A High-end Reconfigurable Computation Platform for Particle Physics Experiments

Ming Liu

Stockholm 2008

Thesis submitted to the Royal Institute of Technology in partial fulfillment of the requirements for the degree of Licentiate of Technology

Liu, Ming
A High-end Reconfigurable Computation Platform for Particle Physics Experiments

ISBN 978-91-7415-145-9
TRITA-ICT/ECS AVH 08:08
ISSN 1653-6363
ISRN KTH/ICT/ECS AVH-08/08--SE

Royal Institute of Technology
School of Information and Communication Technology
Department of Electronic, Computer, and Software Systems
Forum 120
SE-164 40 Kista, Sweden

Abstract

Modern nuclear and particle physics experiments run at a very high reaction rate and are able to deliver a data rate of up to hundred GBytes/s. This data rate is far beyond the storage and off-line analysis capability. Fortunately physicists have only interest in a very small proportion among the huge amounts of data. Therefore in order to select the interesting data and reject the background by sophisticated pattern recognition processing, it is essential to realize an efficient data acquisition and trigger system which results in a reduced data rate by several orders of magnitude. Motivated by the requirements from multiple experiment applications, we are developing a high-end reconfigurable computation platform for data acquisition and triggering. The system consists of a scalable number of compute nodes, which are fully interconnected by high-speed communication channels. Each compute node features 5 Xilinx Virtex-4 FX60 FPGAs and up to 10 GBytes DDR2 memory. A hardware/software co-design approach is proposed to develop custom applications on the platform, partitioning performance-critical calculation to the FPGA hardware fabric while leaving flexible and slow controls to the embedded CPU plus the operating system. The system is expected to be high-performance and general-purpose for various applications especially in the physics experiment domain.

As a case study, the particle track reconstruction algorithm for HADES has been developed and implemented on the computation platform in the format of processing engines. The Tracking Processing Unit (TPU) recognizes peak bins on the projection plane and reconstructs particle tracks in realtime. Implementation results demonstrate its acceptable resource utilization and the feasibility to implement the module together with the system design on the FPGA. Experimental results show that the online track reconstruction computation achieves 10.8 - 24.3 times performance acceleration per TPU module when compared to the software solution on a Xeon 2.4 GHz commodity server.

Acknowledgments

This project has been accomplished under the collaboration between Department of Electronic, Computer and Software Systems (ECS) of Royal Institute of Technology (KTH) in Stockholm, Sweden, II. Physics Institute of Justus-Liebig-University (JLU) in Giessen, Germany, and Institute of High Energy Physics (IHEP) of Chinese Academy of Sciences in Beijing, China. It was supported by grants of the BMBF: 06GI179, 06GI180.

Most of all, I would like to thank my supervisors Professor Axel Jantsch, Professor Wolfgang Kühn and Dr. Zhonghai Lu for providing me the opportunity to do such an interesting interdisciplinary work. Professor Jantsch is a very respectable person for his personality, knowledge and inspiration to students. I still clearly remember his words when I was enrolled as a Ph.D student: "Ph.D study is exactly like the sailing Columbus on the sea. You can never know which new land you will arrive at unless you go ahead and try to search for." Professor Kühn is my local supervisor. I learned from him different cultures in physics and also different methods to solve problems. His broad knowledge in physics, computer and electronics area impresses me very much. Dr. Lu is my direct supervisor who gave me most instructions. He devoted quite much time on my study and work in the last two years. I give my great appreciation to him for his fruitful advice on the process of problem targeting, solution proposal, experimental setup and scientific writing.

I am also thankful to all my colleagues in Stockholm, Giessen, and Beijing. The discussion and suggestion from them are so helpful to improve my professional knowledge and technical skills. My Giessen colleagues, specifically Sören Lange, Vladimir Pechenov, Geydar Agakishiev, Olga Pechenova, Marco Destefanis, Stefano Spataro, Daniel Kirschner, Camilla Kirchhübel, Johannes Roskoss, Andreas Kopp, Johannes Lang, Zoltán Gagyi-Pálfy, Tiago Perez and so on, explained me plenty of physics background knowledge which makes me understand the application very clearly. I also thank Ingo Sander,

Johnny Öberg, and Vladimir Vlassov for their interesting lectures and discussion on the modern techniques, as well as Huimin She, Geng Yang, Xiaolong Yuan, Zhuo Zou, Peng Wang and Jiayi Zhang for exchanging our experience of graduate study. Many thanks are given to the Chinese group including Zhen'an Liu, Hao Xu, Qiang Wang, Dapeng Jin for their nice work on the compute node PCB design.

I appreciate Christa Momberger, Thomas Köster, Lena Beronius, Agneta Herling for their administrative and non-technical assistance in traveling, device ordering, and other issues.

Many thanks to my parents and relatives who are far away in China. Their continuous encouragement and support gave me strength to overcome difficulties in all aspects.

Special thanks go to my girlfriend Shuo Yang who provided her full support to my study. I will forever remember the days we spent altogether, busily but happily.

Ming Liu

September 2008, Giessen

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
ATCA	Advanced Telecommunications Computing Architecture
BSP	Board Support Package
CN	Compute Node
CPLD	Complex Programmable Logic Device
DAQ	Data Acquisition
DDR	Double Data Rate
DSP	Digital Signal Processor
EDA	Electronic Design Automation
EDK	Embedded Development Kit
EMC	External Memory Controller
FEE	Front-End Electronics
FIFO	First-In-First-Out
FPGA	Field Programmable Gate Array
FRU	Field Replaceable Unit
GUI	Graphical User Interface
HDTV	High Definition Television
HW	Hardware
IP	Intelligent Property
IPMC	Intelligent Platform Management Controller
IPU	Image Processing Unit
ISE	Integrated Software Environment
ISR	Interrupt Service Routine
JTAG	Joint Test Action Group
LUT	Look-Up Table
MAC	Media Access Control
MDC	Mini Drift Chamber
MOPS	Mega Operations per Second
MGT	Multi-Gigabit Transceiver

MMU	Memory Management Unit
MPMC	Multi-Port Memory Controller
NFS	Network File System
OS	Operating System
P2P	Point-to-Point
PCB	Printed Circuit Board
PE	Processing Element
RdFIFO	Read First-In-First-Out
RICH	Ring Imaging CHerenkov
RPC	Resistive Plate Chambers
RTL	Register Transfer Level
SADC	Sampling Analog-to-Digital Converter
SDRAM	Synchronous Dynamic Random Access Memory
SRAM	Static Random Access Memory
SW	Software
TOF	Time-Of-Flight
TPU	Tracking Processing Unit
WrFIFO	Write First-In-First-Out

Chapter 1

Introduction

1.1 Background in Particle Physics Experiments

Particle physics is a branch of physics that studies the elementary constituents of matter and the interactions between them. It is also called high energy physics because many elementary particles do not occur under normal circumstances in nature, but can be created and detected during energetic collisions of other particles as is done in particle colliders. In particle physics experiments, beam particles are accelerated to the velocity approaching to the speed of light and then collide with target particles. A huge and complex detector system is constructed to inspect the characteristics of produced particles from the collision, including energy, momentum, mass, charge, etc.. In this domain, an "event" normally describes the result of a single reaction between a projectile particle and a target particle. Typically an "event" consists of "sub-events" referring to the activities of different detectors recording reaction products. Figure 1.1 shows an example event structure which consists of sub-events from various detectors such as Ring Imaging Cherenkov (RICH) detector, Mini Drift Chamber (MDC) detector, Time-Of-Flight (TOF) detector, Shower detector, and so on. The event data generated by detectors are to be recorded during experiments and afterwards physicists will extensively analyze the events to search for interesting ones, such as new types of particles.

Modern nuclear and particle physics experiments, for example HADES [1] and PANDA [2] at GSI, BESIII [3] at IHEP, ATLAS, CMS, LHCb, ALICE at the LHC [4] at CERN, WASA [5] at FZ-Juelich, 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 or even higher (PANDA, up to 200 GBytes/s). Compared to the bandwidth needed by some other applications [6], such as H.264, HDTV, etc., Figure 1.2 lists some experiments by their event sizes and reaction rates. Their data rate, which is the product of the event size and the reaction rate, ranges from 10^7 up to more than 10^{11} Bytes/s. The huge amount of data cannot be entirely recorded in the disk or tape storage medium. Furthermore the supercomputers will take forever to process all the data during offline analysis.

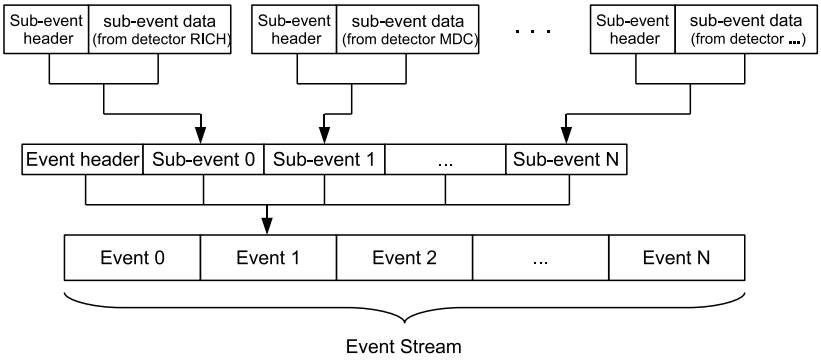


Figure 1.1. Combining sub-events from detectors into events

Fortunately, physicists have interest only in a very small proportion of all the event data. It might occur only once within one million interactions. Hence it provides the possibility to utilize an efficient online Data Acquisition (DAQ) and Trigger system to reject uninteresting events while identify and retain interesting ones in realtime. Depending on the physics focus of the experiment, sophisticated realtime pattern recognition algorithms such as *Cherenkov ring recognition*, *particle track reconstruction*, *Time-Of-Flight processing*, *Shower recognition* [7] [8] [9] and *high level correlations* are implemented for recognizing interesting data. Only the events which meet expected patterns and correlations receive a positive decision and will be forwarded to the mass storage for later offline analysis. Others are discarded on the fly. As a result, the data rate can be reduced by several orders of magnitude and to a reasonable level for storage. Figure 1.3 shows the data flow in the DAQ and trigger system, with the arrow width representing the consecutively reduced data rate.

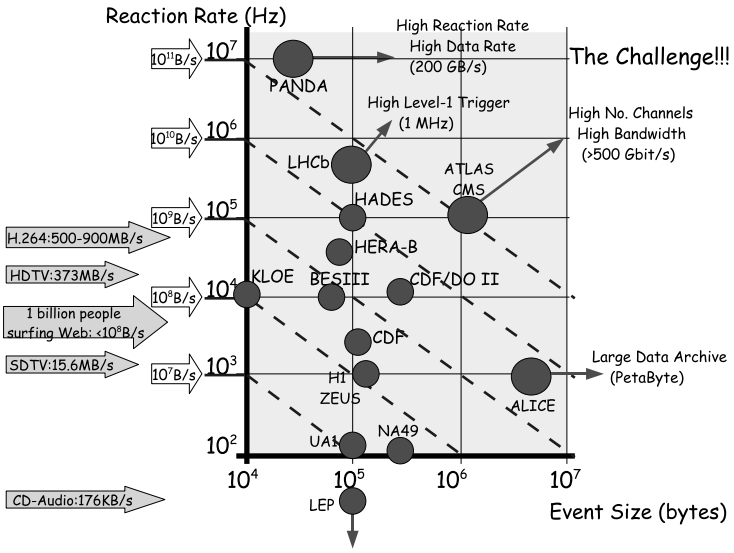


Figure 1.2. Some particle physics experiments with different event sizes and reaction rates

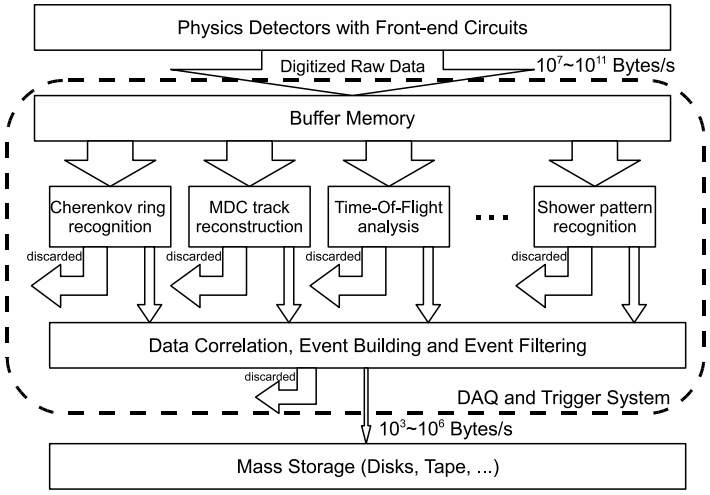


Figure 1.3. Data flow and consecutively reduced data rate in the DAQ and trigger system

1.2 Reconfigurable Computing

Reconfigurable computing is a computing paradigm combining both the flexibility of software and the high performance of hardware by using programmable computing fabrics like FPGAs. The principle difference compared to the ordinary microprocessor computing is the capability to make substantial changes to the datapath in addition to the control flow. Actually the basic concept can be traced back to the 1960s, when Gerald Estrin's landmark paper proposed the concept of a computer made of a standard processor and an array of "reconfigurable" hardware [10] [11]. The main processor controls the behavior of the reconfigurable hardware. The latter can be tailored to perform a specific task as quickly as a dedicated piece of hardware. Unfortunately this idea was far ahead of its time in needed electronics technology. Only after the large development of configurable devices especially FPGAs/PLDs and corresponding EDA tools in the recent decade, reconfigurable computing could be really widely adopted to achieve performance benefits and flexible re-programmability.

Although normally running at a much lower clock frequency, FPGA-based reconfigurable computing is believed to have a 10-100X accelerated performance but far lower power consumption compared to CPUs. The development methodology on the conventional PC clusters focuses on flexible control flows with limited number of Processing Elements (PE), while more concerns are placed on the datapath design and optimization through parallel and pipelined approaches, when developing hardware reconfigurable platforms. The on-chip memory concurrency and fine-grained computation parallelism can overcome the bottleneck in the computer memory system. More efficiently utilizing the expensive chip area, reconfigurable computing creates an unprecedented opportunity for orders of magnitude improvement in MOPS/dollar, MOPS/watt, and just MOPS.

In physics experiment applications, FPGA-based reconfigurable solutions have important advantages to implement the pattern recognition algorithms. We have comparatively simple control flows during data processing, and the application-specific datapath design can result in high performance with the on-chip memory concurrency and fine-grained computation parallelism and/or pipeline support. In addition, the re-programmability enables to change the designs to satisfy different experimental requirements.

1.3 Related Work

Traditionally modular approaches with commercial bus systems, such as VMEbus, FASTbus, and CAMAC, were utilized to construct the DAQ and trigger system for high-energy physics experiments [12][13][14][15]. These bus-based systems can be interfaced to commodity computers, typically high-end workstations at that time. However due to the largely increased data rate generated by the detector systems, these deprecated technologies cannot meet the current experimental requirements any longer. The time-multiplexing nature of the system bus not only exacerbates the data exchange efficiency among algorithms residing on different pluggable modules, but also restricts the flexibility to partition complicated algorithms. Nowadays the networking and switching technologies make it efficient to construct large-scale systems for parallel and pipelined processing. In addition, as the tremendous development on re-programmable devices especially FPGAs, it provides the practicability to migrate some complex algorithms, which were conventionally implemented as software on desktop computers or embedded processors/DSPs, into the FPGA fabric for high-performance hardware processing.

Reconfigurable computing satisfies the simultaneous demands of application performance and flexibility. In the present era when cluster-based supercomputers still dominate the fields of super computation tasks, reconfigurable computing begins showing large potential and perspective on some performance-critical areas such as realtime scientific computing. Currently many commercial and academic projects are developing hardware and software systems to employ the raw computational power of FPGAs. One commercial example is the latest products from Cray Inc., such as XD1, XT4, XT5 series supercomputers. FPGAs are integrated in the system to embody various digital hardware designs and augment the processing capabilities of the AMD Opteron processors [16][17][18]. However most of these projects are augmented computer clusters with FPGAs attached to the system bus as accelerators. One major weakness of such systems is the bandwidth bottleneck between the microprocessor and the FPGA accelerator. An instance in physics experiment applications is the ATLAS level 2 trigger [19]. Their design appears as PCI cards in commodity PCs, in which only those simple but computing-intensive steps are released to FPGAs while others remain on CPUs. In that case, the computation work relies much on the PC and the limited bandwidth between CPUs and FPGAs via the PCI bus becomes the bottleneck when partitioning the algorithm steps between

the CPU and the FPGA. Other standalone platforms for example the Dini Group products [20], target mainly the hardware emulation applications. It is not straightforward to upgrade the system to a supercomputer equivalent scale due to the lack of efficient communication standards for inter-board connectivity. The Berkeley Emulation Engine 2 (BEE2) did provide a good platform which is powerful and scalable for large scale data processing [21][22]. However the external links use only Infiniband, which results in an inflexible and expensive interface to the detector front-end circuits and PC clusters in physics experiments. Moreover its all-board-switched or tree-like topology may cause large communication latency and throughput penalty when complex algorithms are partitioned and span over multiple boards.

1.4 Motivation

Motivated by multiple high-energy physics experiment projects, for instance the HADES and BESIII upgrade and the PANDA construction, we propose a high-end reconfigurable, scalable, and general-purpose computation platform as the solution. The system is entirely built with the commercial off-the-shelf components. Cutting-edge FPGA technologies as well as high-speed communications are adopted to guarantee both high processing capability and high channel bandwidth. Easy scalability is an important feature of the platform. The dynamically and remotely configurable properties have also been considered to overcome temporal and spatial limitations. To standardize the application development on the platform, we propose a hardware/software co-design approach, with which functional tasks are partitioned between embedded microprocessors and modular FPGA cores. Thus the system design could be largely re-utilized for various experimental facilities with little performance penalty or modification effort.

