ECE 587 – Hardware/Software Co-Design
Lecture 10 System Modeling, Software Processor Modeling

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

February 17, 2025

Reading Assignment

- This lecture: 3.3, 3.4
- Next lecture: 3.5

Outline

System Design Challenges

- Input: a high-level system specification
  - Could be functionalities only described in models of computations (MoCs)
- Output: a low-level system implementation
  - Software: programs for the targeted instruction sets
  - Hardware: what level?
- Methodologies and tools are mature for abstraction levels at and below RTL for hardware.
  - For system design, hardware implementations stop at RTL.
- No single step solution for system design
  - Huge semantic gap exists between specification and implementation: many ways to implement a single MoC

System Design Process

- Decompose the whole system design process into a series of smaller steps.
  - Ensure the semantic gap is small enough for a single step
- Each step is defined by a pair of system models.
  - The one at higher abstraction level serves as specification.
  - The one at lower abstraction level serves as implementation.
- Refinement: generate the implementation from the specification for each step
  - Introduce additional details limited to certain scope of the specification
  - Incorporate design decisions to choose one implementation from multiple possible ones
- While more tools are available for refinement, it is critical for designers to provide proper design decisions.
  - Especially when an initial system implementation fails to meet design constraints and multiple design iterations are necessary.

Typical System Design Tool and Design Process

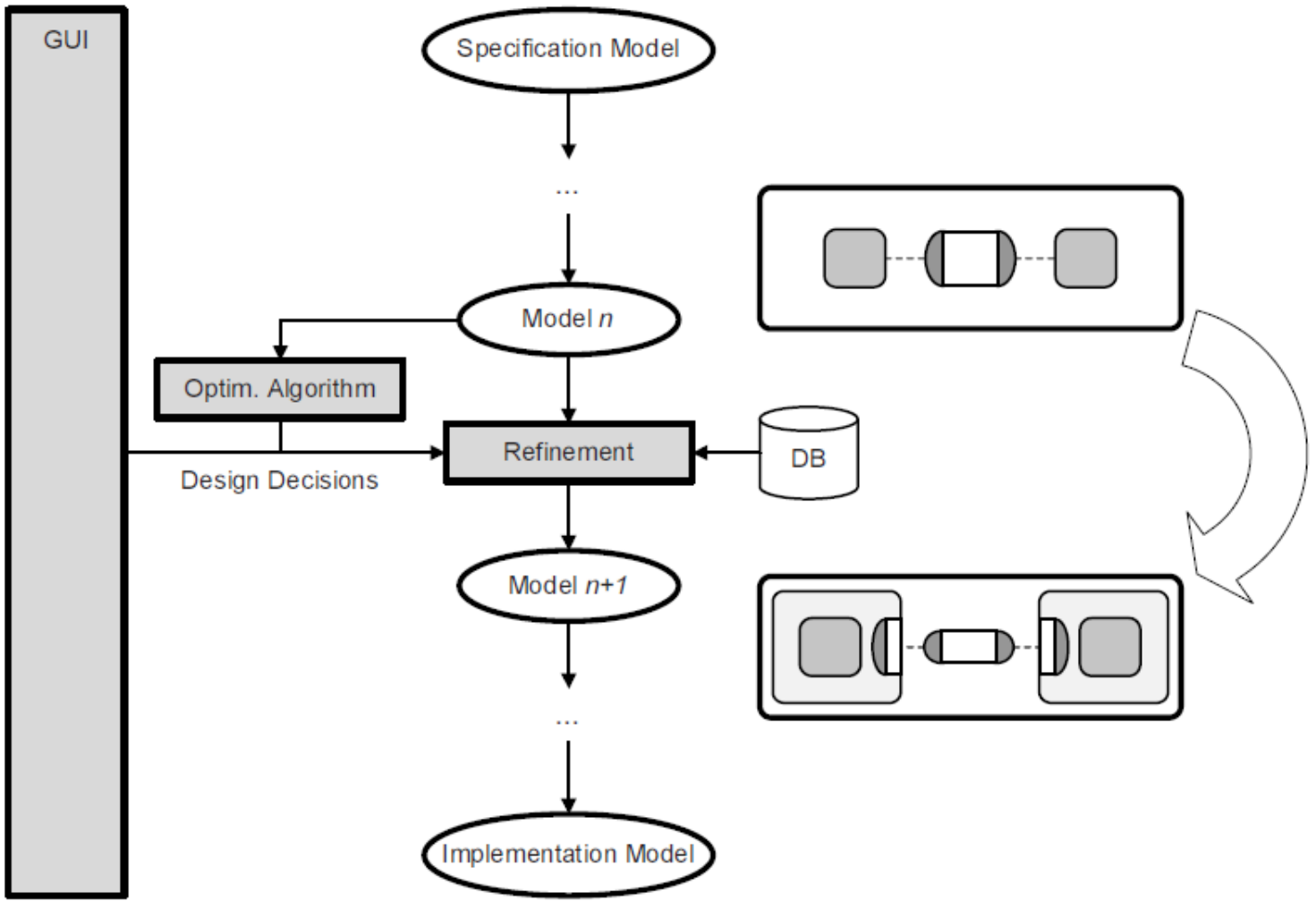

FIGURE 3.6 System design and modeling flow

(Gajski et al.)

Roles of Models

- The implementation of the previous design step will serve as the specification of the next one.
- For implementation, models allow designers to reason about design decisions by simulating and analyzing certain aspects of the system.
- For specification, models document system features that need to be implemented and decided.
- As design progresses,
  - More details are included into the models so simulation and analysis takes more time to finish.
  - Simulation and analysis will become more accurate due to the available details.
  - Designers will be able to afford the increased simulation and analysis time by focusing on the most important parts of the system.

Abstraction Levels for System Design

- At the highest abstraction level, we would assume the system is specified using process-based models while each process is specified using state-based models.
- What intermediate abstraction levels should we introduce for HW/SW implementations?
- Separate communication from computation
  - Compilers help to implement a single process as HW or SW.
  - Communications become limiting factors for system performance.
- Accurate system analysis demands accurate communication modeling.
