Embedded System Hardware
- Processing (Part II)–

Jian-Jia Chen
(Slides are based on Peter Marwedel)
Informatik 12
TU Dortmund
Germany

These slides use Microsoft clip arts. Microsoft copyright restrictions apply.
Embedded System Hardware

Embedded system hardware is frequently used in a loop (**"hardware in a loop"**):

![Diagram](image)

- **A/D converter**
- **Sample-and-hold**
- **Sensors**
- **Information processing**
- **Display**
- **D/A converter**
- **Actuators**
- **(Physical) environment**
- **Cyber-physical systems**
Key requirement #1: Code-size efficiency

- **Overview:** [http://www-perso.iro.umontreal.ca/~latendre/codeCompression/codeCompression/node1.html](http://www-perso.iro.umontreal.ca/~latendre/codeCompression/codeCompression/node1.html)

- **Compression techniques:** key idea
Key requirement #2: Run-time efficiency
- Domain-oriented architectures -

Example: Filtering in Digital signal processing (DSP)

\[
x_S = \sum_{k=0}^{n-1} w_{S-k} \ast a_k
\]

Signal at \( t=t_s \) (sampling points)
### Key requirement #3: energy-efficient and power efficient

<table>
<thead>
<tr>
<th>Execution platform</th>
<th>Relevant during use?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Plugged</td>
</tr>
<tr>
<td>E.g.</td>
<td>Factory</td>
</tr>
<tr>
<td>Global warming</td>
<td>✓</td>
</tr>
<tr>
<td>Cost of energy</td>
<td>✓</td>
</tr>
<tr>
<td>Increasing performance</td>
<td>✓</td>
</tr>
<tr>
<td>Problems with cooling, avoiding hot spots</td>
<td>✓</td>
</tr>
<tr>
<td>Avoiding high currents &amp; metal migration</td>
<td>✓</td>
</tr>
<tr>
<td>Reliability</td>
<td>✓</td>
</tr>
<tr>
<td>Energy a very scarce resource</td>
<td>☐</td>
</tr>
</tbody>
</table>

- Global warming: Relevant during use (Plugged: yes, Unplugged: no).
- Cost of energy: Relevant during use (Plugged: yes, Unplugged: no).
- Problems with cooling, avoiding hot spots: Relevant during use (Plugged: yes, Unplugged: yes).
- Avoiding high currents & metal migration: Relevant during use (Plugged: yes, Unplugged: yes).
- Reliability: Relevant during use (Plugged: yes, Unplugged: yes).
- Energy a very scarce resource: Not relevant during use (Plugged: no, Unplugged: yes).
Key Requirement #4: Real-time capability

- **Timing behavior has to be predictable**

  Features that cause problems:

  - Unpredictable access to shared resources
    - Caches with difficult to predict replacement strategies
    - Unified caches (conflicts between instructions and data)
  - Pipelines with difficult to predict stall cycles ("bubbles")
  - Unpredictable communication times for multiprocessors
  - Branch prediction, speculative execution
  - Interrupts that are possible any time
  - Memory refreshes that are possible any time
  - Instructions that have data-dependent execution times

  Trying to avoid as many of these as possible.
DSP and Multimedia Processors

These slides use Microsoft clip arts. Microsoft copyright restrictions apply.
Saturating arithmetic

- Returns largest/smallest number in case of over/underflows
- Example:

\[
\begin{align*}
  a & \quad 0111 \\
  b & \quad + \quad 1001 \\
\end{align*}
\]

- standard wrap around arithmetic \((1)0000\)
- saturating arithmetic \(1111\)

\[
\begin{align*}
  (a+b)/2: & \quad \text{correct} \quad 1000 \\
  \quad \text{wrap around arithmetic} & \quad 0000 \\
  \quad \text{saturating arithmetic + shifted} & \quad 0111 & \text{“almost correct”}
\end{align*}
\]

- Appropriate for DSP/multimedia applications:
  - No timeliness of results if interrupts are generated for overflows
  - Precise values less important
  - Wrap around arithmetic would be worse.
On-Site Exercise

Suppose that the numbers in the following table are given. Please compute the result of the indicated operation using wrap-around and saturating arithmetic!

