Insider Insights on the ARM11’s Signal-Processing Capabilities

Presentation Goals

By the end of this workshop, you should know:

• How the ARM11 differs from its predecessors
• How the ARM11’s multimedia and general signal processing performance compares to other GPPs’
• How the ARM11’s performance compares to DSPs’
• How to get the most out of the ARM11 in multimedia and signal processing applications
Insider Insights on the ARM11’s Signal-Processing Capabilities

<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>145 MHz</td>
<td>255 MHz</td>
<td>265 MHz</td>
<td>330-335 MHz</td>
</tr>
<tr>
<td><strong>Die Area w/o cache</strong></td>
<td>0.28 mm²</td>
<td>1.7 mm² (estimated)</td>
<td>1.68 mm²</td>
<td>2.3 - 2.5 mm²</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>0.11 mW/MHz</td>
<td>0.25 mW/MHz</td>
<td>0.3 mW/MHz</td>
<td>0.4 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>

* TSMC CL013G/Artisan SAGE-X, worst-case conditions
** ARM1156T2(F)-S only
*** ARM946E-S and ARM966E-S only

<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 1 x 32-bit 1 x 32-bit 1 x 32-bit</td>
<td>1 x 16-bit 2 x 16-bit 4 x 8-bit</td>
<td></td>
<td></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>Non-blocking Load</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>2 cycles</td>
<td>2 cycles</td>
<td>3 cycles</td>
</tr>
</tbody>
</table>

© 2006 Berkeley Design Technology, Inc.
Insider Insights on the ARM11’s Signal-Processing Capabilities

Signal Processing Speed
Certified BDTI DSP Kernel Benchmarks™ Results

Processor Loading for Video Decoder
Certified BDTI Video Decoder Benchmark™ Results
Insider Insights on the ARM11’s Signal-Processing Capabilities

**DSP Energy Efficiency for ARM Cores**
Certified BDTI DSP Kernel Benchmark™ Results

- ARM ARM1136j-S
- ARM ARM1176j-S
- Typical High-Performance DSP Core

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

**Signal Processing Memory Efficiency**
Certified BDTI DSP Kernel Benchmark™ Results

- ARM ARM9E
- ARM ARM1136
- CEVA CEVA-X1620
- Intel PXA27x
- MIPS MIPS524KEc
- TI ‘C55x

© 2006 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!

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

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

\[
\begin{align*}
N &= 40; \\
T &= 16; \\
& \text{for } (n=0; n<N; n++) \\
& \quad \text{for } (k=0; k<T; k++) \\
& \quad \quad \text{SUM} += x[n-k] \times h[k]; \\
& \quad y[n] = \text{SUM};
\end{align*}
\]

Analysis: Compiled FIR Filter

1 stall cycle

3 branches

2 instructions per load

2 instructions per branch

© 2006 Berkeley Design Technology, Inc.
Insider Insights on the ARM11’s Signal-Processing Capabilities

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 |
| SMLAB | 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 |
```

\[
N=40; \quad T=16; \quad \text{for } (n=0; n<N; n++) \{ \\
\quad \text{for } (k=0; SUM=0; k<T; k++) \{ \\
\quad \quad \text{SUM } += x[n-k] \times h[k]; \\
\quad \} \\
\quad y[n] = \text{SUM}; \\
\}
\]

10 cycles per tap = 0.10 taps per cycle

Give the Compiler a Hand

```
N=40; \quad T=16; 
#define N 40; 
#define T 16; 

for (n=0; n<N; n++) { \\
\quad \text{for } (k=0; SUM=0; k<T; k++) \{ \\
\quad \quad \text{SUM } += x[n-k] \times h[k]; \\
\quad \} \\
\quad y[n] = \text{SUM}; \\
\}
```

Use constants instead of variables
**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--) {
    for (k=T, SUM=0; k; k--) {
        short *xt = x++;
        short *ht = h++;
        SUM += *xt-- * *ht++;
    }
    *y++ = SUM;
}
```

**Count downwards in “for” loops**

---

**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**
Insider Insights on the ARM11’s Signal-Processing Capabilities

### 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  
1 stall cycle  

4 taps in 9 cycles = 0.44 taps per cycle

© 2006 Berkeley Design Technology, Inc.
Insider Insights on the ARM11’s Signal-Processing Capabilities

Software Pipelining

```
Loop  LDRD  r6, [r0], #8
LDRD  r8, [r1], #8

19
```

```
Loop  LDRD  r6, [r0], #8
SMLAD  r12, r4, r8, r12
LDRD  r10, [r1], #8
SMLAD  r12, r5, r9, r12
SUBS  r2, r2, #8
LDRGTD  r4, [r0], #8
SMLAD  r12, r6, r10, r12
LDRGTD  r8, [r1], #8
SMLAD  r12, r7, r11, r12
BGT  LOOP
```

8 taps in 10 cycles = 0.80 taps per cycle

```
pipelined second iteration
```

0 stall cycles

```
64 taps in 54 cycles = 1.19 taps/cycle
```

Fully Optimized FIR Inner Loop

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

```
64 taps in 54 cycles = 1.19 taps/cycle
```

```
64 taps in 54 cycles = 1.19 taps/cycle
```

© 2006 Berkeley Design Technology, Inc.
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
SoC Designer enables viewing system-level behavior
Know the simulation models and development boards—and when to use which
• Cycle-accurate simulator is key to optimization
• ETM may produce confusing results
Don’t reinvent the wheel: use off-the-shelf software when appropriate

Conclusions

ARM11 delivers serious DSP horsepower... plus GPP capabilities
• Fast on signal-processing tasks
  • Roughly 2x the speed of the ARM9E
  • Faster than many mid-range DSPs...
  • But slower than high-end DSPs
For pure DSP, not as efficient as DSPs
• Relatively energy-hungry compared to DSPs
• Die area similar to other GPP, DSP cores
  • But smaller than a GPP+DSP combo
Architecture is more complicated than its predecessors'
• Code optimization is more challenging, but manageable
• Simpler than a GPP+DSP combo
Tools lack DSP-savvy, but get the job done
Detailed knowledge of the architecture and tools is key to tapping the ARM11’s DSP capabilities!
Insider Insights on the ARM11’s Signal-Processing Capabilities

For More Information...

Inside DSP newsletter and website, www.insideDSP.com

www.BDTi.com
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

© 2006 Berkeley Design Technology, Inc.