

An Introduction To The Cortex-M4 Processor Architecture

Shyam Sadasivan



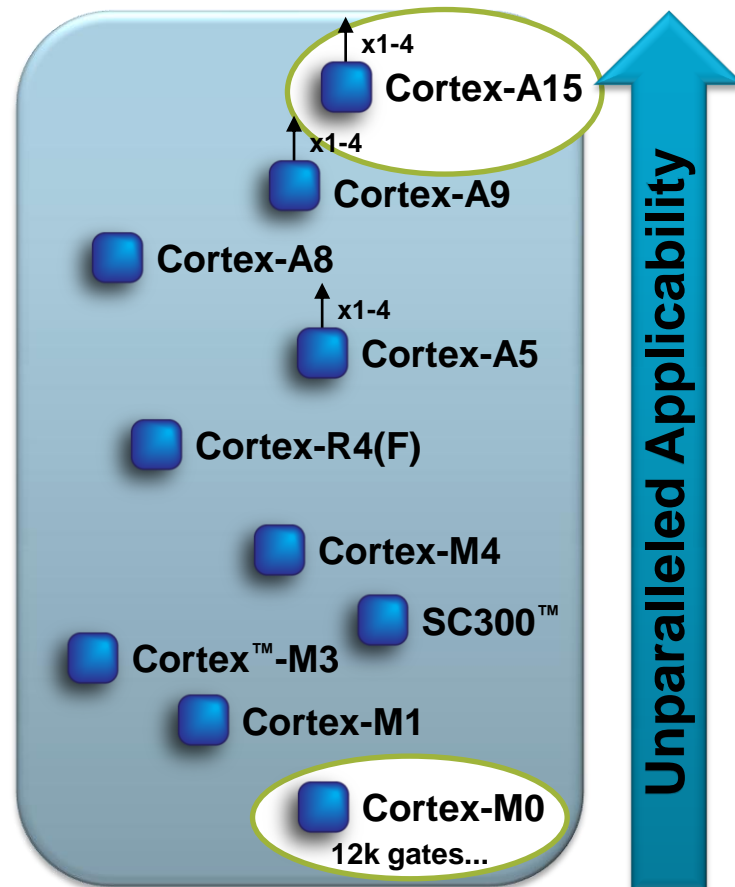
Agenda

- 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 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

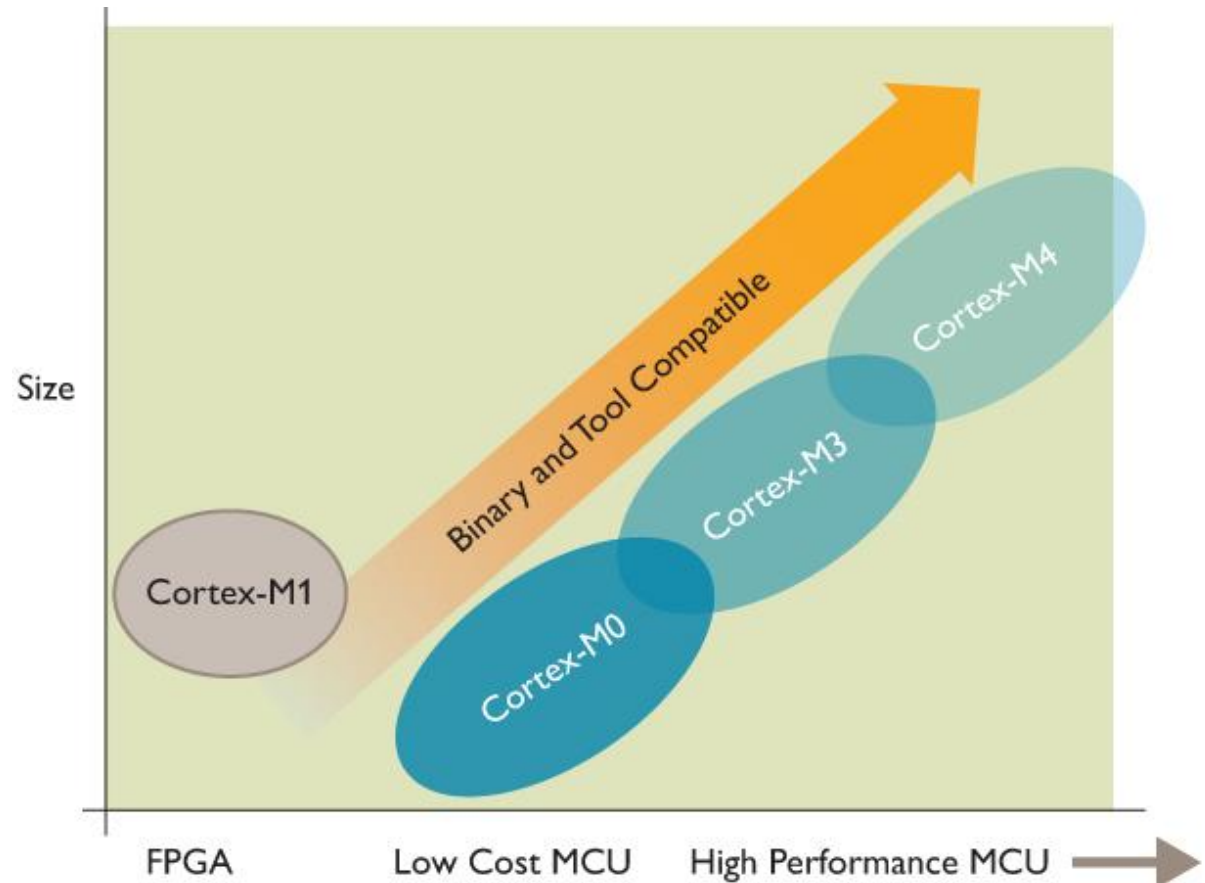
“32-bit/DSP” applications
Efficient digital signal control

