Embedded System Design

Modeling, Synthesis, Verification

Daniel D. Gajski, Samar Abdi, Andreas Gerstlauer, Gunar Schirner

Chapter 1: Introduction

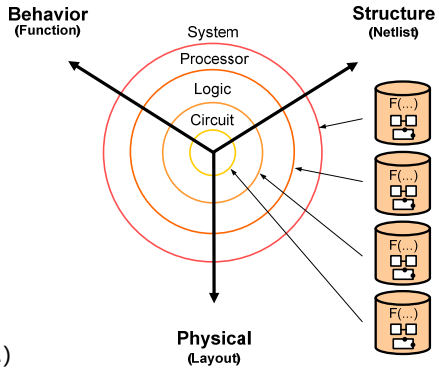
7/8/2009

Overview

- **Abstraction layers**
- **Processor level**
- **System level**
- **Present system design**
  - Missing semantics
  - Modeling algebra
  - Specify-Explore-Refine
- **System models**
- **System platforms**
- **Tools and environments**
- **Conclusion**

Y Chart

- **3 design views**
  - Behavior (functionality)
  - Structure (netlist)
  - Physical (layout)
- **4 abstraction levels**
  - Circuit
  - Logic
  - Processor
  - System
- **5 component libraries**
  - Transistor
  - Logic (standard cells)
  - RTL (ALUs, RF, Memories,…)
  - Processor (standard, custom)
  - System (multi-cores with NoCs)



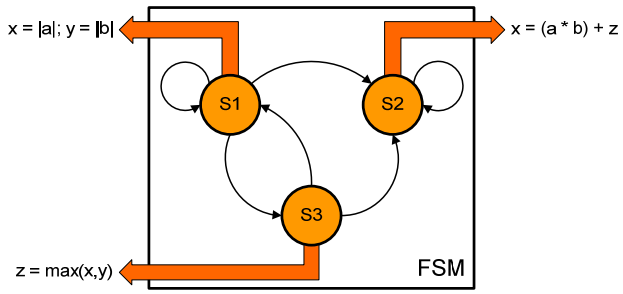
Overview

- ✓ Abstraction layers
- **Processor level**
- System level
- Present system design
  - Missing semantics
  - Modeling algebra
  - Specify-Explore-Refine
- System models
- System platforms
- Tools and environments
- Conclusion

Processor Abstraction Level

- **Processor level generates system components**
  - Computation components
    - Standard processors
    - Custom processor
    - Custom functions
    - Controllers
    - Memories
  - Communication components
    - Arbiters
    - Bridges and transducers
    - Controllers (interrupt, memory, DMA,)
    - Interfaces
- **Processor-level synthesis**
  - Behavior specification (FSMD, CDFG, IS)
  - Component structure
  - Synthesis on processor level

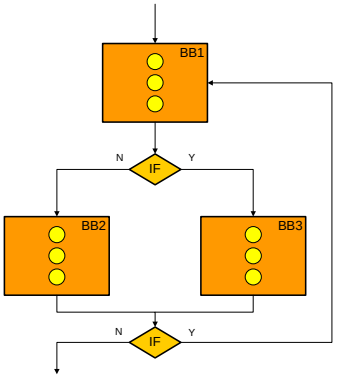
FSMD Model



- **FSM**
  - Set of states and transitions
  - Transition executed under conditional statements of input variables
- **FSMD**
  - FSM + Set of variable statements executed in each state
- **No timing, could be assumed**

