Intel Corporation – Intel® IXP2400 Network Processor - 2nd Generation Intel® NPU

Many trends are driving the need for intelligence and flexibility in network systems. Intel has developed next generation network processors optimized for applications from the customer premises to the core of the network. Intel® IXP2400 network processor delivers a new level of intelligence and performance for access and edge applications, enabling the realization of Quality-of-Service (QoS), enforcement of Service Level Agreements (SLAs), and traffic engineering at OC-48/2.5Gbps and 4Gbps data rates. These capabilities will allow OEMs and Service Providers to offer differentiated and tiered services to their customers while efficiently managing their network resources and bandwidth.

1. Target Applications

Increasingly, packet processing requirements vary significantly by market segment. For example, access networking equipment must support multiple interfaces and protocols. At the same time this equipment needs to meet tight power and real estate requirements dictated by space constraints in wiring closets. Equipment deployed at the edge of the network must support rapid provisioning of services, scalable performance to provide support for emerging services at wire rate and smooth migration to emerging standards. For all applications, minimizing costs and maximizing time-in-market are also critical concerns.

The IXP2400 has the ideal set of features to support these access and edge requirements at line rates up to OC-48/2.5 Gbps and 4x1GbE. The IXP2400 performance and flexibility make it desirable for a wide variety of high performance applications such as multi-service switches, DSLAMs (DSL access multiplexers), CMTS (cable modem termination system) equipment, 2.5G and 3G wireless infrastructure and Layer 4-7 switches including content-based load balancers, and firewalls. The programmability of the IXP2400 also makes it well suited for VoIP Gateways, multi-service access platforms, edge routers, remote access concentrators and VPN Gateways.

Usage models for IXP2400 Network Processor in the target markets listed above are as follows:

- Aggregation, QoS, ATM SAR functions, traffic shaping, policing, forwarding and protocol conversion in DSLAM equipment
- Aggregation, QoS, forwarding and protocol conversion in CMTS equipment
- ATM SAR, encryption, forwarding in base station controller/radio network controllers (BSC/RNC)
- General Packet Radio Services (GPRS) Tunneling Protocol tunneling and Ipv6 in wireless infrastructure
- ATM SAR, Multi-protocol Label Switching (MPLS), QoS, traffic shaping, policing, protocol conversion and aggregation for multi-service switches
- Content-aware load balancing, forwarding and policing for edge server offload

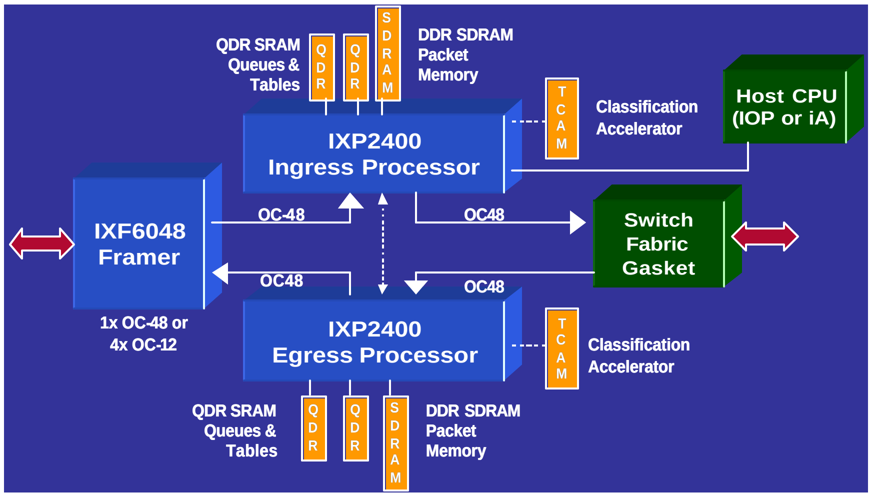
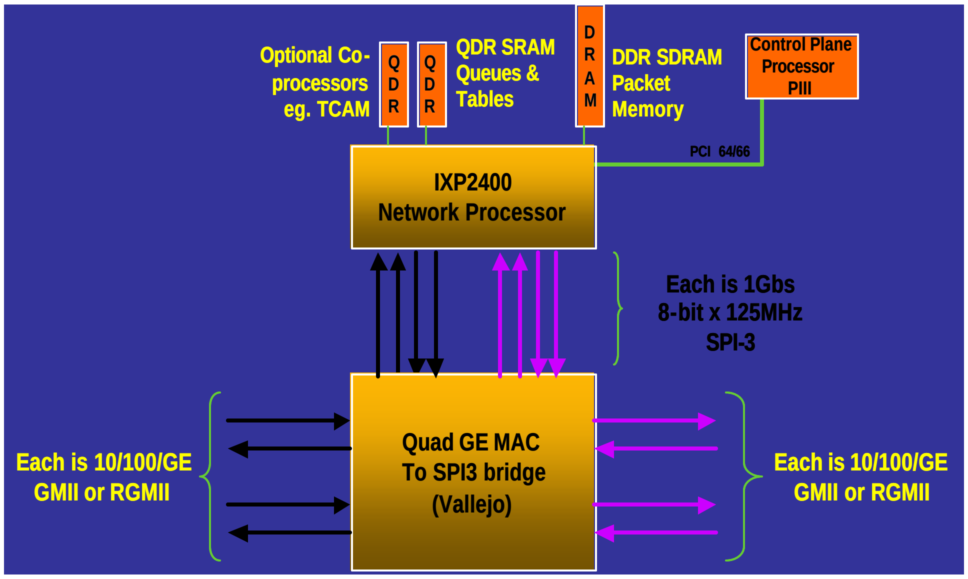


