ARM Processors and Architectures

A Comprehensive Overview
ARM University Program
September 2012

Agenda

- **Introduction**

  ARM Architecture Overview

  ARMv7-AR Architecture

  - Programmer’s Model
  - Memory Systems

  ARMv7-M Architecture

  - Programmer’s Model
  - Memory Systems
  - Floating Point Extensions

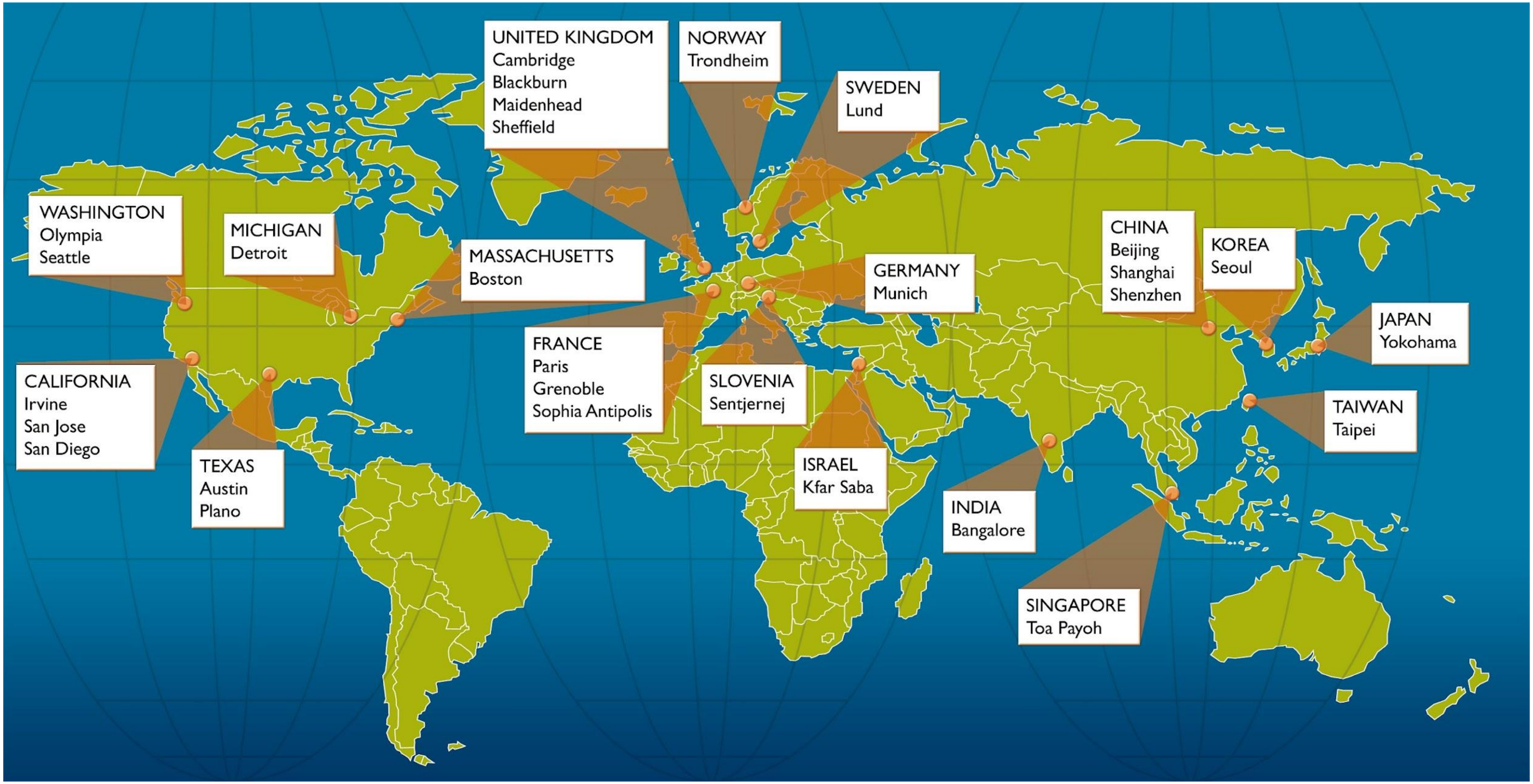
  ARM System Design

  Software Development Tools

ARM Ltd

- **ARM founded in November 1990**
  - Advanced RISC Machines
- **Company headquarters in Cambridge, UK**
  - Processor design centers in Cambridge, Austin, and Sophia Antipolis
  - Sales, support, and engineering offices all over the world
- **Best known for its range of RISC processor cores designs**
  - Other products – fabric IP, software tools, models, cell libraries - to help partners develop and ship ARM-based SoCs
- **ARM does *not* manufacture silicon**
- **More information about ARM and our offices on our web site:**
  - http://www.arm.com/aboutarm/

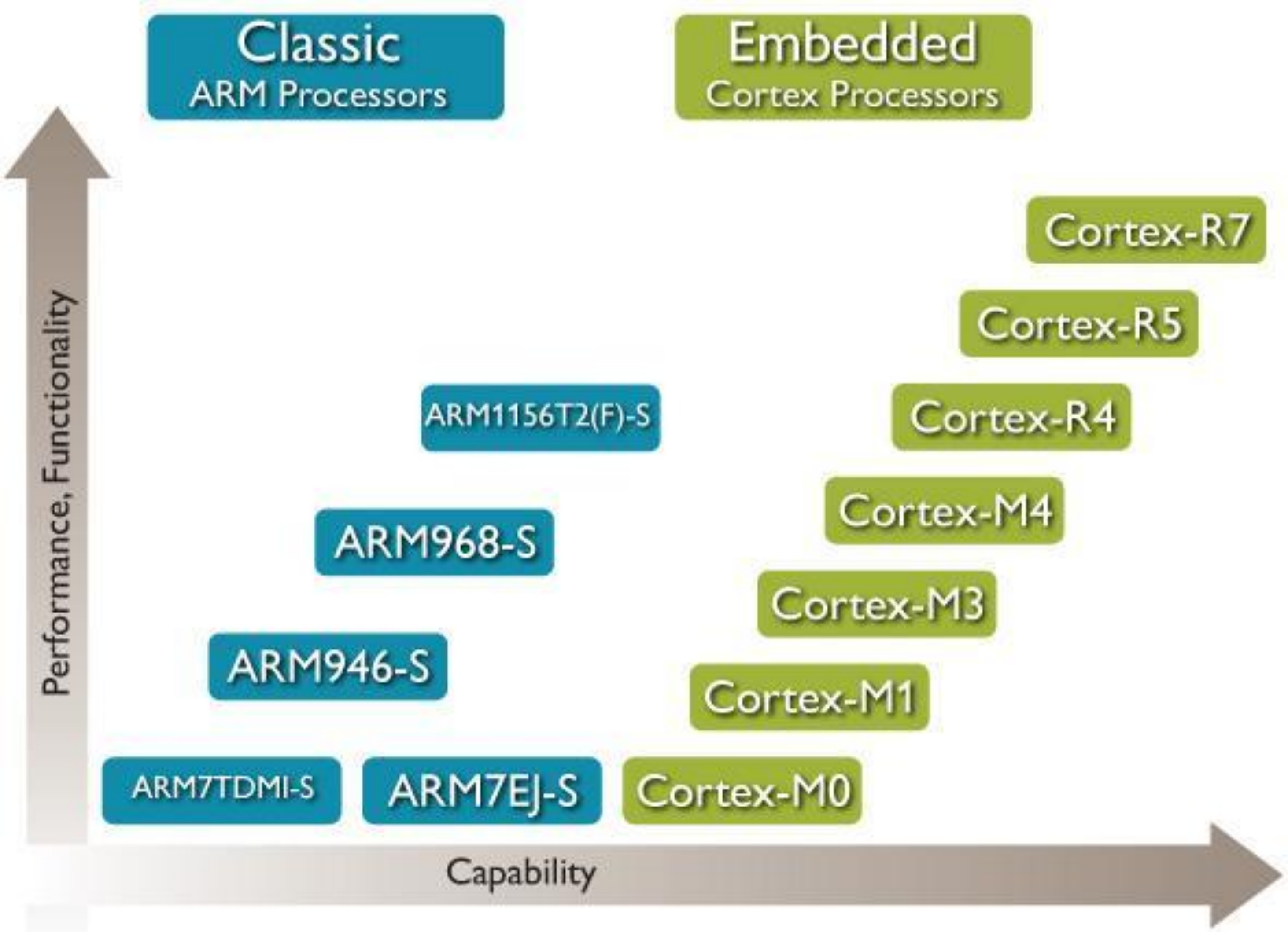
ARM Offices Worldwide



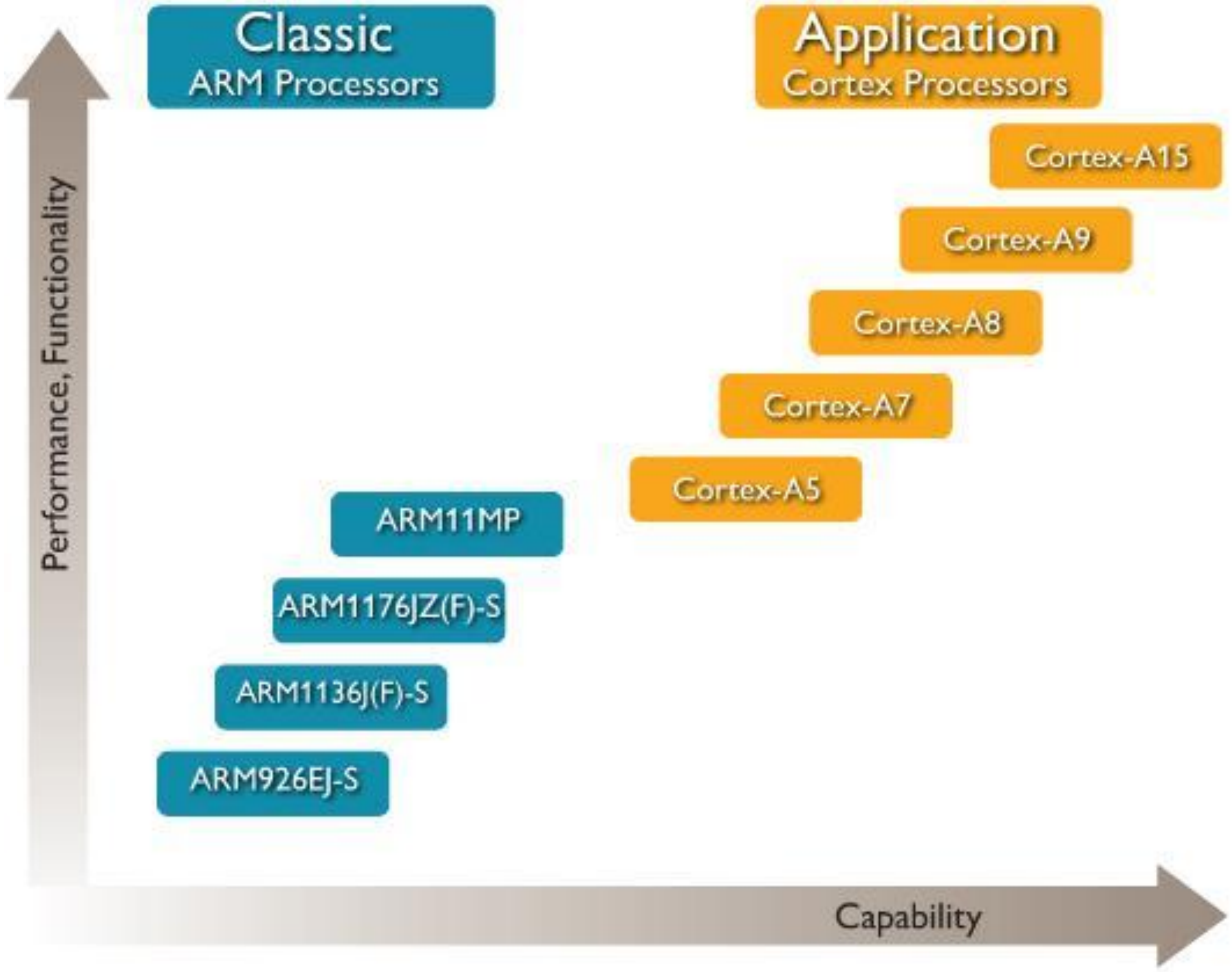
ARM Connected Community – 900+

Connect, Collaborate, Create – accelerating innovation

Embedded Processors



Application Processors



Agenda

Introduction

- ARM Architecture Overview

ARMv7-AR Architecture

Programmer's Model

Memory Systems

ARMv7-M Architecture

Programmer's Model

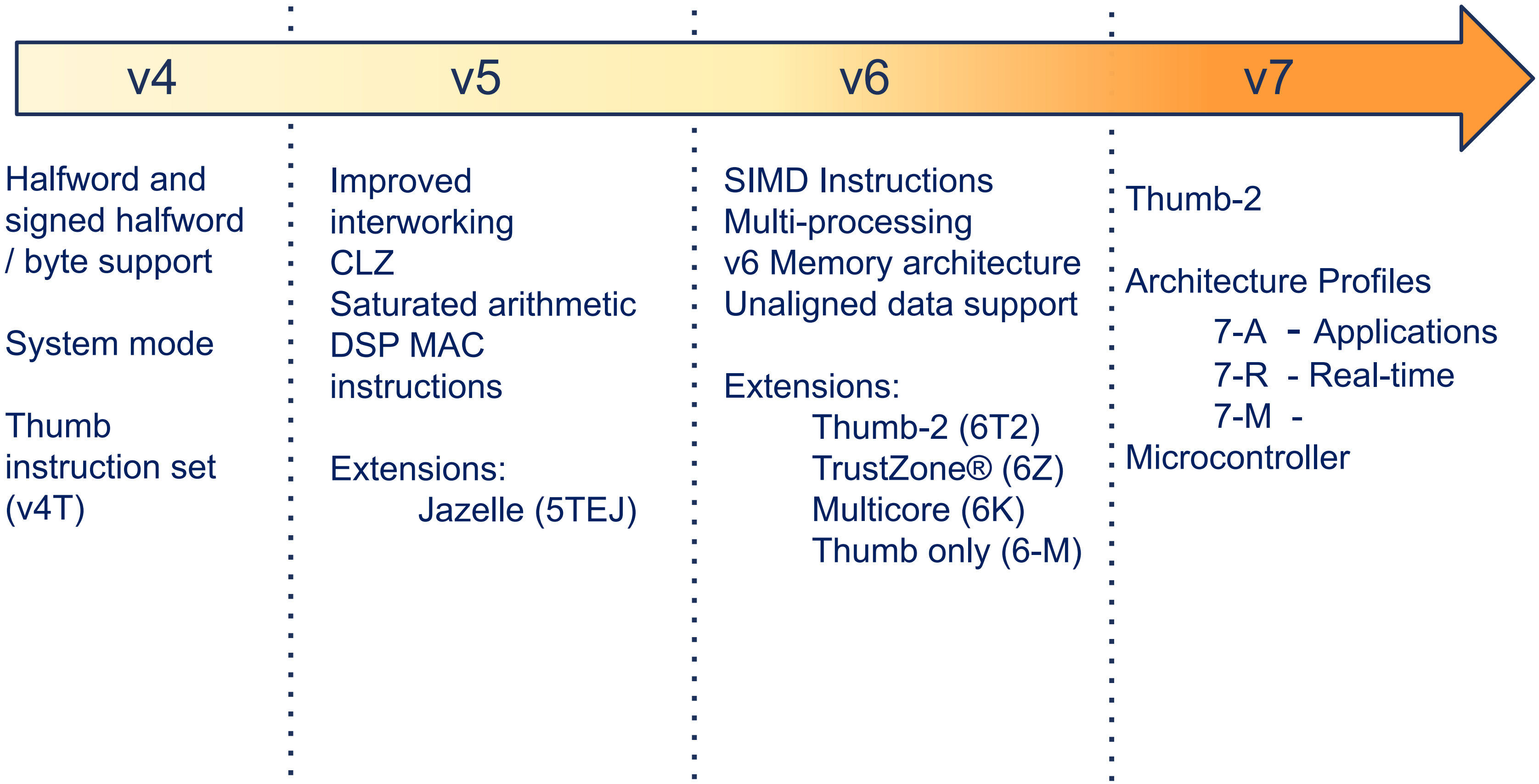
Memory Systems

Floating Point Extensions

ARM System Design

Software Development Tools

Development of the ARM Architecture



v4

Halfword and signed halfword / byte support

System mode

Thumb instruction set (v4T)

v5

Improved interworking
CLZ
Saturated arithmetic
DSP MAC instructions

Extensions:
Jazelle (5TEJ)

v6

SIMD Instructions
Multi-processing
v6 Memory architecture
Unaligned data support

Extensions:
Thumb-2 (6T2)
TrustZone® (6Z)
Multicore (6K)
Thumb only (6-M)

v7

Thumb-2

Architecture Profiles
7-A - Applications
7-R - Real-time
7-M - Microcontroller

- **Note that implementations of the same architecture can be different**
  - Cortex-A8 - architecture v7-A, with a 13-stage pipeline
  - Cortex-A9 - architecture v7-A, with an 8-stage pipeline

