Embedded System Design

Modeling, Synthesis, Verification

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

Chapter 8: Embedded Design Practice

7/8/2009

Embedded Design Practice

- **Introduction**
- System Level Design Tools
- Embedded Software Design Tools
- Hardware Design Tools
- Case Study
- Summary

Introduction

- **System-level tools are available in three different forms:**
  - System-level design
  - Software design
  - Hardware design
- **Some academic tools demonstrate complete process: MoC-to-RTL including custom SW and HW components**
- **Automation of system-level tasks shows large gains as demonstrated on examples of JPEG and MP3**
- **Results also demonstrate potentially large impact on embedded systems technology**

Embedded Design Practice

Overview

- **Electronic System-Level (ESL) design tools**
  - Many that provide single hardware unit only (see HW design tools)
  - ➢ True system-level design across hardware and software boundaries
- **System-level design flow**
  - Frontend
    - – Application & architecture mapping
    - – Design space exploration (DSE)
    - ➢ System models (TLMs) for virtual prototyping
  - Backend
    - – Hardware and software synthesis
    - – Commercial or proprietary (see SW & HW design tools)
    - ➢ Physical system prototype or implementation

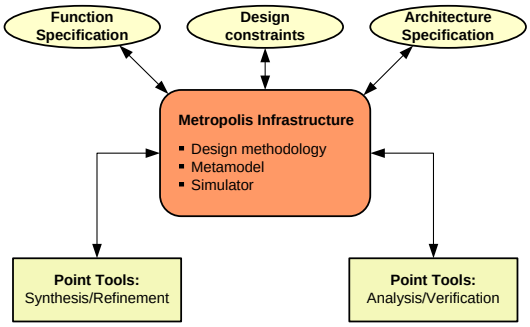
➢ **Commercial tools for modeling and simulation**
➢ **Academic tools for synthesis and verification**

Academic Tools

- **Metropolis**
  - Platform-based design (PBD)
- **SystemCoDesigner**
  - Dynamic dataflow MoC
  - Automated design space exploration
- **Daedalus**
  - KPN MoC for streaming, multi-media applications
  - IP-based MPSoC assembly
- **PeaCE**
  - “Ptolemy extension as a Codesign Environment”
  - Recent extensions for software development (HOPES)
- **SCE**
  - SpecC-based “System-on-Chip Environment”
  - Successive, stepwise Specify-Explore-Refine methodology

Academic Tools: Metropolis

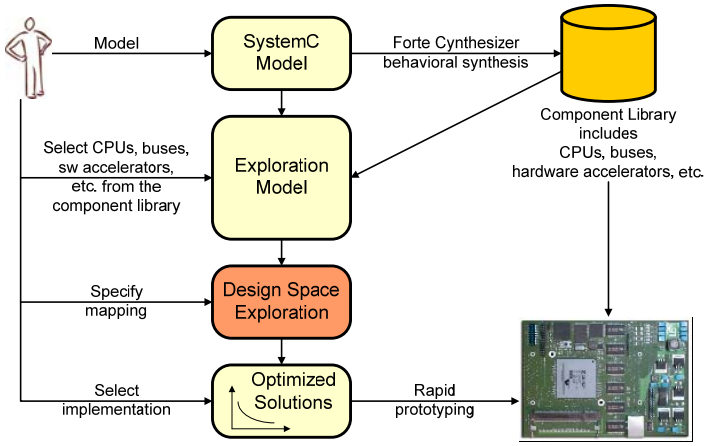
- **Platform-based**
  - Pre-defined target architecture
  - ➢ Reuse
- **Meet-in-the-middle**
  - Platform mapping and configuration



- **General, proprietary meta-modeling language**
  - Capture function, architecture and mapping
- **Modeling framework**
  - Built-in parsing and simulation
  - Back-end point tool integration

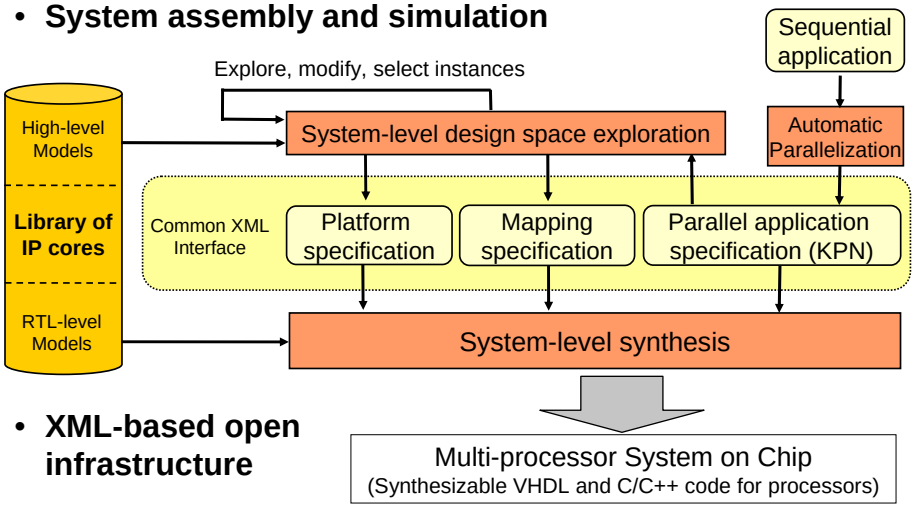
Academic Tools: SystemCoDesigner

- **SysteMoC input model**
  - Dynamic dataflow MoC (actors + FSMDs) in SystemC
- **Fully automatic, multi-objective design space exploration**
  - Genetic algorithms to obtain Pareto-optimal design solutions



Academic Tools: Daedalus

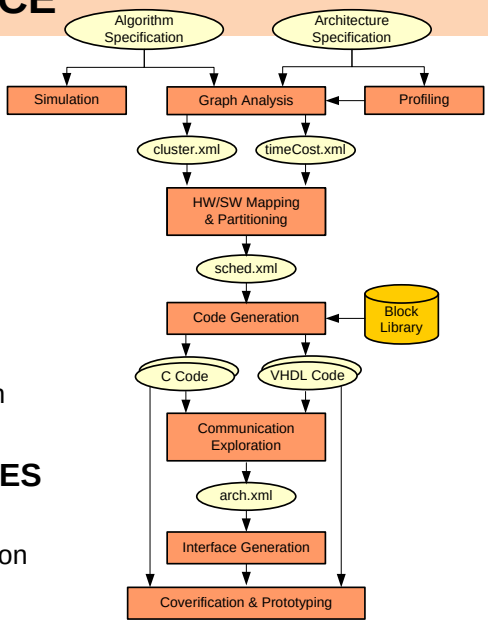
- **KPN input model**
- **System assembly and simulation**



