CHAPTER 4

REAL TIME PERFORMANCE ANALYSIS OF THE DDA-EPON ALGORITHM

It is crucial to implement the algorithm in processor and affirm its compatibility in hardware architecture in the process of realizing the algorithm in commercial switches. This way, the number of clock cycles consumed for the execution of the algorithm could be calculated which is significant, because the number of clock cycles determines the speed of the algorithm. In turn, the speed of the algorithm is the deciding factor to implement the algorithm in a particular network. WiMAX aims to guarantee real time service and this necessitates that the algorithm be tested in a processor beforehand of implementing it for commercial practice. To realize the same, the work is extended to implement and test the performance of the DDA-EPON algorithm in a WiMAX switch using Intel’s 1XP 2400 Network Processor. This chapter provides a short introduction about network processors and explains the various sections of the DDA-EPON algorithm along with their implementation in the processor.

4.1 NETWORK PROCESSOR

Bandwidth explosion over the past decade has resulted in the need of high performance and flexible network devices. The plunging cost of bandwidth provides a way to take advantage of the available Internet connectivity which results in more bandwidth-hungry and computationally intensive applications, like streaming audio and video. In order to support these, the application developers and equipment manufacturers must develop and enhance network service applications that apart from supporting the
simple packet routing process should now include complex services like firewall, Virtual Private Networks (VPN), differentiated services, security, QoS, protocol conversion and others. The existing market necessitates the design and development of network equipments that couples very high throughput along with the flexibility to support new protocols and applications (Coss & Sharp 2004) (Walrand & Varaiya 1999). The choices available for implementing such network equipments are:

- Field Programmable Gate Array – FPGA
- General Purpose Processor – GPP
- Application Specific Integrated Circuit - ASIC
- Application Specific Instruction Processor - ASIP or Network Processor - NP
- Co-processor

Examining the above, Network Processor is the right choice for implementing the network systems since it provides the balance of hardware and software to meet the networking requirements with:

- High performance (has dedicated hardware to perform key computational kernels),
- Flexibility (to adapt to the frequent change in standards and applications, software is made as the major part of the system) and
- Fast time to market (design and realization of any functionality is much faster and cheaper in software than hardware).

Basically, a network processor is an integrated circuit specifically optimized for the networking application domain and is typically a software programmable device with generic characteristics similar to general purpose central processing units (Comer 2005) (Wolf & Franklin 2006) (Li et al 2009) (Xiaobo et al 2009) (Giladi 2008). These features of network processors have
made them highly desirable for network processing implementations and hence the buzz of activity on network processors. The tendency to use network processors is also justified by the facts that:

- Traditionally routers are realized either by software (low-end) or custom made hardware (high-end). Whereas with network processors, high-speed routers are developed using a combination of software and standard, off-the-shelf hardware.
- Network processors are programmable, and hence the developed routers could be made flexible by easily modifying or extending the instructions. A network vendor or third party or users of the equipment – anyone could develop code for the same.

Some of the key players in the network processors market (Allen et al 2003) (AMCC 2003) (EZChip 2002) are:

- Agere,
- AMD,
- Cavium,
- EZChip – NP-x family
- Hifn,
- Intel - IXP2xxx family
- Mindspeed,
- Motorola,
- Netronome - NFP-32xx family
- PMC Sierra – Winpath family
- Xelerated,
- Ubecom.

Intel IXP2400 network processor’s flexibility and performance for access and edge applications make it worthy for services like realization of QoS,
enforcement of SLAs, and traffic engineering at OC-48/2.5Gbps and 4Gbps data rates. Along with managing bandwidth, IXP2400’s performance enables the OEMs and Service Providers to offer differentiated and tiered services to their customers. IXP2400’s capabilities make it suitable for high performance applications like multi-service switches, broadband access equipment, and wireless infra-structure. Also the programmability of the IXP2400 makes it well suited for VoIP Gateways, multi-service access platforms, edge routers, remote access concentrators and VPN Gateways (Intel 2002). The flexible architecture and interfaces, low power consumption, and small footprint of the IXP2400 make it ideal for the target markets. Network processing typically involves extensive queue management. Depending on the applications and algorithms used, the network processor may manage many thousands of packet queues. Moreover, the network processor must execute the desired scheduling algorithm, and select the appropriate packets out of these queues for transmission at wire speed. As a result, effective queue management is the key to high-performance network processing and to reducing development complexity. IXP2400 features high-performance queue management hardware that automates adding data to and removing data from queues. Queues can be accessed simultaneously by multiple threads, and there is no limit on the size or total number of queues which can be managed (Liu et al 2008) (Govind et al 2007).