<table>
<thead>
<tr>
<th></th>
<th>(un)signed</th>
<th>bitvector</th>
<th></th>
<th>(un)signed</th>
<th>bitvector</th>
<th>op</th>
<th></th>
<th>a op b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>signed</td>
<td>00110₂</td>
<td></td>
<td>signed</td>
<td>00110₂</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>signed</td>
<td>01110₂</td>
<td></td>
<td>signed</td>
<td>01110₂</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>signed</td>
<td>11110₂</td>
<td></td>
<td>signed</td>
<td>11110₂</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>signed</td>
<td>01010₂</td>
<td></td>
<td>signed</td>
<td>01010₂</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned</td>
<td>10110₂</td>
<td></td>
<td>unsigned</td>
<td>10110₂</td>
<td>+</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fixed-point arithmetic

Shifting required after multiplications and divisions in order to maintain binary point.
On-Site Exercise

Suppose that the fixed-point numbers in the following table are given. Numbers are assumed to be signed. Please compute the result of the indicated operation!

<table>
<thead>
<tr>
<th>a</th>
<th>bitvector</th>
<th>b</th>
<th>bitvector</th>
<th>op</th>
<th>a op b</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3,2)</td>
<td>00110₂</td>
<td>(3,2)</td>
<td>00110₂</td>
<td>+</td>
<td>(3,2)</td>
</tr>
<tr>
<td>(3,2)</td>
<td>00110₂</td>
<td>(3,2)</td>
<td>00110₂</td>
<td>+</td>
<td>(4,1)</td>
</tr>
<tr>
<td>(3,2)</td>
<td>00110₂</td>
<td>(4,1)</td>
<td>00110₂</td>
<td>+</td>
<td>(4,1)</td>
</tr>
<tr>
<td>(3,2)</td>
<td>00010₂</td>
<td>(4,1)</td>
<td>00010₂</td>
<td>+</td>
<td>(2,1)</td>
</tr>
<tr>
<td>(3,2)</td>
<td>00110₂</td>
<td>(4,1)</td>
<td>00110₂</td>
<td>*</td>
<td>(8,2)</td>
</tr>
</tbody>
</table>
Multimedia-Instructions, Short vector extensions, Streaming extensions, SIMD instructions

- Multimedia instructions exploit that many registers, adders etc are quite wide (32/64 bit), whereas most multimedia data types are narrow

- 2-8 values can be stored per register and added. E.g.:

```
+----------------+----------------+
| a1             | a2             |
+----------------+----------------+
| b1             | b2             |
+----------------+----------------+
| c1             | c2             |
```

- 2 additions per instruction; no carry at bit 16

- Cheap way of using parallelism

- SSE instruction set extensions, SIMD instructions
Key idea of very long instruction word (VLIW) computers (1)

- Instructions included in long instruction packets.
- Multiple instructions are assumed to be executed in parallel.
- Fixed association of packet bits with functional units.

- Compiler is assumed to generate these “parallel” packets
Key idea of very long instruction word (VLIW) computers (2)

- Complexity of finding parallelism is moved from the hardware (RISC/CISC processors) to the compiler;

- Ideally, this avoids the overhead (silicon, energy, ..) of identifying parallelism at run-time.

- A lot of expectations into VLIW machines

- However, possibly low code efficiency, due to many NOPs

- Explicitly parallel instruction set computers (EPICs) are an extension of VLIW architectures: parallelism detected by compiler, but no need to encode parallelism in 1 word.
EPIC: TMS 320C6xxx as an example

1 Bit per instruction encodes end of parallel exec.

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>B C D</td>
</tr>
<tr>
<td>3</td>
<td>E F G</td>
</tr>
</tbody>
</table>

Instructions B, C and D use disjoint functional units, cross paths and other data path resources. The same is also true for E, F and G.
Partitioned register files

- Many memory ports are required to supply enough operands per cycle.
- Memories with many ports are expensive.

Registers are partitioned into (typically 2) sets, e.g. for TI C6xxx:
Microcontrollers
- MHS 80C51 as an example -

- 8-bit CPU optimised for control applications
- Extensive Boolean processing capabilities
- 64 k Program Memory address space
- 64 k Data Memory address space
- 4 k bytes of on chip Program Memory
- 128 bytes of on chip data RAM
- 32 bi-directional and individually addressable I/O lines
- Two 16-bit timers/counters
- Full duplex UART
- 6 sources/5-vector interrupt structure with 2 priority levels
- On chip clock oscillators
- Very popular CPU with many different variations
Energy Efficiency of FPGAs
Reconfigurable Logic

Custom HW may be too expensive, SW too slow.
Combine the speed of HW with the flexibility of SW

- HW with programmable functions and interconnect.
- Use of configurable hardware;
  common form: field programmable gate arrays (FPGAs)

Applications:
- algorithms like de/encryption,
- pattern matching in bioinformatics,
- high speed event filtering (high energy physics),
- high speed special purpose hardware.

