An Introduction To The Cortex-M4 Processor Architecture

Shyam Sadasivan
Agenda

- Introduction
  - ARM and the Cortex™-M Family

- Cortex-M4 high performance and efficiency
  - Integer DSP features - single cycle MAC and SIMD
  - Floating Point Unit
  - Example programs and benchmarks

- Cortex-M4 ease-of-use
  - Nested vectored interrupt controller (NVIC)
  - CoreSight™ debug features
  - CMSIS standard and DSP library
ARM Cortex Advanced Processors

Architectural innovation, compatibility across diverse application spectrum

- **ARM Cortex-A family:**
  - Applications processors for feature-rich OS and 3rd party applications

- **ARM Cortex-R family:**
  - Embedded processors for real-time signal processing, control applications

- **ARM Cortex-M family:**
  - Microcontroller-oriented processors for MCU, ASSP, and SoC applications
Application Examples

Cortex-A
- servers
- set-top boxes
- netbooks
- mobile applications

Cortex-R
- disk drives
- digital cameras
- mobile baseband

Cortex-M
- appliances
- motors
- audio
Cortex-M M Processor Portfolio
ARM Cortex-M Processor Family

- Forget traditional 8/16/32-bit classifications
  - A compatible architecture spanning the embedded application range

**ARM Cortex-M4**
- “32-bit/DSC” applications
- Efficient digital signal control

**ARM Cortex-M3**
- “16/32-bit” applications
- Performance efficiency

**ARM Cortex-M0**
- “8/16-bit” applications
- Low-cost & simplicity
# Standardization - Driven by Software Reuse

- #1 factor in choosing a processor is the software development tools available for it

Factors considered most important when choosing a microprocessor
Cortex Microcontroller Standard (CMSIS)

- Cortex Microcontroller Software Interface Standard
  - Abstraction layer for all Cortex-M processor based devices
  - Developed in conjunction with silicon, tools and middleware partners

- Benefits to the embedded developer
  - Consistent software interfaces for silicon and middleware vendors
  - Simplifies re-use across Cortex-M processor-based devices
  - Reduces software development cost and time-to-market
  - Reduces learning curve for new Cortex microcontroller developers
Cortex-M Processor Industry Adoption

- ARM Cortex-M3 processor momentum continues
  - 35+ licensees in applications from MCU, SoC, wireless sensor nodes

- ARM Cortex-M0 processor success
  - More than 20 licensees already in MCU, mixed-signal and SoC

- New ARM Cortex-M4 already changing industry
  - Released end of ‘09 with licensees including Freescale and NXP
Fundamental Technologies
### Comparing ARM7TDMI® and Cortex-M

- **ARM7TDMI®** is a very successful processor
  - But developed more than 15 years ago
  - Many advances made in the Cortex-M family

<table>
<thead>
<tr>
<th>ARM7TDMI</th>
<th>Cortex-M</th>
</tr>
</thead>
<tbody>
<tr>
<td>No standard interrupt controller</td>
<td>Integrated Nested Vectored Interrupt Controller</td>
</tr>
<tr>
<td>Non-deterministic ISR entry</td>
<td>Deterministic interrupt response</td>
</tr>
<tr>
<td>Significant assembler code required</td>
<td>No assembler required</td>
</tr>
<tr>
<td>Optimal software development requires interworking between ARM and Thumb® code</td>
<td>Thumb-2 simplifies development</td>
</tr>
<tr>
<td>Lack of standardization inhibits apps porting</td>
<td>NVIC, SysTick &amp; Memory Map defined</td>
</tr>
<tr>
<td>Lack of power management support</td>
<td>Architected sleep mode support</td>
</tr>
</tbody>
</table>
Instruction Set Architecture

- **Thumb®**
  - 32-bit operations in 16-bit instructions
  - Introduced in ARM7TDMI processor (‘T’ stands for Thumb)
  - Subsequently supported in every ARM processor developed since

- **Thumb-2**
  - Enables a performance optimised blend of 16/32-bit instructions
  - All processor operations can all be handled in ‘Thumb’ state
  - Supported across the Cortex-M processor range

