ECE382N.23: Embedded System Design and Modeling

Lecture 6 – Host-Compiled & Transaction-Level Modeling

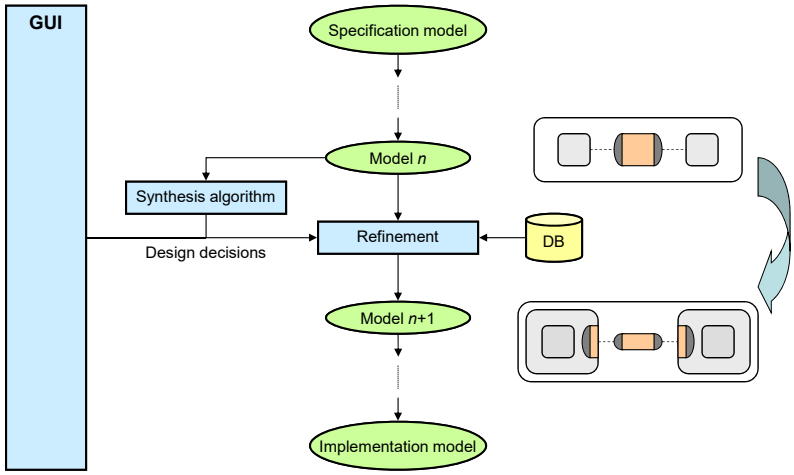
Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

Lecture 6: Outline

- **System-level refinement**
  - Modeling & refinement flow
- **Host-compiled modeling of computation**
  - Source-level application models
  - Operating system and processor models
  - Hardware models
- **Transaction-level modeling of communication**
  - Communication layers and protocol stacks
  - Transaction-level modeling (TLM) of busses
  - Physical models

Refinement Flow

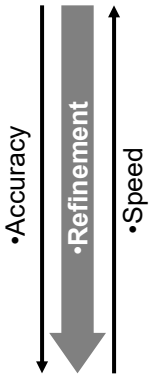
- **Synthesis = Decision making + model refinement**



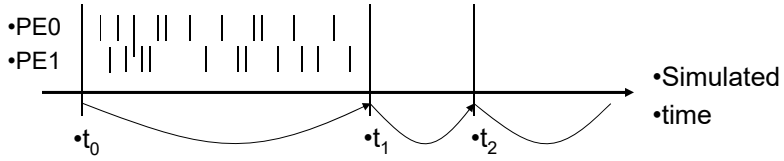
- ➢ **Successive, stepwise model refinement**
- ➢ **Layers of implementation detail**

Model Speed vs. Accuracy

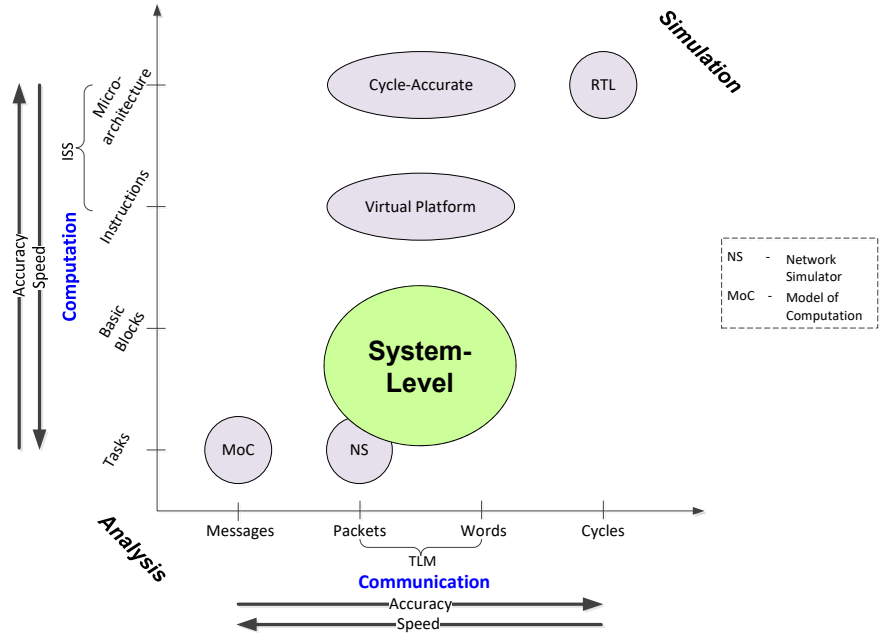
- **Discrete-event simulation speed**
  - Proportional to number of simulated events
  - Proportional to granularity of simulated time/detail
    - "Real-time": simulated vs. simulation time > 1
- **Discrete-event simulation accuracy**
  - Proportional to simulated implementation order
  - Inversely proportional to simulated granularity
    - Where order matters (structural concurrency)



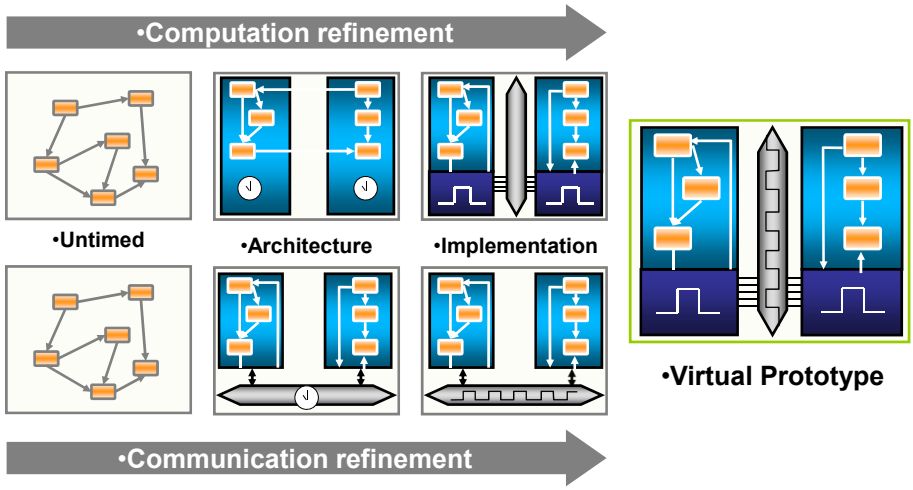
- ➢ **Fundamental modeling tradeoff**



System-Level Modeling Space



System-Level Modeling & Refinement



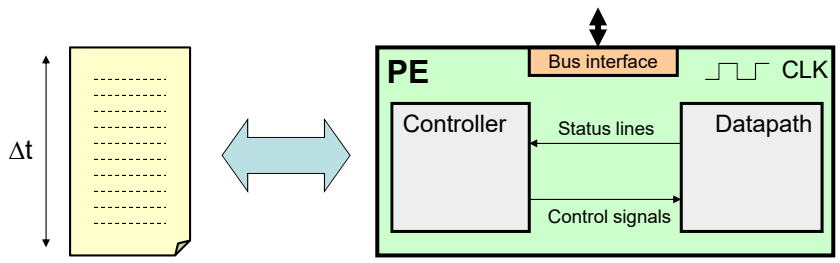
Source: C. Haubelt, Univ. of Rostock

Lecture 6: Outline

- ✓ System-level refinement
  - ✓ Modeling & refinement flow
- **Host-compiled modeling of computation**
  - Source-level application models
  - Operating system and processor models
  - Hardware models
- Transaction-level modeling of communication
  - Communication layers and protocol stacks
  - Transaction-level modeling (TLM) of busses
  - Physical models

Computation Models

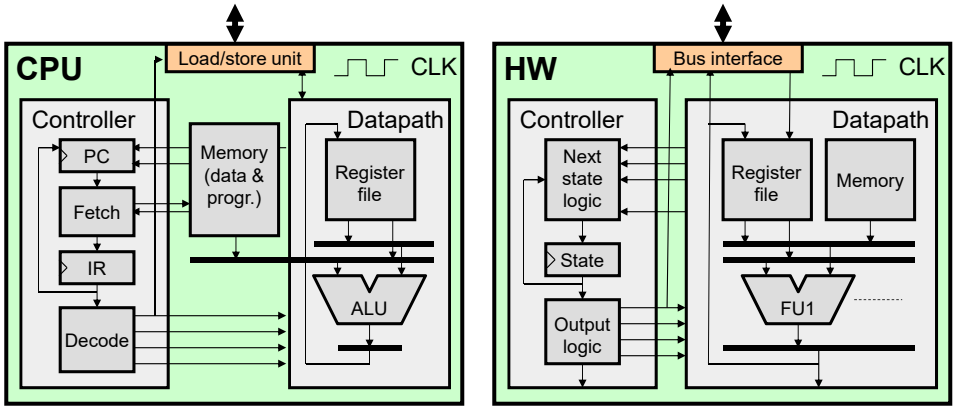
- **Basic component is a *processing element (PE)***
  - Programmable, general-purpose software processor (CPU)
  - Programmable special-purpose processor (e.g. DSPs)
  - Application-specific instruction set processor (ASIP)
  - Custom hardware processor



