

Modeling Flow for Automated System Design and Exploration

Andreas Gerstlauer

Center for Embedded Computer Systems

University of California, Irvine

<http://www.cecs.uci.edu/~gerstl>



Outline

- **Introduction**
- **Design methodology**
- **Computation design**
- **Communication design**
- **Design environment**
- **Experimental results**
- **Summary and conclusion**

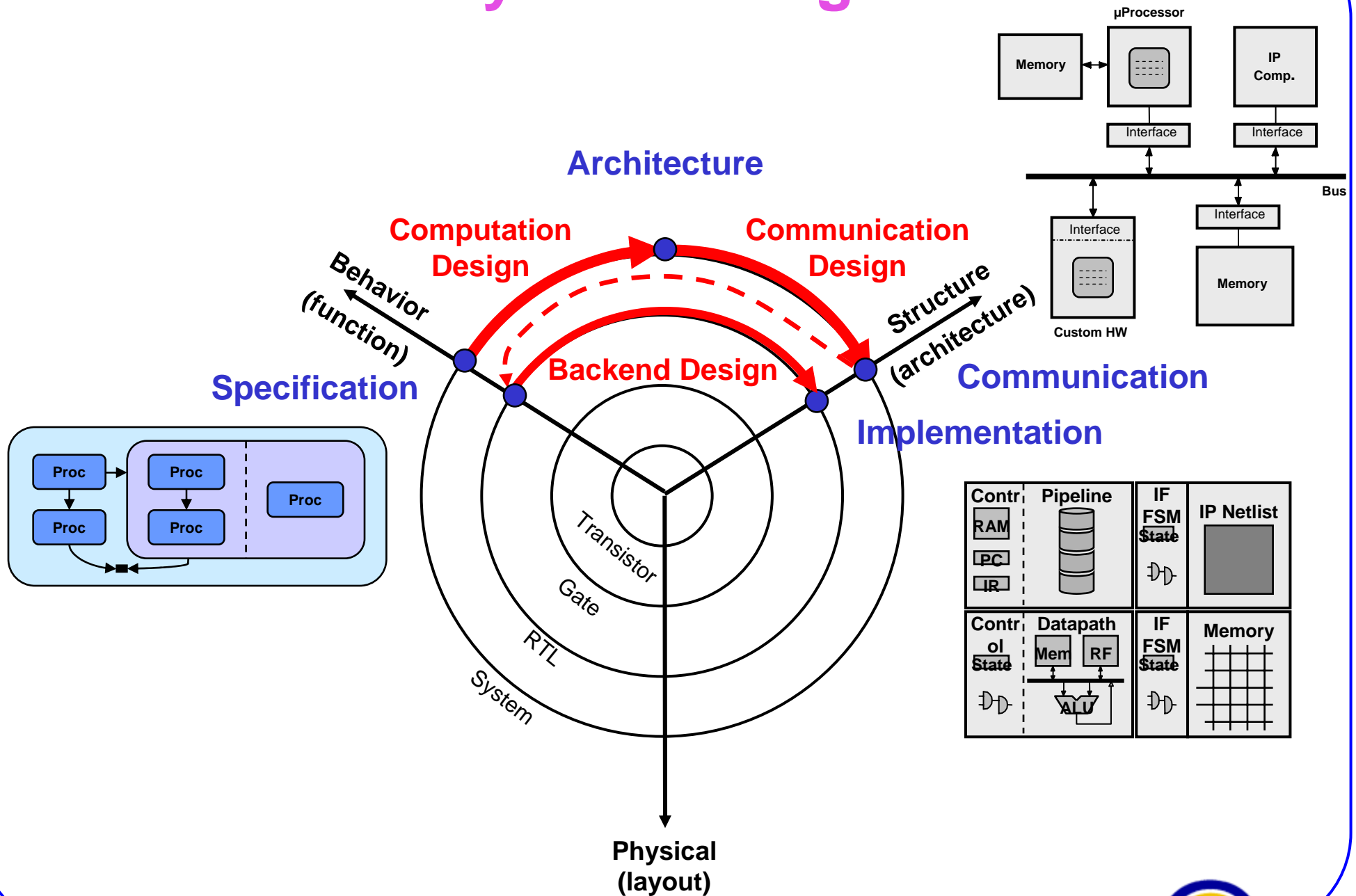


Motivation and Goals

- **Productivity gap, increase in design complexity**
 - Raise level of abstraction
 - Intellectual property (IP) reuse
- **Well-defined, rigorous, structured design flow**
 - Unambiguous abstractions, models, transformations
 - Systematic flow from specification to implementation
 - Reliable feedback at early stages
- Design automation for synthesis, verification
- Rapid, early design space exploration



System Design



Problem Definition

- **Bridge semantic gap**
 - Split into manageable design steps
- **Define intermediate abstraction levels, design models**
 - Synthesizable representation of critical design issues
 - Abstract unnecessary implementation detail
- **Define design steps**
 - Design decisions, model transformations
- **Enable design automation**
 - Automated model refinement, decision making
- **Enable rapid, early design space exploration**
 - Reliable feedback about critical issues at high levels
- **Support for realistic SoC designs**
 - Wide range of applications, target architectures



Related Work

- **System-level design**
 - System-level design languages (SLDL) [SystemC, SpecC]
 - Design methodologies [P-Chart, Rugby]
 - Design environments [OCAPI, POLIS, COSYMA, COSMOS]
 - *No complete, structured flow with specific models, steps & transformations*
 - *Limited applications, limited target architectures*
- **Simulation-centric system models**
 - Co-simulation at lower levels [Coste+99, Gerin+01]
 - Transaction-level models [SystemC TLM, IPSIM]
 - Models of computation for specification [Ptolemy]
 - *Horizontal integration of different models / components*
 - *Lack vertical integration for synthesis-centric approach*
- **Communication abstraction**
 - Communication synthesis [Coware, Lyonnard+01, Siegmund+01, Svarstad+01]
 - *No computational & intermediate abstraction, limited architectures*
- **Computation abstraction**
 - OS modeling [Tomiyama01, Desmet00]
 - *Not fully integrated with other system parts*

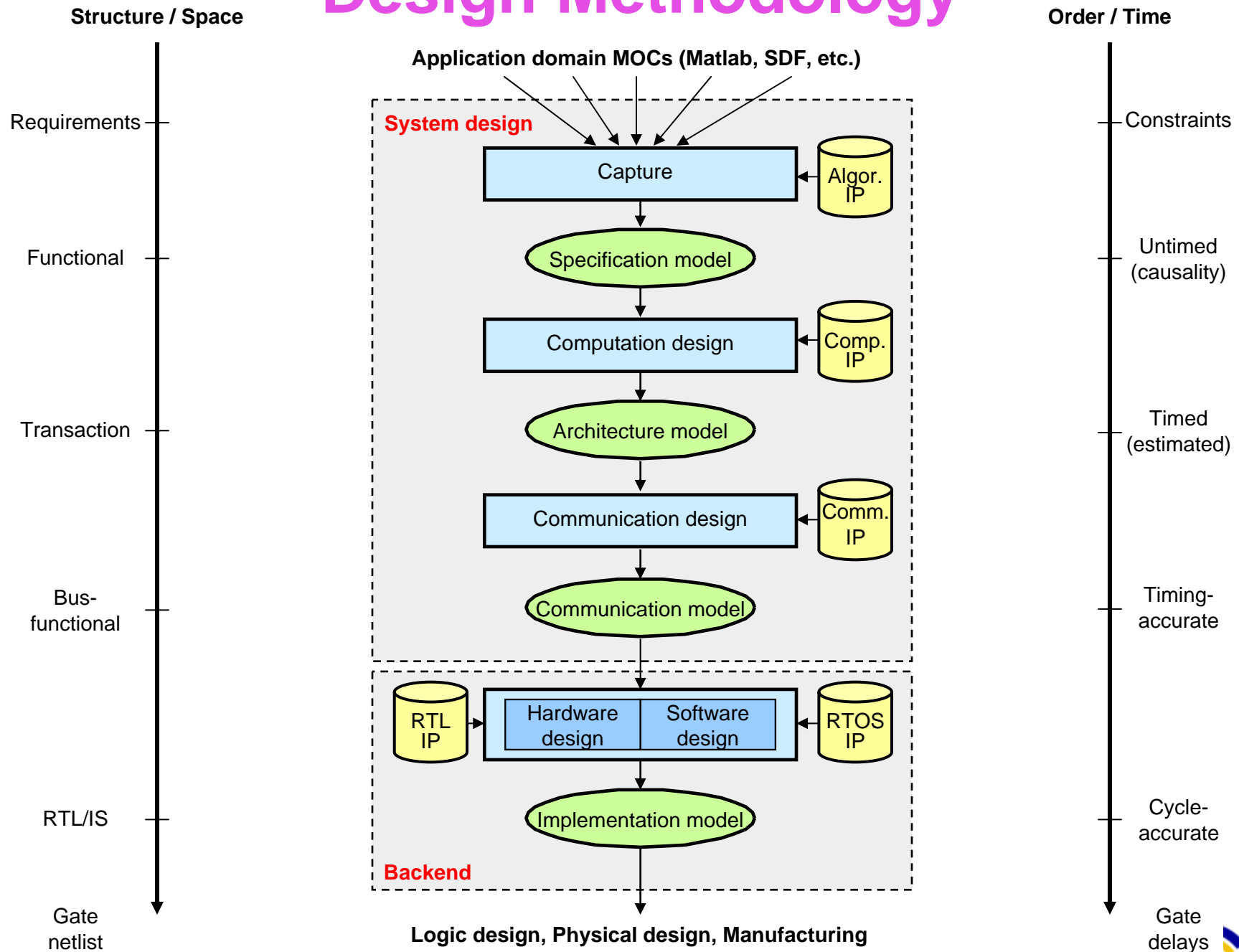


Outline

- Introduction
- **Design methodology**
- Computation design
- Communication design
- Design environment
- Experimental results
- Summary and conclusion

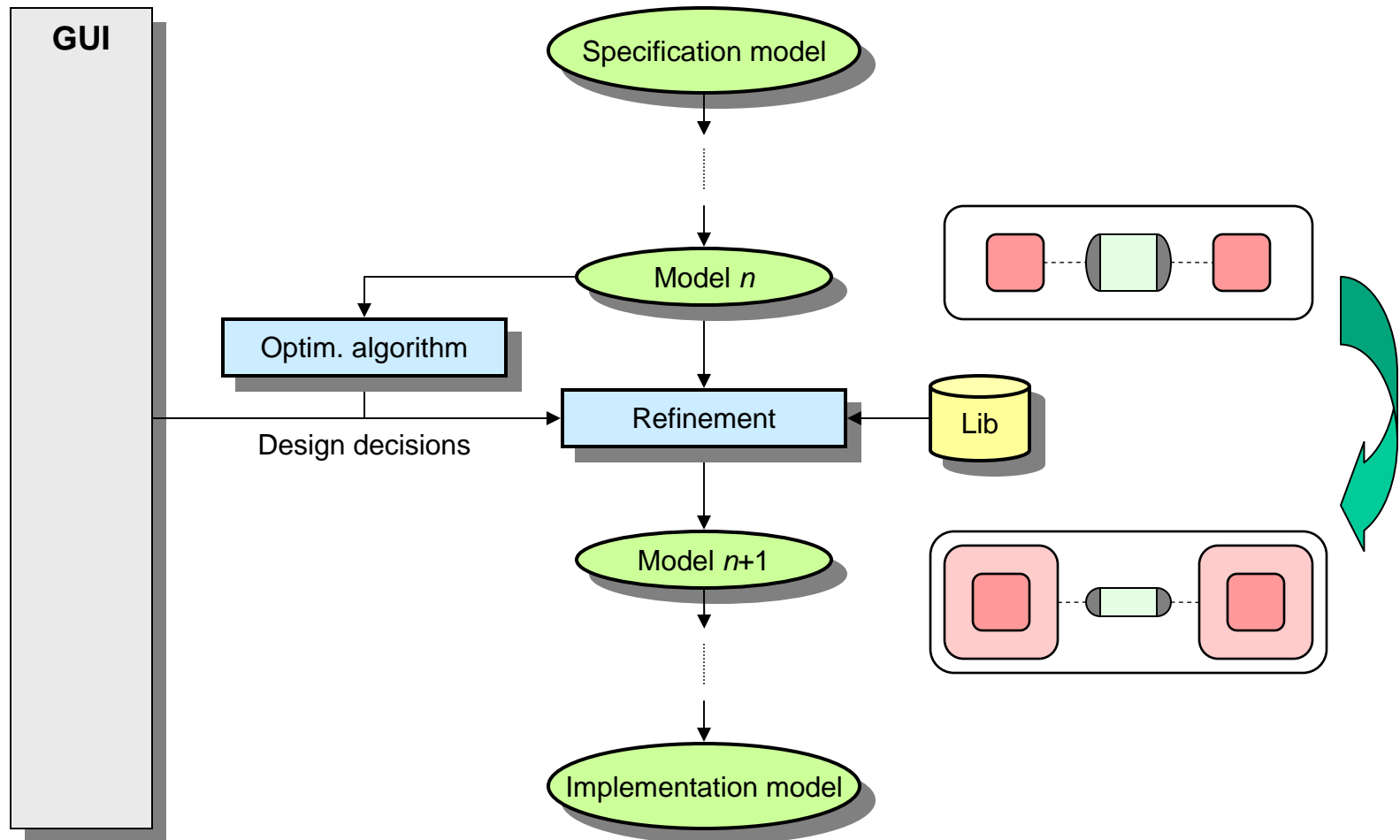


Design Methodology



Design Process

- **Synthesis = Decision making + model refinement**



- **Successive model refinement**
- **Layers of implementation detail**

Specification Model

- **PSM model of computation**
 - Abstract system functionality

Specification = $\langle B, V, C, R \rangle$

B : set of behaviors

V : set of variables

C : set of channels

$R \subseteq B \times (C \cup V)$: connectivity relation

Behavior semigroup (B, \circ) , $\circ \in \{\triangleright, \parallel, |, \vee\}$

