Reading Assignment

- This lecture: 3.4
- Next lecture: 3.5
Outline

Overview

Hardware Abstraction Layer

Hardware Layer
Software Processor Modeling

- **Software processor**
  - To estimate system design metrics, one has to consider not only the programs but the supporting software (e.g. OS).
  - Software processor modeling is not limited only to the processors themselves but should include all pieces of the supporting software.

- **Challenges**
  - Most estimations of system design metrics, e.g. latency, throughput, and power consumption, depend on simulation.
  - Models enable fast simulations while provide accurate (relatively) estimations are desired.
Layers of Computation

- The computation on a software processor is decomposed into layers.
  - 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.

- Typical layers
  - Processor Hardware (HW)
  - Hardware Abstraction Layer (HAL)
  - Operating System (OS)
  - Application
The Application Layer

- Applications are modeled as communicating processes.
  - Processes may be composed hierarchically, e.g. through the fork-join model.
  - Communications are specified via shared variables and various forms of message passing, e.g. synchronization.

- Choosing a proper system design language is vital.
  - Minimize the effort of migrating code
  - Enable fast simulation by emulating computation in host instructions

- Use back-annotation to improve estimation accuracy
  - Back-annotation introduces execution delay and logic timing, which are independent of simulation running times on host.
  - There are always trade-offs between simulation time and estimation accuracy.
The Operating System Layer

- OS provides the illusion that processes are running concurrently,
  - though actually they run sequentially on the processor.
  - The major focus is thus to model the scheduling, with an emphasis to RTOS.
- Transaction-level RTOS models combine fast application simulation in host instructions and instruction-level modeling accuracy.
  - Remove unnecessary implementation details. Only care about multi-tasking, preemption, interrupt handling, inter-process communication, and synchronization
  - Allow designers to consider important OS effects early on in the design process
Outline

Overview

Hardware Abstraction Layer

Hardware Layer
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. ISR 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).
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. ISR 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 transferred 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.
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 transferred 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.
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 transferred 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
}
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/block 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
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.
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
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
What about computations in HW ISRs? Where do they run?
- E.g. IntA to IntD, or hw_isr_0?
HAL Layer Modeling Example

What about computations in HW ISRs? Where do they run?

- E.g. IntA to IntD, or hw_isr_0?

Figure 3.13 Hardware abstraction layer

(Gajski et al.)
Outline

Overview

Hardware Abstraction Layer

Hardware Layer
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.
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

- 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.
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.
**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
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
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
Typical problems solved at various levels

- **Appl.:** Will communications introduce too much overhead for the process mapping?
- **OS:** Will the scheduler guarantee deadlines to be met?
- **HAL:** Is the driver implementation correct and efficient?
- **HW:** Will the system HW work properly with the system software?
- **ISS:** How exactly does the system perform?