HW/SW Design Space Exploration on the Production Cell Setup

Communicating Process Architectures 2009, Formal Methods Week
Eindhoven University of Technology, The Netherlands, 04-11-2009

Marcel A. Groothuis, Jan F. Broenink

Control Engineering, Department of Electrical Engineering,
University of Twente, The Netherlands
Contents

- Introduction
  - Goals & Challenges
  - Embedded Control Systems Software
- Design Space Exploration
- Test Case
  - Demonstration Setup: Production Cell System
  - 6 Embedded Control Systems Software implementations
- Production Cell ECS Implementations
  - CPU implementations (4)
  - FPGA implementations (2)
- Evaluation
- Conclusions & Ongoing work
Introduction

Goals & Challenges

- Realization of Embedded Control System (ECS) software
  - For mechatronics & robotic applications

- Design Methodology
  - Model-driven ECS software design
  - Dependable software
  - Supporting tool chain

- ECS design challenges
  - Large design space
  - Heterogeneous nature
  - Special demands on the software
Essential Properties Embedded Control Software

- **Purpose**: control physical systems
- **Dynamic** behaviour of the physical system essential for SW
- **Dependability**: Safety, Reliability

Embedded Control System (ECS) software

- **Layered structure**

Real-time constraints with low-latency requirement

- Combination of time-triggered & event driven parts
  - Multiple Models of Computation (MoC)
  - Multiple Modeling formalisms
Introduction Design Method ECS SW

- **Approach**
  - Stepwise & local refinement
    - From models towards ECS code
  - Verification by simulation & model checking

- **Way of Working**
  - **Discrete Event**
    - Abstract interactions concurrent actors
    - Interaction between different MoCs
    - Timing low-level behaviour
  - **Continuous Time**
    - Model & Understand Physical system dynamics
    - Simplify model, derive the control laws
    - Interfaces & target
      - Add non-ideal components (AD, DA, PC)
      - Scaling/conversion factors
    - Integrate DE & CT into ECS SW
### Introduction Design Space Exploration

- **Embedded Control System**
  - Large Design Space
  - (Many) Design Choices
    - Restrict solution space
    - Smaller pyramid

- **Examples choices**
  - Modelling formalisms & languages
  - Operating System choice
  - Parallellism
    - Sequential –or-
      Parallel solution \(\Leftrightarrow\) resource usage
  - Architecture
    - CPU \(\Leftrightarrow\) FPGA, distributed \(\Leftrightarrow\) central

- **Reachable solutions**
  - Dependent on all choices
Production cell demonstrator
- Based on:
  - Stork Plastics Molding machine

- Architecture:
  - CPU (ECS / FPGA programmer)
  - FPGA (digital I/O / ECS)

- 6 Production Cell units
  - Action in the production process
    - Moulding, Extraction, Transportation, Storage
  - Synchronize with neighbours
  - Deadlock possible on > 7 blocks
Embedded Control System implementations

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Name</th>
<th>Data type</th>
<th>Target</th>
<th>Realization</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>gCSP RTAI Linux</td>
<td>Floating point</td>
<td>CPU</td>
<td>Yes</td>
</tr>
<tr>
<td>B</td>
<td>POOSL</td>
<td>Floating point</td>
<td>CPU</td>
<td>Yes</td>
</tr>
<tr>
<td>C</td>
<td>Ptolemy II</td>
<td>Floating point</td>
<td>CPU</td>
<td>Yes</td>
</tr>
<tr>
<td>D</td>
<td>gCSP QNX RTOS</td>
<td>Floating point</td>
<td>CPU</td>
<td>Partial</td>
</tr>
<tr>
<td>E</td>
<td>gCSP Handel-C int (CPA 2008)</td>
<td>Integer</td>
<td>FPGA</td>
<td>Yes</td>
</tr>
<tr>
<td>F</td>
<td>gCSP Handel-C float</td>
<td>Floating point</td>
<td>FPGA</td>
<td>Yes</td>
</tr>
<tr>
<td>G</td>
<td>SystemCSP</td>
<td>-</td>
<td>-</td>
<td>No</td>
</tr>
</tbody>
</table>