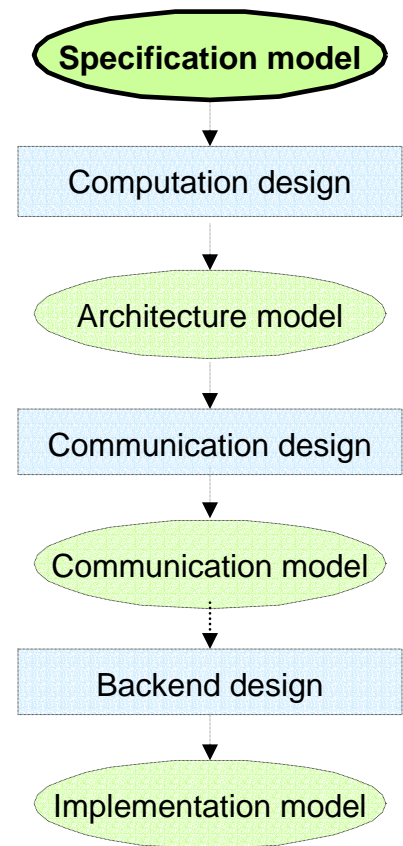
\triangleright : sequential composition

\parallel : parallel composition

$|$: pipelined loop composition (plus guard)

\vee : mutually exclusive (plus guard)

- No implementation detail: untimed / no structure



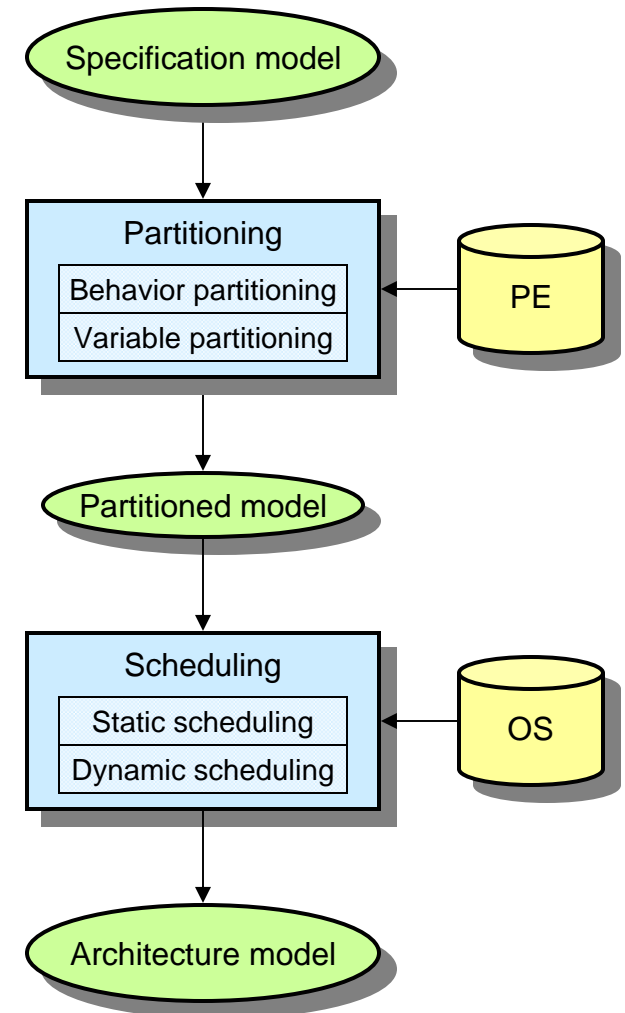
Outline

- Introduction
- Design methodology
- **Computation design**
- Communication design
- Design environment
- Experimental results
- Summary and conclusion



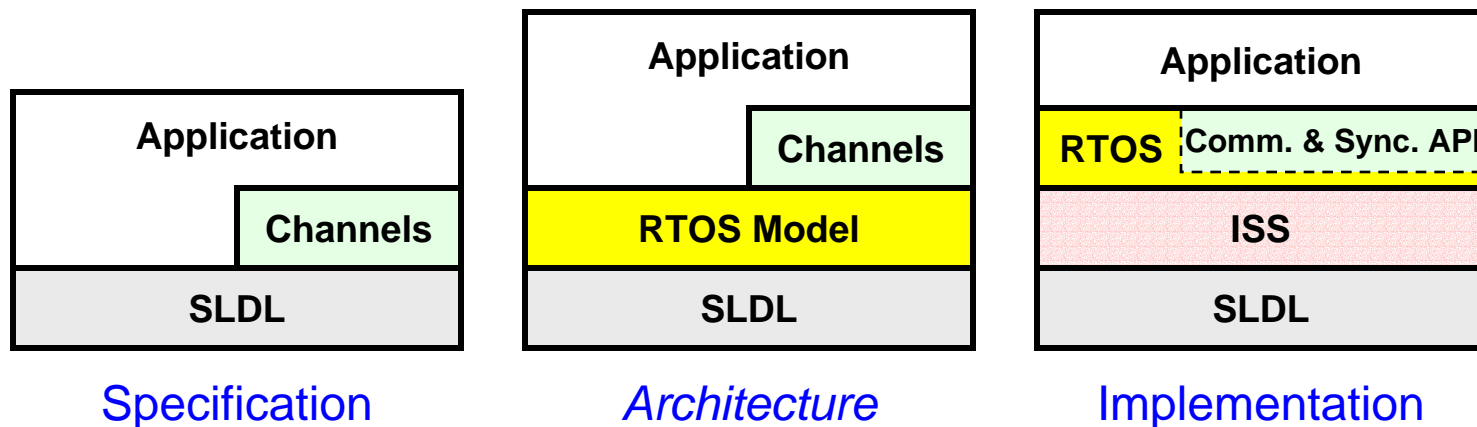
Computation Design Flow

- **Partitioning (structure / space)**
 - Define PE, memory architecture
 - Map behaviors, variables onto PEs, memories
- **Scheduling (order / time)**
 - Serialize behaviors on PEs
 - Pre-defined, fixed order
 - Dynamically under control of OS scheduler



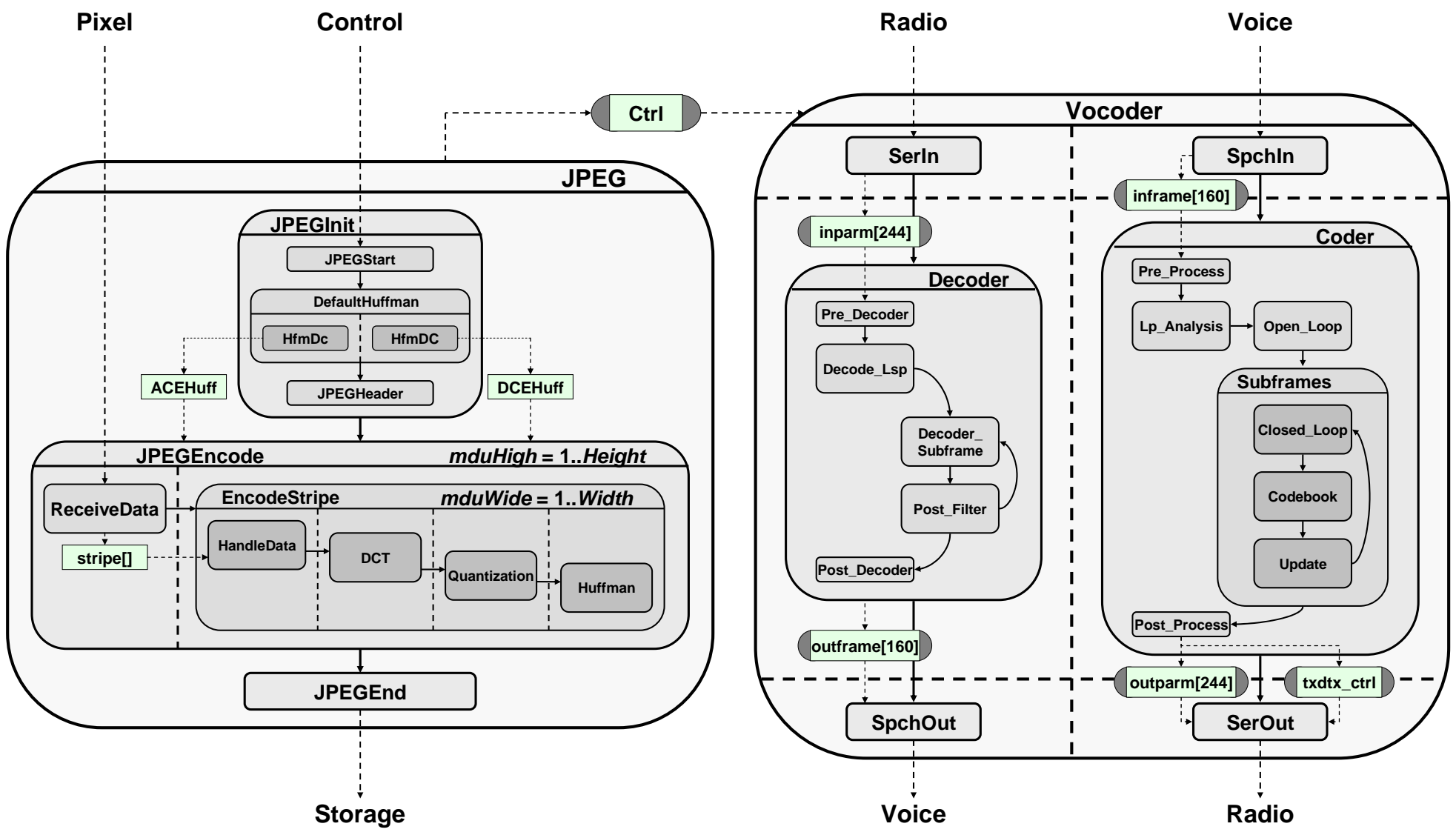
OS Modeling

- **High-level RTOS abstraction**
 - Model standard RTOS concepts
 - Multi-tasking, time-sharing, preemption
 - Real-time scheduling
 - Task synchronization & communication
 - Wrap around SLDL primitives, replace event handling

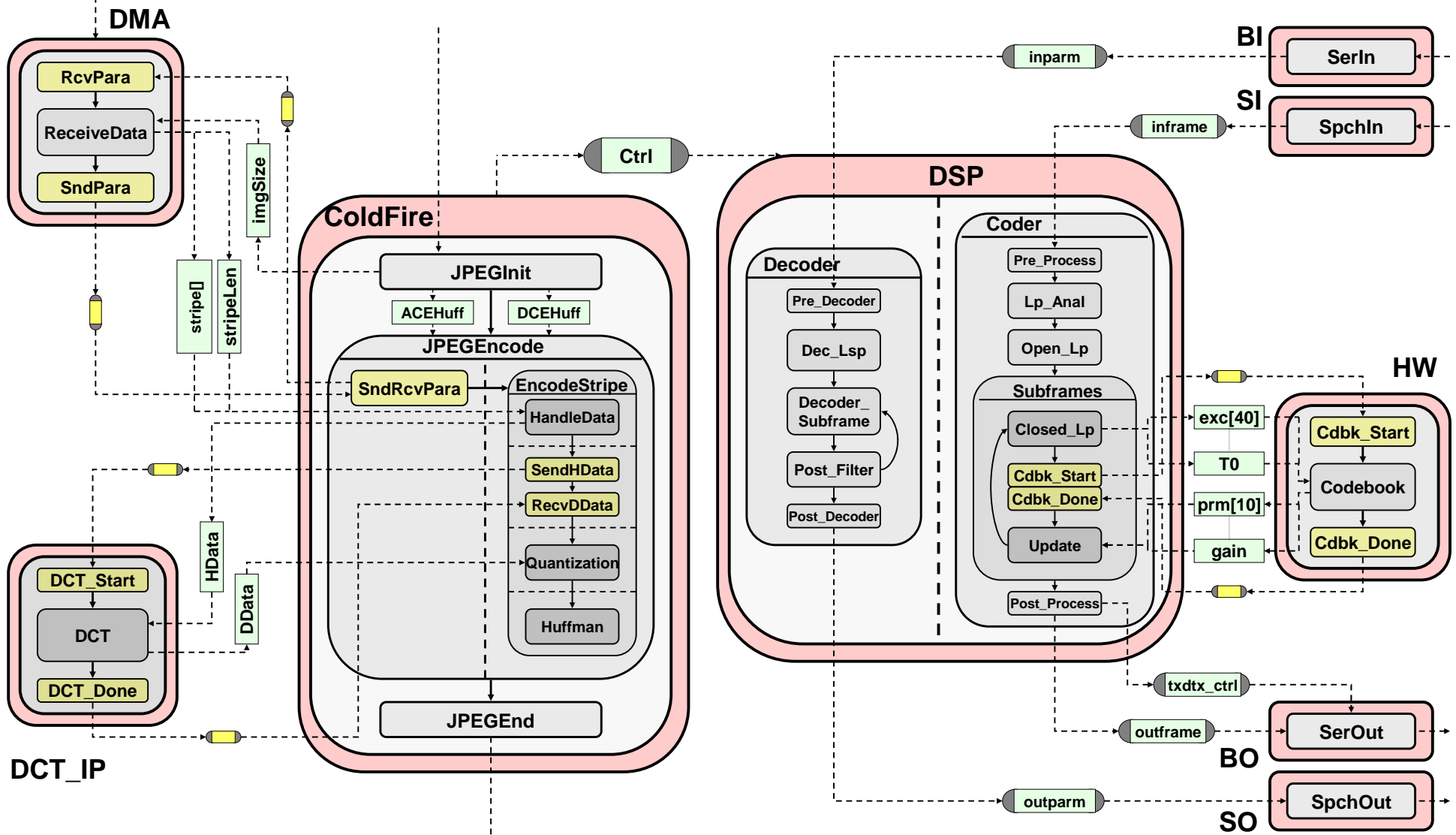


- **Accurate feedback at early stage**
 - Small overhead, low complexity
 - High relative accuracy

