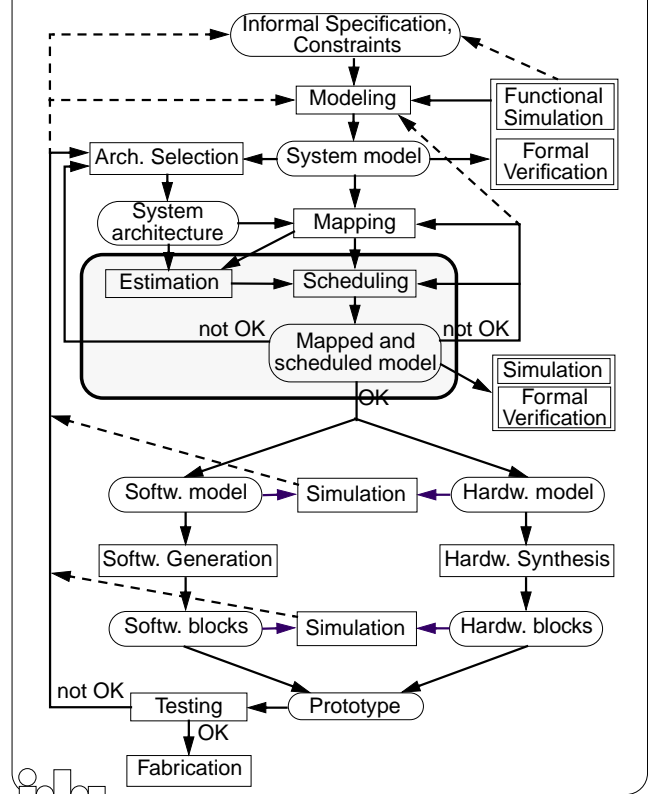


## Real-Time Embedded Systems: Task Scheduling

1. Real-Time Systems and Their Typical Features
2. Task Scheduling Policies
3. Static Cyclic Scheduling
4. What is Good and Bad with Static Cyclic Scheduling
5. Priority Based Preemptive Scheduling
6. Schedulability Analysis

## Remember the Design Flow



## Real-Time Systems

⇒ Many (most) embedded systems are real-time systems.

- A real-time system is a computer system in which the correctness of the system behavior depends not only on the logical results of the computations but also on the time when the results are produced.

### Examples:

- Process control systems
- Computer-integrated manufacturing systems
- Aerospace and avionics systems
- Automotive electronics
- Medical equipment
- Nuclear power plant control
- Defence systems
- Consumer electronics
- Multimedia
- Telecommunications

## Real-Time Systems: Some Typical Features

- They are time - critical. The failure to meet time constraints can lead to degradation of the service or to catastrophe.
- They are made up of concurrent tasks. The tasks share resources (e.g. processor) and communicate to each other. This makes scheduling of tasks a central problem.
- Reliability and fault tolerance are essential. Many applications are safety critical.

### Soft and Hard Real-Time Systems

☞ Time constraints are often expressed as *deadlines* at which tasks have to complete their execution.

A deadline imposed on a task can be:

- **Hard deadline:** has to be met strictly, if not ⇒ "catastrophe".  
- should be guaranteed a-priori, off-line.
- **Soft deadlines:** tasks can be finished after their deadline, although the value provided by completion may degrade with time.
- **Firm deadlines:** similar to hard deadlines, but if the deadline is missed there is no catastrophe, only the result produced is of no use any more.



### Predictability

☞ Predictability is one of the most important properties of any real-time system.

☞ Predictability means that it is possible to guarantee that deadlines are met as imposed by requirements:

- Hard deadlines are always fulfilled.
- Soft deadlines are fulfilled to a degree which is sufficient for the imposed quality of service.



### Predictability (cont'd)

Some problems concerning predictability:

- Determine worst case execution times for each task.
- Determine worst case communication delays on the interconnection network.
- Determine time overheads due to operating system (interrupt handling, task management, context switch, etc.).
- After all the problems above have been solved, comes the "big question":

*Can the given tasks and their related communications be scheduled on the available resources (processors, buses), so that deadlines are fulfilled?*



### Task Scheduling

The scheduling problem:

Which task and communication has to be executed at a certain moment on a given processor or bus respectively, so that time constraints are fulfilled?

- ☞ A set of tasks is *schedulable* if, given a certain scheduling policy, all constraints will be completed (which means, a solution to the scheduling problem can be found).
- ☞ At least for hard real-time systems, it is needed to check off-line, in advance, if the system is schedulable.



### Task Characteristics

What do we assume to know about a task?

- Computation time (worst case computation time),  $c$ .  
For communication, we assume to know communication time.
- Deadline for task completion,  $d$ .
- Regularity of task arrival:
  - *periodic* tasks, with period  $T$  (infinite sequence of identical activities).
  - *aperiodic* tasks: no fixed period of arrival
    - *sporadic* tasks: bound minimum inter-arrival time  $\Rightarrow$  deadlines can be guaranteed off-line.
    - If no bounds on inter-arrival time are known, schedulability cannot be guaranteed.