Different choices

OS:
- RTAI Linux
- QNX
- No OS

Formalisms:
- CSP
- CCS
- Multi MoC

Tools:
- gCSP, FDR2
- 20-sim
- POOSL
- Ptolemy II

And many more…
• Introduction
  • Goals & Challenges
  • Embedded Control Systems Software
• Design Space Exploration
• Test Case
  • Demonstration Setup: Production Cell System
  • 6 Embedded Control Systems Software implementations
• Production Cell ECS Implementations
  • CPU implementations (4)
  • FPGA implementations (2)
• Evaluation
• Conclusions & Ongoing work
Focus: proof of concept gCSP
- Proof of concept gCSP for Embedded Control Systems software
- Combination of untimed CSP and real-time Linux

Realization
- Bottom up
- 6 Semi-independent units ➞ 6 PARs
- PRIPAR for real-time levels
- Periodic timing
  - TimerChannels
  - ECS SW ↔ Environment
  - Rendezvous with OS timer
- Formal check with FDR2
- Generated code from
  - gCSP + 20-sim
Results

- gCSP and CSP are usable for ECS software
  - Graphical process & channel structured
  - Graphical Finite State Machine diagram support wanted
  - Debugging CSP processes difficult (textual) \(\Rightarrow\) gCSP animation CPA2008
- Formal verified process/channel structure (CSPm \(\Rightarrow\) FDR2)
- Real-time behaviour gCSP code + CTC++ library + RTAI Linux
  - Missed deadlines; large process switch overhead; high CPU load
  - Challenge: Discrete Event CSP + Time Triggered loop control

Improvements

- Timing implementation
  - CSP scheduling v.s. hard deadlines \(\Rightarrow\) QNX RTOS version CT library
- Modeling
  - Diagram structure, Interaction, Hierarchy
- **POOSL = Parallel Object Oriented Specification Language**
  - CCS + Timing extension
  - Modeling high level behaviour Embedded Systems

- **Focus**
  - Test timing
  - Integration DE & CT
  - Structured modeling
    - Concurrency & Interaction
    - DE ↔ CT interfacing
  - Timing

- **Realization**
  - Top-down
  - No formal check

- **Results**
  - Separated concurrent design SW layers
    - DE (high level, CT (low level)}
Previous approaches
- Multiple modeling tools (DE, CT), code integration

Ptolemy II: Heterogeneous modeling tool
- Many Models of Computation (MoC)
  - Continuous Time, Discrete Event, Synchronous Dataflow, CSP, Finite State Machine, ...

Focus
- Tryout single modeling tool approach & multi MoC approach

Realization
- Hierarchical model
- Whole setup

No formal checks
### Results

- Single All-in-one design model, no concurrent design possible
- Time saving & easy early integration testing
- Promising approach, but not yet mature enough
  - Extensions & patches to Ptolemy II needed for
    - Code generation: (real-)time support, submodel generation
    - Mechanics (Continuous Time) & Loop Controller modeling (building blocks), …
  - Not all available MoCs can generate code
Introduction
- Goals & Challenges
- Embedded Control Systems Software

Design Space Exploration

Test Case
- Demonstration Setup: Production Cell System
- 6 Embedded Control Systems Software implementations

Production Cell ECS Implementations
- CPU implementations (4)
- FPGA implementations (2)

Evaluation

Conclusions & Ongoing work
Feasibility study on motion control in FPGA
- Exploit parallelism
- Accurate timing
- Model-based design

Choice
- Modeling tools
  - gCSP + 20-sim (output: floating point control algorithm)
- Implementation
  - Handel-C
  - No (soft core) CPU
  - Small size Xilinx Spartan III FPGA
## Loop Controller Floating point alternatives

