consumer electronics or industrial control computers. Firstly, the performance requirement to process at least several tens GBytes/s in realtime is quite critical. Without a powerful interconnected network, this cannot be turned into reality. At the same time, since compute nodes are fundamental units which constitute the entire network, the performance requirement finally falls down on the node design. Obviously the more communicating and computing capabilities each node provides, the smaller node numbers and simpler network topology which both lead to a cost-efficient system could be achieved. Secondly, our design is expected to be highly parameterized and easily reconfigurable and scalable to meet the following cases: 1. The platform should be applicable for many different experiments without much extra effort. What we only need to do for various experimental subjects is to adapt specifically designed processing engine modules

into the system, without changing the systematic architecture and other peripherals; 2. Since most of the operators in the experiments will be physicists who have little experience in hardware design, a parameterized and reconfigurable platform will be convenient for them to control the experimental conditions; 3. Because of the spatial or physical limitations such as the harmful radiation during the experiments, remote reconfiguration capability is also important for our application.

According to the requirements listed above, our proposed solution is a co-design based network platform. Using a defined set of criteria for hardware/software partitioning between FPGA fabric and embedded CPU, in conjunction with the respective implementations of both hardware computing engines and software controls, the flexibility requirements could be met with high performance guaranteed. In Section 2 we will first address the network architecture from a global perspective, considering the case that it is impractical to solve such a heavy computing problem in a single node. After that, we focus on a single FPGA node design which is universal to accommodate different algorithms. In Section 3, the partitioning criteria and strategy are discussed which lead the system to both high performance and cost efficiency. In Section 4 the FPGA-based hardware design will be described, and the embedded Linux Operating System (OS) based software design will be presented in Section 5. Performance measurements and results are reported in Section 6. Finally we conclude the paper and propose the future work in Section 7.

2. System Architecture

2.1. Computation Network Architecture

A network structure consisting of interconnected nodes is proved to be and widely adopted as a good candidate to deal with large amounts of data. In our application, feature extraction algorithms will be partitioned and distributed in many nodes. Without bonding of multiple communication channels together with parallel and pipelined processing in different units, the data rate of up to hundred GBytes/s cannot be managed. Among all the Compute Nodes (CN), high speed interconnections should also be provided for sharing their results and correlating the selected sub-events (see Section 3 for definition) from different algorithms for final decisions. Thus the network architecture is shown in figure 1. For each node in the network, there are two types of links: internal links and external ones. Internal connections make it possible for all compute nodes to exchange and correlate their processing results. Its topology structure is quite dependent on the communication requirement of the *correlation* algorithm, further the partitioning methods and working mechanisms of each specific feature extraction. External links adopt optical links and Gigabit Ethernet to communicate. Optical transceivers are Xilinx RocketIO based and all their channels are bonded to accept data streams from detectors. After being subject to corresponding algorithms, selected sub-events belonging to the same physics process will be encapsulated into a single event. All events could be afterwards pre-filtered for rejecting uninteresting ones, and Gigabit Ethernet is a good choice for conveying the final selected results to the PC farm for higher level analysis and storage. We are utilizing its ease to build the Ethernet network with ordinary PCs, and moreover the reduced data rate after online selection could be successfully managed via multiple channels.

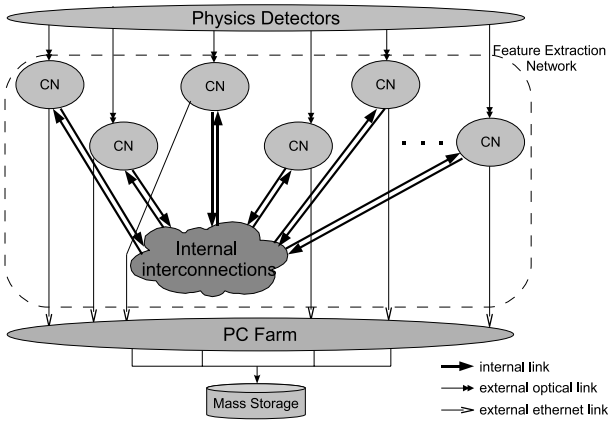

Figure 1. Computation network architecture for online processing in experiments.

