ARM Architecture Overview

Development of the ARM Architecture

- **Processor Architecture = Instruction Set + Programmer’s model**

<table>
<thead>
<tr>
<th>4T</th>
<th>5TE</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM7TDMI</td>
<td>ARM926EJ-S</td>
<td>ARM1136JF-S</td>
<td>Cortex-A8/R4/M3/M1</td>
</tr>
<tr>
<td>ARM922T</td>
<td>ARM946E-S</td>
<td>ARM1176JZF-S</td>
<td>Thumb-2</td>
</tr>
<tr>
<td>Thumb instruction set</td>
<td>Improved ARM/Thumb Interworking</td>
<td>SIMD Instructions</td>
<td>Extensions:</td>
</tr>
<tr>
<td></td>
<td>DSP instructions</td>
<td>Unaligned data support</td>
<td>v7A (applications) – NEON</td>
</tr>
<tr>
<td></td>
<td>Extensions:</td>
<td></td>
<td>v7R (real time) – HW Divide</td>
</tr>
<tr>
<td></td>
<td>Jazelle (5TEJ)</td>
<td></td>
<td>V7M (microcontroller) – HW</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Divide and Thumb-2 only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Note:** Implementations of the same architecture can be very different
  - ARM7TDMI - architecture v4T. Von Neuman core with 3 stage pipeline
  - ARM920T - architecture v4T. Harvard core with 5 stage pipeline and MMU
ARM Architecture profiles

- Application profile (ARMv7-A → e.g. Cortex-A8)
  - 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

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

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

Programmer’s Model
Data Sizes and Instruction Sets

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

- Most ARMs implement two instruction sets
  - 32-bit **ARM Instruction Set**
  - 16-bit **Thumb Instruction Set**

- Latest ARM cores introduce a new instruction set **Thumb-2**
  - Provides a mixture of 32-bit and 16-bit instructions
  - Maintains code density with increased flexibility

- Jazelle-DBX cores can also execute **Java bytecode**

Processor Modes

- The ARM has seven basic operating modes:
  - Each mode has access to own stack 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 Software Interrupt instruction (SWI) 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 low priority (normal) 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>

Confidential
The ARM Register Set

User mode

<table>
<thead>
<tr>
<th>r0</th>
<th>r1</th>
<th>r2</th>
<th>r3</th>
<th>r4</th>
<th>r5</th>
<th>r6</th>
<th>r7</th>
<th>r8</th>
<th>r9</th>
<th>r10</th>
<th>r11</th>
<th>r12</th>
<th>r13 (sp)</th>
<th>r14 (lr)</th>
<th>r15 (pc)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Current mode

<table>
<thead>
<tr>
<th>r8</th>
<th>r9</th>
<th>r10</th>
<th>r11</th>
<th>r12</th>
<th>r13 (sp)</th>
<th>r14 (lr)</th>
<th>r15 (pc)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Banked out registers

ARM has 37 registers, all 32-bits long
A subset of these registers is accessible in each mode

Program Status Registers

<table>
<thead>
<tr>
<th>31</th>
<th>28</th>
<th>27</th>
<th>24</th>
<th>23</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Z</td>
<td>C</td>
<td>V</td>
<td>D</td>
<td>J</td>
<td>GE[3:0]</td>
<td>IT cond_abc</td>
<td>T</td>
<td>A</td>
<td>I</td>
<td>F</td>
<td>J</td>
<td>mode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>a</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Condition code flags
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation Carried out
  - V = ALU operation overflowing
- Sticky Overflow flag - Q flag
  - Architecture STE and later only
  - Indicates if saturation has occurred
- J bit
  - Architecture STEJ and later only
  - J = 1: Processor in Jazelle state
- Interrupt Disable bits
  - I = 1: Disables IRQ
  - F = 1: Disables FIQ
- T Bit
  - T = 0: Processor in ARM state
  - T = 1: Processor in Thumb state
  - Introduced in Architecture 4T
- Mode bits
  - Specify the processor mode
- New bits in V6
  - GE[3:0] used by some SIMD instructions
  - E bit controls load/store endianness
  - A bit disables imprecise data aborts
  - IT [abcde] IF THEN conditional execution of Thumb2 instruction groups
Data alignment