Figure 1 - IXP2400 based line card solution

Figure 1 shows the system configuration for a full duplex OC-48 line card implemented using the IXP2400 network processor.

Figure 2 shows a single chip IXP2400 based line card supporting 4Gb/s data rate.

Figure 2 – Single IXP2400 based solution

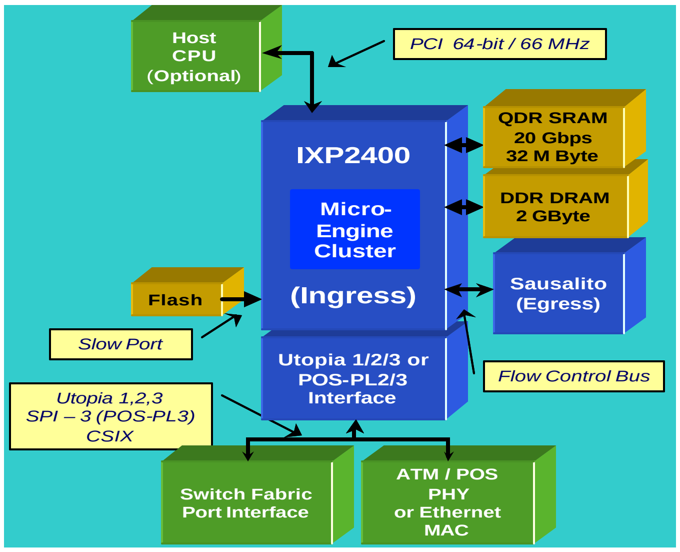

Optional Co-processors eg. TCAM
QDR
QDR
QDR SRAM Queues & Tables
DRAM
DDR SDRAM Packet Memory
Control Plane Processor PIII
PCI 64/66
IXP2400 Network Processor
Each is 1Gbs 8-bit x 125MHz SPI-3
Quad GE MAC To SPI3 bridge (Vallejo)
Each is 10/100/GE GMII or RGMII
Each is 10/100/GE GMII or RGMII

2. Hardware Architecture

This section provides a brief overview of the Sausalito external interfaces and the internal hardware architecture.

Figure 3 shows the external interfaces supported by the IXP2400 network processor.

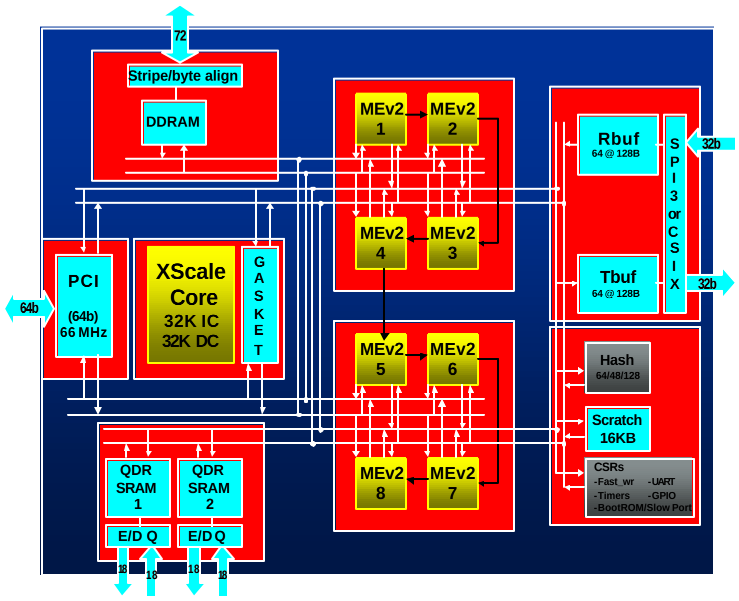
Figure 3 – IXP2400 External Interfaces


Host CPU (Optional)
PCI 64-bit / 66 MHz
IXP2400
Micro-Engine Cluster
(Ingress)
QDR SRAM 20 Gbps 32 M Byte
DDR DRAM 2 GByte
Sausalito (Egress)
Flash
Slow Port
Flow Control Bus
Utopia 1/2/3 or POS-PL2/3 Interface
Utopia 1,2,3 SPI – 3 (POS-PL3) CSIX
Switch Fabric Port Interface
ATM / POS PHY or Ethernet MAC