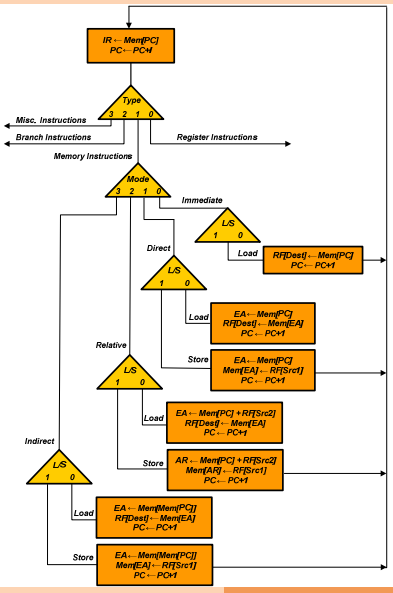
CDFG Model

- **Control/Data Flow Graph (CDFG)**
  - IF statements and LOOP statements
  - Basic Blocks (BBs)
- **Reflects a programming language syntax such as C**
- **Control dependences between IFs and BBs**
- **Data dependences inside BBs**
- **No data dependences outside BBs**



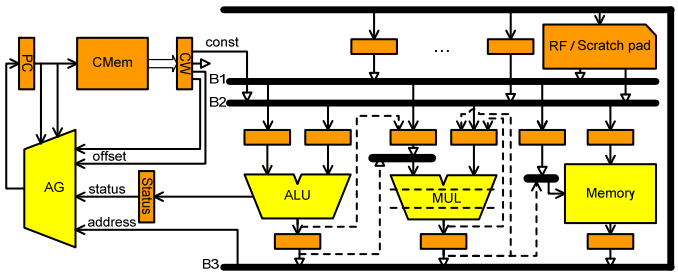
Instruction Set Flow Chart

- **Super-state FSMD**
  - Fetch, execute states
- **Each state several assignments**
  - Variable assignments
  - Register assignments
  - Memory loads and stores
- **Each state may need several clock cycles**

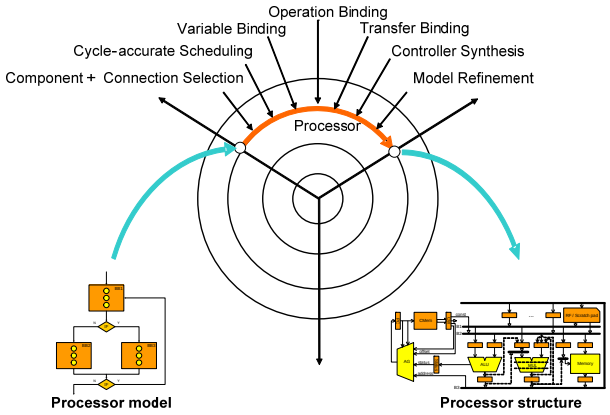


Processor Structural Model

- **Datapath components**
  - Storage (registers, RFs, Scratch pads, data memories)
  - Functional units (ALUs, multipliers, shifters, special functions)
  - Connection (buses, selectors, bridges)
- **Controller components**
  - Registers (PC, Status register, Control word or Instruction register)
  - Others (AG, Control memory or Program memory)
- **Processor structure**
  - Pipelining, chaining, multi-cycling, forwarding



Processor Level Synthesis



- **Several tasks: many different sequences possible**
  - **Different inputs, different libraries, different features, different output**
  - **Different tools, different metrics, different quality**

Overview

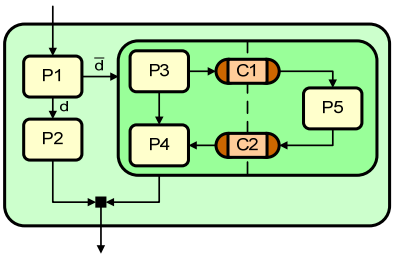
- ✓ Abstraction layers
- ✓ Processor level
- **System level**
- Present system design
  - Missing semantics
  - Modeling algebra
  - Specify-Explore-Refine
- System models
- System platforms
- Tools and environments
- Conclusion

System Abstraction Level

- **Design of multi-core systems**
  - Computation components
  - Communication components
- **Synthesis**
  - From behavior specification: MoCs
  - To system architecture
  - With system SW and HW
  - Synthesis on system level

