ECE 587 – Hardware/Software Co-Design
Lecture 14 System Modeling II

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

February 25, 2026

Reading Assignment

- This lecture: 3.4, 3.5
- Next lecture: 3.5, 7.1

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

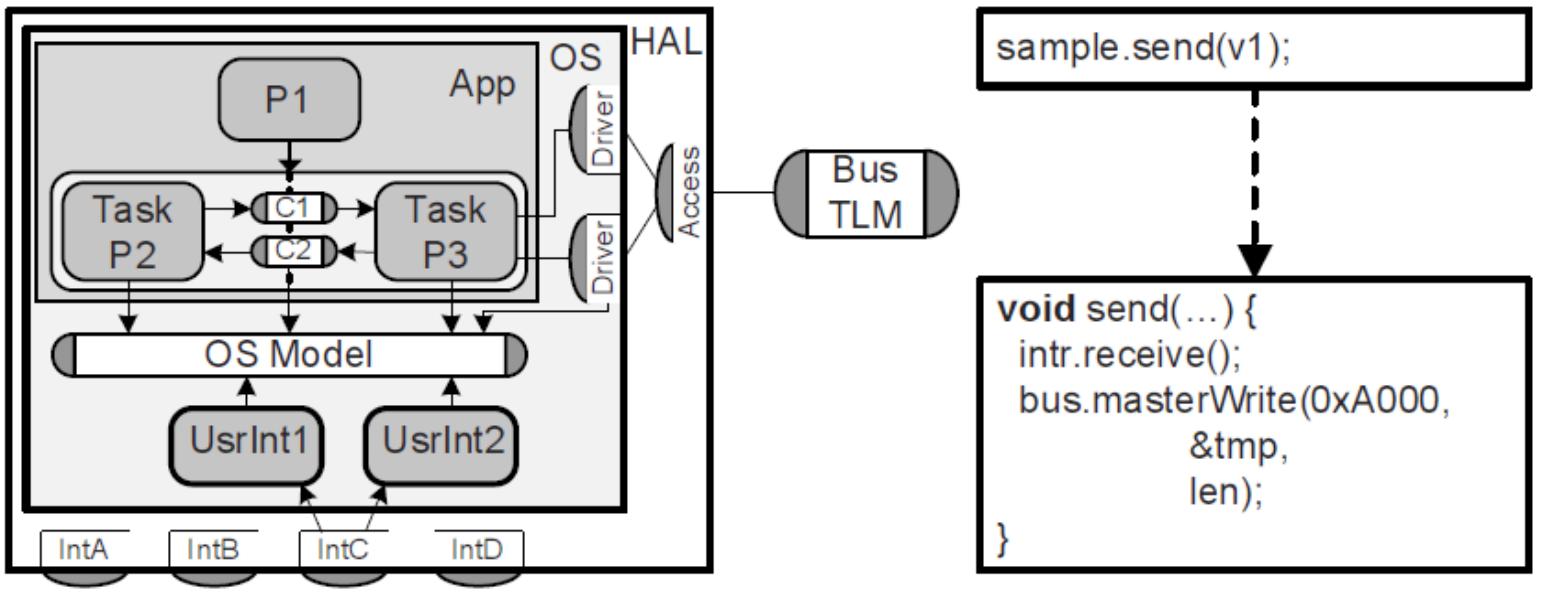


FIGURE 3.13 Hardware abstraction layer

(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

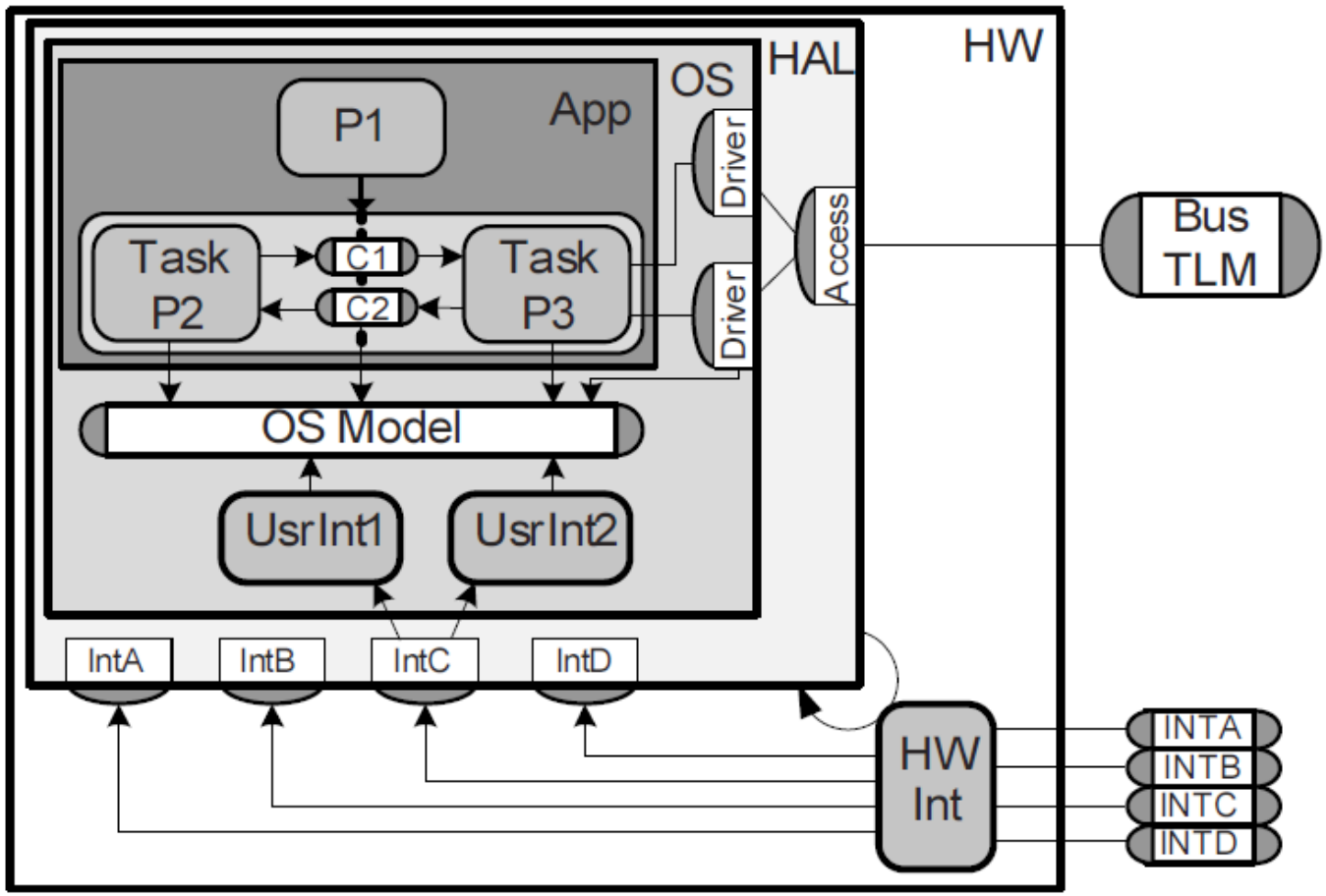


FIGURE 3.15 Hardware layer

(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

Outline

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.

The Upper 5 ISO/OSI Layers

TABLE 3.2 Communication layers

