Intel IXP 1200
Network Processor

Architecture and Programming

Andreas Mitschele-Thiel
Contents

- IXP 1200 Hardware Architecture
- IXP 1200 Programming
IXP 1200 Hardware

- Architecture
- StrongARM
- Microengine
- Interface Units
- Bus bandwidths

References
- IXP 1200 data sheet
- IXP 1200 HW reference manual
Andreas Mitschele-Thiel
IXP 1200 System Architecture

- **SSRAM (8 Mbytes Max)**
- **BootROM (8 Mbytes Max)**
- **SlowPort Devices (2 Mbytes Max)**
- **Network Interface Devices**
- **Intel® IXP1200 Processor**
- **SDRAM (256 Mbytes Max)**
  - Command
  - 64 Data
  - JTAG
  - Serial Interface
- **Another IXP1200**

- **PCI Bus (33-66Mhz)**
- **IX Bus**
- Control
- Data 32
- Control and Status
- Network
- Data
IXP 1200 Architecture Overview

- symmetric array of six RISC data processors (microengines)
  - loosely coupled
  - decoupling of the functional units for the IX Bus, PCI Bus, SDRAM, and SRAM interfaces from the execution pipelines (extensive use of FIFO queues, and event task signaling)
  - semaphores
  - 4 HW threads per microengine (zero-overhead context switching between threads)

- a StrongARM SA-1100 RISC processor core (32-bit)

- two memory interfaces (SDRAM and SRAM)

- a PCI Interface

- an IX Bus Interface
IXP 1200 Key Architecture Features

• Multi-Processing – process multiple network packets in parallel

• Distributed Data Storage architecture - local register file of 256 (32-bit) registers per microengine (128 transfer registers); load-store architecture

• Hardware Multi-Threading - minimal context switching overhead when waiting for data

• Active Memory Optimizations - executes a series of memory requests in the most efficient manner, thereby increasing effective memory bandwidth.

• Concurrent Data Movement – multiple independent data and control busses

• Block Data Movement - efficient movement of large amounts of data

See IXP 1200 HW reference manual, ch 2.3 for details
Data Transfer Architecture

Multiple, independent high speed internal busses
StrongARM SA-1100

Compatible with the StrongARM processor family currently used in applications such as network computers, PDAs, palmtop computers and portable telephones

StrongARM may be used as

- host processor running an RTOS
- exception handling and higher layer protocol processing

StrongARM details

- 5-stage pipelined processor
- Big-Endian or Little-Endian mode
- 16K bytes instruction cache (512 lines of 32 bytes)
- 8K bytes data cache (256 lines of 32 bytes; write-through, allocate on read)
- 512 byte minicache in core intended for data that is read once, operated on, and then discarded (reduce flushing of the main data cache)
# StrongARM Core – IXP Internal Connections

<table>
<thead>
<tr>
<th>Unit</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCI Unit</td>
<td>Full access to the PCI Bus, including all PCI bus transactions.</td>
</tr>
<tr>
<td></td>
<td>Full access to PCI Unit registers.</td>
</tr>
<tr>
<td></td>
<td>Separate, shared 32-bit bus between the StrongARM core (ATU) data bus and the PCI Unit.</td>
</tr>
<tr>
<td>SDRAM Unit</td>
<td>Full access to SDRAM.</td>
</tr>
<tr>
<td>SRAM Unit</td>
<td>Full access to SRAM, including Flash and other devices hooked up to the SRAM Bus.</td>
</tr>
<tr>
<td>IX Bus Unit</td>
<td>Access to Control and Status registers within the IX Bus Unit.</td>
</tr>
<tr>
<td></td>
<td>Access to Scratchpad RAM within the IX Bus Unit.</td>
</tr>
<tr>
<td></td>
<td>No access to the Receive or Transmit FIFOs or the IX data bus.</td>
</tr>
<tr>
<td>Microengines</td>
<td>Access to Microengines’ Program Control Store to program the Microengines.</td>
</tr>
<tr>
<td></td>
<td>Access to Control and Status Registers, including PC (Program Counter).</td>
</tr>
<tr>
<td></td>
<td>No access to Microengine Transfer Registers.</td>
</tr>
</tbody>
</table>
Microengine – Data

- 32-bit, multithreaded RISC microengine
- clock frequency up to 232 MHz, single operation (one stage) per clock
- 5-stage pipeline (instr. fetch, decode, fetch regs, exec, write regs)
- data movement and processing without assistance from the StrongARM
- four independent program counters per microengine
- zero overhead context switching (5 clock cycles to flush pipe)
- hardware semaphores
- both ALU and shift operations in a single cycle
- instruction set for networking and communications applications
  - bit, byte, word (16 bit) and longword (32 bit) operations
  - quick and efficient data forwarding
- local microengine registers
  - 1024 (2048) x 32 bit RAM instruction store (32 bit per instruction)
  - 128 x 32-bit general purpose registers
  - 128 x 32-bit transfer registers to service the SRAM and SDRAM units
## Microengine – IXP Internal Connections

