Jazelle ARM

By: Adrian Cretzu & Sabine Loebner
Table of Contents

• Java
  o Challenge
  o Acceleration Techniques
• ARM Overview
  o RISC
  o ISA
  o Background
• Jazelle
  o Background
  o Jazelle mode
  o bytecode execution
  o Jazelle Interface
  o Future
The Java Challenge

- Java feature: Write Once Run Anywhere Model
  - ideal mobile application development
- Getting Java on portable devices was very difficult.
- The Challenge: How can Java be integrated and accelerated to execute sophisticated applications, yet still be withing the cost, power, memory, and space constraints?
Java Acceleration Techniques

- Software Acceleration
  - Accelerated Java execution, but unsuitable for portable or low-cost systems.
  - Examples: Optimize Java Virtual Machine (JVM) and compilers.
- Hardware Acceleration
  - Requires additional space + power. Don't maximize speed.
  - Examples: Dedicated Java Processor, Java co-processor.
ARM & Java Acceleration

- Increasing demand from ARM customers for better Java performance.
- ARM provided its own solution in executing Java in hardware.
  - Integrate Java execution into the core!
  - Birth of Jazelle!

![Jazelle Logo](image)
Architectural Extensions

- Architectural Extension: Combination of hardware + software solution.
  - Enables performance increase within cost and power requirements.
  - Reuses all existing processor resources without the need to re-engineer the existing architecture.
ARM & RISC

• Advanced RISC Machine (ARM): a 32-bit RISC ISA commonly used in mobile and embedded electronics.
• ARM offers a useful combination of...
  o high performance
  o low power
  o low system cost.

• RISC: Reduced Instruction Set Computing: a CPU design strategy
  o Simplifies instructions as much as possible.
  o Provides higher performance and faster execution of each instruction.
  o Requires more complex coding
ARM ISA

• Most ARM's implement two instruction sets:
  o 32-bit ARM Instruction Set: all instructions are 32-bits long.
  o 16-bit Thumb Instruction Set: compressed most-commonly
    used instructions in 16-bit format.

• ARM has 37 32-bit registers
  o 1 dedicated PC
  o 1 dedicated current program status register (CPSR)
  o 5 dedicated saved program status registers (SPRS)
  o 30 general purpose registers
Status Register Design

**Condition code flags**
- N = Negative result from ALU
- Z = Zero result from ALU
- C = ALU operation Carried out
- V = ALU operation Overflowed

**Interrupt Disable bits.**
- I = 1: Disables the IRQ.
- F = 1: Disables the FIQ.

**T Bit**
- Architecture xT only
- T = 0: Processor in ARM state
- T = 1: Processor in Thumb state

**Mode bits**
- Specify the processor mode

**J bit**
- Architecture 5TEJ only
- J = 1: Processor in Jazelle state

**Sticky Overflow flag - Q flag**
- Architecture 5TE/J only
- Indicates if saturation has occurred
ARM Organization

Cortex M3:
ARM ISA Continued...

- To create Jazelle, a third instruction set that takes Java byte code was required.
- Enables new state where processor behaves like a Java machine
- Maintains ability to switch between ARM/Thumb state and Java state.

Figure 1: Jazelle Technology
Java in Mobile Devices

- **Java feature:** Write Once Run Anywhere Model
  - ideal mobile application development
- **Problems:** handle byte code ISA
  - implemented physically
    - extra power + space required
  - translated by target device in Software
    - inefficient + slow
- **Solution:** built-in JVM in mobile device
  - challenge: integrate + accelerate Java
    - optimal system performance
    - minimize space, cost, power consumption
How to execute Java bytecode?

1) direct interpretation
   -> bytecode translated directly into machine code
   -> pro: memory efficient
   -> con: low performance

2) JIT (Just in time) compiler
   -> compiled, profiled into most executed instructions -> translated to native ARM code
   -> con: compiler requires extra power, memory resources slow start-up, application pauses due to profiling

3) AOT (Ahead of time) compiler
   -> compiled while downloading/ installation
   -> pro: performance
   -> con: large memory requirement

4) Jazelle
   -> combined Hardware/ Software solution interpreting bytecode
   -> high quality, low memory + power consumption, fast application
Enter Jazelle mode

- fetch + decode in separate pipeline stages
  - first fetch, then decide which mode to execute
- CPSR (Current Program Status Status Register)
  - keeps track of current status
  - 2-bit: J, T
    
    J  T  ISA
    
    ---------
    0  0  ARM
    0  1  Thumb
    1  0  Jazelle
    1  1  ThumbEE
- BXJ (branch to Java) instruction
  - checks CPSR -> branch to Jazelle if allowed, otherwise executed as regular branch
  - set J bit in CPSR
Java bytecode - Jazelle mode

1) directly executed in Hardware
   - 95% of all bytecode instructions
   - common, simple JVM instructions
   - reduces number of interpretations
   - reduce need for JIT compilation

2) interpreted in Software
   - JVM instructions not implemented in Hardware
   - interpreted into short sequence of optimized ARM instructions
   - reduces complexity, need for additional logic

3) undefined bytecode
   - leave Jazelle mode, throw exception handled in processor
Jazelle Interface

•Interrupts
  o CPSR keeps track of current status -> used to return to restore state after interrupt and exception handling

•Register assignment
  o hold state within Jazelle JVM
    ▪ all states held within ARM registers -> easily compatible with other existing OS's
  o r0 - r3: first 3 top of stack elements
    ▪ keep logic simple, less additional gates needed
  o r4: local zero operand (pointer to this)
  o r6: Java Stack pointer
  o r14: pointer to next bytecode instruction
    ▪ has to be existing when executing BXJ
  o r15: program counter
Facts/Features

• First introduced 2002
• ARM processors currently supported:
  o ARM926EJ-S (first one supported)
  o ARM1176JZ(F)-S
  o ARM1136J(F)-S
  o ARM1026EJ-S
  o ARM7EJ-S
• 8x less RAM compared to Software only approach
• 4x performance increase compared to commercial JVM
• Start-up timing comparison
Where do we go from here?

- continued performance improvement
  - double performance, minimized memory usage Ver. 2 to 3
- Jazelle MobileVM
  - Multitasking Virtual Machine
  - higher performance, less memory
  - new features:
    - runtime bytecode optimizer
    - on-target debugging
- ThumbEE (or Jazelle-RCT)
  - introduced 2005, Cortex-A8 processor
  - languages: Limbo, Java, C#, Perl, Python
  - code compiled to smaller, simpler pieces
  - new features:
    - automatic null pointer check load + store
    - array bound check
**JTEK 2.x**
- Replaces bytecode interpreter:
- "driver" software
- Simple integration:
- 1-2 day
- Performance boost:
- 4-10x

**JTEK 3.x**
- Optimize JVM parts:
- Method invocation, Garbage collector etc.
- Higher performance:
- >2x JTEK 2.x
- Efficient memory usage:
- 8x less than DAC/UIT

**JTEK 4.x**
- New JVM architecture:
- Multi-tasking
- Configurable API

**Jazelle® MobileVM**
- Debug and profiling
- JRO bytecode optimizer

Version 1.0
Available 2H07
Questions?