2.2. Compute Node Architecture

Figure 2 shows the schematic of a compute node for prototype design. On each board there are five Xilinx Virtex-4 FX60 FPGAs in total. Four of them (No. 1 to 4) perform as algorithm processors and work in a parallel or pipelined fashion to increase processing capability. The fifth FPGA (No. 0) acts as a switch which connects to other CN boards in an ATCA shelf via a full mesh backplane. With these high-speed RocketIO connections all boards in the shelf could talk to each other, and the full mesh topology provides much flexibility to implement the *correlation* algorithm. All five FPGAs are equipped with local DDR2 memories for buffering data and other purposes such as large lookup tables. In addition two optical links and two Gigabit Ethernet ports are connected to each FPGA for communications with external devices.

From now on we will focus on the compute node design, specifically those four identical FPGA nodes working

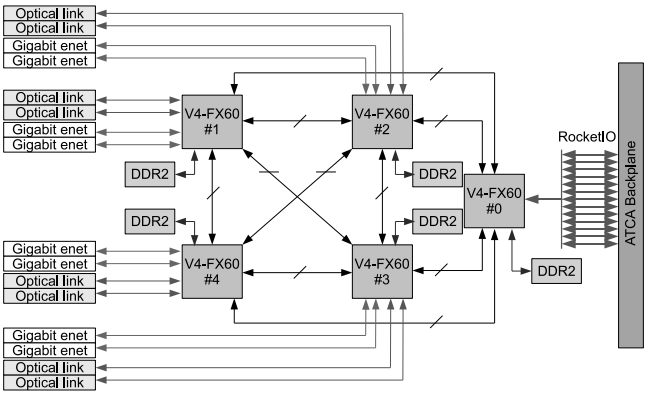

Figure 2. Compute node for prototype design.

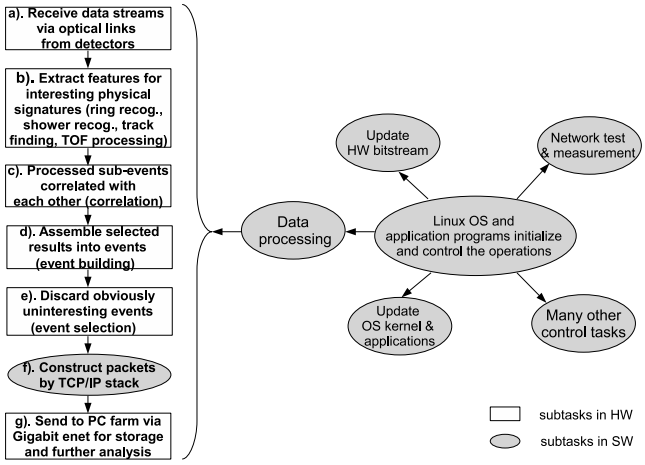

Figure 3. Functional partitioning strategy.

as algorithm processors. With different hardware processing modules fitted inside, different feature extractions could be realized for online processing.

3. Partitioning Criteria and Strategy

System partitioning, also referred to as functional partitioning is an essential problem since it affects overall system cost and performance directly. In our design, we adopt the criteria listed as following:

1. To achieve powerful processing capabilities, all of the performance-intensive computation algorithms are to be implemented in the FPGA fabric in hardware, working in a parallel or pipelined fashion.

2. For easy operation and reconfiguration, the control tasks should be realized preferably in software via high level application programs. A universal operating system is preferred to act as a manager of the hardware and software resources, and Linux is a good candidate.

3. To speed up the development process and save human resources or cost, we use existing and popular communication techniques. For instance, Gigabit Ethernet with the soft Linux TCP/IP stack[1] are utilized in our design.