<table>
<thead>
<tr>
<th>Unit</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>StrongARM Core</td>
<td>The Microengines may interrupt the StrongARM core. The StrongARM core can read a register to determine which Microengine generated the interrupt. No other access to StrongARM Core.</td>
</tr>
<tr>
<td>PCI Unit</td>
<td>No access to the PCI Bus. Access only to the Control and Status Registers for the two DMA Controllers in the PCI Unit. By programming these registers, the Microengines may initiate DMA transfers between a block of SDRAM memory and a PCI device.</td>
</tr>
<tr>
<td>SDRAM Unit</td>
<td>Full access to SDRAM.</td>
</tr>
<tr>
<td>SRAM Unit</td>
<td>Full access to SRAM, including Flash and other devices hooked up to the SRAM Bus.</td>
</tr>
<tr>
<td>IX Bus Unit</td>
<td>Full access to the IX Bus Unit, including the Scratchpad RAM, Hardware Hashing Unit, Receive and Transmit FIFOs, Ready Bus, and Control and Status Registers.</td>
</tr>
<tr>
<td>Microengines</td>
<td>Each Microengine has access to its Program Control Store so it can execute instructions. The Microengines do not have read or write access to the Control Store (they cannot read or write their own Control Store, only the StrongARM core can do that). Inter-thread signalling between Microengine threads is provided. Each Microengine is self-contained, so one Microengine cannot access the Control Store, or Registers of another Microengine.</td>
</tr>
</tbody>
</table>
FBI Unit and IX Bus

IX bus:

- IXP external bus
- 2 modes of operation (2 x 32 bit unidirectional or 1 x 64 bit bidirectional)

Purpose of FBI unit and IX bus:

- servicing fast peripherals, such as MAC-layer devices, on the IX bus
- moving data to and from the IXP1200 receive and transmit FIFOs

FBI unit contains

- transmit and receive FIFO elements
- control and status registers (CSRs)
- a 4 Kbyte scratchpad RAM
- a hash unit for generating 48- and 64-bit hash keys
- drivers and receivers for the IX Bus
External Memory Units

SRAM

• up to 8 MB of fast memory. e.g. for look-up tables
• up to 8 MB bootROM
• 2 MB for address space for peripheral device access

SDRAM

• up to 258 MB of low cost memory for forwarding information and transmit queues
• access from StrongARM, microengines and PCI bus (memory locations, transfer registers, or the transmit and receive FIFOs in the FBI unit)
• operation at half the core frequency (peak bandwidth of 928 MBps at 232 MHz)
**IXP 1200 Performance**

- operating frequencies of 166, 200, or 232 MHz (microengines and StrongARM)

- the StrongARM* core processor and six RISC microengines can forward greater than 3 million Ethernet packets per second

- a multi-IXP1200 system scales linearly

=> a system comprised of eight IXP1200s can process 24 million packets per second
# Performance – Internal and External Bandwidth

<table>
<thead>
<tr>
<th>Read/Write</th>
<th>Bandwidth Description</th>
<th>Calculation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>SRAM internal bandwidth to Microengine</td>
<td>$(32 \text{ bits} \times 232 \text{ MHz}) / 2$</td>
<td>3.7 Gbps</td>
</tr>
<tr>
<td>Write</td>
<td>SRAM internal bandwidth to Microengine</td>
<td>$(32 \text{ bits} \times 232 \text{ MHz}) / 2$</td>
<td>3.7 Gbps</td>
</tr>
<tr>
<td>Read/Write</td>
<td>IX Bus external (pin-side) bandwidth</td>
<td>$64 \text{ bits} \times 104 \text{ MHz}$</td>
<td>6.6 Gbps</td>
</tr>
<tr>
<td>Read</td>
<td>IX Bus internal bandwidth to Microengine</td>
<td>$(32 \text{ bits} \times 232 \text{ MHz}) / 2$</td>
<td>3.7 Gbps</td>
</tr>
<tr>
<td>Write</td>
<td>IX Bus internal bandwidth to Microengine</td>
<td>$(32 \text{ bits} \times 232 \text{ MHz}) / 2$</td>
<td>3.7 Gbps</td>
</tr>
</tbody>
</table>
IXP 1200 Programming

• Data Transfer for Ethernet Example
• Software Development Tools
• Microengine Programming

References
• IXP 1200 HW reference manual
• IXP 1200 development tools user guide
Data Transfer: Ethernet Example

1. The Ethernet MAC receives data.

2. A Microengine instructs the IX Bus Unit (via a Reference Command) to get the data from the MAC and store it in the Receive FIFO (RFIFO). The IX Bus unit takes care of the actual transfer, independent of the Microengine.