- 32b RX and TX interface that supports Utopia 1,2,3, POS-PHY-L2, SPI3 and CSIX protocols. This interface can be independently configured to be 1x32, 2x16, 4x8 or 2x8+1x16 and can be clocked at 25MHz-125MHz. At 125 MHz, the interface provides a peak bandwidth of 4Gb/s in and out of the chip.
- 4b/8b CSIX flow control bus that is used to communicate fabric flow control information to the egress IXP2400. At 125 MHz, this interface provides up to 1Gb/s of peak bandwidth for flow control messages.
- 1 channel of DDR DRAM running at 150/300MHz, providing 19.2Gb/s of peak DRAM bandwidth. The channel can support up to 2 GB of DRAM.
- 2 channels of QDR SRAM running at 200/400 MHz, providing 12.8Gb/s of read bandwidth and 12.8Gb/s of write bandwidth. Up to 32MB of SRAM can be populated on the 2 channels.
- 64b PCI running at 66 MHz, providing a peak bandwidth of 4.2Gb/s

Figure 4 shows the internal architecture of the IXP2400.

Figure 4 - IXP2400 Internal Architecture.



2.1 Intel® XScale Ô Core Features

Intel® XScale™ core is a general purpose 32-bit RISC processor (ARM* Version 5 Architecture compliant) used to initialize and manage the network processor, and used for exception handling, slow-path processing and other control plane tasks. The Intel® XScale™ micro architecture incorporates an extensive list of architecture features that allows it to achieve high performance.

Multiply/Accumulate (MAC)

The MAC unit supports early termination of multiplies/accumulates in two cycles and can sustain a throughput of a MAC operation every cycle. Several architectural enhancements were made to the MAC to support audio coding algorithms, which include a 40-bit accumulator and support for 16-bit packed values.

Memory Management

The Intel® XScale™ micro architecture implements the Memory Management Unit (MMU) Architecture specified in the *ARM Architecture Reference Manual*. The MMU provides access protection and virtual to physical address translation.

The MMU Architecture also specifies the caching policies for the instruction cache and data memory. These policies are specified as page attributes and include:

- Identifying code as cacheable or non-cacheable
- Selecting between the mini-data cache or data cache
- Write-back or write-through data caching
- Enabling data write allocation policy
- Enabling the write buffer to coalesce stores to external memory

Instruction Cache

The Intel® XScale™ micro architecture implements a 32-Kbyte, 32-way set associative instruction cache with a line size of 32 bytes. All requests that “miss” the instruction cache generate a 32-byte read request to external memory. A mechanism to lock critical code within the cache is also provided.

Branch Target Buffer

The Intel® XScale™ micro architecture provides a Branch Target Buffer (BTB) to predict the outcome of branch type instructions. It provides storage for the target address of branch type instructions and predicts the next address to present to the instruction cache when the current instruction address is that of a branch. The BTB holds 128 entries.

Data Cache

The Intel® XScale™ micro architecture implements a 32-Kbyte; 32-way set associative data cache and a 2-Kbyte, 2-way set associative mini-data cache. Each cache has a line size of 32 bytes, and supports write-through or write-back caching. The data/mini-data cache is controlled by page attributes defined in the MMU Architecture and by coprocessor 15.

The Intel® XScale™ micro architecture allows applications to re-configure a portion of the data cache as data RAM. Software may place special tables or frequently used variables in this RAM.

2.2 Microengine (ME)

The Microengines (MEs) do most of the programmable per packet processing in the IXP2400. There are 8 Microengines, connected in two clusters of 4 ME as shown in figure 4. The Microengines have access to all shared resources (SRAM, DRAM, MSF, etc) as well as private connections between adjacent Microengines (referred to as “next neighbors”).

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 interleave operation—there is often at least one thread ready to run while others are blocked. This improves the usage of the ME resources.

Control Store

The Control Store is a RAM, which holds the program that the Microengine executes. It holds 4096 instructions, each of which is 40-bits wide. It is initialized by the Intel® XScale™ core. The Control Store is protected by parity against soft errors.

Contexts

There are eight hardware Contexts (or threads) available in the Microengine. To allow for efficient context swapping, each Context has its own register set, Program Counter, and Context specific Local Registers. Having a copy per Context eliminates the need to move Context specific information to/ from shared memory and Microengine registers for each Context swap. Fast context swapping allows a Context to do computation while other Contexts wait for IO (typically external memory accesses) to complete or for a signal from another Context or hardware unit.

