UNIVERSITY OF TWENTE.

# Embedded Control Software Design with Formal Methods and Engineering models

BCS FACS / FME evening seminar

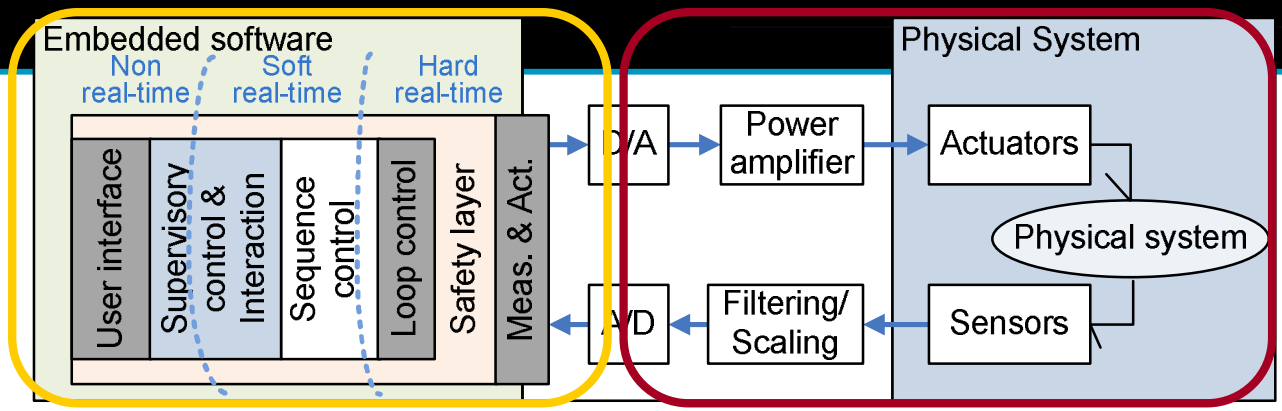
BCS, London, 13-09-2010

**Jan F. Broenink**, Marcel A. Groothuis

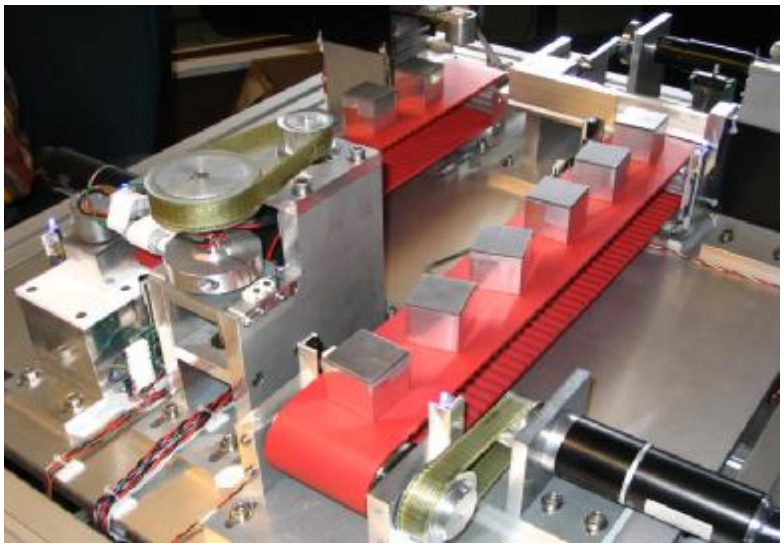
Control Engineering, Department of Electrical Engineering,
University of Twente, The Netherlands

# Contents






- Introduction
  - Goals & Challenges
  - Properties Embedded Control Software
- Method
  - Formalisms: DE / CT
  - Model-driven design, Design Space Exploration
- Test case: production cell
  - Several DE formalisms: CSP, POOSL
  - 2 types of control computers; CPU, FPGA
- Structuring the embedded software
  - Building blocks
  - Separated error handling
- Conclusions & Ongoing Work



- Realization of Embedded Control System (ECS) software
  - For mechatronics & robotic applications
- Design Methodology
  - Model-driven ECS software design
    - Models of Controllers and Robot Behavior
  - Dependable software
    - All code generated!
  - Supporting tool chain
    - Clear work flow -> separation of design steps
- ECS design challenges
  - Heterogeneous nature
  - Large design space
  - Special demands on the software

## Essential Properties Embedded Control Software

- Purpose: control physical systems
- Dynamic behavior of the physical system essential for SW
- Dependability: Safety, Reliability; Real time

## Challenge: Heterogeneity

- Layered structure, building blocks, reuse
- Multiple modeling formalisms / models of computation

## Challenge: Large design space

- Clear design flow, Focus per design step

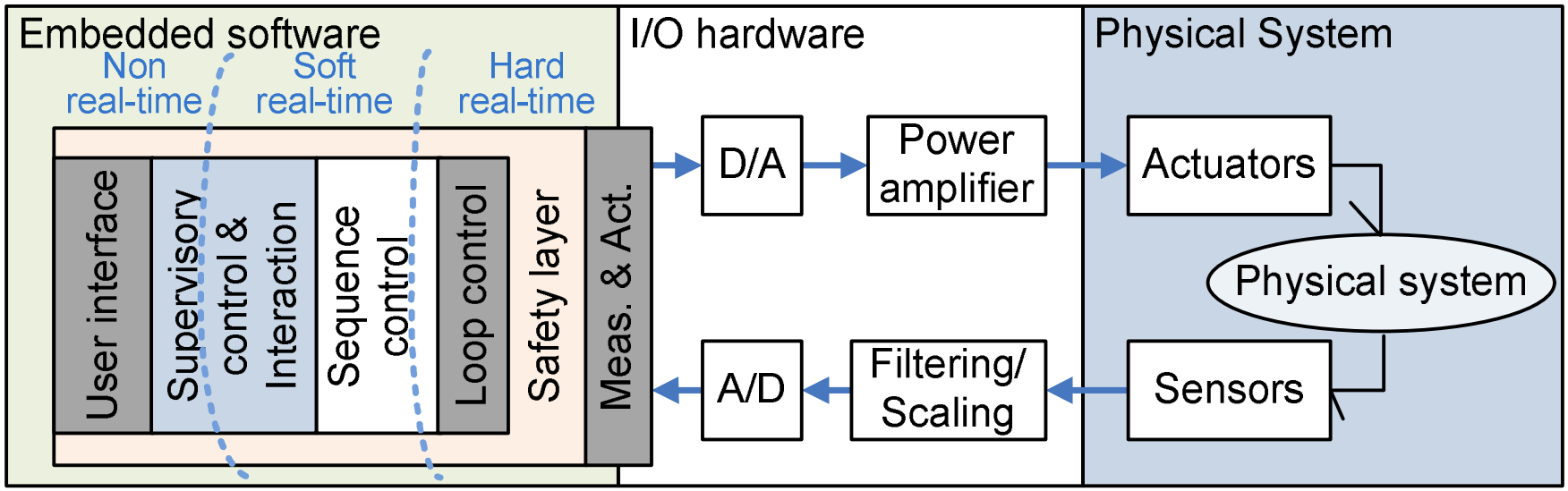
## Challenge: Demands on Software

- Real-time constraints with low latency
- Early-phase testing via (co)-simulation
- Support in method

## 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




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

# Method Starting Points

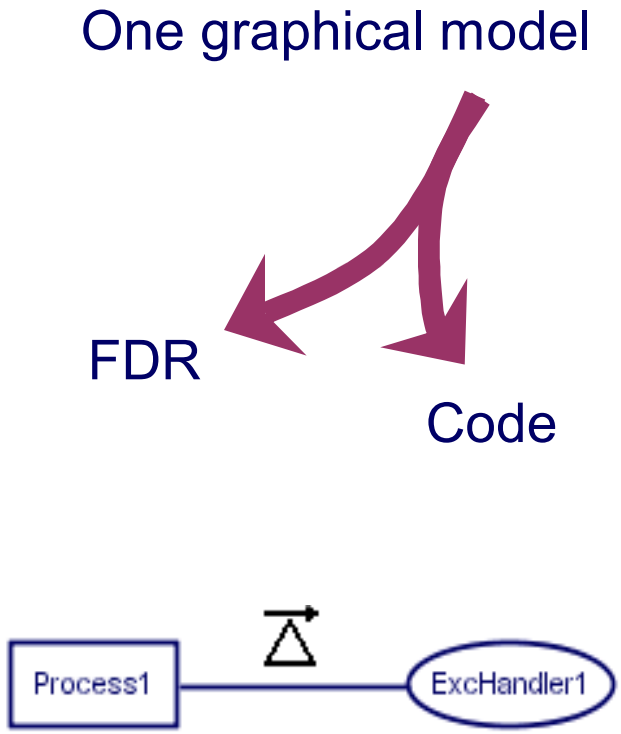


Design Methodology
- Model-driven ECS software design
- Dependable software / supporting tool chain

Use (formal) models for both
- Checking properties (e.g. deadlock)
- Towards code generation

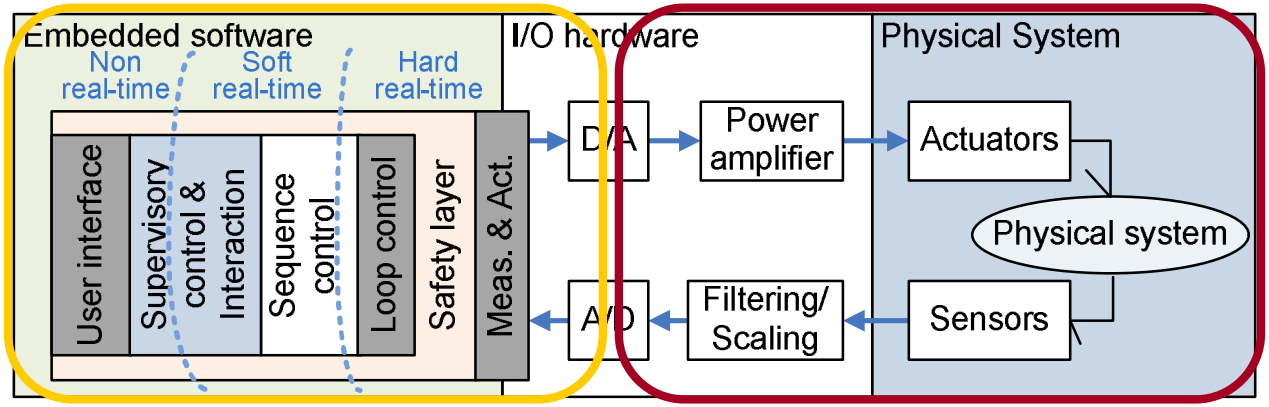
Structuring the Embedded Control Software
- Overview & Support reuse
- Building blocks / design patterns
- Separate normal flow from exception handling

Clear work flow
- Separation of design steps



# Method Formalisms used





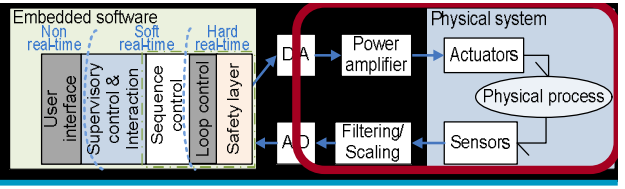

Continuous Time

- Bond graphs
  - to differential equations

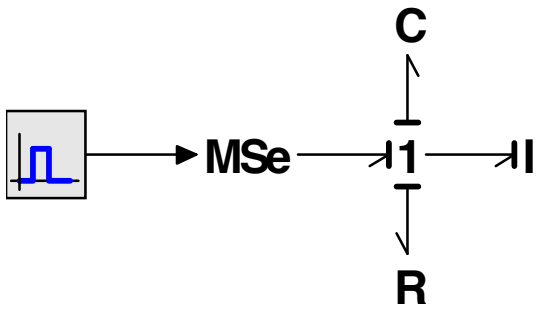
Discrete Event – several possibilities

- gCSP, CSP (communicating sequential processes)
  - to checking
  - to code: CPU / FPGA
- POOSL (parallel object oriented specification language)
  - to interpreted code
- VDM++ (Vienna development method)
  - to checking
  - to code (?!)

