MLX1
A Tiny Multithreaded 586 Core for Smart Mobile Devices

Peter Song
peter.song@memorylogix.com

MemoryLogix
# Mobile x86 core sizes are ~10X ARM10 core

<table>
<thead>
<tr>
<th>Mobile Processors (0.13µm)</th>
<th>Total Cache (KB)</th>
<th>Die Size (mm²)</th>
<th>Est. Core Size (mm²)</th>
<th>Core Size Ratio</th>
<th>Typical Speed (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel ULV PIII-M</td>
<td>544</td>
<td>80</td>
<td>~34</td>
<td>~13</td>
<td>&gt;800</td>
</tr>
<tr>
<td>AMD Duron*</td>
<td>192</td>
<td>55</td>
<td>~37</td>
<td>~14</td>
<td>&gt;800</td>
</tr>
<tr>
<td>TM58000</td>
<td>640</td>
<td>55</td>
<td>~25</td>
<td>~10</td>
<td>&gt;800</td>
</tr>
<tr>
<td>VIA C3</td>
<td>192</td>
<td>52</td>
<td>~31</td>
<td>~12</td>
<td>&gt;800</td>
</tr>
<tr>
<td>ARM 1026EJ-S</td>
<td>32</td>
<td>4.6</td>
<td>2.6</td>
<td>1</td>
<td>&gt;400</td>
</tr>
</tbody>
</table>

ARM 1026EJ-S has DSP and Jazelle, and is synthesizable.

* Duron’s sizes are normalized from 0.18µm.
Why are current x86 cores so large?

• **X86 designed for peak frequency, performance**
  – Large multi-level caches, TLBs (>30-50% of die)
  – Superscalar, speculative execution
  – Branch prediction tables, trace caches
  – Big transistors, replicated logic for speed

• **Many x86 features not in ARM architecture (yet)**
  – FPU, MMX, SSE (20-30% of core)
  – System features (I/O, tasks, SMM, Machine-check, MP, …)
  – Many (>100) model-specific registers (started w/ P6)
  – Variable-length instruction decode (1-16 bytes)
  – Segmentation
MLX1 – Tiny Multithreaded 586 Core for Smart Mobile Devices

• Smart mobile device requirements
  – Balanced communication + entertainment + computing performance
  – High MIPS / Watt
  – Low cost

• MLX1 design goals
  – A “synthesis-friendly” x86 core
  – Support FPU, MMX
  – Support variable size cache and TLB using 1-port SRAMs
  – 2.5X the size of ARM10 core (<1/4 size of mobile x86 cores)
  – 2.5X system performance of single-threaded core / MHz
MLX1 Design Strategy

- **Simple and small**
  - Scalar RISC pipeline
  - Optimized for frequently used instructions
  - Features to map efficiently x86 & Java instructions
  - RISC-style microcode to map complex instructions

- **High speed**
  - No pipeline stall between queued stages
  - 3-cycle access to large L1 cache

- **High performance by high utilization**
  - SMT (simultaneous multithreading) w/ little area overhead
  - Threads share a register file
  - Threads share a non-blocking, 8-bank unified cache
Typical x86 Instruction Set Usage

- **On mostly integer applications**
  - 92% are RISC-like
  - Only 2% are ALU w/ memory operand
  - 80% of all branches use 8-bit offset
  - 35% of loads are relative to stack pointer

- **Frequent register copy**
  - MOV reg,reg 13%
  - MOVQ mmx,mmx varies

- **Float/MMX apps may use more loads and stores**

- **Statistics from Win95 applications**
  - Old but still good
  - Newer optimizations are more RISC-like, using CMOVs and fewer branches

<table>
<thead>
<tr>
<th>Instruction</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXE reg,imm</td>
<td>30%</td>
</tr>
<tr>
<td>EXE reg,reg</td>
<td>20%</td>
</tr>
<tr>
<td>MOV reg,reg</td>
<td>13%</td>
</tr>
<tr>
<td>LOAD reg</td>
<td>10%</td>
</tr>
<tr>
<td>JMP/JCC rel8</td>
<td>10%</td>
</tr>
<tr>
<td>PUSH/POP</td>
<td>5%</td>
</tr>
<tr>
<td>STORE reg</td>
<td>2%</td>
</tr>
<tr>
<td>CALL/RET</td>
<td>2%</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>92%</strong></td>
</tr>
</tbody>
</table>
MLX1’s Multi-fetch, Scalar-execute Pipeline

X86-to-RISC decode

inst. buffer fetch

cache read

translate cache hit

branch target prefetch: 2,3

X86-Flavored RISC

rename IQs, issue

operand read

execute

result cache RF write

load-use: 3

cache read

store queue

load align

cache write

translation

cache hit

cache read

load

align

cache write

branch cond: 3

completion
MLX1’s Multi-fetch, Scalar-execute Pipeline
MLX1’s Multi-fetch, Scalar-execute Pipeline

X86-Flavored RISC

translate
cache hit

rename
IQs, issue

translate
cache hit

cache read

operand
read

cache read

inst. buffer
fetch

X86-to-RISC
decode

executing
load-use : 3

branch target
fetch : 2,3

branch cond : 3

execute

result cache
RF write

store queue

load
align

load-cache

completion
MLX1’s Multi-fetch, Scalar-execute Pipeline

- Translate
  - Cache hit
- Cache read
- Inst. buffer fetch
- X86-to-RISC decode
- Branch target prefetch: 2, 3

