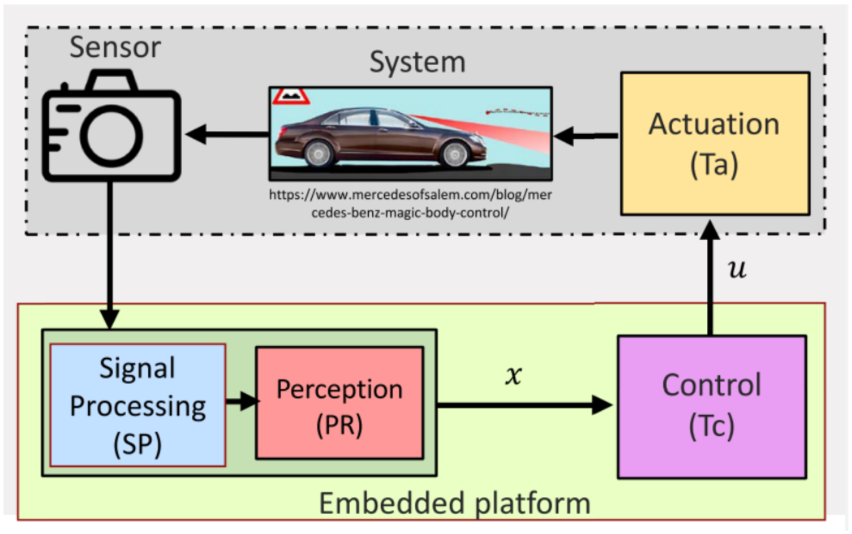
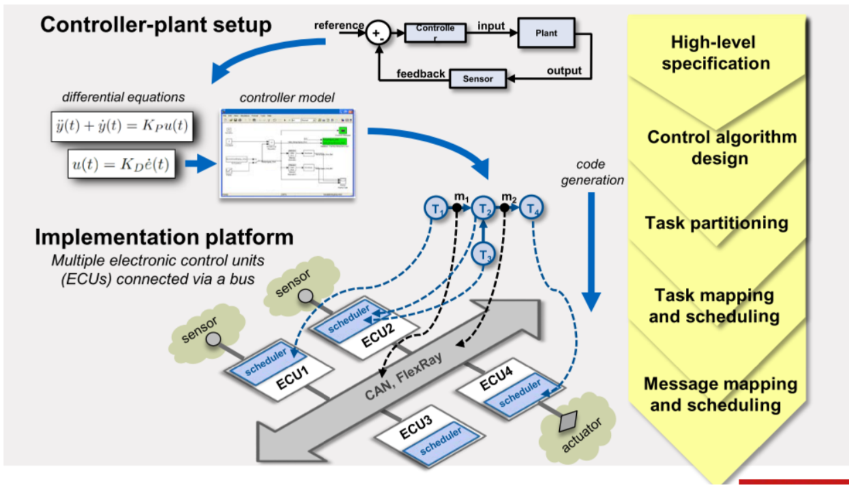
1_1_Introduction

1. Introduction

1.1. Architeture of Embedded Control systems



1.2. Design Flow of Embedded Control Systems



1.3. Difference With Traditional Control Systems

Continuous-time Control

does not consider any restriction on resource; assumes the controller to **sample infinitely often**

Discrete-time Control

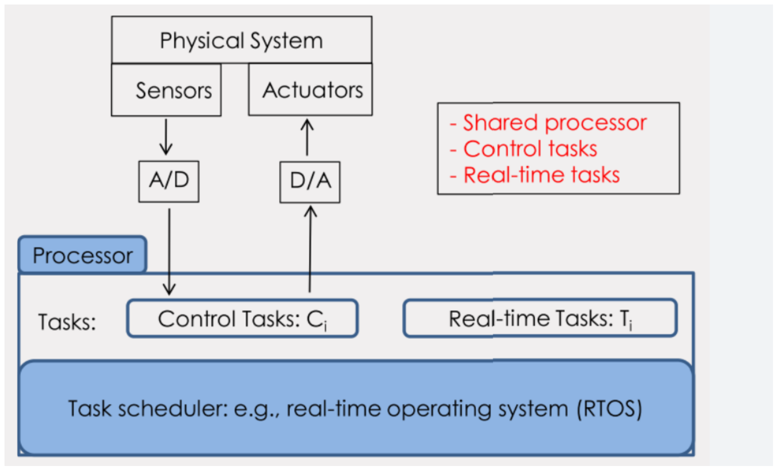
digital nature of the controller and allows for **finite length** of sampling period

Embedded Control System

Limited Resources, scheduling and sharing resource, need to **decide the sampling period**

2. Introduction of Real-time Control Systems

2.1. Architecture of Real-time Control Systems



- Control systems often runs **with other real-time tasks;** the processor is **shared by multiple** real-time and control tasks. For one processor, **only one task can be executed** at any point in time
- Real-time tasks have **timing requirements**, given by the deadlines
- Control tasks performs **sensing, computing and actuating** operations using A/D and D/A converters, sensors and actuators
- Control tasks have **speed and accuracy requirements**
- A processor runs a **real-time operating system (RTOS)** that manages the execution of the tasks according to their schedules. A **task scheduler** chooses which task to execute at a given time

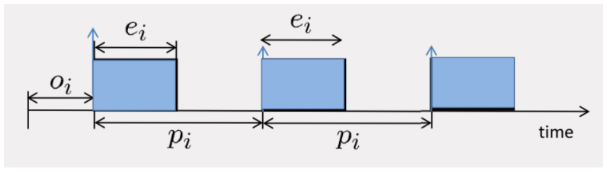
2.2. Different Fashion About Task Scheduler

