Embedded System Design: Concepts and Tools

Daniel D. Gajski
Andreas Gerstlauer
Samar Abdi
Center for Embedded Computer Systems
University of California, Irvine

Outline

- Requirements
- Modeling
- Verification and synthesis
- Summary, conclusions and outlook

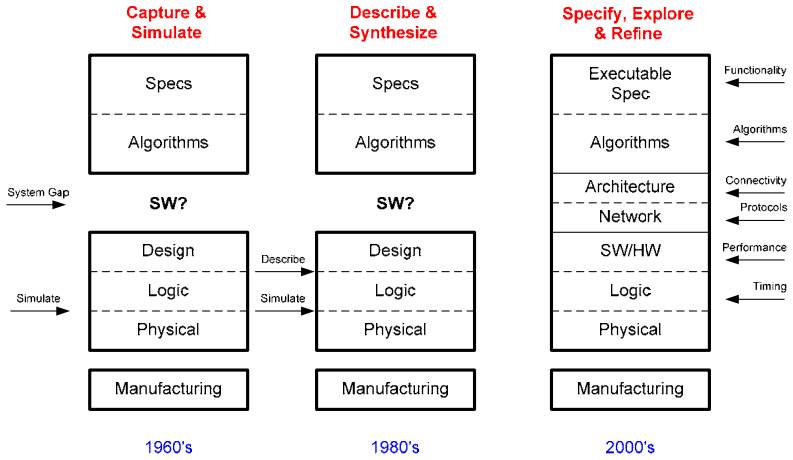
Embedded System Design: Requirements

Daniel D. Gajski
Center for Embedded Computer Systems
University of California, Irvine

Outline

- **Requirements**
  - Issues
  - Models
  - Platforms
  - Tools
- **Modeling**
- **Verification and synthesis**
- **Summary, conclusions and outlook**

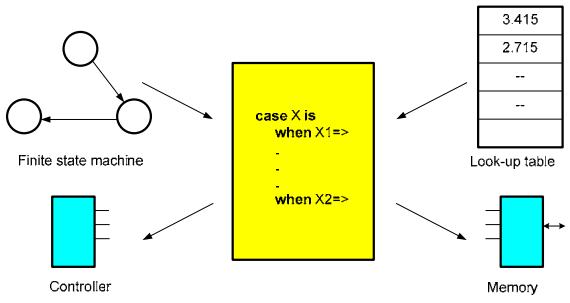
Closing the System Gap



Real gap: behavior and structure (semantics and syntax)

Simulation Based Methodology

Ambiguous semantics of hardware/system level languages



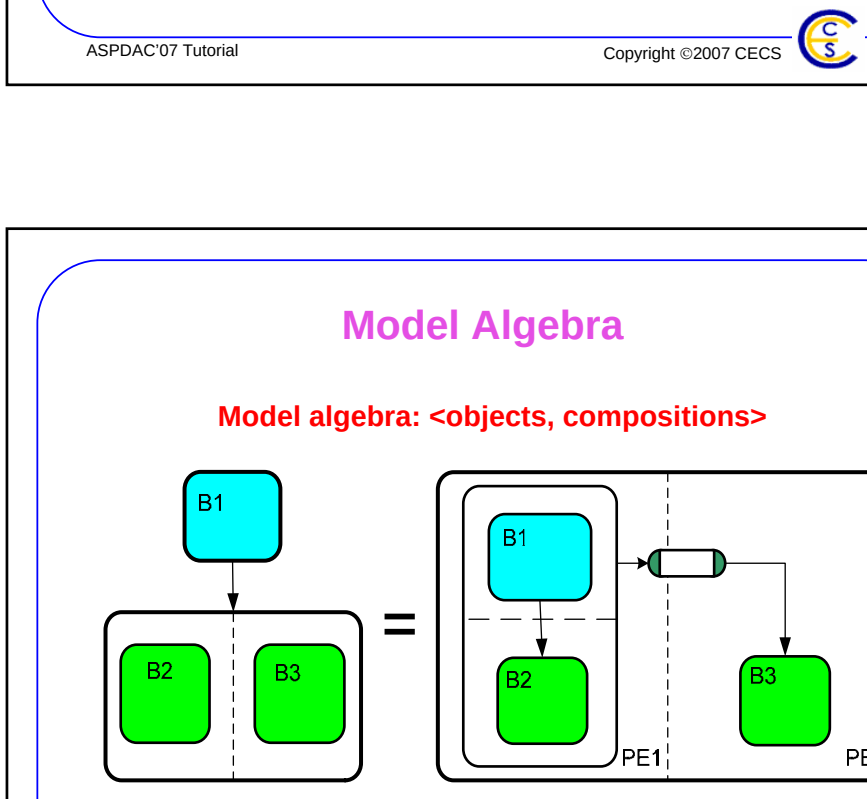
Simulatable but not synthesizable or verifiable

In Search of a Solution

Algebra: < objects, operations>

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

Arithmetic algebra allows creation of expressions and equivalences

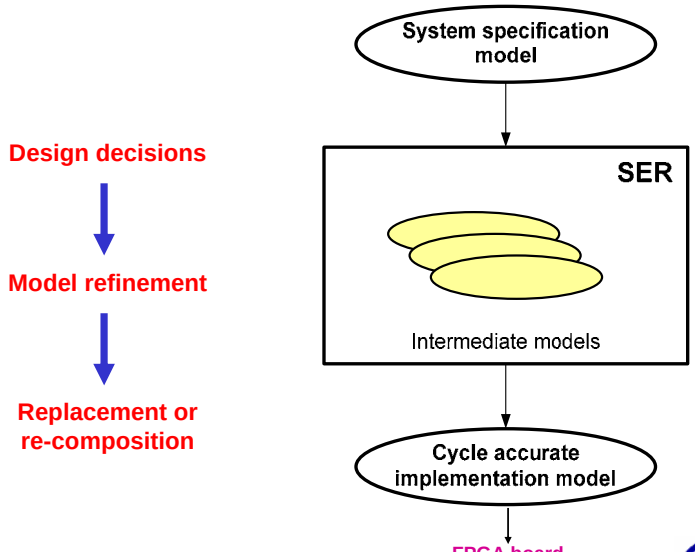


Model Algebra

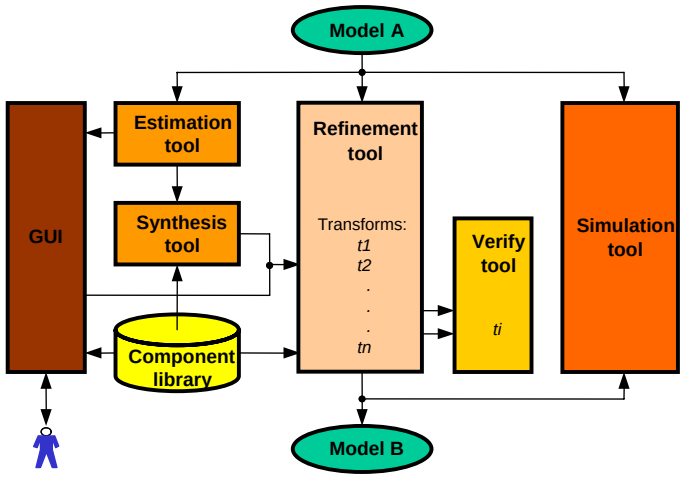
Model algebra: <objects, compositions>

Model algebra allows creation of models and model equivalences

Specify-Explore-Refine Methodology



General System Design Environment



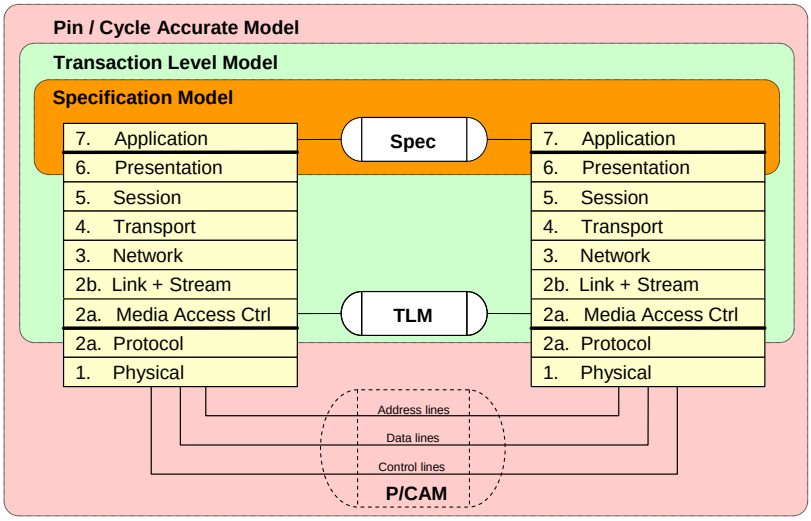
How many models?

Minimal set for any methodology

(Are 3 enough ?)

- System specification model (application designers)
- Transaction-level model (system designers)
- Pin&Cycle accurate model (implementation designers)

Three Models (with Respect to OSI)



Source: G Schirner

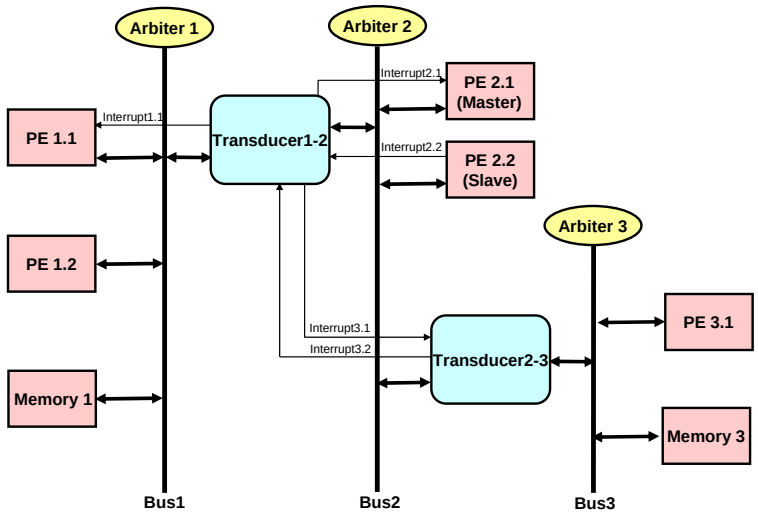
How many components?

Minimal set for any design

(Are 4 enough?)

- Processing element (PE)
- Memory
- Transducer / Bridge
- Arbiter

General System Model



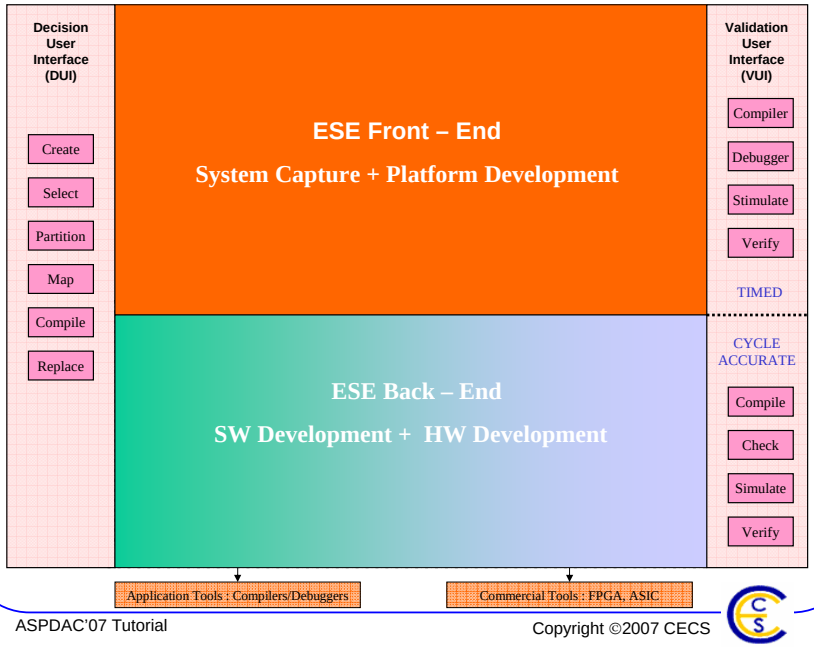
How many tools?

Minimal set for any methodology

(Are 2 enough?)

- Front-End (for application developers)
  - **Input: C, C++, Mathlab, UML, …**
  - **Output: TLM**
- Back-End (for SW/HW system designers)
  - **Input : TLM**
  - **Output: Pin/Cycle accurate Verilog/VHDL**

ES Environment



Does it work?

- Intuitively it does
  - Well defined models, rules, transformations, refinements
  - Worked in the past: layout, logic, RTL?
  - System level complexity simplified
- Following talks will prove the concept
  - ES modeling abstractions
  - Automatic model generation
  - Model synthesis and verification
  - Embedded System Environment (ESE)

Outline

- ✓ Requirements
  - ✓ Issues
  - ✓ Models
  - ✓ Platforms
  - ✓ Tools
- Modeling
- Verification and synthesis
- Summary, conclusions and outlook

Embedded System Design: Modeling

Andreas Gerstlauer
Center for Embedded Computer Systems
University of California, Irvine
http://www.cecs.uci.edu/pub_slides

Outline

- Requirements
- Modeling
  - Introduction
  - Languages and models
  - System modeling semantics
  - Automatic model generation
  - Design example
  - Tools
  - Summary and conclusions
- Verification and synthesis
- Summary, conclusions and outlook

Introduction

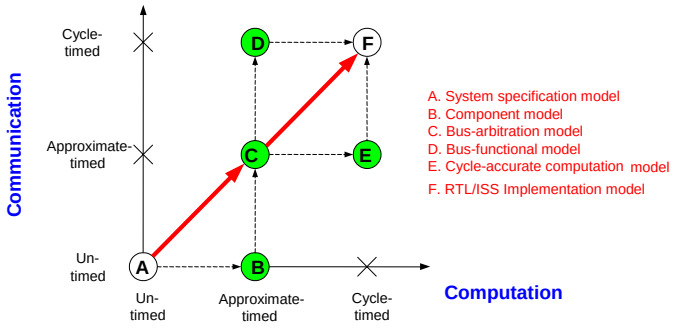
- **Growing SoC complexities and sizes**
  - Heterogeneous multi-processor SoC (MPSoC)
  - Networked, distributed systems

➢ **System modeling**
  - Validation and analysis
  - Concurrent hardware/software development
  - Specification for further implementation/synthesis
  - ➢ Higher levels of abstraction for speed/accuracy tradeoffs
  - ➢ Rapid, early design space exploration

System Modeling

- **Abstraction based on level of detail/granularity**
  - Computation
  - Communication

➢ **System-level design flow**
  - ➢ Path from model A to model F



Source: Lukai Cai, D. Gajski. "Transaction level modeling: An overview", *ISSS* 2003

System Design Languages

- **Netlists**
  - Structure only: components and connectivity
  - ➢ Gate-level [EDIF], system-level [SPIRIT/XML]
- **Hardware description languages (HDLs)**
  - Event-driven behavior: signals/wires, clocks
  - Register-transfer level (RTL): boolean logic
  - ➢ Discrete event [VHDL, Verilog]
- **System-level design languages (SLDLs)**
  - Software behavior: sequential functionality/programs
  - ➢ C-based [SpecC, SystemC, SystemVerilog]

ESL Modeling Today

- **Simulation-centric system modeling**
  - Co-simulation at lower RTL/ISS levels [Axys, VaST]
    - CPU-centric, limited architectures/designs [ARM]
  - Transaction-level models [CoWare, Summit]
    - Multi-processor system simulation [SystemC]
    - Graphical platform assembly [Eclipse]
    - Processor customization [Tensilica, Lisatek]
  - Algorithmic specification [SPW, MATLAB, COSSAP]
    - Varying models of computation (MoC) [Ptolemy]
    - Model-based design [UML, MATLAB/Simulink]
  - ➢ *Horizontal integration of different models / components*
  - ➢ *Lack vertical integration for synthesis-centric approach*