<table>
<thead>
<tr>
<th>Alternative</th>
<th>Benefit</th>
<th>Drawback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating point library, CPA 2009</td>
<td>High precision; re-use existing controller</td>
<td>Very high logic utilization</td>
</tr>
<tr>
<td>Fixed point library</td>
<td>Acceptable precision</td>
<td>High logic utilization</td>
</tr>
<tr>
<td>External FPU</td>
<td>High precision; re-use existing controller</td>
<td>Only high-end FPGA; expensive</td>
</tr>
<tr>
<td>Soft-core CPU+FPU</td>
<td>High precision; re-use existing controller</td>
<td>Only high-end FPGA; expensive</td>
</tr>
<tr>
<td>Soft-core FPU</td>
<td>High precision; re-use existing controller</td>
<td>Scheduler / resource manager required</td>
</tr>
<tr>
<td>Integer, CPA 2008</td>
<td>Native data type</td>
<td>Low precision in small ranges; adaptation of the controllers needed</td>
</tr>
</tbody>
</table>

**Trade-off between numerical precision and logic cell utilization**
Real parallelism
- 6 Production Cell Units run parallel

Integer algorithm (no floating point)
- Manual translation $\Rightarrow$ time consuming

Accurate timing

Estimated FPGA Usage
- Xilinx Spartan 3s1500

<table>
<thead>
<tr>
<th>Element</th>
<th>LUTs (amount)</th>
<th>Flipflops (amount)</th>
<th>Memory</th>
<th>Used ALUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID controllers</td>
<td>13.5% (4038)</td>
<td>0.4% (126)</td>
<td>0.0%</td>
<td>0</td>
</tr>
<tr>
<td>Motion profiles</td>
<td>0.9% (278)</td>
<td>0.2% (72)</td>
<td>0.0%</td>
<td>0</td>
</tr>
<tr>
<td>I/O + PCI</td>
<td>3.6% (1090)</td>
<td>1.6% (471)</td>
<td>2.3%</td>
<td>0</td>
</tr>
<tr>
<td>S&amp;C framework</td>
<td>10.3% (3089)</td>
<td>8.7% (2616)</td>
<td>0.6%</td>
<td>0</td>
</tr>
<tr>
<td>Free</td>
<td>71.7% (21457)</td>
<td>89.1% (26667)</td>
<td>97.1%</td>
<td>32</td>
</tr>
</tbody>
</table>

PID controllers take 50% of the used space, <1% of the code

PID controllers run || @ 1 ms with idle time 99.95%
- Sequential ⇔ Pipelined Handel-C floating point library
- Sequential ⇔ Parallel Production Cell Unit (PCU) execution
- 32 bit Handel-C ⇔ 16-bit Xilinx Coregen floating point
- Soft-core or hard-core CPU with floating point unit.

**Diagram:**

- **Method**: Sequential (Seq) vs. Parallel (Par)
- **Accuracy**: 32 bit vs. 16 bit/32-bit
- **Language**: Handel-C vs. ANSI-C
  - Handel-C
  - Coregen + Handel-C wrapper
  - Softcore or hardcore CPU with FPU
- **Support Library/IP-core**: Floating point library
- **PCU execution order**: (1) Seq, (2) Par, (3) Seq, (4) Par, (5) Seq, (6) Par, (7) Seq
- **Implementation platform**: FPGA

*not yet implemented*
Results FPGA Usage (floating point)

- Less parallelism
  - Sequential PCU execution, but still meeting our deadlines
  - Sequential floating point calculation
  - Central re-used (scheduled) Motion profile + PID controller process

