



**SAAB**

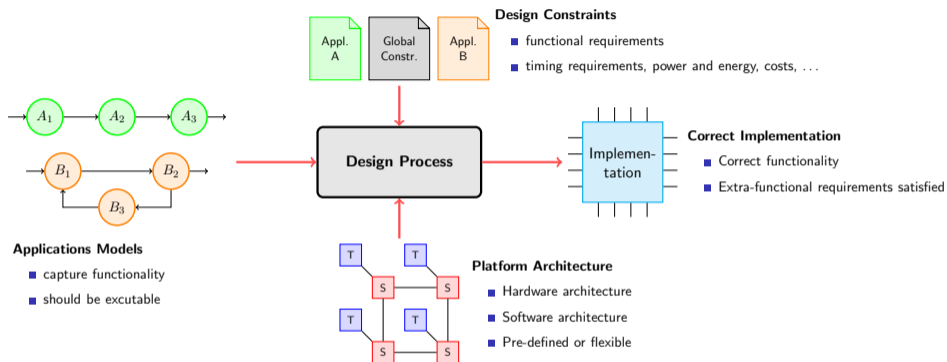
## Towards Seamless System Design from Specification Model to Implementation

FT2025, Stockholm, Sweden, October 14-15, 2025

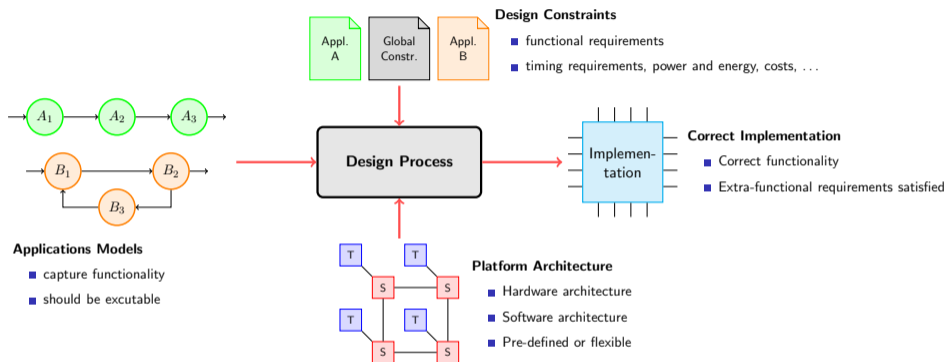
Ingo Sander (KTH), Fahimeh Bahrami (KTH), Rui Chen (KTH),  
Anders Åhländer (Saab), and Ingemar Söderquist (Saab)

Contact: Ingo Sander ([ingo@kth.se](mailto:ingo@kth.se))

# Industrial System Design



# Industrial System Design



## Vision: Correct-by-Construction Design

What is required of a design methodology and accompanying tools to significantly lower or eliminate the verification costs for future industrial system designs?

# System design

## Four principles for system design

According to [Sifakis \(2015\)](#), the success of electronic design automation (EDA) relies on the application of the following four interrelated principles.

- **Separation of concerns:** decomposition of design flow into steps, each step dealing with specific aspects.
- **Component-based design:** reasoned construction of systems by composition of components.
- **Semantic coherency:** descriptions (models) used in successive design steps are related through adequate semantic mappings.
- **Correct-by-construction:** possibility to guarantee functional and extra-functional properties of the designed systems incrementally and compositionally along the design process.

# System design

## Four principles for system design

According to [Sifakis \(2015\)](#), the success of electronic design automation (EDA) relies on the application of the following four interrelated principles.

- **Separation of concerns:** decomposition of design flow into steps, each step dealing with specific aspects.
- **Component-based design:** reasoned construction of systems by composition of components.
- **Semantic coherency:** descriptions (models) used in successive design steps are related through adequate semantic mappings.
- **Correct-by-construction:** possibility to guarantee functional and extra-functional properties of the designed systems incrementally and compositionally along the design process.

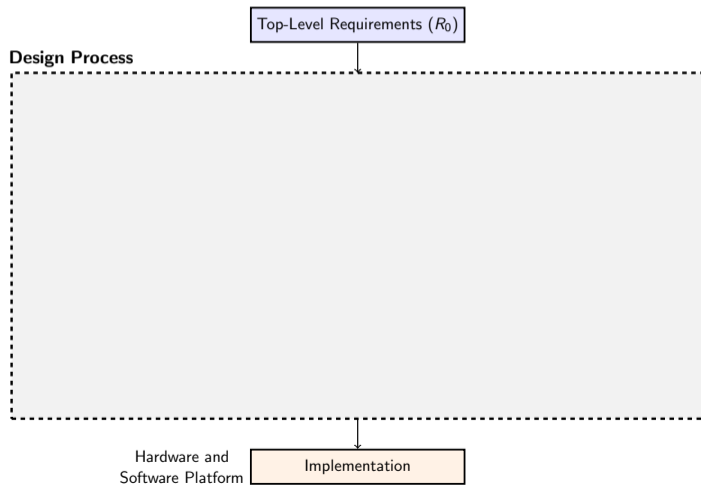


### **System design automation**

These four principles should also form the base for system design automation (SDA).