Data path Registers

Each microengine contains four types of 32-bit data path registers:

1. 256 General Purpose Registers
2. 512 Transfer Registers
3. 128 Next Neighbor Registers
4. 640 32-bit words of Local Memory

GPRs are used for general programming purposes. They are read and written exclusively under program control. GPRs, when used as a source in an instruction, supply operands to the execution data path. When used as a destination in an instruction, they are written with the result of the execution data path. The specific GPRs selected are encoded in the instruction.

Transfer Registers (abbreviated Xfer Registers) are used for transferring data to and from the Microengine and locations external to the Microengine, (for example DRAM, SRAM etc). There are four types of transfer registers. Typically, the external units access the Transfer Registers in response to instructions executed by the Microengines. However, it is possible for an external unit to access a given Microengine’s Transfer Registers either autonomously, or under control of a different Microengine, or The Intel® XScale™ core, etc.

Next Neighbor Registers, when used as a source in an instruction, supply operands to the execution data path. NN registers in a microengine can be written either by an adjacent Microengine or the same microengine.

Local Memory (LM)

Local Memory is addressable storage located in the Microengine. LM is read and written exclusively under program control. LM supplies operands to the execution data path as a source, and receives results as a destination. The specific LM location selected is based on the value in one of the LM_ADDR Registers, which are written by local_csr_wr instructions.

Special Hardware Blocks in the ME

The ME also provides the following special hardware blocks to assist in various packet-processing tasks.

- CRC unit – compute 16b and 32b CRC. This accelerates the performance of ATM AAL5 SAR applications.
- Pseudo random number generator – assist in supporting QoS algorithms for congestion avoidance, e.g. WRED, RED
- Time-stamp, timer – assist in supporting metering, policing, rate shaping functionality required in IP DiffServ and ATM traffic management services.
- Multiply unit – assist in QoS blocks such as policing, congestion avoidance.

2.3 DDR DRAM

The Memory controller is responsible for controlling the off chip DDR DRAM and provides a mechanism for other functional units in Sausalito to access the DRAM. Sausalito supports a single 64-bit channel (72 bit with ECC) of DRAM. DRAM sizes of 64Mb, 128Mb, 256Mb, 512Mb and 1Gb are supported; The DRAM channel can be populated with either a single or double-sided DIMM.

An address space of 2 GB is allocated to DRAM. The memory space is guaranteed to be contiguous from a software perspective. If less than 2 GB of memory is present, the upper part of the address space is aliased into the lower part of the address space and should not be used by software.

Reads and writes to DRAM are generated by Microengines, XScale™ and PCI bus masters. They are connected to the controllers via the Command Bus and Push and Pull Busses. The memory controller takes commands from these sources and services them. ECC (Error Correcting Code) is supported, but can be disabled. Enabling ECC requires that x72 DIMMs be used. If ECC is disabled, x64 DIMMs can be used.

2.4 SRAM

The Sausalito Network Processor has two independent SRAM controllers, which each support pipelined QDR synchronous static RAM (SRAM) and/or a coprocessor that adheres to QDR signaling. Any or all controllers can be left unpopulated if the application does not need to use them. SRAM are accessible by the Microengines, the Intel® XScale™ core, and the PCI Unit (external bus masters and DMA).

The memory is logically four bytes (32-bits) wide; physically the data pins are two bytes wide and are double clocked. Byte parity is supported. Each of the four bytes has a parity bit, which is written when the byte is written and checked when the data is read. There are byte enables that select which bytes to write for writes of less than 32-bits.

Queue Data Structure Commands

The ability to enqueue and dequeue data buffers at a fast rate is key to meeting line-rate performance. This is a difficult problem as it involves dependent memory references that must be turned around very quickly. The SRAM controller includes a data structure (called the Q_array) and associated control logic in order to perform efficient enqueue and dequeue operations. The Q_array has 64 entries, each of which can be used in one of the following ways.

- Linked-list queue descriptor

- Cache of recently used linked-list queue descriptor (the backing store for the cache is in SRAM)
- Ring descriptor.

2.5 Media and Switch Fabric Interface

The Media and Switch Fabric (MSF) Interface is used to connect IXP2400 to a physical layer device (PHY) and/or to a switch fabric. The MSF has the following major features:

Separate and independent 32 bit receive and transmit busses. Each bus may be configured independently. A configurable bus interface; the bus may function as a single 32 bit bus, or it can be channelized into independent busses: two 16 bit busses, four 8 bit, or one 16 bit bus and two 8 bit busses. Each channel may be configured to operate in UTOPIA, POS-PHY, SPI3 or CSIX modes. The Media bus operates from 25 to 125 MHz.

2.6 Scratchpad and Hash Unit

The scratchpad unit provides 16KB of on-chip SRAM memory that can be used for general-purpose operations by the Intel® XScale™ core and the ME. The scratch also provides 16 hardware rings that can be used for communication between MEs and the core.