Thus we present our concrete partitioning strategy shown in figure 3 as follows. The embedded Linux is the control center of the whole system. According to different application programs or commands, operators could issue many kinds of operations such as data processing, updating hardware or software design, and so on. On the data processing path, there are several subtasks which cooperate to select events on-line, including: a). receiving data streams via optical links; b). sub-events from different detectors processed by feature extraction algorithms; c). selected sub-events further correlated with each other and d). building events; e). filtering events and rejecting some uninteresting ones to reduce the data storage requirement; f). selected results constructed into packets by TCP/IP stack and g). packets transmitted over Ethernet for later analysis or storage. Throughout this procedure, data go along the path in the format from sub-event to event. One sub-event is the specific data from one type of all detectors [4]: *RICH*, *MDC*, *TOF* and *Shower detector*, etc.. After their *correlation*, sub-events are combined into events according to their physics tags and with a header attached, as shown in figure 4. Coming back to the criteria described before, all the performance- and computing-intensive subtasks will be implemented in hardware, except the TCP/IP stack. Although it is quite important and a hardwired stack could improve the communication bandwidth a lot [6], dedicating the embedded CPU to protocol processing is a both human-resource-efficient and time-efficient way in our application. Other control subtasks and lightweight processing ones are also expected in software, specifically the embedded Linux OS and application programs.

Now that hardware computing engines and their interface logics need to be customized, FPGA is definitely a suitable platform for development. In addition, as the embedded Linux is expected, we need also a comparatively powerful and well-supported CPU which runs the OS and applications. Thus we choose Xilinx Virtex-4 FX platform FPGA which provides us a good combination of both resources.

[1] What we plan to use is UDP/IP actually, which is part of the TCP/IP protocol suite.

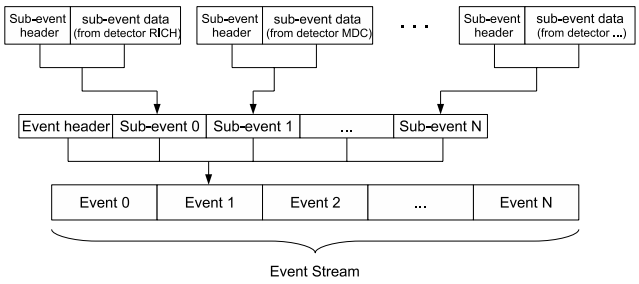

Figure 4. Combining sub-events from various detectors into events.

Inside the FPGA chip, an embedded PowerPC 405 processor core is integrated altogether with many other peripherals such as Gigabit Ethernet MAC and RocketIO multi-gigabit serial transceivers. Linux could run on top of the embedded PowerPC which is easily connected and communicate with other modules inside the FPGA fabric.

4. Hardware Design

Since most peripheral controllers could be integrated in Xilinx Virtex-4 FX FPGA in the format of either hard or soft cores, plus the embedded CPU and bus architecture our design could be treated as a System-on-an-FPGA-chip in some sense. Figure 5 illustrates the block diagram of our development platform.

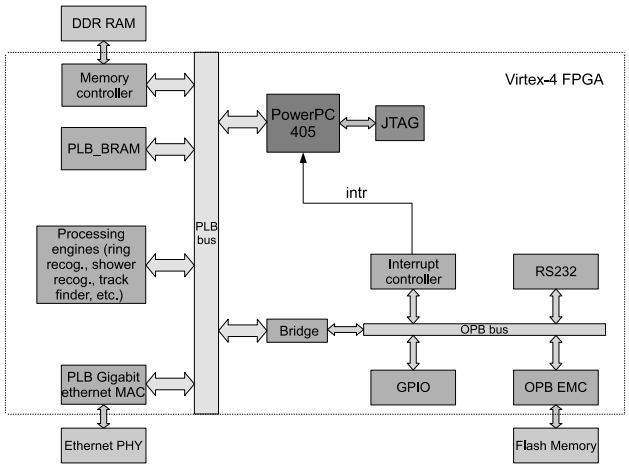

Figure 5. System architecture.

