Native Power Estimation for Embedded System Design-Space Exploration

Eugenio Villar & Héctor Posadas
Microelectronics Engineering Group
University of Cantabria
Context

- HW/SW Embedded Systems Design Flow
  - HW/SW Simulation
  - Performance Analysis
    - avoiding slow design iterations
  - Design Verification
    - At the different abstraction levels

Diagram:

- Requirements
- Functional design
- Executable Specification
- SystemC
- Co-Design
- Embedded SW
- HW Platform
- VHDL/Verilog
- Compilation
- C
- IP Reuse
- Behavioral Synthesis
- RTL Synthesis
- HW/SW Implementation

UML/MARTE

MDA

PATMOS, Faculty of Informatics, UCM, Madrid

26.09.2011
Agenda

- Motivation: Why SW performance analysis
- Technologies: How SW performance analysis
- SCoPE: SW performance analysis for DSE
  - Improvements from Complex
- Conclusions
Motivation

The MPSoC

- Multi-processing platform
  - ASIC
  - FPGA
  - Commercial multi-processing platform

- SW-centric design methodology
  - Most of the functionality implemented as Embedded SW
  - With ‘some’ application-specific HW
Motivation

- Computing needs Time
  - Edward A. Lee

- Computing needs Energy
  - Eugenio Villar
  - Yet to be published
Motivation

- Embedded SW performance analysis
  - SW performance analysis based on SW simulation
    - At any abstraction level
  - As an integral part of the MPSoC simulation
  - Essential for MPSoC verification
    - At any abstraction level
  - Essential for DSE
    - During architectural design
**SW Simulation Technologies**

- **Embedded SW simulation**

  - Functional Simulation
    - Native code simulation
    - Fast Computation & Communication estimations
  - Native co-simulation
    - Accurate Computation & Communication estimations
  - Co-Design
    - Virtual Models
      - ISS Discrete-Time Models
      - HDL Discrete-Event Models
  - HW/SW Implementation
    - VHDL Reuse
      - Behavioral Synthesis
    - Verilog RTL Synthesis
**SW Simulation Technologies**

- HDL simulation

---

**Embedded System Architecture**

**VHDL Verilog**
- Node i
  - CPU1 model
  - Cache models
  - Bus model
  - DMA
  - NoC if.
  - ASHW
  - memory

**Other Nodes**
- NoC model

**Compilation**

**Node i**
- Application Code
  - Task 1
  - ... Task n

**NoC if.**
- Other Nodes

**OS**
- HdS
- OS API
- HdS API

**Cache models**
- ... caches

**CPU1 model**
- CPU1
- memory

**Cache models**
- ... caches

**CPU model**
- CPUp
- memory

**NoC model**
- NoC
SW Simulation Technologies

- ISS simulation

Embedded System Architecture

ISS Model

Other Nodes

Node i

Application Code

Task 1  ...  Task n

OS API  Hds API  OS API  Hds API
OS  Hds  OS  Hds
CPU1  CPUp
caches  ...  caches
Bus
memory  NoC if.  ASHW  DMA

Compilation

NoC model

NoC

PATMOS, Faculty of Informatics, UCM, Madrid

26.09.2011
Virtualization

Embedded System Architecture

Virtual Model

Node i

Application Code

Task 1

... Task n

OS API | HDS API | OS API | HDS API

OS | HDS | OS | HDS

CPU1 | CPUp

caches | caches

Bus

memory | NoC if. | ASHW | DMA

NoC model

NoC

Other Nodes
SW Simulation Technologies

- Virtualization (QEMU)
  - Detailed model
    - High modeling cost
    - Late design steps
  - Faster than ISS

Intel Core i5 (2.40 GHz)

```
# movl_T0_r1
# ebx = env->regs[1]
mov 0x4(%ebp),%ebx
```

```
# addl_T0_im -16 # ebx = ebx - 16
add $0xffffffff0,%ebx # movl_r1_T0

# env->regs[1] = ebx
mov %ebx,0x4(%ebp)
```

