ECE 697J – Advanced Topics in Computer Networks

IXP1200 Microengines

11/06/03
Overview

• More details on Microengines
  – Instruction Store
  – Registers
  – FBI Unit
  – Scratchpad
  – Hash Unit

• Programming Model
  – Active Computing Element (ACE) Abstraction
  – Structure of IXP Software

• Reference System and SDK
  – Next class
Last Class

• Control Processor
  – Basically normal processor with conventional OS

• Microengines
  – Simple microsequencers
  – Functional units have to addressed directly
  – Pipelining and hardware threading
uE Instruction Store

• Why not use SRAM or SDRAM for instruction store?
  – Too slow
  – Need one instruction per cycle
• Special instruction store memory on-chip
• Two design alternatives:
  – Each processing engine gets own instruction store
  – All processing engines share one instruction store
• Pros and cons?
  – Contention on shared storage but no replication needed
  – Most NPs: separated and small
• IXP1200 instruction store:
  – Each uE has own instruction store
  – 2048 instructions per store
• Instruction store is initialized by StrongARM before uE is activated
uE Registers

- Hardware registers are used by the uE to store intermediate results, transfer and control
- General-purpose registers:
  - 128 per uE
  - 32 bit each
- How are registers shared among threads?
  - Either shared among all contexts (requires careful use)
  - Divided among threads
- IXP supports both styles:
  - Absolute register addressing for shared access
  - Relative register address for context-specific access
Register Banks

• Registers are split into banks:

• Addressing specifies bank and register

• What are the benefits of multiple register banks?
  – Multiple data paths

• Programmer must carefully select registers
  – Best performance: each instruction uses one register from bank A and one from bank B
Transfer Registers

- Transfer registers are used for communication with other units
  - Memory: read/write value is placed in transfer register
  - Transfer registers are fast and can act as “buffer”
- IXP transfer registers
  - 128 registers in 4 groups
  - Each group is associated with SRAM or SDRAM interface for read or write
  - Each group is split into 4 contexts (same as gp registers)
- SRAM group can also access mapped I/O and Flash memory
Transfer Registers

<table>
<thead>
<tr>
<th>absolute addr.</th>
<th>used by</th>
<th>relative addr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 - 31</td>
<td>context 3 (8 regs.)</td>
<td>0 - 7</td>
</tr>
<tr>
<td>16 - 23</td>
<td>context 2 (8 regs.)</td>
<td>0 - 7</td>
</tr>
<tr>
<td>8 - 15</td>
<td>context 1 (8 regs.)</td>
<td>0 - 7</td>
</tr>
<tr>
<td>0 - 7</td>
<td>context 0 (8 regs.)</td>
<td>0 - 7</td>
</tr>
<tr>
<td>24 - 31</td>
<td>context 3 (8 regs.)</td>
<td>0 - 7</td>
</tr>
<tr>
<td>16 - 23</td>
<td>context 2 (8 regs.)</td>
<td>0 - 7</td>
</tr>
<tr>
<td>8 - 15</td>
<td>context 1 (8 regs.)</td>
<td>0 - 7</td>
</tr>
<tr>
<td>0 - 7</td>
<td>context 0 (8 regs.)</td>
<td>0 - 7</td>
</tr>
<tr>
<td>24 - 31</td>
<td>context 3 (8 regs.)</td>
<td>0 - 7</td>
</tr>
<tr>
<td>16 - 23</td>
<td>context 2 (8 regs.)</td>
<td>0 - 7</td>
</tr>
<tr>
<td>8 - 15</td>
<td>context 1 (8 regs.)</td>
<td>0 - 7</td>
</tr>
<tr>
<td>0 - 7</td>
<td>context 0 (8 regs.)</td>
<td>0 - 7</td>
</tr>
</tbody>
</table>
Local Control and Status Regs

- Local Control and Status Registers (CSRs)
  - CSRs are mapped into the address space of StrongARM
  - Subset of CSRs are local and control IXP1200

- Access to CSR
  - StrongARM can access all CSRs
  - uE can only access its own CSRs – not those of other uEs
## Local Control and Status Regs

<table>
<thead>
<tr>
<th>Local CSR</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>USTORE_ADDRESS</td>
<td>Load the microengine control store</td>
</tr>
<tr>
<td>USTORE_DATA</td>
<td>Load a value into the control store</td>
</tr>
<tr>
<td>ALU_OUTPUT</td>
<td>Debugging: allows StrongARM to read GPRs and transfer registers</td>
</tr>
<tr>
<td>ACTIVE_CTX_STS</td>
<td>Determine context status</td>
</tr>
<tr>
<td>ENABLE_SRAM_JOURNALING</td>
<td>Debugging: place journal in SRAM</td>
</tr>
<tr>
<td>CTX_ARB_CTL</td>
<td>Context arbiter control</td>
</tr>
<tr>
<td>CTX_ENABLE</td>
<td>Debugging: enable a context</td>
</tr>
<tr>
<td>CC_ENABLE</td>
<td>Enable condition codes</td>
</tr>
<tr>
<td>CTX_n_STS</td>
<td>Determine context status ‡</td>
</tr>
<tr>
<td>CTX_n_SIG_EVENTS</td>
<td>Determine signal status</td>
</tr>
<tr>
<td>CTX_n_WAKEUP_EVENTS</td>
<td>Determine which wakeup events are currently enabled</td>
</tr>
</tbody>
</table>
Inter-Processor Communication