In this design, PowerPC 405 CPU core runs the operating system and applications to manage all other peripheral components. In addition to the Gigabit Ethernet MAC and DDR memory controller, a fraction of partitioned feature extraction engines which might be *ring recognizers*, *track finder*, *event builder*, or even *event selector* is connected with the CPU via a fast-speed PLB bus. Data streams will be guided to corresponding algorithms for processing according to their physical signatures. After the stages of *feature extraction*, *correlation*, *event building* and *event selection* in the internal network, final results will be induced out of Ethernet ports for next level analysis and storage.

We have already designed and implemented a general *event selector* whose use is to filter uninteresting events by event structure checking. Whether a particular event is good or not is decided by looking into sub-events and judging the comprehensive information for all detectors. Currently the design is something like a "toy model", and later more delicate regulations will be added for more careful event selection. As shown in figure 6, this processing module contains a PLB IP interface (IPIF) which communicate with other components via PLB. To release the CPU from the work of moving data back and forth between the memory and the IP, DMA and interrupt are included in the IPIF. DMA takes care of feeding event data to the Write FIFO (WrFIFO) and collecting results from Read FIFO (RdFIFO). After each DMA transfer, the CPU will be noticed by interrupt. In the selector IP core, an event buffer is dedicated to temporarily store an entire event which is under inspection and a complicate FSM controls to analyze the event structure consisting of sub-events from different detectors (refer to figure 4). It is necessary to buffer the whole event since the decision whether this event is interesting or not cannot be made before all its sub-events have already arrived at the buffer and been inspected.

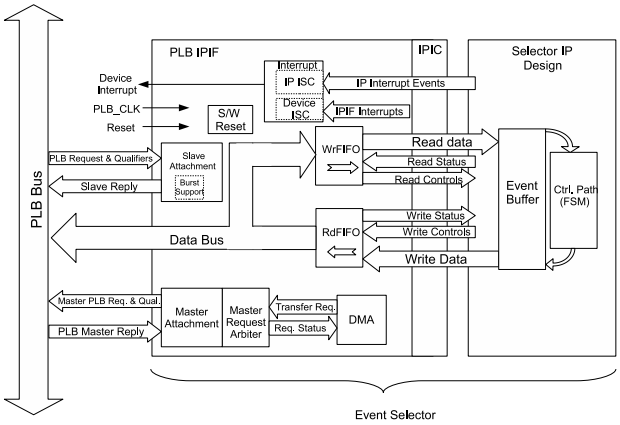

Figure 6. Structure of the event selector.

Also there are some slow peripherals connected to the OPB bus which is bridged with PLB. For instance RS232

provides the console interface to communicate with the embedded OS. Moreover, an External Memory Controller (EMC) enables the PowerPC to address the flash memory space. Details on how to update the flash content with the running Linux will be described in the next section.

5. Software Design

5.1. Embedded Operating System

We have selected an open-source Linux kennel from the mainline kernel tree [7] [8] rather than any commercial distribution for saving cost. In order to get a better support from the Linux community, the latest 2.6 version deserves a higher priority than the old 2.4 one for our choice. How to set up the cross-compilation environment, configure and install Linux on Xilinx boards was described in [9] [10] and [11] in detail. In our case, we have the kernel configuration including device drivers for peripherals, loadable module support, network file system (NFS) [12] support, and many networking options, etc.. In cooperation with the hardware bitstream and the bootloader, the embedded Linux could be booted and operated via a console interface much like its relatives on PCs except that there is no GUI supported yet.

5.2. Device Drivers

Some of the device drivers especially those popular ones such as Ethernet, RS232 and MTD devices, have already been included in the kernel's package. They could be easily enabled by selecting the relevant options during the kernel's configuration. For our customized hardware modules in the system, drivers are also necessary to access them in Linux. Let's consider the *event selector* example, which is a typical character device [13] since it is fed with a stream of event data and outputs interesting candidates. In its driver, common file operations were implemented, for instance "*open*", "*close*", "*read*", "*write*". As well DMA transfer initiation and interrupt handlers were included to support full hardware features and run the device efficiently. Using the device driver programmers could address the hardware with application programs via the entry in "*/dev*" directory.