➢ **Functionality *and* timing (and power and …)**

Computation Modeling (1)

- **Structural RTL models**

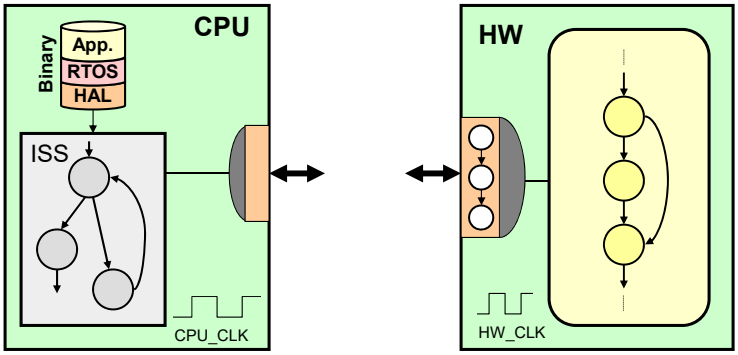


Software processor Hardware processor

➢ **Sub-cycle accurate**

Computation Modeling (2)

- **Behavioral RTL models (FSMD)**
- **Instruction-set simulation (ISS) models**
  - Purely functional (binary translation) [QEMU,…]
  - Micro-architectural (RTL in C) [GEM5,…]

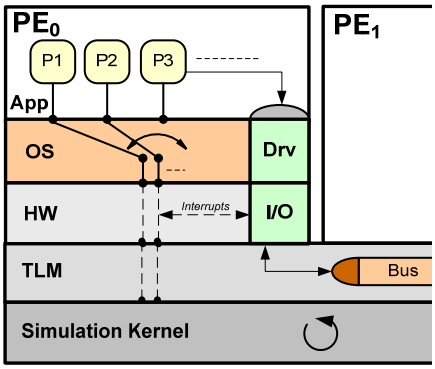


Instruction set simulation (ISS) FSMD

➢ **Cycle or timing accurate**

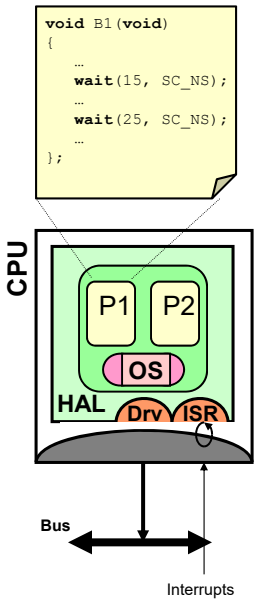
Computation Modeling (3)

- **Host-compiled models**
  - Source-level application model
    - Compile & execute natively
    - Fast functional simulation
  - Back-annotate timing and other metrics
  - Abstract OS and processor models
  - Transaction-level model (TLM) backplane
  - C-based discrete-event simulation kernel [SpecC,SystemC]



➢ **Fast and accurate full-system simulation**

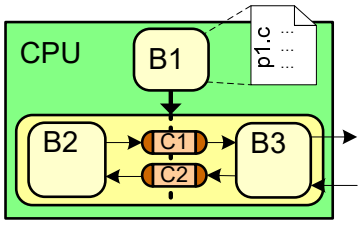
Host-Compiled Modeling Layers



- **Application**
  - Process execution (C code)
  - Execution timing
- **OS & processor**
  - 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

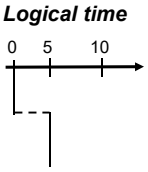
Application Layer

- **High-level, abstract programming model**
  - Hierarchical process graph
    - ANSI C leaf processes
    - Parallel-serial composition
  - Abstract, typed inter-process communication
    - Channels
    - Shared variables



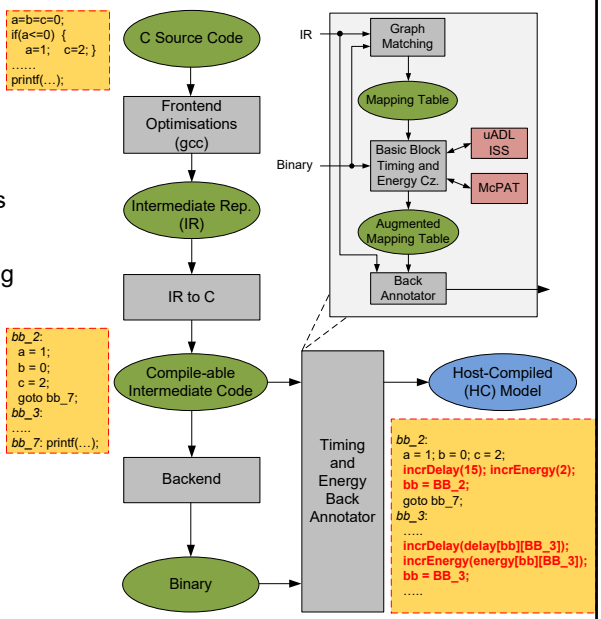
- **Timed simulation of application functionality**
  - Annotate timing, energy, …
    - Granularity?
    - Compiler optimizations?
    - Dynamic architecture effects?
  - Timing, energy estimation
    - Source profiling
    - Back-annotate from ISS
    - Predict from host activity using machine learning

```
...
void f() {
  wait(5,SC_NS);
  ...
}
...
```



Source-Level Modeling

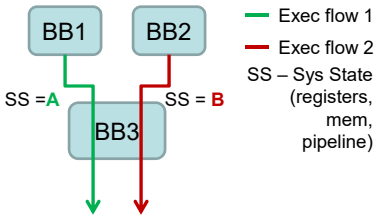
- **Automatic back-annotation**
  - Basic block level
    - Intermediate compiler representation (IR)
    - After frontend optimizations
  - Target binary matching
    - Control-flow graph matching
    - Memory access re-construction
  - Basic block characterization
    - Cycle-accurate or RTL
    - Energy model [McPAT]
  - Back-annotation
    - IR basic block level
    - Optional cache model



Source: Z. Zhao, L. John, A. Gerstlauer. "Source-Level Performance, Energy, Reliability, Power and Thermal (PERPT) Simulation," IEEE TCAD, 2016.

Timing/Energy Characterization

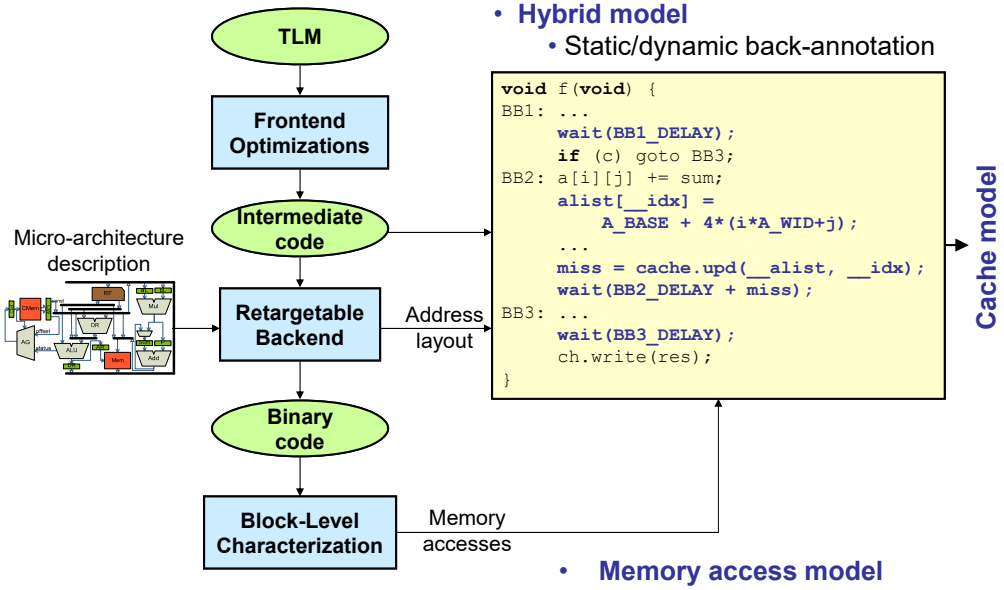
- **Basic block characterization**
  - Execution depends on state
    - Pipeline stalls in case of hazards
    - Pipeline overlaps in multi-issue
  - Pairwise characterization
    - Over all immediate predecessors
    - Across function hierarchy
  - Timing & energy
    - First-to-last instruction fetch time
    - Resource utilization statistics
- **Back-annotation into IR**
  - Path-dependent metrics
    - Capture static branch prediction



Annotated IR

```
bb_2:
  a = 1; b = 0; c = 2;
  goto bb_7;
wait(15); energy(2);
bb_3:
…..
If(prev_bb==3)
   wait(25); energy(5);
else if(prev_bb==1)
   wait(30); energy(6);
…..
bb_7: printf(…);
```