  - The ratio of communication latency to computation latency generally increases as more transistors are packed into a chip.
  - Complex system requires more data to be transfered among subsystems, resulting in latency with limited bandwidth and excessive power consumption.

Processor and Communication Modelings

- Processor modeling provides necessary details to evaluate mappings from processes to processors.
  - Processors: a.k.a. processing elements
    - General-purpose processors
    - DSPs and ASIPs
    - ASICs
  - The computation part of system modeling
- The communication part of system modeling is based on communcation modeling.

Processor Modeling

- Processors specific to certain applications
  - Processes are usually specified with MoCs that can be directly mapped to the processors.
  - Based on hardware metrics, estimating system design metrics could be straight-forward.
  - Otherwise, we can model such processors as special cases of general-purpose processors.
- General-purpose processors, or software processor
  - Processes are usually specified as sequential programs.
  - To estimate system design metrics, one has to consider not only the programs but the supporting software (e.g. OS).
  - Much more complicated than the above case. Will be our focus.
- Challenges for software processors modeling
  - Most estimations of system design metrics, e.g. latency, throughput, and power consumption, depend on simulation.
  - Models enabling fast simulations while providing accurate (relatively) estimations are desired.
  - At what level should we model software processors?

Communication Modeling

- Adopt well-established ISO/OSI 7-layer model
  - Layers are stacked on top of each other.
  - Each layer provides services to layers above by using services of the layer below.
- Layers are tailored to specific system design requirements.
  - E.g. to reflect the HW/SW partitioning of the communication functionality
- Use of layers facilities reasoning about communication stacks
  - However, it should not prevent implementations to merging functionalities across layers for various optimizations.
  - The whole communication stack should be treated as a single specification.

Outline

Modeling Considerations

- At instruction level, processors can be modeled as FSMs, and processes can be modeled as assembly programs. However,
  - Simulations are slow as the supporting software are treated the same way.
  - With all the details mixed together, it is difficult for the designers to make design decisions for improvements, especially when multiple processes are mapped to a single processor.
- Separate the computation into layers
  - E.g. processor, OS/library, application processes
  - Allow designers to reason about various parts of the system.
  - Certain functionality, e.g. those from OS/library, can be simulated faster at levels higher than instructions.
- Software processor modeling is not limited only to the processors themselves but should include all pieces of the supporting software.