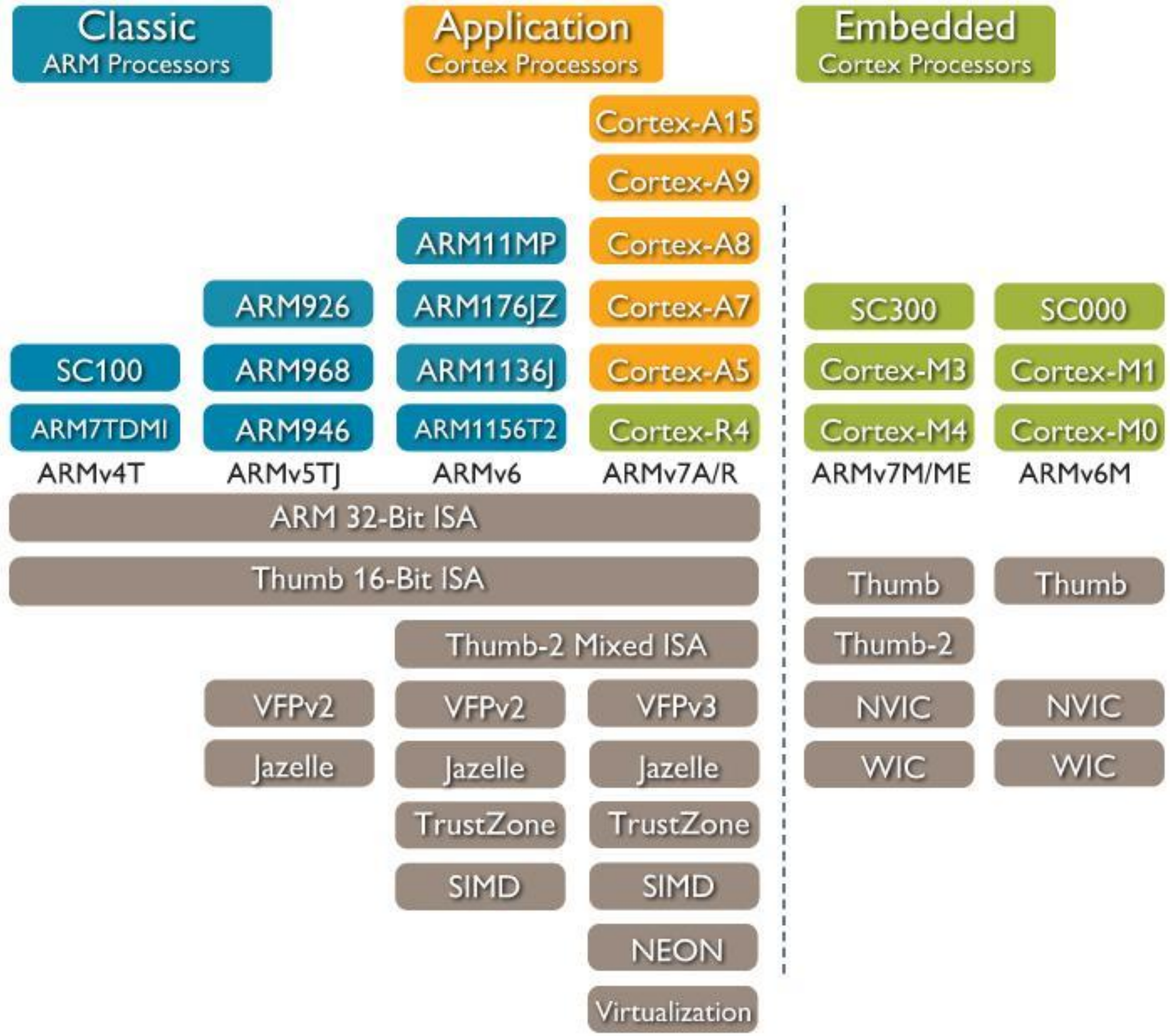
Architecture ARMv7 profiles

- **Application profile (ARMv7-A)**
  - Memory management support (MMU)
  - Highest performance at low power
    - Influenced by multi-tasking OS system requirements
  - TrustZone and Jazelle-RCT for a safe, extensible system
  - e.g. Cortex-A5, Cortex-A9

- **Real-time profile (ARMv7-R)**
  - Protected memory (MPU)
  - Low latency and predictability 'real-time' needs
  - Evolutionary path for traditional embedded business
  - e.g. Cortex-R4

- **Microcontroller profile (ARMv7-M, ARMv7E-M, ARMv6-M)**
  - Lowest gate count entry point
  - Deterministic and predictable behavior a key priority
  - Deeply embedded use
  - e.g. Cortex-M3

Which architecture is my processor?



Classic ARM Processors			Application Cortex Processors	Embedded Cortex Processors	
			Cortex-A15		
			Cortex-A9		
		ARM11MP	Cortex-A8		
	ARM926	ARM176JZ	Cortex-A7	SC300	SC000
SC100	ARM968	ARM1136J	Cortex-A5	Cortex-M3	Cortex-M1
ARM7TDMI	ARM946	ARM1156T2	Cortex-R4	Cortex-M4	Cortex-M0
ARMv4T	ARMv5TJ	ARMv6	ARMv7A/R	ARMv7M/ME	ARMv6M
ARM 32-Bit ISA					
Thumb 16-Bit ISA				Thumb	Thumb
		Thumb-2 Mixed ISA		Thumb-2	
	VFPv2	VFPv2	VFPv3	NVIC	NVIC
	Jazelle	Jazelle	Jazelle	WIC	WIC
		TrustZone	TrustZone		
		SIMD	SIMD		
			NEON		
			Virtualization		

Agenda

Introduction

ARM Architecture Overview

- **ARMv7-AR Architecture**
  - Programmer’s Model
  - Memory Systems

ARMv7-M Architecture

- Programmer’s Model
- Memory Systems
- Floating Point Extensions

ARM System Design

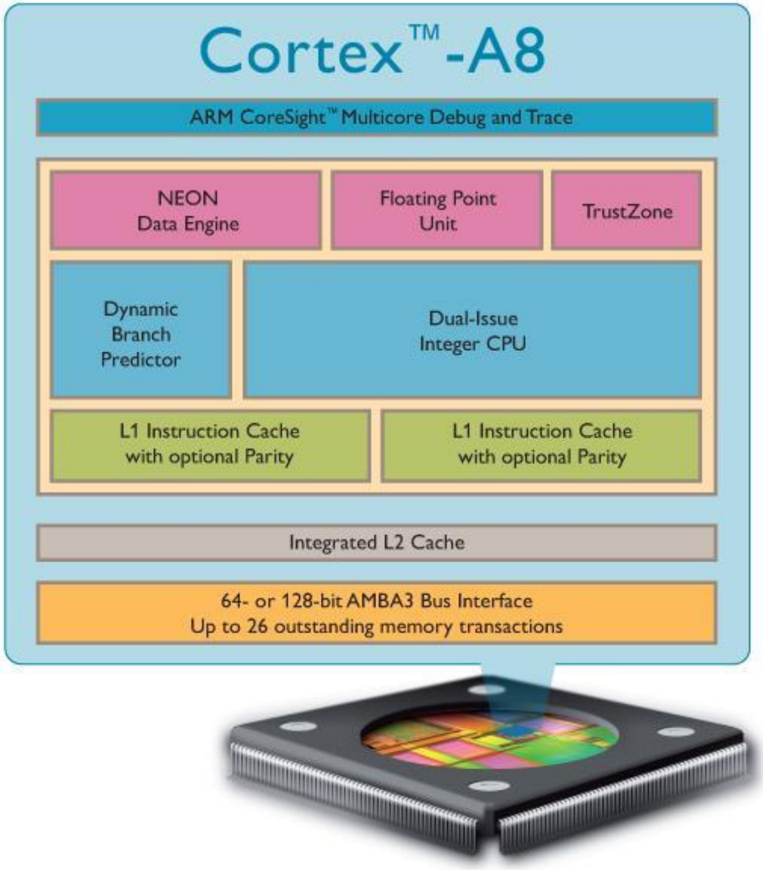
Software Development Tools

Architecture ARMv7-AR profiles

- **Application profile (ARMv7-A)**
  - Memory management support (MMU)
  - Highest performance at low power
  - Influenced by multi-tasking OS system requirements
  - e.g. Cortex-A5, Cortex-A8, Cortex-A9, Cortex-A15

- **Real-time profile (ARMv7-R)**
  - Protected memory (MPU)
  - Low latency and predictability 'real-time' needs
  - Evolutionary path for traditional embedded business
  - e.g. Cortex-R4, Cortex-R5

Cortex-A8

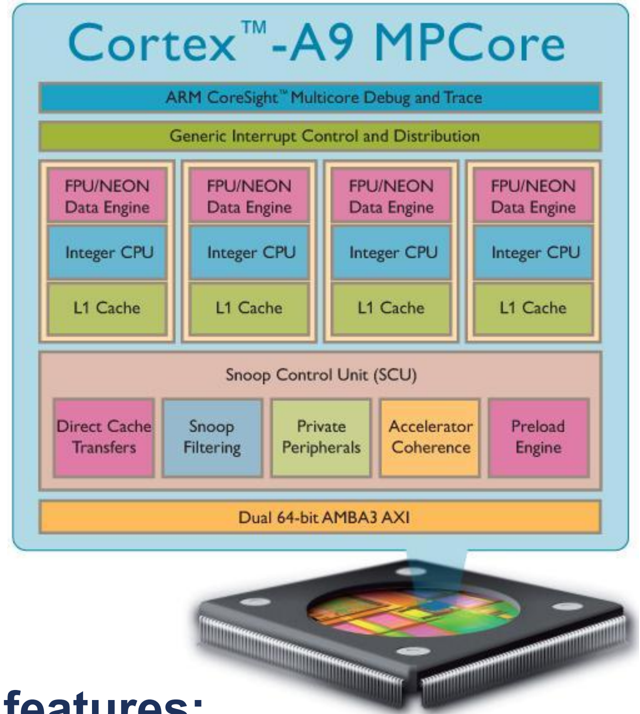

Cortex™-A8
ARM CoreSight™ Multicore Debug and Trace
NEON Data Engine
Floating Point Unit
TrustZone
Dynamic Branch Predictor
Dual-Issue Integer CPU
L1 Instruction Cache with optional Parity
L1 Instruction Cache with optional Parity
Integrated L2 Cache
64- or 128-bit AMBA3 Bus Interface
Up to 26 outstanding memory transactions

- **Dual-issue, super-scalar 13-stage pipeline**
  - Branch Prediction & Return Stack
  - NEON and VFP implemented at end of pipeline

- **ARMv7-A Architecture**
  - Thumb-2
  - Thumb-2EE (Jazelle-RCT)
  - TrustZone extensions
- **Custom or synthesized design**
- **MMU**
- **64-bit or 128-bit AXI Interface**
- **L1 caches**
  - 16 or 32KB each
- **Unified L2 cache**
  - 0-2MB in size
  - 8-way set-associative
- **Optional features**
  - VFPv3 Vector Floating-Point
  - NEON media processing engine

Cortex-A9

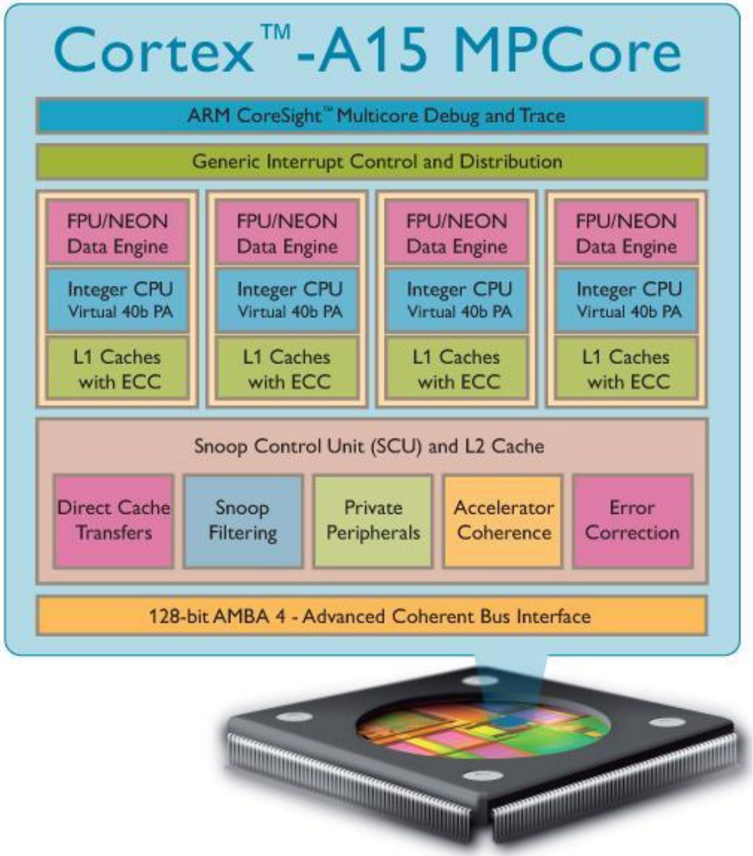
- **ARMv7-A Architecture**
  - Thumb-2, Thumb-2EE
  - TrustZone support
- **Variable-length Multi-issue pipeline**
  - Register renaming
  - Speculative data prefetching
  - Branch Prediction & Return Stack
- **64-bit AXI instruction and data interfaces**
- **TrustZone extensions**
- **L1 Data and Instruction caches**
  - 16-64KB each
  - 4-way set-associative



- **Optional features:**
  - PTM instruction trace interface
  - IEM power saving support
  - Full Jazelle DBX support
  - VFPv3-D16 Floating-Point Unit (FPU) or NEON™ media processing engine

Cortex-A15 MPCore

- 1-4 processors per cluster
- Fixed size L1 caches (32KB)
- Integrated L2 Cache
  - 512KB – 4MB
- System-wide coherency support with AMBA 4 ACE
- Backward-compatible with AXI3 interconnect
- Integrated Interrupt Controller
  - 0-224 external interrupts for entire cluster
- CoreSight debug
- Large Physical Address Extensions (LPAE) to ARMv7-A Architecture
- Advanced Power Management
- Virtualization Extensions to ARMv7-A Architecture



Cortex™-A15 MPCore

ARM CoreSight™ Multicore Debug and Trace

Generic Interrupt Control and Distribution

FPU/NEON Data Engine	FPU/NEON Data Engine	FPU/NEON Data Engine	FPU/NEON Data Engine
Integer CPU Virtual 40b PA	Integer CPU Virtual 40b PA	Integer CPU Virtual 40b PA	Integer CPU Virtual 40b PA
L1 Caches with ECC	L1 Caches with ECC	L1 Caches with ECC	L1 Caches with ECC

Snoop Control Unit (SCU) and L2 Cache

Direct Cache Transfers | Snoop Filtering | Private Peripherals | Accelerator Coherence | Error Correction

128-bit AMBA 4 - Advanced Coherent Bus Interface

Agenda

Introduction

ARM Architecture Overview

ARMv7-AR Architecture

- Programmer's Model

  Memory Systems

ARMv7-M Architecture

  Programmer's Model

  Memory Systems

  Floating Point Extensions

ARM System Design

Software Development Tools

Data Sizes and Instruction Sets

- **ARM is a 32-bit load / store RISC architecture**
  - The only memory accesses allowed are loads and stores
  - Most internal registers are 32 bits wide
  - Most instructions execute in a single cycle

- **When used in relation to ARM cores**
  - **Halfword** means 16 bits (two bytes)
  - **Word** means 32 bits (four bytes)
  - **Doubleword** means 64 bits (eight bytes)

- **ARM cores implement two basic instruction sets**
  - **ARM** instruction set – instructions are all 32 bits long
  - **Thumb** instruction set – instructions are a mix of 16 and 32 bits
    - Thumb-2 technology added many extra 32- and 16-bit instructions to the original 16-bit Thumb instruction set

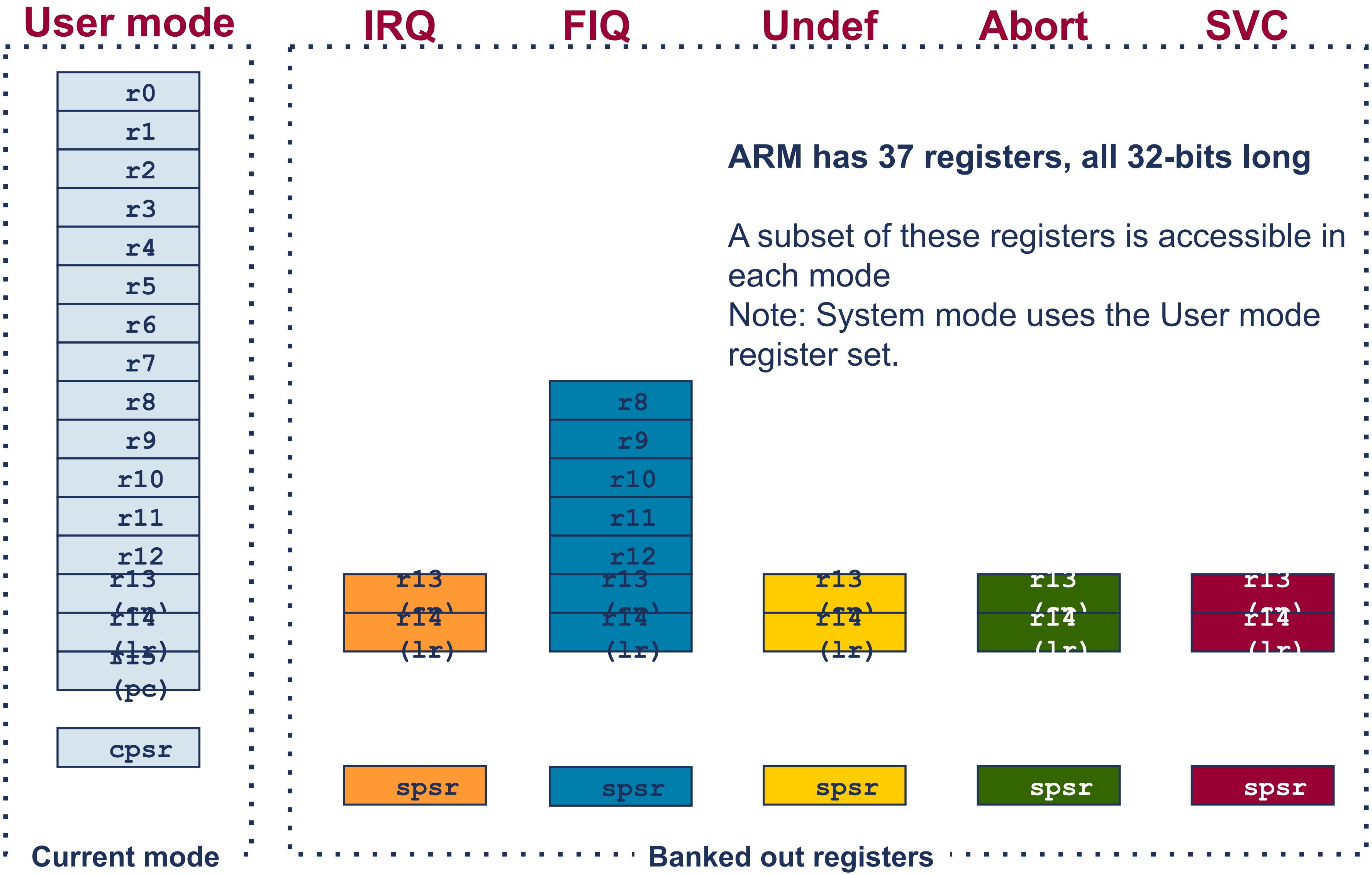
- **Depending on the core, may also implement other instruction sets**
  - **VFP** instruction set – 32 bit (vector) floating point instructions
  - **NEON** instruction set – 32 bit SIMD instructions
  - **Jazelle-DBX** – provides acceleration for Java VMs (with additional software support)
  - **Jazelle-RCT**

Processor Modes

- **ARM has seven basic operating modes**
  - Each mode has access to its own stack space and a different subset of registers
  - Some operations can only be carried out in a privileged mode

	Mode	Description	
Exception modes	**Supervisor (SVC)**	Entered on reset and when a Supervisor call instruction (SVC) is executed	**Privileged modes**
	FIQ	Entered when a high priority (fast) interrupt is raised	
	IRQ	Entered when a normal priority interrupt is raised	
	Abort	Used to handle memory access violations	
	Undef	Used to handle undefined instructions	
	System	Privileged mode using the same registers as User mode	
	User	Mode under which most Applications / OS tasks run	**Unprivileged mode**

The ARM Register Set



User mode	IRQ	FIQ	Undef	Abort	SVC
r0					
r1					
r2					
r3					
r4					
r5					
r6					
r7					
r8		r8			
r9		r9			
r10		r10			
r11		r11			
r12		r12			
r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)
r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)
r15 (pc)					
cpsr					
	spsr	spsr	spsr	spsr	spsr

Current mode

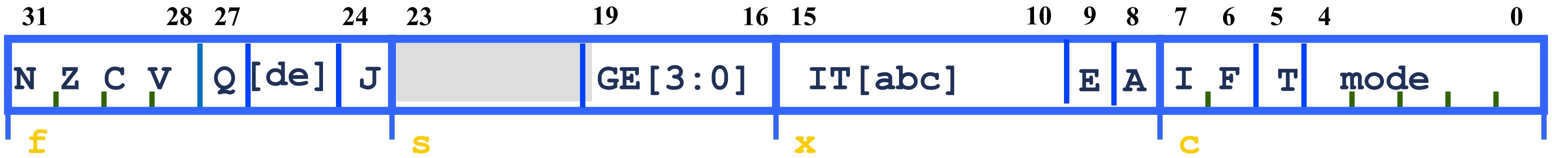
Banked out registers

ARM has 37 registers, all 32-bits long

A subset of these registers is accessible in each mode

Note: System mode uses the User mode register set.

