Presentation Goals

By the end of this workshop, you should know:

- How the ARM11 differs from its predecessors
- How the ARM11’s signal processing performance compares to other GPPs’
- How the ARM11’s signal processing performance compares to DSPs’
- How to get the most out of the ARM11 in signal processing applications
## ARM Family Summary

<table>
<thead>
<tr>
<th></th>
<th>ARM7</th>
<th>ARM9</th>
<th>ARM9E</th>
<th>ARM11</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Max Clock</strong></td>
<td>133 MHz</td>
<td>220 MHz</td>
<td>220 MHz</td>
<td>350 MHz</td>
</tr>
<tr>
<td><strong>Die Area w/o cache</strong></td>
<td>0.32 mm²</td>
<td>1.7 mm² (estimated)</td>
<td>0.59 - 2.2 mm²</td>
<td>2.4 - 2.85 mm²</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>0.11 mW/MHz</td>
<td>0.25 mW/MHz</td>
<td>0.12 - 0.25 mW/MHz</td>
<td>0.45 - 0.8 mW/MHz</td>
</tr>
<tr>
<td><strong>Instruction Sets</strong></td>
<td>ARMv4, Thumb</td>
<td>ARMv4, Thumb</td>
<td>ARMv5E, Thumb</td>
<td>ARMv6, Thumb, Thumb-2**</td>
</tr>
<tr>
<td><strong>FPU</strong></td>
<td>No</td>
<td>No</td>
<td>VFPv2***</td>
<td>VFPv2</td>
</tr>
<tr>
<td><strong>Pipeline</strong></td>
<td>3 stages</td>
<td>5 stages</td>
<td>5 stages</td>
<td>8 stages</td>
</tr>
<tr>
<td><strong>Branch Prediction</strong></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* © 2005 Berkeley Design Technology, Inc.

** No cache

*** ARM946E-S and ARM966E-S only

** ARM1156T2(F)-S only

---

## ARM Family Summary

<table>
<thead>
<tr>
<th></th>
<th>ARM7</th>
<th>ARM9</th>
<th>ARM9E</th>
<th>ARM11</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Maximum Arithmetic Throughput</strong></td>
<td>1 x 32-bit</td>
<td>1 x 32-bit</td>
<td>1 x 32-bit</td>
<td>1 x 32-bit</td>
</tr>
<tr>
<td><strong>Multiplier Latency</strong></td>
<td>Data-dependent</td>
<td>Data-dependent</td>
<td>1 cycle</td>
<td>2+ cycles</td>
</tr>
<tr>
<td><strong>Memory System</strong></td>
<td>Von Neumann</td>
<td>Harvard</td>
<td>Harvard</td>
<td>Harvard</td>
</tr>
<tr>
<td><strong>Data Bus</strong></td>
<td>32-bit</td>
<td>32-bit</td>
<td>32-bit</td>
<td>64-bit</td>
</tr>
<tr>
<td><strong>Parallel Load/Store</strong></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Load Latency</strong></td>
<td>1 cycle</td>
<td>1 cycle</td>
<td>1 cycle</td>
<td>3 cycles</td>
</tr>
</tbody>
</table>

* © 2005 Berkeley Design Technology, Inc.

---

© 2005 Berkeley Design Technology, Inc.
Insider Insights on the ARM11’s Signal-Processing Capabilities

**Signal Processing Throughput**

- Higher is Faster

```
0 500 1000 1500 2000 2500
```

- ARM 220 MHz
- ARM1136 350 MHz
- Intel PXA27x 624 MHz
- TI 'C55x 300 MHz
- CEVA CEVA-X1620 200 MHz
- StarCore SC1400 185 MHz

- BDTIsimMark2000™
- BDTImark2000™

* TSMC CL013G/Artisan SAGE-X, worst-case conditions
** Fastest currently-available chips

– Higher is Faster

**Signal Processing Energy Efficiency**

- Higher is More Efficient

```
0 10 20 30 40 50 60 70 80 90
```

- ARM ARM926E-S
- ARM ARM1136j-S
- CEVA CEVA-X1620
- StarCore SC1400

- BDTIsimMark2000™/mW

* TSMC CL013G/Artisan SAGE-X, nominal conditions; power is for core only

© 2005 Berkeley Design Technology, Inc.
Insider Insights on the ARM11’s Signal-Processing Capabilities

### Signal Processing Memory Efficiency

![Bar Chart](Image)

#### BDTI Benchmarks™

- ARM ARM9E
- ARM ARM1136
- Intel PXA27x
- TI ‘C55x
- CEVA CEVA-X1620
- StarCore SC1400

Higher is More Efficient

© 2005 Berkeley Design Technology, Inc.