Specification Model Example



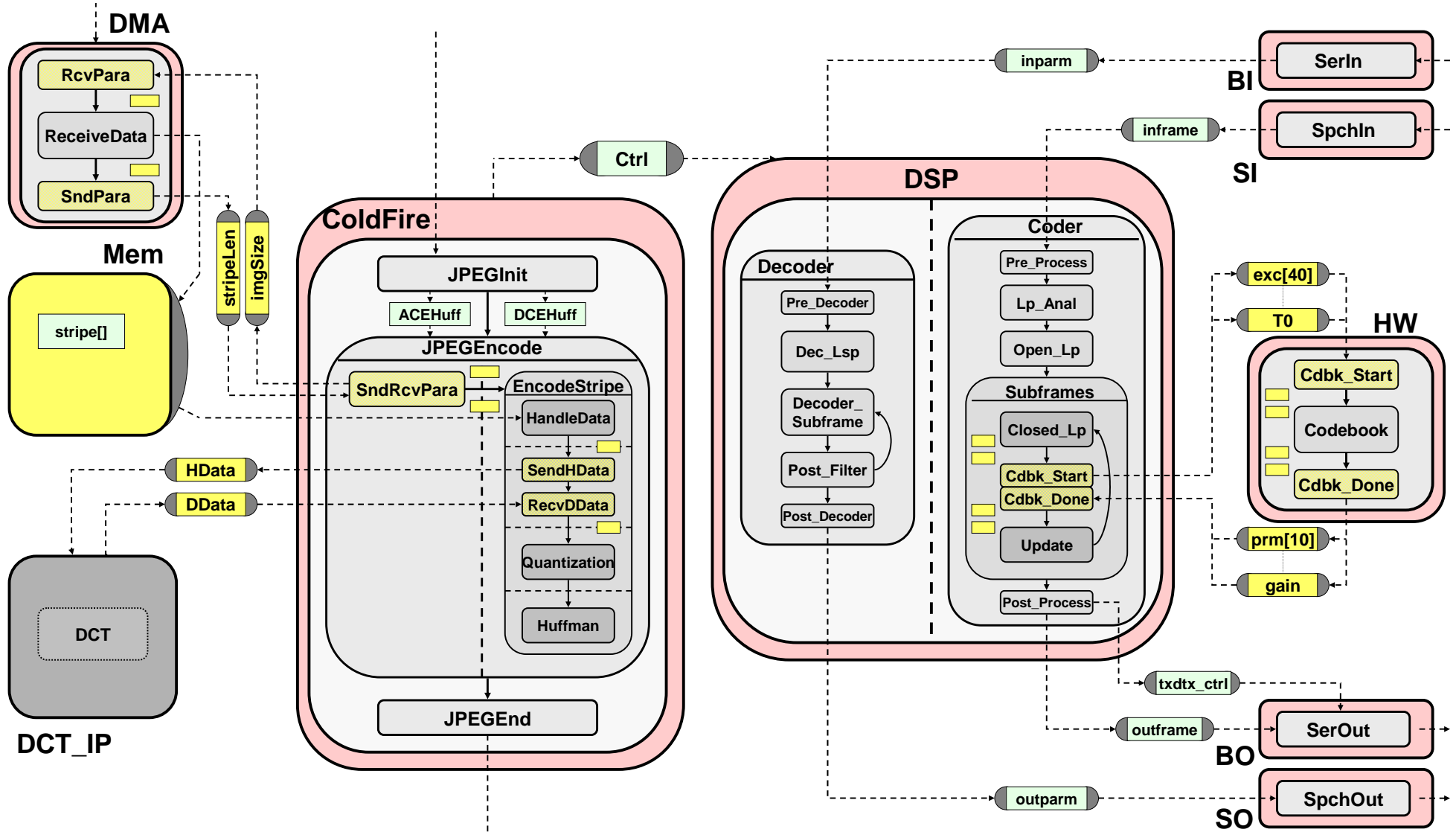
PE Model Example



➤ PE layer, Behavior grouping, Synchronization, Timing

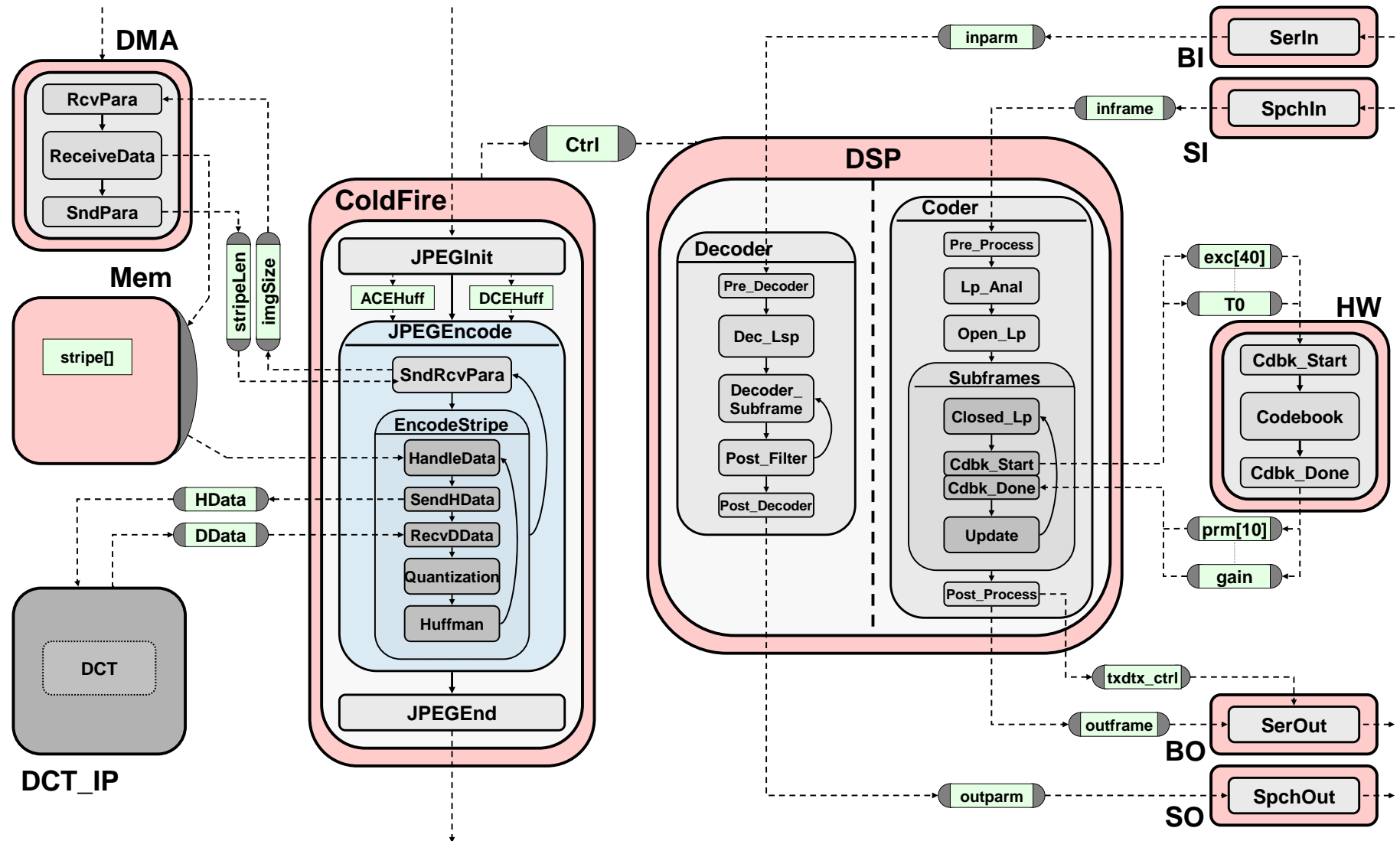


Partitioned Model Example



➤ Memory layer, Variable grouping, Message passing, Memory access

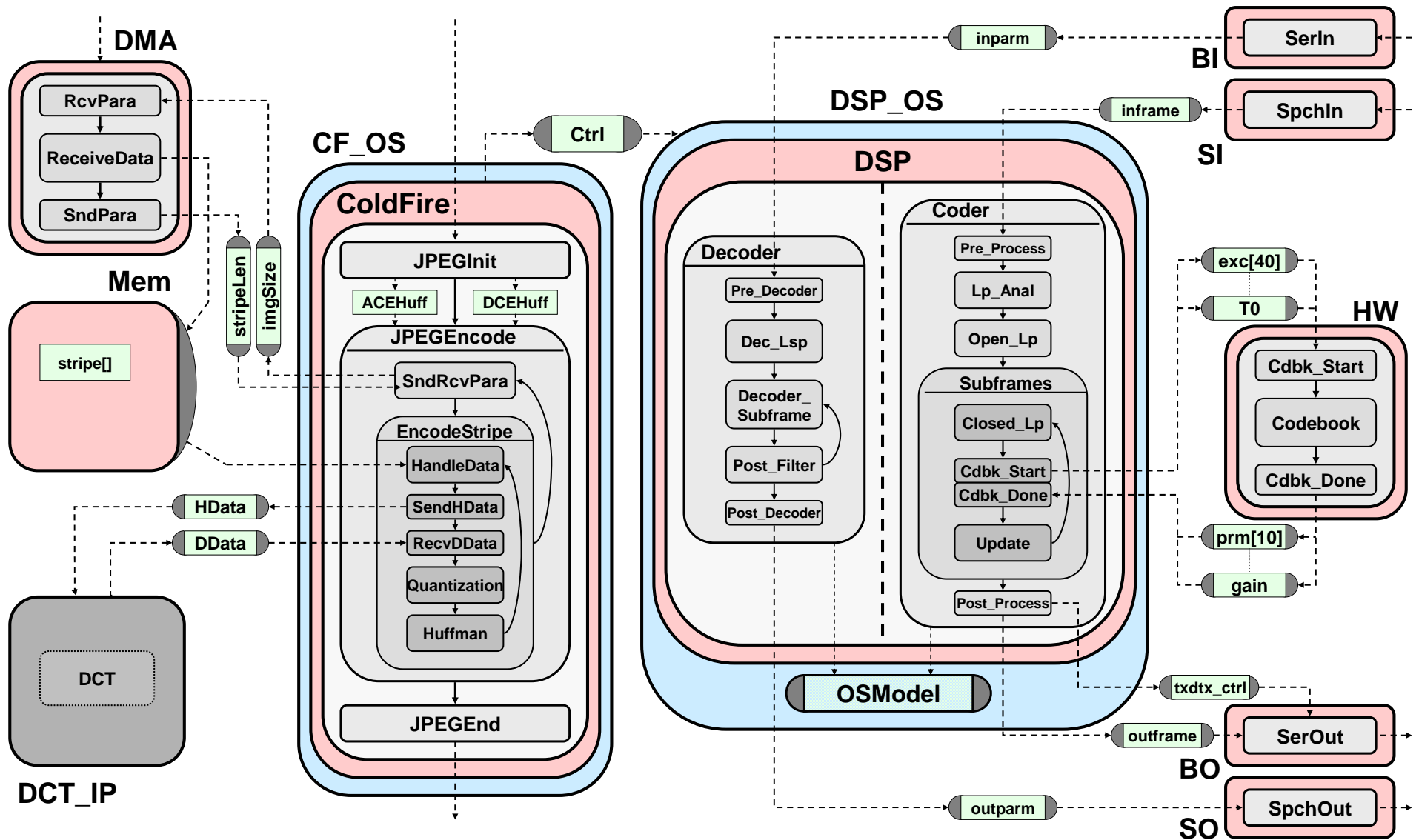
Scheduled Model Example



➤ Behavior serialization, flattening, and reordering



Architecture Model Example



➤ OS layer + OS model, Task creation, Timing and synchronization refinement



Architecture Model

- **Computation structure**
 - Non-terminating, concurrent PEs
 - Sequential, timed PE behaviors
- **Abstract communication**
 - Untimed message-passing
 - Shared memory variable accesses

