UNIVERSITY OF TWENTE.

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

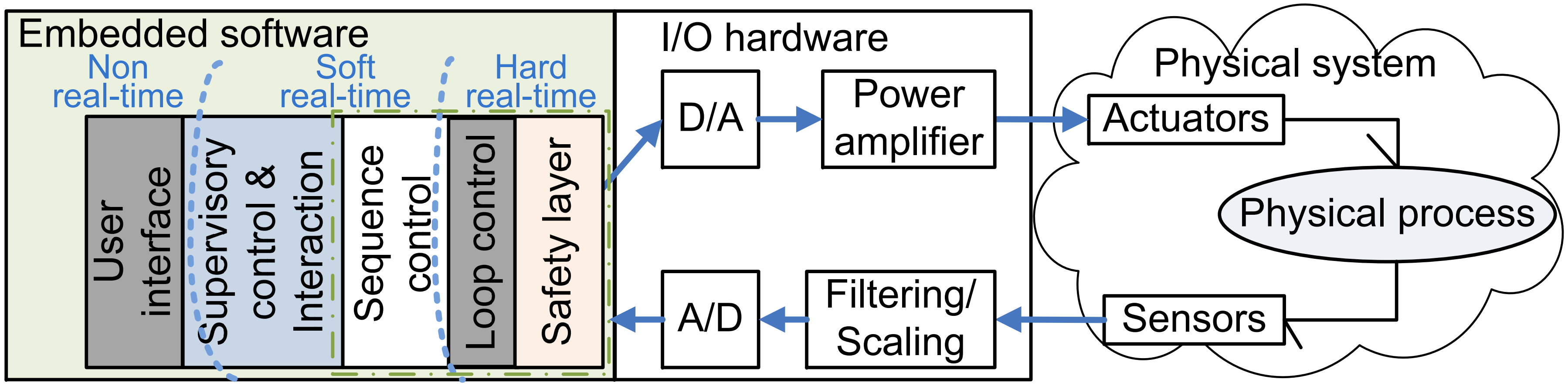
Introduction Goals & Challenges

- 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

Introduction Embedded Control System

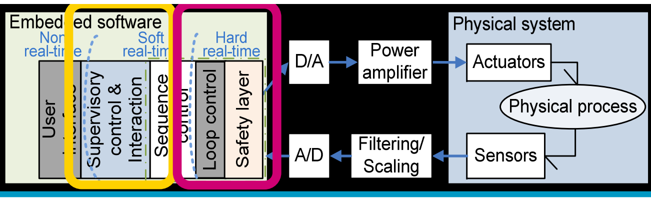
UNIVERSITY OF TWENTE.

- 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

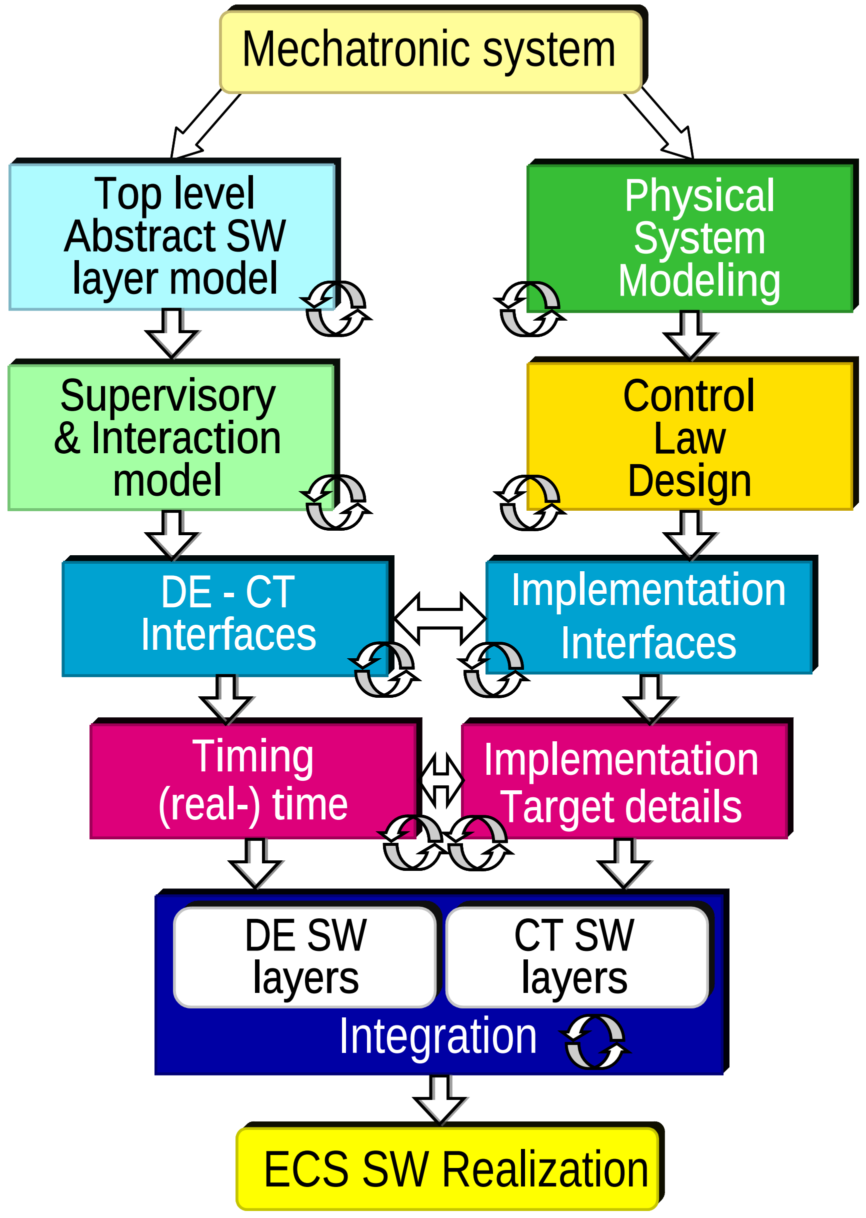


  - Real-time constraints with low-latency requirement
  - Combination of time-triggered & event driven parts
    - Multiple Models of Computation (MoC)
    - Multiple Modeling formalisms