ARM Cortex-M3

“16/32-bit” applications
Performance efficiency

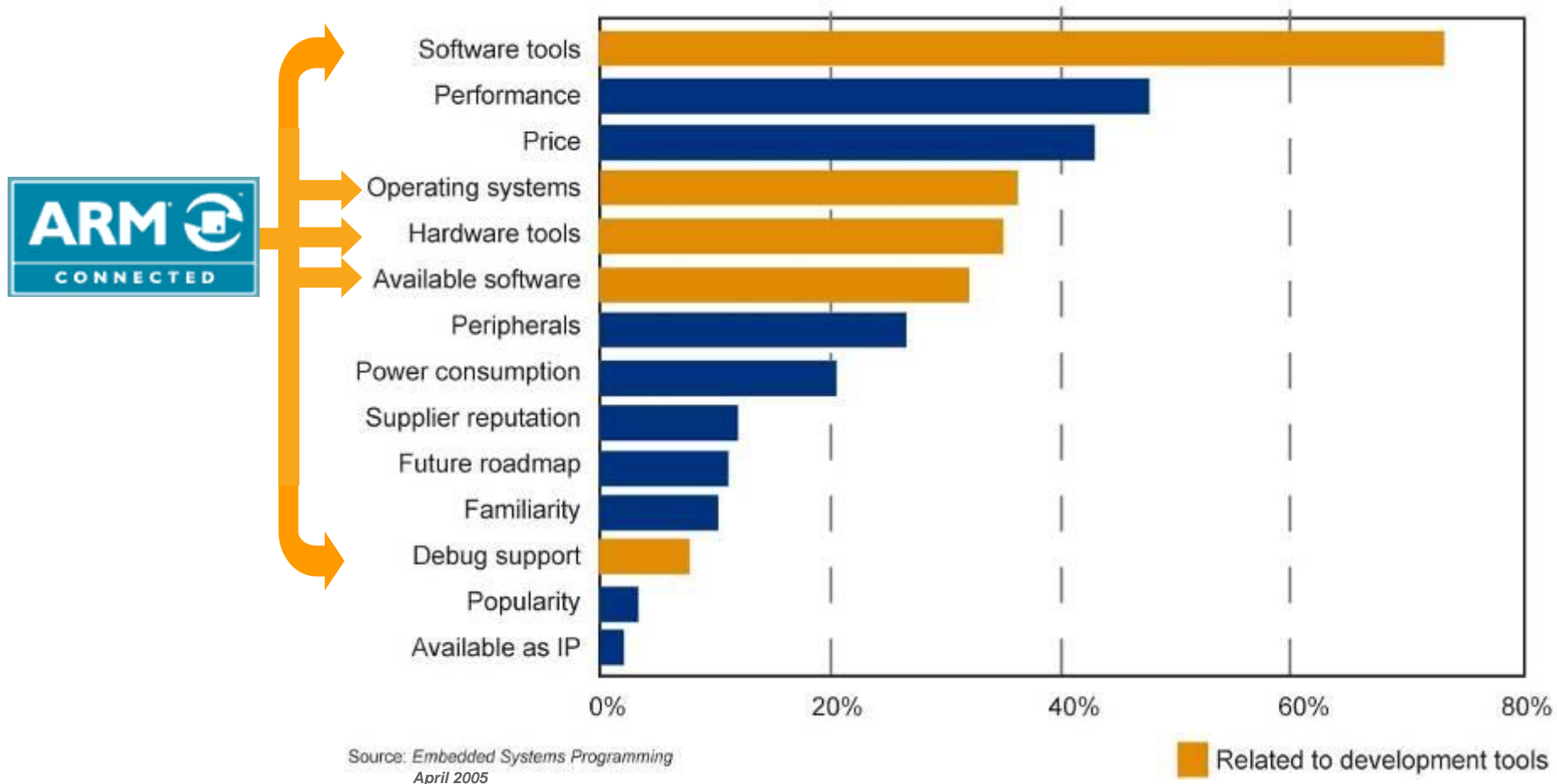
ARM Cortex-M0

“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 ARM7 and Cortex-M

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

ARM7TDMI	Cortex-M
No standard interrupt controller	Integrated Nested Vectored Interrupt Controller ✓
Non-deterministic ISR entry	Deterministic interrupt response ✓
Significant assembler code required	No assembler required ✓
Optimal software development requires interworking between ARM and Thumb [®] code	Thumb-2 simplifies development ✓
Lack of standardization inhibits apps porting	NVIC, SysTick & Memory Map defined ✓
Lack of power management support	Architected sleep mode support ✓

