System-Level Design (and Modeling for Embedded Systems)

Lecture 7 – Computation Modeling & Refinement

Kim Grüttner kim.gruettner@dlr.de
Henning Schlender henning.schlender@dlr.de
Jörg Walter joerg.walter@offis.de

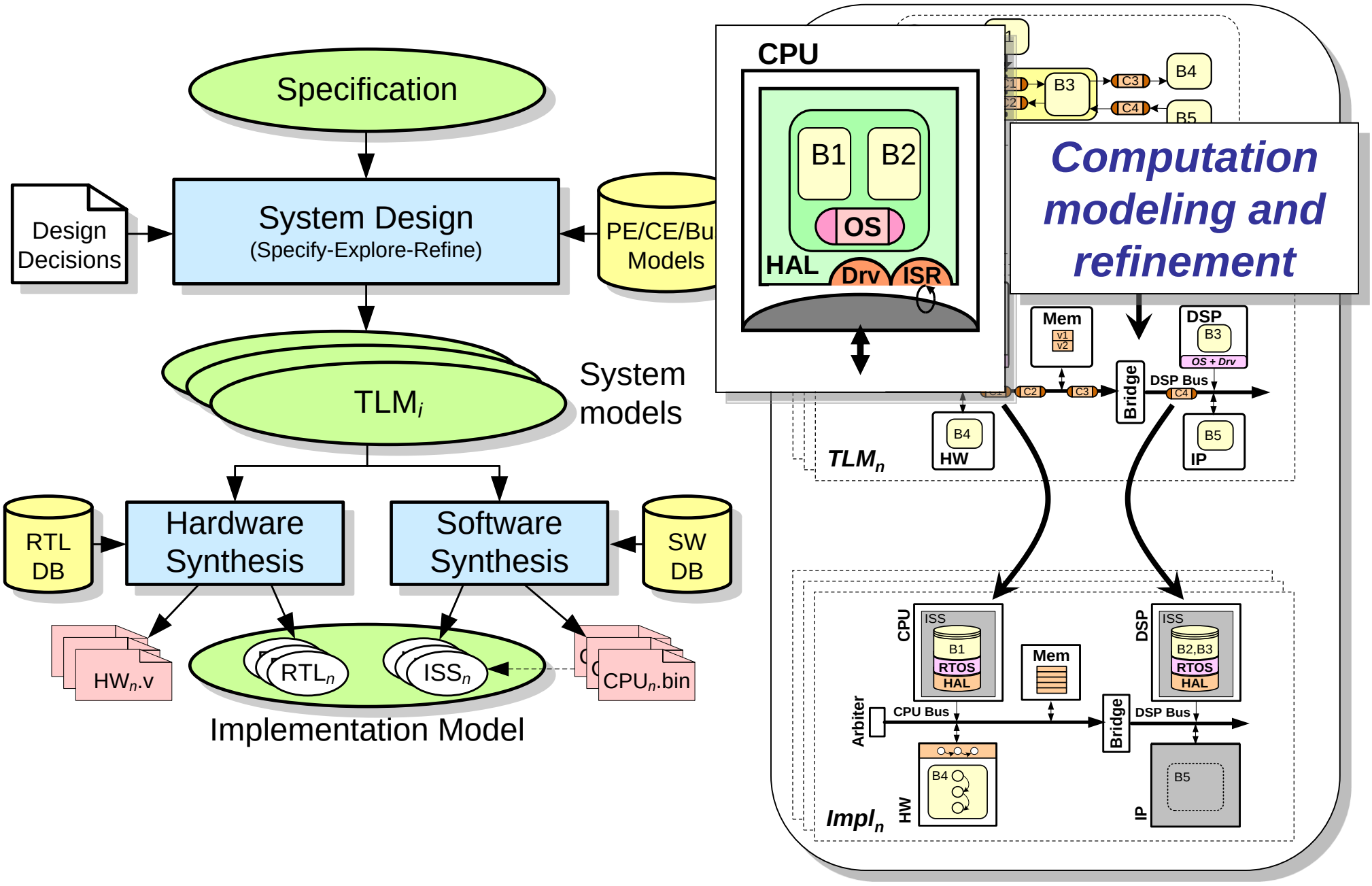
System Evolution and Operation
German Aerospace Center (DLR)
&
Distributed Computation and Communication
OFFIS

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

Lecture 7: Outline

- **Processor layers**
  - Application
  - Task/OS
  - Firmware
  - Hardware
- **Processor synthesis**
  - Software synthesis

System-On-Chip Environment (SCE)



- **Growing system complexities and sizes**
  - Heterogeneous multi-processor systems (MPSoC)
- **Increasing significance of embedded software**
  - Growing software content

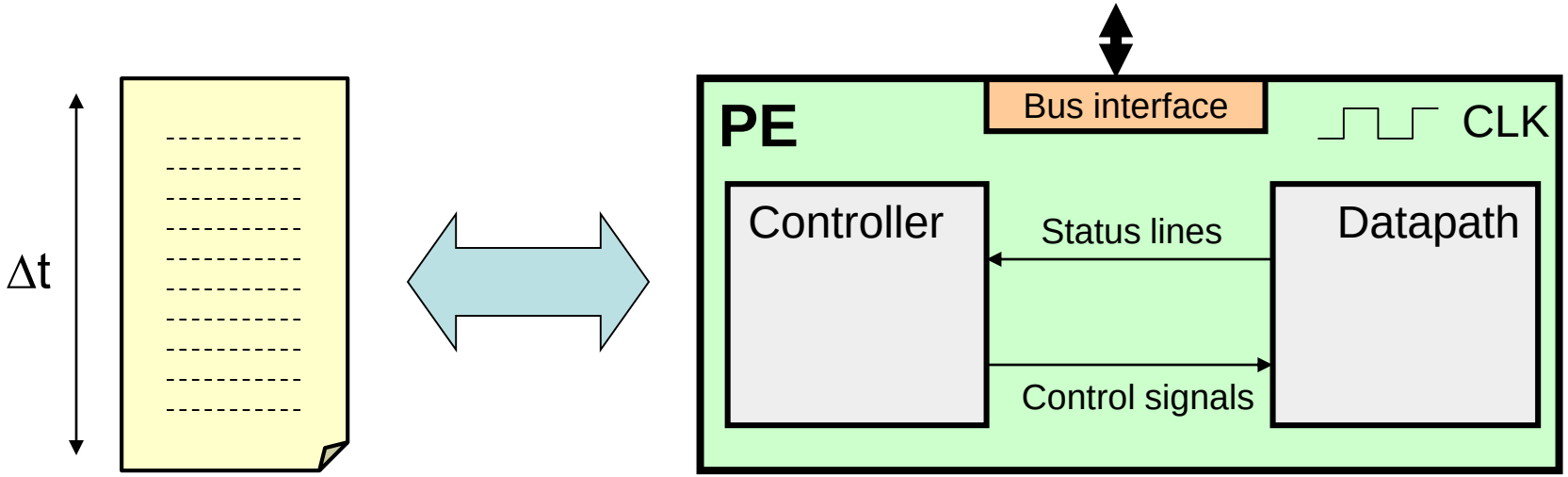
➢ **System design at higher levels of abstraction**
  - Validation and analysis
  - Concurrent hardware and software development
  - Implementation synthesis

➢ **Design of embedded software and processors**
  - Large influence on system performance, power, etc.
  - Actual SW on ISS is accurate but slow

  ➢ High-level models for early and accurate feedback

  ➢ Software synthesis

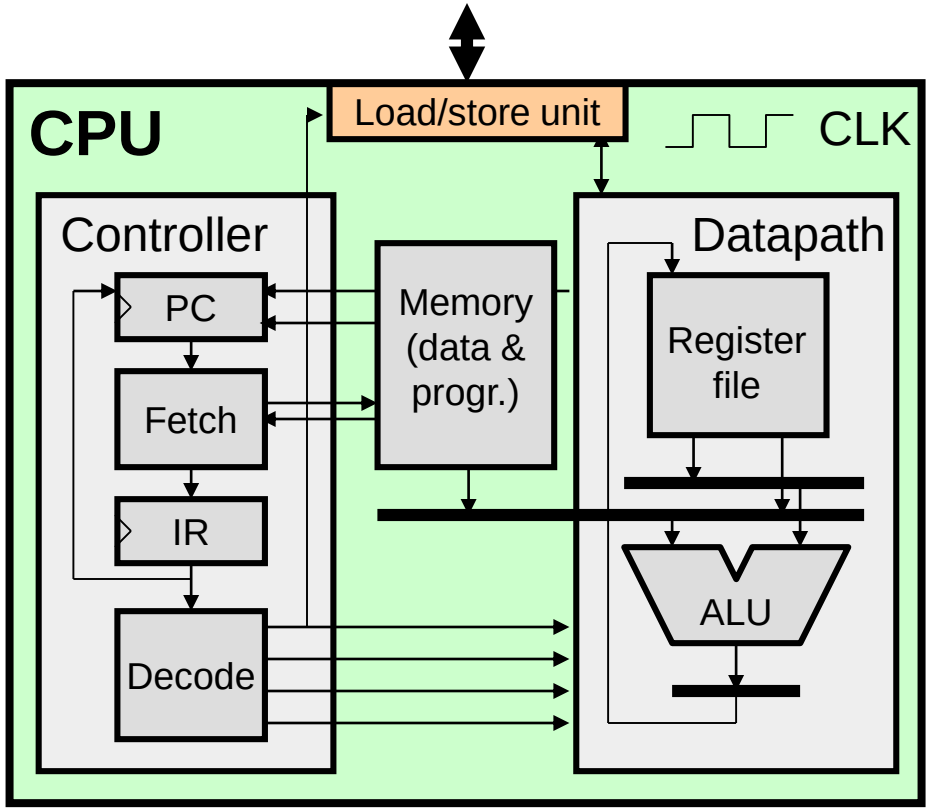
- **Basic system component is a *processor (PE)***
  - Programmable, general-purpose software processor (CPU)
  - Programmable special-purpose processor (e.g. DSPs)
  - Application-specific instruction set processor (ASIP)
  - Custom hardware processor



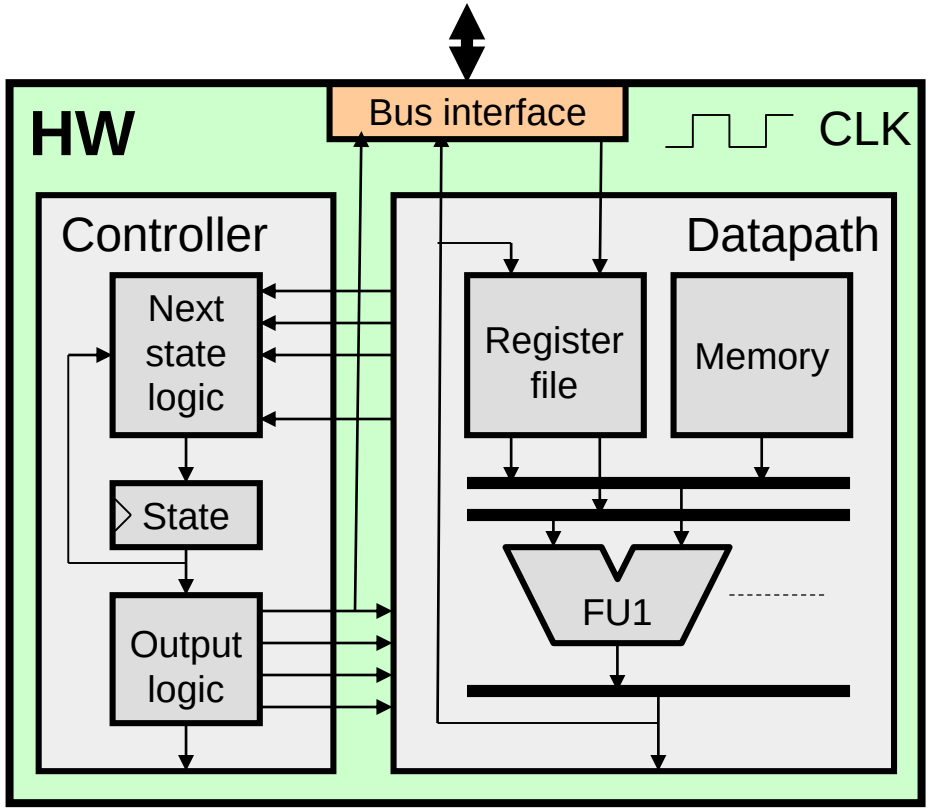
➢ **Functionality *and* timing**

Processor Models (1)

- **Structural RTL models**



Software processor

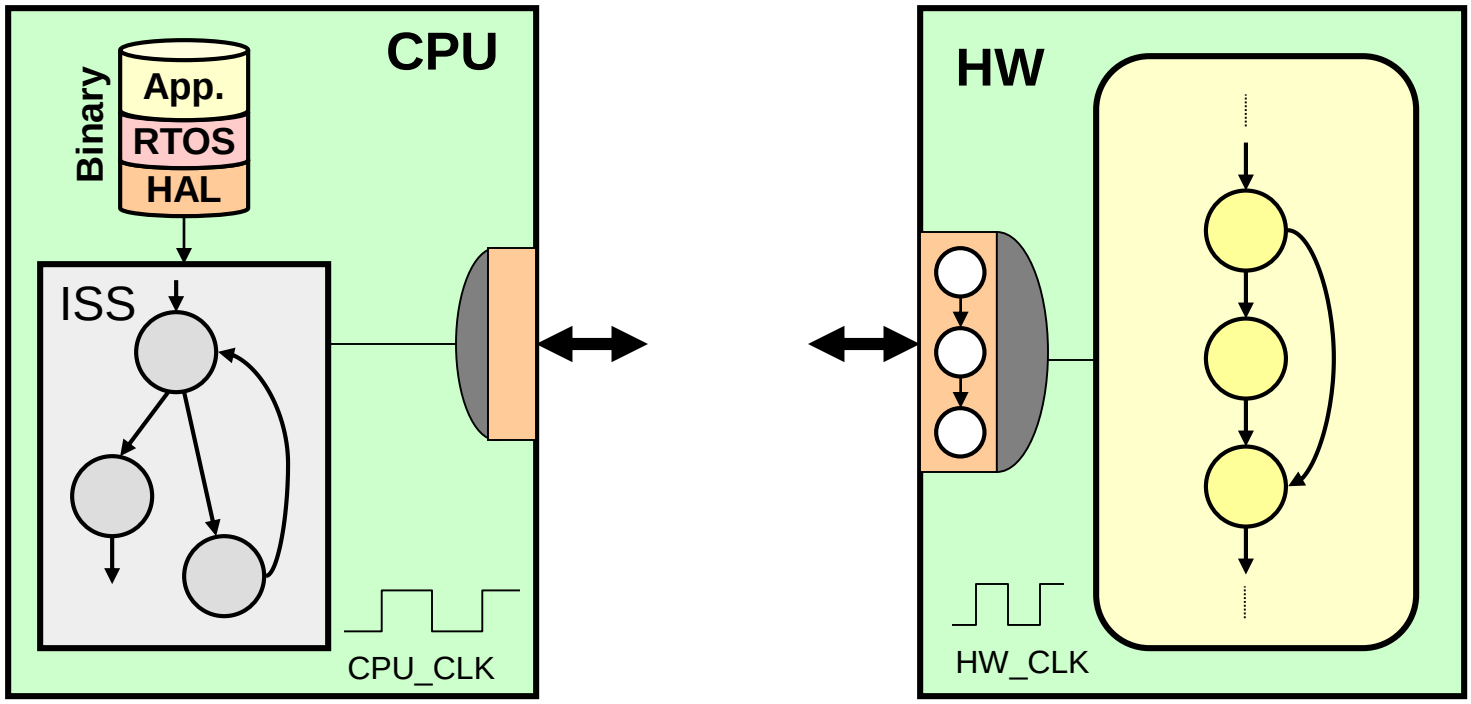


Hardware processor

- **Sub-cycle accurate**

Processor Models (2)

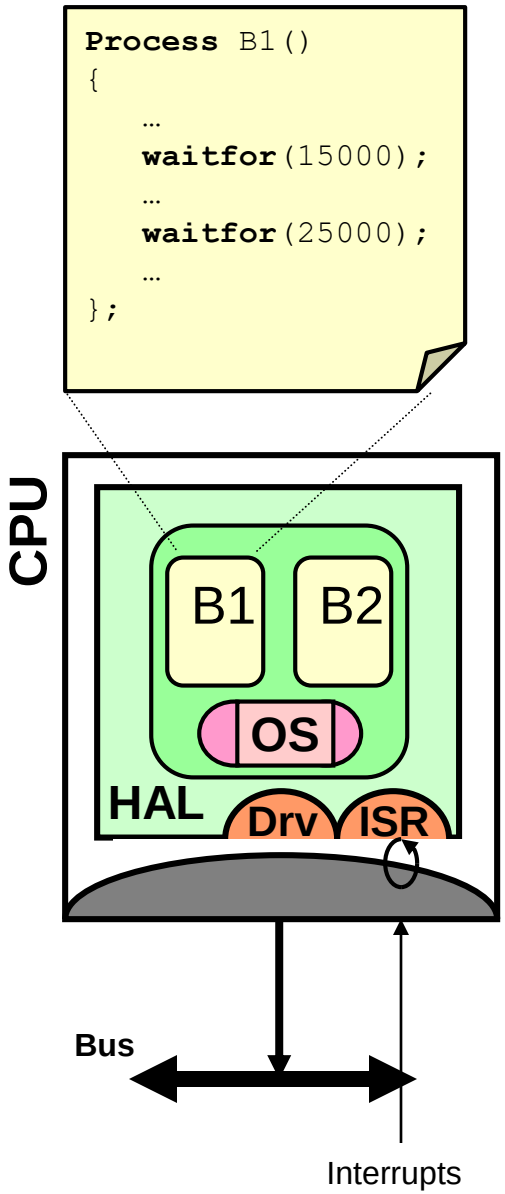
- **Behavioral RTL/IS models**



Instruction set simulation (ISS) FSMD

- **Cycle accurate**

High-Level Computation Modeling

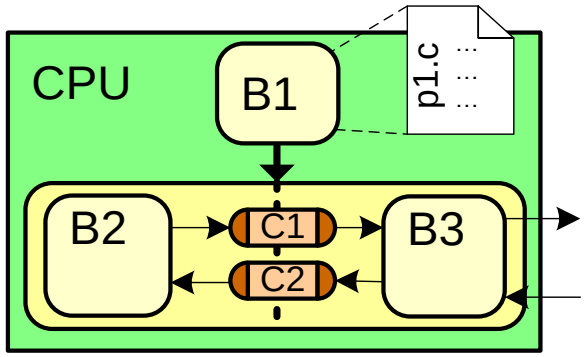


- **Application modeling**
  - Native process execution (C code)
  - Back-annotated execution timing
- **Processor modeling**
  - Operating system
    - Real-time multi-tasking (RTOS model)
    - Bus drivers (C code)
  - Hardware abstraction layer (HAL)
    - Interrupt handlers
    - Media accesses
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

Source: G. Schirner, A. Gerstlauer, R. Doemer. "Abstract, Multifaceted Modeling of Embedded Processors for System Level Design," ASPDAC07

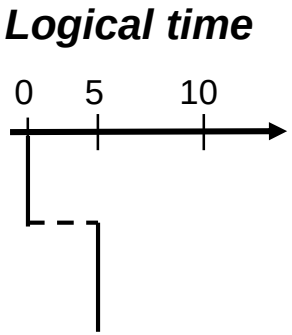
Processor Model: Application Layer

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



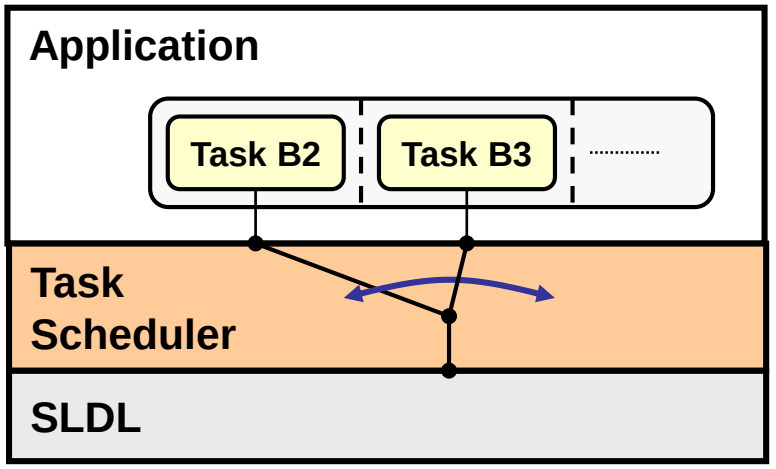
- **Timed simulation of application functionality (SLDL)**
  - Back-annotate timing
    - Estimation or measurement (trace, ISS)
    - Function or basic block level granularity
  - Execute natively on simulation host
    - Discrete event simulator
    - Fast, native compiled simulation

```
...
void f() {
  waitfor(5);
  ...
}
...
```

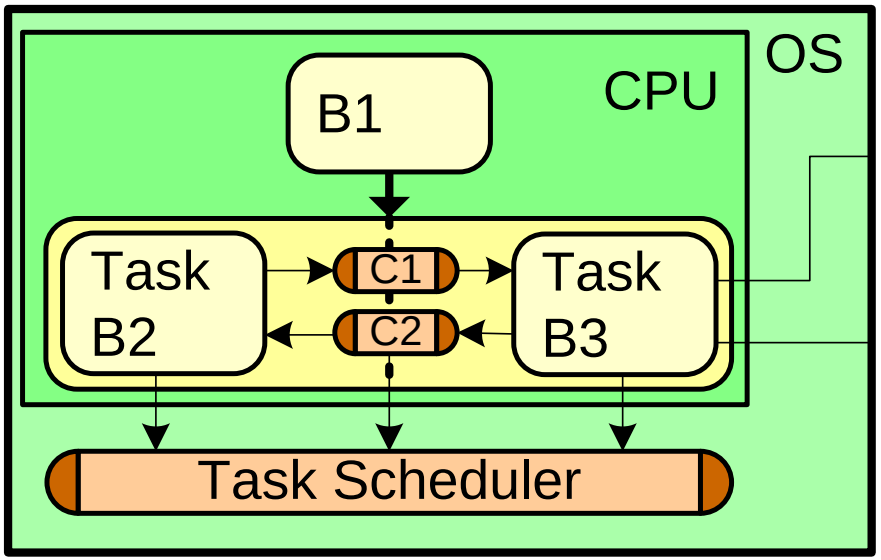


Processor Model: Task Layer

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

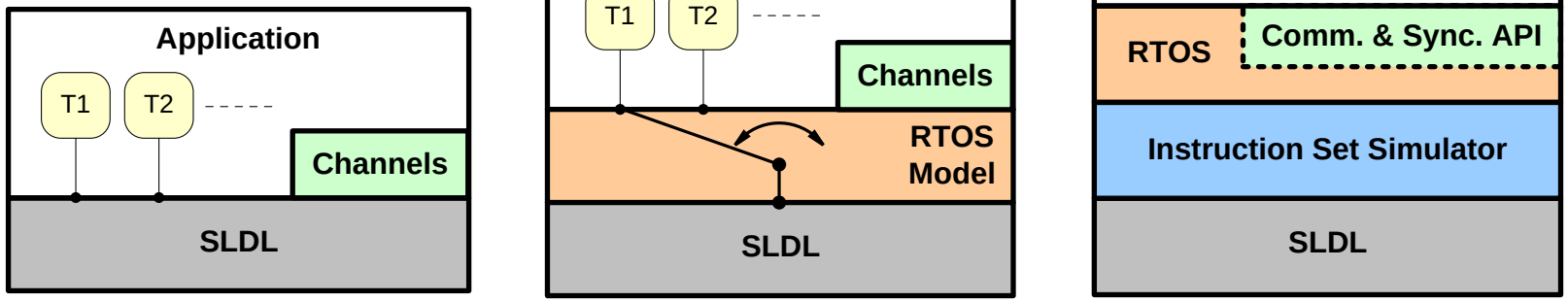


- **OS model on top of standard SLDL**
  - Wrap around SLDL primitives, replace event handling
    - Block all but active task
    - Select and dispatch tasks
  - Target-independent, canonical API
    - Task management
    - Channel communication
    - Timing and all events