The hash unit provides a polynomial hash accelerator. The Intel® XScale™ core and Microengines can use it to offload hash calculations in applications such as ATM VC/VP lookup, IP 5-tuple classification.

2.7 PCI, Intel® XScale Ô Peripherals and Performance Monitors

IXP2400 supports one 64-bit PCI Rev 2.2 compliant IO bus. PCI can be used to either connect to a Host processor, or to attach PCI compliant peripheral devices. The PCI bus is clocked at 66 MHz.

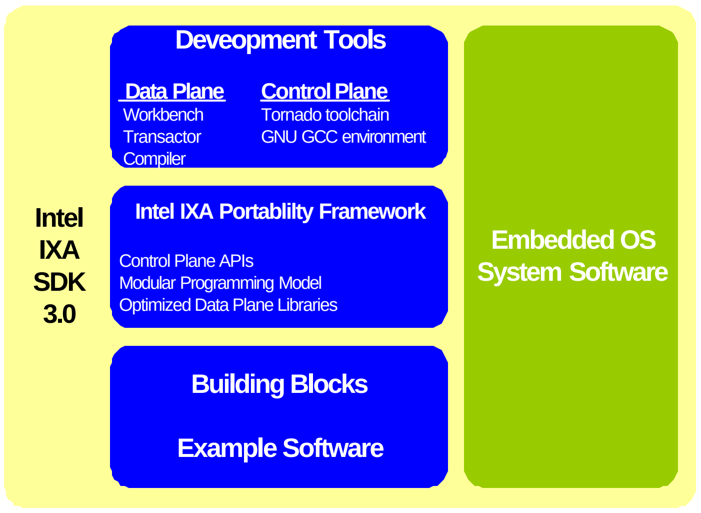
Intel® XScale™ technology Peripherals (XPI) Interface is used to connect the IXP2400 to Interrupt Controller, Timers, UART, General Purpose IO (GPIO) and Flash ROM.

The performance monitor unit provides counters that can be programmed to count selected internal chip hardware events. These counters are used to analyze and tune performance of the software running on the chip.

3. Software Development Environment

As network processors become widely used, the quality and completeness of the associated software development environment takes on greater importance. To benefit from the programmability of the network processor, developers need software tools in several key areas. These include data-plane code development, control-plane code development and system development tools.

Figure 5: Intel IXA SDK 3.0 Components.



The Intel® Internet Exchange Architecture SDK 3.0 enables hardware and software development to proceed in parallel. The SDK provides an easy-to-use graphical development environment that can be used to develop, debug, optimize and simulate code. By using the development tools, network building blocks and the modular development framework, a customer can achieve an unparalled time-to-market advantage. Figure 5 shows the various components of Intel® IXA SDK 3.0. These are components are explained in greater detail below.

3.1 Development Tools

The Microengine Development Environment (MDE) consists of the Developer Workbench and the Transactor. The Developer Workbench provides an integrated development environment for the development, debugging, profiling and simulation of microengine code. The Transactor is a cycle-accurate simulator of the network processor and can be used to identify opportunities for code optimization by capturing history and statistics that show cycle-by-cycle interactions among the microengines and memory units. The transactor also includes a scripting engine for setting test configurations and creating test cases.

The MDE provides developers with a choice of two programming languages to develop microengine code. The Microengine C Compiler offers faster time-to-market, optimum code portability, and the benefits of a familiar high-level programming language that provides isolation from specific hardware. Microengine C extensions to the ANSI C standard allow direct access to specialized hardware functions available on the microengines. The second option is to work in microcode to maximize application performance and minimize space utilization. Microcode can also be used within Microengine C programs for optimum results.

SDK 3.0 supports WindRiver VxWorks 5.4 (big-endian) and Linux (big-endian) operating systems on the XScale™ core of the IXP2400 processor. The Intel® Xscale micro architecture is widely supported by a large ecosystem of tools from leading third-party vendors. This includes cross-development tools such as Wind River Tornado and Linux-hosted or Windows-hosted GNU GCC environment. These tool chains enable cross-compiling, linking and source-level debugging on the target platform. They can be used for building and debugging user applications, OS kernels, kernel-mode drivers and flash boot images.

3.2 Programming Model

The microengines employs a software pipeline model in the fast path processing of the packets. Typically there are two software pipelines: the first is the receive pipeline that receives packets from the POS-PHY L3 (SPI-3) or CSIX interfaces and places the packets onto transmit queues; the second is the transmit pipeline that removes packets from the queues and transmits them out the SPI-3 or CSIX interfaces.

Each pipeline is composed of smaller elements called pipeline stages. Some typical examples of pipeline stages include packet reception, L3 forwarding, metering, WRED. The developer will combine each of these stages to build the full pipeline.