5.3. Application Programs

With the operating system's support, application programs are quite flexible. They might be programmed either in C/C++ or in high level scripts. As an example, the application program set was written in C, which has multiple processes respectively to buffer and feed the incoming sub-events to the processing engines, collect results and send to the next level. On the other hand, it is possible to upgrade the hardware design and the Linux kernel using a bash script, where "*dd*" commands copy the new bitstream or kernel image and overwrite the old ones in flash memories. After that a system reset is scheduled and then the new hardware and OS will be loaded and run in cooperation. With the network support, the upgrading procedure may be initiated in a remotely telneted shell or by a network command packet. This feature protects operators away from the radiative area and upgrading the system manually.

6. Implementation and Performance Measurements

6.1. Implementation Results

Till now the development and design are being executed on Xilinx Dev. board ML403 [14], whose heart is a Xilinx Virtex-4 FX12 platform FPGA with a PowerPC 405 integrated. Figure 7 shows the interconnections of all useful components on-board for our platform. As we said before, most components including the CPU, buses and peripheral controllers are embedded in the FPGA chip. Thus we have put emphasis on the system implementation in FPGA. XPS 8.2 was being used to describe our system and ISE 8.2 to implement it. We also used ModelSim 6.1e and Xilinx Bus Functional Models (BFM) [15] to simulate the bus behaviors of our PLB customized IPs. Table 1 shows both statistics on the resource utilization and percentages out of Virtex-4 FX12. From the listed numbers we can see that Ethernet and the computing engine consume 48% and 43% respectively of the LUT resource in FX12. Then if both fully functional components are desired in the platform, we have to turn to a larger chip, like FX60 that we plan to adopt in the real product.

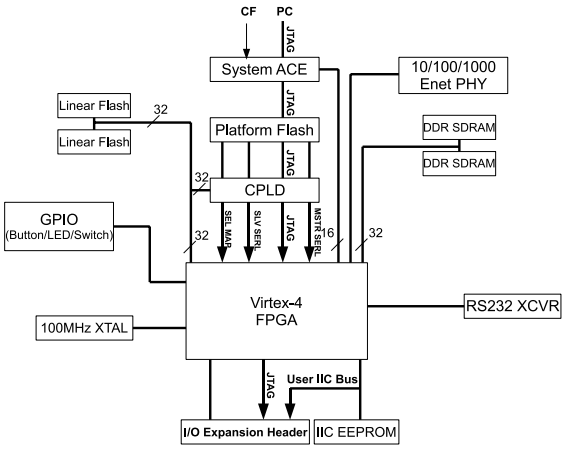

Figure 7. Development system on ML403.

Resources	System without computing engine	Tri-mode Ethernet	*Event Selector* module (4kBytes WrFIFO RdFIFO and Event Buffer)
4-input LUTs	8531 out of 10944 (77%)	5346 out of 10944 (48%)	4674 out of 10944 (43%)
Slice Flip-Flops	5724 out of 10944 (52%)	4093 out of 10944 (37%)	2830 out of 10944 (25%)
FIFO16/ RAMB16s	18 out of 36 (50%)	18 out of 36 (50%)	6 out of 36 (17%)
DSP48s	8 out of 32 (25%)	8 out of 32 (25%)	0
DCMs	3 out of 4 (75%)	0	0

Table 1. Resource consumption.

The timing summary shows that the PLB bus could run at above 100MHz and the PowerPC CPU at 300MHz.