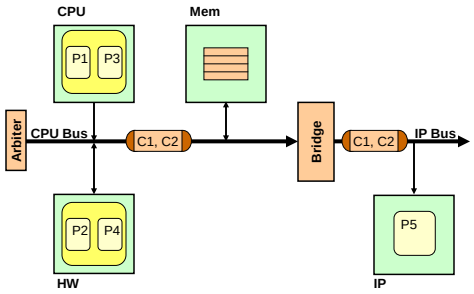
System Behavioral Model

- **Many different MoCs**
  - Process-based models
  - State-based models
- **Universal model**
  - Processes
    - Sequential/parallel, hierarchical
  - Channels
    - Message-passing

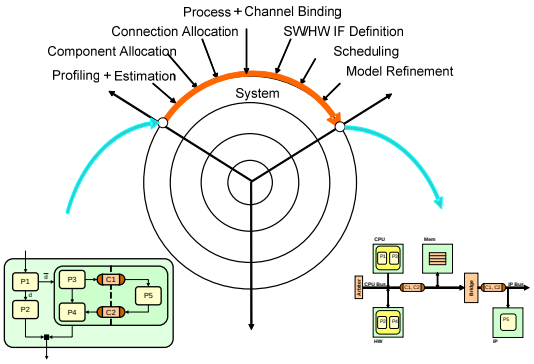


System Structural Model

- **Set of computational components**
  - Processors, IPs, custom HW components
  - Memories
- **Set of communication components**
  - Buses, bridges, arbiters, transducers, interfaces



System Synthesis



- **Several tasks: different sequences possible**
  - Different MoCs, different libraries, different features, different platforms
  - Different tools, different metrics, different quality

Overview

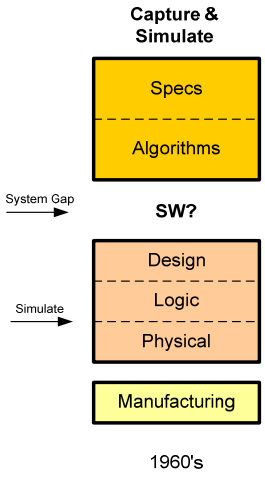
- ✓ Abstraction layers
- ✓ Processor level
- ✓ System level
- **Present system design**
  - Missing semantics
  - Modeling algebra
  - Specify-Explore-Refine
- System models
- System platforms
- Tools and environments
- Conclusion

Evolution of Design Flow

- **Three evolutionary design flows**
  - Capture and Simulate
    - Designers complete design and validate through simulation
  - Describe and Synthesize
    - Introduction of design synthesis
    - For given functionality, design structure generated by tools
  - Specify-Explore-Refine
    - System design flow extended to four levels
    - Large number of levels and metrics for validation
    - Design flow performed in several steps
    - Each step follows describe-and-synthesize concept

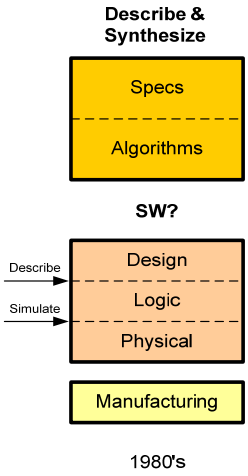
Capture-and-Simulate Design Flow

- **System designers generate**
  - Preliminary specification
  - Basic algorithms
  - No software design
- **HW designers generate**
  - Architecture, RTL, Logic
  - Logic-level netlist for simulation
  - Simulate and optimize
- **System gap between SW and HW**
- **Simulation at the end of design**
- **Manual design, no automation**



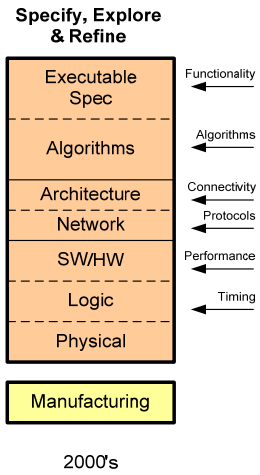
Describe-and-Synthesize Design Flow

- **System designers generate**
  - Preliminary specification
  - Basic algorithms
  - SW design after HW design
- **HW designers generate**
  - Architecture, RTL, Logic behavior
- **Logic synthesis tools generate logic**
  - Designers simulate and optimize
- **Simulation before and after synthesis**
- **Designers describe just functionality, tools synthesize structure**
- **System gap still persists**