Introduction Design Method ECS SW

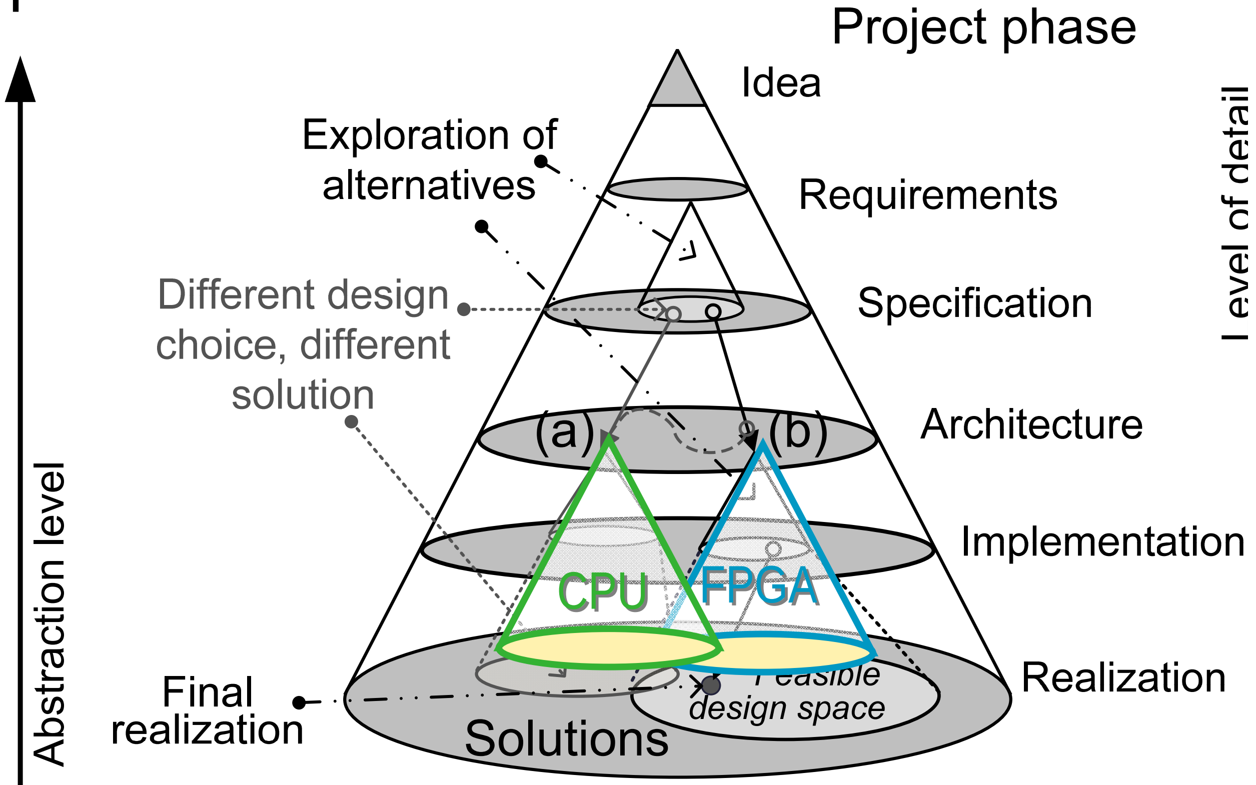


- Approach
  - Stepwise & local refinement
    - From models towards ECS code
  - Verification by simulation & model checking
- Way of Working
  - Discrete Event
    - Abstract interactions concurrent actors
    - Interaction between different MoCs
    - Timing low-level behaviour
  - Continuous Time
    - 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 ⇔resource usage
  - Architecture
    - CPU ⇔ FPGA, distributed ⇔central
- Reachable solutions
  - Dependent on all choices

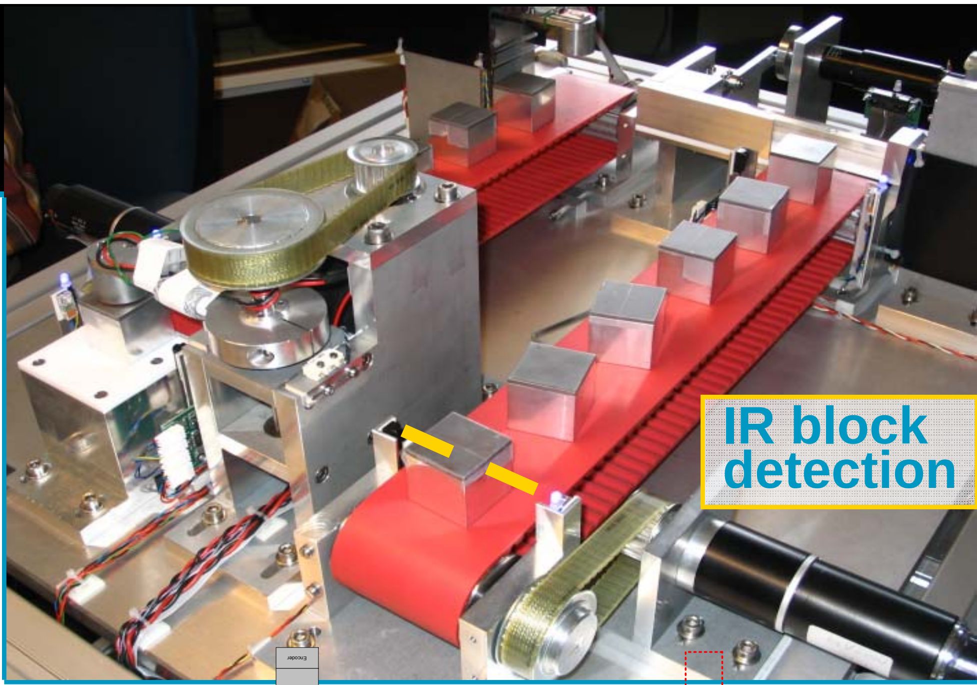


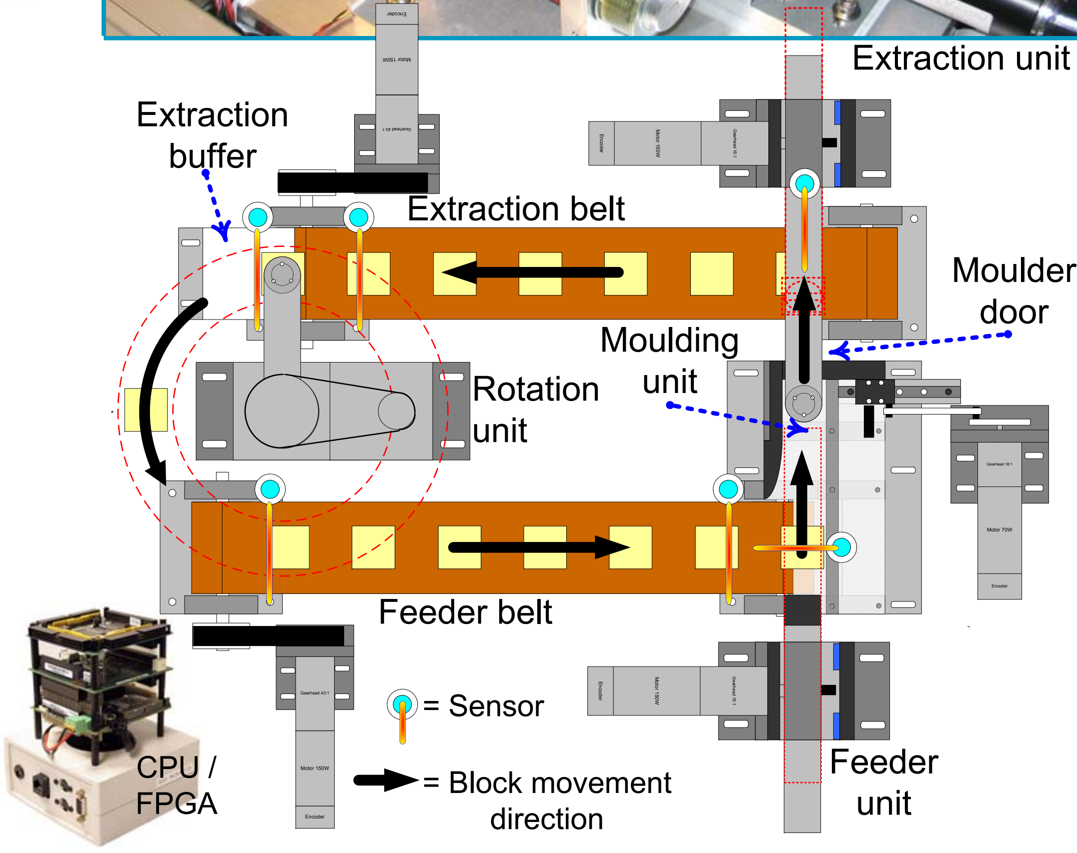
Test Case Production Cell

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





Production Cell ECS implementations

- Embedded Control System implementations

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

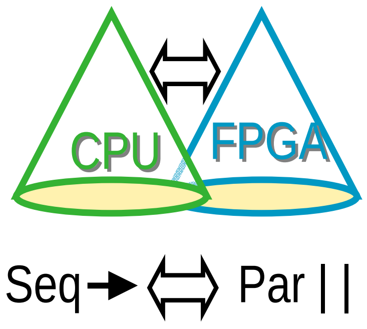
- Different choices

OS:
- RTAI Linux
- QNX
- No OS

Formalisms:
- CSP
- CCS
- Multi MoC

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

Architecture:



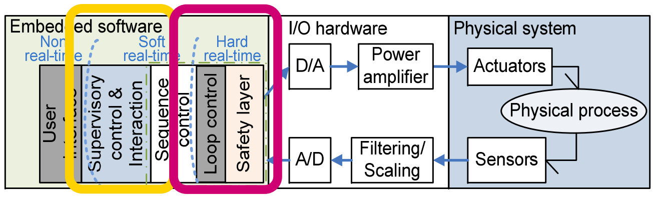
- And many more…

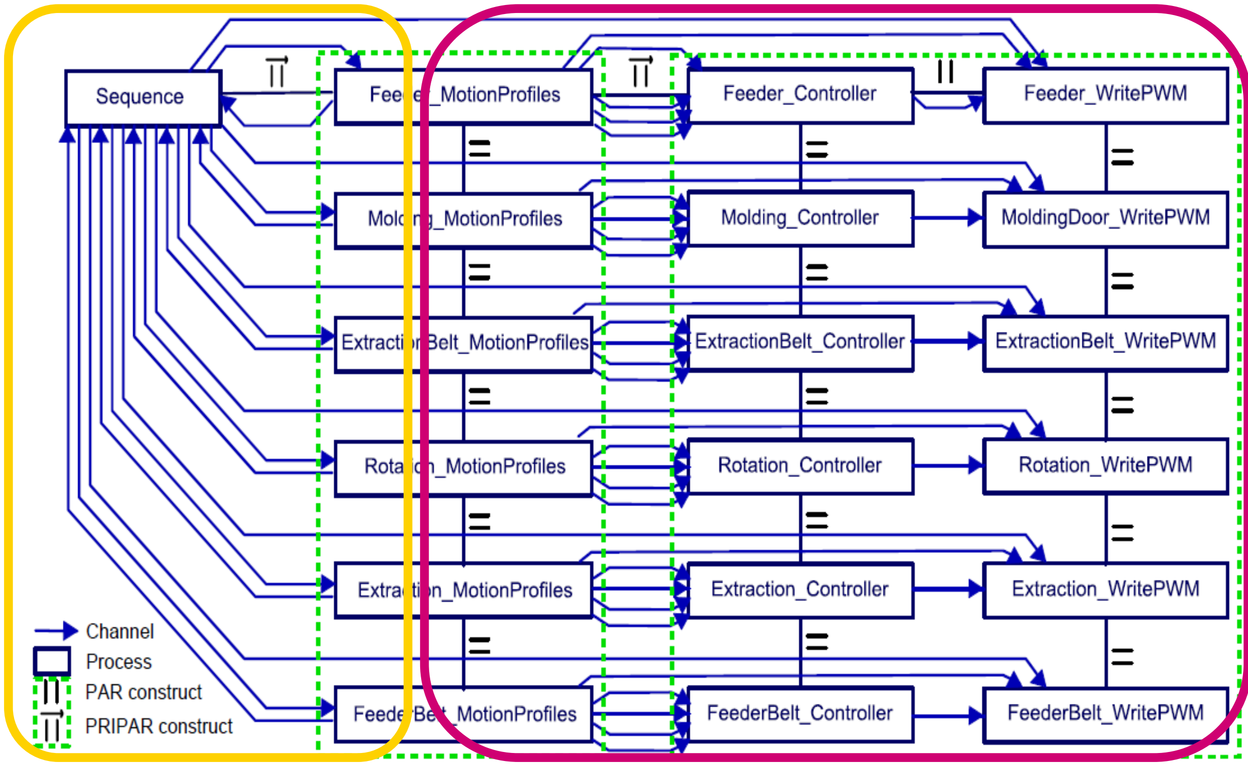
Contents

CPU gCSP RTAI (A)

UNIVERSITY OF TWENTE.

- 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





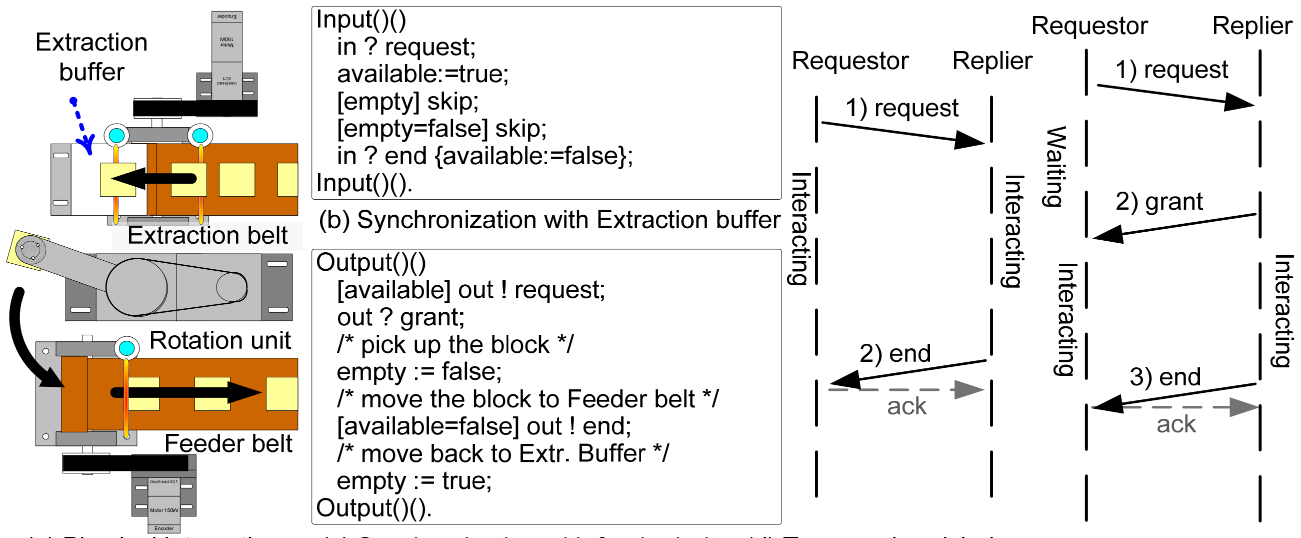
CPU gCSP RTAI (A)

- 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) ➔ gCSP animation CPA2008
  - Formal verified process/channel structure (CSPm ➔ 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 ➔ QNX RTOS version CT library
  - Modeling
    - Diagram structure, Interaction, Hierarchy

CPU POOSL (B)

- POOSL = Parallel Object Oriented Specification Language TU/e
  - 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)