Typical Modeling of Software Processor

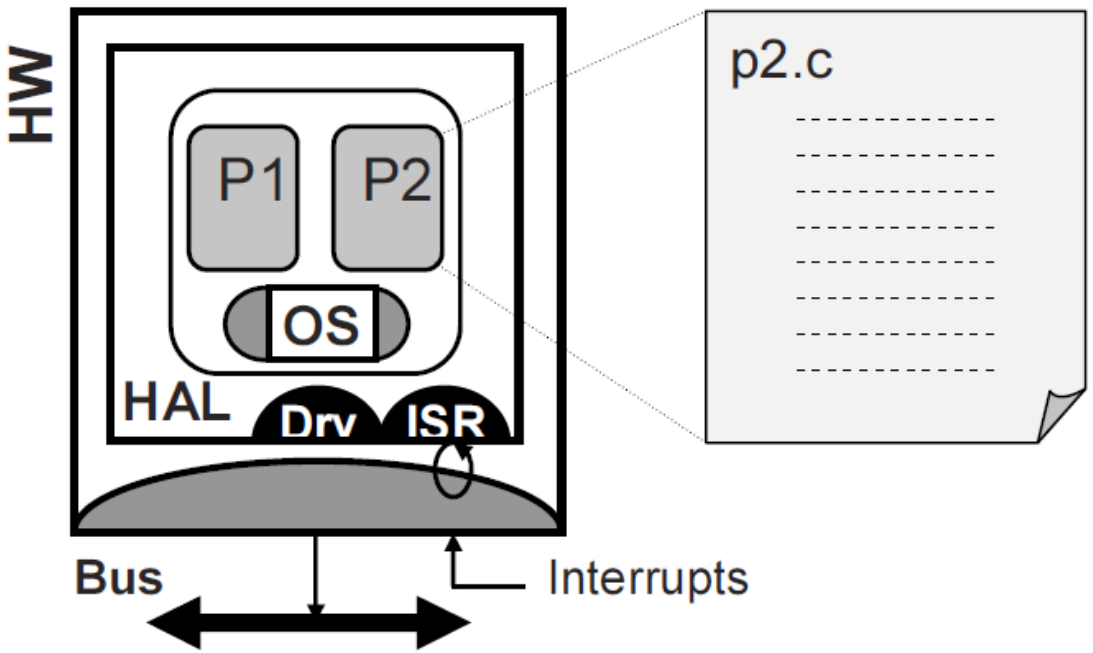

FIGURE 3.8 Processor modeling layers (Gajski et al.)

Typical Layers of Computation

- Processor Hardware (HW)
  - Actual hardware that communicates with the other parts of the system.
- Hardware Abstraction Layer (HAL)
  - Provide abstractions of actual hardware as canonical interfaces as most modern OS are hardware independent.
- Operating System (OS)
  - Provide necessary services like multi-tasking, scheduling, inter-process communication
- Application
  - With the help of OS, we can still model multiple processes mapped to the same processor as they are.
- Notion of time can be introduced as execution delay.
  - Via back-annotation

Outline

Application Modeling

- Applications are modeled as communicating processes.
  - All this processes are running on the SAME processor so somewhat you may ignore the costs associated with their communications.
- Processes may be composed hierarchically, e.g. through the fork-join model.
  - Process scheduling is handled at the OS layer and is irrelevant at this layer, though it may affect the performance of the system.
- Communication mechanisms: shared variable and message passing
  - Message passing could be in its special and high-level forms like events, semaphores, queues, and abstract channels.
  - Message passing is most likely implemented at the OS layer, though we need to include the specification for applications.

Specifying Applications

- It is quite common that the initial application specifications are migrated from some other systems.
  - Computations are specified in a programming language like C/C++
  - Except shared variables, communications are specified as function calls to certain library/OS.
  - Process management are specified in similar ways like communication.
- Methods for migrating code
  - Computations are included directly.
  - Function calls for communications/process management can be replaced by corresponding primitives.

Considerations for Simulation

- Fast simulation
  - Computation could be emulated in host instructions, usually much faster than simulating target instructions.
  - Simulation kernel may be optimized to reduce process management/scheduling overhead and communication costs, and to utilize the multi-processing power of the host.
- Accurate estimations
  - Back-annotation is necessary if timing behavior is of concern as simulation is not performed on targeted platform.
  - System design languages support back-annotations via execution timings, which are independent of their running times on host.

