Hardware Synthesis of Complex System-on-Chip Designs for Embedded Systems Using a Behavioural Programming and Multi-Process Model

Stefan Bosse
University of Bremen, Department of Computer Science, Workgroup Robotics, Germany\(^1\), ISIS Sensorial Materials Scientific Centre, Germany\(^2\)

15.9.2010
Overview

Goals and Questions in Embedded System Design

1. **Requirements** and applications of embedded systems in cyber-physical-systems (CPS) and Sensorial Materials (SM)
2. Design of **embedded systems** using different system architectures and design models
3. Behavioural modelling on programming level using a **multi-process model** with interprocess-communication and atomic guarded actions
4. ConPro: **Concurrent Programming of complex** hardware and software systems
5. **Abstraction of hardware blocks** and access from programming level
6. **Design example**: **SensoNET** - a complete sensor network communication and data processing unit implemented 1. in FPGA/ASIC hardware, and 2. in software
Cyber-Physical-Systems (CPS) And Sensorial Materials (SM)

Cyber-Physical-Systems
- Defined by the interaction of the system with its environment
- Tight integration of computation and control with sensing and actuation physical components
- System components: sensors, actuators, data processing, communication → application specific
- CPS must be reliable, adaptable, easy-to-use, and low-power
- **Operation defined on algorithmic level - requires concurrency**

Sensorial Materials
- Network of smart sensor nodes
- Sensor node: sensor, electronics, and data processing
- SM must be reliable, adaptable, highly miniaturized, and low-power

Figure 1. ModuACT robot arm manipulator with network of sensorial materials and actuator joints
Embedded Systems: Architectures and Design Methodologies

Architectures
- Single-processor (SP)
- Multi-processor (MP)
- System-One-Chip (SoC)
- Multi-processor System-On-Chip (MPSoC)
- Network-On-Chip (NoC)
- Application spec. RTL System-On-Chip
- Application spec./extensible processor systems

Design Methodologies
- Software development (C)
- Application specific: hardware-software-co design (C,SystemC)
- Application specific: hardware design on hardware behavioural level
- Top-down / Bottom-up design flows
- Application specific: from behavioural programming level to hardware SoC using High-level Synthesis
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 the 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 2. Multi-Process Model [mod. CSP/Hoare]
ConPro: From Concurrent Programming to Processing

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 implemented 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 (like monitors)

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

- Multi-stage synthesis flow (HW/SW*):
  I. Parser, Lexer, Analysis
  II. Transformations
  III. Reference-Stack Scheduler & Optimizer
  IV. Optimizations (constant folding...)
  V. Compiling of process instruction syntax tree to linear list of \( \mu \text{Code} \) (intermediate representation) using parameterizable rule sets
  VI. Transformations
  VII. Basicblock Scheduler & Optimizer
  VIII. Compiling of state transition-graphs from \( \mu \text{Code} \), finally VHDL

- **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)

---

**Figure 3. ConPro Synthesis with HW/SW targets**

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

- Synthesis
  - HW: Hardware
  - SW: Software
  - EMI: External Module Interface
  - RTL: Register-Transfer Logic
  - FSM: Finit-State Machine

Stefan Bosse - Hardware Synthesis of Complex System-on-Chip Designs for Embedded Systems
ConPro Programming Language: Highlights

- Execution environment is a process:
  ```
  process pxyz:
  begin ... end;
  ```

- Shared function blocks (process env.):
  ```
  function fxyz (x:int[8])
  return (t: bool):
  begin ... end;
  ```

- Data types: true bit-scaled:
  ```
  int[N], logic[N], char, bool
  ```

- Product types: structures and arrays:
  ```
  type s: { x: int[8]; y: int[10];};
  array a: reg[10] of int[5];
  ```

- Storage objects: registers, variables (in memory blocks), queues:
  ```
  reg xyz: int[21];
  var v1,v2: char in ram1;
  ```

- Exceptions
  ```
  try .. raise .. with
  ```

- Parameterizable block environments:
  ```
  begin
  end with param=value [and p2=v2..]
  ```

- Parameterizable abstract objects:
  ```
  open ADT;
  object o1: adt with width=10;
  o1.write(x,1);
  ```

- Interprocess-communication = abstract object types
  ```
  open Mutex; object mu1: mutex with
  scheduler="static";
  ```

- Control statements: branches, loops:
  ```
  for i = 1 to 10 do ...
  if x < y then ... else
  while a = true do ...
  match c with ...
  z ← fxyz(1); -- Function call
  mu1.lock (); -- ADTO call
ConPro: Abstracting & Interfacing of Hardware Blocks

Component Structures and Signals

- **Signals** are interconnection elements without a storage model
- **Component Structures** bind signals to a port structure
- A component structure can be used 1. to instantiate and access external hardware, 2. to create the toplevel hardware interface
- Signals can be used in expressions

External Module Interface EMI

- **Abstraction & Interconnect** of hardware blocks to algorithmic programing level using abstract *objects* and *methods* to access hardware blocks.
- Hardware blocks are modelled on hardware behaviour level (VHDL) and meta language statements (interpreted during synthesis)
- Hardware blocks are accessed by a set of *methods* from programming level, e.g. read, write, and control operations
- EMI provides software models, too!
Design Example: SensoNET

- Complete application of a sensor node in a sensor network (sensorial mat.)
- Smart and robust communication with Simple Local Intranet Protocol SLIP
- Remote procedure call interface (RPC, application layer)
- Data acquisition with preprocessing of sensor signals

Figure 4. SensoNET used in sensorial material: network of smart strain gauge sensor nodes

Smart Node Architecture

Irregular Network Topology

Twodimensional Sensornetwork
Design Example: SensoNET

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

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

Table 1. Characteristics of SensoNET 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>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 ≈ 2.5mm²</td>
</tr>
<tr>
<td>HW: power consumption (FPGA board)</td>
<td>&lt; 250mW (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 Outlook

Design of parallel SoC

- Complex SoC systems 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-orientated programming style

Design of parallel software

- Parallel software can be synthesized using the same synthesis framework and programming language

Outlook: Design of distributed systems

- From parallel to distributed systems
- Actually shared objects on hardware level are accessed by signals → transformation of signals to message based communication
- Objects and processes distributed over hardware and software components