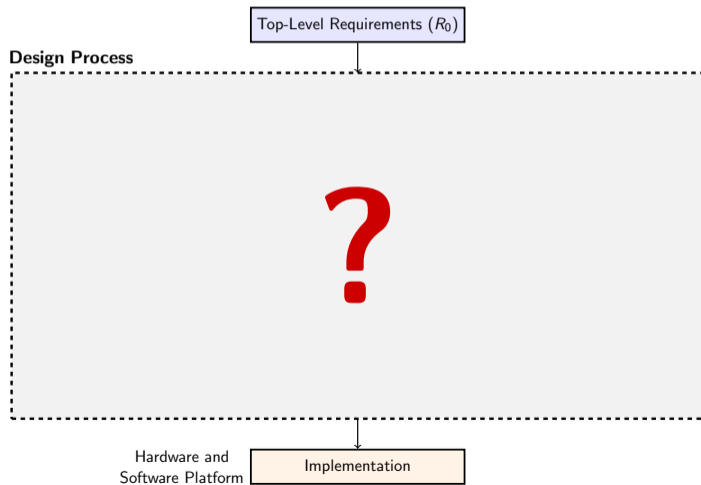
# System design

## Typical design scenario



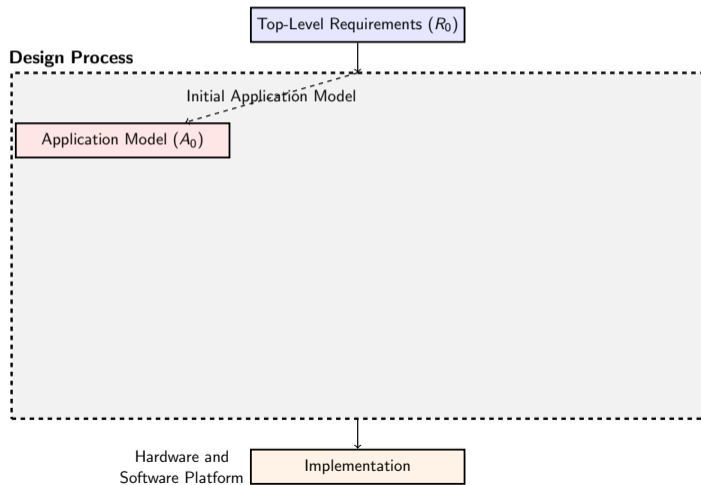
# System design

## Typical design scenario



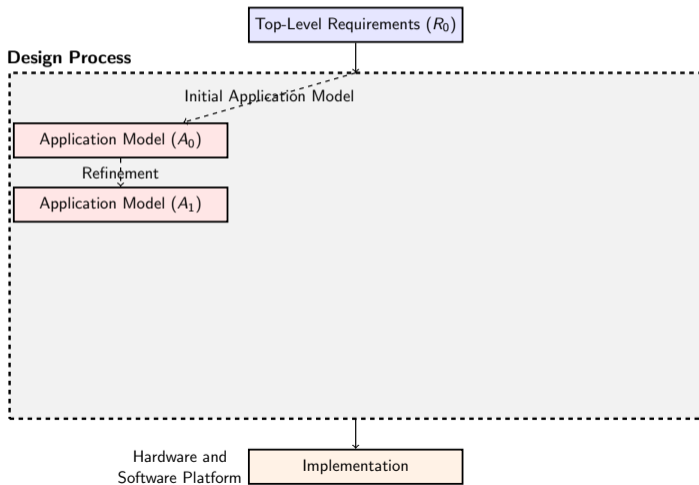
# System design

## Typical design scenario



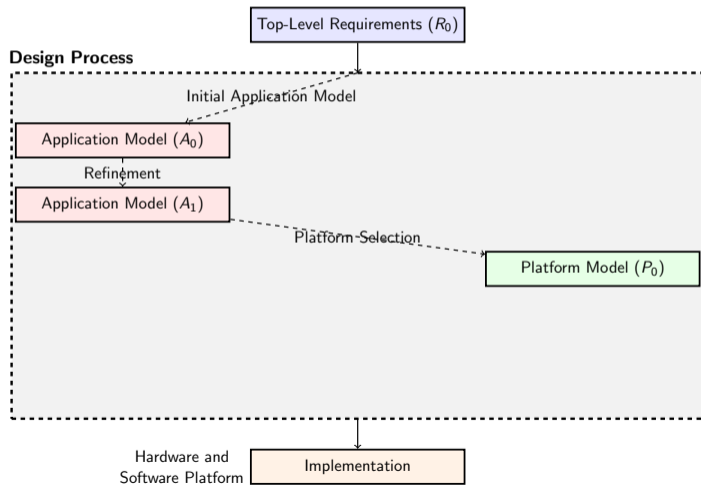
# System design

## Typical design scenario



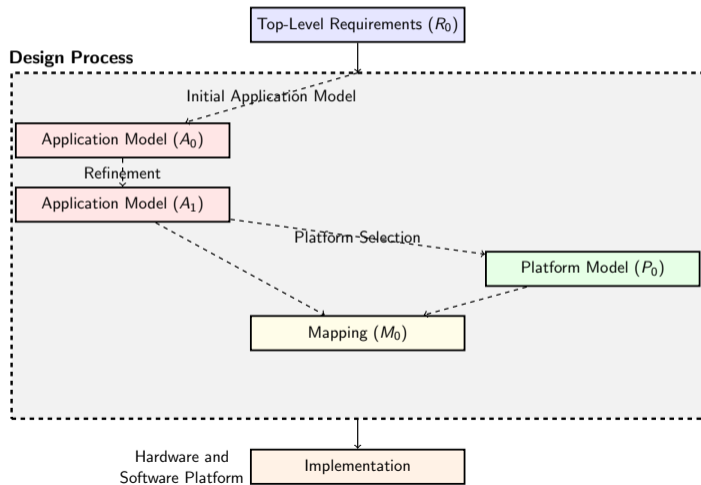
# System design

## Typical design scenario



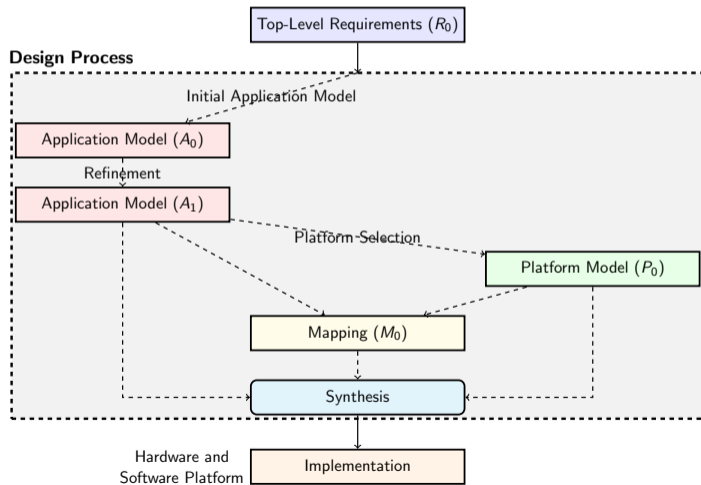
# System design

## Typical design scenario



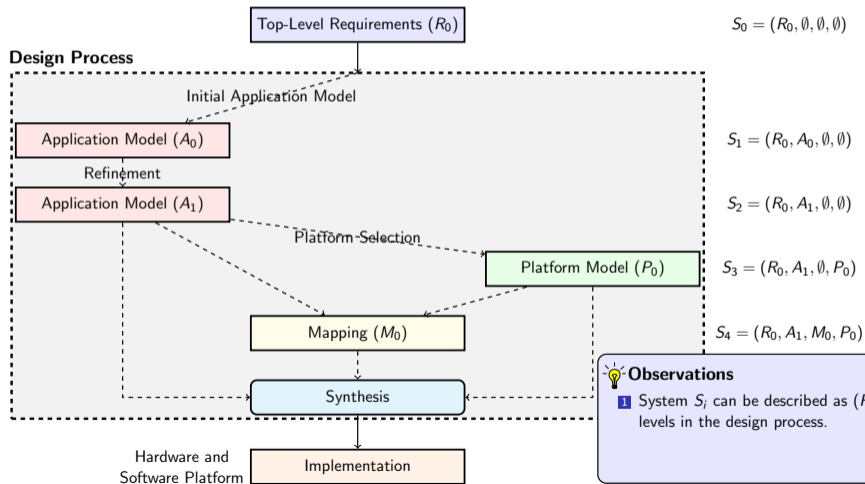
# System design

## Typical design scenario



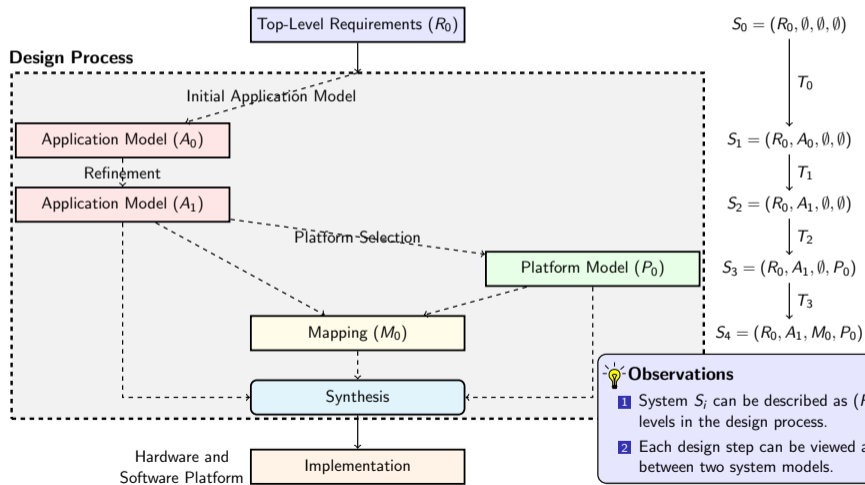
# System design

## Typical design scenario



# System design

## Typical design scenario



# Another view of system design

## Requirements

$$R = R_f \cup R_t \cup R_c$$

End-to-end delay	$R_t$
Functionality	$R_f$
Platform cost	$R_c$

$$S_0 = (R_0, \emptyset, \emptyset, \emptyset)$$

$$S_0 = (R_0, \emptyset, \emptyset, \emptyset)$$

# Another view of system design

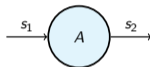
## Requirements

$$R = R_f \cup R_t \cup R_c$$

End-to-end delay	$R_t$
Functionality	$R_f$
Platform cost	$R_c$

$$S_1 = (R_0, A_0, \emptyset, \emptyset)$$

## Initial Application Model

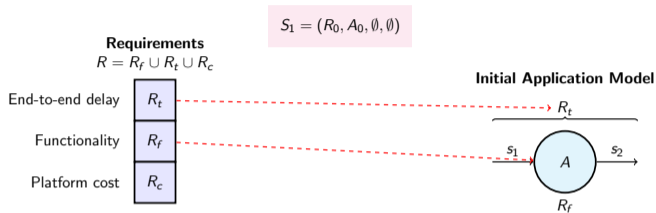


$$S_0 = (R_0, \emptyset, \emptyset, \emptyset)$$

$$\downarrow T_0$$

$$S_1 = (R_0, A_0, \emptyset, \emptyset)$$

# Another view of system design



$$S_0 = (R_0, \emptyset, \emptyset, \emptyset)$$

$$\downarrow T_0$$

$$S_1 = (R_0, A_0, \emptyset, \emptyset)$$

