Hardware dependant Software design

Ahmed A. Jerraya
CEA-LETI
Ahmed.jerraya@cea.fr
Outline

- Multiprocessor System on Chip: HW-SW Architectures
- HW/SW interfaces abstraction: Programming models
- Hardware dependent software design
I. Multiprocessor System on Chip: HW-SW Architectures

SoC Design Cost

Traditional Model

Cost X5 65 to 32nm [Henoff:HDTV ST]

Trend

Developpement Product 3/4

Developpement Plateform 1/4

Développment Product 2/3

Developpment Plateform Techno. 1/3

[MEDEA+Forum]
I. Multiprocessor System on Chip: HW-SW Architectures

Global optimization required

Focus of this talk: HDS = Hardware Dependant Software
I. Multiprocessor System on Chip: HW-SW Architectures

Key Technology: HW/SW Interfaces

- Memory access
- Processor architecture
- OS architecture
- Application
- protocols
- DMA, HW I/O

HW-SW Comm. Design

SW Code / OS
- micro-processor

[O’Nils2001]
HW/SW interfaces design Challenges

- HW/SW interfaces design requires pluridisciplinary knowledge
- HW/SW interfaces are crucial for both cost and performances
  - Cost issues:
    - Hardware dependent Software is tedious to develop, difficult to debug and validate
    - Non optimized software leads to larger footprint and consequently to larger embedded memory
  - Performances:
    - Non optimized HW/SW interface introduces non acceptable global latencies
    - Processor + cache + embedded memory are responsible of the major power consumption part in a SoC

A major challenge in MPSoC design is to efficiently design the HW/SW interfaces
II. HW/SW interfaces abstraction: Programming models

Outline

- Multiprocessor System on Chip: HW-SW Architectures
- HW/SW interfaces abstraction: Programming models
- Hardware dependent software design
II. HW/SW interfaces abstraction: Programming models

Defining HW-SW Interfaces

- Application SW Designer: A set of system calls used to hide the underlying execution platform. Also Called Programming Model
- HW designer: A set of registers, control signals and more sophisticated adaptors to link CPU to HW subsystems.
- System SW designer: Low level SW implementation of the programming Model for a given HW architecture.
- CPU is the ultimate HW-SW Interface
- Sequential scheme assuming HW is ready to start low level SW design
- SoC Designer
  - Abstracts both HW and SW in addition to CPU
  - HW-SW interfaces tradeoff

Sequential SW program
...Call HW (x, y, z)

API

HW-SW Interfaces

HW function wait start
...
II. HW/SW interfaces abstraction: Programming models

Programming Model

- The Classical Solution to Abstract HW-SW Interfaces
  - Programming language with implicit primitives (e.g. module hierarchy & threads in SystemC)
  - API: Application Programming Interface (MPI, Posix Threads)
  - Simulation model (MPICH, Linux)
- Used by SW community to free the SW designer from knowing HW details.
- Facilitate porting of application SW over different architectures that support the same API.
II. HW/SW interfaces abstraction: Programming models

Programming models, the GAP

SoC Design

Distributed SW Design

Programming Model

RTL
(Verilog, BinSW)

Virtual Prototype
(RTL HW, BinSW, e.g. SystemC)

MPI
RT-CORBA
CORBA
SDL

Mixed
Abstract
HW-SW
Interfaces models

Explicit concepts

ALL

ALL

CPU implementation
RTL HW
CPU organization

High Level
Synchronization
NONE

Communication
NONE

- Concurrency
- Threading
NONE
II. HW/SW interfaces abstraction: Programming models

Programming Models for MPSoC

SoC Design ➞ Programming Model ➞ Distributed SW Design

Explicit concepts

- ALL CPU implementation
- ALL RTL HW CPU organization
- Synchro-isation
- Communication
- Concurrency

- ALL - OS
  - Specific I/O
  - Explicit HW modules
- Communication/Computation Modules
- Abstract Interconnect

- NONE

New HW-SW Abstraction levels

- Hiding CPU in addition to HW & SW
II. HW/SW interfaces abstraction: Programming models

Virtual Prototype Model

- Explicit SW (Binary)
- Explicit HW
- Implicit CPU Implementation (ISS)
- Cycle accurate Model
- Typical Model: Cosimulation, SystemC

Binary SW execution on CPU
II. HW/SW interfaces abstraction: Programming models

Transaction Accurate model

- Explicit OS
- Explicit CPU Subsystem Architecture
- Implicit CPU/HAL (TA_BFM)
- Transaction cycle accurate HW
- Typical Model : Native SW Execution [ST/TIMA]
II. HW/SW interfaces abstraction: Programming models

**SW code at Transaction Accurate level**

- **HdS responsible for task and HW resources management**
- **Task code integration with OS and HdS API implementation**
- **Communication (Explicit communication protocol)**

