HW-SW Interfaces
Abstraction and Design
for Multi-Processor SoC

Dr. Ahmed Amine JERRAYA
TIMA Laboratory
46 Avenue Felix Viallet
38031 Grenoble Cedex France
Tel: +33 476 57 47 59
Fax: +33 476 47 38 14
Email: Ahmed.Jerraya@imag.fr
This is Team Work

- **Staff members:** P. Amblard, W. Cesário, X. Chen, F. Rousseau, S. Yoo (left April ’04), N. Zergainoh

- **Ph.D. students:** Y. Atat, I. Bacivarov, M. Bonaciu, A. Bouchhima, A. Grasset, L. Kriaa, Y. Paviot, A. Sarmento, A. Sasongko, W. Youssef

- **Industrial Ph.D.:** A. Blampey, M. Fiandino, F. Hunsinger, L. Perialisi
  
  (STMicroelectronics)

- **Collaborative Ph.D.:** Y. Cho, S. Han (Korea) G. Majauskas (Lithuania) I. Petkov (Bulgaria)

- **Master students and Undergraduates:** F. Dumitrascu, S. Hadhri, R. Khrouf, K. Popovici, M. Yahia,
The SoC Era Challenges

- **SoC**: put on a chip what we used to put on one or several boards (ASIC, CPU, Memories, Analog/RF, MEMS, …)

- **Facts:**
  - 90% of new ASICs already include a CPU in 130nm.
  - Multimedia, network processors, mobile terminals and game applications are already multiprocessors.

- **Fundamental changes:**
  - MPSoC is different from ASIC
  - MPSoC is different from SW
  - MPSoC requires abstract HW-SW interfaces to allow fast integrations.

- **Challenges**
  - Generic MPSoC platform (programmable, reconfigurable, …)
  - Specific MPSoCs using standard IP with specific interconnect.
Generic SoC Platform vs. Application-Specific MPSoC
(MPSoC’03 after dinner discussion)

Example: The GSM History/Roadmap

- 1986  Rack in a van
- 1990  PCB
- 1995  Chip set in a hand-set
- 2002  SoC
- 2006  SW component on a generic platform, e.g. Nomadic (ST)

Same roadmap for game computers, MP3, STB, NP, DVD
Outline

1. HW-SW Interfaces: From Wires to Abstract Interconnect

2. Abstracting HW-SW Interfaces

3. HW-SW Interfaces Design & Debug: The ROSES Environment

4. MPSoC Design
   4.1. MPEG4 Design Example
   4.2. Results Analysis
SoC Platform vs. Embedded Software

- Application SW design:
  - Real time SW design
  - Platform model, e.g., Sony PlayStation, Nomadic
  - Key issue: Complexity (GB, ms)

- Platform_API:
  - Programming model to build software
  - Specific to application
  - Hides HW details

- Hardware dependent SW (HdS)
  - Provided by SoC designer in case of specific HW
  - Lower SW layers to access HW
  - Specific SoC function (e.g., DSP SW code)
  - Key issue: Performances (K&MB, ns)

- Hardware sub-system
  - CPU sub-systems
  - Specific hardware, Analog, memories, ...

- Network-on-Chip

- HW interfaces: required for application specific HW/SW interfaces
Hardware dependent SW Design & Debug is The Bottleneck
Example: SW Debug of an MPEG4 CoDec

- Data dependent computation
- C library bug
- Booting is not synchronized among processors.
- Lost some interrupts
- Wrong interrupt priority levels
- Context switch does not work correctly.
- Incorrect FIFO counter value causes deadlock.
- Result of compressed video is not correct.
- Abnormal execution of a portion of C code

Ahmed A. Jerraya
Outline

1. HW-SW Interfaces: From Wires to Abstract Interconnect

2. Abstracting HW-SW Interfaces

3. HW-SW Interfaces Design & Debug: The ROSES Environment

4. MPSoC Design
   4.1. MPEG4 Design Example
   4.2. Results Analysis
Heterogeneous MPSoC Design Space

- **Software Programming model on an existing platform**
  - Concurrency
  - Decomposition
  - Mapping
  - Communication
  - Synchronisation
  - Interconnect

- **NoC Programming Model**
  - HW Adaptation for application specific communication

- **Computation subsystem model**
  - CPU sub-system for application specific computation
  - SW Adaptation
Which Parallel Programming Model to Use?

