



“Initial Report on protocols FPGA Implementation”

D6.1

'DICONET_D6 1_WP6_v3.doc'

Version: 3

Last Update: 26/3/2010 2:44:00 AM

Distribution Level: PU



The DICONET Project Consortium groups the following Organizations:

Partner Name	Short name	Country
JCP-Consult	JCP	FR
Research and Education Laboratory in Information Technologies	AIT	GR
Center of REsearch And Telecommunication Experimentations for NETworked communities	Create-NET	IT
Institut Telecom, Télécom ParisTech	TPT	FR
Huawei Technologies Düsseldorf GmbH	HWDU	DE
Interdisciplinair Instituut voor Breedband Technologie, VZW	IBBT	BE
Research Academic Computer Technology Institute	CTI	GR
University of Essex	UEssex	UK
Universitat Politècnica de Catalunya	UPC	SP
ADVA AG Optical Networking	ADVA	DE
Deutsche Telekom AG	DTAG	DE
Alcatel-Lucent Bell LabsFrance	A-LBLF	FR
ECI Telecom	ECI	IL

Abstract:

This deliverable reports on architecture, design, implementation and initial test results of DICONET protocol hardware accelerator based on FPGA. The deliverable explains the rationale for using hardware accelerator for DICONET protocols based on FPGA. Furthermore, it proposes a novel architecture and implementation strategy for the DICONET hardware accelerator. Finally, the deliverable report on implementation details of the DICONET hardware accelerator and its test-result. The initial test results indicate significant improvement in performance of the DICONET protocol when the proposed accelerator is deployed.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 216338”



Document Identity

Title:	Initial Report on protocols FPGA Implementation
Subject:	Deliverable
Number:	D6.1
File name:	DICONET_D6 1_WP6_v3.doc
Registration Date:	08/03/2010 18:41:00
Last Update:	26/03/2010 02:44:00

Revision History

No.	Version	Edition	Author(s)	Date
1	0	1	Yixuan Qin, Reza Nejabati, Eduard Escalona, Dimitra Simeonidou	07/02/2010
	Comments:	ToC		
2		2	Yixuan Qin, Reza Nejabati, Eduard Escalona, Dimitra Simeonidou	09/03/2010
	Comments	Content populated		
3		3	Yixuan Qin, Reza Nejabati, Eduard Escalona, Dimitra Simeonidou	26/03/2010
	Comments	Minor revision based on reviewers' comments		
4				
	Comments			
5				
	Comments			
6				
	Comments			
7				
	Comments			
8				
	Comments			
9				
	Comments:			
10				
	Comments:			
11				
	Comments:			
12				
	Comments:			
13				
	Comments:			
14				
	Comments:			
15				
	Comments:			

Table of Contents

LIST OF ACRONYMS	5
LIST OF FIGURES	6
LIST OF TABLES	7
EXECUTIVE SUMMARY	8
1. INTRODUCTION	9
2. DICONET CONTROL PLANE HARDWARE ACCELERATOR	11
2.1. Hardware Accelerator Benefits in DICONET	11
2.2. Hardware Accelerator Options	13
2.2.1. Network Processor Based Approach	13
2.2.2. FPGA Based Approach	14
2.3. DICONET Approach	14
2.3.1. FPGA Background	14
2.3.2. FPGA Design Methods	18
2.3.3. Pure FPGA implementation	19
2.3.4. FPGA + Embedded Processor implementation	19
2.3.5. Host Server + pure FPGA implementation	19
2.3.6. Host Server + FPGA + Embedded Processor implementation	20
3. HARDWARE ACCELERATED Q-TOOL IMPLANTATION	21
3.1. Targeted Hardware	21
3.2. Inter-connection models between FPGA logic and embedded processor in FPGA fabric	22
3.2.1. Embedded processor bus connected model	22
3.2.2. I/O based model	23
3.2.3. Extended instruction set model	23
3.2.4. Shared memory model	23
3.3. DICONET Hardware accelerated QoT estimation tool architecture	23
4. PRELIMINARY TEST RESULTS	26
REFERENCES	28

List of Acronyms

APU	Auxiliary Processor Unit
ASE	Amplifier Spontaneous Emission
ASIC	Application Specific Integrated Circuit
ASON	Automatically Switched Optical Networks
CLB	Configurable Logic Block
CPLD	Complex PLD
DDR	Double Data Rate
DMA	Direct Memory Access
DPBRAM	Dual Port Block RAM
DSP	Digital Signal Processing
EEPROM	Electrically Erasable Programmable ROM
EPROM	Erasable Programmable ROM
FF	Flip Flop
FPGA	Field Programmable Gate Array
FWM	Four-Wave Mixing
FSL	Fast Simplex Link
LUT	Look Up Table
MAC	Media Access Control
NIC	Network Interface Card
NMS	Network Management System
OCC	Optical Connection Controller
OCM	On Chip Memory
PCE	Path Computation Element
PCEP	Path Computation Element Protocol
PLD	Programmable Logic Device
PMD	Polarization Mode Dispersion
PROM	Programmable Read Only Memory
QoS	Quality of Service
QoT	Quality of Transmission
ROM	Read-Only Memory
RWA	Routing and Wavelength Assignment
SPLD	Simple PLD
SPM	Self Phase Modulation
SRAM	Static Random-Access Memory
TED	Traffic Engineering Database
WLAN	Wireless Local Area Network
XPM	Cross Phase Modulation

List of Figures

Figure 1: Centralized/PCE based CP	10
Figure 2: Reconfigurable computing device evolution timeline.....	15
Figure 3: Generic internal structure of CPLD	16
Figure 4: Generic internal structure of FPGA.....	17
Figure 5: Typical internal structure of LUT	17
Figure 6: Modern FPGA	18
Figure 7: DN8000KPCI Function Blocks	21
Figure 8: Different solutions for integrating an embedded processor with the FPGA logic ...	22
Figure 9: Hardware Accelerated QoT estimation tool	24
Figure 10: APU controller Processing Operative block Diagram.....	24
Figure 11: The QoT estimation tool average executing time in two different network sizes v.s. (a) the number of wavelength per link for 100 lightpaths and (b) the number of lightpaths for 32 wavelengths per link.....	26

List of Tables

Table 1: Speed Comparison between FPGA based and Software only processors	12
Table 2: Comparison between Non-Accelerated and Accelerated Q-Tool.....	26

Executive Summary

In highly dynamic networks, the network resource provisioning operations (mainly path calculation, signalling and switching) must be performed as fast as possible. In an impairment aware network with impairment aware control plane, the operations and algorithms required dramatically increase the delay of the bandwidth provisioning process, thus, an accelerated system has to be implemented in order to make the whole provisioning process suitable in a dynamic environment deploying fast network elements.

Traditionally, network control plane software and protocols are in the form of conventional software programs executed on PC or servers. The traditional approach of using PC/servers for control plane execution offers great flexibility at the cost of other design factors like size, power consumption and more importantly severely degraded speed performance.

One of the main achievements of DICONET project is the development of the Quality of Transmission (QoT) estimation tool (Q-Tool) that calculates the Q factor for the existing lightpaths and estimates the Q of the new lightpath to be established in the presence of currently established lightpaths. Then these values are evaluated and depending on the threshold of values the lightpath is accepted or blocked, requiring a new path calculation and the subsequent Q estimations. The implementation of the QoT estimation tool is purely software based and it requires intensive computational resources, thus, in a normal PC the execution time can be in the range of 10 to 1000 seconds depending on the number of lightpaths that we are going to feed to Q-Tool and also number of channels (wavelengths) per link. This performance is not suitable for dynamic lightpath provisioning, which should be in the order of milliseconds.

As the Q-Tool is the main speed bottleneck in the DICONET control plane, therefore, in the dynamic context of the DICONET scenario, the acceleration of the QoT estimation process is crucial to demonstrate the feasibility of deployment in a real environment.

In DICONET for hardware based acceleration of control protocols an FPGA based approach is chosen due to the importance of protocol algorithm and specifically Q-Tool acceleration. During DICONET project, four methods were studied and analysed for hardware accelerator implementation. This section provides an overview of these methods:

- Pure FPGA implementation
- FPGA + Embedded Processor
- Host Server+ pure FPGA implementation
- Host Server+ FPGA + Embedded Processor implementation

Based on the above analysis and study the option “FPGA + Embedded Processor implementation” is identified as the most suitable option for the hardware accelerator of DICONET. Due to the challenge posed by the complexity of the QoT estimation tool (Q-Tool), our approach focuses on the implementation of only the time-critical operations of the Q factor computation in the logic part of the targeted FPGA, and keep the non-time critical operations in software to be implemented in the Embedded processor (IBM PowerPC 405 hard core 300MHz which is embedded inside the FPGA fabric).