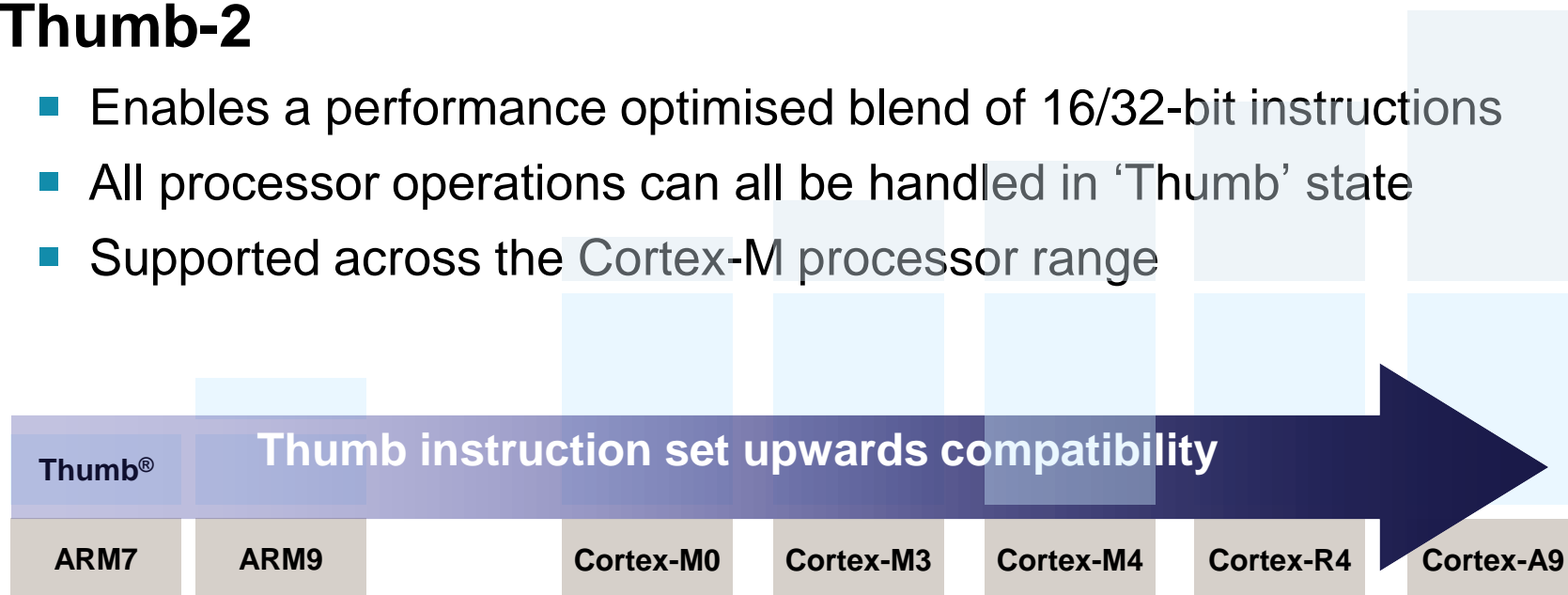
Instruction Set Architecture

■ Thumb®

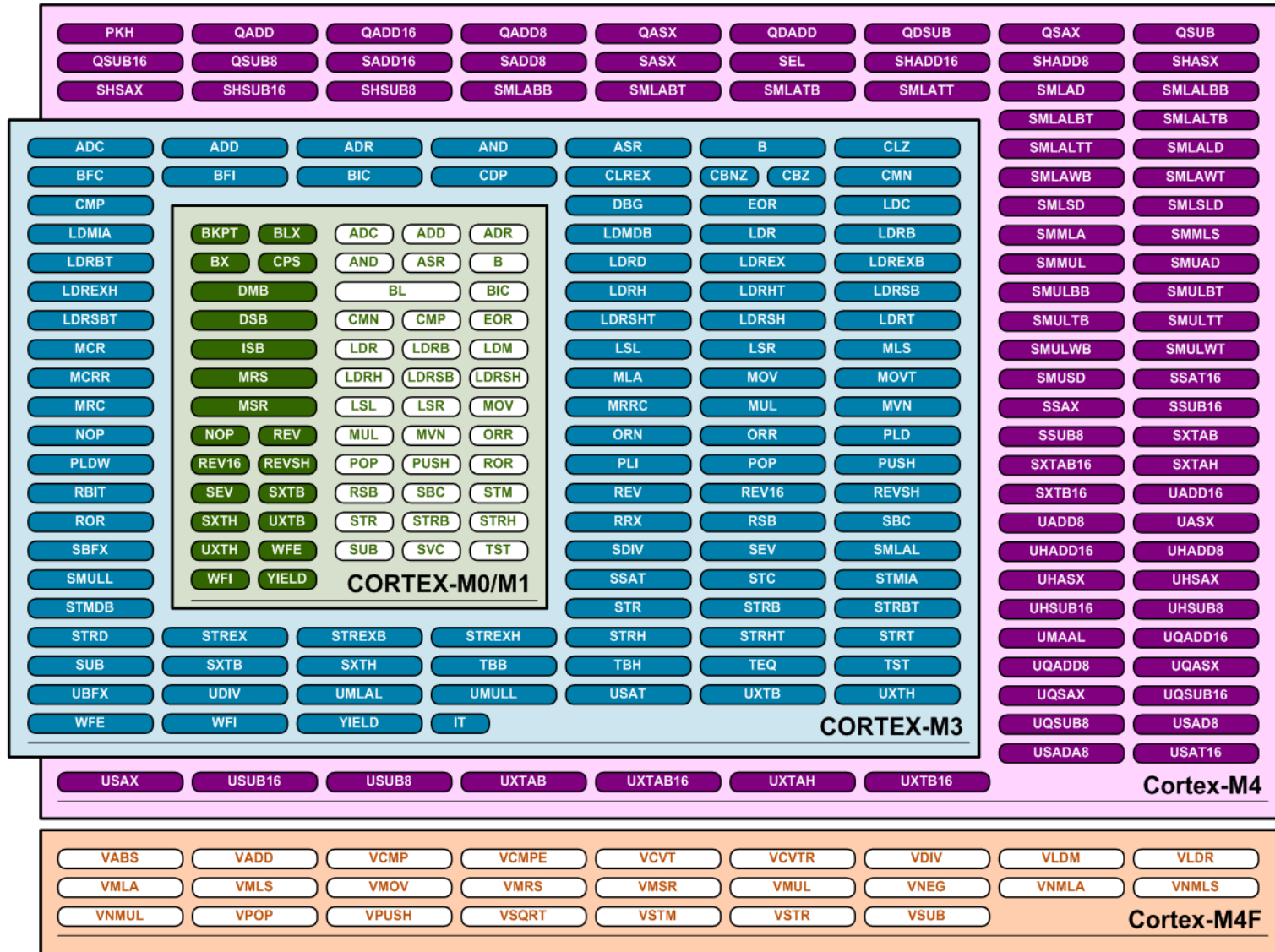
- 32-bit operations in 16-bit instructions
- Introduced in ARM7TDMI processor ('T' stands for Thumb)
- Subsequently supported in every ARM processor developed since

■ 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



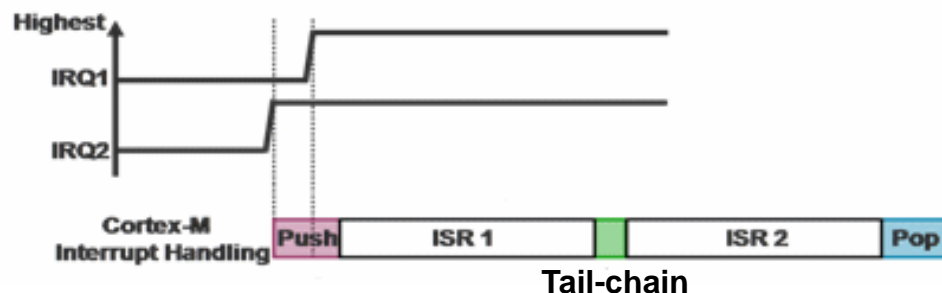
Instruction Set Architecture



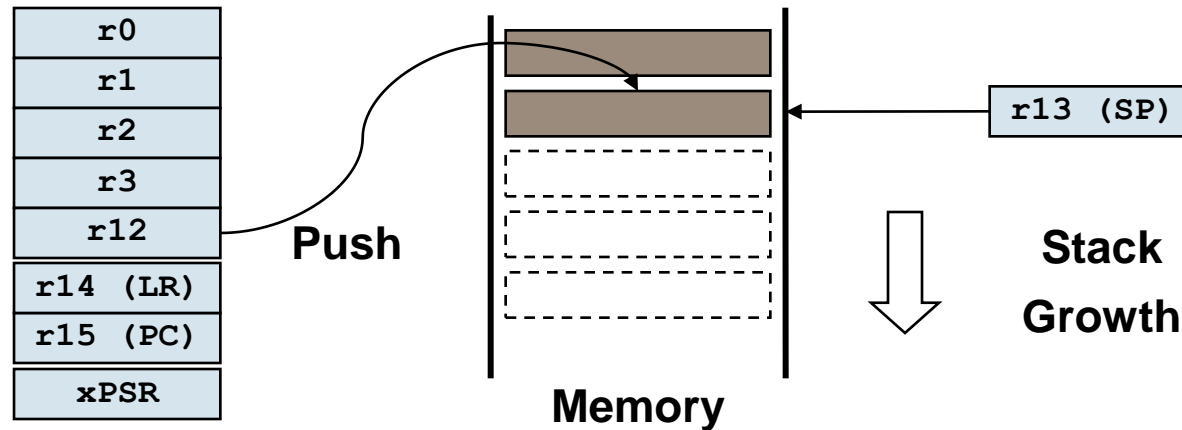
Nested Vectored Interrupt Controller

- **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

8051	Cortex-M
<ol style="list-style-type: none">1. SJMP/LJMP from vector table to handler2. PUSH PSW3. ORL PSW, #00001000b (to switch register bank)4. Starting real handler code	<ol style="list-style-type: none">1. Starting real handler code



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

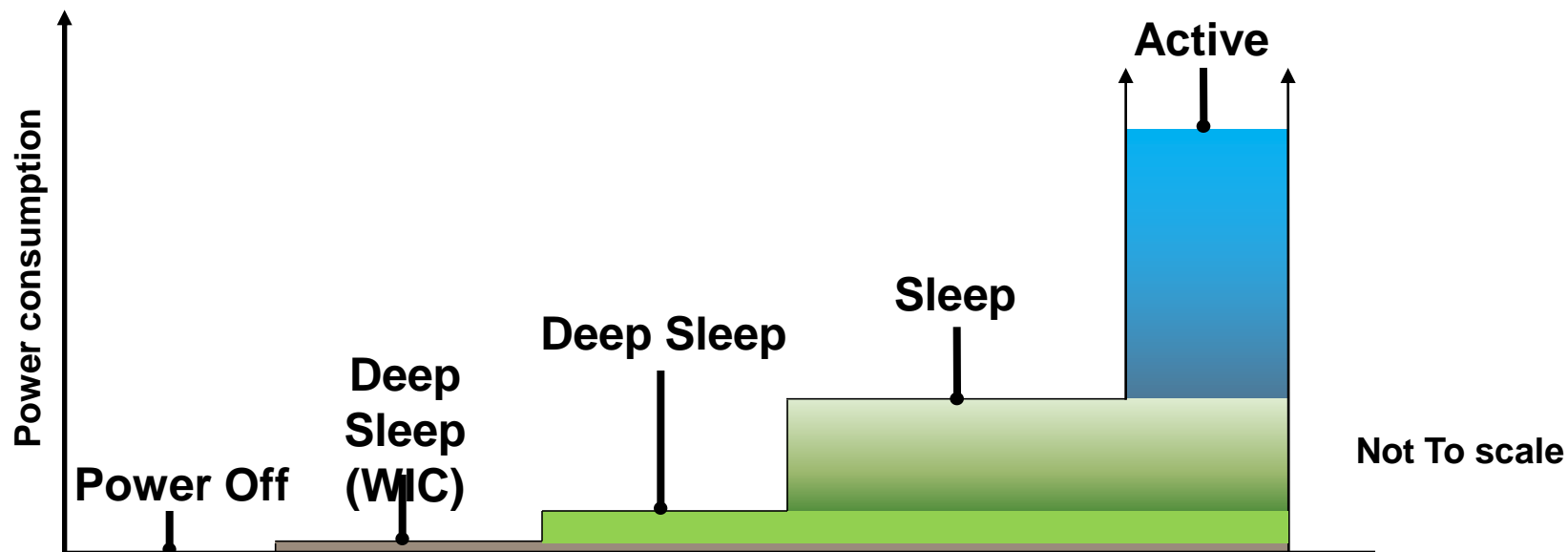
- Cortex-M shows smaller code size than 8/16-bit devices
- Consider a 16-bit multiply operation
 - Required for 10-bit ADC data filtering, encryption algorithms, audio