1.5 Thesis Outline and Author's Contributions

The thesis is constructed as follows:

- Chapter 1: Background introduction. The targeted application of high-energy physics experiments is first introduced. We describe the functionalities and requirements of a powerful data acquisition and trigger system, which processes massive data with pattern recognition

algorithms in nuclear and particle physics experiments. After that the scheme of this thesis project, reconfigurable computing, is briefly addressed. Some related work in the domain of reconfigurable computer design is discussed then. We point out a few disadvantages of those designs in the physics experiment scenario and propose the motivation to build our own computation platform.

- Chapter 2: We present the computation platform architecture in a top-down approach. First the global computation network architecture is described with the focus on the hierarchical interconnections in the system. Second we come to the compute node design, which appears as elementary Field Replaceable Units (FRU) in the ATCA crates. The main contributions of this chapter have been published in paper 1, 4, 5 and poster 8 listed below.

- Chapter 3: Corresponding to the platform design, we propose a hardware/software co-design approach as the standard method to develop applications on our computation platform. Versatile control tasks will be allocated to the embedded microprocessors as application programs, while performance-critical data processing is implemented in the FPGA fabric in the format of hardware processing modules. These application-specific modules are customized and integrated in the universal bus-based or Multi-Port Memory Controller (MPMC) based system design. To facilitate the development of software applications, an embedded Linux operating system is ported to the PowerPC architecture in the FPGA. The device drivers provide the interface with which the embedded CPU manages all peripherals in the design. The main contributions concerning to the HW/SW co-design can be referred to in the published papers 1 and 3 listed below.

- Chapter 4: In this chapter, one physics application for the HADES upgrade project is implemented and evaluated on the platform as a case study. The Tracking Processing Unit (TPU) is integrated in the system design to recognize track candidates and reconstruct particle tracks in realtime. The modular structure is presented in detail, and from the experimental results we see its acceptable resource utilization, as well as the performance speedup compared to the software solution on commodity PCs. The contributions of the algorithm implementation on FPGAs can be referred to in paper 2 and poster 6 and 7 listed below.

- Chapter 5: Summary of the thesis. Conclusions are drawn and future work is proposed in this chapter.

This thesis is mainly based on the following publications:

Papers:

1. Ming Liu, Johannes Lang, Shuo Yang, Tiago Perez, Wolfgang Kuehn, Hao Xu, Dapeng Jin, Qiang Wang, Lu Li, Zhenan Liu, Zhonghai Lu, and Axel Jantsch, "ATCA-based Computation Platform for Data Acquisition and Triggering in Particle Physics Experiments", In Proceedings of the International Conference on Field Programmable Logic and Applications 2008 (FPL'08), Heidelberg, Germany, Sep. 2008.

   Introduction: The paper presents both the ATCA-based computation platform architecture and the compute node design for data acquisition and triggering purpose in particle physics experiments. The FPGA development approach and some experimental results are also described and demonstrated in the paper.

   Author's contribution: The author established the network model of interconnected nodes, developed applications on the FPGA with a HW/SW co-design approach, and did experiments to evaluate the performance of the compute node.

2. Ming Liu, Wolfgang Kuehn, Zhonghai Lu and Axel Jantsch "System-on-an-FPGA Design for Real-time Particle Track Recognition and Reconstruction in Physics Experiments", In Proceedings of the 11th EUROMICRO Conference on Digital System Design (DSD'08), Parma, Italy, Sep. 2008.

   Introduction: The paper demonstrates a hardware implementation of the particle track recognition and reconstruction algorithm for the HADES [1] experiment. The modular design structure is described in the paper, and experimental results of FPGA resource utilization as well as performance measurements are presented.

   Author's contribution: The author implemented the algorithm in hardware and performed the evaluation experiments.

3. Ming Liu, Wolfgang Kuehn, Zhonghai Lu, Axel Jantsch, Shuo Yang, Tiago Perez and Zhenan Liu, "Hardware/Software Co-design of a

General-Purpose Computation Platform in Particle Physics", In Proceedings of the IEEE International Conference on Field Programmable Technology 2007 (ICFPT'07), Kitakyushu, Kokurakita, Japan, Dec. 2007.

Introduction: The paper describes the HW/SW co-design of the system-on-an-FPGA on the computation platform. The hardware architecture is presented, with which customized algorithm processors are integrated in the system for application-specific processing. The software development for slow control tasks based on the embedded Linux OS is also introduced.

Author's contribution: The author implemented the hardware system design and brought up the embedded Linux platform for software application development.

4. Wolfgang Kuehn, Ming Liu, et al., "FPGA-Based Compute Nodes for the PANDA - Experiment at FAIR", 15th IEEE-NPSS Real Time Conference 2007, Fermilab, Batavia, IL, USA, Apr. 2007.

   Introduction: The paper introduces the FPGA-based compute node design for the future PANDA [2] experiment.

   Author's contribution: The author was engaged in the FPGA interconnection model design of the compute node.

5. Tiago Perez, Camilla Gilardi, Ming Liu and Shuo Yang, "A FPGA-based Compute Node for the PANDA Data Acquisition and Trigger System", In Proceedings of the XLV International Winter Meeting on Nuclear Physics, Bormio, Italy, Apr. 2007.

   Introduction: The paper presents the hierarchical architecture of the PANDA data acquisition and trigger system. The compute node design is proposed to be utilized to construct a powerful network for high-performance data acquisition and triggering computation.

   Author's contribution: The author participated in the compute node based architecture design of the data acquisition and trigger system for the PANDA project.

Posters:

6. Ming Liu, Wolfgang Kuehn, Tiago Perez, Vladimir Pechenov, and Shuo Yang, "Implementation of an FPGA-based MDC Track Reconstruction Algorithm for the HADES Upgrade", German Physics Society 2008 Meeting (DPG'08), Darmstadt, Germany, Mar. 2008. (Best Awards)

7. Johannes Roskoss, Daniel Kirschner, Andreas Kopp, Ming Liu, and Shuo Yang, "New Event-building and Trigger Algorithms for the HADES DAQ Upgrade", German Physics Society 2008 Meeting (DPG'08), Darmstadt, Germany, Mar. 2008.

8. Tiago Perez, Daniel Kirschner, Wolfgang Kuehn, Soeren Lange, Ming Liu, Zhenan Liu, "An FPGA Compute Node for the PANDA DAQ", German Physics Society 2007 Meeting (DPG'07), Giessen, Germany, Mar. 2007.

In addition, a paper not included in this thesis, is:

9. Zhonghai Lu, Ming Liu and Axel Jantsch, "Layered Switching for Networks on Chip", In Proceedings of the 44th Design Automation Conference (DAC'07), San Diego, CA, USA, Jun. 2007.

Chapter 2

Computation Platform Architecture

To manage the large data rate from detectors, we construct a hierarchical network architecture which consists of multiple interconnected Compute Nodes (CN). The connectivity is sorted in two categories: external connections and internal ones. The external channels are used to communicate with detectors and the PC farm, to receive raw data for processing and forward selected results for storage and offline analysis. Specifically they provide optical links and Ethernet interfaces. The internal connections bridge all algorithms or algorithm steps for parallel/pipelined processing. Both on-board I/O channels and the inter-board backplane interface function as internal links. The details of the system architecture will be introduced in the following sections from the interconnected network top-down to the node design.

The topics of this chapter cover paper 1, 4, 5 and poster 8 listed in Section 1.5.

2.1 Global Computation Network

Figure 2.1 [23] shows the DAQ and trigger system for the PANDA project, as an instance of modern particle physics experiments. It demonstrates the overall steps in which detector data are processed and analyzed. Physics detectors generate analog signals for detected particle reactions. This information is converted into digital signals by Sampling Analog-to-Digital Converters (SADC) in Front-End Electronic (FEE) circuits mounted

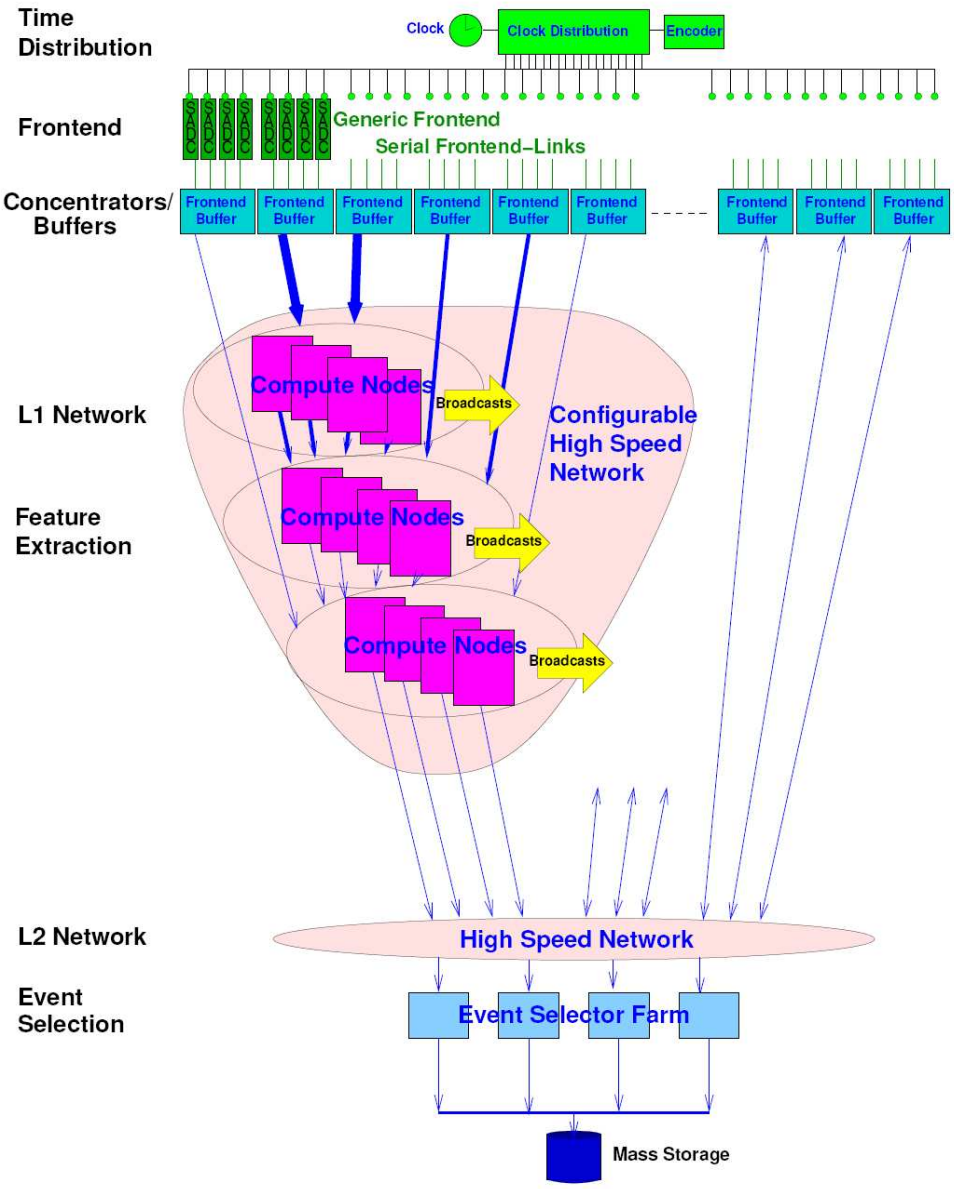


Figure 2.1. DAQ and Trigger system for PANDA experiment

on the detector devices. Through a large bunch of high-speed serial links (commonly more than 10^5 individual channels) and via massive storage capacity of concentrators or buffers, huge amounts of data are supplied to the computation network for pattern recognition processing. The background data which are not interesting to physicists will be discarded on the fly. The interesting ones which meet the expected patterns are forwarded to PC clusters for storage and offline analysis.

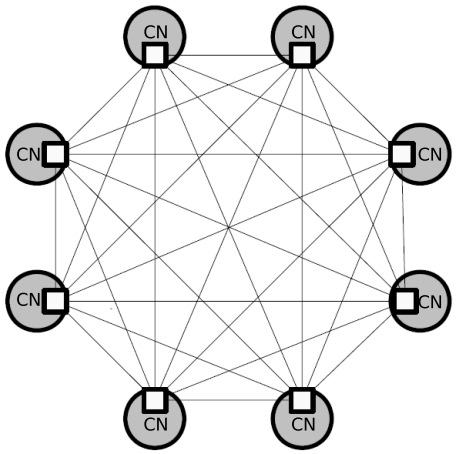


Figure 2.2. Full-mesh network topology of compute nodes via the ATCA backplane (only 8 nodes shown)

In this thesis project, our focus is on the reconfigurable computation network design for online data processing. In practice, pattern recognition algorithms are to be partitioned and distributed in many compute nodes to work in parallel and/or pipeline for high processing throughput. The Advanced Telecommunications Computing Architecture (ATCA or AdvancedTCA) [24] standard was architected to provide the bandwidth needed for the next generation computation platform. The backplane provides Point-to-Point (P2P) 100 Ω differential signal connections between the boards and does not use a data bus like VMEbus or FASTbus. It is typically used to move data between the Field Replaceable Units (FRU) in each slot and the outside network. The Fabric Interface on the backplane supports different architectures such as Dual-Star, Dual-Dual-Star, Full-Mesh, Replicated-Mesh, and others. A full-mesh shelf (the topology shown in Figure 2.2), that we plan to use to build our platform, can host 2.1 Tbps of data transport

when using 3.125 GHz signaling and 8B/10B [25] encoding on the backplane. These direct connections provide much flexibility for network configurations, such as the vertical pipelined processing, or the horizontal parallel processing, or hybrid solutions with more complicated interconnections vertically and horizontally. This feature makes the platform general-purpose for various applications with different network architectures. It also provides significant freedom and convenience to partition the processing logic across multiple boards. In one ATCA chassis (see Figure 2.3) , up to 14 nodes can be fitted. Practically we plug in 13 compute nodes plus one Ethernet switch.

Figure 2.3. ATCA chassis

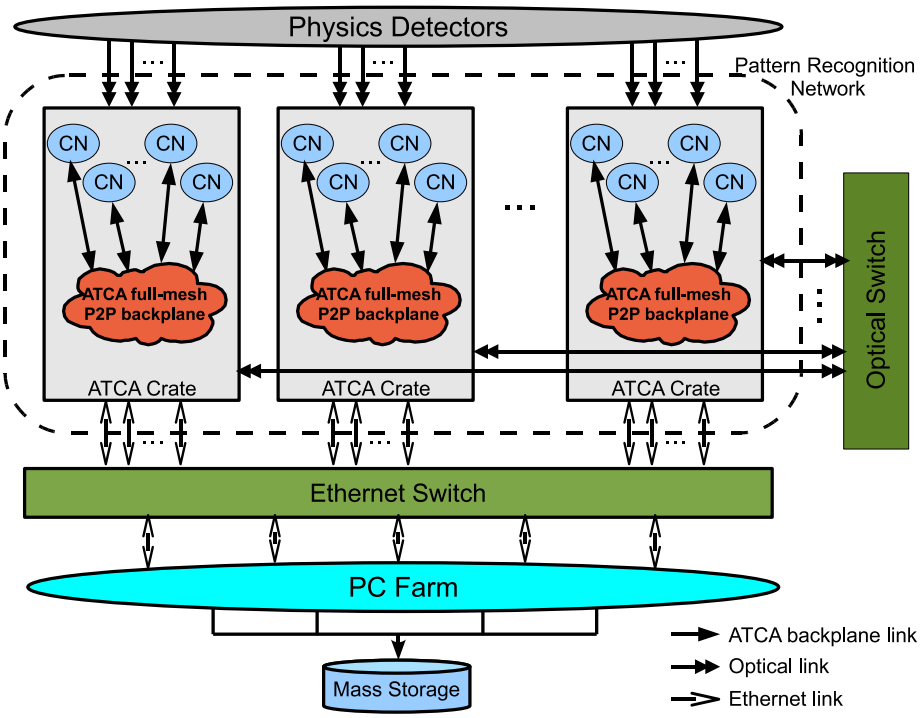


Figure 2.4. Computation network for online pattern recognition

Figure 2.4 shows the computation network architecture in the experimental facilities, where multiple ATCA chassises are required to meet high communication and computation requirements. Through bonded optical channels, raw data are dumped from the detector front-end circuits into the computation network. The front-end circuits samples and digitizes the analog signals directly from detectors. After that, all the data will be processed in the pattern recognition network, which is made up of compute nodes and ATCA crates, for pattern recognition, correlation, event building and event filtering. All these processing algorithms are partitioned and distributed in the FPGA logic cells. And all of the steps constitute the complete algorithm computation by communicating through the hierarchical interconnections including on-board I/O links, inter-board shelf backplane connections, and perhaps the inter-chassis optical link or Ethernet switching if necessary. Of course the on-board communication has the fastest speed, while the inter-chassis switching has more communication overhead and will introduce extra

latency and throughput penalty. Thus trying to locate the computing steps with more communication requirements on the same board or next in the same chassis is a basic rule to implement these algorithms in practice. After pattern recognition computation and event selection in the network, a large proportion of events is discarded on the fly while only a small part will be labeled as interesting and forwarded to the PC cluster over Ethernet for storage and in-depth analysis.

2.2 Compute Node

2.2.1 FPGA Component and Interconnections

The computation platform uses Xilinx platform FPGAs as the primary processing components. The first prototype chose Virtex-4 FX60. In the real products the latest generation FPGAs might be adopted instead. Table 2.1 [27] lists the available resources of Virtex-4 FX family chips.

Virtex-4 FX family platform FPGAs target mainly the embedded processing and serial connectivity application domains. Except for plenty of programmable logic cells, storage RAM components, and General-Purpose I/Os (GPIO), some hard IP cores are also embedded on the die. Specifically PowerPC 405 processors are useful for designs which expect high-level software management of the system or HW/SW hybrid processing. Tri-mode Ethernet MACs ease interconnections of the embedded systems and desktop PCs using the popular Ethernet and TCP/IP protocol. RocketIO [26] serial transceiving is well suited for high-bandwidth data transport. It supports many types of commercial standards including Infiniband, Gigabit Ethernet, Fibre channel, PCI Express, Aurora, and also customized definitions. These factors were all in the range of our consideration when component devices were selected. In our case we chose FX60 which is a good balance between the resources and the price.