- Prior to architecture v6 data accesses must be appropriately aligned for access size
  - Unaligned addresses will produce unexpected/undefined results
- Unaligned data can be accessed using multiple aligned accesses combined with shift/mask operations

<table>
<thead>
<tr>
<th>Byte access (byte aligned)</th>
<th>Halfword access (halfword aligned)</th>
<th>Word access (word aligned)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 1 0</td>
<td>2 0</td>
<td>0</td>
</tr>
<tr>
<td>7 6 5 4</td>
<td>6 4</td>
<td>4</td>
</tr>
<tr>
<td>1 0 9 8</td>
<td>a 8</td>
<td>8</td>
</tr>
<tr>
<td>1 0 9 8</td>
<td>a 8</td>
<td>8</td>
</tr>
</tbody>
</table>

Exception Handling

- When an exception occurs, the core:
  - Copies CPSR into SPSR_<mode>
  - Sets appropriate CPSR bits
    - Change to ARM state
    - 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>
- Must be done in ARM state in most cores, but...
  ...Thumb-2 capable cores can do this in Thumb state
Introduction to Instruction Sets

ARM Instruction Set

- All instructions are 32 bits long / many execute in a single cycle
- Instructions are conditionally executed
- A load / store architecture

- Example data processing instructions
  
  ```
  SUB r0, r1, #5
  ADD r2, r3, r3, LSL #2
  ADDEQ r5, r5, r6
  ```

- Example branching instruction
  ```
  B <Label>
  ```

- Example memory access instructions
  ```
  LDR r0, [r1]
  STRNEB r2, [r3, r4]
  STMFD sp!, {r4-r8, lr}
  ```

Examples:
- \( r0 = r1 - 5 \)
- \( r2 = r3 + (r3 \times 4) \)
- IF EQ condition true \( r5 = r5 + r6 \)
- Branch forwards or backwards relative to current PC (+/- 32MB range)
- Load word at address \( r1 \) into \( r0 \)
- IF NE condition true, store bottom byte of \( r2 \) to address \( r3+r4 \)
  Store registers \( r4 \) to \( r8 \) and \( lr \) on stack. Then update stack pointer
Thumb Instruction Set

- Thumb is a 16-bit instruction set
  - Optimized for code density from C code (~65% of ARM code size)
  - Improved performance from narrow memory
  - Subset of the functionality of the ARM instruction set

- Thumb is not a “regular” instruction set!
  - Constraints are not generally consistent
  - Targeted at compiler generation, not hand coding

Thumb-2 Instruction Set

- Thumb-2 is a major extension to the Thumb ISA
  - Adds 32-bit instructions to implement almost all of the ARM ISA functionality
  - Retains the complete 16-bit Thumb instruction set

- Design objective: ARM performance with Thumb code density
  - No switching between ARM-Thumb states
  - Compiler automatically selects mix of 16 and 32 bit instructions
Thumb 2 Performance / Density

- 100% ARM code
- Random mix
- Profilled mix
- 100% Thumb code

Processor Cores
ARM7TDMI Processor

- Architecture v4T
- 3-stage pipeline
- Single interface to memory

ARM926EJ-S Processor

ARM926EJ-S
- Architecture v5TE
- 5-stage pipeline
- Single-cycle 32x16 multiplier
- Caches and TCMs
- Memory management unit (MMU)
- 2 AHB memory interfaces
- Jazelle technology
ARM1176JZ(F)-S Processor Core

- TrustZone
- 8-stage pipeline
- Branch prediction
- Four AXI memory ports
- IEM (Intelligent Energy Management)
- Integrated VFP coprocessor

ARM11 MPCore Processor

- 1 – 4 MP11 processors
- Cache coherency
- Distributed interrupt controller
ARM Cortex-M3 Processor

- Architecture v7-M (Thumb-2 only) → Very different from previous ARM processors
  - No CPSR register
  - Vector table contains addresses, not instructions
  - Processor automatically saves/restores state in exceptions
  - Only 2 processor modes (Thread/Handler)
  - No Coprocessor 15 3-stage pipeline with static branch prediction

- Atypical Implementation
  - Fixed memory map
  - Integrated interrupt controller
  - Serial-Wire Debug

ARM Cortex-A8 Processor

- Architecture v7-A
- 14 stage pipeline
- NEON media processor
The Instruction Pipeline

The ARM7TDMI uses a 3-stage pipeline in order to increase the speed of the flow of instructions to the processor:

- Allows several operations to be performed simultaneously, rather than serially.