PowerPC (200 MHz)

```
# r1 = r1 - 16
addi r1,r1,-16
```

```
# ebx = env->regs[1]
mov 0x4(%ebp),%ebx
```

```
# addl_T0_im -16 # ebx = ebx - 16
add $0xffffffff0,%ebx # movl_r1_T0

# env->regs[1] = ebx
mov %ebx,0x4(%ebp)
```
SW Simulation Technologies

- Virtualization (QEMU)
  - Functional emulation
  - Rough timed simulation
    - i.e. 1 cycle per instruction
  - Large effort needed for more accurate modeling
    - Execution times
    - Power consumption
    - Caches
    - ...
  - Requires a specific Virtual Model for each processor

- Commercial tools
  - OVP, FastModels, Synopsys (CoWare), etc.
- Native simulation
  - Embedded code directly executed by the host
  - Good accuracy by back-annotation
  - Fast execution time
SW Simulation Technologies

- Native simulation based on HAL API
SW Simulation Technologies

- Native simulation based on OS API

Virtual Model

- Node i
  - Application Code
    - Task 1
    - Task n
  - OS API
  - Hds API
  - Abstract model of OS & CPU
  - TLM Bus model
  - DMA
  - NoC if.
  - ASHW
  - memory

Embedded System Architecture

- Node i
  - Application Code
    - Task 1
    - Task n
  - OS API
  - Hds API
  - Hds
  - Abstract model of OS & CPU
  - OS
  - Hds
  - CPU
  - CPUp
  - caches
  - memory
  - NoC if.
  - ASHW
  - DMA

Other Nodes

- NoC model

- NoC
Basic code annotation in native simulation

```c
... Overflow = 0;
    s = 1L;
    for (i = 0; i < L_subfr; i++) {
        Carry = 0;
        s = L_macNs(s, xn[i], y1[i]);
        if (Overflow != 0) {
            break;
        }
        if (Overflow == 0) {
            exp_xy = norm_l(s);
            if (exp_xy<=0)
                xy = round(L_shr (s, -exp_xy));
            else
                xy = round(L_shl (s, exp_xy));
        }
        mq_send(queue1, &xy, p, t);
    }
...```

Global variable

```
int Sim_Time = 0;
Sim_Time += 20;
Sim_Time += 25;
Sim_Time += 15;
Sim_Time += 10;
Sim_Time += 10;
Sim_Time += 10;
wait included
```
SW Simulation Technologies

- Functional simulation based on code
SW Simulation Technologies

- Power estimation based on traces
  - Accurate but slow
SW Simulation Technologies

- Power estimation based on back-annotation

Virtual Model

- Same technique as with execution times

Global variable

```c
int Sim_Energy = 0;
```

- Best ratio accuracy/speed
### SW Simulation Technologies

#### Performance/Error comparison

<table>
<thead>
<tr>
<th>Technology</th>
<th>Time Estimation</th>
<th>Time &amp; Power Estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functional</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td>5,000</td>
<td>N.A.</td>
</tr>
<tr>
<td>Error</td>
<td>N.A.</td>
<td>N.A.</td>
</tr>
<tr>
<td><strong>Native</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td>1,000</td>
<td>500</td>
</tr>
<tr>
<td>Error</td>
<td>1.3</td>
<td>1.4</td>
</tr>
<tr>
<td><strong>Virtualization</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td>200</td>
<td>T.B.M.</td>
</tr>
<tr>
<td>Error</td>
<td>1.5</td>
<td>T.B.M.</td>
</tr>
<tr>
<td><strong>ISS (cycle-accurate)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Error</td>
<td>1.1 (DT)</td>
<td>1.1</td>
</tr>
<tr>
<td><strong>HDL</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>Error</td>
<td>1 (DE)</td>
<td>1</td>
</tr>
</tbody>
</table>

Rough approximate figures

PATMOS, Faculty of Informatics, UCM, Madrid 26.09.2011
**SCoPE: SW Performance Estimation for DSE**

- **Key features**
  - Abstract OS modeling
  - Instruction cache modeling
  - Data cache modeling
  - System power estimation

- **Novel features**
  - Physical memory accesses
  - Separate memory spaces
  - Configurability for Design-Space Exploration
  - Dynamic Voltage-Frequency Scaling
  - Thermal modeling
  - System composition from IP-XACT components
  - Win32 API
**Instruction cache modeling**
- Similar to time modeling