The initial results show that the accelerated QoT estimation tool outperforms the non-accelerated one by almost 100% in all the test cases. Furthermore, test results shows that the hardware accelerated Q-Tool scales smoothly and mitigates the impact of increasing load is a very important desirable feature in a real dynamic network to provide a consistent network provisioning delay.

The initial result is very promising and has resulted a paper publication [34]. Further, work is ongoing to improve performance of the hardware accelerated Q-tool and it is expected that initial result to be significantly improved which will be a major achievement for DICONET.

1. Introduction

Generalized Multi-Protocol Label Switching (GMPLS) is the strongest candidate for the control plane of Automatically Switched Optical Networks (ASON) [1]. GMPLS allows dynamic provisioning of network resources by means of signalling and routing protocols [2]. Signalling protocols (e.g. RSVP-TE [3]) are used to request, cancel and modify network connections whilst routing protocols (e.g. OSPF-TE [4]) are mainly used to advertise link properties throughout the network. In standard GMPLS, this information is collected either in a centralised or distributed Traffic Engineering Database (TED) which, in turn, feeds the Routing Controller, that can be implemented using a Path Computation Element (PCE) [5], responsible of the Routing and Wavelength Assignment (RWA). In high-speed transmission bit rates (i.e. bitrate > 10Gbps) supported by new generation of optical networks, physical layer impairments play a critical role on Quality of Transmission (QoT). Many impairment-aware algorithms have been proposed in the literature to include physical layer impairments such as Amplifier Spontaneous Emission (ASE) and Polarization Mode Dispersion (PMD) in order to minimise the blocking probability in dynamic networks, while maintaining an acceptable level of QoT [9].

In order to support impairment awareness in a dynamic optical network, extensions to current GMPLS standards have to be implemented [6]. One of the approaches considers extending the OSPF-TE IGP routing protocol to distribute the impairment characteristics of the network elements and links. These extensions are implemented as new Type-Length-Value objects that allow the inclusion of impairment information [7] in routing protocols. This way, the TED is updated to contain not only basic link properties such as state and bandwidth availability, but also information about physical impairments that affect the QoS of the established connections.

The Path Computation Client (PCC) uses the PCE protocol (PCEP) to request a route to the PCE given source address, destination address and some constraints. Then the PCE uses the extended TED for performing RWA and the QoT is estimated to allow or reject the computed path based on the agreed policies [8]. In DICONET architecture, an extended GMPLS Control Plane based on centralised/distributed PCE models is deployed. Path computation is performed along with QoT estimation for each of the already established connections. Impairment Aware Control Plane function block interaction is shown in Figure 1. The QoT estimation identifies the impact that the new allocated connection would have on the existing connections, to assure that QoT is not degraded. This operation is carried out until a suitable path is found or, if none, the connection is blocked. Upon a successful response, the information is transferred back to the optical connection controller (OCC) via PCEP, and then standard RSVP-TE messages are triggered to set-up the lightpath. If the path reservation fails or the QoT estimation tool gives unacceptable Q factor for alternative paths, the Network Management System (NMS) is notified accordingly.

In highly dynamic networks, the operations involved in the network resource provisioning (mainly path calculation, signalling and switching) need to be performed as fast as possible, as a delay in the provisioning may result in undesired collisions and blocked connections, which can severely affect the performance of the network as well as the guaranteed QoS [9].

Furthermore, emerging and new generation of optical network elements are capable to perform switching at millisecond speed regime [10]. To take advantage of this fast switching speed in a dynamic network environment, it is essential that the control plane algorithms and procedures can be performed at speed higher or comparable to switching speed of these network elements.

In an impairment aware network with impairment aware control plane, the operations and algorithms required dramatically increase the delay of the bandwidth provisioning process, thus, an accelerated system has to be implemented in order to make the whole provisioning process suitable in a dynamic environment deploying fast network elements.

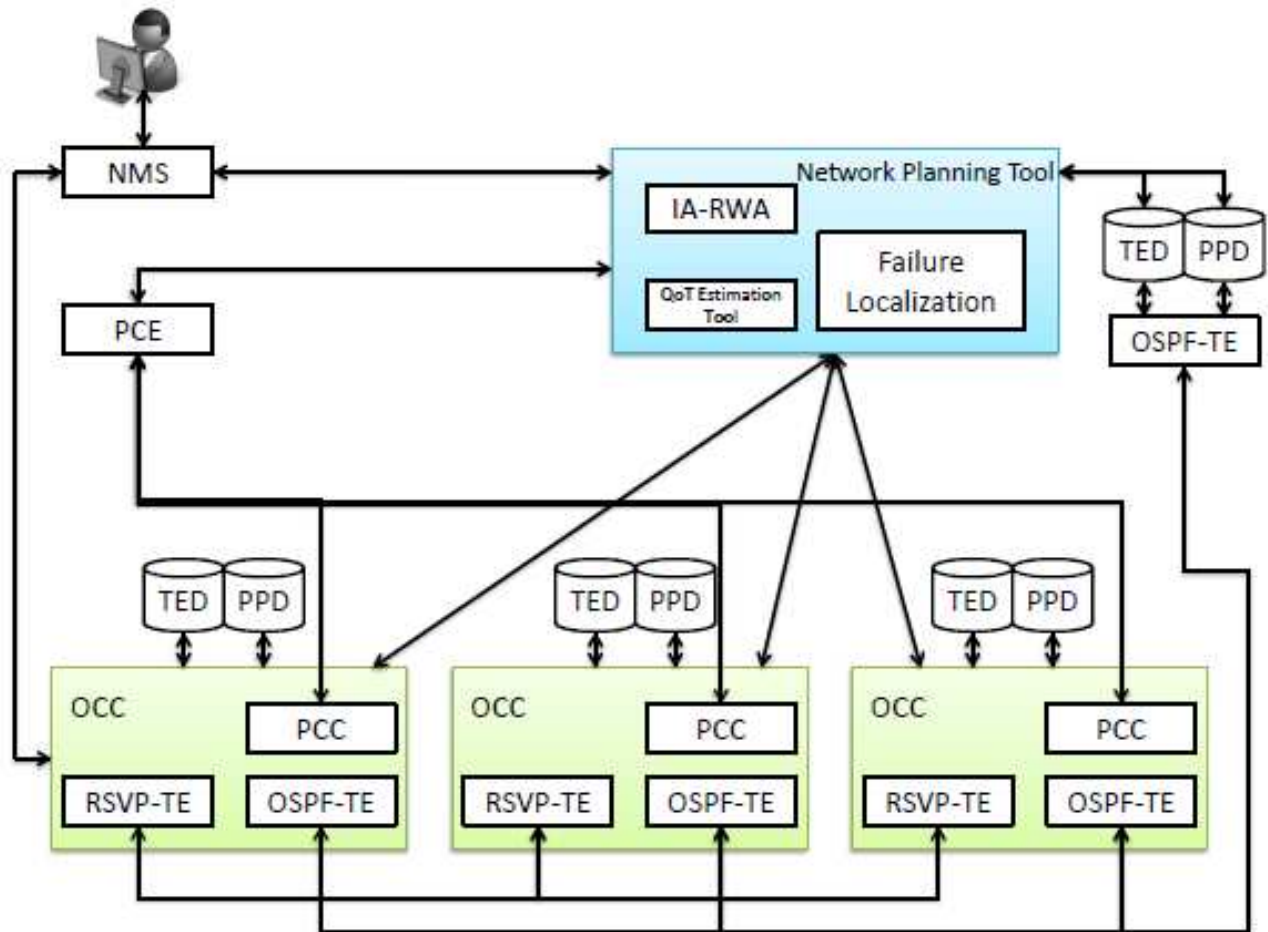


Figure 1: Centralized/PCE based Control Plane

2. DICONET Control Plane Hardware Accelerator

2.1. Hardware Accelerator Benefits in DICONET

Traditionally, network control plane software and protocols are in the form of conventional software programs executed on PC or servers. In a network deploying centralized control plane a single PC/server execute the control plane procedures and configure network elements accordingly. In a network with distributed control plane, there is a network of servers/PCs each control a single network element by executing an instance of the control plane software. The traditional approach of using PC/servers for control plane execution offers great flexibility at the cost of other design factors like size and power consumption. Due to their general purpose nature and architecture, they may not be the strongest performer compared to other type of hardware. PC and servers deploy CPU and there are four basic steps for an instruction to be executed by the processor when running a software procedure: fetch, decode, execute, and write back. These steps are normally pipelined in order to improve efficiency and speed. Apart from pipelining, parallelism is also a natural method to increase performance. However, parallelism does not improve the speed of a single application. It only helps when multiple programs are running on a single processor. Not until recently, have modern processors had more than one processing ‘core’ so that instructions can be processed in parallel by different cores. However, this kind of parallelism is coarse-grained and is not scalable since it is restricted by the number of cores. As a result, the efficiency of the software implemented on a processor cannot be optimized by hardware designers.