### Programming Tips

- Know how to use the compiler
  - Understand compiler behavior
  - Know the ARM11 architecture
- Know when to use the compiler—and when to write assembly code
- Make effective use of SIMD operations
  - Organize data for SIMD
  - Process multiple samples when possible
  - Consider all the available instructions
  - Sometimes SISD is better!

© 2005 Berkeley Design Technology, Inc.
Programming Tips

Memory access is costly
- Organize data flow to minimize cache misses
- Keep data in registers and re-use it
- Use large loads and stores

Use software pipelining to mask multiplier and load/store latencies

Register pressure is tight; use fewer registers
- CAUTION: Don’t increase memory accesses in the process!

Example: FIR Filter Kernel

C implementation of FIR kernel

\[ y[n] = \sum_{k=0}^{T-1} x[n-k]h[k] \]

\[
\begin{align*}
N &= 40; \\
T &= 16; \\
\text{for} \ (n=0; \ n<N; \ n++) \{} \\
\text{for} \ (k=0; \ SUM=0; \ k<T; \ k++) \{} \\
\text{SUM} &= x[n-k] * h[k]; \\
\} \\
y[n] &= \text{SUM}; \\
\end{align*}
\]
Insider Insights on the ARM11’s Signal-Processing Capabilities

### Analysis: Compiled FIR Filter

```
| L1.16 | CMP r6, #0
| MOV r3, #0
| MOV r12, #0
| BLE | L1.68 |
| L1.32 | SUB r4, r5, r3
| ADD r8, r1, r3, LSL #1
| ADD r4, r0, r4, LSL #1
| LDRH r8, [r8, #0]
| LDRH r4, [r4, #0]
| ADD r3, r3, #1
| CMP r3, r6
| SMLABB r12, r4, r5, r12
| BLT | L1.32 |
| L1.68 | ADD r3, r2, r5, LSL #1
| ADD r5, r5, #1
| CMP r5, r7
| STRH r12, [r3, #0]
| BLT | L1.16 |
```

3 branches  
2 instructions per branch

2 instructions per load

1 stall cycle

---

### Inner Loop Cycle Count

```
| L1.16 | CMP r6, #0
| MOV r3, #0
| MOV r12, #0
| BLE | L1.68 |
| L1.32 | SUB r4, r5, r3
| ADD r8, r1, r3, LSL #1
| ADD r4, r0, r4, LSL #1
| LDRH r8, [r8, #0]
| LDRH r4, [r4, #0]
| ADD r3, r3, #1
| CMP r3, r6
| SMLABB r12, r4, r8, r12
| BLT | L1.32 |
| L1.68 | ADD r3, r2, r5, LSL #1
| ADD r5, r5, #1
| CMP r5, r7
| STRH r12, [r3, #0]
| BLT | L1.16 |
```

```
{ 
  N=40; 
  T=16; 
  for (n=0; n<N; n++) {
    for (k=0; SUM=0; k<T; k++) {
      SUM += x[n-k] * h[k];
    }
    y[n] = SUM;
  }
}

10 cycles per tap = 0.10 taps per cycle
```
Give the Compiler a Hand

Human-friendly

```
N=40;
T=16;
for (n=0; n<N; n++) {
    for (k=0; SUM=0; k<T; k++) {
        SUM += x[n-k] * h[k];
    }
    y[n] = SUM;
}
```

Compiler-friendly

```
#define N 40
#define T 16
for (n=N; n; n--) {
    for (k=T; SUM=0; k--; k++) {
        SUM += x[n-k] * h[k];
    }
    y[n] = SUM;
}
```

Use constants instead of variables

Give the Compiler a Hand

Human-friendly

```
N=40;
T=16;
for (n=0; n<N; n++) {
    for (k=0; SUM=0; k<T; k++) {
        SUM += x[n-k] * h[k];
    }
    y[n] = SUM;
}
```

Compiler-friendly

```
#define N 40
#define T 16
for (n=N; n; n--) {
    for (k=T; SUM=0; k--; k++) {
        SUM += x[n-k] * h[k];
    }
    y[n] = SUM;
}
```

Count downwards in “for” loops
Insider Insights on the ARM11’s Signal-Processing Capabilities

Give the Compiler a Hand

**Human-friendly**

```c
N=40;
T=16;
for (n=0; n<N; n++) {
  for (k=0; SUM=0; k<T; k++) {
    SUM += x[n-k] * h[k];
  }
  y[n] = SUM;
}
```