**Example of TA SW code**

**main.c**

```c
extern void task_t2 (void);
void __start (void){
...
create_task(task_t2);
...}
```

**Communication SW**

```c
void recv_data (ch,dst,size) {
switch (ch.protocol){
case FIFO:
    if (ch.state==EMPTY) __schedule();
    ...
case REG:
    dst = _io_read_reg (ch.addr,size);
}
```

**task_T2.c**

```c
channel ch_fifo,ch_reg;
void task_t2(void ) {...
recv_data (ch_fifo,B, 10);
recv_data (ch_reg,C,20);
...}
```

**OS**

```c
void __schedule(void) {
int old_tid=cur_tid;
cur_tid=new_tid;
__cxt_switch(old_tid,cxt,cur_tid,cxt);
...}
```
II. HW/SW interfaces abstraction: Programming models

Virtual Architecture model

- Explicit SW communication API
- Explicit System Interconnect
- Implicit HDS (OS, Communication Implementation)
- Typical Models: TTL, DSoC, Pthreads

HW Node

SW Node

SW Node

Explicit Interconnect

VA_BFM to Abstract HDS & CPU Sub-System Model

Software Thread 1

Software Thread 2

HDS API

Native SW execution on Host
II. HW/SW interfaces abstraction: Programming models

SW code at Virtual Architecture level

- Memory declaration
- Computation code
- Communication code using HdS API
  - Specific mapping to platform channels

**Task code of T1**

```c
static int B[10], C[20], D[10];
void task_T1()
{
    while(1)
    {
        recv_data (reg_ch, B, 10);
        F1(B,C);
        F2(C,D);
        send_data (gmem_ch, D, 10);
    }
}
```

**Communication primitive**

```c
void recv_data (REG_Ch* ch, void* dst, int size) {
    dst = ch->read (ch->address, size);
}
```

```c
class REG_Ch : public sc_prim_channel {
    word *buffer;
    public: word * read (unsigned int addr,int size) {
        for (i=0; i<size; i++)
            *(ret+i)=*(buffer + addr +i);
        return ret;
    }
    ...
};
```
II. HW/SW interfaces abstraction: Programming models

System Architecture model

- Explicit thread/subsystems & HW-SW partition
- Implicit communication
- Typical Models: MPI, Simulink, KPN

Implicit SW execution as a simulation module
II. HW/SW interfaces abstraction: Programming models

System Architecture Modeling in Simulink

- Capture application and mapping to architecture
- Task (set of functions: Simulink blocks, S-functions)
- Subsystem (set of tasks)
- Abstract communication types (Intra-SS, Inter-SS)
- Communication protocol (Generic Simulink I/Os, Explicit annotation for implementation)
III. Hardware dependent software design

Outline

- Multiprocessor System on Chip: HW-SW Architectures
- HW/SW interfaces abstraction: Programming models
- Hardware dependent software design
III. Hardware dependent software design

Heterogeneous MPSoC

- Diopsis Tile [ESWEEK06]
  - ARM9 SS
  - DSP SS
  - MEM SS
  - POT SS (Periph. On Tile):
    - I/O peripherals
    - System peripherals
  - Interconnect: AMBA bus

- Local & global memories accessible by both processing units

- Different communication schemes between CPUs
III. Hardware dependent software design

Multiple SW Stacks

- Reduce the design cost
  - Small memory footprint
- Increase performances
  - Specific platform devices
  - Efficient communication schemes

- ARM specific OS and COM
  - DXM & DSP REG access
- DSP specific OS and COM
  - PIC, Mailbox, DMA
  - REG & DXM access
III. Hardware dependent software design

SW Stack organization

• Layered SW Stack
  – Application code
    • SW code of tasks mapped on the CPU
    • INIT
  – HdS (Hardware dependent SW)
    • OS + Communication + HAL (Hardware Abstraction Layer)

• Different SW components need to be validated incrementally
  – Layers corresponds to Different SW abstraction levels
  – SW development platforms at each abstraction level
III. Hardware dependent software design

Classical SW design flow to interface HW

- Programming Model: Abstract HW at Different level

- Discontinuities:
  - Compilation: Generally ignore the CPU environment (Interrupts, Complex I/O)
  - Sys.lib: adapt for different HW
  - MMAP: Adapt to different CPU-memory architecture
  - User.lib: to make the flow efficient for the application
III. Hardware dependent software design

Code generation from a high level Model d

- **High level description**
  - Contains the application behavior, partitioning and resources mapping
  - Difficult to capture architecture specificities

- **SW architecture exploration**

- **Application SW generation**
  - Multitask and multiprocessor
  - Various operating systems and specific communications support
  - Speed and/or size code optimizations

- **Software stack composition**
  - Using existing or generated tasks
  - Various operating systems and specific communications support

Diagram:
- High level description
  - Application SW generation
    - Task codes
      - SW stack composition
        - HDS
        - Final binary production
          - Final binary
III. Hardware dependent software design

Software generation environment

High level application model (function + mapping)

Software architecture parameters

Application software generation

Tasks code

Tasks init.