Very popular devices from
- XILINX, Actel, Altera and others
Fine-Grained Reconfigurable Fabric

**CLB:** Configurable Logic Block

**PSM:** Programmable Switch Matrix

Additionally: I/O Blocks, RAM Blocks, Multiplier, CPUs, …

src: Xilinx Data Sheets
Fine-Grained Reconfigurable Fabric

Logic Layer: Perform Computations
Configuration Layer: Which Computations to Perform
Partial Reconfiguration: Only parts of the Fabric are reconfigured

src: Kalenteridis et al. “A complete platform and toolset for system implementation on fine-grained reconfigurable hardware”, Microprocessors and Microsystems 2004
Coarse-Grained Reconfigurable Fabric

Pros:
• Reduced Area
• Higher Frequency
Reduced reconfiguration latency

Cons
• Reduced Flexibility
• Area Wastage
→ In case computations do not suite the fabric (e.g. bit-level operations, bit shuffling)

src: PACT’s XPP 64-A1 architecture
Reconfigurable Computing: an example

Runtime reconfigurable processor:

+ Efficient use of hardware: through high degree of parallelism, multiplexed used of reconf. fabric => energy efficient

- Reconfiguration Latency and Energy
Memory

These slides use Microsoft clip arts. Microsoft copyright restrictions apply.
For the memory, efficiency is again a concern:

- capacity
- energy efficiency
- speed (latency and throughput); predictable timing
- size
- cost
- other attributes (volatile vs. persistent, etc)
Memory capacities expected to keep increasing

Figure ORTC2  ITRS Product Function Size Trends:
MPU Logic Gate Size (4-transistor); Memory Cell Size [SRAM (6-transistor); Flash (SLC and MLC), and DRAM (transistor + capacitor)] -- Updated
Where is the power consumed? - Stationary systems -


- Switching (dynamic) power, logic dominating

- Overall power consumption a nightmare for environmentalists
Where is the power consumed?  
- Consumer portable systems -


- Based on current trends

- Memory and logic, static and dynamic relevant
Energy consumption and access times of memories

Example CACTI: Scratchpad (SRAM) vs. DRAM (DDR2):

16 bit read; size in bytes;
65 nm for SRAM, 80 nm for DRAM

Source: Olivera Jovanovic, TU Dortmund, 2011
Access times and energy consumption for multi-ported register files

Cycle Time (ns)  Area ($\lambda^2 \times 10^6$)  Power (W)

![Graphs showing cycle time, area, and power consumption for different register file sizes and port counts.](image)

Source and © H. Valero, 2001
Trends for the Speeds

Speed gap between processor and main DRAM increases

Similar problems also for embedded systems & MPSoCs

- Memory access times
- >> processor cycle times
- “Memory wall” problem

[P. Machanik: Approaches to Addressing the Memory Wall, TR Nov. 2002, U. Brisbane]
However, clock speed increases have come to a halt.

The speed gap does not disappear – may be, it is not getting larger any more.
Memory Hierarchy

- **Register (SRAM)**
- **SRAM (Cache or Scratchpad)**
- **Main Memory (DRAM)**
- **File Cache (DRAM, Flash)**
- **Secondary Storage (Disks, Flash)**
- **Tertiary Storage (Optische Speicher, Bänder)**

**persistent**
Recover Your Knowledge on Cache

- Fully-associative mapping, direct mapping, set-associative mapping
- Cache replacement policy
- Temporal locality and spatial locality
Set-associative cache $n$-way cache

$|\text{Set}| = 2$

Address

Tag | Index
---|---
way 0 | $(\text{€})$
way 1 |

$\geq 1$

Data
Hierarchical memories
using scratch pad memories (SPM)

SPM is a small, physically separate memory mapped into the address space

Address space

0

scratch pad memory

FFF...

no tag memory

Examples:

- Most ARM cores allow tightly coupled memories
- IBM Cell
- Infineon TriCore
- Many multi-cores, due to high costs of coherent caches