Operation of IP-based networking equipment can be partitioned into three logical planes:

- Data plane,
- Control plane and
- Management plane.

Data plane: The data plane receives and sends packets from an interface, processes them as defined by the protocol, and delivers, drops, or forwards
them as appropriate. The hardware at this layer is very fast and designed with fast packet processing and forwarding in mind. It is typically the most performance sensitive since all packets processed by the device pass through this plane. It consists of:

- The fast path: Packets that need minimal or normal processing take the fast path. Microengines are used for fast path processing and process the majority of incoming packets. It runs outside the OS environment and hence does not incur any of the OS overheads that degrade overall performance. Only those packets that require complex processing are forwarded to the higher layer for management, signaling and control functions (Xu et al 2008).

- The slow path: Packets that need complex processing take the slow path. XScale core is used for slow path processing and handles a few exceptional packets that cannot be handled on the fast path because of the complexity of the processing involved. Examples include forwarding IP packets with options in the header, fragmented packets, etc.

**Control Plane:** The control plane is responsible for the setup, configuration, and update of tables and data sets used by the data plane and it handles protocol messages. Control plane handles complex signaling protocols which when implemented in the data plane would lead to poor forwarding performance. Data plane detects incoming signaling packets and locally forwards them to the control plane. The control plane then updates the data plane information and injects outgoing signaling packets into the data plane. For example, processing of Routing Information Protocol (RIP) packets and IPv4 forwarding table updates used by the data plane are done by the control plane. In the IXP environment, the XScale core may function as the control plane, or this functionality could be supported by an external processor.
Management plane: The management plane is responsible for system configuration, statistics collection, aggregation and reporting, implementation of management protocols, command line interface, graphical user configuration interfaces, and stopping or starting applications in response to user input or messages from other applications.

The Intel Internet Exchange Architecture (IXA) portability framework is used primarily to develop data plane software and to help interface with code running on the control plane. Intel IXA software framework as depicted in Figure 4.1 provides the associated software infrastructure to help develop modular, reusable software building blocks for network processors (Carlson 2003) (Intel 2003).

![Figure 4.1 IXA Software Framework](image)

The Microengines (MEs) do most of the programmable per packet processing in the IXP2400. There are 8 microengines, connected in two clusters of 4 microengines as shown in Figure 4.2. Each microengine is its
own processor running in perfect clock synchronization with all the other microengines. The speed and flexibility of the microengines is provided by their fast access to most special purpose subsystems of the IXP2400 chip. The eight microengines are highly programmable packet processors, and support multi-threading up to eight threads each. These microengines provide a variety of network processing functions in hardware, and process data at OC-48 wire speed. The Microengines have access to all shared resources (SRAM, DRAM, MSF, etc.) as well as private connections between adjacent Microengines. Figure 4.2 depicts the functional block diagram of IXP2400 processor.

![Functional block diagram of IXP2400 processor](image)

**Figure 4.2 Functional block diagram of IXP2400 processor**

The Microengine provides support for software controlled multi-threaded operation. Given the disparity in processor cycle times vs. external memory times, a single thread of execution will often block waiting for the
memory operation to complete. Having multiple threads available allows for threads to interleaver operation - there is often at least one thread ready to run while others are blocked. This improves the usage of the microengine resources.

The fast path processing on the microengines is composed of logical networking functions called microblock. A microblock is a software component in the packet processing pipeline performing a specific operation on packets and each microblock is independent of others. Each microblock is a macro using underlying low-level libraries provided by data plane libraries. A microblock has an associated management component on the XScale core. The application is typically written so that the microblock will process most common cases and pass exception cases to the XScale component for further processing. The core component implements the configuration, management and exception processing code for an associated microblock. The resource manager library is a software component on the XScale core, which provides an API for hardware initialization and configuration, and communication between the microblocks and their associated core components. If the packet processing is fairly expensive, the code per microblock should utilize more than one microengine to give a significant performance advantage over using only one microengine. This performance advantage only comes to its right when traffic loads are high, at low traffic rates there is no benefit of using more than one microengine per microblock.