- **XML-based open infrastructure**

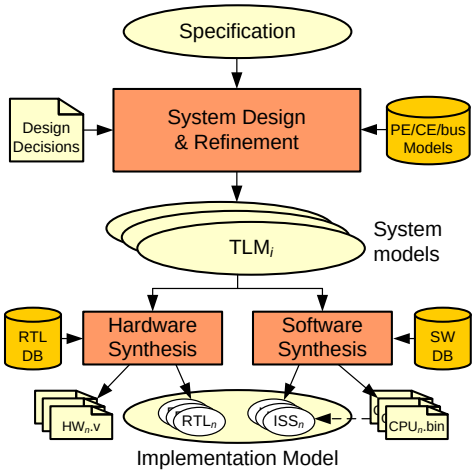
Academic Tools: PeaCE

- **Ptolemy-based**
  - Heterogeneous SDF+FSM application MoC
- **Stepwise flow**
  - Application partitioning
  - Communication architecture exploration
  - Code and interface generation
- **Software extensions: HOPES**
  - Parallel programming API
  - Multi-processor code generation



Academic Tools: SCE

- **SpecC based**
  - PSM input MoC
  - Specify-Explore-Refine
  - Interactive, successive, stepwise refinement
- **Frontend**
  - Compile specification onto user-defined MPSoC arch.
  - Automatically generate TLM
- **Backend**
  - Hardware/software synthesis
  - RTL + ISS implementation



➢ **Commercial derivative: SER (JAXA)**

Commercial Tools (1)

- **CoFluent**
  - SystemC-based modeling and simulation
    - Networks of timed processes
    - Communication through queues, events, variables
  - Early, high-level interactive design space exploration
    - Graphical application, architecture and mapping capture
    - Fast TLM simulation with estimated timing
- **Space Codesign**
  - Graphical application, architecture and mapping capture (Eclipse)
    - Process network with message-passing or shared-memory channels
  - SystemC TLM simulation
    - Annotated, host-compiled or cycle-accurate ISS models
  - FPGA-based prototyping
    - Cross-compilation and third-party hardware synthesis (Forte/Catapult)

Commercial Tools (2)

- **CoWare**
  - Virtual system platforms
    - SystemC TLM capture, modeling and simulation
    - Extensive library of IP, processor and bus models
    - Application-specific processor ISS models (LISAtek acquisition)
  - Proprietary SystemC simulation framework
    - Optimized SystemC kernel
    - Graphical debugging, visualization and analysis capabilities
- **Soc Designer**
  - Proprietary, C++ based modeling and simulation
    - Fast, statically scheduled cycle-accurate simulation
    - Special cycle-callable component models
- **VaST and Virtutech**
  - Proprietary SW-centric virtual platform modeling and simulation
    - Fast, cycle-approximate binary translated or compiled ISS + peripherals

Outlook

- **State of the art**
  - Commercial focus still only on modeling and simulation
  - Academic approaches towards true system-level design
  - Emerging commercial solutions for backend HW/SW design
- **Future complete, automated system design flows**
  - Further research and development of system-level synthesis and design space exploration solutions
  - Continuing technology transfer from academia into commercial settings and startups

➢ **Productivity gains**

➢ **Closing gap between application and implementation**

Embedded Design Practice

- ✓ Introduction
- ✓ System Level Design Tools
- **Embedded Software Design Tools**
  - Overview
  - Academic Tools
  - Commercial Tools
  - Outlook
- Hardware Design Tools
- Case Study
- Summary

Overview

- **Tight connection to underlying HW**
  - Processor, custom hardware, physical process integration
  - Requires:
    - Processor-specific code generation
      - » e.g. DSP v.s. general purpose processor
    - Processor-specific compiler (cross compiler)
    - Processor-specific simulator
      - » Virtual platform
      - » Instruction Set Simulator (ISS)
    - Non-intrusive analysis/tracing
    - Real-time analysis
  - Specialized point solutions
    - Processor vendor: e.g. ARM RealView Development Suite
    - FPGA Vendor: e.g. Xilinx EDK
    - OS Vendor: WindRiver WorkBench

Academic Tools (1)

- **POLIS**
  - HW/SW co-design environment
  - Input: Esterel or graphical FSM notation
  - Centered around Codesign Finite State Machine (CFSM)
    - Locally synchronous, globally asynchronous
  - Formalism for verification, co-simulation, partitioning and synthesis

- **METROPOLIS**
  - Platform based design
  - Meta-model; supports many MoCs
  - Separate function, architecture and MoC into separate inputs
  - Co-simulation heterogeneous PEs with different MoCs

Academic Tools (2)

- **DESCARTES**
  - Targets real-time signal processing systems
  - Input:
    - Asynchronous Data Flow (ADF), and
    - extension of Synchronous Data Flow (SDF)
  - Computation node scheduling observing
    - Latency
    - Throughput
    - Memory consumption
  - C code generation of each computation node

Academic Tools (3)

- **Software generation from SystemC models**
  - Herrara et al.
    - Single source solution
      - » Same C++ code on SystemC and on target
    - Simplifies debugging / maintenance
    - Overload SystemC primitives for target implementation
    - Subset of SystemC
  - PROTOS (Krause et al.)
    - Input
      - » SystemC threads communicating though point-to-point channels
    - Parses SystemC, generates RTOS targeted code for selected RTOS
      - » Replaces SystemC calls (comm., threads) with RTOS equivalent calls
      - » Attempts to recreate SystemC events
    - Captures RTOS characteristics in XML
      - » API call signatures, thread fork join, static / dynamic

Academic Tools (4)

- **Eclipse**
  - Multi-language development platform
    - IDE: Compiler, debugger, source code browser
    - Extensible with well defined plug-in system
  - Free, open source; managed by Eclipse Foundation
  - Main focus JAVA, but supports many other languages
  - Very popular framework for custom (also embedded) extensions in academic and commercial projects, e.g.
    - Tensilica Xtnesa Xplorer IDE
      - » Custom processor generation, cross compilation and debugging
    - Greensys AUTOSAR Builder
      - » Develop AUTOSAR (automotive) software components
      - » Capture system and application level description aiding integration

Commercial Tools (1)

- **MathWorks: Real-Time Workshop**
  - Simulink
    - Model-based design tool
    - Block diagram capturing of system functionality
      - » Compose of predefined blocks (e.g. filters, control functions)
    - Hierarchical composition
    - Discrete time and continues time models
  - Real-Time Workshop generates target code based on Simulink model
    - ANSI C / C++
    - Stand alone / RTOS based
- **dSpace Cooperation: TargetLink**
  - Integrates into Matlab/Simulink
  - Automotive focus
    - Supports OSEK/VDX compliant OS
    - Target code for Electronic Control Units (ECU)
    - Extensions to support AUTOSAR

Commercial Tools (2)

- **Esterel Technologies:**
  Software Critical Application Development Environment (SCADE)
  - Targets safety critical applications
  - Graphical notation of hierarchical data flow and safe state machines
    - Rich set of predefined blocks (operators, linear functions, filters, state machines, model composition)
  - Internally based on Lustre, synchronous data flow language
  - KCG: C code generator certified for airborne systems
    - Generates code for each block
  - Worst Case Execution Time (WCET) analysis integration
  - Extensible through gateway (e.g. Matlab/Simulink, UML/SysML)

Commercial Tools (3)

- **UML/SysML Products**
  - Universal Modeling Language (UML)
    - Specification of software systems, early in process
      - » Construction, documentation
    - Modeling language, not programming language
    - Defines 13 diagram types
      - » System structure, System behavior, Interaction of system elements
    - Use std. programming language to capture algorithms
  - Systems Modeling Language (SysML)
    - Extension and subset of UML (extending SW focus to System)
      - » E.g. adds: requirement diagram (perf. analysis), MoC for continues systems
  - Many commercial tools for capture, analysis, validation and framework code generation:
    - IBM Telelogic Rhapsody
    - Spark Systems' Enterprise Architect
    - Gentleware's Poseidon
    - Artisan Software's Artisan Studio

Outlook

- **Status**
  - Vendor specific solutions / domain specific solutions
    - E.g. processor, FPGA fabric or OS vendors
    - Automotive, signal processing
- **More attention to reusable and scalable implementations**
  - Component-based approaches (e.g. AUTOSAR)
  - Integrated documentation / design (e.g. UML, SysML)
- **Platform complexities increase**
  - Many-core platforms, heterogeneity
  - Manual implementation increasingly inefficient
- **Increasing focus on generation / synthesis**
  - Develop systems as composition of algorithms
  - Automatic generation of embedded software
  - Focus on essential function aspects instead of implementation detail

Embedded Design Practice

- ✓ Introduction
- ✓ System Level Design Tools
- ✓ Embedded Software Design Tools
- **Hardware Design Tools**
  - Overview
  - Academic Tools
  - Commercial Tools
  - Outlook
- Case Study
- Summary

Overview (1)

- **Historical Perspective: Four Phases**
  - Concept Phase (1970s)
    - Basic definition for languages, methods, tools
    - Instruction-Set Processor Specification/ CMU RT-CAD System (1976)
    - MIMOLA at U of Kiel (1978)
  - Algorithms Phase (1980s)
    - Allocation, binding, scheduling algorithms
    - Design flow for controllers, datapaths, custom processors
    - Early tools: Yorktown Silicon Compiler (IBM), Cathedral (IMEC), System Architects' Workbench (CMU), Design Environment (U of Karlsruhe)

Overview (2)

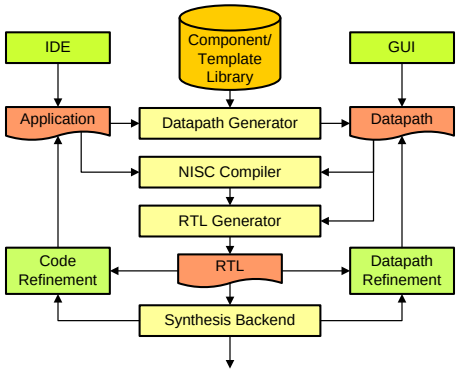
- Consolidation Phase (1990s)
  - HLS books: System Architect's Workbench (1990), and others
  - Commercial tools: Behavioral Compiler (Synopsys), Monet (Mentor), Cyber Synthesis Tool (NEC)
  - Obstacles: Tool-dependent language subsets, simple controller and datapath architecture, non-programmable, fixed, FSM controller, interfacing components not defined, consumer market not prepared
- Acceptance phase (2000s)
  - HLS tools acceptance forced by system complexities
  - Standard programming or system languages as input (C/C++, SystemC)
  - More sophisticated algorithms
  - Complex IPs and custom architectures with programmable controllers

Academic Tools

- **GAUT**
  - Custom processors for digital signal processing application
  - Bit-accurate specification in C/C++
  - Pipelined architecture of processor, memory and interface unit
- **No-Instruction Set Computer (NISC)**
  - Custom processor with control memory vs. program memory
- **SPARK High-Level Synthesis**
  - C-to-VHDL HLS framework with pre-synthesis optimizations
- **xPilot Synthesis System**
  - Platform-based behavioral synthesis with multiple metric optimization

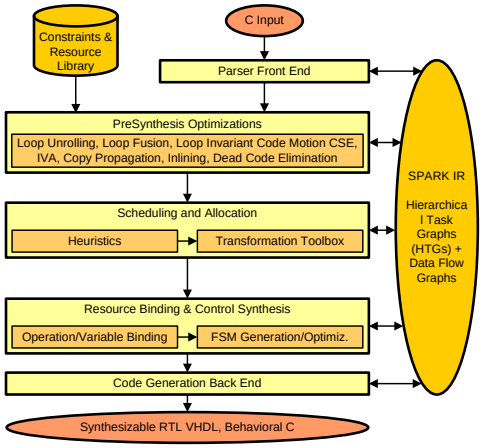
Academic Tools: NISC

- **NISC features programmability**
  - Parametrizable architecture
  - Programmable controller with control-word memory
    - Large codes accommodated
- **NISC features metric closure**
  - Separation of allocation from binding & scheduling
  - Datapath completely defined before binding and scheduling by compiler
  - Architecture-cell concept
- **NISC tools**
  - Datapath generator generates datapath from source
    - Manual override possible
  - Retagertable cycle-accurate compiler
  - RTL generator for FPGA prototyping
  - Optimization by manual code or datapath refinement



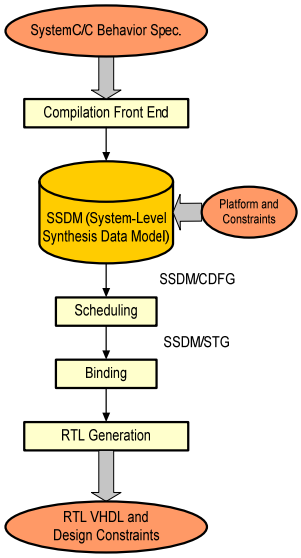
Academic Tools: SPARK

- **SPARK is HLS framework**
  - For multimedia and image applications
  - For control intensive functional blocks
- **Input: ANSI-C, resource library, constraints and user directives**
- **Output: Synthesizable RTL VHDL code**
- **Tasks:**
  - Pre-synthesis optimizations
  - Scheduling and allocation
  - Binding and control synthesis
  - RTL generation



Academic Tools: xPilot

- **Platform-based behavioral synthesis**
- **Input: C or SystemC**
- **Output: RTL and constraints files**
- **SSDM models process network**
- **Tasks:**
  - Pre-synthesis optimization by LLVM compiler
  - Physically-aware optimizations during scheduling and binding
  - RTL generation with physical location constraints



Commercial Tools (1)

- **Catapult Synthesis**
  - C++-to-RTL
  - Block architecture for different C functions with communication channels between
  - User directives for interface and memory mappings, loop unrolling and pipelining, HW hierarchy, block communication, resource allocation, latency and cycle constraints
- **Cynthesizer**
  - Pin- and protocol-accurate SystemC as input
  - Hybrid scheduling approach for protocol and computation sections
  - Gate-level library generated for estimation
  - Custom datapath components are created from user indicated C++ code

Commercial Tools (2)

- **PICO**
  - C-to-RTL mapping under performance constraints (throughput, cycle-time) for data streaming applications
  - Complex application engines for system platforms
  - Compile-time configurable architecture template based on Khan-process-network model
  - Advanced parallelizing compiler
- **CyberWorkBench (CWB)**
  - C-based HLS and verification tool (“All-in-C” approach)
  - Legacy RTL blocks as black boxes
  - Cycle-accurate simulation model generated for validation
  - Input C code verified through assertions
- **Bluespec**
  - An alternative to loop-and-array paradigm
  - Bluespec System Verilog (BSV) language specifies concurrent system behavior as a collection of rewrite rules
  - BSV is translated into Verilog or SystemC RTL by Bluespec Compiler

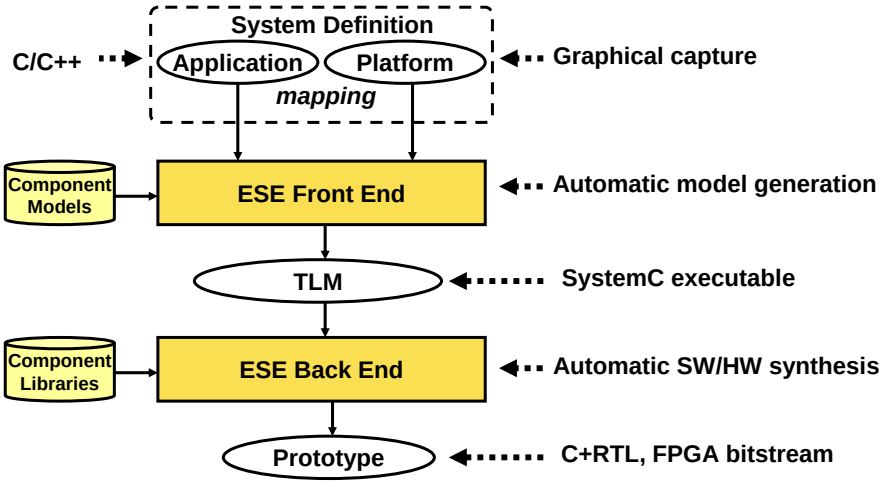
Outlook

- **Status**
  - Designers acceptance of C-to-RTL concepts
  - Increasing supply of HLS tools
  - C/C++ is favored as input description
  - Pre-synthesis optimization for better results
- **Open Issues**
  - Synthesized architecture needs additional features
    - Control and datapath pipelining
    - Programmable controllers
    - Architecture cells or custom-processor templates
    - Retargetable compilers
  - Platform generation and synthesis
    - Merging components into platform and mapping application
    - Interfacing synthesized components (Interface cells)

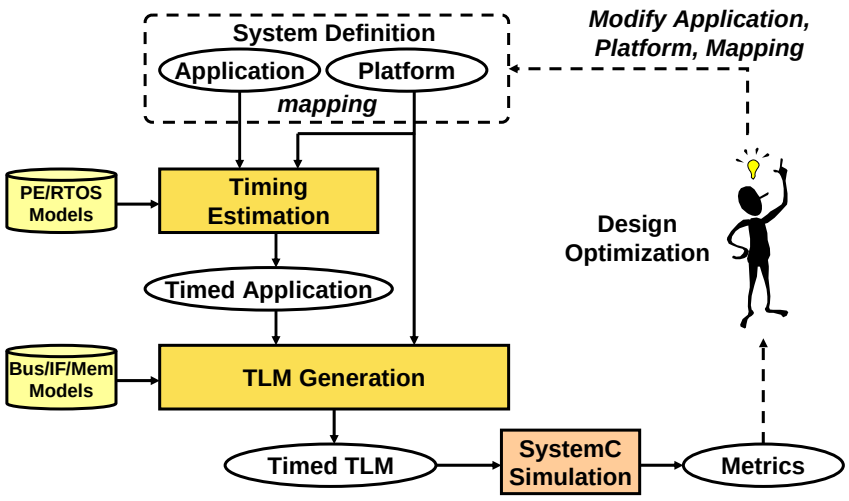
Embedded Design Practice

- ✓ Introduction
- ✓ System Level Design Tools
- ✓ Embedded Software Design Tools
- ✓ Hardware Design Tools
- **Case Study**
  - Embedded System Environment
  - Design Driver: MP3 Decoder
  - Results
- Summary

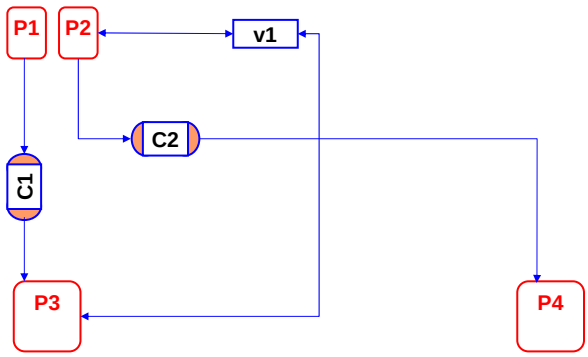
Embedded System Environment (ESE)



ESE Front End Design Flow



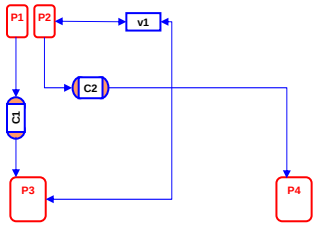
Input: Application Model



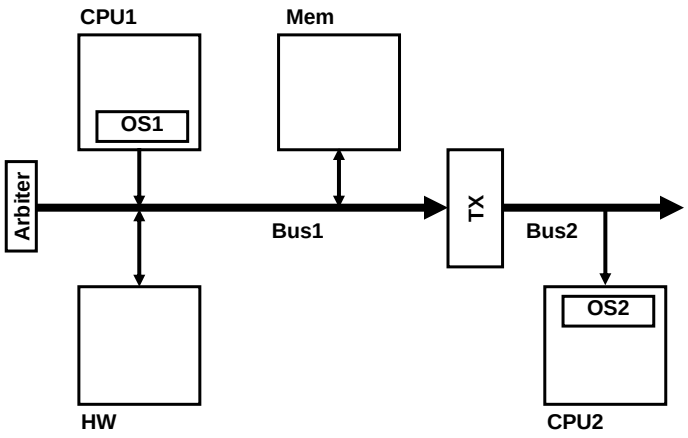
- **Application model consists of**
  - Processes for computation (eg. P1, P2, P3, P4)
  - Channels for communication (eg. C1 between P1 and P3)
  - Variables for storage (eg. v1)

Application Model Objects

- **Processes**
  - Symbolic representation of computation
  - Contain C/C++ code imported from reference
- **Process ports**
  - Symbolic representation of communication services required by processes
  - Provide object orientation by allowing processes to connect to different channels
- **Channels**
  - Symbolic representation of inter-process communication
  - Implement communication services such as blocking, non-blocking, handshake, FIFO etc.
  - Encapsulation for communication functions
- **Variables**
  - Symbolic representation of data storage



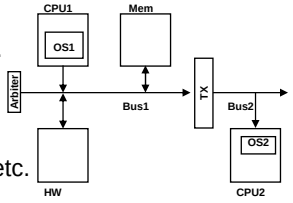
Input: Platform Architecture



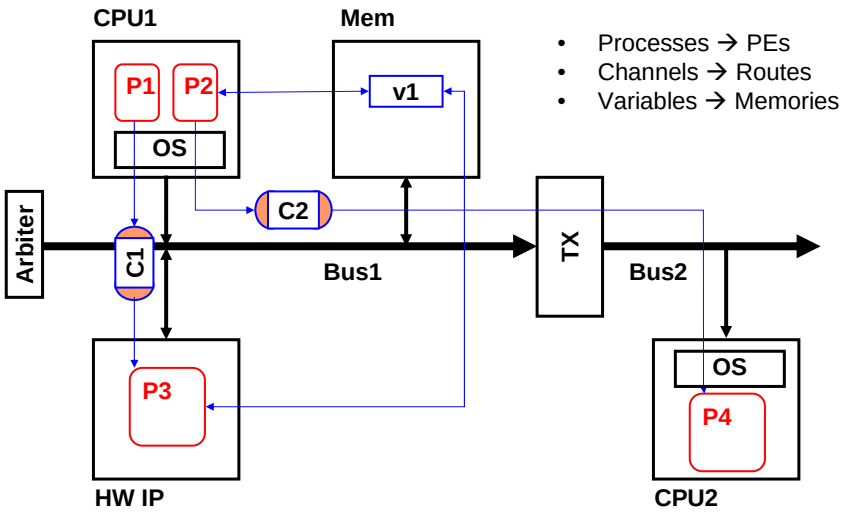
- **Platform consists of**
  - Hardware: PEs (eg. CPU1, HW), Buses (eg. Bus1), Memories (eg. Mem), Interfaces (eg. Transducer)
  - Software: Operating systems (eg. OS1) on SW PEs

Platform Objects

- **Processing element (PE)**
  - Symbolic representation of computation resources
  - Different types such as SW processors, HW IPs etc.
- **Bus**
  - Symbolic representation of communication media
  - Types include shared, point-to-point, link, crossbar etc.
- **Memory**
  - Symbolic representation of physical storage
  - May contain shared variables or SW program/data
- **Transducer**
  - For protocol conversion and store-forward routing
  - Necessary for PEs with different bus protocols
- **Operating system (OS)**
  - Software platform for individual PEs
  - Needed for scheduling multiple processes on a PE



Input: Mapping

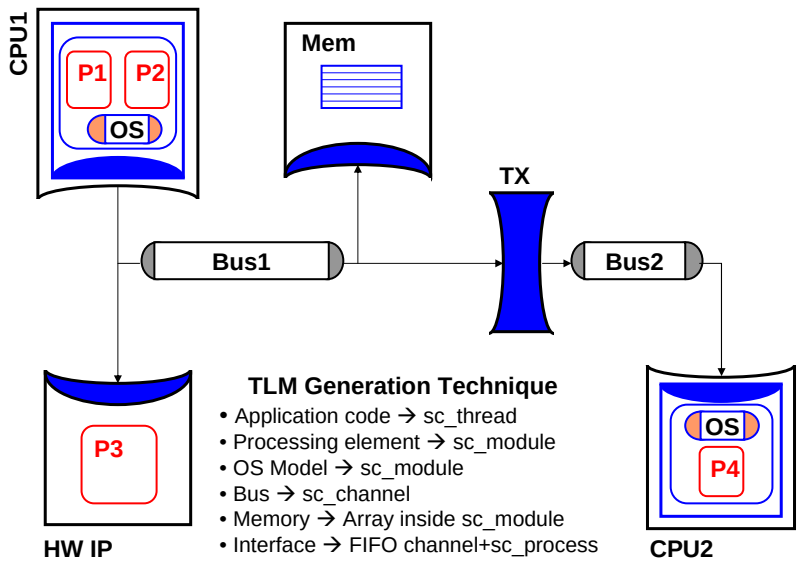


- Processes → PEs
- Channels → Routes
- Variables → Memories

Mapping rules

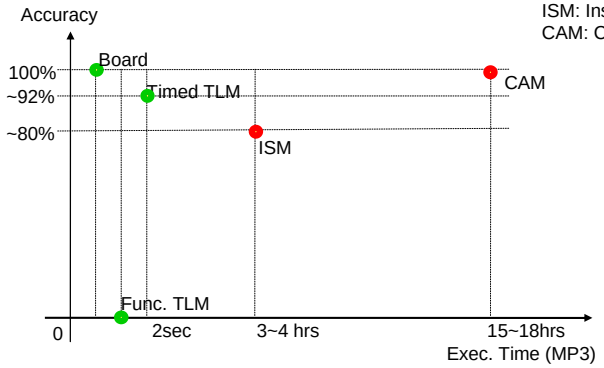
- **Processes to PEs**
  - Each process in the application must be mapped to a PE
  - Multiple processes may be mapped to SW PE with OS support
  - Example: P1, P2 → CPU1
- **Channels to Routes**
  - All channels between processes mapped to different PEs are mapped to routes in the platform
  - Route consists of bus segments and interfaces
  - Channel on each bus segment is assigned a unique address
- **Variables to Memories**
  - Variables accessed by processes mapped to different PEs are mapped to shared memories
  - All variables are assigned an address range depending on size

Output: SystemC TLM



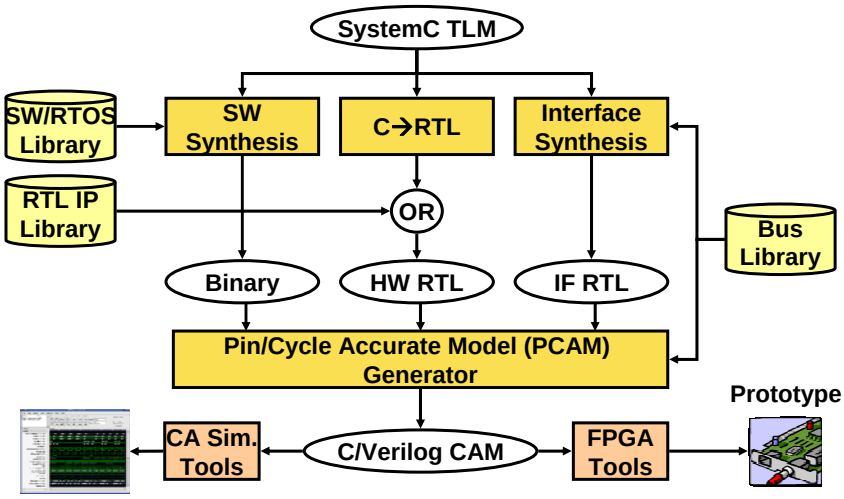
Model Accuracy vs. Execution Time

TLM: Transaction Level Model
ISM: Instruction Set Model
CAM: Cycle-Accurate Model

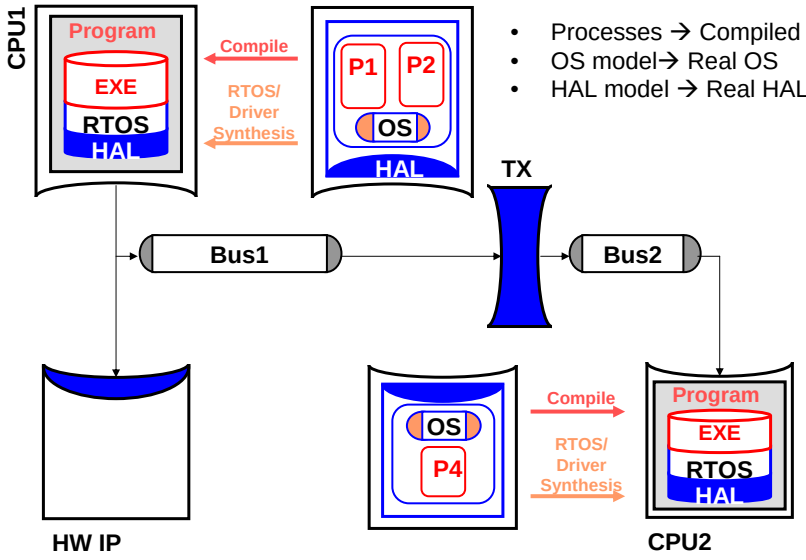


- Board implementation: Reference for model accuracy
- CAM: Accurate but simulates extremely slow
- ISM: Faster than CAM, but inaccurate
- **Functional TLM: No timing, fast simulation (Ideal for SW development)**
- **Timed TLM: Very fast and accurate (Ideal for early estimation)**

ESE Back End Prototyping Flow



Cycle-Accurate Software Synthesis

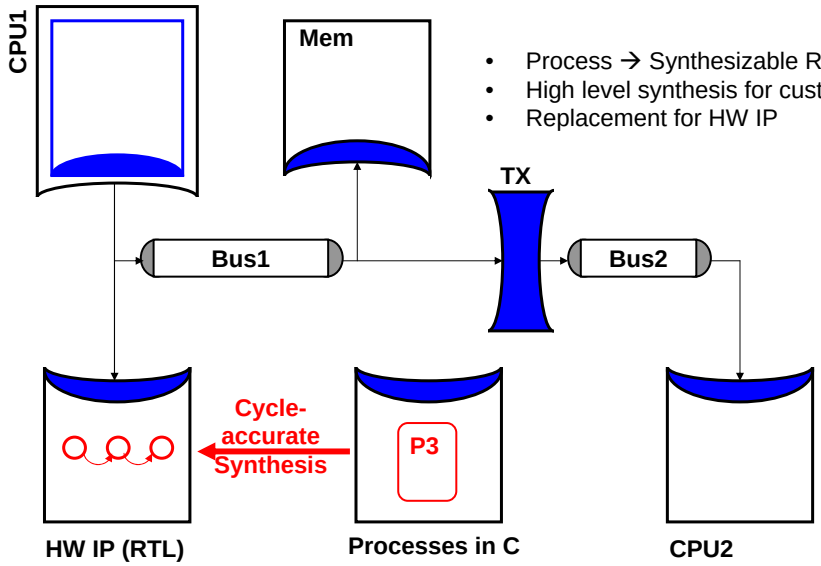


- Processes → Compiled App.
- OS model→ Real OS
- HAL model → Real HAL

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
- **OS selection and targeting**
  - Designer selects an OS for the processor
  - OS model is replaced by real RTOS and SW is re-targeted
  - C code for drivers is generated from Hardware Abstraction Layer (HAL) model
- **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