Hierarchy

main

SPM

processor

Selection is by an appropriate address decoder (simple!)
Influence of the associativity

[P. Marwedel et al., ASPDAC, 2004]
Summary

- Processing
  - VLIW/EPIC processors
  - MPSoCs
- FPGAs
- Memories
  - “Small is beautiful”
  - (in terms of energy consumption, access times, size)
Key requirement #3: Run-time efficiency
- Domain-oriented architectures -

Example: Filtering in Digital signal processing (DSP)

\[ x_s = \sum_{k=0}^{n-1} w_{s-k} * a_k \]

Signal at \( t = t_s \) (sampling points)
Filtering in digital signal processing

\[ x_s = \sum_{k=0}^{n-1} w_{s-k} * a_k \]

-- outer loop over
-- sampling times \( t_s \)

\{ MR:=0; A1:=1; A2:=s-1; \\
MX:=w[s]; MY:=a[0]; \\
for (k=0; k <= (n-1); k++) \\
{ MR:=MR + MX * MY; \\
MX:=w[A2]; MY:=a[A1]; \\
A1++; A2--; \\
} \\
x[s]:=MR; \}

Maps nicely
DSP-Processors: multiply/accumulate (MAC) and zero-overhead loop (ZOL) instructions

MR:=0; A1:=1; A2:=s-1; MX:=w[s]; MY:=a[0];

for ( k:=0 <= n-1 )
{MR:=MR+MX*MY; MY:=a[A1]; MX:=w[A2]; A1++; A2--}

Multiply/accumulate (MAC) instruction

Zero-overhead loop (ZOL) instruction preceding MAC instruction.
Loop testing done in parallel to MAC operations.
Heterogeneous registers

Example (ADSP 210x):

Address registers
A0, A1, A2

Address generation unit (AGU)

Different functionality of registers An, AX, AY, AF, MX, MY, MF, MR
Separate address generation units (AGUs)

Example (ADSP 210x):

- Data memory can only be fetched with address contained in A,
- but this can be done in parallel with operation in main data path (takes effectively 0 time).
- \( A := A \pm 1 \) also takes 0 time,
- same for \( A := A \pm M \);
- \( A := \text{<immediate in instruction>} \) requires extra instruction
  - Minimize load immediates
  - Optimization in optimization chapter
Modulo addressing:

Am++ \equiv Am:=(Am+1) \mod n

(implements ring or circular buffer in memory)

$n$ most recent values

\[
\begin{cases}
  \ldots \\
  w[t_1-1] \\
  w[t_1] \\
  w[t_1-n+1] \\
  w[t_1-n+2] \\
  \ldots 
\end{cases}
\]

sliding window

Memory, $t=t_1$

\[
\begin{align*}
  w & \rightarrow w[t_1-1] \\
  w & \rightarrow w[t_1] \\
  w & \rightarrow w[t_1-n+1] \\
  w & \rightarrow w[t_1-n+2] \\
  \ldots
\end{align*}
\]

Memory, $t_2=t_1+1$
Cycles/access as a function of the size of the list

Figure 3.10: Sequential Read Access, NPAD=0

* prefetching succeeds

§ prefetching fails
Impact of TLB misses and larger caches

Elements on different pages; run time increase when exceeding the size of the TLB

Larger caches are shifting the steps to the right

Figure 3.14: Advantage of Larger L2/L3 Caches
Comparison of currents using measurements

E.g.: ATMEL board with ARM7TDMI and ext. SRAM

![Image of ATMEL board]

<table>
<thead>
<tr>
<th>Current Source</th>
<th>Main Memory Current (mA)</th>
<th>Core+SPM (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prog Main/ Data Main</td>
<td>116</td>
<td>48.2</td>
</tr>
<tr>
<td>Prog Main/ Data SPM</td>
<td>77.2</td>
<td>50.9</td>
</tr>
<tr>
<td>Prog SPM/ Data Main</td>
<td>44.4</td>
<td>44.4</td>
</tr>
<tr>
<td>Prog SPM/ Data SPM</td>
<td>53.1</td>
<td>1.16</td>
</tr>
</tbody>
</table>

32 Bit-Load Instruction (Thumb)
Why not just use a cache?

2. Energy for parallel access of sets, in comparators, muxes.

[R. Banakar, S. Steinke, B.-S. Lee, 2001]
Influence of the associativity

Parameters different from previous slides

[P. Marwedel et al., ASPDAC, 2004]