# Another view of system design

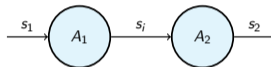
## Requirements

$$R = R_f \cup R_t \cup R_c$$

End-to-end delay	$R_t$
Functionality	$R_f$
Platform cost	$R_c$

$$S_2 = (R_1, A_1, \emptyset, \emptyset)$$

## Refined Application Model



$$S_0 = (R_0, \emptyset, \emptyset, \emptyset)$$

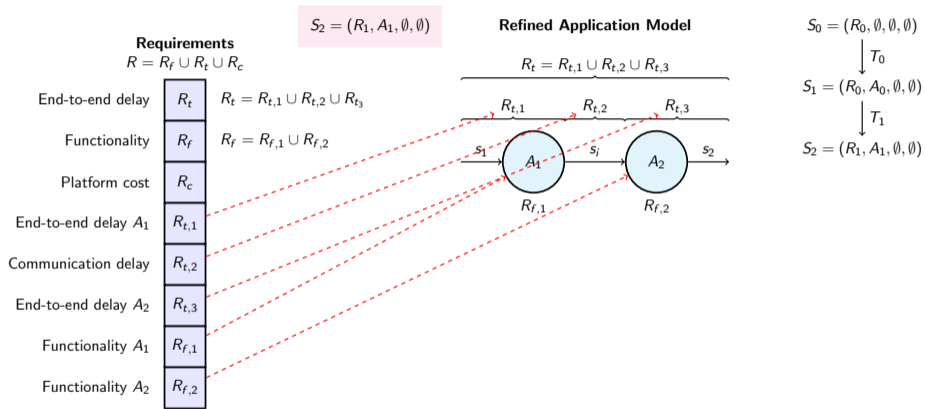
$$\downarrow T_0$$

$$S_1 = (R_0, A_0, \emptyset, \emptyset)$$

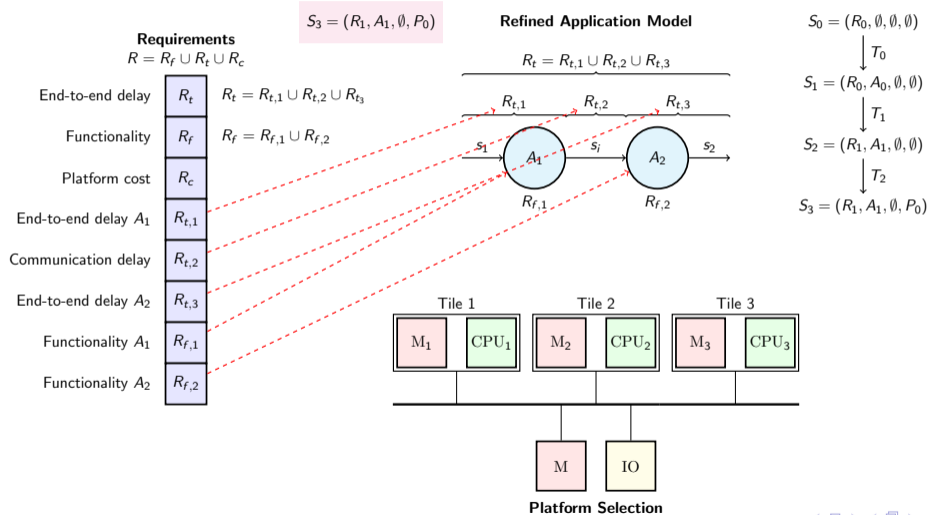
$$\downarrow T_1$$

$$S_2 = (R_1, A_1, \emptyset, \emptyset)$$

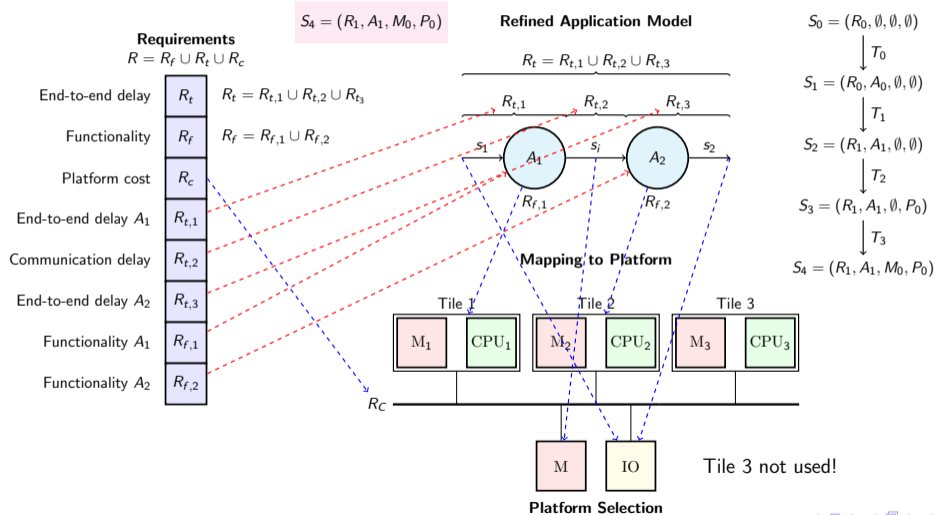
# Another view of system design



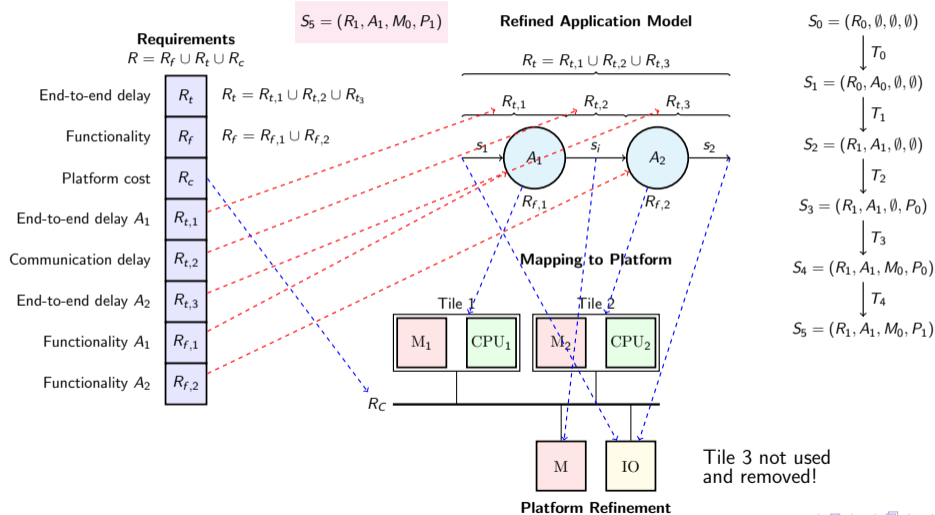
# Another view of system design



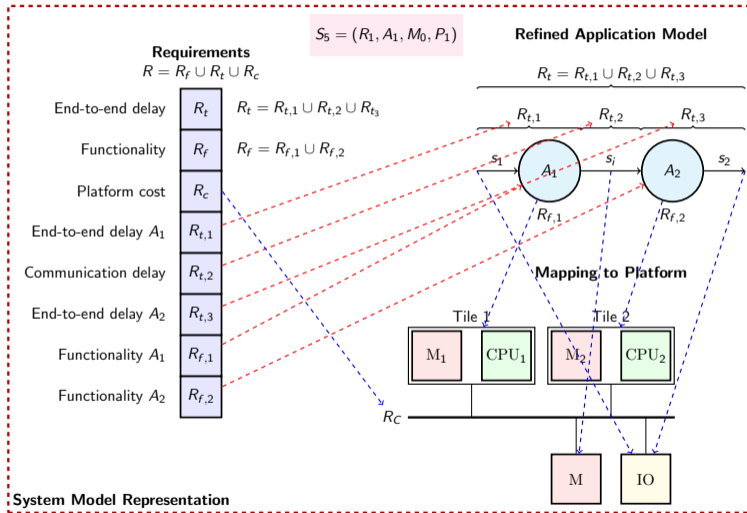
## Another view of system design



## Another view of system design



# Another view of system design



$$\begin{aligned}
 S_0 &= (R_0, \emptyset, \emptyset, \emptyset) \\
 &\downarrow T_0 \\
 S_1 &= (R_0, A_0, \emptyset, \emptyset) \\
 &\downarrow T_1 \\
 S_2 &= (R_1, A_1, \emptyset, \emptyset) \\
 &\downarrow T_2 \\
 S_3 &= (R_1, A_1, \emptyset, P_0) \\
 &\downarrow T_3 \\
 S_4 &= (R_1, A_1, M_0, P_0) \\
 &\downarrow T_4 \\
 S_5 &= (R_1, A_1, M_0, P_1)
 \end{aligned}$$

### Observation

Formal representation of the system model is needed to capture the complete design process!

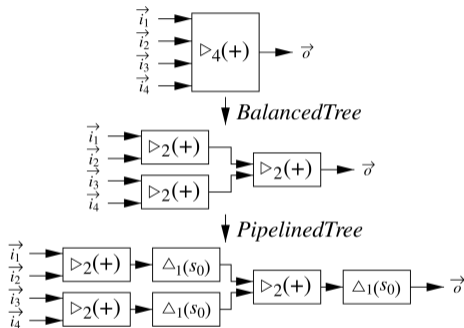
## Assumptions for transformational system design

System design can be viewed as step-wise application of design transformations converting an abstract specification into a physical implementation