System Modeling Needs

- **Language ambiguities**
  - Simulation vs. synthesis (subsets)
  - Impossible to automatically discern implicit meaning

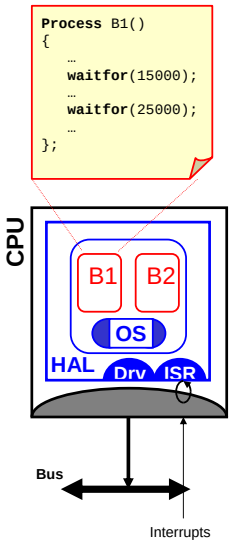
➢ **Well-defined, rigorous semantics**
  - Unambiguous, explicit abstractions, models
    - Objects and composition rules
    - Computation and communication
  - Systematic flow from specification to implementation
    - Transformations and refinements

  ➢ Reliable feedback at early stages
  ➢ Design automation for synthesis, verification
  ➢ Rapid, early design space exploration

Outline

- Requirements
- **Modeling**
  - Introduction
  - Languages and models
  - **System modeling semantics**
  - Automatic model generation
  - Design example
  - Tools
  - Summary and conclusions
- Verification and synthesis
- Summary, conclusions and outlook

Computation Modeling

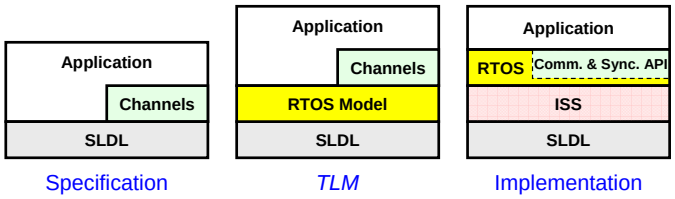


- **Application modeling**
  - Native process execution (C code)
  - Back-annotated execution timing
- **Processor modeling**
  - Operating system
    - Real-time multi-tasking (RTOS model)
    - Bus drivers (C code)
  - Hardware abstraction layer (HAL)
    - Interrupt handlers
    - Media accesses
  - Processor hardware
    - Bus interfaces (I/O state machines)
    - Interrupt suspension and timing

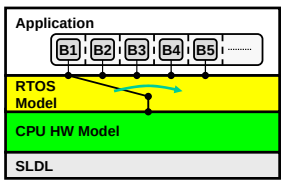
Source: G. Schirner, A. Gerstlauer, R. Doemer. "Abstract, Multifaceted Modeling of Embedded Processors for System Level Design," *ASPDAC*, 2007.

RTOS Modeling

- **High-level RTOS abstraction**
  - Wrap around SLDL primitives, replace event handling



  - High-level, abstract model
  - Target-independent, canonical API
  - Small overhead, low complexity
  - Dynamic scheduling behavior
    - Real-time scheduling, multi-tasking
    - Preemption, interrupt handling
  - IPC channels
    - Task communication, synchronization



Source: A. Gerstlauer, H. Yu, D. Gajski. "RTOS Modeling for System-Level Design," *DATE*, 2003.

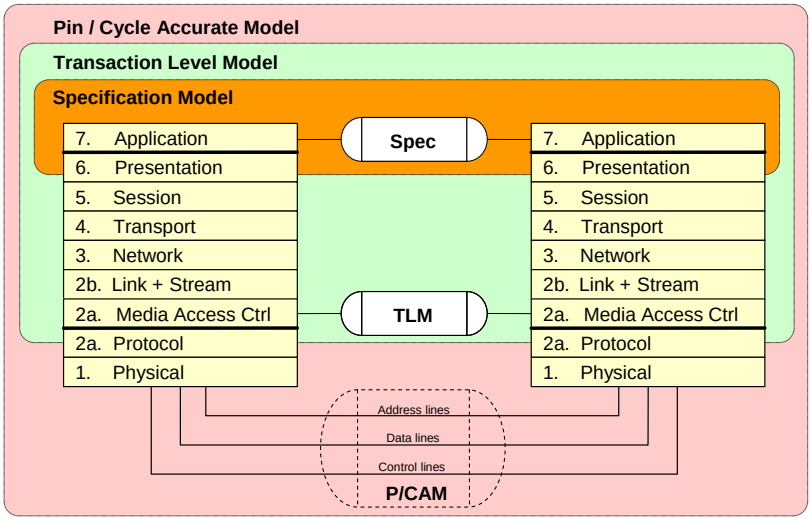
Communication Modeling

- **SoC communication layers**

Layer	Semantics	Functionality	Implementation	OSI
Application	Channels, variables	Computation	Application	7
Presentation	End-to-end typed messages	Data formatting	Application	6
Session	End-to-end untyped messages	Synchronization, Multiplexing	OS kernel	5
Transport	End-to-end data streams	Packeting, Flow control, Error correction	OS kernel	4
Network	End-to-end packets	Routing	OS kernel	3
Link	Point-to-point logical links	Station typing, Synchronization	Driver	2b
Stream	Point-to-point control/data streams	Multiplexing, Addressing	Driver	2b
Media Access	Shared medium byte streams	Data slicing, Arbitration	HAL	2a
Protocol	Media (word/frame) transactions	Protocol timing	Hardware	2a
Physical	Pins, wires	Driving, sampling	Interconnect	1

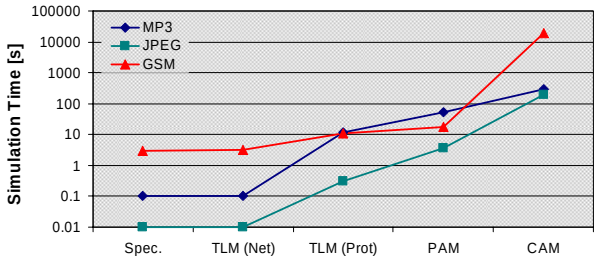
Source: A. Gerstlauer, D. Shin, R. Dömer, D. Gajski, "System-Level Communication Modeling for Network-On-Chip Synthesis," ASPDAC, 2005.

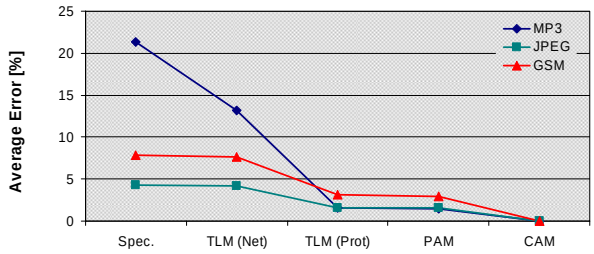
Transaction-Level Modeling



Source: G. Schirner

Modeling Results





Source: G. Schirner, A. Gerstlauer, R. Doemer

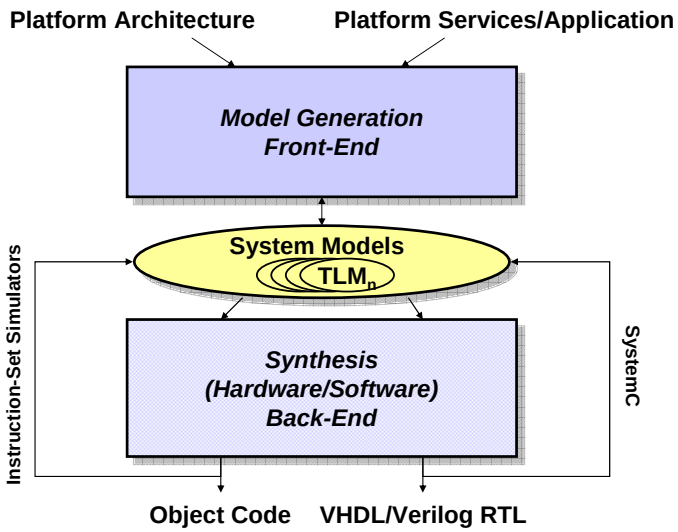
Outline

- Requirements
- **Modeling**
  - Introduction
  - Languages and models
  - System modeling semantics
  - **Automatic model generation**
  - Design example
  - Tools
  - Summary and conclusions
- Verification and synthesis
- Summary, conclusions and outlook

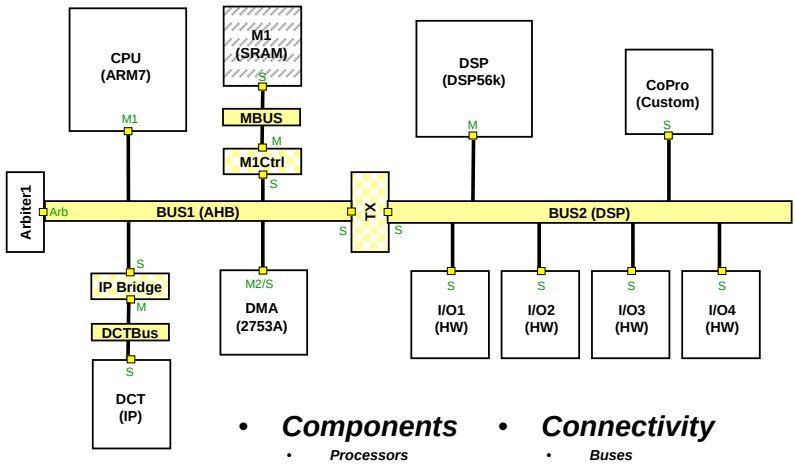
Automatic Model Generation

- **Problem: Writing of system models is**
  - Time consuming, error-prone, tedious
- **Solution:**
  - Automatic model generation
- **Refinement-based approach**
  - System designer : makes design decisions
  - Refinement tool : automatically generates the model
- **Benefits**
  - No manual model writing, focus on design decisions
  - Low error rate by automating error-prone tasks
  - Easy change/upgrade for incremental/derivative design
  - No change in basic design methodology
  - ➢ Enables fast, extensive design space exploration
  - ➢ Productivity gains (1000x)
  - ➢ Shorter time-to-market

ESL Design Flow

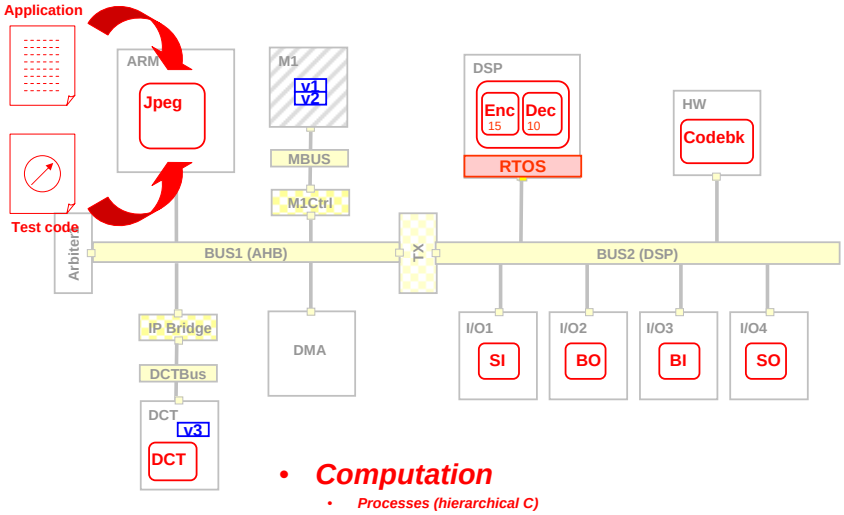


Platform Architecture



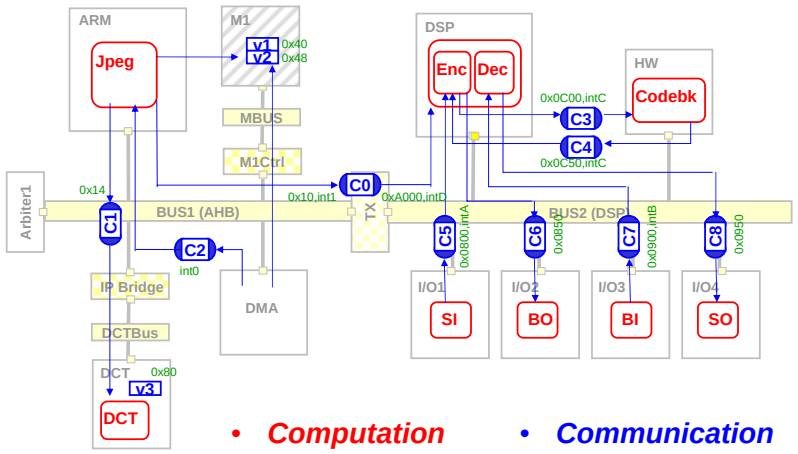
- ***Components***
  - *Processors*
  - *Memories*
  - *IPs*
  - *Custom HW*
- ***Connectivity***
  - *Buses*
  - *Bridges*
  - *Ports*

Platform Services



- ***Computation***
  - *Processes (hierarchical C)*
  - *Scheduling (static/OS)*

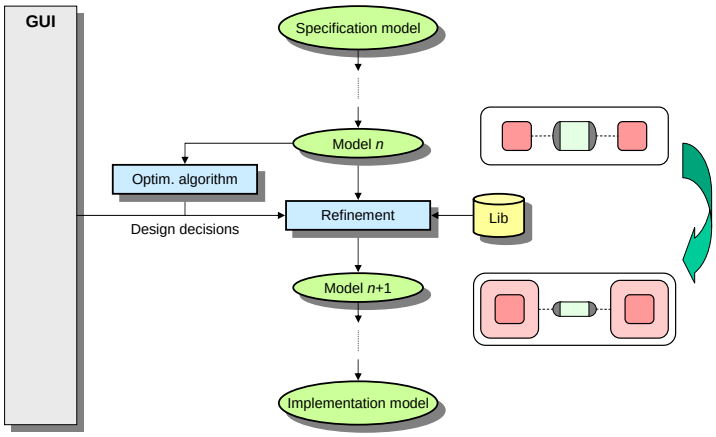
Platform Services



- ***Computation***
  - ***Processes (hierarchical C)***
  - ***Scheduling (static/OS)***
- ***Communication***
  - ***High-level IPC (channels)***
  - ***Storage (variables)***