Thumb instruction set upwards compatibility

ARM7  ARM9  Cortex-M0  Cortex-M3  Cortex-M4  Cortex-R4  Cortex-A9
Nested Vectored Interrupt Controller

- Faster interrupt response
  - With less software effort

- ISR written directly in C
  - Interrupt table is simply a set of pointers to C routines
  - ISRs are standard C functions

- Integrated NVIC handles:
  - Saving corruptible registers
  - Exception prioritization
  - Exception nesting

<table>
<thead>
<tr>
<th>8051</th>
<th>Cortex-M</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. SJMP/L JMP from vector</td>
<td>1. Starting real handler</td>
</tr>
<tr>
<td>table to handler</td>
<td>code</td>
</tr>
<tr>
<td>2. PUSH PSW</td>
<td></td>
</tr>
<tr>
<td>3. ORL PSW, #00001000b</td>
<td></td>
</tr>
<tr>
<td>(to switch register bank)</td>
<td></td>
</tr>
<tr>
<td>4. Starting real handler</td>
<td></td>
</tr>
<tr>
<td>code</td>
<td></td>
</tr>
</tbody>
</table>

Diagram: Interrupt Handling

Tail-chain
Interrupt Behaviour

- On interrupt, hardware automatically stacks corruptible state
- Interrupt handlers can be written fully in C
  - Stack content supports C/C++ ARM Architecture Procedure Calling Standard
- Processor fetches initial stack pointer from 0x0 on reset
# Code Density

- Cortex-M shows smaller code size than 8/16-bit devices
- Consider a 16-bit multiply operation
  - Required for 10-bit ADC data filtering, encryption algorithms, audio

### 8-bit example (8051) vs 16-bit example

<table>
<thead>
<tr>
<th>8-bit example (8051)</th>
<th>16-bit example</th>
<th>ARM Cortex-M</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A, XL ; 2 bytes</td>
<td>MUL AB; 1 byte</td>
<td>MULS r0,r1,r0</td>
</tr>
<tr>
<td>MOV B, YL ; 3 bytes</td>
<td>ADD A, R1; 1 byte</td>
<td></td>
</tr>
<tr>
<td>MUL AB; 1 byte</td>
<td>MOV R1, A; 1 byte</td>
<td></td>
</tr>
<tr>
<td>MOV R0, A; 1 byte</td>
<td>MOV A, B; 2 bytes</td>
<td></td>
</tr>
<tr>
<td>MOV R1, B; 3 bytes</td>
<td>ADDC A, R2; 1 byte</td>
<td></td>
</tr>
<tr>
<td>MOV A, XL ; 2 bytes</td>
<td>MOV R2, A; 1 byte</td>
<td></td>
</tr>
<tr>
<td>MOV B, YH ; 3 bytes</td>
<td>MOV R2, A; 1 byte</td>
<td></td>
</tr>
<tr>
<td>MUL AB; 1 byte</td>
<td>MOV A, XH ; 2 bytes</td>
<td></td>
</tr>
<tr>
<td>ADD A, R1; 1 byte</td>
<td>MOV B, YH ; 3 bytes</td>
<td></td>
</tr>
<tr>
<td>MOV R1, A; 1 byte</td>
<td>MUL AB; 1 byte</td>
<td></td>
</tr>
<tr>
<td>MOV A, B; 2 bytes</td>
<td>ADD A, R2; 1 byte</td>
<td></td>
</tr>
<tr>
<td>ADDC A, #0 ; 2 bytes</td>
<td>MOV R2, A; 1 byte</td>
<td></td>
</tr>
<tr>
<td>MOV R2, A; 1 byte</td>
<td>MOV A, B; 2 bytes</td>
<td></td>
</tr>
<tr>
<td>MOV A, XH ; 2 bytes</td>
<td>MOVC A, #0 ; 2 bytes</td>
<td></td>
</tr>
<tr>
<td>MOV B, YL ; 3 bytes</td>
<td>MOV R3, A; 1 byte</td>
<td></td>
</tr>
</tbody>
</table>

| Time: 48 clock cycles*                                      | Time: 8 clock cycles                                      | Time: 1 clock cycle                       |
| Code size: 48 bytes                                         | Code size: 8 bytes                                        | Code size: 2 bytes                        |