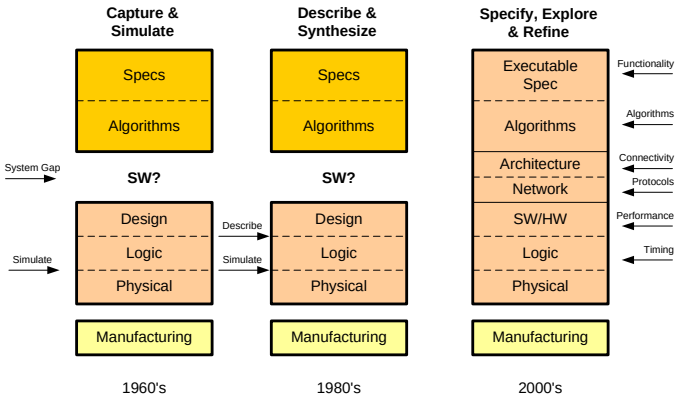
Specify-Explore-Refine Design Flow

- **Application designers generate**
  - Executable specification
  - Application algorithms
  - Platform model for application/platform optimization
- **System designers generate**
  - Detailed architecture and network
  - SW and HW components
  - Logic and layout
- **Many design levels and models**
- **Many metrics for validation**
- **Solution: Step-by-step strategy**
  - For each model explore design decisions
  - Refine model after exploration
  - Refined model = specification for the next level



Evolution of Design Flow

- **Concept of describe-and synthesize was not upgraded adequately to system level**

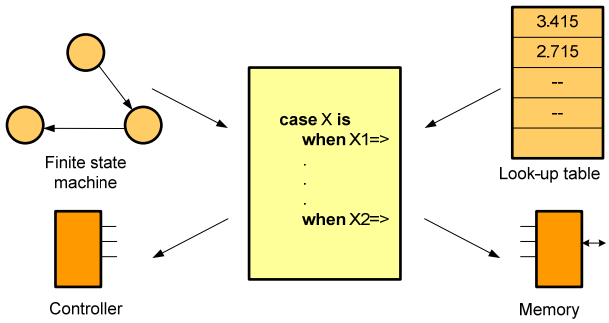


In Search of Solution

- **Semantics issue in description languages**
  - Modeling languages do not reflect design decisions
  - Modeling languages do not reflect implementation
  - Models are ambiguous for synthesis and verification
- **Better model formalism needed**
  - Definition of set of models
  - Definition of model composition
  - Relation between design decision and model transformation
- **Automatic model refinement and generation**
  - Model automation
- **Automatic metric estimation**

Missing Semantics in Languages

- **Ambiguous semantics of hardware and system languages**
  - Describes functionality, but not implementation
  - Same model may represent two completely different implementations
- **Simulatable but not synthesizable or verifiable**
  - Needs stricter semantics, describing design decisions



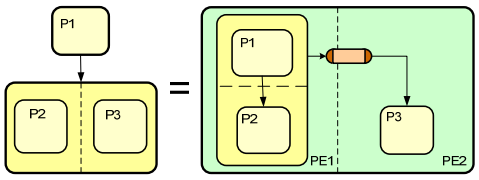
Model Algebra

Arithmetic algebra: <variables, operations>

a*(b+c) = a*b + a*c

- **Arithmetic algebra allows creation of expressions and proving their equivalences**

