Hardware-Software-Co-Design of Parallel and Distributed Systems Using a unique Behavioural Programming and Multi-Process Model with High-Level Synthesis

Stefan Bosse
University of Bremen, Department of Computer Science, Workgroup Robotics, Germany, ISIS Sensorial Materials Scientific Centre, Germany(2)

19.4.2011
Overview

Goals and Requirements in Embedded System Design

1. Design of parallel and distributed embedded systems from algorithmic level
2. Behavioural modelling on programming level using a multi-process model with interprocess-communication and atomic guarded actions
3. ConPro: Concurrent programming and design of parallel hardware and software systems
4. Abstraction of operational hardware blocks and access from programming level
5. Transition from parallel to distributed systems: Required communication architecture for distributed systems
6. Design example: SLIP - a robust and efficient communication protocol stack for multidimensional network topologies (HW/SW implementation)
Concurrent Programming with a Multi-Process Model

- Execution Environment: processes executing instructions in sequential (imperative) order \(\Rightarrow\) Finite State Machine
- Interaction between processes: always using global shared objects \(\Rightarrow\) Interprocess-Communication (IPC)
- Interprocess-Communication = Synchronization: Mutex, Semaphore, ...
- Access of shared resources is serialized: guarded atomic actions
- Access of shared resources is managed by a scheduler: processes blocked until resource is available.
- Hardware Implementation: Mapping of processes to concurrently executing state machines and RTL
- Software Implementation: Mapping of processes to threads (simulated multiprocessing)

Figure 1. Multi-Process Model [mod. CSP/Hoare]

<table>
<thead>
<tr>
<th>P</th>
<th>P</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>O</td>
<td></td>
</tr>
</tbody>
</table>

Multi-Process Model
P: Sequential Process
O: Shared Object
\(-\): Atomic Guarded Action (Communication)

queue q: int;

process a:
begin
  reg x: int;
  x <- 0;
  for i = 1 to 10
    x <- x + q;
  done;
end;

process b:
begin
  reg y: int;
  y <- 0;
  for i = 1 to 10
    q <- y+i;
    y <- y*2;
  done;
end;
**ConPro: Language & Highlevel-Synthesis**

**Synthesis of massive parallel application specific SoC designs AND parallel software from algorithmic & behavioural programming level**

**Programming Model**
- Communicating Sequential Processes
- Guarded shared objects

**Concurrency Model**
- **Control path**: concurrently executed processes
- **Data path**: bounded instruction blocks

**Synchronization**
- Interprocess-Communication directly implementable in hardware: Mutex, Semaphore, Event, Timer, Queue, ...
- Shared objects guarded by mutex scheduler (atomic guarded access)

**Execution Model**
- Process: strict sequential
- HW: Finite-State-Machine & RTL
- SW: light weighted process/thread

**Objects**
- Data storage: registers (CREW), variables (RAM, EREW), ...
- Object orientated programming: abstract objects accessed with methods (hardware blocks)

**Programming Language**
- Imperative with data and control statements
- Explicitly modelled parallelism
- Parameterization on block level: synthesis, scheduling, allocation, object parameters, ...
ConPro Synthesis

- One synthesis compiler is used for hardware- and software targets.
- **Hardware Implementation**: Mapping of processes to concurrently executing state machines and RTL
- **Software Implementation**: Mapping of processes to threads with *different abstraction levels* (high, mid, low)
- Central part for HW/SW co-design: External Module Interface
- Operational blocks like Interprocess-Communication are modelled with unified abstract objects
- Objects can be implemented directly in hardware or software
Multi-stage synthesis flow (HW/SW):

I. Parser, Lexer, Analysis

II. First Intermediate Representation (IR): Abstract Syntax Tree ($p_i \rightarrow \text{AST}_i \forall p\in P$)

III. Second IR: Compiling of AST to linear list of $\mu$Code using parameterizable rule sets

IV. Third IR: Mapping of $\mu$Code to state transition-graphs (STG) and data path: Register-Transfer Level Architecture

V. HW: Compiling of state transition-graphs to hardware model VHDL

VI. SW: Compiling of AST or $\mu$Code to software model C (imperative) or ML (functional)
**ConPro: Abstract Objects and External Module Interface**

**Abstract Objects**

- **Abstraction & Interconnect** of hardware blocks to programming level
- A set of *methods* is used to access objects (read, write, control).
- Unified access interface (HW/SW)
- HW: method access is mapped to handshaked hardware signals
- SW: method access is mapped to function calls

**External Module Interface EMI**

- HW: Objects are modelled on hardware behaviour level (VHDL) and meta language statements (interpreted during synthesis)
- SW: For each object there is a software model, too.