In contrast, the hardware accelerator not only offers the programmability of software systems, indeed, it also provides the parallel architectures within the hardware that can be (re)-defined to suit the application. For example, while a 3 GHz Pentium class processor is a highly optimized general purpose programmable “software system”, it can be outperformed by a “hardware accelerator system” working at 300 MHz that has been designed for specific applications. A Hardware accelerator offers the ability to give both high-performance and flexibility. Table 1 summarizes performance results of different types of algorithms in hardware accelerator based on FPGA compared to a general-purpose processor.

Table 1: Speed Comparison between FPGA based and Software only processors

Application	FPGA based	Software only
Hough & intensive Hough processing	2 sec of processing time @20 MHz 370× faster	12 mins processing time Pentium 4 - 3 GHz
AES 1MB data processing rate Encryption Decryption	424 ms/19.7 MB/s 424 ms/19.7 MB/s 13× faster	5,558 ms/1.51 MB/s 5,558 ms/1.51 MB/s
Smith-Waterman sssearch34 from FASTA	100 sec FPGA processing 64× faster	6161 sec processing time Opteron - 2.2 GHz
Multi-dimensional hypercube search	1.06 sec FPGA @140 MHz Virtex II 113× faster	119.5 sec Opteron - 2.2 GHz
Callable Monte-Carlo Analysis 64,000 paths	10 sec of processing @200 MHz FPGA system 10× faster	100 sec processing time Opteron - 2.4 GHz
BJM Financial Analysis 5 million paths	242 sec of processing @61 MHz FPGA system 26× faster	6300 sec processing time Pentium 4 - 1.5 GHz
Mersenne Twister Random Number Generation	319M 32bit integers/sec 3× faster	101M 32bit integers/sec Opteron - 2.2 GHz

One main objective of DICONET is to develop an impairment-aware control plane in dynamic optical networks. This kind of networks are characterized by fast and efficient bandwidth provisioning which is typically enabled by the deployment of an intelligent network control plane on top of a high-speed and fast reconfigurable transport plane. The definition of "fast" in fast bandwidth provisioning depends on many factors, but a connection allocation delay should usually range from milliseconds to few seconds. Other factors such as resilience mechanisms (protection and restoration) or resource contention may infer more stringent delay requirements.

One of the main achievements of DICONET project is the development of the Quality of Transmission estimation tool (Q-Tool) that calculates the Q factor for the existing lightpaths and estimates the Q of the new path to be established. Then these values are evaluated and depending on the threshold of values the path is accepted or blocked, requiring a new path calculation and the subsequent Q estimations. The implementation of the QoT estimation tool is purely software based and it requires intensive computational resources, thus, in a normal PC the execution time can be in the range of 10 to 1000 seconds depending the number of lightpaths and number of channels per link, the network load and deployed RWA algorithms, which is based on the RWA performance tests reported in [11]. This performance is not suitable for dynamic lightpath provisioning, which should be in the order of milliseconds.

As the Q-Tool is the main speed bottleneck in the DICONET control plane, therefore, in the dynamic context of the DICONET scenario, the acceleration of the QoT estimation process is crucial to demonstrate the feasibility of deployment in a real environment.

2.2. Hardware Accelerator Options

There are two options available for hardware based acceleration of DICONET control protocol:

- ❖ Network processor based approach
- ❖ Field Programmable Gate Array (FPGA) based approach

2.2.1. Network Processor Based Approach

Network processors are programmable integrated circuits like general purpose microprocessors, but usually have a customized instruction set optimized for application in the network domain.

These network processors can be deployed in network equipments like routers, switches, firewalls, content-aware switches, application-aware gateways, WLAN and 3G/4G access and aggregation devices, storage arrays, storage networking equipments, servers, and intelligent NICs. Examples of these network processors are [references]:

- Cavium Networks OCTEON Family [12]
- Ezchip NP Network Processor Family [13]
- Intel IXP Network Processor [14]
- Bay Microsystems Chesapeake [15]
- LSI Advanced PayloadPlus® (APP) Family [16]
- Xelerated X10q Family [17]
- Freescale C-Port Network Processor Family [18]
- AMCC nP Network Processor Family [19]

Network processors are generally suitable for application that require acceleration in network tasks like packet processing, switching and traffic management. The advantage of network processors is that their upgrade and modification does not involve hardware changes, only software has to be modified which can be handy and time efficient.

However using network processor for hardware acceleration of DICONET protocols has several disadvantages as listed below:

- Communication between different network processors can be tricky since each device or software suite has a different approach to network processing. The same applies to porting a design from one type to another.
- Many network processors support only a stripped-down version of C (e.g. removing floating-point instructions and pointers), which makes porting existing software more difficult.
- Most importantly, they are not designed for algorithm acceleration which is a crucial requirement in DICONET. In a bit more details, the network processor is manufactured for dataflow-oriented process, for example, used in the hardware firewall for packet filter [20]. Another example is to use network processor in the network router/switch for packet processing and forwarding. The network processor is customised for fast processing the network data and application, however it has neither capability nor flexibility to accelerate any custom algorithms.

In DICONET, porting Q-tool to network processor environment is not feasible and will not provide the required speed up (acceleration).

2.2.2. FPGA Based Approach

There are several advantages for using FPGA in DICONET as compared to using a network processor. An FPGA can be programmed to perform any task thus provide maximum flexibility, especially when new protocols are introduced in network environment. Furthermore, algorithm acceleration on hardware can be realized easily by hardware design techniques, which can vastly improve performance. Once the design has been implemented into an FPGA, the latency and throughput is fixed since it only contains pure logic. This is important in time critical tasks. Additionally, modern FPGA provides versatile hardware intellectual property cores for different applications, e.g. telecommunication, DSP, networking, floating point calculation, and many others. In DICONET, the provided off-shelf IP cores dramatically eases the complex Q factor estimation algorithm implementation which involves lots of floating point mathematic operations. In particular, the modern Xilinx FPGAs are populated with two IBM PowerPC 405 hard cores which are able to run any general software program and communicate with FPGA fabric. This facility gives users great flexibility to port any algorithm from theory to reality, i.e. from Matlab to C and in turn to FPGA, which is perfectly suitable for the DICONET work flow. However, the abovementioned big advantages need great hardware language programming expertise, and upgrading small portions of the design means hardware changes which may also be time consuming.

There are several major FPGA vendors focusing on product suitable for telecommunication applications and protocols. Xilinx, Inc. is the world's largest FPGA developer and manufacturer. Other main vendors include Altera Corporation, Lattice Semiconductor Corporation and Actel Corporation [21][22][23][24]. Manufacturers release different FPGA families to target different markets. Examples are the Xilinx Virtex family (for high performance) and Spartan family (for low cost).

2.3. DICONET Approach

Based on the discussion in section 2.2, an FPGA based approach is chosen for DICONET due to the importance of protocol algorithm and specifically Q-Tool acceleration.

2.3.1. FPGA Background

The FPGA technology is used intensively throughout WP6, readers with the background of optical networking may need brief FPGA background knowledge before understanding the following sections.

FPGA evolved from the programmable logic device (PLD). The original PLD components had the very simple form of programmable read only memory (PROM), until the end of 1970s, when significantly more new versions with additional features became available, named complex PLDs (CPLD) to distinguish from their less-sophisticated ancestors. All the simple PLDs (SPLD) shown in Figure 2 have the option of being programmed in batches in a factory or in the field (field programmable), however programmable logic was hardwired between logic gates [25]. In addition, as indicated by its name (SPLD), it has rather simple internal logic. Consequently the CPLD was designed so as to have a smaller footprint, faster speed, cheaper and more powerful. In simple terms, the CPLD can be thought of as a

combination of up to 50 typical SPLDs with programmable interconnects on a single chip [26].