* 8051 needs at least one cycle per instruction byte fetch as they only have an 8-bit interface
## Cortex-M Processor Power Modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Power Consumption</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active mode</td>
<td>Leakage + dynamic</td>
<td>Running Dhrystone 2.1 benchmark</td>
</tr>
<tr>
<td>Sleep mode</td>
<td>Leakage + some dynamic</td>
<td>Core clock gated, NVIC awake</td>
</tr>
<tr>
<td>Deep Sleep mode</td>
<td>Leakage only</td>
<td>Power still on, most clocks off</td>
</tr>
<tr>
<td>Deep Sleep mode</td>
<td>State retention (WIC)</td>
<td>Most power off, all clocks off</td>
</tr>
<tr>
<td>Power off</td>
<td>Zero power</td>
<td>Power off</td>
</tr>
</tbody>
</table>

Not To scale
32-bit Energy Efficiency Advantage

Average power = 13 µW

Average power = 6.8 µW
47% lower!
ARM Cortex-M4

In-depth
Cortex-M4 Blends MCUs and DSPs

**MCU**
- Ease of use
- C Programming
- Interrupt handling
- Ultra low power

**DSP**
- Harvard architecture
- Single cycle MAC
- Floating Point
- Barrel shifter
Cortex-M4 Processor Overview

- Wake-Up Interrupt Controller: for Low Power Stand by Operation
- Integrated Nested Vectored Interrupt Controller and SYSTICK Timer
- Central Core: 1.25 DMIPS/MHz Single cycle MAC
- Single precision Floating Point Unit
- Memory Protection Unit (MPU) 8-Region
- Embedded Trace Macrocell (ETM) for Instruction Trace
- Instrumentation Trace Macrocell (ITM) for Data Trace via Single Wire Output
- Flash Patch & Breakpoint Unit 8x Hardware Breakpoints
- Debug Access Port: JTAG or Serial Wire
- Instrumentation Trace Macrocell (ITM ) for Data Trace via Single Wire Output
- Data Watch Point and Trace Unit (DWT) 4x Data Watchpoints & Event Monitors
- Flash Patch & Breakpoint Unit 8x Hardware Breakpoints
- 2x AHB-Lite Buses I_CODE (Instruction Code Bus) D_CODE (Data / Coefficients Code Bus)
- 1x AHB-Lite Buses SYSTEM (SRAM & Fast Peripherals) 1x APB Bus ARM Peripheral Bus (Internal & Slow Peripherals)

Optional blocks, please consult your silicon manufacturer’s data sheet
Cortex-M4 - What’s Unique About it?

Most energy efficient 32-bit embedded processor for MCU+DSP requirements

Brings high performance signal processing within the reach of the typical MCU programmer
Agenda

- Introduction
  - Cortex-M4 unique value proposition

- Cortex-M4 high performance and efficiency
  - Integer DSP features - single cycle MAC and SIMD
  - Floating Point Unit
  - Example programs and benchmarks

- Cortex-M4 ease-of-use
  - Nested vectored interrupt controller (NVIC)
  - CoreSight debug features
  - CMSIS standard and DSP library
Highest In-class Efficiency

The Cortex-M4 is ~2X more efficient on most DSP tasks than leading 16 and 32 bit MCU devices with DSP extensions.

Cycle counts on DSP tasks compared, smaller is better
SIMD Operations

SIMD extensions perform multiple operations in one cycle

\[ Sum = Sum + (A \times C) + (B \times D) \]

