The COMPLEX ESL Framework for Timing and Power Aware Rapid Prototyping of HW/SW Systems

Kim Grüttner (OFFIS)
2 Motivation

- Rising complexity of applications (Shannon) and execution platforms (Moore)
  - but the design gap becomes larger
  - including the uncertainty of platform selection

- Power consumption is the next main limiting physical factor

- Limitation in memory access times requires smarter memory organization

![Graph showing the performance of Shannon's, Moore's, and Eveready's laws over time with notes on complexity, power reduction, and memory access time.](Source: Jan M. Rabaey)
3 Motivation

- No generic framework available for estimating execution artifacts of application on target platform early in the design process:
  - timing/performance,
  - power consumption, and
  - memory/communication bottle-necks

- Available point-tools need to be bundled and properly interfaced in a holistic framework
  - MDA software design tools,
  - behavioral synthesis,
  - optimizing cross-compilers,
  - virtual platform generators,
  - automatic design-space exploration tools
4 Requirements

- Consideration of functional, power and timing behavior on system-level
  - Executable model of entire system
  - Capable to simulate large sequences of domain-specific workloads
  - In an acceptable time

- Executable Specification
  - Parallel specification enabling same description for HW and SW
  - Separation of behavior and communication
  - Communication should abstract from platform communication resources

- Estimation and Representation of Non-Functional Properties
  - Timing and power dissipation needs to be studied under domain-specific workload scenarios
  - Full custom HW, SW, Communication and IP power & timing estimation

- System Simulation Including Non-Functional Properties
  - Fast and scalable simulation due to raising system complexity
5 The proposed framework
General Overview

- Innovative design environment
- Productive design methodology
- Holistic framework
- Iterative design space exploration
- Five main stages:
  - MDA design entry
  - Executable specification
  - Estimation & model generation
  - Simulation
  - Exploration & optimization
6 Hardware/Software Task Mapping and Separation

- Determine individual HW/SW/IP tasks of the system
- Collect information from the SystemC specification
- Collect information from architecture description
- Elaborate the SystemC model
- Read mapping information from IP-XACT
- Create internal design representation
- Extract individual tasks
- Forward individual tasks to appropriate tool
- Testbench/Transactor generation
  - Simulate each task decoupled from the rest of the system
  - Characterisation is done with realistic data
  - Task is used as in the overall system
  - Created model is appropriate for the overall system
7 Analysis and Model Generation

General

- Characterise each task with the appropriate tool
  - HW  $\Rightarrow$ Power aware High Level Synthesis
  - SW  $\Rightarrow$ Power aware target specific cross-compiler
  - IP  $\Rightarrow$ Use information from datasheet (Power State Machines)
- Generate augmented version of the task
- Self-simulating model of:
  - Timing (clock cycles)
  - Dynamic power of functional behaviour
  - Static power of the data path
  - Controller, clock tree, etc.
- Model parameter can be influenced during simulation
  - Leakage (Voltage scaling, power gating, etc.)
  - Timing (frequency scaling, delay by power mode switching, etc.)
8 Analysis and Model Generation

Example: Hardware tasks

```c
SC_THREAD(faculty);
void faculty() {
    while(true){
        i = in.read();
        r = 1;
        while(i>1) {
            r = r * i;
            i--;
        }
        faculty.write(r);
        wait();
    }
}
```

```
SC_THREAD(faculty);
void faculty() {
    LEAKAGE(1,uW); // per process
    CTRL(3,uW); // per process
    CLOCKTREE(2,uW); // per process
    AREA(2,3,uM2); // per process
    while(true){
        i = in.read();
        r = 1;
        POWER(2,uW);
        DELAY(1,nS);
        wait(sc_time(1,SC_NS));
        tmp0 = i>1;    // while(i>0)
        while(tmp0) {  // while(i>0)
            tmp1 = r * i; // r = r * i;
            r    = tmp1;  // r = r * i;
            tmp2 = i - 1; // i--;
            i    = tmp2;  // i--;
            tmp0 = tmp2>0;// while(i>0)
            POWER(6,uW);
            DELAY(2,ns);
            wait(sc_time(2,SC_NS));
        }
        faculty.write(r);
        wait();
    }
}
```
9 Virtual System Generation

- Assemble all characterised tasks
- Add TLM2-based communication
- Synthesise appropriate interfaces
- CPU
  - Computation notes are augmented
  - Explicit communication
  - Fetches are handled by a cache model
- Custom hardware
  - Parallel execution
  - Computation nodes are augmented
- TLM2 Router
  - Contains bus power and timing model
  - Programmers view
  - Bus-cycle approximate
  - Bus-cycle accurate
Monitors allow different types of granularity

- Allows a trade-off between accuracy and simulation speed
- Selectable for each task individually
- Adjustable during simulation time
11 Conclusion

- HW/SW task separation and testbench generation
  - Mapping tasks to hardware resources
  - Extracting individual tasks based on the hardware they are mapped to
  - Tasks are clustered and forwarded to appropriate characterisation tools

- Task characterisation
  - Each task is simulated and estimated, using a specific testbench
  - An augmented version of each task, containing a self-simulating model is created

- Virtual system generation
  - Synthesises interfaces and connects augmented versions of tasks
  - Uses a TLM2-based communication model

- Simulation
  - Shall be enhanced in terms of simulation speed
  - Shall provide different types of granularity
Thank you very much for your attention

COMPLEX project partners:

- ST
- SYNOPSYS
- OFFIS
- THALES
- chipvision
- gmV
- EDALab
- universidad de cantabria
- magilem
- ECSI
- imec

Project Website: http://complex.offis.de