Program Status Registers



- **Condition code flags**
  - N = **N**egative result from ALU
  - Z = **Z**ero result from ALU
  - C = ALU operation **C**arried out
  - V = ALU operation o**V**erflowed
- **Sticky Overflow flag - Q flag**
  - Indicates if saturation has occurred
- **SIMD Condition code bits – GE[3:0]**
  - Used by some SIMD instructions
- **IF THEN status bits – IT[abcde]**
  - Controls conditional execution of instructions

- **T bit**
  - T = 0: Processor in ARM state
  - T = 1: Processor in Thumb state
- **J bit**
  - J = 1: Processor in Jazelle state
- **Mode bits**
  - Specify the processor mode
- **Interrupt Disable bits**
  - I = 1: Disables IRQ
  - F = 1: Disables FIQ
- **E bit**
  - E = 0: Data load/store is little endian
  - E = 1: Data load/store is bigendian
- A bit

Instruction Set basics

- **The ARM Architecture is a Load/Store architecture**
  - No direct manipulation of memory contents
  - Memory must be loaded into the CPU to be modified, then written back out

- **Cores are either in ARM *state* or Thumb *state***
  - This determines which instruction set is being executed
  - An instruction must be executed to switch between states

- **The architecture allows programmers and compilation tools to reduce branching through the use of conditional execution**
  - Method differs between ARM and Thumb, but the principle is that most (ARM) or all (Thumb) instructions can be executed conditionally.

Data Processing Instructions

- **These instructions operate on the contents of registers**
  - They DO NOT affect memory

	arithmetic		logical		move
manipulation (has destination register)	ADC ADD	SBC SUB RSB RSC	BIC AND	ORR EOR ORN	MVN MOV
comparison (set flags only)	CMN (ADDS)	CMP (SUBS)	TST (ANDS)	TEQ (EORS)	

- **Syntax:**

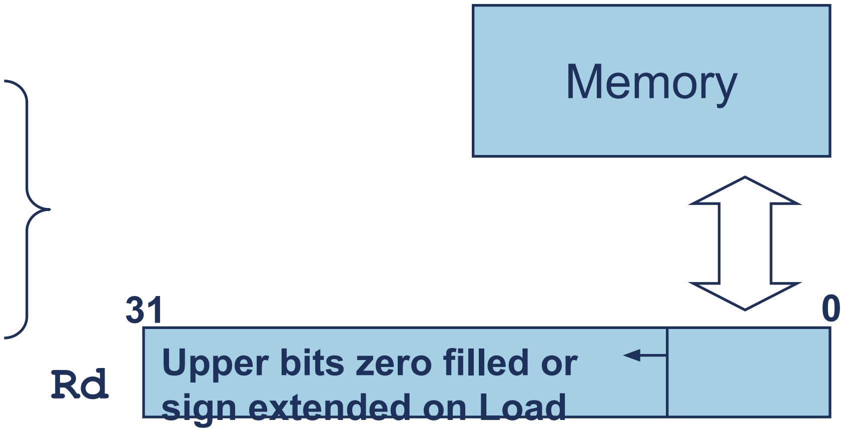
```
<Operation>{<cond>}{S} {Rd,} Rn, Operand2
```

- **Examples:**
  - `ADD r0, r1, r2   ; r0 = r1 + r2`
  - `TEQ r0, r1       ; if r0 = r1, Z flag will be set`
  - `MOV r0, r1       ; copy r1 to r0`

Single Access Data Transfer

- Use to move data between one or two registers and memory

LDRD	STRD	Doubleword
LDR	STR	Word
LDRB	STRB	Byte
LDRH	STRH	Halfword
LDRSB		Signed byte load
LDRSH		Signed halfword load



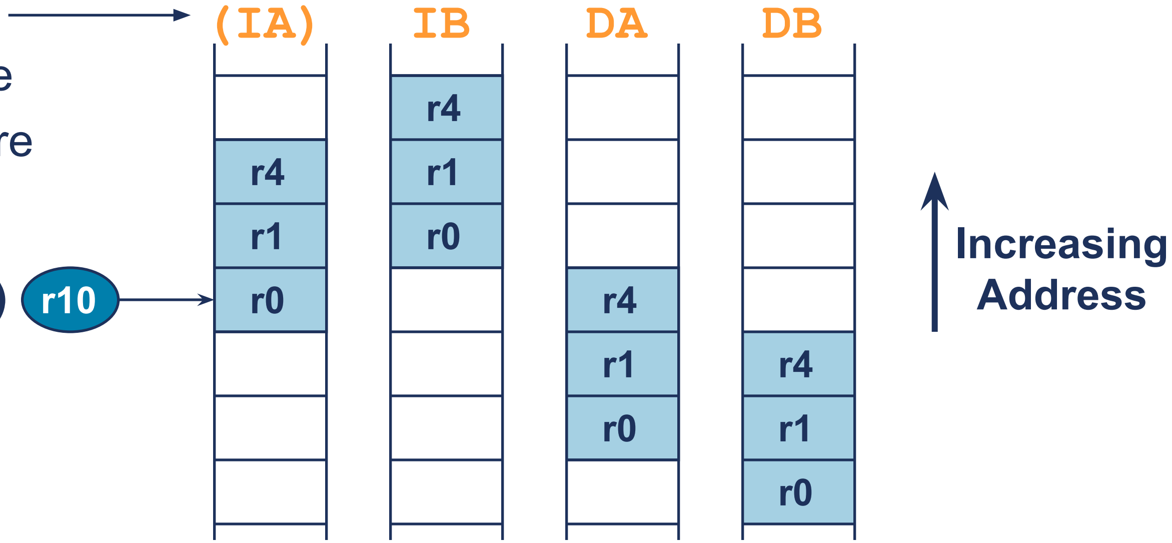
- Syntax:
  - `LDR{<size>}{<cond>} Rd, <address>`
  - `STR{<size>}{<cond>} Rd, <address>`

- Example:

  -
    ```
    LDRB r0, [r1]       ; load bottom byte of r0 from the
                        ; byte of memory at address in r1
    ```

Multiple Register Data Transfer

- These instructions move data between multiple registers and memory
- Syntax
  - `<LDM|STM>{<addressing_mode>}{<cond>} Rb{!}, <register list>`
- 4 addressing modes
  - Increment after/before
  - Decrement after/before

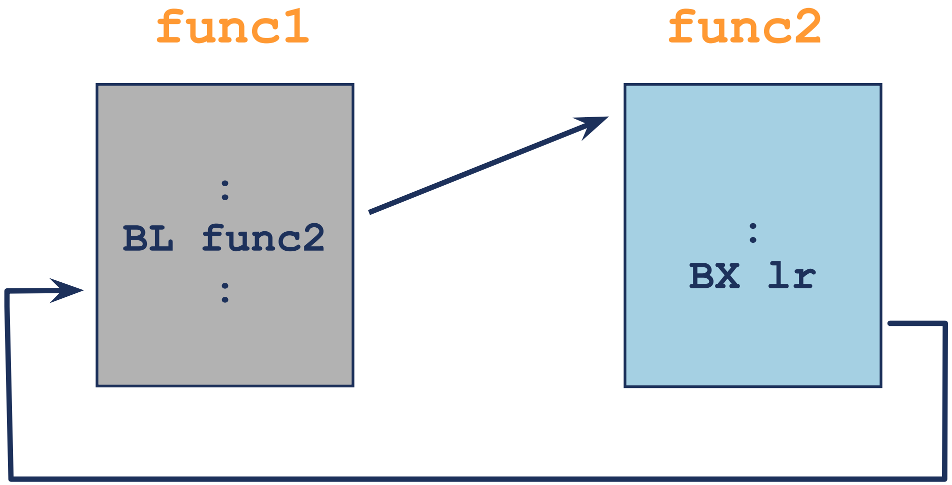


- Also
  - `PUSH/POP`, equivalent to `STMDB/LDMIA` with `SP!` as base register
- Example
  - `LDM     r10, {r0,r1,r4}  ; load registers, using r10 base`
  - `PUSH    {r4-r6,pc}       ; store registers, using SP base`

Subroutines

- **Implementing a conventional subroutine call requires two steps**
  - Store the return address
  - Branch to the address of the required subroutine
- **These steps are carried out in one instruction, `BL`**
  - The return address is stored in the link register (`lr/r14`)
  - Branch to an address (range dependent on instruction set and width)
- **Return is by branching to the address in `lr`**

```
void func1 (void)
{
    :
    func2();
    :
}
```



func1

:
BL func2
:

func2

:
BX lr

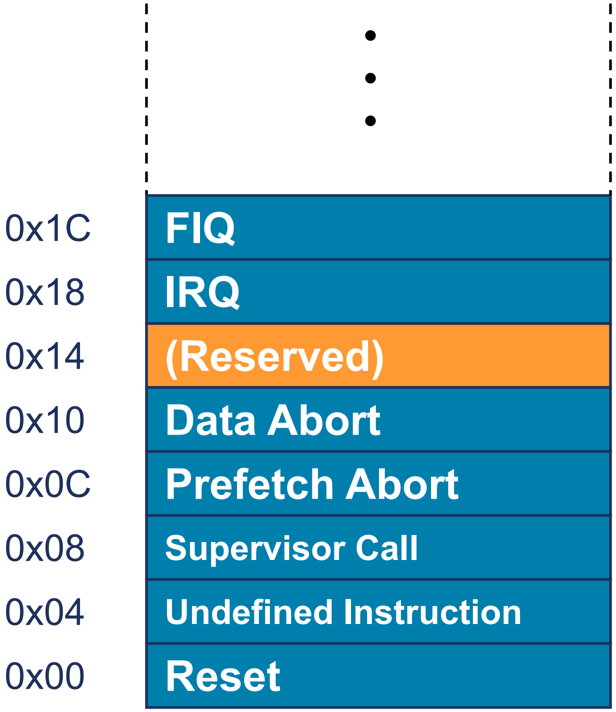
Supervisor Call (SVC)

```
SVC{<cond>} <SVC number>
```

- **Causes an SVC exception**
- **The SVC handler can examine the SVC number to decide what operation has been requested**
  - But the core ignores the SVC number
- **By using the SVC mechanism, an operating system can implement a set of privileged operations (system calls) which applications running in user mode can request**
- **Thumb version is unconditional**

Exception Handling

- **When an exception occurs, the core…**
  - Copies CPSR into SPSR_<mode>
  - Sets appropriate CPSR bits
    - Change to ARM state (if appropriate)
    - Change to exception mode
    - Disable interrupts (if appropriate)
  - Stores the return address in LR_<mode>
  - Sets PC to vector address
- **To return, exception handler needs to…**
  - Restore CPSR from SPSR_<mode>
  - Restore PC from LR_<mode>
- **Cores can enter ARM state or Thumb state when taking an exception**
  - Controlled through settings in CP15

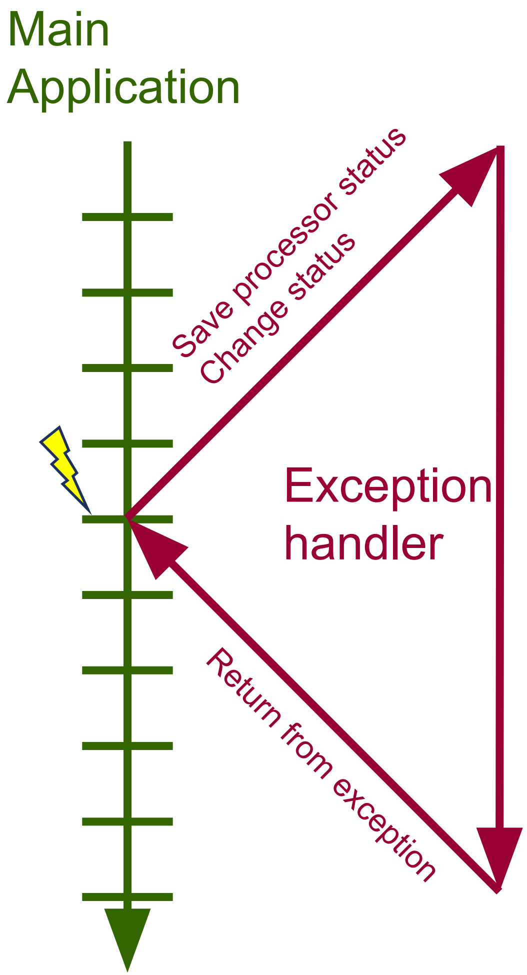


	⋮
0x1C	FIQ
0x18	IRQ
0x14	(Reserved)
0x10	Data Abort
0x0C	Prefetch Abort
0x08	Supervisor Call
0x04	Undefined Instruction
0x00	Reset

Vector Table

Vector table can also be at `0xFFFF0000` on most cores

Exception handling process

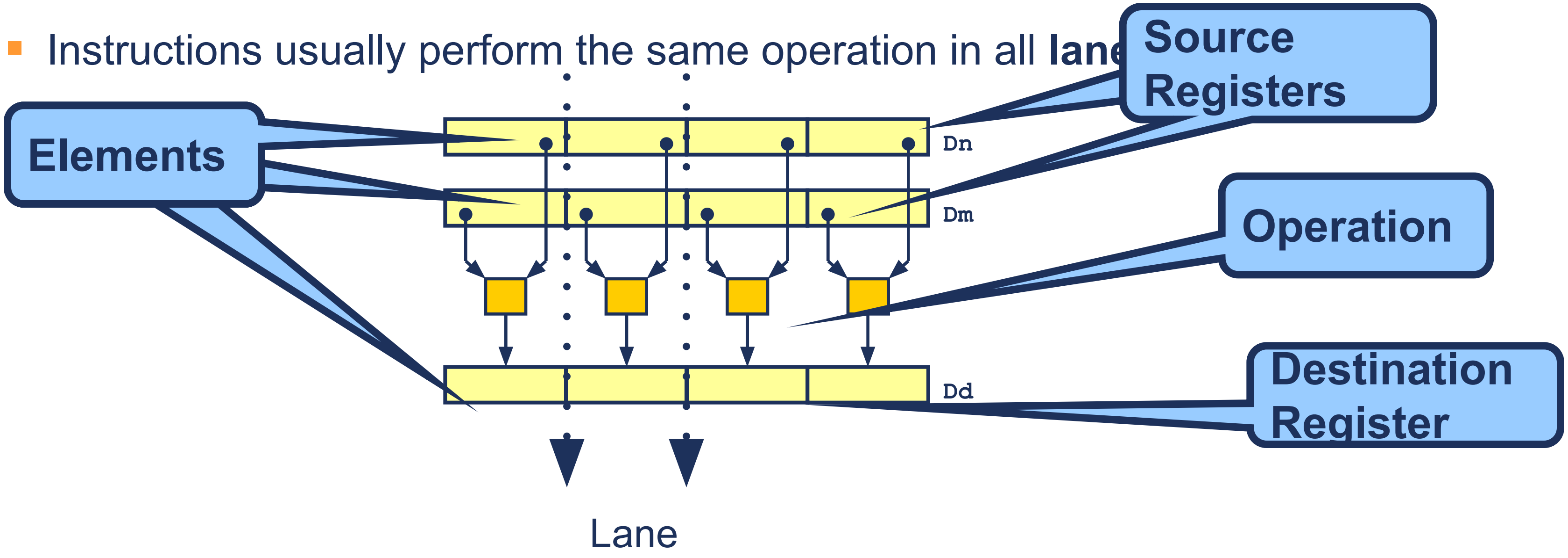


1. **Save processor status**
   - Copies `CPSR` into `SPSR_<mode>`
   - Stores the return address in `LR_<mode>`
   - Adjusts LR based on exception type
2. **Change processor status for exception**
   - Mode field bits
   - ARM or Thumb state
   - Interrupt disable bits (if appropriate)
   - Sets PC to vector address
3. **Execute exception handler**
   - <users code>
4. **Return to main application**
   - Restore `CPSR` from `SPSR_<mode>`
   - Restore PC from `LR_<mode>`

- **1 and 2 performed automatically by the core**

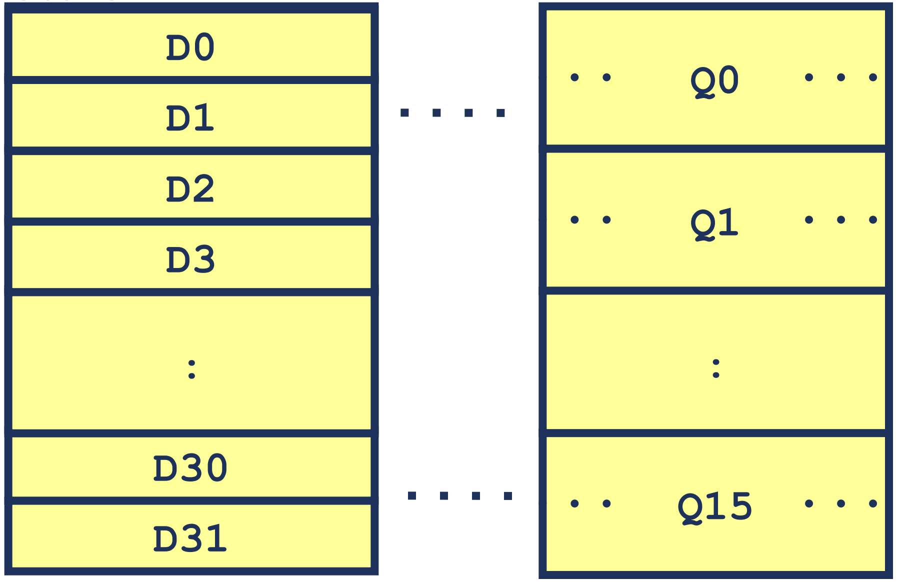
What is NEON?

- **NEON is a wide SIMD data processing architecture**
  - Extension of the ARM instruction set (v7-A)
  - 32 x 64-bit wide registers (can also be used as 16 x 128-bit wide registers)
- **NEON instructions perform “Packed SIMD” processing**
  - Registers are considered as **vectors** of **elements** of the same data type
  - Data types available: signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit, single prec. float
  - Instructions usually perform the same operation in all **lane**



NEON Coprocessor registers

- **NEON has a 256-byte register file**
  - Separate from the core registers (r0-r15)
  - Extension to the VFPv2 register file (VFPv3)
- **Two different views of the NEON registers**
  - 32 x 64-bit registers (D0-D31)
  - 16 x 128-bit registers (Q0-Q15)
- **Enables register trade-offs**
  - Vector length can be variable
  - Different registers available



NEON vectorizing example

- **How does the compiler perform vectorization?**

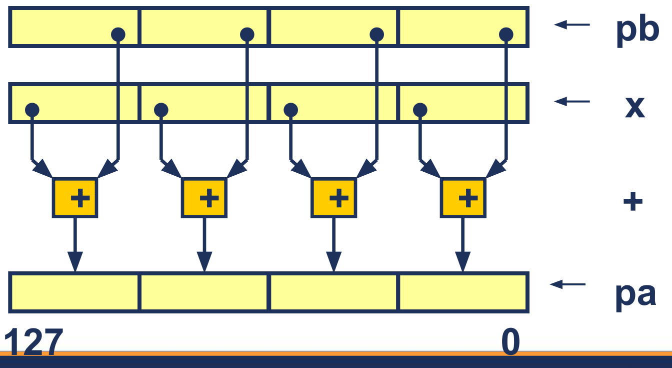
```
void add_int(int * __restrict pa,
             int * __restrict pb,
             unsigned int n, int x)
{
  unsigned int i;
  for(i = 0; i < (n & ~3); i++)
    pa[i] = pb[i] + x;
}
```

1. Analyze each loop:
   - Are pointer accesses safe for vectorization?
   - What data types are being used? How do they map onto NEON vector registers?
   - Number of loop iterations

2. Unroll the loop to the appropriate number of iterations, and perform other transformations like pointerization