8-bit example (8051)	16-bit example	ARM Cortex-M
<pre> MOV A, XL ; 2 bytes MOV B, YL ; 3 bytes MUL AB; 1 byte MOV R0, A; 1 byte MOV R1, B; 3 bytes MOV A, XL ; 2 bytes MOV B, YH ; 3 bytes MUL AB; 1 byte ADD A, R1; 1 byte MOV R1, A; 1 byte MOV A, B ; 2 bytes ADDC A, #0 ; 2 bytes MOV R2, A; 1 byte MOV A, XH ; 2 bytes MOV B, YL ; 3 bytes </pre>	<pre> MUL AB; 1 byte ADD A, R1; 1 byte MOV R1, A; 1 byte MOV A, B ; 2 bytes ADDC A, R2 ; 1 bytes MOV R2, A; 1 byte MOV A, XH ; 2 bytes MOV B, YH ; 3 bytes MUL AB; 1 byte ADD A, R2; 1 byte MOV R2, A; 1 byte MOV A, B ; 2 bytes ADDC A, #0 ; 2 bytes MOV R3, A; 1 byte </pre>	<pre> MOV R1, &MulOp1 MOV R2, &MulOp2 MOV SumLo, R3 MOV SumHi, R4 (Memory mapped multiply unit) MULS r0, r1, r0 </pre>
<p>Time: 48 clock cycles* Code size: 48 bytes</p>	<p>Time: 8 clock cycles Code size: 8 bytes</p>	<p>Time: 1 clock cycle Code size: 2 bytes</p>

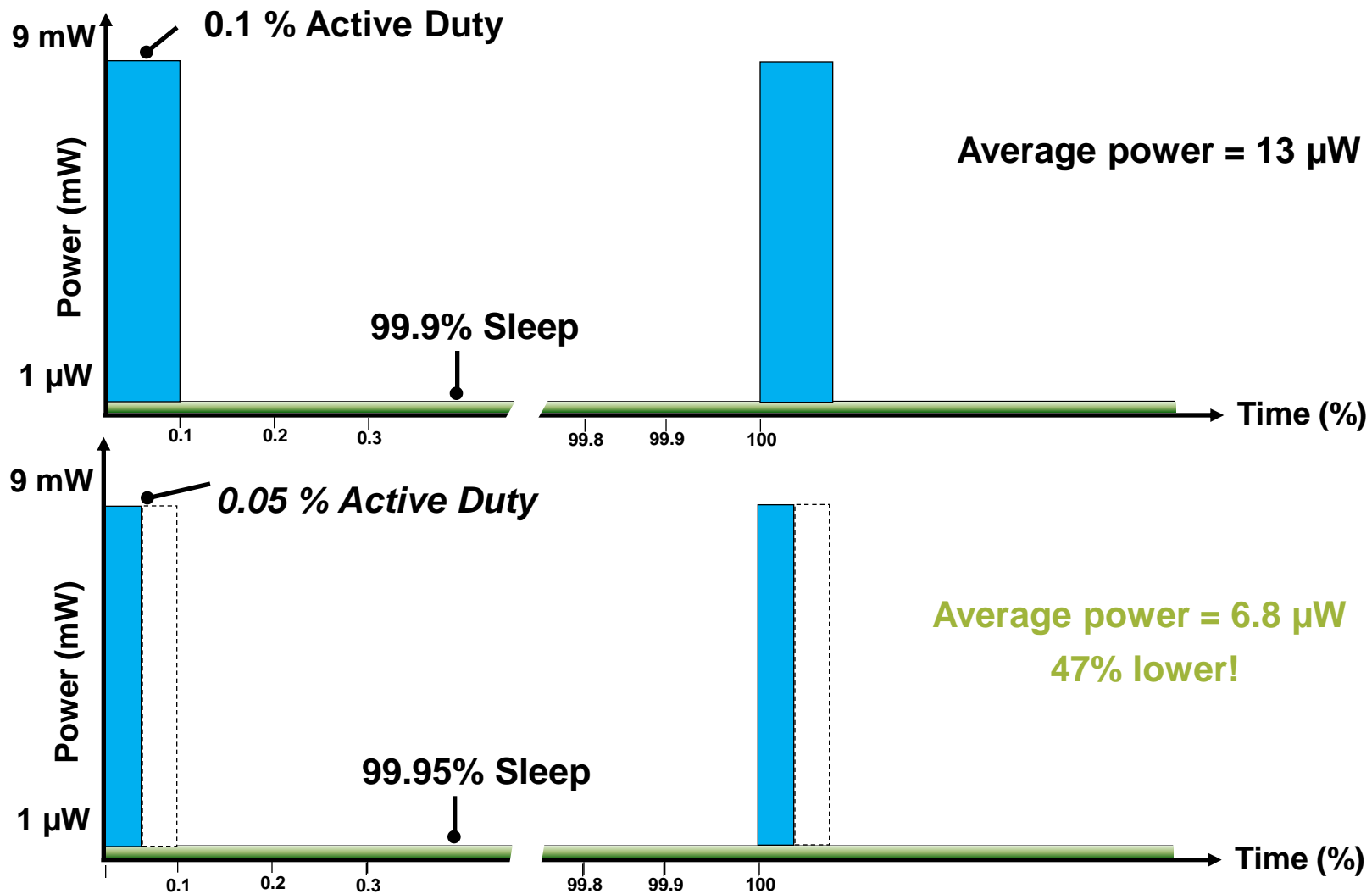
* 8051 needs at least one cycle per instruction byte fetch as they only have an 8-bit interface

Cortex-M Processor Power Modes

Active mode	Leakage + dynamic	Running Dhrystone 2.1 benchmark
Sleep mode	Leakage + some dynamic	Core clock gated, NVIC awake
Deep Sleep mode	Leakage only	Power still on, most clocks off
Deep Sleep mode	State retention (WIC)	Most power off, all clocks off
Power off	Zero power	Power off