Application Layer Modeling Example

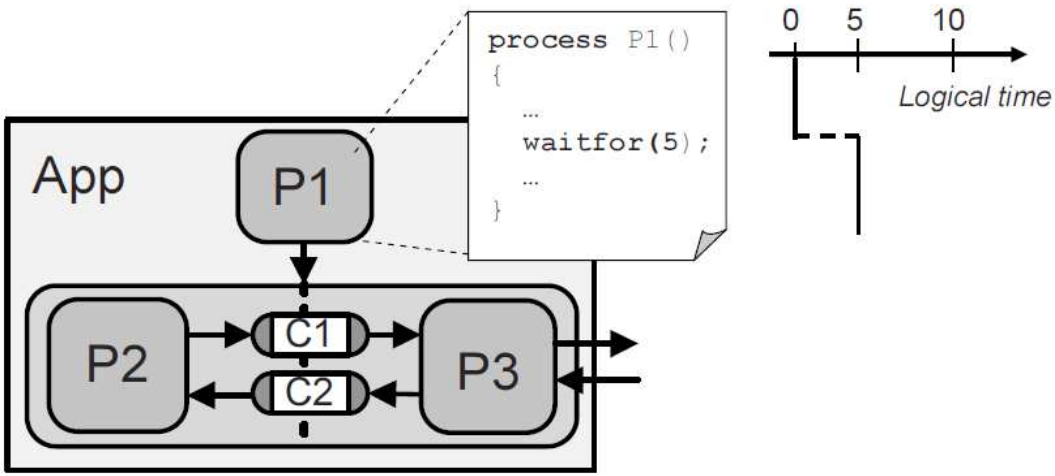

FIGURE 3.9 Application layer

(Gajski et al.)

More about Back-Annotation

- Back-annotations may be included at various level of computations.
  - E.g. the whole process or basic blocks
- As a process may react differently for different inputs, assigning it a simple delay number could be inaccurate.
- Associating delay numbers with basic blocks makes it possible to capture such dynamic behavior and thus improve accuracy.
  - However, it will slow down simulation because of the excessive interactions with the simulation kernel required by back-annotation.
- There is always a trade-off between simulation speed and accuracy.
- The delay numbers may be obtained by estimation or measurement.

Outline

OS and Multi-Processing

- We assume processes are running truly *concurrently* at application layer.
- However, when mapped to a single processor, that's not possible.
- OS provides the illusion that processes are running concurrently,
  - though actually they run sequentially on the processor.
- The major focus of the OS layer is thus to model the scheduling of parallel processes on the sequential processor.

Static vs. Dynamic Scheduling

- Static scheduling combines processes into a singe task following a pre-defined static schedule.
  - E.g. you may interleave the statements of multiple processes to achieve a static round-robin scheduling.
  - Modeling of static scheduling is no harder than modeling of a sequential program.
- What if a process need to communicate for either data or synchronization and have to wait?
  - The task has to wait.
  - All the processes in the task halt.
  - May hurt system latency/throughput or even lead to artificial deadlocks
- Dynamic scheduling allows OS, at runtime, to hang tasks that have to wait and to execute tasks that are ready to go.
  - For embedded systems, typically a Real-Time Operating System (RTOS) is used.
  - And thus a model of RTOS should be established.
  - Real-time here doesn't imply low computing latency, but means mechanism to meet certain deadlines.

Other OS Layer Modelings

- Process/task management
- Synchronization
- Timing, i.e. delay modeling
- Realize communication channels as IPC primitives
  - Synchronization may be required.

Interaction between Application and OS Layers

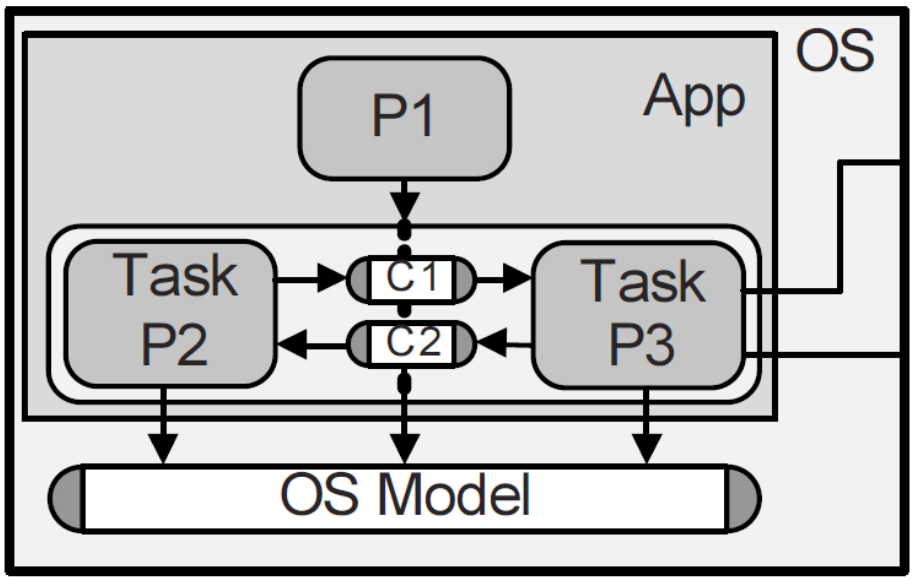

FIGURE 3.10 Operating system layer (Gajski et al.)

