SoCKET Collaborative Project

(SoC toolKit for critical Embedded sysTems)

June 2008 – November 2011

Astrium

Noordwijk,
V.LEFFTZ & J. LACHAIZE
(Astrium)
Agenda

- SoCKET project presentation
- Proposed Hardware/Software Co-Design Flow
- Focus on key technologies + perspectives
  - SystemC/TLM Modelling
  - Assertion Based Verification
  - High Level Synthesis
Agenda

- SoCKET project presentation

- Proposed Hardware/Software Co-Design Flow

- Focus on key technologies + perspectives
  - SystemC/TLM Modelling
  - Assertion Based Verification
  - High Level Synthesis
SoCKET: Objectives

Define a "seamless" development flow, integrating the equipment qualification/certification, from the system level, to the IC and validated SW on these ICs;

Master the SoC solutions for critical embedded systems;

Master the "system dimension" (software + hardware) into the SoCs integration problematics;

Master the complexity, the time cycle reduction, design optimisation of SoC-based systems;

Evaluate the HW simulation models (get from the design flow) usage for the integration and the validation of the critical embedded SWs.
“Seamless” design flow

Formalisms unification
- Remove any semantic holes into HW/SW interfaces

Models transformation operators
- Automation
- Traceability
- Overall coherency insurance

Tools interoperability
- Keystone of 2 previous points
SoCKET: Consortium

- **International groups:**
  Airbus, Astrium, STMicroelectronics, Thalès R&T

- **PMEs:**
  PSI-S, PSI-E, Magillem Design Services

- **Academics and Research Centers:**
  CNES, IRIT, Lab-STICC, TIMA

- **Security Camera Use Case**
- **IP-XACT**
- **SystemC/TLM modelling Heterogeneous Simulation Techniques**
- **ABV ISIS & HORUS**
- **SoC Debug & Trace**
- **SW Properties WCET OTAWA**
- **SW Secure Architecture**
- **HLS Gaut**
- **Secondary Flight Control Computer**
- **Swarm Magnetometer Computer**
- **Image Processing Moving Object Tracking & Compression**
- **Moving Object Tracking & Compression**
- **Grenoble**
- **Paris**
- **Toulouse**
- **Lorient**
- **HLS Gaut**

**Use Case:**
- Secondary Flight Control Computer
- Image Processing Moving Object Tracking & Compression
Agenda

- SoCKET project presentation
- Proposed Hardware/Software Co-Design Flow
- Focus on key technologies + perspectives
  - SystemC/TLM Modelling
  - Assertion Based Verification
  - High Level Synthesis
SoCKET Co-Design flow

1. **System requirements**
   - Global SoC spec.

2. **SoC Architecture**
   - HLS
   - Metrics
   - Platform assembly
   - IP-Xact
   - Requirement traceability

3. **Functional validation**
   - TLMₜ
   - Functional validation

4. **SW Performance validation**
   - TLMₘ
   - SW Performance validation

5. **Co-simulation/Co-emulation**
   - RTL
   - Co-simulation/Co-emulation

6. **Device execution**
   - Silicon
   - Device execution

7. **Software**
   - C/C++/ASM
   - Instruction Set Simulator

---

**HW Properties**
- Functionality
- Timing

**SW Properties**
- Functionality
- Header generation

---

**System Properties**
Content management with IP-Xact

Generation of HDS layers
- Header files
- Structure and Functions
- Parts of drivers

- Generation of TLM model skeleton
- Top netlist for virtual platform
- Hardware platform documentation

Consistency vs. heterogeneity
- Various formalisms (architectures, timing, functionality, power, …)
- Various levels of abstraction and languages
- Implementation in hardware and software
Define the appropriate partitioning between hardware and software
- Conform with system requirements

An architect-driven decision process guided by metrics
- High Level Synthesis
  - Provides IP internal information

IP Traffic Generators
- Assess bandwidth and latency scenarios

Resulting architecture is described in the IP-Xact format
- Refined requirements are associated to the architecture
**Objective**
- Analyze interconnect bandwidth and latency

