Overview

- Embedded system design process
  - Traditional
  - Codesign
- The COSYMA system
- The POLIS approach
- SpecSyn

Credits: Rolf Ernst, Sushant Jain, Vivek Sinha
Design process

Co-synthesis design flow
Co-design using co-synthesis and design space exploration

- specification parameter change
- high level transformations

estimations

OS, component & communication libraries
intermediate-code generation
code generation
object code

HDL generation
HW/SW partitioning, scheduling
RLS synthesis

system function
cost, performance, ...

hardware designer

customer

software developer

system architect

COSYMA

- COSYMA (COSYnthesis for eMbedded micro Architectures)

Achim Österling, Thomas Benner, Rolf Ernst, Dirk Herrmann, Thomas Scholz, Wei Ye

Technische Universität Braunschweig
Germany
COSYMA - an Overview

- A platform for exploration of the HW/SW co-synthesis techniques
- Covers *almost* entire design flow
- Limited target architecture
- Is used mainly for design-space exploration where it gives fast response times

The COSYMA Design Flow
The COSYMA Design Flow

C Processes
- Compiler
- Simulation and profiling
- Process scheduling
- HW/SW Partitioning
- Constrains and user directives

Constraints and user directives
- Communication Models

C-code generator
& comm. synth.
- HW/SW Target model
- Peripheral modules

HDL-code generator
& comm. synth.
- SW Synthesis
- HL synthesis
- Synthesis directives
- HDL netlist

Run-time analysis
- obj. code
- Target model

COSYMA Architecture

- Standard RISC processor core (a SPARC architecture model with 33 MHz clock and floating point coprocessor with COSYMA)

- A fast RAM for program and data with single clock cycle access time
COSYMA Architecture

- An automatically generated application specific coprocessor

- Peripheral units must be inserted by the designer

- Processor and coprocessor communicate via shared memory in mutual exclusion

The COSYMA Design Flow

- C Processes
- Constraints and user directives
- Communication Models
- Constraints and user directives
- HW/SW Partitioning
- C-code generator & comm. synth.
- HDL-code generator & comm. synth.
- SW Synthesis
- HL synthesis
- Synopsys DC
- VHDL netlist
- Peripheral modules
- HW/SW Target model
- obj. code
- run-time analysis
- Communication Models
- Process scheduling
- Simulation and profiling
System Spec

- The input description may consist of several communicating processes with timing requirements, specified in Cx (extension of C supporting parallel processes and timing constraints).

- Internal data structure: Extended Syntax Graph (ESG)

The COSYMA Design Flow

- C Processes
  - Compiler
  - Simulation and profiling
  - Process scheduling
  - HW/SW Partitioning
  - C-code generation & comm. synthesis
  - SW Synthesis
  - Run-time analysis

- Constraints and user directives
  - Communication Models
  - HW/SW Target model
  - VHDL netlist
  - Peripheral modules

- HDL-code generation & comm. synthesis
  - HL synthesis
  - Synopsys DC
  - obj. code
  - VHDL netlist

- Synthesis directives
  - SW Synthesis
  - SM
  - C-code generation & comm. synth.
Scheduling

- $C^X$ processes are simulated on a RTL model or analyzed symbolically
- Scheduling is done by using Scalable Performance Scheduling (SPS)
  - Resulting a single serialized process
  - Done before partitioning
- Alternative approach - combination of scheduling and partitioning

Scheduling

- Speedup factor - to estimate the acceleration factor of the target architecture compared to the reference processor
- Information can be retrieved in an early design stage (before partitioning)
The COSYMA Design Flow

Partitioning

- Inputs:
  - ESG with profiling information
  - CDR file
  - Synthesis directives
- Basic block level and is automated

Timing constraints!
Partitioning

- Partitioning goals:
  - meet real-time constraints
  - minimize hardware costs
  - minimize the CAD system response time - allow user intervention

- Simulated Annealing is deployed as an optimization algorithm

Partitioning

- Communication is implicit

- Requires communication analysis and communication synthesis

- DOES NOT require explicit description of the communication from the user
The COSYMA Design Flow

The COSYMA Design Flow involves the following steps:

1. C Processes
   - Compiler
   - Simulation and profiling
   - Process scheduling
   - HW/SW Partitioning
   - Constraints and user directives