# Formalism Continuous-Time Modeling



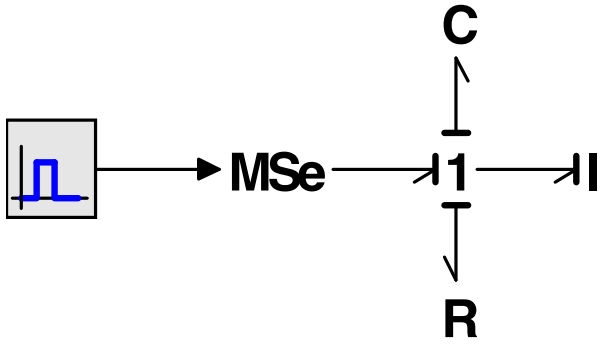

- Essential idea
  - Describe relevant dynamic behaviour
  - Diagrams to show the overview / structure
    - As nature is inherently concurrent
  - Result in ODE: ordinary differential equations
  - Can be simulated to show behaviour: f(t)
- Port-based approach -> Bond Graphs
  - Directed graph: submodels & ideal connections
  - Domain-independent
    - Analogies between physical domains
- Encapsulation of contents
  - Interface: ports with 2 variables
    - (u, i): voltage & current; (F, v): force & velocity;
  - Equations as equalities (math. Equations)
    - Not as algorithm: u = i * R -> u := i * R of i := u / R



# Formalism CT with Bond Graphs

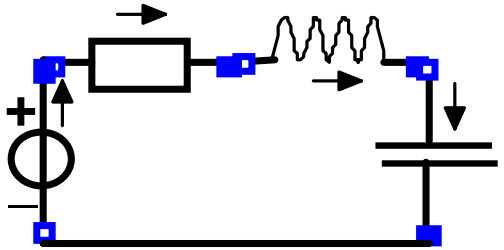
Bond graphs: labeled and directed graphs

- vertices: submodels
  - idealized descriptions, concepts
- edges: ideal energy connection
  - called bonds, bilateral signal flow

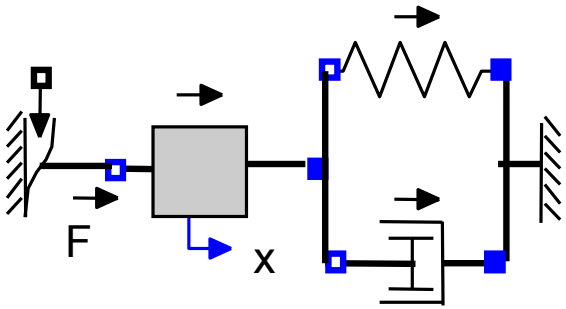



Analogies

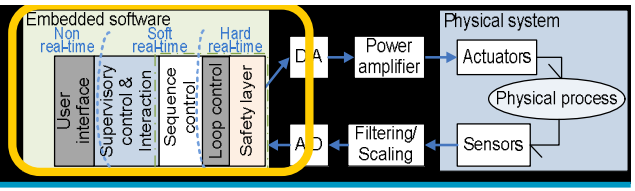
| | | |
|---|---|---|
| Capacitor | Spring | C |
| Coil | Mass | I |
| Resistor | Friction | R |
| Voltage source | Force source | Se |



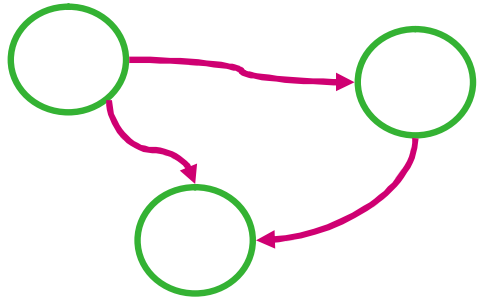
| | | | | |
|---|---|---|---|---|
| Voltage | u | Force | F | e – effort |
| Current | i | Velocity | v | f – flow |

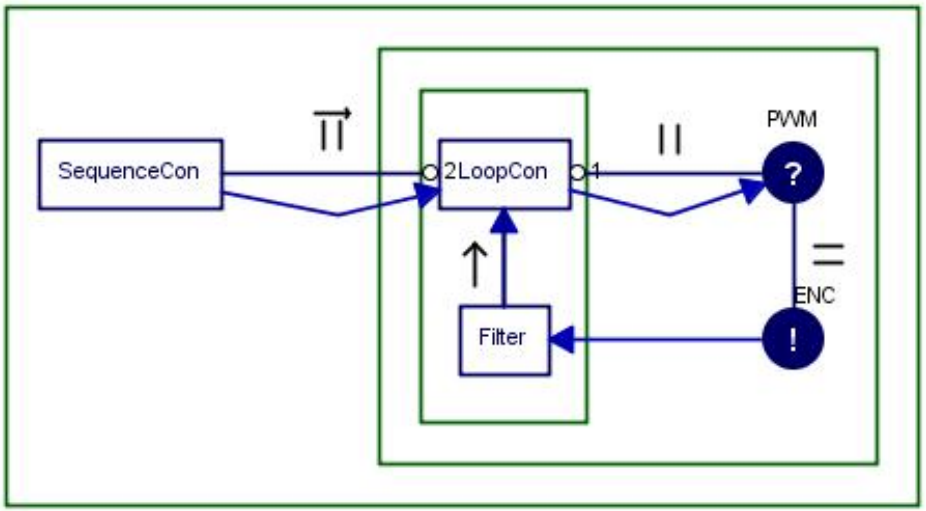


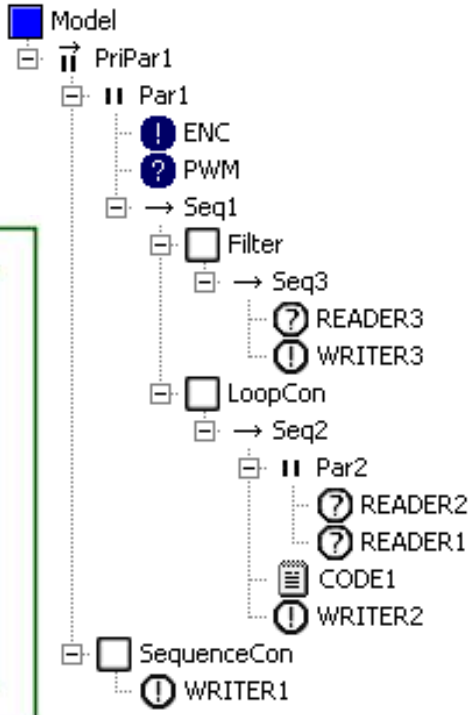
# Formalism Discrete-Event Modeling




- Dataflow diagrams based on CSP process algebra
  - Modeling concurrent behavior
  - Communicating Sequential Processes (Hoare)
  - Synchronous behavior (rendezvous communication)
  - Formally verify-able: model checker FDR2
- Processes & Events
  - Process alphabet: set of events
  - Communication: channels
  - Scheduling at rendezvous: in application
- Process operators
  - PAR, ALT, SEQ
  - PRI-PAR
- Example
  - gCSP



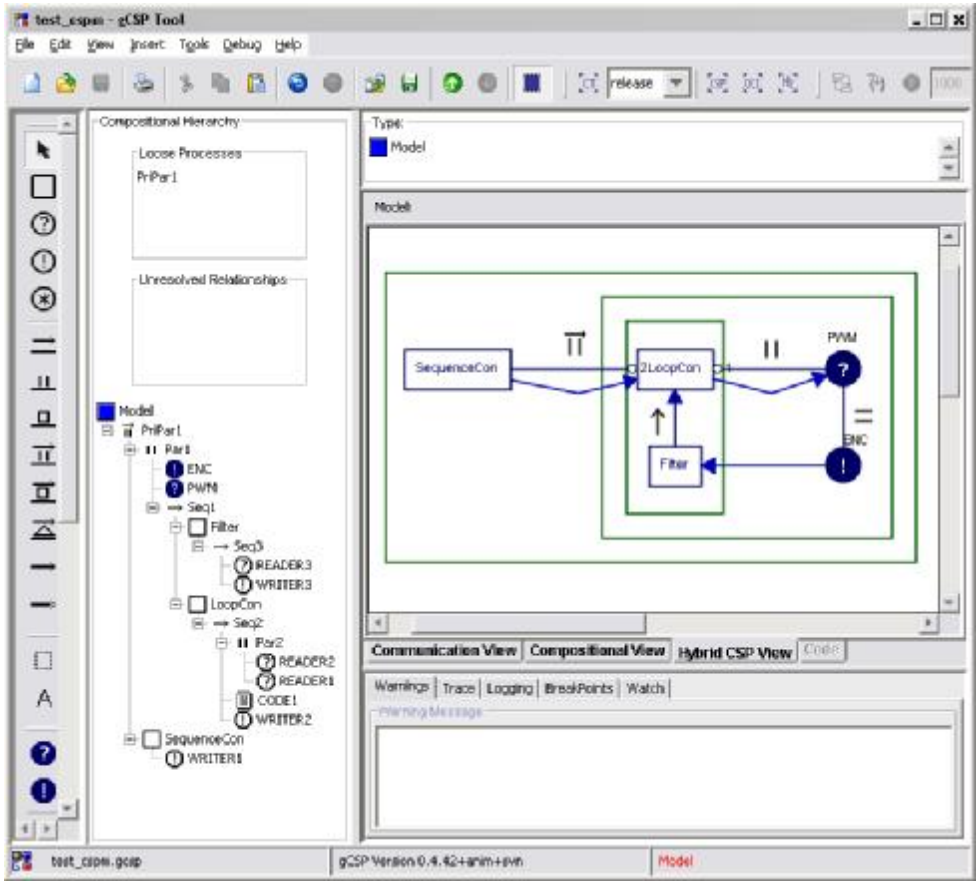


# Formalism DE with gCSP



- gCSP
  - Graphical CSP design tool
  - Animation
  - Code generation:
    - C++, HandelC, CSPm, Occam




**CSPm:**

```
PriPar1 = Par1 [| {| setState |} |] SequenceCon
Par1 = ENC [| {| encread |} |] (PWM [| {| pwmWrite |} |] Seq1)
Seq1 = Filter ; LoopCon
SequenceCon = setState -> SKIP
LoopCon = Par2 ; CODE1 ; pwmWrite -> SKIP
Par2 = filt2loop -> SKIP ||| setState -> SKIP
CODE1 = SKIP
Filter = encread -> SKIP ; filt2loop -> SKIP
PWM = pwmWrite -> SKIP
ENC = encread -> SKIP
```

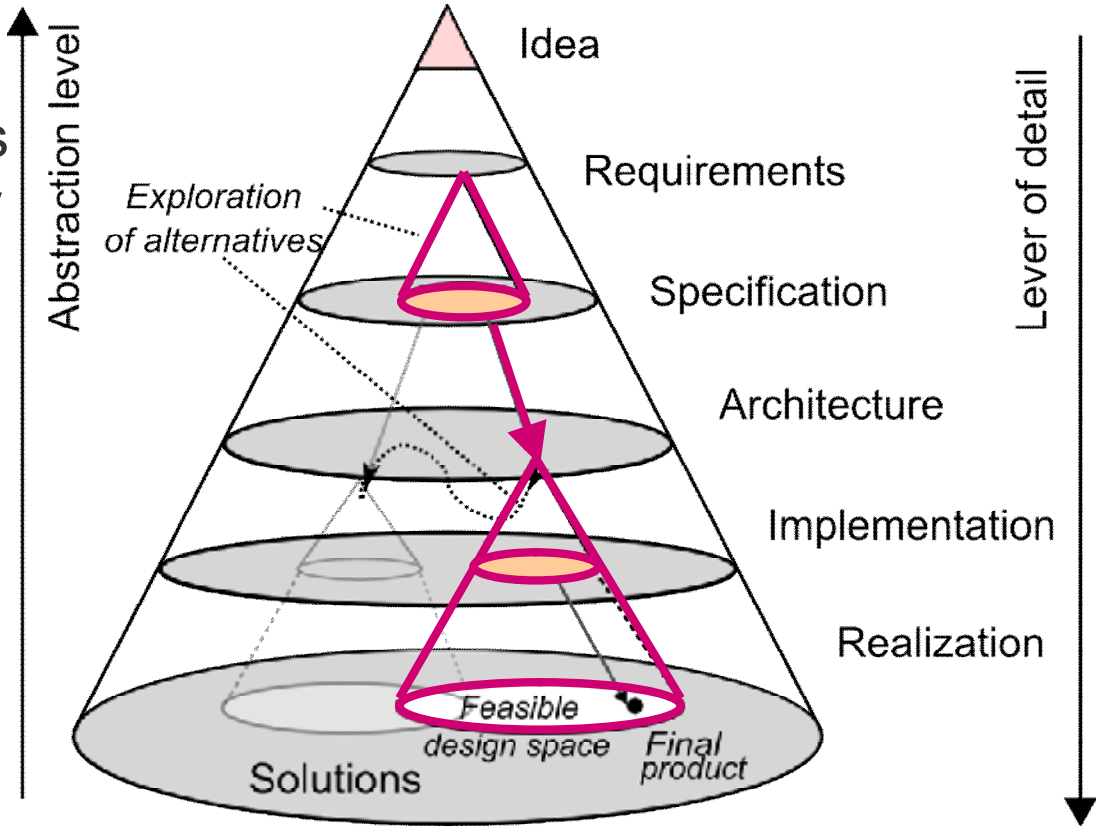
**Handel-C:**

```
void main( void ) {
    chan  filt2loop; chan  pwmWrite; chan  encread;
    chan  setState;
    par {
        par {
            ENC( &encread );
            PWM( &pwmWrite );
            seq {
                Filter( &filt2loop, &encread );
                LoopCon( &filt2loop, &pwmWrite, &setState );
            }
        }
        SequenceCon( &setState );
    }
}
```