Architecture = $\langle PE, C, R \rangle$

$PE = P \cup IP \cup M$: set of processing elements

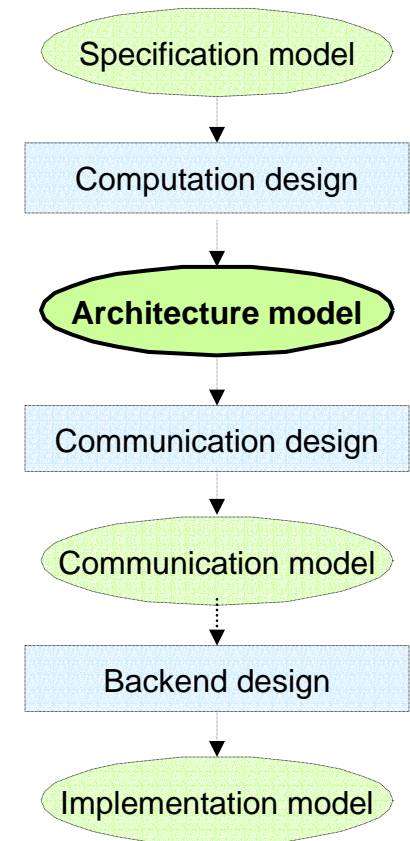
C : set of system channels

$R \subseteq B \times (C \cup A)$: system connectivity relation

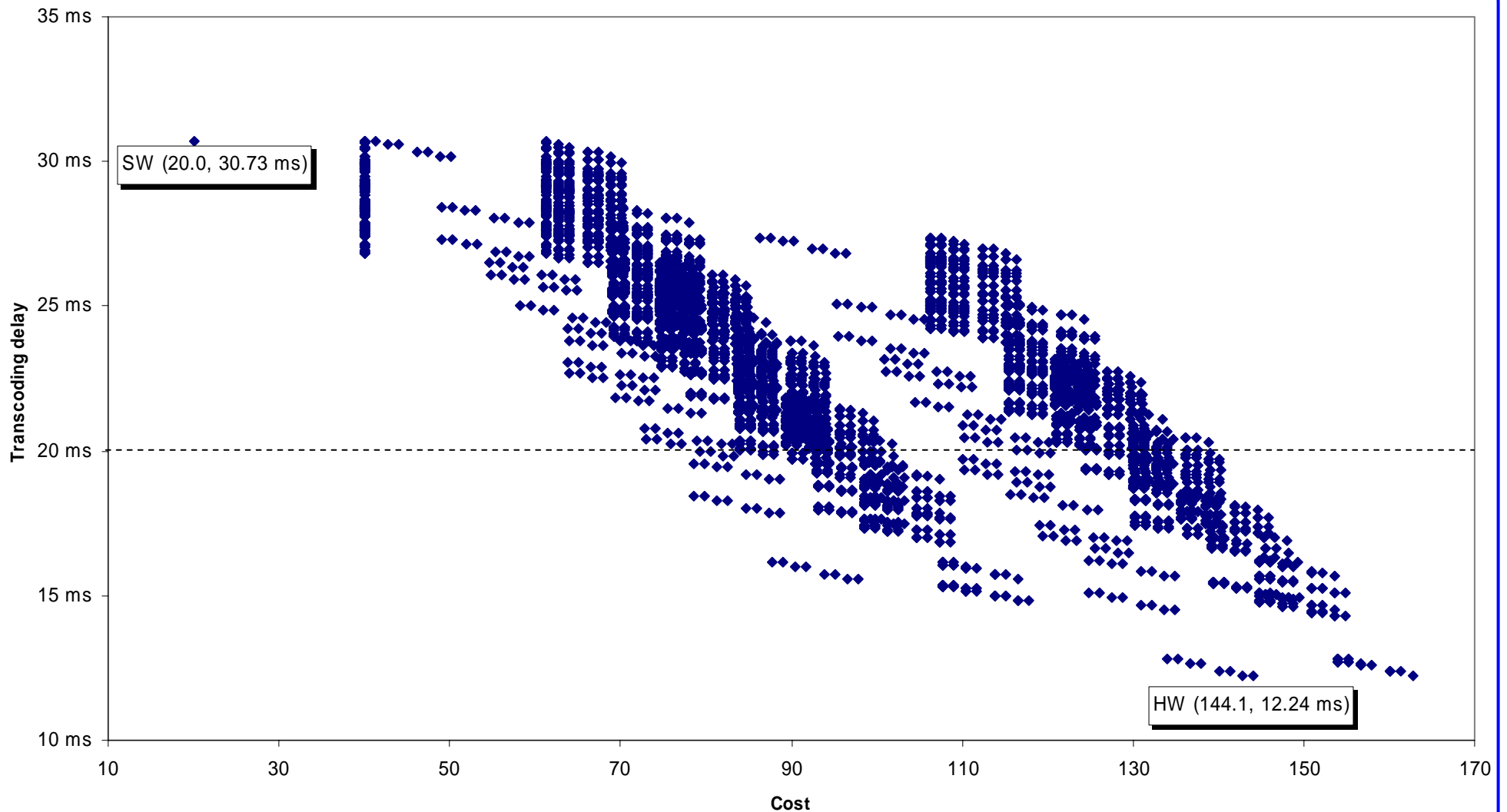
\forall processor $p \in P : p = \langle B_p, V_p, C_p, R_p \rangle$

\forall memory $m \in M : m = \langle V_m, A_m \rangle$

\forall IP $ip \in IP : ip = \langle B_{ip}, V_{ip}, A_{ip} \rangle$



Vocoder PE Exploration



- Mapping of 8 top-level encoder behaviors onto ColdFire + DSP + HW
- 85:04h for 6561 alternatives (1.7s simulation + 3s refinement each)
- 100% fidelity

Vocoder OS Exploration

	Modeling		Simulation	
	Lines of code	Simulation Time	Context switches	Transcoding delay
Partitioned	12,601	16.7 s	0	9.37 ms
Round-robin	13,920	18.1 s	64	10.9 ms
Decoder > Encoder	13,939	17.8 s	2	10.2 ms
Encoder > Decoder	13,939	17.8 s	8	11.3 ms
Implementation	~ 115,500	~ 5 h	2	10.7 ms

- **Modeling effort**

- Automatic scheduling refinement: seconds
- Manual scheduling refinement: < 1 h
 - 104 lines of code added / changed (< 1%)
- Implementation: weeks



Outline

- Introduction
- Design methodology
- Computation design
- **Communication design**
- Design environment
- Experimental results
- Summary and conclusion



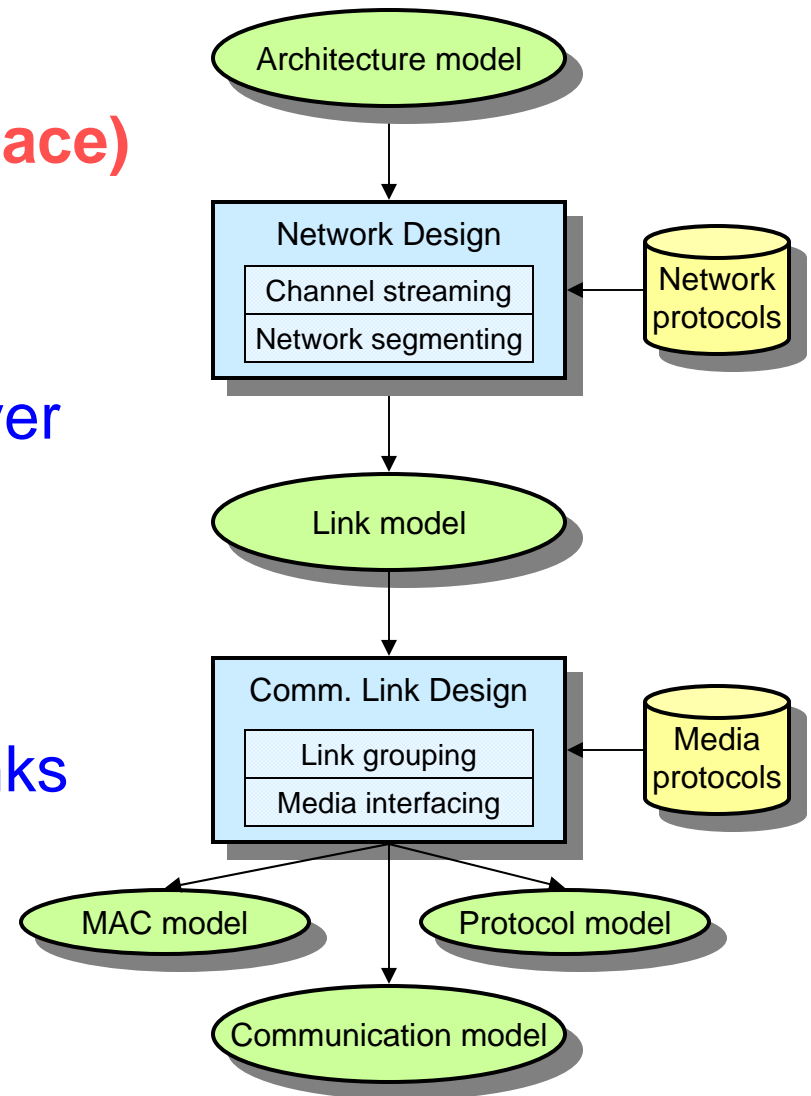
Communication Design Flow

- **Network design (structure / space)**

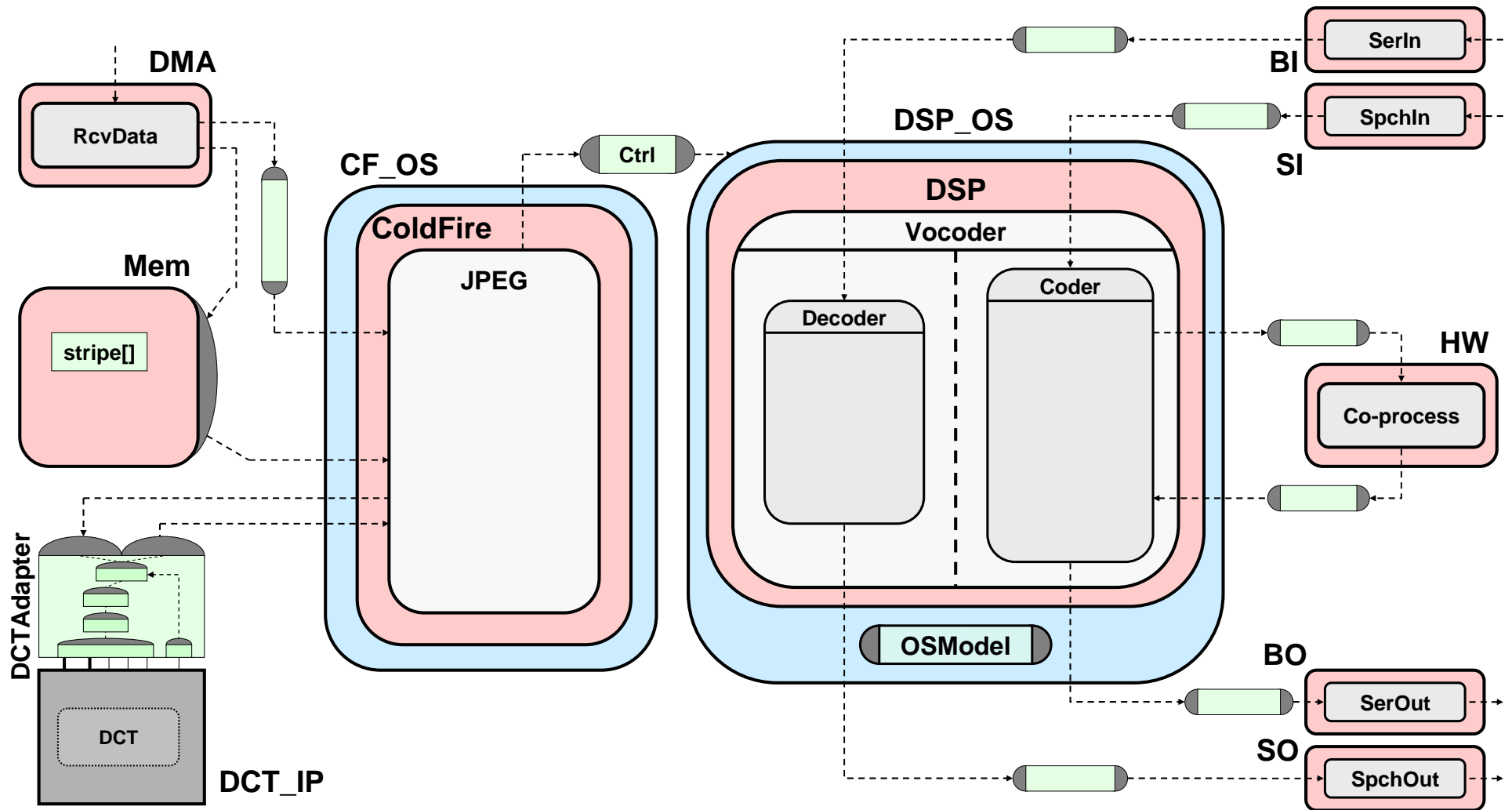
- Define network topology
- Merge channels into streams
- Route end-to-end streams over point-to-point network links