RTOS Modeling Choices

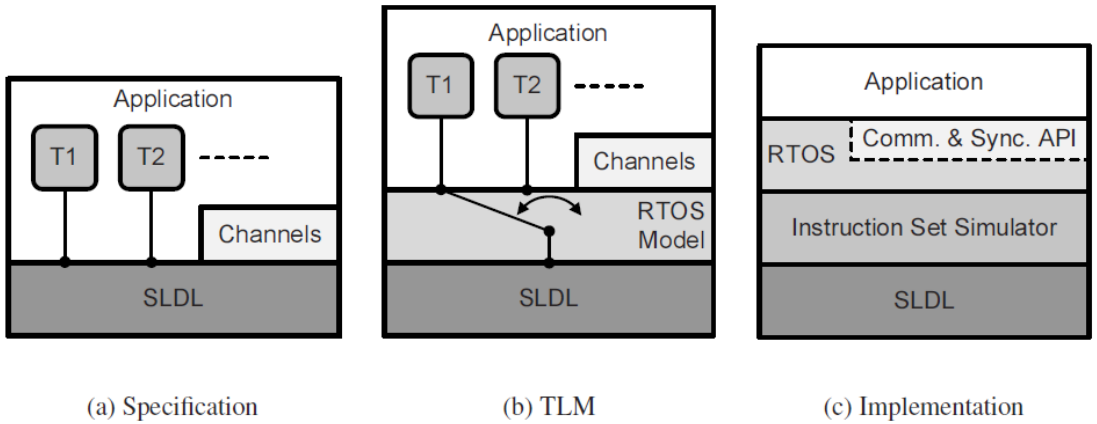

FIGURE 3.11 Operating system modeling

(Gajski et al.)

- For specification (a), tasks execute directly on the simulation kernel and scheduling is omitted.
- There are two choices to model the actual RTOS scheduling algorithm.
  - (c) simulation at instruction level
  - (b) provide a transaction-level RTOS model

Instruction vs. Transaction-Level RTOS Modelings

- Instruction level simulation
  - Need to compile all codes, including applications, to target instructions.
  - Simulation is done in an instruction-set simulator (ISS).
  - More accurate but slower than transaction-level RTOS models.
  - Less accurate but faster than cycle-accurate simulation of actual hardware that consider micro-architectural details.
- Transaction-level RTOS models
  - Enable fast application simulation in host instruction.
  - Use an abstract RTOS model that removes unnecessary implementation details that slow down the simulation.
  - Only care about multi-tasking, preemption, interrupt handling, inter-process communication, and synchronization
  - Add only a negligible simulation overhead
  - Allow designers to consider important OS effects early on in the design process