```
void add_int(int *pa, int *pb,
             unsigned n, int x)
{
  unsigned int i;
  for (i = ((n & ~3) >> 2); i; i--)
  {
    *(pa + 0) = *(pb + 0) + x;
    *(pa + 1) = *(pb + 1) + x;
    *(pa + 2) = *(pb + 2) + x;
    *(pa + 3) = *(pb + 3) + x;
    pa += 4; pb += 4;
  }
}
```

3. Map each unrolled operation onto a NEON vector lane, and generate corresponding NEON instructions



Agenda

Introduction

ARM Architecture Overview

ARMv7-AR Architecture

- Programmer’s Model
- Memory Systems

ARMv7-M Architecture

- Programmer’s Model
- Memory Systems
- Floating Point Extensions

ARM System Design

Software Development Tools

Memory Types

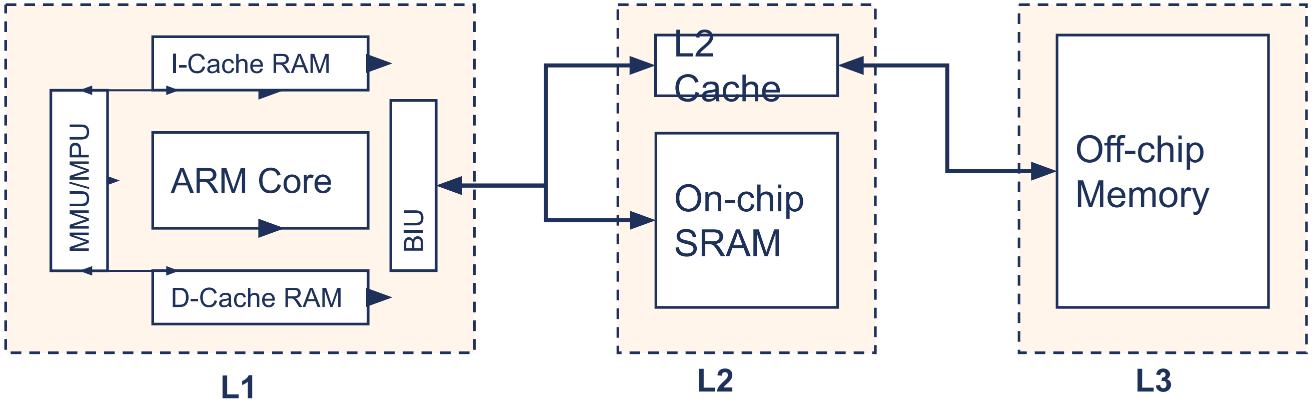
- **Each defined memory region will specify a memory type**

- **The memory type controls the following:**
  - Memory access ordering rules
  - Caching and buffering behaviour

- **There are 3 mutually exclusive memory types:**
  - Normal
  - Device
  - Strongly Ordered

- **Normal and Device memory allow additional attributes for specifying**
  - The cache policy
  - Whether the region is Shared
  - Normal memory allows you to separately configure Inner and Outer cache policies (discussed in the Caches and TCMs module)

L1 and L2 Caches

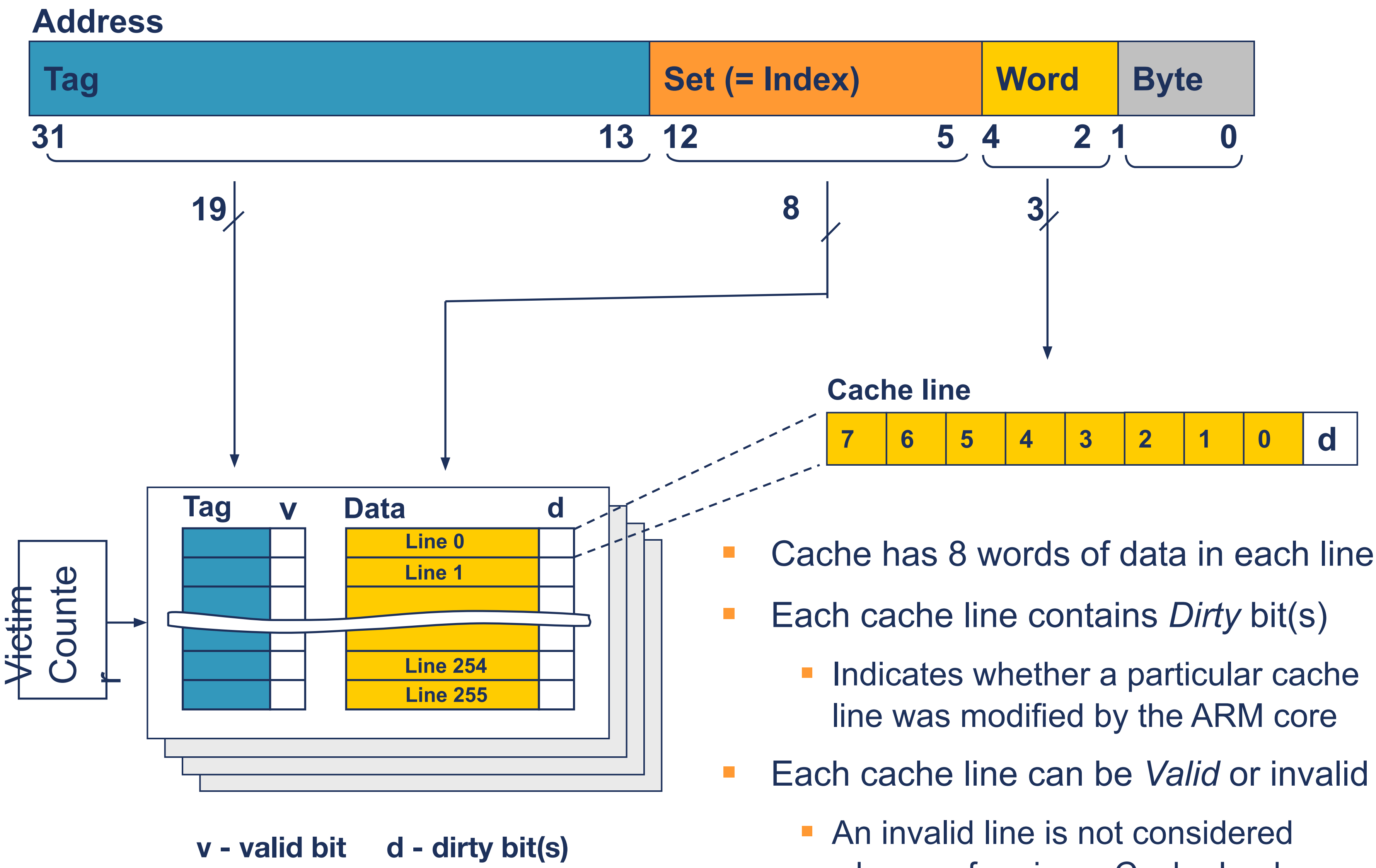


- **Typical memory system can have multiple levels of cache**
  - Level 1 memory system typically consists of L1-caches, MMU/MPU and TCMs
  - Level 2 memory system (and beyond) depends on the system design
- **Memory attributes determine cache behavior at different levels**
  - Controlled by the MMU/MPU (discussed later)
  - Inner Cacheable attributes define memory access behavior in the L1 memory system
  - Outer Cacheable attributes define memory access behavior in the L2 memory system (if external) and beyond (as signals on the bus)

ARM Cache Features

- **Harvard Implementation for L1 caches**
  - Separate Instruction and Data caches
- **Cache Lockdown**
  - Prevents line Eviction from a specified Cache Way (discussed later)
- **Pseudo-random and Round-robin replacement strategies**
  - Unused lines can be allocated before considering replacement
- **Non-blocking data cache**
  - Cache Lookup can hit before a Linefill is complete (also checks Linefill buffer)
- **Streaming, Critical-Word-First**
  - Cache data is forwarded to the core as soon as the requested word is received in the Linefill buffer
  - Any word in the cache line can be requested first using a 'WRAP' burst on the bus
- **ECC or parity checking**

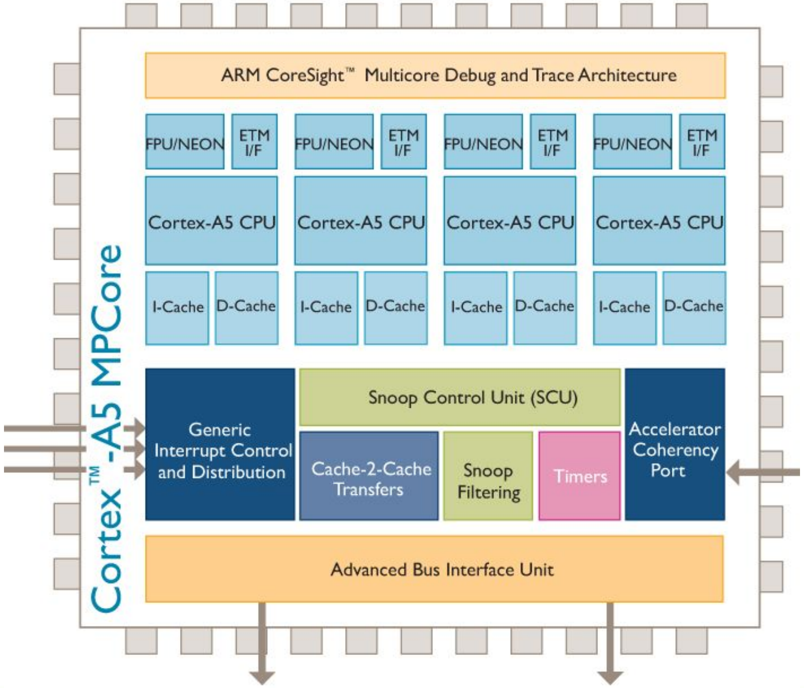
Example 32KB ARM cache

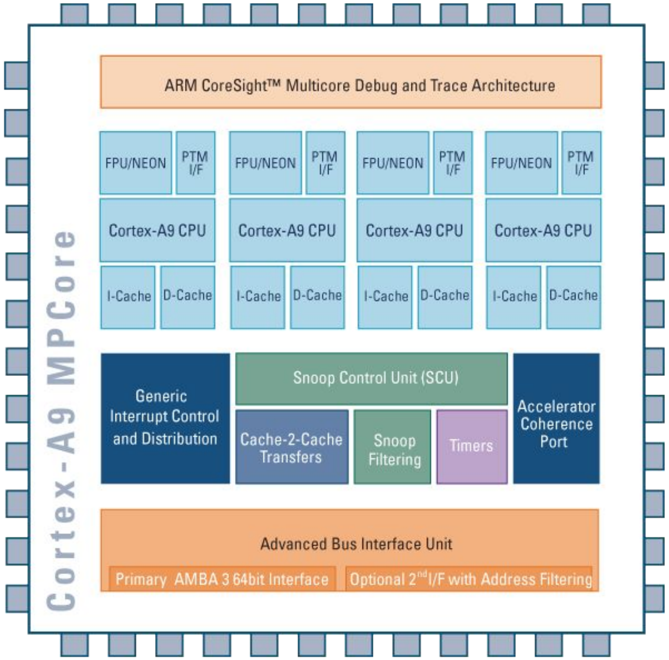


- Cache has 8 words of data in each line
- Each cache line contains *Dirty* bit(s)
  - Indicates whether a particular cache line was modified by the ARM core
- Each cache line can be *Valid* or invalid
  - An invalid line is not considered when performing a Cache Lookup

Cortex MPCore Processors

- **Standard Cortex cores, with additional logic to support MPCore**
  - Available as 1-4 CPU variants
- **Include integrated**
  - Interrupt controller
  - Snoop Control Unit (SCU)

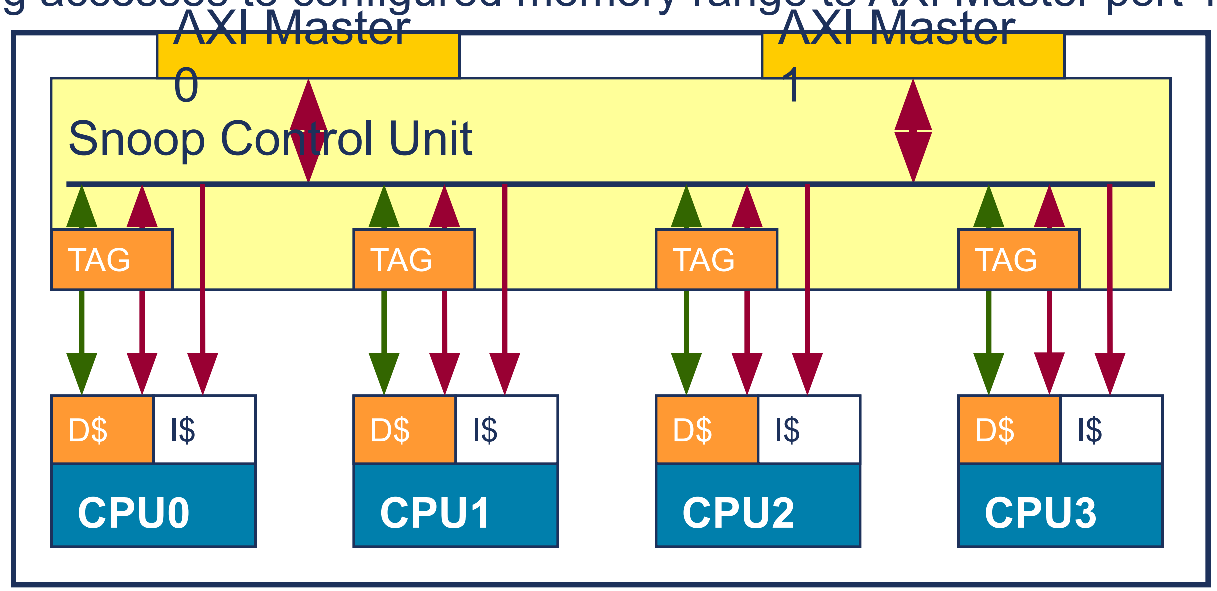




Snoop Control Unit

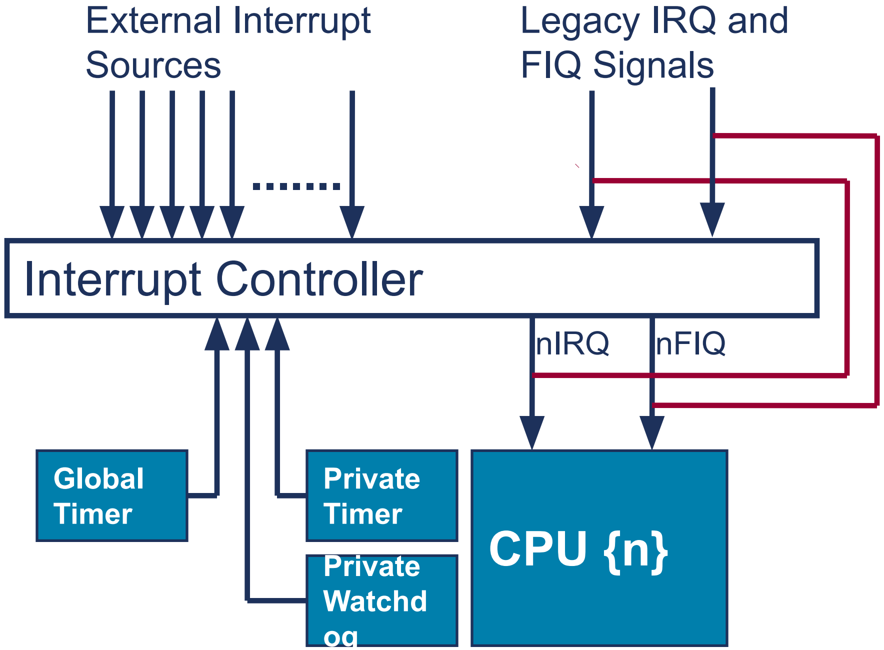
- **The Snoop Control Unit (SCU) maintains coherency between L1 data caches**
  - Duplicated Tag RAMs keep track of what data is allocated in each CPU’s cache
    - Separate interfaces into L1 data caches for coherency maintenance
  - Arbitrates accesses to L2 AXI master interface(s), for both instructions and data

- **Optionally, can use address filtering**
  - Directing accesses to configured memory range to AXI Master port 1



Interrupt Controller

- **MPCore processors include an integrated Interrupt Controller (IC)**
  - Implementation of the Generic Interrupt Controller (GIC) architecture
- **The IC provides:**
  - Configurable number of external interrupts (max 224)
  - Interrupt prioritization and pre-emption
  - Interrupt routing to different cores
- **Enabled per CPU**
  - When not enabled, that CPU will use legacy nIRQ[n] and nFIQ[n] signals



Agenda

Introduction

ARM Architecture Overview

ARMv7-AR Architecture

- Programmer’s Model
- Memory Systems

- **ARMv7-M Architecture**
  - Programmer’s Model
  - Memory Systems
  - Floating Point Extensions

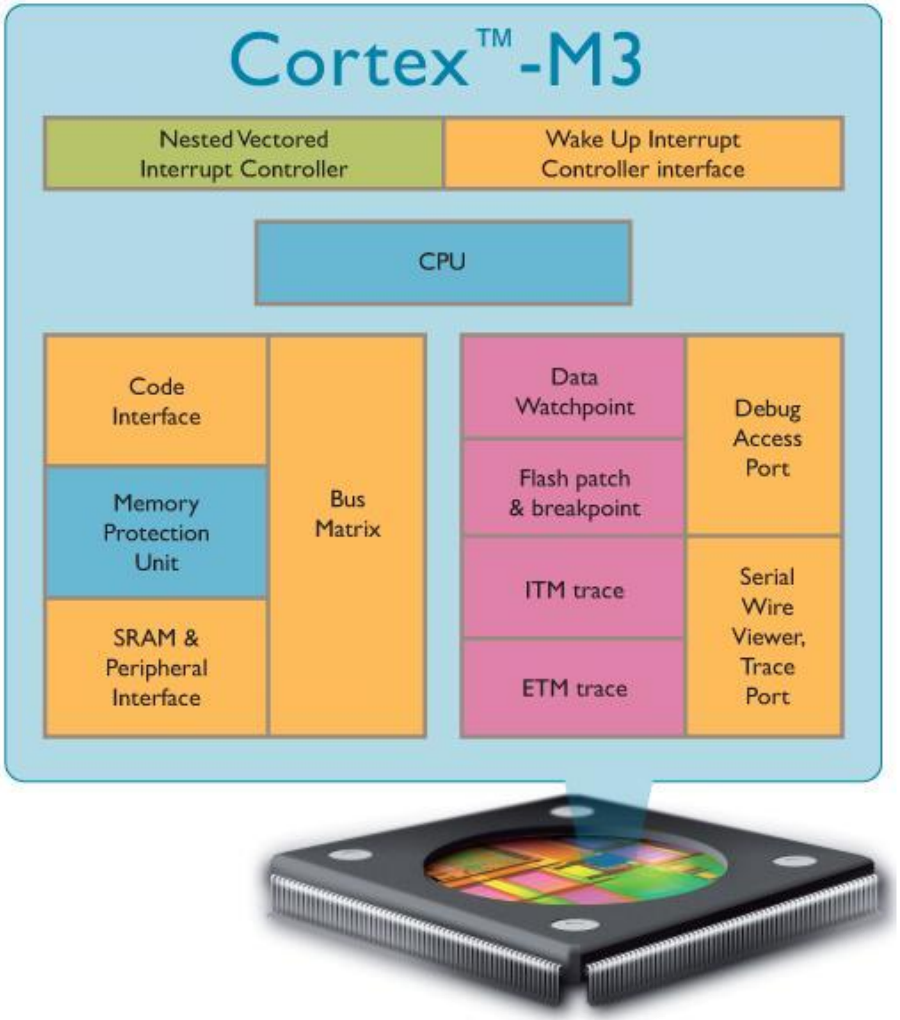