- Estimated FPGA Usage
  - Xilinx Spartan 3s1500

<table>
<thead>
<tr>
<th>Element</th>
<th>LUTs (amount)</th>
<th>Flipflops (amount)</th>
<th>Memory</th>
<th>Used ALUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating point library + wrappers</td>
<td>27.4% (8191)</td>
<td>19.7% (5909)</td>
<td>0.0%</td>
<td>4</td>
</tr>
<tr>
<td>PID controllers</td>
<td>4.2% (1251)</td>
<td>0.3% (91)</td>
<td>0.0%</td>
<td>0</td>
</tr>
<tr>
<td>Motion profiles</td>
<td>1.1% (314)</td>
<td>0.5% (163)</td>
<td>0.0%</td>
<td>0</td>
</tr>
<tr>
<td>I/O + PCI</td>
<td>4.1% (1250)</td>
<td>1.8% (534)</td>
<td>2.3%</td>
<td>0</td>
</tr>
<tr>
<td>S&amp;C framework</td>
<td>5.6% (1666)</td>
<td>4.2% (1250)</td>
<td>0.3%</td>
<td>0</td>
</tr>
<tr>
<td>Free</td>
<td>57.6% (21457)</td>
<td>73.5% (22005)</td>
<td>97.4%</td>
<td>28</td>
</tr>
</tbody>
</table>

Red = more resource usage, Green = less resource usage compared to int version

Floating point library takes 37% of the used space
Evaluation

- Common CPU & FPGA
  - Hierarchical process-oriented implementations
  - Layered ‘software’ structure with DE + CT/DT parts
  - Create re-usable standardized building blocks

- Modeling process structures ⇔ Implementation efficiency
  - Many small processes ⇔ scheduling overhead
  - Often multiple channels between them ⇒ Needed: buses

- Formal verification
  - User-friendly model-to-formal language translation still lacking

- FPGA implementations
  - Alternative for common CPU / PLC solutions
  - Accurate timing
  - Design time is higher and black box debugging is more difficult
Evaluation Example Building Block

Production Cell Unit (PCU)

Controller

Command

Safety

FPGA

Host PC

User Interface

Low-level Hardware

Controller handshake channel
State channel
User interface channel
Hardware interface channel
Error channel

From Previous PCU
To previous PCU
From previous PCU

To next PCU
From Next PCU
To Next PCU

To Production Cell
From Production Cell

04-11-2009 HW/SW Design Space Exploration on the Production Cell Setup
Conclusions & Ongoing work

- Insight maturity (academic) tools for ECS design
- Standardized process-oriented layered ECS structure
- Trade-off CPU / FPGA solution
  - CPU: low design time, real-time behaviour → critical issue
  - FPGA: higher design time, more complicated, accurate timing

- Design Space Exploration results
  - 7 different implementations for same setup
  - Valuable information for improvement design methods & tooling

- Ongoing work
  - gCSP version 2
  - Design methodology
Production Cell, CPU controlled: gCSP RTAI version
Approach
- Stepwise & local refinement
  - From models towards ECS code
  - Verification by simulation & model checking

Way of Working
- Discrete Event
  - Abstract interactions between concurrent actors
  - Interaction between different MoCs
  - Timing low-level behavior
- Continuous Time
  - Model & Understand Physical system dynamics
  - Simplify model, derive the control laws
  - Interfaces & target
    - Add non-ideal components (AD, DA, PC)
    - Scaling/conversion factors
  - Integrate DE & CT into ECS SW

Background Design Method ECS SW

- Approach
- Stepwise & local refinement
  - From models towards ECS code
  - Verification by simulation & model checking

- Way of Working
  - Discrete Event
    - Abstract interactions between concurrent actors
    - Interaction between different MoCs
    - Timing low-level behavior
  - Continuous Time
    - Model & Understand Physical system dynamics
    - Simplify model, derive the control laws
    - Interfaces & target
      - Add non-ideal components (AD, DA, PC)
      - Scaling/conversion factors
    - Integrate DE & CT into ECS SW
Focus:
- Test QNX real-time operating system

QNX
- Real-time μ-kernel
- Message passing: channels
- Transparent distribution
  - Network & local channels are /dev/ nodes