Layer	Semantics	Functionality	Implementation	OSI
Application	Channels, variables	Computation	Application	7
Presentation	End-to-end typed messages	Data formatting	OS	6
Session	End-to-end untyped messages	Synchronization, multiplexing	OS	5
Transport	End-to-end data streams	Packeting, flow control	OS	4
Network	End-to-end data packets	Subnet bridging, routing	OS	3

(Gajski et al.)

The Lower 2 ISO/OSI Layers

TABLE 3.2 Communication layers

Layer	Semantics	Functionality	Implementation	OSI
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

(Gajski et al.)

Outline

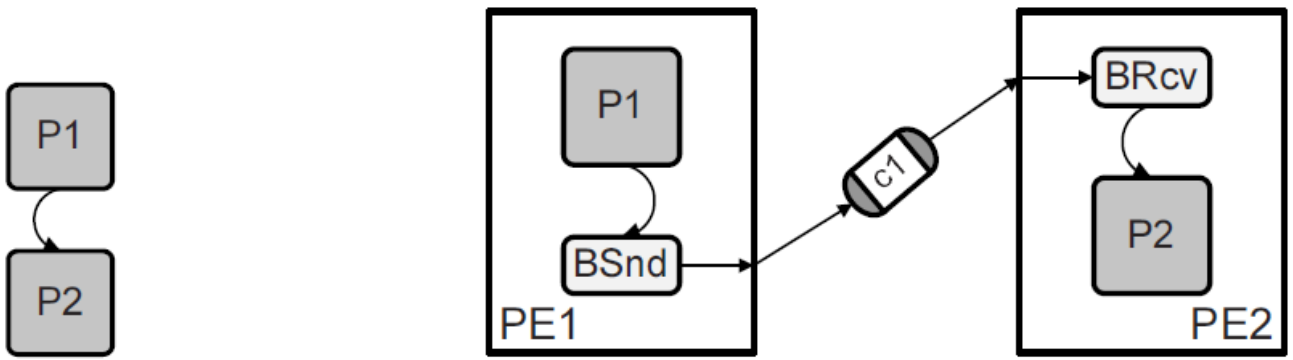
Application-Level Communication Primitives