Cache-Aware Back-Annotation



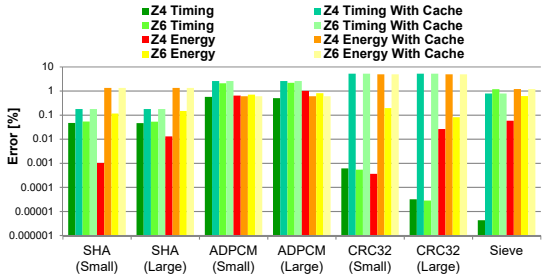
- **Hybrid model**
  - Static/dynamic back-annotation

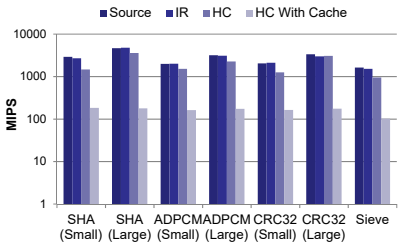
```
void f(void) {
BB1: ...
     wait(BB1_DELAY);
     if (c) goto BB3;
BB2: a[i][j] += sum;
     alist[__idx] =
         A_BASE + 4*(i*A_WID+j);
     ...
     miss = cache.upd(__alist, __idx);
     wait(BB2_DELAY + miss);
BB3: ...
     wait(BB3_DELAY);
     ch.write(res);
}
```

- **Memory access model**
  - Stack, heap address tracing

Source-Level Simulation

- **One-time back-annotation overhead**
  - 3min. to 3s runtime (function of code size)

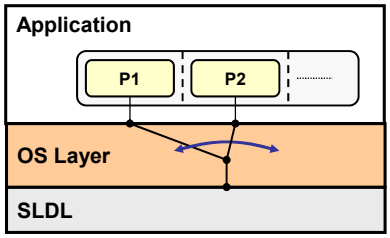




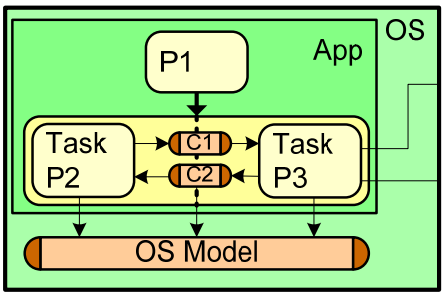
- **Close to cycle-accurate at source-level speeds**
  - >98% timing and energy accuracy @ 2000 MIPS
  - >95% accuracy @ 160 MIPS including cache
- **Extensions to power, thermal & reliability modeling**

Operating System Layer

- **Scheduling**
  - Group processes into tasks
    - Static scheduling
  - Schedule tasks
    - Dynamic scheduling, multitasking
    - Preemption, interrupt handling
    - Task communication (IPC)

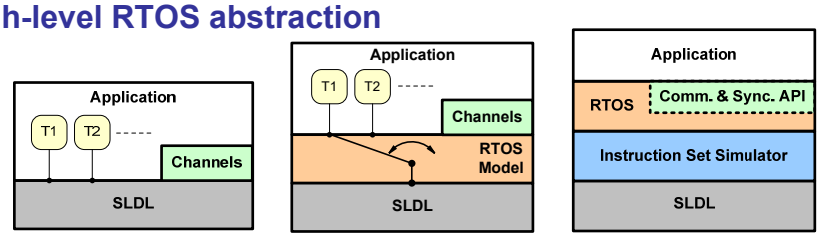


- **Scheduling refinement**
  - Flatten hierarchy
  - Reorder behaviors
- **OS refinement**
  - Insert OS model
  - Task refinement
  - IPC refinement



OS Modeling

- **High-level RTOS abstraction**



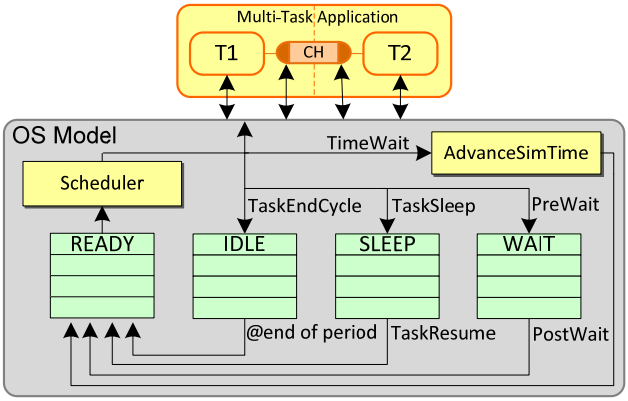
Specification *Host-Compiled* ISS/cycle-accurate

  - Specification is fast but inaccurate
    - Native execution, truly concurrent model
  - Traditional ISS-based validation infeasible
    - Accurate but slow (esp. in multi-processor context), requires full binary

  ➢ Model of operating system (task interleaving in time)
    - ➢ High accuracy but small overhead at early stages
    - ➢ Focus on key effects, abstract unnecessary implementation details
    - ➢ Model all concepts: multi-tasking, scheduling, preemption, interrupts, IPC

Source: A. Gerstlauer, H. Yu, D. Gajski. "RTOS Modeling for System-Level Design," DATE03.

Abstract RTOS Model



- **Emulate the sequential execution of concurrent tasks**
  - Task scheduler
    - Maintain task queues, determine task(s) to run & emulate context switching
  - Timing model
    - Simulate back-annotated task delays, call scheduler to allow for preemptions

Task Refinement

```
1  SC_MODULE(B2) {

5    SC_CTOR(B2) { SC_THREAD(task); }

     void task(void) {
10
       ...
       /* model execution delay */
       wait(BLOCK1_DELAY);
15     ...
       send();
       /* model execution delay */
       wait(BLOCK2_DELAY);

20     ...

     }

     void send() {
25

       wait(ack);

     }
30 };
```


Task Refinement

- **Convert processes into tasks**
  - Task initialization
    - Register task with OS model
  - Task activation
    - Wait for task start trigger from OS
  - Replace delay model
    - Trigger rescheduling in OS
    - ➢ Preemption points
- **Convert channels into IPC**
  - Communication and synchronization
    - Wrap around SLDL event handling

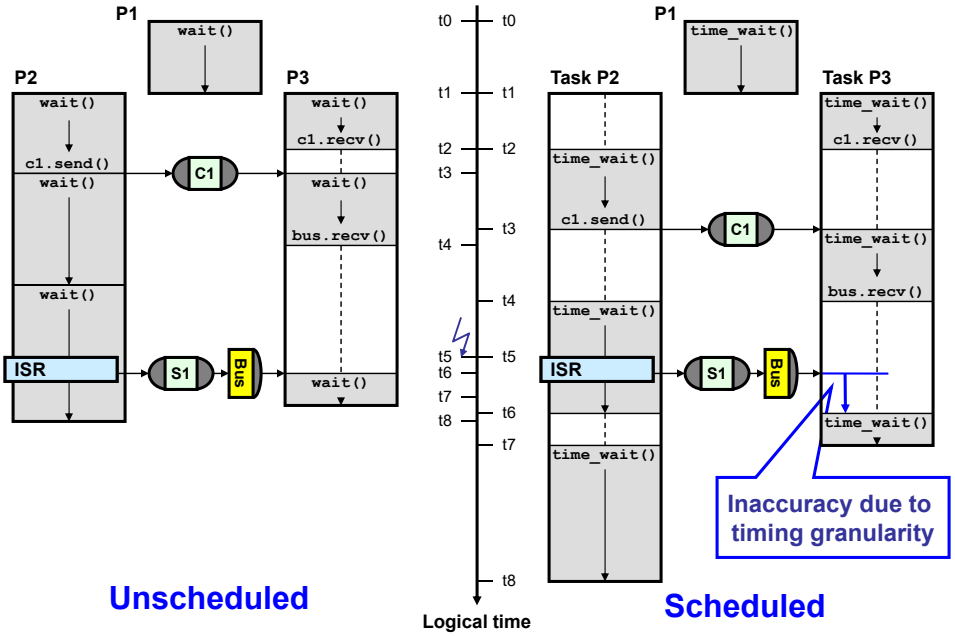
```
1  SC_MODULE(B2) {
     sc_port<OSAPI> os;
     Task h;
5    SC_CTOR(B2) { SC_THREAD(task); }
     void end_of_elaboration() {
       h = os->task_create("B2"); }
     void task(void) {
10     os->task_activate(h);
       ...
       /* model execution delay */
       os->time_wait(BLOCK1_DELAY);
15     ...
       send();
       /* model execution delay */
       os->time_wait(BLOCK2_DELAY);

20     ...
       os->task_terminate(h);
     }

     void send() {
25     t = os->pre_wait();
       wait(ack);
       os->post_wait(t);
     }
   };
```


