Overview

- Introduction to Hardware-Software Co-design
- Hardware Options for Embedded Systems
- CPU with an Accelerator
- CPU-Accelerator Co-design Process
Embedded System Design

Real-time System Design
- Performance analysis
- Scheduling and allocation

Accelerated systems
Use additional computational unit dedicated to some functions?
- Hardwired Logic e.g. FPGA
- Multiple processing elements (PEs) or an extra CPU

Hardware/software co-design: joint design of hardware and software architectures
Traditional Design Practice

Performance Requirements make it impossible to execute entire application in software.

Intensive portions are extracted and realized as custom hardware.

Early Design Cycle Partitioning
- Design Space is not fully Explored
- High Cost Design
- Inefficient
Traditional Design Practice

- Requirements Definition
- Architecture Definition
- Hardware Design
- Software Design
- Interface Design
- Hardware Manufacture and Test
- Software Code and Test
- Hardware Software Implementation and Test
- Deliverables: Documentation, Deployment
- Field Test
- Documentation
- Deployment
- Field Test

Timeframes:
- Requirements Definition: 6-12 Months
- Architecture Definition: 6-12 Months
- Hardware Design: 25-49 Months
- Software Design: 25-49 Months
- Interface Design: 25-49 Months
- Hardware Manufacture and Test: 25-49 Months
- Software Code and Test: 25-49 Months
- Hardware Software Implementation and Test: 25-49 Months
- Field Test: 6-12 Months
- Deployment: 6-12 Months
- Documentation: 6-12 Months
Advancements

VLSI Technology
- Smaller, Faster IP Cores
- Reconfigurable Logic

Matured Hardware Design Methodology

Matured Software Design Methodology

Joint design – Still in Infancy!
Hardware Software Codesign

An approach utilizing the maximum efficiency of Hardware and Software is needed

Recent developments in CAD Tools

Result -- Hardware Software Codesign

- A unified approach
- Large Design Space Exploration
- Improved Time to Market
Codesign Methodology

1. Analysis of Constraints and Requirements
   - System Specs

2. Hardware Software Partitioning
   - Hardware Description
     - Hardware Synthesis and Configuration
       - Configuration Modules
       - Hardware Components
   - Software Description
     - Interface Synthesis
     - Software Gen. and Parameterization
       - HW/SW Interfaces
       - Software Modules

3. HW/SW Integration and Cosimulation

4. Integrated System
   - System Evaluation
   - Design Verification
Hardware-Software Codesign

Study of the design of embedded computer systems encompassing the following parts:

- **Co-Specification**
  Developing system specification that describes hardware and software modules and relationship among them.

- **Co-Synthesis**
  Automatic and semi-automatic design of hardware and software elements to meet the specification.

- **Co-Simulation**
  Simultaneous simulation of hardware and software.

Hardware Software Co-Verification

Hardware Software Co-Estimation
HW/SW Co-Specification

System functionality at an Abstract Level
No Concept of Hardware or Software
Must Capture System functionality precisely

Common Specification Approaches

- **Using High Level Languages**
  - C/C++
    - Inclined to Software description
  - VHDL
    - Inclined to Hardware description
  - System C
    - Combined features for Hardware and Software Representation
Common Specification Approaches

Using Task Graphs

- Grey Area
  - High Level descriptions are usually translated to some form of a task graph

- Common Forms
  - Data flow graphs
  - Control flow Graphs
  - Control-Data Flow Graphs
Hardware Software Partitioning

Assignment of System parts to implementation units (Hardware and Software)

Goals of Partitioning Algorithm

- Meet constraints (Timing)
- Minimize cost (Area, Power)

Directly affects the cost and performance of final system

**Major problem: Computation time**

Significant Research Area
Hardware Software Partitioning

Granularity: Size of System Components assigned to Hardware or Software

Coarse Grain Approaches

Assign Complete Function or Processes to hardware or software

• A single task is a large block of system functionality
• Prevent Excess Communications
• Example tasks: MPEG decode, JPEG-2000 encode, FFT, DCT, etc.
• Also referred to as High Level Granularity
Hardware Software Partitioning

Fine Grain Approaches: Operate on basic operations (add, subtract, multiply etc.)

