ELEC6234
Embedded Processor Synthesis

3. ARM embedded processor cores adopted from ARM University Programme material

Tom Kazmierski
Faculty of Physical Sciences and Engineering,
University of Southampton, UK
tjk@ecs.soton.ac.uk

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 website:
  - http://www.arm.com/aboutarm/

ARM Embedded Processors

ARM Application Processors
**Development of the ARM Architecture**

- v4
  - Halfword and signed halfword / byte support
  - Improved interworking
  - CLZ
  - Saturated arithmetic
  - DSP MAC instructions
  - Extensions: Jazelle (STEJ)

- v5
  - Improved interworking
  - CLZ
  - Saturated arithmetic
  - DSP MAC instructions
  - Extensions: Jazelle (STEJ)

- v6
  - Improved interworking
  - CLZ
  - Saturated arithmetic
  - DSP MAC instructions
  - Extensions: Jazelle (STEJ)

- 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

**ARM architectures**

- Classic ARM processors
- Application processors
- Embedded ARM processors

- ARM1026
- ARM1022
- ARM1136
- ARM1134
- ARM1176
- ARM1146
- ARM1142
- ARM1147
- ARM1214
- ARM1210
- ARM1174
- ARM1173
- Thumb-2
- Thumb-2E (Jazelle-RTC)
- TrustZone extensions
- Custom or synthesized design
- MMU
- 64-bit or 128-bit AXI Interface
- L1 caches
  - 16 or 32KB each
  - Unified L2 cache
  - 6-2MB in size
  - 8-way set-associative

**Cortex-A8**

- ARMv7-A Architecture
  - Thumb-2
  - Thumb-2EE (Jazelle-RTC)
  - TrustZone extensions

- Optional features
  - VFPv3 Vector Floating-Point
  - NEON media processing engine

- 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
- 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

**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)

**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
- Advanced Power Management
- Large Physical Address Extensions (LPAE) to ARMv7-A Architecture
- Virtualization Extensions to ARMv7-A Architecture

**Data Sizes and Instruction Sets – cont.**

- **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** - provides support for interpreted languages
**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

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

**The ARM Register Set**

<table>
<thead>
<tr>
<th>User mode</th>
<th>IRQ</th>
<th>FIQ</th>
<th>Undef</th>
<th>Abort</th>
<th>SVC</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Program Status Registers**

- Condition code flags
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation Carried out
  - V = ALU operation overflowed

- 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 Thumb instructions

**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

<table>
<thead>
<tr>
<th>manipulation</th>
<th>arithmetic</th>
<th>logical</th>
<th>move</th>
</tr>
</thead>
<tbody>
<tr>
<td>(has destination</td>
<td>ADC</td>
<td>BIC</td>
<td>MVN</td>
</tr>
<tr>
<td>register)</td>
<td>SUB</td>
<td>ORR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RSB</td>
<td>EOR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RSC</td>
<td>ORN</td>
<td></td>
</tr>
</tbody>
</table>

| comparison            |           |         |      |
| (set flags only)      | CMN       | TST     | TEQ  |
|                       | (ADDS)    | (ANDS)  | (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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD</td>
<td>STR</td>
<td>Doubleword</td>
<td>Word</td>
</tr>
<tr>
<td>LDRB</td>
<td>STRB</td>
<td>Byte</td>
<td></td>
</tr>
<tr>
<td>LDRH</td>
<td>STRH</td>
<td>Halfword</td>
<td></td>
</tr>
<tr>
<td>LDRSB</td>
<td>STRSB</td>
<td>Signed byte load</td>
<td></td>
</tr>
<tr>
<td>LDRSH</td>
<td>STRSH</td>
<td>Signed halfword load</td>
<td></td>
</tr>
</tbody>
</table>

- 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
  - Decrements after/before

- Also
  - `PUSH/POP`, equivalent to `STMSB/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`

```c
void func1 (void)
{
    func2 ();
}
```
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) 0x1C
    - Change to exception mode 0x18
    - Disable interrupts (if appropriate) 0x14
  - Stores the return address in LR_<mode>
  - Sets PC to vector address 0x0C
- 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
- Note that v7-M and v6-M exception model is different

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
   - 3 and 4 responsibility of software

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 lanes

Vector Table

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

Source Registers

Destination Register

Elements

Operation

Lane
How does the compiler perform vectorization?

• **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

---

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)

• Before caches can be used, software setup must be performed

---

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?

```c
void add_int(int *pa, int *pb, unsigned int n, int x) {
    for (i = 0; i < (n & ~3); i++)
        pa[i] = pb[i] + x;
    pa += 4; pb += 4;
}
```

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

```c
void add_int(int *pa, int *pb, unsigned int 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
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

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)
  - Timers and Watchdogs

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
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
  - No coprocessor 15 – controlled through memory mapped control registers

Cortex-M0

- 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
- Low power support
- Core configured with or without debug
  - Variable number of watchpoints and breakpoints

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

ARM Processor

Application Code

Thread Mode

Reset

Exception Entry

Exception Code

Handler Mode

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

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)
  – Allows handler to be written entirely in ‘C’

Instruction Set Examples:

• Data Processing:
  MOV r2, r5
  ADD r5, #0x24
  ADD r2, r3, r4, LSL #2
  LSL r2, #3
  MLA r0, r1, r2, r3

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

• Memory Access:
  STRB r2, [r10, r1]
  LDR r0, [r1, r2, LSL #2]

• Program Flow:
  BL <label>

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

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

INTNMI
INTISR[0]
...
INTISR[N]

NVIC
Cortex-Mx Integration Layer

Cortex-Mx Processor Core
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)
  - Must still fetch new vector and refill pipeline (6-cycle latency on M3/M4)
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
- Buffers are not supported and the PPB is marked Strongly Ordered

Cortex-M4

- 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

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)
### ARM Software Development Tools

**Software Tools**
- DS-5 (Development Studio)
  - Application Edition
  - Linux Edition
  - Professional Edition
- MDK: Keil Microcontroller Development Kit

**JTAG Debug and Trace**
- DSTREAM

**Development Platforms**
- Fast Models
- Versatile Platform baseboards

- Keil MCU development boards
- Keil µVision simulator

- ULINK

### AXI Multi-Master System Design

AXI - Advanced eXtensible Interface

**Inter-connection architecture**

- Master interface
- Slave interface

### 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
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 System, CAN Driver Library, USB Device Interface
- Debug Hardware
- Evaluation boards
- Separate support channel
- See www.keil.com

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

• See www.keil.com