Figure 3 shows the internal block diagram of a CPLD. Intuitively, each of the six logic blocks is the equivalent of one SPLD. In practice, the CPLD normally has more than six logic blocks.

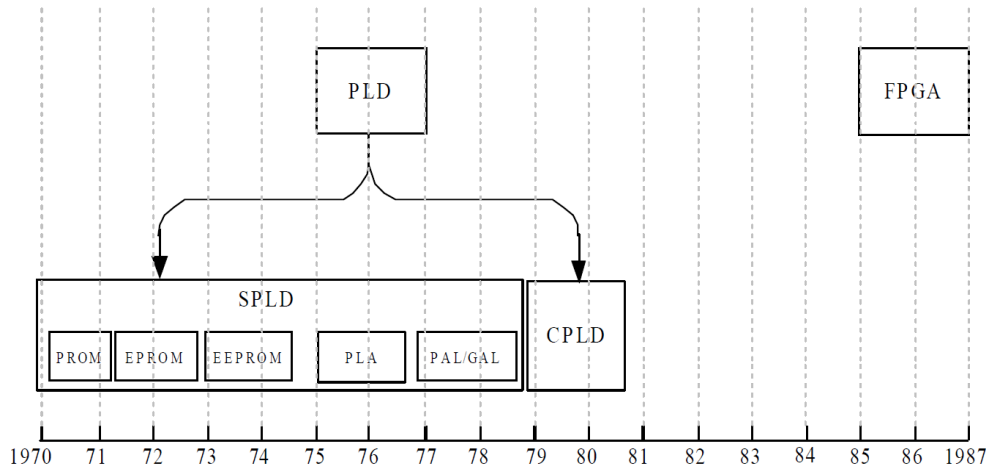


Figure 2: Reconfigurable computing device evolution timeline

Although the CPLD was designed to replace the SPLD, the switch matrix within a CPLD is not fully connected as is the case with the SPLD. In other words, in a CPLD, some of the theoretically possible connections between a given logic block inputs and outputs may not be supported in practice. That is because if the size of SPLD were to be doubled, the size of interconnect array would be quadrupled. Consequently, it could result in a huge decrease in speed, along with higher power dissipation and component costs. Thus the CPLD will normally have less than 100% connectivity. The downside of this is that making full utilisation of a given CPLD is very difficult, even when there are sufficient logic blocks, and requires more complex software design tools to route the signals. The positives of this are that it keeps the speed, power, and cost of these devices scalable.

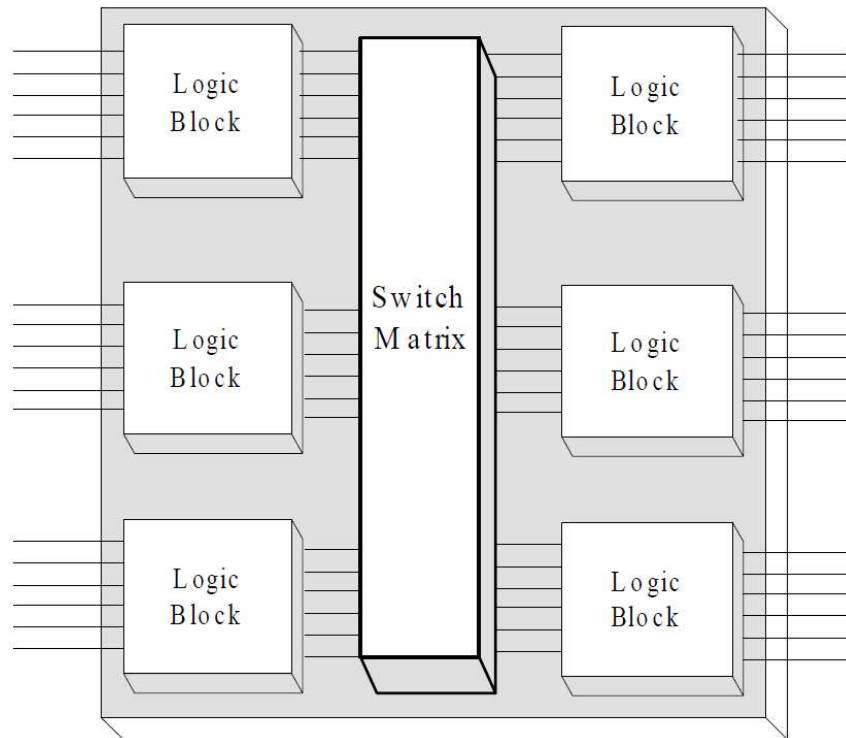


Figure 3: Generic internal structure of CPLD

Even though a SPLD is normally simply called a PLD, in this deliverable, the SPLD and CPLD are still categorised as PLDs which are highly configurable and quick to take to market, but cannot support large or complex logic functions. In contrast with PLD devices, the other end of the spectrum is the application specific integrated circuit (ASIC) which can support extremely large and complex logic functions, but have relatively expensive development and manufacturing costs. In addition, the design cycle is very long, and last but not least, the ASIC is not reprogrammable; once the implementation has been carried out, the ASIC is effectively frozen in the silicon. To overcome this gap in the digital IC continuum, the first commercial viable FPGA—XC2064 was invented by Xilinx Co-Founders, Ross Freeman and Bernard Vonderschmitt in 1985[27].

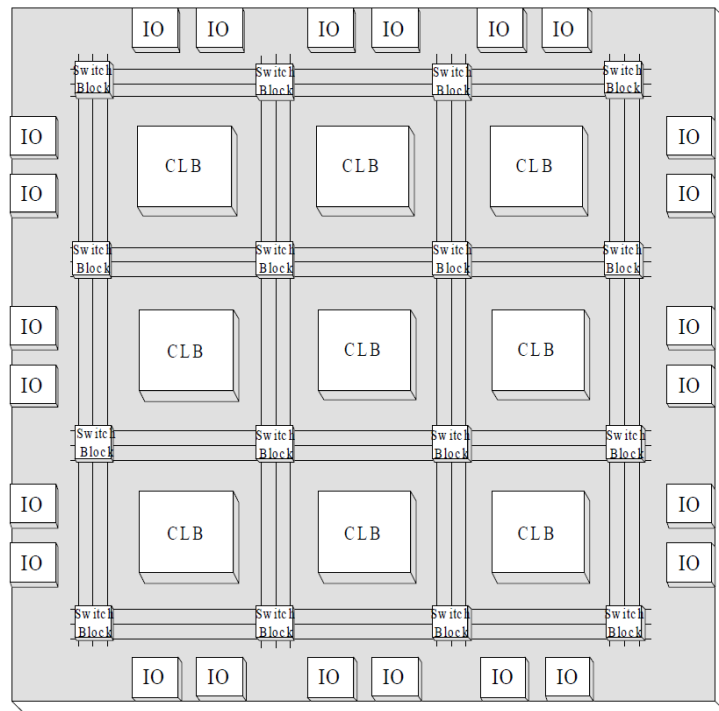


Figure 4: Generic internal structure of FPGA

As shown in Figure 4, the common FPGA architecture consists of an array of configurable logic blocks (CLB), I/O pads, switch blocks and wire segments [28]. A typical CLB consists of a look up table (LUT), a flip flop (FF) and a multiplexer as shown in Figure 5. Newer FPGAs have much more advanced and complex CLBs, for example, the latest Virtex 6 FPGA CLB consists of eight 6 input LUTs, 16 FFs, and other logic resources, i.e. slices, arithmetic and carry chains, RAM, and registers [29]. Both the CLBs and I/O pads have full connectivity

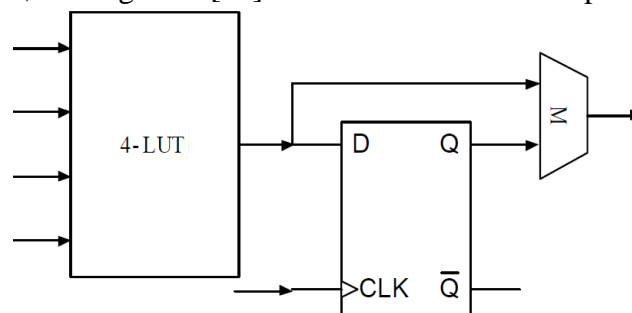


Figure 5: Typical internal structure of LUT

to the wire segments in the adjacent channels. When spanning the FPGA routing path between CLBs, the switch blocks are used. By turning on the programmable switch in a switch block, longer paths are produced. The longest path does not necessarily result in the largest delay in time. But the longest delay in the FPGA determines the highest possible clock speed that it can run.