6.2. Performance Measurements

The performance of our platform could be represented by two respects: communication bandwidth and computing capability; Latency is not a big problem in our application. Since Gigabit Ethernet links are external channels to convey processing decisions to high level analysis and storage, they stand for half of the external communication performance of the proposed computation network directly. We measured the maximum network throughput with a Point-to-Point connection between a development board and a powerful enough PC with a Gigabit Ethernet card. On both PC and board runs a benchmark software "Netperf" [16] to measure the end-to-end bulk data transfer capacity. With all hardware/software supported features enabled which improve performance, explicitly Scatter/Gatter DMA, checksum offloading, data realignment engines, interrupt coalescing, jumbo frame of 8982, etc., we could achieve a maximum throughput of around 400 Mbps for UDP transmitting and receiving. Details are shown in table 2, as well TCP transfers. Although not so elegant as what a desktop PC could provide, these results are quite reasonable for an embedded application taking into account the weak embedded CPU which runs only at 300MHz and the overhead added by the soft TCP/IP stack. In the procedure of searching for the bottleneck which restricts the performance lower than the physical gigabit limitation, we found that the CPU utilization was almost one hundred percent during transmitting and receiving. Actually the results match the famous thumb rule of network communication quite well: driving a line at 1 bps requires a 1 Hz processor. So we conclude that it is the CPU processing capability that decides the Ethernet throughput in our application. In [17] [18] [19] [20] there are details to analyze the problem and provide methods to improve the performance.

Protocol Type	Direction	Max. Throughput (Mbps)
UDP/IP	Board → PC	394.5 (TX)
UDP/IP	PC → Board	$\geq$ 394.5[1] (RX)
TCP/IP	Board → PC	297.8
TCP/IP	PC → Board	316.6

Table 2. Ethernet maximum throughput.

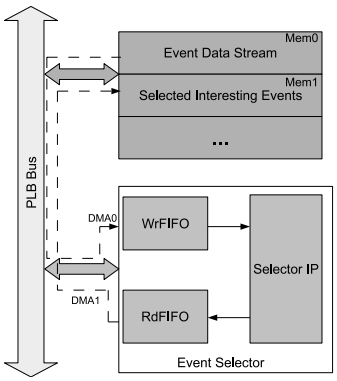

Figure 8. Computing throughput measurements of the event selector.

The other important factor for performance is the processing capability of the hardware computing engine. We still discuss the *event selector*, and the measurement could be done in the following method. As shown in figure 8, we reserve two large blocks in the DDR memory to buffer the incoming event stream and the selected interesting results respectively. Through DMA0 events to be processed could be fed from Mem0 into WrFIFO of the selector, and results will be collected from RdFIFO to Mem1 by DMA1 operations. With continuous event stream buffered in Mem0, the computing throughput of the selector module could be calculated by the division of data size and spent time. After all measurements and calculations at different event selection rates (100% and 25% interesting event rates respectively), processing capabilities are shown in figure 9 as functions of DMA transfer sizes which are equal to both WrFIFO and RdFIFO sizes to maximize the transfer at each time. We draw the conclusion from the results that a lower event selection rate could increase the processing capabilities of the computing engine. It is due to the fact that lower selection rate decreases DMA1 transfer times for collecting

[1] We didn't get an exact number for the maximum receiving speed. However the board could successfully receive all packets sent at the speed of 394.5 Mbps. So its receiving capability should be no less than 394.5 Mbps.

interesting events. From the figure we can see that with a WrFIFO and RdFIFO size of 32kBytes, the top processing power of 148.1 and 97.3 MBytes/s could be achieved at the interesting event rates of 25% and 100% respectively. In the real experimental conditions where the selection rate will be quite low, the computing throughput might be even higher.

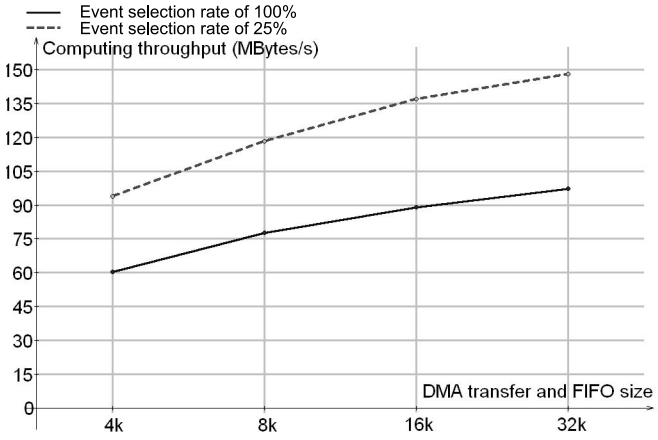

Figure 9. Computing throughput vs. DMA transfer and FIFO size @ different event selection rates.