**Means**
- For each IP, characterize traffic profile
- Assemble a platform with a BCA/RTL interconnect and memory models
- Generate traffic
- Exploit results with analysis tools
Transaction Level models (LT)

- System requirements
- Global SoC spec.
- SoC Architecture
- HLS
- Metrics
- Traffic generators
- System Properties
- HW Properties
- IP-Xact
- Requirement traceability
- SW Properties
- Platform assembly
- Metrics
- Header generation
- HLS
- RTL
- TLM _LT_ 
- Functional validation
- SW Performance validation
- Co-simulation/Co-emulation
- SoC Recorder/Emulator
- C/C++/ASM
- Device execution
- SILICON
- SW Properties
- RTL
- TLM _AT_ 
- Functionality + timing
- HLS

Requirements:
- Functional validation
- SoC Architecture
- HLS
- Traffic generators
- SW Properties
- Platform assembly
- Metrics
- Header generation
- HLS
- RTL
- TLM _LT_ 
- Functional validation
- SW Performance validation
- Co-simulation/Co-emulation
- SoC Recorder/Emulator
- C/C++/ASM
- Device execution
- SILICON
Transaction Level models (LT)

- Model IPs/subsystems at the transaction level
  - Bit true behavior & communication
  - System synchronization points
  - No clock/cycle, but functional timing (e.g. *timer*)
  - Fast to implement and simulate

- TLM LT models often built using
  - C reference model
  - TLM wrapper
    - Model registers
    - serve read/write accesses

- Used for
  - Embedded software development
  - Functional verification activities for RTL IPs
Validation of LT models

- The TLM LT model is the golden reference
- Functional test suite is built using the LT model
- TLM LT testbench is used to validate the RTL IP
- High level of confidence for reusing the model
Transaction Level Models (AT)

- System Properties
- Global SoC spec.
- SoC Architecture
- Functional validation
- SW Performance validation
- System requirements
  - Metrics
  - HLS
  - Metrics
  - Traffic generators
  - SW Properties
  - HLS
  - IP-Xact
  - Header generation
  - C/C++/ASM
  - Requirement traceability
  - Platform assembly
  - TLMAT
  - TLM LT
  - HW Properties
  - Functionality
  - RTL
  - SW Properties
  - SW Performance validation
  - Co-simulation/Co-emulation
  - Device execution
  - Silicon
Transaction Level Models (AT)

- Targeting performance evaluation
  - Hardware architecture
  - Software
  - Captures micro-architecture information/timing
  - Timing accuracy may be trimmed

- Several technical options
  - LT Model refinement -> rewriting
  - LT Model annotation
  - Composition of LT and T models
  - Generation of a CA model from RTL
Validation of AT models

- Functionality of AT models can be assessed with the LT tests
- Building the temporal model is difficult
  - Extract timing information from the RTL
  - Implement the micro-architecture model
  - Statistical approach
- And impacts significantly the simulation speed
- The hard topic is the temporal validation
  - Reuse of Implementation Verification Patterns when available
  - As difficult as the validation of cycle accurate models
Comparing abstraction levels

- TLM LT
  - Same functional behavior

- TLM AT
  - Same functional behavior
  - Same timed behavior

- RTL
  - Same timed behavior
RTL Level

System requirements

Global SoC spec.

SoC Architecture

System Properties

HW Properties

Functionality

TLM_LT

Functional validation

IP-Xact

SoC

TLM_AT

SW Performance validation

C/C++/ASM

Software

Requirement traceability

Metrics

Traffic generators

HW Properties

Functionality

Timing

HLS

RTL

Co-simulation/Co-emulation

Silicon

Device execution

Software

System Properties

Platform assembly

Software

IP-Xact

Header generation

Software

Co-simulation/Co-emulation

Software
RTL level

Entry point for logic synthesis flow

- RTL models might be
  - Implemented manually
  - Generated from higher description (HLS flow)

Co-simulation
- Joint simulation of SystemC/TLM and VHDL/Verilog models

Co-emulation
- Simulation of SystemC with the execution of VHDL/Verilog models mapped on a hardware emulators
Each model is a golden model for its level of abstraction.
Verification at a lower level extends the validation capacities.
Verification flow is connected to requirement management.
Requirement update impacts all relevant models.
Agenda

- SoCKET project presentation
- Proposed Hardware/Software Co-Design Flow
- Focus on key technologies + perspectives
  - SystemC/TLM Modelling (Time modelling)
  - Assertion Based Verification
  - High Level Synthesis
Time is not represented, only functionality is modelled.

