Introduction to Embedded System Processor Architectures

Contents crafted by Professor Jari Nurmi
Tampere University of Technology Department of Computer Systems
Motivation – Why Processor Design?

Embedded systems emerging increasingly
- Processor-based devices
- Not user-programmable
- Consumer products, office automation, vehicles, robotics
- Mobile and ubiquitous computing
→ high demand for different kinds of processors

Embedded processors integrated in System-on-Chip
- Enabled by high integration capacity of modern technologies
  - From 100,000 to 1,000,000,000 transistors in past 15-17 years
- Low cost and low power consumption requirements
  - Avoidance of overheads in area, timing and power consumption
  - Flexibility requirement calls for programmable solutions
→ application-specific and customizable processors needed
Motivation – Why Processor Design? (cont’d)

FPGA technology advances
- Increased capacity allows integrating processors on FPGA
- Inherently avoiding ASIC back-end problems and NRE
- Use of most advanced technologies! (but carrying some overheads because of the generic chips)

→ relatively straightforward to run your own processor

Better tools for processor design, exploration, and support
- Architecture description and design space exploration
  - Compiler-centric approach (e.g. Target Compiler Technologies)
  - Architecture description language based flow (e.g. CoWare LISA)
  - Customizable base architecture approach (e.g. Tensilica Xtensa)
- Hardware generation and SW tool chain generation

→ easier entry for digital design and SW professionals
Components of an embedded computer

Embedded System / Computer

- Peripheral Devices
- I/O
- Processor
  - Control
  - Datapath
- Memory

Interconnections
The processor ISA is made up from

- **Instruction set** (including addressing modes and data types)
- Programmer’s view on the processor HW resources
  - GP registers
  - Special registers
  - Memory spaces

**Additionally, conventions of use, e.g.**

- Certain registers used as
  - subroutine arguments
  - return values
  - stack pointer
  - frame pointer
  - pointer to the global variable area
### Example: ARM 11 Register Set

#### ARM state general registers and program counter

<table>
<thead>
<tr>
<th>System and User</th>
<th>FIQ</th>
<th>Supervisor</th>
<th>Abort</th>
<th>IRQ</th>
<th>Undefined</th>
<th>Secure monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
</tr>
<tr>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
</tr>
<tr>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
</tr>
<tr>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
</tr>
<tr>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
</tr>
<tr>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
</tr>
<tr>
<td>r6</td>
<td>r6</td>
<td>r6</td>
<td>r6</td>
<td>r6</td>
<td>r6</td>
<td>r6</td>
</tr>
<tr>
<td>r7</td>
<td>r7</td>
<td>r7</td>
<td>r7</td>
<td>r7</td>
<td>r7</td>
<td>r7</td>
</tr>
<tr>
<td>r8</td>
<td>r8_fiq</td>
<td>r8</td>
<td>r8</td>
<td>r8</td>
<td>r8</td>
<td>r8</td>
</tr>
<tr>
<td>r9</td>
<td>r9_fiq</td>
<td>r9</td>
<td>r9</td>
<td>r9</td>
<td>r9</td>
<td>r9</td>
</tr>
<tr>
<td>r10</td>
<td>r10_fiq</td>
<td>r10</td>
<td>r10</td>
<td>r10</td>
<td>r10</td>
<td>r10</td>
</tr>
<tr>
<td>r11</td>
<td>r11_fiq</td>
<td>r11</td>
<td>r11</td>
<td>r11</td>
<td>r11</td>
<td>r11</td>
</tr>
<tr>
<td>r12</td>
<td>r12_fiq</td>
<td>r12</td>
<td>r12</td>
<td>r12</td>
<td>r12</td>
<td>r12</td>
</tr>
<tr>
<td>r13</td>
<td>r13_fiq</td>
<td>r13_svc</td>
<td>r13_svc</td>
<td>r13_irq</td>
<td>r13_un</td>
<td>r13_mon</td>
</tr>
<tr>
<td>r14</td>
<td>r14_fiq</td>
<td>r14_svc</td>
<td>r14_svc</td>
<td>r14_irq</td>
<td>r14_un</td>
<td>r14_mon</td>
</tr>
<tr>
<td>r15</td>
<td>r15(PC)</td>
<td>r15(PC)</td>
<td>r15(PC)</td>
<td>r15(PC)</td>
<td>r15(PC)</td>
<td>r15(PC)</td>
</tr>
</tbody>
</table>

