

## EE382N.23: Embedded System Design and Modeling

---

### Lecture 8 – Implementation Modeling

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



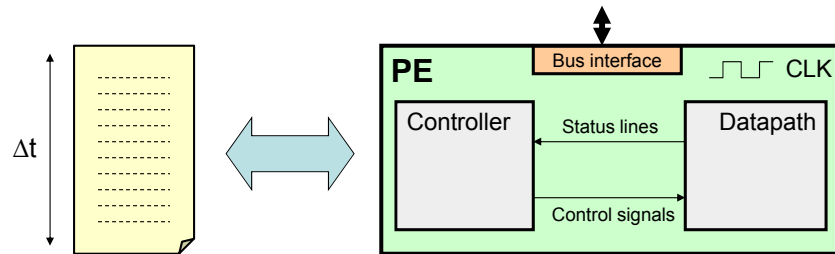
### Lecture 8: Outline

---

- **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

## General Processor Micro-Architecture

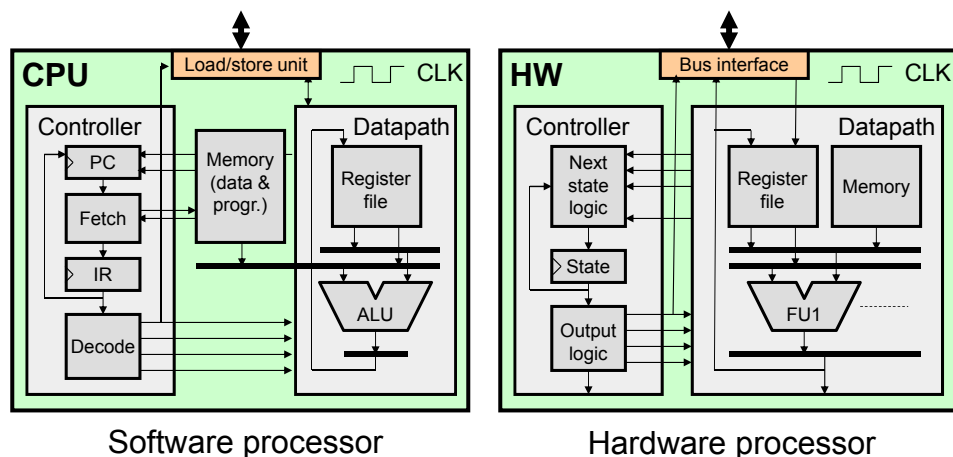
- **Basic computation component is a *processor (PE)***
  - Programmable, general-purpose software processor (CPU)
  - Programmable special-purpose processor (e.g. DSPs)
  - Application-specific instruction set processor (ASIP)
  - Custom hardware processor



- **Functionality *and* timing (and power and ...)**

## Computation Modeling (1)

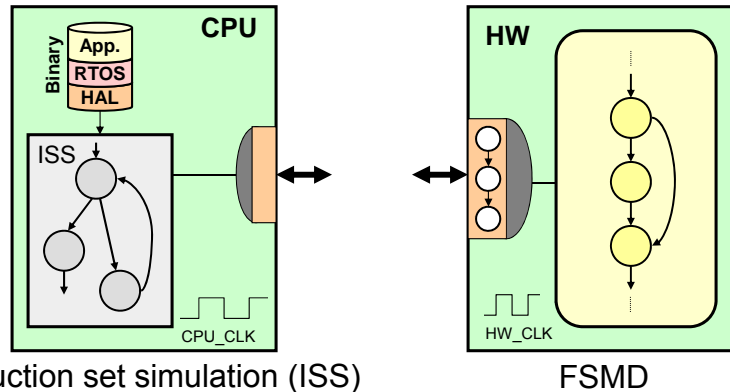
- **Structural RTL models**



- **Sub-cycle accurate**

## Computation Modeling (2)

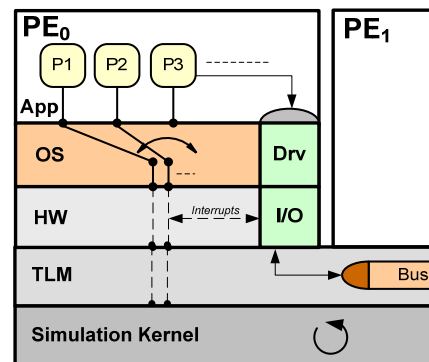
- Behavioral RTL models (FSMD)
- Instruction-set simulation (ISS) models
  - Purely functional (binary translation) [QEMU,...]
  - Micro-architectural (RTL in C) [GEM5,...]



- Cycle or timing accurate

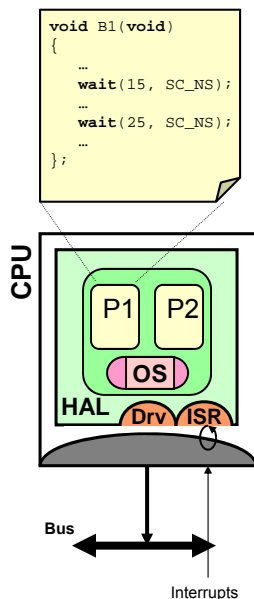
## Computation Modeling (3)

- 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 Computation 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

EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

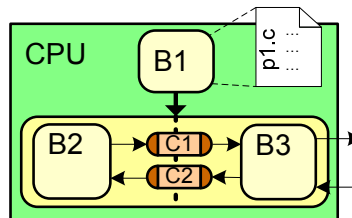
© 2019 A. Gerstlauer

7

## 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?

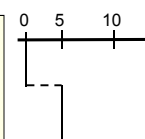
### ➤ Timing, energy estimation

- Source profiling
- Back-annotate from ISS
- Predict from host activity using machine learning

```

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

Logical time



EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

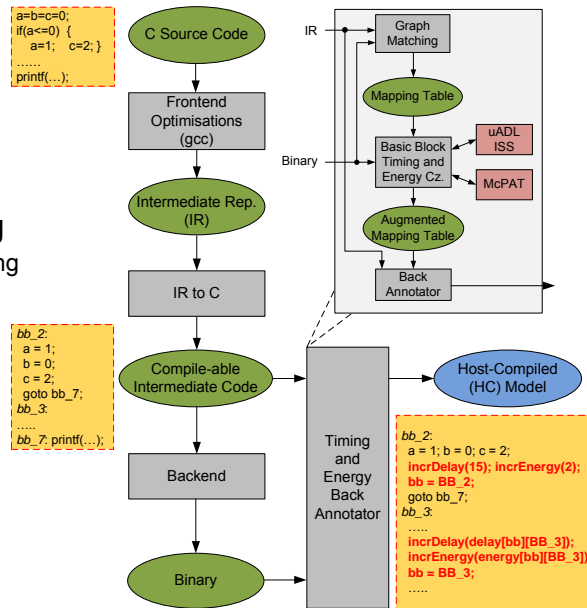
© 2019 A. Gerstlauer

8

## Source-Level Back-Annotation

- **Retargetable back-annotation**

- Intermediate representation (IR)
  - Frontend optimizations
  - IR to C conversion [gcc]
- 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.

EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

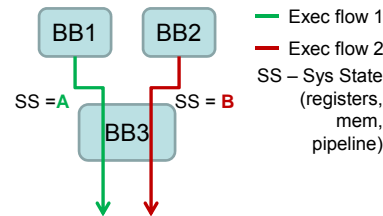
© 2019 A. Gerstlauer

9

## 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