RTOS Model Implementation

- **RTOS model**
  - OS, task, event management
    - Descriptors & queues
  - Context switching
    - Block all but active task on SLDL level
  - Scheduling
    - Select and dispatch task based on algorithm
  - Preemption
    - Allow rescheduling at simulation time increases
  - Event handling
    - Remove task temporarily from OS while waiting for SLDL event
- **RTOS model library**
  - RTOS models for different scheduling strategies
    - Round robin, priority based
  - Parametrizable
    - Task parameters (priorities)

```
1  class OS: public sc_channel,
              public OSAPI {
     Task current = 0;
     os_queue rdyq;
5
     void dispatch(void) {
       current = schedule(rdyq);
       if(current)
         current.event.notify();
10   }
     void yield() {
       task = current;
       rdyq.insert(task);
       dispatch();
15     wait(task.event);
     }
     void time_wait(sc_time t) {
       wait(t);
20     yield();
     }
     Task pre_wait(void) {
       Task t = current;
25     dispatch(); return t;
     }
     void post_wait(Task t) {
       rdyq.insert(t);
       if (!current) dispatch();
30     wait(t.event);
     }
};
```


Simulated Dynamic Behavior



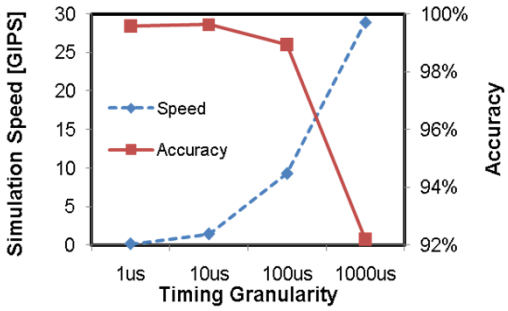
OS Modeling Results

- **Configurable, generic and flexible OS model**
  - Configurable scheduling strategies and parameters
    - Round-robin or priority-based scheduling
  - ➢ Scheduling exploration

➢ **Accuracy & speed**

- Artificial task set example

Granularity	Avg. speed per core	Avg. err.
1 µs	140 MIPS	0.4 %
10 µs	1500 MIPS	0.4 %
100 µs	9000 MIPS	1.0 %
1000 µs	29000 MIPS	8.0%



Advanced OS Modeling Approaches

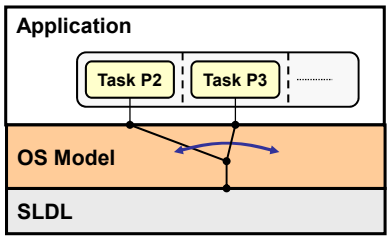
- **Conservative**
  - Predict possible preemption points
  - Simulate until next predicted point
  - Fall back to fine granularity if prediction is not possible
  - ➢ Automatic timing granularity adjustment (ATGA) [Razaghi'12]

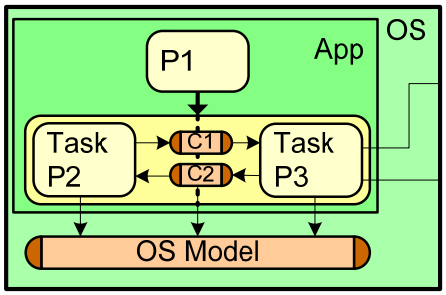
- **Optimistic**
  - Simulate at coarse granularity assuming no preemptions
  - Record disturbing influences
  - Correct and roll back if necessary
  - ➢ Result-oriented modeling (ROM) [Schirner'08]

Sources: P. Razaghi, A. Gerstlauer. "Predictive OS Modeling for Host-Compiled Simulation of Periodic Real-Time Task Sets," Emb. Sys. Letters '12.
P. Razaghi, A. Gerstlauer. "Automatic Timing Granularity Adjustment for Host-Compiled Software Simulation," ASPDAC'12.
G. Schirner, R. Dömer, " Introducing preemptive scheduling in abstract RTOS models using result oriented modeling," DATE'08.

Operating System Layer

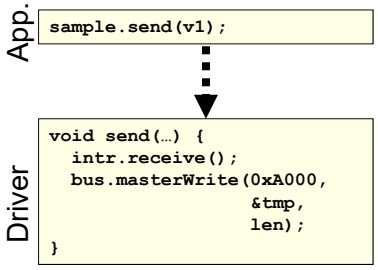
- **OS model**
  - On top of standard SLDL
  - Wrap around SLDL primitives, replace event handling
    - Block all but active task
    - Select and dispatch tasks
  - Target-independent, canonical API
    - Task management
    - Channel communication
    - Timing and all events

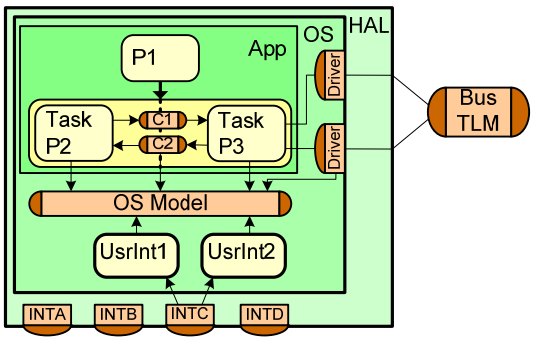




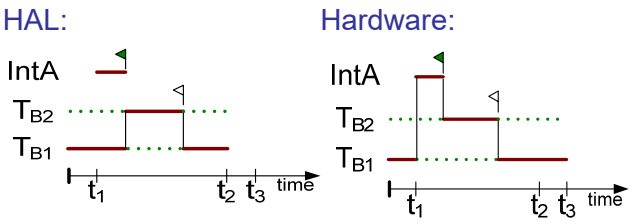
Hardware Abstraction Layer (HAL)

- **External communication**
  - Software Drivers
    - Presentation, session, network communication layers
    - Synchronization (interrupts)
  - Hardware/software boundary
    - Low-level HW access
    - Bus drivers and interrupt handlers
    - Canonical HW/SW interface
  - External interface
    - Bus transactions (TLM)
    - Interrupt trigger

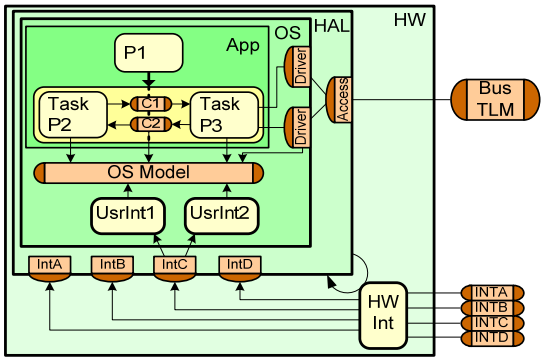




Hardware Layer (1)

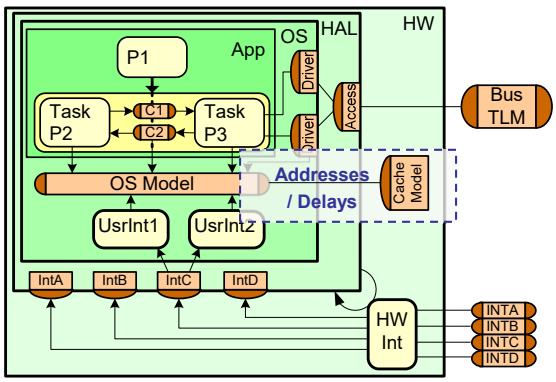


- **Processor model**
  - HW interrupt handling
    - Interrupt logic
      - » Suspend user code
    - Interrupt scheduling
      - » Priority, nesting
  - Peripherals
    - Interrupt controller
    - Timers
  - TLM bus model
    - Bus transactions



Hardware Layer (2)

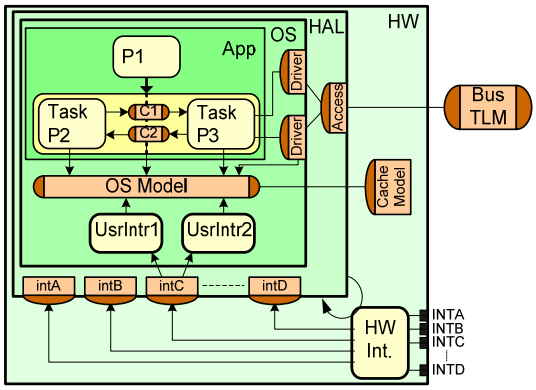
- **Cache modeling**
  - Pure behavioral modeling
    - Tag state
    - Hits/misses
    - Replacement policy
  - Integrated into back-annotation
    - Called with accessed address trace
    - Update cache state
    - Return delay penalties
  - ➢ Implemented as SystemC channel
    - < 200 lines of code



Source: A. Pedram, D. Craven, T. Amimeur, A. Gerstlauer. "Modeling Cache Effects at the Transaction Level," IESS 2009.