```c
struct icache_line { char num_set; char hit; }
...
static icache_line line_124 = {0};
static icache_line line_125 = {0};
static icache_line line_126 = {0};
...
If (line_124.hit == 0) insert_line(&line_124);
If (line_125.hit == 0) insert_line(&line_125);
If (line_126.hit == 0) insert_line(&line_126);
```
**SCoPE: SW Performance Estimation for DSE**

- **Data cache modeling**
  - Use modified native addresses to get data variable addresses
  - Global array with all memory line status
  - L2 modeling

```c
bool cache[dcache_size/line_size];

Overflow = 0;
s = L_subfr;
for (i = 0; i < L_subfr; i++) {
    Carry = 0;
s = L_macNs(s, xn[i], y1[i]);
    if (Overflow != 0) {
        break;
    }
}
if (Overflow == 0) {
    exp_xy = norm_l(s);
    if (exp_xy<=0)
        xy = round(L Shr (s, -exp_xy));
    else
        xy = round(L Shl (s, exp_xy));
}
```

If (cache[GET_TAG(&Overflow)] == 0)
    insert_line(GET_TAG(& Overflow));
If (cache[GET_TAG(&s)] == 0)
    insert_line(GET_TAG(& s));
If (cache[GET_TAG(&Carry)] == 0)
    insert_line(GET_TAG(& Carry));
If (cache[GET_TAG(&i)] == 0)
    insert_line(GET_TAG(& i));
If (cache[GET_TAG(&xn[i])] == 0)
    insert_line(GET_TAG(& xn[i]));

If (cache[GET_TAG(&exp_xy)] == 0)
    insert_line(GET_TAG(& exp_xy));
```
SCoPE: SW Performance Estimation for DSE

- System power estimation
  - Application code
    - Instruction counting from binary
  - HW-dependent SW
    - Function power estimation
  - Caches
    - Counting memory accesses
    - Cache misses
  - Bus
    - Actual bandwidth
      - Cache misses
    - DMA accesses
    - HW accesses
  - HW & NoC
    - SystemC power models
SCoPE: SW Performance Estimation for DSE

- Design-Space Exploration
  - Configurable model

![Diagram of Design-Space Exploration](image)
SCoPE: SW Performance Estimation for DSE

- Dynamic Voltage-Frequency Scaling

\[ T_0(F_0, V_0) \]
\[ E_0(F_0, V_0) \]

\[ T = T_0 \cdot \frac{F_0}{F} \]
\[ E = E_0 \cdot \frac{V^2}{V_0^2} \]
\[ P = E/T \]
SCoPE+: Improvements from Complex

- Limitations of application code simulation
  - Valid for co-simulation after partitioning

OS+HdS code developed

HW platform decided
SW Simulation Technologies

- Functional, back-annotated simulation based on code
SCoPE+: Improvements from Complex

- Performance estimation before partitioning
Conclusions

- SW simulation and performance analysis
  - Essential Design Technology
  - HW/SW Embedded Systems
  - At different design steps
    - Different modeling and simulation technologies
    - Various performance*accuracy products

- SCoPE
  - SystemC Native Co-Simulation Technology
  - Specially tuned to performance analysis
    - Design-Space Exploration
Thank you for your attention

- Slides available at:
  - www.teisa.unican.es/en/publicaciones

- Open-source SCoPE available at:
  - www.teisa.unican.es/scope