Towards Very High Bandwidth Wireless Handheld Devices

John Glossner\textsuperscript{1}, David Routenberg\textsuperscript{1}, Erdem Hokenek\textsuperscript{1}, Mayan Moudgill\textsuperscript{1}, Michael Schulte\textsuperscript{2}, Pablo Balzola\textsuperscript{2}, and Stamatis Vassiliadis\textsuperscript{3}

\textsuperscript{1}Sandbridge Technologies, White Plains, NY
\textsuperscript{2}Lehigh University, Bethlehem, PA
\textsuperscript{3}Delft University of Technology, Delft, Netherlands

glossner@SandbridgeTech.com
Introduction

Market
- DSP
- Wireless

Application Domain
- Broadband Communications
- Performance Requirements
- Compute + Control

Low Power Techniques
- Saturating Multi-operand Arithmetic

Programmability
- DSP Compilation

Java execution
- Hardware vs. Software
Market Outlook
Programmable DSP Market

- **CAGR 34.4%**
- **Growing faster than the general semiconductor market**

Source: Forward Concepts
Wireless Handset Market

- 416M units shipped in 2000
- 1B per year estimated by 2005!
- 70% using traditional DSPs
  - TI – 83% (242M units)
  - MOT – 10%
  - ADI/LU – 7%
- 30% using DSP core ASICs
Application Domain
Datarate Evolution

- Voice Modems
- Data Modems
- Wireless
- Ethernet
- Fiber - TDM
- Fiber - WDM
Performance vs. Power

DSP Performance vs. Power
(Log Log scale)

MMAC/s vs. mWatts

100 GMAC/sec

10M/mW

5M/mW

1M/mW

50M/mW

10000

1000

100

10

mWatts

SANDBRIDGE
Reducing Power

- Batteries are not improving dramatically
- Typical high-end SOC’s dissipate 1W
  - 1-10mW required for handset
- Attention must be paid at every step of the design cycle
  - Algorithms, OS, Software, Architecture, Microarchitecture, Logic Design, Circuits, Process
  - Any one particular area does not yield 100x
A 3G convergence device

- **A convergence device**
  - cell phone + pda

- **Java Applications**
  - Dynamic applets
  - Web browsing
  - Email/calendar/PIM

- **Network Processing**
  - Firewall
  - Web server

- **Operating System**
  - Linux / PocketPC / PalmOS for Applications
  - RTOS for Signal Processing

- **DSP**
  - **High-speed connectivity**
    - Multiple reconfigurable protocols
      - 2.5 / 3G (GPRS, CDMA-2000, WCDMA)
      - Bluetooth
      - 802.11
    - Constant connection

  - E911
  - GPS
  - MPEG Video
  - Speech recognition (?)
Processor Classification

Processor

DSP

General Purpose
Processor Classification

Processor

DSP

Floating Point

32 bit IEEE

Other

General Purpose

Floating Point

32/64 bit IEEE

Other (80 bit)
Processor Classification

Processor

DSP

Fixed Point
- 16 bit
- 20 bit
- 24 bit
- 32 bit
- 32 bit IEEE

Floating Point
- 32 bit + subsets
- 64 bit + subsets

General Purpose

Integer
- 32 bit + subsets
- 64 bit + subsets
- 32/64 bit IEEE

Floating Point
- Other (80 bit)
Numeric Representations

**Fixed Pt. Fractional**
- Sign
- Radix Point

**Integer 2's Complement**
- Sign

**Floating Point**
- Sign
- Mantissa
- Exponent
- Implied mantissa (always 1)

- $2^0$  $2^{-1}$  $2^{-2}$  $2^{-3}$  $2^{-4}$  $2^{-5}$  $2^{-6}$  $2^{-7}$
- $2^0$  $2^1$  $2^2$  $2^3$  $2^4$  $2^5$
- $2^0$  $2^{-1}$  $2^{-2}$  $2^{-3}$  $2^{-4}$  $2^{-5}$

- $-2^0 + 2^{-2} + 2^{-4} + 2^{-5} = 0.65625$
- $-2^7 + 2^5 + 2^3 + 2^2 = 1.625$

- $-1 + .25 + .0625 + .03125 = -0.65625$
- $-128 + 32 + 8 + 4 = -84$
- $1 + .5 + .125 = 1.625$