- A system model  $S_i$  can at any level be described as a four-tuple  $(R_i, A_i, M_i, P_i)$
- A design transformation changes at least one parameter of the system model
- A design transformation can be formal or informal (ad hoc)
- A design transformation can preserve or change the semantics of the system model
- Discontinuities in the design process have to be bridged by well-defined semantic mappings

# Semantic- and non-semantic design transformations

## *BalancedTree* and *PipelinedTree*

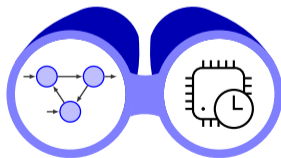


Source: (Sander and Jantsch, 2004)

- *BalancedTree* is a semantic-preserving transformation, which does not change the semantics of the application model.
- *PipelinedTree* is a non-semantic-preserving transformation, which changes the semantics of the model, since delay elements are introduced to achieve pipelining and a possibly higher clock frequency.
- Both transformations can be combined to the non-semantic-preserving transformation *PipelinedBalancedTree*.

## ForSyDe (Formal System Design)

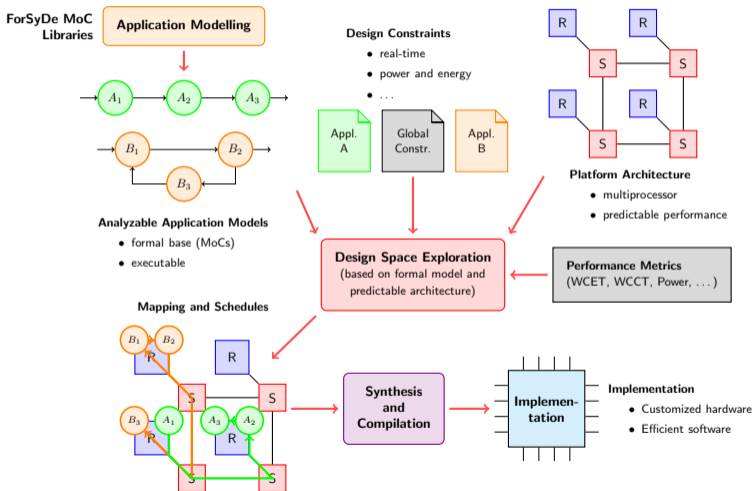
- ForSyDe is a design methodology targeting heterogeneous embedded systems design
- ForSyDe envisions a correct-by-construction design process
- ForSyDe consists of modelling libraries and tools for different steps of the design flow
- ForSyDe is based on the functional programming paradigm and a sound theoretical base in form of models of computation
- The ForSyDe modelling framework is implemented in Haskell and SystemC
- ForSyDe supports several models of computation