SIMD techniques operate with packed data

- 32-bit packed data
- 64-bit packed data

The Architecture for the Digital World®
# 8,16-bit SIMD Arithmetic

<table>
<thead>
<tr>
<th>Prefix</th>
<th>S Signed</th>
<th>Q Signed Saturating</th>
<th>SH Signed Halving</th>
<th>U Unsigned</th>
<th>UQ Unsigned Saturating</th>
<th>UH Unsigned Halving</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD8</td>
<td>SADD8</td>
<td>QADD8</td>
<td>SHADD8</td>
<td>USADD8</td>
<td>UQADD8</td>
<td>UHADD8</td>
</tr>
<tr>
<td>SUB8</td>
<td>SSUB8</td>
<td>QSUB8</td>
<td>SHSUB8</td>
<td>USUB8</td>
<td>UQSUB8</td>
<td>UHSUB8</td>
</tr>
<tr>
<td>ADD16</td>
<td>SADD16</td>
<td>QADD16</td>
<td>SHADD16</td>
<td>UADD16</td>
<td>UQADD16</td>
<td>UHADD16</td>
</tr>
<tr>
<td>SUB16</td>
<td>SSUB16</td>
<td>QSUB16</td>
<td>SHSUB16</td>
<td>USUB16</td>
<td>UQSUB16</td>
<td>UHSUB16</td>
</tr>
<tr>
<td>ASX</td>
<td>SASX</td>
<td>QASX</td>
<td>SHASX</td>
<td>UASX</td>
<td>UQASX</td>
<td>UHASX</td>
</tr>
<tr>
<td>SAX</td>
<td>SSAX</td>
<td>QSAX</td>
<td>SHSAX</td>
<td>USAX</td>
<td>UQSAX</td>
<td>UHSAX</td>
</tr>
<tr>
<td>USAD8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USADA8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**USAD8**
- Unsigned Sum of Absolute Difference (8 bits)

**USADA8**
- Unsigned Sum of Absolute Difference and Accumulate (8 bits)

**Instr ASX**
1. Exchanges halfwords of the second operand register
2. Adds top halfwords and subtracts bottom halfwords

**Instr SAX**
1. Exchanges halfwords of the second operand register
2. Subtracts top halfwords and adds bottom halfwords
DSP Operations – MAC is Key Operation

- **FIR Filter**
  \[ y[k] = \sum_{k=0}^{N-1} h[k] x[k] - k_0 \]

- **IIR Filter**
  \[ y[k] = b_0 x[k] + b_1 x[k-1] + b_2 x[k-2] + a_1 y[k-1] + a_2 y[k-2] \]

- **FFT**
  \[ Y_1 = X_1 + X_2 \]
  \[ Y_2 = \langle X_1 - X_2 \rangle e^{-j\omega} \]

Most operations are dominated by MACs
These can be operations on 8, 16 or 32 bit data
## Extended Single Cycle MAC

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>INSTRUCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$16 \times 16 = 32$</td>
<td>SMULBB, SMULBT, SMULTB, SMULTT</td>
</tr>
<tr>
<td>$16 \times 16 + 32 = 32$</td>
<td>SMLABB, SMLABT, SMLATB, SMLATT</td>
</tr>
<tr>
<td>$16 \times 16 + 64 = 64$</td>
<td>SMLALBB, SMLALBT, SMLALTB, SMLALTT</td>
</tr>
<tr>
<td>$16 \times 32 = 32$</td>
<td>SMULWB, SMULWT</td>
</tr>
<tr>
<td>$(16 \times 32) + 32 = 32$</td>
<td>SMLAWB, SMLAWT</td>
</tr>
<tr>
<td>$(16 \times 16) \times (16 \times 16) = 32$</td>
<td>SMUAD, SMUADX, SMUSD, SMUSDX</td>
</tr>
<tr>
<td>$(16 \times 16) \times (16 \times 16) + 32 = 32$</td>
<td>SMLAD, SMLADX, SMLSD, SMLSDX</td>
</tr>
<tr>
<td>$(16 \times 16) \times (16 \times 16) + 64 = 64$</td>
<td>SMLALD, SMLALDX, SMLSLD, SMLSDX</td>
</tr>
<tr>
<td>$32 \times 32 = 32$</td>
<td>MUL</td>
</tr>
<tr>
<td>$32 \times (32 \times 32) = 32$</td>
<td>MLA, MLS</td>
</tr>
<tr>
<td>$32 \times 32 = 64$</td>
<td>SMULL, UMULL</td>
</tr>
<tr>
<td>$(32 \times 32) + 64 = 64$</td>
<td>SMLAL, UMLAL</td>
</tr>
<tr>
<td>$(32 \times 32) + 32 + 32 = 64$</td>
<td>UMAAL</td>
</tr>
<tr>
<td>$32 \times (32 \times 32) = 32 \text{ (upper)}$</td>
<td>SMMLA, SMMLAR, SMMLS, SMMLSR</td>
</tr>
<tr>
<td>$(32 \times 32) = 32 \text{ (upper)}$</td>
<td>SMMUL, SMMULR</td>
</tr>
</tbody>
</table>