Example for Scheduling and Other OS Services

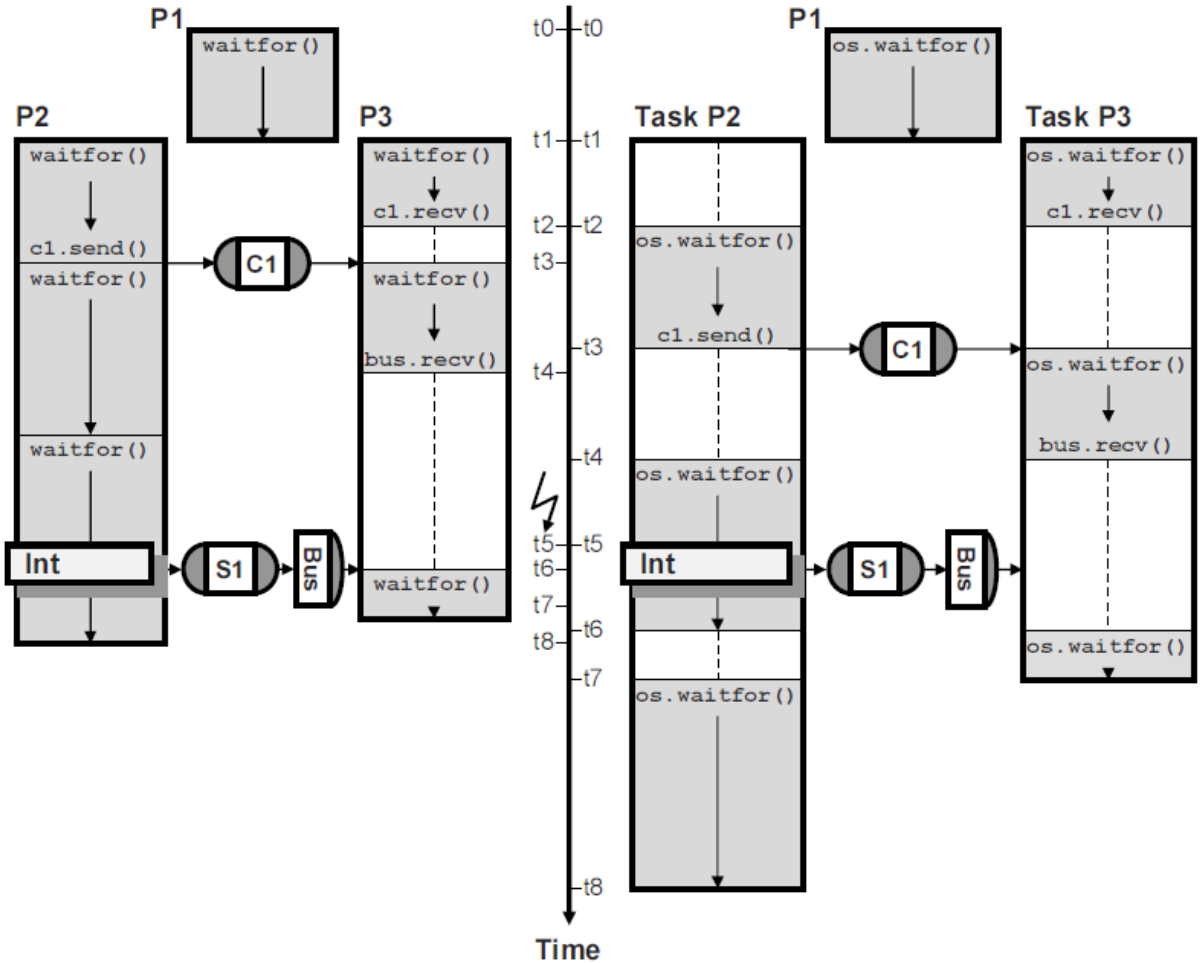

FIGURE 3.12 Task scheduling

(Gajski et al.)

- Back-annotations are supported via `waitfor()`.
- Why `S1` arrives at the same time but not `C1`?

Outline

Hardware Abstraction Layer (HAL)

- HAL provides the lowest level of software functionality.
  - Serve as interface between software and hardware
  - Implement canonical interfaces/services for use by OS, e.g. interrupt service routines (ISRs) and I/O drivers, enabling communications to other components.
- HAL is processor dependent.
  - Implementations (templates) are associated with processors, possible managed by a database when there are multiple types of processors in the system (the PE database).

Modeling I/O Drivers

- Communication at a bus interface may need to follow certain protocol.
  - Procedures to acquire/release the bus
  - Notifications via HW interrupts
  - The amount of bytes that could be transfered may be predefined.
  - Alignment requirements may be enforced for src/dest addresses.
- HAL use drivers to hide the details
  - Utilize two primitives: send and recv
  - May transfer arbitrary amounts of bytes at arbitrary addresses
- These driver implementations are directly integrated with the OS model.
  - Communications between processes on different processors are then realized based on bus models, which are seen per processors as interrupts and bus transactions.

A Possible send Function

```
send_0(device, data, len) { // send via certain bus
  bus.write_ctrl(ACQUIRE); // try to acquire the bus
  while (bus.read_ctrl() != MASTER) {
    // poll to see if the bus is acquired
    //
    // depending on bus implementations, this polling
    // may need to be replaced with waiting for certain
    // interrupt
  }

  msg = {device, DMA_SEND, data, len};
  bus.write_data(msg); // send the DMA request to the device

  bus.write_ctrl(RELEASE);

  wait for the interrupt that indicates the completion
  of the DMA operation from the device
}
```

Modeling HW Interrupts

- HW interrupts are triggered by external events.
  - For HAL, we are not interested in modeling interrupts caused by system calls, which should be modeled directly at OS level.
- HW interrupts need to be handled quickly.
  - So you won't miss other HW interrupts since they should be masked/blocked inside the ISRs.
  - The ISRs should contain minimum amount of codes, only those absolutely necessary.
  - What if there is a lot of work to do for certain HW interrupt?
- In most modern OS', further processings are deferred to user-level tasks.
  - They are created/triggered by HW ISRs and are scheduled at OS level.
  - Can be modeled in similar ways as application tasks