There are two types of pipelines: context pipeline and functional pipeline. These are explained in greater detail below.

3.2.1 Context Pipeline

The advantage of the context pipeline is that the entire ME program memory space can be dedicated to a single function. This is important when a function supports many variations that result in a large program memory footprint. The context pipeline is also desirable when a pipe stage needs to maintain state (bit vectors, or tables) to perform its work. The context pipe stage can use the local memory to store this state eliminating the latency of accessing external memory.

Figure 6 shows the implementation of context pipelines in the ME.

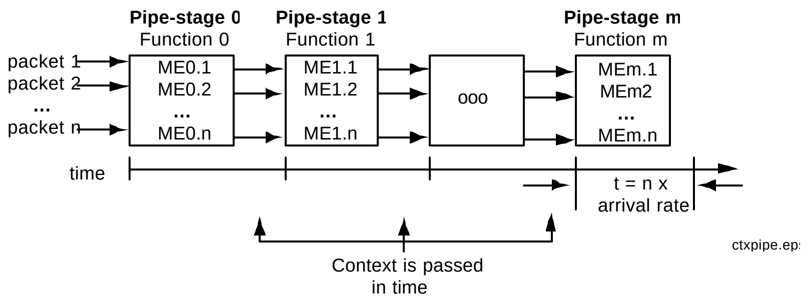


Figure 6 – Context Pipeline

Cases where the context pipeline is not desirable are ones in which the amount of context passed to and from the pipe stage is so large that it affects system performance. Another disadvantage of the context pipe stage is that all pipe stages must execute at minimum packet arrival rates. This may make partitioning the application into stages more difficult.

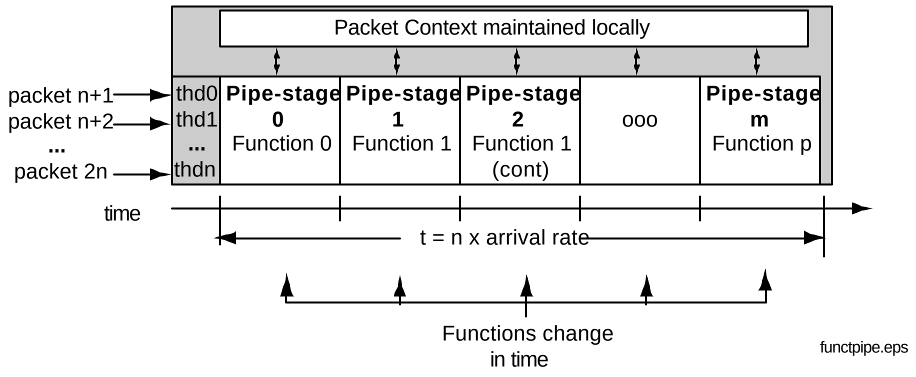
3.2.2 Functional Pipeline

In a functional pipeline, the context remains with an ME while different functions are performed on the packet as the time progresses. The ME execution time is divided into "n" pipe-stages and each pipe-stage performs a different function. A single ME can constitute a functional pipeline. The functional pipeline gets its name from the fact that it is the function that moves through the pipeline.

Packets are assigned to the ME threads in strict order, so that if there were "n" threads executing on an ME, the first thread, "A" must complete processing its first packet before "n" + 1st packet arrives so that it can begin processing the "n" + 1 packet.

Figure 7 shows a functional pipeline implementation.

Figure 7 – Functional Pipeline



There is little benefit to dividing a single ME's execution time into functional pipe stages. The real benefits comes from having more than one MEs execute the same functional pipeline in parallel.

- A packet remains with a thread for a longer period of time as more MEs are added to the Functional pipe-stage. For e.g., with 4 ME in the functional pipeline, the packet remains with a thread 32 packet arrival times (8 threads x 4 MEs) because thread ME0.0 is not required to accept another packet until all the other threads get their packets.
- The number of pipe-stages is equal to the number of MEs in pipeline. This ensures that a particular pipe-stage only executes in a one ME at any one time. This is required to provide a pipeline model that supports critical sections. A critical section is one in which an ME thread is

provided exclusive access to a resource (such as CRC residue, reassembly context, or a statistic) in external memory. Critical sections are described in more detail below.

- Functions can be distributed across one or more pipe-stages, however the exclusive access to resources as describe above cannot be supported in these pipe-stages.

When designing a functional pipeline, the goal is to identify critical sections and place them into their own pipeline stage. The non-critical section code then natural falls into pipe stages that become interleaved with the critical sections. Non-critical code that takes longer than a pipe-stage time to execute can be allocated more than one pipe-stage.

The advantages of Functional pipelines are:

- Unlike the Context pipeline, there is no need to pass the context between each pipe-stage since it remains locally within the ME.
- Supports a longer execution period than context pipe-stages.
- If one pipe stage does not consume it allocated time budget, this time can be used by another pipe stage (if the longer pipe stage is not a critical section).