3. The IX Bus Unit signals a Microengine thread that the data has been transferred.

4. The Microengine instructs the SDRAM Unit to transfer the data from the Receive FIFO to SDRAM (this actually takes place in the background). The SDRAM unit takes care of the actual transfer, independent of the Microengine.

5. The Microengine transfers the first few bytes (header) into its Transfer Registers (via a Reference Command to the IX Bus Unit).

6. It processes the header (or however much of the packet needs to be inspected), and determines what to do. It can make use of tables stored in SRAM (or SDRAM) to do lookups.

7. It modifies the packet header, if necessary, and writes out the new header to SDRAM.

8. It instructs the SDRAM Unit to write the packet data out to the Transmit FIFO. The SDRAM unit takes care of the actual transfer, independent of the Microengine. When the transfer is complete, the SDRAM Unit notifies the Microengine thread.

9. The Microengine instructs the IX Bus Unit to transfer the data to the appropriate MAC.
Data Transfer: Ethernet Example

1. The Ethernet MAC receives data.
Data Transfer: Ethernet Example

2. A Microengine instructs the IX Bus Unit (via a Reference Command) to get the data from the MAC and store it in the Receive FIFO (RFIFO). The IX Bus unit takes care of the actual transfer, independent of the Microengine.
Data Transfer: Ethernet Example

3. The IX Bus Unit signals a Microengine thread that the data has been transferred.
Data Transfer: Ethernet Example

4. The Microengine instructs the SDRAM Unit to transfer the data from the Receive FIFO to SDRAM (this actually takes place in the background). The SDRAM unit takes care of the actual transfer, independent of the Microengine.
Data Transfer: Ethernet Example

5. The Microengine transfers the first few bytes (header) into its Transfer Registers (via a Reference Command to the IX Bus Unit).
Data Transfer: Ethernet Example

6. The microengine processes the header (or however much of the packet needs to be inspected), and determines what to do. It can make use of tables stored in SRAM (or SDRAM) to do lookups.
Data Transfer: Ethernet Example

7. It modifies the packet header, if necessary, and writes out the new header to SDRAM.
Data Transfer: Ethernet Example

8. The microengine instructs the SDRAM Unit to write the packet data out to the Transmit FIFO. The SDRAM unit takes care of the actual transfer, independent of the Microengine. When the transfer is complete, the SDRAM Unit notifies the Microengine thread.
Data Transfer: Ethernet Example

9. The Microengine instructs the IX Bus Unit to transfer the data to the appropriate MAC.
Software Development Tools

Assembler for the Microengine instruction set

StrongARM tools (C Compiler and Assembler) can be obtained from ARM Limited, or a number of other software tool vendors

Linker and Loader is provided to link and load the Microengine code with the StrongARM code.

Transactor

- a software model of the IXP1200 to support debugging (data accurate model of the IXP1200 on a cycle by cycle basis)
- software simulation interface (called foreign model) to link models of external devices (MACs or any custom devices) into the transductor

Developer’s Workbench

- graphically displays a history of each of the twenty four threads
- execution profiling, statistics gathering
- C-like scripting facility
- data-watch windows, breakpoints, queue statistics, etc.
Development Libraries

• IXP 1200 macro library (assembler macros)
• Microengine C networking library (C library, API portable across IXP processor family)
• Active Computing Element (ACE) library
  – high-level building blocks aiding in development of networking applications (based on C library and/or macro library)
  – e.g. layer 3 forwarding, NAT
IP Router on IXP Evaluation Board

Example: IP Router with Ethernet Interfaces

The IXP1200’s six Microengines are partitioned as follows:

- four Microengines (16 threads) are used for receive processing
- two Microengines (4 threads) are used for transmit processing
## Microengine Threads – Summary