OS Modeling

- **High-level RTOS abstraction**



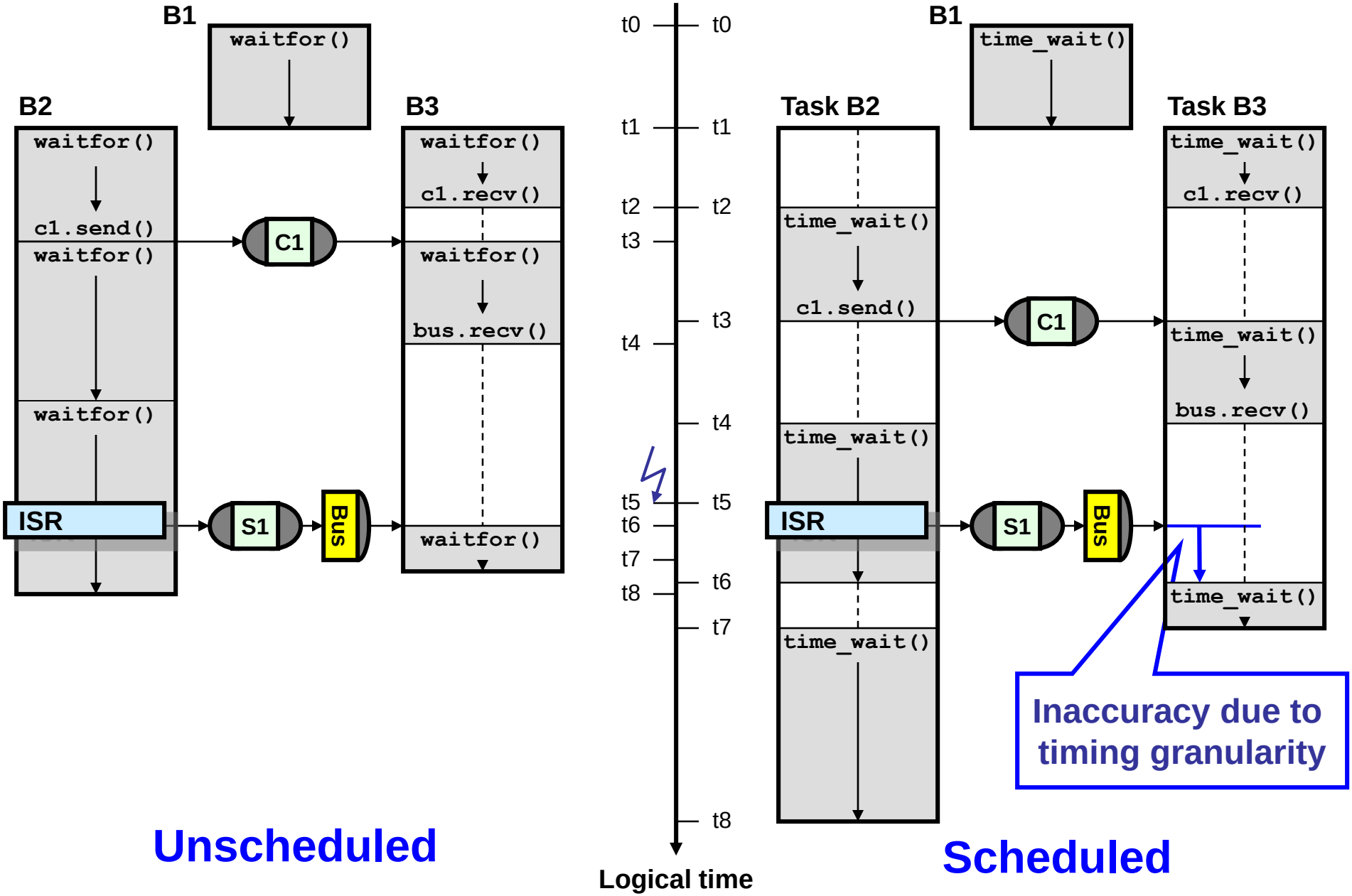
Specification *TLM* Implementation

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

➢ Model of operating system
  - ➢ 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.

Simulated Dynamic Behavior



RTOS Model Implementation

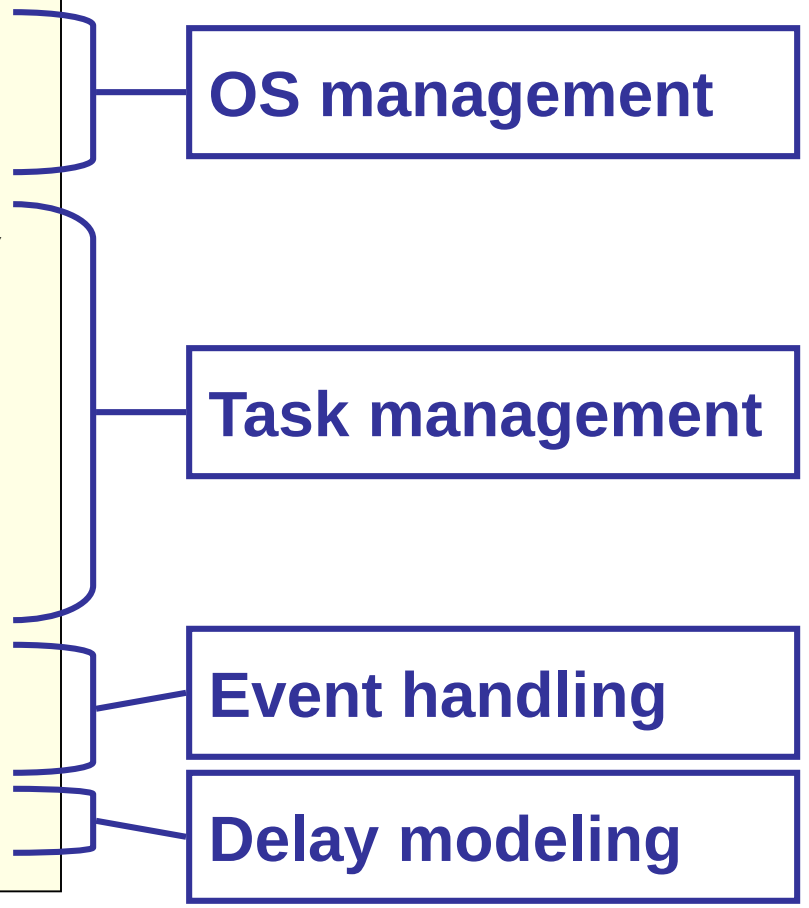
- **RTOS model**
  - OS, task, event management
    - Descriptors & queues
  - Scheduling
    - Select and dispatch task based on algorithm
    - Block all but active task on SLDL level
  - Preemption
    - Allow rescheduling at simulation time increases
  - Event handling
    - Remove task temporarily 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  channel OS implements OSAPI {
     Task current = 0;
     os_queue rdyq;

5    void dispatch(void) {
       current = schedule();
       notify(current.event);
     }
     void yield() {
10     task = current;
       dispatch();
       wait(task.event);
     }