Computation Models

- **Processor layers**
  - Application
    - Native, host-compiled C
    - Back-annotation
  - OS
    - OS model
    - Middleware, drivers
  - HAL
    - Firmware
  - Processor hardware
    - Bus interfaces
    - Interrupts
    - Cache

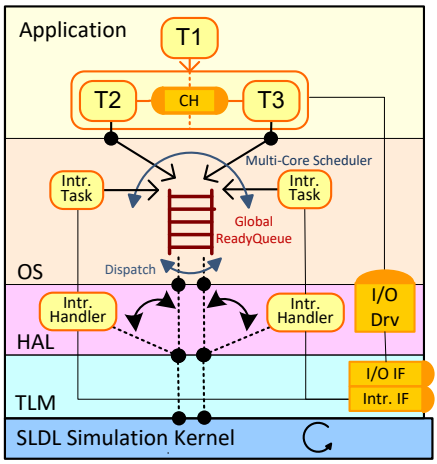


Features	
Target approx. computation timing	Appl.
Task mapping, dynamic scheduling	OS
Task communication, synchronization	HAL
Interrupt handlers, low level SW drivers	HW-TLM
HW interrupt handling, int. scheduling	HW-BFM
Cycle accurate communication	BFM - ISS
Cycle accurate computation	

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

Multi-Core OS & Processor Models

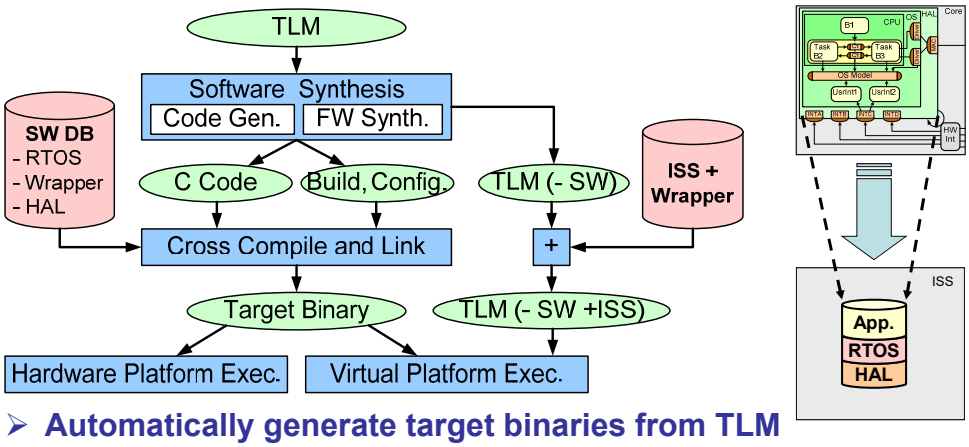
- **Multi-core OS model**
  - SMP scheduler model
    - Global or partitioned queue
  - Configurable
    - Number of cores
    - Scheduling policies
    - Priorities, affinity, time slice
- **Multi-core processor model**
  - Multi-core cache model
    - Out-of-order simulation handling
  - Multi-core interrupt models
    - Interrupt handlers & tasks
    - Interrupt controller



- **Full-system MPSoC simulation in close to real time**
  - 1400 MIPS at > 99% timing accuracy

Source: P. Razaghi, A. Gerstlauer. "Host-Compiled Multi-Core System Simulation for Early Real-Time Performance Evaluation," ACM TECS '14.

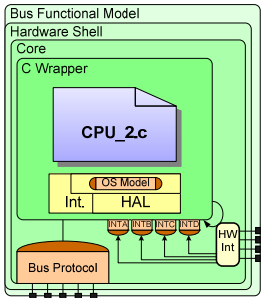
Software Synthesis



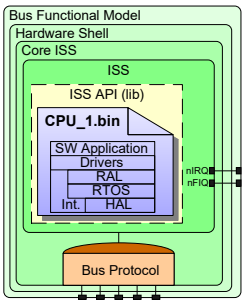
- **Automatically generate target binaries from TLM**
  - Generate code for application (tasks and IPC)
  - Synthesize firmware (drivers, interrupt handlers)
  - OS wrappers and HAL implementations from DB
  - Compile and link against target RTOS and libraries

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

Processor Implementation Models



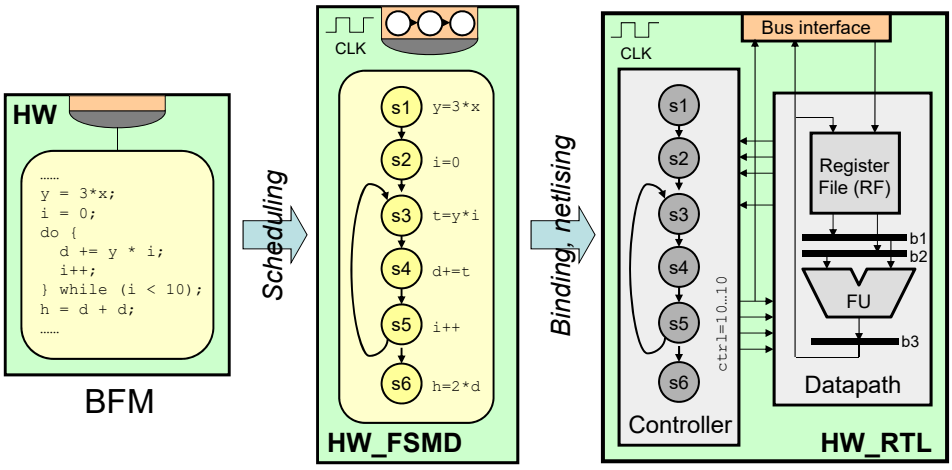
- **Software C model**
  - Generated application C code
    - Flat standard ANSI C code
  - Firmware and hardware models
    - RTOS model, HAL model
    - Low-level &hardware interrupt handling
    - External bus communication protocol/TLM



- **Software ISS model**
  - Reintegrared processor ISS
    - Bus-functional ISS wrapper
  - Running generated binary
    - Application, RTOS, drivers, HAL

Hardware Synthesis

- **C-to-RTL high-level synthesis (HLS)**
  - Allocation, scheduling, binding



Source: D. Shin, A. Gerstlauer, R. Doemer, D. Gajski. "An Interactive Design Environment for C-based High-level Synthesis of RTL Processors," TVLSI, 2008.

Lecture 6: Outline

- ✓ **System-level refinement**
  - ✓ Modeling & refinement flow
- ✓ **Host-compiled modeling of computation**
  - ✓ Source-level application models
  - ✓ Operating system and processor models
  - ✓ Hardware models
- **Transaction-level modeling of communication**
  - Communication layers and protocol stacks
  - Transaction-level modeling (TLM) of busses
  - Physical models

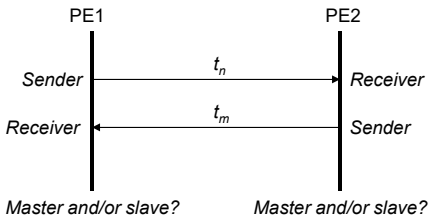
Specification-Level Communication

- **Events, transitions**
  - Pure control flow, no data
- **Shared variables**
  - No control flow, no synchronization
- **Synchronous message passing**
  - No buffering, two-way control flow
- **Asynchronous message passing**
  - Only control flow from sender to receiver guaranteed
  - May or may not use buffers (implementation dependent)
- **Queues**
  - Fixed, defined queue length (buffering)
- **Complex channels**
  - Semaphores, mutexes

➢ **Reliable communication primitives (lossless, error-free)**

Implementation-Level Communication

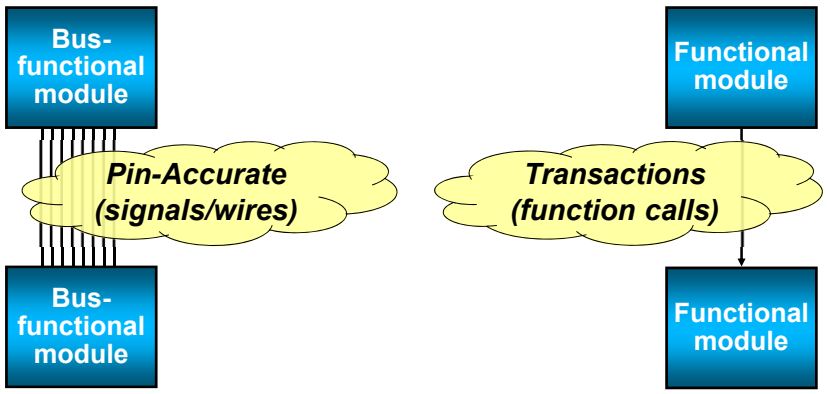
- **For each transaction between two communication partners**
  - 1 sender, 1 receiver
  - 1 master (initiator), 1 slave (listener)



➢ **Any combination of master/slave, sender/receiver**
- ➢ Master/Slave bus
  - ➢ Statically fixed master/slave assignments for each PE pair
  - ➢ PEs can be masters, slaves or both (dual-port)
- ➢ Node-based bus (e.g. Ethernet, CAN):
  - ➢ Sender is master, receiver is slave

➢ **Reliable (loss-less, error-free)??**

Communication Modeling



- **Pin-accurate model (PAM)**
  - Simulate every event (protocols)
- ➢ **Transaction-level model (TLM)**
  - Communications by transactions (abstract channels)
  - Granularity of transactions? Dynamic effects?