Figure 2.5 shows the peripheral connections of each FPGA on the board. We will discuss them one by one in the following part.

Figure 2.6 shows the block diagram of the Compute Node (CN). Each board consists of five FPGAs, four of which (No. 1 to 4) work as algorithm processors and the fifth (No. 0) as a switch interfacing to other modules via the ATCA backplane. Each processor FPGA has two RocketIO Multi-Gigabit Transceiver (MGT) based optical links, which may run at the maximum baud rate of 6.5 Gbps per channel. In addition, all the FPGAs

		Virtex-4 FX (Embedded Processing & Serial Connectivity)					
		FX12	FX20	FX40	**FX60**	FX100	FX140
CLB Resources	CLB Array	64 x 24	64 x 36	96 x 52	**128 x 52**	160 x 68	192 x 84
	Slices	5,472	8,544	18,624	**25,280**	42,176	63,168
	Logic Cells	12,312	19,224	41,904	**56,880**	94,896	142,128
	Flip Flops	10,944	17,088	37,248	**50,560**	84,352	126,336
Memory Resources	Max Distributed RAM bits	87,552	136,704	297,984	**404,480**	674,816	1,010,688
	Block RAM/FIFO (18 kbits each)	36	68	144	**232**	376	552
	Total Block RAM (kbits)	648	1,224	2,592	**4,176**	6,768	9,936
Clock Resources	Digital Clock Managers (DCM)	4	4	8	**12**	12	20
	Phase-matched Clock Dividers (PMCD)	0	0	4	**8**	8	8
I/O Resources	Max Select I/O	320	320	448	**576**	768	896
	Total I/O Banks	9	9	11	**13**	15	17
	Digitally Controlled Impedance	Yes	Yes	Yes	**Yes**	Yes	Yes
	Max Differential I/O Pairs	160	160	224	**288**	384	448
	I/O Standards	LDT-25, LVDS-25, LVDSEXT-25, BLVDS-25, ULVDS-25, LVPECL-25, LVCMOS25, LVCMOS18, LVCMOS15, PCI33, LVTTL, LVCMOS33, PCI-X, PCI66, GTL, GTL+, HSTL I (1.5V,1.8V), HSTL II (1.5V,1.8V), SSTL2I, SSTL2II, SSTL18 I, SSTL18 II					
DSP Resources	XtremeDSP Slices	32	32	48	**128**	160	192
Embedded Hard IP Resources	PowerPC Processor Blocks	1	1	2	**2**	2	2
	10/100/1000 Ethernet MAC Blocks	2	2	4	**4**	4	4
	RocketIO Serial Transceivers	0	8	12	**16**	20	24
Speed Grade	Commercial (slowest to fastest)	-10, -11, -12	-10, -11, -12	-10, -11, -12	**-10, -11, -12**	-10, -11, -12	-10, -11
	Industrial (slowest to fastest)	-10, -11	-10, -11	-10, -11	**-10, -11**	-10, -11	-10, -11

Table 2.1. Available resources of the Virtex-4 FX family FPGAs

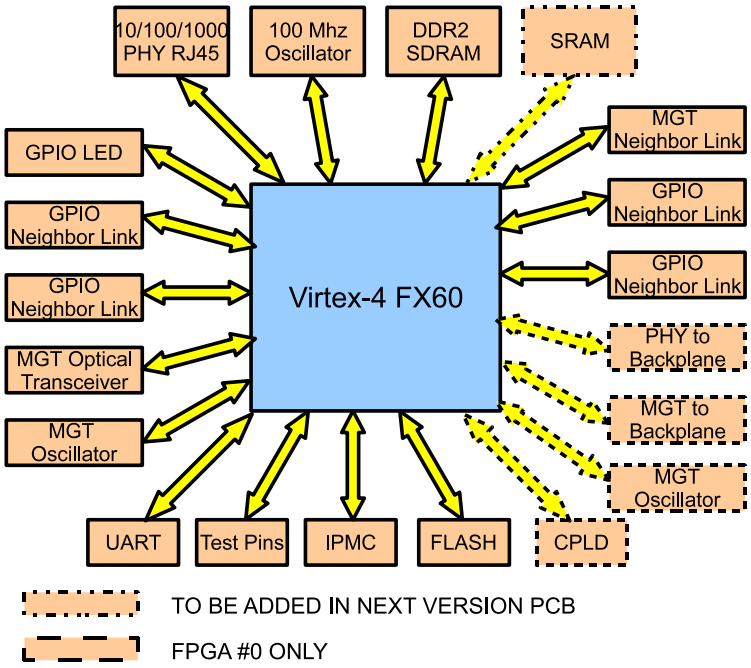


Figure 2.5. Peripherals connected to the FPGA

are equipped with one Gigabit Ethernet each, over which the results are forwarded to the PC farm.

All four processor FPGAs are interconnected with each other in a full-mesh topology. The connectivity includes both 32-bit General-Purpose IO (GPIO) buses and one full-duplex RocketIO link per connection. These processor FPGAs also connect to the switch FPGA with dedicated 32-bit GPIO channels. Depending on application requirements, either circuit-switching or packet-switching design can be configured in the FPGA fabric to communicate with other CNs in the chassis. 16 RocketIO channels to the backplane feature the bandwidth of 104 Gbps at 6.5 GHz signaling. Except for the switch fabric, sub-event data from all four processor FPGAs can be collected in the switch FPGA and do the event building and filtering. With the on-board P2P interconnections, it is convenient to partition complex algorithms and aggregate all five FPGAs as a virtual one with five times capacity.

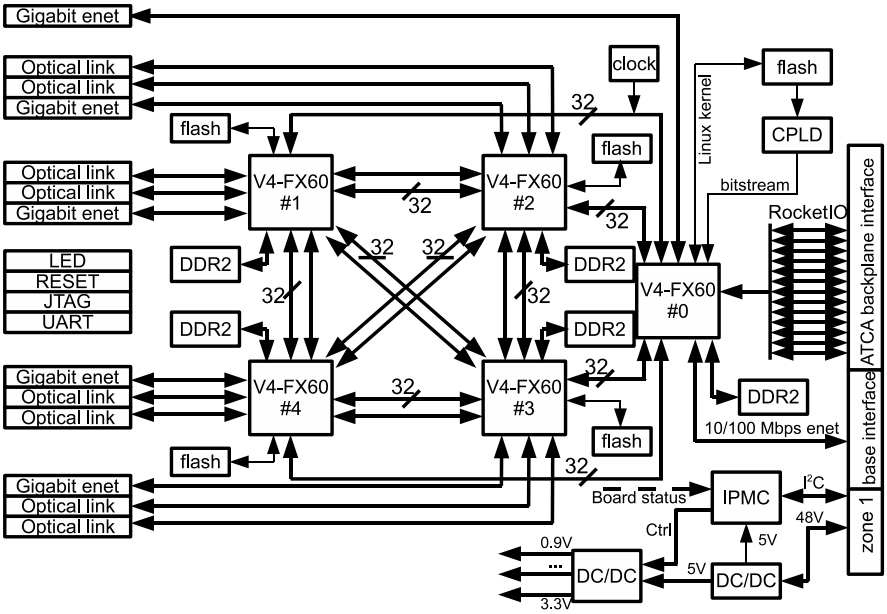


Figure 2.6. Compute node schematic

2.2.2 DDR2 Memory Module

On the compute node, all five FPGAs have their respective 2 GBytes DDR2 memories. The total 10 GBytes SDRAM memory capacity is mainly used for data buffering and large Look-Up Tables (LUT). In the design, we employ five ultra low profile Small Outline Dual In-line Memory Modules (SO-DIMM) DDR2 SDRAM modules. The data bus is 64-bit wide and the maximum data rate is 6400 MBytes/s (PC2-6400@800 MHz).

Considering the large overhead of the DDR2 memory access, SRAM chips may be added in later versions of the prototype board. They will be used to improve the memory access speed for middle-size LUTs or data buffering.

2.2.3 FPGA Configuration and OS Loading

Each FPGA has 64 MBytes NOR flash memory to store the embedded Linux Operating System (OS) kernel and other non-volatile data. Two flash chips, each of which has 16 bit data bus, are combined to form a

32 bit data bus interfacing to GPIOs of the FPGA. All the bitstreams for five FPGAs are merged and placed in the flash for the switch FPGA. A CPLD (Xilinx XC95144) helps to configure the FPGA chain in a serial mode during power on. After the hardware configuration of all five FPGAs, a bootloader program executes to copy the OS kernel to the main memory (DDR2 SDRAM) and start the OS for each respective FPGA.

The JTAG chain is a good choice to download bitstreams as well as software applications to FPGAs during system debugging. Figure 2.7 lists the components in the JTAG chain including the CPLD and five FPGAs.

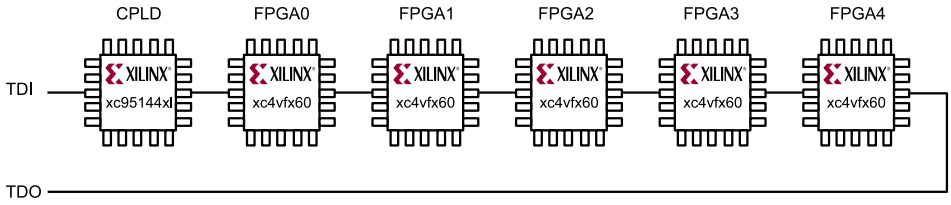


Figure 2.7. JTAG chain on the compute node

The file system of the embedded OS will reside in the storage device of PC clusters. They will be mounted as the root directory through Ethernet with the support of the Network File System (NFS) [28] protocol in Linux (see sub-section 3.3.1). To remotely reconfigure the system, including both the FPGA configurations and the OS kernels, what we need to do is to overwrite the old bitstreams and kernel image files in the NOR flash memory with the new versions and then restart the platform. Afterwards the system will be configured and booted with new hardware designs and operating systems. The Memory Technology Device (MTD) driver provides the application interface to read and write the flash memory in Linux. Operators can remotely login the embedded Linux from a PC and issue the upgrade and restart commands. To avoid fatal errors during the upgrading process, which may destroy the data in the flash and disable normal booting in the next power-on, backups are stored in the flash memory (see Figure 2.8). In case of upgrading errors, Linux can be booted from the backup bitstream and the backup kernel image to resume the data copy to the normal bitstream and kernel space in the flash chips. Switching between the normal booting and the backup booting is controlled by the Intelligent Platform Management Controller (IPMC) module on the compute node.

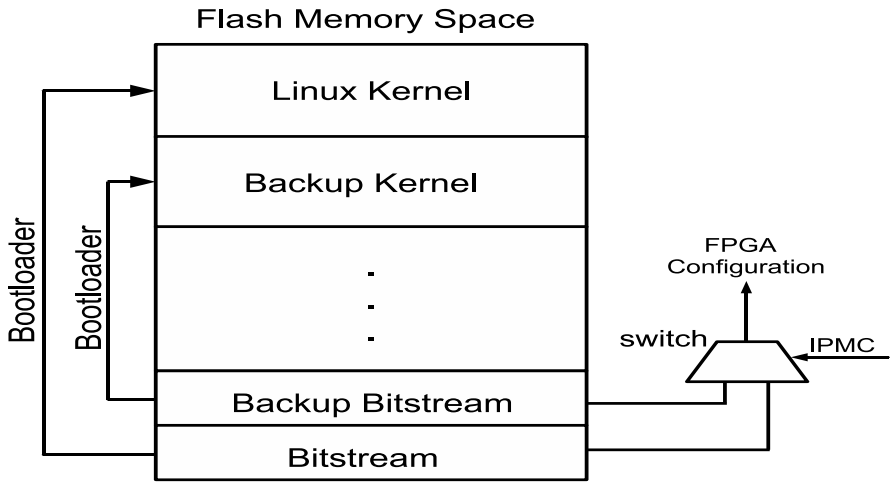


Figure 2.8. Backup configuration in the flash memory

2.2.4 IPMC module

A customized IPMC fulfills the ATCA requirements on power negotiation, voltage monitoring, temperature sensoring, and FPGA configuration check, etc.. It talks to the on-board sensors and the ATCA shelf manager with three I^2C buses. The design is based on the AVR microcontroller and appears as an add-on card on the compute node.

2.2.5 Clock and Power Distribution

The FPGA network is a pseudo-synchronous clock domain, i.e. the clock frequency for each FPGA node is the same but the clock phases may vary. On the board, each FPGA has a dedicated 100 MHz oscillator providing the clock for the system design. In addition, each Ethernet PHY chip has a 125 MHz and each RocketIO column has a 200 MHz high precision oscillator, both of which are in separate packages.

ATCA crates provide -48 V DC power supply. With the on-board conversion, different voltage supplies are obtained for chips and circuits. Figure 2.9 shows the power distribution tree on the board. We pessimistically accumulated all the power hungry components and arrived at the maximum consumption of 170 W per board. The power budget for each ATCA slot is 200 Watt at most, which should be enough for each module.

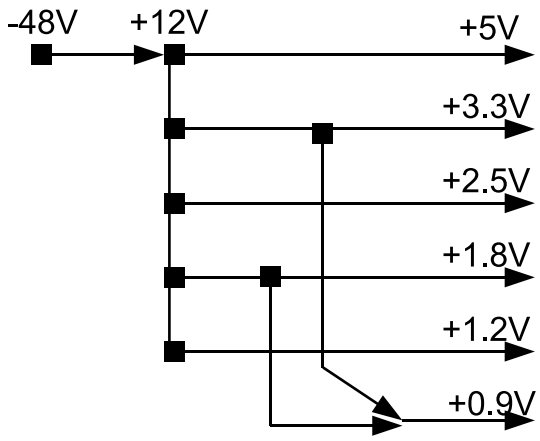


Figure 2.9. Power distribution tree on the compute node

2.2.6 PCB Design

Figure 2.10 is the picture for our first version PCB of the compute node. To meet the dimension requirement from the ATCA chassis, all main components are placed on the top side except five SO-DIMM DDR2 SDRAM memories. The board is in a standard ATCA 12U size of 280 x 322 mm and the layout has 14 layers. In the next version PCB product, two or more layers may be added due to newly placed components such as SRAMs.

Each CN module resides in one of the 14 slots in ATCA chassis. When all 14 modules are plugged in, such a chassis can host up to 1890 Gbps inter-FPGA on-board connections (assuming GPIO running at 300 Mbps), 1456 Gbps inter-board backplane connections, 728 Gbps full-duplex optical bandwidth, 70 Gbps Ethernet bandwidth, 140 GBytes of DDR2 SDRAM, and all computing resources of 70 Virtex-4 FX60 FPGAs.

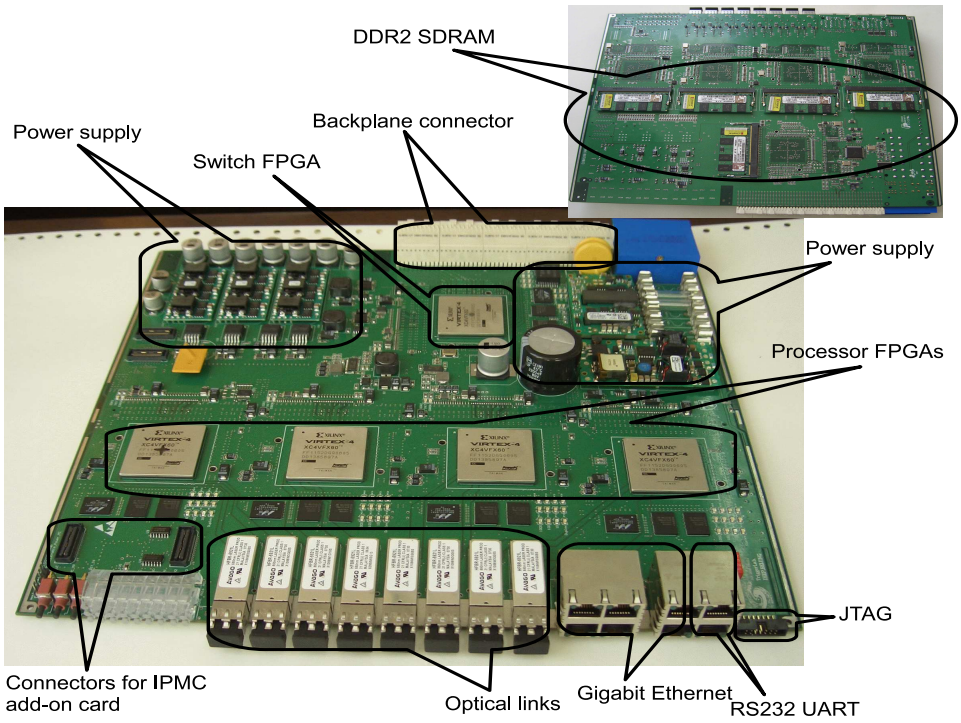


Figure 2.10. Prototype PCB of the compute node

Chapter 3

HW/SW Co-design of the System-on-an-FPGA

Xilinx platform FPGAs, such as the Virtex-4 FX family we used, contain both programmable resources and embedded hardcore PowerPC processors. They will be both utilized as computational resources in our design. In this chapter, we begin from the concrete partitioning strategy for various tasks in particle physics experiments. Afterwards, the hardware design and the software development will be addressed. This HW/SW co-design approach is explicitly formulated, which will be adopted as a general method for different application development on the computation platform.

This chapter presents the contributions published in paper 1 and 3 listed in Section 1.5.

3.1 Partitioning Strategy

System partitioning, also referred to as functional partitioning is an essential problem, since it affects overall system cost, development effort, and performance directly. In our hybrid processing scenario, embedded PowerPC processors execute software programs and the FPGA fabric is customized for mathematical and logic operations. For such a huge and complex system as the DAQ and trigger in particle physics experiments, many features beyond the fundamental functionality of data processing are expected for experiment operations. For example, due to temporal and spatial limitations, operators would like to remotely and dynamically reconfigure and control the platform.