### Task Characteristics (cont'd)

Inter-task dependencies:

- Precedence relations:
  - Because of application dependent ordering (task  $T_2$  has to be performed after  $T_1$ , although  $T_2$  does not receive any input data from  $T_1$ ).
  - Generated by data-dependencies (like, for example, specified in a data-flow model)
- Resource dependencies
  - Because of shared resources (processors, buses, peripherals, buffers, etc.).



### Scheduling Policies

☞ Static cyclic scheduling

- A table is generated off-line containing activation times for each task (communication). The activation sequence captured by the table is repeated cyclically.

☞ Priority based scheduling

- Tasks are activated in response to a certain event. In case of conflict (several tasks ready to be executed on the same processor), priorities are considered.
- Priorities can be assigned:
  - statically (fixed off-line and kept unchanged during execution)
  - dynamically (change during execution)



### Scheduling Policies (cont'd)

☞ Preemptive scheduling

- A running task can be interrupted in order to execute another task.

☞ Non-preemptive scheduling

- A task, once started, may not be stopped.



### Static Cyclic Scheduling

- Generate off-line activation times for all tasks.
  - This activation times determine the behavior of the system over a (hyper)period  $T^h$ .
  - This sequence of activations is repeated in a cyclic manner.
- If all tasks have the same period  $T \Rightarrow T^h = T$ .
- If the tasks have different periods  $T_1, T_2, \dots, T_n$ 
  - $\Rightarrow T^h = \text{LCM}(T_1, T_2, \dots, T_n)$ .

See also examples in F6 1-2 (slides 24, 28, 29, 30, 32).



### Static Cyclic Scheduling (cont'd)

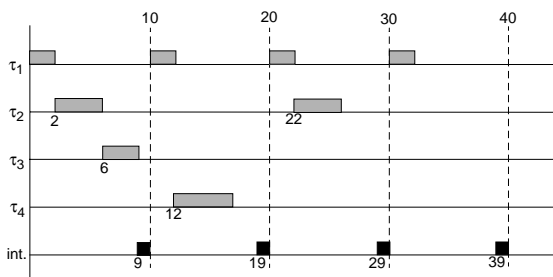
An Example: 4 independent tasks on one processor

	Period=deadline	Worst case comp. time
$\tau_1$	10	2
$\tau_2$	20	4
$\tau_3$	40	3
$\tau_4$	40	5
System management	10	1

$$T^h = \text{LCM}(10, 20, 40) = 40$$

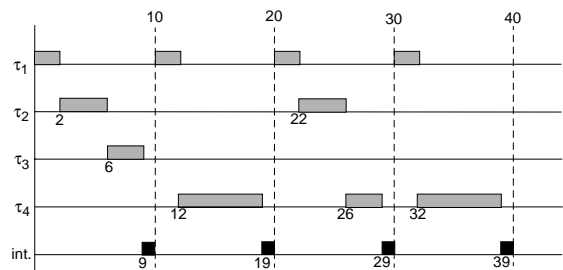


### Static Cyclic Scheduling (cont'd)



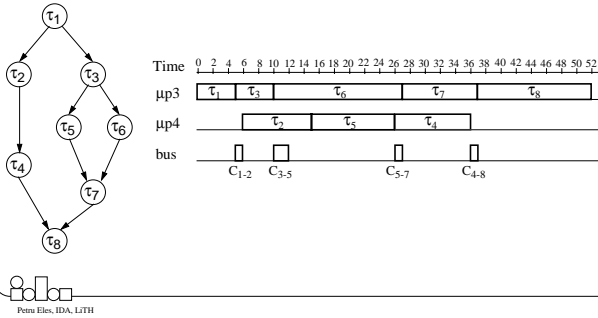
### Static Cyclic Scheduling (cont'd)

- Consider the same example as before, but computation time for  $\tau_4$  is 17  $\Rightarrow$  without preemption we cannot build a schedule!



### Static Cyclic Scheduling (cont'd)

⇒ Often we have to schedule data dependent tasks mapped on several processor nodes. Example from Lecture 1-2:



### List Scheduling

List scheduling is a classical algorithm for static cyclic scheduling.

Preliminary operations for priority assignment

**for each** processing element  $pe_i$ ,  $i=1,2,\dots,N_{pe}$ , **do**  $free_{pe_i}=0$  **end for**;

$T_{current}=0$ ; schedule  $\tau_0$  at  $T_{current}$ ; --  $\tau_0$  is the source node of the task graph

**repeat**

Update ready task lists  $List_{pe_i}$ ;

**for each** processing element  $pe_i$ ,  $i=1,2,\dots,N_{pe}$ , **do**

**if**  $T_{current} \geq free_{pe_i}$  **then**

$p=Select(List_{pe_i})$ ; schedule task  $p$  at time  $T_{current}$ ;  $free_{pe_i}=T_{current}+c_p$

**end if**

**end for**

$T_{current}=t_{next}$ , where  $t_{next}$  is the first time when a scheduled and yet active task will terminate;

**until** all direct predecessors of  $\tau_N$  are scheduled; --  $\tau_N$  is the sink of the polar graph

### List Scheduling (cont'd)

- For each processing element a *list of ready tasks* is maintained. It contains those tasks mapped to that processing element which have not yet been scheduled, but all their predecessors have been scheduled and finished.
- Tasks are extracted from the ready lists, in order to be scheduled, based on a *certain priority function*.
- A very simple priority function: critical path.