<table>
<thead>
<tr>
<th>Microengine</th>
<th>Thread</th>
<th>Port / Task Assignment</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3 Receive</td>
<td>Thread 0</td>
<td>16 Receive Threads</td>
<td>Each of the receive threads is assigned to a specific port. Microengine 0 to receive ports 0, 1, 2, &amp; 3</td>
</tr>
<tr>
<td>Microengines</td>
<td>Thread 1</td>
<td></td>
<td>Microengine 1 to receive ports 4, 5, 6, &amp; 7</td>
</tr>
<tr>
<td></td>
<td>Thread 2</td>
<td></td>
<td>Microengine 2 to receive ports 8, 9, 10, &amp; 11</td>
</tr>
<tr>
<td></td>
<td>Thread 3</td>
<td></td>
<td>Microengine 3 to receive port 12, 13, 14, 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Thread ID = Port Number = RFIFO Element</td>
</tr>
<tr>
<td>4-5 Transmit</td>
<td>Thread 0</td>
<td>Transmit Scheduling</td>
<td>Microengine 4 transmits on even ports (0, 2,..14). (One scheduler thread and three transmit threads)</td>
</tr>
<tr>
<td>Microengines</td>
<td>Thread 1</td>
<td></td>
<td>Microengine 5 transmits on odd ports (1, 3, 15) (One scheduler thread and three transmit threads)</td>
</tr>
<tr>
<td></td>
<td>Thread 2</td>
<td>6 dynamically assigned Transmit Fill</td>
<td>Thread 0 of Microengine 5 is also used to do hardware initialization (calls ixp_hw_init()) and freelist (buffer) allocation.</td>
</tr>
<tr>
<td></td>
<td>Thread 3</td>
<td>Threads</td>
<td></td>
</tr>
</tbody>
</table>
Receive Processing

Receiving microengines:

– Each of the 16 receive threads runs identical code
– Each thread is bound to a specific port number and Receive FIFO element number.
– The relationship is:

  Receive Thread ID = Port Number = Receive FIFO Element Number

A receive thread obtains the receive ready flags to determine if its port has incoming data available.

If data is available, it creates and writes a Receive Request which causes the Receive State Machine (in the IX Bus Unit) to transfer the packet data from the MAC to the Receive FIFO element.

After the Receive Request completes, the thread moves the packet into a buffer in SDRAM and reads the packet header into the Microengine. It filters based on Ethernet protocol, verifies the header, performs an IP forwarding lookup, and modifies the header with the new address.

After writing the updated header to the packet buffer, the receive thread then enqueues the packet for output to either the StrongARM core or to an outgoing port via the transmit Microengines.
Receive Processing

- transfer the packet data from the IXF440 Octal MAC device to the receive FIFO
- store the packet data into SDRAM
- filter based on Ethernet protocol
- verify IP header
- perform IP lookup
- enqueue the packet for output to either the StrongARM core or to an outgoing port via the transmit Microengines

When a receive thread receives an mpacket that is the start of a new packet (that is, the SOP bit is asserted), the receive thread performs SOP processing.

IP lookup will be performed to get the output port number and also destination MAC address. Source and destination MAC addresses will be modified accordingly. When the input mpacket is not the last mpacket of a packet (EOP) the mpacket data is stored sequentially into SDRAM.

When a receive thread determines that the last mpacket of a packet has been received (EOP signalled) the receive thread enqueues the packet to either the StrongARM core for further processing or to a transmit scheduler so that it can be scheduled for transmit.
Transmit Processing

Transmitting Microengine:
- thread 0 in each of the transmit Microengines serves as transmit scheduler
- threads 1, 2, and 3 are transmit fill threads.

Transmit Scheduler:
- sequentially checks to see if any port has a packet waiting to be transmitted
- issues transmit assignments to Transmit Fill threads on the same Microengine
- each assignment specifies a port number.

Transmit Fill threads:
- all sixteen ports served by the transmit fill threads are serviced dynamically on a round-robin basis (transmit fill threads are not bound to specific ports)
- check to see if there is a valid assignment (make use of the packet link and queue descriptors to find the packet in SDRAM)
- packet data is then moved from SDRAM to the Transmit FIFO (the packet status bits are set (for example, SOP and/or EOP))
- the packet (or mpacket) is then transmitted to the IXF440 MAC port

See uC_Example.pdf for details.
Transmit Processing – Transmit Scheduler

One transmit scheduler per transmit microengine

- Transmit scheduler continuously polls for packets to be output on the ports it is responsible for.
- If a packet is ready for output, the transmit scheduler assigns a fill thread the task of transmitting one mpacket.
- If no packet data is available for output, the transmit scheduler assigns a fill thread a skip assignment (skip assignments are required because of the distributed transmit scheduler software architecture of the example design).

The transmit scheduler communicates with the fill threads, and the fill threads communicate with each other, using absolute registers.

Communication between the scheduler and the fill threads includes information such as transmit assignments and status updates.

The fill threads also communicate port status and output packet information with each other.
Data Paths
IXP Family

IXP 2400
• 8 microengines (5.4 GIPS)
• Intel 32-bit Xscale core (StrongARM successor)

IXP 2800
• 16 microengines (22.4 GIPS)
• Intel 32-bit Xscale core
References


• Intel: IXP 1200 HW reference manual

• Intel: IXP 1200 development tools user guide