A clear and friendly user interface also helps physicists to easily adjust experimental parameters and monitor the system status. In our platform, two hardcore PowerPC processors in each FPGA could be just utilized to implement those versatile control tasks, while leaving the performance-critical calculation to the FPGA fabric in hardware. Figure 3.1 shows the functional partitioning strategy and concrete criteria are described as follows:

1. All pattern recognition algorithms are to be customized in the FPGA fabric as hardware processor modules, working in parallel and/or pipeline to identify interesting events.

2. Slow control tasks, including monitoring the system status, modifying experimental parameters, re-programming HW/SW designs, etc., are implemented in software by high-level application programs which execute on top of embedded microprocessors and operating systems.

3. The existing soft TCP/IP stack in the operating system is employed for Ethernet communication.

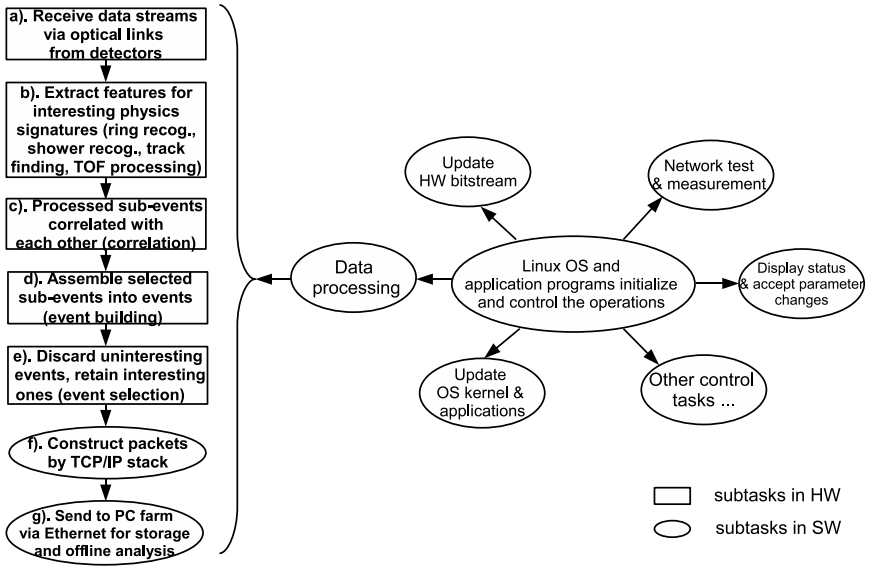


Figure 3.1. Functional partitioning strategy

3.2 Hardware Design on the FPGA

Before obtaining the prototype compute node PCB, most development work has been done on the Xilinx commercial board ML405 [29]. It has a Virtex-4 FX20 FPGA chip whose main features are identical with FX60, except for smaller capacity. Figure 3.2 [29] shows all peripheral components on the board. Except the 32-bit DDR SDRAM instead of the 64-bit DDR2, we see ML405 has similar peripheral connections and is suited for the compute node development.

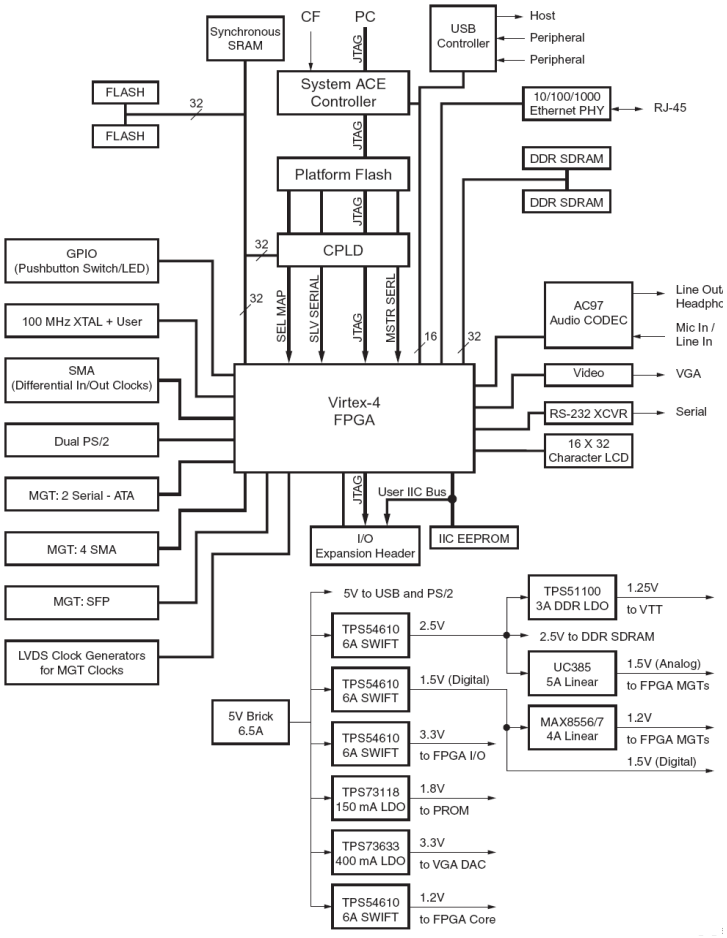


Figure 3.2. Block diagram of the ML405 development board

In Xilinx software suite including EDK, ISE, Chipscope, etc., we develop the hardware system in the FPGA with both hardcore and softcore components. Specifically the PowerPC 405 processors, RocketIO Multi-Gigabit Transceivers (MGT), Digital Clock Managers (DCM) and Gigabit Ethernet MACs are hardcores residing in the silicon fabric of the FPGA chip. Some other peripheral components have to be implemented as softcores by programmable resources. Table 3.1 summarizes all useful cores with their types and functions in our design.

IP cores	Hard or Soft	Functions
PowerPC 405	Hardcore	Embedded hardcore processor for software execution.
RocketIO MGTs	Hardcore	High-speed duplex serial links for communication. It supports many industrial standards. In our system, the optical links and the backplane channels are all RocketIO based.
Tri-mode Ethernet MACs	Hardcore	Ethernet communication between the computation platform and the commodity PC farm.
DCMs	Hardcore	Digital clock managers.
SDRAM memory controller	Softcore	Memory controller for external DDR or DDR2 SDRAM.
PLB_BRAM	Softcore	Small but fast system memory made out of the on-chip Block RAM resource.
Interrupt controller	Softcore	Interrupt management for PowerPC 405.
RS232 UART	Softcore	UART console terminal of PowerPC 405.
External Memory Controller (EMC)	Softcore	Memory controller for external flash or SRAM. It is useful to upgrade the bitstream and OS kernel in the flash memory.
PLB	Softcore	Fast system bus.
OPB	Softcore	Slow system bus.
PLB2OPB	Softcore	Bus bridge of PLB and OPB.
JTAGPPC	Softcore	JTAG controller for PowerPC 405.
GPIO	Softcore	IO pins for buttons and displays.
Chipscope_icon	Softcore	Chipscope integrated controller.
Chipscope_ila	Softcore	Chipscope integrated logic analyzer.
Chipscope_iba	Softcore	Chipscope integrated bus analyzer.
Customized Processing Engines	Softcore	Processing Units for application-dependent data processing.

Table 3.1. IP cores in the system

The interconnections among these components are flexible. They might be a traditional bus-based topology, as well as a high-performance P2P switching architecture. Figure 3.3 shows the two level bus system, in which

high speed components are connected to the Peripheral Local Bus (PLB) while slow ones to the bridged On-chip Peripheral Bus (OPB). The hardware processing engines for pattern recognition appear also as fast devices on the PLB bus. They address memory modules and receive controls from the CPU through the bus. All communications among components go through the buses, which may be congested in case of heavy traffic of data transport. To overcome the bandwidth bottleneck, a Multi-Port Memory Controller (MPMC) [30] based solution (see Figure 3.4) is employed to connect some memory-hungry modules directly to the memory controller. With dedicated links and time-sharing the memory bandwidth, the memory access time will be significantly reduced. Hence the pattern recognition processors benefit from such an architecture when they frequently address buffered detector data and LUT data in the memory. In this system, the PLB bus is used only for low data rate communications and controls.

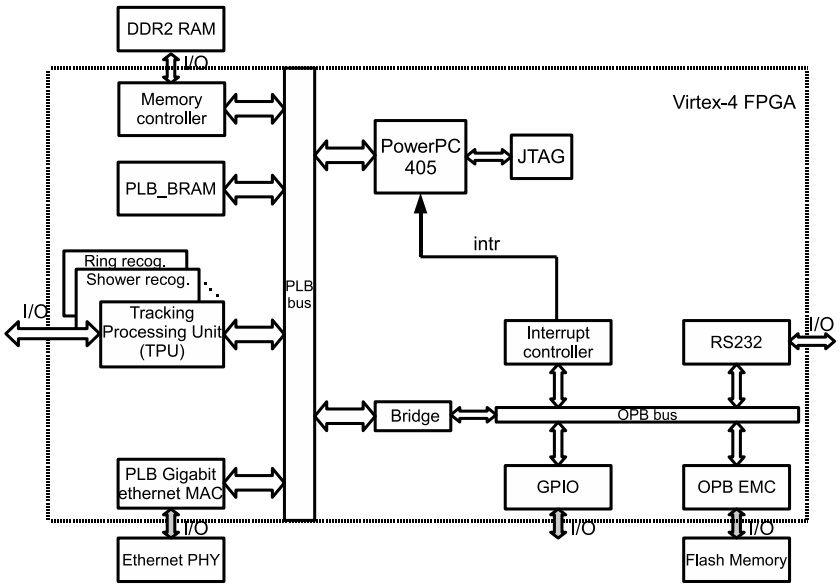


Figure 3.3. Bus-based hardware design

In both Figure 3.3 and 3.4, The I/O arrows from/to the pattern recognition processors may stand for either RocketIO-based optical links from detectors or General-Purpose I/O (GPIO) connections with other modules. Through these channels, custom processors receive incoming data and trans-

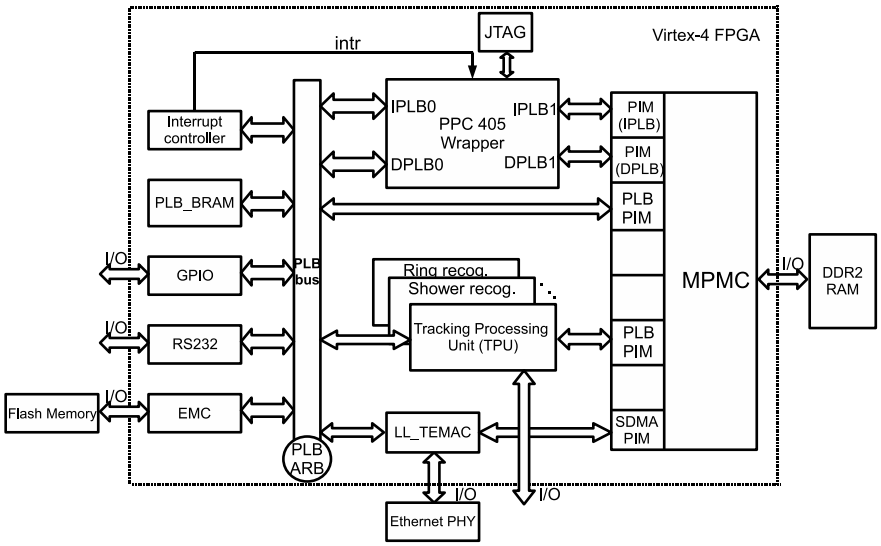


Figure 3.4. MPMC-based hardware design with P2P switching connections

mit processing results to the next stage. The data may also be temporarily buffered in the system memory.

In various application designs, the system architecture is intended to be fixed as a template and only to replace customized processing engines. This maximizes the design re-usability and largely shrinks the time-to-market.

3.3 Software Development

3.3.1 Porting Linux to the Compute Node

To make the software development easy and flexible, an open-source embedded Linux kernel of version 2.6 was ported to the PowerPC 405 architecture in Virtex-4 FX series FPGAs. The main steps [31][32][33][34] can be referred to in Appendix A for details.

3.3.2 Device Drivers in Linux

Device drivers for the Xilinx peripherals in the system design, such as Ethernet, RS232 UART, flash memories, etc., have been already included in

the kernel's package and can be enabled during the kernel configuration process. Other customized modules, for example the Tracking Processing Unit (TPU) which will be discussed in the next chapter, need to be programmed by ourselves. These processing units are modeled as character devices [35], which process data streams coming from the memory or I/O ports. In their drivers, common file operations are implemented including "open", "close", "read", "write", and "ioctl", as well as the DMA initiation and Interrupt Service Routines (ISR). Their device entries will appear in the "/dev" directory after loading driver modules. The devices can then be opened and accessed in application programs.

3.3.3 Application Programs

With the support from the OS and the Application Programming Interface (API), flexible applications can be exploited on the computation platform. The developing tools vary from C/C++ programming to high level scripts. Given as an example, an Apache HTTP server [36] was ported to the embedded Linux. We are using the Common Gateway Interface (CGI) [37] to develop web pages on the server, with which the current experimental status can be remotely shown to operators and commands can be issued to the system. Another example is Linux socket programming in C used to forward selected interesting events to the PC farm over Ethernet.

Chapter 4

Algorithm Implementation and Evaluation

In our project timeline, HADES is the first experiment which will use the computation platform to upgrade the existing DAQ and trigger system. The detectors generate a raw data rate of less than 10 GBytes/s and we plan to employ one single ATCA crate, specifically two compute nodes for each detector sector and twelve for six sectors in total. Plus one more redundant board for backup purpose, there will be thirteen compute nodes in the chassis. Driven by the HADES requirements, we have been developing and evaluating the pattern recognition and correlation algorithms on our platform, including *Cherenkov ring recognition* (for RICH detector), *MDC particle track reconstruction* (MDC detector), *Time-Of-Flight (TOF) processing* (TOF detector and RPC, short for Resistive Plate Chambers detector), *Shower recognition* (Shower detector), *event building*, and *event selection*. All algorithm processors receive readout data from their corresponding detector and search for specific patterns, as the data flow shown in Figure 4.1. Their processing results will be shared and correlated with each other. And all sub-events which meet expected patterns and have been successfully correlated, are preserved and assembled into the pre-defined event structure before storage. According to the data flow, Figure 4.2 shows a feasible approach to partition and distribute all algorithms on two compute nodes for one sector. Representatively the *particle track reconstruction in MDCs* is described in the following as a case study.

This chapter focuses on the main topics in paper 2 as well as poster 6 and 7 listed in Section 1.5.

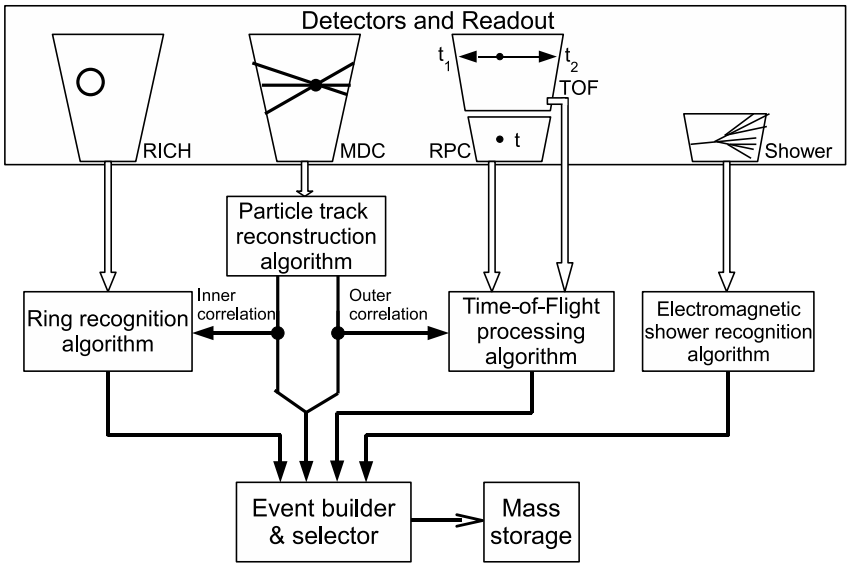


Figure 4.1. Data flow in HADES DAQ and trigger system

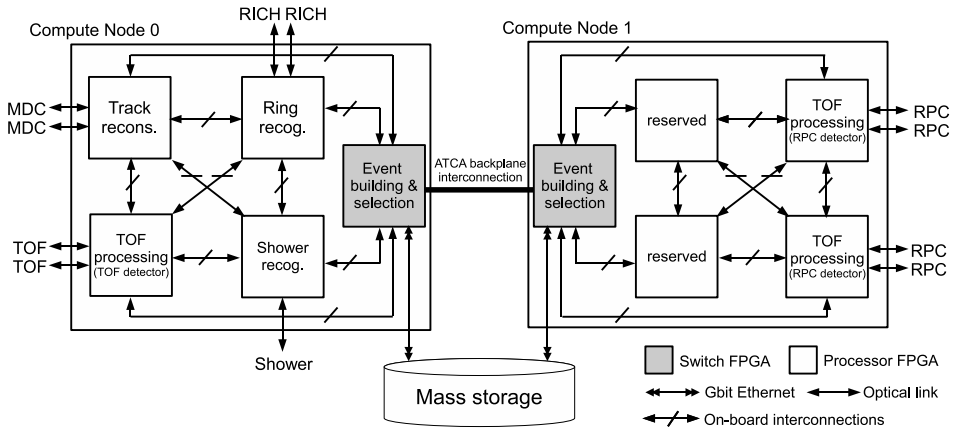


Figure 4.2. Algorithm partition and distribution on two CNs for one sector

4.1 The HADES Spectrometer

The High Acceptance Di-Electron Spectrometer (HADES) [1] is installed at the heavy ion synchrotron of the GSI facilities in Darmstadt, Germany. Its main objective is the measurement of lepton pairs (dileptons) produced in the decay of light vector mesons at a reaction rate of less than 10^6 Hz with an accuracy of $\frac{\triangle p}{p} \simeq 1\%$ particle momentum resolution [38][39]. The HADES spectrometer is composed of several different specialized particle detectors, as shown in Figure 4.3(a) for exploded view, Figure 4.3(b) for mounted shape, and Figure 4.3(c) giving a lateral cut view.