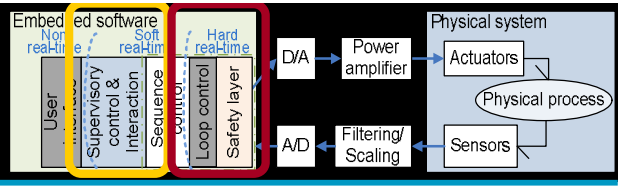
# Method Overview Model-Driven Design



- Way of Working
  - Abstraction
    - Hierarchy
      - Split into Subsystems
      - Cope with complexity
  - Model-driven design
    - Design Space Exploration
      - Aspect models
      - Make choices
      - Limit solution space
    - Step-wise refinement
      - Add detail
      - Lower abstraction
    - Implementation
    - Realization
  - Concurrent design trajectory
  - Early Integration where possible



# Method Design Method ECS SW

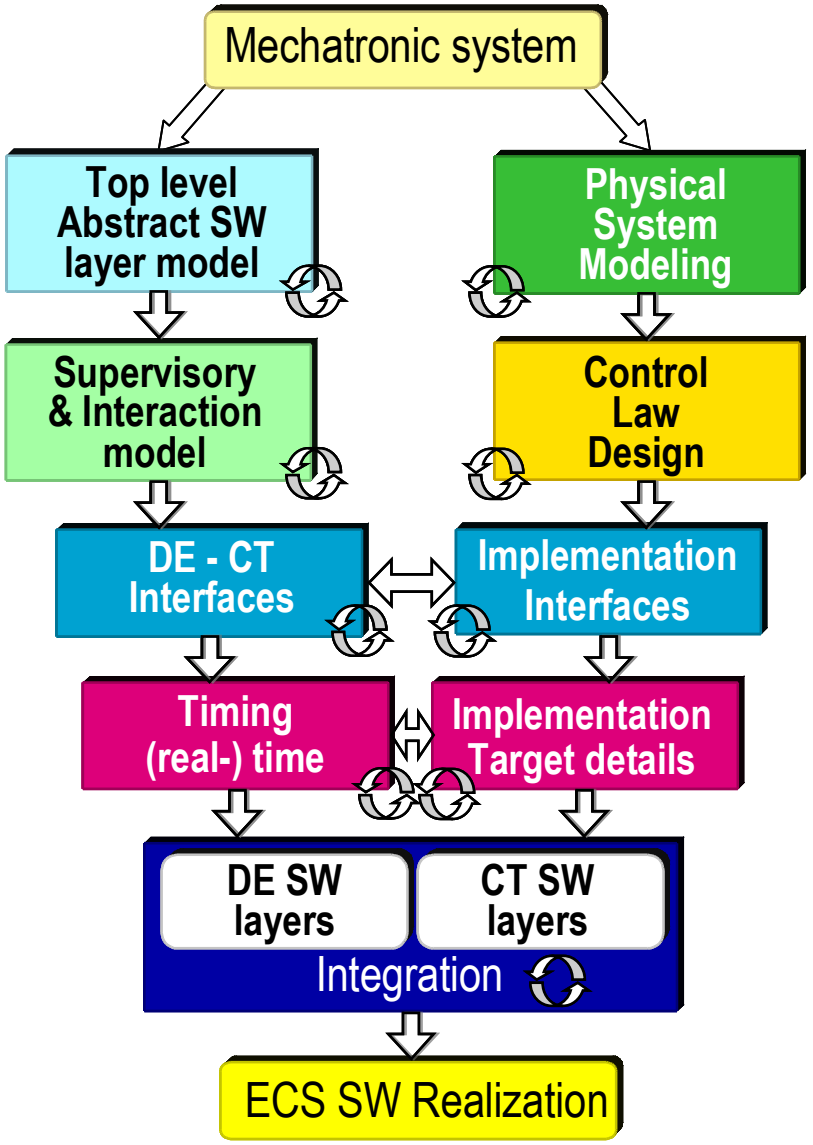



## Approach

- 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



# Method MoC & Method Steps

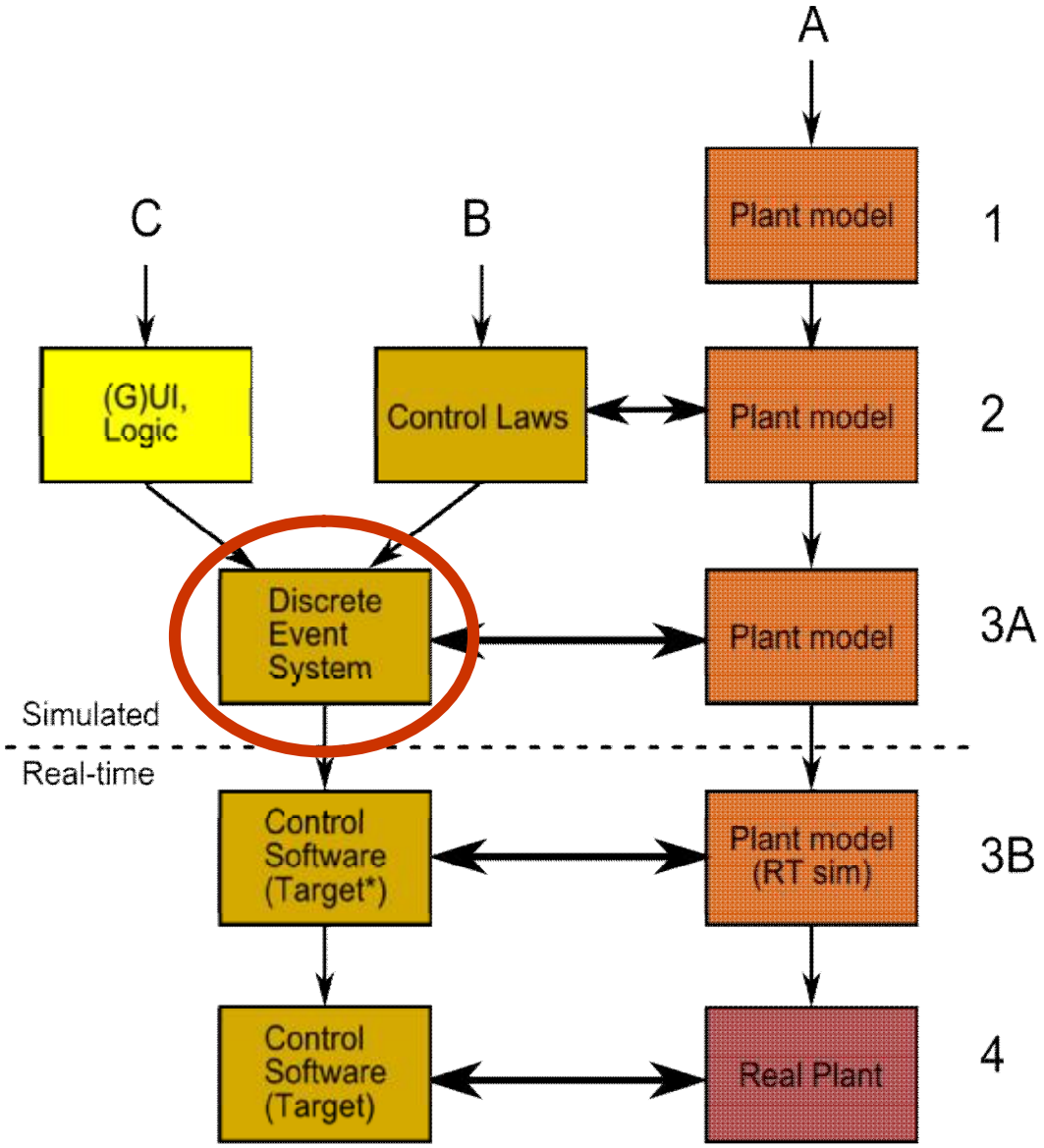


## Models of Computation

- A: Plant (bond graphs)
  -> simulation
- B: Loop Control Laws
  -> into code on target
- C: Supervisory / Interaction
  -> code on target

## Co-simulation

- Combines models
  Discrete Event
  & Continuous Time
- Stepwise refinement
- ‘Better’ testing
- Concurrent engineering



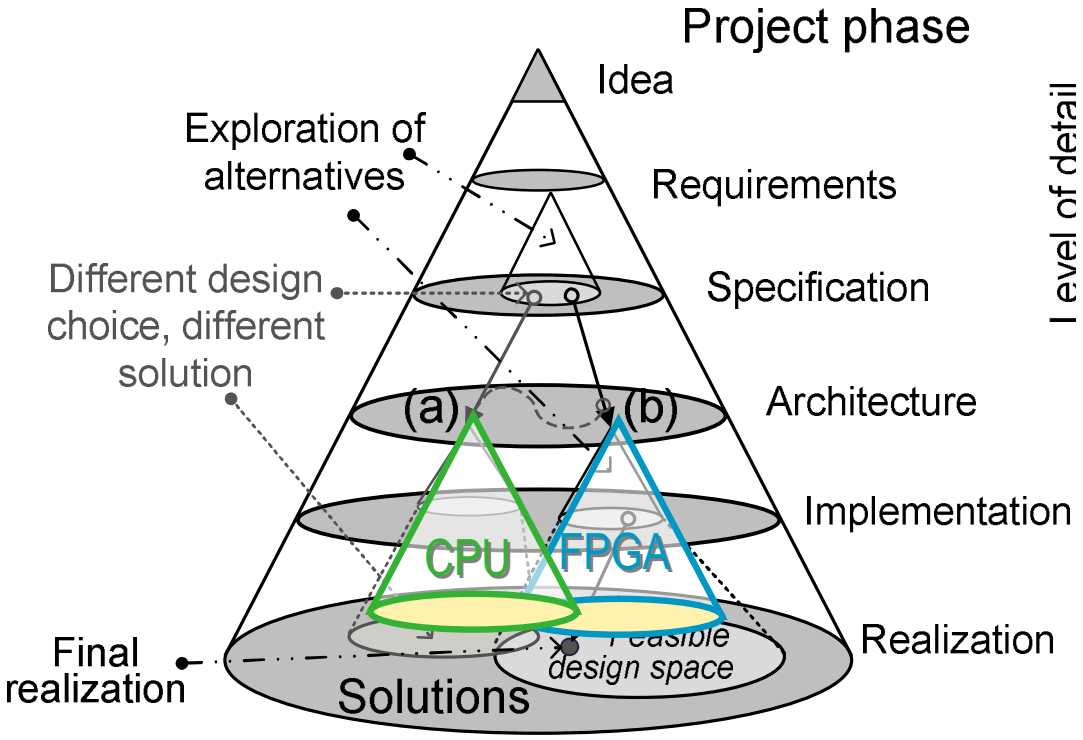
# Method MDD example, also DSE



- 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
  - Architecture
    - CPU FPGA, distributed central
- Reachable solutions
  - Dependent on all choice

Abstraction level

resource usage




Design Space Exploration (DSE)

# Contents

# Test Case Production Cell

Production cell demonstrator

Based on:

Stork Plastics Moulding machine

Architecture:

CPU (ECS / FPGA programmer)

FPGA (digital I/O / ECS)