```

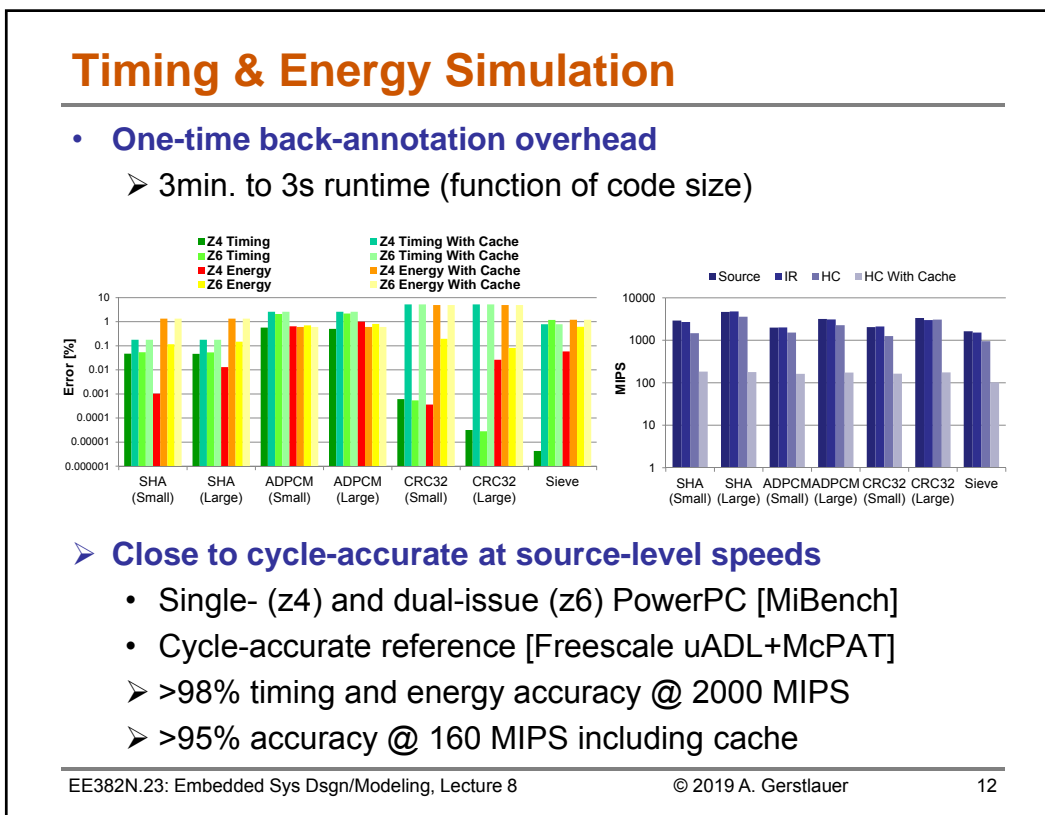
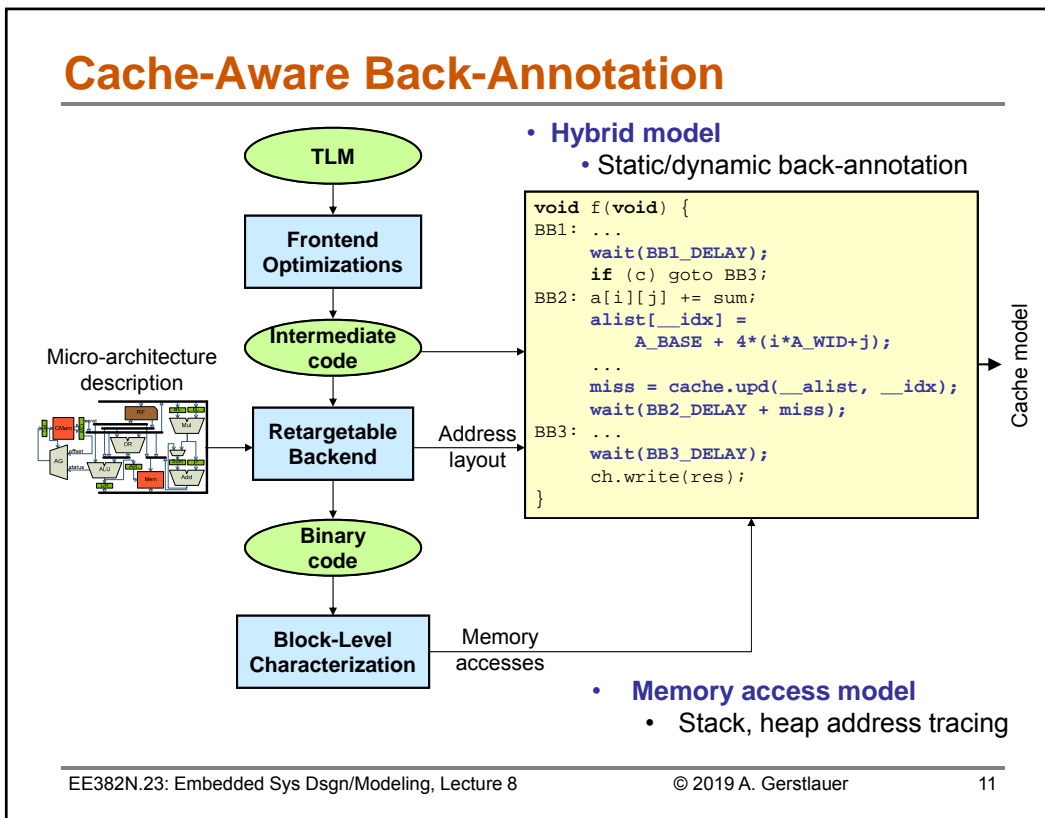
bb_2:
a = 1; b = 0; c = 2;
goto bb_7;
wait(15); energy(2);
bb_3:
.....
If(prev_bb==3)
  wait(25); energy(5);
else if(prev_bb==1)
  wait(30); energy(6);
.....
bb_7: printf(...);

```

EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

© 2019 A. Gerstlauer

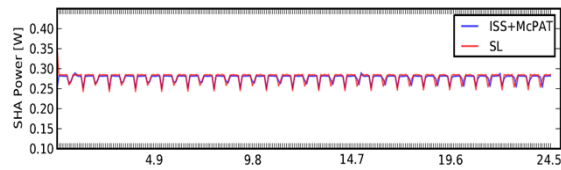
10



## Power & Thermal & Reliability

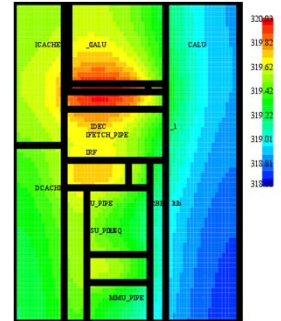
- **Power**

- Source-level (SL) energy and timing back-annotation



- **Thermal**

- HotSpot (HS) or discrete-time (DTTEM) thermal models
- FastSpot (FS) [ISQED'14]



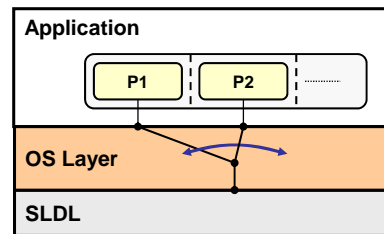
- **Soft-error vulnerability**

- Architectural vulnerability factor (AVF) estimation at source level [SELSE'15]

## Operating System Layer

- **Scheduling**

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

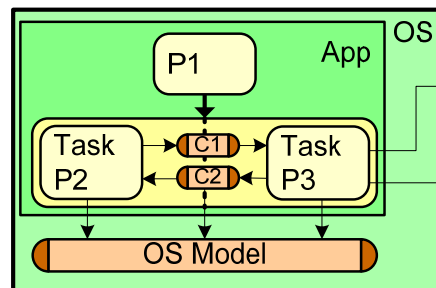


- **Scheduling refinement**

- Flatten hierarchy
- Reorder behaviors

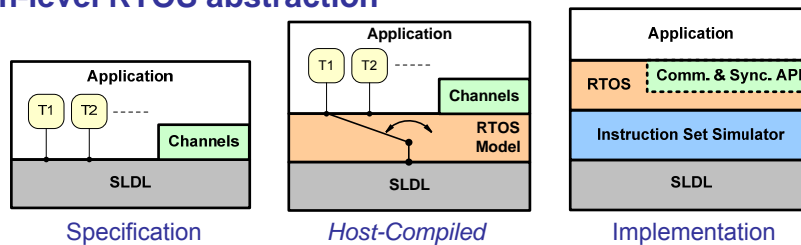
- **OS refinement**

- Insert OS model
- Task refinement
- IPC refinement



## OS Modeling

### • High-level RTOS abstraction



- Specification 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

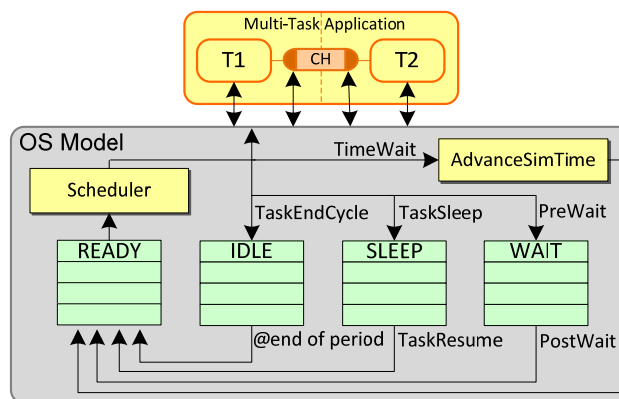
Source: A. Gerstlauer, H. Yu, D. Gajski. "RTOS Modeling for System-Level Design," DATE03.

EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

© 2019 A. Gerstlauer

15

## Abstract RTOS Model



### • Emulate the sequential execution of concurrent tasks

- Task scheduler
  - Maintain task queues, determine task(s) to run & perform context switch
- Timing model
  - Simulate back-annotated task delays, call scheduler to allow for preemptions

EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

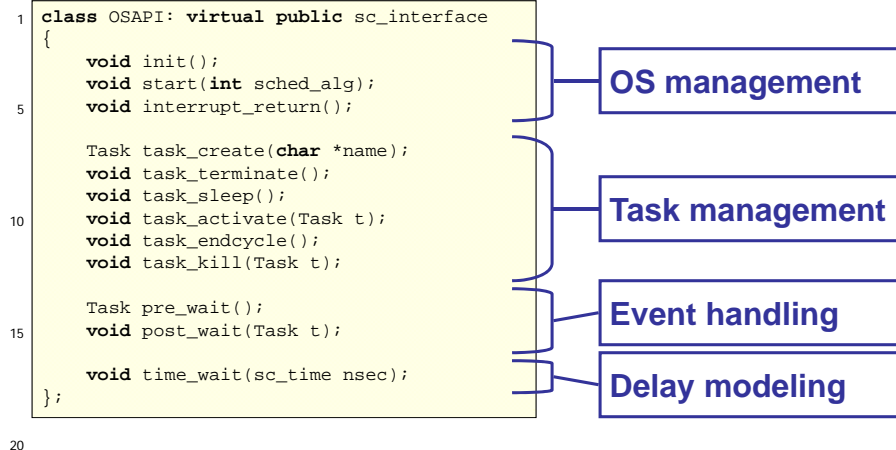
© 2019 A. Gerstlauer

16



## RTOS Model Interface

- Canonical, target-independent API



EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

© 2019 A. Gerstlauer

17

## 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();
    wait(task.event);
  }

  15 void time_wait(sc_time t) {
    wait(t);
    yield();
  }

  20 Task pre_wait(void) {
    Task t = current;
    dispatch(); return t;
  }

  25 void post_wait(Task t) {
    rdyq.insert(t);
    if (!current) dispatch();
    wait(t.event);
  }

  30 };