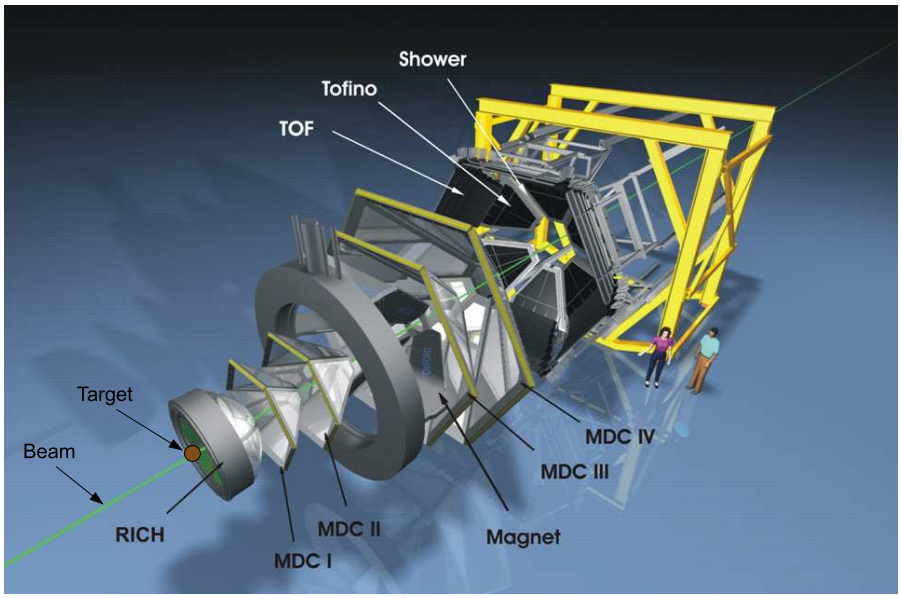
From the figures, we see in the HADES spectrometer, there have been RICH, MDC, TOF, TOFino, Shower detectors installed. In the upgrade project, the RPC detector will also be mounted as the replacement of the TOFino detector for higher timing resolution. All the detectors consist of 6 assembled trapezoidal sectors. When the accelerated beam particles collide the target, product particles will be emitted, and they fly through these detectors and generate signals on them.

4.2 Particle Track Reconstruction in MDCs

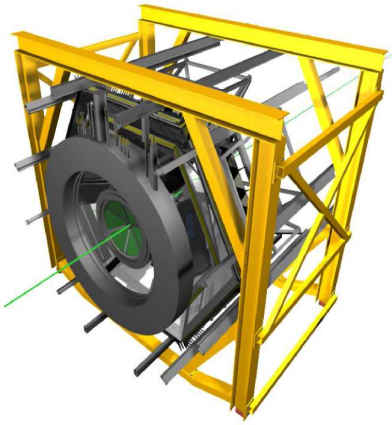
4.2.1 Physics Principle of the Tracking Algorithm

The tracking of charged particles in HADES is accomplished by the so-called Mini Drift Chambers (MDC) detectors. The name "MDC" arises from the comparably small size of the individual drift cells. The HADES tracking system, as an instance of modern experimental facilities, consists of four MDC modules which have six identical trapezoidal sectors (see Figure 4.3). Two MDC layers are located before and two behind the toroidal magnetic field produced by 6 superconducting coils (see Figure 4.3(c)). Practically the magnetic field does not penetrate into MDCs. Thus particle tracks only bend in the magnetic field and the segments before or behind the coil can be approximately described by straight lines. The two segments can be reconstructed separately with the inner (I - II) and the outer (III - IV) MDC information. The basic principle is similar and hence we focus only on the inner part implementation.

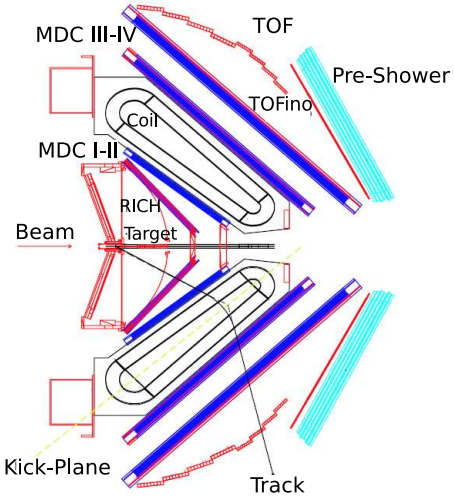
In the two inner MDC modules, a total number of 12660 sense wires (6 sectors) are arranged in 12 layers and 6 orientations: +40°, -20°, 0°, 0°, +20°, -40°, with Figure 4.4 showing one sector. When beam particles hit the target, charged particles are emitted from the target position and go



(a) Exploded exhibition of the HADES detector system



(b) HADES detector system view when mounted



(c) Lateral cut view of the HADES detector system

Figure 4.3. The HADES detector system

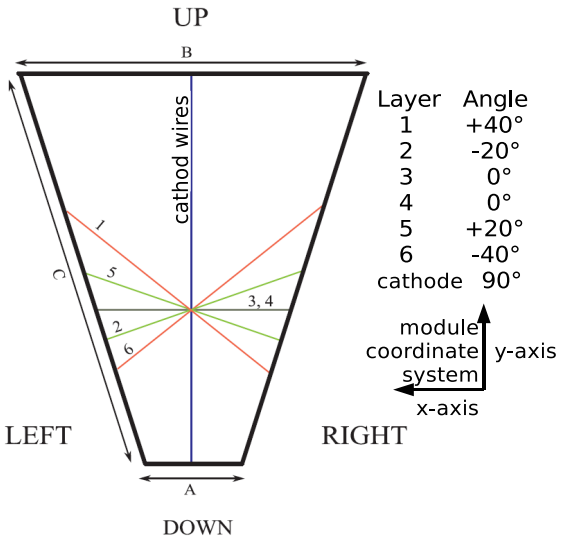


Figure 4.4. One sector of the MDC with six orientation wires

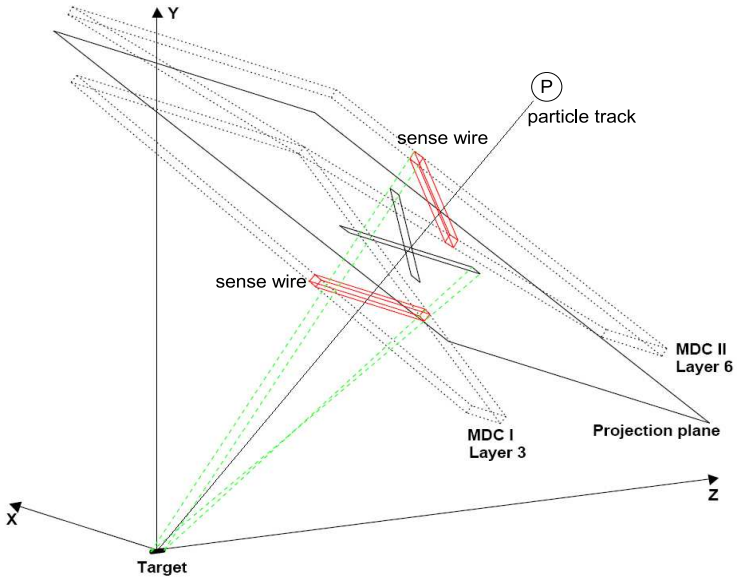


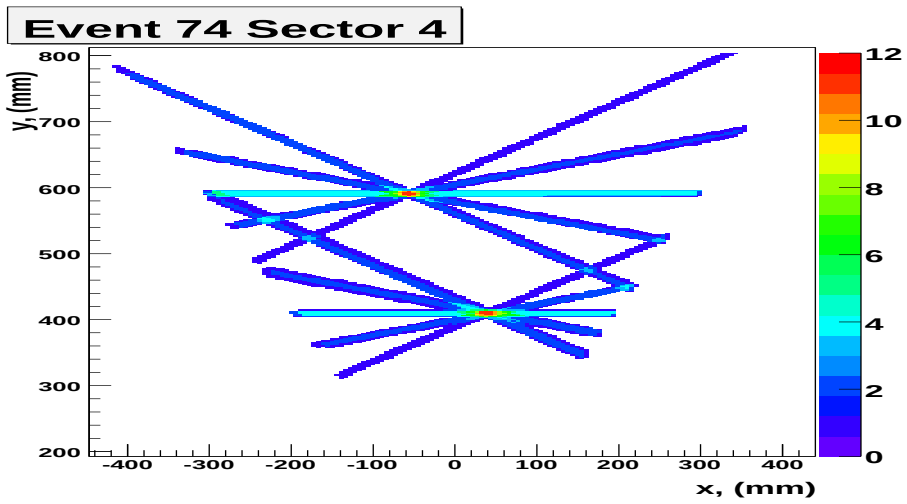
Figure 4.5. Track recognition and reconstruction in inner MDCs

forward through different wire layers in straight paths. Along their flying ways, pulse signals are generated on the wires close to the tracks with high probability (>95%). We also say that the sense wires are "fired" by flying particles. As shown in Figure 4.5, if the sensitive volumn of each wire is projected from the boundary of the target onto a plane located between two inner chambers, apparently the particle passed through the projection plane at the point where all projections of fired wires from different layers overlap. To search for such regions the projection plane is treated as a two dimensional histogram with the projection area as bins (grids). For each fired sense wire, its projection bins are all increased by one. By finding the locally maximum bins whose values are also above a given threshold, track candidates can be recognized and the inner segments of tracks are reconstructed as straight lines from the point-like target to those bins.

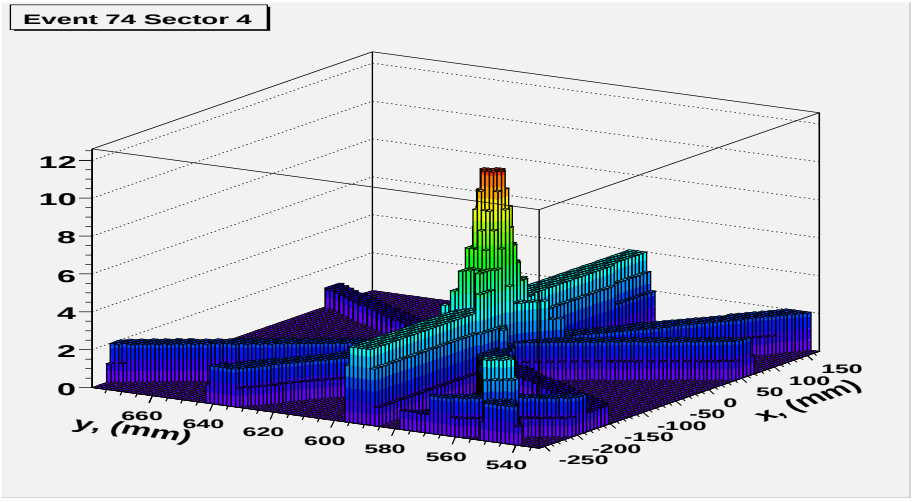
Figure 4.6(a) demonstrates the 2D projection plane for one sector with 2 passed particles. The scale on the right side shows the correspondence between the bin values and the colors in the plot. Figure 4.6(b) is the 3D display of Figure 4.6(a) for a single track, where the coordinates of the peak in the center are recognized as the track's position. Normally we do not consider the peak with less than 8 fired wires out of 12 layers. Therefore assuming the threshold is 10, the peak bin in the center is recognized as the candidate which is most likely caused by a charged particle passing through this point of the projection plane.

4.2.2 Hardware Design on FPGA

The Tracking Processing Unit (TPU) has been developed and implemented on FPGA for online particle track reconstruction during experiments [40]. It is to be integrated in the system architecture for MDC data processing. The TPU module receives the serial numbers of fired wires in each MDC sub-event as inputs and outputs the position of track candidates. The position of track candidates is possible to be represented by the position of peak bins, which can be further converted into coordinate information in space. The tracking information will then be correlated with both RICH and TOF results; If there is a charged particle passing through the MDC detectors, in principle there should also be corresponding indications on the RICH and the TOF detector. This correlation is to be done by supplying the tracking results to the RICH and the TOF processing modules which reside also in the computation platform.



(a) Projection plane with two passed tracks



(b) 3D display of the accumulated bins for a single track

Figure 4.6. Particle tracks in the projection plane of one sector

The TPU design can be decomposed into sub-modules as shown in Figure 4.7, specifically the wire number Write FIFO (WrFIFO), the address Look-Up Table (LUT), the bus master, the projection LUT, the accumulate unit, and the peak finder. For the buffered fired wire numbers in WrFIFO,

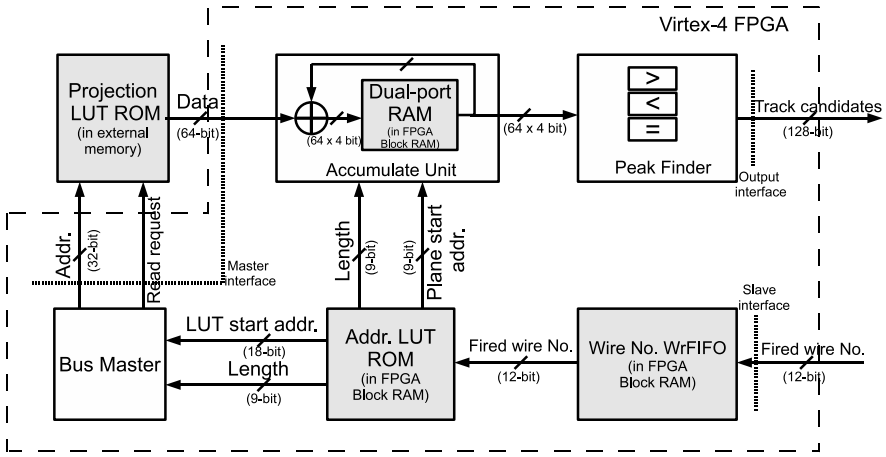


Figure 4.7. Block diagram of the TPU structure

the address LUT stores their storage and projection address information. With the help of the bus master, the projection LUT supplies data to the accumulate unit, to show which bins on the projection plane have been touched by the wires' projection. The accumulate unit accumulates the histogram of the touched times for all the bins. Finally the peak finder compares the bins in the neighbourhood and figures out the exact peak points where particles probably passed through. We will describe these sub-modules in detail as follows.

Wire Number WrFIFO:

The wire number Write FIFO is an asynchronous FIFO which buffers the incoming fired wire sequential numbers. The source of the input might be either the previous level data buffer in the external large-capacity memory, or directly from the I/O inputs of the FPGA chip. Due to the small size of the WrFIFO, probably in the Kilo-byte order of magnitude, it is implemented on FPGA with the Block RAM resource.

Projection LUT and Address LUT:

For each fired wire, we should decide which bins on the projection plane will be touched by its projection shadow and increased by one on the his-

togram correspondingly. This geometrical problem is too complicated to be calculated in realtime, considering there are 2110 sense wires with six orientations in each inner MDC sector. Hence a simpler solution is to lookup the touched bins of a specific wire in a "projection LUT" with its serial number as the entry. The LUT is built offline, and thus if any modification on it, we have to download the new one into the memory. The LUT's size is proportional to the quantity of bins on the projection plane. At present we configure the resolution as 128 x 256 bins for each sector.

In principle each wire should have the LUT mapping for all the bins. One bit per bin is to show whether it is touched by the projection ("1") or not ("0"). Hence 2110 wires need around 8.6 MBytes for the 128 x 256 resolution configuration (128 x 256 x 2110 = 69140480 bits $\approx$ 8.6 MBytes). In practice, we observe that some wires will never touch the bins far away from their shadow on the projection plane. For example, Figure 4.8 shows the situation of a fired $+20^\circ$ sense wire. We see only the bins in the bottom are covered by the shadow and it is not necessary to include the ones on the top in the projection LUT of this wire. Thus only the relevant bins from the *plane starting address* and within the *length* will be built into the LUT mapping for a specific sense wire. In this mechanism, another small LUT is adopted to derive a wire's *plane starting address* in the projection plot and the *length* from its sequential number. We call it "address LUT" to distinguish from the "projection LUT". Compared to the solution which covers all the bins for wire entries, the shrinked projection LUT has two advantages: (1) It results in a smaller size of 1.5 MBytes for storage; (2) Although it needs extra clock cycles to derive the *plane starting address* and the *length* from the address LUT, the shrinked projection LUT saves many cycles during the procedure of supplying data for accumulation. This feature avoids meaningless data transportation and computation. It significantly improves the processing throughput and decreases the output latency of the TPU device.

Except for the *plane starting address*, the address LUT provides also the *LUT starting address*, which indicates the physical starting locations of the wires' mapping in the projection LUT memory. Altogether with the *length*, the *LUT starting address* is fed to the bus master to initiate data transfers from the memory to the TPU module.