15   void time_wait(time t) {
       waitfor(t);
       yield();
     }

20   Task pre_wait(void) {
       Task t = rdyq.get(current);
       dispatch(); return t;
     }
     void post_wait(Task t) {
25     rdyq.put(t);
       wait(t.event);
     }
};
```

RTOS Model Interface

- **Canonical, target-independent API**

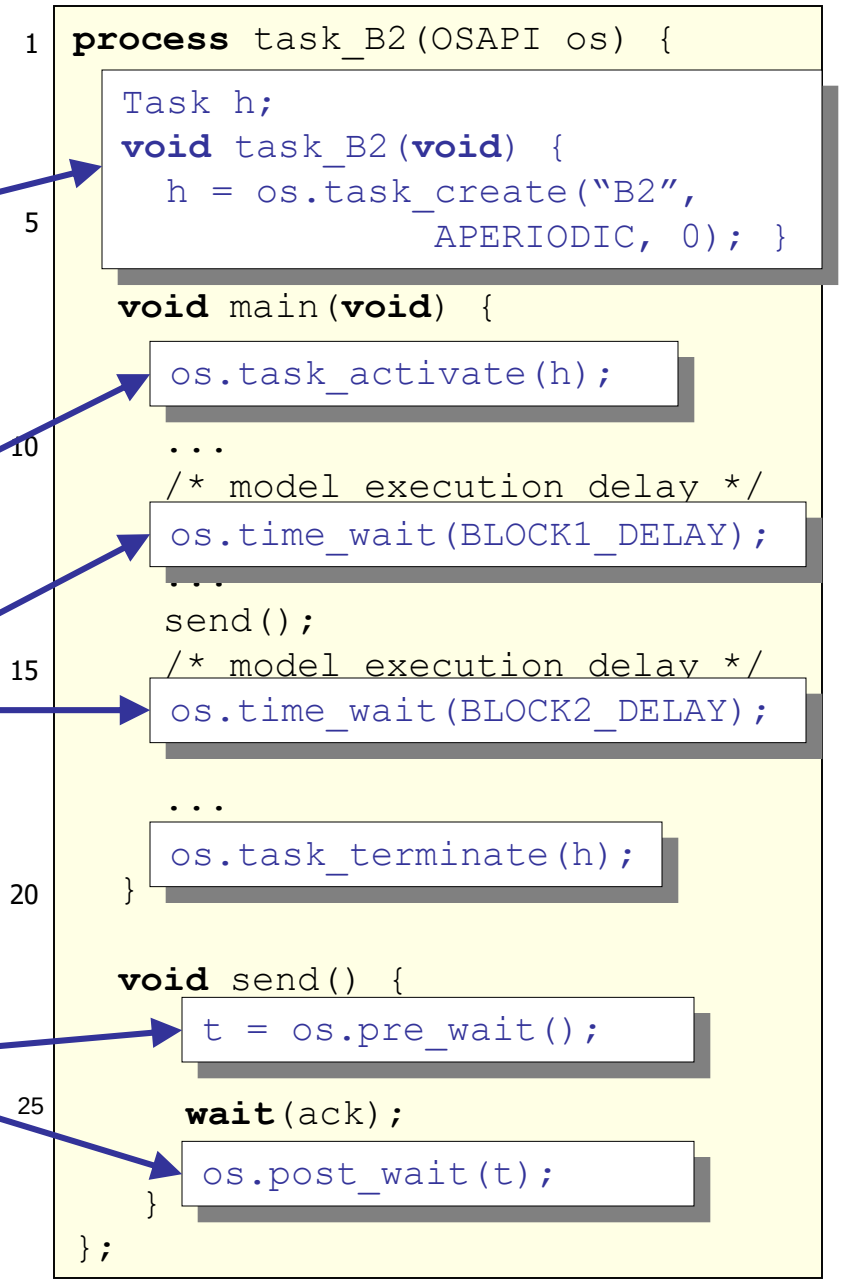
```
1  interface OSAPI
   {
       void init();
       void start(int sched_alg);
5      void interrupt_return();

       Task task_create(char *name, int type,
                        sim_time period);
       void task_terminate();
10     void task_sleep();
       void task_activate(Task t);
       void task_endcycle();
       void task_kill(Task t);
       Task par_start();
15     void par_end(Task t);

       Task pre_wait();
       void post_wait(Task t);