32-bit Energy Efficiency Advantage

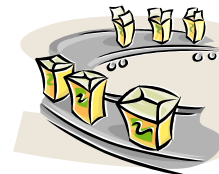
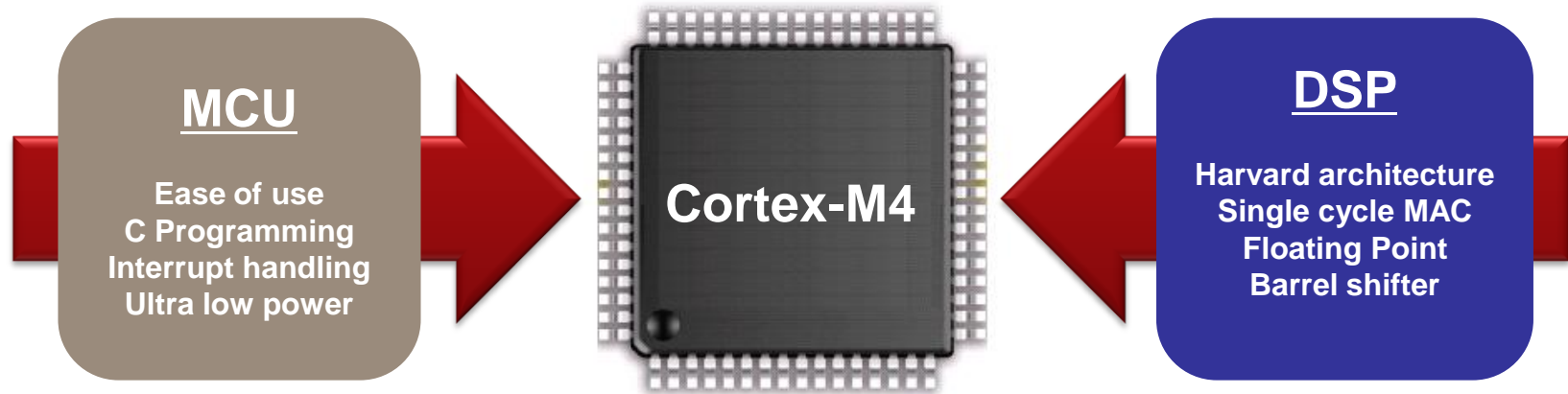


ARM Cortex-M4

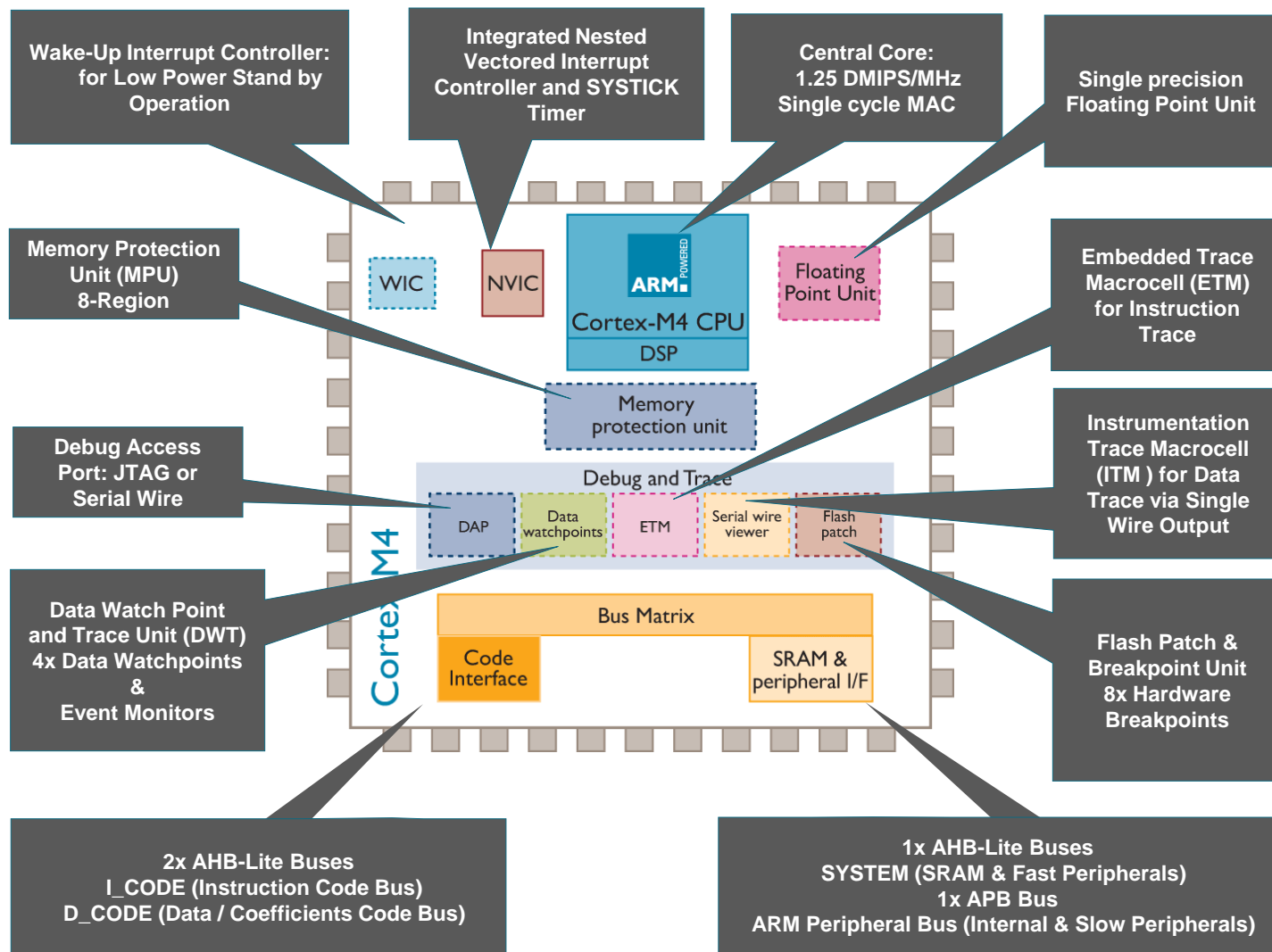
In-depth




Cortex-M4 Blends MCUs and DSPs



Cortex-M4 Processor Overview



 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



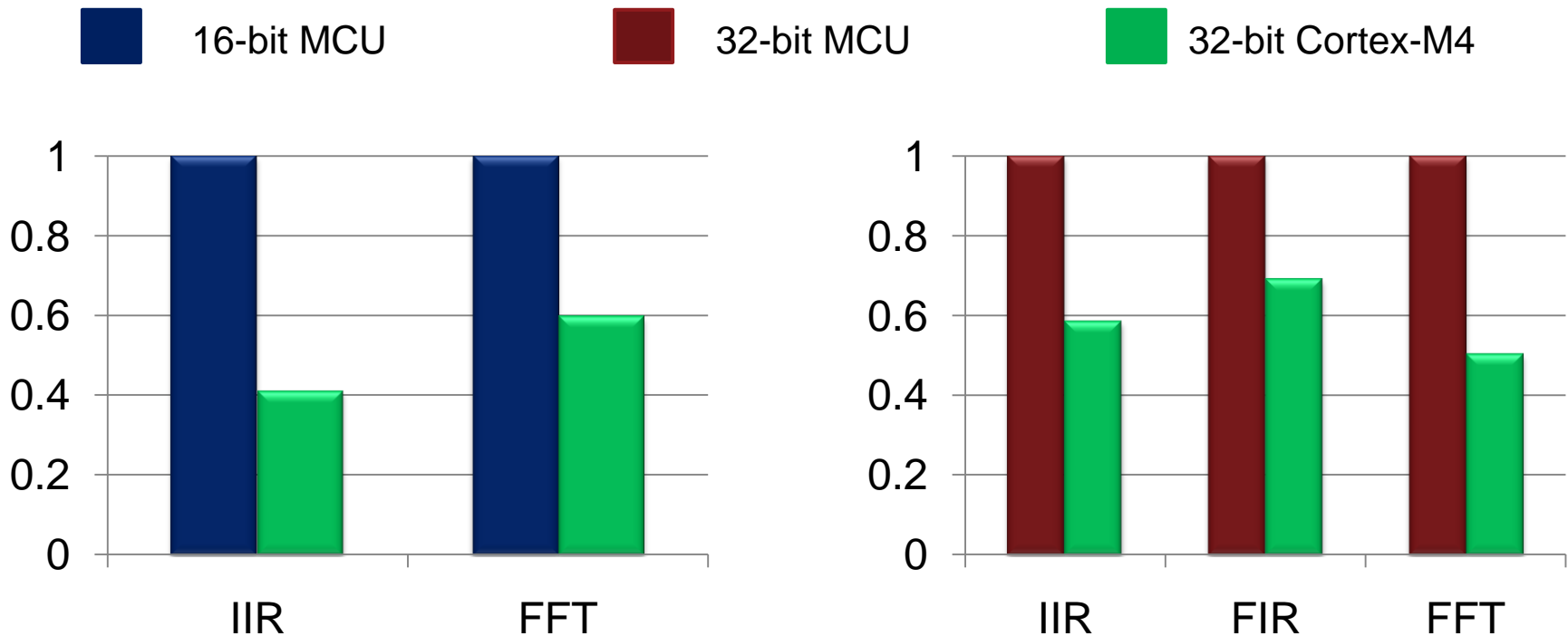
Brings high performance signal processing within the reach of the typical MCU programmer