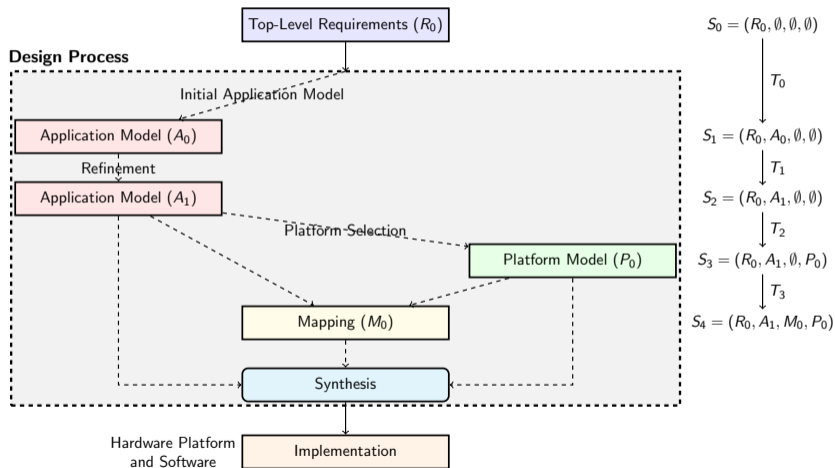
### ForSyDe vision

From high-level model to real-time implementation

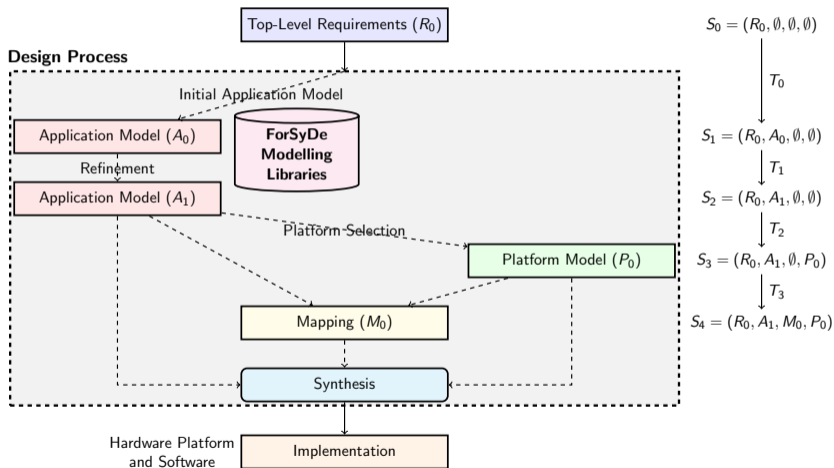
# 'Classic' ForSyDe System Design Flow



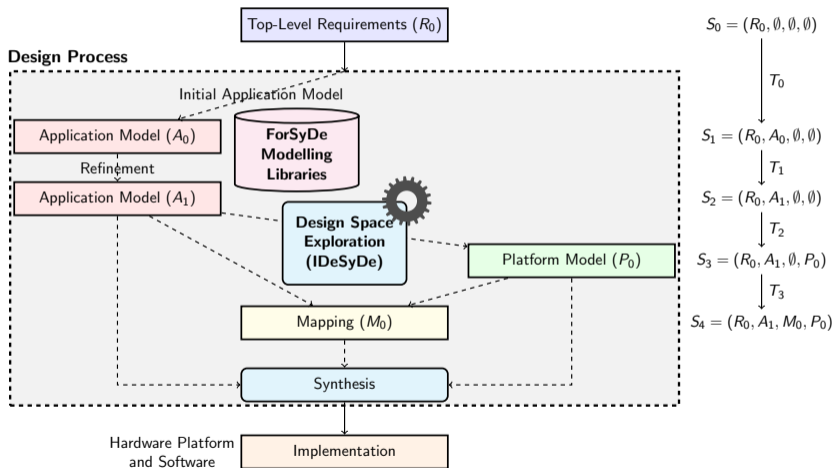
# Transformational ForSyDe System Design Flow



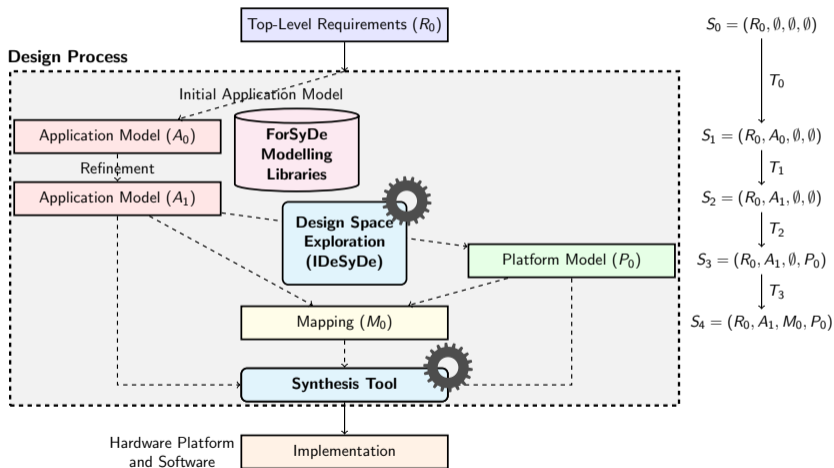
# Transformational ForSyDe System Design Flow



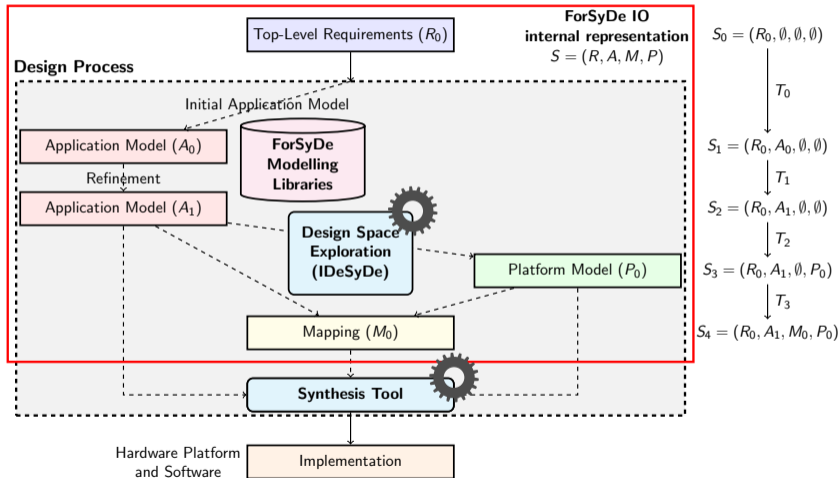
# Transformational ForSyDe System Design Flow