- $1.625 \times 2^5 = 52.0$

Multiplication complicates fractional representations

Source: BDTI
DSP vs. General Purpose

**Execution Predictability**
- Required to guarantee real-time constraints

1 cycle MAC
0-overhead Loop Buffer
Complex Instructions
- Multiple Operations Issued
Harvard Memory Architecture
- Multiple memory access
Specialized Addressing Modes
Operate on Vector Stream Data
Data-independent Execution
Fractional Arithmetic
Pipeline Non-interlocked
- Shallow Pipeline (3-5 stage)
Delayed Branch

**Fast But Non-predictable**
- Dynamic Instruction Issue
- Non-deterministic caches
Multicycle MAC
Branch Prediction
RISC Superscalar Instructions
- Multiple Instructions Issued
Von Neumann Architecture
- Split Cache has similar benefit
Typically Linear Addressing
Caches Assume Locality
Data-dependent Execution
- Dependent upon operands
Integer Arithmetic
Pipeline Typically Interlocked
- Deep Pipeline (5+ stage)
Multicycle Branch
Modern DSP Architectures

- Focus on compilability
- RISC based with Control + DSP processing
- Highly Parallel
- Multiple instruction issue
- Multiple operation issue
  - MAC
  - ALU
  - Load/Store
- Predominately VLIW
  - Some use of SIMD
- 32-bit unified address space
Parallel Saturating Arithmetic
Motivation

- **GSM (Global System for Mobile Communications)**
  - Leading wireless digital technology in the world

- Extensive use of saturating dot products on vectors

```c
for (j = 0; j < n; j++)
    sum = L_mac(sum, x[j], y[j]);
```

- For GSM compliance, results produced must exactly match serial results

- Saturating arithmetic operations are not associative
  - severely limits parallelism

- **Goal:** Develop techniques for performing parallel saturating dot products with bit exact results
Parallel Saturating MACs

Our approach:
- Parallel saturating multiply operations
- m-input saturating multioperand addition
- A k element dot product requires $1 + \lceil k/m \rceil$ cycles

$Z_5 = \ll\ll\ll P_1 + P_2, P_3, P_4, P_5 >$

$P_{i+1} = < X_i Y_i >$
Saturating Addition

With two’s complement addition, overflow only occurs if $P_1$ and $P_2$ have the same sign, and the sign of their sum is different.

\[
P_1 = 0.101 = 0.625 \quad P_1 = 1.011 = -0.625
\]
\[
+ P_2 = 0.111 = 0.875 \quad + P_2 = 1.001 = -0.875
\]
\[
= T_1 = 1.100 = -0.500 \quad = T_1 = 0.100 = 0.500
\]
\[
Z_2 = 0.111 = 0.875 \quad Z_2 = 1.000 = -1.000
\]

Overflow can be detected as

\[
o_1 = sp_1 \, sp_2 \, st_1 + sp_1 \, sp_2 \, st_1
\]

If overflow occurs, the saturated result is

\[
V_2 = sp_2 \, sp_2 \, sp_2 \ldots \, sp_2 \, sp_2
\]
Saturating 2-MAC Results

16-bit designs for sat. and no-sat. units were implemented using the Synopsys Module Compiler with a 0.25 um CMOS std. cell lib.

Delay/area estimates and increases due to adding saturation logic are shown in the following table:

<table>
<thead>
<tr>
<th>Unit</th>
<th>Saturation</th>
<th>No Saturation</th>
<th>Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Delay</td>
<td>Area</td>
<td>Delay</td>
</tr>
<tr>
<td>Adder</td>
<td>1.1</td>
<td>2373</td>
<td>1.0</td>
</tr>
<tr>
<td>Multiplier</td>
<td>6.6</td>
<td>16695</td>
<td>6.6</td>
</tr>
<tr>
<td>MAC</td>
<td>7.5</td>
<td>18567</td>
<td>7.2</td>
</tr>
<tr>
<td>Dual MAC</td>
<td>8.1</td>
<td>42982</td>
<td>7.2</td>
</tr>
</tbody>
</table>
Area Comparison