20     void time_wait(sim_time nsec);
   };
```



Back

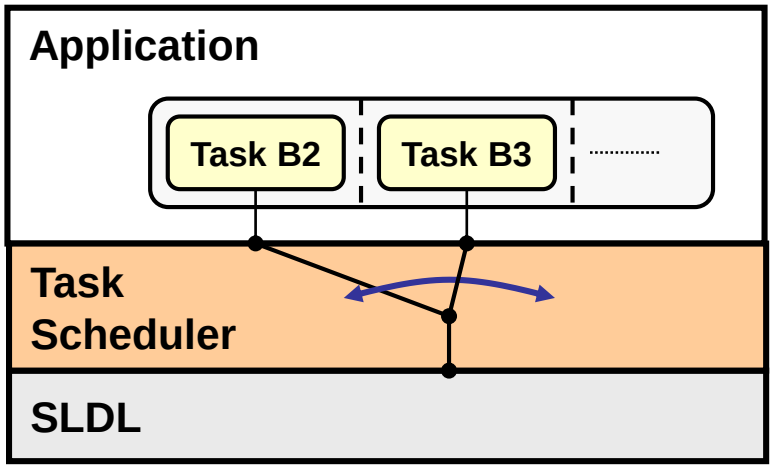
Task Refinement

- **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
  - Communication and synchronization
    - Wrap around SLDL event handling

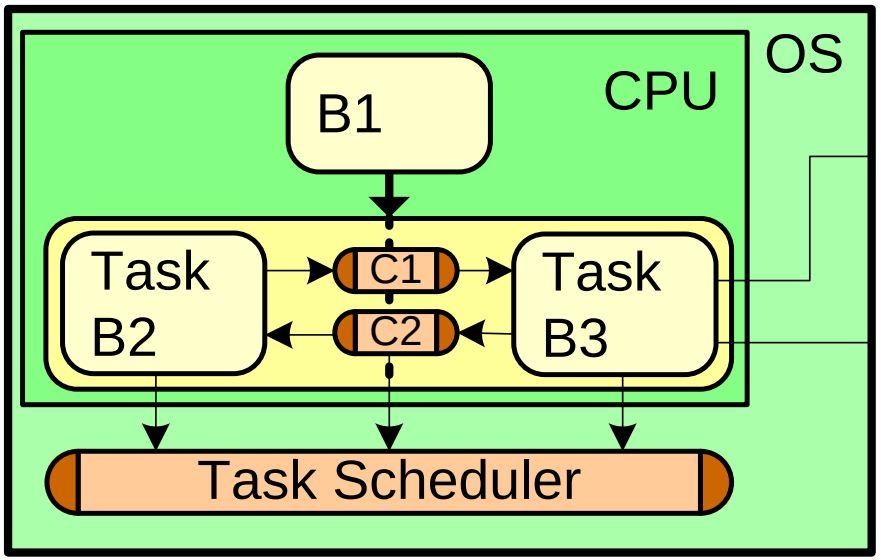


Processor Model: Task Layer

- **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



Processor Model: Firmware Layer

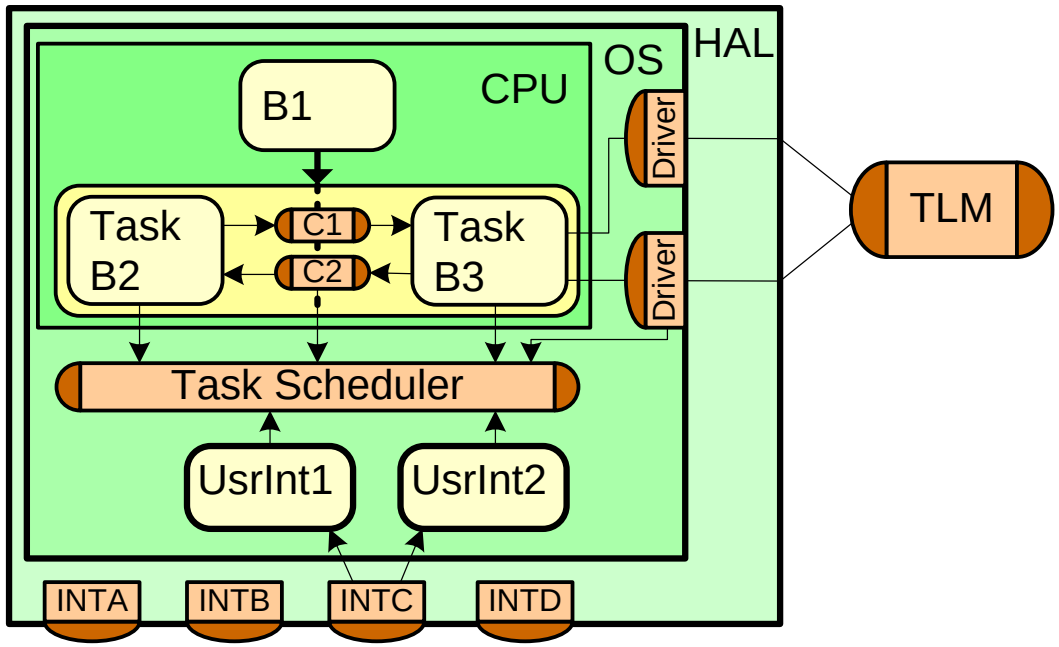
- **External communication**
  - Software Drivers
    - Presentation, Session, Packeting
    - Synchronization (e.g. Interrupts)
  - TLM Bus model
    - User transactions
  - However, interrupts are unscheduled

App.

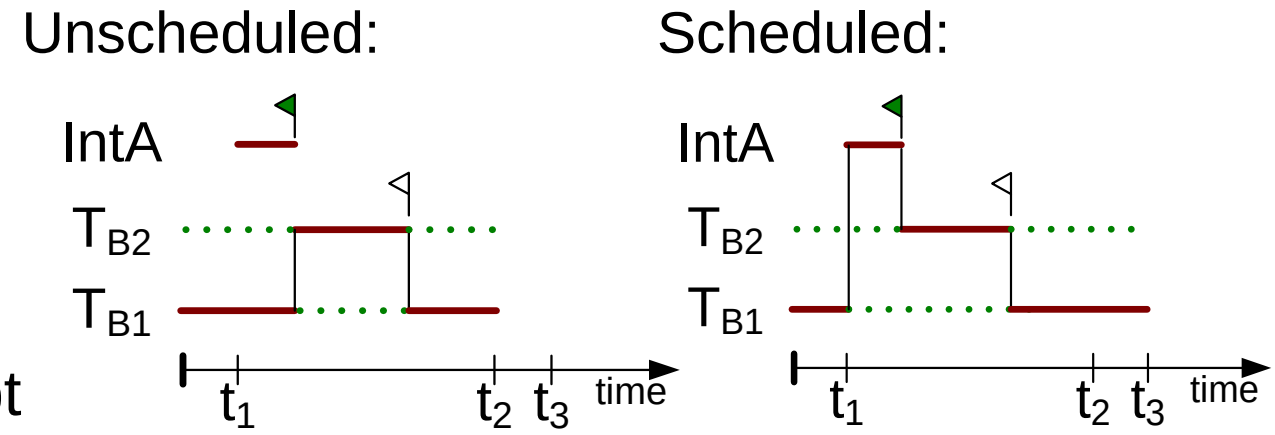
```
sample.send(v1);
```

Driver

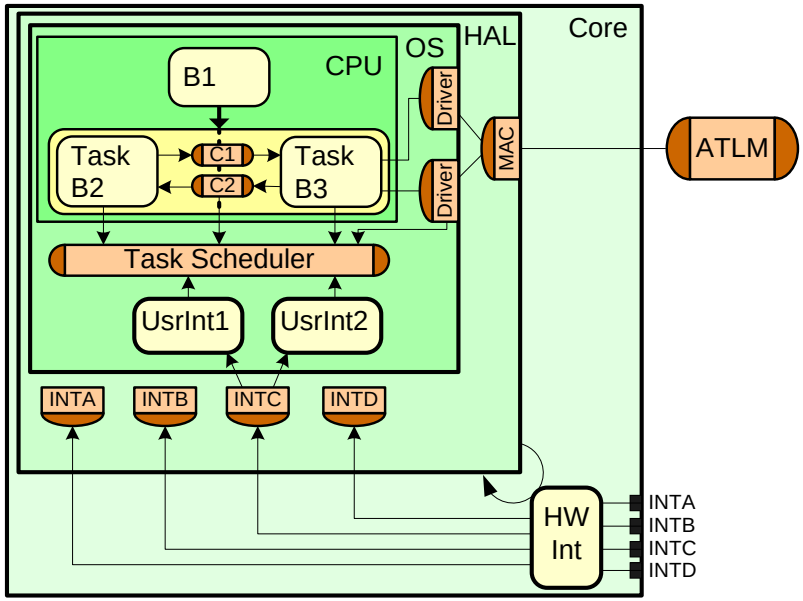
```
void send(…) {
  intr.receive();
  bus.masterWrite(0xA000,
                  &tmp,
                  len);
}
```



Processor Model: TLM Layer

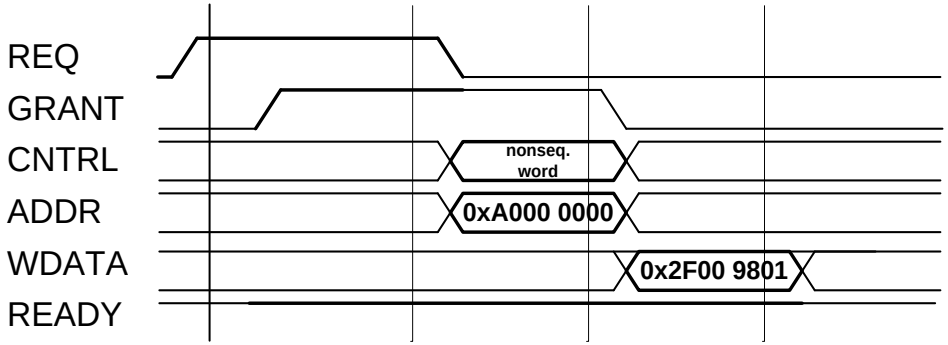


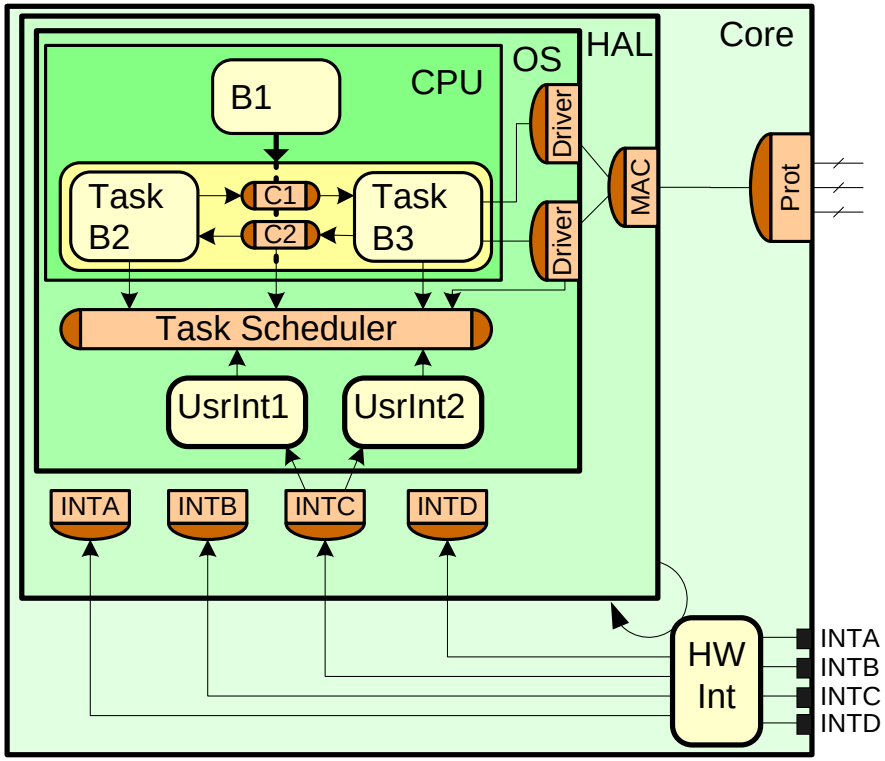
- **Processor TLM**
  - Hardware interrupt handling
    - Interrupt Scheduling
      - » Suspend user code
      - » Priority, Nesting
  - Media Access Control (MAC) for bus interface
    - Split user transaction into bus transaction
  - Arbitrated TLM bus model



Processor Model: Bus-Functional Layer

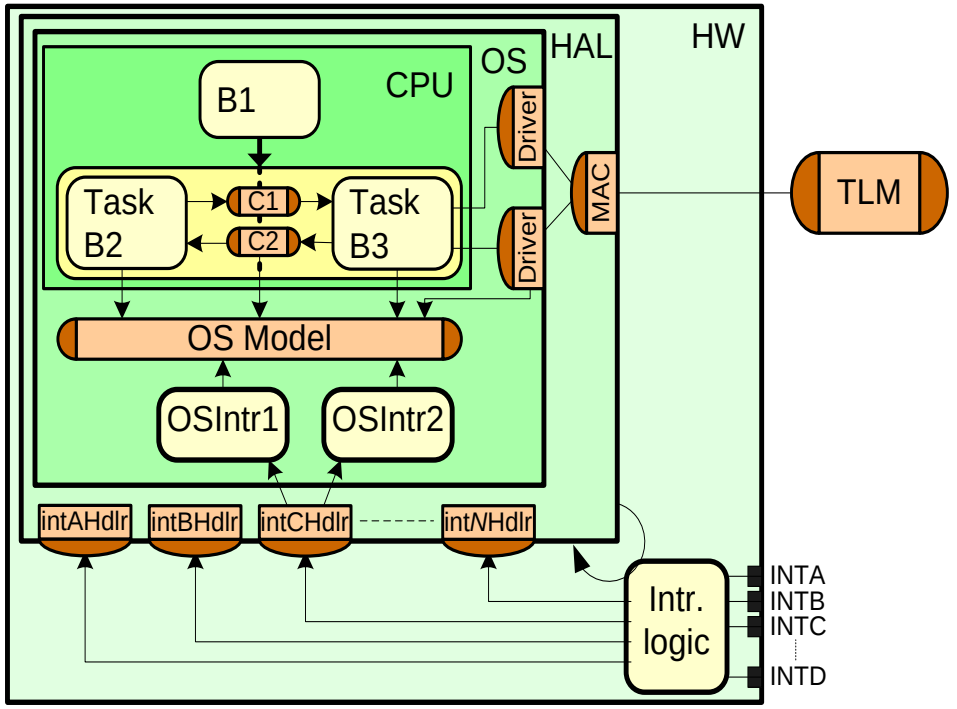