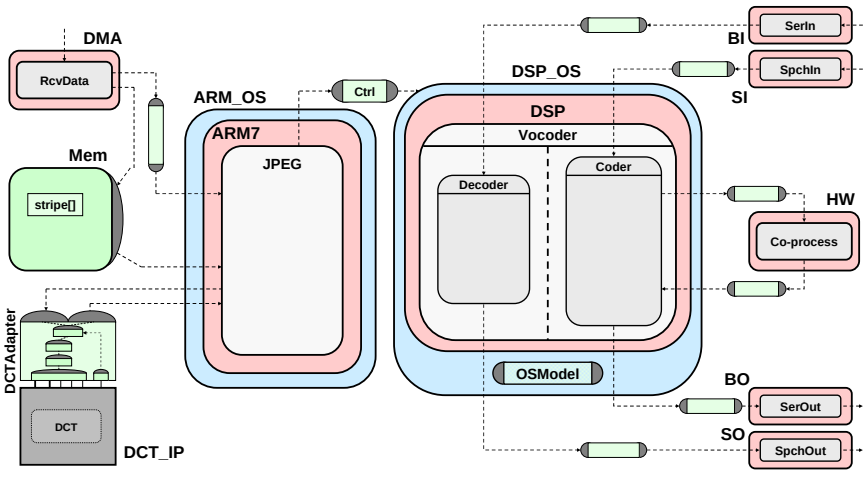
Model Generation Process

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

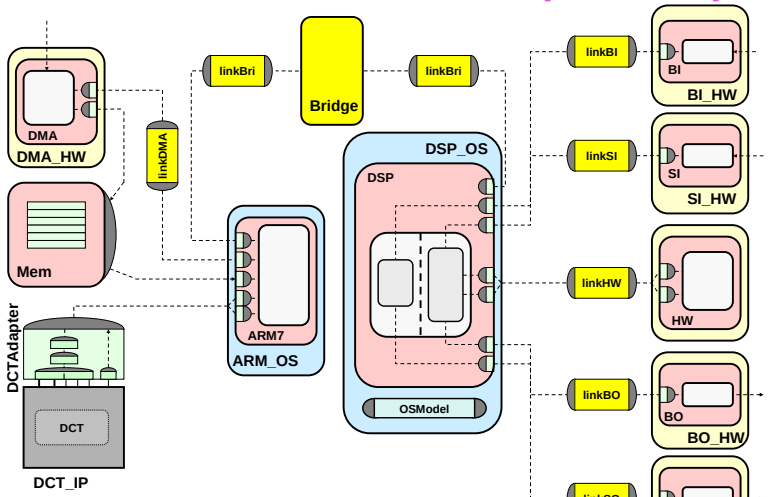


- **Successive model refinement**
- **Layers of implementation detail**

Specification Model

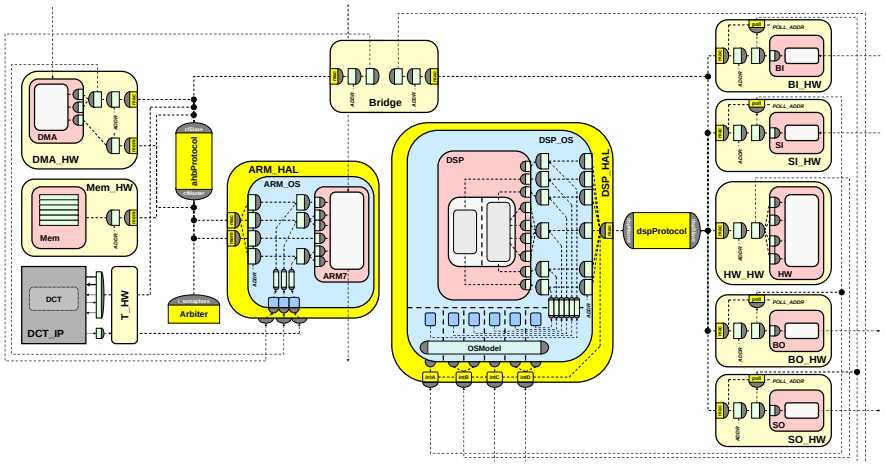


Transaction-Level Model (Network)



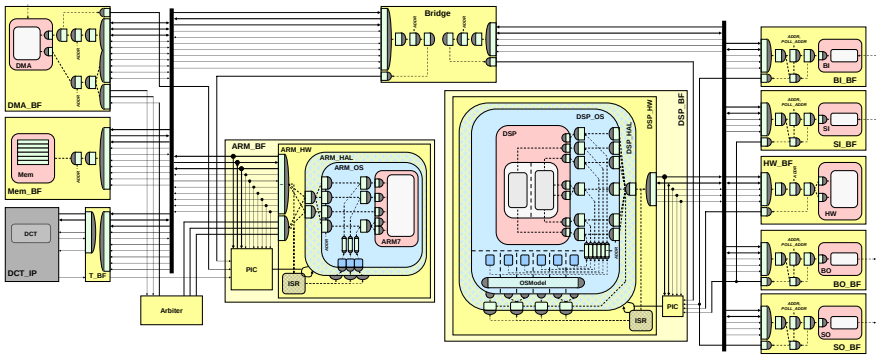
- Data conversion, channel merging
- Transducer insertion, Packeting, Routing

Transaction-Level Model (Protocol)



- Synchronization, addressing, media acces
- Arbitration, data slicing, interrupt handling

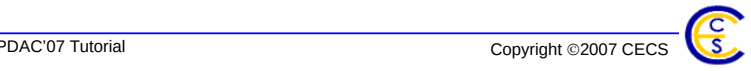
Pin-Accurate Model



- Implementation synthesis in backend tools
  - Interface synthesis on hardware side
  - Bus driver and RTOS synthesis on the software side

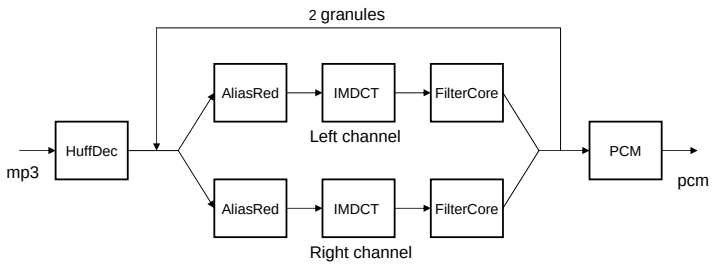
Outline

- Requirements
- Modeling
  - Introduction
  - Languages and models
  - System modeling semantics
  - Automatic model generation
  - **Design example**
  - Tools
  - Summary and conclusions
- Verification and synthesis
- Summary, conclusions and outlook



Example: MP3 Decoder

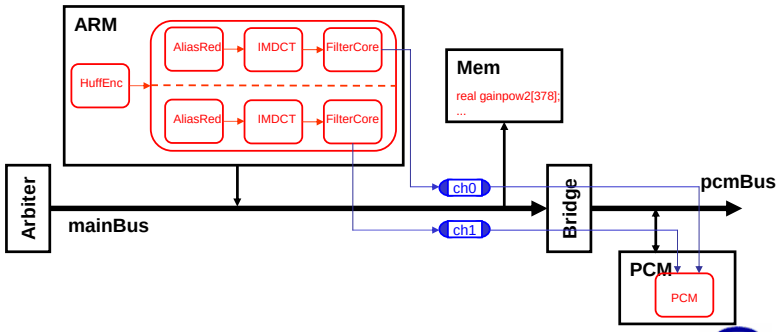
- **Functional block diagram (major blocks only)**



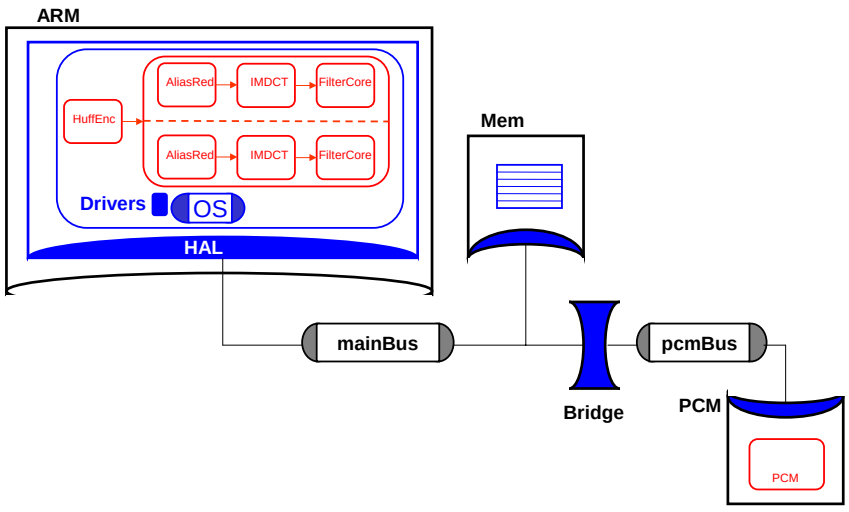
- **Timing constraints**
  - 38 frames per second
  - Frame delay < 26.12ms

System Definition

- **Components allocated from database**
  - Bridge used to interface two busses
- **Hierarchical processes inside PEs**
  - Parallel, sequential, leaf (C code)
- **Channels and variables between PEs**

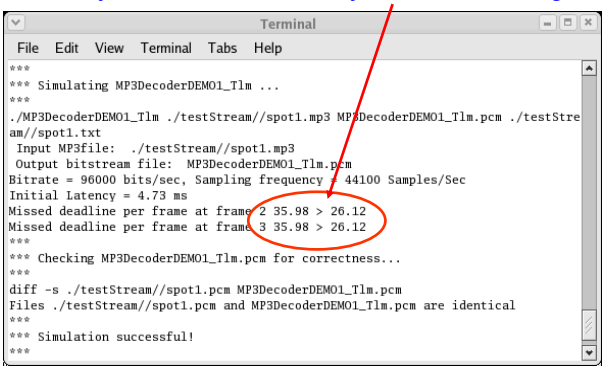


Output: Transaction-Level Model (TLM)



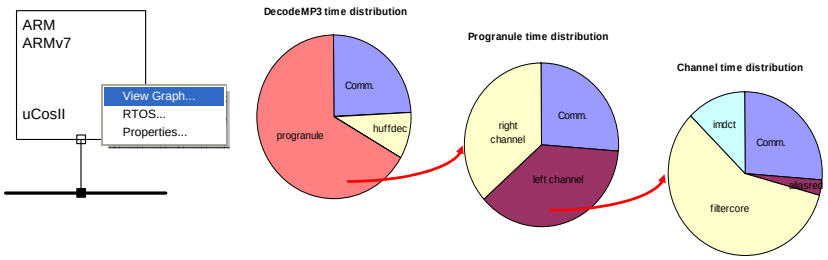
TLM Simulation

- **Timed-simulation**
  - Computation delays back-annotated in processes
  - Communication delays modeled in the bus channels
- **Simulation of TLM**
  - Functionally correct but frame delay far exceeds timing constraint



Computation Analysis

- **View computation time of processes**
  - *FilterCore* is the most computation-intensive

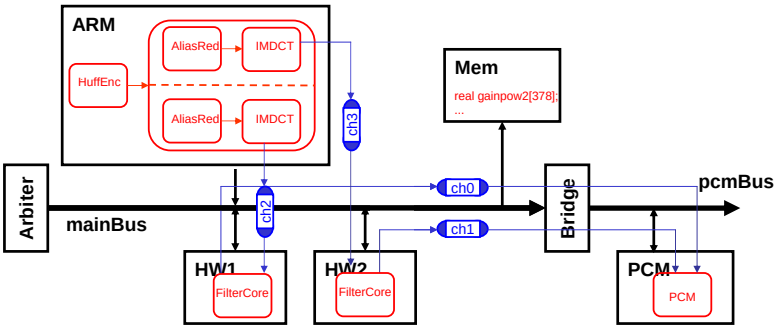


- **Look for parallelism in process hierarchy**
  - *FilterCore* processes are running in parallel

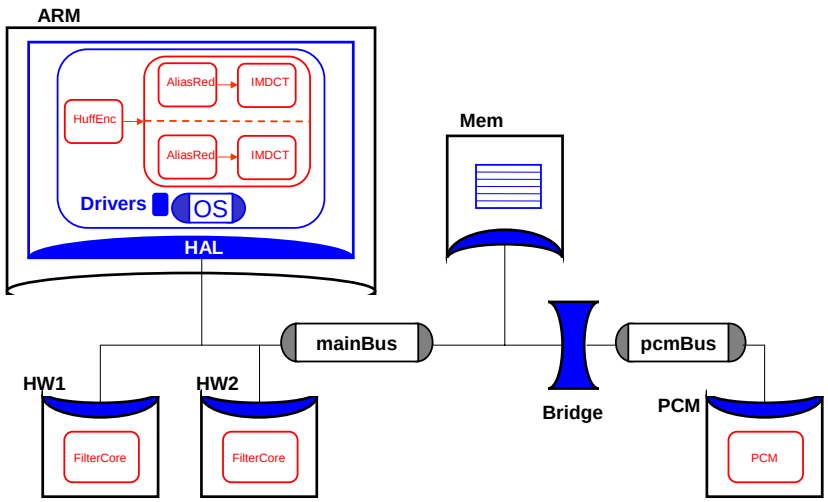
    →Use two identical custom HWs

System Modifications

- Add two more PEs: HW1 and HW2
- Connect HW1 and HW2 to the mainBus
- Move processes from ARM to HWs
  - Channels are automatically inserted

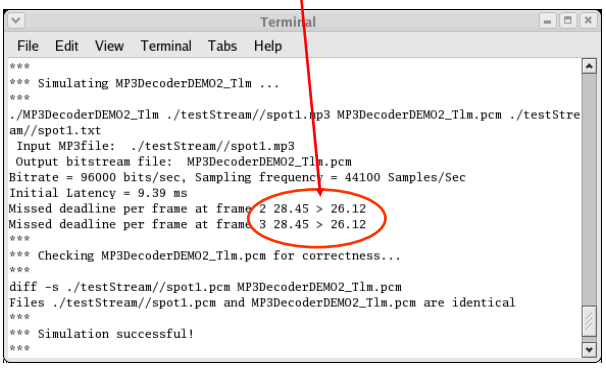


Modified TLM



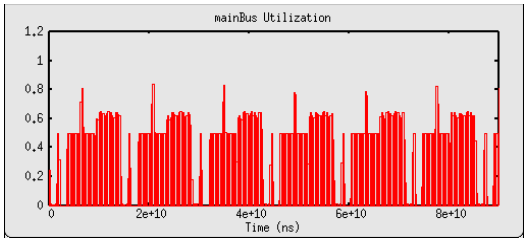
Modified TLM Simulation

- **Simulation of TLM**
  - Functionally correct
  - Frame delay still violates timing constraint

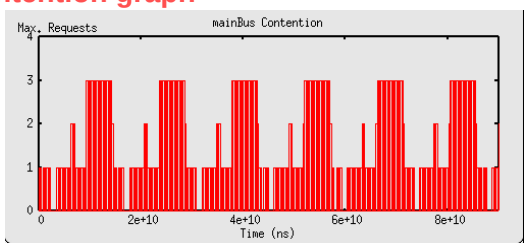


Communication Analysis

- **Bus utilization graph**

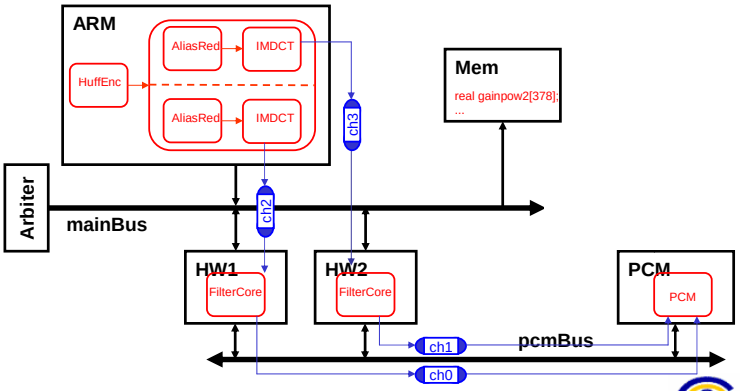


- **Bus contention graph**

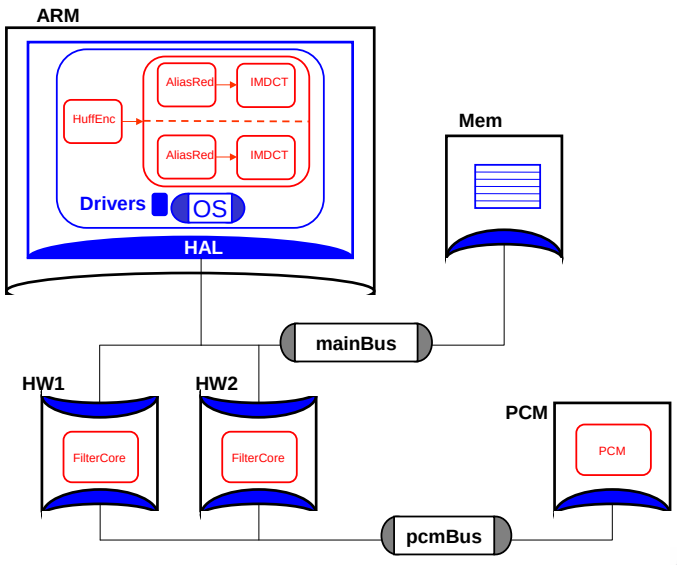


More System Modifications

- Make two ports in HW1, HW2
- Connect HWs directly to the DHS bus
- Remove the bridge