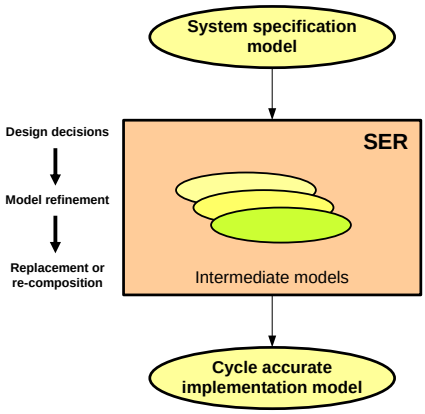
Model algebra: <objects, compositions>



- **Model algebra allows creation of models and proving their equivalences**

Specify-Explore-Refine Methodology

- **SER**
  - Sequence of models generated through a sequence of design decisions
  - Each design decision requires a model transformation
  - Transformations generate model refinement
  - Model refinement requires object replacement or re-composition



Necessary models, platforms and tools

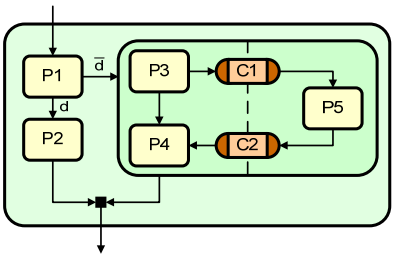
- **Design flow with 3 types of designers**
  - Application designers
  - System designers
  - Implementation designers
- **Design flow with 3 design models**
  - Executable specification model (SM)
  - Transaction-level model (TLM)
  - Cycle-accurate model (CAM)
- **Platform design with 4 component types**
  - Processing components
  - Storage components
  - Communication components
  - Interface components
- **Environment with 4 types of tools**
  - Metric evaluation with estimation tool
  - Design decisions with synthesis tool
  - Model generation with refinement tools
  - Validation with simulation/verification tool

Overview

- ✓ Abstraction layers
- ✓ Processor level
- ✓ System level
- ✓ Present system design
- **System models**
- System platforms
- Tools and environments
- Conclusion

System Specification Model (SM)

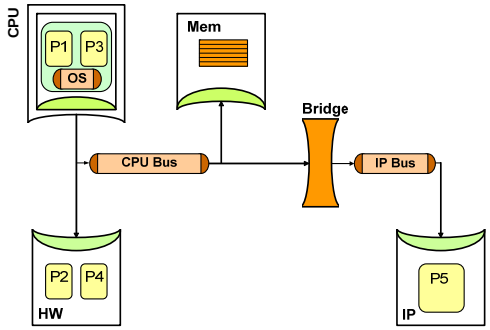
PSM example



- **SM includes application code and system requirements**
- **SM is for platform mapping and optimization**
- **SM uses a MoC (such as PSM)**
- **SM is an executable specification**
  - Processes communicating through channels

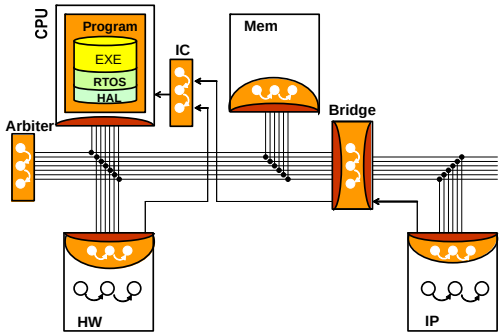
System Transaction-Level Model (TLM)

- **System TLM for system exploration**
  - Application algorithms
  - Platform structure
  - Design metrics
- **TLM annotations**
  - different metrics (timed TLM)
- **TLM with different details**
  - network TLM,
  - protocol TLM
- **CPU model**
  - Processes
  - OS model
- **IP, HW models**
  - Processes
- **Bus model**
  - UBC



System Cycle-Accurate Model (CAM)

- **CAM is for cycle-accurate exploration and design**
- **CAM is for input to RTL tools**
  - ASIC design
  - FPGA design
- **CPU model**
  - Compiled binary
  - Inserted RTOS
  - HAL RTL
- **IP and HW model**
  - Process RTL
  - IF RTL
- **Memory model**
  - Controller RTL
  - IF RTL
- **Bus model**
  - Arbiter RTL
  - IC RTL
  - Bridge RTL
  - IF RTL