- **Link design (order / time)**

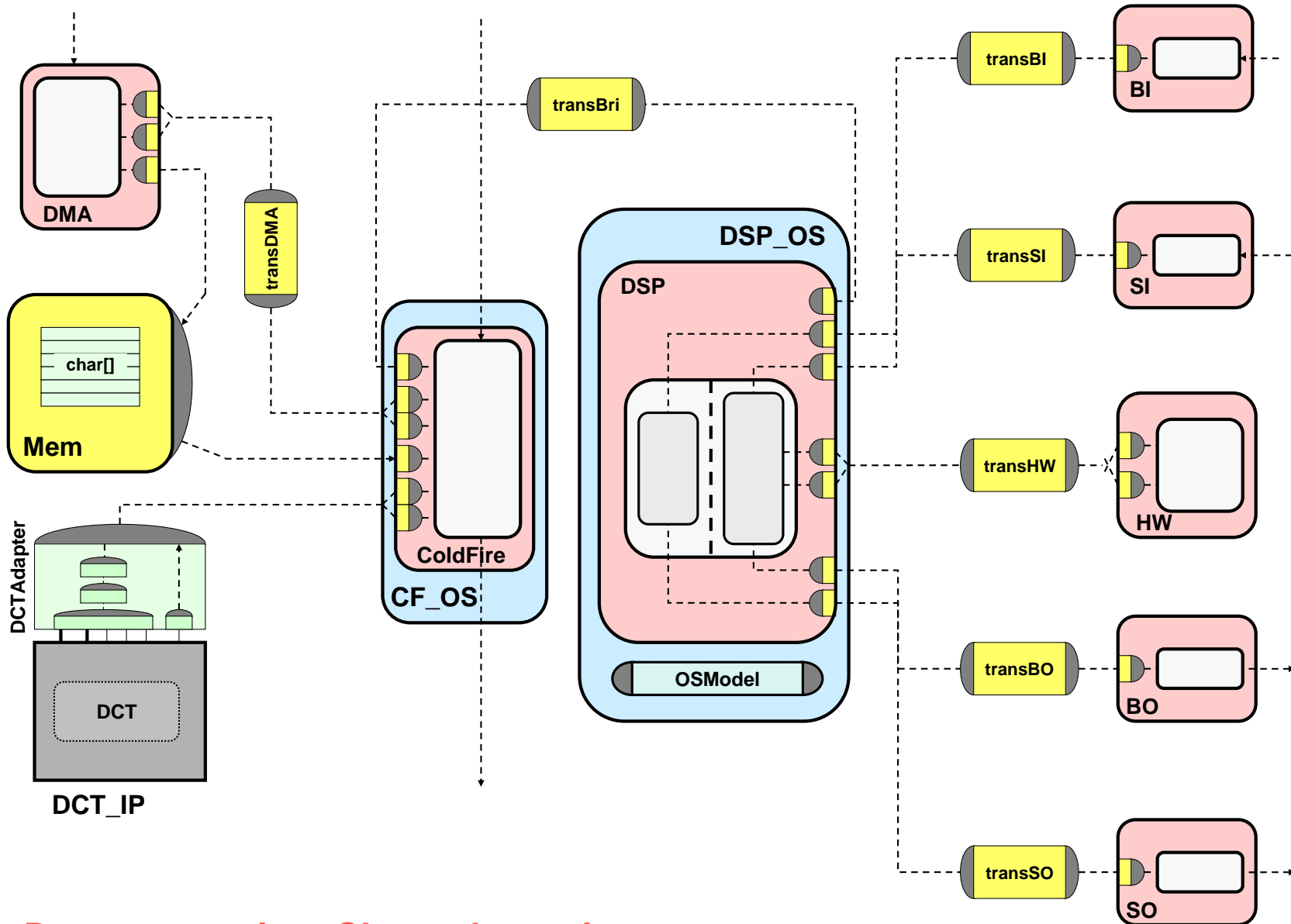
- Group logical into physical links
- Implement links over shared media, protocols, wires