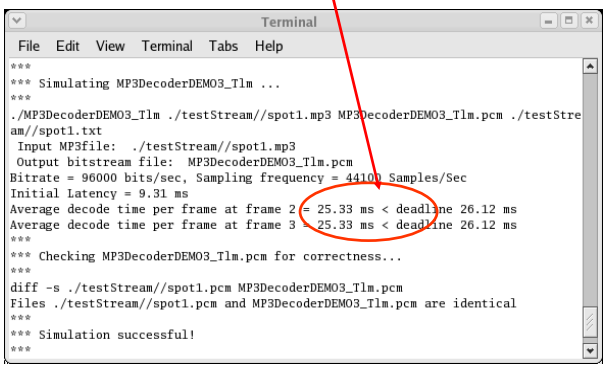


Modified TLM



Modified TLM Simulation

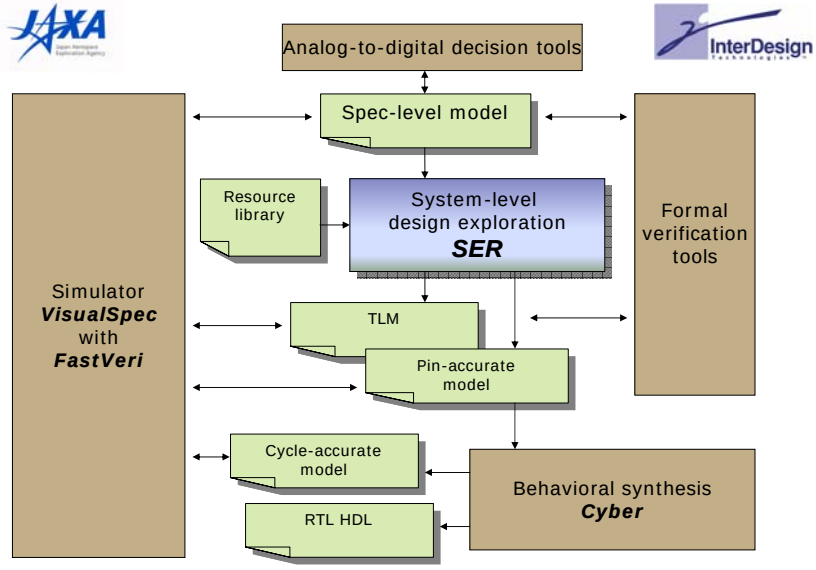
- **Simulation of TLM**
  - Functionally correct
  - Frame delay now meets timing constraint



Outline

- Requirements
- **Modeling**
  - Introduction
  - Languages and models
  - System modeling semantics
  - Automatic model generation
  - Design example
  - **Tools**
  - Summary and conclusions
- Verification and synthesis
- Summary, conclusions and outlook

ELEGANT Environment

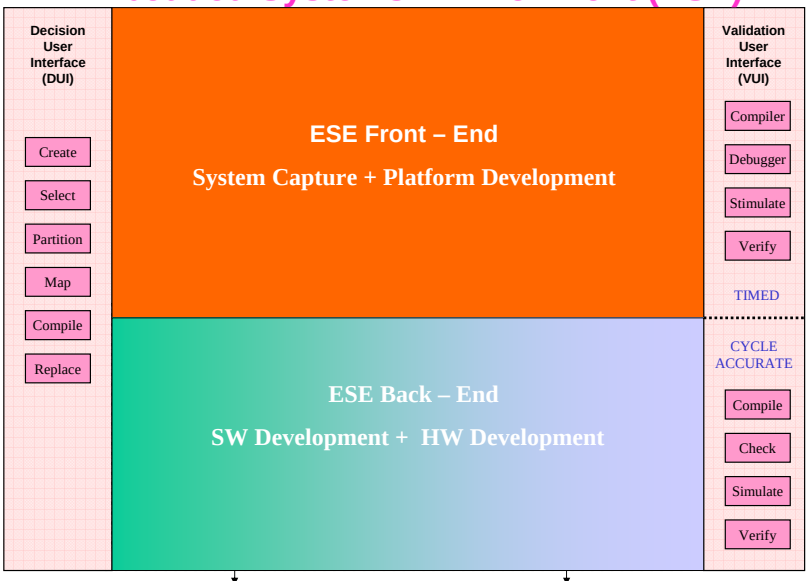


Source: InterDesign Technologies, Inc.

ELEGANT Environment

- **Complete ESL design environment**
  - Japanese Aerospace Exploration Agency (JAXA)
  - Focus on correctness and reliability
  - Integration of tools for simulation, synthesis, verification
- **Specify-Explore-Refine (SER) tool**
  - 1st-generation of automatic model generation
  - Top-down design
    - Specifiy desired system functionality
    - Explore architectural design alternatives
    - Refine specification into automatically generated TLM or PAM

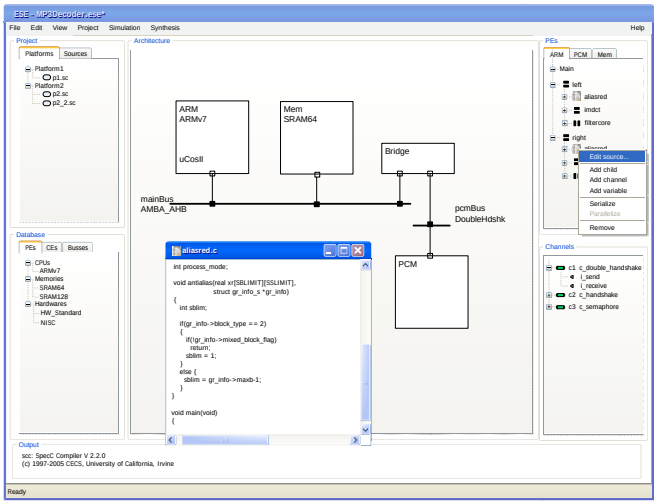
Embedded Systems Environment (ESE)



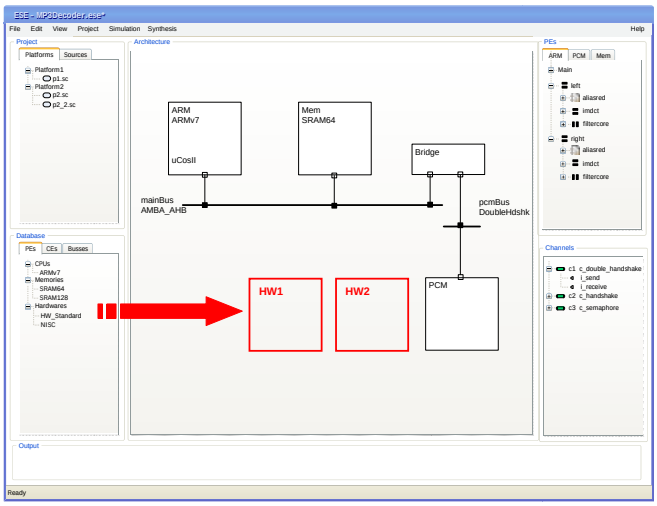
ES Environment (ESE)

- **Next-generation ESL tool set**
  - ESE Front-End
    - Platform capture and virtual prototyping (modeling, simulation)
    - Embedded software development environment
    - Communication network model (advanced TLM) generation
    - Heterogeneous multi-processor SoC (MPSoC) target support
  - ESE Back-End
    - Platform customization, optimization and implementation
    - Software and hardware synthesis
    - FPGA board prototyping

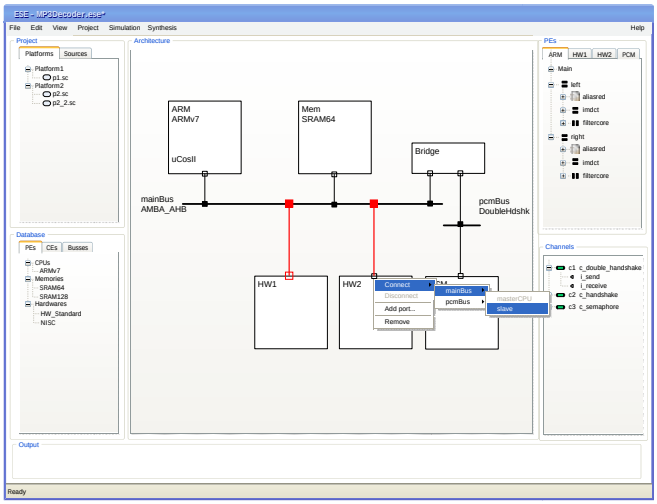
ESE: System Definition

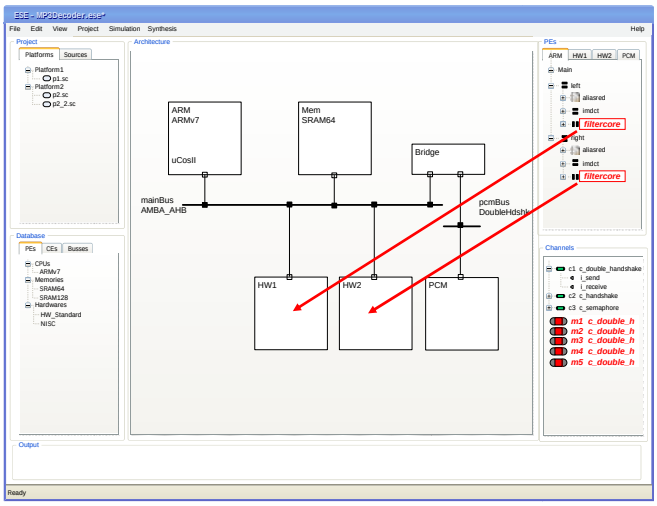


ESE: System Modifications

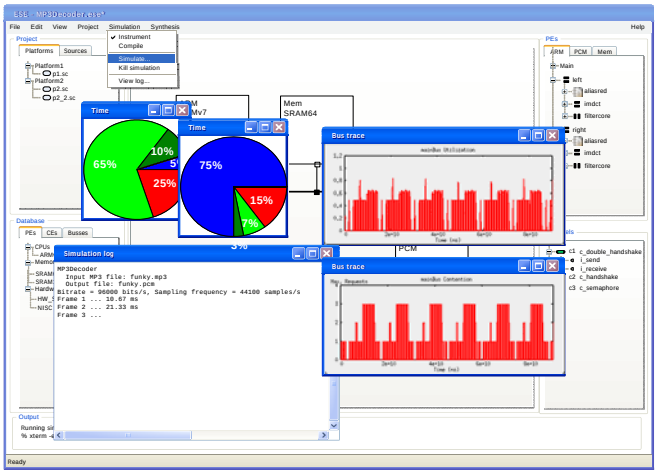


ESE: System Modifications





ESE: TLM Simulation



Summary and Conclusions

- **Technology advantages**
  - Platform can be easily captured using GUI
  - TL models are automatically generated for development and testing of application code
  - Legacy or preliminary SW can be easily captured and mapped to the platform
  - ESL allows concurrent development of platform SW, HW and application code
  - ESL allows easy upgrade of platform and reuse of legacy application SW and RTL HW code

➢ **Early, rapid prototyping for design space exploration**

➢ **Significant productivity gains**

References

- **System-level languages and modeling**
  - T. Grötker, S. Liao, G. Martin, S. Swan. *System Design with SystemC*. Kluwer, 2002.
  - A. Gerstlauer, R. Dömer, J. Peng, D. D. Gajski, *System Design: A Practical Guide with SpecC*, Kluwer, 2001.
  - F. Ghenassia. *Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*, Springer, 2006.
- **System prototyping and simulation**
  - L. Benini, D. Bertozzi, A. Bogoliolo, F. Menichelli, M. Olivieri. "MPARM: Exploring the Multi-Processor SoC Design Space with SystemC," *Journal of VLSI Signal Processing*, Sept. 2005.
  - T. Kempf, M. Dörper, R. Leupers, G. Ascheid, H. Meyr, T. Kogel, B. Vanthournout. "A Modular Simulation Framework for Spatial and Temporal Task Mapping onto Multi-Processor SoC Platforms," *DATE Conference*, March 2005.
- **System-level design and design environments**
  - F. Balarin, H. Hsieh, L. Lavagno, C. Passerone, A. Pinto, A. Sangiovanni-Vincentelli, Y. Watanabe, G. Yang. "Metropolis: a Design Environment for Heterogeneous Systems," *Multiprocessor Systems-on-Chips* (Editors: W. Wolf, A. Jerraya), Morgan Kaufmann, 2004.
  - A. A. Jerraya, F. Petrot, A. Bouchhima. "Programming Models and HW-SW Interfaces Abstraction for Multi-Processor SoC," *DAC*, June 2006.
  - D. Gajski, A. Gerstlauer, R. Doemer, S. Abdi, J. Peng, D. Shin, R. Ang, "ESE Front End," http://www.cecs.uci.edu/pub_slides, March 2006.

Outline

- ✓ **Requirements**
- ✓ **Modeling**
- **Verification and synthesis**
- **Summary, conclusions and outlook**

Embedded System Design: Verification and Synthesis

Samar Abdi
Center for Embedded Computer Systems
University of California, Irvine
http://www.cecs.uci.edu/pub_slides

Outline

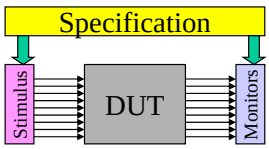
- ✓ **Requirements**
- ✓ **Modeling**
- **Verification and synthesis**
  - Existing verification methods
  - TLM verification using transformations
  - Issues in ES synthesis
  - Synthesis of MP3 player design
  - Results
- **Summary, conclusions and outlook**

Design Verification Methods

- **Simulation based methods**
  - Specify input test vector, output test vector pair
  - Run simulation and compare output against expected output
- **Formal Methods**
  - Check equivalence of design models or parts of models
  - Check specified properties on models
- **Semi-formal Methods**
  - Specify inputs and outputs as symbolic expressions
  - Check simulation output against expected expression

Simulation

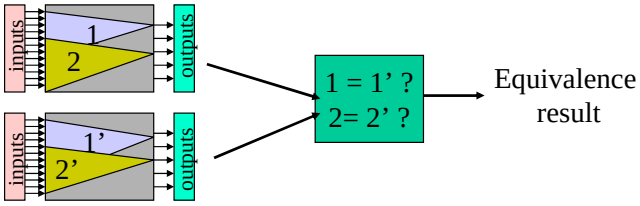
- **Task : Create test vectors and simulate model**
- **Tools: VCS(Synopsys), ModelSim(Mentor), NC-Sim(Cadence)**
- **Inputs**
  - Specification
    - Typically natural language, incomplete and informal
    - Used to create interesting stimuli and monitors
  - Model of DUT
    - Typically written in HDL or C or both
- **Output**
  - Failed test vectors
    - Pointed out in different design representations by debugging tools



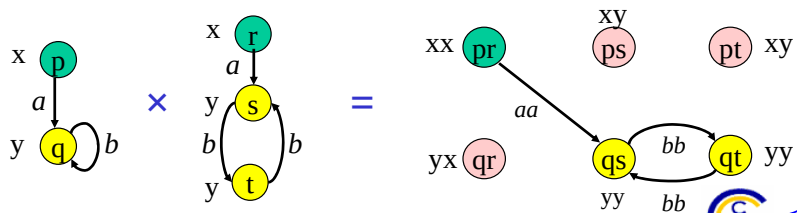
Typical simulation environment

Logic and FSM Equivalence Checking

- **LEC uses boolean algebra to check for logic equivalence**

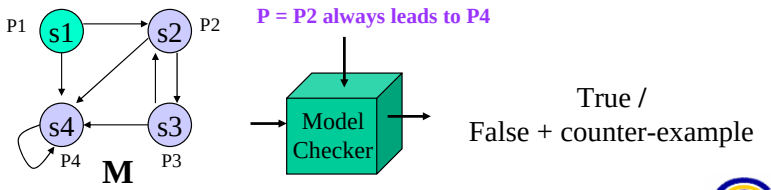


- **SEC uses FSMs to check for sequential equivalence**



Model Checking

- **Model M satisfies property P? (Clarke, Emerson '81)**
- **Inputs**
  - State transition system representation of M
  - Temporal property P as formula of state properties
- **Output**
  - True (property holds)
  - False + counter-example (property does not hold)