All the above operations are single cycle on the Cortex-M4 processor
# DSP Instructions Compared

<table>
<thead>
<tr>
<th>CLASS</th>
<th>INSTRUCTION</th>
<th>ARM9E-S</th>
<th>CORTEX-M3</th>
<th>Cortex-M4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>ALU operation (not PC)</td>
<td>1 - 2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>ALU operation to PC</td>
<td>3 - 4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>CLZ</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>QADD, QDADD, QSUB, QDSUB</td>
<td>1 - 2</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>QADD8, QADD16, QSUB8, QSUB16</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>QDADD, QSUB</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>QASX, QSAX, SASX, SSAX</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SHASX, SHSAX, UHASX, UHSAX</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SADD8, SADD16, SSUB8, SSUB16</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SHADD8, SHADD16, SHSUB8, SHSUB16</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>UQADD8, UQADD16, UQSUB8, UQSUB16</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>UHADD8, UHADD16, UHSUB8, UHSUB16</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>UADD8, UADD16, USUB8, USUB16</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>UQASX, UQSAX, USAX, UASX</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>UXTAB, UXTAB16, UXTAH</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>USAD8, USADA8</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td>Multiplication</td>
<td>MUL, MLA</td>
<td>2 - 3</td>
<td>1 - 2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>MULS, MLAS</td>
<td>4</td>
<td>1 - 2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SMULL, UMULL, SMLAL, UMLAL</td>
<td>3 - 4</td>
<td>5 - 7</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SMULBB, SMULBT, SMULTB, SMULTT</td>
<td>1 - 2</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SMLABBB, SMLABTB, SMLATT</td>
<td>1 - 2</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SMULWBB, SMULWT, SMLAWB, SMLAWT</td>
<td>1 - 2</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SMLALBB, SMLALBT, SMLALTB, SMLALTT</td>
<td>2 - 3</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SMLAD, SMLADX, SMLALD, SMLALDX</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SMLSD, SMLSDX</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SMLSDL, SMLSDL</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SMMLA, SMMLAR, SMMLS, SMMLSR</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SMMLUL, SMMLUR</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SMUAD, SMUADX, SMUSD, SMUSDX</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>UMAAL</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td>Division</td>
<td>SDIV, UDIV</td>
<td>n/a</td>
<td>2 - 12</td>
<td>2 - 12</td>
</tr>
</tbody>
</table>

**Cycle counts**

- **Single cycle MAC**
Single Precision Floating Point

- Floating point critical for
  - Motor control – extended range necessary, double precision overkill
  - Industrial/factory automation – strong alignment with meta-languages
  - Sensing and control – highly accurate measurement requirements

- Cortex-M4 FPU
  - IEEE 754 standard compliant
  - Single-precision floating point math

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>CYCLE COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add/Subtract</td>
<td>1</td>
</tr>
<tr>
<td>Divide</td>
<td>14</td>
</tr>
<tr>
<td>Multiply</td>
<td>1</td>
</tr>
<tr>
<td>Multiply Accumulate (MAC)</td>
<td>3</td>
</tr>
<tr>
<td>Fused MAC</td>
<td>3</td>
</tr>
<tr>
<td>Square Root</td>
<td>14</td>
</tr>
</tbody>
</table>
Code Example – Cortex-M4 FIR

\[ y_{32}[n] = \sum_{i=0}^{N} x_{16}[n-i] \times c_{16}[i] \]

- \( y_{32} \) – filter output (32 bit); \( N \) - filter order
- \( x_{16} \) – 16 bit input data; \( c_{16} \) – 16 bit filter coefficient