Source: OSCI TLM-2.0

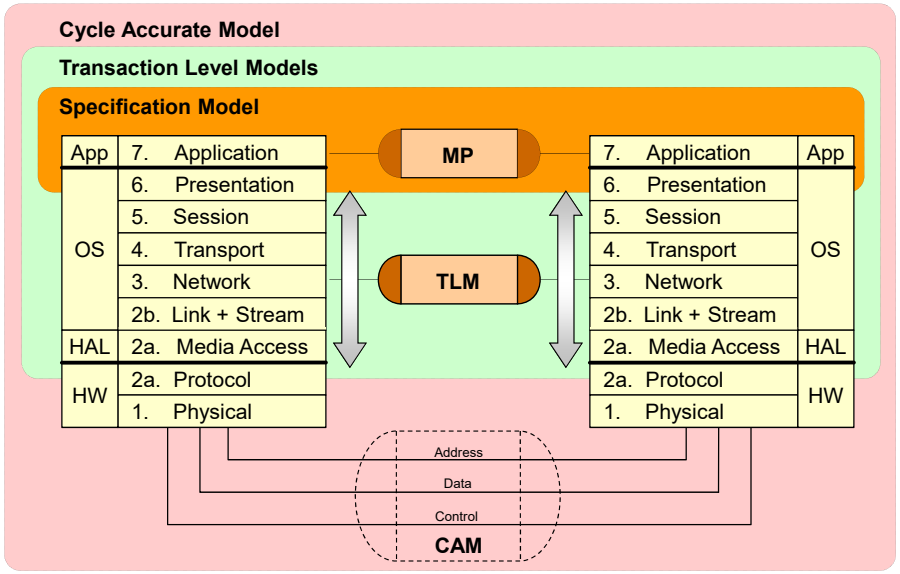
Communication Layers

- **ISO/OSI 7-layer network model**

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

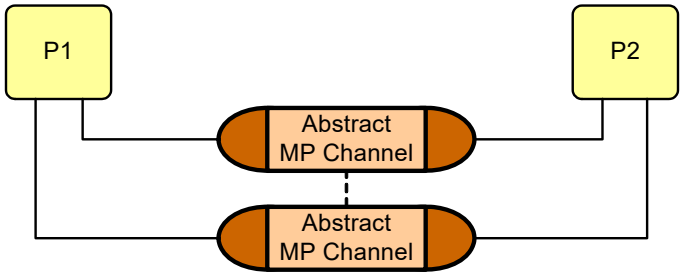
- ➢ **A *model*, not an implementation !**

Communication Models

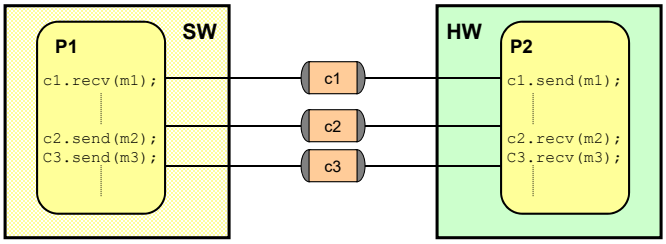


Specification Model

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



Architecture Model

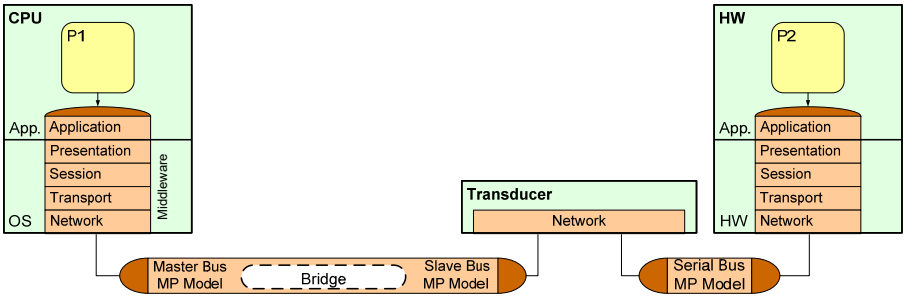


- **Application layer (virtual system architecture)**
  - Mapped computation
    - Processing elements (functionality)
    - Memories (storage)
  - Abstract end-to-end communication
    - Queues, semaphores
    - Sync./async. message-passing
    - Shared variables/memories
    - Events, transitions

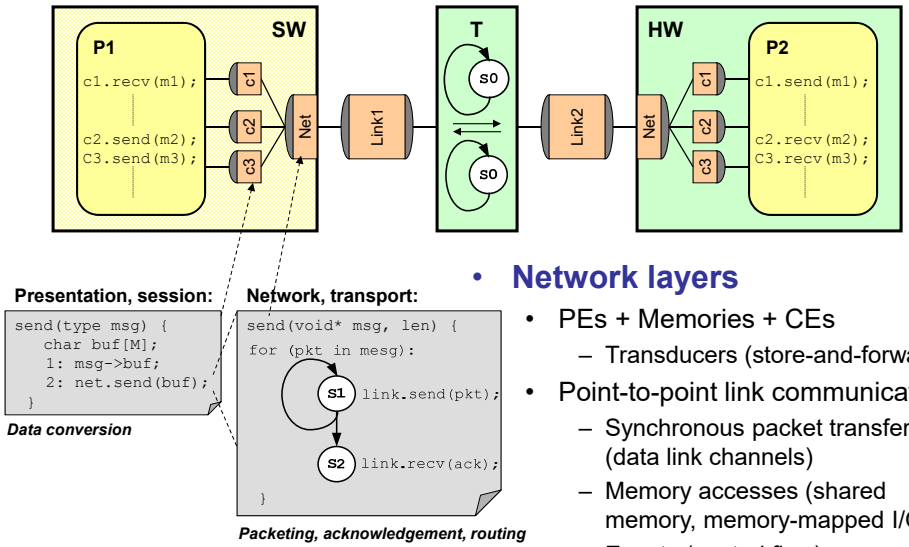
➢ **Reliable, loss-less application communication**

Network Model

- **Topology of communication architecture**
  - PEs + Memories + Communication Elements (CEs)
    - Bus bridges and transducers/routers
  - Upper protocol layers inserted into PEs/CEs
  - Communication via point-to-point links
    - Synchronous packet transfers (data transfers)
    - Memory accesses (shared memory, memory-mapped I/O)



Network Model

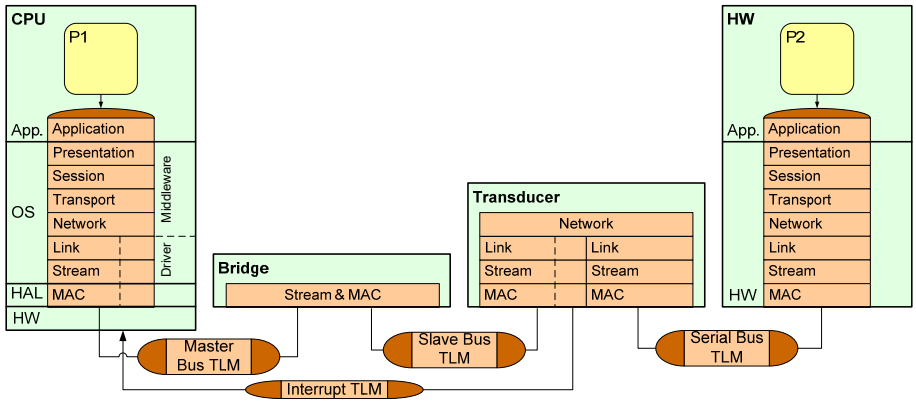


- **Network layers**
  - PEs + Memories + CEs
    - Transducers (store-and-forward)
  - Point-to-point link communication
    - Synchronous packet transfers (data link channels)
    - Memory accesses (shared memory, memory-mapped I/O)
    - Events (control flow)