Process technology is the technology that provides the programmability of the programmable logic on the FPGA. There are a number of different process technology types for programming FPGAs. One of them is the fusible link technology. The programmable cell contains fusible links which can be blown (fuse) or grown (antifuse) in order to represent a logical function. However, the links cannot be restored back to its original state and, therefore, they are one-time programmable. Instead of fuses, there is another category of programmable technology based on read-only memory (ROM). They are erasable

programmable ROM (EPROM), electrically erasable programmable ROM (EEPROM) and FLASH-erase EPROM. These kinds of technology allow the FPGA to be re-programmed. Most of the modern FPGA uses static random-access memory (SRAM) as the process technology because SRAM can be reprogrammed quickly and repeatedly. However, one disadvantage of using SRAM is that the configuration data (state of the cell) will be lost if power is removed. This means the FPGA will have to be reprogrammed after the system is powered on.

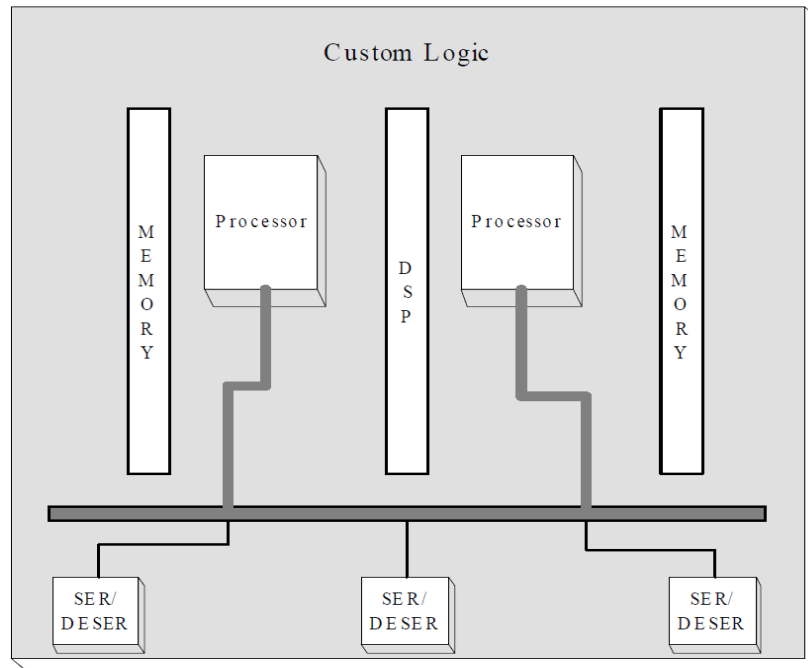


Figure 6: Modern FPGA

Apart from the above common internal FPGA components, modern FPGAs are able to use 40nm manufacturing technology and can be equipped with embedded higher level ASIC function blocks in the fabric to extend capacity, reduce the area required, and increase clock speed. In addition to the FPGA vendor dependent primitives which are fundamental function blocks, examples of these build-in complex function blocks include embedded processors, generic DSP slices, high speed SER/DESER, block memories and various IP cores, to name a few, Ethernet MAC, PCI-E controllers, DDR SDRAM controllers and many others. Figure 6 shows a modern FPGA system. The build-in ASIC function blocks enables it to be taken to market quickly, reduces the power required, and provides the user with the biggest possible space for customer functions. Coupled with these facilities, and decreasing prices, FPGAs are not only used for system validation, and prototypes, but are also increasingly used for end-products.

2.3.2. FPGA Design Methods

Parallelism and pipelining are common methods which are widely used in FPGA design in order to achieve better performance and higher throughput. Hardware parallelism has significant advantages over loosely coupled software parallelism. Software parallelism relies upon a sequential set of statements that can be loosely aligned through operating system parallelism constructs and, unless the number of microprocessor cores is very large, only allows a limited level of parallelism. However, hardware parallelism allows a large number of operations to be carried out in a tightly synchronous fashion with advantages in the number of

operations that can truly be carried out at the same time and without the difficulties in aligning the operations compared to software parallelism.

Pipelining allows operations to be performed on a fast data throughput in a manageable fashion. Data is stored in the pipeline so that it can be processed in parallel. This allows operations that take more than one clock cycle to be performed in a synchronous fashion one clock cycle at a time, while the data moves through the pipeline. This requires an operation to be transformed from a straightforward software algorithm (as it is usually described and tested) into an implementation suitable for performing in a pipelined fashion.

While it is essential that both parallelism and pipelining to be employed effectively in hardware implementations, they also cause a significant design complexity that increases the time-to-market. Furthermore, complex parallelism and pipelining consume more fabric resources and power. One example of the design penalty introduced by intensive parallelism and pipelining is that a far more complex mathematic function is needed to align the parallel pipelines. Another example is the greater area used when implementing a high clock rate and high throughput that necessitates increasing the amount of pipelining used. Thus it is crucial for the hardware accelerated Q-tool to retain parallelism and pipelining but implement the full functionality cost-effectively.

During DICONET project, four methods were studied and analysed for protocol implementation. This section provides an overview of these methods:

2.3.3. Pure FPGA implementation

This method relies only on utilising gate level resources of the targeted FPGA. The main advantage of this method is its potential to achieve the highest possible performance achievable by the FPGA. However this approach has several disadvantages including: Long development time which makes it not feasible in the time scale of DICONET, Inflexibility on upgrading and modification and finally, all Q-tool codes need to be rewritten to hardware description language. Another disadvantage the interfacing of Q-Tool to the rest of impairment aware control plane (in particular Network Planning and Operation Tool (NPOT)).

2.3.4. FPGA + Embedded Processor implementation

This method relies only on utilising gate level resources of the targeted FPGA together with the available embedded processor capability of new FGPAs. The advantages of this method include: its potential to achieve the high performance, flexibility for future upgrades (and interfacing?) and modification and more importantly possibility to implement processing intensive and complex part of Q-Tool in embedded processor of FPGA. However this approach has also few disadvantages including: requirement for a complex proprietary platform to combine embedded processor and FPGA, necessity for converting Q-tool code into an Embedded C code compatible with the embedded processor of the FPGA and finally, requirement for a complex method to partition Q-Tool to two parts for implementation in logical gate and embedded processor of FPGA.

2.3.5. Host Server + pure FPGA implementation

This method uses control plane host server (PC) for main part of Q-tool and uses pure logic of FPGA to implement a co-processor for the Q-tool. The advantages of this method include:

high flexibility for future upgrades and modification. However this approach has some major disadvantages including: poor performance due to partial implementation of Q-tool in the host server, requirement for a complex method to partition Q-Tool into two parts for implementation in logical gate of FPGA and the host server. Finally there is a need for a complex interface between FPGA and host server which potentially can be a major speed bottleneck.

2.3.6. Host Server + FPGA + Embedded Processor implementation

This method uses control plane host server (PC) for main part of Q-tool and uses logic of FPGA as well as its embedded processor to implement a co-processor for the Q-tool. The advantages of this method include: high flexibility for future upgrades and modification. However this approach has some major disadvantages including: poor performance due to partial implementation of Q-tool in the host server, requirement for a complex method to partition Q-Tool to three parts for implementation in logical gate of FPGA, the embedded processor and the host server and finally requirement for a complex interface between FPGA and host server which potentially can be a major speed bottleneck.

Based on the above analysis and discussion it is evident that the option “FPGA + Embedded Processor implementation” is the most suitable option for the hardware accelerator of DICONET.