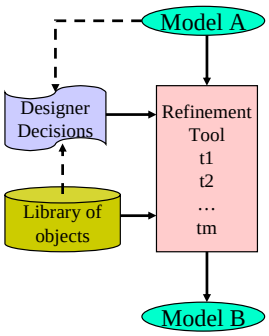


New Verification Challenges for SoC Design

- **Design complexity**
  - Size
    - Verification either takes unreasonable time (eg. Logic simulation)
    - Or takes unreasonable memory (eg. Model Checking)
  - Heterogeneity
    - HW / SW components on the same chip
    - Interface problems
  - Modeling abstraction
    - No formalism exists to prove equivalence of high level models
    - LEC and SEC are not applicable above RTL
- **Possible directions**
  - Methodology
    - Unified HW/SW models
    - Model formalization
    - Automatic model transformations

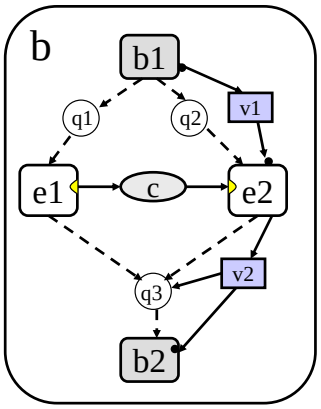
System Verification through Refinement

- **Set of models**
- **Designer Decisions => transformations**
- **Transformations preserve equivalence**
  - Same partial order of tasks
  - Same input/output data for each task
  - Same partial order of data transactions
  - Equivalent replacements
- **All refined models will be "equivalent" to input model**
  - Still need to verify
    - First model
    - Correctness of replacements



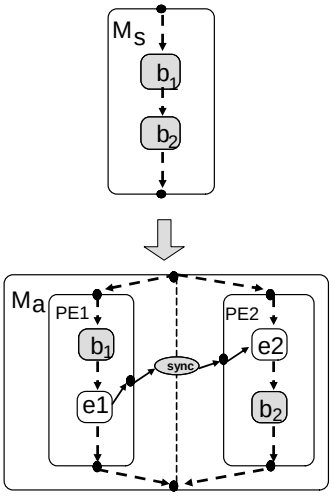
Formal Model Representation

- **Model Algebra**
  - < {objects}, {composition rules} >
- **Objects**
  - Behaviors (for computation)
    - Identity behaviors (output identical to input)
  - Channels (for synchronized communication)
  - Variables (for storage)
  - Ports (for hierarchy / connections)
- **Composition rules**
  - Control dependency
  - Channel transaction
  - Variable read/write
- **Hierarchical behavior**
  - Grouping of sub-behaviors, channels, variables and their compositions
- **System = Top level behavior**

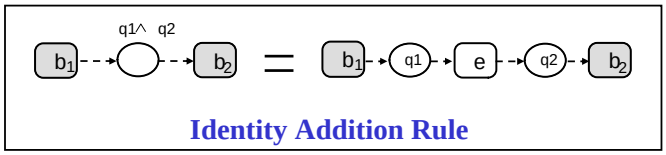


Refinement Example

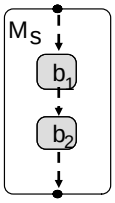
- **Design decisions**
  - Select PEs
  - Map leaf behaviors to PEs
- **Resulting model refinement**
  - Additional hierarchy for PEs
  - Distribute leaf behaviors inside PEs
  - Add Identity behaviors and channels to preserve control flow
    - Each PE has independent control
- **Refinement verification**
  - Express refinement as sequence of model transformations



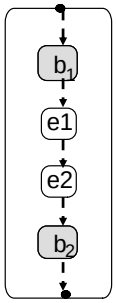
Transformation Step 1



Identity Addition Rule

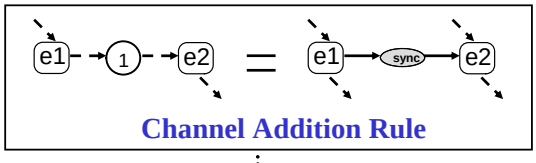


(Original Model)

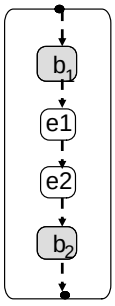


(Intermediate Model 1)

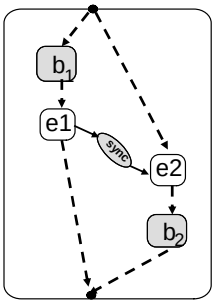
Transformation Step 2



Channel Addition Rule

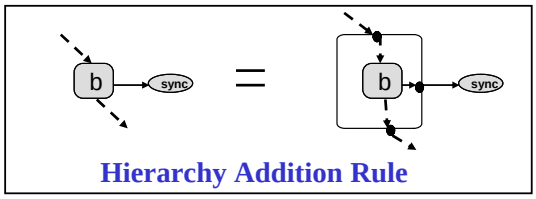


(Intermediate Model 1)

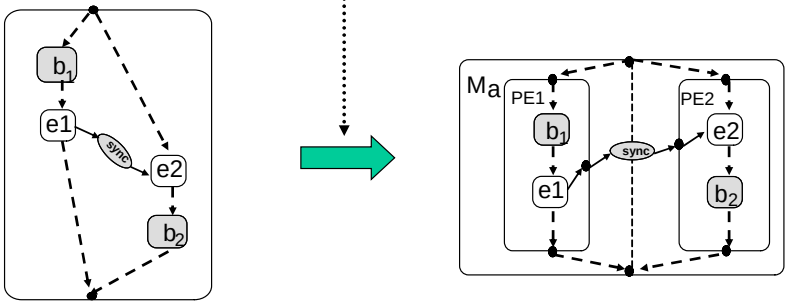


(Intermediate Model 2)

Transformation Step 3



Hierarchy Addition Rule



(Intermediate Model 2) (Final Model)

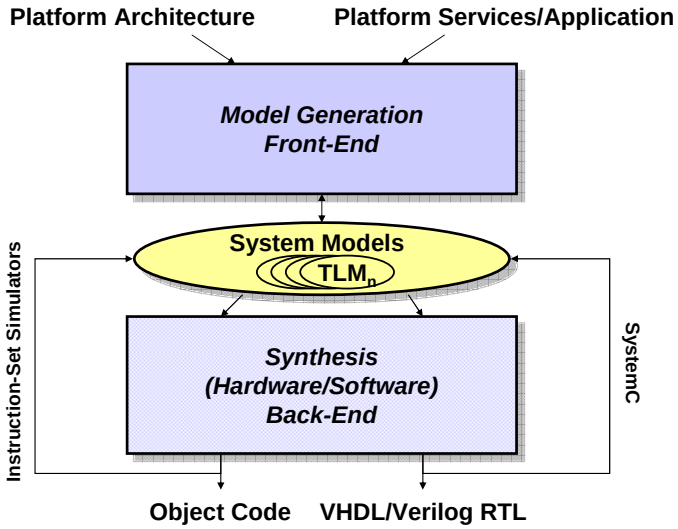
Summary of ES Verification

- **Variety of verification techniques available**
  - But they apply to traditional design flow based on RTL
- **Challenges for verification of large system designs**
  - Simulation based techniques take way too long
  - Most formal techniques cannot scale in size or abstraction level
- **Future design and verification**
  - Well defined semantics for models at different abstraction levels
  - Well defined transformations for design decisions
    - Verify transformations
    - Automate refinements
- **Modeling semantics are the key to verification !**

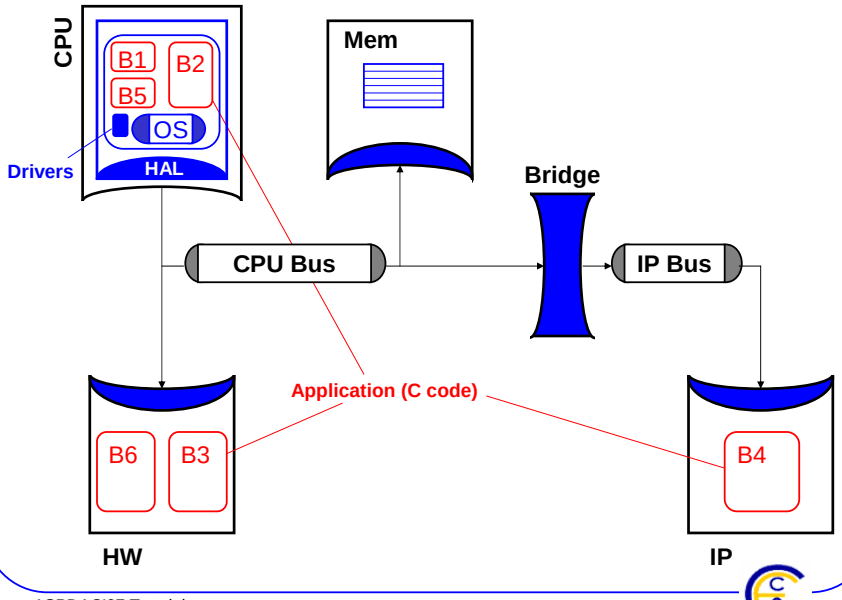
Outline

- ✓ Requirements
- ✓ Modeling
- Verification and synthesis
  - ✓ Existing verification methods
  - ✓ TLM verification using transformations
  - Issues in ES synthesis
  - Synthesis of MP3 player design
  - Results
- Summary, conclusions and outlook

ESL Design Flow



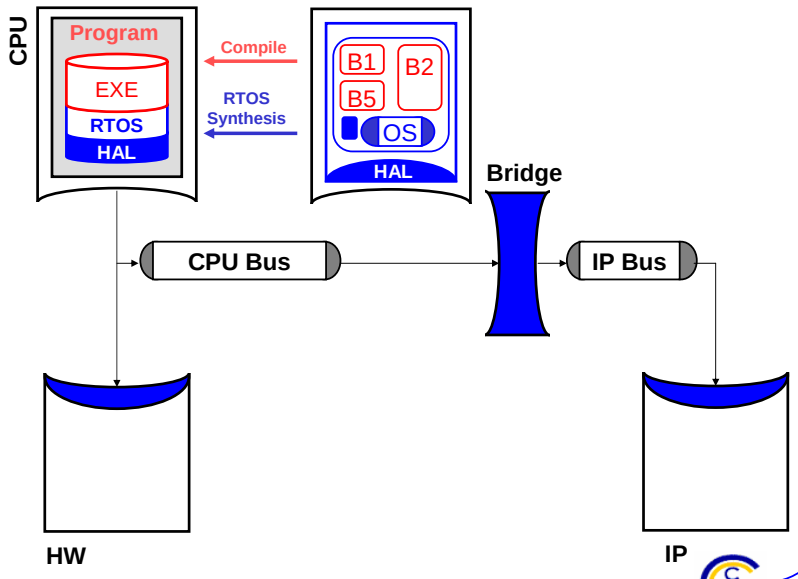
Input: Transaction Level Model (TLM)



TLM Features

- **Universal Bus Channel (UBC)**
  - Bus is modeled as universal channel with send/recv, read/write functions
  - Well defined functions for routing, synchronization, arbitration and transfer
- **SW modeling**
  - Application SW is modeled as processes in C
  - A RTOS model or real RTOS is used for dynamic scheduling of processes
  - Communication with peripherals, memory or other IP is done using UBC
- **HW modeling**
  - Application HW is modeled as processes written in C
  - Communication with processor, memory or other IP is done using UBC
- **Memory modeling**
  - Memory is modeled as array in C
  - Controller is modeled by function in UBC

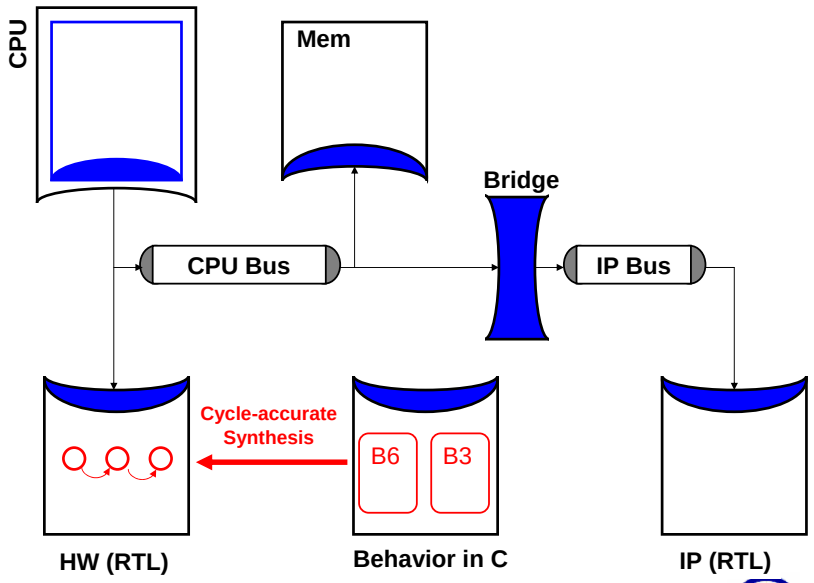
Cycle-Accurate Software Synthesis



SW Synthesis Issues

- **Compiler selection**
  - The designer specifies which compiler is used for the SW
- **Library selection**
  - Libraries are selected for SW support such as file systems, string manipulation etc.
  - Prototype debugging requires selection of additional libraries
- **RTOS selection and targeting**
  - Designer selects an RTOS for the processor
  - RTOS model is replaced by real RTOS and SW is re-targeted
- **Program and data memory**
  - Address range for SW program memory is assigned
  - Address range for data memory used by program is assigned
  - For large programs or data, off-chip memory may be allocated

Cycle-Accurate Hardware Synthesis



HW Synthesis Issues

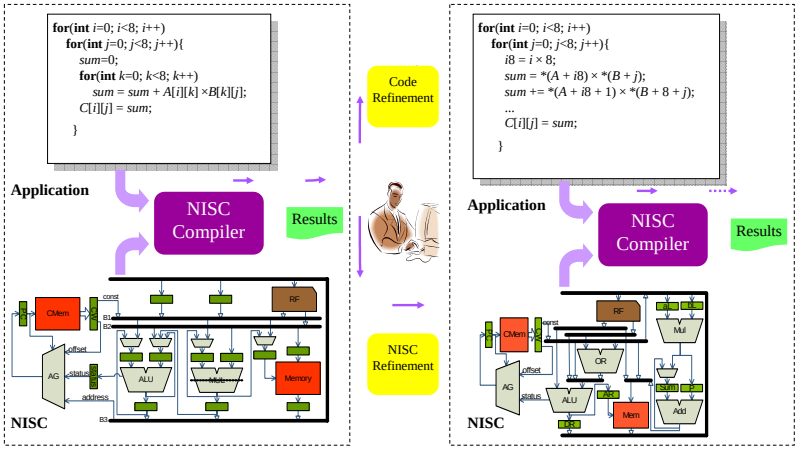
- **IP insertion**
  - C model of HW is replaced with pre-designed RTL IP, if available
- **RTL synthesis tool selection**
  - RTL synthesis tool must be selected for custom HW design
- **C code generation**
  - C code for input to RTL synthesis tool is generated
- **Synthesis directives**
  - RTL architecture and clock cycle time is selected
  - UBC calls are treated as special interface operations, to be later expanded during interface synthesis
- **HDL generation**
  - RTL synthesis results in cycle accurate synthesizable Verilog code