- Application-level communication modeling should support a rich set of primitives.
  - For processes to communicate no matter they are on the same processor or not.
  - The primitives are expected to provide guaranteed delivery.
- Events for one-way synchronization, e.g. control flow transitions
- Shared variables
- Message-passing channels
  - Synchronous: both parties block
  - Asynchronous: receiver blocks
- Queues as a special case of asynchronous message-passing with well defined, fixed buffer sizes.
- Other complex and user-defined channels with extended semantics.
  - E.g. semaphores or mutexes.

Application Layer Communication Modeling

- Once we map processes to processors, communications among them should also be mapped.
- We are interested in communications between processes running on different processors here.
  - OS will utilize similar technique to model communications between processes running on a single processor.
- The application layer is responsible to translate the previous mentioned primitives to a restricted set of primitives supported by the following design process.

Making Synchronizations Explicit



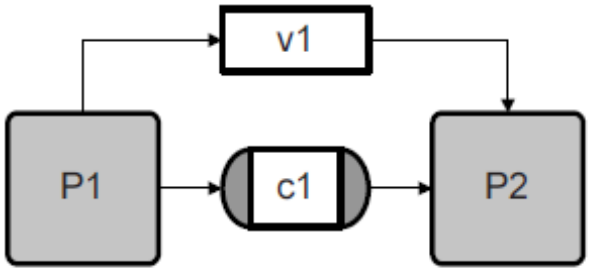
(a) Behavioral model (b) Synchronization

FIGURE 3.16 Application layer synchronization

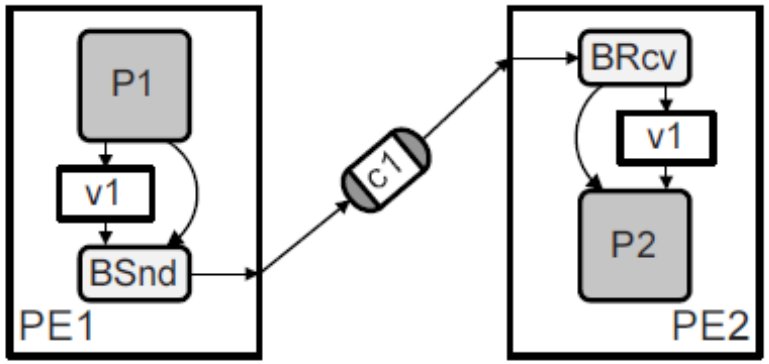
(Gajski et al.)

- There exists a dependency among P1 and P2 running on different processors.
- Explicit synchronization between the processes are necessary.
- Two processes and one channel are introduced.
  - No modification of the processes P1 and P2.