3. Hardware Accelerated Q-tool Implantation

3.1. Targeted Hardware

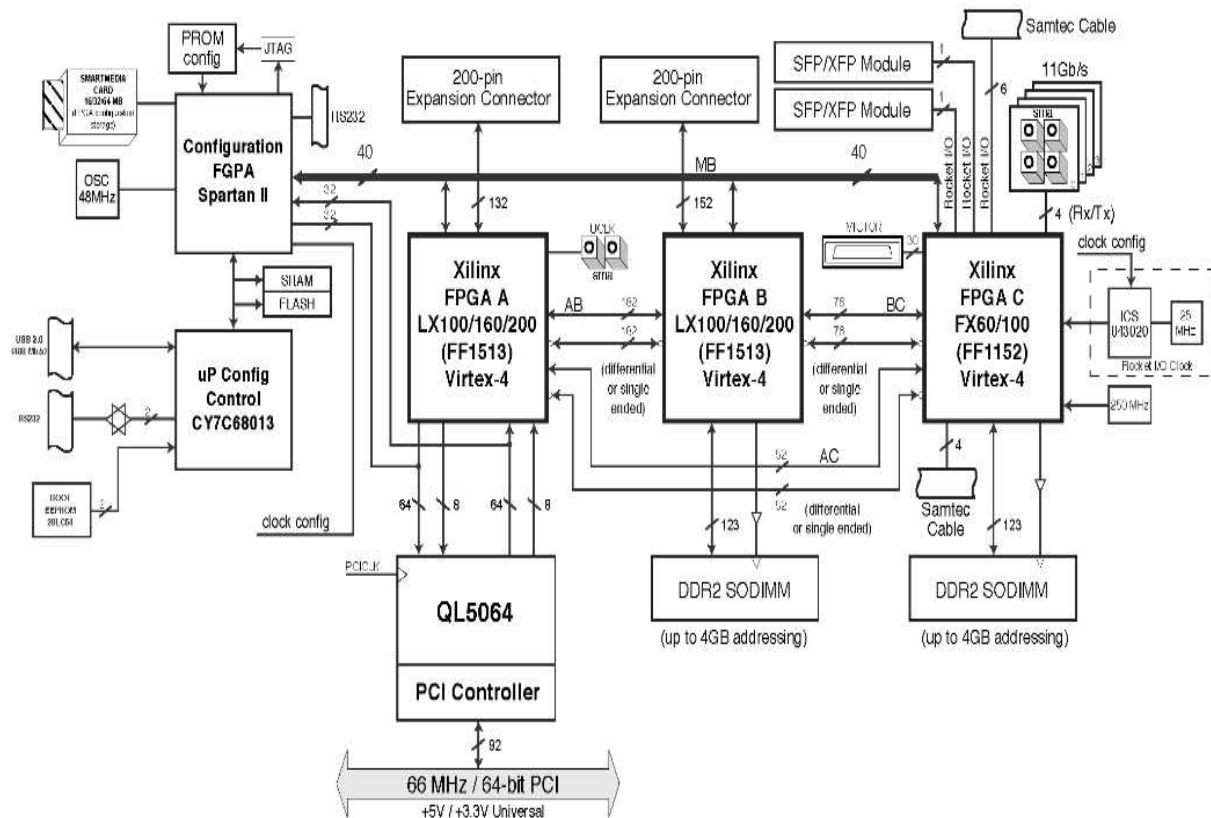


Figure 7: DN8000KPCI Function Blocks

In order to meet the tight acceleration requirements of DICONET, the Xilinx Virtex™-4 FPGA families were chosen¹. They are LX, FX, and SX --- offering multiple feature choices and combinations to address all complex applications. The wide array of Virtex-4 internal resources includes the PowerPC™ processors (with a new APU interface), tri-mode Ethernet MACs, 622 Mb/s to 6.5 Gb/s serial transceivers, dedicated DSP slices, high-speed clock management circuitry, and source-synchronous interface blocks. LX family is for high-performance logic applications solution; SX family is for high-performance solution for digital signal processing (DSP) applications; FX family is for high-performance, full-featured solution for embedded platform applications [30]. In DICONET, which deploys network based complex algorithm application, either LX or FX families are most suitable.

The hardware platform of the DICONET accelerator is the FPGA evaluation board DN8000KPCI from DiniGroup [31] populated with two LX160FF1513-12s and one FX100FF1152-12 which are high density and fastest FPGAs available in Xilinx Virtex™-4 families as shown in Figure 7. This platform has been chosen for implementation of the DICONET protocol hardware accelerator for the following reasons:

- The large FPGA fabric gives sufficient Flip-flop/LUT to implement all the Q-tool modules.

¹ Virtex™-4 FPGA was the latest product when the project started at 2008.

- High speed grade of FPGAs with in platform makes easier to achieve high clock frequency which is particularly important for DICONET hardware accelerator when using relatively high level synthesis language, e.g. Handel-C [32].
- The build-in PowerPC processor eases the network communication between the hardware accelerator and the NPOT.
- The off-shelf hardware Gigabit Ethernet MAC shortens the developing time and provides easy interfacing and good bandwidth.
- Sufficient external interfaces, e.g. SFPs and SMAs enable the high speed transmission up to 6.5 Gb/s by taking advantage of RocketIO.

Due to the challenge posed by the complexity of the QoT estimation tool (Q-Tool), our approach focuses on the implementation of only the time-critical operations of the Q factor computation in the logic part of the targeted FPGA, and keep the non-time critical operations in software to be implemented in the Embedded processor (IBM PowerPC 405 hard core 300MHz which is embedded inside the FPGA fabric). Another reason to partition the QoT estimation tool is that the FPGA achieves high performance results when it executes computing intensive operations, but is overshadowed by control intensive operations while the converse is true for the general purpose embedded processor. The data exchange between the reconfigurable logic in the FPGA fabric and the pipeline of integrated embedded processor is critical for the best acceleration performance. This is because the large amount of mathematical operations involves intense data exchange between the FPGA logics and the embedded processor. For this reason, for designing the QoT estimation tool a special effort is required to reduce the time spent for the communication between the FPGA logic and the embedded processor. This is achievable by exploiting the versatile integrating features of the embedded processors inside the FPGA fabric.

3.2. Inter-connection models between FPGA logic and embedded processor in FPGA fabric

There are different solutions provided for deploying embedded processors inside the FPGA fabrics. In DICONET project, four approaches have been studied and analysed as summarized below:

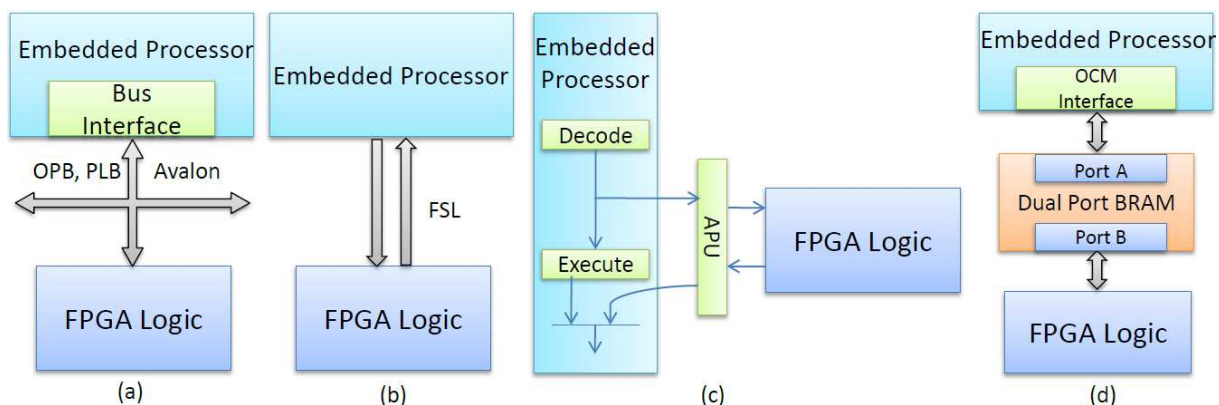


Figure 8: Different solutions for integrating an embedded processor with the FPGA logic

3.2.1. Embedded processor bus connected model

As shown in Figure 8 (a), processor bus connected to the FPGA logics needs the embedded processor to move data and send commands through a bus. Consequently, a single data

transaction can require many processor cycles and need bus arbitration. A direct memory access (DMA) can be deployed to enable the FPGA logic to operate on data located on bus connected memory without the interaction with the embedded processor at the cost of additional FPGA logics.

3.2.2. I/O based model

In this approach, Fast Simplex Link (FSL) as shown in Figure 8 (b) which is a more efficient interface than embedded processor bus through a dedicated point to point communication channel to exchange data between the FPGA logic and embedded processor without any arbitration overhead. The reduced control complexity enables lower latency and higher rate data movement but one channel only supports one direction data movement.

3.2.3. Extended instruction set model

In this approach, Auxiliary Processor Unit (APU) as shown in Figure 8 (c), it attaches directly to the instruction pipeline of the embedded processor and extends the instruction set. As the most highly integrated interface, this approach also clocks faster than an embedded processor bus. The brief data flow of APU is that the instructions from cache/memory are simultaneously presented to the embedded processor decoder and the APU controller. If the embedded processor recognizes the instruction, it is executed, otherwise, the APU has the opportunity to acknowledge the instruction then send to the FPGA logic to execute it.