HW Synthesis Today

- **HW synthesis tools**
  - SystemC/C to RTL [Mentor, FORTE, Synfora]
    - Based on high level synthesis technology
    - C/C++ support with user constraints (Catapult)
    - SystemC TLM support (Cynthesizer)
    - C based language support (HandelC, Celoxica)
  - SystemVerilog to RTL [BlueSpec]
    - Synthesis from assertions in SystemVerilog (BS Compiler)
    - Correct-by-construction
  - MATLAB to FPGA RTL [Xilinx]
    - Matlab models to DSP HW (AccelChip)

➢ *Component based approach*
➢ *Lack of HW-SW co-synthesis support*
➢ *Capacity and quality issues*

IP synthesis with NISC technology



Source: M. Reshadi

C → RTL

FSM based controller

One-pass synthesis

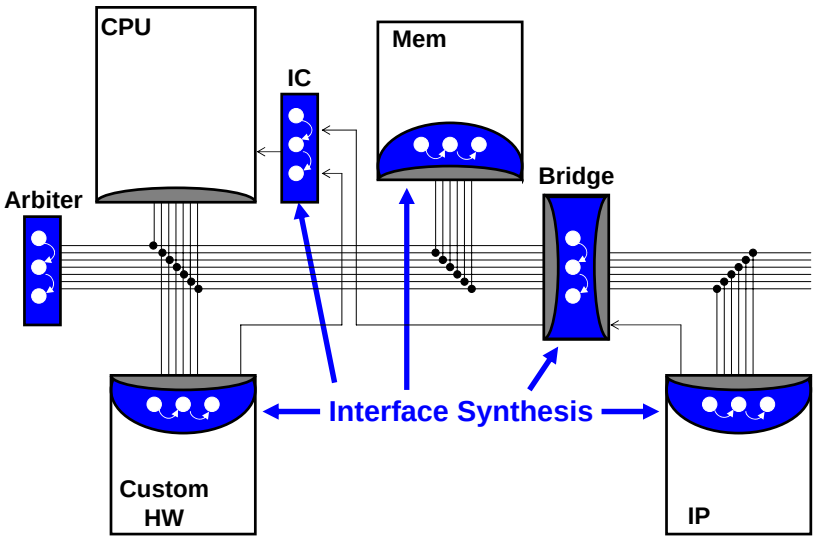
Synthesis→Architecture→Layout

Present C-to-RTL synthesis	C-to-RTL synthesis with NISC technology
Only subset of C	**Full C**
Synthesis for small block/kernel in application	Compile complete application. Any size C possible
No reprogramming / No Reuse (synthesized design cannot be reused for other code)	**Reprogramming / Reuse possible by recompiling** (with matching reduction possible)
Timing problems in controller (long delays in FSM controller)	No timing problems in templatized controller (memory-based, pipelined controller)
Local optimization for blocks/kernels	Global optimization for the whole application
No design for manufacturability (DFM) (layout generated after synthesis)	**Full DFM** (by pre lay out or use of templates)
Estimation based algorithms (no final architecture information available)	Compiling algorithm based on measured metrics (compiler knows the final architecture netlist)
No refinement closure (no layout info., no improvement)	**Good refinement closure** (through preliminary layout)
No designer control over final architecture	Designer control over final architecture
Architecture features not considered (cache, interrupt, I/O, branch prediction,…)	Architecture features possible
No assembly programming	**Available assembly programming**

Memory based controller Synthesis + feedback

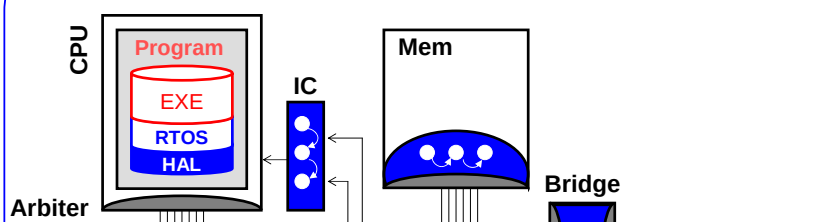
Architecture←Layout←Compile

Cycle-Accurate Interface Synthesis



Interface Synthesis Issues

- **Synchronization**
  - UBC has unique flag for each pair of communicating processes
  - Flag access is implemented as polling, CPU interrupt or interrupt controller
- **Arbitration**
  - Selected from library or synthesized to RTL based on policy
- **Bridge**
  - Selected from library or synthesized using universal bridge generator
- **Addressing**
  - All communicating processes are assigned unique bus addresses
- **SW communication synthesis**
  - UBC functions are replaced by RTOS functions and assembly instructions
- **HW communication synthesis**
  - DMA controller in RTL is created for each custom HW component
  - Send/Recv operations are replaced by DMA transfer states



Final Pin-Accurate Model

Outline

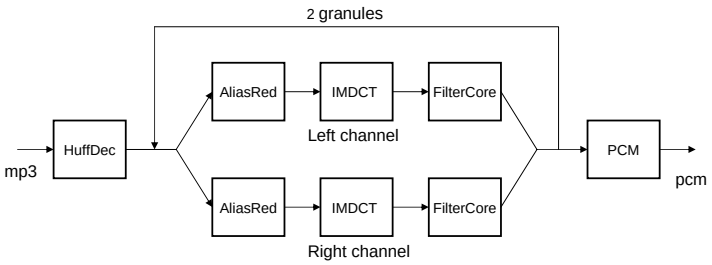
- ✓ Requirements
- ✓ Modeling
- Verification and synthesis
  - ✓ Existing verification methods
  - ✓ TLM verification using transformations
  - ✓ Issues in ES synthesis
  - Synthesis of MP3 player design
  - Results
- Summary, conclusions and outlook

MP3 Player Synthesis

- TLM Input for MP3 application and platform
- Synchronization and Arbitration synthesis
- SW synthesis including RTOS selection and address generation
- HW and Bridge synthesis
- Export to FPGA design tools
- FPGA download and test

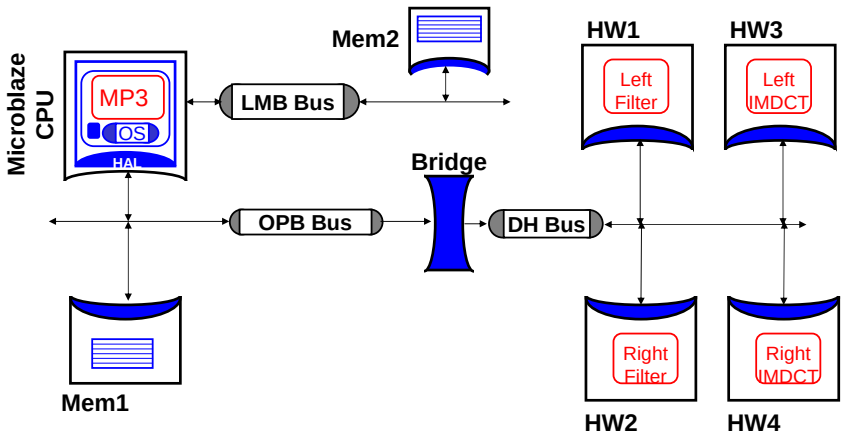
Example: MP3 Decoder

- **Functional block diagram (major blocks only)**



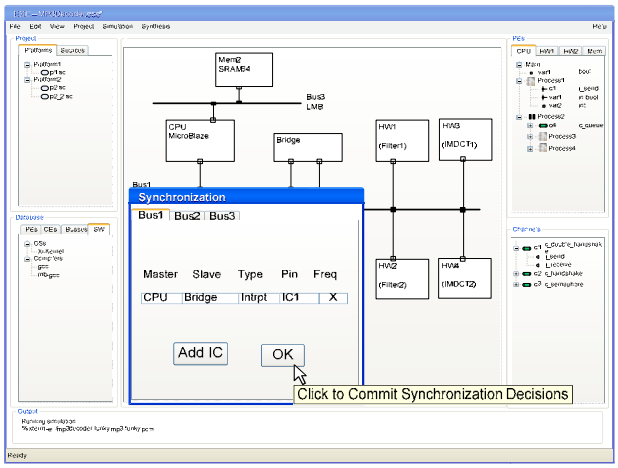
- **Timing constraints**
  - 38 frames per second
  - Frame delay < 26.12ms

MP3 Player TLM



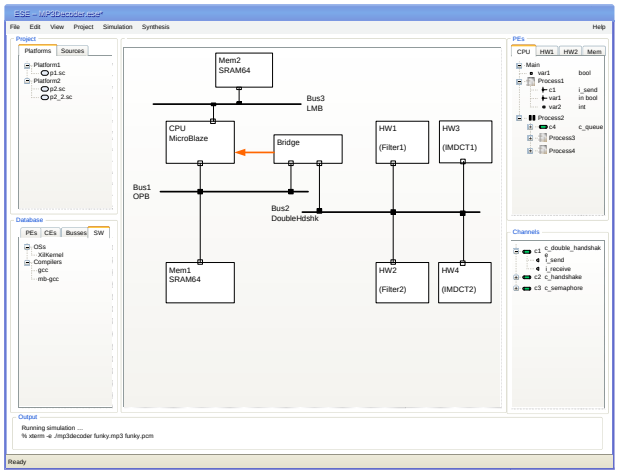
- **MP3 encoder mapped to SW (MicroBlaze), filters and IMDCT to HW**
- **Mem1 (on OPB bus) for data, Mem2 (on LMB bus) for program**
- **Custom HWs on DoubleHdshk (DH) bus, with bridge to OPB**

Synchronization Selection



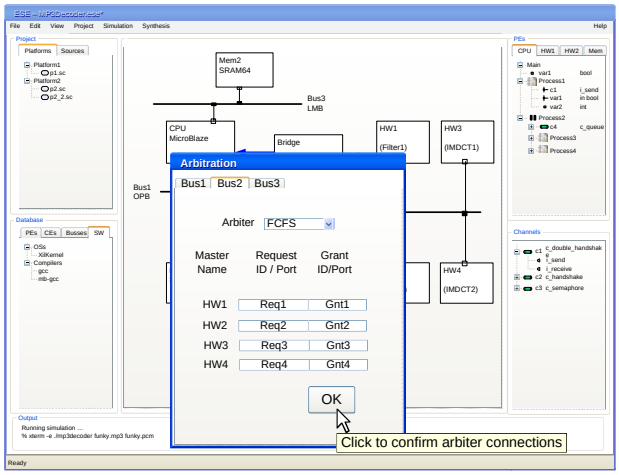
- **Interrupt signals and connections are selected**

Model with Synchronization



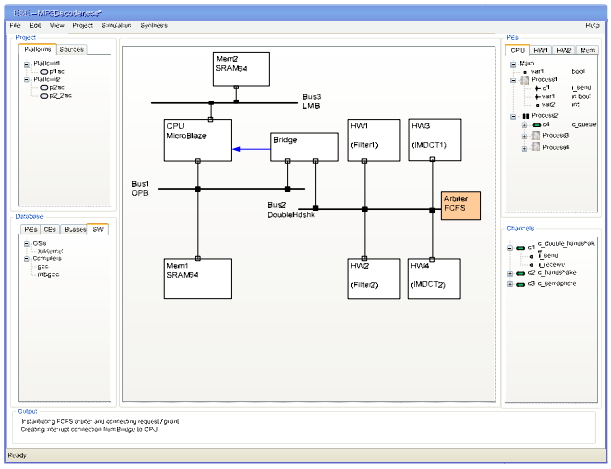
- **Interrupt signals and connections are created**

Arbiter Selection



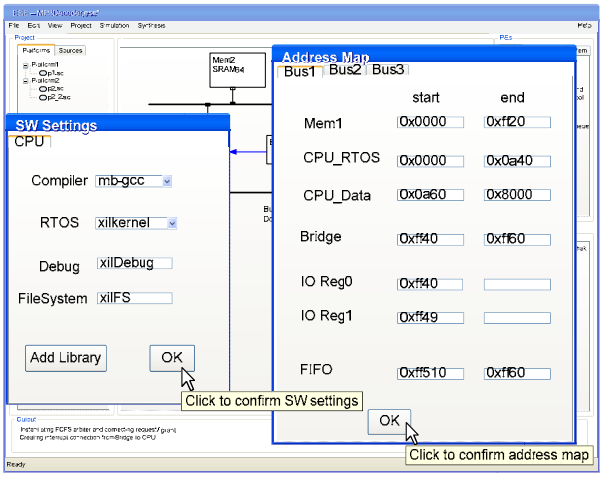
- **Arbiter is selected and request / grant pins are connected**

Model with Arbitration



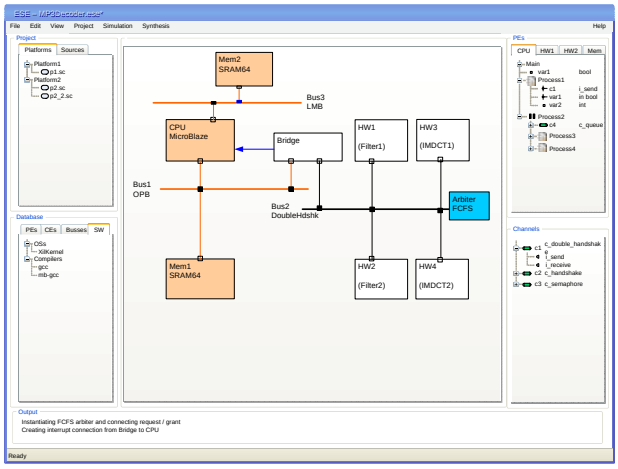
- **The selected arbiter is instantiated and signals are added to create arbiter connections**

SW synthesis



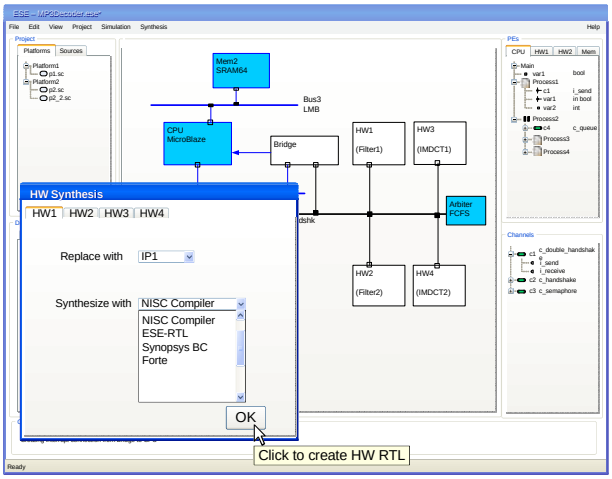
- **Compiler, RTOS and libraries are selected for SW**
- **Default addresses for all addressable memory/bus is generated by ESE**

Model after SW synthesis



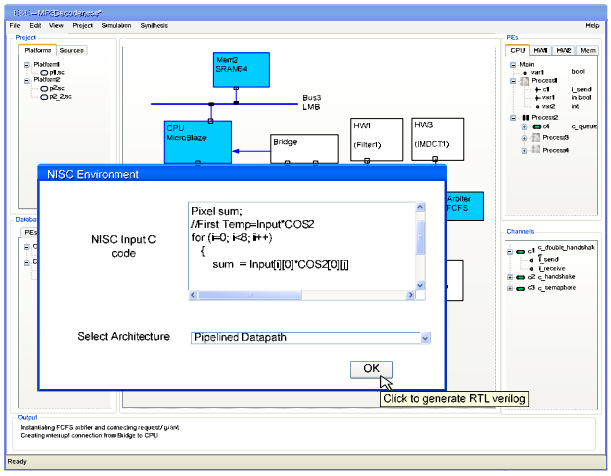
- **SW application and drivers are targeted for RTOS and ready for compilation on MicroBlaze**