- **Processor bus-functional model (BFM)**
  - Pin-accurate model of processor
    - Cycle approximate for SW execution
  - Bus model
    - Pin-accurate
    - Cycle-Accurate





Processor Model

- **Layered model**
  - Feature levels
- **Processor layers**
  - Application
    - Native C
  - Task
    - OS model
  - Firmware
    - Middleware
  - Processor hardware
    - Bus I/F
    - Interrupts, suspension



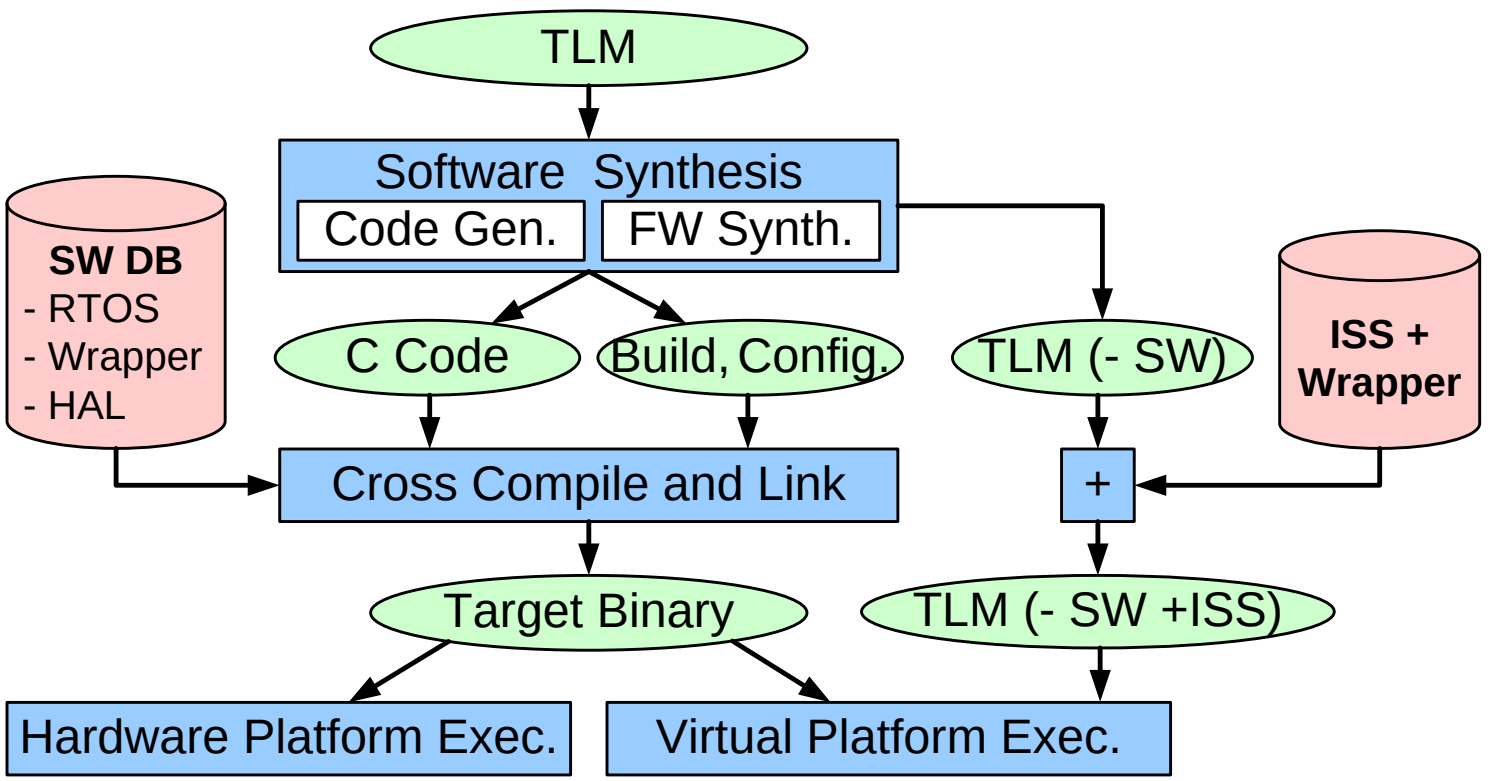
Features	
Target approx. computation timing	Appl.
Task mapping, dynamic scheduling Task communication, synchronization	Task
Interrupt handlers, low level SW drivers	Firmware
HW interrupt handling, int. scheduling	TLM
Cycle accurate communication	BFM
Cycle accurate computation	BFM - ISS

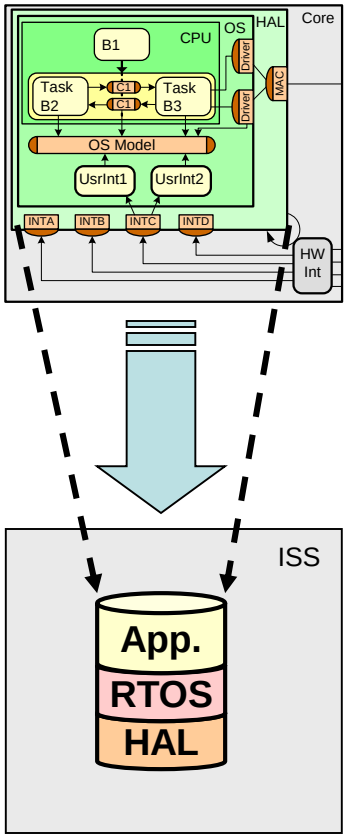
Lecture 7: Outline

- ✓ **Processor layers**
  - ✓ Application
  - ✓ Task/OS
  - ✓ Firmware
  - ✓ Hardware

- **Processor synthesis**
  - Software synthesis

Software Synthesis

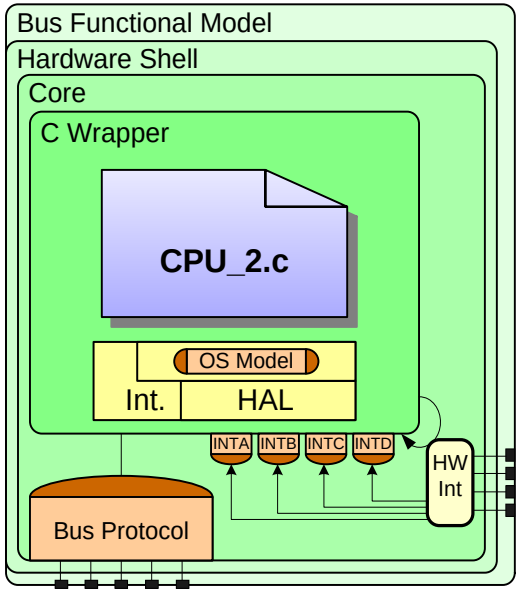




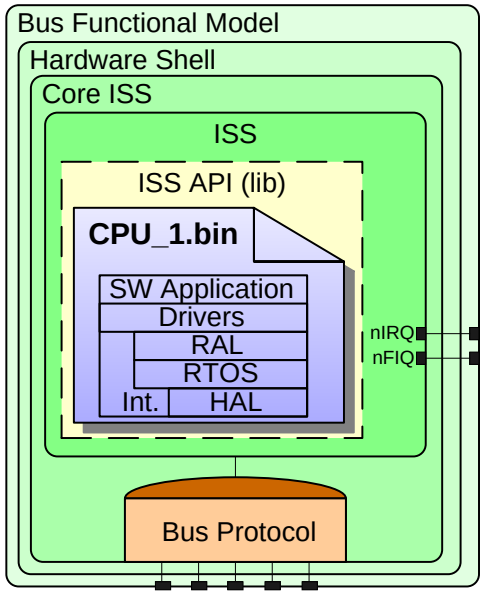
- **Automatically generate target binaries from TLM**
  - Generate code for application (tasks and IPC)
  - Synthesize firmware (drivers, interrupt handlers)
  - OS wrappers and HAL implementations from DB
  - Compile and link against target RTOS and libraries

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