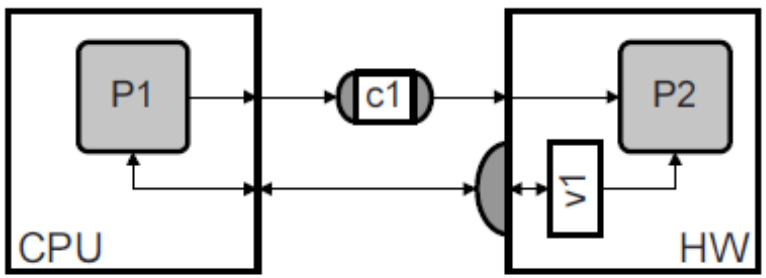
Resolving Memory Visits



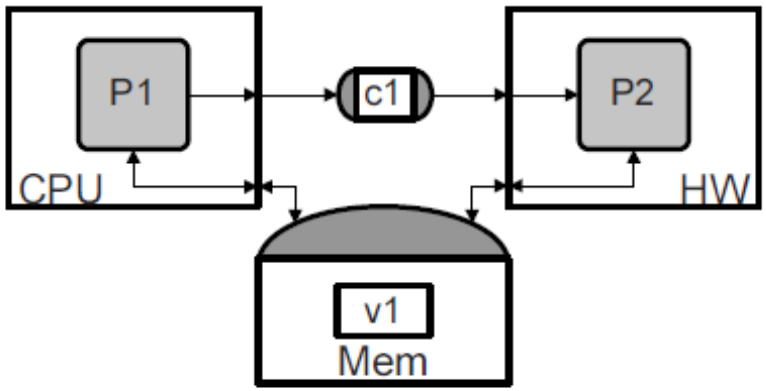
(a) Behavioral model



(b) Distributed message-passing



(c) Memory-mapped I/O

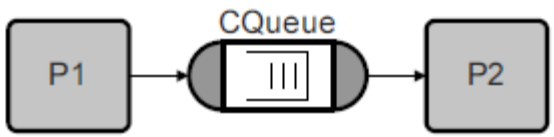


(d) Shared memory

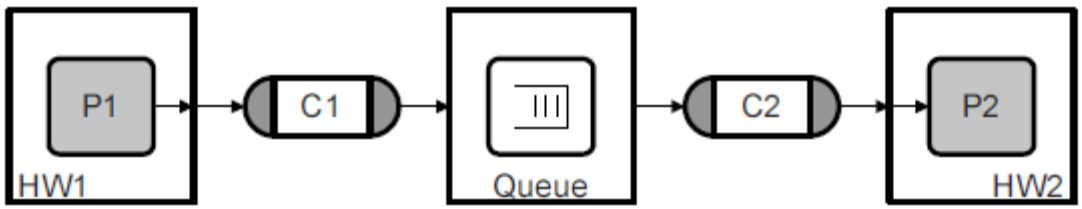
FIGURE 3.17 Application layer storage

(Gajski et al.)

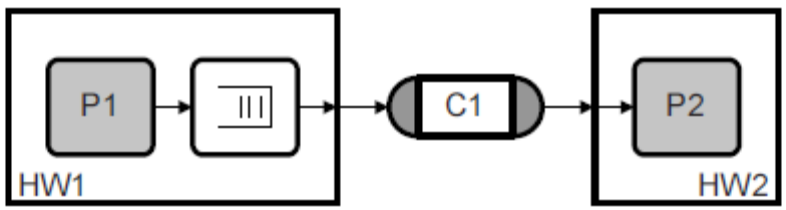
Incorporating Computation for Complex Channels



(a) Behavioral model



(b) Dedicated PE



(c) Local processes

FIGURE 3.18 Application layer channels

(Gajski et al.)

Outline

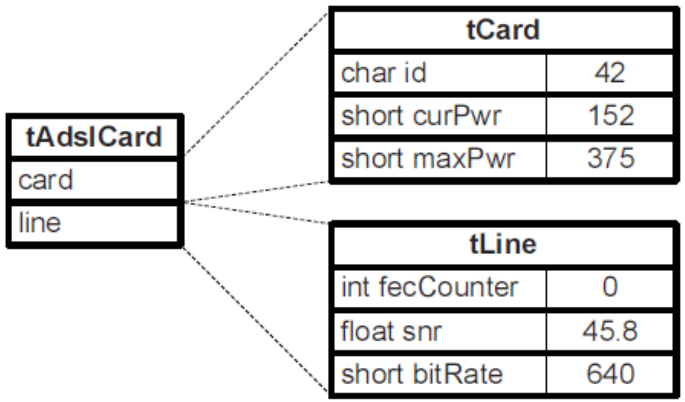
Presentation Layer

- Layout of data is processor dependent.
- Presentation layer is responsible for explaining the meaning of bytes.
  - Formatting of data
  - Conversion between data in abstract types to untyped blocks of bytes
- Layout parameters
  - Bitwidth of machine character, i.e. the smallest addressable unit
  - Size of primitive data types
  - Alignment of primitive data type
  - Endianess, i.e. ordering of characters in primitive data types
- Presentation layer adds the details to model the storage/performance overhead associated data formatting and conversion.

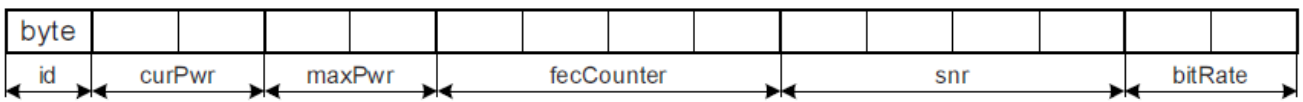
Presentation Layer Communication Modeling

- For each set of communication partners, a common data layout is chosen to exchange data.
  - The communication mechanism is based on message passing or shared memory.
- The chosen layout can be the same as in both, one or none of the involved processors.
  - Any difference in the layout parameters of network, memory, and processor would require conversions.
  - You may have an empty presentations layer if all parameters are the same.
- The layouts may either be globally defined or chosen differently.
  - Globally, one can save data traffic by choosing a layout without alignment requirement.
  - One can alternatively choose different layouts to match parameters of participating processors to reduce conversion overhead.

Presentation Layer Modeling Example



(a) Application data structure



(b) Network byte stream

FIGURE 3.19 Presentation layer

(Gajski et al.)