Architecture Model Example

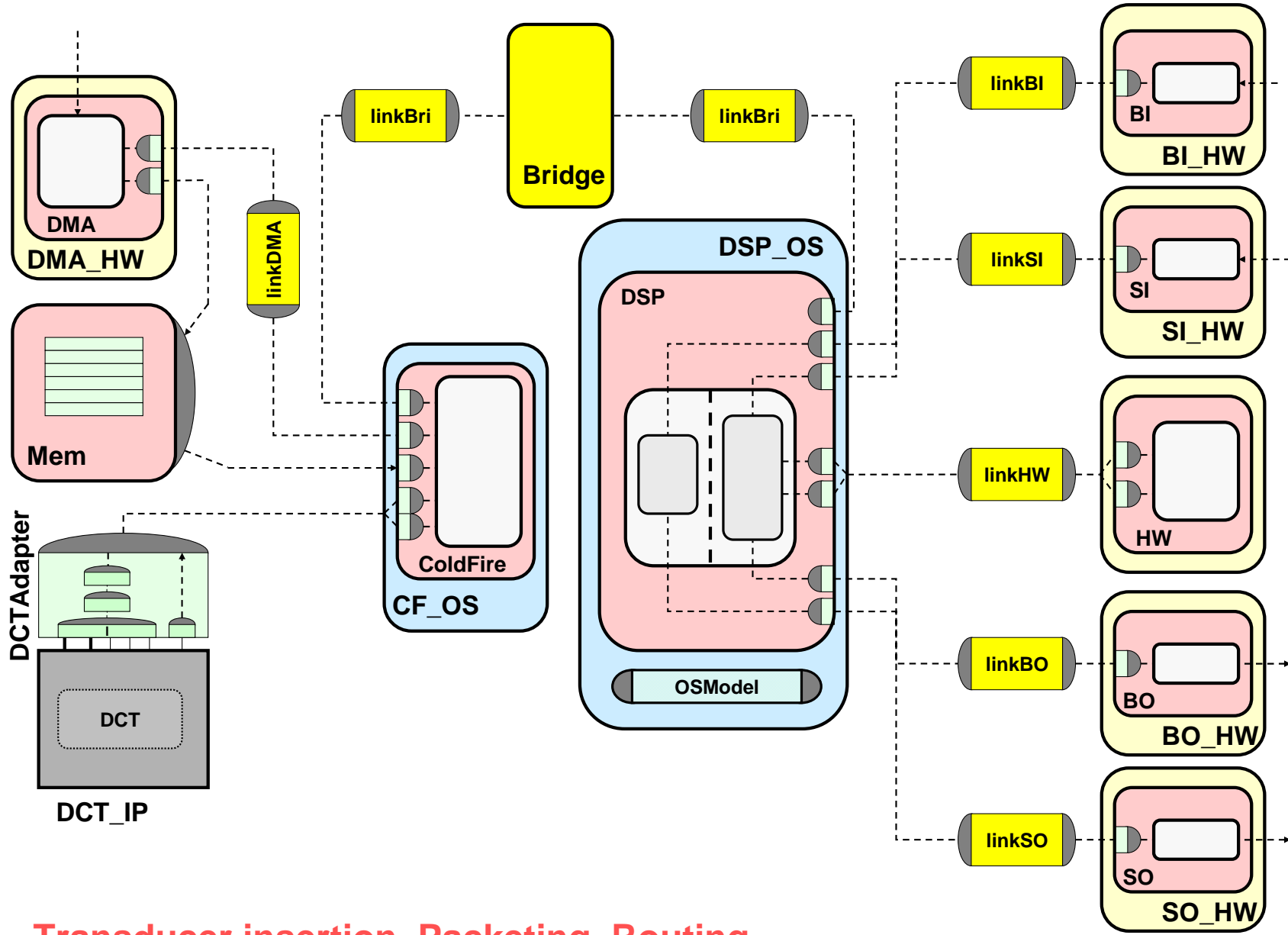


Transport Model Example



➤ Data conversion, Channel merging

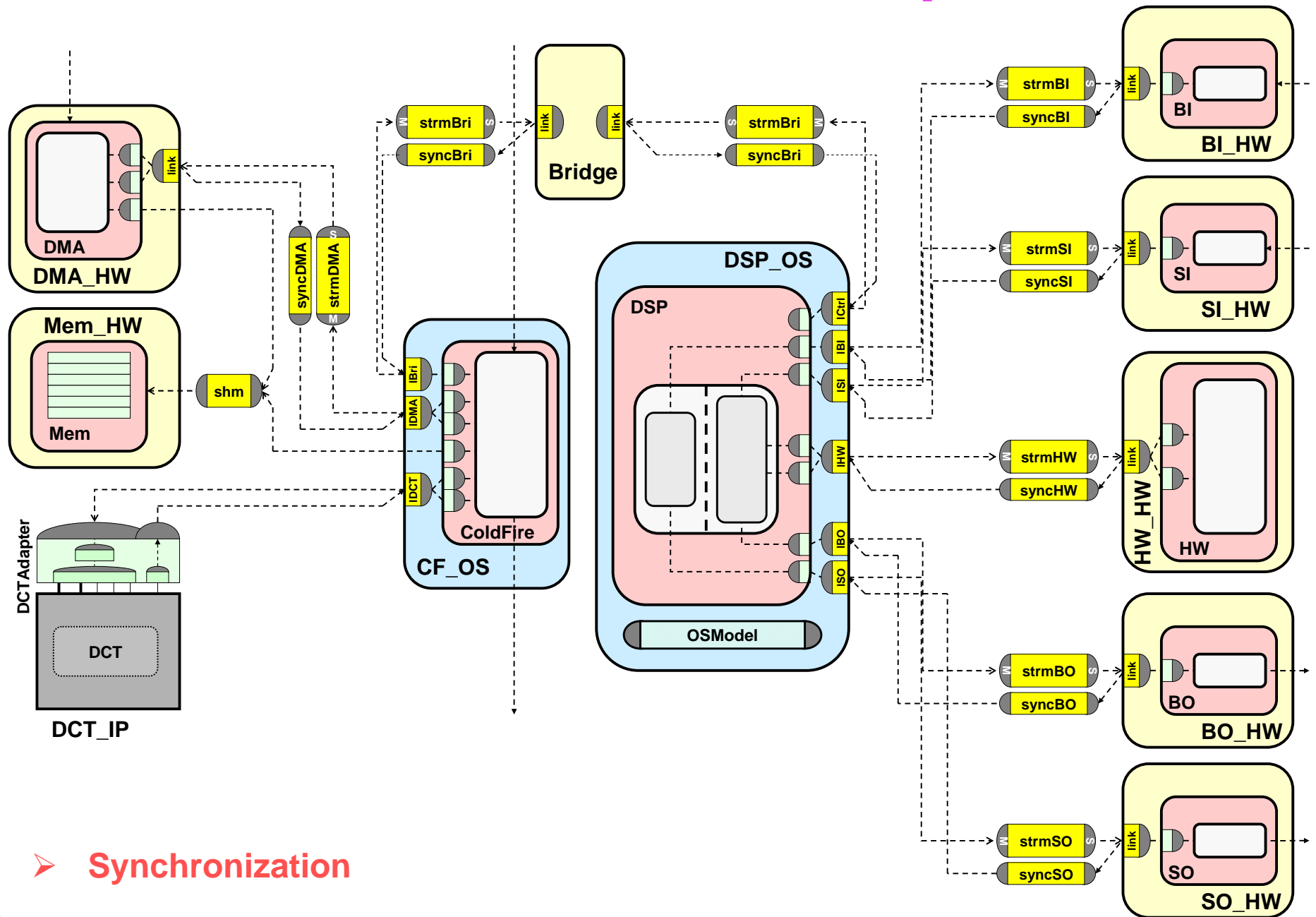
Link Model Example



➤ **Transducer insertion, Packeting, Routing**

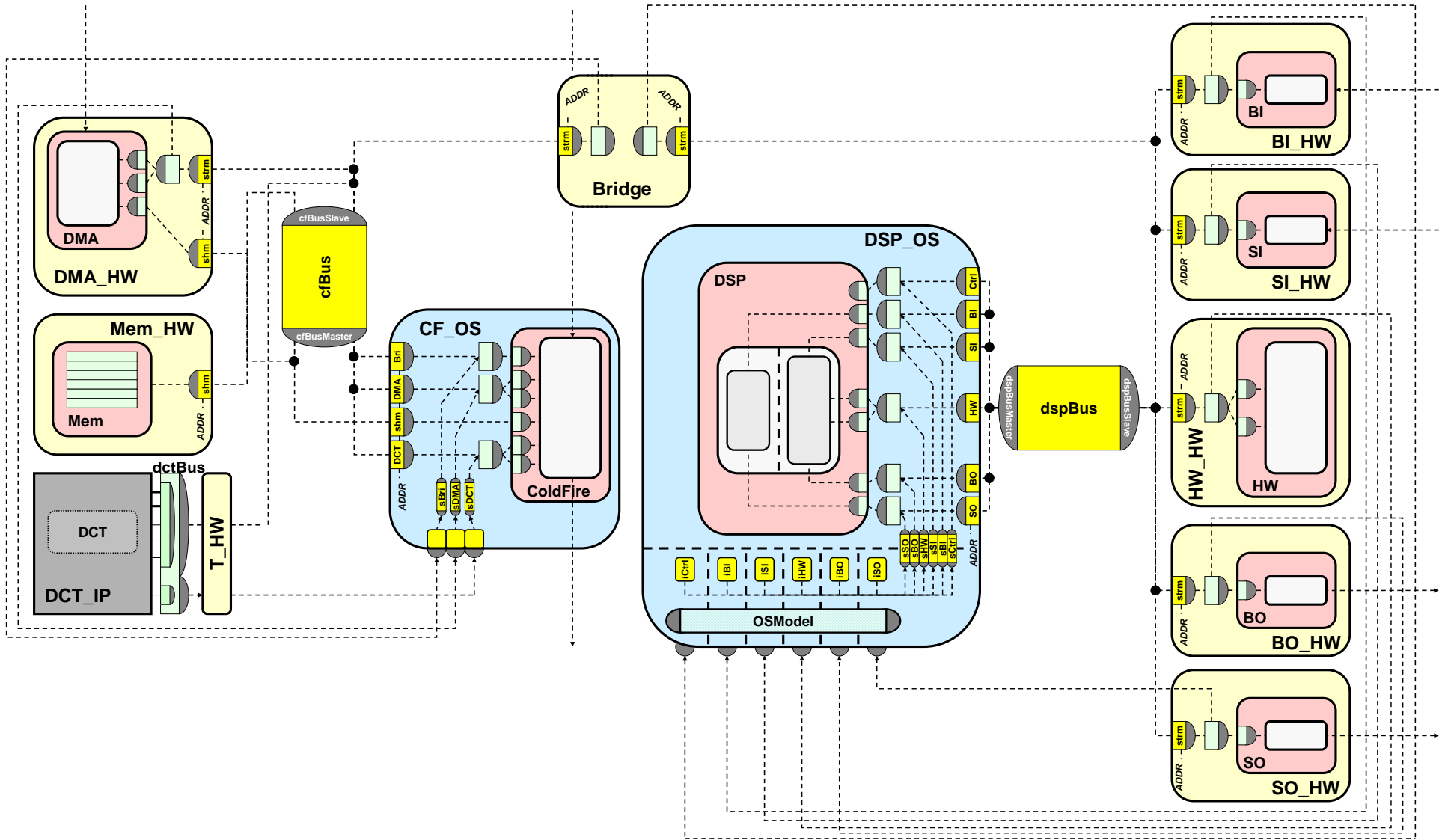


Stream Model Example



➤ Synchronization

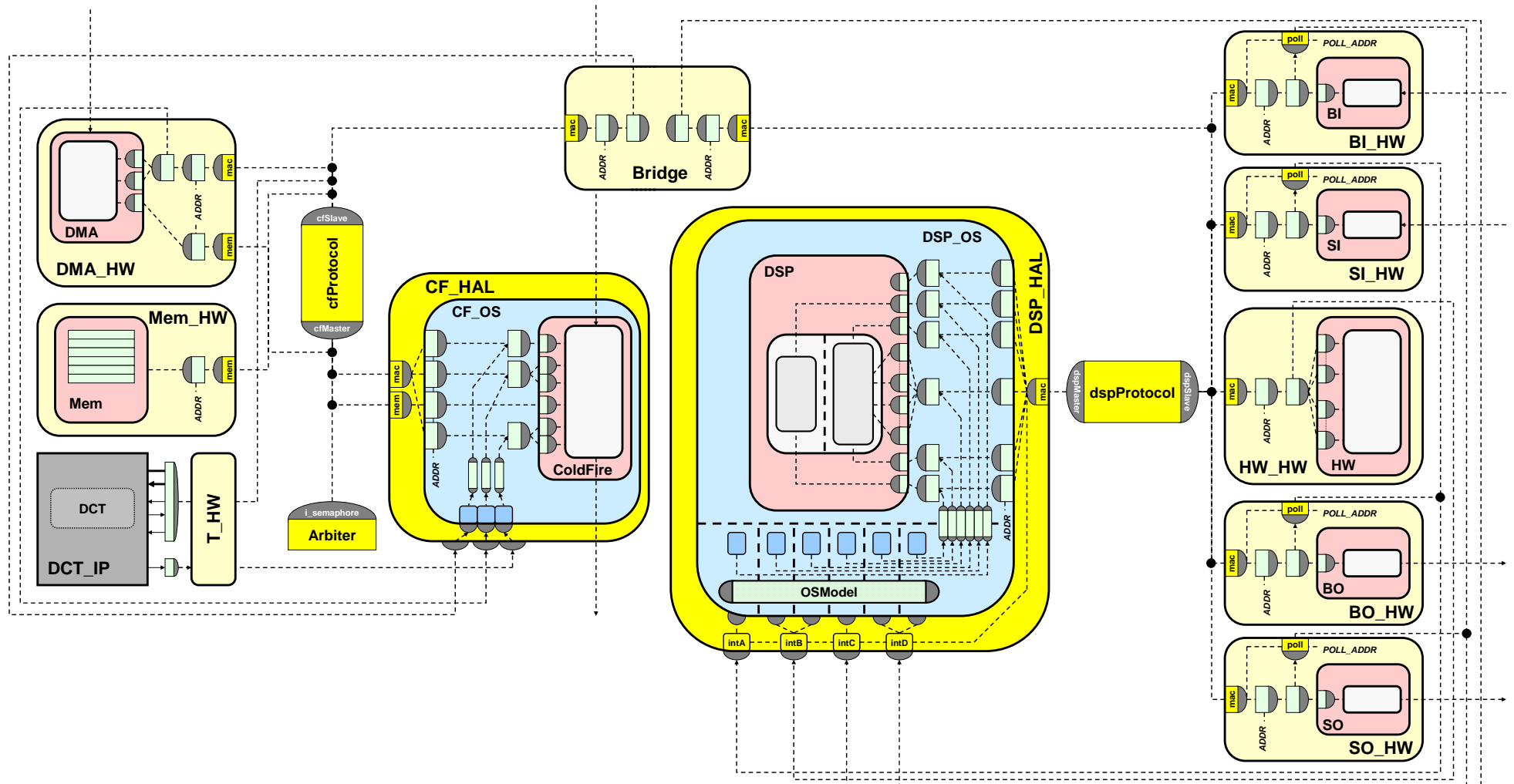
Media Access Model Example



➤ Multiplexing, Addressing, Interrupt tasks

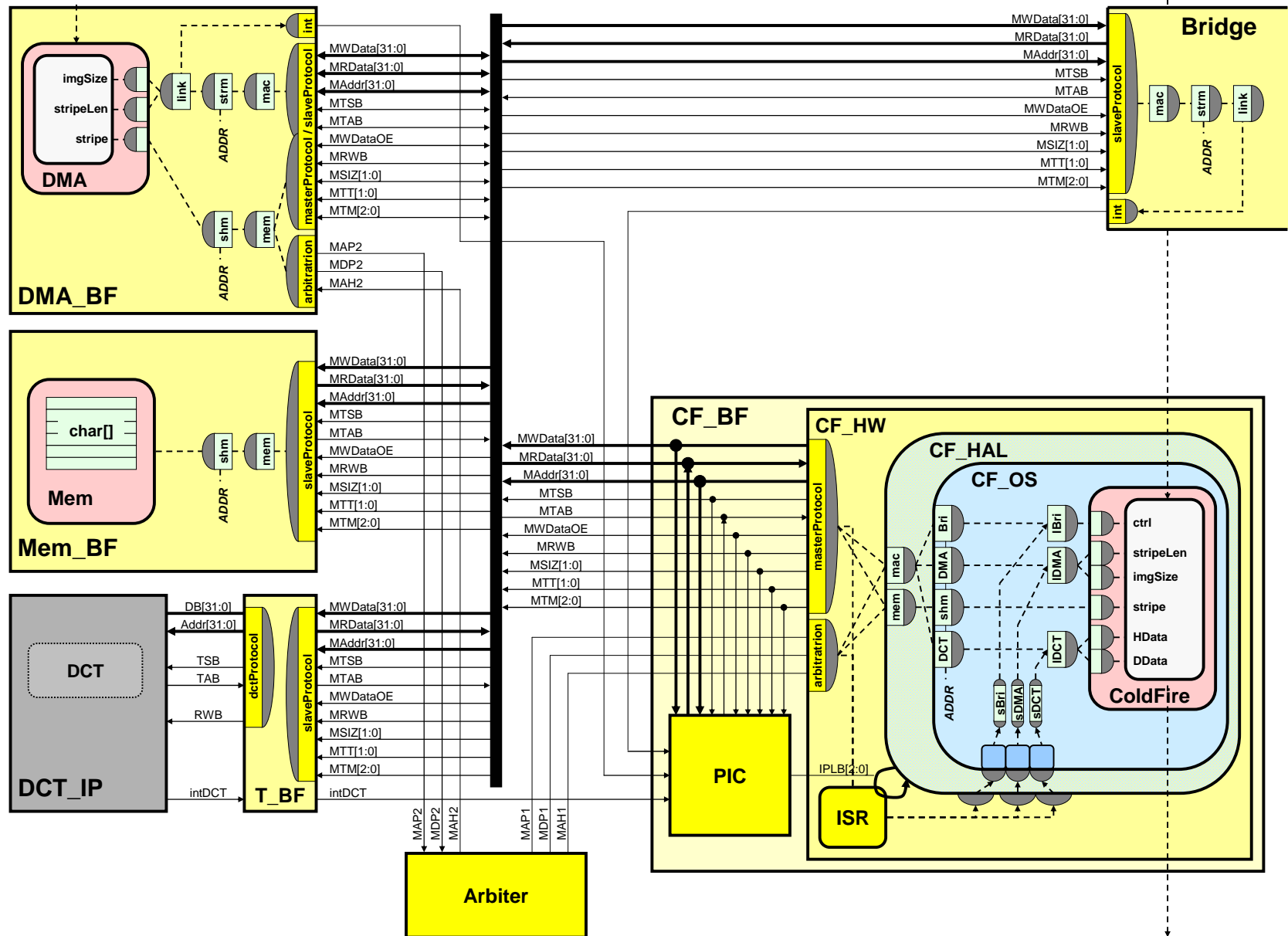


Protocol Model Example



- Hardware abstraction layer (HAL), Arbitration, Data slicing, Interrupt handling

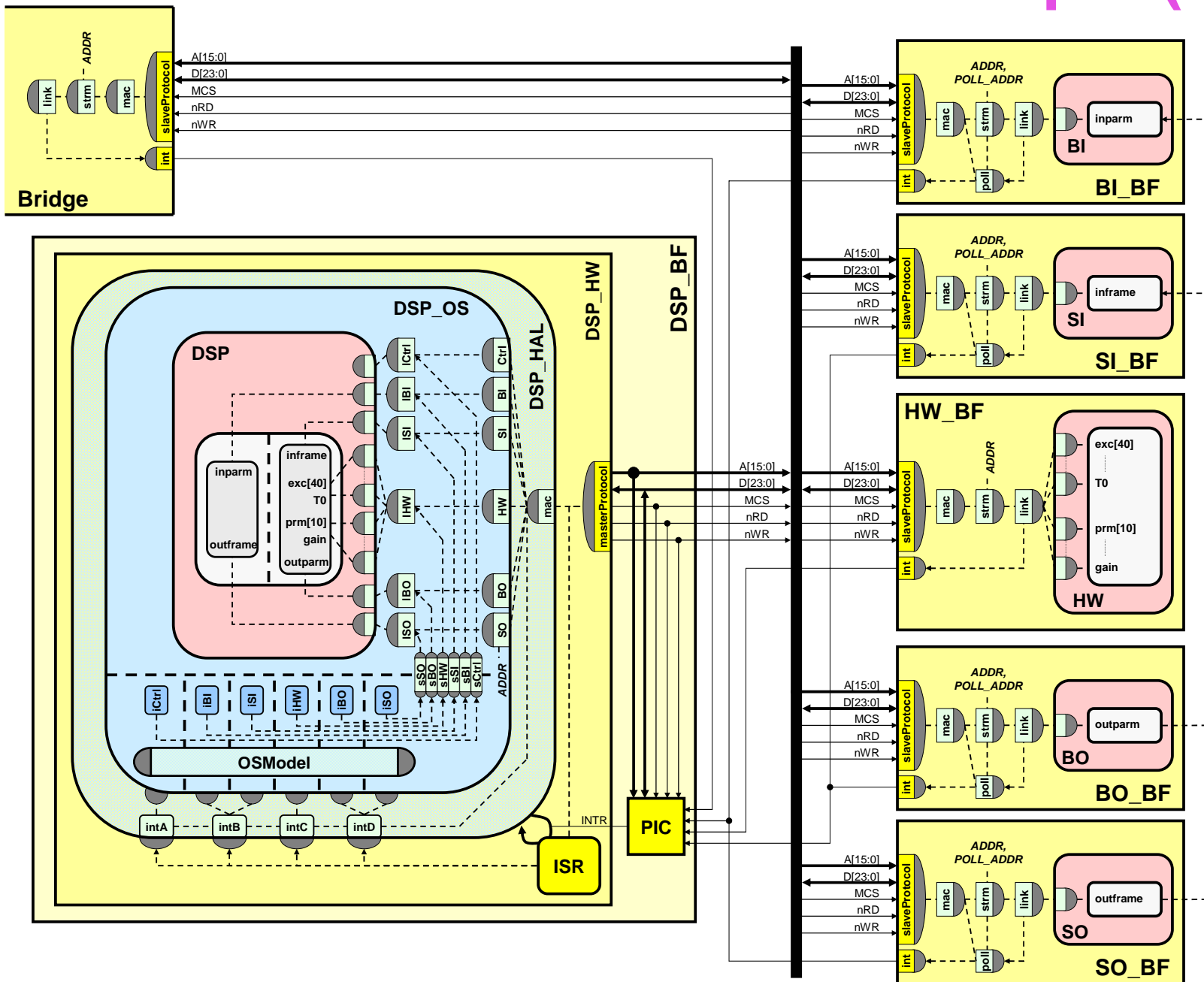
Communication Model Example (1)



➤ Hardware layer, Protocol insertion, Interrupt routing



Communication Model Example (2)



Communication Model

- **System architecture**
 - Computation & communication structure
 - Timed, bus-functional

Communication = $\langle PE \cup CE, W, c \rangle$

$PE = P \cup IP \cup M$: set of processing elements

$CE = T \cup A \cup IC$: set of communication elements

W : set of bus wires

$c : \bigcup_{p \in (PE \cup CE)} O_p \mapsto W$ port mapping function

\forall bus - functional processor $p \in P : p = \langle B_p, V_p, C_p, D_p, O_p, R_p \rangle$