---

Figure 4. Abstract Objects modelled with the External Module Interface
Distributed Architecture and Communication

Distributed Architecture

- Processes and objects can be distributed in domains on algorithmic programming level

Communication Architecture

- Object access is mapped to message based communication and domain routing using serial links

Figure 5. Left: local access, Right: global access of objects using message based communication
Communication

Network Topology

- Each node is connected in a two-dimensional mesh-grid with their direct neighbours.
- Each node is a service end-point and a message router, too.

Delay Insensitive Link

- Problem: local: synchronous system, global: asynchronous system.
- Solution: asynchronous serial links between nodes (GALS wrapper).
- But asynchronous links requires asynchronous logic or.
- Asynchronous link is implemented with synchronous finite state-machine and input signal oversampling.
- Dual-rail encoding and four-phase handshaked protocol is used.
Synthesis of Distributed Systems

Programming Model

- Concurrently executing communicating processes
- Processes and objects are assigned to domains on programming level
- Communication: always using abstract object access (IPC...)
- Local object access: communication still with handshaked hardware signals (HW) or function calls (SW)
- Remote object access: hidden message based communication

Partitioner

- A common programming source is partitioned into independent domain sources and compiled independently by ConPro to HW/SW targets
- Add communication architecture

Figure 7. Extended Design flow using a partitioner
Design Example: SLIP

- Smart and robust communication with Simple Local Intranet Protocol SLIP based on smart delta-routing
- Remote procedure call interface (RPC, application layer)

Figure 8. SLIP network topology (2-dim.) and massive parallel implementation of protocol stack
**Design Example: SLIP**

- Mapping of algorithms and massive parallel data processing to SoC node with high-level synthesis using Con-Pro: 
  - 1. low power
  - 2. minaturization
  - 3. low latency

- Mapping of same sources to software (C) using ConPro, too: 
  - 1. interfacing computers
  - 2. test/simulation

### Table 1. Characteristics of SLIP implementation (HW: Hardware, SW: Software)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLS source code, ConPro</td>
<td>∼ 4000 lines, 34 processes</td>
</tr>
<tr>
<td></td>
<td>30 shared objects (16 queues, 2 timers)</td>
</tr>
<tr>
<td>HW: synthesized VHDL sources</td>
<td>∼ 32000 lines</td>
</tr>
<tr>
<td>SW: synthesized C sources</td>
<td>∼ 5500 lines</td>
</tr>
<tr>
<td>HW: FPGA, Xilinx Spartan III - 1000k</td>
<td>11261/15360 LUT (73 %), 2925 FF</td>
</tr>
<tr>
<td>HW: ASIC, standard cell library LSI_10K</td>
<td>∼ 244k gates, 15k FF  ( \equiv 2.5 \text{mm}^2 \mid 0.18 \mu \text{m} )</td>
</tr>
<tr>
<td>HW: power consumption (FPGA board)</td>
<td>(&lt; 100 \text{mW} ) (including analog electr.)</td>
</tr>
<tr>
<td>HW: performance benchmark R1*</td>
<td>82 clock cycles</td>
</tr>
<tr>
<td>SW: performance benchmark R1*</td>
<td>2305 unit machine instructions</td>
</tr>
</tbody>
</table>

*R1: Sequential part of message routing in SLIP*
Summary and Conclusions

Design of parallel hardware systems

- Complex SoC hardware devices with concurrency on control- and data path level can be efficiently designed from programming level
- The concurrent multi-process model with interprocess-communication and guarded atomic access of shared resources allows designing of complex parallel systems
- Hardware blocks are abstracted and accessed using a method based object-oriented programming style
- Access of objects is fast and efficient (at least 2 clock cycles)

Design of parallel software systems

- Parallel software can be synthesized using the same synthesis frame work and programming language
- Access of objects is fast and efficient (< 100 machine operations)

Design of distributed systems

- Distributed systems can be synthesized using the same synthesis frame work and programming language with an additional partitioner and message based communication
- Preliminary results: automated synthesis of distributed systems using a partitioner results in significant increase of complexity, resources, and latency of remote object access (> 100 clock cycles).
Hardware-Software-Co-Design of Parallel and Distributed Systems Using a unique Behavioural Programming and Multi-Process Model with High-Level Synthesis

Stefan Bosse
University of Bremen, Department of Computer Science, Workgroup Robotics, Germany, ISIS Sensorial Materials Scientific Centre, Germany(2)

19.4.2011