Software stack composition

Simulation, performances analysis

Software Stack

I  T1  T4

HDS API

COM

OS

HAL API

HAL

OS library

COM library

HAL library
IV. Long term trends: HW-SW codesign

Outline

- Multiprocessor System on Chip: HW-SW Architectures
- HW/SW interfaces abstraction: Programming models
- Hardware dependent software design
IV. Long term trends: HW-SW codesign

Challenges

- Affordable programming model for MPSoC end users
  - Easy to port new application by end user
  - Based on high level programming model

- Seamless HW-SW Integration
  - Architecture exploration and Application SW mapping

- Quality Proven SW Modules
  - Ensure better design success using proven IP
IV. Long term trends: HW-SW codesign

Ideal Design Flow

- **Simulink Model**
- **Global Interconnect**
- **Hardware Architecture Template**
- **Virtual Prototype**
- **HW Lib**
- **SW Lib**
- **HW Gen**
- **SW Gen**
- **CPU SS1**
  - ISS
  - PIC
  - Local bus
  - Mail box
  - NI
  - Timer
  - Interconnect
- **CPU SS2**
- **Software Main**
- **HDS API**
- **Comm. Library**
- **OS**
- **HAL API**
- **HAL**

Diagram showing the integration of hardware and software design flows, highlighting the use of Simulink Model, HW Lib, SW Lib, HW Gen, SW Gen, and various components like CPU SS1, CPU SS2, ISS, PIC, Local bus, Mail box, NI, Timer, and software stacks.
IV. Long term trends: HW-SW codesign

Classical HW/SW Design: The GAPS

[Gerin ASPDAC 07]
IV. Long term trends: HW-SW codesign

**HW and SW Mixed Models**

- Seamless refinement at four abstraction levels:
  - Simulink Combined Algorithm and Architecture Model (Simulink CAAM)
  - HW/SW Interfaces refinements

---

**Software Design**

- Automatic Code Generation

**Hardware Design**

- Platform based Generators

---

**System level model (Simulink CAAM)**

- Virtual architecture Model (SystemC)
- Transaction accurate (SystemC)
- Virtual prototype (SystemC)
IV. Long term trends: HW-SW codesign

Simulink CAAM of M-JPEG Decoder

- Three CPUs (ARM, Xtensa)
- Communication Units (GFIFO, HWFIFO, SWFIFO)

- 7 S-Functions
- 7 Delays
- 26 Links
- 4 IASs

© CEA 2007. Tous droits réservés. Any reproduction in whole or in part on any medium is prohibited without the prior written consent of CEA.
IV. Long term trends: HW-SW codesign

Experiment Results of MJPEG

10-frame QVGA (320x240) JPEG stream

**RTW:** sequential program on the host machine

**Simulink:** Simulation in Simulink GUI

**VA:** System C model without HW information

**TA:** Abstract CPU + Other devices (SC TLM)

**VP:** Cycle Accurate ISS + Other devices (SC TLM)

<table>
<thead>
<tr>
<th>System</th>
<th>CPU1</th>
<th>CPU2</th>
<th>CPU3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3ARM</td>
<td>ARM7</td>
<td>ARM7</td>
<td>ARM7</td>
</tr>
<tr>
<td>2XT1ARM</td>
<td>Xtensa</td>
<td>Xtensa</td>
<td>ARM</td>
</tr>
<tr>
<td>1ARM2XT</td>
<td>ARM</td>
<td>Xtensa</td>
<td>Xtensa</td>
</tr>
<tr>
<td>3XT</td>
<td>Xtensa</td>
<td>Xtensa</td>
<td>Xtensa</td>
</tr>
</tbody>
</table>

The architecture models at different abstraction levels provide trade-off alternatives between simulation time and architecture detail.

Three ARM7 Subsystems

Execution time (MegaCycles)

218 110 85 78 59
IV. Long term trends: HW-SW codesign

Performance Analysis of MJPEG

Using these performance analysis results, designers can optimize *task allocation* to the processors so that each processor has *enough margins for future additional functions*.
Conclusions

- **HW-SW interfaces**
  - Heterogeneous MPSoC require multiple SW stacks
  - Application-specific hardware & software

- **Programming models for SoC**
  - Abstract specific architectures for a better match between HW & SW
  - Different abstraction levels for early application SW & HDS validation

- **HDS design: Platform based SW development at different abstraction levels**

- **Future trends: Architecture Exploration**
  - HDS-CPU codesign
  - HW-SW interfaces architecture exploration
IV. Long term trends: HW-SW codesign

Acknowledgement

- Prof F. Pétrot, Prof. F. Rousseau from TIMA
- PhD Students K. Popovici, P. Gerin from TIMA
- JR Lequepeys, CEA LETI
Innovation for industry
You will be welcome to the
10th Leti Annual Review
24 - 26 June 2008 at Minatec

For more information:
http://www.leti.fr