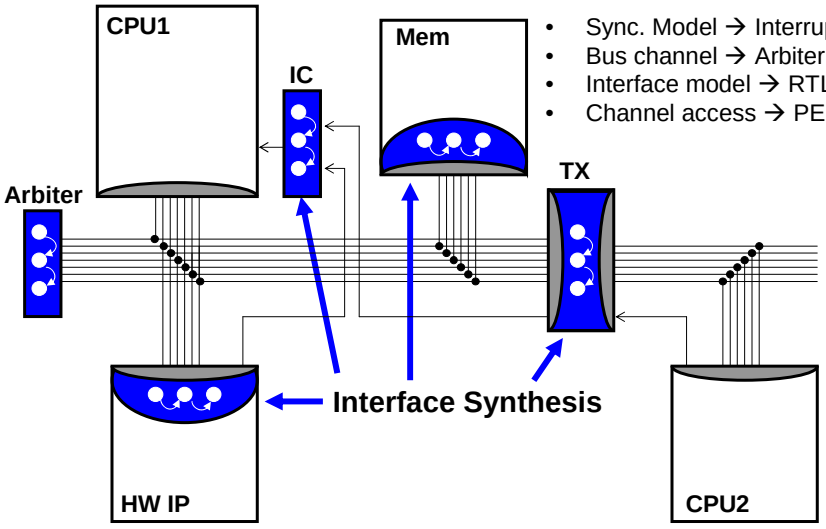


- Process → Synthesizable RTL
- High level synthesis for custom
- Replacement for HW IP

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 single cycle operations, to be later expanded during interface synthesis
- **HDL generation**
  - RTL synthesis result in cycle accurate synthesizable Verilog code

Cycle-Accurate Interface Synthesis

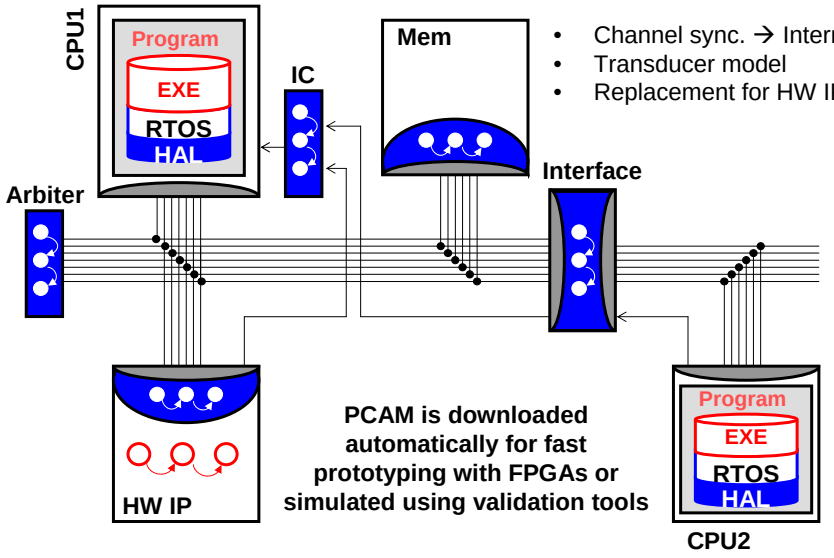


- Sync. Model → Interrupts
- Bus channel → Arbiter + Signals
- Interface model → RTL
- Channel access → PE interface

Interface Synthesis Issues

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

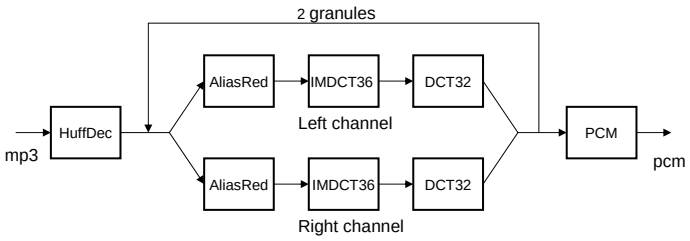
Cycle-Accurate Model



- Channel sync. → Interrupts
- Transducer model
- Replacement for HW IP

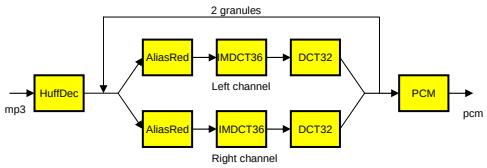
Example: MP3 Decoder Application

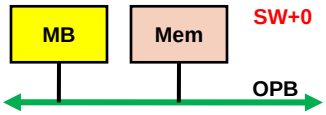
- **Functional block diagram (major blocks only)**



- **Characteristics**
  - Over 12K lines of C code in Spec
  - IMDCT36 and DCT32 compute intensive functions
  - Constraint: Frame processing delay < 26.12ms
- **Design objective**
  - Select platform and mapping to meet constraint

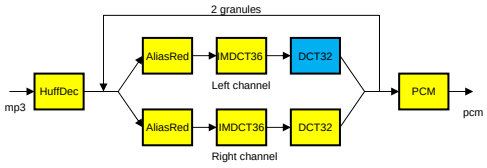
MP3 Platform SW+0

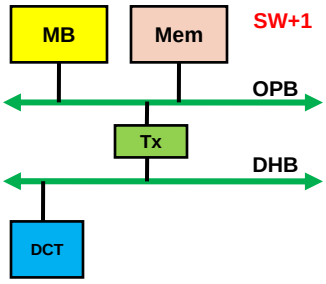




- **MP3 mapped to Microblaze on Xilinx board**
  - Pure software solution
  - Easy to implement, debug and upgrade
  - Frame decoding delay estimated by TLM at 35.66 ms
  - Does not meet the frame delay constraint of 26.12 ms