Several microblocks running on a microengine thread can be combined into a microblock group. A microblock group has a dispatch loop that defines the dataflow for packets between microblocks and typically looks like:

```c
// start of dispatch loop:
(1) Receive packet
(2) Process packet
```
(3) Transmit packet

// end of dispatch loop

A microblock group runs on each thread of one or more microengines. The same dispatch loop may be instantiated on more than one microengine thread to process multiple packets at the same time. A packet processing pipeline consists of many microblocks chained together.

The following sections provide a detailed explanation about the implementation of the algorithm in network processors and the results of the process along with a summary.

4.2 IMPLEMENTATION

First the Conventional or Centralized ONU processing system is implemented in the network processor and the machine cycles that is required to execute the algorithm is estimated. The DDA-EPON ONU system is then implemented using the network processor and the performance is evaluated.

Three microengines are required to construct an ONU hardware. The first microengine receives the packets, the DDA-EPON and conventional algorithms are implemented in the second microengine and the third microengine is used to transmit the packets. Once the construction of the ONU is elaborated, the systematic description of the process of construction of the OLT will be discussed.

4.2.1 Microengine 0:0 – Reception

Microengine 0:0 is used to receive the packets from the ONU. The first microengine receives all the incoming packets which are then stored in memory for further processing. Reassembling of packet frames is done in the POS (Packet over SONET)-PHY Level3 physical media interface and the
receive / transmit device are configured to work in POS-PHY Level3 mode. In this mode the MAC device supports 32 ports and these ports together share the 32-bit bus. The receive microengine reassembles the incoming packets that are in the receive buffer (RBUF), writes the packet data to a DRAM buffer and queues them in a scratch ring (Ring 0) for further processing. The scratch ring holds the DRAM address of every new packet's starting address. The algorithm for the packet reception functionality is given below.

Void function PacketReceive
{
    Receive:
    Wait for (Evaluate signal);
    Extract Buffer element, Byte count, Start of Packet (SOP), Middle of Packet (MOP), End of Packet (EOP) details;
    Transfer the packet element from RBUF to DRAM;
    If (SOP == 1)
    {
        Push the address pointed by DRAM pointer in to Scratch Ring;
    }
    Update Packet size;
    Add thread to Thread FreeList;
    Increment the DRAM pointer;
    If (EOP != 1)
        goto Receive;
}
Table 4.1 lists the register used for the process and their offset to RAM.
Table 4.1 Network Processor Register Initialization

<table>
<thead>
<tr>
<th>Register</th>
<th>Offset</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSF_Rx_Control</td>
<td>0x0000</td>
<td>Control register that defines the number of receive configuration parameters.</td>
</tr>
<tr>
<td>Rx_Thread_Freelist</td>
<td>0x0030</td>
<td>Adds a Context to the Rx_Thread_Freelist.</td>
</tr>
<tr>
<td>RBUF_Element.Done</td>
<td>0x0044</td>
<td>RBUF element can be freed and reused by writing to this address.</td>
</tr>
<tr>
<td>Rx_Up_Control_0</td>
<td>0x0080</td>
<td>Register used to configure various receiving modes (SPHY, MPH, etc.).</td>
</tr>
</tbody>
</table>

4.2.2 Microengine 0:1 – ONU Processing

ME 0:1 runs the ONU processing functionality and its performance is checked in both the conventional and the DDA-EPON scheme.

4.2.2.1 Conventional ONU Processing

The conventional processing algorithm of ONU in the microengine 0:1 is first implemented. The microengine 0:1 reflects the functionality of the conventional ONU. A conventional ONU estimates the bandwidth requirement for the next cycle and generates the REPORT message filling the Report bitmap and Queue#n Report fields of the REPORT message. Then, the ONU sends the request message to the OLT.
Void function ONUProcessing
{
    Estimate the bandwidth requirement for the next cycle;
    Fill Report bitmap and Queue#n Report fields;
    Generate REPORT message;
    Send it to OLT;
}

The sequence of actions involved in the Conventional ONU processing algorithm is detailed above. Once the performance of the conventional algorithm is evaluated, the DDA-EPON algorithm is then implemented in the microengine 0:1 and processed.