ARM System Design

Software Development Tools

ARMv7-M Profile Overview

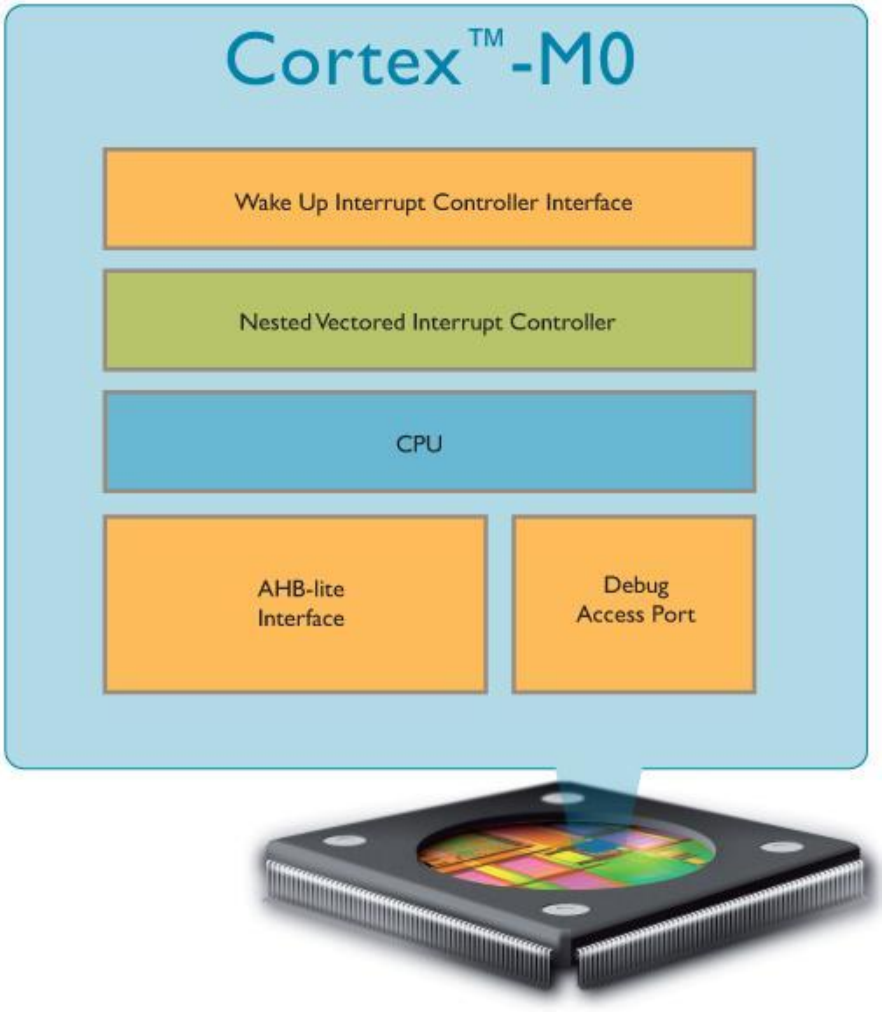
- **v7-M Cores are designed to support the microcontroller market**
  - Simpler to program – entire application can be programmed in C
  - Fewer features needed than in application processors
- **Register and ISA changes from other ARM cores**
  - No ARM instruction set support
  - Only one set of registers
  - xPSR has different bits than CPSR
- **Different modes and exception models**
  - Only two modes: Thread mode and Handler mode
  - Vector table is addresses, not instructions
  - Exceptions automatically save state (r0-r3, r12, lr, xPSR, pc) on the stack
- **Different system control/memory layout**
  - Cores have a fixed memory map

Cortex-M3



- **ARMv7-M Architecture**
  - Thumb-2 only
- **Fully programmable in C**
- **3-stage pipeline**
- **von Neumann architecture**
- **Optional MPU**
- **AHB-Lite bus interface**
- **Fixed memory map**
- **1-240 interrupts**
  - Configurable priority levels
  - Non-Maskable Interrupt support
  - Debug and Sleep control
- **Serial wire or JTAG debug**
- **Optional ETM**

Cortex-M0


Cortex™-M0
Wake Up Interrupt Controller Interface
Nested Vectored Interrupt Controller
CPU
AHB-lite Interface
Debug Access Port

- **ARMv6-M Architecture**
  - 16-bit Thumb-2 with system control instructions
- **Fully programmable in C**
- **3-stage pipeline**
- **von Neuman architecture**
- **AHB-Lite bus interface**
- **Fixed memory map**
- **1-32 interrupts**
  - Configurable priority levels
  - Non-Maskable Interrupt support
- **Low power support**
- **Core configured with or without debug**
  - Variable number of watchpoints and breakpoints

Agenda

Introduction

ARM Architecture Overview

ARMv7-AR Architecture

Programmer's Model

Memory Systems

ARMv7-M Architecture

- Programmer's Model

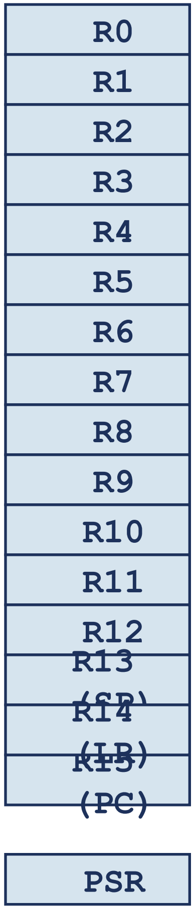
Memory Systems

Floating Point Extensions

ARM System Design

Software Development Tools

Processor Register Set



R0

R1

R2

R3

R4

R5

R6

R7

R8

R9

R10

R11

R12

R13 (SP)

R14 (LR)

R15 (PC)

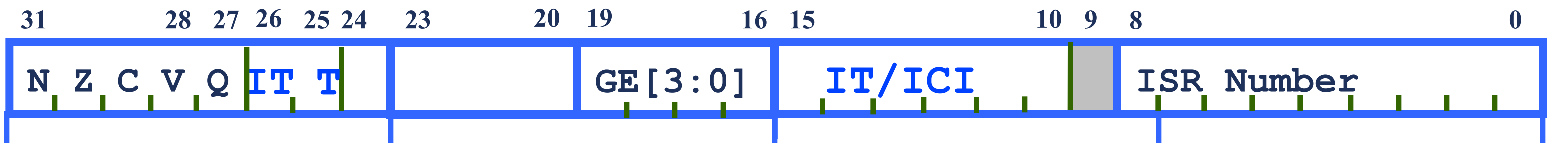
PSR

- **Registers R0-R12**
  - General-purpose registers
- **R13 is the stack pointer (SP) - 2 banked versions**
- **R14 is the link register (LR)**
- **R15 is the program counter (PC)**
- **PSR (Program Status Register)**
  - Not explicitly accessible
  - Saved to the stack on an exception
  - Subsets available as APSR, IPSR, and EPSR

Special Purpose Registers

- **Program Status Register**
  - Described in upcoming slides
- **Special Purpose Mask Registers : PRIMASK, FAULTMASK, BASEPRI**
  - Used to modify exception priorities
  - To set/clear PRIMASK and FAULTMASK, use CPS instructions
    - CPSIE i / CPSID i / CPSIE f / CPSID f
- **Special Purpose CONTROL Register**
  - 2 bits
    - Bit 0 defines Thread mode privilege
    - Bit 1 defines Thread mode stack
- **The Special Purpose Registers are not memory-mapped**
- **Accessed via specific instructions**
  - MRS – Move special purpose register to general-purpose register
  - MSR – Move general-purpose register to special purpose register

xPSR - Program Status Register



- **xPSR stored on stack during exceptions**
- **Condition code flags**
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation carry out
  - V = ALU operation overflow
  - Q = Saturated math overflow
- **IT/ICI bits**
  - Contain IF-THEN base condition code or Interrupt Continue information
- **ISR Number**
  - Stacked xPSR shows which exception was pre-empted
- **T=1**

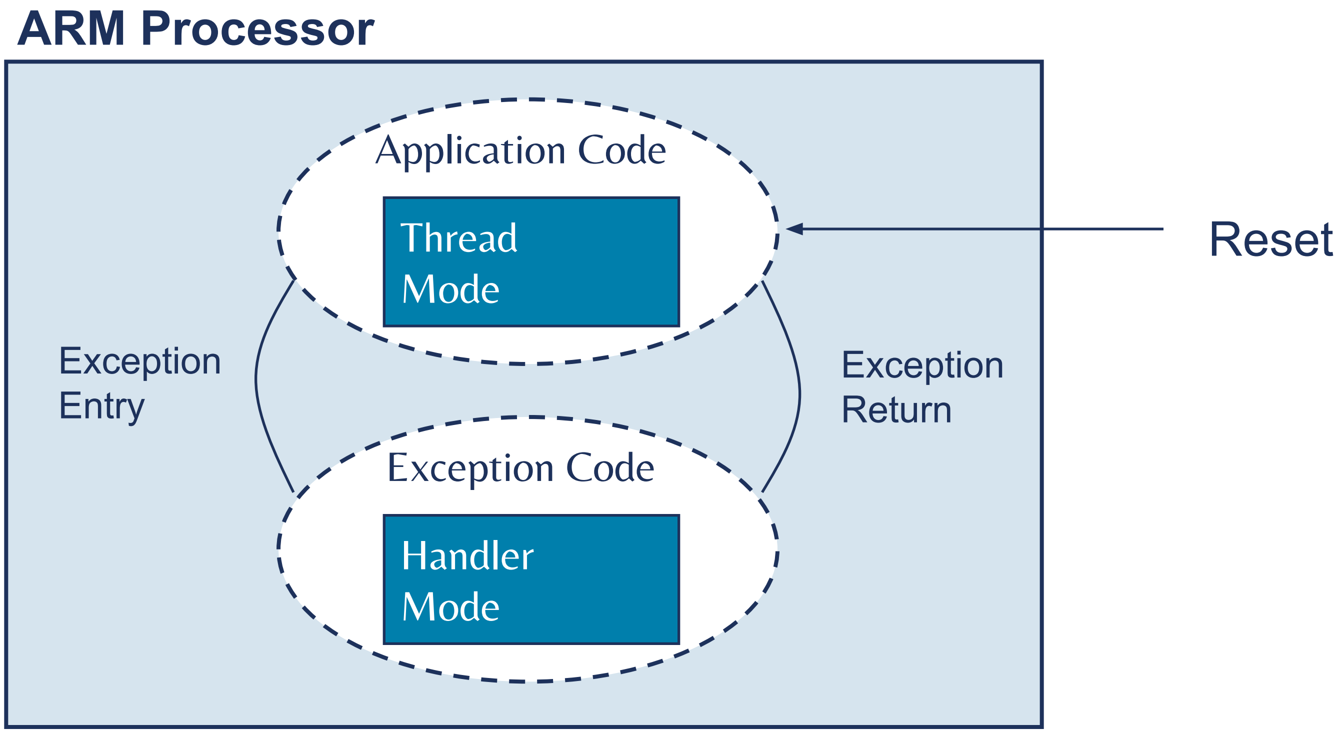
System Timer – SysTick

- **Flexible system timer**
  - 24-bit self-reloading down counter
    - Reload on count == 0
    - Optionally cause SysTick interrupt on count == 0
  - Reload register
  - Calibration value

- **Clock source is CPU clock or optional external timing reference**
  - Software selectable if provided
  - Reference pulse widths High/Low must exceed processor clock period
    - Counted by sampling on processor clock

- **Calibration Register provides value required for 10ms interval**
  - STCALIB inputs tied to appropriate value

Modes Overview



Not shown: Handler mode can also be re-entered on exception return

Instruction Set Examples:

- Data Processing:

```
MOV r2, r5                    ; r2 = r5
ADD r5, #0x24                 ; r5 = r5 + 36
ADD r2, r3, r4, LSL #2            ; r2 = r3 + (r4 * 4)
LSL r2, #3                    ; r2 = r2 * 8
MOVT r9, #0x1234                  ; upper halfword of r9 = #0x1234
MLA     r0, r1, r2, r3        ; r0 = (r1 * r2) + r3
```

- Memory Access:

```
STRB r2, [r10, r1]            ; store lower byte in r2 at
      address {r10 + r1}
LDR     r0, [r1, r2, LSL #2]    ; load r0 with data at address
                {r1 + r2 * 4}
```

- Program Flow:

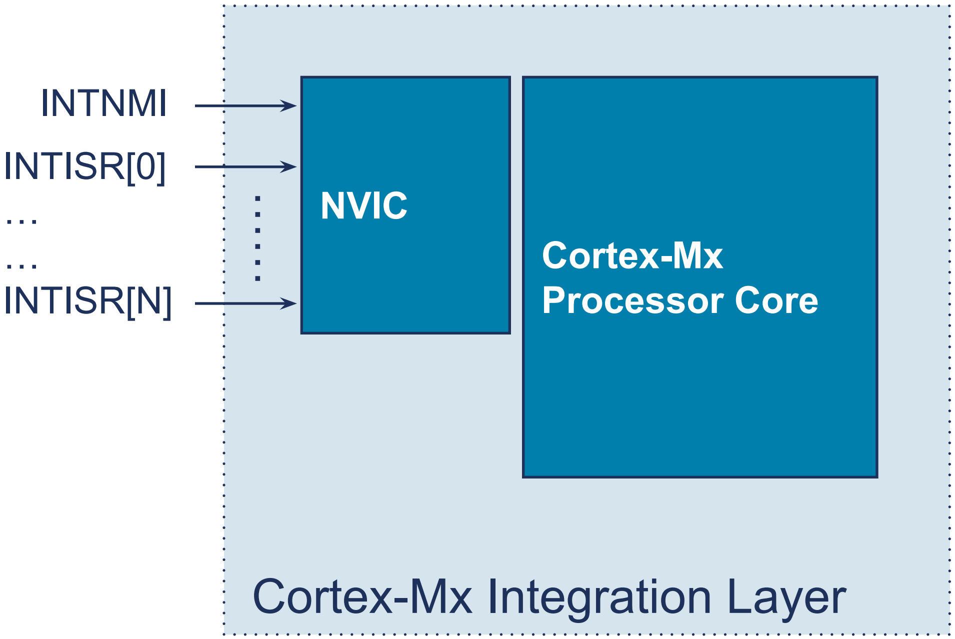
```
BL  <label>             ; PC relative branch to <label>
  location, and return address                        stored in LR
(r14)
```

Exception Handling

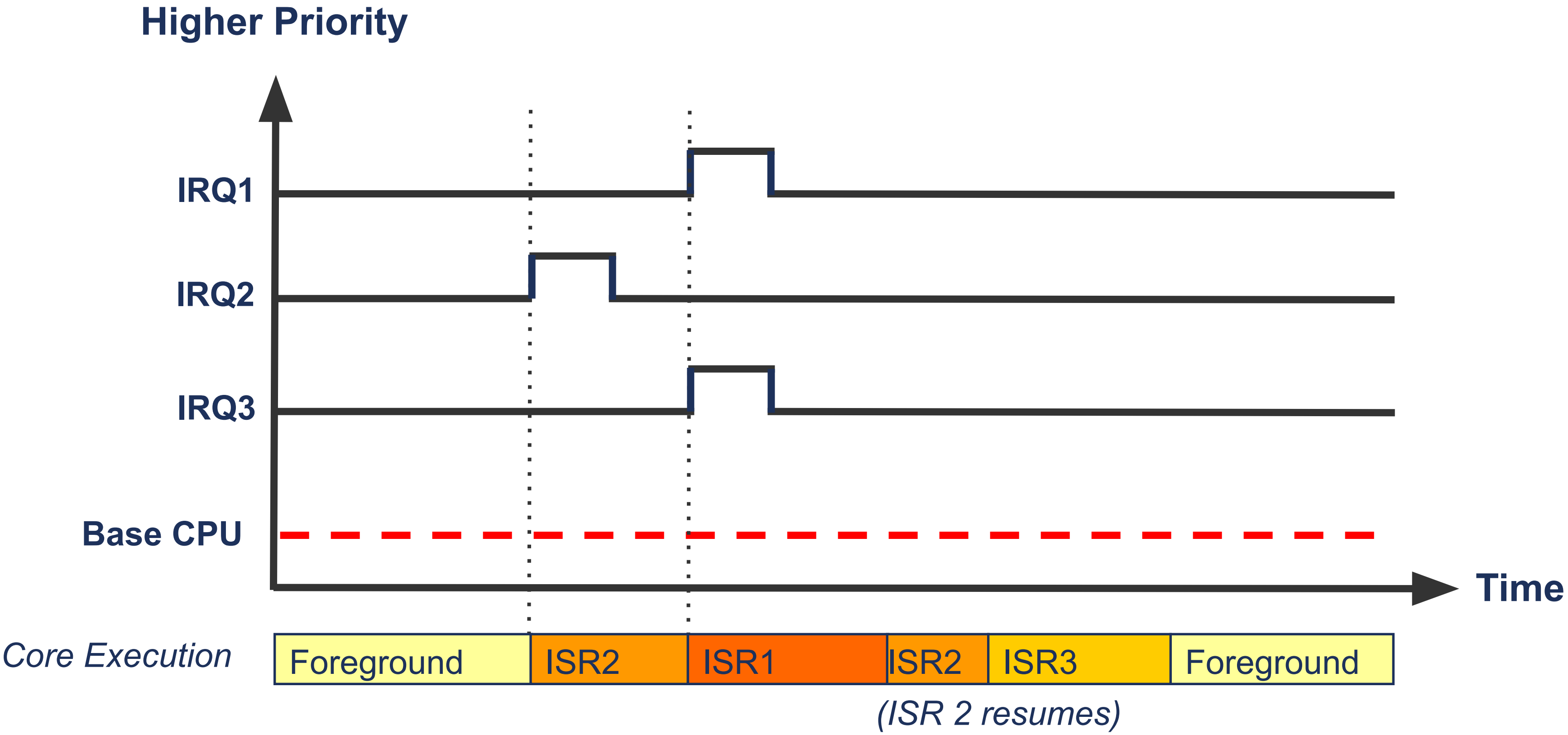
- **Exception types:**
  - Reset
  - Non-maskable Interrupts (NMI)
  - Faults
  - PendSV
  - SVCall
  - External Interrupt
  - SysTick Interrupt
- **Exceptions processed in Handler mode (except Reset)**
  - Exceptions always run privileged
- **Interrupt handling**
  - Interrupts are a sub-class of exception
  - Automatic save and restore of processor registers (xPSR, PC, LR, R12, R3-R0)
  - handler to be written entirely in C

External Interrupts

- **External Interrupts handled by Nested Vectored Interrupt Controller (NVIC)**
  - Tightly coupled with processor core
- **One Non-Maskable Interrupt (NMI) supported**
- **Number of external interrupts is implementation-defined**
  - ARMv7-M supports up to 496 interrupts



Exception Handling Example

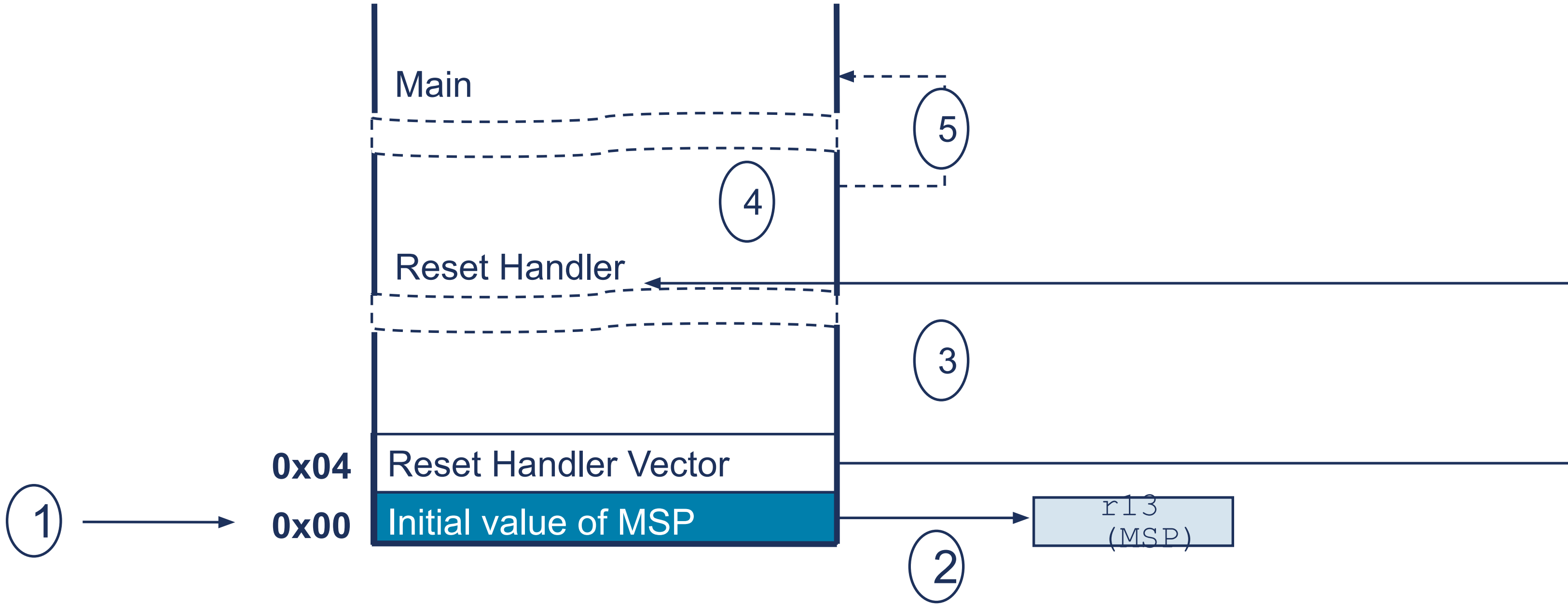


Vector Table for ARMv7-M

- **First entry contains initial Main SP**
- **All other entries are addresses for exception handlers**
  - Must always have LSBit = 1 (for Thumb)
- **Table has up to 496 external interrupts**
  - Implementation-defined
  - Maximum table size is 2048 bytes
- **Table may be relocated**
  - Use Vector Table Offset Register
  - Still require minimal table entries at 0x0 for booting the core
- **Each exception has a vector number**
  - Used in Interrupt Control and State Register to indicate the active or pending exception type
- **Table can be generated using C code**
  - Example provided later

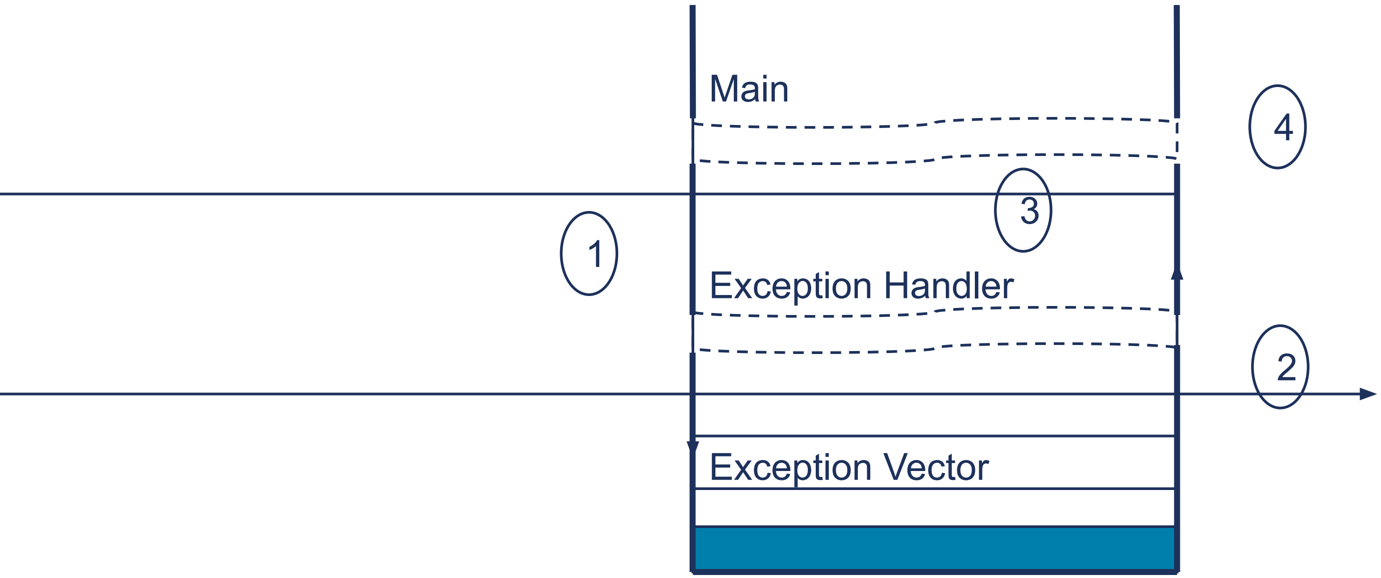
Address		**Vector #**
0x40 + 4*N	**External N**	16 + N
…	**…**	…
0x40	**External 0**	16
0x3C	**SysTick**	15
0x38	**PendSV**	14
0x34	**Reserved**	13
0x30	**Debug Monitor**	12
0x2C	**SVC**	11
0x1C to 0x28	**Reserved (x4)**	7-10
0x18	**Usage Fault**	6
0x14	**Bus Fault**	5
0x10	**Mem Manage Fault**	4
0x0C	**Hard Fault**	3
0x08	**NMI**	2
0x04	**Reset**	1
0x00	**Initial Main SP**	N/A

Reset Behavior



1. A reset occurs (Reset input was asserted)
2. Load MSP (Main Stack Pointer) register initial value from address 0x00
3. Load reset handler vector address from address 0x04
4. Reset handler executes in Thread Mode
5. Optional: Reset handler branches to the main program

Exception Behaviour



1. **Exception occurs**
   - Current instruction stream stops
   - Processor accesses vector table
2. **Vector address for the exception loaded from the vector table**
3. **Exception handler executes in Handler Mode**
4. **Exception handler returns to main**

Interrupt Service Routine Entry

- **When receiving an interrupt the processor will finish the current instruction for most instructions**
  - To minimize interrupt latency, the processor can take an interrupt during the execution of a multi-cycle instruction - see next slide
- **Processor state automatically saved to the current stack**
  - 8 registers are pushed: PC, R0-R3, R12, LR, xPSR
  - Follows ARM Architecture Procedure Calling Standard (AAPCS)
- **During (or after) state saving the address of the ISR is read from the Vector Table**
- **Link Register is modified for interrupt return**
- **First instruction of ISR executed**
  - For Cortex-M3 or Cortex-M4 the total latency is normally 12 cycles, however, interrupt late-arrival and interrupt tail-chaining can improve IRQ latency
- **ISR executes from Handler mode with Main stack**

Returning From Interrupt

- **Can return from interrupt with the following instructions when the PC is loaded with "magic" value of 0xFFFF_FFFX (same format as EXC_RETURN)**
  - LDR PC, …..
  - LDM/POP which includes loading the PC
  - BX LR (most common)

- **If no interrupts are pending, foreground state is restored**
  - Stack and state specified by EXC_RETURN is used
  - Context restore on Cortex-M3 and Cortex-M4 requires 10 cycles

- **If other interrupts are pending, the highest priority may be serviced**
  - Serviced if interrupt priority is higher than the foreground's base priority
  - Process is called Tail-Chaining as foreground state is not yet restored
  - Latency for servicing new interrupt is only 6 cycles on M3/M4 (state already saved)

- **If state restore is interrupted, it is abandoned**
  - New ISR executed without state saving (original state still intact and valid)
  - [illegible]

Vector Table in C

```
typedef void(* const ExecFuncPtr)(void) __irq;

#pragma arm section rodata="exceptions_area”

ExecFuncPtr exception_table[] = {
   (ExecFuncPtr)&Image$$ARM_LIB_STACK$$ZI$$Limit,   /* Initial SP */
   (ExecFuncPtr)__main,                         /* Initial PC */
   NMIException,
   HardFaultException,
   MemManageException,
   BusFaultException,
   UsageFaultException,
   0, 0, 0, 0,           /* Reserved */
   SVCHandler,
   DebugMonitor,
   0,                   /* Reserved */
   PendSVC,
   SysTickHandler
   /* Configurable interrupts start here...*/
};
#pragma arm section
```

The vector table at address 0x0 is minimally required to have 4 values: stack top, reset routine location, NMI ISR location, HardFault ISR location

The SVCall ISR location must be populated if the SVC instruction will be used

Once interrupts are enabled, the vector table (whether at 0 or in SRAM) must then have pointers to all enabled (by mask) exceptions

Vector Table in Assembly

```
              PRESERVE8
              THUMB
              IMPORT ||Image$$ARM_LIB_STACK$$ZI$$Limit||
              AREA    RESET, DATA, READONLY
              EXPORT  __Vectors