#### ARM state program status registers

<table>
<thead>
<tr>
<th>CPSR</th>
<th>CPSR</th>
<th>CPSR</th>
<th>CPSR</th>
<th>CPSR</th>
<th>CPSR</th>
<th>CPSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPSR_fiq</td>
<td>SPSR_svc</td>
<td>SPSR_abt</td>
<td>SPSR_irq</td>
<td>SPSR_un</td>
<td>SPSR_mon</td>
<td></td>
</tr>
</tbody>
</table>

= banked register

*Copyright © 2004-2007 ARM Limited.*
Instruction set

Different types of executable operations
- arithmetic and logic instructions
- load and store instructions
- change of flow (control)
- miscellaneous (system calls, mode setting, etc.)
- vector or multimedia instructions
- other (application) specific instructions

Different architectural styles
- Single operation per instruction
- Compound instructions (multiple operations)

Different extents
- Simple / reduced instruction sets
- Extensive / rich / complex instruction sets
Addressing Modes

<table>
<thead>
<tr>
<th>Addressing Mode</th>
<th>Instruction</th>
<th>Address operations</th>
<th>Final operand or instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>immediate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>register direct</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>register indirect</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>offset (base)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>indexed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>post-increment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pseudo-direct</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Instruction Address operations:
- Reg. File
- Memory
- PC

Final operand or instruction:
- Extension
Special Addressing Modes, Data Representation

DSP addressing modes
- Modulo addressing (for circular buffer implementation)
- Bit-reverse addressing (for FFT indexing)

A binary number – a bunch of bits can be interpreted as...
- A bunch of bits!
- unsigned integer
- signed integer
- signed fraction
- floating-point number
- character string
- binary-coded decimal (BCD) number
- Grey coded number

Different number of bits may be used, e.g.
- double words, words, half-words, and bytes
- single and double precision floating-point numbers
Operand convention is one way to classify architecture styles

- Stack
- Accumulator
- (GP) Register
  - load-store
  - RISC architecture
    - 2-register
    - 3-register
- Register-memory
- Memory-memory
A Reduced Instruction Set Computer architecture

- Provides basic primitives, not “complete solutions” as instructions; this leads to the “reduced” instruction set
- Orthogonality and regularity in the instructions as much as possible (in real life compromises have to be made)
- Single-cycle execution of most instructions
- Easy to pipeline
- A lot of general purpose registers
- Arithmetic and logic operations (and address computation) are done for register operands or immediates (the load-store architecture principle)

Practically all modern general-purpose computing and most of the embedded processor architectures are heavily based on the RISC paradigm (except maybe traditional DSP processors)
Parallelism

Pipelining
  • the most applied approach for parallelism
  • allows parallel use of resources in different pipeline stages

Other forms of parallelism include
  • ILP, Instruction Level Parallelism
  • DLP, Data Level Parallelism
  • TLP, Thread Level Parallelism
  • PLP, Process(or) Level Parallelism
Instruction-Level Parallelism Approaches – what is done at compile time vs. in hardware

- Application
  - Frontend
    - Determine dependencies
      - Determine independencies
        - Bind function units
        - Bind datapaths
          - Compile time
  - Sequential (superscalar)
    - Dependence (dataflow)
      - Independence (EPIC)
        - Independence (VLIW)
          - Independence (TTA)
            - Execute
          - Runtime
Data-Level Parallelism

Short vector processing by splitting datapath

64-bit

4 x 16-bit

2 x 32-bit

8 x 8-bit
### Thread-Level Parallelism - Multithreading

<table>
<thead>
<tr>
<th>coarse-grain</th>
<th>fine-grain</th>
<th>simultaneous</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="c) Matrix" /></td>
<td><img src="image" alt="b) Matrix" /></td>
<td><img src="image" alt="a) Matrix" /></td>
</tr>
</tbody>
</table>
Process-Level Parallelism

Processes are allocated on multiple processors or cores

MIMD (Multiple Instruction, Multiple Data streams) in the classical Flynn classification

Relatively little communication between processes

However, some synchronization points may be needed

Cache coherence is the major concern when distributing processing on multiple autonomous processor subsystems
  - E.g., multi-core processors often share L2 or L3 level cache
  - More about caches to follow...
Physical implementations of memory vary

- Register
- SRAM
- ROM (non-volatile)
- Flash, EEPROM (non-volatile)
- DRAM (SDRAM, DDR, Rambus)

The speed varies, too

- Register, SRAM, DRAM, Flash/EEPROM (fastest to slowest)
- Large memory is slower and more expensive than a small one
  → Memory hierarchy
Illusion of a large AND fast memory by using a hierarchy of memories

- Based on principle of locality
  - Temporal locality
  - Spatial locality
- Lower levels copied automatically upwards when needed
- Misses occur relatively seldom
Cache Misses

**Misses** (the item is not found on the particular level but needs to be loaded from the lower level) **and the time penalty associated with them are the main factor lowering the memory subsystem performance; the factors are**