```

EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

## 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
};

```

EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

## 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

```

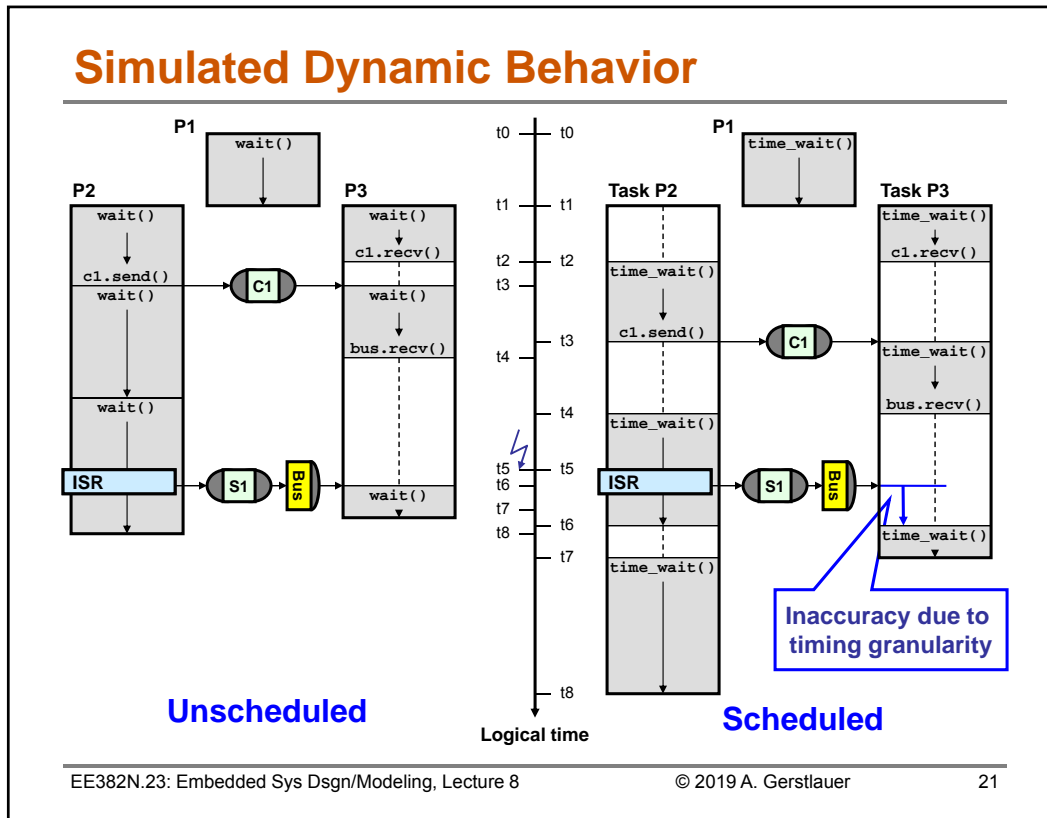
1  SC_MODULE(B2) {
   sc_port<OSAPI> os;
   Task h;
5
   SC_CTOR(B2) { SC_THREAD(task); }
   void end_of_elaboration() {
       h = os->task_create("B2"); }