Processor Implementation Models



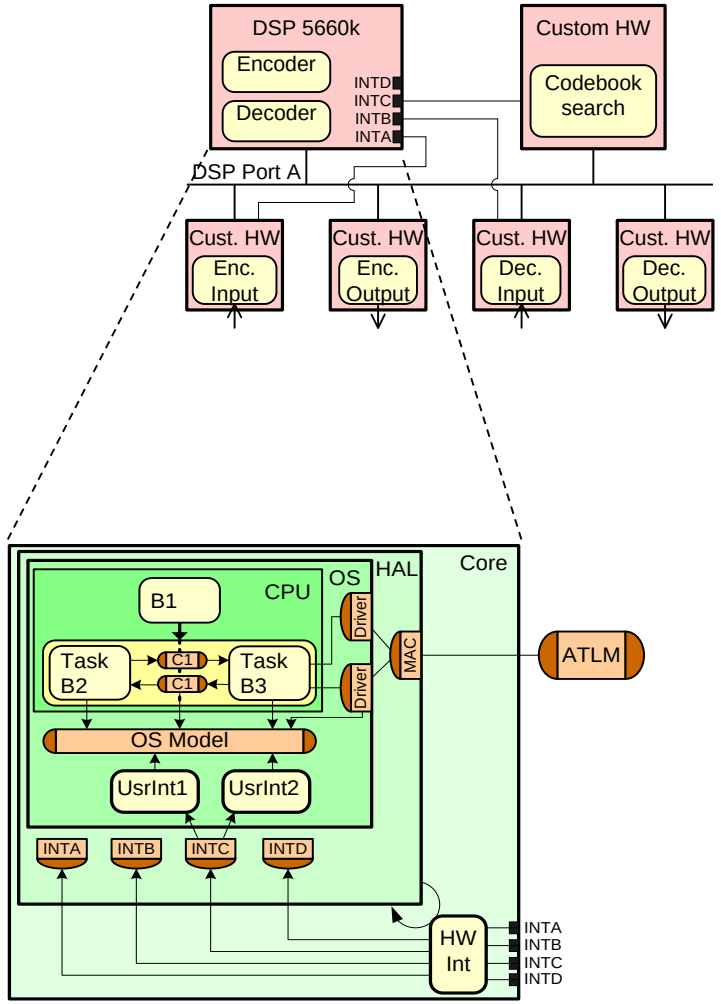
- **Software C model**
  - Generated application C code
    - Flat standard ANSI C code
  - Firmware and hardware models
    - RTOS model, HAL model
    - Low-level & hardware interrupt handling
    - External bus communication protocol/TLM



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

Single-Processor Experiments

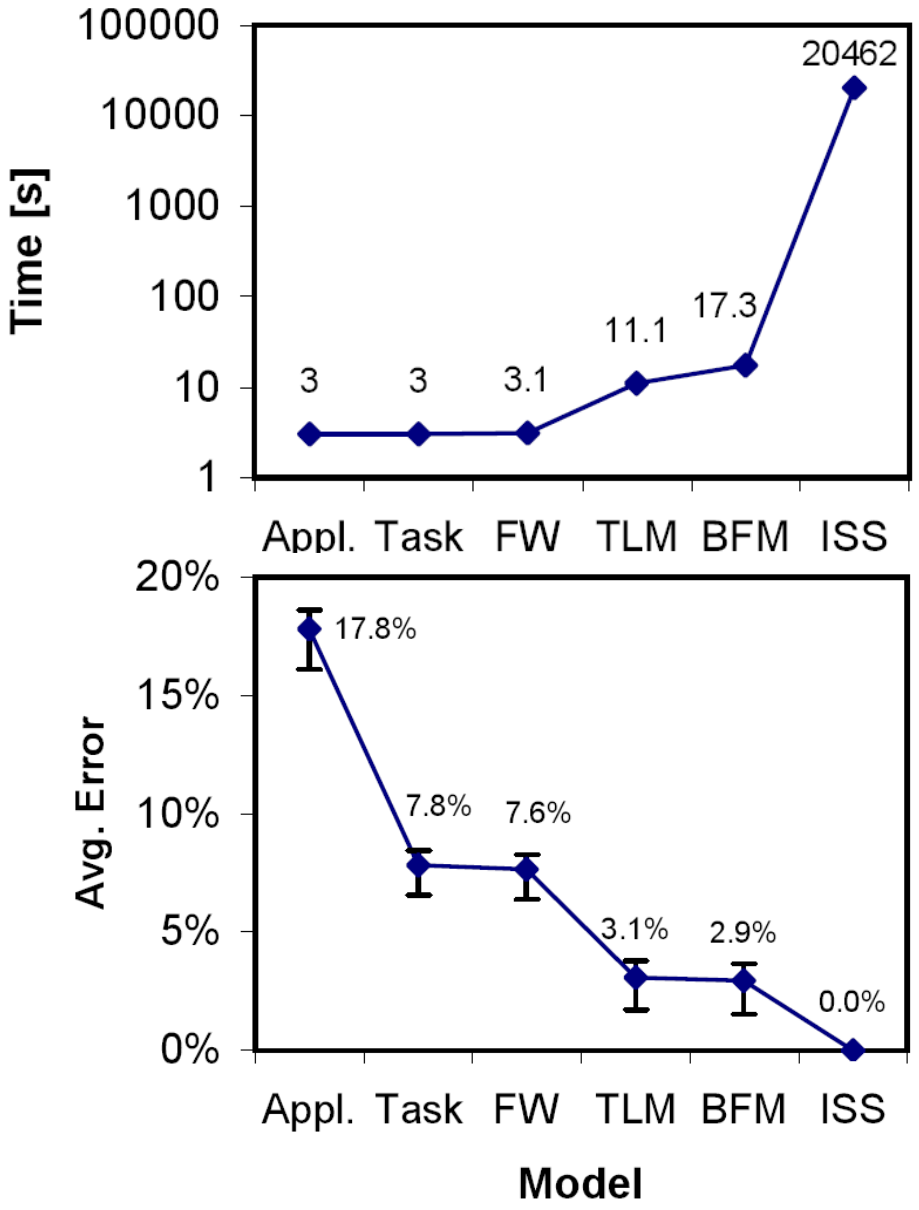
- **Voice encoding and decoding**
  - Motorola DSP 56600
    - Encoding & decoding tasks
    - custom OS
  - 4 custom I/O blocks
  - 1 custom HW co-processor
    - Codebook search
- **Processor models**
  - Perfect timing
    - Back-annotated from ISS
  - Priority-based OS model
    - EDF: Decoder > Encoder
  - HW interrupt scheduling
    - 4 non-preempted priority levels
- **Reference**
  - Motorola proprietary ISS



Processor Modeling Results

- **Execute on Sun Fire V240 (1.5 GHz)**
  - 163 speech frames
- **Speed vs. accuracy**
  - ➢ OS model (Appl ⇨ Task)
  - ➢ Interrupts (FW ⇨ TLM)

➢ **1800x speed w/ 3% error (vs. cycle-accurate ISS)**



Lecture 7: Summary

- **OS and Processor 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
- **Backend processor synthesis**
  - Software synthesis
    - Code generation, RTOS targeting, cross-compilation & linking
    - Fully automatic final target binary generation