- Rename
  - IQs, issue
- Operand read
- Execute
- Result cache RF write

- Branch cond: 3

- Translate
  - Cache hit
- Cache read
- Load align
- Cache write
- Store queue
- Load-use: 3

- Completion

X86-Flavored RISC
Multi-fetch Using 3 Identical Fetch Units

- Each fetch unit
  - Operates independently
  - Holds five 8-byte blocks
  - Prefetches up to 3 blocks from sequential path
  - Prefetches 2 blocks from target path as condition is evaluated

- Cache-location aware logic
  - Determines cache location of the next sequential 64B line
  - Remembers cache locations of two previous 64B lines
Threads Share a Single-instruction Decoder

- **ROM-based decode**
  - Easy to design & modify
  - Complexity independent
  - 512 words x D wide
  - 430MHz, 0.05 mm² (0.13μm) or
  - 700MHz, 0.1mm² fast SRAM
  - 1-cycle size decode is speed critical

- **Thread switch can occur**
  - Thread’s decode buffer not full
  - Thread’s issue queue full
  - After a branch, 4-cycle load, …
  - After a serialization instruction
  - After 8\textsuperscript{th} consecutive decode
Threads Share Rename Registers

• **Pointer-based register renaming**
  – 3 threads share a queue of 64 rename registers (8 GPRs + EIP)
  – Rename registers hold future, architectural, temporary values
  – Thread has own future register map, updated at rename
  – Thread has own arch. register map, updated at completion
  – On exception, future map restored from architectural map

• **A few-bit reference counter per rename register**
  – Enables register copy w/o copying values
  – Example: MOV R1, R2 → copy R2 pointer to R1 map
  – Increment count on reference
  – Decrement count on dereference
  – Supports fast & efficient inter-thread register copy
 Threads Share a Unified Cache

• **3-cycle access**
  - 1: Tag access for hit check
  - 2: Data access when needed
  - 3: Data alignment & routing
  - Use prefetch to reduce penalty
  - Supports large L1 cache

• **Up to 8 accesses / cycle**
  - 8 banks / line, 8 bytes / bank
  - Support multiple instruction fetch

• **N-way set-associative**
  - N can be any integer
  - True LRU replacement
  - Set partitioning and locking
  - Reduces multithread conflict
One-cycle Load From Stack

- 35% of loads are relative to stack or base pointers
  - POP reg
  - MOV reg, [ESP/EBP,im8] ; “C” local variables & arguments

- Cache-location aware logic for top 256 bytes on stack
- Cuts load-use penalty from 3 to 0 cycle
- Supports Java hardware acceleration w/o register stack

<table>
<thead>
<tr>
<th>ISSUE</th>
<th>pop</th>
<th>add</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td></td>
<td>add</td>
</tr>
<tr>
<td>EXE</td>
<td></td>
<td>add</td>
</tr>
<tr>
<td>$ HIT</td>
<td></td>
<td>pop</td>
</tr>
<tr>
<td>$ RD</td>
<td>pop</td>
<td>pop</td>
</tr>
<tr>
<td>ALIGN</td>
<td></td>
<td>pop</td>
</tr>
</tbody>
</table>
# MLX1 Targets System-on-Chip Solutions

>400 MHz synthesized, 0.13μm

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SMT 386</td>
<td>3.5 mm²</td>
</tr>
<tr>
<td>MMX</td>
<td>1.0 mm²</td>
</tr>
<tr>
<td>FPU</td>
<td>1.5 mm²</td>
</tr>
<tr>
<td>MLX1</td>
<td>6.0 mm²</td>
</tr>
</tbody>
</table>

**Ultra-low power premium devices:**
1G DRAM : 95%
MLX1 + $ : 5%
(0.09 μm)

- **Embedded PC:**
  MLX1 + $ + PC peripherals

- **Mobile devices:**
  MLX1 + $ + mobile peripherals

- **Mobile Xbox:**
  \(N \times MLX1 + $ + 3D\) graphics + game peripherals
MLX1 Leverages from X86 PC Platform

- **PC platform has the best industry support**
  - Supported by 100’s of companies and $Billions every year
  - Most number of low-cost peripherals
  - Most amount of application code and device drivers
  - Best software development tools
  - Most number of programmers

- **PC platform will expand**
  - To higher performance, more functionality
  - To lower cost & power consumption, smaller form factors

- **PC platform will enable successful Post-PC devices**
  - Simpler, smaller, cheaper, lower-power PC in user-friendly shells
Summary

• An x86 core can be made as small as an ARM core
  – More complexity does not necessarily require larger die area
  – Design as small scalar core + FPU + MMX
  – ARM also has integer + Thumb + DSP + Jazelle + VFP + SIMD
  – A core will occupy increasingly smaller portion of an SoC

• MLX1
  – A tiny “synthesis-friendly” 586 core for SoC solutions
  – For smart mobile devices that demand high MIPS / W
  – Uses SMT to deliver more performance in smaller die area
  – Leverages from the PC platform

• Contact us for more detail