HW synthesis



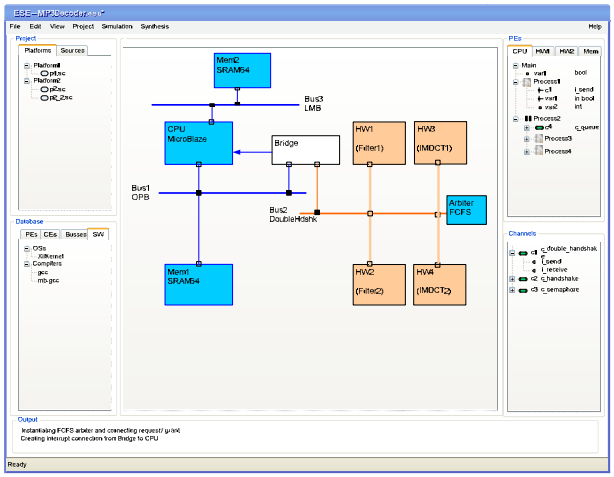
- **RTL code for HW components is generated using NISC compiler**

NISC compilation



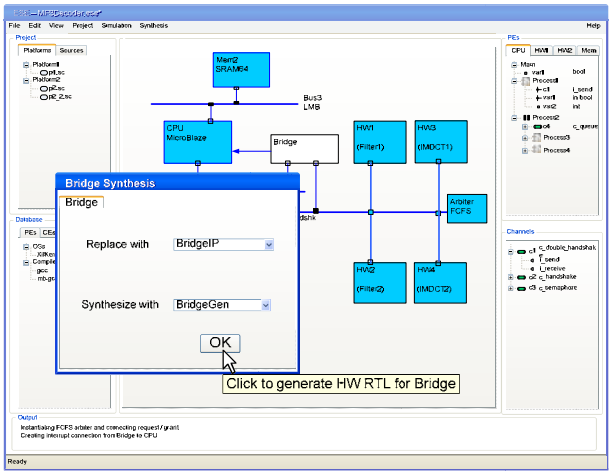
- **NISC compiler generates synthesizable RTL verilog from application code for selected architecture**

Model after HW synthesis



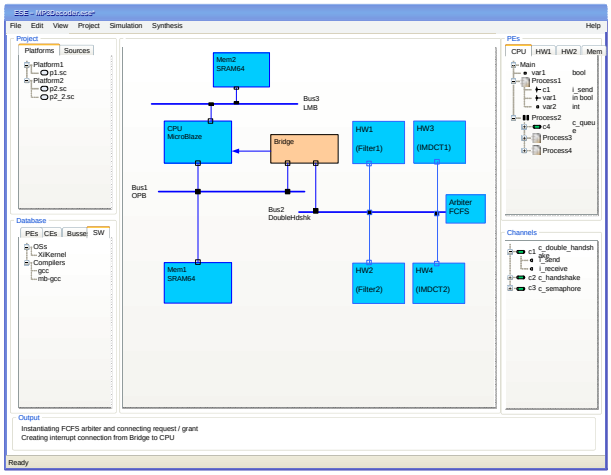
- **Model is updated with RTL code for HW units**

Bridge synthesis



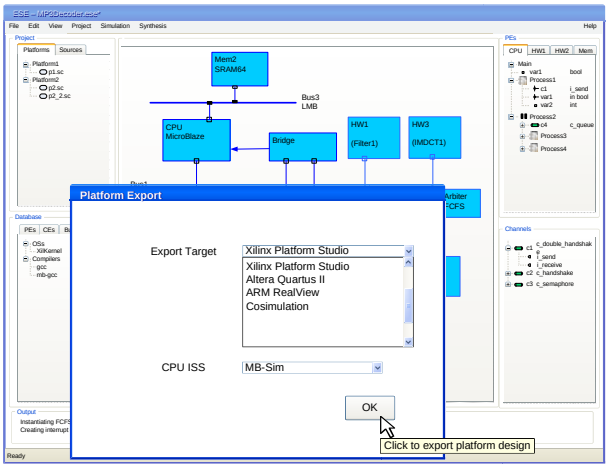
- **RTL code for Bridge is generated using BridgeGenerator**

Model after Bridge synthesis



- **Design is ready for prototyping after SW, HW and Interface synthesis**

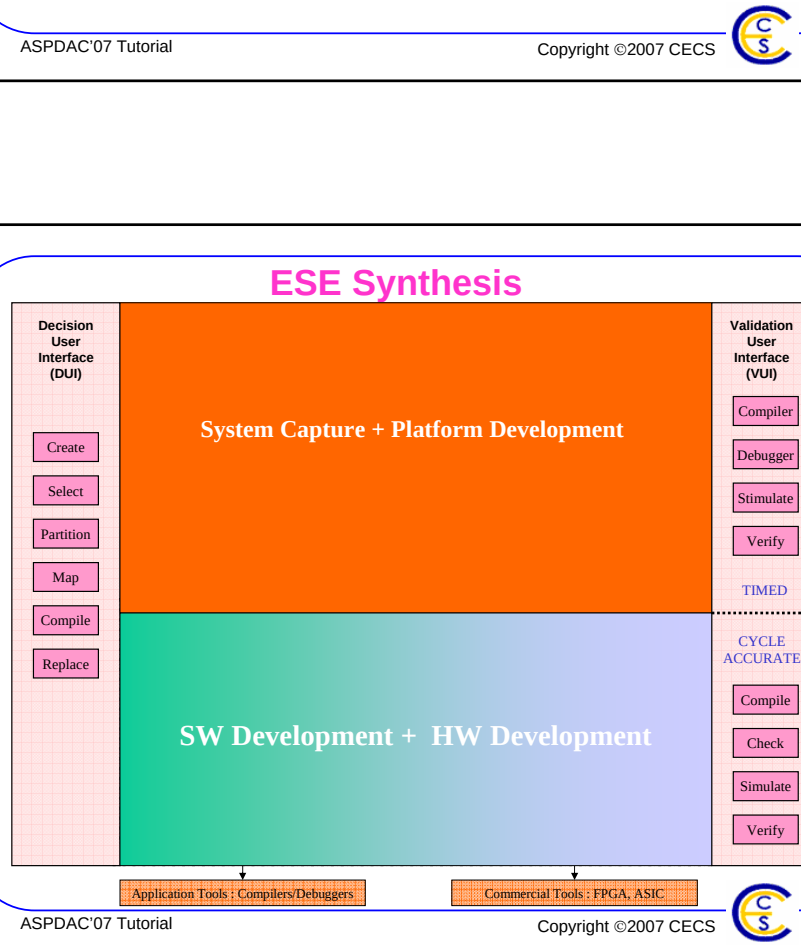
Export to FPGA Design Tools



- **Platform and SW specification files are created for FPGA design tools**
- **C code for Microblaze and Verilog for HWs and Bridge is exported**

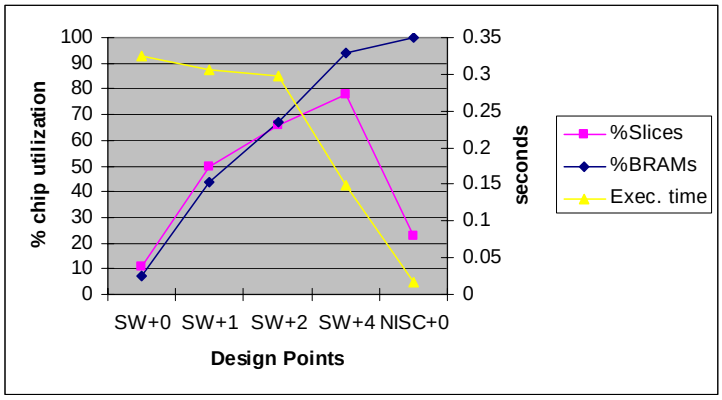
Outline

- ✓ Requirements
- ✓ Modeling
- Verification and synthesis
  - ✓ Existing verification methods
  - ✓ TLM verification using transformations
  - ✓ Issues in ES synthesis
  - ✓ Synthesis of MP3 player design
  - Results
- Summary, conclusions and outlook



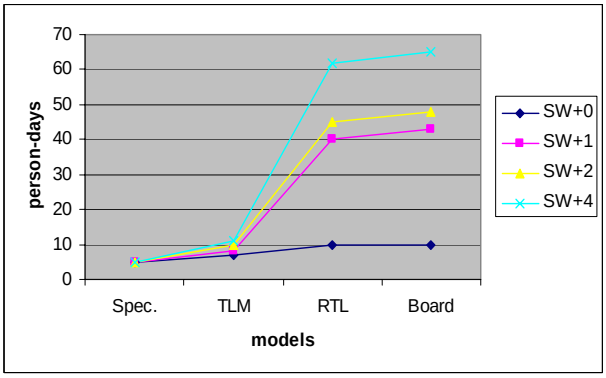
ESE Synthesis

Design Quality



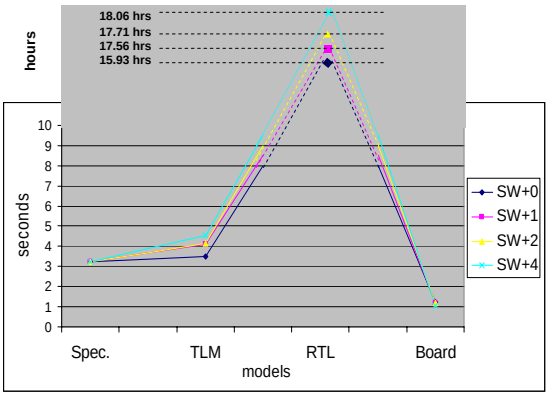
- **Area**
  - % of FPGA slices and BRAMS
- **Performance**
  - Time to decode 1 frame of MP3 data

Manual Development Time



- **Model Development time**
  - Includes time for C, TLM and RTL Verilog coding and debugging

Validation Time



- **Simulation time measured on 3.3 GHz processor**
- **Emulation time measured on board with Timer**

Summary and Conclusions

- **Technology advantages**
  - No need to code and debug large RTL HDL models
  - Bridge and interface synthesis allows flexibility to include heterogeneous IP in the design
  - No need for SW developers to understand HW details
  - Easy application upgrade at TL
  - C and graphical input of TL model allows even non-experts to develop and test HW/SW systems
- **Simplified system development**
- **Huge productivity gains**
- **Short time to market**

References

- **Embedded System Verification**
  - Devadas, Ma, Newton, "On the verification of sequential machines at different levels of abstraction", 24th DAC, pp.271-276, June 1987
  - Clarke, Grumberg, Peled, "Model Checking" , MIT Press
  - K.L. McMillan, "Symbolic Model Checking: An approach to the State Explosion Problem" , Kluwer Academic 1993
  - McFarland, "Formal Verification of Sequential Hardware: A tutorial", IEEE Transaction on CAD, pp. 633-653, May 1993.
- **HW Synthesis**
  - P. Bannerjee et al. "A Behavioral Synthesis Tool for Exploiting Fine Grain Parallelism in FPGAs ," *LNCS*, Sept. 2005.
  - J. Cong et al. "*"xPilot: A Platform-Based Behavioral Synthesis System"*, SRC TechCon'05 .
- **System synthesis**
  - F. Balarin, H. Hsieh, L. Lavagno, C. Passerone, A. Pinto, A. Sangiovanni-Vincentelli, Y. Watanabe, G. Yang. "Metropolis: a Design Environment for Heterogeneous Systems," *Multiprocessor Systems-on-Chips* (Editors: W. Wolf, A. Jerraya), Morgan Kaufmann, 2004.
  - P. Chou, R. Ortega, G. Borriello. "The Chinook hardware/software co-synthesis system ," *ISSS 1996*.
  - D. Gajski, "ESE Back End," http://www.cecs.uci.edu/pub_slides, March 2006.

Outline

- ✓ **Requirements**
- ✓ **Modeling**
- ✓ **Verification and synthesis**
- **Summary, conclusions and outlook**

Embedded System Design: Summary, Conclusions and Outlook

Daniel D. Gajski
Center for Embedded Computer Systems
University of California, Irvine

Outline

- ✓ **Requirements**
- ✓ **Modeling**
- ✓ **Verification and synthesis**
- • **Summary, conclusions and outlook**
  - Models
  - Platforms
  - Tools
  - Benefits
  - Conclusion

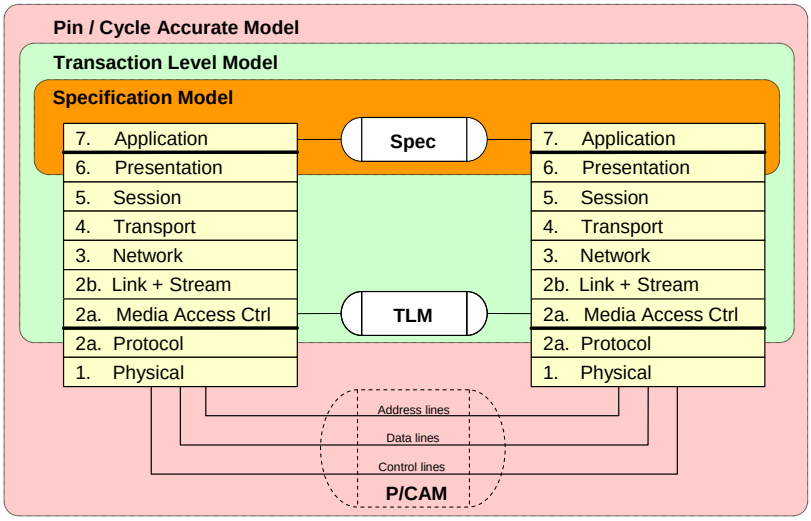
How many models?

Minimal set for any methodology

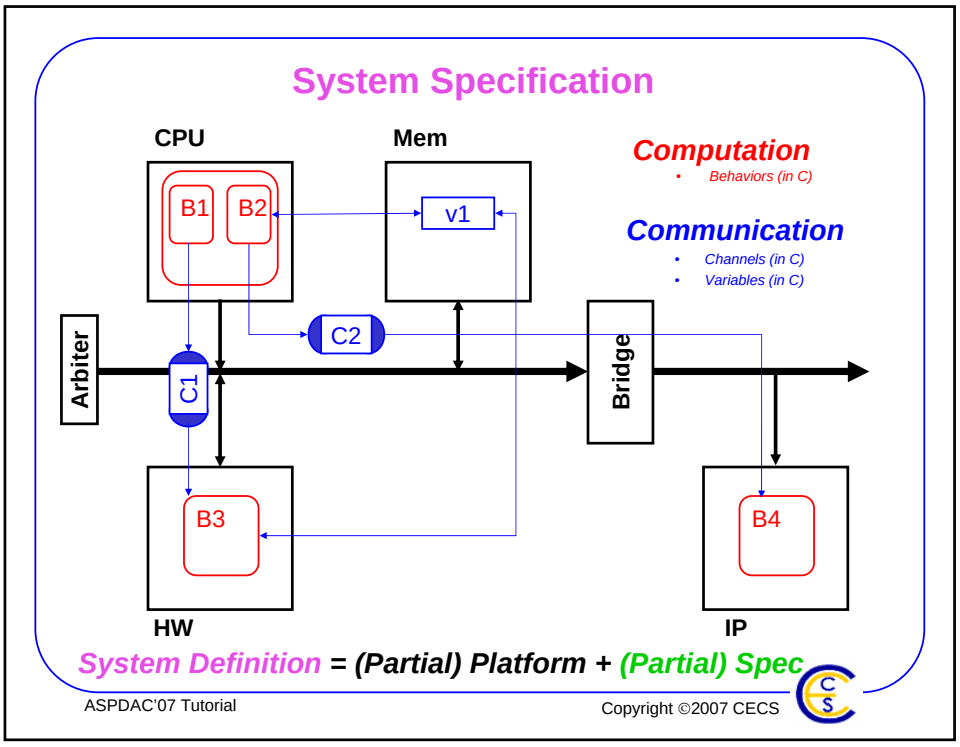
(3 are enough)