Bus Master:

The master interface to the memory device initiates data transfers from the projection LUT to the accumulate unit sub-module, with the address

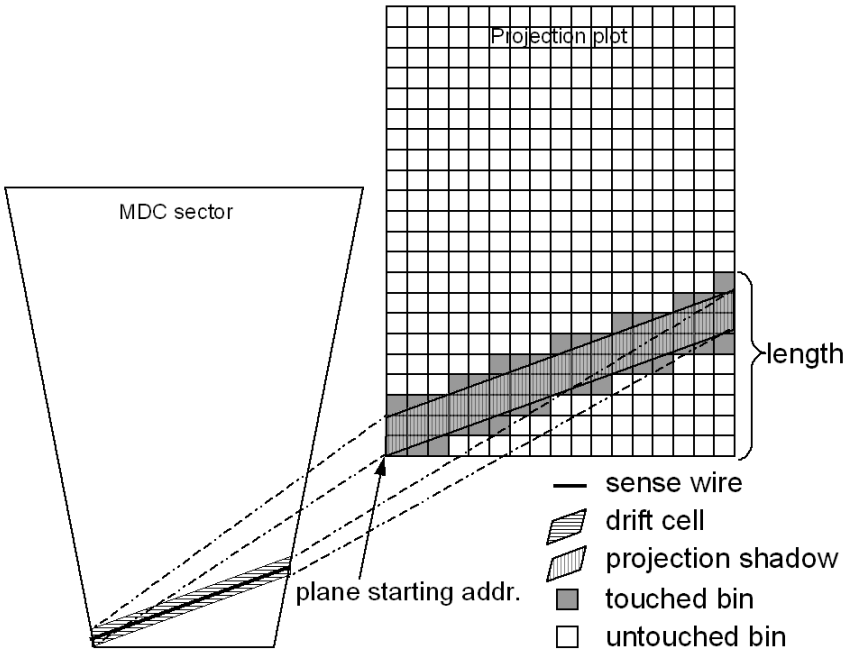


Figure 4.8. Projection and touched bins of a $+20^\circ$ wire on the projection plane

information from the address LUT. It supports burst mode transfers to reduce the data movement overhead. In the bus-based system design, the master device will be interfaced to the system PLB; In an MPMC-based system, it will be directly connected to an MPMC port to move data via the local link using also the PLB protocol.

Accumulate Unit:

The accumulate unit is the sub-module which accumulates the histogram of the shadow-touched-times for all the bins on the projection plane. It is constructed by using a dual-port BRAM block, adders and registers, as shown in Figure 4.9. The BRAM block is 3-dimensional and has 4 bits per bin which allow to represent the maximum 12 layer wires in two MDCs. Initially all the bits are reset to zero. As the projection LUT provides data which show whether a specific bin is touched (a bit of '1') by a fired wire or not (a bit of '0'), the values of bins are correspondingly either increased by

one or not. For each wire, the accumulation only happens on those bins from the *plane starting address* and within the *length*, which are both from the address LUT outputs. The computation process is pipelined in two stages. One is to write the accumulated results into the BRAM block. The other is to read out the values of the next address and get them ready for the next cycle accumulation. After the processing of all fired wires within one MDC sub-event, the entire BRAM block will be exported to the next level peak finder, searching for peak bins or track candidates. Then the BRAM block will be reset again and get ready for the next round computation.

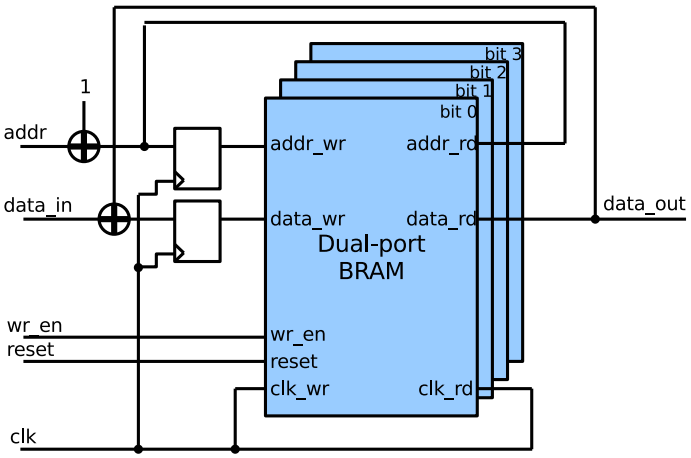


Figure 4.9. Pipelined structure of the accumulate unit

For each bin on the projection plane, there are extra 12 BRAM bits needed to record whether the 12 layers have already been accumulated on this specific bin. It is used to resolve the situation where multiple wires belonging to the same layer are all fired and take effect on one bin. This happens most frequently when two neighboured wires in the same layer have projection overlap on some bins. In this case duplicated accumulation should be avoided. If a bin has already increased by one with a fired wire belonging to a specific layer of the total twelve, this layer should be recorded in the corresponding bit and not take effect on this bin any longer until the next sub-event.

Peak Finder:

The peak finder is the most computation-intensive part in the entire TPU design. It indicates not only which bins have values not less than the threshold, but also the exact peak bins in their neighbourhoods where particles probably passed. Figure 4.10 shows the necessity to build such a sub-module. There are seven bins meeting the threshold requirement assuming the threshold is 10. However in fact they belong to a single track. So more delicate computation should be done to find out the peak in this area, which is most probably the point a particle passed through on the projection plane. In case of multiple peak bins with the same value and neighboured with each other, we can export any one of them or directly all of them according to the system requirement.

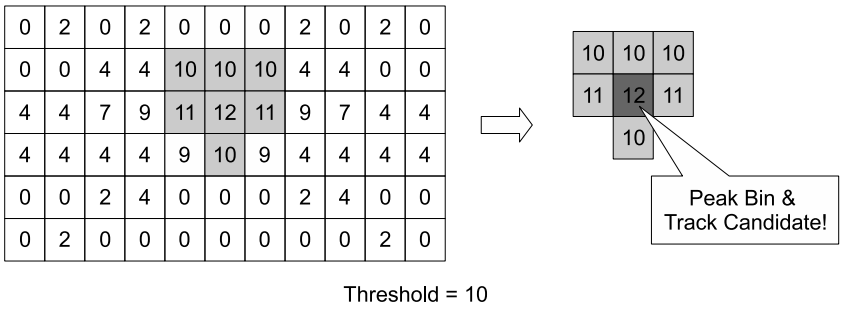


Figure 4.10. Selection of the peak bin in the neighbourhood

To indicate the peak in one area, each bin is arranged to be compared with all its eight neighbours, except for those in the boundaries which have fewer neighbours. If none of the neighbours is larger than that bin, it is identified as the peak. Otherwise the larger neighbours overcome it and keep on searching. This calculation was implemented as pipelined comparisons in our design. As shown in Figure 4.11, the input data go downwards cycle by cycle from "data_in", passing two levels of registers "Reg1_array" and "Reg2_array" until the results come out. Using the same data from Figure 4.10, the pipelined peak finding process is demonstrated by four subfigures for four clock cycles. Each square in "Reg1_array" which represents a single bin with its value inside, is being compared with four of its neighbours: the upper-left, the upper, the upper-right, and the left, with an arrow representing a comparison. It is also being compared with its right neighbour

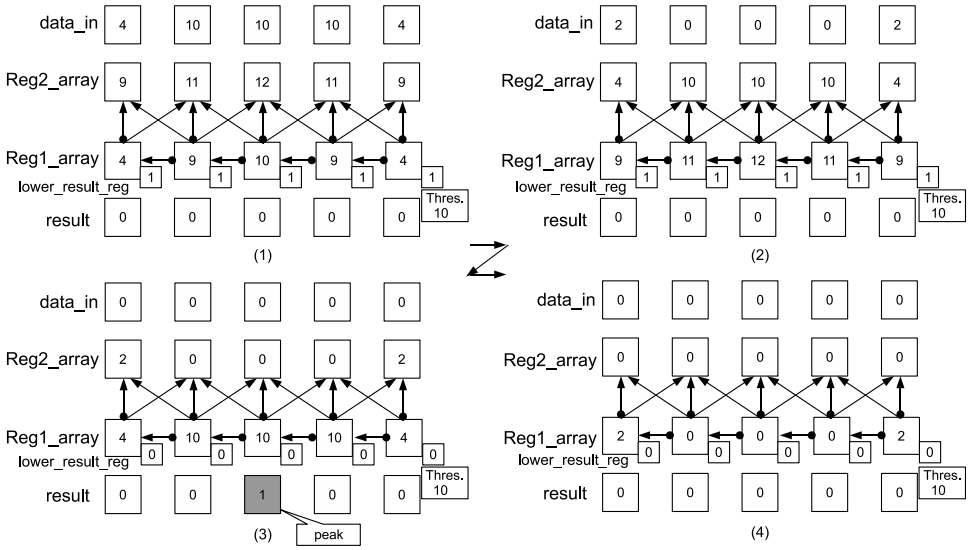
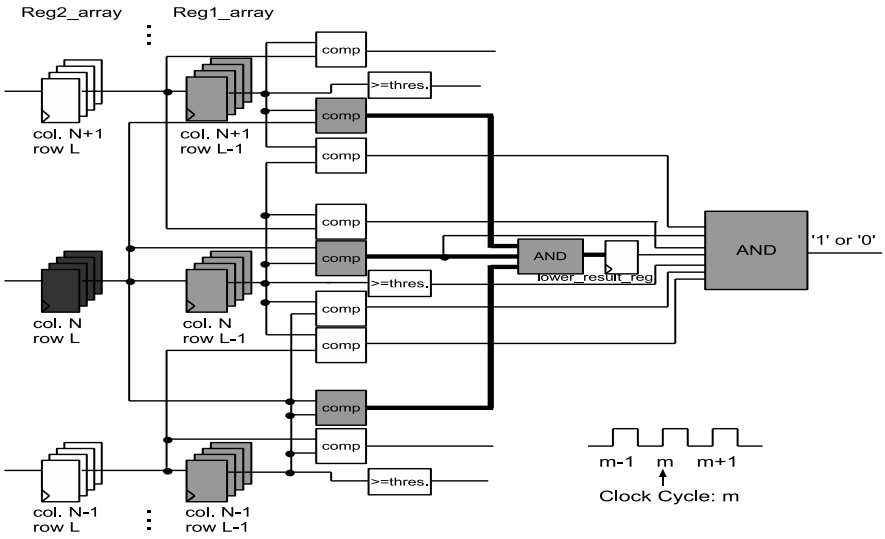
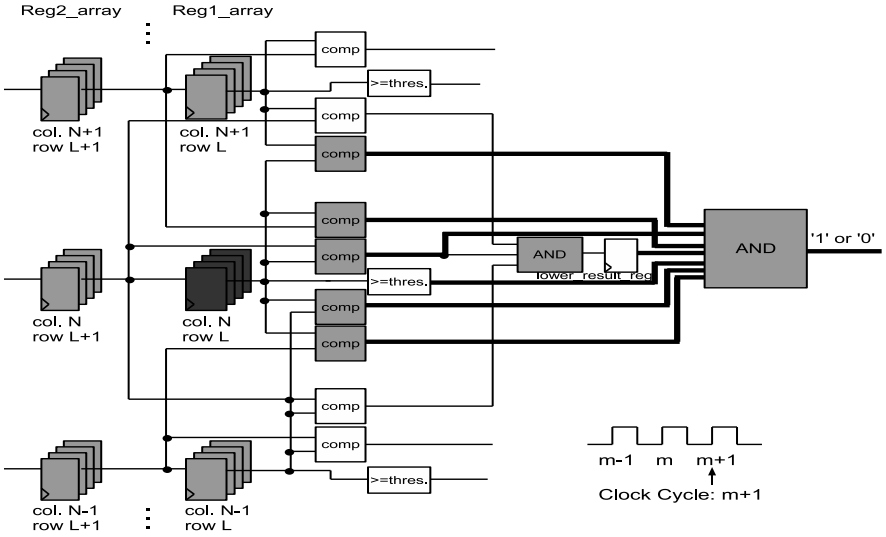


Figure 4.11. Pipelined peak finding process

since the comparison is initiated by its right neighbour bin. Moreover the bins in "Reg2_array" are being compared with their respective lower-left, lower and lower-right neighbours by using the same comparators. Hence in summary, each bin has been compared with its three lower neighbours when it stays in "Reg2_array", and with its five upper and horizontal neighbours after it steps to "Reg1_array" in the next clock cycle. The comparison results with the lower neighbours are stored in the register "lower_result_reg" temporarily. A bit of "1" means none of its lower neighbours is larger than that bin while "0" means it cannot be the peak and will be ignored. In case of two neighboured bins having the same value, the one in the higher address direction (right and upper) wins the comparison and may go on being compared with those at even higher addresses. The bit from "lower_result_reg" will be ANDed with the results from the five upper and horizontal comparisons and the threshold comparison. If the final result is '1', that bin is the peak in its area and is recognized as a track candidate. Figure 4.12 illustrates the Register Transfer Level (RTL) schematic of the pipelined peak finder. We can also observe the signal transmission process leading to the final decision within two clock cycles. For simplicity, only the registers for one bin and its eight neighbours are shown in the figure.



(a) The object bin (col. N and row L) is compared with its three lower neighbours. The result will be buffered in "lower_result_reg".



(b) The object bin (col. N and row L) is compared with its five upper and horizontal neighbours. The results will be ANDed with the previous one in "lower_result_reg" and the one from the threshold comparison to make the final decision. An output of '1' means that the object bin is a peak bin and there is a track candidate.

Figure 4.12. The RTL structure of the peak finder

4.2.3 Device Driver

In order to fully utilize the hardware acceleration and avoid the large software overhead, as much calculation as possible has been partitioned in hardware to prevent the PowerPC processor from interfering the track reconstruction computation. The CPU takes only care of feeding fired wire numbers from the DDR/DDR2 buffer to the WrFIFO, or initializing DMA transfers to do this. Hence in the device driver for Linux, the "write" operation or the DMA initiation are implemented to supply wire numbers to the WrFIFO. Also experimental parameters such as the threshold can be stored in device registers and addressed with "ioctl" operations, to read and display the experiment status or modify them according to different experimental requirements.

4.2.4 Implementation Results

The TPU design was described in VHDL and implemented on FPGA using Xilinx ISE software. Its master device can be either interfaced to the system PLB bus, or connected to one MPMC port directly for accessing the projection LUT data in the external DDR/DDR2 memory. We prefer the latter solution in order to relieve the system bus from heavy data transport. When the resolution of the projection plane is configured as 128 x 256 bins, the resource consumption statistics are shown in table 4.1. The utilized percentages of the Xilinx Virtex-4 FX60 FPGA are also demonstrated. This module will be integrated in the MPMC-based system and so the resource utilization of the FPGA system design is also listed. From the statistics, we observe that 12.3% LUT resource and 5.9% Flip-Flops contribute to mainly construct the computational logic and registers. In addition 19.4% Block RAM resource is dedicated to the storage components in the TPU design, including the dual-port BRAM block after accumulation, the address LUT, and the wire number FIFO. The last column in the table indicates the resource consumption of the entire system with one TPU integrated. From the figures we conclude that the total resource utilization is acceptable, and it is feasible to implement the online inner track reconstruction computation on the Virtex-4 FX60 FPGA.

The timing summary shows that the TPU design can run at 125 MHz without further optimization effort. To match the speed of the MPMC core and the PLB bus, we fix its clock frequency at 100 MHz.

Resources	MPMC-based FPGA system design (no application processor)	TPU module	MPMC-based system with the *TPU*
4-input LUTs	10008 out of 50560 (19.8%)	6210 out of 50560 (12.3%)	16218 out of 50560 (32.1%)
Slice Flip-Flops	8440 out of 50560 (16.7%)	2966 out of 50560 (5.9%)	11406 out of 50560 (22.6%)
Block RAMs	53 out of 232 (22.8%)	45 out of 232 (19.4%)	98 out of 232 (42.2%)
DSP Slices	0	0	0

Table 4.1. Resource utilization of the MPMC-based system and the TPU

4.2.5 Performance Measurements

We measured the processing capability of the TPU module in the following experimental setup: the TPU is incorporated in the system design. Its slave interface accepts fired wire numbers as input and the master interface requests projection LUT data from the external DDR2 memory via the MPMC port. The TPU and the MPMC port run at 100 MHz. The MPMC core as well as the DDR2 memory run at 200 MHz. The PowerPC CPU initializes the projection LUT and supplies randomly generated wire numbers to the WrFIFO. Due to the case that wires in different positions or orientations have different projection areas, they have various data block sizes in the projection LUT for transportation and computation. With the randomly generated wire numbers, the average projection LUT size is around 5.7 Kbits per wire (1.5 MBytes for 2110 wires). In case of heavy ion reactions implying more emitted particle tracks from the target and hence more fired wires in each MDC sub-event, it takes more clock cycles to fetch the projection LUT data and execute the track reconstruction processing. Therefore we took into account the number of fired wires in each sub-event during the experiments and measured the processing speed of the TPU module at five wire multiplicities, specifically assuming that 10, 30, 50, 200, and 400 wires are fired in a single sub-event. The results demonstrate the processing capability of 32.3, 12.2, 7.5, 2.0, 1.0 Kilo sub-events/s respectively. Compared to the software solution in C program, which runs on an Intel Xeon 2.4 GHz CPU/1GB DDR2 memory server with the Gentoo Linux OS, the measured results are shown in Figure 4.13. This reveals a highest hardware acceleration of 24.3 times for light ion reactions and a lowest one of 10.8 times for heavy reactions. The processing speedup of the TPU design over the

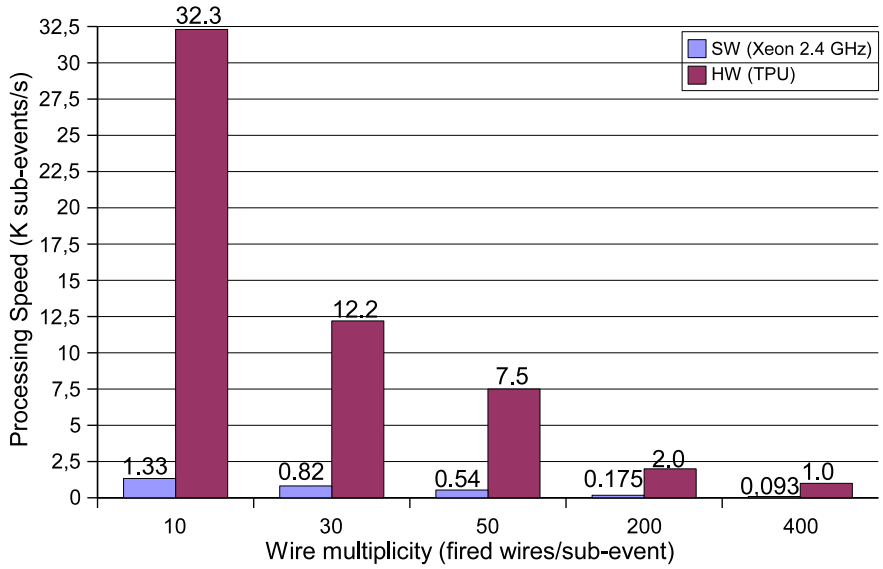


Figure 4.13. HW & SW processing capability on MDC sub-events

software solution is listed in Figure 4.14 for five assumed wire multiplicities.