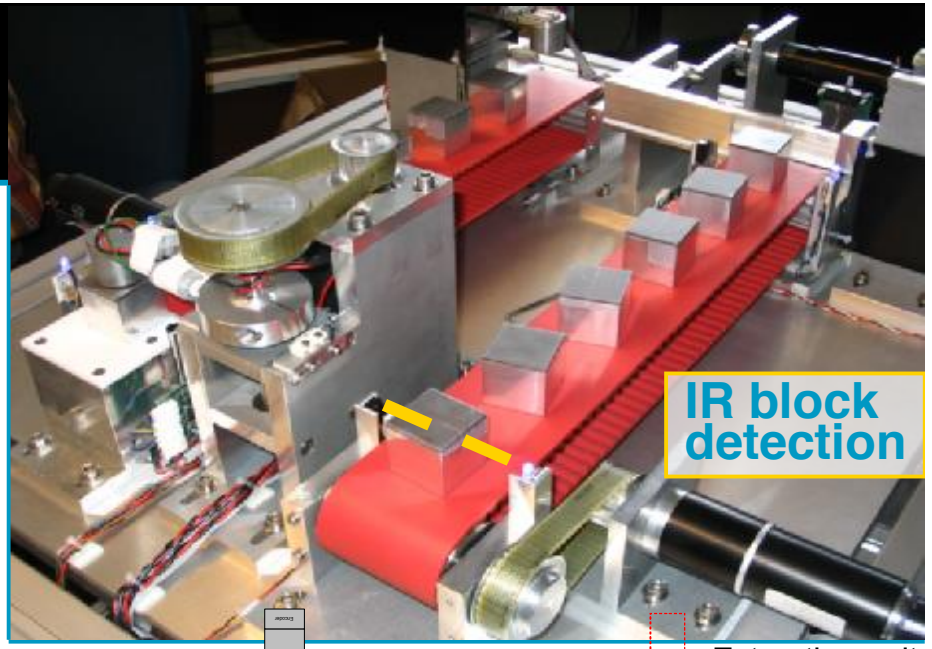
6 Production Cell units

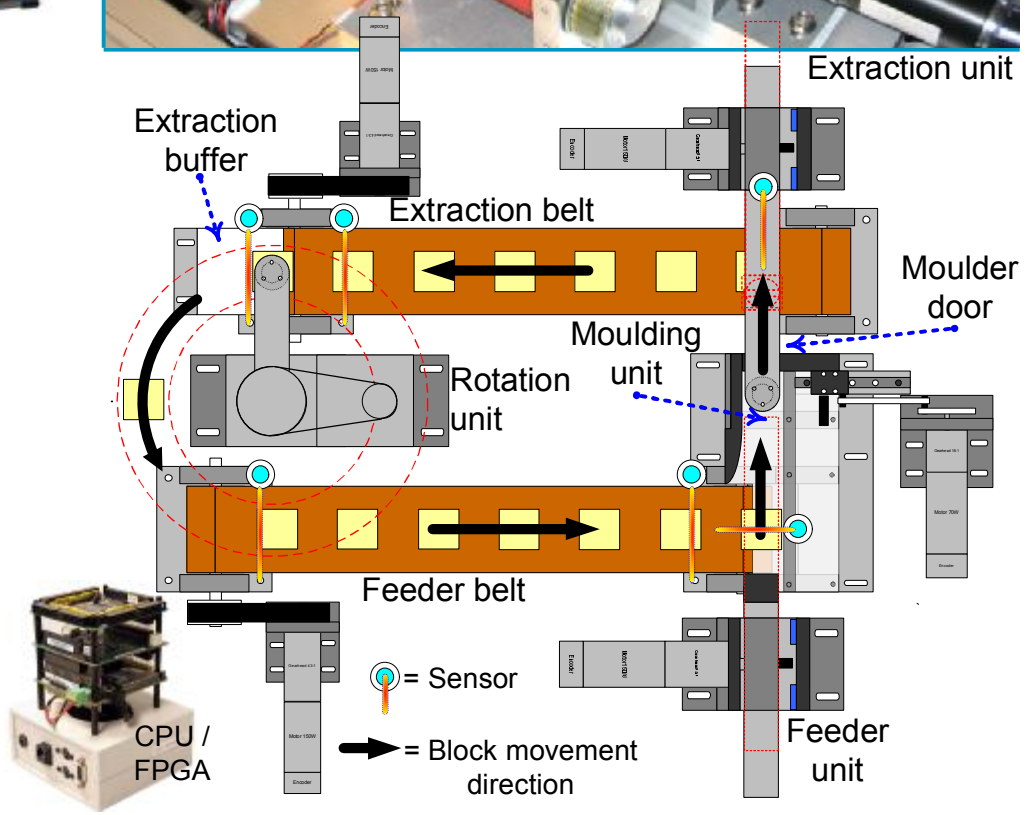
Action in the production process

Moulding, Extraction, Transportation, Storage

Synchronize with neighbours

Deadlock possible on > 7 blocks



# Production Cell ECS implementations

UNIVERSITY OF TWENTE.

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 | Integer | FPGA | Yes |
| F | gCSP Handel-C float | Floating point | FPGA | Yes |
| G | SystemCSP | - | - | No |
| H | VDM++ | | | Idea |

Different choices

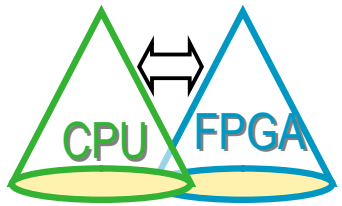
**OS:**
- RTAI Linux
- QNX
- No OS

And many more

**Formalisms:**
- CSP
- CCS
- Multi MoC

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

**Architecture:**




Seq → ⇔ Par | |

# CPU gCSP RTAI (A)

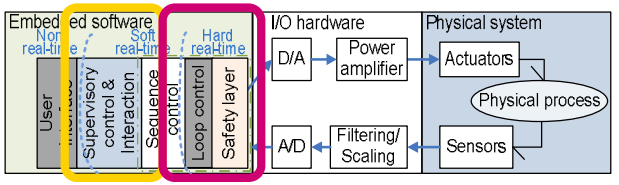


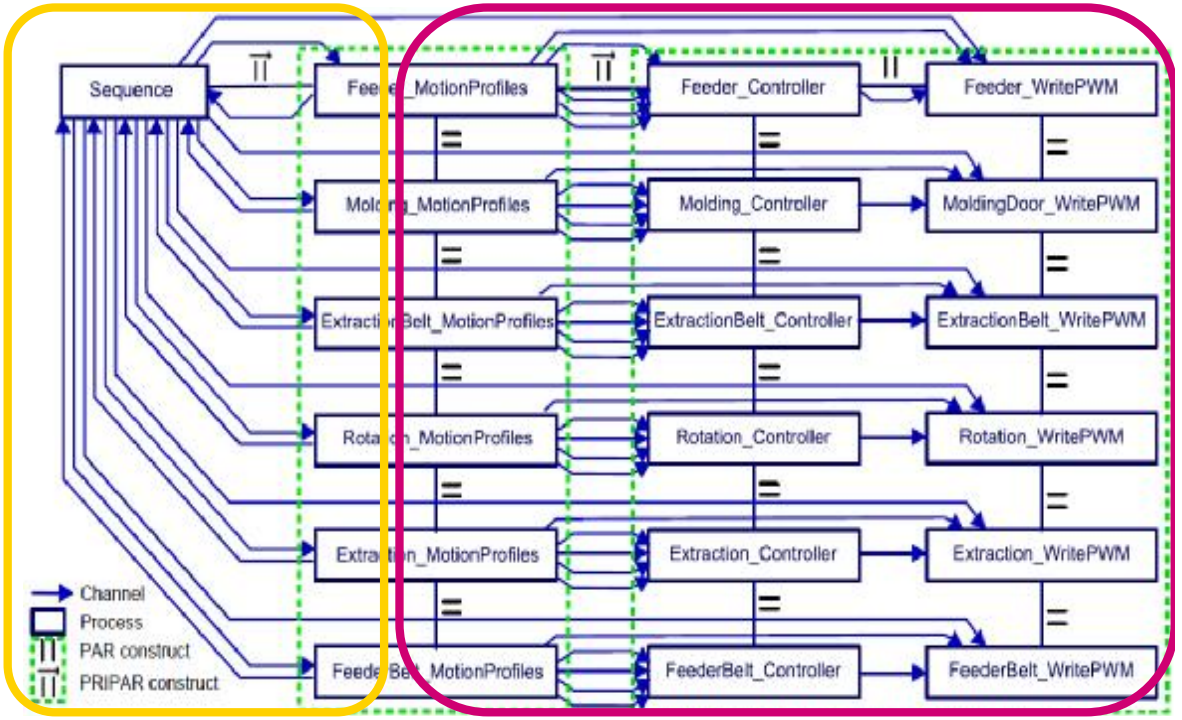
Focus: proof of concept gCSP

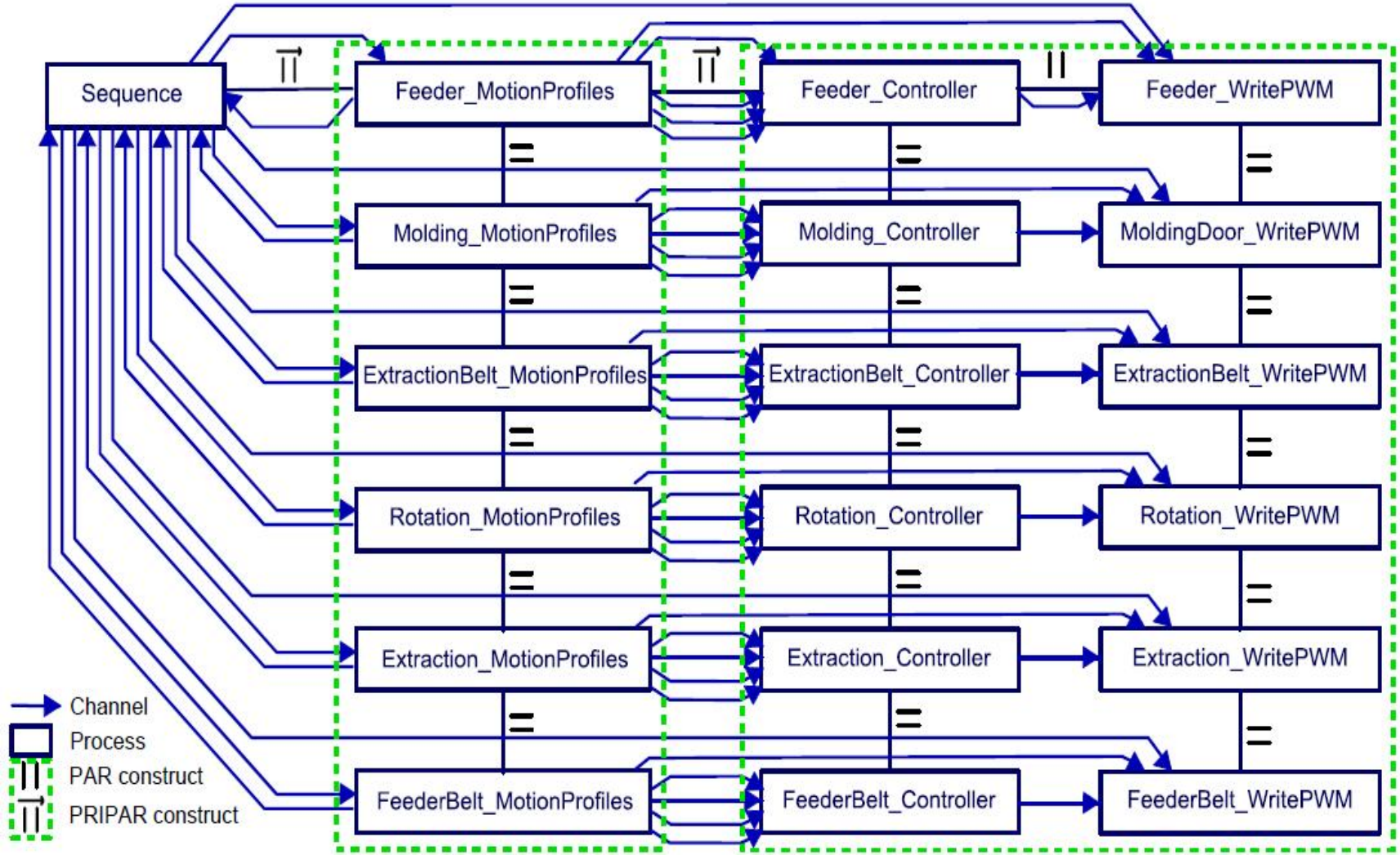
- Proof of concept gCSP for Embedded Control Systems software
- Combination of untimed CSP and real-time Linux

Realization

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




Sequence
Feeder_MotionProfiles
Feeder_Controller
Feeder_WritePWM
Molding_MotionProfiles
Molding_Controller
MoldingDoor_WritePWM
ExtractionBelt_MotionProfiles
ExtractionBelt_Controller
ExtractionBelt_WritePWM
Rotation_MotionProfiles
Rotation_Controller
Rotation_WritePWM
Extraction_MotionProfiles
Extraction_Controller
Extraction_WritePWM
FeederBelt_MotionProfiles
FeederBelt_Controller
FeederBelt_WritePWM
Channel
Process
PAR construct
PRIPAR construct

# CPU gCSP RTAI (A)



Results

- gCSP and CSP are usable for ECS software
  - Graphical process & channel structure
  - Debugging CSP processes difficult (textual) → gCSP animation
- 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 CTC++ library
- Modeling
  - Diagram structure, Interaction, Hierarchy

POOSL = Parallel Object Oriented Specification Language TU/e

- CCS process algebra + Timing extension
- Modeling high level behaviour Embedded Systems

## Focus

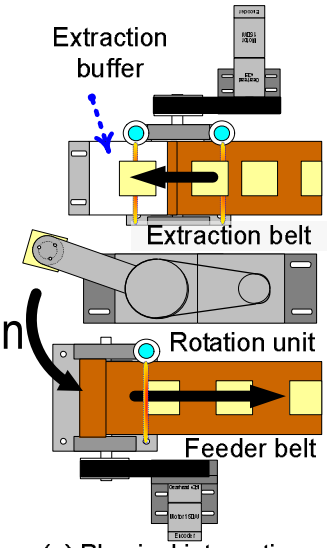
- Test timing
- Integration DE & CT
- Structured modeling
  - concurrency & Interaction
  - DE CT interfacing
  - Timing