Avoid the problem of dealing with poorly defined functional specifications

• Very useful when dealing with partially re-configurable processors (IP cores that can be modified during the design process)
  • Here tasks are often referred to as base blocks
  • Also referred to as Low Level Granularity
Flexible Granularity: Computationally intensive parts are small loops which are hidden inside a function or process

- Coarse grain approaches provide costly designs, some redundant parts may be moved to hardware
- Fine Grain approaches make up such a large design space which is usually very hard to explore
- Perform HW/SW partitioning handling decisions at both high and low levels
- Complicated problem compared to high or low level partitioning
Hardware Software Partitioning

Granularity Effects

Granularity
- Process
- Function *(Global Data)*
- Basic Block *(Local Data)*
- Operation *(Statements/Variables)*

Optimization Potential

Communication Overhead

Design Effort
Partitioning Common Approaches

Optimal Partitioning Approaches

**Exhaustive**
- Computationally very Intensive
- Limited to small task graphs

**Branch and Bound**
- Start from a good Initial Condition
- Try pre-sorted combinations
- Exhaustive in limiting case

Heuristic Approaches

**Heuristic Optimization Techniques**
- Simulated Annealing, Tabu Search and Genetic Algorithm

**Greedy Approaches:** Start from all SW or HW and move parts to HW or SW
Hardware-Software Co-Synthesis

Four Principle Phases of Co-synthesis:

- **Partitioning**
  Dividing the functionality of an embedded system into units of computation.

- **Scheduling - Pipelining**
  Task Start Timing: Choosing time at which various computation units will occur.

- **Allocation**
  Determining the processing elements (PEs) on which computations will occur.
  
  Selection, type of Processing Elements and Communication Structure (System Architecture)

- **Assignment**
  Task Mapping: Choosing particular component types for the allocated units (of computations).
Hardware Software Co-Synthesis

Automatic System Architecture definition (e.g., bus, tree, ...)

Tightly coupled with HW/SW Partitioning

Must schedule system to determine feasibility of each solution being evaluated
Hardware Software Co-Synthesis

Approaches

Optimal
- Try all possible combinations
- Limited use due to Large Design Space
- Examples
  - Exhaustive, Constraint Solving, Integer Linear Programming

Heuristic
- Avoid large Execution times
- Usually give ‘Good’ results
- Iterative or Constructive
Iterative Heuristic Approaches

- Start with an initial solution
- With each iteration of the algorithm improve the solution somewhat
- As the algorithm progresses the solution is refined

Iterative Approach Example

- Presented by Wayne Wolf
  - Start with an Initial Solution
  - Improve the solution at each Iteration
  - As the algorithm progresses the solution is refined
Cosynthesis Environments

**COSYMA**

Ernst et. al. [93]
- Simulated Annealing
- Fixed fine granularity

Henkel and Ernst [97,01]
- Simulated Annealing
- Variable granularity
Cosynthesis Environments

VULCAN

Gupta et. al. [93]

- Iterative Improvement
- Fixed fine granularity
Embedded System Hardware Structure

Various Hardware Options:

- CPU
- Cache
- Memory
- I/O
- ASIC
- FPGA
- Coprocessor
- PEs
System Partitioning

Introduces a design methodology that uses several techniques:

- **Partition system specification into tasks (processes).** The best way to partition a specification depends on the characteristics of the underlying hardware platform. Try different functional partitions.

- **Determine the performance of the function when executed on the hardware platform.** We usually rely on approximating. Exact performance depends on hardware-software details.

- **Allocate processes onto various processing elements e.g. ASIC or FPGA.**
Hardware-Software Partitioning

Hardware/software system design involve:
Modeling, Validation and Implementation

- System Implementation involves:
  Hardware-software partitioning (Cosynthesis)

- Hardware-Software Partitioning
  Identifying parts of the system model best implemented in hardware and software modules

- Such partitions can be decided by the designer (with a successive refinement) or determined by the codesign (CAD) tools.
Hardware-Software Partitioning

For embedded systems, such partitioning represents a physical partition of the system functionality into:

- **Hardware (CPU: 1 or more), ASIC, FPGA, etc.)**
- **Software executing on one or more CPUs**

Various formation of the Partitioning Problem that are based on:

- **Architectural Assumptions**
- **Partitioning Goals**
- **Solution Strategies**

**COWARE**: A design environment for application specific architectures targeting telecom applications.
Partitioning Techniques

Hardware-Software Homogeneous System Model => task graph

For each node of the task graph, determine the implementation choices (HW or SW)

- Keep the scheduling of nodes at the same time Meeting the real-time constraints
- There is intimate relationship between partitioning and scheduling.
- Wide variation in timing properties of the hardware and software implementation of a task. That effects the overall system latency significantly.
Accelerated (ASIC, FPGA, etc.) System Architecture

- CPU
- Accelerator
- Memory
- I/O

request
result
data
data
Accelerator vs Coprocessor

A co-processor executes instructions
- Instructions are dispatched by the CPU

Accelerator appears as a device on the bus
- The accelerator is controlled by registers

Accelerator implementations
Application-specific integrated circuit
Field-programmable gate array (FPGA)
Standard component

Example: graphics processor.
System Design Tasks

Design a heterogeneous multiprocessor architecture.

- Processing Element (PE)
  CPU, accelerator, etc.

Program the system
Why Accelerators?

Better Cost/Performance

- Custom logic may be able to perform operation faster than a CPU of equivalent cost
- CPU cost is a non-linear function of performance
Better Real-time Performance

- Put time-critical functions on less-loaded processing elements.
- Remember RMS utilization (<100%) --- extra CPU cycles must be reserved to meet deadlines.

G. Khan Hardware Software Codesign
Why Accelerators?

- Good for processing I/O in real-time.
- May consume less energy.
- May be better at streaming data.
- May not be able to do all the work on even the largest single CPU.
Accelerated System Design

First, determine that the system really needs to be accelerated.

- How much faster is the accelerator on the core function?
- How much data transfer overhead?

Design the accelerator itself

Design CPU interface to the accelerator
Performance Analysis

Critical parameter is the speedup: How much faster is the system with the accelerator?

Must take into account:
- Accelerator execution time
- Data transfer time
- Synchronization with the master CPU
Total accelerator execution time:
\[ t_{\text{accel}} = t_{\text{in}} + t_{x} + t_{\text{out}} \]
Data Input/Output Times

Bus transactions include:

- Flushing register/cache values to main memory
- Time required for CPU to set up transaction
- Overhead of data transfers by bus packets, handshaking, etc.
Accelerator Speedup

Assume loop is executed \( n \) times.

Compare accelerated system to non-accelerated system:

\[
\text{Speedup} = n(t_{\text{CPU}} - t_{\text{accel}})
\]

\[
= n[t_{\text{CPU}} - (t_{\text{in}} + t_{x} + t_{\text{out}})]
\]

Execution time on CPU of the accelerated functions
Single- vs. Multi-threaded

One critical factor is the available parallelism:

- **Single-threaded/blocking**: CPU waits for accelerator;
- **Multithreaded/non-blocking**: CPU continues to execute along with accelerator.

For multithread, CPU must have useful work to do.

- But software must also support multi-threading.

**Blocking**: CPU waits for the accelerator call to complete.
Total Execution Time

**Single-threaded:**
Count execution time of all component processes.

**Multi-threaded:**
Find longest path through execution.

- **P1**
- **P2**
- **P3**
- **P4**

Accelerator

- **P2, P3** are independent
- **-- After P1 CPU start P3**
- **-- P2 depends on A1**
Sources of Parallelism

Overlap I/O and the Accelerator Computation.

- Perform operations in batches, read in second batch of data while computing on first batch.

Find other work to do on the CPU.

- May reschedule operations to move work after accelerator initiation.
Accelerated Systems

Several off-the-shelf boards are available for acceleration in PCs:

- FPGA-based core;
- PC bus interface.
Accelerator/CPU Interface

Accelerator registers provide control registers for CPU.

Data registers can be used for small data objects.

Accelerator may include special-purpose read/write logic.
  - Especially valuable for large data transfers
Memory Caching Problems

Main memory provides the primary data transfer mechanism to the accelerator.

Programs must ensure that caching does not invalidate main memory data.