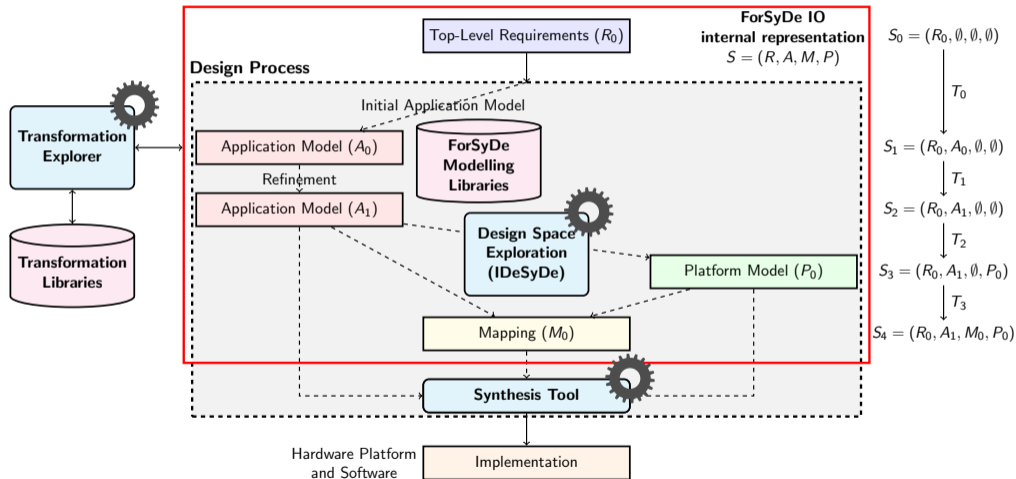
# Transformational ForSyDe System Design Flow



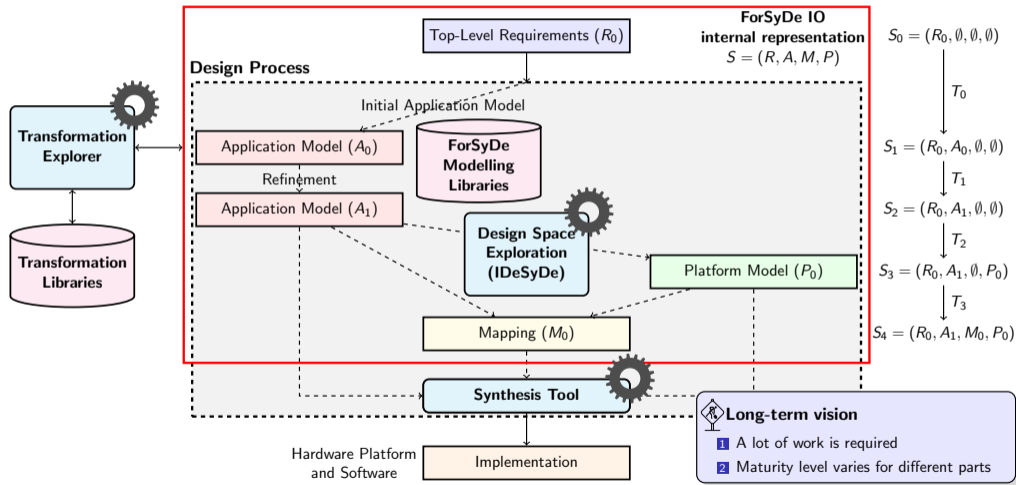
# Transformational ForSyDe System Design Flow



# Transformational ForSyDe System Design Flow



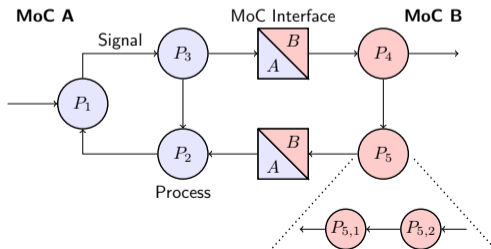
# Transformational ForSyDe System Design Flow



## ForSyDe Application Model

ForSyDe follows the tagged-signal model (Lee and Sangiovanni-Vincentelli, 1998).

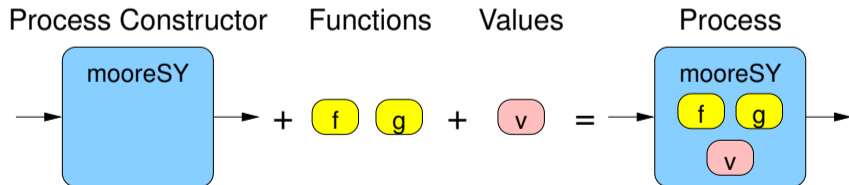
- An application is modelled as hierarchical concurrent process network
- Processes belonging to different models of computation communicate via MoC interfaces
- ForSyDe libraries in Haskell and SystemC to support the designer to create a formal model exist for several MoCs
  - e.g. synchronous MoC, continuous time MoC, SDF MoC, SADF MoC



## ForSyDe Process

A process takes  $m$  input signals as argument and produces  $n$  output signals. ForSyDe processes are deterministic.

- A process is always designed by means of a **process constructor**
- The process constructor defines the **model of computation** and the **communication interface** of the process
- The process constructor takes side-effect free **functions** and **values** as arguments and returns a process



The process constructor *mooreSY* constructs a Moore FSM process belonging to the synchronous MoC

## ForSyDe IO (Jordão et al., 2022)

ForSyDe's intermediate system representation, which captures the system model  $S$  as a graph  $(R, A, M, P)$

- The graph can capture any entity and the relationship between these entities
- Trait hierarchies are used to explicitly define specific characteristics of a submodel (e.g. the specific MoC or the tile-based platform)
- The graph can be manipulated type-safely due to trait hierarchies through supporting access functions
- This enables the definition of any transformation on the system model graph, which is used for design space exploration and transformation exploration.

## ForSyDe IO (Jordão et al., 2022)

ForSyDe's intermediate system representation, which captures the system model  $S$  as a graph  $(R, A, M, P)$

- The graph can capture any entity and the relationship between these entities
- Trait hierarchies are used to explicitly define specific characteristics of a submodel (e.g. the specific MoC or the tile-based platform)
- The graph can be manipulated type-safely due to trait hierarchies through supporting access functions
- This enables the definition of any transformation on the system model graph, which is used for design space exploration and transformation exploration.



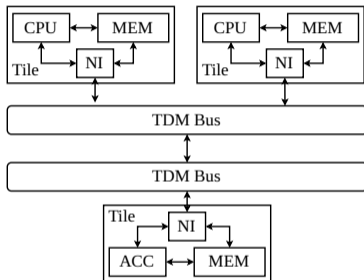
### **Accompanying proofs need to be supplied for transformations**

The correctness of a formal transformation rule needs to be verified separately. ForSyDe IO mainly gives the mechanisms for a tool that can support a transformation-based design process.