**Cortex-M3 Code Segment:**

```
FIR_LOOP:
LDR R2, [R0], #4 ; (2) Load input \( x_{16} \)
LDR R3, [R1], #4 ; (2) Load coeff \( c_{16} \)
SXTH R4, R2 ; (1) Extract \( x_{16}[n-i] \)
ASR R2, R2, #16 ; (1) Extract \( x_{16}[n-i-1] \)
SXTH R5, R3 ; (1) Extract \( c_{16}[i] \)
ASR R3, R3, #16 ; (1) Extract \( c_{16}[i+1] \)
MLA R6, R4, R5 ; (2) \( y_{32} \) += \( x_{16}[n-i] \times c_{16}[i] \)
MLA R6, R2, R3 ; (2) \( y_{32} \) += \( x_{16}[n-i-1] \times c_{16}[i+1] \)
SUBS R7, R7, #2 ; (1) loop count -= 2
BNE FIR_LOOP ; (2)
```

**Cortex-M4 Code Segment:**

```
FIR_LOOP:
LDR R2, [R0], #4 ; (1) Load input \( x_{16}[n-i], x_{16}[n-i-1] \)
LDR R3, [R1], #4 ; (2) Load coeff \( c_{16}[i], c_{16}[i+1] \)
SUBS R5, R5, #2 ; (1) loop count -= 2
SMLAD R4, R2, R3 ; (1) \( y_{32} \) += \( x_{16}[n-i], n-i-1 \) \( c_{16}[i,i+1] \)
BNE FIR_LOOP ; (2)
```

**Processor** | Kernel cycles | Total Cycles | Number of Instructions | Register usage
---|---|---|---|---
Cortex-M3 | 8 | 15 | 10 | 7
Cortex-M4 | 1 | 7 | 5 | 5
Advantage | 8x | \(~2.2x\) | 2x | 1.4x

Note:
1. In these examples, FIR_LOOP is unrolled by 2
2. This example assumes number of taps is even.

Example (non-binding) from Ittiam Systems, a leading provider of signal processing software on ARM platforms
Cortex-M4 - MP3 Playback in <10MHz

MHz required for MP3 decode, smaller is better

- General Purpose MCUs
- Discrete DSPs
- Cortex-M4 (estimated)
- Specialised Audio DSPs

MHz bandwidth requirement for MP3 decode

<0.5mW processor power!
Agenda

- Introduction
  - Cortex-M4 unique value proposition

- Cortex-M4 high performance and efficiency
  - Integer DSP features - single cycle MAC and SIMD
  - Floating Point Unit
  - Example programs and benchmarks

- Cortex-M4 ease-of-use
  - Nested vectored interrupt controller (NVIC)
  - CoreSight debug features
  - CMSIS standard and DSP library
Cortex-M4 – Easy to Use

- Nested Vector Interrupt Controller
  - High performance inbuilt interrupt controller

- Advanced debug features
  - CoreSight for “on the fly” debug

- CMSIS
  - Standard for writing, maintaining and porting code on Cortex-M4
  - CMSIS support for Cortex-M4 already available

- Cortex-M4 processor can be fully programmed in C
  - Programming fully in C leads to high optimization, full compiler support already available through software tools
Nested Vector Interrupt Controller

- NVIC is a core peripheral
  - Consistent between Cortex-M cores
  - Tailored towards fast and efficient interrupt handling
- Number of interrupts/priorities can be configured by manufacturer
  - 1 ... 240 interrupt channels, 8 – 256 interrupt priorities
- NVIC configured via memory-mapped control registers
- Interruptible LDM/STM (and PUSH/POP) for low interrupt latency
  - Continued on return from interrupt
- When an interrupt occurs:
  - The exception vector is fetched over the ICODE bus
  - In parallel, the processor state is saved over the SYSTEM bus
  - Automatic save and restore of processor state
    - Provides low latency interrupt/exception entry and exit
    - Allows handler to be written entirely in ‘C’
- Interrupt latency – Just 12 cycles to enter an interrupt
Cortex-M4 Interrupt Response [1]

Tail chaining
- 12 cycles from IRQ1 to ISR1 (Interruptible/Continual LSM)
- 6 cycles from ISR1 exit to ISR2 entry
- 12 cycles to return from ISR2