7. Conclusion and Future Work

We have presented a hardware/software co-design case study of the computation network in experimental particle physics. With a reasonable partitioning strategy, we put the performance- and computing-intensive subtasks in hardware designs while TCP/IP stack and other lightweight ones in Software. In the hardware design section, a computer architecture was demonstrated which includes customized hardware computing engines connected to the PLB bus. Specifically an *event selector* design was described from the aspects of its structure and working mechanism as an example of various algorithms. The software design is based on the Linux OS with matching device drivers for hardware modules. With the support from the OS and device drivers, the platform could be run flexibly by application programs in C/C++ or high level scripts. Finally the implementation results show the resource utilization, and performance measurements indicate the system's communication capacity of 400 Mbps UDP/IP transfers per Ethernet link and up to 148.1 MBytes/s for event selection at an interesting event rate of 25%.

In the future, various feature extraction algorithms will be implemented and optimized in FPGA. Much work is also to be devoted in the research of the high-speed internal network, which connects all the nodes and correlates the results from different algorithms.

Acknowledgement

This work was supported in part by BMBF under contract Nos. 6GI179 and 6G180.

References

[1] High Acceptance Di-Electron Spectrometer @ GSI, Darmstadt, Germany, www-hades.gsi.de.
[2] antiProton ANnihilations at DArmstadt @ GSI, Darmstadt, Germany, www.gsi.de/panda.
[3] BEijing Spectrometer @ Institute of High Energy Physics, Beijing, China, http://bes.ihep.ac.cn/bes3/index.html.
[4] Michael Traxler, Real-Time Dilepton Selection for the HADES Spectrometer, November 2001, Ph.D thesis, II. Physics Institute of Justus-Liebig-University Giessen.
[5] PANDA Collaboration, Technical Progress Report for PANDA, http://www-panda.gsi.de/db/papersDB/PC19-050217_panda_tpr.pdf, February 2005.
[6] Joe Rash, A case for hard-wired TCP/IP offload, *Network systems Designline*, February, 2007.
[7] www.kernel.org.
[8] www.penguinppc.org.
[9] Brent Nelson and Brad Baillio, Configuring and Installing Linux on Xilinx FPGA Boards, November, 2005, BYU Configurable Computing Laboratory.
[10] Dan Burke, James Player, and Nacho Navarro, Building a Xilinx ML300/ML310 Linux Kernel, November, 2004, UIUC Soft Systems Lab.
[11] Wolfgang Klingauf and Uwe Klingauf, Virtex2Pro & Linux, January, 2004, www.klingauf.de.
[12] Hal Stern, Mike Eisler, and Ricardo Labiaga, *Managing NFS and NIS (Second Edition)*, O'REILLY & Associates, Inc., June, 2001.
[13] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman, *Linux Device Drivers (Third Edition)*, O'REILLY & Associates, Inc., February, 2005.
[14] Xilinx, Inc., ML401/ML402/ML403 Evaluation Platform User Guide, UG080(v2.3), January 13, 2006.
[15] Xilinx Inc., BFM Simulation in Platform Studio, November 11, 2005.
[16] www.netperf.org.
[17] Xilinx Inc., Getting the most TCP/IP from your Embedded Processor.
[18] J. Chase, A. Gallatin, and K. Yocum, End-System Optimizations for High-Speed TCP, *Communications Magazine*, IEEE, Apr 2001, Volume: 39, Issue: 4 pp. 68-74.
[19] A. Gallatin, J. Chase, and K. Yocum, Trapeze/IP: TCP/IP at near-gigabit speeds, *In Proc. 1999 USENIX Technical Conference*, Freenix Track, 1999.
[20] A. P. Foong , T. R. Huff , H. H. Hum , J. R. Patwardhan , G. J. Regnier, TCP performance re-visited, *Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software*, pp.70-79, March 06-08, 2003.