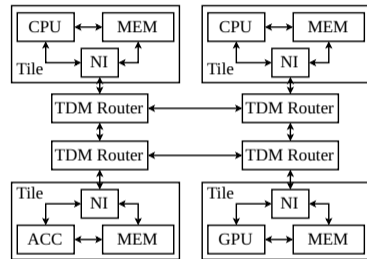
# ForSyDe IO supported Platform Model

Composable predictable hardware platform

Bus-based multiprocessor



Network-on-chip-based multiprocessor



- Define platform as composition of basic building blocks, which can be combined in different ways
- Guaranteed access and bandwidth to buses or communication links, e.g. using time division multiplex for reserving time slots for each processor tile

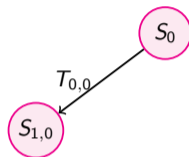
# Transparent Transformational System Design

- ForSyDe IO can capture the system model  $S = (R, A, M, P)$  at different stages of the design process
- Formal and informal transformations can be defined and applied within ForSyDe IO
- Using version control concepts enables to record the sequence of transformations and to revert transformations if they do not lead to a feasible implementation



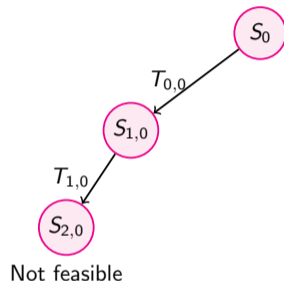
## Transparent Transformational System Design

- ForSyDe IO can capture the system model  $S = (R, A, M, P)$  at different stages of the design process
- Formal and informal transformations can be defined and applied within ForSyDe IO
- Using version control concepts enables to record the sequence of transformations and to revert transformations if they do not lead to a feasible implementation



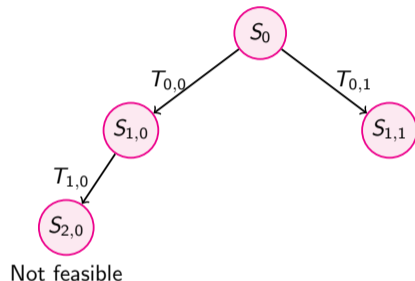
## Transparent Transformational System Design

- ForSyDe IO can capture the system model  $S = (R, A, M, P)$  at different stages of the design process
- Formal and informal transformations can be defined and applied within ForSyDe IO
- Using version control concepts enables to record the sequence of transformations and to revert transformations if they do not lead to a feasible implementation



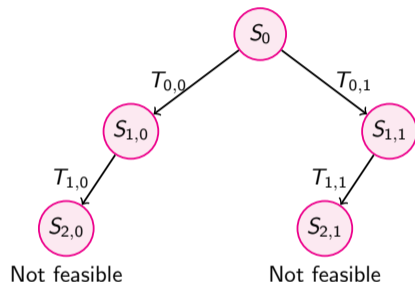
## Transparent Transformational System Design

- ForSyDe IO can capture the system model  $S = (R, A, M, P)$  at different stages of the design process
- Formal and informal transformations can be defined and applied within ForSyDe IO
- Using version control concepts enables to record the sequence of transformations and to revert transformations if they do not lead to a feasible implementation



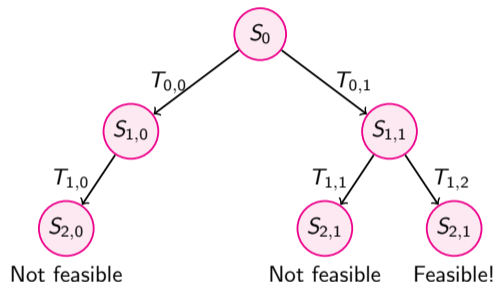
## Transparent Transformational System Design

- ForSyDe IO can capture the system model  $S = (R, A, M, P)$  at different stages of the design process
- Formal and informal transformations can be defined and applied within ForSyDe IO
- Using version control concepts enables to record the sequence of transformations and to revert transformations if they do not lead to a feasible implementation

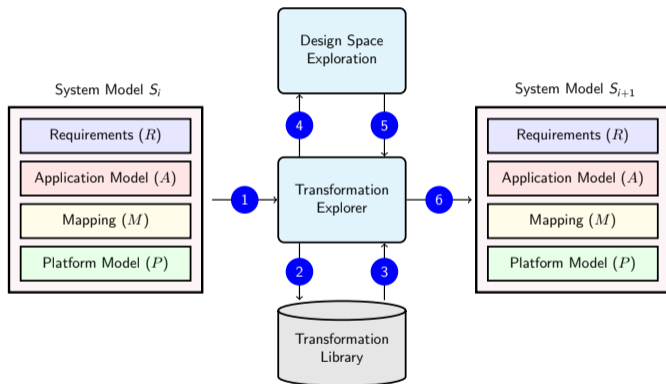


## Transparent Transformational System Design

- ForSyDe IO can capture the system model  $S = (R, A, M, P)$  at different stages of the design process
- Formal and informal transformations can be defined and applied within ForSyDe IO
- Using version control concepts enables to record the sequence of transformations and to revert transformations if they do not lead to a feasible implementation



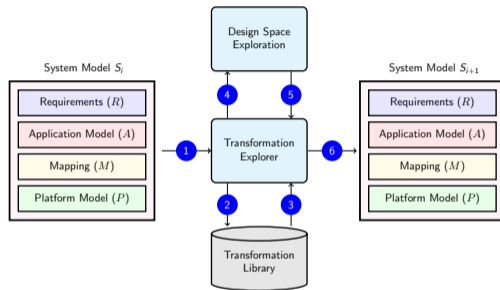
# Transformation Exploration



## Transformation Exploration

- 1-3 Transformation explorer (TE) analyses system model for possible transformations in the transformation library
- 4 TE applies transformations and sends a set of system models to design space exploration (DSE) tool
- 5 DSE tool returns evaluation for the set of candidate transformations
- 6 TE applies most promising transformation.

# Transformation Exploration



The concept of **transformation exploration** can support different scenarios.

- **Non-automated transformation exploration:** the TE suggests available transformations to the designer, and the designer selects, which transformation to apply.
- **Fully-automated transformation exploration:** the TE selects the design transformations and applies them until a feasible solution is found or all possible sequences of transformations have been applied.

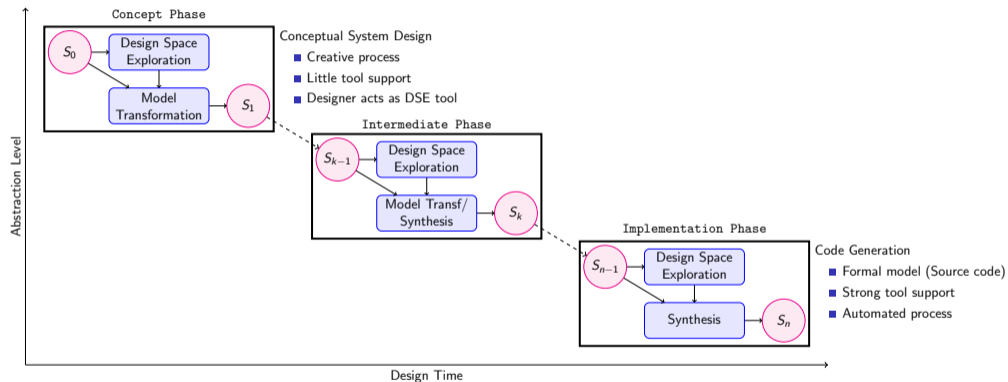


## Transformation Exploration and Transformation Explorer Tool

For more information check [Bahrami et al. \(2024, 2025\)](#); [Bahrami and Sander \(2025a,b\)](#).