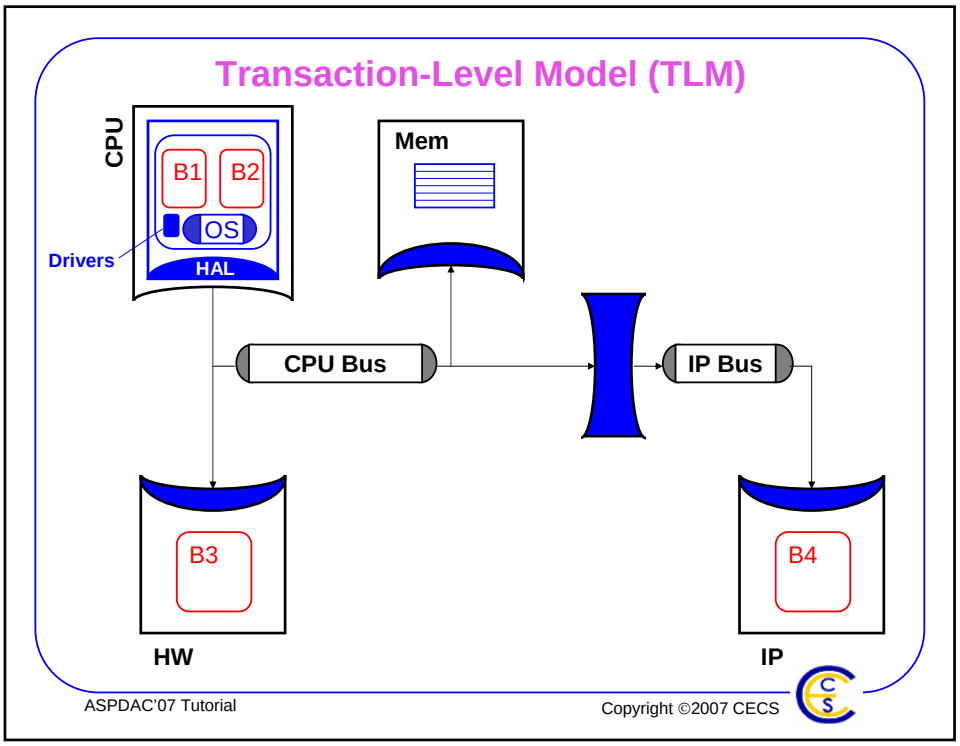
- System specification model (application designers)
- Transaction-level model (system designers)
- Pin&Cycle accurate model (implementation designers)

Three Models (with Respect to OSI)

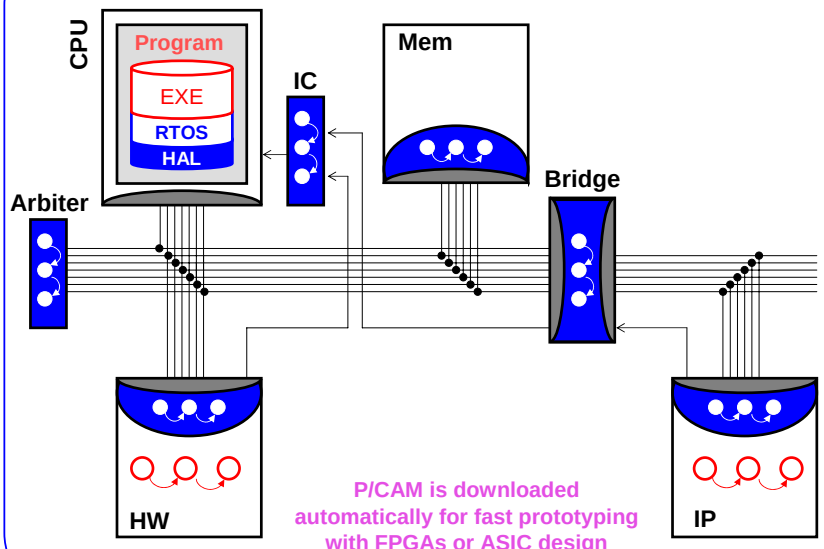


Source: G Schirner


System Specification
CPU
Mem
Computation
Behaviors (in C)
B1
B2
v1
Communication
Channels (in C)
Variables (in C)
C2
Arbiter
C1
Bridge
B3
B4
HW
IP
System Definition = (Partial) Platform + (Partial) Spec
ASPDAC'07 Tutorial
Copyright ©2007 CECS


Transaction-Level Model (TLM)
CPU
B1
B2
OS
HAL
Drivers
Mem
CPU Bus
IP Bus
B3
B4
HW
IP
ASPDAC'07 Tutorial
Copyright ©2007 CECS

Pin/Cycle Accurate Model (P/CAM)



P/CAM is downloaded automatically for fast prototyping with FPGAs or ASIC design

Source: D. Gajski et al.

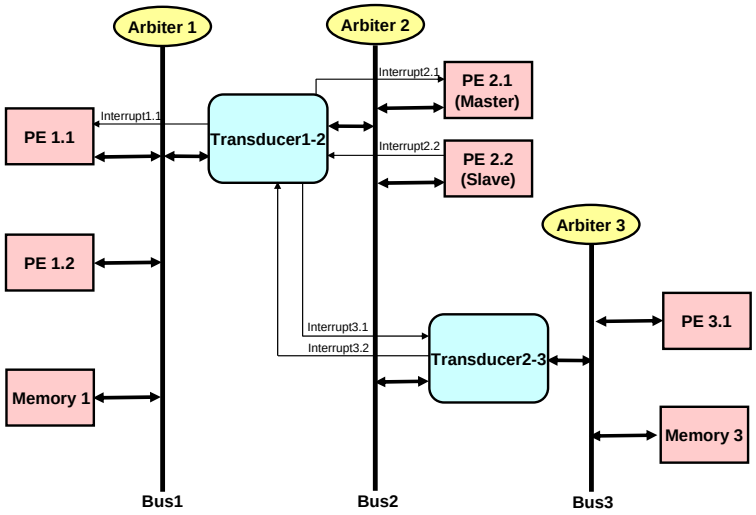
How many components?

Minimal set for any design

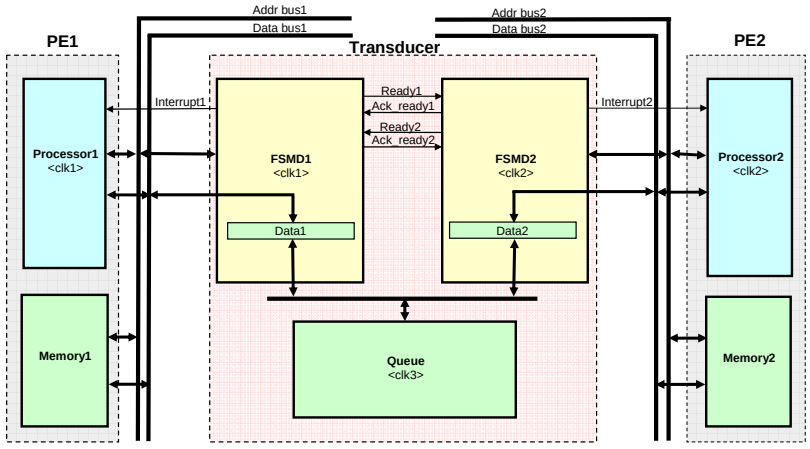
(4 are enough)

- Processing element (PE)
- Memory
- Transducer / Bridge
- Arbiter

General System Model



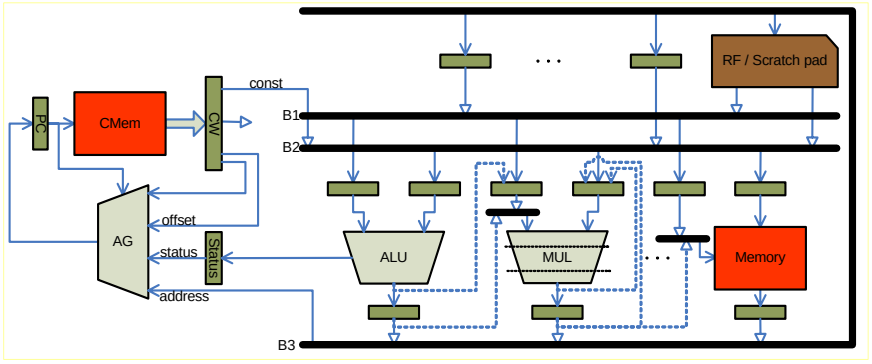
Transducer Model



Source: H. Cho

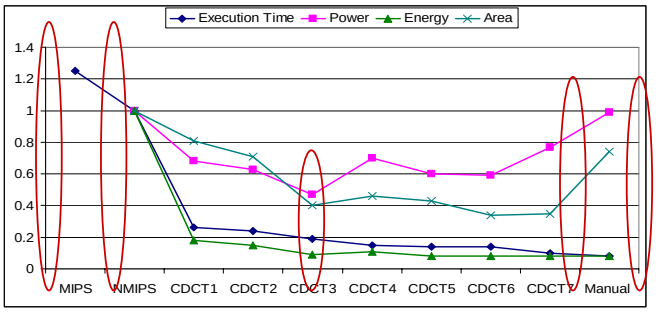
NISC technology architectures

- Direct compilation of C to HW (fastest possible execution)
- Statically and dynamically reconfigurable (anytime, anywhere)
- Designed for manufacturability (solving timing closure)



Programmable controller | Datapath | Multi-cycle units | Pipelined units | Controller pipelining | Datapath pipelining | Data forwarding

DCT with NISC technology



	Performance	Power saving	Energy saving	Area reduction
NMIPS vs. MIPS	1.25X	NA	NA	NA
CDCT3 vs. NMIPS	5.3X	2.1X	11.6X	2.5X
CDCT7 vs. NMIPS	10X	1.3X	12.8X	3X
CDCT7 vs. Manual	0.83X	1.3X	0	2.1X

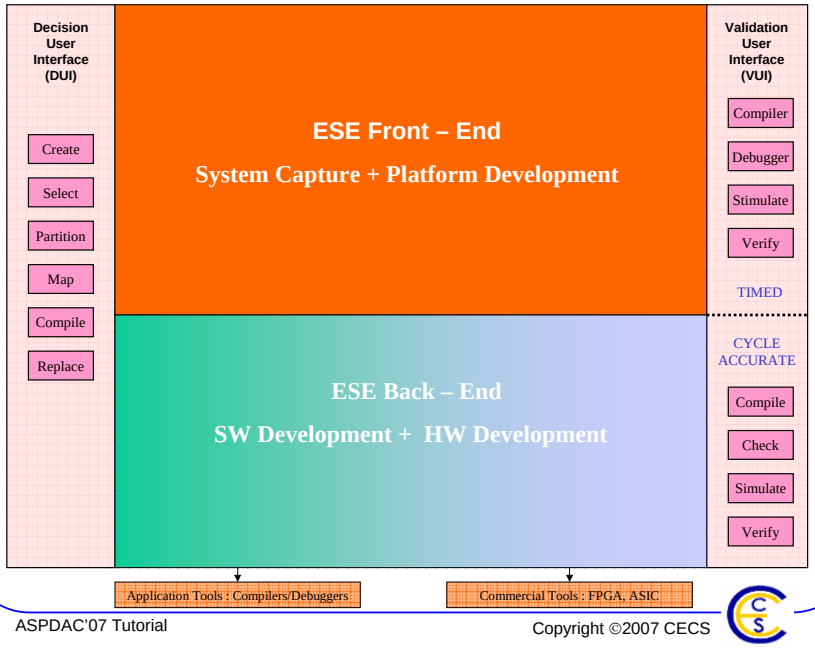
How many tools?

Minimal set for any methodology

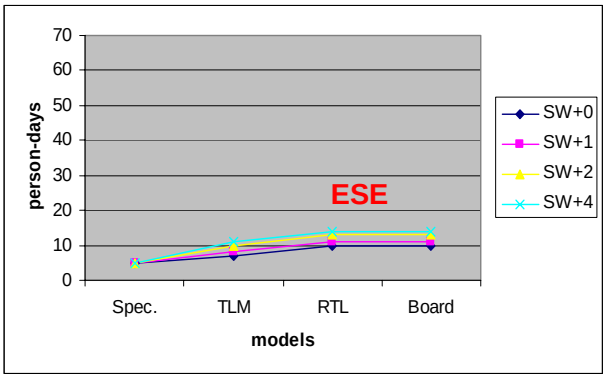
(2 are enough)

- Front-End (for application developers)
  - **Input: C, C++, Mathlab, UML, …**
  - **Output: TLM**
- Back-End (for SW/HW system designers)
  - **Input : TLM**
  - **Output: Pin/Cycle accurate Verilog/VHDL**

ES Environment



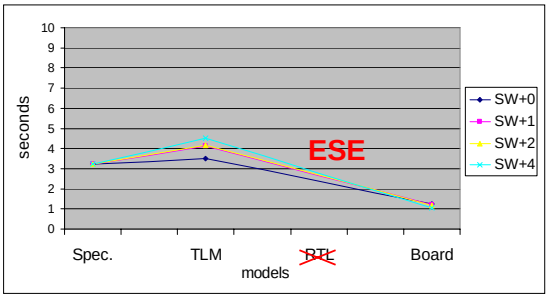
Development Time with ESE



- **ESE drastically cuts RTL and Board development time**
  - Models can be developed at Spec and TL
  - Synthesizable RTL models are generated automatically by ESE

Source: S. Abdi

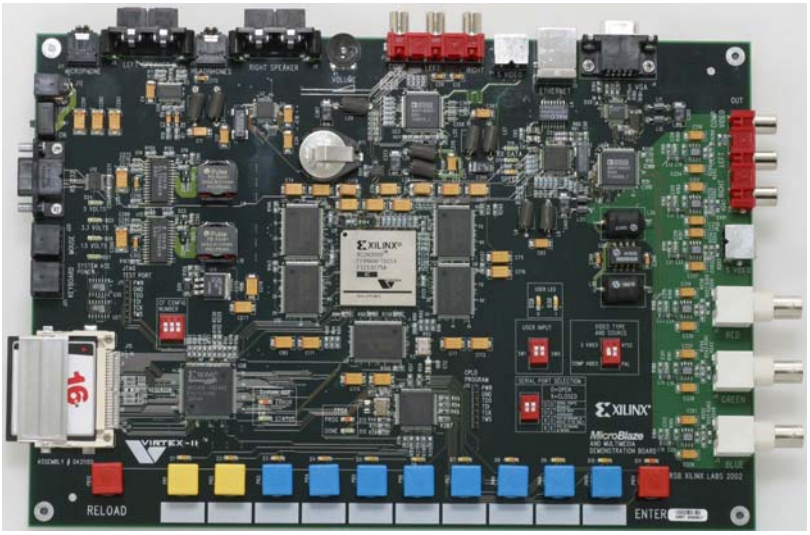
Validation Time with ESE



- **ESE cuts validation time from hours to seconds**
  - No need to verify RTL models
  - Designers can perform high speed validation at TLM and board

Source: S. Abdi

Benefit: Spec-to-Prototype in 1 Week



Summary

- **Underlying concepts**
  - Well defined models, rules, transformations, refinements
  - Analogous to layout, logic, RTL
  - System level complexity simplified
- **Proof of concept demonstrated**
  - Embedded System Environment (ESE)
  - Automatic model generation
  - Model synthesis and verification
  - Universal IP technology (NISC)
  - Productivity gains greater then 1000
- **Benefits**
  - Large productivity gains
  - Easy design management
  - Easy derivatives
  - Shorter TTM

Conclusions

- Extreme makeover is necessary for a new paradigm, where
  - SW = HW = SOC = Embedded Systems
  - Simulation based flow is not acceptable
  - Design methodology is based on scientific principles
- Model algebra is enabling technology for
  - System design, modeling and simulation
  - System synthesis, verification, and test
- What is next?
  - Change of mind
  - Application oriented EDA
  - Looking for early adapters

Outline

- ✓ Requirements
- ✓ Modeling
- ✓ Verification and synthesis
- ✓ Summary, conclusions and outlook

Thank You

Daniel D. Gajski
Andreas Gerstlauer
Samar Abdi
Center for Embedded Computer Systems
University of California, Irvine