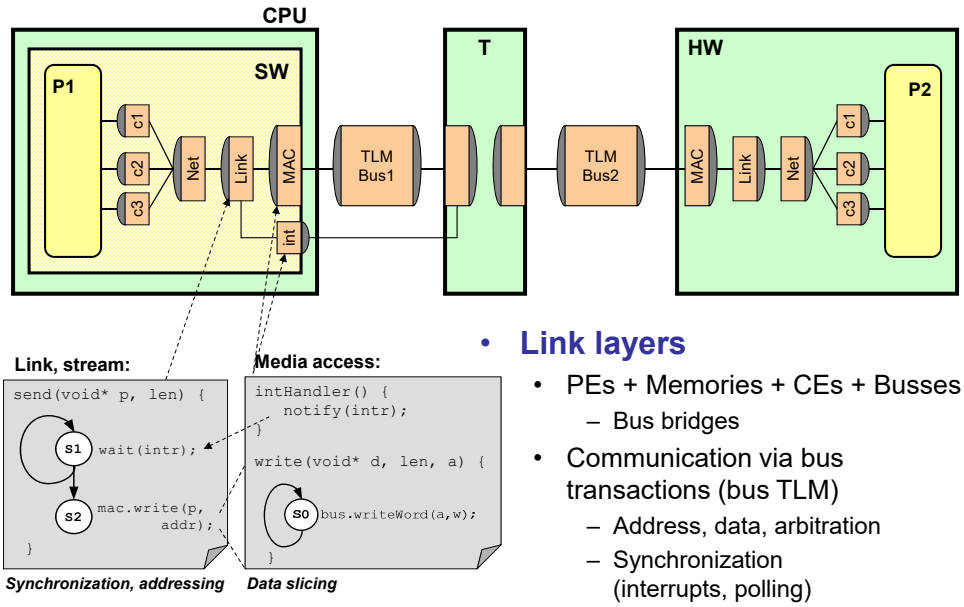
➢ **Communication topology**

Transaction-Level Model (TLM)

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



Transaction-Level Model (TLM)



- **Link layers**
  - PEs + Memories + CEs + Busses
    - Bus bridges
  - Communication via bus transactions (bus TLM)
    - Address, data, arbitration
    - Synchronization (interrupts, polling)

➢ **System communication architecture**

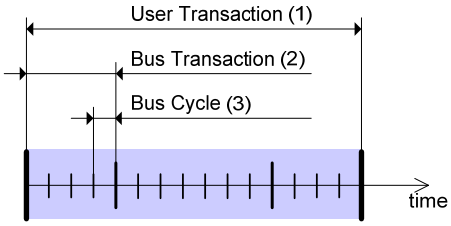
TLM Abstraction Levels

- **ISO/OSI reference layer-based architecture**
  - Granularity of data and arbitration handling
- **Abstraction levels**

1) Media Access Control (MAC)
   - User Transaction
     - » Contiguous block of bytes
     - » Arbitrary length, base address

2) Protocol
   - Bus Transaction
     - » Bus primitives (e.g. store word)
     - » Observes bus address restrictions

3) Physical
   - Bus Cycle
     - » Drive or sample bus wires on bus cycle



- **Abstraction levels define granularity**
  - Higher level yields a more coarse-grain model

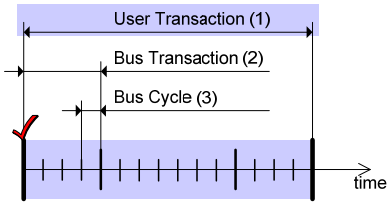
Source: G. Schirner, R. Dömer, "Quantitative Analysis of the Speed/Accuracy Tradeoff in Transaction-Level Models," ACM TECS, 2008.

Media Access Model (MAC)

Implemented Layers:

MAC

Granularity:



- **User transaction (message)**
  - Arbitrary length, contiguous block of bytes
- **No arbitration: contention avoidance by semaphore**
  - Resolution depends on simulator
- ➢ **Expected to be the fastest model**
  - Single **`memcpy`**, Single **`time wait`**

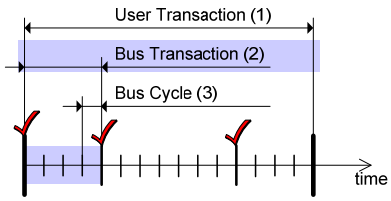
Transaction Level Model (TLM)

Implemented Layers:

MAC

Protocol

Granularity:



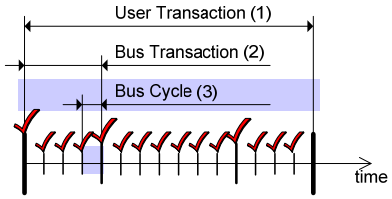
- **Bus primitives**
  - StoreWord, StoreBurst4
- **Abstract model**
  - Not pin accurate, not bus cycle accurate in all cases
- **Priority arbitration per bus transaction**
  - ➢ May lead to wrong arbitration decision, depending on execution order

Bus Functional Model (BFM)

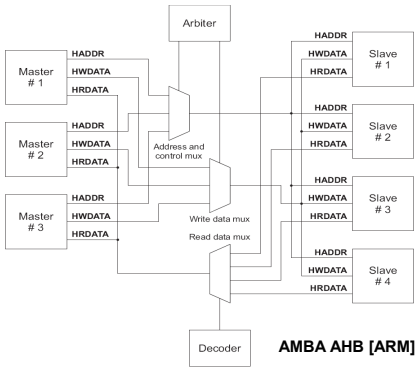
Implemented Layers:

- MAC
- Protocol
- Physical

Granularity:



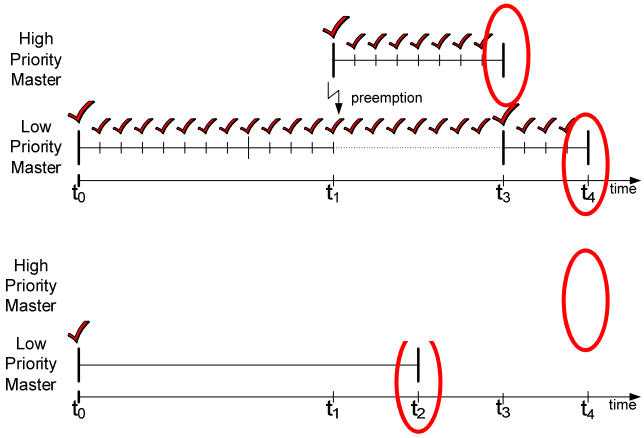
- **May or may not be pin accurate**
- **Bus cycle accurate**
  - Arbitration check ✓ on each cycle
- **Includes additional active components**
  - Multiplexers (tri-state-free bus)
  - Arbiter
  - Address decoder
  - Clock generator



AMBA AHB [ARM]

TLM Accuracy Limitations

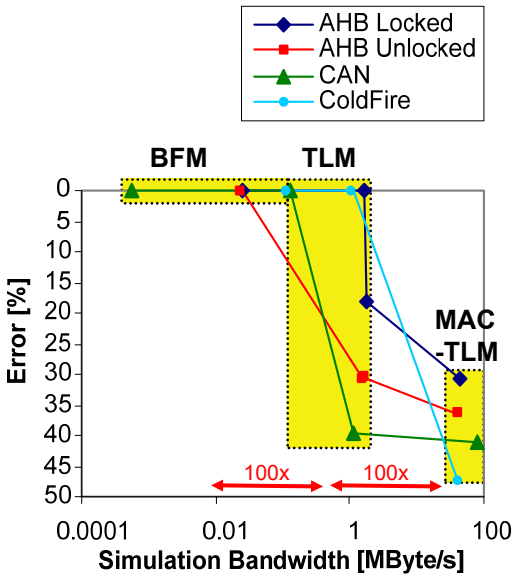
- **Example:**
  - Low priority burst starting at $\mathbf{t_0}$
  - High priority preemption at $\mathbf{t_1}$
- BFM:
  - check every cycle
  - slow
  - accurate
- TLM:
  - coarse grain check
  - fast
  - inaccurate: low prio. burst ends at t_2 instead of t_4



TLM Trade-Off

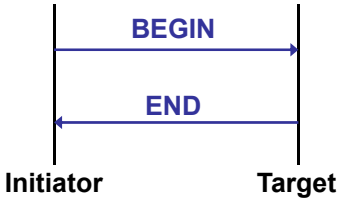
- **Bus models**
  - AMBA AHB, CAN, ColdFire
  - BFM, TLM, MAC-TLM
- **Performance analysis**
  - 2 masters, 2 slaves
  - Randomly distributed traffic
  - 100 byte transactions
  - 40% bus contention
  - ➢ Transfer duration

Model	Speed	Error
M-TLM	<100 MByte/s	32% - 47%
TLM	~ 1 MByte/s	18% - 39%
BFM	<0.2 MByte/s	0%

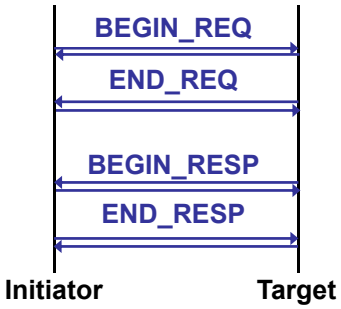


SystemC/TLM

- **Loosely-timed**
  - Sufficient timing detail to boot OS and simulate multi-core systems
  - Each transaction has 2 timing points: *begin* (call) and *end* (return)
  - Typically *blocking transport* calls