__Vectors     DCD     ||Image$$ARM_LIB_STACK$$ZI$$Limit||       ; Top of Stack
              DCD     Reset_Handler             ; Reset Handler
              DCD     NMI_Handler               ; NMI Handler
              DCD     HardFault_Handler         ; Hard Fault Handler
              DCD     MemManage_Handler         ; MemManage Fault Handler
              DCD     BusFault_Handler          ; Bus Fault Handler
              DCD     UsageFault_Handler        ; Usage Fault Handler
              DCD     0, 0, 0, 0,          ; Reserved x4
              DCD     SVC_Handler,             ; SVCall Handler
         DCD     Debug_Monitor             ; Debug Monitor Handler
         DCD     0                ; Reserved
              DCD     PendSV_Handler            ; PendSV Handler
              DCD     SysTick_Handler           ; SysTick Handler
              ; External vectors start here
```

Agenda

Introduction

ARM Architecture Overview

ARMv7-AR Architecture

- Programmer's Model
- Memory Systems

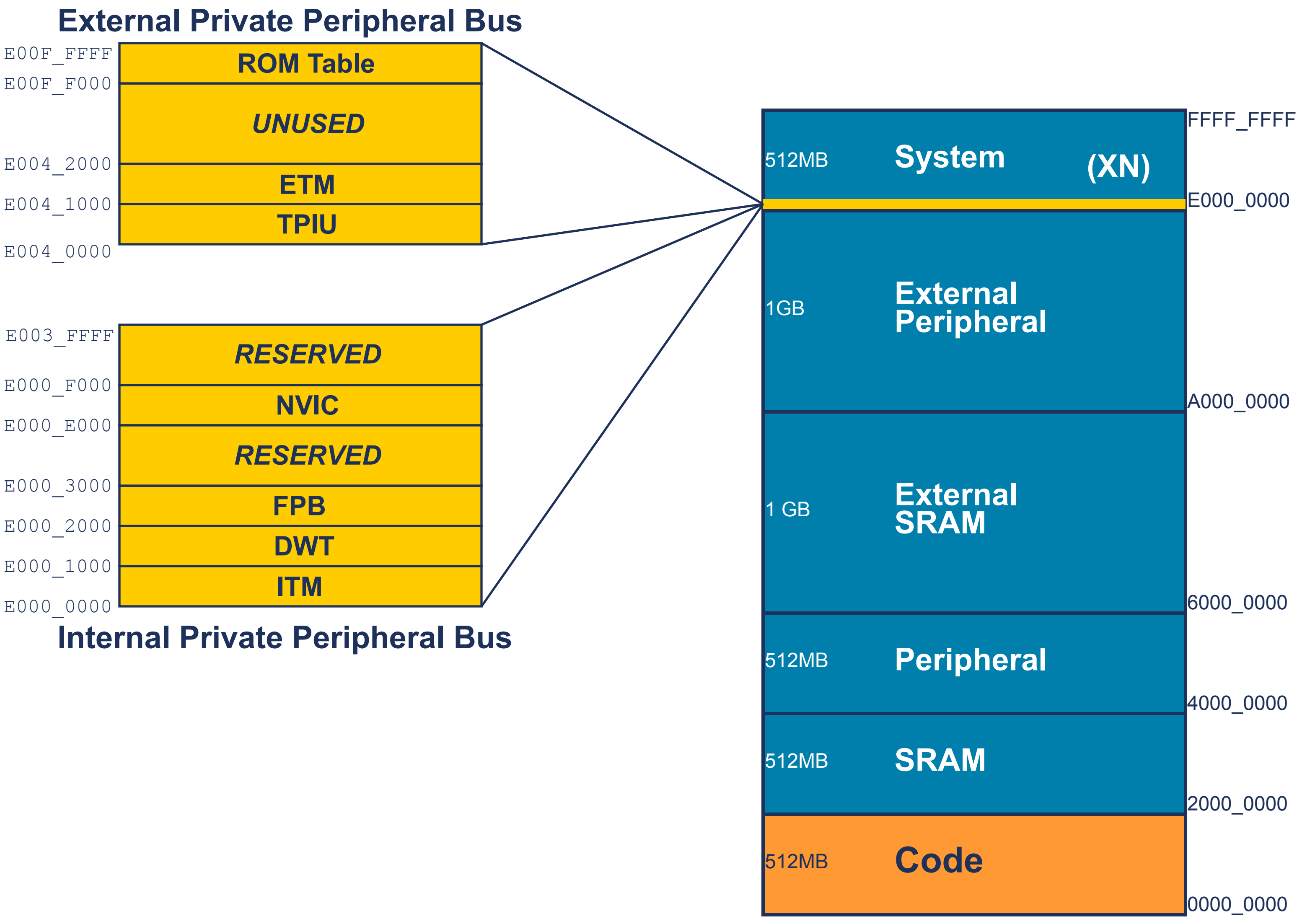
ARMv7-M Architecture

- Programmer's Model
- Memory Systems
- Floating Point Extensions

ARM System Design

Software Development Tools

Processor Memory Map



Memory Types and Properties

- **There are 3 different memory types:**
  - Normal, Device and Strongly Ordered

- **Normal memory is the most flexible memory type:**
  - Suitable for different types of memory, for example, ROM, RAM, Flash and SDRAM
  - Accesses may be restarted
  - Caches and Write Buffers are permitted to work alongside Normal memory

- **Device memory is suitable for peripherals and I/O devices**
  - Caches are not permitted, but write buffers are still supported
  - Unaligned accesses are unpredictable
  - Accesses must not be restarted
    - Load/store multiple instructions should not be used to access Device memory

- **Strongly ordered memory is similar to Device memory**

System Control Block

- **Memory mapped space containing registers to configure, control, and deal with interrupts, exceptions, and debug**
  - Replaces co-processor #15 in older ARM cores

Address	Type	Reset Value	Function
0xE000E000	Read/Write	0x00000000	Master Control register - RESERVED
0xE000E004	Read Only	IMP DEFINED	Interrupt Controller Type Register
0xE000ED00	Read Only	IMP DEFINED	CPUID Base Register
0xE000ED04	Read/Write	0x00000000	Interrupt Control State Register
0xE000ED08	Read/Write	0x00000000	Vector Table Offset Register
0xE000ED0C	Read/Write	Bits[10:8] = 000	Application Interrupt/Reset Control Register

More SCB Registers

Address	Type	Reset Value	Function
0xE000ED10	Read/Write	0x00000000	System Control Register
0xE000ED14	Read/Write	0x00000000	Configuration Control Register
0xE000ED18	Read/Write	0x00000000	System Handlers 4-7 Priority Register
0xE000ED1C	Read/Write	0x00000000	System Handlers 8-11 Priority Register
0xE000ED20	Read/Write	0x00000000	System Handlers 12-15 Priority Register
0xE000ED24	Read/Write	0x00000000	System Handler Control and State Register
0xE000ED28	Read/Write	n/a - status	Configurable Fault Status Registers (3)
0xE000ED2C	Read/Write	n/a - status	HardFault Status Register
0xE000ED30	Read/Write	n/a - status	DebugFault Status Register
0xE000ED34	Read/Write	Unpredictable	MemManage Address Register
0xE000ED38	Read/Write	Unpredictable	BusFault Address Register
0xE000ED3C	Read/Write	Unpredictable	Auxiliary Fault Status Register (vendor specific)
0xE000EF00	Write Only		Software Trigger Interrupt Register

Agenda

Introduction

ARM Architecture Overview

ARMv7-AR Architecture

- Programmer's Model
- Memory Systems

ARMv7-M Architecture

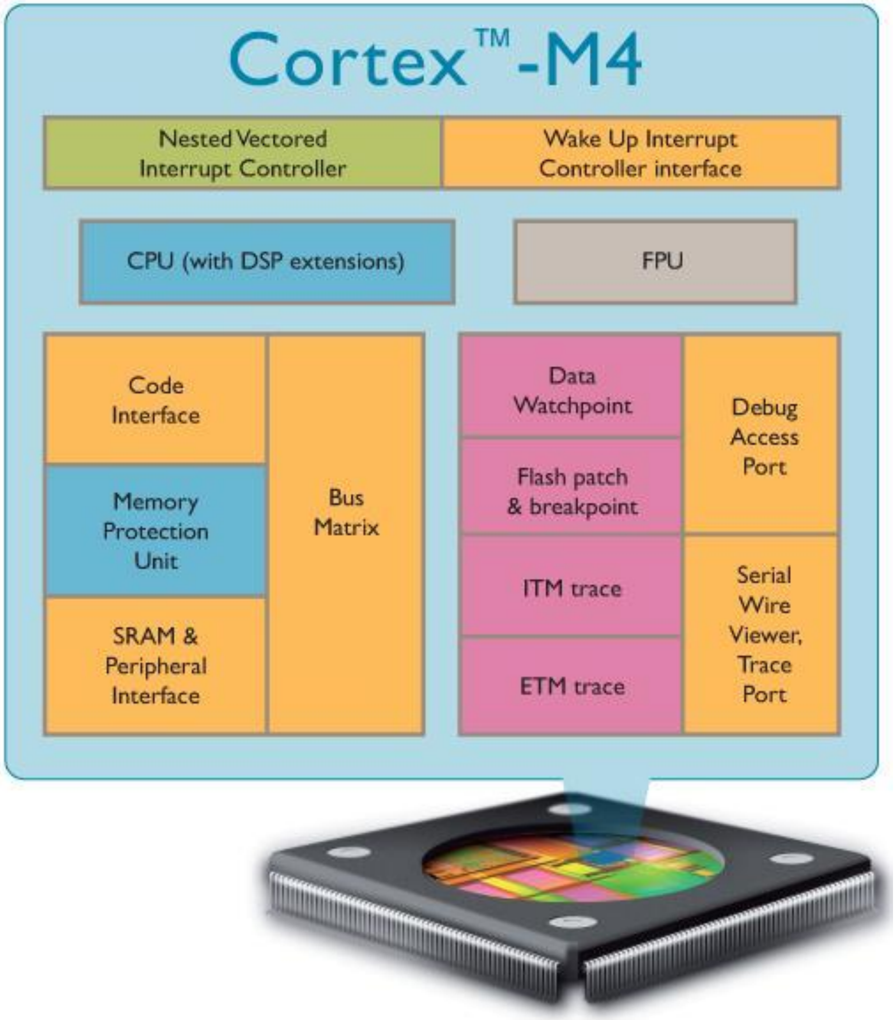
- Programmer's Model
- Memory Systems

- Floating Point Extensions

ARM System Design

Software Development Tools

Cortex-M4

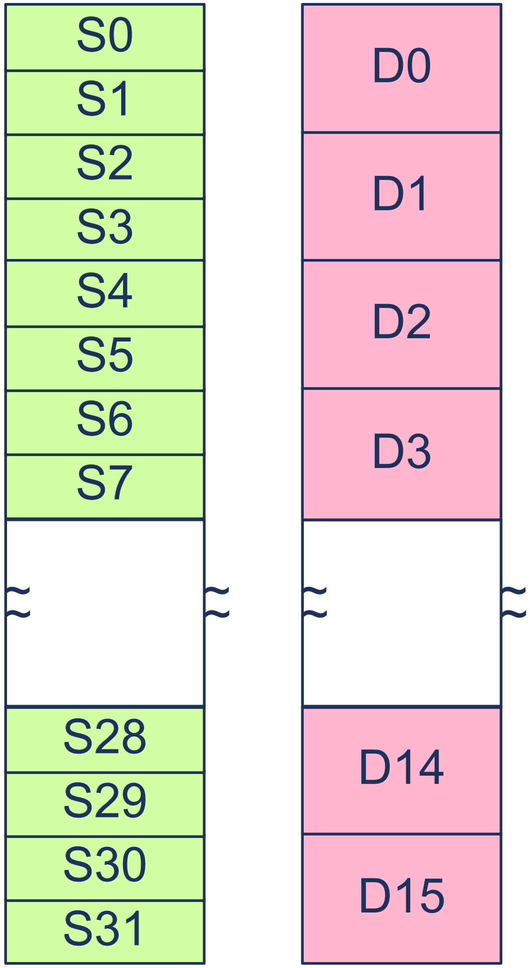


Cortex™-M4
Nested Vectored Interrupt Controller
Wake Up Interrupt Controller interface
CPU (with DSP extensions)
FPU
Code Interface
Memory Protection Unit
SRAM & Peripheral Interface
Bus Matrix
Data Watchpoint
Flash patch & breakpoint
ITM trace
ETM trace
Debug Access Port
Serial Wire Viewer, Trace Port

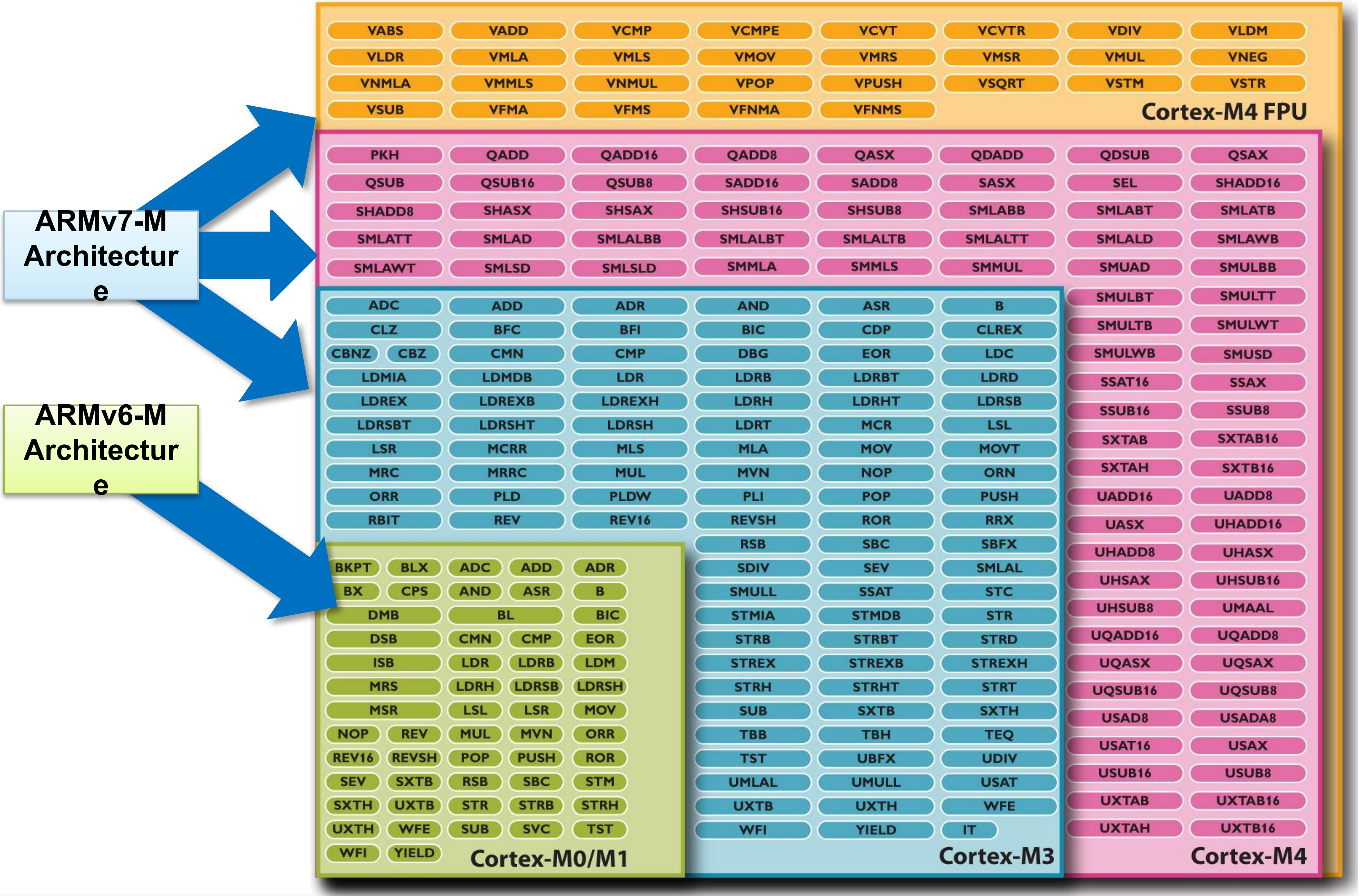
- **ARMv7E-M Architecture**
  - Thumb-2 only
  - DSP extensions
- **Optional FPU (Cortex-M4F)**
- **Otherwise, same as Cortex-M3**
- **Implements full Thumb-2 instruction set**
  - Saturated math (e.g. QADD)
  - Packing and unpacking (e.g. UXTB)
  - Signed multiply (e.g. SMULTB)
  - SIMD (e.g. ADD8)

Cortex-M4F Floating Point Registers

- **FPU provides a further 32 single-precision registers**
- **Can be viewed as either**
  - 32 x 32-bit registers
  - 16 x 64-bit doubleword registers
  - Any combination of the above



Binary Upwards Compatibility



ARMv7-M Architecture

ARMv6-M Architecture

VABS VADD VCMP VCMPE VCVT VCVTR VDIV VLDM
VLDR VMLA VMLS VMOV VMRS VMSR VMUL VNEG
VNMLA VMMLS VNMUL VPOP VPUSH VSQRT VSTM VSTR
VSUB VFMA VFMS VFNMA VFNMS

Cortex-M4 FPU

PKH QADD QADD16 QADD8 QASX QDADD QDSUB QSAX
QSUB QSUB16 QSUB8 SADD16 SADD8 SASX SEL SHADD16
SHADD8 SHASX SHSAX SHSUB16 SHSUB8 SMLABB SMLABT SMLATB
SMLATT SMLAD SMLALBB SMLALBT SMLALTB SMLALTT SMLALD SMLAWB
SMLAWT SMLSD SMLSLD SMMLA SMMLS SMMUL SMUAD SMULBB