### What is Good with Static Cyclic Scheduling?

- High predictability
- Easy to debug
- Low execution time overhead (not much to do for the real-time kernel during execution time)

### What is Bad with Static Cyclic Scheduling?

- Not flexible:
  - quality degrades rapidly if periods and execution times deviate from those predicted;
  - if new tasks are added, the whole schedule has to be regenerated.
- Urgent events (interrupts) are handled purely:
  - time slots are statically allocated for polling and handling such events.
- Very long hyper-periods have to be avoided:
  - the periods of individual tasks have to be adjusted; this can lead to artificially reduced periods  $\Rightarrow$  artificially increased load  $\Rightarrow$  waste of processor time.
- Tasks have to be "manually" split, in order to fit into available slots.

### Priority Based Preemptive Scheduling

- No schedule (predetermined activation times) is generated off-line. Tasks are activated as response to events (e.g. arrival of a signal, message, etc.).
- At any given time the highest priority ready task is running. If several tasks are ready to be activated on a processor, the highest priority task will be executed.
- Tasks can be preempted at any moment. If a task becomes ready to be executed (the respective event has occurred), and it has a higher priority than the running task, the running task will be preempted and the new one will execute.

### Priority Based Preemptive Scheduling (cont'd)

- Priorities can be assigned statically (off-line) or dynamically (during execution).
- Will the tasks meet their deadlines?  
*Schedulability analysis* tries to answer this question.

### Schedulability Analysis

Schedulability analysis is possible for certain, more or less restricted, task models:

- Task periods and execution times are known
- Task priorities are given statically  
or  
Certain restricted dynamic priority policies are used like, for example, earliest deadline first (EDF).
- Tasks are executed on a single processor  
or  
on multiprocessor systems with a communication infrastructure such that communication delays are predictable (CAN bus, TDMA protocols, etc.).

### Schedulability Analysis (cont'd)

⇒ As result of research in the real-time systems, a mathematical apparatus has been developed (in the '90s) for schedulability analysis.

This results are in form of conditions which can be used in order to check if a certain task set is schedulable (all tasks meet their deadline) or not.

- ⇒ Schedulability analysis can be based on
- Sufficient conditions (sometimes too pessimistic).
  - Necessary and sufficient conditions (sometimes difficult to apply).
- ⇒ We will show some of the simpler formulas, only to give a "feeling".



### Schedulability Analysis (cont'd)

- A set of  $n$  tasks, with period  $T_i$  and worst case execution time  $c_i$ .  
Their deadline is equal with their period:  $d_i = T_i$ .

Task priorities are statically assigned to tasks, according to their period: the task with shorter period gets the higher priority.

- A sufficient (not necessary) condition for the task set to be schedulable:

$$\sum_{i=1}^n \frac{c_i}{T_i} \leq n \left( 2^{\frac{1}{n}} - 1 \right)$$



### Schedulability Analysis (cont'd)

- A set of  $n$  tasks, with period  $T_i$  and worst case execution time  $c_i$ .

Arbitrary deadline:  $d_i \leq T_i$

Task priorities are statically assigned, but can be arbitrary.

- The response time for each task can be calculated based on the following recurrence relation (which converges if processor utilisation is less than 100%):

$$r_i = c_i + \sum_{\forall k \in hp_i} \left\lceil \frac{r_i}{T_k} \right\rceil c_k$$

This is the interference from higher priority tasks.

- A necessary and sufficient condition for schedulability:  $r_i \leq d_i$



### Schedulability Analysis (cont'd)

- Schedulability conditions have been developed to handle more general systems:

- Deadlines which can be larger then the period
- Tasks with share critical resources
- Multiprocessors



### Summary

- The correctness of real-time systems depends not only on the logical results computed, but also on the time when these results are produced.
- The deadlines imposed on real-time systems can be hard or soft.
- In order to guarantee the correctness of real-time systems these systems have to be predictable.
- Task scheduling determines which task (communication) to be executed at certain moment, so that time constraints are fulfilled.
- A task set is schedulable if it is possible to schedule the tasks such that all constraints are fulfilled.

### Summary (cont'd)

- With static cyclic scheduling all activation times are determined off-line and fixed in a schedule table. List scheduling is a possible algorithm for constructing a static schedule.
- Static cyclic scheduling provides good predictability. The execution overhead is small, and systems are easy to debug. However, systems are not flexible and urgent events cannot be handled properly.
- With priority based preemptive scheduling, no schedule is determined off-line. Tasks are activated as response to events (messages), based on their relative priority.
- Schedulability tests are available in order to check if a certain set of tasks is schedulable or not.