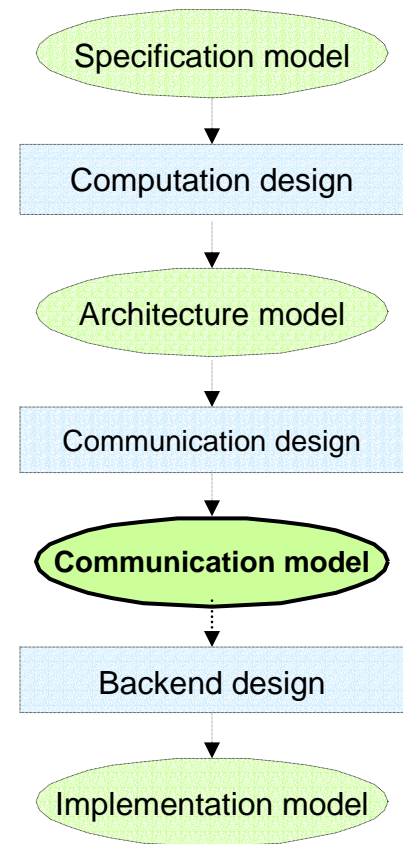
B_p : set of behaviors

C_p : set of local channels

D_p : set of bus drivers

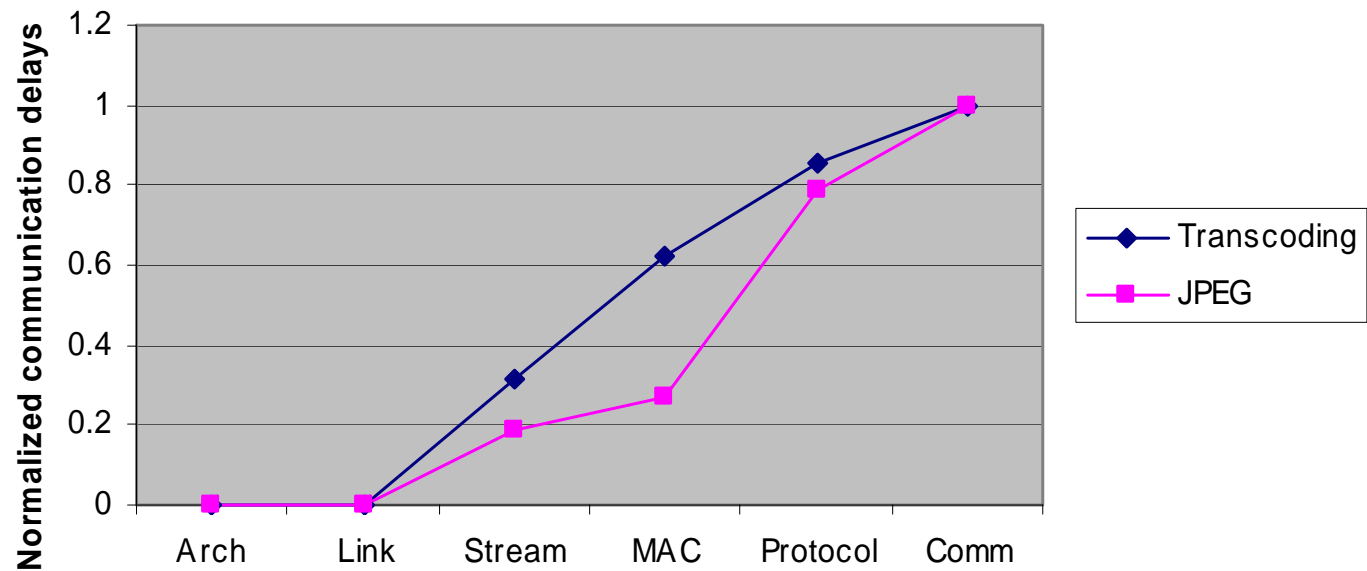
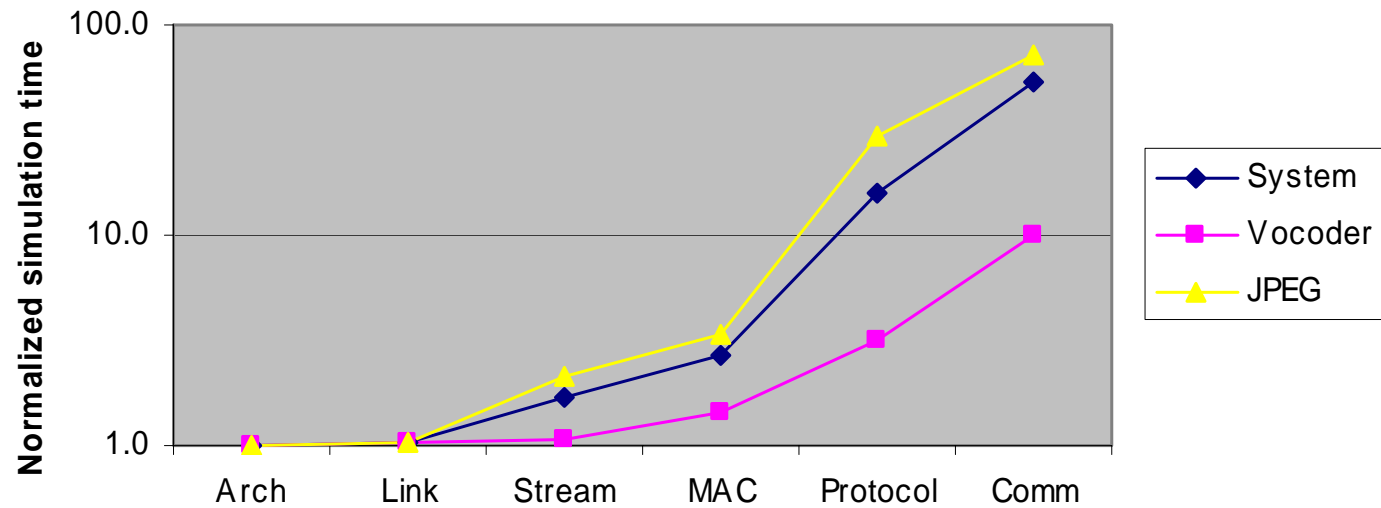
O_p : set of PE ports

$R_p \subseteq B_p \times (C_p \cup D_p)$: local connectivity relation



Communication Modeling

- Simulation overhead vs. accuracy

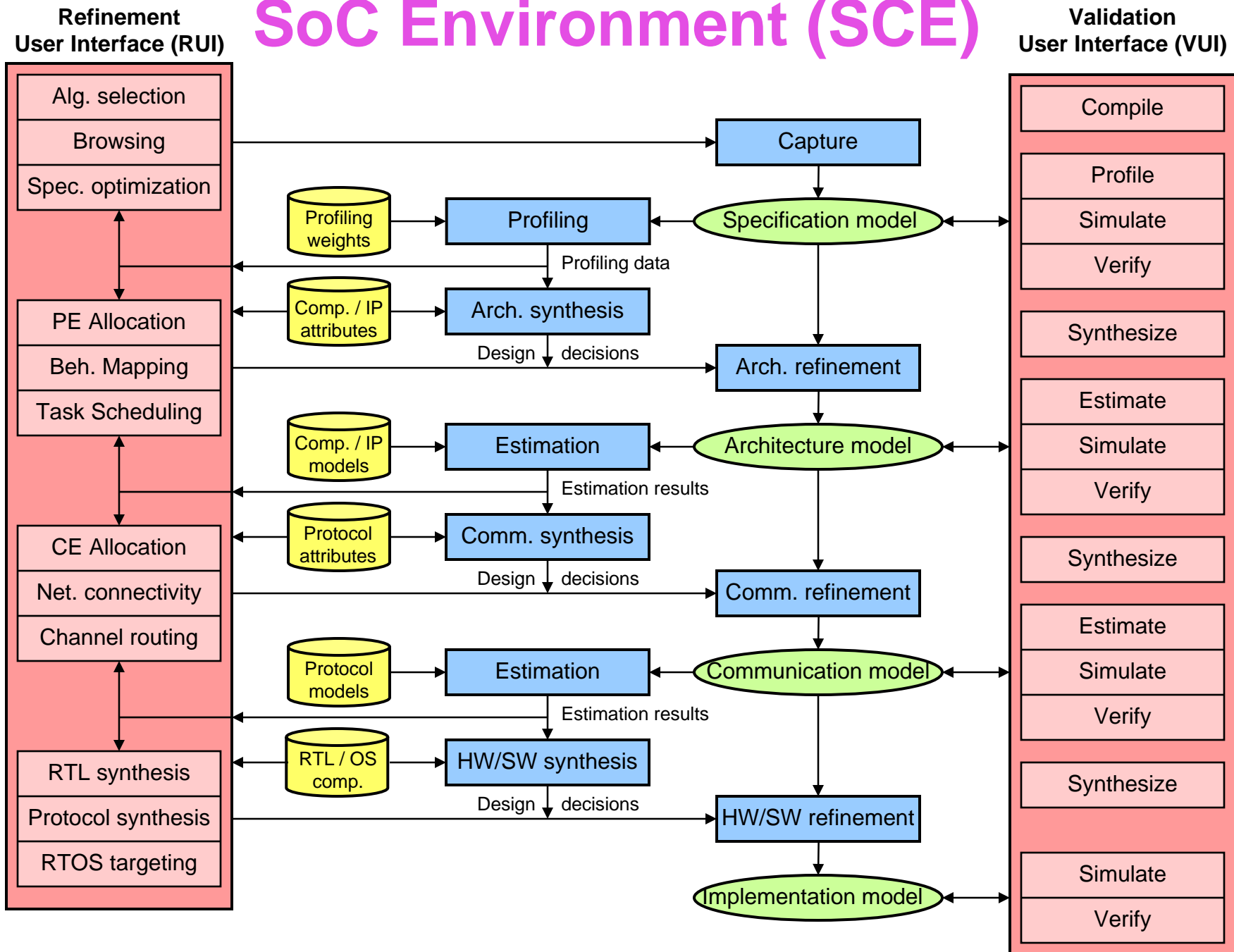


Outline

- Introduction
- Design methodology
- Computation design
- Communication design
- **Design environment**
- Experimental results
- Summary and conclusion



SoC Environment (SCE)