## Realization

- Top-down
- No formal check

## Results

- Separated concurrent design SW layers
  - DE (high level, CT (low level)



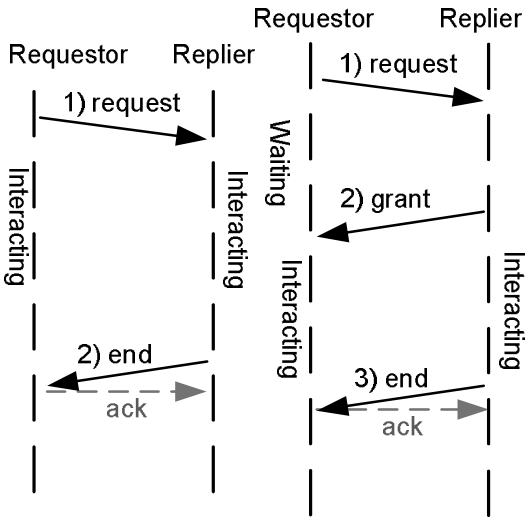

(a) Physical interactions

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

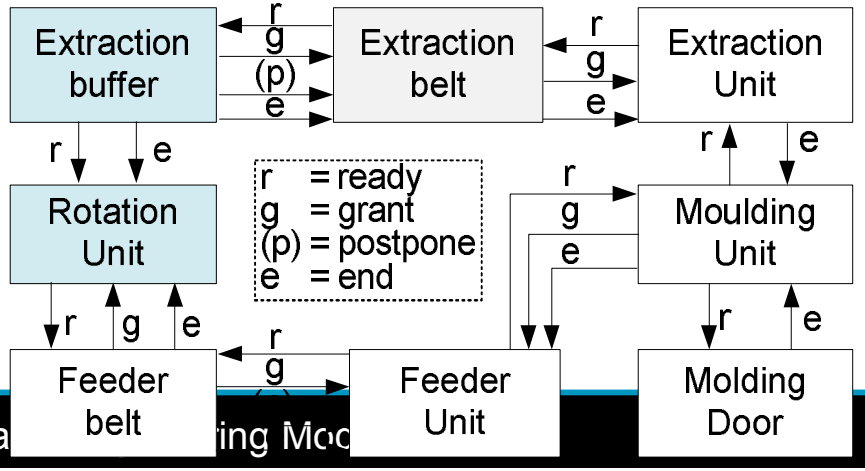
(b) Synchronization with Extraction buffer

```
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()().
```

(c) Synchronization with feeder belt



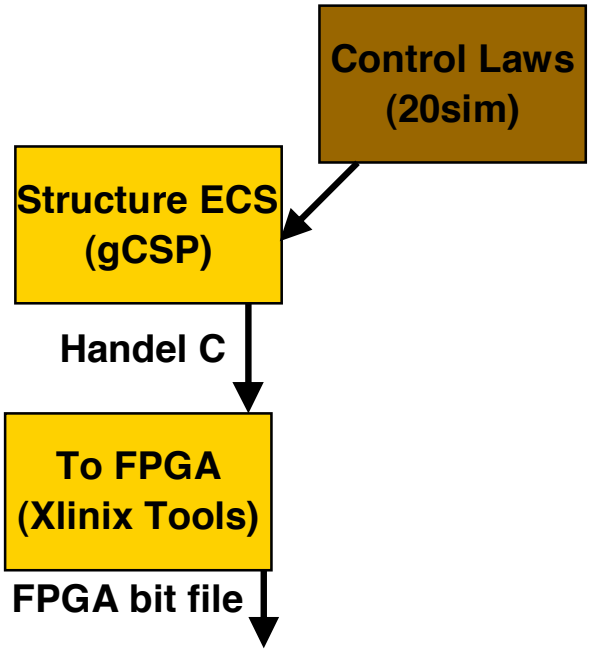

(d) Two-way handshake

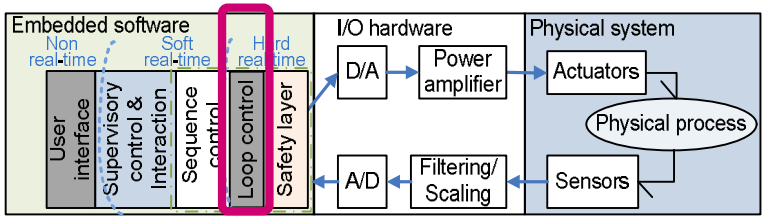


# FPGA gCSP Handel-C

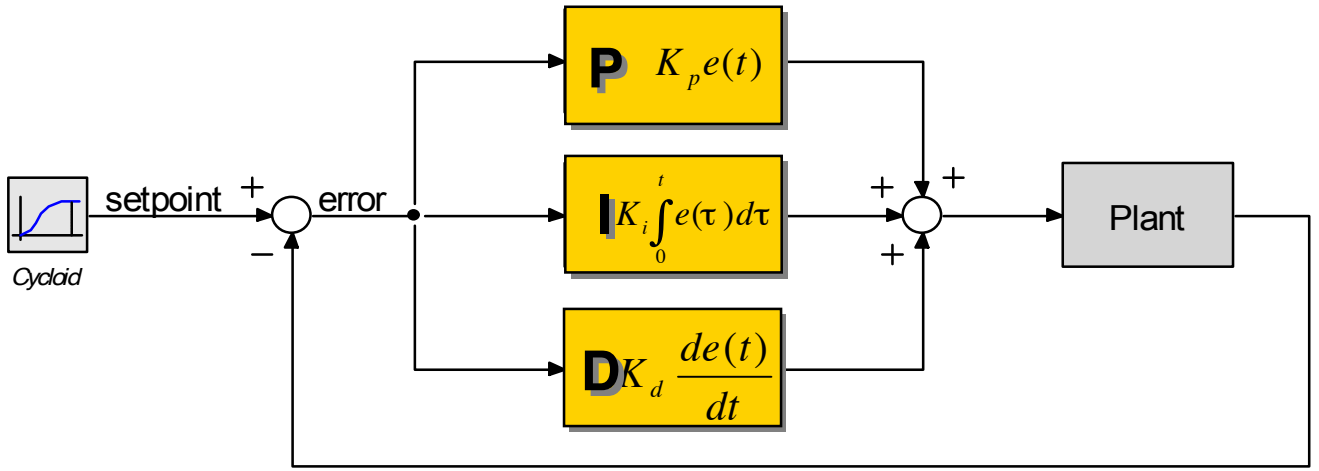


- Feasibility study on motion control in FPGA
  - Exploit parallelism
  - Accurate timing
  - Model-driven design
- Choice
  - Modeling tools
    - gCSP + 20-sim
    - output: floating point control algorithm
  - Implementation
    - Handel-C programming language / hardware description language
    - No (soft core) CPU
    - Small size Xilinx Spartan III FPGA
  - Challenges
    - Design Space Exploration
      - Numerical precision versus logic cell utilization (FPGA)
    - Embedded Software structuring
      - Object orientation, processes, channels FPGA: ????






- PID loop-controller algorithm
  - Proportional, Integral & Derivative terms
  - Floating point
  - Feedback loop: error minimization
    - Error fluctuates around 0; calculation accuracy needed




```
factor = 1 / ( sampletime + tauD * beta );
uD = factor * ( tauD * previous(uD) * beta + tauD * kp * ( error - previous(error)) + sampletime * kp * error );
uI = previous( uI ) +  sampletime * uD / tauI;
output = uI + uD;
```

# Loop Controller Floating point alternatives



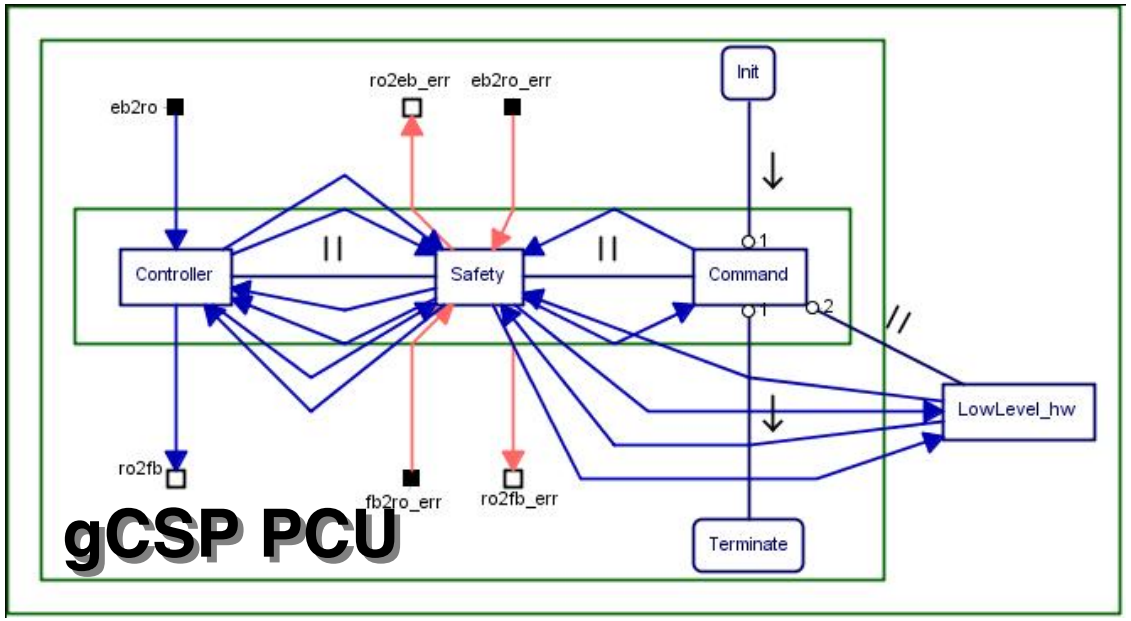
| Alternative | Benefit | Drawback |
|---|---|---|
| Floating point library, CPA 2009 | High precision; re-use existing controller | Very high logic utilization |
| Fixed point library | Acceptable precision | High logic utilization |
| External FPU | High precision; re-use existing controller | Only high-end FPGA; expensive |
| Soft-core C | | ation unless sign challenges |
| 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**

# Realization Handel-C

```
void Rotation(chan* eb2ro_err, chan* ro2eb_err, chan* fb2ro_err, chan* ro2fb_err, chan* eb2ro, chan* ro2fb)
{
    /* Declarations */
    chan int cnt0_w encoder_in;
    chan int 12 pwm_out;
    chan int 2 endsw_in;
    chan int 1 magnet_out;
    chan state_w setState;
    chan state_w currentState;
    chan state_w saf2ctrl;
    chan state_w override;
    chan int 12 ctrl2hw;
    chan state_w ctrl2saf;
    chan cnt0_w hw2ctrl;
    chan int 1 magnet_saf;

    /* Process Body */
    par {
        LowLevel_hw(&encoder_in, &pwm_out, &endsw_in, &magnet_out);
        seq {
            Init(&encoder_in, &magnet_out, &pwm_out);
            par {
                Command(&setState, &currentState);
                Safety(&eb2ro_err, &saf2ctrl, &ro2eb_err, &override, &encoder_in, &fb2ro_err, &pwm_out,
                        &setState, &ro2fb_err, &ctrl2hw, &currentState, &ctrl2saf, &hw2ctrl);
                Controller(&saf2ctrl, &override, &eb2ro, &ctrl2hw, &ctrl2saf, &ro2fb, &hw2ctrl, &magnet_saf);
            }
            Terminate(&encoder_in, &magnet_out, &pwm_out);
        }
    }
}
```



# Results FPGA Usage (integer)



Real parallelism

- 6 Production Cell Units run parallel

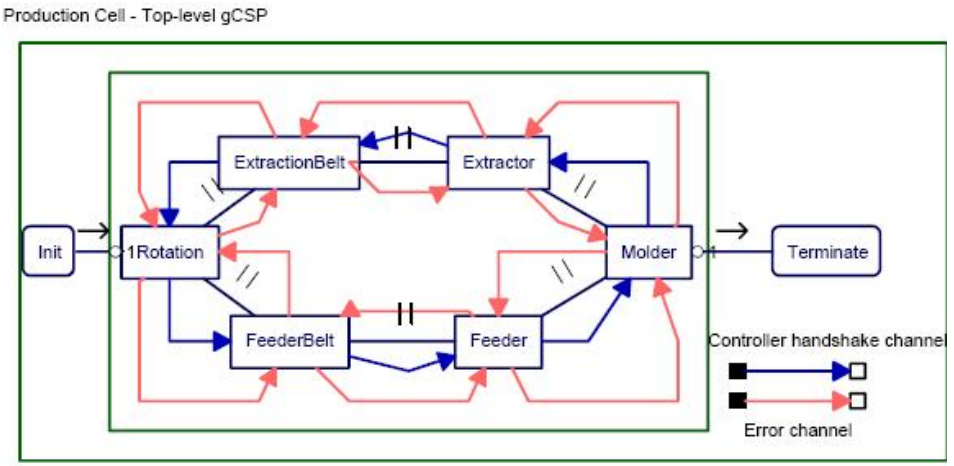
Integer algorithm (no floating point)