A Possible HW ISR

```
hw_isr_0() { // isr for certain HW interrupt
  block_all_interrupts();
  msg = extract data associated with interrupt
  add_user_task(process_isr_0, msg);
  if (preempt)
    switch to the context of the OS scheduler
  unblock_all_interrupts();
}

process_isr_0(msg) {
  if (msg.reason == DMA_SEND_DONE) {
    sem_dma_done[msg.device].post(); // notify sender by semaphores
  }
  ...
}

send_0(device, data, dest_addr) {
  ...
  sem_dma_done[device].wait(); // this is how the waiting is implemented
}
```

HAL Layer Modeling Example

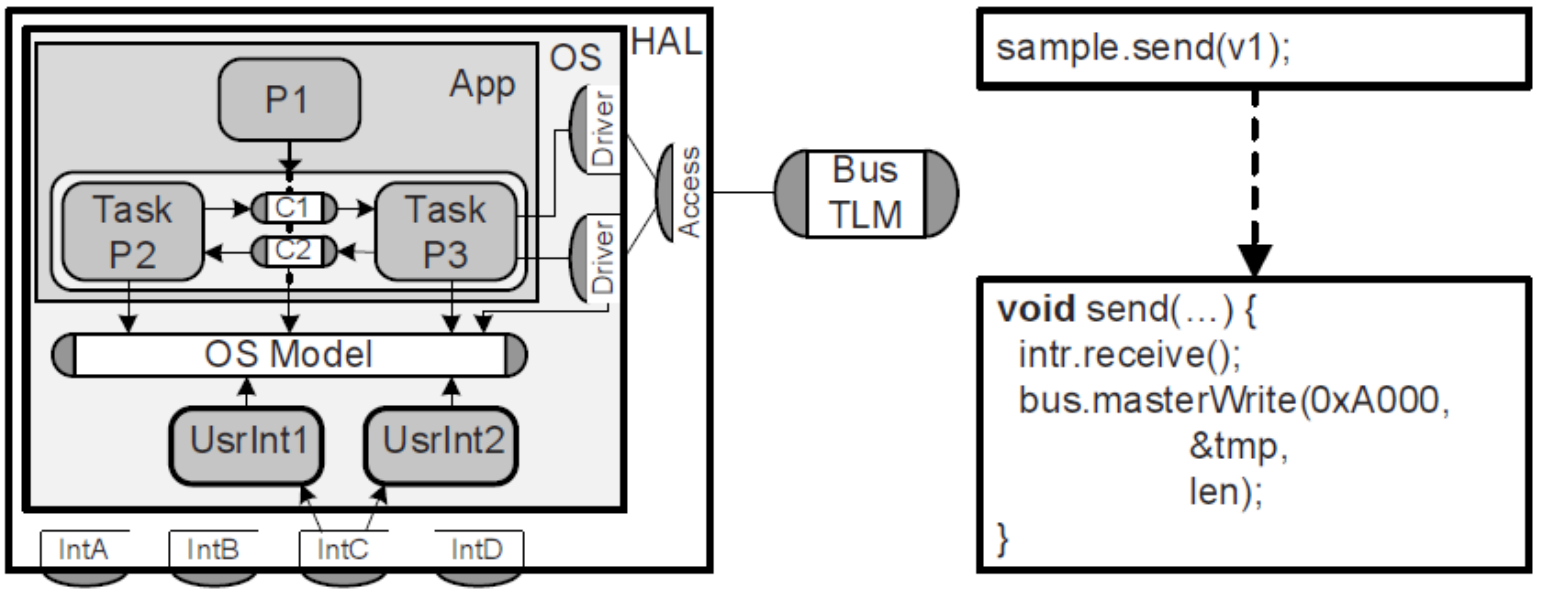

FIGURE 3.13 Hardware abstraction layer

(Gajski et al.)

- What about computations in HW ISRs? Where do they run?
  - E.g. IntA to IntD, or hw_isr_0?

Outline

Interrupt Scheduling

(a) HAL (b) Hardware

FIGURE 3.14 Interrupt scheduling

(Gajski et al.)

- The HW ISR (IntC) would also consume processor cycles for computation
  - If such HW ISRs do consume a significant amount of processor cycles, they have to be "scheduled" with other processes (P1 and P2) for accurate modeling.