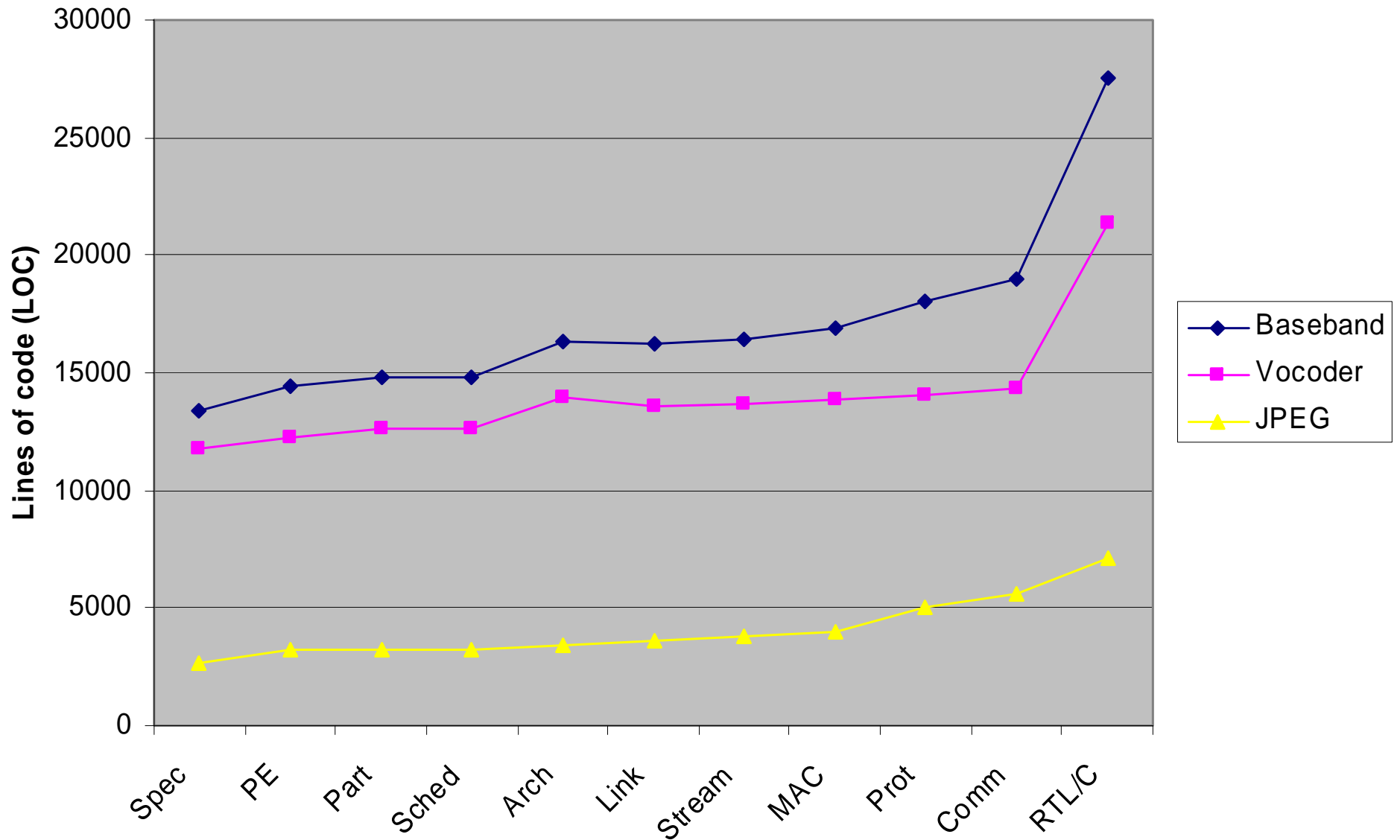


Outline

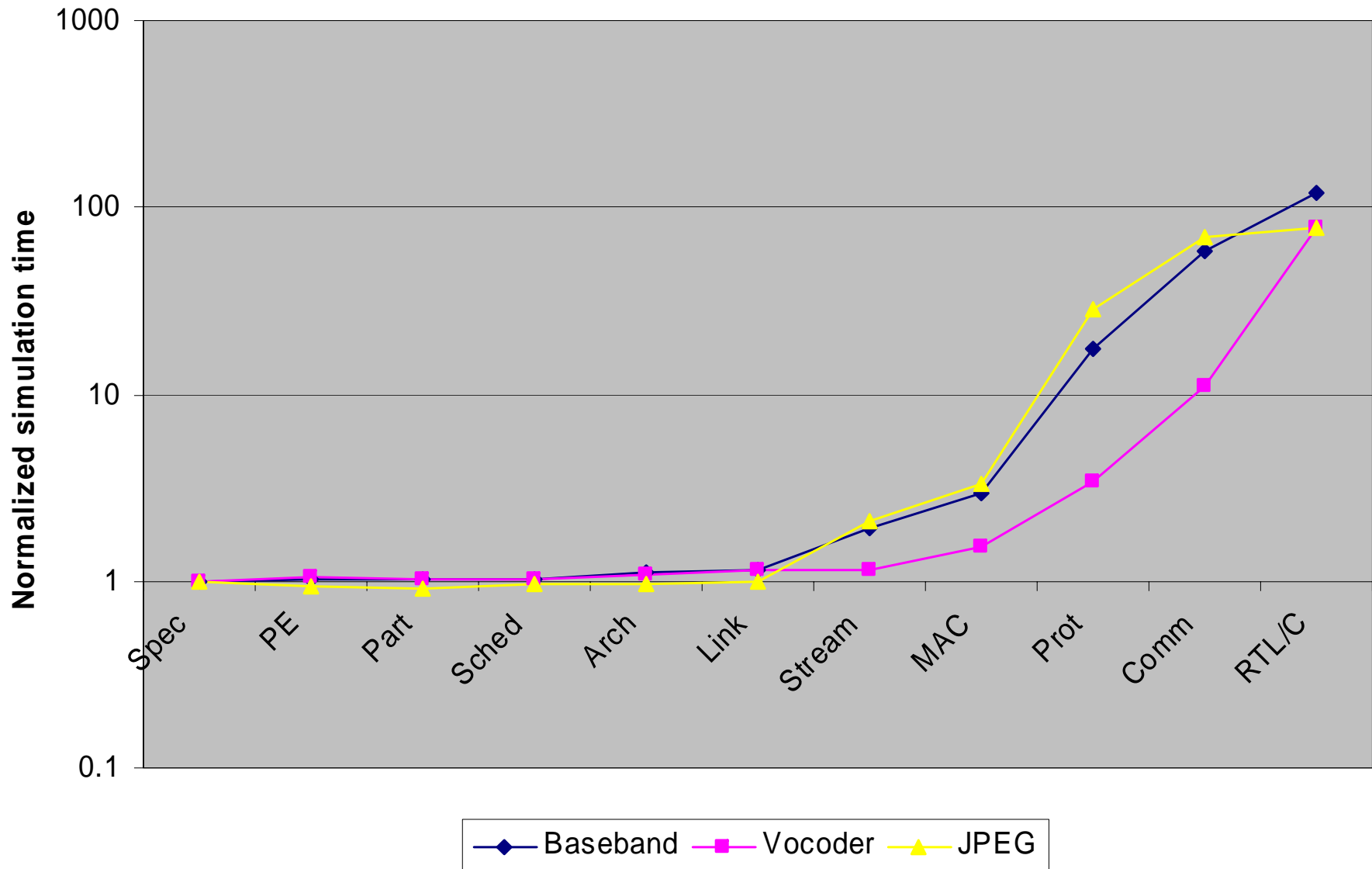
- Introduction
- Design methodology
- Computation design
- Communication design
- Design environment
- **Experimental results**
- Summary and conclusion



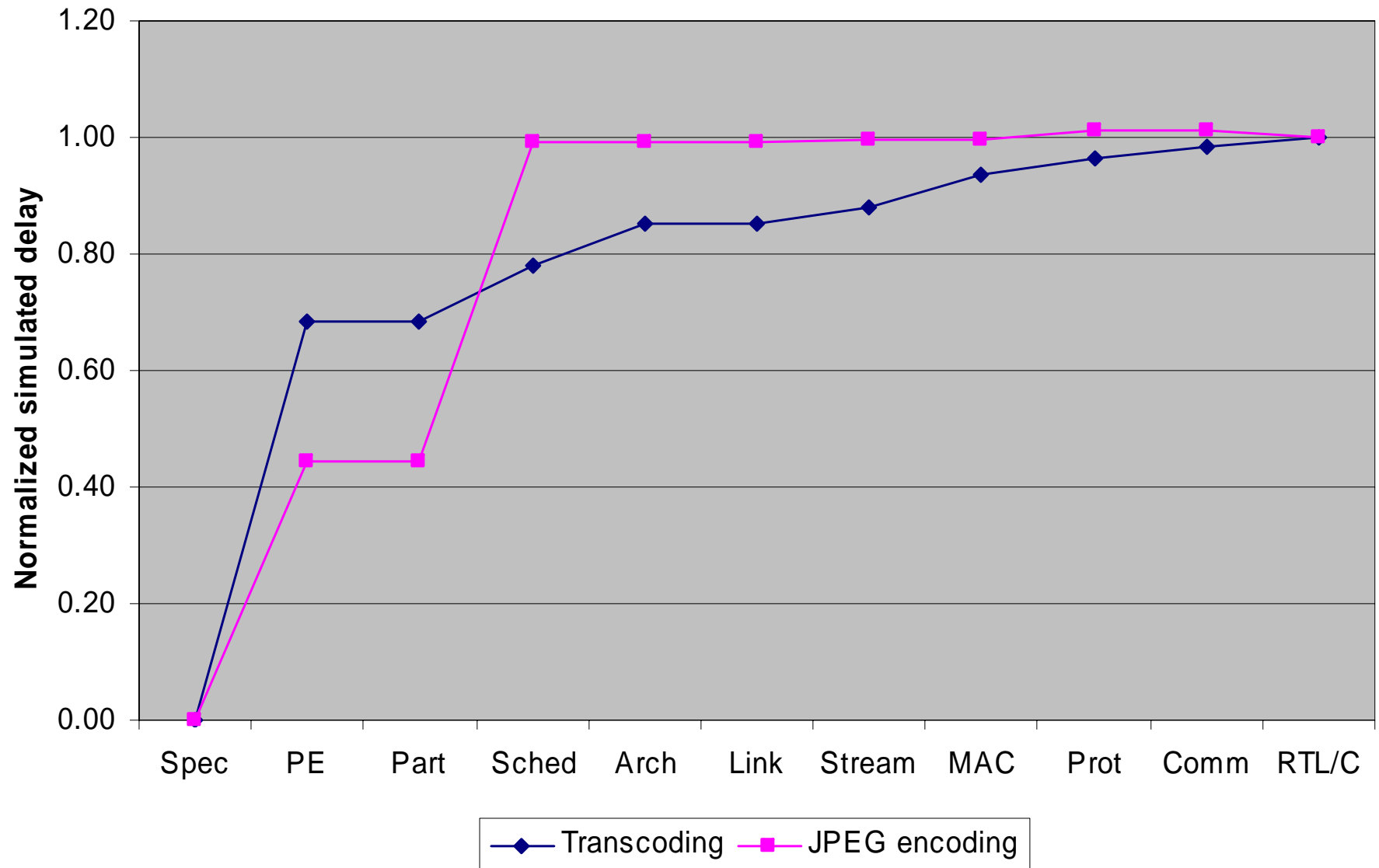
Model Complexities



Simulation Runtimes



Model Accuracies



Outline

- Introduction
- Computation design
- Communication design
- Design environment
- Experimental results
- **Summary and conclusion**



Contributions

- **Systematic, structured, well-defined system design flow**
 - Specification to implementation
 - Computation, communication, backend design tasks
 - Support for realistic applications, target architectures
- **Defined abstraction levels, models**
 - PE, memory, IP modeling for computation abstraction
 - OS model for dynamic scheduling abstraction
 - Communication abstractions at several levels
- **Defined design steps**
 - Design decisions + model transformations
- **Identified intermediate models for exploration**
 - Reliable feedback about critical issues at early stages
- **Defined interactive system design framework**
 - Tool flow, databases, architecture, interfaces
 - Graphical user interfaces for decision entry + model visualization
- **Productivity gains**
 - Automation of model refinement and decision making
 - Rapid, early design space exploration



Selected Publications

- **Books**

- A. Gerstlauer, R. Dömer, J. Peng, D. Gajski, “**System Design: A Practical Guide with SpecC**”, Kluwer, 2001.
- D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, S. Zhao, “**SpecC: Specification Language and Methodology**”, Kluwer, 2000.

- **Book chapters**

- A. Gerstlauer, H. Yu, D. Gajski, “**RTOS Modeling for System-Level Design**”, *Embedded Software for SoC*, Kluwer, 2003.
- A. Rettberg, F. Rammig, A. Gerstlauer, D. Gajski, W. Hardt, B. Kleinjohann, “**The Specification Language SpecC within the PARADISE Design Environment**”, *Architecture and Design of Distributed Embedded Systems*, Kluwer, 2001.

- **Conference Papers**

- L. Cai, A. Gerstlauer, D. Gajski, “**Retargetable Profiling for Rapid, Early System-Level Design Space Exploration**”, *DAC* 2004.
- A. Gerstlauer, H. Yu, D. Gajski, “**RTOS Modeling for System-Level Design**”, *DATE* 2003.
- A. Gerstlauer, D. Gajski, “**System-Level Abstraction Semantics**”, *ISSS* 2002.
- W. Mueller, R. Dömer, A. Gerstlauer, “**The Formal Execution Semantics of SpecC**”, *ISSS* 2002.
- A. Gerstlauer, S. Zhao, D. Gajski, A. Horak, “**SpecC System-Level Design Methodology Applied to the Design of a GSM Vocoder**”, *SASIMI* 2000.



Backup Slides



Behavior Partitioning

- **Design decisions**
 - PE allocation and selection
 - $PE = \text{set of } (name, type) \text{ tuples}$
 - Behavior mapping
 - mapping function $m_b : B \mapsto PE$
- **Model transformations**
 - PE layer
 - Additional layer of behavior hierarchy representing PEs
 - Grouping
 - Group behaviors under PEs according to mapping
 - Synchronization
 - Insert synchronization to preserve transition semantics
 - Timing refinement
 - Annotate behaviors with estimated execution delays



Variable Partitioning

- **Design decisions**
 - Memory allocation and selection
 - $M = \text{set of } (name, type) \text{ tuples}$
 - Variable mapping
 - mapping function $m_v : V_s \subseteq V \mapsto M$
- **Model transformations**
 - Memory layer
 - Insert behaviors representing shared memories
 - Grouping
 - Group global variables under shared memories according to mapping
 - Message passing
 - Distribute unmapped global variables, insert message passing
 - Memory accesses
 - Create memory interface, update shared variables accesses



Static Scheduling

- **Design decisions**

- Behavior order

- $\forall b \in B_s, B_s \subseteq B$: schedule S_b = totally ordered set of children

- **Model transformations**

- Serialization

- Sequentialize concurrent behavior compositions

- Flattening

- Move children into parent behavior as requested

- Reordering

- Arrange behaviors in selected execution order



Dynamic Scheduling

- **Design decisions**
 - Scheduling algorithm selection
 - OS selection function $os : PE \mapsto$ set of algorithms OS
 - Task priority assignment
 - task priority function $p : B_t \subseteq B \mapsto Z^+$
- **Model transformations**
 - OS layer
 - Additional OS layer around programmable PEs
 - Insert abstract OS model for selected scheduling strategy
 - Task creation
 - Turn concurrent behaviors into OS tasks
 - Task refinement
 - Replace delay primitives
 - Synchronization refinement
 - Replace event handling primitives



Channel Streaming

- **Design decisions**

- Network byte layout

- layout function l over data types $d \in D$:

$$D \mapsto \mathbb{Z}^* \times \mathbb{Z}^* \times \{\text{b}, \text{l}\}, l(d) = (\text{size}, \text{alignment}, \text{endianess})$$

- Channel merging

- merging function m_c : set of channels $C_s \mapsto$ set of streams S

- **Model transformations**

- Presentation layer

- Conversion of abstract data types into network bytes
 - Memory data byte layout

- Session layer

- Merge channels into message streams



Network Segmenting

- **Design decisions**

- Transducer allocation

- $T = \text{set of } (name, type) \text{ tuples}$

- Channel routing & packeting

- $\forall \text{ stream } s \in S : \text{route } R_s = \text{ordered set of hops } r \in (PE \cup T)$

- packet function $p : S \mapsto \mathbb{Z}^+$, $p(s) = \text{packet size}$

- **Model transformations**

- Transport layer

- Splitting of message streams into packet streams

- Flow control, error correction

- Network layer

- Insert transducers and links

- Routing of packets over links between PEs and transducers



Link Grouping

- **Design decisions**

- **Bus/protocol allocation**

- $BUS = \text{set of } (name, type) \text{ tuples}$

- **Station connectivity**

- connectivity relation $N \subseteq (PE \cup T) \times BUS$

- connection type function $if : N \mapsto \text{interface types } IF$

- **Link parameters**

- $\forall l \in \text{links } L, \text{ parameter function } m :$

- $L \mapsto N \times N \times Z^+ \times Z^*, m(l) = (src, dst, addr, intr)$

- **Model transformations**

- **Link layer**

- Splitting of links into control and data transactions

- **Stream layer**

- Multiplexing of data over media transaction via media addressing

- Implementation of interrupt tasks for control transactions



Media Interfacing

- **Design decisions**

- Arbiter allocation, bus master priority assignment

- $A = \text{set of } (name, type) \text{ tuples, connectivity: } A \mapsto BUS$
 - bus master priority function $a : MASTER \subseteq N \mapsto Z^*$

- Interrupt controller allocation, bus slave interrupt assgn.

- $IC = \text{set of } (name, type) \text{ tuples, connectivity } IC \mapsto PE$
 - bus slave interrupt function $i : SLAVE \subseteq N \mapsto \text{set of interrupts}$

- **Model transformations**

- Media access layer, hardware abstraction layer (HAL)

- Slicing of data packets into media words/frames, media arbitration
 - Implementation of interrupt handlers, slave polling

- Protocol layer, hardware layer

- Protocol transaction timing for sampling/driving wires
 - Insert programmable PE interrupt hardware model
 - Insert, connect arbiters and interrupt controllers