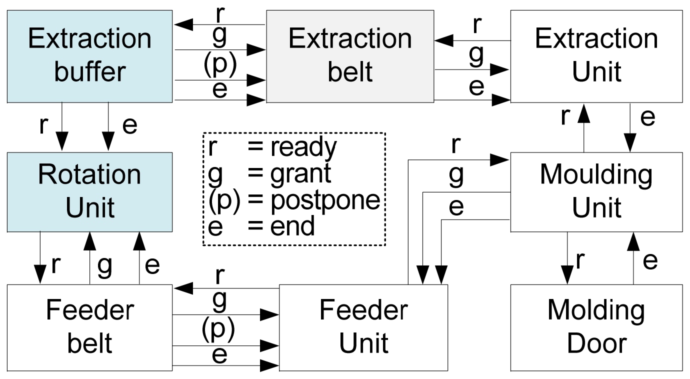


```
Input()()
   in ? request;
   available:=true;
   [empty] skip;
   [empty=false] skip;
   in ? end {available:=false};
Input()().
```

(b) Synchronization with Extraction buffer

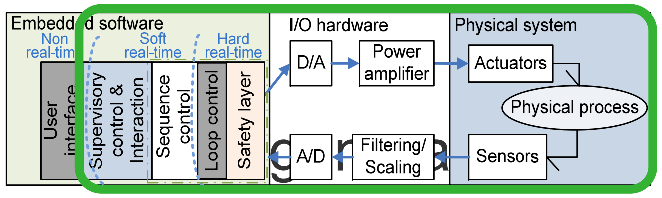
```
Output()()
   [available] out ! request;
   out ? grant;
   /* pick up the block */
   empty := false;
   /* move the block to Feeder belt */
   [available=false] out ! end;
   /* move back to Extr. Buffer */
   empty := true;
Output()().
```

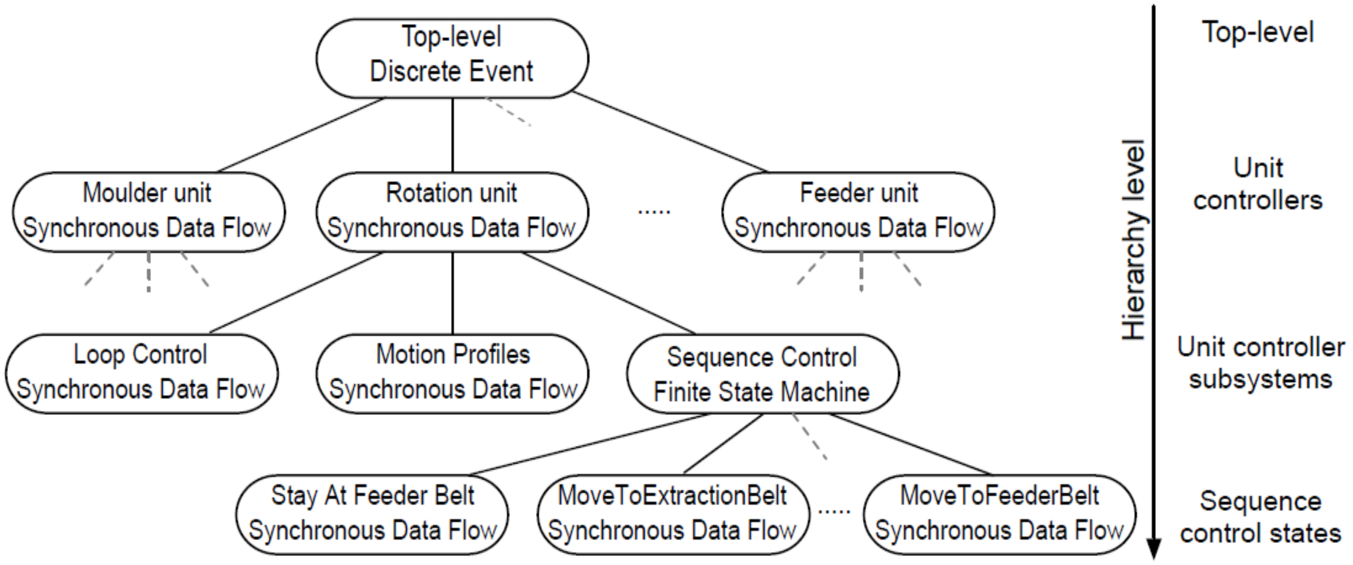
(a) Physical interactions (c) Synchronization with feeder belt (d) Two-way handshake



CPU Ptolemy II (C)

- Previous approaches
  - Multiple modeling tools (DE, CT), code integration
- Ptolemy II: Heterogeneous modeling tool
  - Many Models of Computation (MoC)
    - Continous Time, Discrete Event, Synchronous Dataflow, CSP, Finite State Machine, …
- Focus
  - Tryout single modeling tool approach & multi MoC approach
- Realization
  - Hierarchical model
    - Whole setup



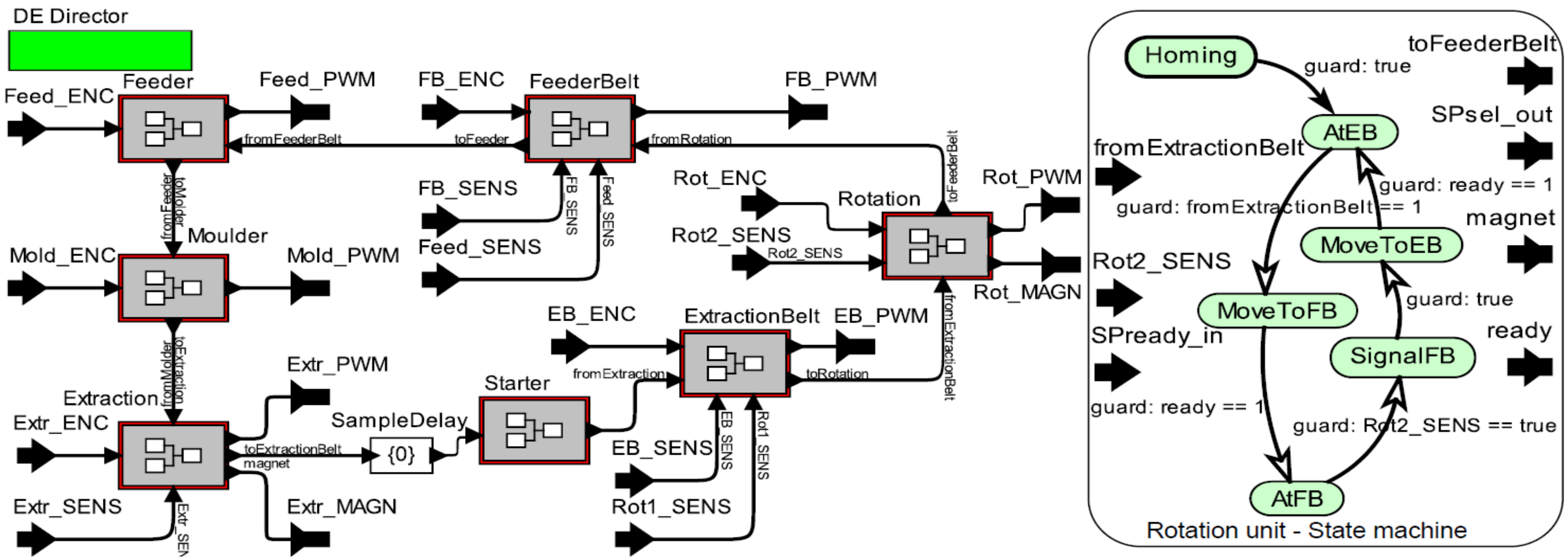

Top-level Discrete Event
Moulder unit Synchronous Data Flow
Rotation unit Synchronous Data Flow
.....
Feeder unit Synchronous Data Flow
Loop Control Synchronous Data Flow
Motion Profiles Synchronous Data Flow
Sequence Control Finite State Machine
Stay At Feeder Belt Synchronous Data Flow
MoveToExtractionBelt Synchronous Data Flow
.....
MoveToFeederBelt Synchronous Data Flow
Hierarchy level
Top-level
Unit controllers
Unit controller subsystems
Sequence control states

  - No formal checks

CPU Ptolemy II (C)

UNIVERSITY OF TWENTE.