- 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 I I @ 1 ms with idle time 99,95%**

# FPGA Trade-offs floating point

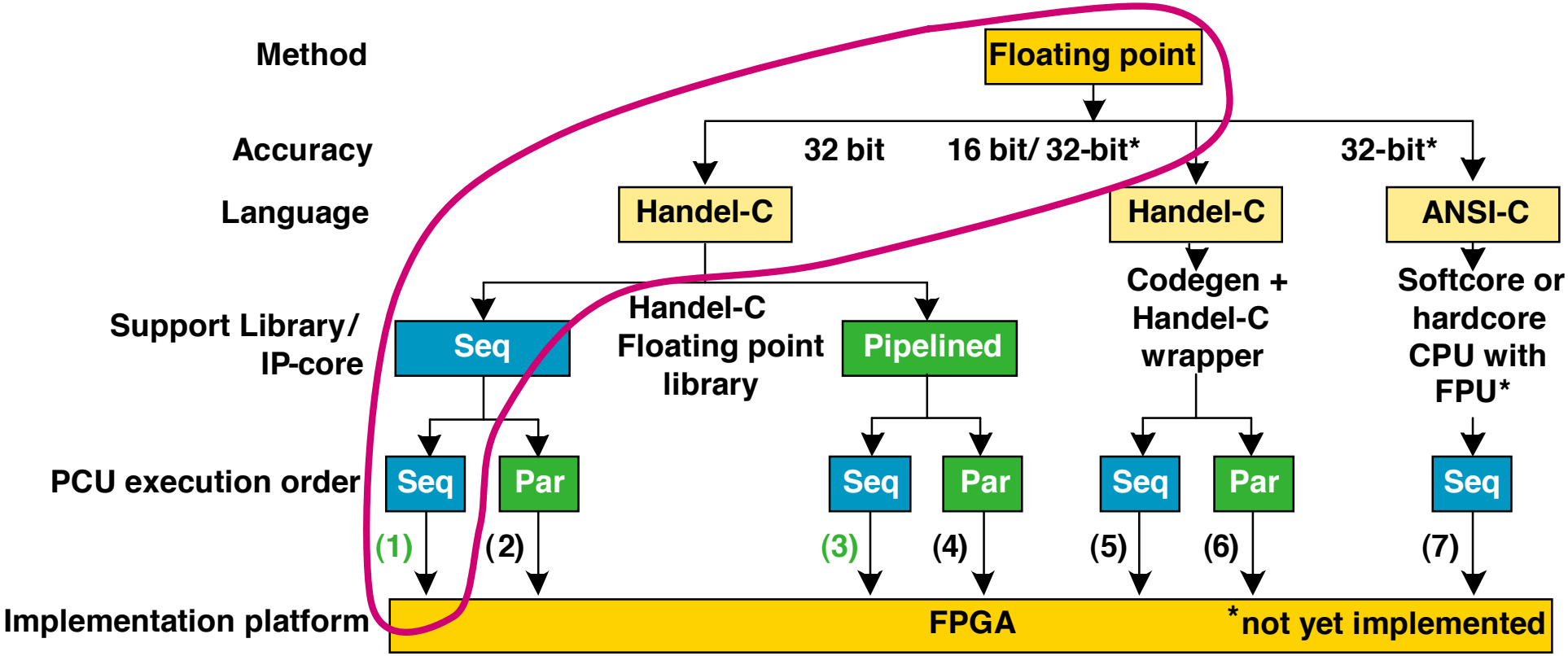


Sequential Pipelined Handel-C floating point library

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)



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

# Contents

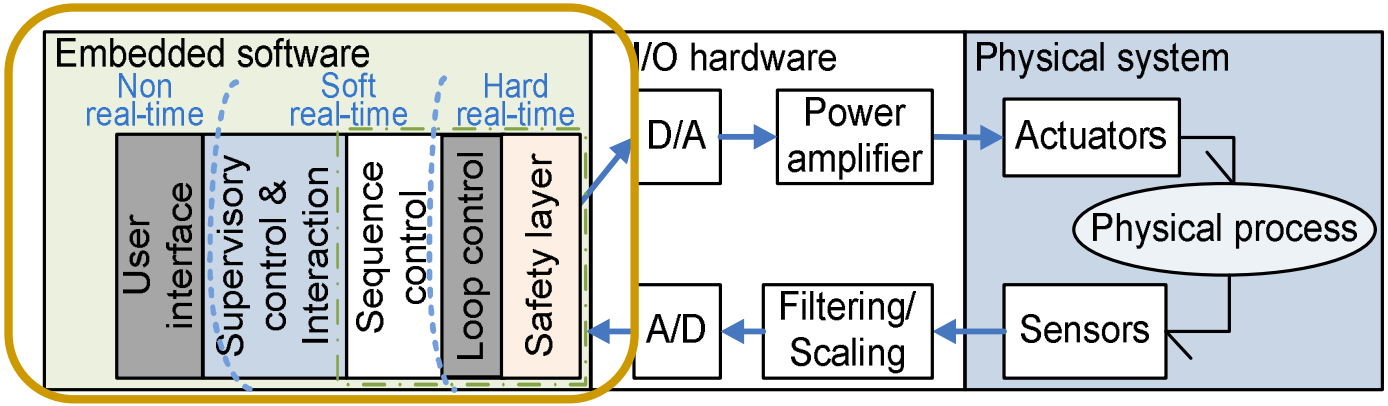
# Structuring ECS

## Structuring the Embedded Control Software

- Overview for the designer
- Re-usable framework for SW and HW based designs
- Building blocks / design patterns
- Separate normal flow from fault handling

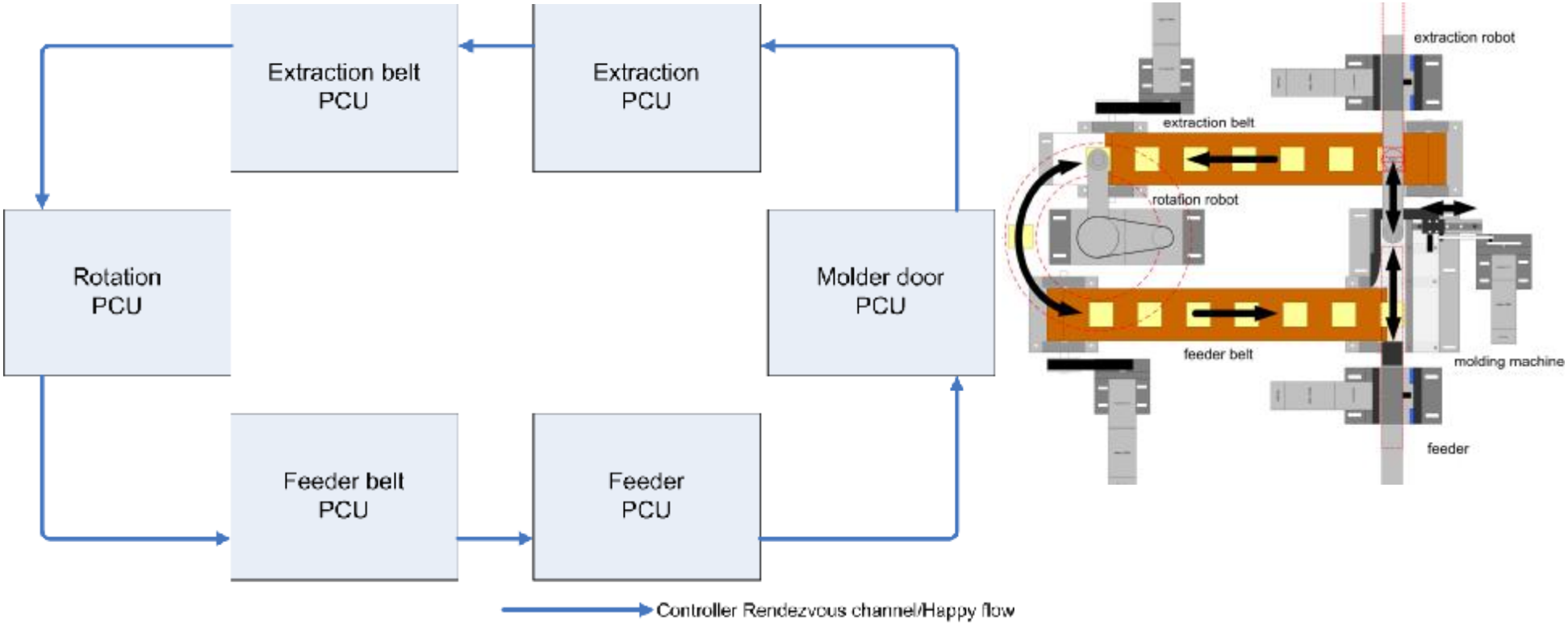
## Process-oriented approach

- Inspired by CSP, block diagrams, bond graphs
- Describe concurrent behaviour
- Structuring in layers is supported
- Use benefits of CSP (checking)