**Instructions:**

- **FETCH**
  - Instruction fetched from memory
- **Decode**
  - Decoding of registers used in instruction
- **Execute**
  - Register(s) read from Register Bank
  - Shift and ALU operation
  - Write register(s) back to Register Bank

- The PC points to the instruction being fetched, not executed.
  - Debug tools will hide this from you.
  - This is now part of the ARM Architecture and applies to all processors.
### Optimal Pipelining

<table>
<thead>
<tr>
<th>Cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORR</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORR</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EOR</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- All operations here are on registers (single cycle execution)
- In this example it takes 6 clock cycles to execute 6 instructions
- Clock cycles per Instruction (CPI) = 1

### Branch Pipeline Example

<table>
<thead>
<tr>
<th>Cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Operation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x8000</td>
<td>BL 0x8FEC</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x8004</td>
<td>SUB</td>
<td>F</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x8008</td>
<td>ORR</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x8FEC</td>
<td>AND</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x8FF0</td>
<td>ORR</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x8FF4</td>
<td>EOR</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Breaking the pipeline
- Note that the core is executing in ARM state
Cortex-A8 Integer Pipeline

- Optimising code to make use of the processor pipeline is very difficult
- Leave it to the compiler!!

Reference Slides
Reference Material

- ARM ARM ("Architecture Reference Manual")
  - ARM DDI 0100E covers v5TE DSP extensions
  - Can be purchased from booksellers - ISBN 0-201-737191 (Addison-Wesley)
  - Available for download from ARM's website
  - ARM v7-M ARM available for download from ARM's website
  - Contact ARM if you need a different version (v6, v7-AR, etc.)

- Steve Furber “ARM system-on-chip architecture” - 2nd edition
  - ISBN 0-201-67519-6 (Addison-Wesley)

- Sloss, Symes & Wright – “ARM System Developer's Guide”

- RVCT Assembler Guide
  - Available for download from ARM's website

- Technical Reference Manuals for processor core being used
  - Available for download from ARM's website

Naming Conventions

- ARMx1z (e.g. ARM710T) indicates cache & full MMU
- ARMx2z (e.g. ARM720T) indicates cache, MMU & Process ID support
- ARMx3z (e.g. ARM1136J-S) indicates physically mapped caches and MMU
- ARMx4z (e.g. ARM740T) indicates cache and MPU
- ARMx5z (e.g. ARM1156T2-S) indicates cache, MPU and error correcting memory
- ARMx6z (e.g. ARM966E-S) indicates write buffer but no caches
- ARMx7z (e.g. ARM1176JZ-S) indicates AXI bus, & physically mapped caches and MMU
- ARMxy6 (e.g. ARM946E-S) indicates TCMs
Which architecture is my processor?

- **ARM7TDMI family**
  - ARM720T, ARM740T
- **ARM9TDMI family**
  - ARM920T, ARM922T, ARM940T
- **ARM9E family**
  - ARM946E-S, ARM966E-S, ARM926EJ-S
- **ARM10E family**
  - ARM1020E, ARM1022E, ARM1026EJ-S
- **ARM11 family**
  - ARM1136J(F)-S, ARM1156T2(F)-S
  - ARM1176JZF-S, v6Z
  - ARM11 MPCore, v6
- **Cortex family**
  - ARM Cortex-A8, v7-A
  - ARM Cortex-R4(F), v7-R
  - ARM Cortex-M3, v7-M
  - ARM Cortex-M1, v6-M

For ARM processor naming conventions and features, please see the Appendix.