- 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



Contents

UNIVERSITY OF TWENTE.

FPGA gCSP Handel-C

- 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

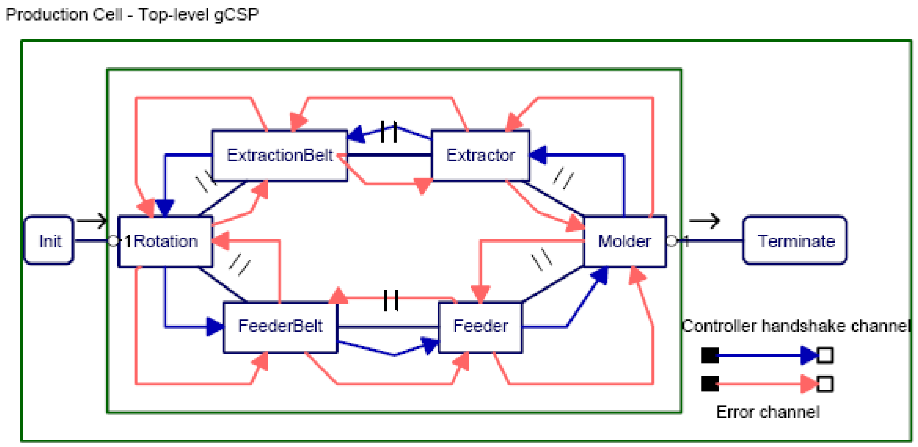
UNIVERSITY OF TWENTE.

Alternative	Benefit	Drawback
Floating point library, CPA 2009	High precision; re-use existing controller	Very high logic utilization
Fixed point library	Acceptable precision	High logic utilization
External FPU	High precision; re-use existing controller	Only high-end FPGA; expensive
Soft-core C		ation unless
Soft-core FPU	High precision; re-use existing controller	Scheduler / resource manager required
Integer, CPA 2008	Native data type	Low precision in small ranges; adaptation of the controllers needed

Trade-off between numerical precision and logic cell utilization

Results FPGA Usage (integer)

- Real parallelism
  - 6 Production Cell Units run parallel
- Integer algorithm (no floating point)
  - Manual translation ⇔ time consuming
- Accurate timing
- Estimated FPGA Usage
  - Xilinx Spartan 3s1500



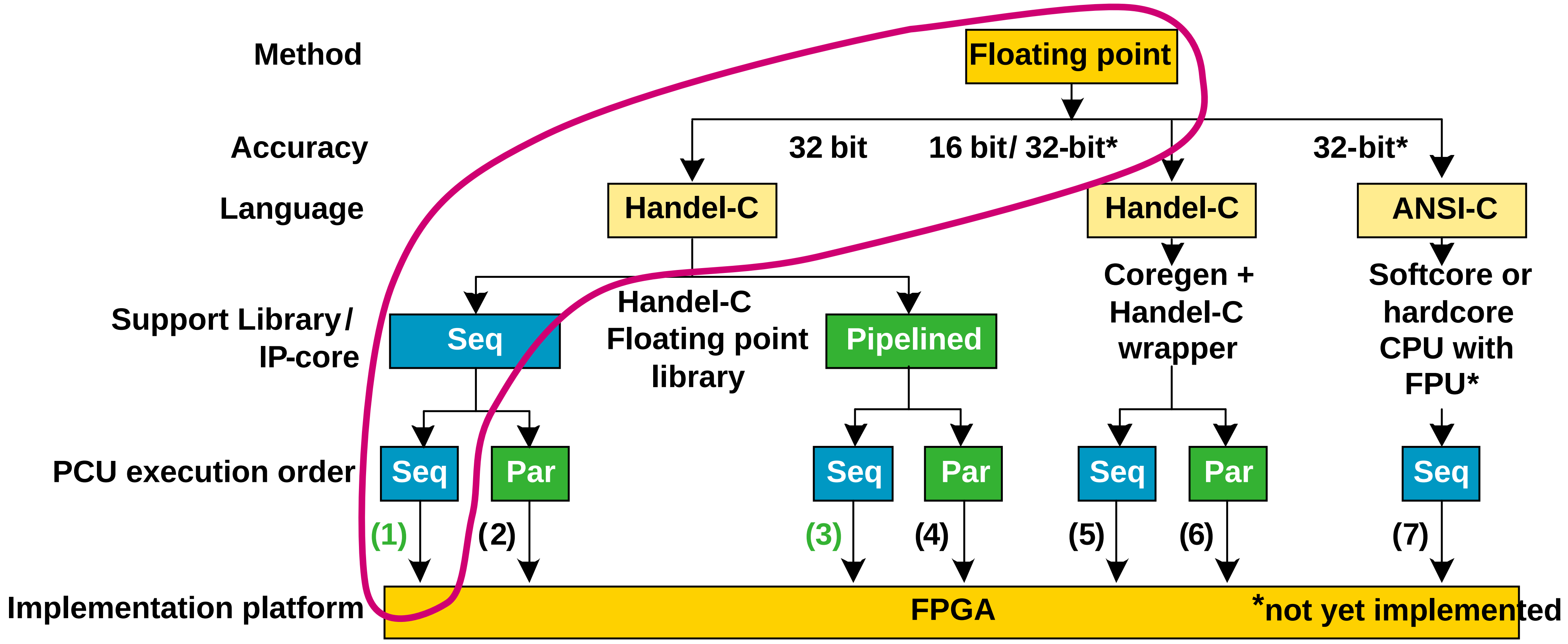
Element	LUTs	(amount)	Flipflops	(amount)	Memory	Used ALUs
PID controllers	13.5%	(4038)	0.4%	(126)	0.0%	0
Motion profiles	0.9%	(278)	0.2%	(72)	0.0%	0
I/O + PCI	3.6%	(1090)	1.6%	(471)	2.3%	0
S&C framework	10.3%	(3089)	8.7%	(2616)	0.6%	0
Free	*71.7%*	*(21457)*	*89.1%*	*(26667)*	*97.1%*	*32*

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

PID controllers run | | @ 1 ms with idle time 99,95%

FPGA Trade-offs floating point

- 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.



Results FPGA Usage (floating point)

UNIVERSITY OF TWENTE.