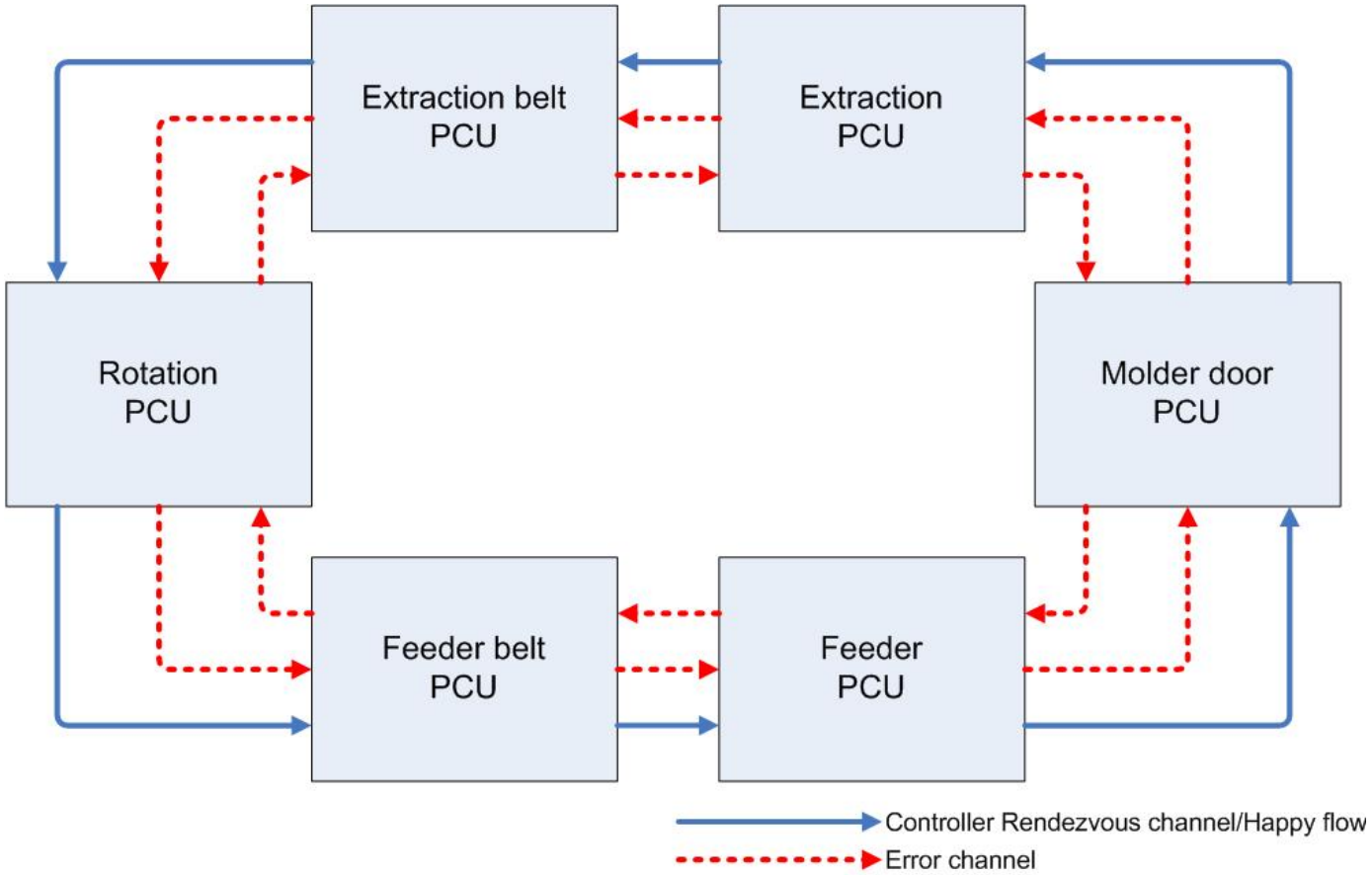
6 independent Production Cell Units

- Interaction: handshake for block delivery
- No central supervisor



## Add Error communication

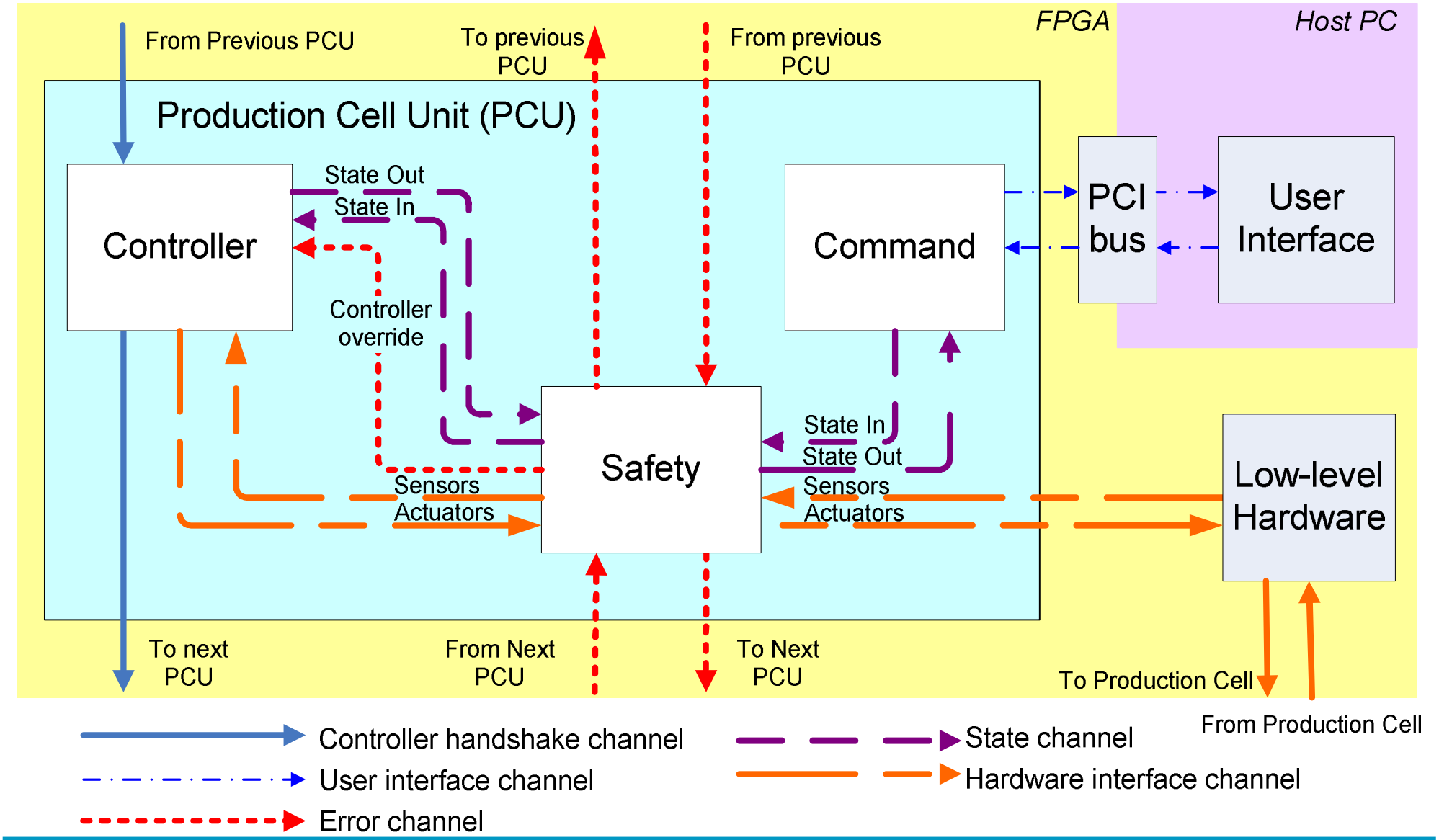
### Inform neighbours when needed




PCU: Production cell Control Unit

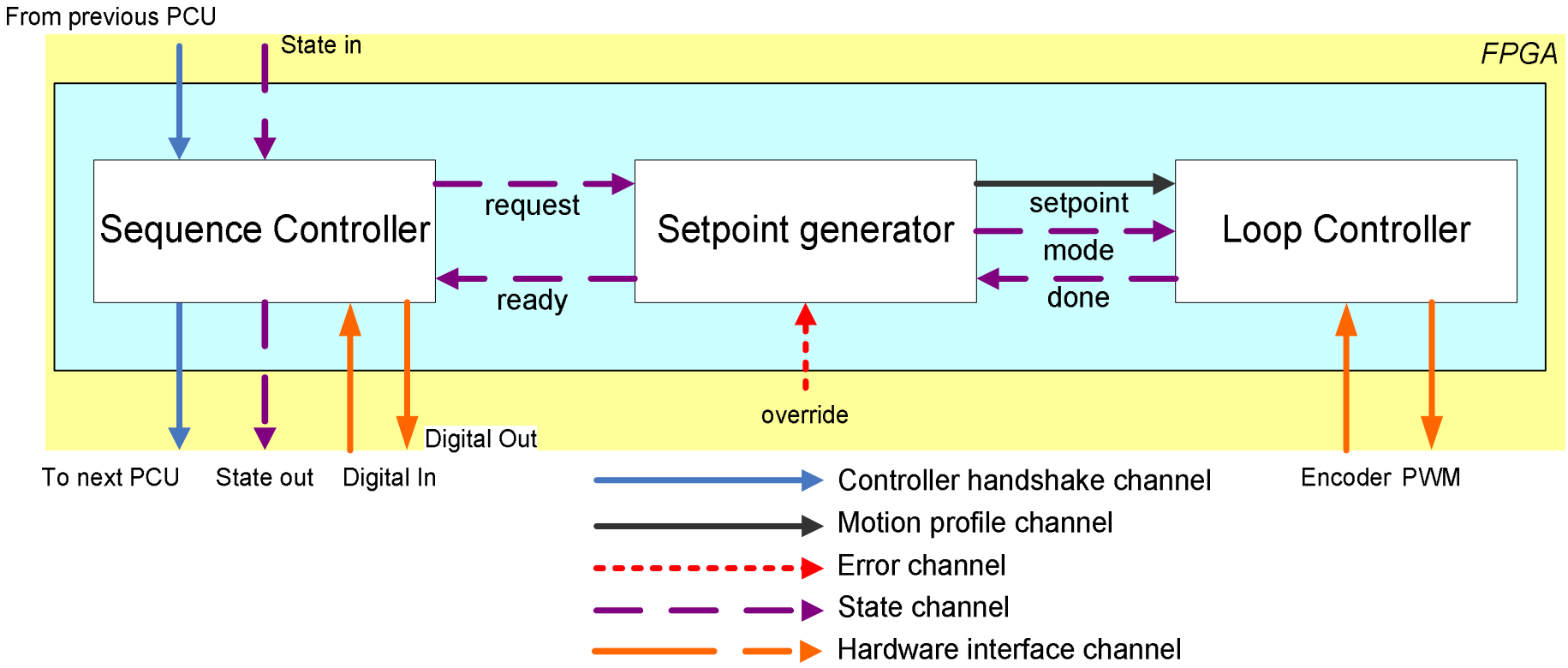
# ECS structure one PCU

UNIVERSITY OF TWENTE.



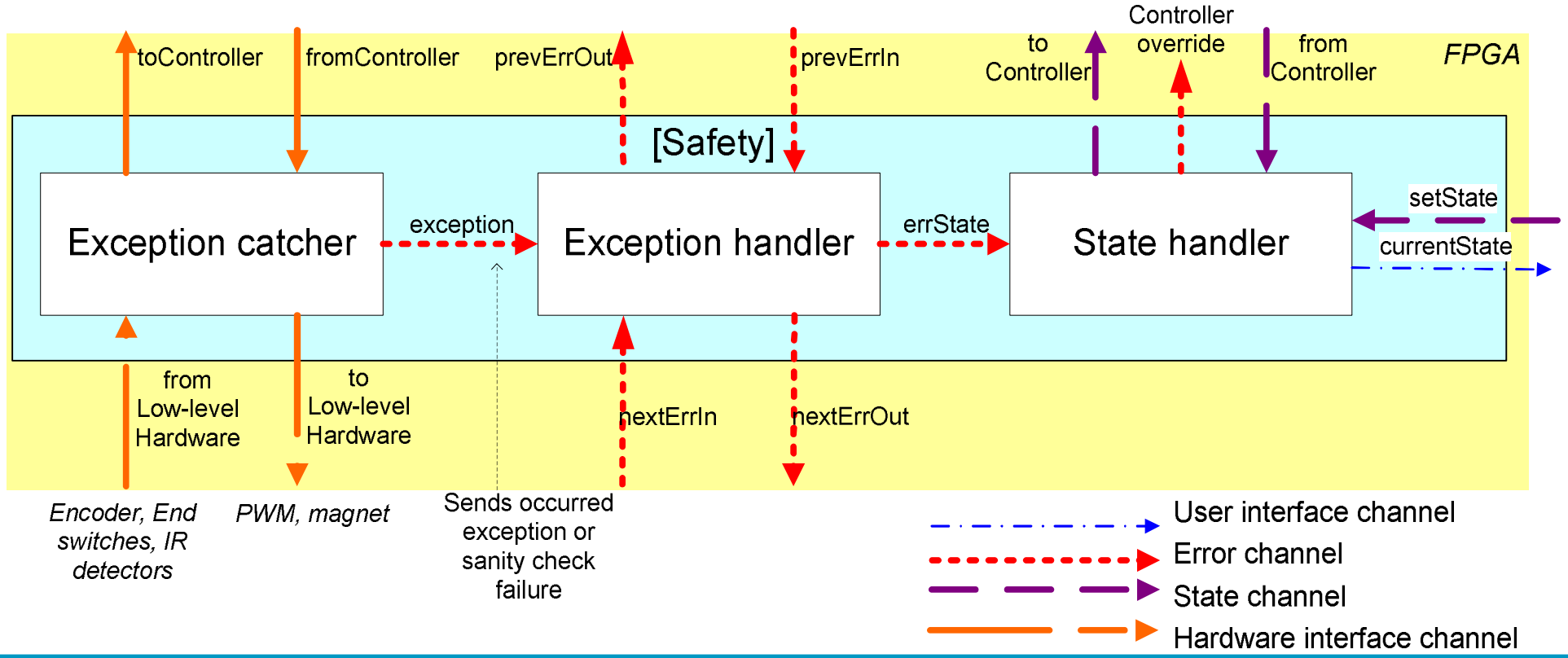
Production Cell Unit Controller

- Sequence controller: determine order of actions
- Setpoint generator: generate motion profiles
- Loop controller: control law