- Explicit concurrency, decomposition, mapping; Implicit communication, synchronization, Interconnection and Interface
  - SDL, compositional C++
- Explicit concurrency, decomposition, mapping, communication, synchronization; Implicit Interconnection and Interface
  - MPI, TLM Message, thread package, concurrent C
- Explicit concurrency, decomposition, mapping, communication, synchronization, Interconnection; Implicit Interface
  - TLM Transaction
- All explicit
  - ISA SW + RTL HW
Abstracting HW-SW Interfaces for A Software Sub-system

- API-SW = SW programming model
- API-HW = NoC programming model
- Abstract CPU sub-system
  - HAL = HW abstraction layer
  - HW services: local architecture (e.g. bus)
- SW adaptation: implement programming model on CPU sub-system
- HW adaptation: adapt CPU sub-system to NoC
The Virtual Component Model

- Virtual component
  - Component
    - Hardware IP
    - Software IP
    - Functional IP
  - Abstract Interfaces
    - Required Services
    - Provided Services
    - Control Services
    - Synchronization
    - Parameters, ….

- Execution Environment
  - Abstract Platform (e.g. NoC, Cosimulation backplane, …)
- Heterogeneous components thanks to adaptation between different programming models.
The Virtual Component Model for MPSoC

- Basic model: a set of hierarchically interconnected virtual modules and an execution environment
- Virtual Module:
  - Content: Tasks/Instances + communication channels)
  - Abstract interface: set of virtual ports
    - Internal/external ports
    - Structure and services

- Colif: An XML object-oriented database for virtual architectures
  - Components programming models
  - NoC programming models
- MPSoC programming model is the composition of NoC and components programming models.
Outline

1. HW-SW Interfaces: From Wires to Abstract Interconnect

2. Abstracting HW-SW Interfaces

3. HW-SW Interfaces Design & Debug: The ROSES Environment

4. MPSoC Design
   4.1. MPEG4 Design Example
   4.2. Results Analysis
System-level SoC Design Flow

- System specification is a virtual architecture: virtual modules using specific programming models connected through an execution environment.

- Architecture implementation: heterogeneous components and sophisticated communication interconnect to adapt different programming models.

- Automatic generation of application-specific HW/SW interface sub-systems from basic interface components and CPU sub-system models.
Key Technology: Composing Interfaces

- Component interface
  - Required/Provided services
  - Control and Synchronization services
  - Parameters
- Interface sub-system composition
  - Services matching
  - User-extensible library
  - Code specialization

Interface sub-system composition

Works for HW, SW, and Functional interface sub-systems
Outline

1. HW-SW Interfaces: From Wires to Abstract Interconnect
2. Abstracting HW-SW Interfaces
3. HW-SW Interfaces Design & Debug: The ROSES Environment

4. MPSoC Design
   4.1. MPEG4 Design Example
   4.2. Results Analysis
MPSoC Design of a DivX Encoder

- OpenDivX
  - Open source Mpeg4 encoder/decoder
  - Modified to work concurrently on 1/4th of each frame

- Goals
  - Refinement of HW/SW interfaces
  - Multi-level simulation and early validation
  - SW debug before HW platform is ready.
DivX Encoder: Overview

- **INPUT**: Split coming frame in 4 parts and send it to CPUs
- **CPU_#**: Treat coming data and prepare it for compression
- **VLC**: Finalize compression and prepare the whole image
- **COMBINER**: prepare for output and adjust compression parameters
- **DMA**: Direct access to local memories of processors.
DivX Encoder: Overview

- **Major architecture specificities**
  - Specific Memory Controller: Switch bank service
  - Specific Interface: Core IT + 2 Synchronization Signals
  - Point to Point communication scheme
CPU Sub-system Architecture With An ARM9 Core

- MemCtrl
- SRAM0
- SRAM1
- Bus Matrix
- Memory Controller
- SRAM
- ROM
- Link to DMA

AHB AMBA
Programming Model for DivX (DMA)

Shared-memory Programming Model

Shared memory:  p1.switch_banks()

DMA control:  
- p2.WaitEvent()
- p3.SendEvent()

RT-level channels

SystemC transaction level channels

memory_bank_struct *memory_io;
// initialize encoder library
initialize(5, true, 0, 900);
// loop forever
while(1) {
    // waits for data
    p1.WaitEvent();
    // gets the data address
    memory_io = (memory_bank_struct*)
                p2.switch_banks();
    // signals computation starting
    p3.SendEvent();
    // calls encoding function
    divx_compress(&memory_io->ins,
                &memory_io->outs, 1);
    // signals computation ended
    p3.SendEvent();
    wait();
}
Programming Model for DivX (DMS)