![Area Comparison Chart](chart.png)

- **Area (equivalent gates)** vs **Number of MACs**
- Comparison between **Serial** and **Parallel** architectures

- **X-axis**: Number of MACs (1 to 8)
- **Y-axis**: Area (equivalent gates)
Delay Comparison

![Bar chart showing comparison between serial and parallel delay with respect to the number of MACs. The x-axis represents the number of MACs (1 to 8), and the y-axis represents delay (in ns). The chart compares serial and parallel methods, with bars in blue for serial and red for parallel.](image)
Programmability
10x Complexity every 10 years
Compiler Productivity

1. Design Algorithms
2. Map to Fixed Point C
3. Write DSP Specific C
4. Write DSP Assembly
5. Hand Schedule Operations on DSP
6. Final Product

6-9 Months!
Compiler Productivity

1. Design Algorithms
2. Map to Fixed Point C
3. Write DSP Specific C
4. Write DSP Assembly
5. Hand Schedule Operations on DSP
6. Compile
7. If floating point implemented
   - 6-9 Months!

Final Product

6-9 Months!
Compilable Architecture

- **Compile Architecture**
- **Optimize**
  - Cost / Power
  - Performance
- **Compiler**
- **Architecture**
- **Implementations**
  - GSM
  - DSL
  - VoIP
  - 3G

**Algorithms**
DSP Compilation Problem

Mismatch between C & DSP
- 16-bit fixed point
- 40-bit accumulators with mixed type arithmetic
- Saturation arithmetic vs. modulo semantics

Historically...
- DSPs have had compiler unfriendly architectures
  - very complex instructions
  - non-orthogonal, specialized resources
  - exposed pipelines
- DSP compiled performance
  - Typical: 1/10 speed of handwritten assembly
  - Assembly code is required for performance
DSP Compilation Solutions

- **Extensive libraries**
  - Often more than 1000 functions
  - Resource consuming but high reuse

- **C language extensions (DSP-C)**
  - Type support (Q15)
  - Memory disambiguation

- **Intrinsics**

- **Handwritten assembly code**
DSP Intrinsics

- Intrinsics allow programmers to use instructions a compiler can not generate
- Has appearance of a function call in C
  - Replaced with assembly statements by compiler
  - Highly architecture dependent
- Often condense 10 assembly instructions into 1
- Early attempts were blocking
  - Inlined asm statement
- Non-blocking pioneered by Lucent
  - Written in the compiler’s intermediate language
  - Semantics of side effects well defined
  - Allowed for further optimization
  - Architecturally neutral
**DSP Compilation Solution**

- **Intrinsics work well but...**
  - Compiler writers become DSP assembly language programmers
  - Only work for a specific application

- **DSP Solution: Semantic Analysis**
  - Type inference
  - no intrinsics: out-of-the-box C compiler
  - near-parity with assembly code
  - novel DSP optimizations
  - existing optimizations adapted for DSPs
  - power-driven optimizations
Java Execution
Java & 3G

Java may be required for all 3G devices

- NTT DoCoMo issued statement requiring use of Java

Java may execute applications but...

- Is Java an efficient Signal Processing Language?
- Is special hardware required?
Java Properties

- **Object-Oriented Programming Language**
  - Inheritance and Polymorphism Supported

- **Programmer Supplied Parallelism (Threads)**

- **Dynamically Linked**
  - Resolved C++’s fragile class problem but imposes performance constraints on class access
  - Entire set of objects in system not required at compile time

- **Strongly Typed**
  - Statically determinable type state enables simple on-the-fly translation of bytecodes into efficient machine code
  - [Gos95]

- Garbage collected

- Compiled to Platform Independent Virtual Machine
Methods of Executing Java

- **Interpretation**
  - 1-10% performance of compiled code

- **JIT Compilation**
  - 5-10x better than interpretation

- **Flash Compilation**
  - 10x better than JIT

- **Off-line compilers**
  - 10x better than interpretation found in Toba

- **Native Compilers**
  - Should provide near parity

- **Direct Execution**
  - 5-50x better than JIT
Java Performance

Can Java match C performance...
Experiments

**Traditional FFT**
- C algorithm from [Press92]
- Multiple Sizes

**Rewrote in Java**
- Does not use Java-specific capabilities

**Comparison**
- gcc 2.7.2
- Matlab built-in
- Java Interpreted
- Java JIT
- Toba off-line compiler

**Other Experiments included**
- Tensor FFT optimized for MT/MP Applications
- Coded in Matlab and Java
### Traditional FFT Rel. Results