SMULBT SMULTT
SMULTB SMULWT
SMULWB SMUSD
SSAT16 SSAX
SSUB16 SSUB8
SXTAB SXTAB16
SXTAH SXTB16
UADD16 UADD8
UASX UHADD16
UHADD8 UHASX
UHSAX UHSUB16
UHSUB8 UMAAL
UQADD16 UQADD8
UQASX UQSAX
UQSUB16 UQSUB8
USAD8 USADA8
USAT16 USAX
USUB16 USUB8
UXTAB UXTAB16
UXTAH UXTB16

Cortex-M4

ADC ADD ADR AND ASR B
CLZ BFC BFI BIC CDP CLREX
CBNZ CBZ CMN CMP DBG EOR LDC
LDMIA LDMDB LDR LDRB LDRBT LDRD
LDREX LDREXB LDREXH LDRH LDRHT LDRSB
LDRSBT LDRSHT LDRSH LDRT MCR LSL
LSR MCRR MLS MLA MOV MOVT
MRC MRRC MUL MVN NOP ORN
ORR PLD PLDW PLI POP PUSH
RBIT REV REV16 REVSH ROR RRX
RSB SBC SBFX
SDIV SEV SMLAL
SMULL SSAT STC
STMIA STMDB STR
STRB STRBT STRD
STREX STREXB STREXH
STRH STRHT STRT
SUB SXTB SXTH
TBB TBH TEQ
TST UBFX UDIV
UMLAL UMULL USAT
UXTB UXTH WFE
WFI YIELD IT

Cortex-M3

BKPT BLX ADC ADD ADR
BX CPS AND ASR B
DMB BL BIC
DSB CMN CMP EOR
ISB LDR LDRB LDM
MRS LDRH LDRSB LDRSH
MSR LSL LSR MOV
NOP REV MUL MVN ORR
REV16 REVSH POP PUSH ROR
SEV SXTB RSB SBC STM
SXTH UXTB STR STRB STRH
UXTH WFE SUB SVC TST
WFI YIELD

Cortex-M0/M1

University Program Material

Agenda

Introduction

ARM Architecture Overview

ARMv7-AR Architecture

- Programmer's Model
- Memory Systems

ARMv7-M Architecture

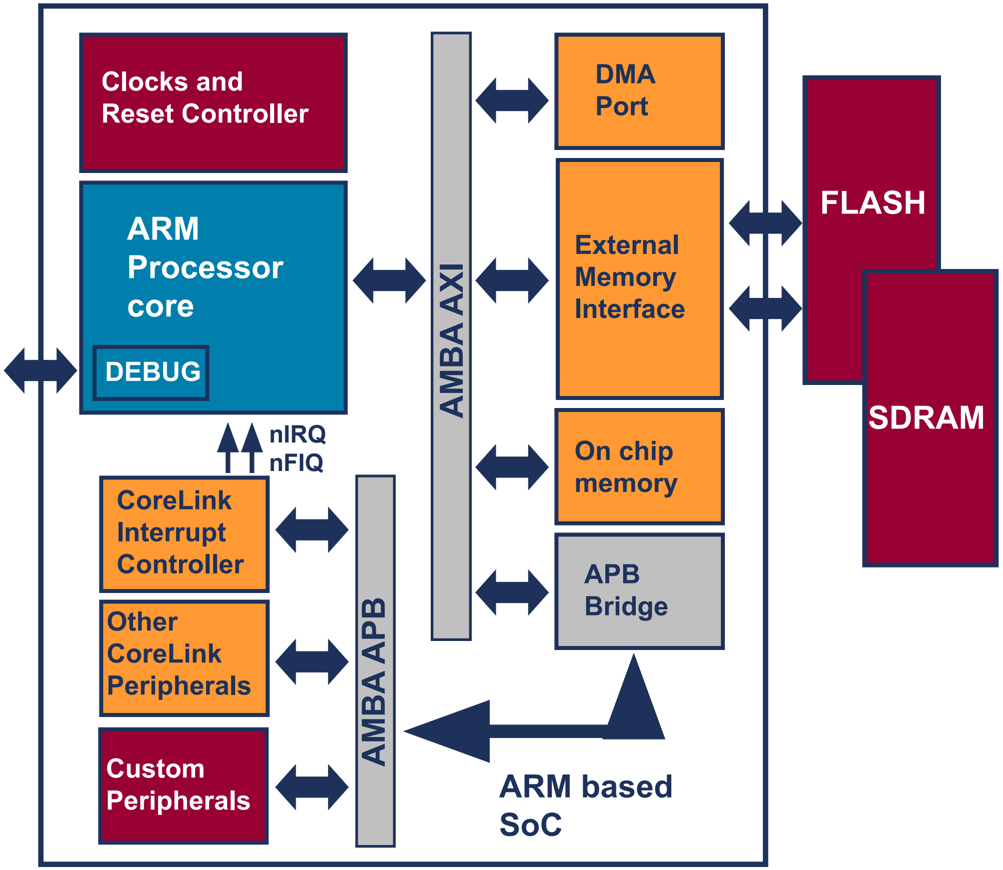
- Programmer's Model
- Memory Systems
- Floating Point Extensions

- **ARM System Design**

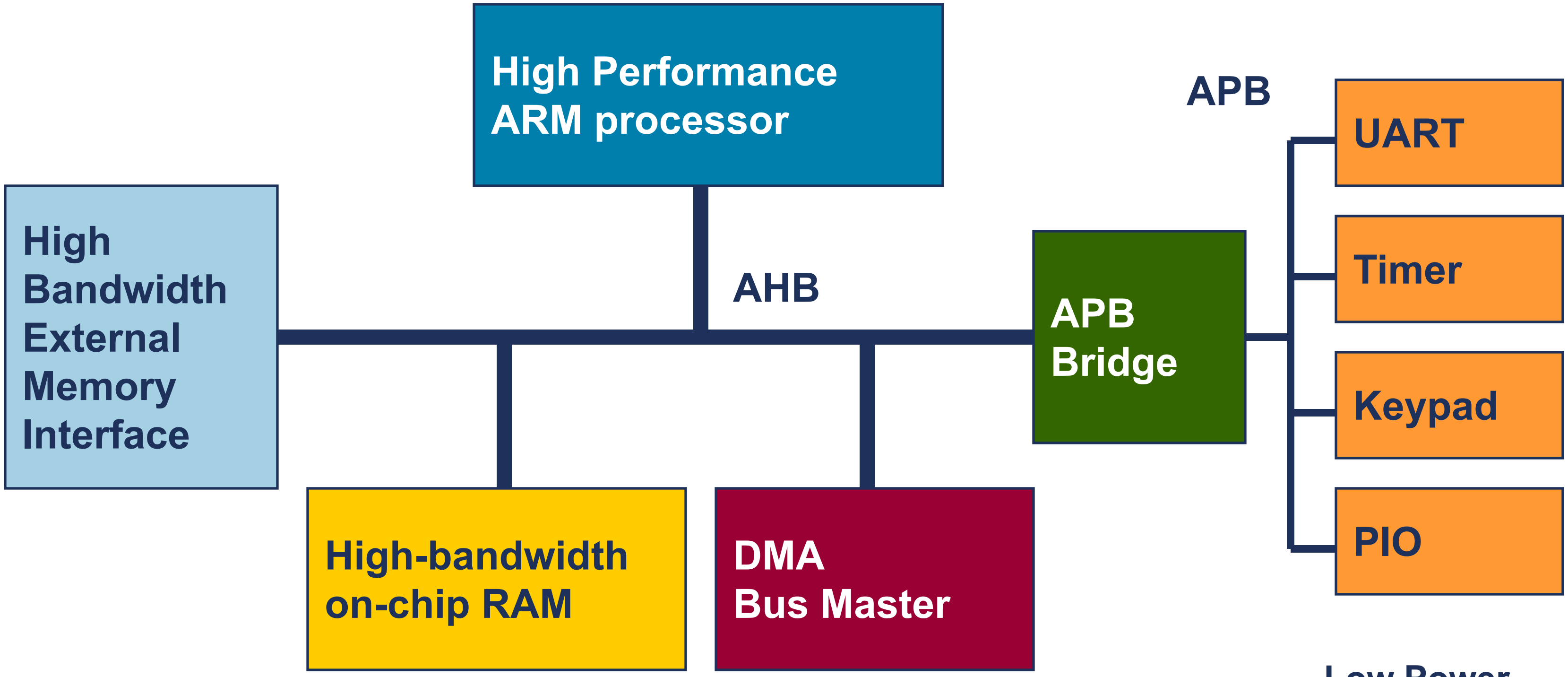
Software Development Tools

Example ARM-based system

- **ARM core deeply embedded within an SoC**
  - External debug and trace via JTAG or CoreSight interface
- **Design can have both external and internal memories**
  - Varying width, speed and size – depending on system requirements
- **Can include ARM licensed CoreLink peripherals**
  - Interrupt controller, since core only has two interrupt sources
  - Other peripherals and interfaces
- **Can include on-chip memory from ARM Artisan Physical IP Libraries**
- **Elements connected using AMBA (Advanced Microcontroller Bus Architecture)**



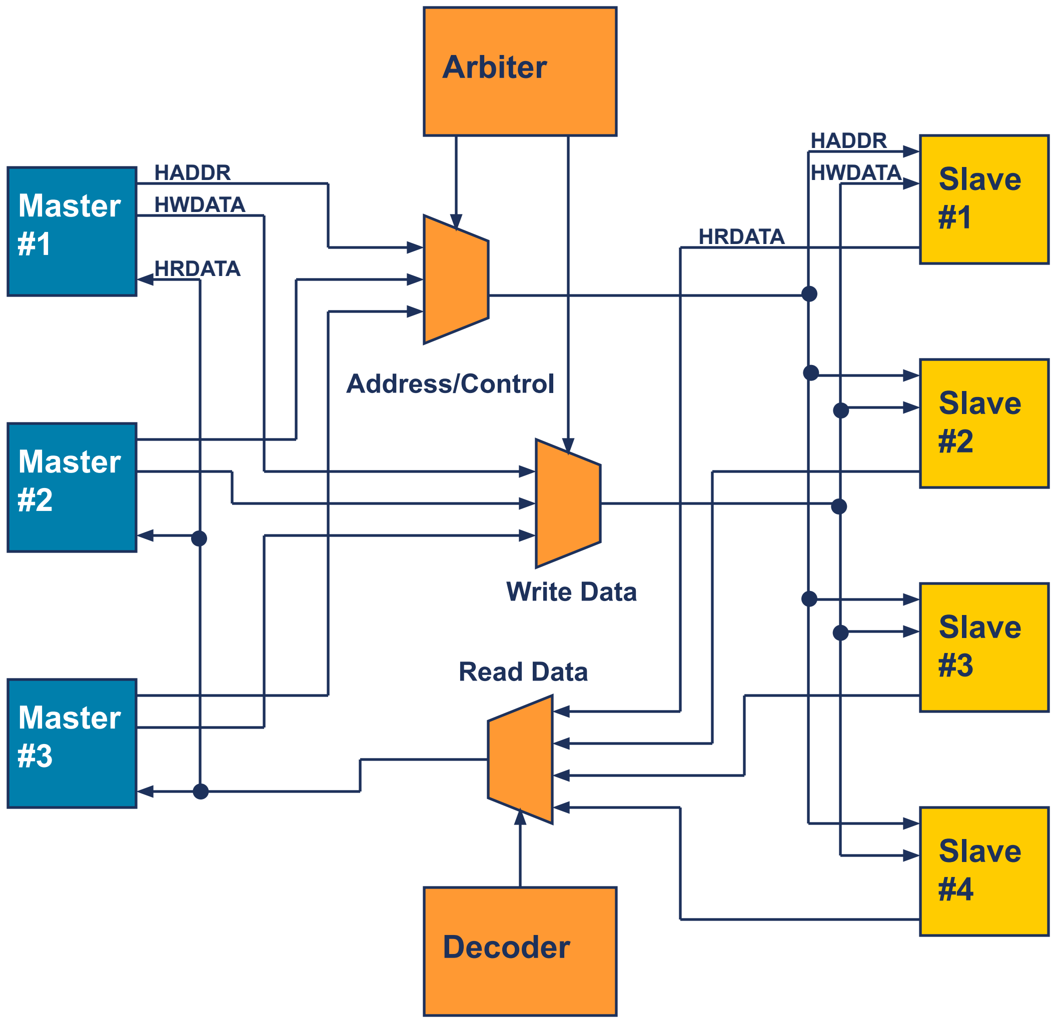
An Example AMBA System


High Performance ARM processor
APB
UART
Timer
AHB
APB Bridge
Keypad
PIO
High Bandwidth External Memory Interface
High-bandwidth on-chip RAM
DMA Bus Master

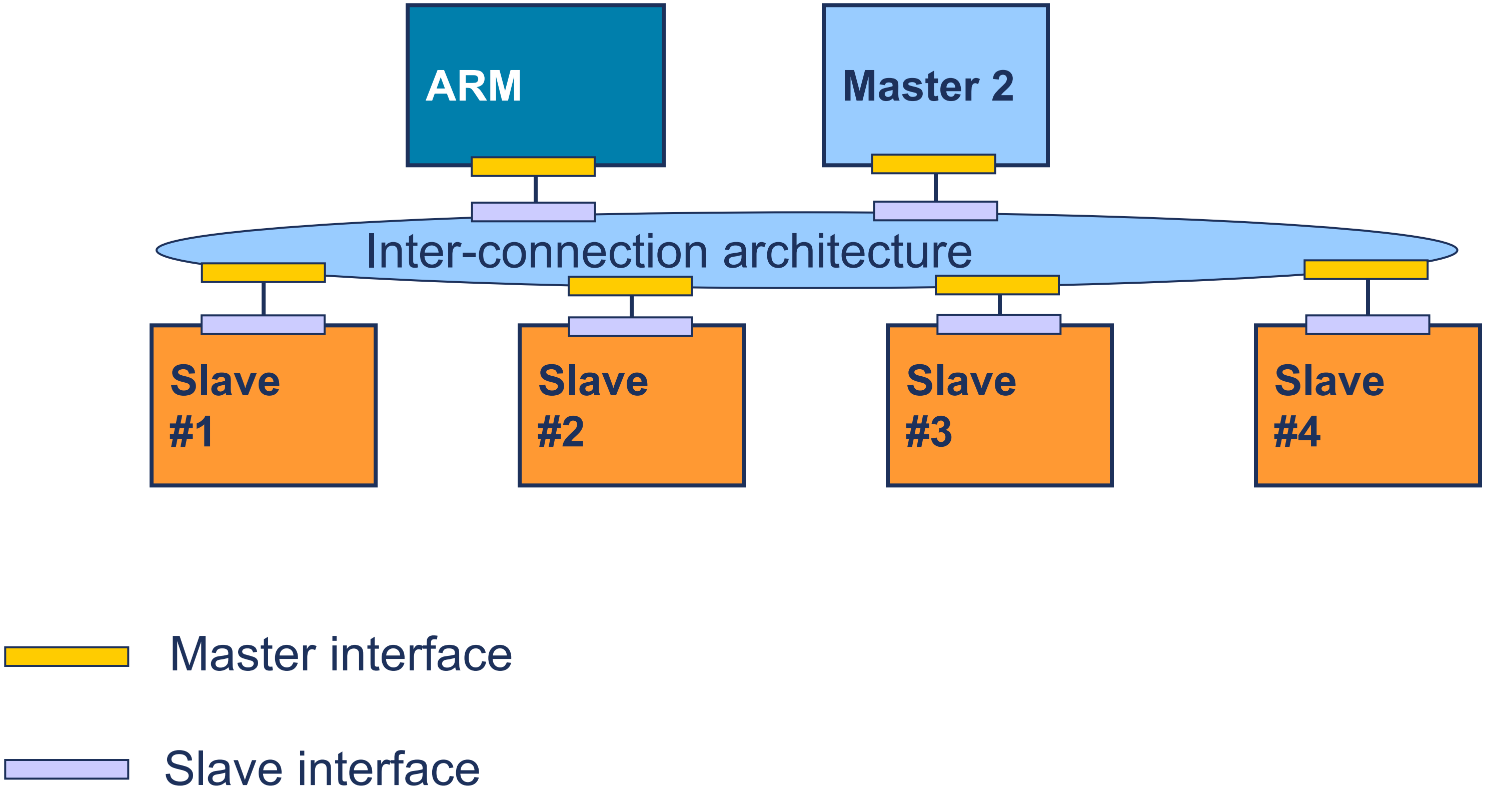
Low Power
Non-pipelined
Simple Interface

High Performance
Pipelined
Burst Support
Multiple Bus Masters

AHB Structure



AXI Multi-Master System Design



Agenda

Introduction

ARM Architecture Overview

ARMv7-AR Architecture

- Programmer’s Model
- Memory Systems

ARMv7-M Architecture

- Programmer’s Model
- Memory Systems
- Floating Point Extensions

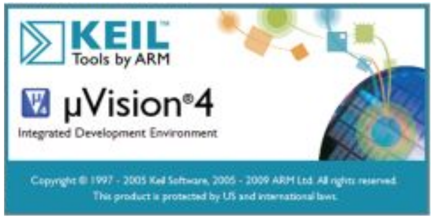
ARM System Design

- **Software Development Tools**

ARM Software Development Tools

Software Tools

- **DS-5**
  - Application Edition
  - Linux Edition
  - Professional Edition
- **MDK: Keil Microcontroller Development Kit**



KEIL™ Tools by ARM

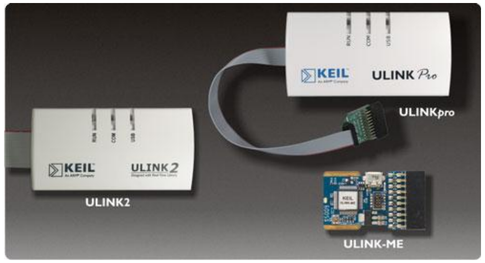
µVision®4

Integrated Development Environment

Copyright © 1997 - 2005 Keil Software, 2005 - 2009 ARM Ltd. All rights reserved.
This product is protected by US and international laws.

JTAG Debug and Trace

- DSTREAM
- **ULINK**

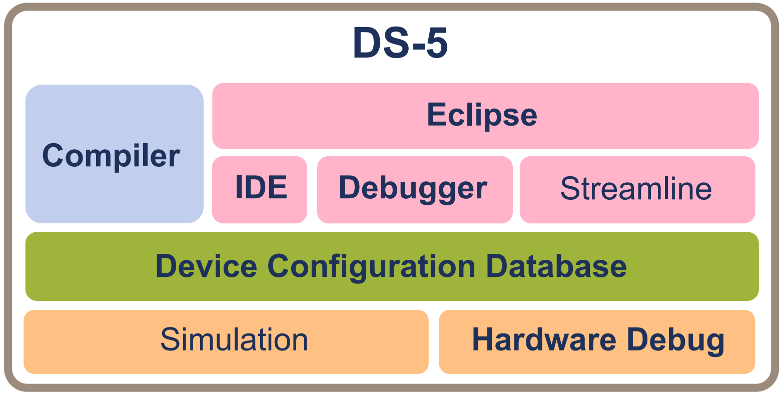


Development Platforms

- Fast Models
- Versatile Platform baseboards
- Keil MCU development boards
- Keil µVision simulator

DS-5 Professional at a Glance

- **Integrated solution, professionally supported and maintained**
  - End-to-end development, from SoC bring-up to application debug
- **Powerful ARM compiler**
  - Best code size and performance
- **Intuitive DS-5 debugger**
  - Flexible graphical user interface
  - DSTREAM probe with 4GB trace buffer
- **Fast SoC simulation models**
  - Develop in a controlled environment
  - Examples and applications
- **Streamline performance analyzer**
  - System-wide analysis of Linux and Android systems



DS-5

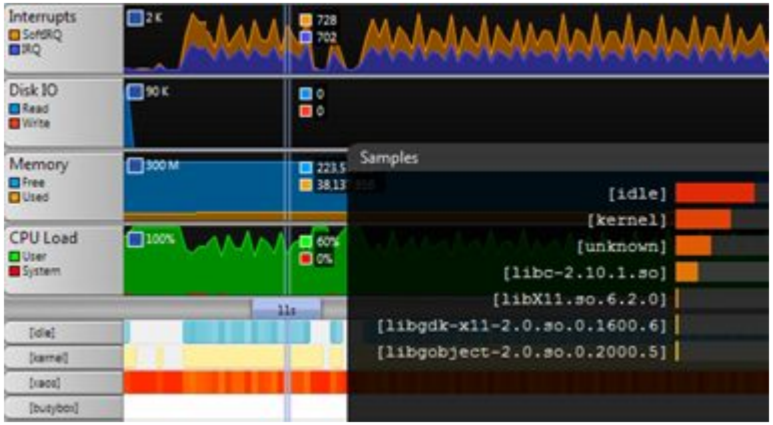
Compiler

Eclipse

IDE | **Debugger** | Streamline

Device Configuration Database

Simulation | **Hardware Debug**

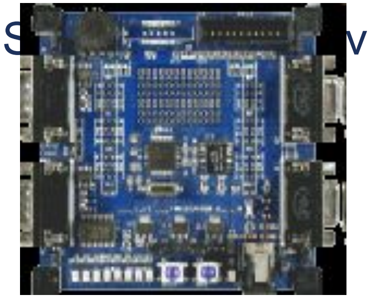


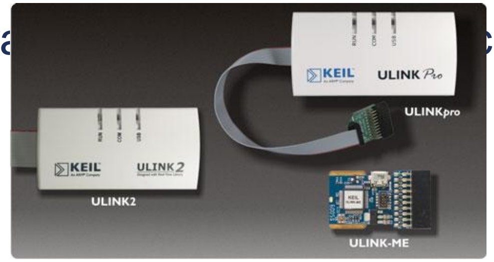
Development Suite: MDK

- **Low cost tools for ARM7, ARM9, Cortex-M and Cortex-R4 MCUs**
  - Extensive device support for many devices
  - Core and peripheral simulation
  - Flash support

- **Microcontroller Development Kit (MDK)**
  - IDE, optimized run-time library, KEIL RTX RTOS
  - ARM Compiler
  - Realtime trace (for Cortex-M3 and Cortex-M4 based devices)

- **Real-Time Library**
  - KEIL RTX RTOS + Source Code
  - TCP networking suit, Flash File S ver Libra ce
- **Debug Hardware**
- **Evaluation boards**





GNU Tools and Linux

- **GNU/GCC Tools Support**
  - ARM works with CodeSourcery to keep the GNU toolchain current with the latest ARM processors

- **Linux Support**
  - Pre-built Linux images are available for ARM hardware platforms
  - DS-5 accepts kernel images built with the GNU toolchain
    - Can also debug applications or loadable kernel modules
  - RVCT can be used to build Linux applications or libraries
    - Giving performance benefits

- **ARM does not provide technical support for the GNU toolchain, or Linux kernel/driver development**

ARM University Program

August 2012