The above results come from the practical measurements on the compute node. In this experimental setup, there are some non-TPU factors which introduce overhead and restrict the TPU's performance. Particularly the DDR2 memory altogether with its controller has complex address mechanism which results in a large access latency. And it supports the data transfer burst mode of only 8 beats at most. In addition the MPMC core leads to clock cycle loss when arbitrating the memory access among multiple ports. All these factors interleave the data transport from the DDR2 projection LUT and waste clock cycles during this process, as clearly shown in the waveform of figure 4.15. In order to efficiently utilize the data bus bandwidth of the TPU module, SRAM memory devices would replace the DDR2 SDRAM to store the projection LUT and be interfaced directly to the TPU master interface. With the dedicated bandwidth and more efficient memory access features of SRAM, the processing speed of the TPU module is foreseen to be doubled, achieving a speedup ranging from about 20 to 50 times for various wire multiplicities over the software solution.

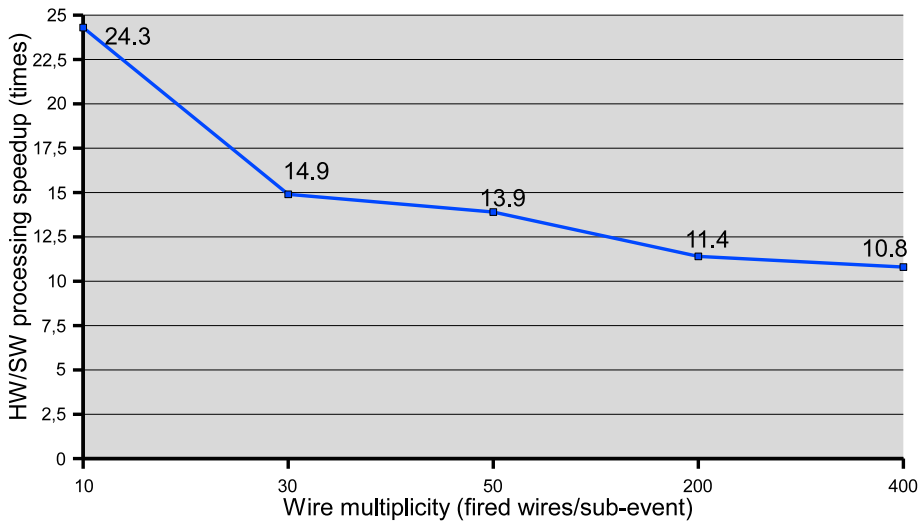


Figure 4.14. Speedup of the TPU module over SW solution

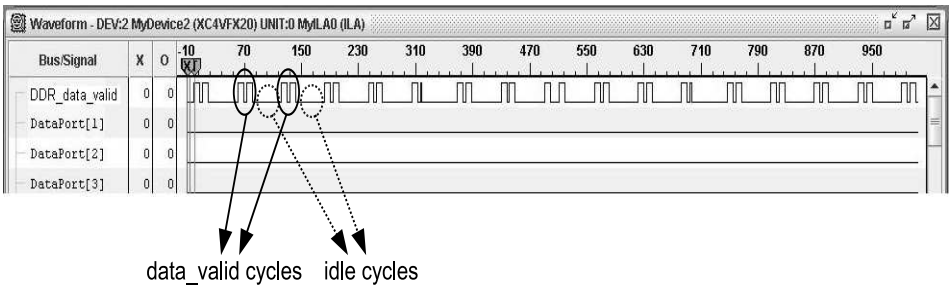


Figure 4.15. Interleaved data transport from the DDR/DDR2 projection LUT

Chapter 5

Summary

5.1 Conclusion

In this thesis, we have presented a hierarchical computation platform based on the ATCA standard and FPGA technologies. It features high bandwidth interconnections, large processing power and large storage capacity. Specifically in each ATCA crate which hosts 14 compute nodes, up to 1890 Gbps inter-FPGA on-board channels, 1456 Gbps inter-board backplane connections, 728 Gbps full-duplex optical links, 70 Gbps Ethernet, 140 GBytes DDR2 SDRAM, and all computing resources of 70 Xilinx Virtex-4 FX60 FPGAs are available. The system is easily scalable, reconfigurable, and well suited for various applications of nuclear and particle physics experiments, for large-scale data acquisition and triggering computation.

In order to easily and quickly develop different applications on the platform, we propose a hardware/software co-design approach. With a reasonable functional partition between the embedded microprocessor and the FPGA fabric, slow control tasks and Ethernet protocol processing are implemented in software programs running on top of device drivers and the embedded Linux operating system, while the application-specific computation is customized in hardware as processing modules to accelerate performance. This development approach is expected to be standard for various application designs, where only specific processing modules and the corresponding software are customized, trying to reuse the system design as much as possible.

Driven by the HADES upgrade project, the particle track reconstruction computation has been studied and implemented as a hardware processor on

FPGA. Implementation results have been listed to show the integration feasibility. Experimental results reveal significant performance speedup of 10.8 - 24.3 times compared to the software solutions on a Xeon 2.4 GHz commodity server.

5.2 Future Work

Future work is planned in the following respects:

- All pattern recognition algorithms are supposed to be implemented on FPGAs as modular designs. They should be reasonably partitioned and distributed in multiple FPGA nodes to process the massive data from detectors in parallel and/or pipeline.
- More studies and experiments are to be done on the overall network architecture for performance evaluation and improvement. In the next version of compute nodes, we will produce 3 or 4 PCBs which are interconnected by the ATCA backplane for network experiments.
- The switching mechanism using packet switching or circuit switching will be studied and the switch design in the switch FPGA is to be implemented.
- More FPGA features such as the dynamic partial reconfiguration are interesting to be exploited in our development methodology. It is useful during the experiments when custom processing units need to be modified or re-parameterized while the rest system cannot stop functioning normally.
- We are going to program the software applications running on the embedded PowerPC microprocessor and the Linux OS, which provide a user interface to operators for monitoring the system status and issuing control commands.

Appendix A

Porting Linux on Xilinx FPGA Boards

A.1 Introduction

Xilinx Virtex-4 FX and Virtex-5 FXT series FPGAs have embedded hardcore PowerPC processors (PowerPC 405 in Virtex-4 and PowerPC 440 in Virtex-5 respectively) on the die. For embedded systems designs, installing an Operating System (OS) makes the application development easier, more portable and more flexible when compared to the standalone approach. Among all kinds of embedded OSes, Linux is very preferable to many engineers and researchers due to its merits of open-source, stable running, a wide range of hardware architecture support, advanced memory management techniques, powerful network support, etc.. In this technical report, we will introduce our work on porting embedded Linux on the Xilinx FPGA boards.

The experimental devices we adopted are Xilinx commercial boards ML405 and ML403, with Virtex-4 FX20 and FX12 FPGAs installed respectively. We worked also on our customized PCBs, and it needs a little more effort and tricks to run all customized peripherals on the board. Concerning the PowerPC 440 architecture in Virtex-5 FXT family FPGAs, it shares the same principle and process to bring up the Linux kernel. Recently Xilinx has added the Memory Management Unit (MMU) functionality to its soft-core CPU Microblaze. It enables the possibility to run a "full Linux" on the Microblaze architecture. However for the MMU-less Microblaze design,

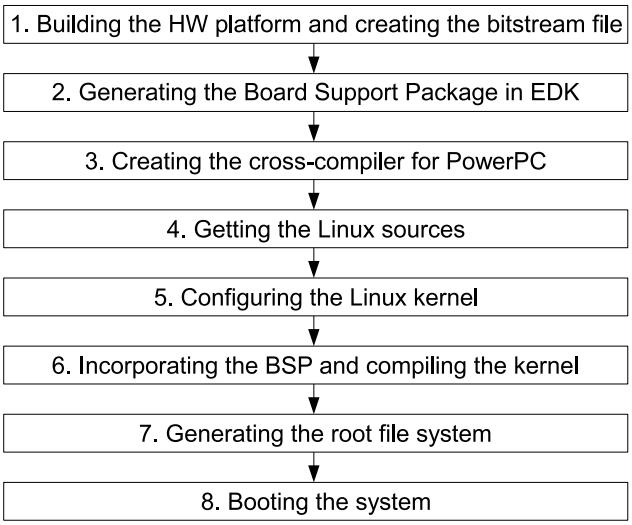


Figure A.1. Steps to bring up Linux on Xilinx boards

μClinux should be used instead. In this report, we focus only on the PowerPC 405 architecture, on which we have worked for our project.

A.2 Steps to Port Linux on Xilinx Boards

To port Linux on FPGA boards concerns both hardware and software design effort. First of all, the hardware platform design should be verified to work correctly. Then aiming at the specific hardware architecture, the kernel image containing device drivers is to be compiled with cross-compilation tools running on a host system. Main steps are briefly shown in Figure A.1. We will discuss more details as follows.

1. Building the HW platform and creating the bitstream file

   To get started, build a hardware project using Base System Builder in EDK for the specific board. Necessary modules should be included in the design, in our case specifically PowerPC 405 processor, Tri-mode Ethernet MAC, SDRAM controller, PLB_BRAM, interrupt controller, UART, external memory controller for flash memory, and other needed customized modules. They are interconnected altogether, either in the PLB/OPB bus-based topology or the MPMC-based architecture.

After synthesis and implementation, the hardware bitstream is generated. A bootloader program can be assembled in the bitstream file and located in the reset vector of the PowerPC processor to load the OS kernel from the flash memory into the DDR/DDR2 SDRAM and start executing. Alternatively when using JTAG to debug the system, bootloader is not needed and the OS kernel image file can be downloaded to DDR/DDR2 by PowerPC through the JTAG chain.

2. Generating the Board Support Package (BSP) in EDK

   The BSP contains device drivers and header files required by the OS kernel. It is automatically generated by EDK, with the specification of board parameters.

   In EDK's "Software Platform Settings" window:

   (a) Set PPC405_0 processor to generate BSP for Linux kernel (e.g. linux_2_6 and version 1.01b in EDK 10.1)

   (b) Specify the OS and library parameters for the board, including connected peripherals, memory size, UART clock frequency, target directory, etc.. All the peripheral information will be included in the generated header file "xparameters_ml40x.h".

   (c) Set all device drivers with the same name as the peripherals.

   Then "Generate Libraries and BSPs" will start generating the BSP package, which will be located in the folder specified by the target directory.

3. Creating the cross-compiler for PowerPC

   A cross-compiler will compile C codes to PowerPC machine code on an arbitrary host. In our case we use an X86 PC running Linux as the host to develop PowerPC executables.

   Building a cross-compiling toolchain is typically regarded as an awkward and error-prone process. Fortunately, Dan Kegel [41] has developed a set of shell scripts to automate the toolchain build process. After successful compilation and installation on the host machine, the cross-compilation toolchain provides us binaries, for instance powerpc-405-linux-gnu-gcc, powerpc-405-linux-gnu-as, powerpc-405-linux-gnu-ld, and so on.

4. Getting the Linux sources

   Different versions of Linux kernels may be downloaded from the Linux kernel archives at http://www.kernel.org/. To make the development easier, Xilinx, Secretlab [42] and MontaVista [43] have all set up their kernel trees for PowerPC 405 architecture at http://git.xilinx.com, http://git.secretlab.ca/, and http://source.mvista.com/ respectively. In their kernel versions, device drivers of some standard IP cores in Xilinx FPGAs have been included in the source package. During the kernel configuration, they can be enabled optionally. In our work, we have downloaded and brought up two version kernels, 2.6.10 and 2.6.24 for the bus-based and the MPMC-based PowerPC 405 architectures respectively.

5. Configuring the Linux kernel

   The Linux kernel will be built using the above created cross-compiler. The file .config is used to control which features or functions in Linux are included in the kernel image file. It is not recommended to edit .config directly. Instead, "make xconfig" or "make menuconfig" should be used to configure the kernel. Figure A.2 shows the top level configuration interface of "make menuconfig" and in the hierarchical structure, many options are available to be enabled or disabled.

   If not properly configured, the kernel is very error prone during compilation. Hence the best solution is to make a minimal ".config" file as a starting point and add more options later step by step. The following items show a fundamental example configuration for our platform in the kernel version of 2.6.24.

   - General Setup
     * Prompt for development and/or incomplete code/drivers

   - Enable Loadable Module Support
     * Module unloading
     * Module versioning support
     * Automatic kernel module loading

   - Processor
     Processor Type → 40x
     * Math emulation
     IBM 4xx options

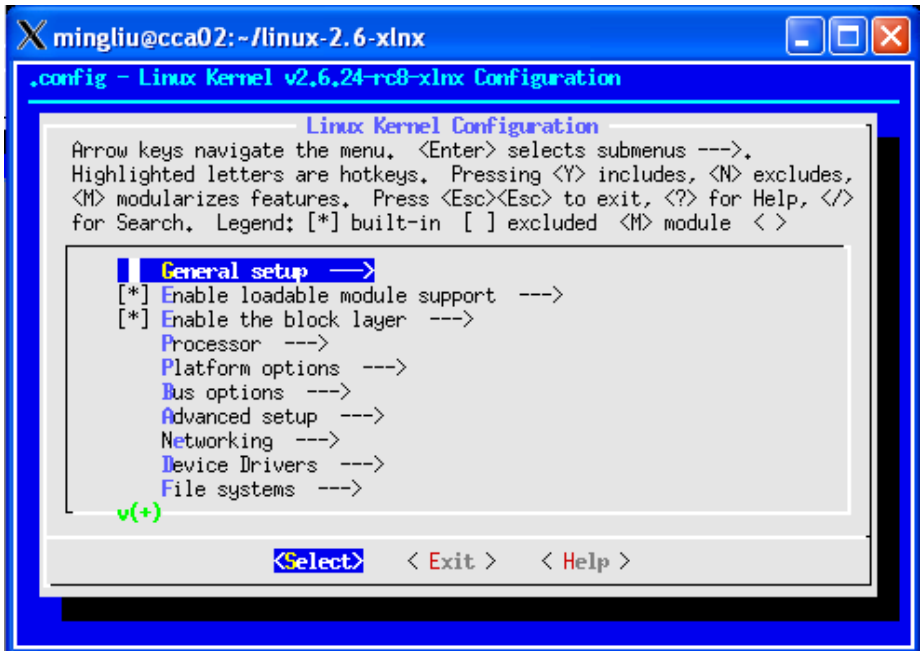


Figure A.2. Linux kernel configuration interface

Machine type (Xilinx-ML405)
TTYS0 device and default console (UART0)

- Platform Options
  * Kernel support for ELF binaries
  * Kernel support for MISC binaries
  * Default bootloader kernel arguments
  ("console=ttyUL0, 38400 root=/dev/nfs ip=192.168.0.4:192.168.0.1:192.168.0.1:255.255.255.0, rw nfsroot=192.168.0.1:/home/mingliu/ml403_rootfs mem=64M")

- Networking
  * Networking support
  Networking options
    * Packet socket
    * Unix domain sockets
    * TCP/IP networking

- Device Drivers
  - * Memory Technology Device (MTD) support
    - * MTD partitioning support
    - * Direct char device access to MTD devices

    RAM/ROM/FLASH chip drivers →
    - * Detect flash chips by Common Flash Interface (CFI) probe
    - * Support for Intel/Sharp flash chips

    Mapping drivers for chip access →
    - * Support non-linear mappings of flash chips
    - * CFI Flash device in physical memory map

    (0xff000000) Physical start address of flash mapping

    (0x00800000) Physical length of flash mapping

    (4) Bank width in octets
  - * Block devices
    - * Loopback device support
    - * Network block device support
  - * Network device support
    - * Ethernet (1000 Mbit)
      - * Xilinx LLTEMAC 10/100/1000 Ethernet MAC driver

  Character devices
  - * Virtual terminal
  - * Support for console on virtual terminal

    Serial drivers →
    - * Xilinx uartlite serial port support
    - * Support for console on Xilinx uartlite serial port

- File Systems
  - * Kernel automounter support

  Pseudo filesystems →
    - * /proc file system support
    - * sysfs file system support
    - * Virtual memory file system support
  - * Network File Systems
    - * NFS file system support
    - * Provide NFSv3 client support
    - * Root file system on NFS
    - * Support for rpcbind version 3 & 4

- Kernel hacking
  - * Kernel debugging
  - * Compile the kernel with debug info

6. Incorporating the BSP and compiling the kernel

   If the kernel source code is from Xilinx, Secretlab or MontaVista, they have already integrated device drivers in the package. The only thing one needs to do is to copy and overwrite the header file "xparameters_ml40x.h", where system parameters are defined. If using the package from the Linux kernel archives, generated BSP files should be copied into the kernel source and correspondingly modify "Kconfig" and "Makefile" to visualize and enable driver options in the kernel configuration interface.

   Before compilation, the architecture and software toolchain in the "Makefile" need to be specified as:
   ARCH := ppc
   CROSS_COMPILE = powerpc-405-linux-gnu-
   Then "make dep" and "make zImage" will start the compilation and generate the executable file "zImage.elf" in "arch/ppc/boot/images/" directory. "zImage.elf" is the executable file which will be downloaded to the memory for starting the Linux OS. By the "powerpc-405-linux-gnu-objcopy" tool, the .elf file is also able to be converted into the .srec format, which is used for the flash memory storage.

7. Generating the root file system

   The root file system contains the startup files necessary to get the system up to a fully running state where users can log in. It provides also some utilities (ls, chmod, ...) as well as hierarchical file directories. In our case the root file system will reside in a desktop PC and be accessed through Network File System (NFS).

   The root file system can be generated by BusyBox [44]. After generation, all folders are copied to the PC at "/home/mingliu/ml403_rootfs", which behaves as the NFS server.