   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);
   }
30
};

```

EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8



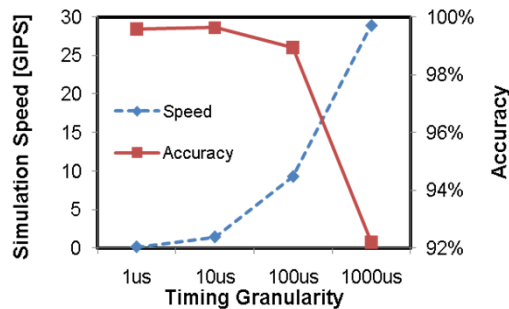
## OS Modeling Results

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

### ➤ Accuracy & speed

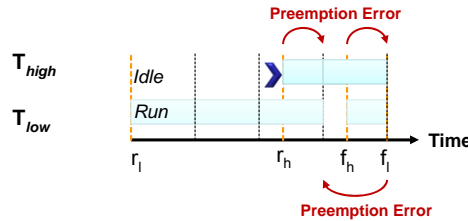
- Artificial task set example

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



## Speed and Accuracy Tradeoffs

- Errors in discrete preemption models



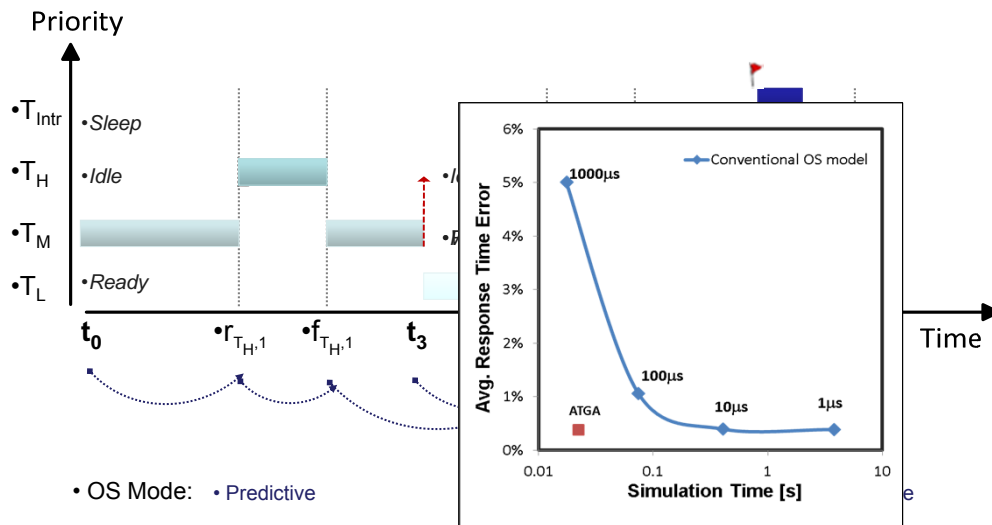
- Potentially large preemption errors
  - Not bounded by simulation granularity

- Automatic Timing Granularity Adjustment (ATGA)

- Observe system state to predict preemption points
- Dynamically and optimally control timing model
- Transparently integrated into OS model
- Eliminate preemption errors

Source: 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

## ATGA Model Execution Example



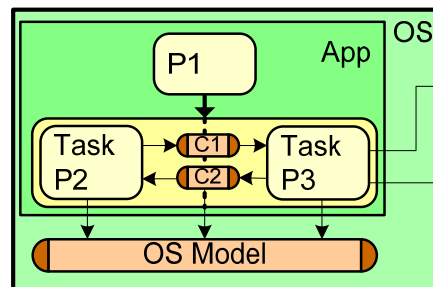
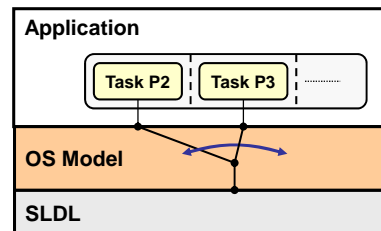
- Fast and accurate

## 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]

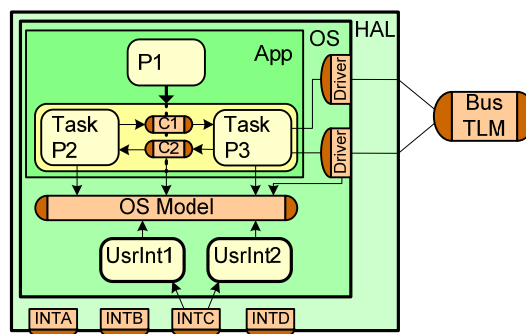
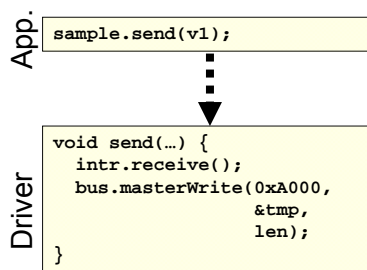
## Operating System Layer

- **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

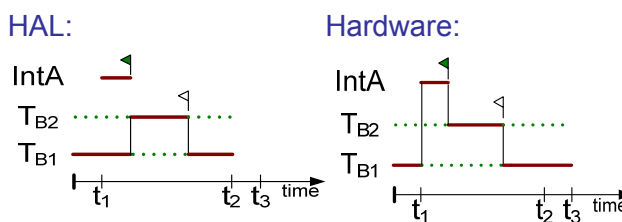


## Hardware Abstraction Layer (HAL)

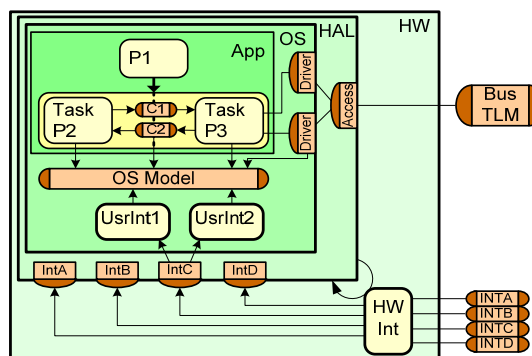
- External communication
  - Software Drivers
    - Presentation, session, network communication layers
    - Synchronization (interrupts)
  - Hardware/software boundary
    - Low-level HW access
    - Bus drivers and interrupt handlers
    - Canonical HW/SW interface
- External interface
  - Bus transactions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



## Hardware Layer (1)



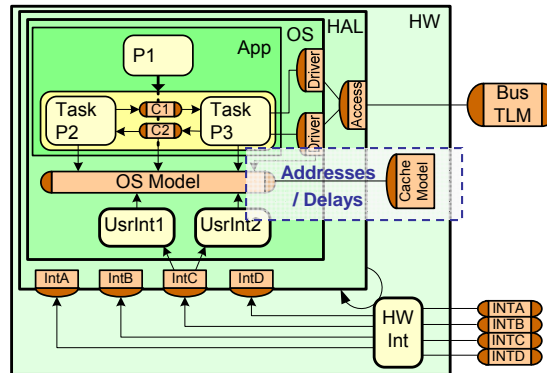
- Processor TLM
  - HW interrupt handling
    - Interrupt logic
      - » Suspend user code
    - Interrupt scheduling
      - » Priority, nesting
  - Peripherals
    - Interrupt controller
    - Timers
  - TLM bus model
    - Bus transactions



## 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 SpecC channel
  - < 200 lines of code



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

EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

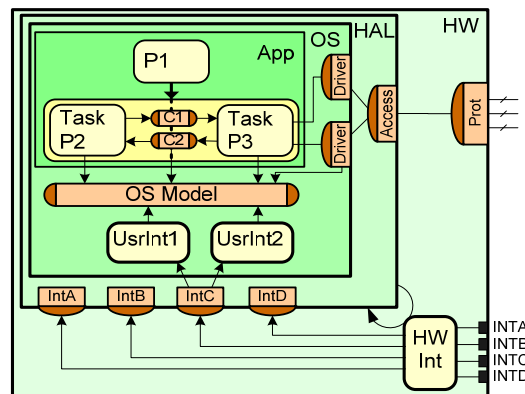
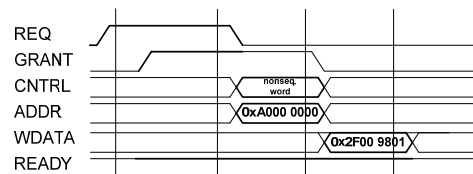
© 2019 A. Gerstlauer

29

## Hardware Layer (3)

- **Bus-functional model (BFM)**

- Pin-accurate processor model
  - Timing-accurate bus and interrupt protocols
- Bus model
  - Pin- and cycle-accurate
  - Driving and sampling of bus wires



EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

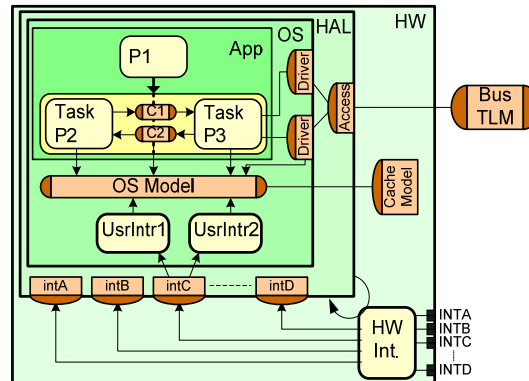
© 2019 A. Gerstlauer

30

## Processor 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.

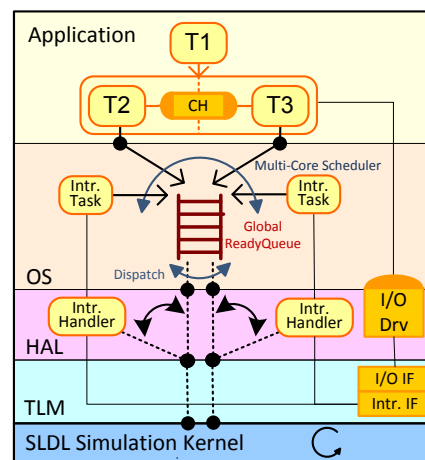
## Multi-Core OS & Processor Models

### • Multi-core OS model

- SMP scheduler model
  - Global or partitioned queue
- Configurable parameters
  - Number of cores
  - FIFO, round-robin, priority-based scheduling policies
  - Priorities, affinity, time slice (for round-robin)

### • Multi-core processor model

- Multi-core cache model
  - Out-of-order simulation handling
- Multi-core interrupt models
  - Interrupt handlers & tasks
  - Configurable generic interrupt controller (GIC) model
- TLM bus interfaces

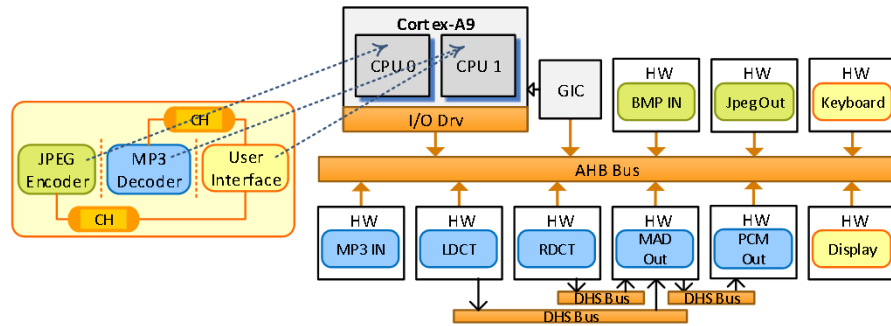


Source: P. Razaghi, A. Gerstlauer. "Host-Compiled Multi-Core System Simulation for Early Real-Time Performance Evaluation," ACM TECS '14.



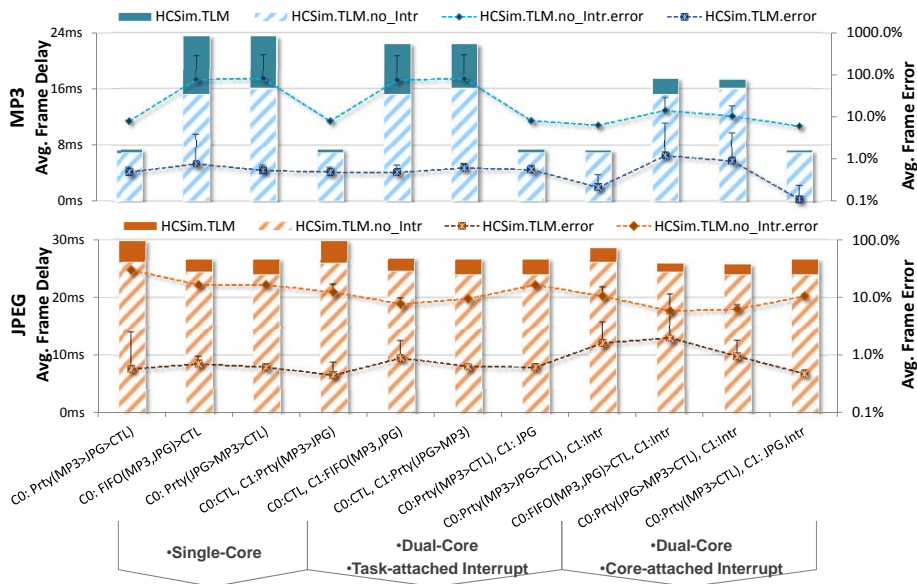
## Platform Simulation Example

- **Cellphone baseband MPSoC**