The disadvantages of Functional pipelines are:

- The entire ME program memory space must support multiple functions
- Function control must be passed between stages therefore it should be minimized.
- Mutual exclusion may be more difficult because you have multiple MEs accessing the same data structures.

3.2.3 Elasticity Buffers

When a pipe-stage transition occurs between two MEs it is beneficial if the two stages do not need to run in lock-step. Instead an elasticity buffer (implemented as a ring) is used and this has an advantage over executing in lock-step. Each pipe-stage must be designed to execute in the time allocated to the pipe-stage, however elasticity buffers accommodate jitter in a pipe-stage execution. So if a pipe stage falls behind in execution, due to a system anomaly such as an unusually high utilization of a memory unit over a short time period, the elasticity buffer will allow the pipe-stage context to be buffered so that the previous pipe-stage are not stalled waiting for the next stage to complete. Statistically, the pipe-stage that falls behind will be able to catch up with its processing and system will normalize. Elasticity buffers also allow different heartbeats to the different pipelines.

The IXP2400 processor supports multiple methods for implementing elasticity buffers for communication between the pipe stages:

- SRAM/Scratch rings: These are multi producer-multi consumer message passing queues maintained by hardware.

- Next neighbor rings: These are single producer-single consumer optimized message passing queues between adjacent microengines.

3.2.4 Critical Sections

A critical section is a section of code where it is assured that only one ME thread has exclusive modification privileges for a global resource (such as a location in memory) at anyone time. The first part of the discussion focuses on providing exclusive modification privileges between the MEs. The second part of the discussion focuses on providing supporting exclusive access between the threads in an ME.

Exclusive Modification Privileges between MEs

To ensure exclusive modification privileges between MEs, the following requirements must be met:

1. Only one function modifies the critical section resource.
2. The function that modifies the critical section resource executes in a single pipe-stage.
3. The pipeline is design so that only one ME executes a pipe-stage at any one time.

In a context pipeline, each ME is assigned exclusive modification privileges to the critical data, satisfying requirement 1. Requirement 2 and 3 are satisfied because each pipe-stage is partitioned in to different functions and only on ME executes a specific function.

In a Functional pipeline, an ME should not transition into a critical section pipe-stage unless it can be assured that its "next" MEs has transitioned out of the critical section. This can be accomplished by placing a fence around the critical section using inter-thread signaling. There are four ways to signal another thread using inter-thread signaling

Exclusive Modification Privileges between Threads in an ME

A critical section involves three steps:

- Reading a resource
- Modifying the resource
- Writing back the modified data.

If more that one thread in a pipe stage is required to modify the same critical data, a latency penalty will be incurred if each thread reads the data from external memory modifies it and writes the data back. To reduce the latency penalty associated with the read and write, the ME threads can use the CAM to fold these operations into a single read, multiple modifications and depending on the cache eviction policy either one or more write operations. In this case, the ME thread is assured exclusive access to the data by performing the modification and write operations on the critical data without swapping out.

3.3 Portability Framework

The Intel® IXA Portability Framework is one of the foundations of the Intel® Internet Exchange Architecture. It enables fast and cost-effective code development, while protecting software investments through software portability and the reuse of code for the microengines and the Intel® XScale™ core across current and future generations of IXA network processors.

The portability framework enables the development of modular, portable code blocks and integration of third-party software products for longer product life and easier maintenance, while eliminating the time-consuming development of crucial infrastructure software. The Intel IXA portability framework includes:

- Optimized data-plane libraries for hardware abstraction, protocol processing and utility functions.
- A modular programming model for software pipelining
- A library of standards-based Network Processor Forum (NPF) APIs for communication with control plane protocol stacks.

The microengine modular programming model allows partitioning of functional blocks across microengines and threads, as well as combining independent modular building blocks into a single managed pipeline. This model facilitates the retargeting of code between Intel IXA network processors with differing number of microengines and threads, simplifies code reuse and enables easier integration with software supplied by third-party vendors.

4. IXP2400 system configurations and performance analysis

IPv4 and IPv6 forwarding, IP DiffServ and QoS, ATM AAL5 and AAL2 SAR function, ATM policing and traffic shaping, IP/UDP/RTP header compression, encap/decap, tunneling etc are key functions performed by equipment in the target market space for the IXP2400. This section presents the IXP2400 performance for some of the key applications. Table 1 summarizes performance of the IXP2400 running applications representative of the target market segments.