8. Booting the system

   If all the previous steps are correct, the system is runnable now. No matter what booting mechanism is used (JTAG, flash memory, ...), the booting information of the Linux kernel will show on the UART console terminal. The information can be referred to as follows. Except no Graphical User Interface (GUI), the embedded PowerPC platform with the Linux OS can be operated in the same way as we do on a normal desktop PC with Linux.

```
loaded at:     00400000 0059F19C
board data at: 0059D120 0059D19C
relocated to:  004050FC 00405178
zimage at:     00405F73 0059CCAC
avail ram:     005A0000 08000000

Linux/PPC load: root=/dev/nfs ip=192.168.0.4:192.168.0.1:192.168.0.1:255.255.255.0 rw
nfsroot=192.168.0.1:/home/mingliu/ml403_rootfs console=ttyUL0,38400 mem=64M
Uncompressing Linux...done.
Now booting the kernel
Linux version 2.6.24-rc8-xlnx-g1db182b8-dirty (mingliu@cca02) (gcc version 3.4.1) #19 Fri Jul 18 14:11:15 CEST 2008
Xilinx Generic PowerPC board support package (Xilinx ML405) (Virtex-4 FX)
Zone PFN ranges:
  DMA             0 ->    16384
  Normal      16384 ->    16384
  HighMem     16384 ->    16384
Movable zone start PFN for each node
early_node_map[1] active PFN ranges
    0:        0 ->    16384
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 16256
Kernel command line: root=/dev/nfs ip=192.168.0.4:192.168.0.1:192.168.0.1:255.255.255.0 rw
nfsroot=192.168.0.1:/home/mingliu/ml403_rootfs console=ttyUL0,38400 mtdparts=flash_mem:5M(kernel),2M(others),-
(bitstream) mem=64M
Xilinx INTC #0 at 0x81800000 mapped to 0xFDFFF000
PID hash table entries: 256 (order: 8, 1024 bytes)
Console: colour dummy device 80x25
Dentry cache hash table entries: 8192 (order: 3, 32768 bytes)
Inode-cache hash table entries: 4096 (order: 2, 16384 bytes)
Memory: 61312k available (2552k kernel code, 936k data, 84k init, 0k highmem)
SLUB: Genslabs=11, HWalign=32, Order=0-1, MinObjects=4, CPUs=1, Nodes=1
Mount-cache hash table entries: 512
net_namespace: 64 bytes
NET: Registered protocol family 16
Registering device uartlite:0
Fixup MAC address for xilinx_lltemac:0
Registering device xilinx_lltemac:0
NET: Registered protocol family 2
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 2048 (order: 2, 16384 bytes)
TCP bind hash table entries: 2048 (order: 1, 8192 bytes)
TCP: Hash tables configured (established 2048 bind 2048)
TCP reno registered
sysctl table check failed: /kernel/l2cr .1.31 Missing strategy
Call Trace:
[c3c11e50] [c0008b70] show_stack+0x40/0x194 (unreliable)
[c3c11e90] [c003aed4] set_fail+0x68/0x80
[c3c11eb0] [c003b4ec] sysctl_check_table+0x600/0x77c
[c3c11ef0] [c003b4d4] sysctl_check_table+0x5e8/0x77c
[c3c11f30] [c002605c] register_sysctl_table+0x64/0xb4
[c3c11f50] [c034379c] register_ppc_htab_sysctl+0x18/0x2c
[c3c11f60] [c034282c] kernel_init+0x94/0x2bc
[c3c11ff0] [c0004d58] kernel_thread+0x44/0x60
Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
fuse init (API version 7.9)
io scheduler noop registered
io scheduler anticipatory registered
io scheduler deadline registered
io scheduler cfq registered (default)
uartlite.0: ttyUL0 at MMIO 0x84000003 (irq = 3) is a uartlite
console [ttyUL0] enabled
loop: module loaded
nbd: registered device at major 43
xilinx_lltemac xilinx_lltemac.0: MAC address is now  0: a:35: 1: 2: 3
xilinx_lltemac xilinx_lltemac.0: XLITemac: using DMA mode.
XLITemac: Dma base address: phy: 0x84600100, virt: 0xc5008100
XLITemac: buffer descriptor size: 32768 (0x8000)
XLITemac: Allocating DMA descriptors with kmalloc<6>XLITemac: (buffer_descriptor_init) phy: 0x3d18000, virt: 0xc3d18000,
size: 0x8000
XTemac: PHY detected at address 7.
xilinx_lltemac xilinx_lltemac.0: eth0: Xilinx TEMAC at 0x81C00000 mapped to 0xC5004000, irq=2
console [netcon0] enabled
```

```
Linux version 2.6.24-rc8-xlnx-g1db182b8-dirty (mingliu@cca02) (gcc version 3.4.1) #19 Fri Jul 18 14:11:15 CEST 2008
Xilinx Generic PowerPC board support package (Xilinx ML405) (Virtex-4 FX)
Zone PFN ranges:
  DMA             0 ->    16384
  Normal      16384 ->    16384
  HighMem     16384 ->    16384
Movable zone start PFN for each node
early_node_map[1] active PFN ranges
    0:        0 ->    16384
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 16256
Kernel command line: root=/dev/nfs ip=192.168.0.4:192.168.0.1:192.168.0.1:255.255.255.0 rw
nfsroot=192.168.0.1:/home/mingliu/ml403_rootfs console=ttyUL0,38400 mem=64M
Xilinx INTC #0 at 0x81800000 mapped to 0xFDFFF000
PID hash table entries: 256 (order: 8, 1024 bytes)
Console: colour dummy device 80x25
Dentry cache hash table entries: 8192 (order: 3, 32768 bytes)
Inode-cache hash table entries: 4096 (order: 2, 16384 bytes)
Memory: 61312k available (2552k kernel code, 936k data, 84k init, 0k highmem)
SLUB: Genslabs=11, HWalign=32, Order=0-1, MinObjects=4, CPUs=1, Nodes=1
Mount-cache hash table entries: 512
net_namespace: 64 bytes
NET: Registered protocol family 16
Registering device uartlite:0
Fixup MAC address for xilinx_lltemac:0
Registering device xilinx_lltemac:0
NET: Registered protocol family 2
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 2048 (order: 2, 16384 bytes)
TCP bind hash table entries: 2048 (order: 1, 8192 bytes)
TCP: Hash tables configured (established 2048 bind 2048)
TCP reno registered
sysctl table check failed: /kernel/l2cr .1.31 Missing strategy
Call Trace:
[c3c11e50] [c0008b70] show_stack+0x40/0x194 (unreliable)
[c3c11e90] [c003aed4] set_fail+0x68/0x80
[c3c11eb0] [c003b4ec] sysctl_check_table+0x600/0x77c
[c3c11ef0] [c003b4d4] sysctl_check_table+0x5e8/0x77c
[c3c11f30] [c002605c] register_sysctl_table+0x64/0xb4
[c3c11f50] [c034379c] register_ppc_htab_sysctl+0x18/0x2c
[c3c11f60] [c034282c] kernel_init+0x94/0x2bc
[c3c11ff0] [c0004d58] kernel_thread+0x44/0x60
Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
fuse init (API version 7.9)
io scheduler noop registered
io scheduler anticipatory registered
io scheduler deadline registered
io scheduler cfq registered (default)
uartlite.0: ttyUL0 at MMIO 0x84000003 (irq = 3) is a uartlite
console [ttyUL0] enabled
loop: module loaded
nbd: registered device at major 43
xilinx_lltemac xilinx_lltemac.0: MAC address is now  0: a:35: 1: 2: 3
xilinx_lltemac xilinx_lltemac.0: XLITemac: using DMA mode.
XLITemac: Dma base address: phy: 0x84600100, virt: 0xc5008100
XLITemac: buffer descriptor size: 32768 (0x8000)
XLITemac: Allocating DMA descriptors with kmalloc<6>XLITemac: (buffer_descriptor_init) phy: 0x3d18000, virt: 0xc3d18000,
size: 0x8000
XTemac: PHY detected at address 7.
xilinx_lltemac xilinx_lltemac.0: eth0: Xilinx TEMAC at 0x81C00000 mapped to 0xC5004000, irq=2
console [netcon0] enabled
netconsole: network logging started
physmap platform flash device: 00800000 at ff000000
physmap-flash.0: Found 2 x16 devices at 0x0 in 32-bit bank
 Intel/Sharp Extended Query Table at 0x0031
Using buffer write method
cfi_cmdset_0001: Erase suspend on write enabled
RedBoot partition parsing not available
mtd: Giving out device 0 to physmap-flash.0
mice: PS/2 mouse device common for all mice
TCP cubic registered
NET: Registered protocol family 1
NET: Registered protocol family 17
```

```
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
eth0: XLITemac: Options: 0x3fa
eth0: XLITemac: allocating interrupt 0 for dma mode tx.
eth0: XLITemac: allocating interrupt 1 for dma mode rx.
eth0: XLITemac: speed set to 1000Mb/s
eth0: XLITemac: Send Threshold = 24, Receive Threshold = 4
eth0: XLITemac: Send Wait bound = 254, Receive Wait bound = 254
IP-Config: Complete:
     device=eth0, addr=192.168.0.4, mask=255.255.255.0, gw=192.168.0.1,
    host=192.168.0.4, domain=, nis-domain=(none),
    bootserver=192.168.0.1, rootserver=192.168.0.1, rootpath=
Looking up port of RPC 100003/2 on 192.168.0.1
Looking up port of RPC 100005/1 on 192.168.0.1
VFS: Mounted root (nfs filesystem).
Freeing unused kernel memory: 84k init

Welcome to ML405 powerpc linux 2.6.24, E.I.S. edition

Starting system...
mounting /proc: done.
Mounting '/' read-write: done.
brining up loopback interface: done.
Mounting /tmp: done.
Starting syslogd: done.
Starting klogd: done.
Starting inetd: done.
System started.

ML405 powerpc linux 2.6.24-pre7 E.I.S. edition
192.168.0.4 login: root


Welcome to the ML405, EIS edition

Be careful, it's blue.

#
```

A.3 Summary

In this report, we have presented the work to install an open-source Linux on Xilinx Virtex-4 FX series FPGA boards. The discussion above includes only summaried steps which lead the direction to a running Linux OS. Much more effort needs to be devoted to solve specific problems both in hardware and in software, for example, kernel compilation errors and hardware design bugs.

References

[1] High Acceptance Di-Electron Spectrometer (HADES) @ GSI, Darmstadt, Germany, www-hades.gsi.de.

[2] antiProton ANnihilations at DArmstadt (PANDA) @ GSI, Darmstadt, Germany, www.gsi.de/panda.

[3] BEijing Spectrometer (BES) @ Institute of High Energy Physics, Beijing, China, http://bes.ihep.ac.cn/bes3/index.html.

[4] The Large Hadron Collider (LHC) @ CERN, the European Organization for Nuclear Research, http://lhc.web.cern.ch/lhc/.

[5] Wide Angle Shower Apparatus (WASA) @ Research Center Juelich, Juelich, Germany, http://www.fz-juelich.de/ikp/wasa/index.shtml.

[6] Pieter van der Wolf, " Applications and Memory Organization", *Design Automation and Test Conference (DATE) Tutorial - NoCs at the Age of six*, Apr. 2007.

[7] I. Froehlich, A. Gabriel, D. Kirschner, J. Lehnert, E. Lins, M. Petri, T. Perez, J. Ritman, D. Schaefer, A. Toia, M. Traxler, and W. Kuehn, "Pattern recognition in the HADES spectrometer: an application of FPGA technology in nuclear and particle physics", *In Proc. of the 2002 IEEE International Conference on Field-Programmable Technology*, pages 443-444, Dec. 2004.

[8] Michael Traxler, "Real-time dilepton selection for the HADES spectrometer", November 2001, Ph.D thesis, II.Physikalisches Institut, Justus-Liebig-Universitaet Giessen.

[9] C. Hinkelbein, A. Kugel, R. Manner, M. Muller, M. Sessler, H. Simmler and H. Singpiel, "Pattern recognition algorithms on FPGAs and CPUs

for the ATLAS LVL2 trigger", *IEEE Transactions on Nuclear Science*, Volume 48, Issue 3, Part 1, pp. 296-301, Jun. 2001.

[10] G. Estrin, Organization of Computer Systems – The Fixed Plus Variable Structure Computer, *In Proc. of the Western Joint Computer Conference*, New York, 1960, pp. 33-40.

[11] G. Estrin, Reconfigurable Computer Origins: The UCLA Fixed-Plus-Variable (F+V) Structure Computer, *IEEE Annals of the History of Computing*, Volume 24, Issue 4, Oct.-Dec. 2002, pages 3-9.

[12] R. Merl, F. Gallegos, C. Pillai, F. Shelley, B. J. Sanchez and A. Steck, "High speed EPICS data acquisition and processing on one VME board", *In Proc. of the 2003 Particle Accelerator Conference*, volume 4, pages 2518-2520, May. 2003.

[13] Y. Tsujita, J. S. Lange, and C. Fukunaga, "Construction of a compact DAQ-system using DSP-based VME modules", *In Proc. of the 11th IEEE NPSS Real Time Conference*, pages 95-98, Jun. 1999.

[14] M. Drochner, W. Erven, P. Wustner, and K. Zwoll, "The second generation of DAQ-Systems at COSY", *IEEE Transactions on Nuclear Science*, Volume 45, Issue 4, Part 1, pp. 1882-1888, Aug. 1998.

[15] Y. Nagasaka, I. Arai and K. Yagi, "Data acquisition and event filtering by using transputers", *In Proc. of the Nuclear Science Symposium and Medical Imaging Conference 1991*, pp. 841-844, Nov. 1991.

[16] N. Bhatia, S. R. Alam, and J. S. Vetter, "Performance Modeling of Emerging HPC Architectures", *In Proc. of HPCMP Users Group Conference 2006*, pp. 367-373, Jun. 2006.

[17] J. Fernando, D. Dalessandro, A. Devulapalli, and K. Wohlever, "Accelerated FPGA Based Encryption", *In Proc. of the 2005 Cray Users Group Conference*, May. 2005.

[18] C. B. Cameron, "Using FPGAs to Supplement Ray-Tracing Computations on the Cray XD-1", *In Proc. of the DoD High Performance Computing Modernization Program Users Group Conference 2007*, pp. 359-363, Jun. 2007.

[19] C. Hinkelbein, A. Kugel, R. Manner, M. Muller, M. Sessler, H. Simmler and H. Singpiel, Pattern Recognition Algorithms on FPGAs and CPUs for the ATLAS LVL2 Trigger, *IEEE Transactions on Nuclear Science*, Volume 48, Issue 3, Part 1, pp. 296-301, Jun. 2001.

[20] Dini Group, "www.dinigroup.com"

[21] C. Chang, J. Wawrzynek, and R. W. Brodersen, "BEE2: a high-end reconfigurable computing system", *IEEE Design & Test of Computers*, Volume 22, Issue 2, pp. 114-125, March-April 2005.

[22] A. Tkachenko, D. Cabric, and R. W. Brodersen, "Cognitive Radio Experiments using Reconfigurable BEE2", *In Proc. of the Fortieth Asilomar Conference on Signals, Systems, and Computers 2006*, pp. 2041-2045, Oct. -Nov. 2006.

[23] PANDA Collaboration, "Technical Progress Report for PANDA", http://www-panda.gsi.de/db/papersDB/PC19-050217_panda_tpr.pdf, February 2005.

[24] PCI Industrial Computers Manufactures Group (PICMG), PICMG 3.0 Advanced Telecommunications Computing Architecture (ATCA) specification, Dec. 2002.

[25] A. X. Widmer, P. A. Franaszek, "A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code", *IBM Journal of Research and Development*, Volume 27, Issue 5, 1983, pages 440-451.

[26] Xilinx, Inc., "RocketIO Transceiver User Guide", UG024(v3.0), Feb, 2007.

[27] Xilinx, Inc., "Virtex-4 Product Table", Jan, 2007.

[28] Hal Stern, Mike Eisler, and Ricardo Labiaga, "Managing NFS and NIS (Second Edition)", O'REILLY & Associates, Inc., ISBN: 1-56592-510-6.

[29] Xilinx, Inc., "ML405 Evaluation Platform User Guide", UG080(v1.3) May 2, 2007.

[30] Xilinx, Inc., Multi-Port Memory Controller (MPMC) (v4.01.a), DS643, March 12, 2008.

[31] Brent Nelson and Brad Baillio, "Configuring and Installing Linux on Xilinx FPGA Boards", November, 2005, BYU Configurable Computing Laboratory.

[32] Dan Burke, James Player, and Nacho Navarro, "Building a Xilinx ML300/ML310 Linux Kernel", November, 2004, UIUC Soft Systems Lab.

[33] Wolfgang Klingauf and Uwe Klingauf, "Virtex2Pro & Linux", January, 2004, www.klingauf.de.

[34] Jonathon W. Donaldson, "Porting MontaVista Linux to the XUP Virtex-II Pro Development Board", August, 2006, Master Thesis, Department of Computer Science, Rochester Institute of Technology.

[35] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman, "Linux Device Drivers (Third Edition)", O'REILLY & Associates, Inc., ISBN: 0-596-00590-3.

[36] Apache HTTP Server Project, "http://httpd.apache.org/"

[37] The Common Gateway Interface, "http://hoohoo.ncsa.uiuc.edu/cgi/"

[38] HADES collaboration, "A proposal for a High Acceptance Di-Electron Spectrometer", 1994, GSI Darmstadt.

[39] Daniel Kirschner, "Level 3 Trigger Algorithm and Hardware Platform for the HADES Experiment", Oct. 2007, Ph.D thesis, II. Physics Institute of Justus-Liebig-University Giessen.

[40] Ming Liu, Wolfgang Kuehn, Zhonghai Lu, and Axel Jantsch, "System-on-an-FPGA Design for Real-time Particle Track Recognition and Reconstruction in Physics Experiments", *In Proc. of the 11th EUROMICRO Conference on Digital System Design*, Parma, Italy, Sep. 2008.

[41] Dan Kegel's Web Hostel, "http://www.kegel.com/"

[42] Secret Lab, "http://www.secretlab.ca/"

[43] MontaVista Linux, "http://www.mvista.com/"

[44] Busybox, "http://www.busybox.net/"