- 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

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

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

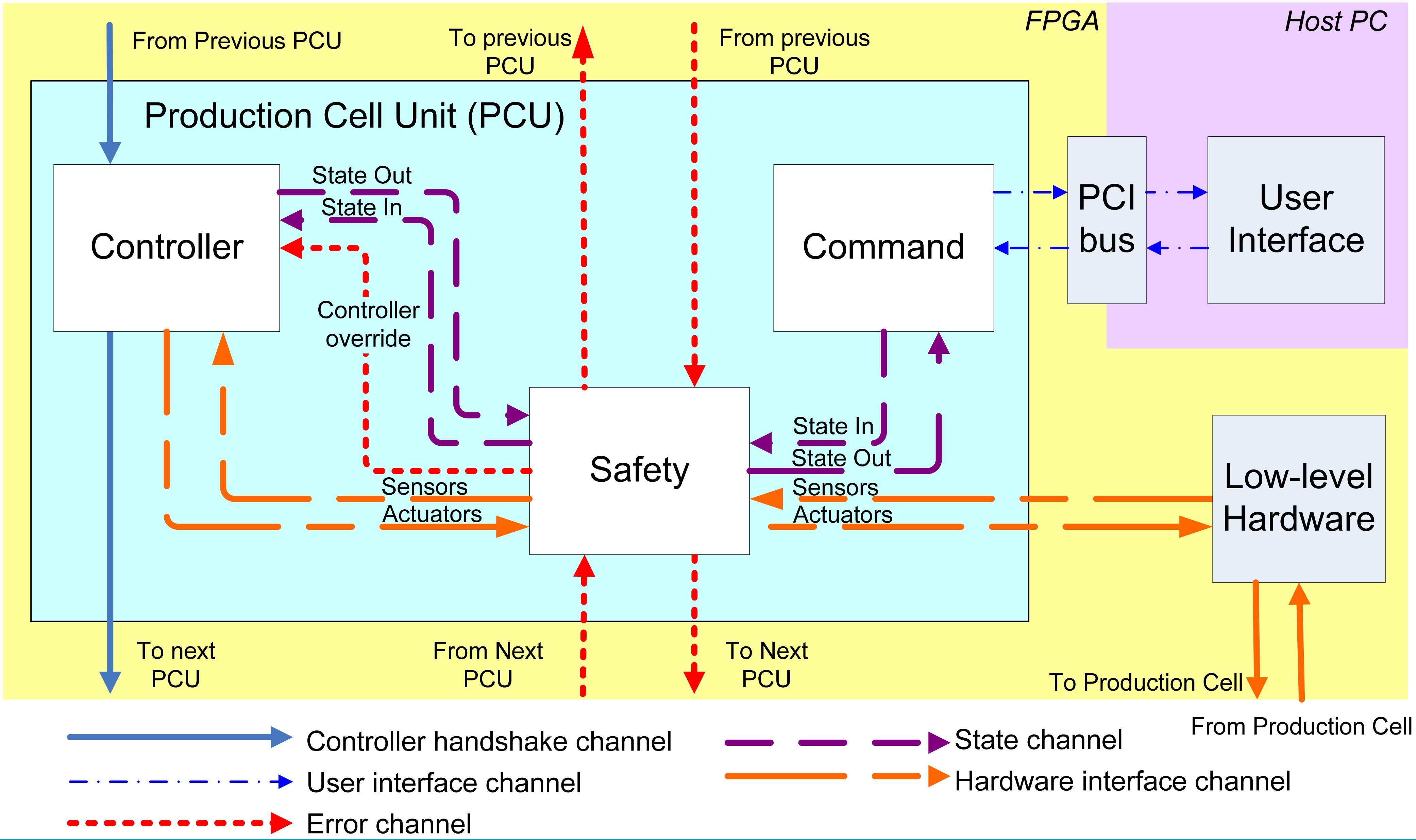
Floating point library takes 37% of the *used* space

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

UNIVERSITY OF TWENTE.



Conclusions & Ongoing work

UNIVERSITY OF TWENTE.

- 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

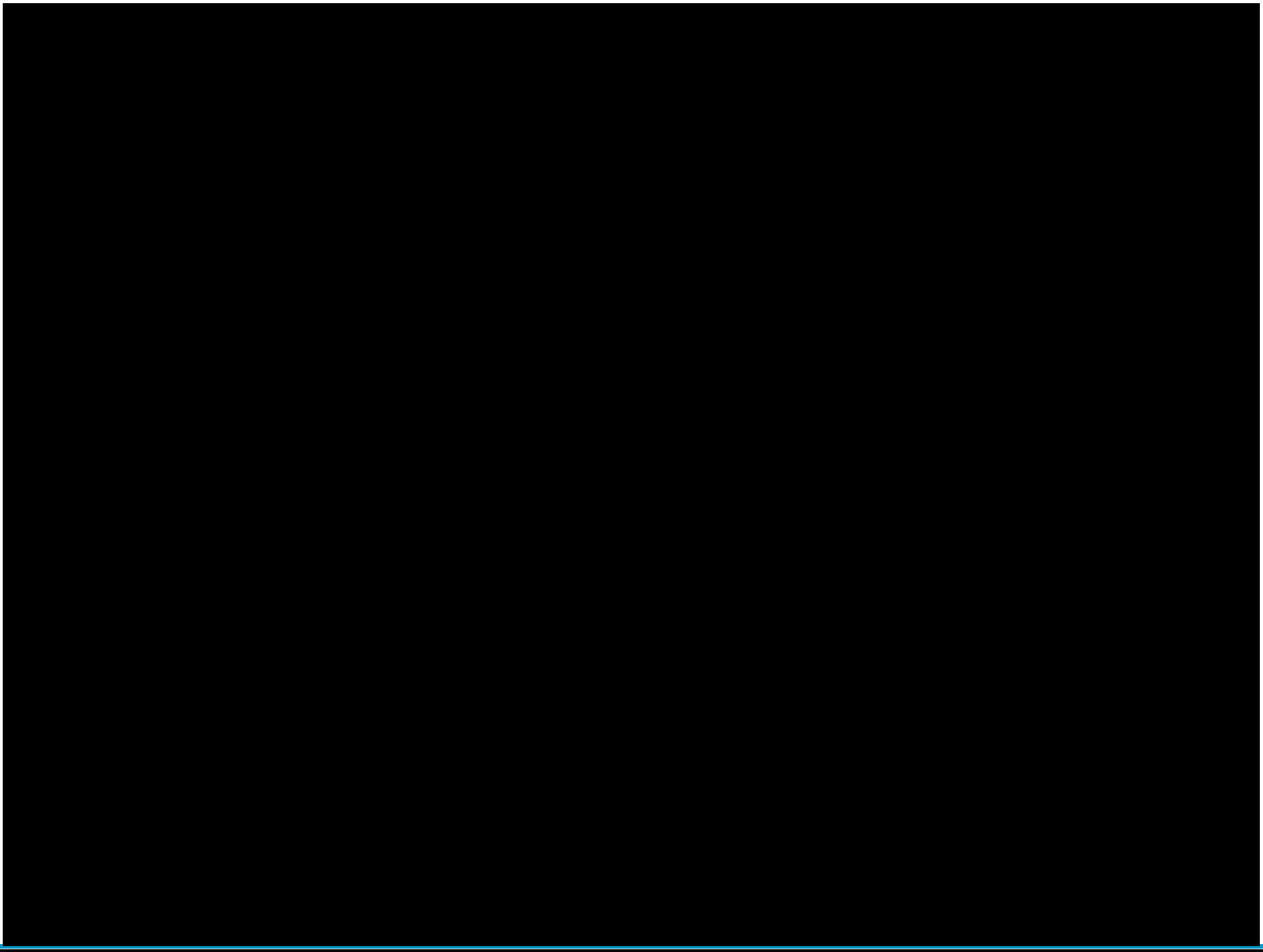
- Design Space Exploration results
  - 7 different implementations for same setup
  - Valuable information for improvement design methods & tooling