RT-level channels

Message Passing Programming Model

Message passing:
- `p1.sram_init(base_address)`
- `p2.Conn_Setup(rmt_id,lch,rch)`
- `p2.Send(lch,laddress,size)`
- `p2.Recv(lch,laddress,size)`
- `p2.RWrite(lch,laddr,raddr,size)`
- `p2.RRead(lch,laddr,raddr,size)`

DMS control:
- `p2.IWait(lch)`
- `p2.PWait(lch)`

```
memory_bank_struct *memory_io;
// initialize message structure
p1.sram_init(&mes);
// loop forever
while(1) {
  // input data
  p2.Recv(...); p2.PWait(...);
  // gets the data
  while (mes != end_data) {
    memory_io[mes.addr] = mes.data;
  }
  // calls encoding function
  divx_compress(&memory_io->ins), &memory_io->outs, 1);
  // sends output data
  for (...) {
    p2.Send(...); p2.PWait(...); ...
  }
  wait();
}
The ROSES Environment

This group investigates Application-Specific Multi-processor System-on-Chip (ASoC). Research topics include:

- Multi-level and multi-domain specification of heterogeneous systems
- HOTSM on-chip communication design for application-specific multi-processor System-on-Chip
- Memory design for multi-processor System-on-Chip
- Software code for Hardware debug

This group has a long standing experience in the area of system-level synthesis and hardware/software co-design. Previous research has generated more than 150 international publications.
Outline

1. HW-SW Interfaces: From Wires to Abstract Interconnect

2. Abstracting HW-SW Interfaces

3. HW-SW Interfaces Design & Debug: The ROSES Environment

4. MPSoC Design
   4.1. MPEG4 Design Example
   4.2. Results Analysis
Key Results

- Early and multi-level simulation allows for:
  - Architecture exploration
  - Debug cost reduction
    - Debug software before hardware is ready
    - Mitigate hardware prototyping step
- Automatic generation of HW and SW adaptation layers: a drastic improvement of design productivity.
Multi-level Simulation Speed-up and Accuracy

**SW simulation at programming model level**
- Application SW
- Abstract SW Interface Model
- RTL (or TLM)
- HW

**Native SW simulation with abstract CPU sub-system model (HAL)**
- Application SW
- OS
- HAL Model
- RTL
- HW

**HW/SW co-simulation with ISS**
- ISS
- RTL
- HW

<table>
<thead>
<tr>
<th>Speed-up</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>~500 (&gt;&gt;)</td>
<td>75% (&lt;&lt;)</td>
</tr>
<tr>
<td>~100</td>
<td>85%</td>
</tr>
<tr>
<td>1</td>
<td>100%</td>
</tr>
</tbody>
</table>

Ahmed A. Jerraya
Architecture Exploration for QCIF Resolution, 25 frames/s

QCIF RESOLUTION, 25 frames/s
Solution 1: QCIF using ARM7 (60MHz) Processors
Solution 2: QCIF Using ARM9SE46-4kI$,4kD$ (60MHz) Processors

Frame
Architecture Exploration for CIF Resolution, 25 frames/s
Performance Results: CIF Using ARM7 (60MHz) Processors (+3 for VLCs)
Performance Results: CIF Using ARM9SE46- 4kI$,4kD$ (60MHz) Processors (+2 for VLC)
Early Simulation to Reduce HW/SW Interface Debug Cycle

- Validate HdS at several levels of abstraction:

<table>
<thead>
<tr>
<th>Applied to case study</th>
<th>% HdS bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>13%</td>
</tr>
<tr>
<td></td>
<td>83%</td>
</tr>
<tr>
<td></td>
<td>5+5+30 = 40%</td>
</tr>
<tr>
<td></td>
<td>12+13+5 = 30%</td>
</tr>
<tr>
<td></td>
<td>17% 100%</td>
</tr>
</tbody>
</table>

- Simulation Model: 
  - μ-Kernel + HAL + CPU
  - Simulation Model (HAL + CPU) Instruction Set Simulator (ISS)
  - Simulation Model CPU ISS
  - On HW prototype

Ahmed A. Jerraya
MPSoC Design Issues

- Generic MPSoC platform vs. Application specific MPSoC
  - HdS vs. Application specific HW-SW interfaces
  - SW programming model vs. a composition of heterogeneous programming models
- Application specific HW-SW interfaces
  - Computation specific CPU sub-system
  - Interconnect
  - SW adaptation: HdS
  - HW adaptation
- Early validation to reduce design and debug cost.