- CPU reads location S
- Accelerator writes location S
- CPU writes location S

PROBLEM
Synchronization

As with cache, main memory writes to shared memory may cause invalidation:

- CPU reads S
- Accelerator writes S
- CPU reads S

Solve the Memory and Cache Inconsistency Problem
Partitioning

Divide functional specification into units.
- Map units onto PEs
- Units may become processes

Determine proper level of parallelism:

\[ f_3(f_1(), f_2()) \quad \text{vs.} \quad f_3() \]
Scheduling and Allocation

Must:
- Schedule operations in time
- Allocate computations to processing elements

Scheduling and allocation interact, but separating them will help.
- Alternatively allocate, then schedule.
Scheduling and Allocation

Example:

Task graph

Hardware Modules

Hardware platform
## Process Execution Times

### Example

Time to perform P1, P2, P3 on M1 and M2 hardware module (Library)

<table>
<thead>
<tr>
<th></th>
<th>M1</th>
<th>M2</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>P2</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>P3</td>
<td>-</td>
<td>5</td>
</tr>
</tbody>
</table>
Communication Model

Example

Assume communication within PE is free

Cost of communication from P1 to P3 is $d_1 = 2$

Cost of P2->P3 communication is $d_2 = 4$
First Design Option

Allocate P1, P2 -> M1; P3 -> M2.

Time = 19
Second Design Option

Allocate P1 -> M1; P2, P3 -> M2:

Time = 18

M1

P1

M2

P2

P3

d1

network
System Integration and Debugging

Try to debug the CPU/accelerator interface separately from the accelerator core.

Build scaffolding to test the accelerator.

Hardware/software co-simulation can be useful.

- Seamless: Mentor Graphics
- Coware
Accelerator Case Study

Example:

Video accelerator
Concept

Build accelerator for block motion estimation, a step in video compression.

Perform two-dimensional correlation:
Block Motion Estimation

MPEG divides frame into 16 x 16 macroblocks for motion estimation.

Search for best match within a search range.

Measure similarity with sum-of-absolute-differences (SAD):

$$\sum | M(i,j) - S(i-o_x, j-o_y) |$$
Best Match

Best match produces motion vector for motion block:
Full Search Algorithm

bestx = 0; besty = 0;
bestsad = MAXSAD;
for (ox = - SEARCHSIZE; ox < SEARCHSIZE; ox++) {
    for (oy = -SEARCHSIZE; oy < SEARCHSIZE; oy++) {
        int result = 0;
        for (i=0; i<MBSIZE; i++) {
            for (j=0; j<MBSIZE; j++) {
                result += iabs(mb[i][j] - search[i-ox+XCENTER][j-oy-YCENTER]);
            }
        }
        if (result <= bestsad) {
            bestsad = result;
            bestx = ox; besty = oy;
        }
    }
}

Computational Requirements

Let $MBSIZE = 16$, $SEARCHSIZE = 8$.

Search area is $8 + 8 + 1$ in each dimension

Must perform:

$$n_{ops} = (16 \times 16) \times (17 \times 17) = 73984 \text{ ops}$$

CIF format has $352 \times 288$ pixels $=>$

$22 \times 18$ macroblocks
# Accelerator Requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>block motion estimator</td>
</tr>
<tr>
<td>Purpose</td>
<td>block motion est. in PC</td>
</tr>
<tr>
<td>Inputs</td>
<td>macroblocks, search areas</td>
</tr>
<tr>
<td>Outputs</td>
<td>motion vectors</td>
</tr>
<tr>
<td>Functions</td>
<td>compute motion vectors with full search</td>
</tr>
<tr>
<td>Performance</td>
<td>as fast as possible</td>
</tr>
<tr>
<td>Manufacturing Cost</td>
<td>hundreds of dollars</td>
</tr>
<tr>
<td>Power</td>
<td>from PC power supply</td>
</tr>
<tr>
<td>Physical Size/Weight</td>
<td>PCI card</td>
</tr>
</tbody>
</table>

G. Khan

Hardware Software Codesign

Page: 61
Sequence Diagram

:PC

Search area

:Motion-estimator

compute-mv()

memory[]

memory[]

memory[]
Architectural Considerations

Requires large amount of memory:
- Macroblock has 256 pixels;
- Search area has 1,089 pixels.

May need external memory (especially if buffering multiple macroblocks/search areas).