Table 1 – IXP2400 Performance Summary

Application	Media Interface	Min Packet Size	IXP2400 Performance
IPv4 forwarding + IP DiffServ	SPI3	46B	OC-48, 2.5Gb/s
IPv4 forwarding + IP DiffServ	Ethernet	64B	4Gb/s
ATM AAL5 SAR + TM4.1 compliant traffic shapers	Utopia	53B	OC-48, 2.5Gb/s
IPv6 forwarding + IP DiffServ	SPI3	66B	OC-48, 2.5Gb/s
VoAAL2	Utopia/SPI3	40B	16K voice channels
AAL2/AAL5 in wireless (Node-B/RNC)	Utopia/SPI3	53B	Full duplex OC12

- IXP2400 supports OC-48 line rate (2.5Gb/s) executing an IPv4/IPv6 forwarding + IP DiffServ application for min sized POS packet. IXP2400 also supports 4Gb/s line rate for this application with 64B min Ethernet packets.
- IXP2400 supports ATM AAL5 segmentation & re-assembly and TM4.1 compliant CBR, VBR and UBR traffic shapers for min sized AAL5 frames (53B) at OC-48 line rate.
- IXP2400 supports 16K voice channels (2*OC12 each direction) for the VoAAL2 application that involves AAL2 SAR (ITU-T I.363.2) and audio services processing such as jitter buffering etc (as specified in ITU-T I.366.2)
- IXP2400 supports full duplex OC12 data rate for AAL2 CPS SAR & SSSAR (ITU-T 363.2 and ITU-T I.366.1), IP/UDP processing and AAL5 SAR processing applications in the wireless RNC and Node-B.

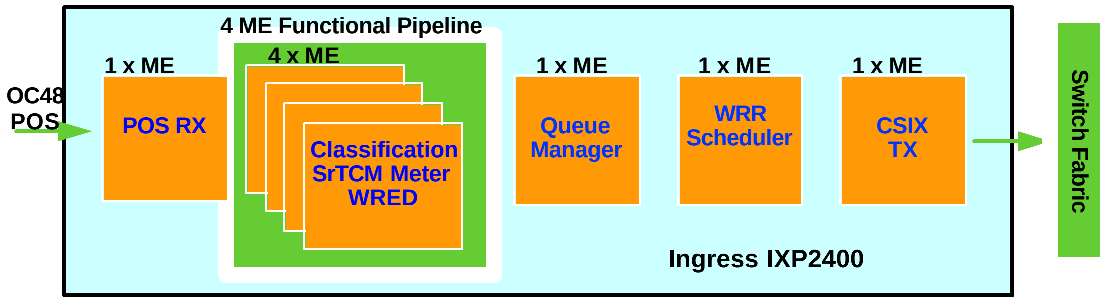
4.1 IPv4 forwarding + IP DiffServ application

The configuration shown in Figure 1 is used for analyzing the performance for this application. At OC48 line rate, the inter-arrival time for a minimum size POS packet of 46B is 88 ME cycles (assuming ME frequency is 600 MHz). In order to achieve line rate, the ME must execute each pipeline stage within this budget of 88 cycles. The blocks of this application running on the ingress and egress IXP2400 are specified below.

- Ingress processor blocks
    - POS frame receive, IP header classification using 5-tuple lookup, route lookup, SrTCM meter, WRED, queuing, WRR scheduler and CSIX segmentation and transmit
- Egress processor blocks
    - CSIX cell re-assembly, further IP classification, SrTCM meter, WRED, queuing, DRR scheduler, POS transmit.

Figure 8 shows the ME partition for this application.

Figure 8 – ME Partition for IPv4 forwarding + DiffServ application



5. Summary

Next generation access and edge equipment require flexible programming, high performance, low power consumption, and small real estate. These applications require support for a wide range of functions such as IP forwarding with QoS, ATM AAL5 and AAL2 Segmentation and Reassembly and ATM traffic shaping, header compression etc. The IXP2400 network processor has been optimized to meet these requirements. The eight microengines supporting 5.4 Giga ops/sec provide fully flexible programming and processing power to meet 2.5Gbps and 4Gbps wire-rate performance. The flexible media interface allows a variety of media devices ranging from OC3 to OC48 speeds to be connected without glue logic to the IXP2400 for easier design and lower system cost. Performance analysis demonstrates that the Intel® IXP2400 network processor is an ideal product for meeting these requirements at OC-48 wire-rate.

Table 2 provides the technology summary for the IXP2400 Network Processor

Table 2 – IXP2400 Technology Summary

Technology:	Manufactured in 0.18 micron Intel® process.
Package:	1356 Ball FCBGA 37.5mm x 37.5mm, 1mm solder ball pitch
Required Voltages:	1.3 V core, 1.5 V QDR, 2.5 V DDR and 3.3 V I/O
Power:	10W typical and 12.4 W max

References

Additional information on Intel Corporation network processor offerings are available at

[1] http://www.intel.com/design/network/products/npfamily/index.htm

[2] http://www.intel.com/design/network/devnet/