  - Design space exploration: mapping & scheduling

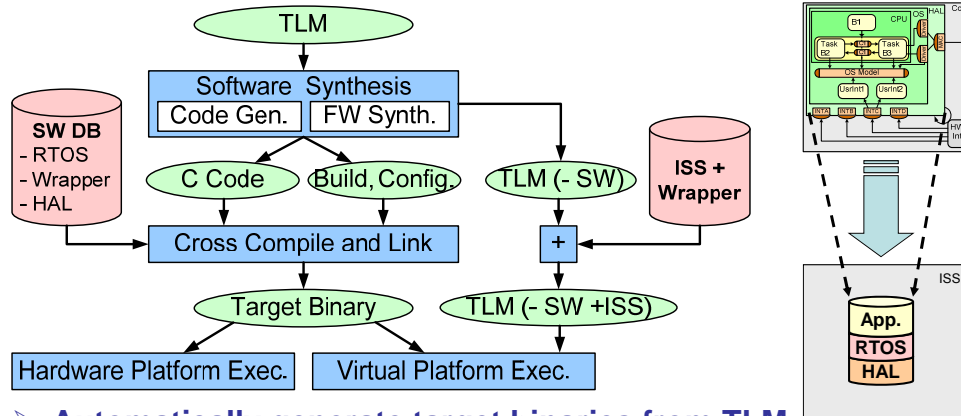


- **Full-system simulation in close to real time**
  - 1400 MIPS at > 99% timing accuracy

## MPSoC Exploration Results



## Software Synthesis

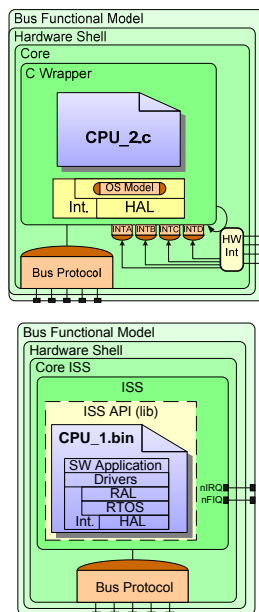


### ➤ Automatically generate target binaries from TLM

- 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

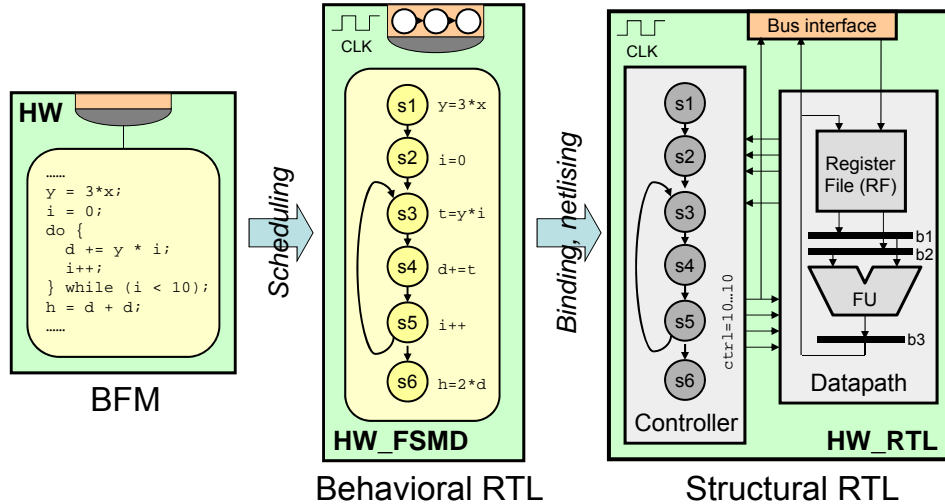
- Generated application C code
  - Flat standard ANSI C code
- Firmware and hardware models
  - RTOS model, HAL model
  - Low-level & hardware interrupt handling
  - External bus communication protocol/TLM

### • Software ISS model

- Reintegrated 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.

EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

© 2019 A. Gerstlauer

37

## Lecture 8: Outline

- ✓ **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

EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

© 2019 A. Gerstlauer

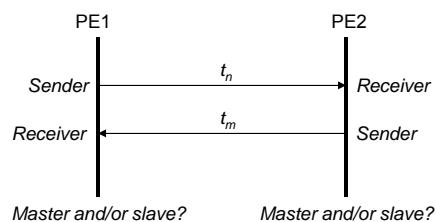
38

## 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)**

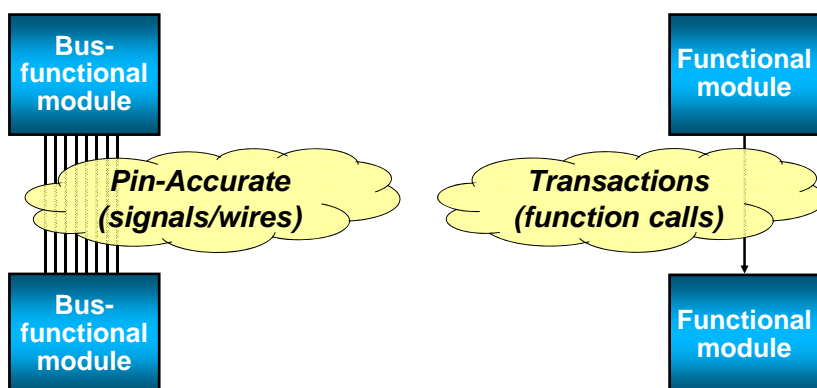
## Taxonomy of “Busses”

- **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

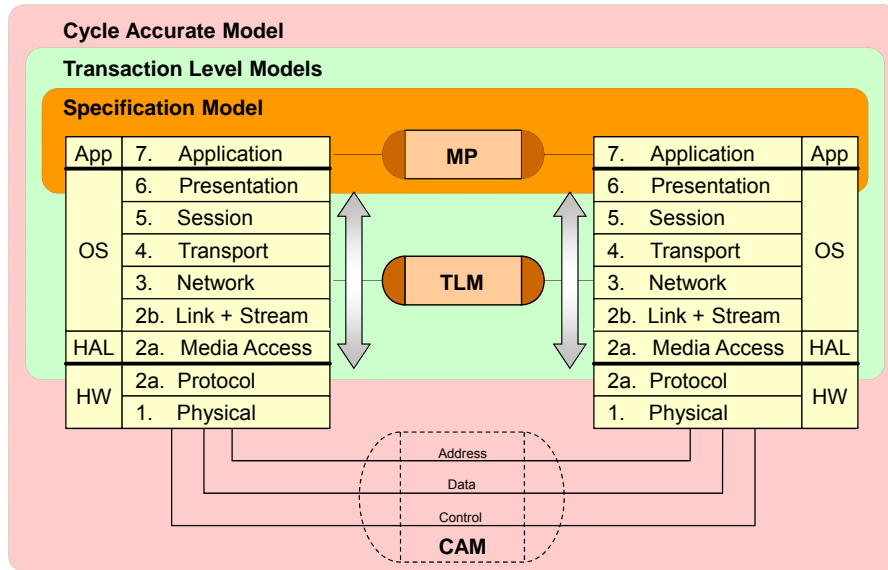
## Communication Layers

- **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	Packaging, 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

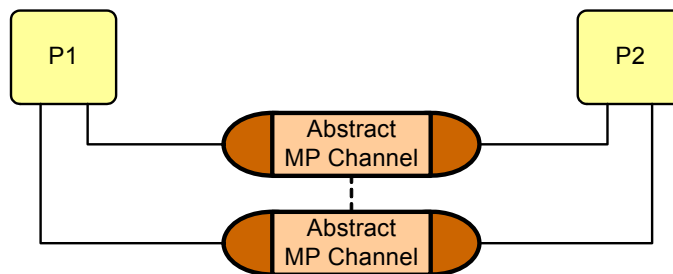
- **A model, not an implementation !**

## From Layers to System Models...

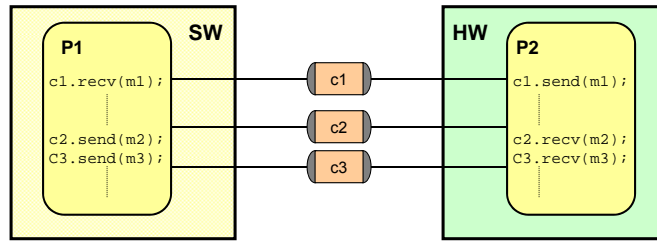


## Specification Model

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



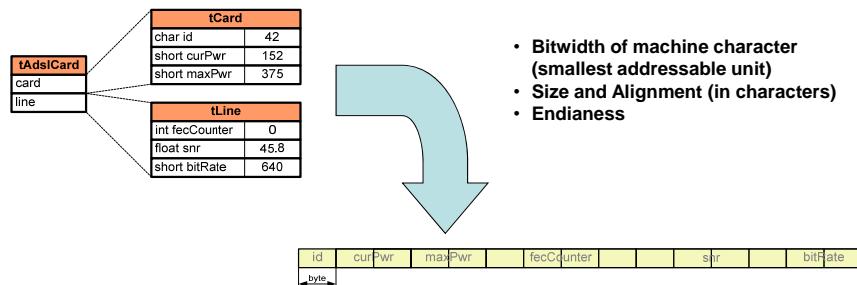
## Computation Model



- **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**

## Presentation Layer

- **Data formatting**
  - Translate abstract data types into canonical network byte layout
    1. Global network data layout
    2. Shared, optimized layout for each pair of communicating PEs
  - Convert typed messages into untyped, ordered byte streams
  - Convert variables into memory byte layout

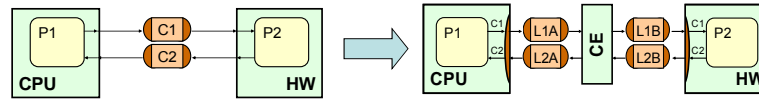


- Bitwidth of machine character (smallest addressable unit)
- Size and Alignment (in characters)
- Endianness

## Network Layer

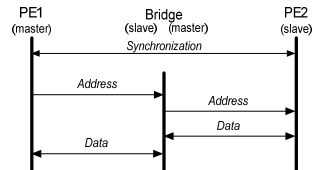
- **Split network into subnets**

- Routing of end-to-end paths over point-to-point links
- Insert communication elements (CEs) to connect busses



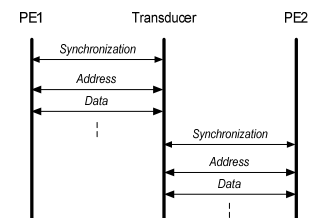
- **Bridges**

- Transparently connect slave & master sides at protocol level
- Bridges maintain synchronicity, no buffering



- **Transducers**

- Store-and-forwarding of data packets between incompatible busses
- Intermediate buffering, results in asynchronous communication



## Transport Layer

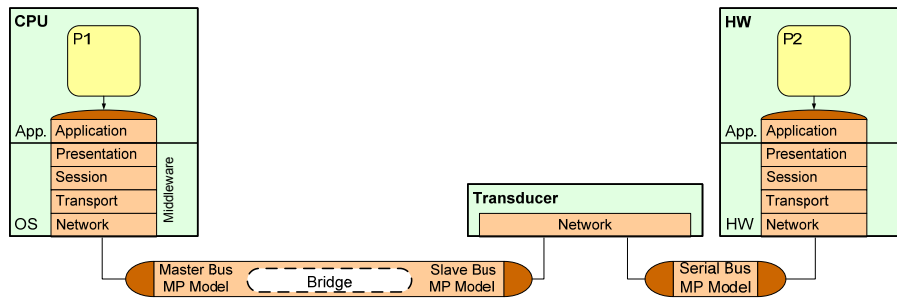
- **Packeting and routing**

- Packetization to reduce buffer sizes
  1. Fixed packet sizes (plus padding)
  2. Variable packet size (plus length header)
- Protocol exchanges (ack) to restore synchronicity
  - Iff synchronous message passing and transducer in the path
- Packet switching and identification (logical routing)
  1. Dedicated logical links (defer identification to lower layers)
  2. Network endpoint addressing (plus packet address headers)
- Physical routing in case of multiple paths between PEs
  1. Static, predetermined routing (based on connectivity/headers)
  2. Dynamic (runtime) routing

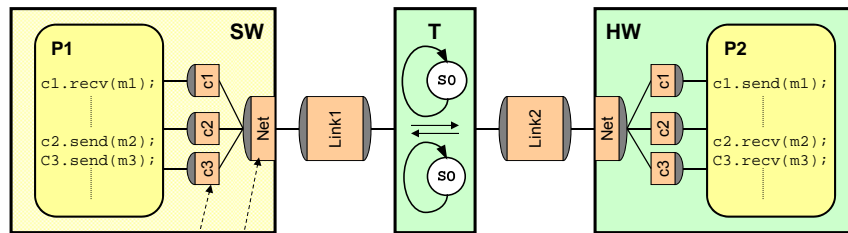


## Network Model

- **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



### Presentation, session:

```
send(type msg) {
    char buf[M];
    1: msg->buf;
    2: net.send(buf);
}
```

*Data conversion*

### Network, transport:

```
send(void* msg, len) {
    for (pkt in msg):
        s1 link.send(pkt);
        s2 link.recv(ack);
}
```

*Packeting, acknowledgement, routing*

### • Network layers

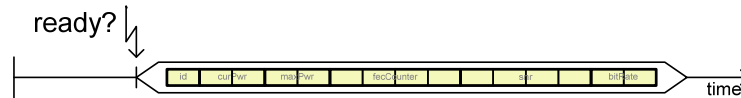
- PEs + Memories + CEs
  - Transducers (store-and-forward)
- Point-to-point link communication
  - Synchronous packet transfers (data link channels)
  - Memory accesses (shared memory, memory-mapped I/O)
  - Events (control flow)

### ➤ Communication topology

## Link Layer

### • Synchronization

- Ensure slave is ready before master initiates transaction
  1. Always ready slaves (memories and memory-mapped I/O)
  2. Defer to fully synchronized bus protocol (e.g. RS232)
  3. Separate synchronization mechanism



- Events from slave to master for master/slave busses
  - Dedicated or shared interrupts, polling
- Synchronization packets for node-based busses

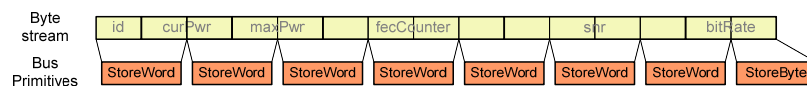
## Media Access (MAC) Layer

### • Addressing

- Multiplexing of links over shared medium
  - Dedicated physical bus addresses or packet headers

### • Data slicing

- Split data packets into multiple bus transactions
  - Optimized slicing utilizing bus modes (e.g. burst)

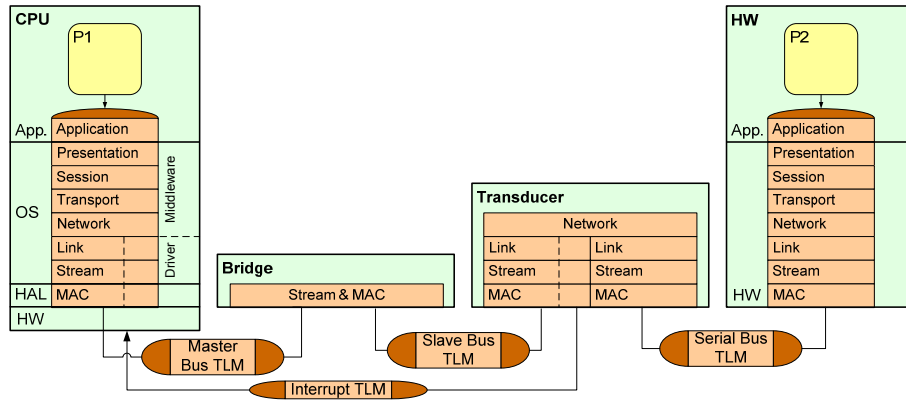


### • Arbitration

- Separate individual bus transactions in time
  - Distributed or centralized (using additional arbiter components)

## Protocol 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)

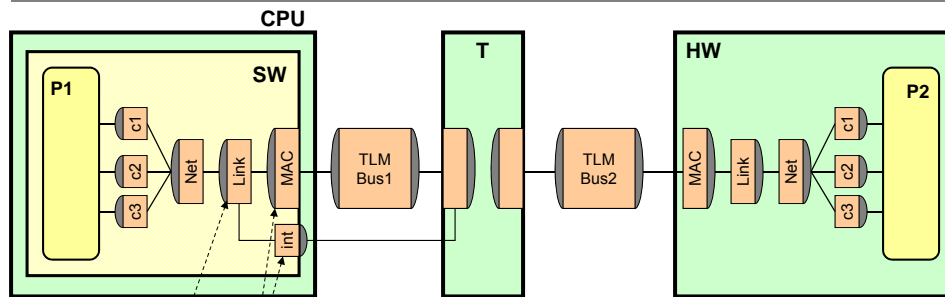


EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

© 2019 A. Gerstlauer

53

## Protocol TLM



```

Link, stream:
send(void* p, len) {
    s1 wait(int);
    s2 mac.write(p, addr);
}

Media access:
intHandler() {
    notify(int);
}
write(void* d, len, a) {
    s0 bus.writeWord(a,w);
}
    