4.2.2.2 DDA-EPON ONU Processing

The microengine 0:1 reflects the functionality of a DDA-EPON ONU. ONU estimates the bandwidth requirement for the next cycle. Then the ONU checks if the required bandwidth is less than (lightly-loaded) or greater than (heavily-loaded) its SLA value. If lightly-loaded, the ONU generates REPORT message with the Report bitmap and Queue#n Report fields filled with the required bandwidth details and sends the request message to OLT. If heavily-loaded, the ONU generates REPORT message with the Report bitmap, Queue#n Report, Status bitmap, and Status#n Report fields filled with the required bandwidth and SLA details and sends the request message to OLT. The algorithm used for DDA-EPON ONU processing is detailed below.

Void function DDA-EPONONUProcessing
{
    Estimate the bandwidth requirement for the next cycle;
Check the required bandwidth to determine whether the ONU is lightly-loaded or heavily-loaded;

Fill Report bitmap and Queue#n Report fields for lightly-loaded ONU;

Fill Report bitmap, Queue#n Report, Status bitmap, and Status#n Report fields for heavily-loaded ONU;

Generate REPORT message;

Send it to OLT;

}

After implementing the algorithm, the performance of the same in the microengine is evaluated.

4.2.3 Microengine 0:2 – Transmission

Microengine 0:2 is used to transmit the packets from the ONU. This microengine is designed to segment the packets and transmit over POS-PHY Level3 physical interface and the transmit device is configured to POS-PHY Level3 mode during initialization. The MAC device supports 32 ports and these ports together share the 32 bit bus in POS-PHY Level3 mode. TBUF buffers the packets which are then forwarded. The steps involved in the packet transmission algorithm are given below.

Void function PacketTransmit
{
    Read the Transmit Request from Scratch Ring;
    Read the Packet data from SRAM;
    Store the Packet Information to the Local Memory;
    Allocate TBUF;
    Read the Packet header from SRAM;
    Move the Payload data from DRAM to TBUF;
    Move the header to SRAM;
Validate Transmit Control Word;
Free the Buffer;

} 

Now the implementation of the conventional and DDA-EPONOLT models using network processor is discussed. The implementation requires four microengines. Microengine 0:0 receives the packets from the ONU and stores the packet in the DRAM. Microengine 0:1 retrieves the packet from the DRAM and classifies the packet. Microengine 0:2 run the conventional and DDA-EPONOLT algorithms and forward the packets to the Microengine 0:3 for transmission. The receiving process carried out by Microengine 0:0 is similar to the ONU model as explained in the previous section and the transmission process carried out Microengine 0:3 is similar to the process handled by Microengine 0:2 in the ONU model.

4.2.4 Microengine 0:1 – Classification

The scratch ring extracts the chunks of packets from the DRAM, opens the header and classifies the packets based on ToS. The second Microengine 0:1 reflect the functionality of the QoS guarantor that handles the QoS mapping between EPON 802.1p priority queues and the WiMAX connections, done in an attempt to improve the DDA-EPON algorithm. Eight priority levels are supported by EPON and EPON’s bandwidth request packet includes QoS for the service represented by the numbered priority queue in the 802.1P nomenclature. WiMAX supports five QoS levels and an effective mapping mechanism is required between EPON priority queues and WiMAX different levels of QoS. WiMAX flow packets are stored in appropriate EPON priority queues such that the mapping provides equivalent QoS levels at both ends as detailed in Table 3.1. Given below is the packet classification algorithm.

Void function DDA-EPONQoSMapping
{  
    Read the Packet data from SRAM;
    Read the Packet header from SRAM;
    Map the WiMAX packets with the equivalent priority levels of EPON priority queue;
}

4.2.5 Microengine 0:2 – OLT Processing

ME 0:2 runs the OLT processing functionality and the performance of the OLT processing is checked in both the conventional and the DDA-EPON scheme.