2. Constraints and user directives lead to:
   - Communication Models

3. Communication Models lead to:
   - Hardware synthesis
   - Software synthesis
   - Simulation and profiling

4. Hardware synthesis includes:
   - SW Synthesis
   - HW/SW Target model
   - Peripheral modules
   - HDL code
   - VHDL netlist

5. Software synthesis includes:
   - HL synthesis
   - Compilation
   - Obj. code
   - Process scheduling

6. Compilation leads to:
   - Compilation directives
   - C-code generator & compiler
   - HDL code generator & compiler

7. Co-simulation

Synthesis

- For high-level synthesis: Braunschweig Synthesis System (BSS) → for fast coprocessor designs

- Netlist by Synopsys Design Compiler

- For software: Standard C compiler

- Co-simulation
COSYMA Conclusions

- Software oriented approach
- Specification can be handled easily (C^X)
- Supports automated partitioning and co-processor synthesis
- Design space exploration is possible during synthesis
- Synthesis is driven by timing and HW constraints
COSYMA Conclusions

- Does not support concurrent modules
- Architecture is limited
- There is no support for formal verification
- Quality depends on partitioning and cost estimation techniques

POLIS

- Luciano Lavagno, Ellen Sentovich, Kei Suzuki, Alberto Sangiovanni-Vincentelli et al.
- UC Berkeley, Cadence, Magnetti-Marelli, PdT
POLIS

- HW/SW codesign framework for reactive (control-dominated) embedded systems
- Includes both synthesis and simulation
- Environment based on a uniform representation for hardware and software - a network of Co-design Finite State Machines (CFSMs)

POLIS

- CFSM has a finite, non-zero, unbounded reaction time
- The model is Globally Asynchronous, Locally Synchronous
- Each element of a network of CFSMs describes a component of the system to be modeled
- Hardware and software have different delay characteristics
System Specification

- The system can be specified in **Esterel to be** directly translated into CFSMs
  - Reactive synchronous language

- System is a set of Concurrent Esterel modules (can be hierarchical)

- Communication via signals and events

CFSM

- Classical FSM has a synchronous communication model
- CFSM has a finite, non-zero, unbounded reaction time

- A CFSM consists of
  - sets of input events
  - sets of output events
  - a transition relation
Each transition emits after an unbounded non-zero time the output event

- A synchronous hardware implementation of CFSM can execute a transition in 1 clock cycle
- A software implementation can require more than 1 clock cycle

Events move between communicating CFSMs in zero time (like in Esterel)
Design Partitioning

- Implementation selection for every CFM
- CFM specification is *a priori* unbiased towards a hardware or software implementation
- No support for automatic partitioning
Synthesis

- Each HW partition is implemented as a fully synchronous circuit
  - CFSM2BLIF (XNF, VHDL, Verilog)
  - Logic synthesis

- Each SW partition is implemented as a standalone C program
  - High-level, processor independent representation
  - Portable C code

Synthesis

- Simple Real-time Operating System to:
  - provide communication between modules (HW-SW and SW-SW)
  - schedule SW CFSMs (Rate-Monotonic and Deadline-Monotonic)
  - generate device drivers for communication
  - generate an event driven layer which implements the CFSM event emission/detection primitives
Interfacing

- Interfaces between different implementation domains (hardware-software) are automatically synthesized.
- Interfaces come in the form of cooperating circuits and software procedures (I/O drivers) embedded in the synthesized implementation.
- Communication can be through I/O ports available on the micro-controller, or general memory mapped I/O.
Formal Verification

- Direct interface with existing FSM based formal verification tools
- **POLIS** includes a translator from the CFSM to the FSM formalism
- A methodology which incorporates a set of abstraction and assumption rules specific to **POLIS** and CFSMs

System Simulation

- System level HW-SW Co-simulation is a way to give designers feedback on their design choices.
  - HW-SW partitioning
  - CPU selection
  - Scheduler selection.
- Fast timed co-simulation due to the software synthesis and performance estimation techniques.
- **PTOLEMY** co-simulation environment
POLIS - Conclusions

- Suitable for small control dominated systems
- FSM based approach
- Basic communication primitives: events
- Flexible design space exploration (HW & SW are treated similarly, can be derived from the same CFSM)
POLIS - Conclusions