<table>
<thead>
<tr>
<th></th>
<th>F4</th>
<th>F16</th>
<th>F64</th>
<th>F256</th>
<th>F1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc -O3</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Solaris 2.6 w/JIT</td>
<td>1.2</td>
<td>1.6</td>
<td>2.2</td>
<td>2.6</td>
<td>2.8</td>
</tr>
<tr>
<td>Toba 1.0b6 -O3</td>
<td>1.3</td>
<td>1.8</td>
<td>2.5</td>
<td>4.0</td>
<td>3.2</td>
</tr>
<tr>
<td>gcc unopt</td>
<td>1.7</td>
<td>2.5</td>
<td>3.6</td>
<td>4.2</td>
<td>2.5</td>
</tr>
<tr>
<td>Kaffe 0.9.2</td>
<td>2.1</td>
<td>3.3</td>
<td>5.0</td>
<td>5.8</td>
<td>6.3</td>
</tr>
<tr>
<td>Matlab built-in</td>
<td>6.7</td>
<td>4.9</td>
<td>4.9</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>Java 1.1.4 Interp</td>
<td>7.2</td>
<td>15.0</td>
<td>25.0</td>
<td>30.0</td>
<td>34.0</td>
</tr>
</tbody>
</table>
Java FFT Results

Java May Offer Sufficient FFT Performance
- Small FFT’s Are Within 20% to 60% of C performance
- Larger FFT’s Are 2-3x Less Efficient
- Object Oriented FFT’s Are Much Less Efficient (>>30x)
- Better Java Compiler Technology may decrease this gap

JIT Compilers Offer 6-12x speed-up on C-like Java
- Traditional FFT speedup of 6-12x over Interpreted Java
Java Hardware

Does Java require special hardware acceleration?
Delft-Java Engine

- **RISC-style Architecture**
  - 32-bit Instructions
  - Multiple Register Files

- **Concurrent Multithreaded Organization**
  - Multiple Hwdr Thread Units
  - Multiple Instruction Issue Per Thread

- **Indirect Register Access**

- **Supervisory Instructions**
  - Branch Java View (bex)

- **Integer & Floating Point**
  - 8, 16, 32, and 64-bit Signed & Unsigned Integers
  - IEEE-754 Floating Point

- **Multimedia Instructions**
  - SIMD Parallelism

- **DSP Arithmetic Extensions**
  - Saturation Logic
  - Rounding Modes

- **32-bit Address Space**
  - Base + Offset + Displacement
Delft-Java Organization
Java Hardware Support

- **Transparent Extraction of Parallelism**
  - Multiple concurrent thread units

- **Dynamic Java Instruction Translation**
  - Register file caches stack with indirect access

- **JVM Reserved Instruction Used For BEX**

- **Link Translation Buffer For Dynamic Linking**
  - Associates a caller’s object reference and constant pool entry ID with a linked object invocation

- **Logical Controller For Non-Supported Translations**
  - Thin interpretive layer and Java run-time
Java H/W Execution

- **Single-threaded Dynamic Translation**
  - A form of hardware register allocation
  - Transform stack bottlenecks into pipeline dependencies
  - Pipeline dependencies are removed using superscalar techniques

- 3.2x speedup achieved over a pipelined stack model

- Up to 60% of stack bottlenecks removed

- For translated instruction streams, out-of-order execution realized a 50% performance improvement when compared with in-order execution
Conclusions

- Power control is a major enabler of convergence devices
  - Special purpose low power hardware techniques required
  - Attention to implementation details a must

- DSP Applications are becoming more complex and will be written in C
  - Highly optimizing compilers will be required

- Java may become the predominant programming platform for 3G wireless
  - Limited to applications code

- Embedded/DSP applications have distinct requirements from general purpose applications