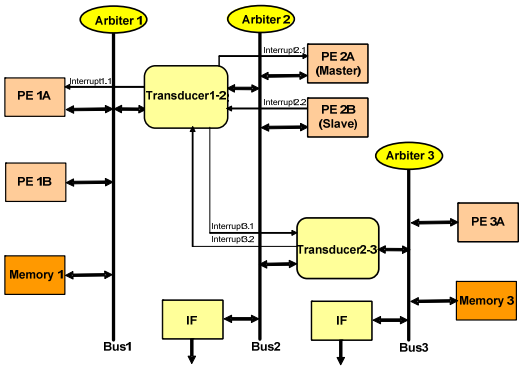


Overview

- Abstraction layers
- Processor level
- System level
- Present system design
  - Missing semantics
  - Modeling algebra
  - Specify-Explore-Refine
- System models
- ✓ **System platforms**
- Tools and environments
- Conclusion

General Platform Architecture

- **Processing components (PEs)**
  - Standard and custom processors
- **Storage components**
  - Local and global memories
- **Communication components**
  - Bridges, transducers
  - NoCs
- **Interface components**
  - Arbiters
  - Controllers
  - DMAs, UARTs

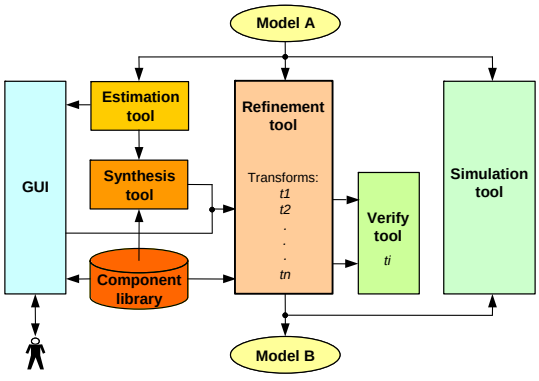


Overview

- ✓ Abstraction layers
- ✓ Processor level
- ✓ System level
- ✓ Present system design
- ✓ System models
- ✓ System platforms
- • **Tools and environments**
- • Conclusion

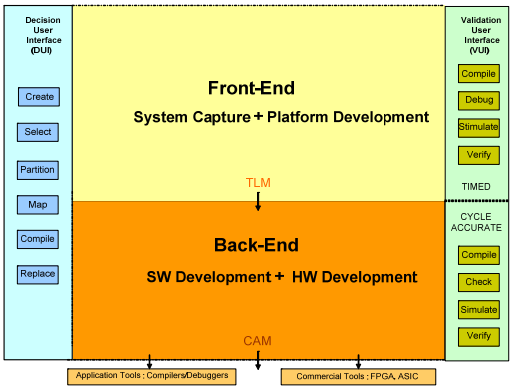
Model Refinement Tools

- **Model A to Model B refinement**
  - Estimation tool for metrics m1,…,mx
  - Synthesis tool for design decisions d1,…,dn
  - Refinement tool for transformations t1,…,tn
  - Verify tool for model equivalence after ti
  - Simulation tool for equivalence of A and B
  - Component library for model refinement
  - GUI for manual override of synthesis tool



General Environment for System Design

- **Front-End**
  - for application developers
  - for testing product concepts
- **Back-End**
  - for SW and HW development
  - product prototyping



Conclusion

- **Embedded system synthesis! Does it work? Intuitively it does**
  - Well-defined models, decisions, transformations, refinements
  - Worked in the past: layout, logic, RTL
  - System level complexity simplified
- **Extreme makeover is necessary for this new paradigm**
  - SW = HW = SOC = Embedded Systems
  - Only simulation based flow is not acceptable
  - Design methodology must be based on scientific principles
- **Benefits**
  - Large productivity gains
  - Easy design management
  - Easy derivatives and shorter TTM
- **What is next?**
  - Change to more structured system design
  - Application oriented EDA