- Co-simulation supported by Ptolemy
- Support for formal verification both at specification and implementation levels
- Not suitable for very large computationally dominated systems
- Architecture is limited - single processor surrounded by a combinational HW. No support for shared memory

A Comparison of the Features

<table>
<thead>
<tr>
<th>Features</th>
<th>POLIS</th>
<th>Cosyma</th>
<th>Chinook</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Spec. Language</td>
<td>Esterel</td>
<td>C^</td>
<td>Verilog</td>
</tr>
<tr>
<td>Constraint Spec.</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Abstract Comm.</td>
<td>Event</td>
<td>Send/receive</td>
<td>Signals</td>
</tr>
<tr>
<td>Model of Computation</td>
<td>CFMS</td>
<td>RAM Model</td>
<td>Concurrent modules</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Concurrent modules</td>
<td>Single thread of execution</td>
<td>Concurrent modules</td>
</tr>
<tr>
<td>Partitioning</td>
<td>Manual User defined modules</td>
<td>Automated C Instruction level</td>
<td>Manual module and task level</td>
</tr>
<tr>
<td>Granularity Level</td>
<td>Supported</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Formal Verification</td>
<td>PTOLEMY</td>
<td>CoSim</td>
<td>Pia</td>
</tr>
<tr>
<td>Co-Simulation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Comparison of the Features

<table>
<thead>
<tr>
<th>Features</th>
<th>Polis</th>
<th>Cosyma</th>
<th>Chinook</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW Synthesis</td>
<td>BLIF, VHDL, XNF</td>
<td>BSS Tool</td>
<td>Netlist</td>
</tr>
<tr>
<td>SW Synthesis</td>
<td>S-Graphs; C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>OS Synthesis</td>
<td>Automated</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Process Scheduling</td>
<td>Part of OS</td>
<td>Static</td>
<td>Static</td>
</tr>
<tr>
<td>SW Performance Estimation</td>
<td>S-Graphs and Empirical results</td>
<td>Sparc Simulator</td>
<td></td>
</tr>
<tr>
<td>HW Estimation</td>
<td>Single cycle execution</td>
<td>List scheduling</td>
<td></td>
</tr>
<tr>
<td>HW/SW Communication</td>
<td>I/O Ports</td>
<td>Shared Memory</td>
<td>I/O Ports</td>
</tr>
<tr>
<td>Target Architecture</td>
<td>Processor and CFSMs implemented in HW</td>
<td>Processor, coprocessor and shared memory</td>
<td></td>
</tr>
<tr>
<td>Processor supported</td>
<td>MIPS R3000</td>
<td>Sparc</td>
<td></td>
</tr>
</tbody>
</table>

SpecSyn

- SpecSyn environment for SER paradigm
- UC Irvine
- Daniel Gajski et al.
SpecSyn Environment

SpecSyn

- Outputs a system-level description, which differs from the input only by the addition of system-level architectural features

- SpecSyn was intended to support wide variety of implementation component technologies, architectures and heuristics, and new versions of such items could be added
Why SpecSyn?

- There was no model available to support embedded systems design (state-transitions, exceptions, forking and program-like computations)

- New program-state machine model (PSM)
  - Combination of hierarchical FSM (Statecharts) and Communicating sequential processes (CSP)

- Supports subset of constraints

Internal Representation

- Specification-level intermediate format (SLIF)

- Access Graph (AG) to show relations between the behaviors/variables (access)

- AG is generic version of a procedure-call graph
Exploration

- Any number of standard processors, custom processors, memories and buses can be allocated

- Three types of functional objects to be partitioned:
  - Variables ⇒ memory components
  - Behaviors ⇒ custom or standard processors
  - Channels ⇒ buses

Partitioning

- Hardware/Hardware and Hardware/Software - both are supported

- Generic partitioning engine (evolving)

- Supports manual partitioning

- Partitioning is guided by cost functions
Estimation

- Two level approach:
  - Preestimation
  - Online-Estimation

- Three metric types:
  - Performance
  - Hardware size
  - Software size

Transformations

- Procedure exlining
  - Redundancy exlining
  - Distinct-comutation exlining
- Procedure inlining
- Process merging
- Process splitting
- Preclustering
- Procedure calling
- Port calling
Refinement

- Interface generation
- Memories
- Arbitration
- Generation
- Validation

What’s next?
Next slide: