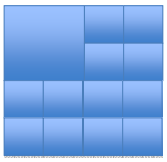
Software Synthesis for Embedded Multicore Systems

Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
http://www.ece.utexas.edu/~gerstl

UT ECE

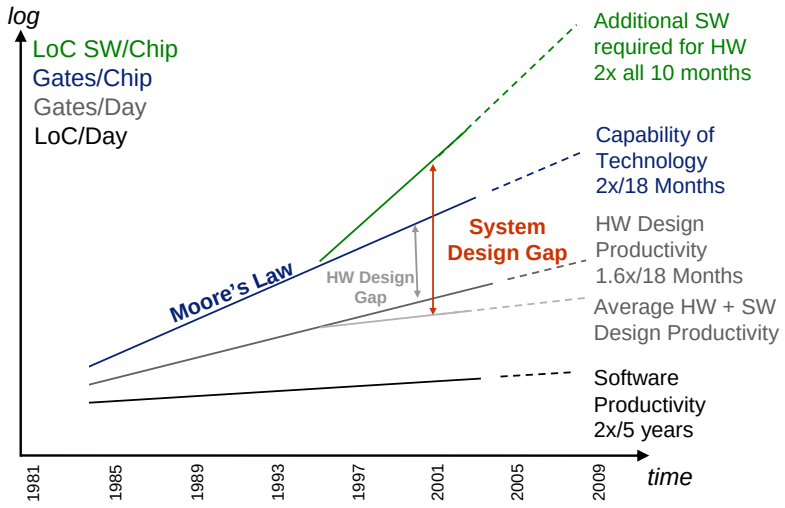
Two Worlds Are Merging

- **Embedded systems**
  - Application-specific, tight constraints
  - ➢ Increasingly networked and programmable
- **General-purpose computing**
  - Physical limits of scaling, power walls and "dark silicon"
  - ➢ Increasingly parallel and heterogeneous



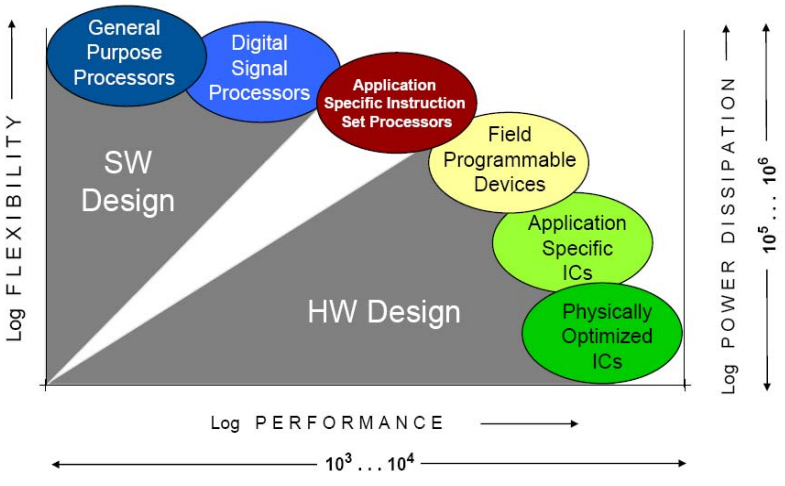
- ➢ **Hardware specialization**
- ➢ **Flexibility through programming**

Complexity Trends



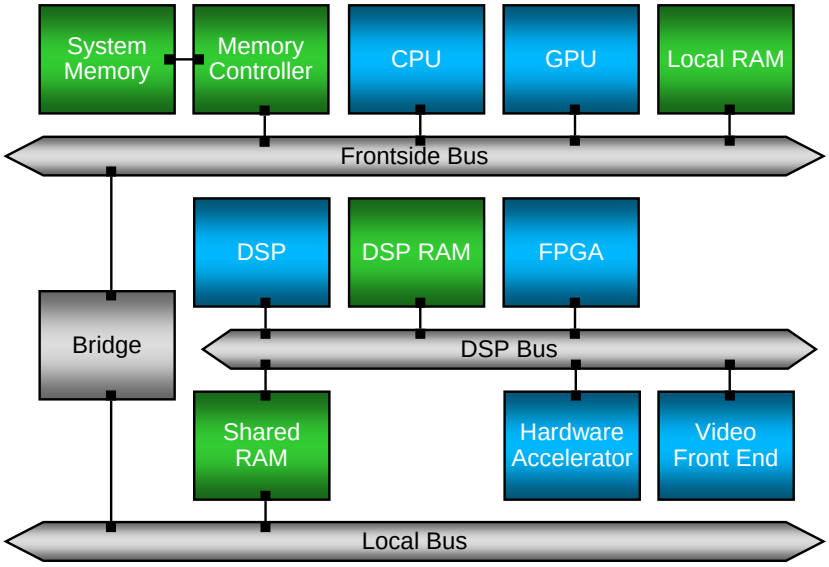
Source: W. Ecker, W. Müller, R. Dömer, *Hardware-dependent Software - Principles and Practice*, Springer 2009.

Heterogeneity Spectrum



Source: T. Noll, RWTH Aachen, via R. Leupers, "From ASIP to MPSoC", Computer Engineering Colloquium, TU Delft, 2006

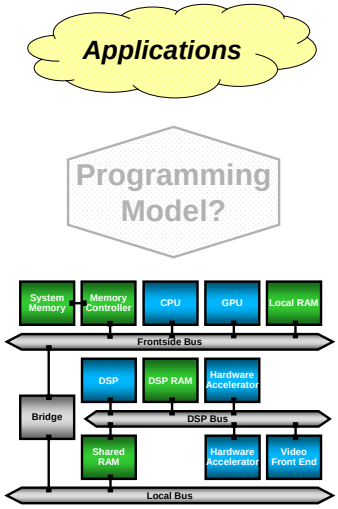
Multi-Processor System-on-Chip (MPSoC)



Challenges

- **Complexity**
  - High degree of parallelism at various levels
  - Real-time, reactive
  - Reliability
- **Heterogeneity**
  - Of architectures
  - Of applications
  - Of tools

➢ **Application mapping**
  - ➢ System “compiler”?



General-Purpose Software Trends

- **Paradigm shift: software synthesis**
  - From a single, very high-level, formal specification
  - To parallel, multi-core target architectures

➢ **Regular applications: linear algebra [FLAME]**
  - Programming model: Matrix & vector math
  - Compiler encodes algorithm knowledge
  - Output: linear algebra library optimized for target arch.
    - Hardware heterogeneity: linear algebra processor (LAP) [ASAP11]
      10x-100x better performance & GFLOPS/W as GPUs and CPUs

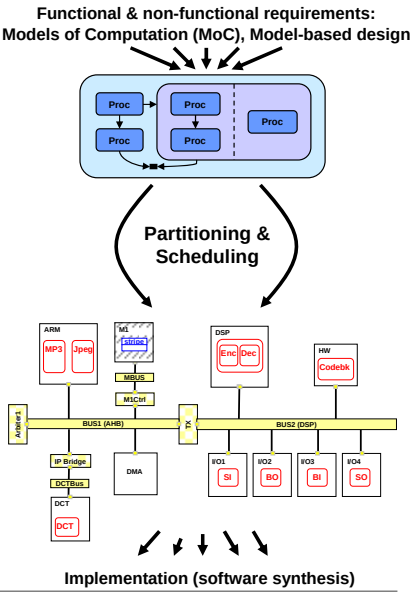
➢ **Irregular applications: graph traversals [Galois]**
  - Layered programming model: Graph data structures
  - Library to extract amorphous data parallelism
  - Output: runtime system to dynamically schedule execution

Source: Keshav Pingali, CS, UT Austin

An (Embedded) System Compiler

- **From high-level specification**
  - Application functionality
    - Programming model
- **To implementation**
  - Heterogeneous MPCSoC
    - Across hardware & software

➢ **Automation**
  - Synthesis
    - Task mapping
    - Software synthesis
  - Modeling
    - Programming models
    - Simulation models

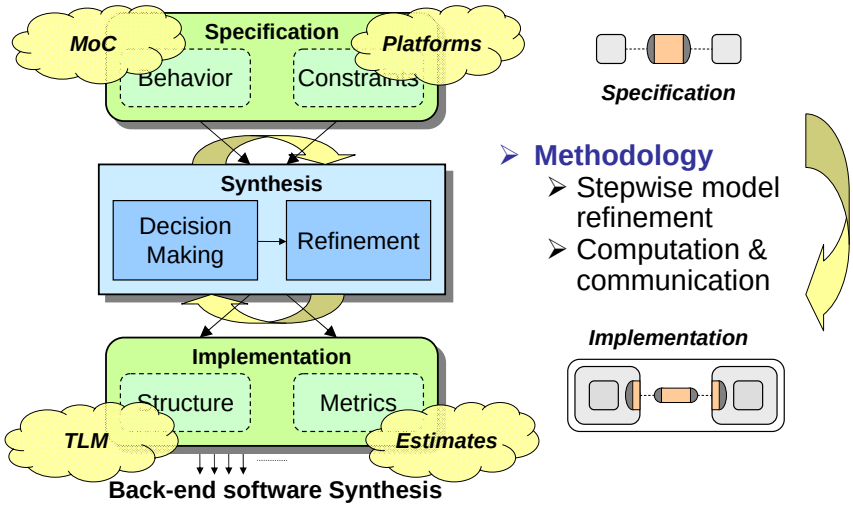


The Key Ingredients…

- **Formal methods and models**
  1. Specification = programming model
     - Models of Computation (MoCs)
  2. Software synthesis = compilation
     - Mapping: partitioning and scheduling
     - Design space exploration: decision making
     - Refinement: target code & binary generation
  3. Evaluation = simulation
     - Transaction-level modeling (TLM)
     - Virtual platform prototyping
     - Estimation

Synthesis

- **X-Chart**



- **Methodology**
  - Stepwise model refinement
  - Computation & communication

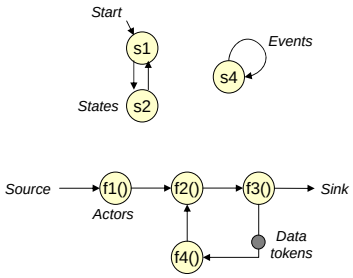
Source: A. Gerstlauer, C. Haubelt, A. Pimentel, et al., "Electronic System-Level Synthesis Methodologies," TCAD, 2009.

Programming Models

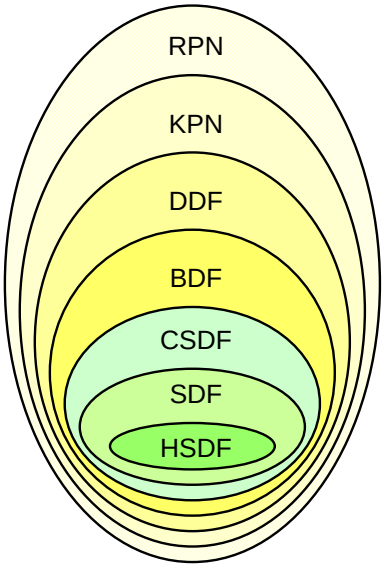
- **Models of Computation (MoCs)**
  - Rigorous, formal modeling as a basis for automation
    - Concurrency and communication
    - Analysis and optimization (implementation decisions)
  - Domain-specific
    - Inherent tradeoffs between expressiveness vs. analyzability

➢ **State-based models**
  - Control-dominated, reactivity
    - Automotive, aerospace
      - ➢ FSMs, StateCharts [Statemate]

➢ **Process-based models**
  - Data-dominated, streaming
    - Communications, signal processing
      - ➢ KPNs, dataflow [LabView]



Dataflow Models



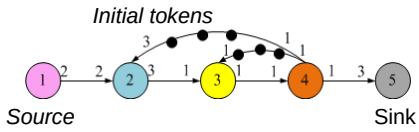
Yellow: *Turing complete*
Green: *Statically analyzable*

RPN	Reactive Process Network
KPN	Kahn Process Network
DDF	Dynamic Dataflow
BDF	Boolean Dataflow
CSDF	Cyclo-Static Dataflow
SDF	Synchronous Dataflow
HSDF	Homogeneous SDF

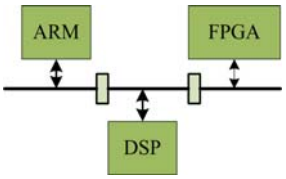
Source: T. Basten, MoCC 2008.

Dataflow Synthesis (1)

- **Synchronous data flow (SDF) model**

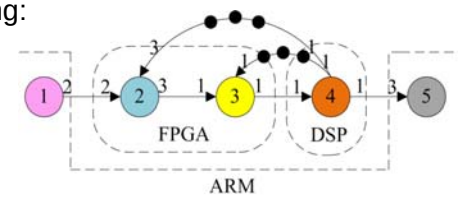


  - Well-know methods for single-processor software synthesis
    - E.g. repetitive static schedule: 1-2-3-4-3-4-3-4-5
    - Fixed throughput, minimize buffer usage & code size
  - Multi-processor synthesis more challenging
    - MPSoC mapping of dataflow graphs
    - Traditionally solely throughput-oriented / targeting homogeneous multi-core architectures
    - Resource usage (processors/hardware, buffers/memories)?
    - Latency?

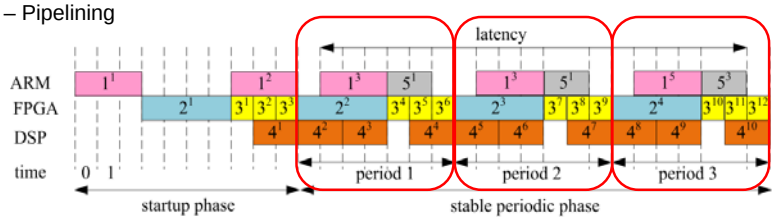


Dataflow Synthesis (2)

- **Mapping SDF models to MPSoCs**
  - Allocation and partitioning:
    - Resource sharing



  - Scheduling:
    - Pipelining

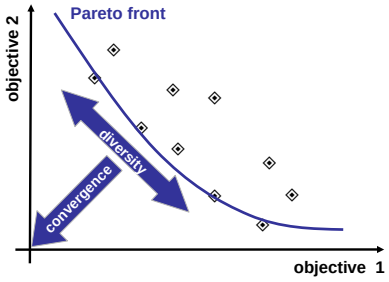


Period = 1 / Throughput

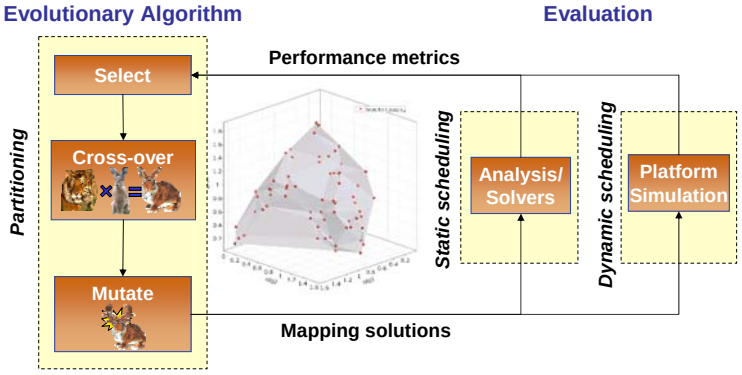
Latency = (End of the *n*-th exec. of Sink) – (Start of the *n*-th exec. of Source)

Dataflow Synthesis (3)

- **Mapping of general SDF graphs onto heterogeneous multi-processor architectures**
  - Multi-objective optimization
    - Throughput
    - Latency
    - Cost (area/resource usage)
    - Power
    - Thermal
    - …
  - Design space exploration
    - Find Pareto-optimal solution set



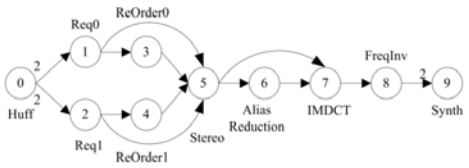
Design Space Exploration



- **Automatic decision making and design space exploration**
  - Near-optimal, hierarchical exploration heuristics
    - Combinations of evolutionary algorithms & integer linear programming
  - Pareto-optimal multi-objective solution space
    - Performance, energy, reliability, power, thermal (PERPT)

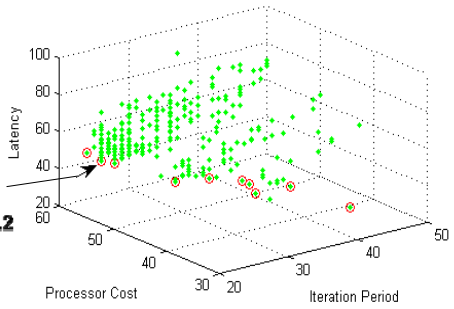
SDF Synthesis Results

- **Design space exploration for an MP3 decoder**



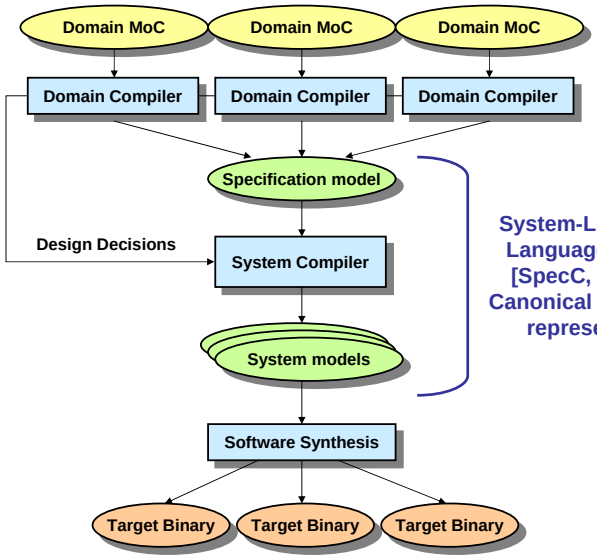
- **Convergence to Pareto front**
  - Within 10^{-6} of optimum
  - 12x better runtime
    - Polynomial in size of SDF graph

Solution of global ILP with $\lambda_1 = 0.8$ and $\lambda_2 = 0.2$



J. Lin, A. Srivasta, A. Gerstlauer, B. Evans, "Heterogeneous Multiprocessor Mapping for Real-time Streaming Systems," ICASSP'11

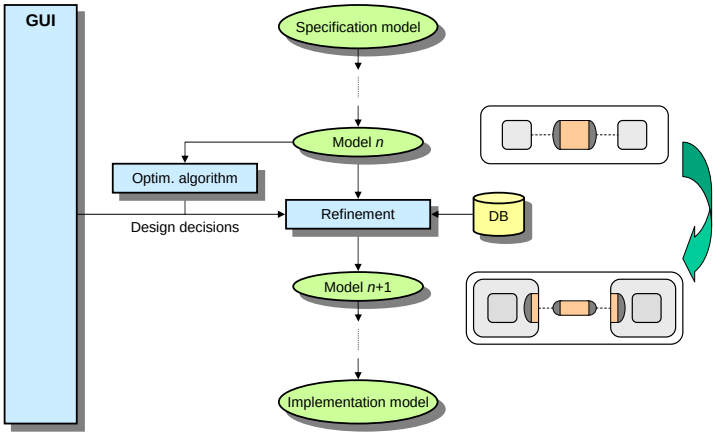
Synthesis Flow



System-Level Design Languages (SLDLs) [SpecC, SystemC]: Canonical intermediate representations

Compilation Process

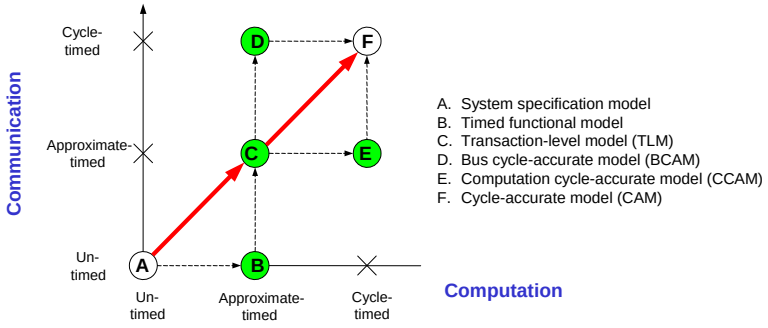
- **Synthesis = Decision making + model refinement**



- **Automatic, successive & stepwise model refinement**
- **Layers of implementation detail, code generation**

System Compilation Flow

- **Fundamental separation of orthogonal concerns**
  - Computation and communication
- **System refinement flow**
  - Path from model A to model F

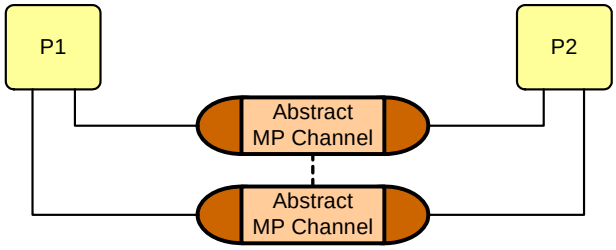


Source: L. Cai, D. Gajski. "Transaction level modeling: An overview", ISSS 2003

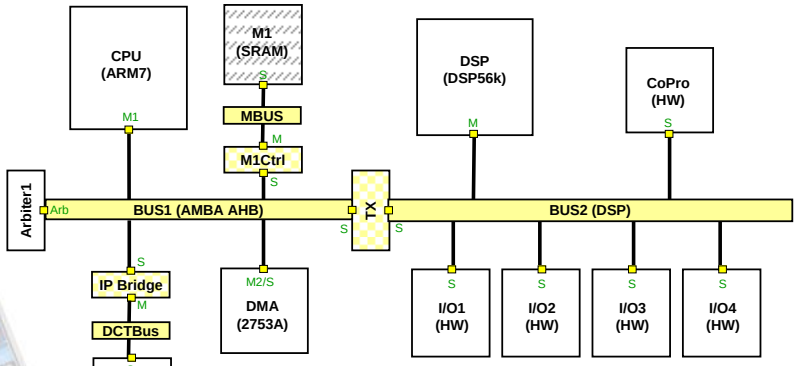
- **Design methodology and compilation flow**
  - Set of models and transformations between models

Specification

- **Abstract, high-level system functionality**
  - Computation
    - Processes
    - Variables
  - Communication
    - Sync./async. message-passing
    - Variables
    - Events

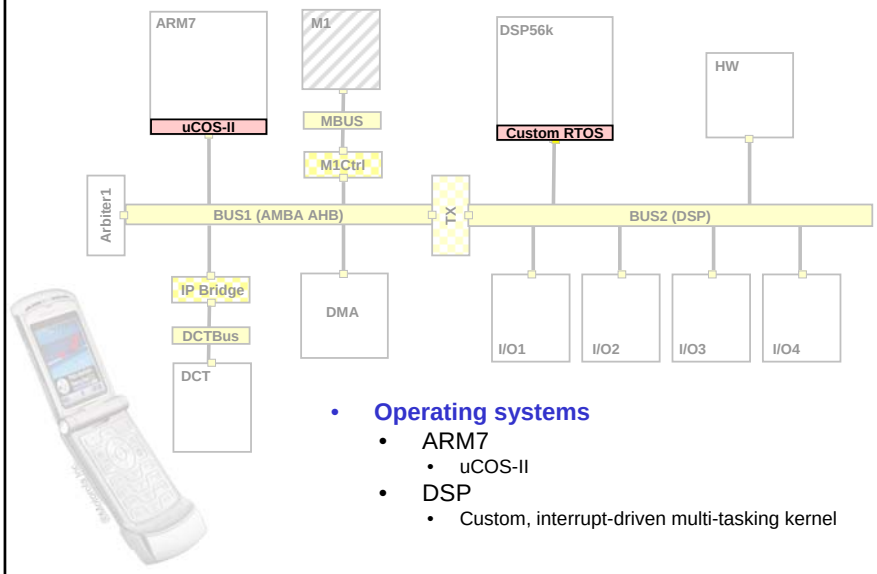


Cellphone Example: Hardware Platform



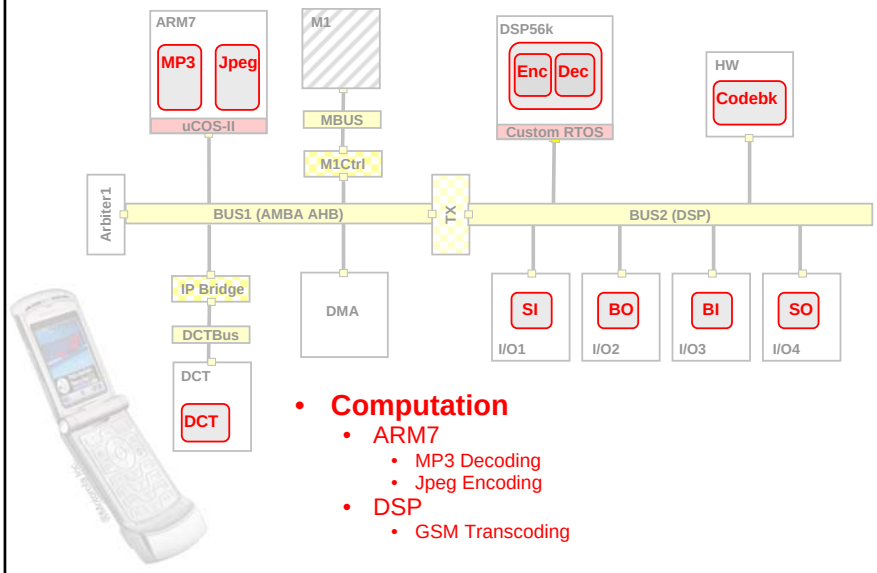
- **2 Subsystems**
  - ARM7TDMI
  - Motorola DSP 56600k
- **4 Busses**
  - AMBA AHB
  - DSP bus
  - IP & memory busses
- **2 Accelerator HW/IP blocks**
  - DCT IP
  - Custom codebook HW
- **10 I/O HW blocks**

Cellphone Example: Software Platform



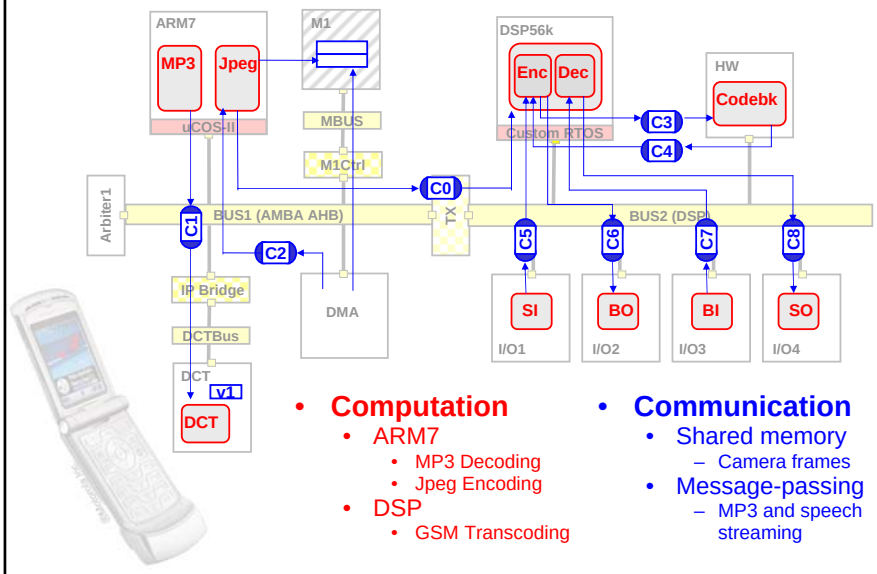
- **Operating systems**
  - ARM7
    - uCOS-II
  - DSP
    - Custom, interrupt-driven multi-tasking kernel

Cellphone Example: Application



- **Computation**
  - ARM7
    - MP3 Decoding
    - Jpeg Encoding
  - DSP
    - GSM Transcoding

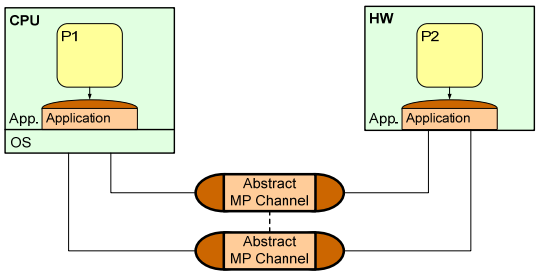
Cellphone Example: Specification



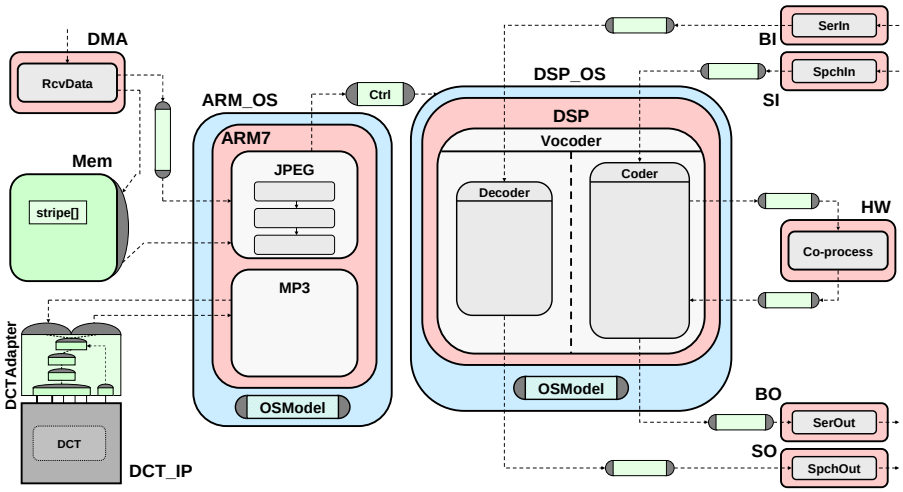
- **Computation**
  - ARM7
    - MP3 Decoding
    - Jpeg Encoding
  - DSP
    - GSM Transcoding
- **Communication**
  - Shared memory
    - Camera frames
  - Message-passing
    - MP3 and speech streaming

Architecture Synthesis

- **Computation partitioning & scheduling**
  - PEs + Memories
  - Static and dynamic scheduling (OS)
  - Abstract communication via channels
    - Synchronous packet transfers (data transfers)
    - Memory accesses (shared memory, memory-mapped I/O)
    - Events (control flow)



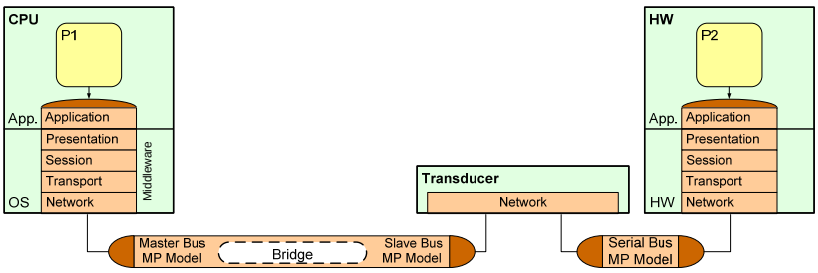
Cellphone Example: Architecture Model



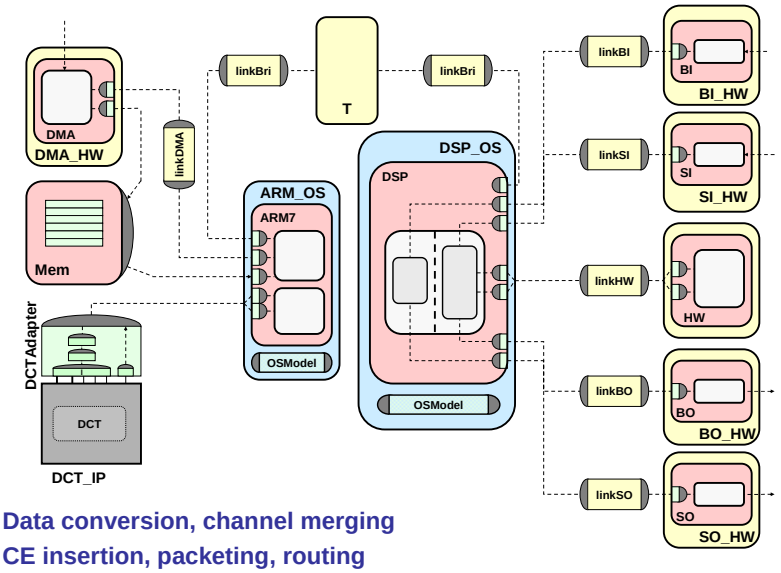
- **Scheduling, task refinement, OS model insertion**

Network Synthesis

- **Topology of communication architecture.**
  - PEs + Memories + CEs
  - Upper protocol layers inserted into PEs/CEs
    - Custom, lightweight middleware
  - Communication via point-to-point links
    - Synchronous packet transfers (data transfers)
    - Memory accesses (shared memory, memory-mapped I/O)
    - Events (control flow)



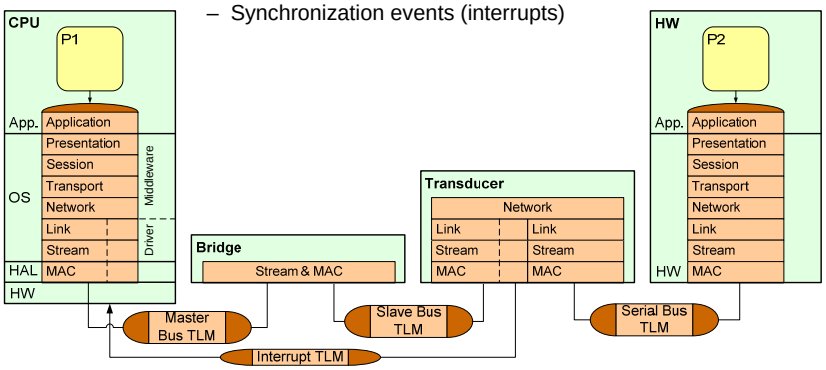
Cellphone Example: Network TLM



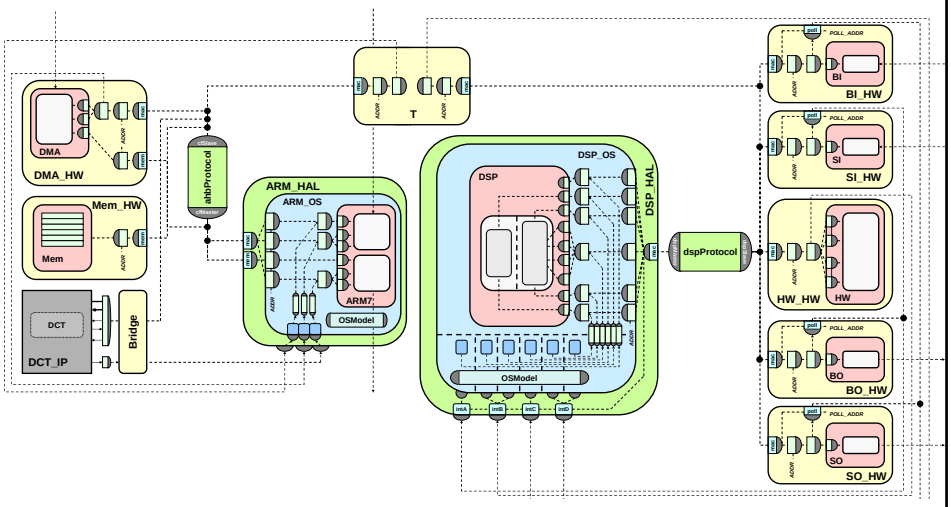
- Data conversion, channel merging
- CE insertion, packeting, routing

Protocol Synthesis

- **Abstract bus structure/architecture**
  - PEs + Memories + CEs + Busses
  - Communication layers down to protocol transactions
    - Hardware-dependent software (HdS)
  - Communication via transaction-level channels
    - Bus protocol transactions (data transfers)
    - Synchronization events (interrupts)



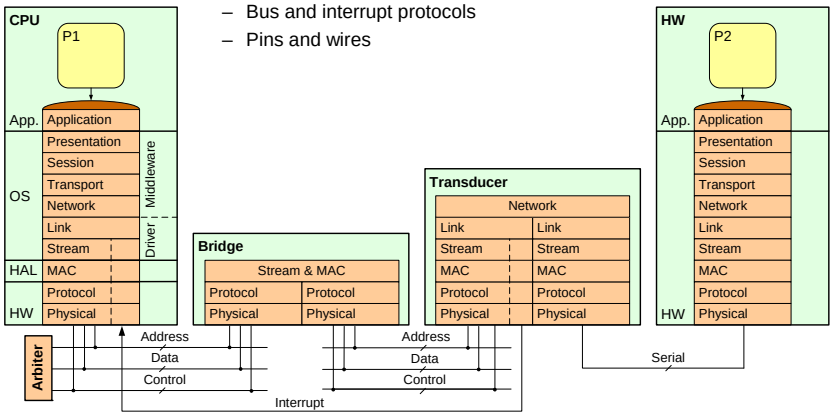
Cellphone Example: Protocol TLM



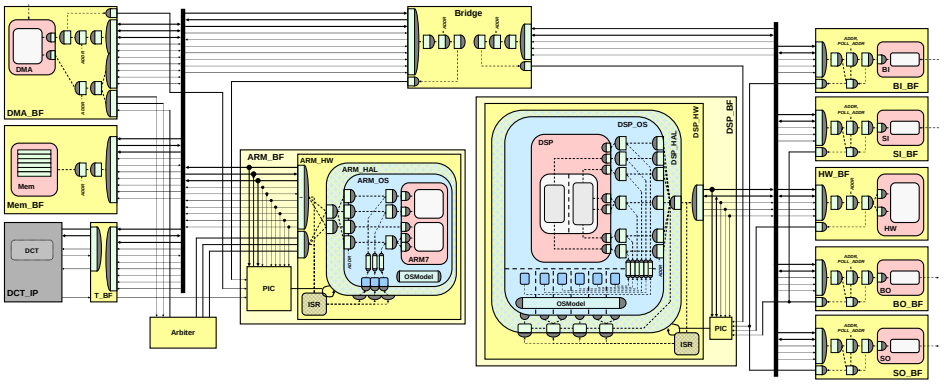
- Synchronization, addressing, media acces
- Arbitration, data slicing, interrupt handling

Bus Cycle-Accurate Model (BCAM)

- Component & bus structure/architecture
  - PEs + Memories + CEs + Busses
  - Pin-accurate bus-functional components
  - Pin- and cycle-accurate communication
    - Bus and interrupt protocols
    - Pins and wires

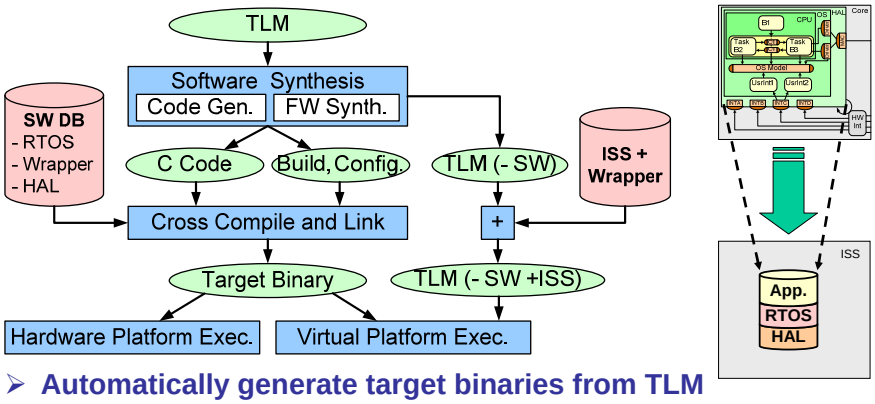


Cellphone Example: BCAM



- **Implementation synthesis in backend tools**
  - Interface and high-level synthesis on hardware side
  - Firmware, RTOS and C synthesis on the software side

Backend Software Synthesis

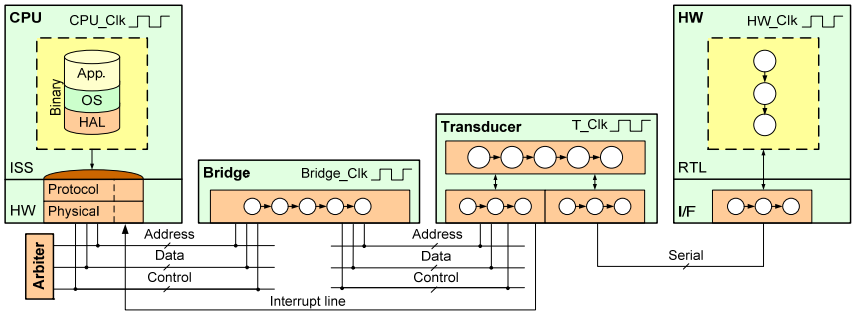


- **Automatically generate target binaries from TLM**
  - Generate code for application (tasks and IPC)
  - Synthesize firmware (drivers, interrupt handlers)
  - OS wrappers and HAL implementations from DB
  - Compile and link against target RTOS and libraries

Source: G. Schirner, A. Gerstlauer, R. Doemer. "Automatic Generation of Hardware dependent Software for MPSoCs from Abstract System Specifications," ASPDAC08

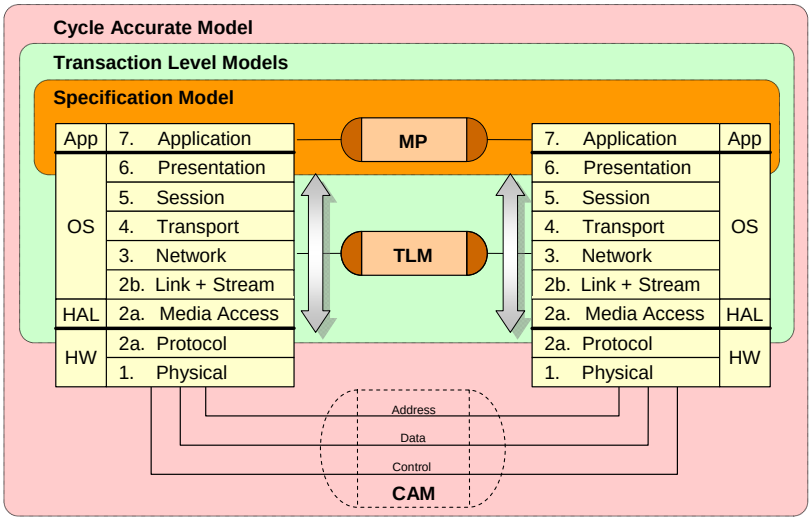
Cycle-Accurate Model (CAM)

- **Component & bus implemenation**
  - PEs + Memories + CEs + Busses
  - Cycle-accurate components
    - Instruction-set simulators (ISS) runing final target binaries
    - RTL hardware models
    - Bus protocol state machines

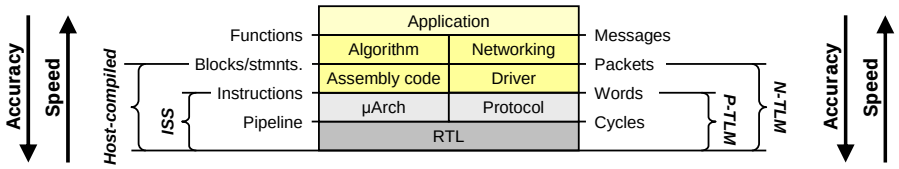


System Models

- **From refinement layers to system models…**



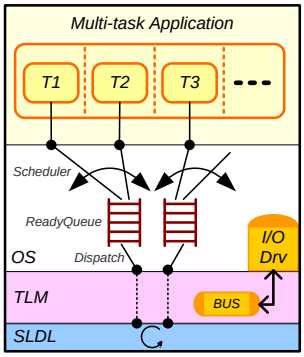
Modeling Layers



- **Computation**
  - Host-compiled modeling
    - Abstract execution (at source level) above instructions
  - ➢ Functionality and timing
    - ➢ Native execution of functionality
    - ➢ Back-annotation of timing & power estimates
    - ➢ Models of execution environment (OS & HW)
- **Communication**
  - Transaction-level modeling (TLM)
    - Abstract transactions (function calls) above pins and wires
    - Below complete messages
  - ➢ Functionality and timing
    - ➢ Varying levels of granularity
    - ➢ Speed vs. accuracy

Host-Compiled Modeling

- **Software simulation**
  - Compile and execute application natively
    - Fast functional simulation
  - Annotate application with target timing and power
    - Accurate feedback
  - Wrap with SLDL code for platform integration
    - Add OS and processor models for integration into TLM backplanes

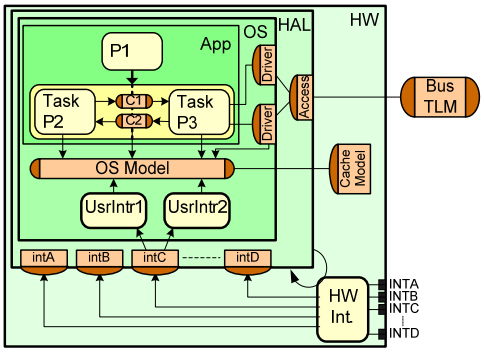


- ➢ **Fast and accurate evaluation of real-time behavior**
  - ➢ 300-1500 MIPS @ 97% accuracy

Source: A. Gerstlauer. "Host-Compiled Simulation of Multi-Core Platforms," RSP10.

Processor Modeling

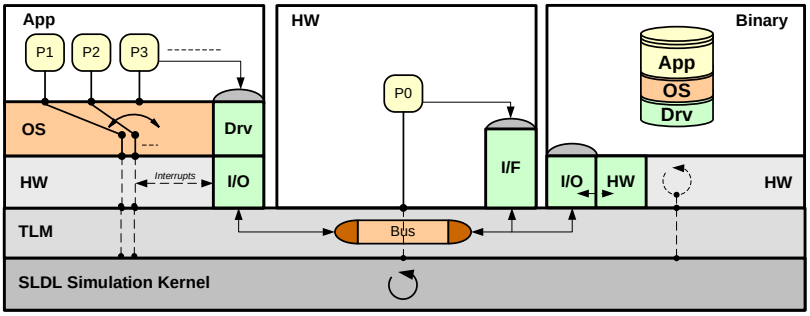
- **Processor layers [TODAES10]**
  - Application
    - Native, host-compiled C
    - Profiling & back-annotation [DAC04]
  - OS
    - OS model [DATE03, DATE11, ASPDAC12]
    - Middleware, drivers
  - HAL
    - Firmware
  - Processor hardware
    - Bus interfaces
    - Interrupts
    - Cache [IESS09]



Features	
Target approx. computation timing	Appl.
Task mapping, dynamic scheduling	OS
Task communication, synchronization	HAL
Interrupt handlers, low level SW drivers	HW-TLM
HW interrupt handling, int. scheduling	HW-BFM
Cycle accurate communication	BFM - ISS
Cycle accurate computation	

Source: G. Schirner, A. Gerstlauer, R. Doemer. "Fast and Accurate Processor Models for Efficient MPSoC Design," TODAES, 2009.

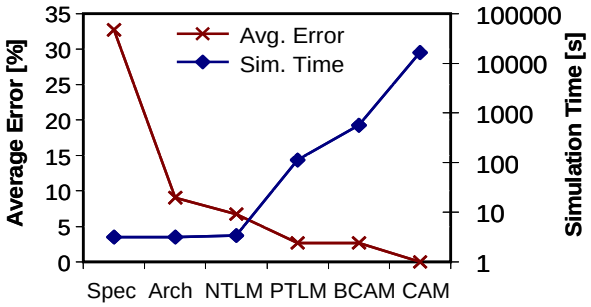
Virtual Platform Prototyping



- **Host-compiled model**
  - Native functionality
  - Timing & power back-annotation
  - RTOS models
  - Processor models
- **SLDL & TLM backplane**
  - Discrete events [SpecC, SystemC]
  - Interconnect
- **Hardware model**
  - Functionality & timing
- **ISS model**
  - Functionality [QEMU, OVP]
  - Cycle-accurate [SimpleScalar]

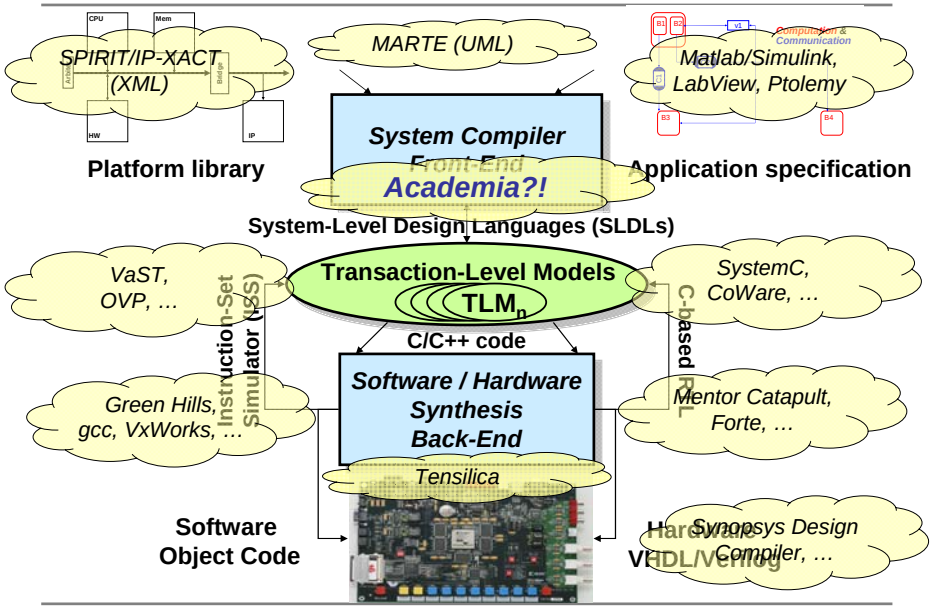
Cellphone Example: Modeling Results

- **MPSoC cellphone platform**
  - ARM + DSP subsystems
  - 4 accelerators, 10 I/O blocks
  - 5 busses
- **Full-system simulation**
  - 300 Mcycles/s (MIPS)
  - <3% timing error
  - ➢ 3s = 300M/180M ARM/DSP cycles in less than 1s of real time



- ➢ **Transaction-level modeling (TLM) of communication**
- ➢ **Host-compiled software, OS and processor modeling**

The System-Level Landscape

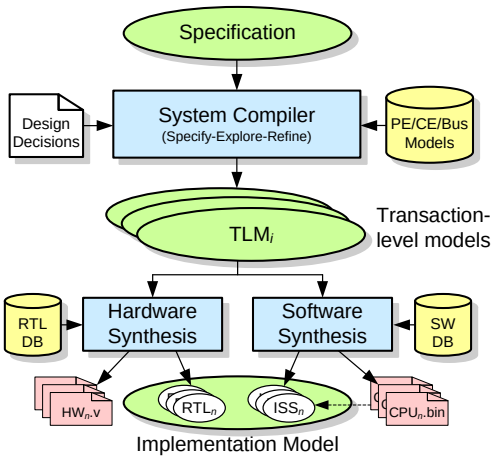


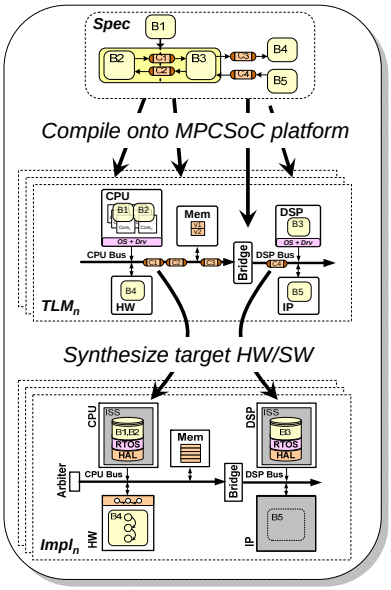
Academic MPSoC Design Tools

Approach	DSE	Comp. decision	Comm. decision	Comp. refine	Comm. refine
Daedalus	●	●	○	●	○
Koski	●	●	○	●	○
Metropolis		○		○	
PeaCE/HoPES	○	○		●	○
SCE				●	●
SystemCoDesigner	●	●	●	●	

Source: A. Gerstlauer, C. Haubelt, A. Pimentel, et al., "Electronic System-Level Synthesis Methodologies," TCAD, 2009.

System-on-Chip Environment (SCE)





➢ **Commercialized w/ Japanese Aerospace Exploration Agency**

SCE Synthesis Results

- **Suite of industrial size examples**

Example		System Architecture (masters → slaves)	Model size (LOC)				Generation time
			Spec	Arch	Net	BCAM	
JPEG	A1	CF→HW	1806	2732	2780	4642	1.01 s
Vocoder	A1	DSP→HW	7385	9594	9775	10679	4.77s
	A2	DSP→HW1,HW2		9632	9913	10989	5.21s
	A3	DSP→HW1,HW2,HW3		9659	9949	11041	5.76s
Mp3float	A1	CF→HW1	6900	28190	28204	29807	5.86s
	A2	CF→HW1,HW2,HW3 HW1↔HW3 HW2↔HW3		28275	28633	31172	6.18s
	A3	CF →HW1, HW2, HW3, HW4 HW1↔HW3↔HW5 HW2↔HW4↔HW5		28736	30202	32795	17.69s
Mp3fix	A1	ARM→2 I/O	13363	17131	17270	21593	3.85s
	A2	ARM→2 I/O, LDCT, RDCT		18300	18564	23228	7.01s
	A3	ARM→2 I/O, LDCT, RDCT LDCT↔I/O RDCT↔I/O		18748	19079	24471	7.40s
Baseband	A1	DSP→HW, 4 I/O, T CF, DMA→Mem, BR, T, DMA BR→DCT_IP	11481	17020	17685	21711	8.99s
Cellphone	A1	ARM→4 I/O, 2 DCT, T LDCT, RDCT→I/O DSP→HW, 4 I/O, T	16441	21936	22570	30072	9.49s

The Holy Grail

- **A multi-processor system compiler**
  1. Programming models
     - Domain-specific models of computation
     - Expressability vs. analyzability
  2. Synthesis and decision making
     - Multi-objective design space exploration
  3. Refinement and code generation
     - Custom middleware and hardware-dependent software generation
     - High-level hardware/FPGA synthesis

➢ **Seamless compilation of parallel applications onto heterogeneous target platforms**