- Miss rate (or its complement, hit rate)
- Miss penalty
- Hit access time

**Misses can be classified to**

- Compulsory (in ”cold start”) – partial cure: larger block size
- Capacity – cure: larger cache size
- Conflict misses – cure: larger size and/or associativity (see →)
Cache Operation and Design Parameters

Cache organized in cache lines
- BLOCK of B words of payload (B ≥1 is a design parameter)
- Tag to identify the lower level address associated with payload
- Valid bit
- Possibly other bits for replacement or coherence information

Cache design parameters include
- Total size or capacity C
- Block size B
- Associativity, N ways, N=C/B (see →)
- Block replacement policy
- Write policy
- Allocate-on-write-miss policy
Associativity

Direct-mapped (1-way set associative)
- a single location for each data to be mapped directly according to the least significant part of the address (address mod cache-size).

Set-associative
- the memory has been divided into two or more ways. The data can be mapped into one of these ways according to a (re)placement algorithm.
- the set (cache area where a particular memory block can be placed) is a cross-section of the ways at the address formed by the modulus of the address and the way size. So, only one Nth of the cache indices are used in an N-way set-associative cache compared to a direct-mapped one with the same capacity.

Fully associative (∞-way set associative)
- cache the data can be placed anywhere. This sounds like a good idea, but it also means that every location has to be searched when accessing the cache.

Tag comparison is done to define if the item searched from the cache is there (in one of the ways). From 1 to N (# of ways) comparisons!
Block Replacement Policies

Others than direct-mapped caches need a block replacement policy
• Direct-mapped always replaces the single possible one!

The possible choices include
• Random
• Least Recently Used (LRU)
• Pseudo-LRU
• Least Frequently Used (LFU)
• First-In-First-Out (FIFO).

All of these except the random replacement need some book-keeping to be done in the status bits of each block
• A simple method in two-way set-associative caches is to maintain a LRU-bit for each block in cache. When accessing the block, the corresponding LRU bit is set and in the other block’s LRU-bit (within the same set) it is cleared.
• To simplify this, the bits may be cleared every now and then by the operating system or cache controller and only the single bit is set in the block accessed (pseudo-LRU). The least recently used one of the blocks may be also encoded to a common status section such that only $\log_2(N)$ bits are needed for $N$ ways of the cache.
• LFU requires an access counter for each cache block and is thus a bit more expensive solution.
• FIFO does not take into account the access patterns but replaces always the oldest block.
• Random placement: simple but relatively good performance.
Write Allocation and Allocate-on-Write Miss?

Write policies

- Write-back
  - modified block is not written to the lower hierarchy level until it is replaced
  - reduces the bus traffic to the memory
  - a *dirty* bit is used to mark an updated block
  - drawback: increased miss penalty (when a dirty block is replaced)
  - drawback: in multi-processor systems cache-coherence protocols and cache-to-cache data submission are needed to provide the most current version of the block for the cache updates

- Write-through
  - the next level is kept up-to-date
  - drawback: slow to the main memory would stall the processor in case of frequent subsequent writes
  - a **write buffer** is used to avoid this

Allocate block on write miss?

- Yes: increases the write miss penalty, especially if a write buffer is used
- No: typical in write-back caches (large penalty), requires a write buffer!
In embedded systems, different local storage solutions have also remained instead / in parallel with caches:

- A *scratchpad* memory can be used to store temporarily data or intermediate results. It is a fast local memory which can be also implemented as a part of the data cache in hierarchical memory systems. (e.g. ARM terminology uses Tightly Coupled Memory in a similar context)

- Compiler Controlled Memory (CCM) has been introduced especially to hold values spilled out from the register file due to register re-allocation. CCM is a fast local memory in a separate address space, taking the compiler-induced memory traffic out of the memory hierarchy.
Virtual memory separates the logical address space from the physical memory implemented in the system. The address space is divided into pages, which is a lot larger than a cache block. Segments can be used instead of (or sometimes on top of) pages. Segments can be of any size, while the page size is a fixed system parameter. The mapping is conventionally done by using page tables in memory.

- A part of the virtual address is the virtual page number (VPN), which is used to acquire the physical page number (PPN) from the page table.
- The lower end of the virtual address is typically the offset within the page directly.

We notice that one memory operation now requires two memory accesses!
Translation Lookaside Buffer (TLB)
- contains the virtual–physical address pairs
- a small associative cache memory, where the virtual page number forms the tag and the physical page number is the “data”

Virtual or physical addresses to be used in cache?
- physical addressing of cache
  - requires sequential TLB lookup – cache access (→ slow)
- virtual addressing of cache
  - allows accessing cache in parallel with TLB lookup
  - requires additional process tags or invalidating cache blocks at context switch!