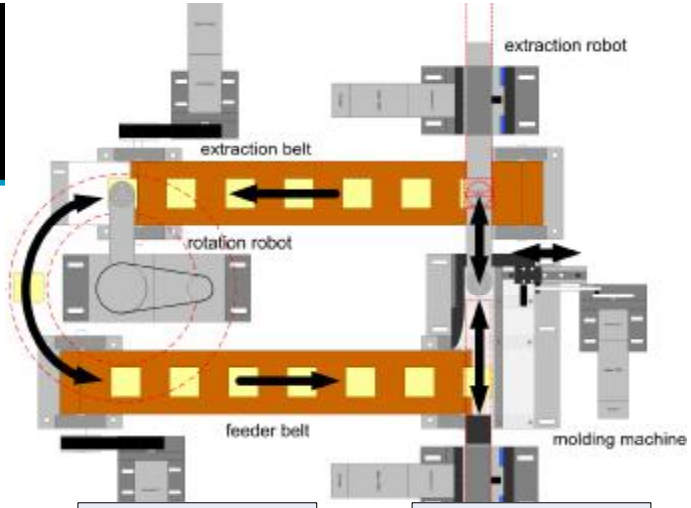


Production Cell Unit Safety layer

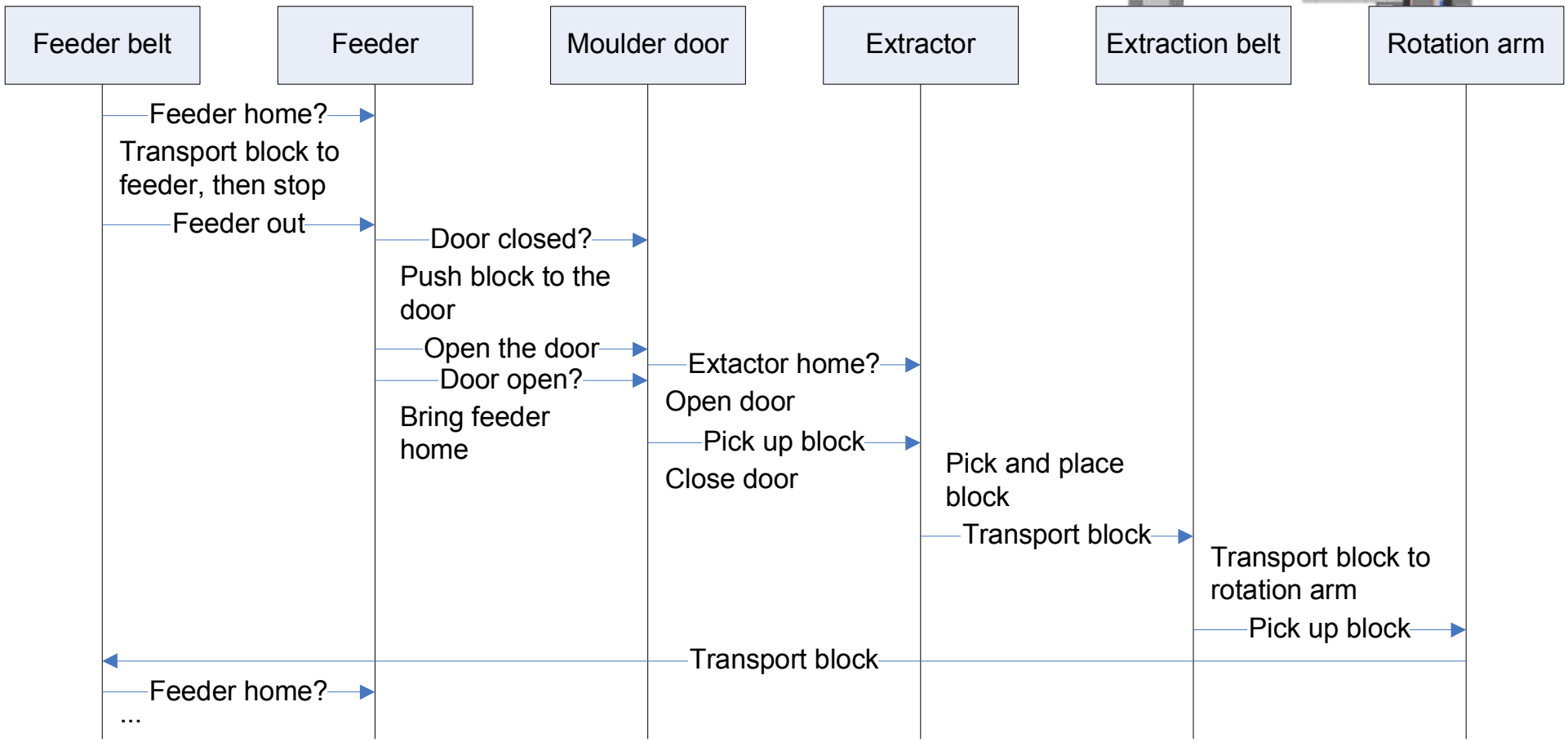
- Exception catcher: detect errors
- Exception handler: handle & select required action
- State handler: put PCU in a safe state



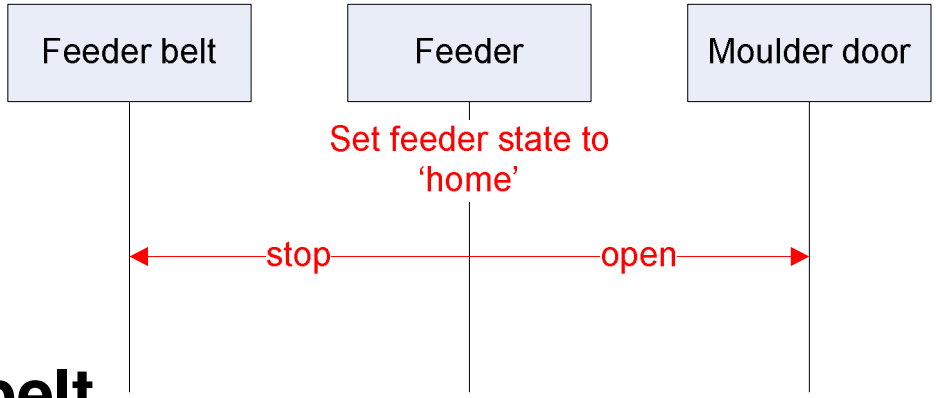
# ECS structure Control Sequence




## Flow normal operation

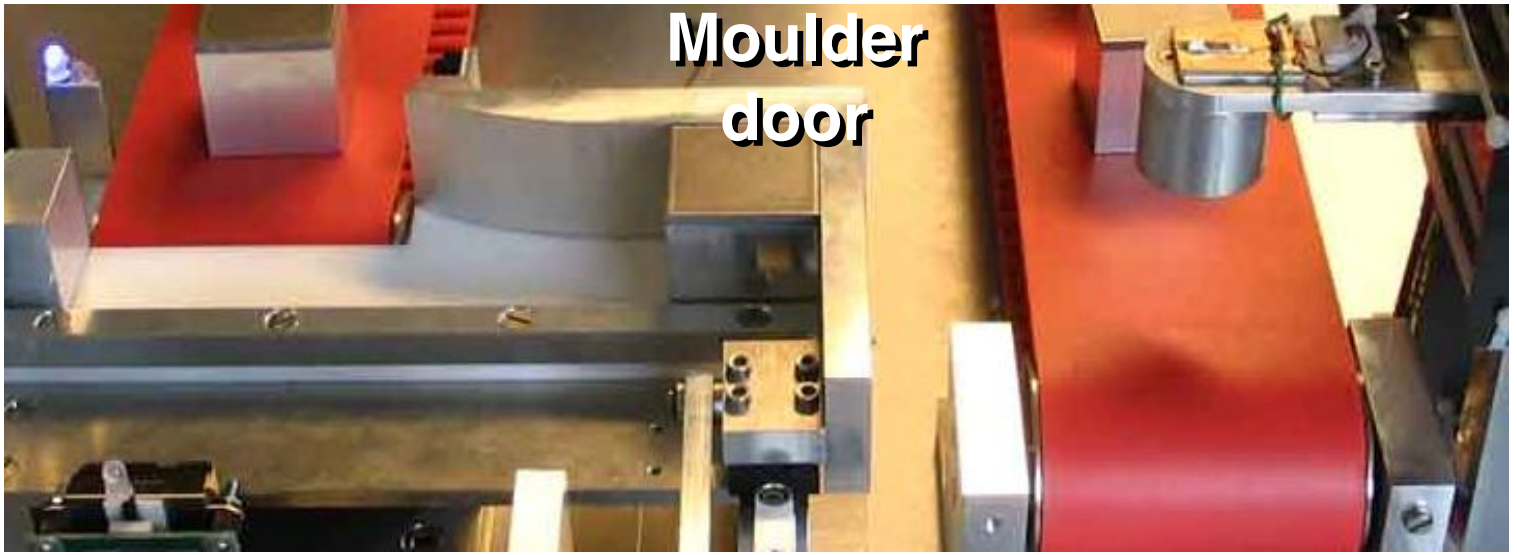


Flow of error operation




**Feeder belt**

**Feeder**



# 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: short design time, real-time behaviour  critical issue
- FPGA: longer 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
- Improvement Design Methodology

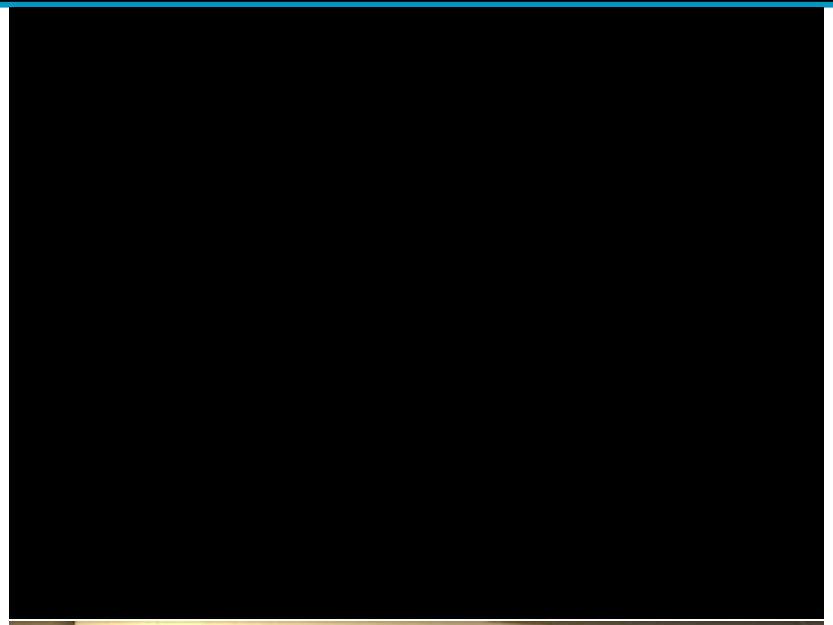
# Movies

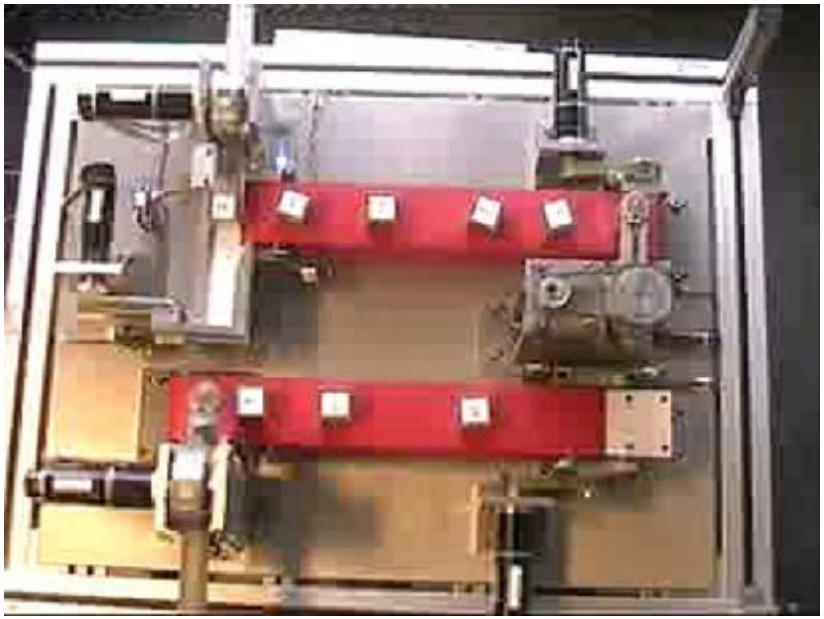


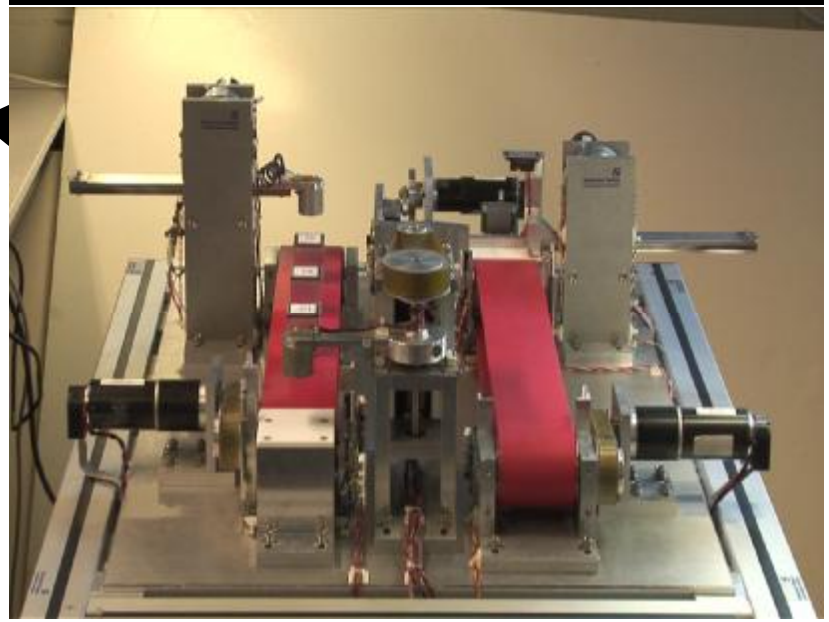
CPU controlled
- gCSP RTAI version

CPU controlled
- Ptolemy version
- Compared to 3D model

FPGA controlled
- Parallel, integer controllers







UNIVERSITY OF TWENTE.

# Embedded Control Software Design with Formal Methods and Engineering models

BCS FACS / FME evening seminar

BCS, London, 13-09-2010

**Jan F. Broenink**, Marcel A. Groothuis

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