4.2.5.1 Conventional OLT Processing

The third Microengine 0:2 reflect the functionality of the conventional OLT. The conventional OLT on receiving the REPORT message from an ONU, extracts the Report bitmap and Queue#n Report fields from the REPORT message and calculates the requested bandwidth by the ONU. The process is repeated until REPORT message from all active ONUs are received. Now, the OLT calculates the available bandwidth and the amount of bandwidth that could be allotted to each ONU and sends a GATE message to each ONU with the bandwidth allotted details specified in the Grant fields. The ‘Force Report’ flag is set in these grant messages since a REPORT message is expected in response from the ONU. The conventional ONU processing algorithm is detailed below.

Void function OLTProcessing
{
    Collect REPORT message from all active ONUs;
    Extract the Report bitmap and Queue#n Report fields;
Frame and send GATE message to each ONU with Grant fields having the allotted bandwidth details;

‘Force Report’ flag is set in these grant messages since a REPORT message is expected in response from the ONU;

Once the performance of the conventional algorithm is evaluated, the DDA-EPON algorithm is then implemented in the Microengine 0:2 and processed.

4.2.5.2 DDA-EPON OLT Processing

The third Microengine 0:2 reflect the functionality of the DDA-EPON OLT. The OLT on receiving the REPORT message from the ONU extracts the Status bitmap field from the REPORT message and determines if the ONU is lightly or heavily loaded. For lightly-loaded ONUs, the OLT grants the requested bandwidth and sets the ‘Force Report’ flag in the GATE message. Whereas, for heavily-loaded ONUs the OLT grants only the bandwidth as specified during the SLA and the ‘Force Report’ flag is reset to inform the ONU that a REPORT message need not be generated for this GATE message. The above is repeated for all active ONUs. Now, OLT does the second round of allocation for all heavily-loaded ONUs with the remaining bandwidth in the cycle in a round-robin fashion and the ‘Force Report’ flag is set in these grant messages since a REPORT message is expected in response from the ONU. The steps involved in the DDA-EPON OLT processing algorithm are detailed below.

Void function DDA-EPONOLTProcessing
{
    Receive REPORT message from an ONU;
Extract the Report bitmap, Queue#n Report, Status bitmap, and Status#n Report fields;

Frame and send GATE message to each lightly-loaded ONU with the requested bandwidth and ‘Force Report’ flag is set in these since a REPORT message is expected in response from the ONU;

Frame and send GATE message to each heavily-loaded ONU with the SLA bandwidth and ‘Force Report’ flag is reset in these since a REPORT message is not expected in response from the ONU;

Repeat the above for all active ONUs;

Frame and send GATE message to each heavily-loaded ONU with the remaining available bandwidth in a round-robin fashion and ‘Force Report’ flag is set in these since a REPORT message is expected in response from the ONU;

}  

After implementing the algorithm, the performance of the same in the microengine is evaluated.

4.3 RESULTS AND DISCUSSION

The conventional and DDA-EPON architecture are implemented in IXP2400 network processor which is configured to work as a WiMAX switch and studied for their performance.

The performance of the ONU configuration is first studied. Figure 4.3 illustrates the performance study of the microengines in the ONU configuration. Microengine 0:0 performs the task of reception in the implementation of both ONU and OLT configuration. It could be inferred from Figure 4.3 that the Microengine 0:0 is free initially, but the execution of
the same increases as the packets are received and transferred to the scratch ring for further classification.

![Machine clock cycles for Conventional (left) and DDA-EPON (right) ONU Processing Algorithm](image)

Figure 4.3 Machine clock cycles for Conventional (left) and DDA-EPON (right) ONU Processing Algorithm