- virtually indexed, physically tagged cache is a good compromise
  - allows making TLB lookup in parallel with (L1) cache
  - requires that the index to cache is within the part of the address that is equal between the two memory spaces (virtual and physical).
  - in direct mapped caches this means that the cache size cannot be any larger than the page size
  - In set-associative caches each of the ways must stay below this limit, allowing for larger total capacity
Buses are the conventional interconnect in embedded processors

- Drawback: limited to a single data transfer at a time
- Circumvented by using multiple buses or wider buses

Crossbar switch can be used to replace the bus

- Allows multiple simultaneous transfers, e.g. processor and Direct Memory Access (DMA) controller transferring data at the same time
- Grows in complexity in proportion to the number of inputs and outputs required

Network-on-Chip (or network, in general)

- Separates communication from computation
- Can be used mainly in multiprocessor systems for system-wide communication
- Allows high bandwidth but typically increases the latency of a single transfer
- Grows in proportion to the number of routing nodes in the network and to the degree (connectivity) of the network
**Input/Output Operations**

**Memory-mapped**
- I/O becomes loads and stores to an address in (one of) the processor data address space(s)

**Explicit**
- Special commands are used to input or output some data from/to a certain device

**Interrupt-driven**
- Data is acquired in an interrupt service routine evoked by an interrupt originating from, e.g., A/D converter clock, FIFO full signal, or some external device interrupt request

**Polling based**
- A certain part of a program sets attempts to acquire data
- A timer (interrupt!) might be also used to time the polling
The actual devices carrying out I/O operations with the processor include:

- Timers
- Different type of serial and parallel interfaces, such as
  - RS-232
  - Universal Serial Bus (USB)
  - Firewire
  - Peripheral Component Interconnect (PCI)
  - Small Computer System Interconnect (SCSI)
- Interrupt controller
- Direct Memory Access (DMA) controller
- Analog-to-Digital (A/D) and Digital-to-Analog (D/A) converters
Putting it all together: ARM1176JZF-S

ARM, Thumb, Jazelle (Java) instruction sets, DSP extensions and SIMD (Single Instruction Multiple Data) media processing extensions
Enhanced system security (TrustZone technology)
In-order single issue with 3 parallel 8-stage pipelines
High performance memory system
  • Extended Harvard architecture
  • Supports 4-64k cache sizes
  • Optional tightly coupled memories with DMA for multi-media apps
  • Quad-ported AMBA 3 AXI bus interface
  • ARMv6 memory system architecture accelerates OS context-switch
Optional Vector Floating Point coprocessor

Copyright © 2004-2007 ARM Limited.
ARM 11JZF-S Memory Subsystem

Level 1 memory
- Virtually indexed 4-way set associative data cache + MicroTLB + write buffer + TCM
- Similar instruction cache + MicroTLB+ TCM
- MMU with main TLB

Level 2 interface
- Several ports to AMBA AXI protocol based bus
  - 64-bit instruction port (up to 2 outstanding read accesses)
  - 64-bit data port (up to 2 outstanding read and 2 write accesses)
  - 64-bit DMA port
  - 32-bit peripheral port
### ARM 11JZ-S Integer Core Pipeline

**8-stage pipeline**

**3 parallel pipes: integer, MAC, Load-Store**

- **Fe1** First stage of instruction fetch where address is issued to memory and data returns from memory.
- **Fe2** Second stage of instruction fetch and branch prediction.
- **De** Instruction decode.
- **Iss** Register read and instruction issue.
- **Sh** Shifter stage.
- **ALU** Main integer operation calculation.
- **Sat** Pipeline stage to enable saturation of integer results.
- **WBex** Write back of data from the multiply or main execution pipelines.
- **MAC1** First stage of the multiply-accumulate pipeline.
- **MAC2** Second stage of the multiply-accumulate pipeline.
- **MAC3** Third stage of the multiply-accumulate pipeline.
- **ADD** Address generation stage.
- **DC1** First stage of data cache access.
- **DC2** Second stage of data cache access.
- **WBls** Write back from the Load Store Unit.

---

**Copyright © 2004-2007 ARM Limited.**
Embedded processor architectures features overviewed

- ISA (instructions, addressing modes, registers,...)
- Organization (RISC, others)
- Parallelism (pipelining, ILP, DLP, TLP, PLP)
- Memory subsystem (hierarchy, special memories, VM)
- Interconnects
- I/O, peripherals

Concluding example

- ARM 1176JZ-S
- Memory subsystem
- Pipeline

Next

- How NOT to design a processor