- Ongoing work
  - gCSP version 2
  - Design methodology

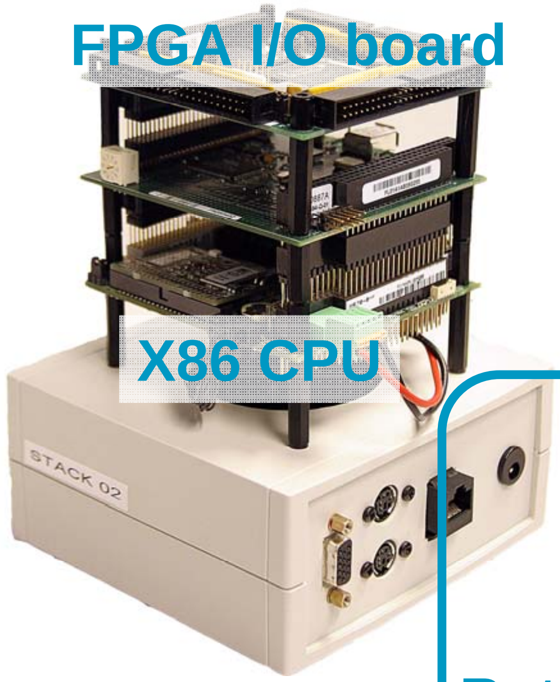
Movie

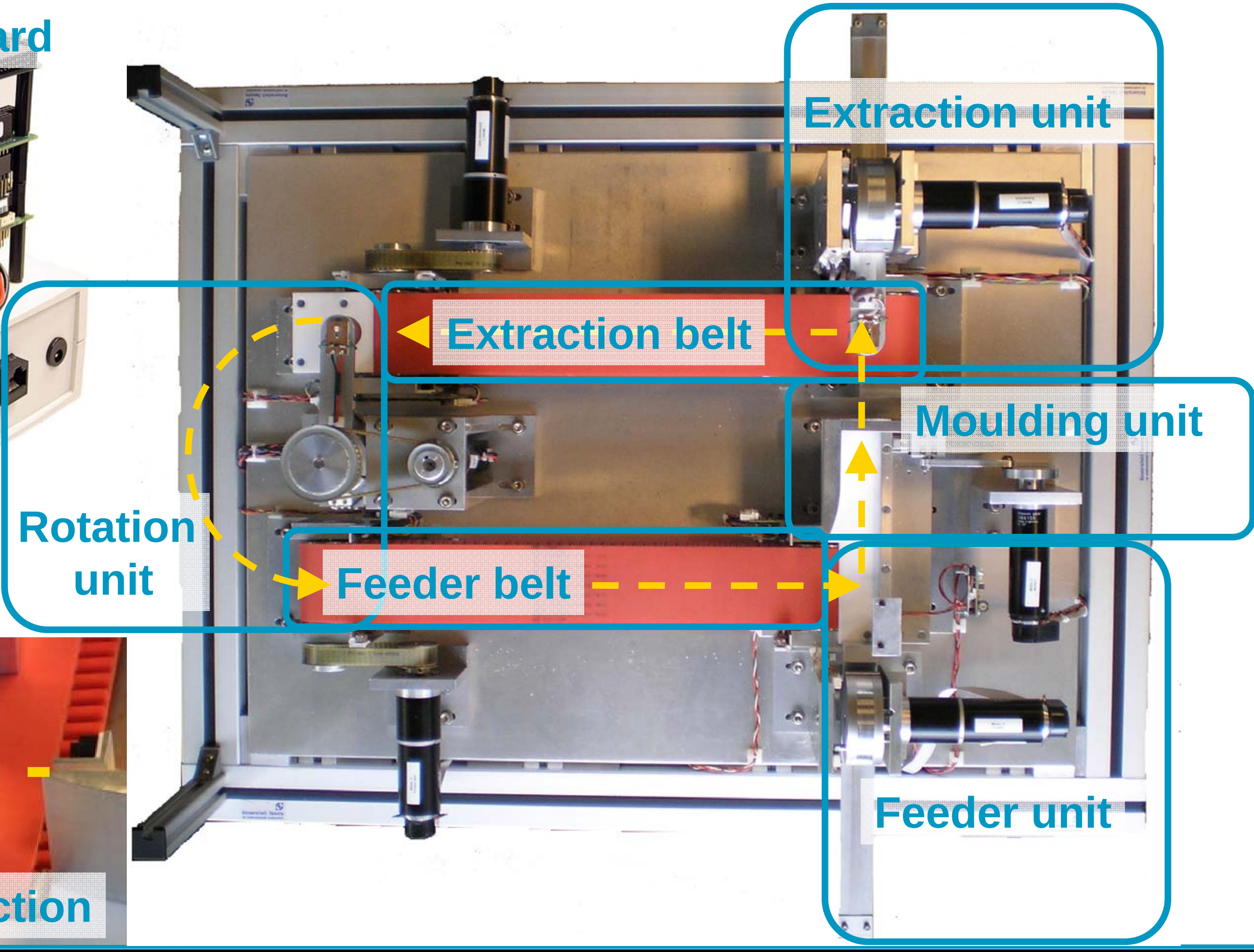
UNIVERSITY OF TWENTE.

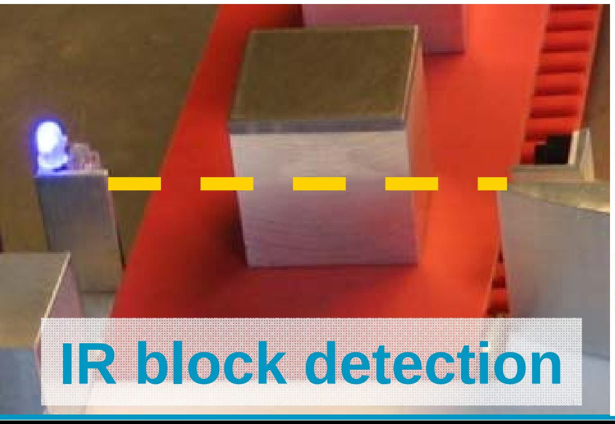
- Production Cell, CPU controlled: gCSP RTAI version