Hardware Layer

- Provide means to model the timing of HW ISRs
- HW ISRs are NOT scheduled by OS.
  - They are triggered by external events and are usually fired at the instruction boundary.
  - At the end of these ISRs, you may choose to simply return to the current pending task or hand over the control to OS scheduler.
- Hardware layer modeling should consider and include such implementation details.
  - Introduce a separate model of the processor's hardware interrupt logic
  - Suspend HAL/OS/application when HW interrupts arrive
  - Resume execution when exiting ISRs, possibly causing OS to re-schedule tasks.

Hardware Layer Modeling Example

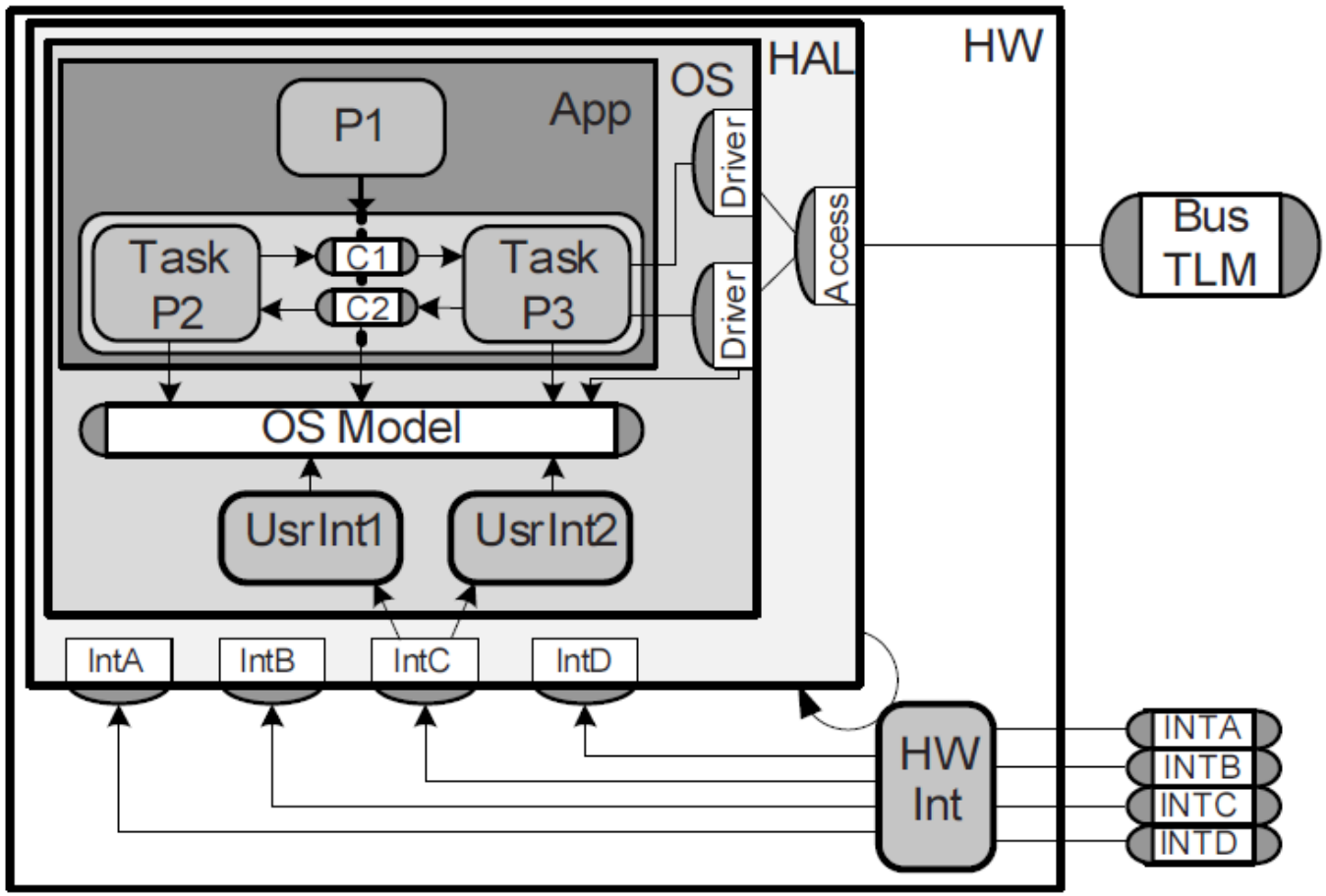

FIGURE 3.15 Hardware layer

(Gajski et al.)

Other Hardware Layer Modelings

- Peripherals, e.g. timers, immediately associated with the processor.
- (Programmable) interrupt controller, e.g. for interrupt priorities.
- Communications via bus models
  - Transaction-level bus models
  - Pin-accurate and cycle-accurate bus models