```

*Synchronization, addressing      Data slicing*

### • Link layers

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

### ➤ System communication architecture

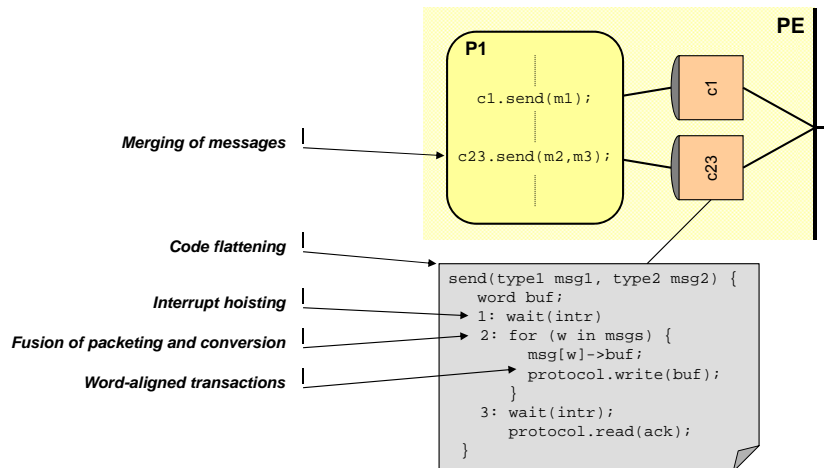
EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

© 2019 A. Gerstlauer

54

## Protocol Stack Optimizations

- Apply automatic code optimizations during refinement
  - Code optimized for HW/SW synthesis
  - Layer merging and cross-optimizations (inline/interleave)



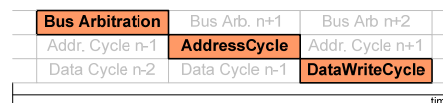
EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

© 2019 A. Gerstlauer

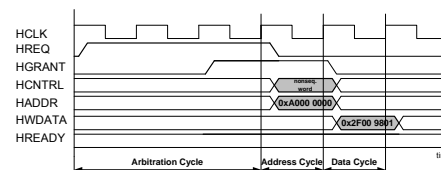
55

## Protocol, Physical Layers

- **Bus interface**
  - Generate state machines implementing bus protocols
  - Timing-accurate based on timing diagrams and timing constraints



- Bus protocol database



- **Port mapping and bus wiring**
  - Connectivity of component ports to bus, interrupt wires/lines
  - Generate top-level system netlist

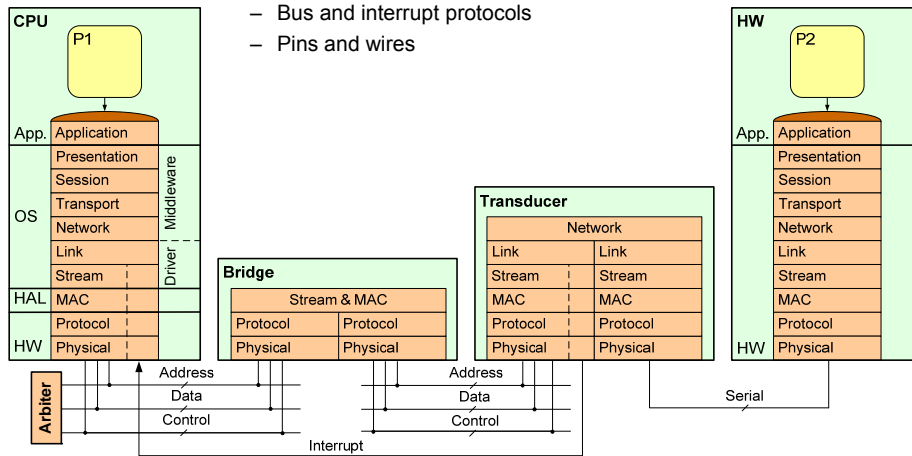
EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

© 2019 A. Gerstlauer

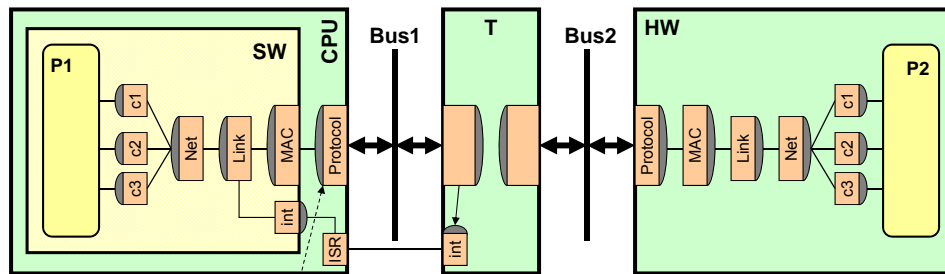
56

## Pin-/Bus Cycle-Accurate Model (P/BCAM)

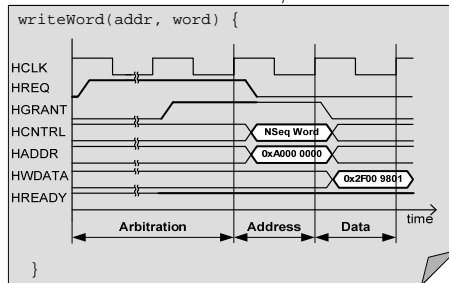
- **Component & bus structure/architecture**
  - PEs + Memories + CEs + Busses
  - Pin-accurate bus-functional components
  - Pin- and cycle-accurate communication



## Pin-Accurate Model (PAM)



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

## TLM Abstraction Levels

- **ISO/OSI reference layer-based architecture**

- Granularity of data and arbitration handling

- **Layers:**

- 1) Media Access Control (MAC)

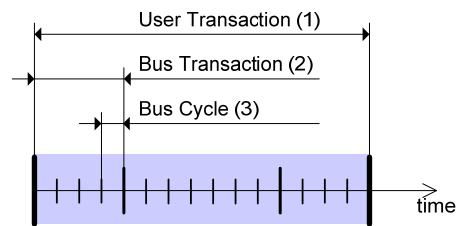
- User Transaction
  - » Contiguous block of bytes
  - » Arbitrary length, base address

- 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



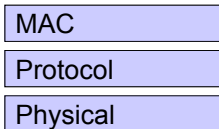
- **Models are composed of layers**

- Using fewer layers yields a more abstract model

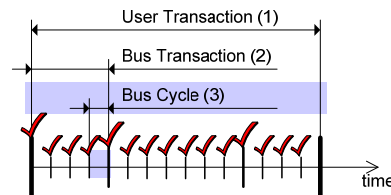
Source: G. Schirmer, R. Dömer, "Quantitative Analysis of the Speed/Accuracy Tradeoff in Transaction-Level Models," ACM TECS, 2008.

## Bus Functional Model (BFM)

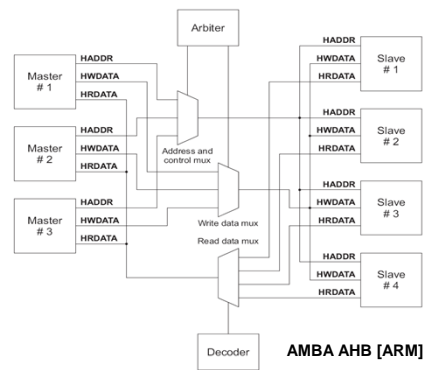
Implemented Layers:



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

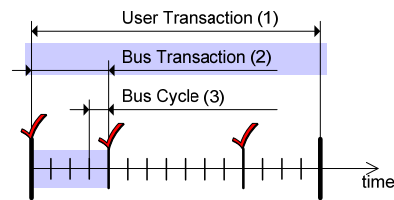


## Transaction Level Model (TLM)

Implemented Layers:



Granularity:



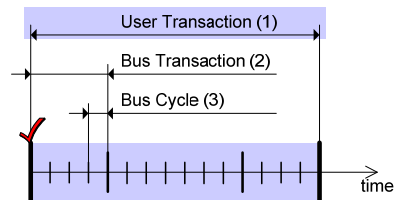
- **Bus primitives**
  - StoreWord, StoreBurst4
- **Abstract model**
  - Not pin accurate, not bus cycle accurate in all cases
- **Priority arbitration per bus transaction**
  - May lead to wrong arbitration decision, depending on execution order

## MAC-TLM

Implemented Layers:



Granularity:



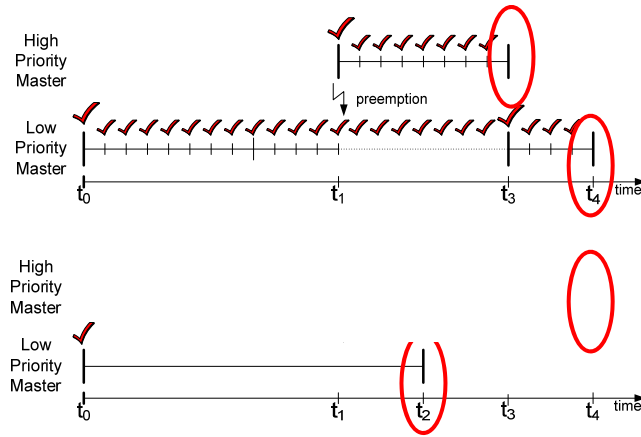
- **User transaction (message)**