- StrongARM can communicate with uE over CSRs
- Other paths of communication:
  - Thread-to-StrongARM
  - Thread-to-thread within on IXP1200
  - Thread-to-thread across multiple IXP1200
- Communication methods:
  - Interrupts
  - Shared memory
- uE-to-StrongARM:
  - uE raises interrupt or uses shared memory and polling
- Thread-to-thread:
  - On one IXP: signal event on internal “command bus”
  - On multiple IXPs: signal event via “ready bus”
FBI Unit

- Interface between processors and high-speed I/O components
- FBI has control over:
  - Scratchpad memory
  - Hash unit
  - FBI control and status registers
  - Control and operation of ready bus
  - Control and operation of IX bus
  - Data buffers that hold data arriving from the IX bus
  - Data buffers that hold data sent to the IX bus
- FBI unit offloads FIFO processing from uEs
Transmit and Receive FIFOs

- FIFOs are only communication between I/O and uE
- One FIFO in each direction: transmit and receive
- Microengine can instruct FIFO to receive packet via IX
- Once packet is in FIFO, microengine can have it moved to memory
  - Same for other direction
- FIFO really is RFIFO (random access FIFO 😊)
  - Each slot in FIFO can be accessed at any time
- IXP FIFOs:
  - Each FIFO contains 16 slots with 10 quadwords (=80 bytes)
- MAC hardware can divide packets to fit into slots
FBI Unit

- Command bus for communication with uEs
- Push and pull engine operate independently and move data to/from transfer register and FIFOS
Scratchpad Memory

- FBI Unit controls on-chip scratchpad memory
- Scratchpad memory:
  - 1K words (= 4kB)
- Scratchpad supports two functions:
  - Test and set operation
  - Autoincrement operation
Hash Unit

- ALU in uE does not support multiplication or division
  - Is used for protocol processing for hashing
- Hashing unit provides hardware implementation of hash function
- FBI unit handles access to hash unit
  - uE can request 1-3 hash operations in single instruction
  - 1-3 data values are stored by uE in consecutive SRAM tx regs
Hash Function

- Hash computes: $A(x) \times M(X) / G(x) \Rightarrow Q(x) + R(x)$
  - $A(x)$: input value
  - $M(x)$: hash multiplier – can be set in CSRs in FBI
  - $G(x)$: built-in value, depends on hash length (only two choices)
  - $Q(x)$: quotient
  - $R(x)$: remainder – result of hash computation
- Binary input can be seen as polynomial
- Hash can be 48 bit or 64 bit:
  - $G(x) = 100100200040116 = x^{48}+x^{36}+x^{25}+x^{10}+1$ (48 bit)
  - $G(x) = 1004000080002000116 = x^{64}+x^{54}+x^{35}+x^{17}+1$ (64 bit)
Hash Example

• Example values:
  – A = 800000000001_{16}
  – G = 1001002000401_{16}
  – M = 20D_{16}

• Hash is remainder:
  – H(A) = R = A * M \mod G
  – A * M = x^{56}+x^{50}+x^{49}+x^{47}+x^9+x^3+x^2+1
  – A * M = Q * G + R with Q(x) = x^8 + x^2 + x^1
  – H(A) = R = 90620C041B0B_{16}
StrongArm and uE Summary

Diagram showing the architecture of a StrongArm and uE system, including connections between SRAM, FLASH, Memory Mapped I/O, SDRAM, IXP1200 chip, PCI bus, and serial line.
• What kind of software abstractions are used on IXP?

• **Active Computing Element (ACE):**
  – Fundamental software building block
  – Used to construct packet processing system
  – Runs on StrongARM, uE, host
  – Handles control plane and fast or slow path packet processing
  – Coordinates and synchronizes with other ACEs
  – Can have multiple outputs
  – Can serve as part of pipeline

• Protocol processing is implemented by combining multiple ACEs
ACE Terminology

• Library ACE:
  – ACE that has been provided by Intel for basic functions

• Conventional ACE or Standard ACE:
  – ACE build by customer
  – Might make use of Intel’s Action Service Libraries

• Micro ACE
  – ACE with two components:
    • Core component (runs on StrongARM)
    • Microblock component (runs on uE)

• Terminology for mikrobloks:
  – Source microblock: initial point that receives packets
  – Transform microblock: intermediate point that accepts and forwards packets
  – Sink microblock: last point that sends packets
ACE Parts

• An ACE contains four conceptual parts:
  • Initialization:
    – Initialization of data structures and variables before code execution
  • Classification:
    – ACE classifies packet on arrival
    – Classification can be chosen or use default
  • Actions:
    – Based on classification an action is invoked
  • Message and event management:
    – ACE can generate or handle messages
    – Communication with another ACE or hardware
ACE Binding

- ACE can be bound together to implement protocol processing:

- Binding happens when loading ACE into NP
- Binding can be changed dynamically
- Unbound targets perform silent discard
ACE Division

Stack ACE

ingress ACE (core)

IP ACE (core)

egress ACE (core)

StrongARM microengine

input ports

ingress ACE (microblock)

IP ACE (microblock)

egress ACE (microblock)

output ports
Next Class

• More on ACE
  – How to assign components to microengines
  – Dispatch loops, packet queues

• SDK
  – Hopefully a demo

• Question:
  – Tuesday 11/11 is Veterans Day
  – Class for 12/12 needs to be moved