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

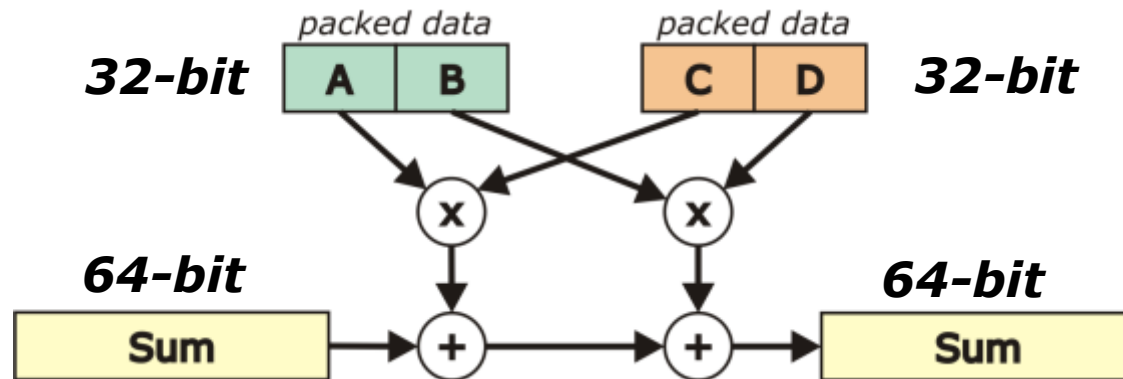


Cycle counts on DSP tasks compared, smaller is better

SIMD Operations

SIMD extensions perform multiple operations in one cycle

$$Sum = Sum + (A \times C) + (B \times D)$$



SIMD techniques operate with packed data

8,16-bit SIMD Arithmetic

Prefix Instr	S Signed	Q Signed Saturating	SH Signed Halving	U Unsigned	UQ Unsigned Saturating	UH Unsigned Halving
ADD8	SADD8	QADD8	SHADD8	USADD8	UQADD8	UHADD8
SUB8	SSUB8	QSUB8	SHSUB8	USUB8	UQSUB8	UHSUB8
ADD16	SADD16	QADD16	SHADD16	UADD16	UQADD16	UHADD16
SUB16	SSUB16	QSUB16	SHSUB16	USUB16	UQSUB16	UHSUB16
ASX	SASX	QASX	SHASX	UASX	UQASX	UHASX
SAX	SSAX	QSAX	SHSAX	USAX	UQSAX	UHSAX
USAD8	Unsigned Sum of Absolute Difference (8 bits)					
USADA8	Unsigned Sum of Absolute Difference and Accumulate (8 bits)					

ASX

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

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

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

- IIR Filter

$$y[n] = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] \\ + a_1 y[n-1] + a_2 y[n-2]$$

- FFT

$$Y[k_1] = X[k_1] + X[k_2] \\ Y[k_2] = X[k_1] - X[k_2] e^{-j\omega}$$

Most operations are dominated by MACs
These can be operations on 8, 16 or 32 bit data

Extended Single Cycle MAC

OPERATION	INSTRUCTIONS
$16 \times 16 = 32$	SMULBB, SMULBT, SMULTB, SMULTT
$16 \times 16 + 32 = 32$	SMLABB, SMLABT, SMLATB, SMLATT
$16 \times 16 + 64 = 64$	SMLALBB, SMLALBT, SMLALTB, SMLALTT
$16 \times 32 = 32$	SMULWB, SMULWT
$(16 \times 32) + 32 = 32$	SMLAWB, SMLAWT
$(16 \times 16) \quad (16 \times 16) = 32$	SMUAD, SMUADX, SMUSD, SMUSDX
$(16 \times 16) \quad (16 \times 16) + 32 = 32$	SMLAD, SMLADX, SMLSD, SMLSDX
$(16 \times 16) \quad (16 \times 16) + 64 = 64$	SMLALD, SMLALDX, SMLSLD, SMLSLDX
$32 \times 32 = 32$	MUL
$32 \quad (32 \times 32) = 32$	MLA, MLS
$32 \times 32 = 64$	SMULL, UMULL
$(32 \times 32) + 64 = 64$	SMLAL, UMLAL
$(32 \times 32) + 32 + 32 = 64$	UMAAL
$32 \quad (32 \times 32) = 32$ (upper)	SMMLA, SMMLAR, SMMLS, SMMLSR
$(32 \times 32) = 32$ (upper)	SMMUL, SMMULR

All the above operations are single cycle on the Cortex-M4 processor

DSP Instructions Compared

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

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

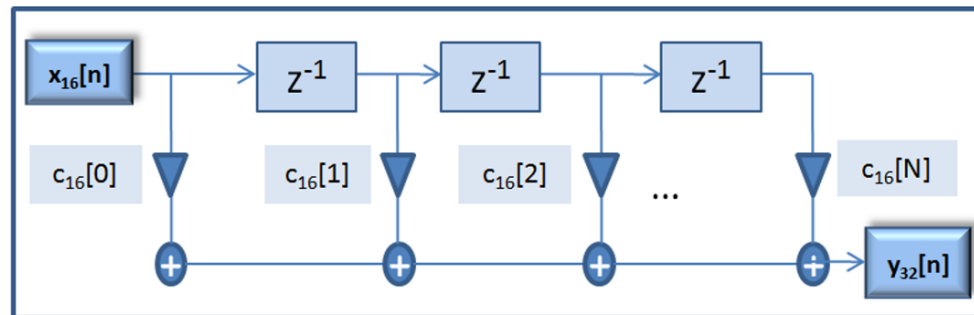
OPERATION	CYCLE COUNT
Add/Subtract	1
Divide	14
Multiply	1
Multiply Accumulate (MAC)	3
Fused MAC	3
Square Root	14

Code Example – Cortex-M4 FIR

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

y_{32} – filter output (32 bit); N - filter order

x_{16} – 16 bit input data; c_{16} – 16 bit filter coeff



Cortex-M3 Code Segment:

FIR_LOOP:

```

LDR    R2, [R0], #4    ;(2) Load input x16
LDR    R3, [R1], #4    ;(2) Load coeff c16
SXTH   R4, R2          ;(1) Extract x16[n-i]
ASR    R2, R2, #16     ;(1) Extract x16[n-i-1]
SXTH   R5, R3          ;(1) Extract c16[i]
ASR    R3, R3, #16     ;(1) Extract c16[i+1]
MLA    R6, R4, R5      ;(2) y32 += x16[n-i]*c16[i]
MLA    R6, R2, R3      ;(2) y32 += x16[n-i-1]*c16[i+1]
SUBS   R7, R7, #2      ;(1) loop count -= 2
BNE    FIR_LOOP       ;(2)
    
```

Note:

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

Cortex-M4 Code Segment:

FIR_LOOP:

```

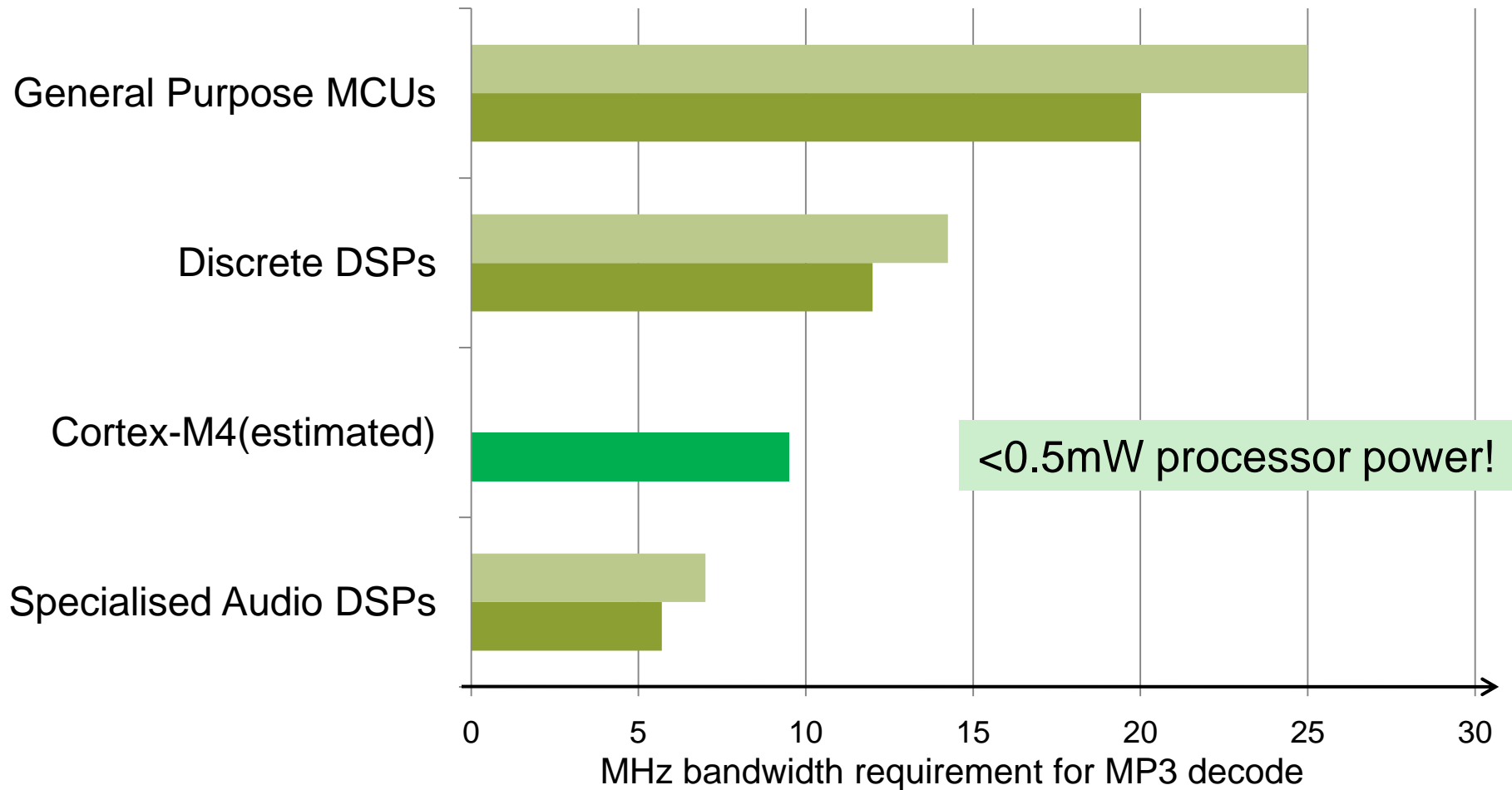
LDR    R2, [R0], #4    ;(1) Load input x16[n-i], x16[n-i-1]
LDR    R3, [R1], #4    ;(2) Load coeff c16[i], c16[i+1]
SUBS   R5, R5, #2      ;(1) loop count -= 2
SMLAD  R4, R2, R3      ;(1) y32 += x16[n-i, n-i-1]*c16[i, i+1]
BNE    FIR_LOOP       ;(2)
    
```

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

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



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

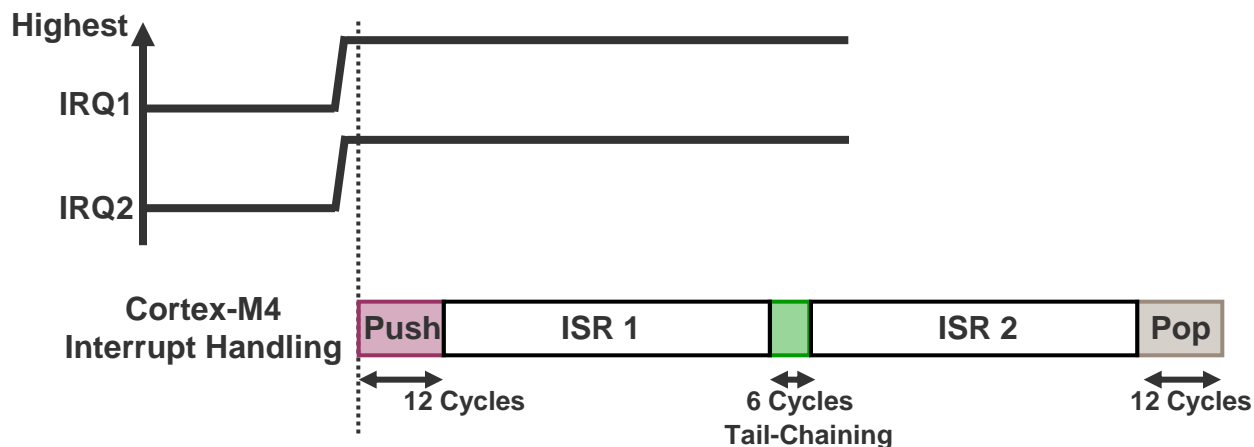
Nested Vector Interrupt Controller

- **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]