  - Arbitrary length, contiguous block of bytes
- **No arbitration: contention avoidance by semaphore**
  - Resolution depends on simulator
- **Expected to be the fastest model**
  - Single `memcpy`, Single `time wait`

## TLM Accuracy Limitations

- **Example:**
  - Low priority burst starting at  $t_0$
  - High priority preemption at  $t_1$

- **BFM:**
  - check every cycle
  - slow
  - accurate

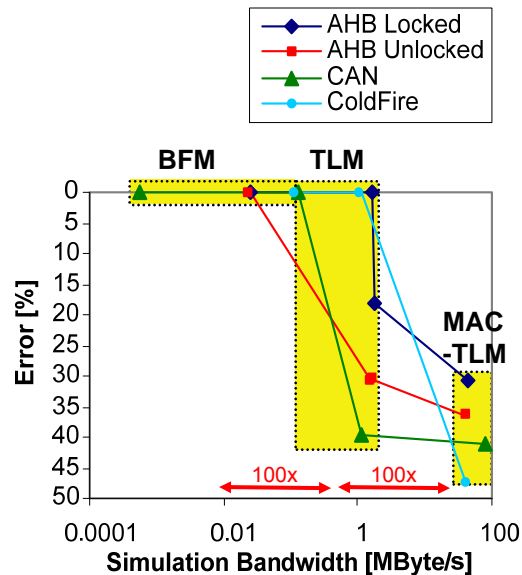
- **TLM:**
  - coarse grain check
  - fast
  - inaccurate: low prio. burst ends at  $t_2$  instead of  $t_4$



## 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%

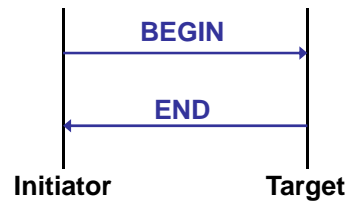




## SystemC/TLM

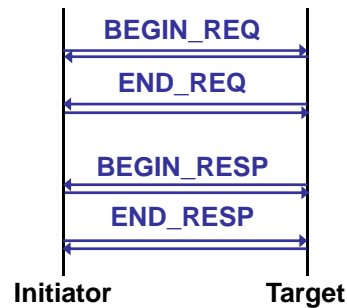
- **Loosely-timed**

- 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)

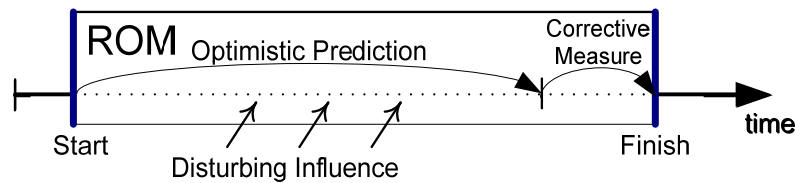
- **Advanced TLM techniques (see OS modeling)**

- Conservative [ATGA]
- Optimistic [ROM]

## Result-Oriented Modeling (ROM)

### • Characteristics

- Observability at boundary of transaction
  - Internal state changes not visible, may not be modeled
- *Optimistically predict the end result* at beginning
  - optimistic == earliest time to finish
- Record *disturbing influence*
- *Corrective measures* at the end



Source: G. Schirner, R. Dömer, "Fast and Accurate Transaction Level Models using Result Oriented Modeling," ICCAD, 2006.

EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

© 2019 A. Gerstlauer

67

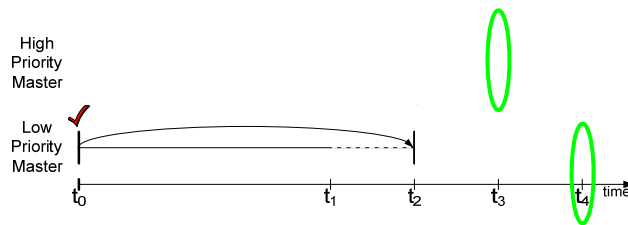
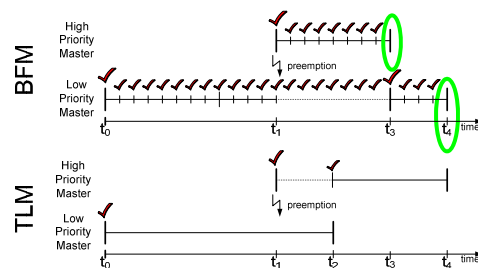
## Result Oriented Modeling (ROM)

### • Same example transfer:

- Low priority burst at  $t_0$
- High priority preemption at  $t_1$

### • ROM:

- $t_0$  low: predict  $t_2$
- $t_1$  high: preempt, predict  $t_3$ , record preemption for low
- $t_2$  low: detect disturbance, prediction update  $t_4$
- $t_3$  high: no preemptions, finish
- $t_4$  low: no preemptions, finish
- **Accurate**



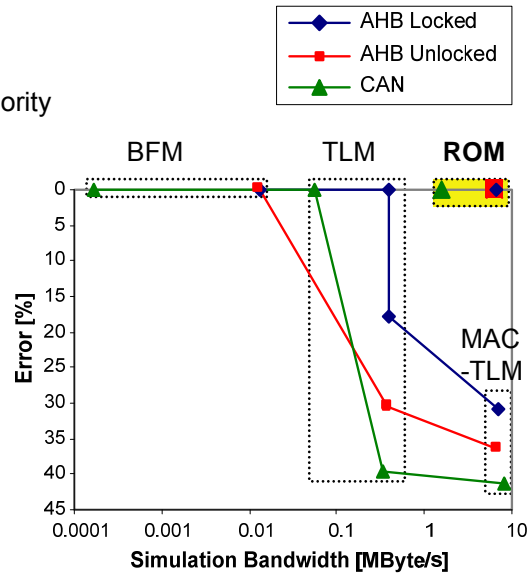
EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

© 2019 A. Gerstlauer

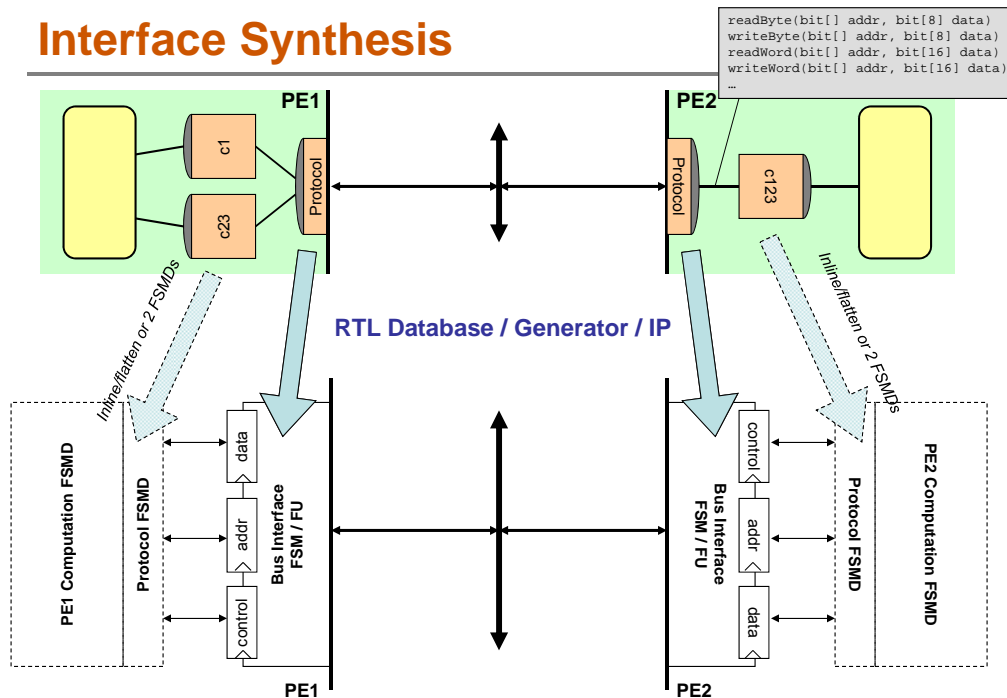
68

## ROM Results

- Two concurrent masters
  - High priority: 33% bus load
  - Measure bandwidth of low priority
- 100% accuracy
  - ROM, BFM both on top
  - No error
- At coarse-grain speeds
  - ROM and MAC-TLM are in same magnitude!
  - CAN ROM can not outperform MAC-TLM
    - Data dependent length
    - Bit inspection
      - CRC, stuffing bits

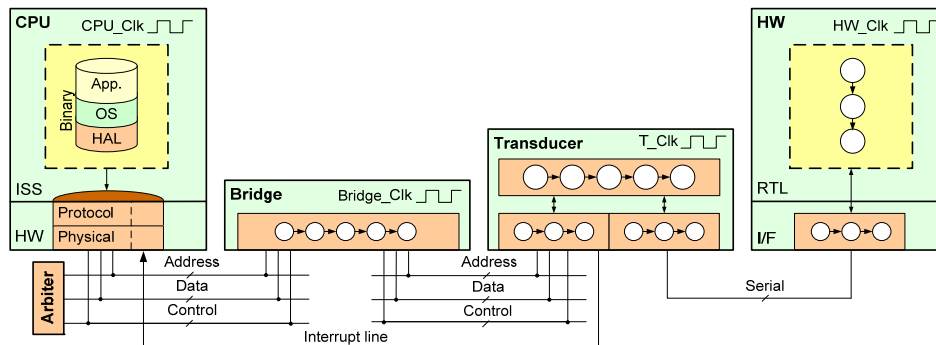


## Interface Synthesis



## Cycle-Accurate Model (CAM)

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



EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

© 2019 A. Gerstlauer

71

## Lecture 8: 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

EE382N.23: Embedded Sys Dsgn/Modeling, Lecture 8

© 2019 A. Gerstlauer

72