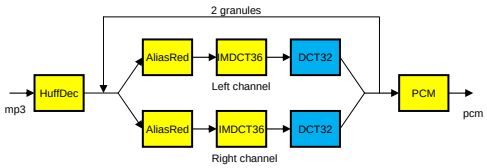
MP3 Platform SW+1

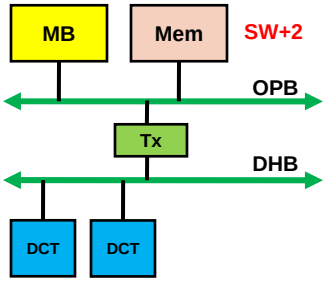




- **MP3 mapped to Microblaze (MB) SW and 1 HW component**
  - DCT32 from left channel moved to custom HW for acceleration
  - Everything else in SW on Microblaze
  - Transducer (Tx) added to connect HW module's DHB interface to OPB
  - Frame decoding delay estimated by TLM at 32.89 ms
  - Faster than SW+0 but does not meet frame delay constraint of 26.12 ms

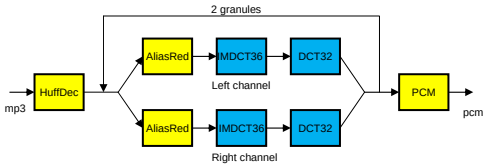
MP3 Platform SW+2

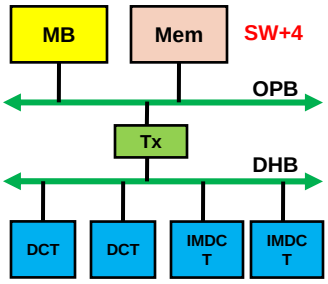




- **MP3 Decoder mapped to Microblaze SW and 2 HW components**
  - DCT32 from both left and right channels moved to HW
  - DCT32 functions for the two channels execute concurrently in HW
  - Bridge added to connect HW module's DHB interface to OPB
  - Frame decoding delay estimated by TLM at 29.99 ms
  - Faster than SW+1 but does not meet frame delay constraint of 26.12 ms

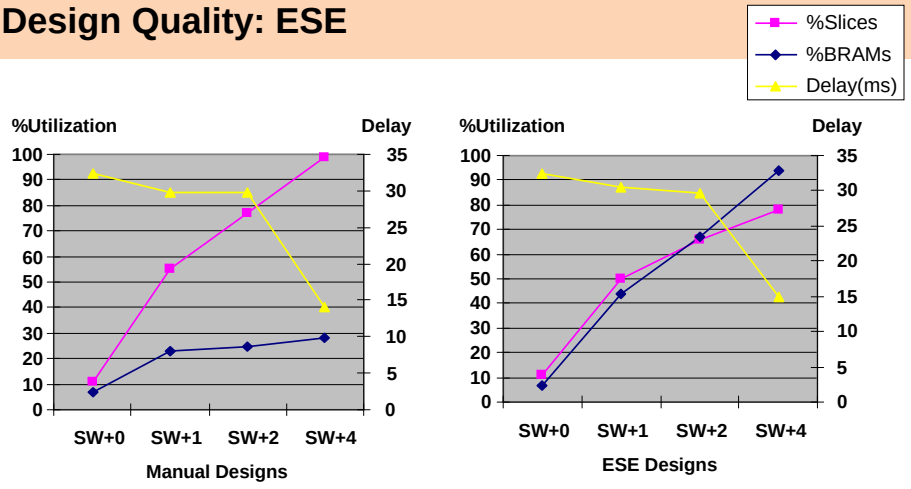
MP3 Platform SW+4





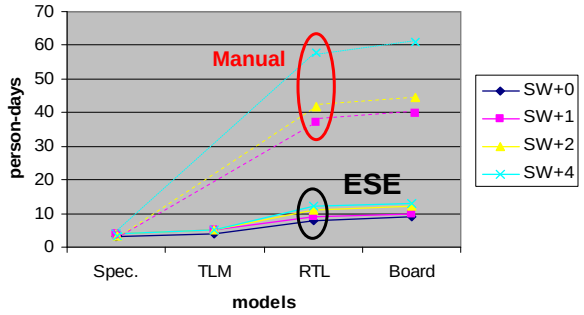
- **MP3 Decoder mapped to Microblaze SW and 4 HW components**
  - Both DCT32 and IMDCT36 from both channels moved to HW
  - Everything else in SW on Microblaze
  - Bridge added to connect HW module's DHB interface to OPB
  - Frame decoding delay estimated by TLM at 15.96 ms
  - Significantly faster than SW+2 and **meets frame delay constraint!**

Design Quality: ESE



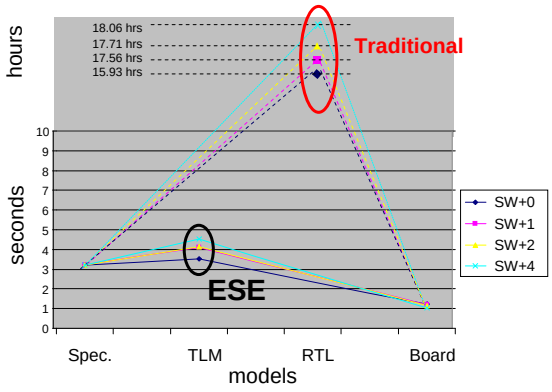
- **Area**
  - ESE designs use fewer FPGA slices and more BRAMs than manual HW: Controller implemented with memory vs. gates
- **Performance**
  - ESE designs execute at similar speed as manual designs

Development Time: ESE vs. Manual



- **ESE drastically cuts RTL and Board development time**
  - Manual development includes months of RTL coding
  - Models can be developed at Spec level with ESE
  - TLM, RTL and Board models are generated automatically by ESE

Validation Time: ESE vs. Traditional



- **ESE cuts validation time from hours to seconds**
  - No need to verify RTL models for every design change
  - Designers can perform high speed validation with TLM and board

ESE Technology Summary

- **C based application input**
  - Supports model based design and legacy reuse
- **Automatic functional and timed TLM generation**
  - Enables early design validation and reliable estimation
- **Automatic SW synthesis**
  - Provides modular, verifiable, platform specific SW code
- **Automatic interface synthesis**
  - Allows rapid implementation of heterogeneous networks
- **FPGA and C/HDL export**
  - Generates standard input for commercial prototyping and CA validation tools