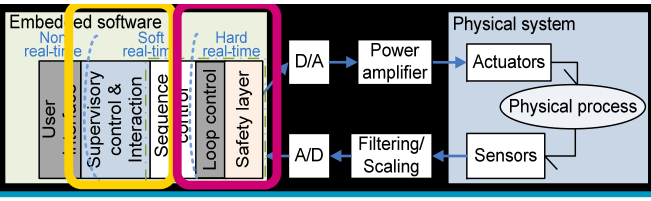
Background Production cell



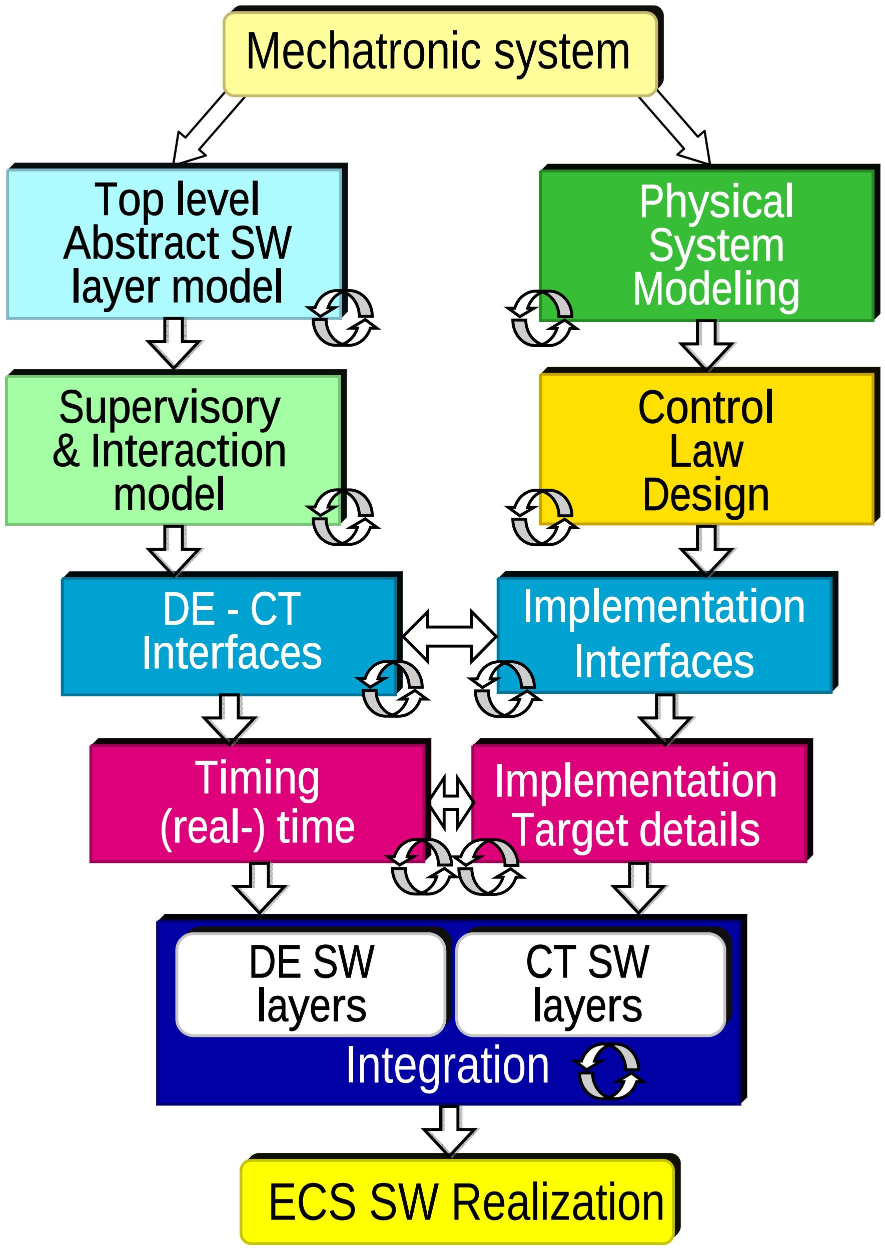




Background Design Method ECS SW



- 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

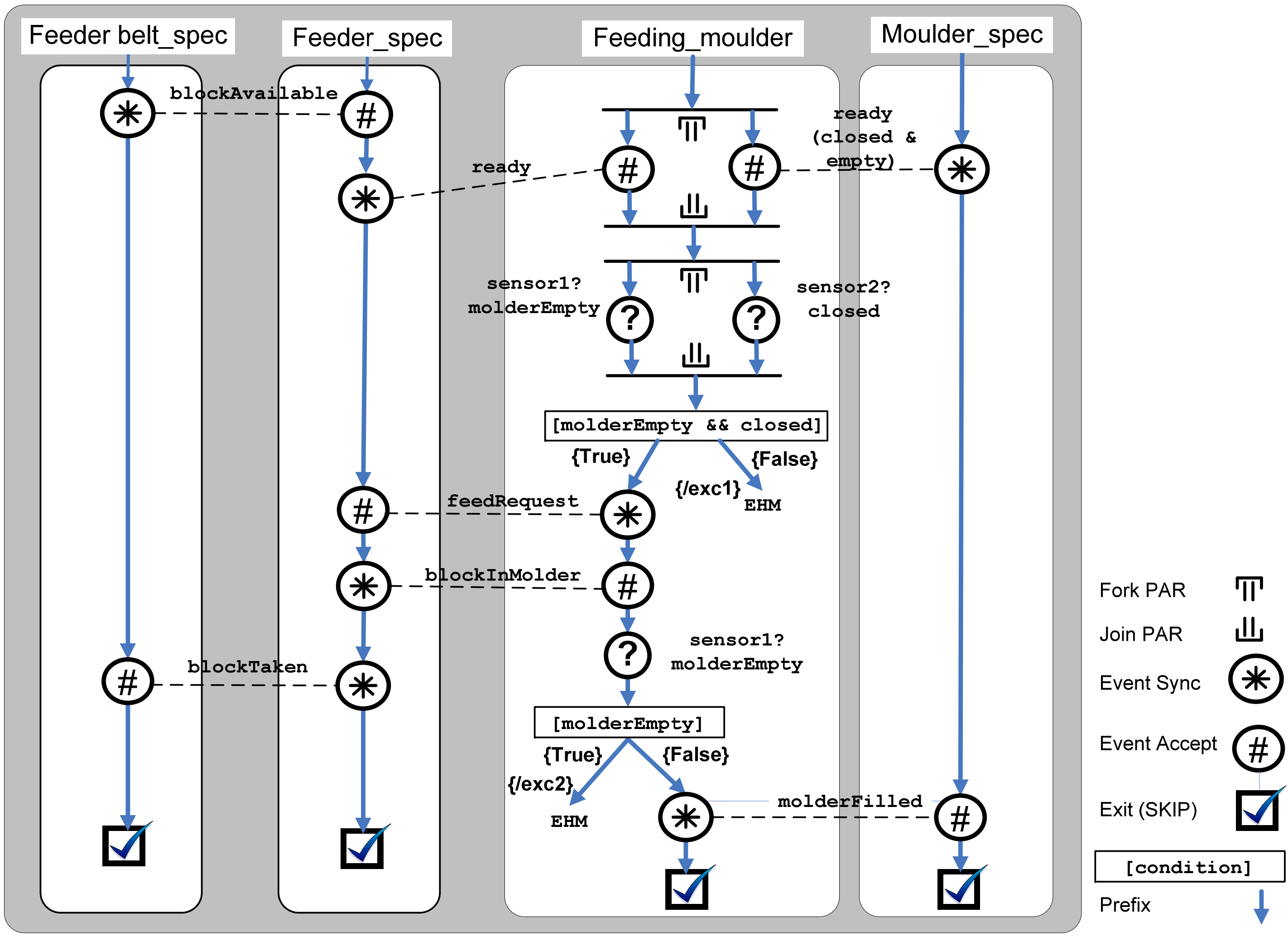


CPU gCSP QNX (D)

- Focus:
  - Test QNX real-time operating system

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

CPU SystemCSP (G)


Feeder belt_spec
Feeder_spec
Feeding_moulder
Moulder_spec
blockAvailable
ready
ready (closed & empty)
sensor1? molderEmpty
sensor2? closed
[molderEmpty && closed]
{True}
{False}
{/exc1}
EHM
feedRequest
blockInMolder
sensor1? molderEmpty
[molderEmpty]
{True}
{False}
{/exc2}
EHM
molderFilled
blockTaken
Fork PAR
Join PAR
Event Sync
Event Accept
Exit (SKIP)
[condition]
Prefix