3.2.4. Shared memory model

On Chip Memory (OCM) approach (or model?) as shown in Figure 8 (d). An on chip dual port block RAM (DPBRAM) is used between the FPGA logic and embedded processor which both can directly and separately access the same data memory, e.g. one port can be employed for embedded processor data side interface connection, while the second port for external FPGA logic data access.

3.3. DICONET Hardware accelerated QoT estimation tool architecture

Based on the discussion in section 3.2, a combination of the extended instruction set approach and embedded processor bus connected approach is deployed. At the cost of additional logic, the FPGA logic, i.e. QoT estimation tool, receives commands and returns status through a fast, low latency interface (i.e. APU) while operating on blocks of data located in bus connected memory.

The hardware accelerated QoT estimation tool in DICONET work flow is as follows (Figure 9):

1. The NPOT sends the QoT estimation request to the FPGA via the network interface, the embedded processor i.e. PowerPC.
2. While the embedded processor performs Q calculation, some computational intensive instructions are bypassed and are forwarded to the APU for handling.
3. Then the FPGA logics connected to the APU performs the instructions, e.g. partial Q-tool computation, and returns to the APU, in turn, back to the embedded processor to resume the Q calculation.
4. During the calculation, all the intermediate data is stored in the processor bus connected memory, i.e. the external DDR (Double Data Rate) memory.

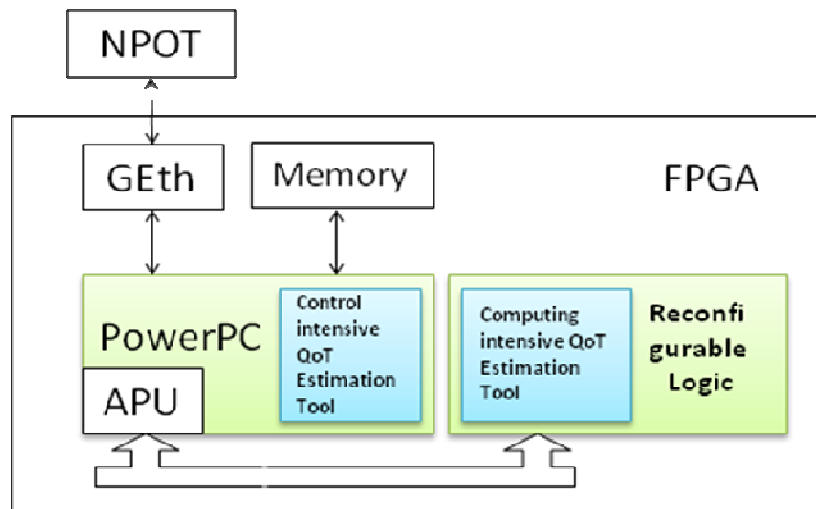


Figure 9: Hardware Accelerated QoT estimation tool

One of the challenges to accelerate the QoT estimation tool is to accurately identify computationally-intensive operations and offload them to the FPGA via the APU interface, in order to significantly improve the hardware-based QoT estimation tool performance. In this work, the GNU software “gprof” [33] is used to identify the most time-consuming routines. Then, this information is used as a reference to carry out a careful design. In order to achieve the highest performance, a two-level acceleration scheme is proposed — at instruction level, and at routine level. The former is mainly used for mathematic computation, i.e. arrays and complex number operations. The latter is used when many individual instruction level accelerations need to be done; then the APU interface overhead becomes significant. In this case, the whole routine can be migrated into the FPGA reconfigurable logic if it is allowed by the FPGA resource, and the APU will only be invoked once when returning the results.

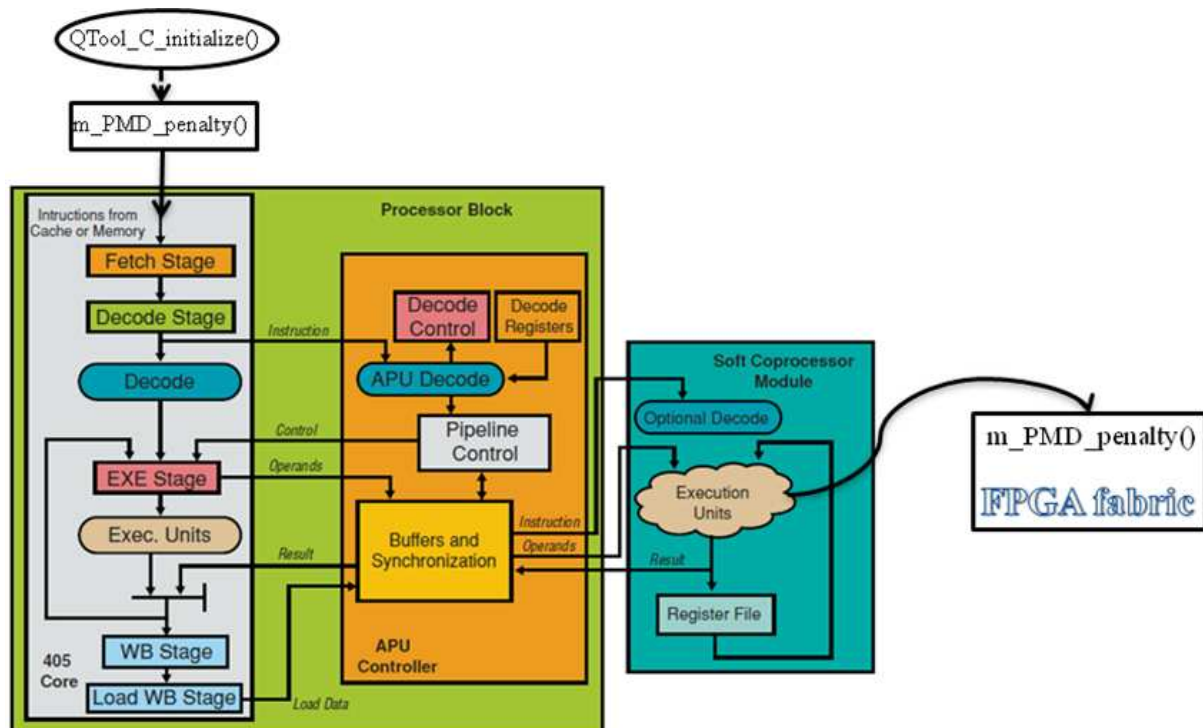


Figure 10: APU controller Processing Operative block Diagram

The detailed APU working flow with interaction with Q-tool is shown in Figure 10. It gives an example, i.e. function *PMD_penalty()*, on how the Q-tool invokes FPGA logics to calculate *PMD_penalty()* via APU. When PowerPC executes *PMD_penalty()*, it fetches and decodes the instruction from memory. When the PowerPC recognizes the *PMD_penalty()* instruction is not registered with PowerPC native instruction, it will forward this instruction to APU decoder. In turn the APU fetches the data and operands, then send to FPGA logic to calculate the *PMD_penalty()*. Once the calculation finishes, the APU will forward the returned results to PowerPC.

4. Preliminary Test Results

Table 2: Comparison between Non-Accelerated and Accelerated Q-Tool

Non-accelerated Q-tool				Accelerated Q-tool			
“CPU”		Memory		“CPU”		Memory	
3.2GHz Intel Quad Core Extreme		4GB DDR3		100 MHz Xilinx Virtex 4 + 300 MHz IBM PowerPC 405		1 GB DD2 at	

The non-accelerated QoT estimation tool is evaluated in a very fast general purpose computer which is populated with an Intel Quad Core Extreme 3.2GHz CPU and 4GB DDR3 memory, while the accelerated version runs in the FPGA evaluation board mentioned in previous section. The evaluation testbed specification is summarized in Table 2. The experiments carried out in this work are based on a national network topology provided by Deutsche Telekom (DT). Two variations (“network sizes”) of the DT topology have been used: one with original link lengths, and the other where all link lengths are doubled. In addition, the QoT estimation tool performance is evaluated in terms of the number of available wavelengths per link, varying from 8 to 32 with a step of 8, and for a varying number of lightpaths established in the network. The QoT estimator includes single channel effects (ASE noise, PMD, SPM and chromatic dispersion) and multichannel effects (XPM and FWM), so that all of the network size, number of channels and number of established lightpaths impact its running time.