Late arrival
- 12 cycles to ISR entry
- Parallel stacking & inst. fetch
- Target ISR may be changed until last cycle
- When IRQ1 occurs new target ISR set
Cortex-M4 Interrupt Response [2]

**Pop pre-emption**

- Hardware un-stacking interruptible
- If interrupted only 6 cycles required to enter ISR2

![Cortex-M4 Interrupt Handling Diagram]

- ISR 1
- ISR 2
- Pop
- Abandon Pop (1-12 Cycles)
- 6 Cycles Tail-Chaining
- 12 Cycles
System Debug Challenges

- Debug a running system
  - Many embedded systems cannot be stopped for debug
    - Brushless DC motor control
    - Communication protocols lose their handshaking
  - Analyze dynamic system behaviour - just a snapshot will not do
  - Optimize performance bottlenecks for real-time systems

- Traditional debugging
  - Intrusive debug
  - Limited or prohibitive trace support
  - Few breakpoints and watchpoints
CoreSight Introduction

- Debug and trace technology in Cortex-M devices
- On-the-fly debugging
  - Debug application while the processor is running
    - Set breakpoints, read/write memory locations
  - Direct access to memory, no need to go via processor
  - Increased number of breakpoints and watchpoints
- Flexible trace options
  - Integrated Data Trace
  - Optional Instruction Trace (ETM)
- Reduced pin count interface
  - 2-pin Serial Wire Debug (SWD)
  - 1-pin Serial Wire Viewer (SWV)
  - Uses standard JTAG connectors
Cortex Microcontroller Standard (CMSIS)

- Cortex Microcontroller Software Interface Standard
  - Abstraction layer for all Cortex-M processor-based devices
  - Developed in conjunction with silicon, tools and middleware partners

- Benefits to the embedded developer
  - Consistent software interfaces for silicon and middleware vendors
  - Simplifies re-use across Cortex-M processor-based devices
  - Reduces software development cost and time-to-market
  - Reduces learning curve for new Cortex microcontroller developers
CMSIS files

Compiler Vendor-Independent Files:

- Cortex-Mx Core Files *(provided by ARM)*
  - core_cm4.h+core_cm4.c  core_cm4.h+core_cm4.c
- Device-specific Files *(provided by Silicon Vendors)*
  - Register Header File (*device.h*)
  - System Startup File (*system_\_device.c*)
- Compatible with all supported Compilers (IAR, RealView, GNU..)

Compiler-Vendor + Device-Specific Startup File:

- Device Specific Compiler Startup Code *(provided by Silicon Vendors)*
  - startup_device.s

CMSIS Files are available via [www.onARM.com](http://www.onARM.com):

- Device Database that lists all available devices
  - CMSIS Files can be downloaded
CMSIS DSP Library for Cortex-M4

- Designed to help MCU users develop DSP programs easily
- CMSIS compliant library with DSP algorithms in C
- Large number of algorithms in 8, 16 and 32-bit data formats
  - Basic math – vector mathematics
  - Fast math – sin, cos, sqrt etc
  - Interpolation – linear, bilinear
  - Statistics – max, min, RMS etc
  - Filtering – IIR, FIR, LMS etc
  - Transforms – FFT(real and complex), Cosine transform etc
  - Matrix functions
  - PID Controller, Clarke and Park transforms
  - Support functions – copy/fill arrays, data type conversions etc
- Available from ARM in Q4 2010
Summary

- What is the Cortex-M4 processor?
  - Cortex-M processor specifically designed for MCU+DSP requirements
  - Key features – Single cycle MAC, Floating point unit, Debug/trace
  - Key markets - Motor control, industrial automation, automotive, audio

- Why this processor?
  - Efficient blend of control and DSP features is key to growth markets
  - Advanced debug/trace features key for markets like motor control

- How is it different?
  - Most efficient 32-bit processor for MCU+DSP requirements
    - MP3 decode within 0.5 mW processor power
  - Easy to use tools through strong software ecosystem
    - CMSIS ensures portability
Thank You

Please visit www.arm.com for ARM related technical details

For any queries contact < Salesinfo-IN@arm.com >