---

**ARMv4T Cores:**

<table>
<thead>
<tr>
<th></th>
<th>7TDMI</th>
<th>720T</th>
<th>740T</th>
<th>920T</th>
<th>940T</th>
<th>SA1100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache</td>
<td>None</td>
<td>8K Unified 4 words/line</td>
<td>8K Unified 4 words/line</td>
<td>16K Instr + 16K Data 8 words/line</td>
<td>4K Instr + 4K Data 4 words/line</td>
<td>16K Instr + 16K Data 4 words/line</td>
</tr>
<tr>
<td>Associativity</td>
<td>N/A</td>
<td>4-way</td>
<td>4-way</td>
<td>64-way</td>
<td>64-way</td>
<td>32-way</td>
</tr>
<tr>
<td>TCM</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Replacement</td>
<td>N/A</td>
<td>Random</td>
<td>Random</td>
<td>Random Round Robin</td>
<td>Random</td>
<td>Round Robin</td>
</tr>
<tr>
<td>Write Strategy</td>
<td>N/A</td>
<td>Write Through</td>
<td>Write Through</td>
<td>Write Through Write Back</td>
<td>Write Through Write Back</td>
<td>Write Back</td>
</tr>
<tr>
<td>Write Buffer</td>
<td>None</td>
<td>8 Words 4 Addresses</td>
<td>8 Words 4 Addresses</td>
<td>16 Words 4 Addresses</td>
<td>8 Words 4 Addresses</td>
<td>8 Words 4 Addresses</td>
</tr>
<tr>
<td>MMU/MPU</td>
<td>None</td>
<td>MPU</td>
<td>MPU</td>
<td>MPU</td>
<td>MPU</td>
<td>MPU</td>
</tr>
<tr>
<td>Hi Vectors</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Streaming</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Standby Mode</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### ARMv5 Cores:

<table>
<thead>
<tr>
<th></th>
<th>926EJ-S</th>
<th>946E-S</th>
<th>966E-S</th>
<th>968E-S</th>
<th>1026EJ-S</th>
<th>XScale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache</td>
<td>4-128K Instr 4-128K Data 8 words/line</td>
<td>0-1024K Instr 0-1024K Data 8 words/line</td>
<td>None</td>
<td>None</td>
<td>0-128K Instr 0-128K Data 8 words/line</td>
<td>32K Instr 32K Data 8 words/line</td>
</tr>
<tr>
<td>Associativity</td>
<td>4-way</td>
<td>4-way</td>
<td>N/A</td>
<td>N/A</td>
<td>4-way</td>
<td>32-way</td>
</tr>
<tr>
<td>TCM</td>
<td>0-1024K Instr 0-1024K Data</td>
<td>0-1024K Instr 0-1024K Data</td>
<td>0-64M Instr 0-64M Data</td>
<td>0-64M Instr 0-64M Data</td>
<td>0-1024K Instr 0-1024K Data</td>
<td>No</td>
</tr>
<tr>
<td>Write Strategy</td>
<td>Write Through Write Back</td>
<td>Write Through Write Back</td>
<td>Write Through Write Back</td>
<td>Write Through Write Back</td>
<td>Write Through Write Back</td>
<td>Write Through Write Back</td>
</tr>
<tr>
<td>Write Buffer</td>
<td>16 Words 4 Addresses</td>
<td>16 Words Data or Address</td>
<td>12 Words Data or Address</td>
<td>12 Words Data or Address</td>
<td>8 Words Data or Address</td>
<td>8 x 16 Bytes Coalescing</td>
</tr>
<tr>
<td>MMU/MPU</td>
<td>MMU</td>
<td>MPU</td>
<td>None</td>
<td>None</td>
<td>MMU or MPU</td>
<td>MMU With extensions</td>
</tr>
<tr>
<td>Hi Vectors</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Streaming</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Standby Mode</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### ARMv6 Cores:

<table>
<thead>
<tr>
<th></th>
<th>1136EJ(F)-S</th>
<th>1156T2(F)-S</th>
<th>1176JZ(F)-S</th>
<th>MPCore11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Harvard</td>
<td>Harvard</td>
<td>Harvard</td>
<td>Harvard</td>
</tr>
<tr>
<td>Cache</td>
<td>4-64K Instr 4-64K Data 8 words/line</td>
<td>0-64K Instr 0-64K Data 8 words/line</td>
<td>4-64K Instr 4-64K Data 8 words/line</td>
<td>16-64K Instr 16-64K Data 8 words/line</td>
</tr>
<tr>
<td>Associativity</td>
<td>4-way</td>
<td>4-way</td>
<td>4-way</td>
<td>4-way</td>
</tr>
<tr>
<td>TCM</td>
<td>0-64K Instr 0-64K Data</td>
<td>0-64K Instr 0-64K Data</td>
<td>0-64K Instr 0-64K Data</td>
<td>None</td>
</tr>
<tr>
<td>Replacement</td>
<td>Random Round Robin</td>
<td>Random Round Robin</td>
<td>Random Round Robin</td>
<td>Random Round Robin</td>
</tr>
<tr>
<td>Write Strategy</td>
<td>Write Through Write Back</td>
<td>Write Through Write Back</td>
<td>Write Through Write Back</td>
<td>Write Through Write Back</td>
</tr>
<tr>
<td>MMU/MPU</td>
<td>MMU</td>
<td>MPU</td>
<td>MMU</td>
<td>MMU</td>
</tr>
<tr>
<td>Hi Vectors</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Streaming</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>Standby Mode</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Bus</td>
<td>AHB/APB</td>
<td>AXI</td>
<td>AXI</td>
<td>AXI</td>
</tr>
<tr>
<td>VFP Support</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Cortex Cores:

<table>
<thead>
<tr>
<th></th>
<th>Cortex-M3</th>
<th>Cortex-M1</th>
<th>Cortex-R4</th>
<th>Cortex-A8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Harvard</td>
<td>Harvard</td>
<td>Harvard</td>
<td>Harvard</td>
</tr>
<tr>
<td>Cache</td>
<td>None</td>
<td>None</td>
<td>4-64K Instr</td>
<td>4-64K Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8 words/line</td>
<td>16 or 32 Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16 or 32 Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16 words/line</td>
</tr>
<tr>
<td>Associativity</td>
<td>N/A</td>
<td>N/A</td>
<td>4-way</td>
<td>4-way</td>
</tr>
<tr>
<td>TCM</td>
<td>None</td>
<td>0-1M Instr</td>
<td>0-8M Instr</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0-1M Data</td>
<td>0-8M Data</td>
<td></td>
</tr>
<tr>
<td>Replacements</td>
<td>N/A</td>
<td>N/A</td>
<td>Random</td>
<td>Random</td>
</tr>
<tr>
<td>Write Strategy</td>
<td>N/A</td>
<td>N/A</td>
<td>Write Through</td>
<td>Write Through</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Write Back</td>
<td>Write Back</td>
</tr>
<tr>
<td>MMU/MPU</td>
<td>MPU</td>
<td>None</td>
<td>MPU (optional)</td>
<td>MMU</td>
</tr>
<tr>
<td>Hi Vectors</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Streaming</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Standby Mode</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Bus</td>
<td>AHB Lite/APB</td>
<td>AHB Lite/APB</td>
<td>AXI</td>
<td>AXI</td>
</tr>
<tr>
<td>VFP Support</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

TrustZone Computing

- TrustZone adds a “parallel world” to allow trusted programs and data to be safely separated from the OS and applications
- Introduced for ARM1176, standard for ARmv7-A Cores
- **Features:**
  - New Secure Monitor Mode: gate-keeper for secure state
  - New S-bit in CP15 to indicate when the processor is running in a secured state
  - Security state exposed on external bus accesses to permit security-aware memory and peripherals
  - Ability to restrict debug to non-secure state
NEON Media Processor Features

- Single Instruction Multiple Data (SIMD) Media Processor
- Targets audio and video codecs, image and speech processing, graphics, baseband processing, and general signal processing
- 3 Processing pipelines: Integer/fixed point, single precision floating point, IEEE vector floating point
- Efficient data handling
  - Best use of available memory bandwidth
  - Eliminates data arrangement overhead
  - Operates on separate register file
  - SIMD Framework excellent target for compilers

End