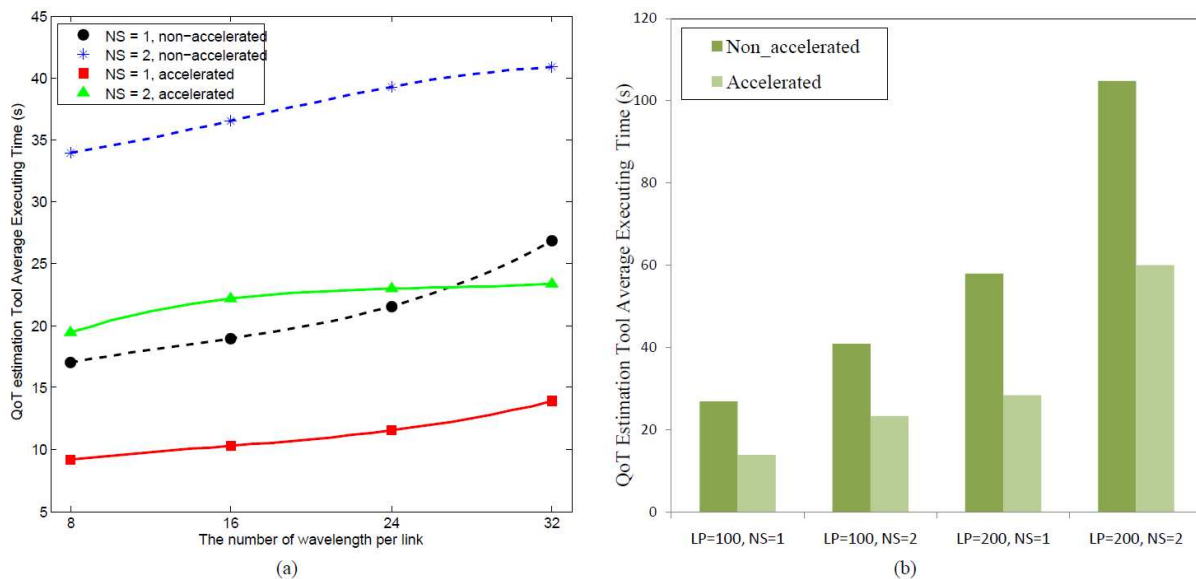


Figure 11: The QoT estimation tool average executing time in two different network sizes v.s. (a) the number of wavelength per link for 100 lightpaths and (b) the number of lightpaths for 32 wavelengths per link

Given the above scenarios, Figure 11(a) shows the QoT estimation tool performance for 100 existing lightpaths; we report the average executing time against the number of wavelength per link for the two different network sizes ($NS=\{1,2\}$). For the original network size ($NS=1$), the non-accelerated QoT estimation tool execution time (dotted line with circle) is ranging from 17s to 27s; the corresponding accelerated computation (solid line with square) is nearly halved, a very significant speed-up. When network size is doubled, generally the computation

time is doubled as well. The non-accelerated QoT estimation tool takes 34s to 41s to run depending on the number of available wavelengths per link; while the accelerated QoT estimation tool execution time drops down to about half of this. In the worst case (23s) the accelerated tool is still much faster than best-case non-accelerated tool.

The QoT estimation tool performance for combinations of two numbers of lightpath ($LP = \{100, 200\}$) and two network sizes ($NS = \{1, 2\}$) is shown in Figure 11b for a fixed number of wavelengths per link (32). As shown in the figure, the accelerated QoT estimation tool outperforms the non-accelerated one by almost 100% for each combination of LP and N; it scales smoothly and mitigates the impact of increasing load. It is a very important desirable feature for a QoT estimation tool in a real dynamic network to provide a consistent network provisioning delay.

The initial result is very promising and has resulted a paper publication [34]. Further, work is ongoing to improve performance of the hardware accelerated Q-tool and it is expected that initial result to be significantly improved which will be a major achievement for DICONET. The detailed further work will include investigating and designing more efficient data exchange mechanism between PowerPC and FPGA, making it more suitable for Q-tool operation, taking more advantages of high performance FPGA logic, optimizing Q-tool hardware language code, collecting more numerical results. The developed Q-tool hardware accelerator will be tested in UESSEX testbed then ported to UPC testbed for the final demo in September 2010.

References

- [1] I.-T. Recommendation, “G.8080: Architecture for the automatically switched optical network (ASON),” 2005, with amendment.
- [2] E. Mannie, “IETF RFC 3945: Generalized Multi-Protocol Label Switching (GMPLS) Architecture,” 2004.
- [3] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, “RSVP-TE: extensions to RSVP for LSP tunnels,” 2001.
- [4] K. Kompella and Y. Rekhter, “OSPF Extensions in Support of General Multi-Protocol Label Switching (GMPLS).” RFC 4203, October 2005.
- [5] A. Farrel, J. Vasseur, and G. Ash, “IETF RFC 4655: A Path Computation Element (PCE)-Based Architecture,” 2006.
- [6] R. Martinez, C. Pinart, F. Cugini, N. Andriolli, L. Vakarengi, P. Castoldi, L. Wosinska, J. Comellas, and G. Junyent, “Challenges and requirements for introducing impairment-awareness into the management and control planes of ason/gmpls wdm networks,” IEEE Communications Magazine, vol. 44, pp. 76–85, 2006.
- [7] G. Pavani, L. Zuliani, H. Waldman, and M. Magalhães, “Distributed Approaches for Impairment-aware Routing and Wavelength Assignment Algorithms in GMPLS Networks,” Computer Networks, 2008.
- [8] Y. Lee, G. Bernstein, D. Li, and G. Martinelli, “A Framework for the Control of Wavelength Switched Optical Networks (WSO) with Impairments.” Internet Draft, October 2009.
- [9] S. Azodolmolky et al. “A dynamic impairment aware networking solution for transparent mesh optical networks,” IEEE Com. Mag, vol. 47, pp. 38 – 47, 2009.
- [10] Wang, H.; Veeraraghavan, M.; Karri, R.; Li, T.; , "Design of a High-Performance RSVP-TE Hardware Signaling Accelerator," Selected Areas in Communications, IEEE Journal on , vol.23, no.8, pp. 1588- 1595, Aug. 2005.
- [11] K. Christodoulopoulos et al. “A comparison of offline IA-RWA approaches (invited),” in Proc. of NOC, 2009.
- [12] Cavium Networks, <http://www.caviumnetworks.com/>
- [13] EZchip, <http://www.ezchip.com/>
- [14] Intel, <http://developer.intel.com/design/network/products/npfamily/ixp425.htm>
- [15] BAYmicrosystems, <http://www.baymicrosystems.com/products/network-silicon/chesapeake.php>
- [16] LSI, http://www.lsi.com/networking_home/networking_products/network_processors/
- [17] Xelerated, <http://www.xelerated.com/en/core-x10/>
- [18] Freescale, <http://www.freescale.com/>
- [19] Appliedmicro, <http://www.appliedmicro.com/>
- [20] Accardi, K., Bock, T., Hady, F., and Krueger, J. 2005. Network processor acceleration for a Linux* netfilter firewall. In Proceedings of the 2005 ACM Symposium on Architecture for Networking and Communications Systems.
- [21] Xilinx, <http://www.xilinx.com/>
- [22] Altera, <http://www.altera.com/>
- [23] Latticesemi, <http://www.latticesemi.com/>
- [24] Actel, <http://www.actel.com/>
- [25] C. M. Maxfield, ed., The Design Warrior’s Guide to FPGAs. Newnes, 2004.
- [26] S. Brown and J. Rose, “FPGA and CPLD architectures: a tutorial,” Design & Test of Computers, IEEE, vol. 13, pp. 42–57, summer 1996.
- [27] Wikipedia, “Xilinx.” <http://en.wikipedia.org/wiki/Xilinx>.

- [28] I. Kuon, R. Tessier, and J. Rose, "FPGA Architecture: Survey and Challenges," Foundations and Trends® in Electronic Design Automation, vol. 2, no. 2, pp. 135–253, 2007.
- [29] X. W. Paper), "Virtex-6 FPGA Configurable Logic Block: User Guide," June 2009, UG364 (v1.0).
- [30] Xilinx Datasheet, Virtex-4 Family Overview,
http://www.xilinx.com/support/documentation/data_sheets/ds112.pdf
- [31] Dinigroup, <http://www.dinigroup.com>
- [32] Handel-C, <http://en.wikipedia.org/wiki/Handel-C>
- [33] GNU gprof, http://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html.
- [34] Y. Qin, Y. Pointurier, E. Escalona, S. Azodolmolky, M. Angelou, I. Tomkos, K. Ramantas, K. Vlachos, R. Nejabati, and D. Simeonidou, "Hardware accelerated impairment aware control plane," Proceedings of the IEEE/OSA Optical Fiber Communication Conference (OFC), San Diego, CA, USA, 21-25 March 2010.