After this stage, the microengine becomes completely free as when there are no packets to receive. Microengine 0:1 implements the conventional and DDA-EPONONU functionality. In the conventional mode, the clock cycle consumption is initially low but gradually increases as packets are received and the REPORT messages are framed for each request with the bandwidth request details in it and after a while the consumption reduces when there are no more packets to be processed. In the DDA-EPON mode, the utilization of the machine clock cycle is high than the conventional mode since the ONU is also involved in the bandwidth allocation process by filling in the REPORT message with more details like the SLA bandwidth which the traditional ONU does not do. The observation of the clock cycles in Figure 4.3 indicates that the processor requires around 14000 cycles to execute the conventional ONU algorithm and 18000 cycles to execute the DDA-EPONONU algorithm which is acceptable as per industrial standards.
Microengine 0:2 transmit the packets from the ONU to the OLT. The percentage of utilization of the machine clock cycle for Microengine 0:2 is initially very low and underutilized because only after the packets are received and processed by Microengine 0:0 and Microengine 0:1 are they scheduled for transmission. The consumption gains momentum after the initial delay and this is the reason why there is a sharp increase in the number of clock cycles after Microengine 0:1 has completed the processing of the packets and the packets are then transmitted.

The performance of the OLT configuration is then studied. Figure 4.4 illustrates the performance study of the microengines in the OLT configuration.

![Graph](image)

**Figure 4.4 Machine clock cycles for Conventional (left) and DDA-EPON (right) OLT Processing Algorithm**

Since the functionality of Microengine 0:0 is similar to that in the ONU model and the functionality carried out by Microengine 0:3 is similar to the process handled by Microengine 0:2 in the ONU model, the focus is on the functionalities of Microengine 0:1 and Microengine 0:2. Microengine 0:1 implements the packet classification functionality and from the consumed
machine clock cycles it is clear that the Microengine 0:1 is free initially but increases drastically when the packets are transferred from the Microengine 0:0 to the scratch ring and Microengine 0:1 retrieves and classifies the packets based on the Type of Service. Microengine 0:2 implement the conventional and DDA-EPONOLT functionality. In the conventional mode, the clock cycle consumption starts after Microengine 0:1 completes processing the packets and the machine clock cycle gradually increases and becomes high after receiving the packets from all ONUs. This is because the OLT takes the sole responsibility of calculating the available bandwidth and accordingly allocates bandwidth to each ONU after receiving requests from all active ONUs and then processes them. In the DDA-EPON mode, the utilization of the machine clock cycle is lower than the conventional mode since the workload on OLT is reduced by the ONU taking part in the bandwidth allocation process.

Unlike the conventional OLT, the DDA-EPONOLT machine clock cycles consumption starts earlier since the OLT now sends a GATE message to every request immediately and after receiving request from all ONUs, the OLT does the second round of bandwidth allocation. The observation from the clock cycles indicates that the processor requires around 16000 cycles to execute the conventional OLT algorithm and 14000 cycles to execute the DDA-EPONOLT algorithm. Also it could be noted that there is continuous machine clock cycle utilization in the DDA-EPON algorithm compared to the conventional scheme in which there is a peak utilization of the machine clock cycles.

Figure 4.5 and Figure 4.6 illustrates the percentage of utilization of the clock cycles by the conventional and DDA-EPONONU and OLT configurations and could be seen that none of the microengines is overloaded i.e., the percentage of utilization of the clock cycles has always been less than 100 percent for all the process. The maximum percentage of utilization in all the processes is 96 percent by Microengine 0:2 which is executing the
conventional OLT algorithm and is a proof for the efficiency of the DDA-EPON algorithm.

**Figure 4.5** Percentage of Utilization of Microengines by Conventional (left) and DDA-EPON (right) ONU Algorithm

**Figure 4.6** Percentage of Utilization of Microengines by Conventional (left) and DDA-EPON (right) OLT Algorithm
If there has been a case of overloading of any of the microengines, then the algorithm should be fine-tuned to make it compatible for real time implementation. As this is not the case, the algorithm could be readily deployed for commercial purpose in a WiMAX switch. Also the increase in the machine clock cycles utilization by the DDA-EPONONU is not quite high and hence will not affect the performance of the network switches i.e., the processing time for the algorithm does not increase tremendously.

From the percentage of utilization of the clock cycles by the conventional and DDA-EPON configurations, it could be inferred that none of the microengine is overloaded i.e., the percentage of utilization of the clock cycles has always been less than 100 percent for all the process. The maximum percentage of utilization in all the processes is 96 percent by the microengine which is executing the conventional OLT algorithm and is a proof for the efficiency of the DDA-EPON algorithm.