**Compiler-friendly**

```c
#define N 40
#define T 16

for (n=N; n; n--) {
  short *xt = x++;
  short *ht = h;
  for (k=T, SUM=0; k; k--) {
    SUM += *xt-- * *ht++;
  }
  *y++ = SUM;
}
```

Make pointer increment explicit

Analysis: Compiled FIR Filter

```
| L1.8 | MOV  r3,r0
| ADD  r0,r0,#2
| MOV  r12,r1
| MOV  r4,#0x10
| MOV  r5,#0

| L1.28 | LDRH r6,[r3],#-2
| LDRH r7,[r12],#2
| SUBS r4,r4,#1
| SMLABB r5,r6,r7,r5
| BNE   |L1.28|
| SUBS r8,r8,#1
| STRH r5,[r2],#2
| BNE   |L1.8|
```

1 instruction per load
2 stall cycles
2 branches
1 instruction per branch

7 cycles per tap = 0.14 taps per cycle*

*Inner loop only
Adding SIMD: The Simple Approach

```
LOOP  LDRD  r6, [r0], #8
      LDRD  r10, [r1], #8
      SUBS  r2, r2, #4
      SMLAD r12, r6, r10, r12
      SMLAD r12, r7, r11, r12
      BGT   LOOP
```

2 stall cycles

```
LDRD  r4, [r0], #8
      LDRD  r8, [r1], #8
```

4 taps in 9 cycles = 0.44 taps per cycle

Software Pipelining

```
LOOP  LDRD  r6, [r0], #8
      LDRD  r10, [r1], #8

SMLAD r12, r4, r8, r12
      LDRGTD r4, [r0], #8
      SMLAD r12, r4, r8, r12
      LDRGTD r8, [r1], #8
      SMLAD r12, r7, r11, r12
      BGT   LOOP
```

pipelined second iteration

8 taps in 10 cycles = 0.80 taps per cycle

0 stall cycles
Insider Insights on the ARM11’s Signal-Processing Capabilities

Fully Optimized FIR Inner Loop

```
loop ldrd rx, [rx, #x]  
smuad rx, rx, rx  
smuad rx, rx, rx  
smlad rx, rx, rx, rx  
smlad rx, rx, rx, rx  
ldrd rx, [rx, #x]  
smuad rx, rx, rx  
smuad rx, rx, rx  
smlad rx, rx, rx, rx  
smlad rx, rx, rx, rx  
ldrd rx, [rx, #x]  
smuad rx, rx, rx  
smuad rx, rx, rx  
smlad rx, rx, rx, rx  
smlad rx, rx, rx, rx  
ldrd rx, [rx, #x]  
smuad rx, rx, rx  
smuad rx, rx, rx  
smlad rx, rx, rx, rx  
smlad rx, rx, rx, rx  
smlad rx, rx, rx, rx  
ldrd rx, [rx, #x]  
smlad rx, rx, rx, rx  
smlad rx, rx, rx, rx  
smlad rx, rx, rx, rx  
smlad rx, rx, rx, rx  
ldrd rx, [rx, #x]  
smlad rx, rx, rx, rx  
smlad rx, rx, rx, rx  
smlad rx, rx, rx, rx  
smlad rx, rx, rx, rx  
ldrd rx, [rx, #x]  
smlad rx, rx, rx, rx  
smlad rx, rx, rx, rx  
smlad rx, rx, rx, rx  
ldrd rx, [rx, #x]  
```

64 taps in 54 cycles = 1.19 taps/cycle

Tools and Other Considerations

Compiler excels on code size, not on code speed
- Compiler’s job is much harder with ARM11

ARM debugger provides limited visibility
- Cannot view registers, memory in common formats
- No easy way to view system-level behavior

Know the simulation models and development boards—and when to use which
- New cycle-accurate simulator is key to optimization
- WARNING: ETM may produce confusing results!

Don’t reinvent the wheel: use off-the-shelf software when appropriate
Insider Insights on the ARM11’s Signal-Processing Capabilities

Conclusions

ARM11 emphasizes speed over efficiency
- Fast on signal-processing tasks
- Relatively energy-hungry
- Relatively large, but good memory efficiency

ARM11 is more complicated than its predecessors
- Challenges are manageable with good planning
- Likely easier to program than a multiprocessor SoC

New tools help ease the pain
- Tools are still missing important features

Learning the tools and techniques is the key to success!

For More Information…

www.BDTi.com

Inside [DSP] newsletter and quarterly reports
Benchmark scores for dozens of processors
Pocket Guide to Processors for DSP
- Basic stats on over 40 processors
Articles, white papers, and presentation slides
- Processor architectures and performance
- Signal processing applications
- Signal processing software optimization
comp.dsp FAQ

© 2005 Berkeley Design Technology, Inc.