- **Approximately-timed**
  - Cycle-approximate or cycle-count-accurate (incl. bus preemption, pipelining, split transactions)
  - Sufficient for architectural exploration
  - Each transaction has at least 4 timing points
  - Typically *non-blocking transport* calls



Source: OSCI TLM-2.0

Advanced TLM Approaches

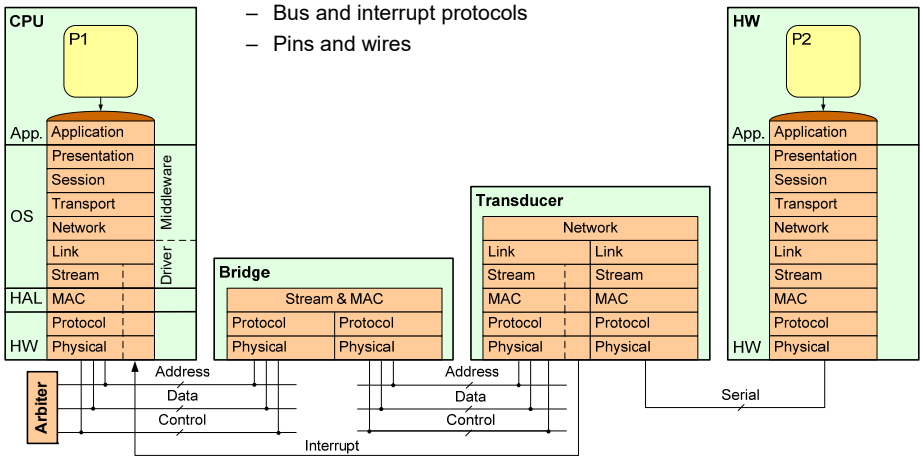
- **System modeling tricks [SystemC/TLM]**
  - Early completion of non-blocking REQ/RESP sequence
  - Timing annotation on call/return (pass)
  - Temporal decoupling & quantum (lump, out-of-order)
  - Direct interface (bypass)

- **Advanced TLM techniques (see OS modeling)**
  - Conservative: Temporal decoupling [Lu'13]
  - Optimistic: Result-oriented modeling (ROM) [Schirner'06]

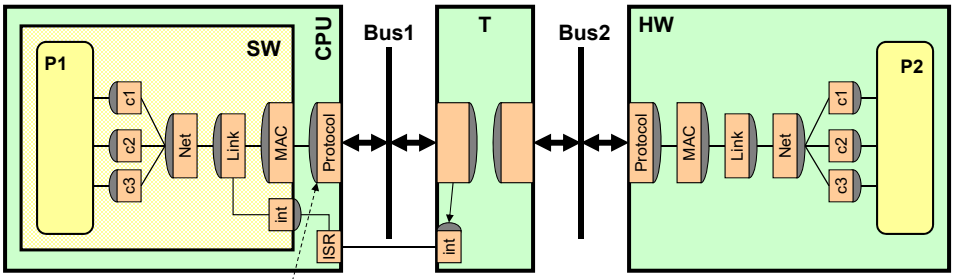
Sources: G. Schirner, R. Dömer, "Fast and Accurate Transaction Level Models using Result Oriented Modeling," ICCAD, 2006.
K. Lu, D. Müller-Gritschneder, U. Schlichtmann, "Analytical timing estimation for temporally decoupled TLMs considering resource conflicts," DATE, 2013.

Bus-Functional Model (BFM)

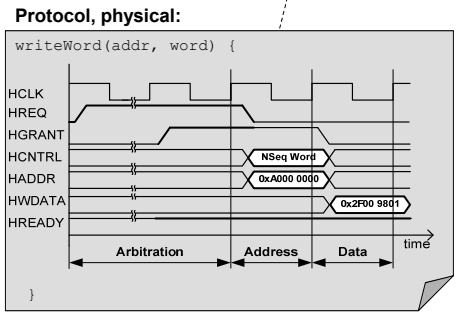
- **Component & bus structure/architecture**
  - PEs + Memories + CEs + Busses
  - Pin-accurate bus-functional components
  - Pin- and cycle-accurate communication
    - Bus and interrupt protocols
    - Pins and wires



Bus-Functional Model (BFM)



Protocol, physical:

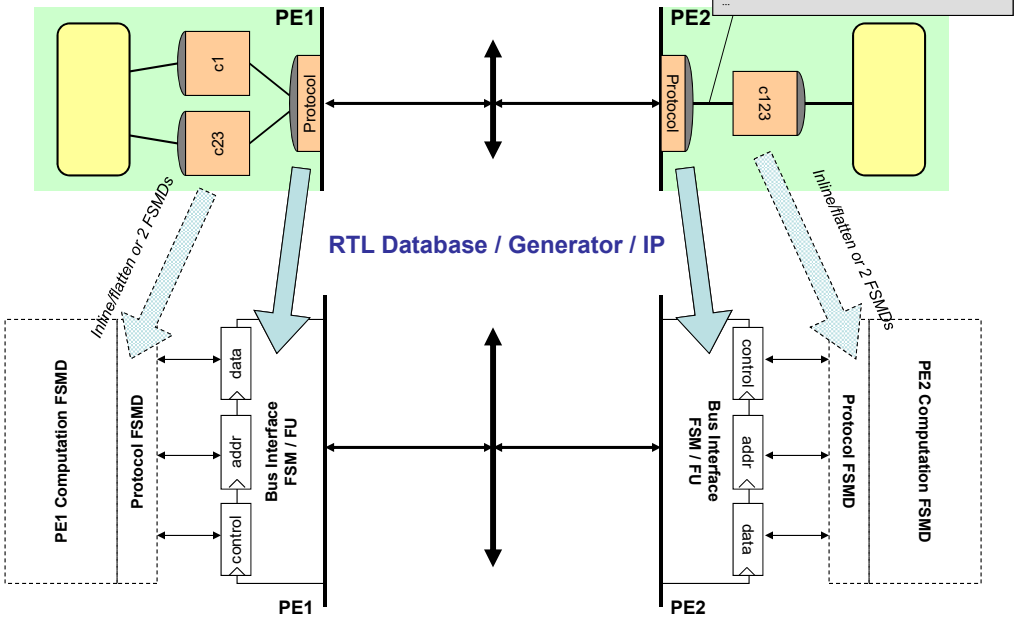


Protocol timing, wire driving & sampling

- **Bus-functional layers**
  - PEs + Memories + CEs + Busses
    - Pin-, cycle- and bit-accurate bus-functional components
  - Communication via ports and wires
    - Address, data, control busses
    - Interrupts

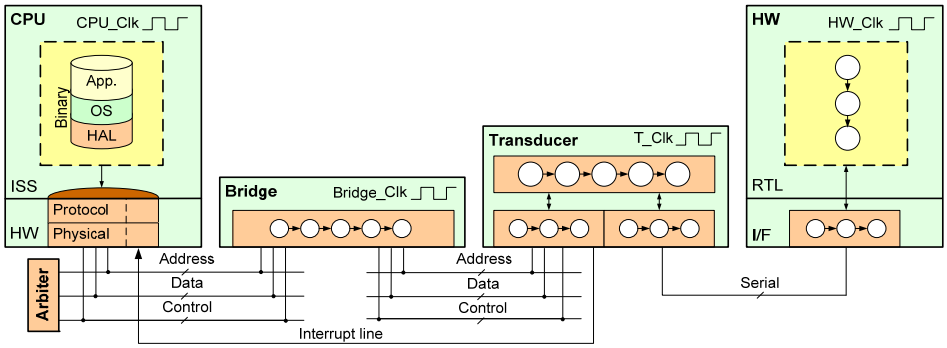
Interface Synthesis

```
readByte(bit[] addr, bit[8] data)
writeByte(bit[] addr, bit[8] data)
readWord(bit[] addr, bit[16] data)
writeWord(bit[] addr, bit[16] data)
…
```



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



Lecture 6: Summary

- **Host-compiled computation modeling**
  - Model of software running in execution environment
    - Timed application, OS, bus drivers, interrupt handlers
    - Processor hardware model, suspension, bus interfaces
  - ➢ Virtual platform prototype
    - ➢ Embedded software development and validation
    - ➢ Viable complement to ISS-based validation
- **Transaction-level communication modeling**
  - Systematic, structured communication design flow
    - Protocol stacks and communication layers
    - Layer-based modeling and refinement
  - ➢ Transaction-level modeling (TLM)
    - ➢ Rapid, early feedback, validation and exploration
    - ➢ Various levels of abstraction, accuracy vs. speed tradeoffs