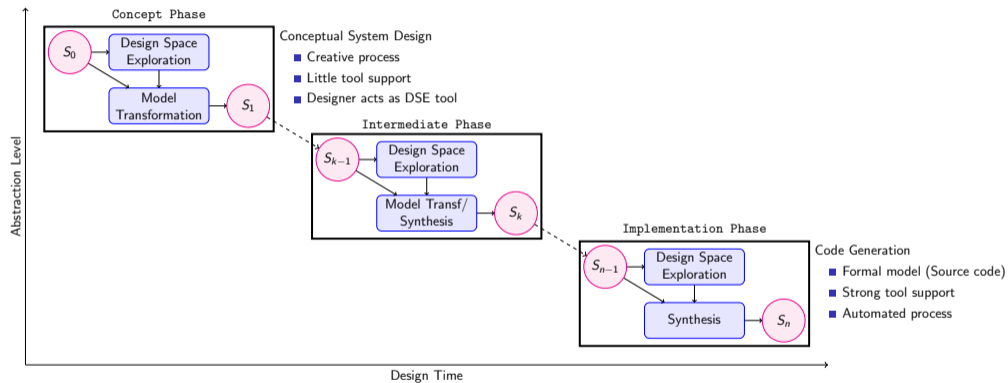
# Seamless System Design from Concept Phase to Implementation

Project: Early Bird (KTH, Saab, Ericsson, 2021–2025)



# Seamless System Design from Concept Phase to Implementation

Project: Early Bird (KTH, Saab, Ericsson, 2021–2025)



- **Observation:** Each design phase includes DSE and model transformation
- Presented transformational design and accompanying tools support such a design flow

## Summary

- The presentation proposed a seamless transformational system design flow as a step towards correct-by-construction design of embedded real-time multiprocessor systems.
- The vision is enabled by
  - common view of a system model  $S = (R, A, M, P)$  at all stages of the design process
  - well-defined modelling technique based on functional paradigm and tagged-signal model
  - composable platform model based on predictable architectures to provide service guarantees
  - intermediate representation to capture the system model at different stages of the design flow
  - concept for the definition of transformations, so that the implication of applying the transformation becomes visible to the designer
  - synthesis mechanism applicable to different MoCs and target platforms
  - initial tool support for parts of the methodology: modelling, synthesis, design space exploration

# There remains a lot of work and open problems...

## ■ Transformation Library

- sufficient set of transformations needed and proof required for formal transformations
- ⇒ focus on restricted domain (data-parallel systems using skeletons<sup>1</sup>)
- ⇒ capture design patterns and allow informal transformations under the designer's responsibility

## ■ Design Space and Transformation Exploration

- design space grows quickly for larger designs
- ⇒ capture designer's experience into tool to quickly identify suitable transformations
- ⇒ support combination and interplay of different solvers (metaheuristics and CP)

## ■ Platforms and Platform Model

- Few industrial platforms are composable and predictable
- ⇒ focus on composable and predictable platforms

## ■ Bridging Huge Abstraction Gap between Specification and Implementation

- Design flow will use several models at different levels of abstraction
- ⇒ Suitable abstractions and semantic coherent transformations have to be defined

---

<sup>1</sup>Skeletons are supported in ForSyDe by higher-order functions

## Supporting Projects and Links to ForSyDe

The work in this presentation has been supported by the following projects:

- CORRECT (Vinnova, Sweden, 2018–2022): Correct-by-Construction Design Flow,
- PANORAMA (ITEA3, EU, 2019–2022): Design Space Exploration,
- TRANSFORM (Vinnova, Sweden, 2019–2023): Transformational System Design,
- EARLY BIRD (Vinnova, Sweden, 2021–2025): Early Stage System Design.



## Supporting Projects and Links to ForSyDe

The work in this presentation has been supported by the following projects:

- CORRECT (Vinnova, Sweden, 2018–2022): Correct-by-Construction Design Flow,
- PANORAMA (ITEA3, EU, 2019–2022): Design Space Exploration,
- TRANSFORM (Vinnova, Sweden, 2019–2023): Transformational System Design,
- EARLY BIRD (Vinnova, Sweden, 2021–2025): Early Stage System Design.



The work will be continued in the following project:

- PARTI (Vinnova, Sweden, 2024–2028): Time-critical Parallel Applications on Avionics Platforms.

## Supporting Projects and Links to ForSyDe

The work in this presentation has been supported by the following projects:

- CORRECT (Vinnova, Sweden, 2018–2022): Correct-by-Construction Design Flow,
- PANORAMA (ITEA3, EU, 2019–2022): Design Space Exploration,
- TRANSFORM (Vinnova, Sweden, 2019–2023): Transformational System Design,
- EARLY BIRD (Vinnova, Sweden, 2021–2025): Early Stage System Design.



The work will be continued in the following project:

- PARTI (Vinnova, Sweden, 2024–2028): Time-critical Parallel Applications on Avionics Platforms.

### More information on ForSyDe

- ForSyDe web page: <https://forsyde.github.io/>
- ForSyDe tools are developed under a permissive open source license and publicly available on <https://github.com/forsyde>

# References I

- Fahimeh Bahrami and Ingo Sander. A transformation strategy for process partitioning in hierarchical concurrent process networks. *Journal of Systems Architecture*, 167:103509, 2025a. ISSN 1383-7621. doi: <https://doi.org/10.1016/j.sysarc.2025.103509>. URL <https://www.sciencedirect.com/science/article/pii/S138376212500181X>.
- Fahimeh Bahrami and Ingo Sander. Automating transformation strategy via attributed graphs for process network parallelization. In *2025 Forum on Specification and Design Languages (FDL)*, pages 1–10, 2025b. doi: 10.1109/FDL68117.2025.11165274.
- Fahimeh Bahrami, Rodolfo Jordão, Ingo Sander, and George Ungureanu. Automatic parallelization of embedded software via hierarchical process network transformations. In *2024 Forum on Specification and Design Languages (FDL)*, pages 1–9, 2024. doi: 10.1109/FDL63219.2024.10673845.
- Fahimeh Bahrami, Rodolfo Jordão, Ingo Sander, and Ingemar Söderquist. Bridging the abstraction gap: A systematic approach to rule-based transformational design for embedded systems. *ACM Trans. Embed. Comput. Syst.*, January 2025. ISSN 1539-9087. doi: 10.1145/3714412. URL <https://doi.org/10.1145/3714412>. Just Accepted.
- Rodolfo Jordão, Fahimeh Bahrami, Rui Chen, and Ingo Sander. A multi-view and programming language agnostic framework for model-driven-engineering. In *Forum on Specification & Design Languages 2022*, Linz, September 2022.
- Edward A. Lee and Alberto Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(12):1217–1229, December 1998.

## References II

- Ingo Sander and Axel Jantsch. System modeling and transformational design refinement in ForSyDe. [IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems](#), 23(1):17–32, January 2004. doi: 10.1109/TCAD.2003.819898.
- Joseph Sifakis. System design automation: Challenges and limitations. [Proceedings of the IEEE](#), 103(11):2093–2103, 2015. doi: 10.1109/JPROC.2015.2484060.