Functional synchronization is necessary. It is done at System Synchronization Points (SSP): configuration registers access, interrupts and all state alternating accesses.
The need for time

- Performance measurements
- Design Space Exploration
- Feasibility assessment
  ...how ??
- Precision?
- Modelling granularity?
- Simulation performance?
The obvious solution: mixing time and functionality

- It works !!!

...but...

- Functional modifications cannot be verified without having to verify all timed aspects as well
- Modelling granularity is hard to modify once it has been set
- Modules cannot be easily reused for other platforms
Each model has a functional (PV) part and a timed part (T).
Simulation switches between PV and T phases: when all PV models have reached a SSP.
T models record time relevant data (address, size of transfer…) during PV phases.
During T phases, T models “replay” the last PV phase and simulate time accordingly.

Enriching a functional platform
Functional simulation phase

ISS PVT

PV router

Detailed bus model

Memory PVT

Initiator port

Target port

T= Δt has
### Advantages and limitations

<table>
<thead>
<tr>
<th>PV &amp; T mixed</th>
<th>PV &amp; T separated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modelling is “natural”. Platforms are simple.</td>
<td>Parallel development and debug of reusable PV and T models</td>
</tr>
<tr>
<td>Asynchronous events can be modelled easily</td>
<td>Granularity can be controlled easily (by changing T model)</td>
</tr>
<tr>
<td>Granularity is fixed</td>
<td>Modelling is more abstract. Platforms are complex</td>
</tr>
<tr>
<td>Mixed debugging</td>
<td>Asynchronous events are harder to model</td>
</tr>
<tr>
<td>no control over simulation performance</td>
<td></td>
</tr>
</tbody>
</table>
Agenda

- SoCKET project presentation
- Proposed Hardware/Software Co-Design Flow
- Focus on key technologies + perspectives
  - SystemC/TLM Modelling
  - Assertion Based Verification
  - High Level Synthesis
**What is ABV?**

**Assertion**: statement about the *intended behaviour* or a *requirement* of the design

- Temporal logics: CTL, LTL, ...
- Specification languages: SVA (IEEE Std 1800), PSL (IEEE Std 1850)

**Assertion-Based Verification**: does the design obey these temporal assertions?

- Static analysis (model checking)
- **Dynamic verification** (during simulation)
Assertions at the system level:

Any time a source address is transferred to the DMA, a read access eventually occurs and the right address is used.
ISIS tool from TIMA Lab

PSL assertion

Automatic transformation

SystemC assertion monitor (checker)

Automatic instrumentation

Design under verification
Astrium’s REX

- Good expressivity of PSL
- Moderate time overhead induced by monitoring (5-8%)
- The property checkers will provide a valuable help for non-regression testing
- Careful (natural language) expression of the requirements
  - Specify at the TLM interfaces
  - Disambiguate the properties, in particular the meaning of communication actions, and specify your preferred observation points
Among the future works

- Perform other experiments and refine the definition rules for the expression of properties
- Relation between PSL properties at the transactional level and at the RT level?