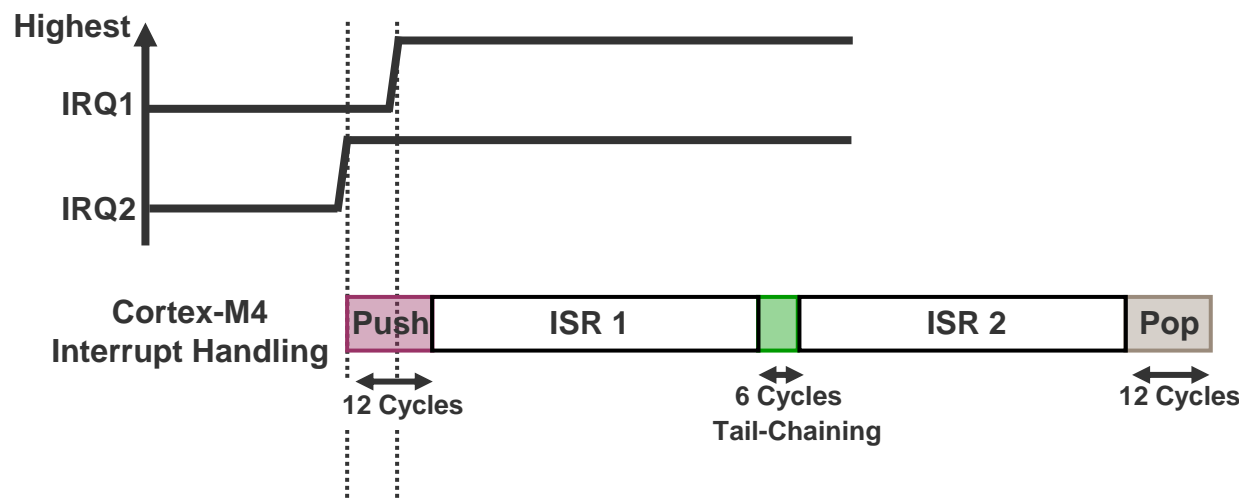
Tail chaining

- 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

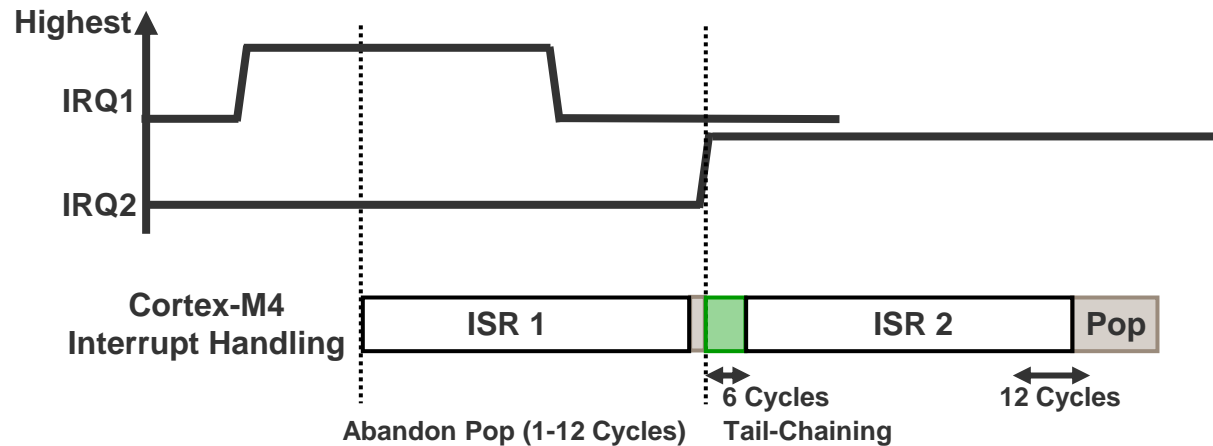
- 12 cycles to ISR entry
- Parallel stacking & inst. fetch
- Target ISR may be changed until last cycle
- When IRQ1 occurs new target ISR set



Cortex-M4 Interrupt Response [2]

Pop pre-emption

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



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



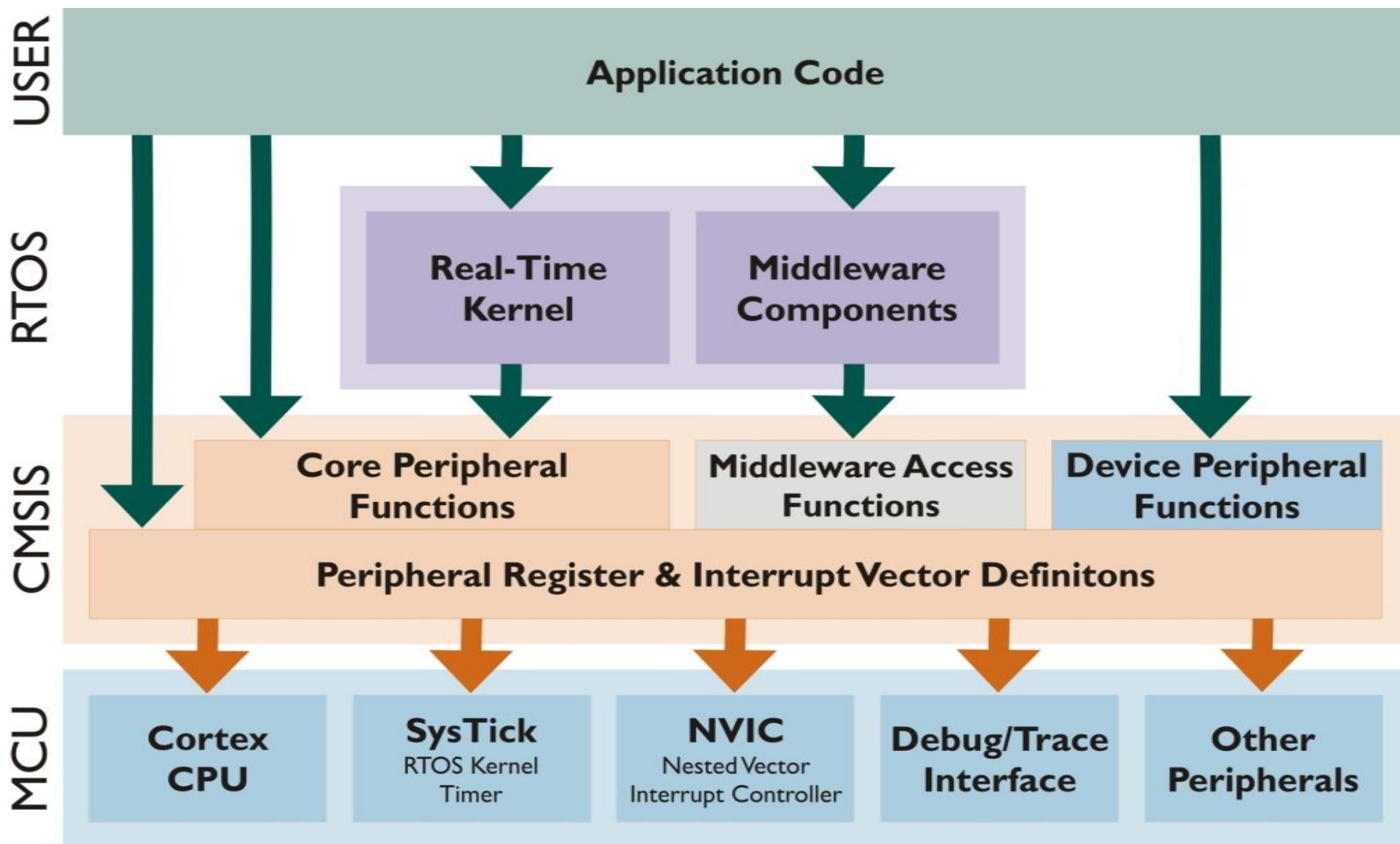
TEXAS
INSTRUMENTS



TOSHIBA



CMSIS - Structure



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:

- 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 >