Correctness, and mitigation of radiation effects
Agenda

- SoCKET project presentation
- Proposed Hardware/Software Co-Design Flow
- Focus on key technologies + perspectives
  - SystemC/TLM Modelling
  - Assertion Based Verification
  - High Level Synthesis
1st step: Algorithm -> HLS

Validation
- Intermediate results required
- Validity criteria (computation precision)
HLS for architecture exploration

- Metrics (area, performance) - 20% pessimistic - Usable for tradeoffs
- Help for bit accurate arithmetic migration (fixed point (ac_type/sc_type))
- HLS requires to consider any IO architecture bottlenecks
- HLS incremental refinement try/test loop: heuristic approach
- Allows to measure latency introduced by pipelining
- Separation of the processing process and the IO constraints
Tools evaluation report

- Celoxica HandleC
  - 2007: GAIA project – metrics for DSE
  - Obsolete since 2009

- Mentor/Calypto CatapultC
  - 2005 evaluation: great tool, but too expensive
  - 2010—: HLS bluebook (Michael Fingeroff)

- Mathworks HDL Coder
  - 2011: introduction in production process

- Université de Bretagne Sud Lab-STIC CC GAUT
  - SoCKET
High Level Synthesis: GAUT

- **Academic tool**
  - Public Domain (CECILL-B License)
  - Open source and free
- **Dedicated to DSP applications**
  - Data-dominated algorithm

- **Inputs:**
  - Algorithm written in bit-accurate C/C++
    - Bit-accurate integer and fixed-point from Mentor Graphics
  - Synthesis constraints (data average throughput, clock, I/O constraints...)

- **Outputs:**
  - RTL Architecture written in VHDL (IEEE 1076)
  - Simulation model in SystemC
  - Automated Test-bench generation
GAUT REX

- Powerful academic tool for Data Flow Graph
- Good support of Xilinx targets
- Generated component directly pluggable on a bus (not used).
- ATC18RHA targeting on-going study
- Generated HDL efficiency (# gates and speed)
- Hierarchic synthesis support (not tested)
- Further evolution in the frame of project P
  - CDFG support
  - IO communication pattern instantiation
Pipelining needs to be externally handled
  - A valid signal is not propagated
Control of loops unrolling
  - Automatic loop unrolling under constraint No manual override
Output timing constraint propagation
Add traceability between the C code and generated VHDL code
Operators set extension (e.g: fixed-point division)
Smart IO management (optimal data organization, ...)
Would require some additional work for tool industrialization/qualification (documentation, validation of generated HDL, IO interface configuration/control, ...)

GAUT expected enhancement
HLS highlights

- Ease IP maintenance/evolution.
- Requires both hardware competence and software skills.
- It’s quite natural to transform Matlab to C then to RTL.
- Not optimal for data handling (FIFO, Cache, prefetch)
- ! Manager: no speed up development process but the exploration process and avoid some dead-end
ESL techniques very useful:

- For DSE,
- For removing ambiguities at HW/SW interfaces level,
- For improving the fast prototyping phase

Need of

- Building a Space ESL ecosystem
  - Common Model Library and Toolbox
  - Common Model Coding Rules for model exchange
- Can we avoid any dependence with commercial tools
Public website:  http://socket.imag.fr

Public “final” workshop:
- 23 & 24 November 2011 at Toulouse, France
- 1\textsuperscript{st} day: Tutorial on pillar technologies & Results presentation
- 2\textsuperscript{nd} day: Industrial Return of Experience

YOU ARE ALL WELCOMED
Thank you for your attention

Any questions?
Technical pillars (1)

- High level synthesis (Lab_STICC)
  - Under time constraint
  - Under resources constraint

- Heterogeneous simulation techniques (STM)
  - SystemC/TLM
  - LT/AT/CABA abstraction levels

- IPs encapsulation and interoperability (MDS, STM)
  - SPIRIT/IP-XACT (structural information)
    => configuration/documentation/deployment
  - OCP-IP
Validation techniques

- Formal and semi-formal methods (TIMA, IRIT)
  - Formal verification by model-checking
  - Semi-formal verification by automatic generation of monitors

- Mutation analysis techniques (STM)
  - Atomic modification of the design to check if the validation environment detect it
  - Today, at RTL level, SoCKET will try to extend it to system level

- Test cases automatic generation (STM)
  - E language OK for IPs, performance issues for SoCs
  - Evaluation of techniques based on IP-XACT description and C generators