2.2.1. Preliminary

An instance of a task is called **job**

2.2.2. Time-triggered

- Schedules for $T_i : \{o_i, e_i, p_i\}$
  - o_i task-offset
  - e_i worst-case execution time
  - p_i task period



- A **periodic dispatcher** is used to trigger the tasks
- Start time of k^{th} job of a periodic time-triggered task T_i is given by

$$t_i(k) = o_i + k \times p_i$$

- Finish time of k^{th} job of a periodic time-triggered task T_i is given by:

$$t_i(k) = o_i + k \times p_i + e_i$$

<u>**PROS**</u>

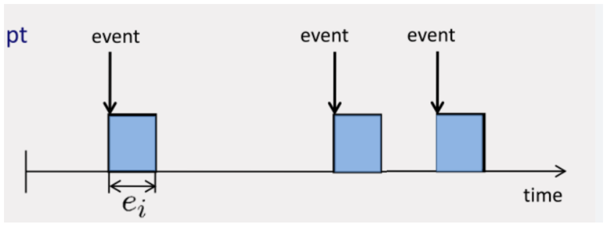
time-predictable

<u>**CONS**</u>

inflexible: the task schedules are pre-defined and poor resource utilization **when the jobs are not ready in the allocated slots** (and the processor goes **idle**)

2.2.3. Event-triggered

- The task is triggered in an event-driven fashion;
  - event can be a processor interrupt or an external signal (e.g., voltage pulse, switch, interrupt)



- $T_i : \{e_i\}$
  - e_i : worst-case execution time (WCET)

<u>**PROS**</u>

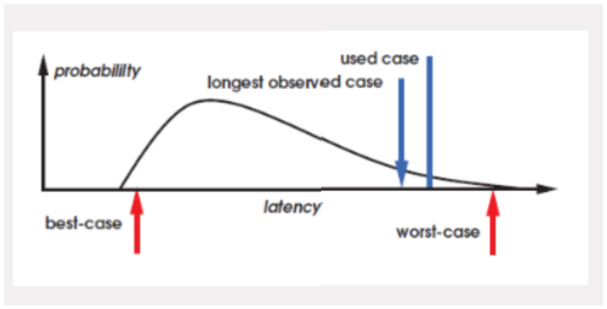
Better resource utilization and flexible

<u>**CONS**</u>

Not time-predictable

2.3. Worst-case execution time (WCET)

the **upper bound** on the processor time it takes to execute the task **without any interfering** tasks (alone on the processor)

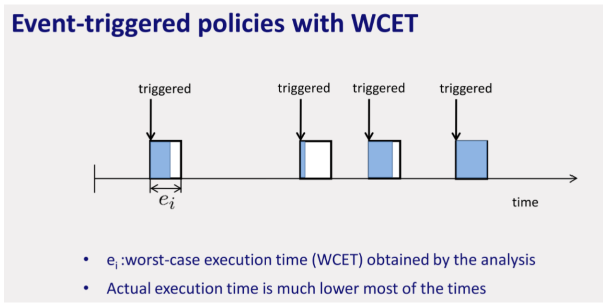


2.3.1 Static Analysis

an analysis tool **takes the source code of a task**. Automatically and formally find the longest execution time

2.3.2. Pessimism

- Actual WCET is **much shorter than the analytical WCET**, no way to formally prove it
- Moreover, WCET **happens rarely**, the average case execution time is much lower

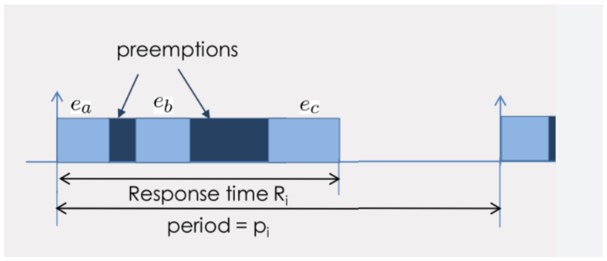


Event-triggered policies with WCET

- e_i :worst-case execution time (WCET) obtained by the analysis
- Actual execution time is much lower most of the times

2.4. Preemption

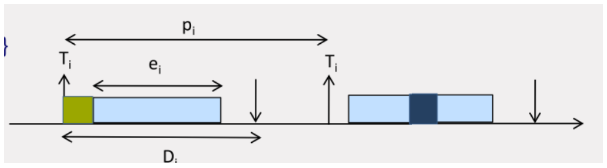
when preempted: the processor either **runs another higher priority task** or **be idle** (for some other reason such as heating and thermal problems)



2.5. Model of Real-time Tasks

Periodic real-time tasks $T_i : p_i, D_i, e_i$

- Period p_i
- Relative deadline D_i
- WCET e_i



2.5. Processor Utilization

- Often, a design criteria is $U \leq U_{limit}$
- Total processor utilization: $U = \sum_i \frac{e_i}{p_i}$

3. Co-design problem in Real-time Control

3.1. Schedulability

- A task set is **schedulable if and only if** all jobs of all task meet their relative deadline D_i, i.e. $R_i \leq D_i, \forall i$
- A system is **not schedulable** if the scheduler will not find a way to switch between the tasks such that all the deadlines are met

3.1.1. Test of Schedulability

- The test is **sufficient** if, when it answers "Yes", all deadlines will be met "**no negative positive**"
- The test is **necessary** if, when it answers "No", there really is a situation where deadlines could be missed "**no positive negative**"
- The test is **exact** if it is both sufficient and necessary
- A **sufficient test is an absolute requirement** and one likes it to be as close to necessary as possible (known as tightness of an analysis)

3.2. The critical instant

Critical Instant

In the **single processor case**, the **worst workload situation**, from a schedulability perspective, occurs when all tasks want to **start their execution at the same time instant**

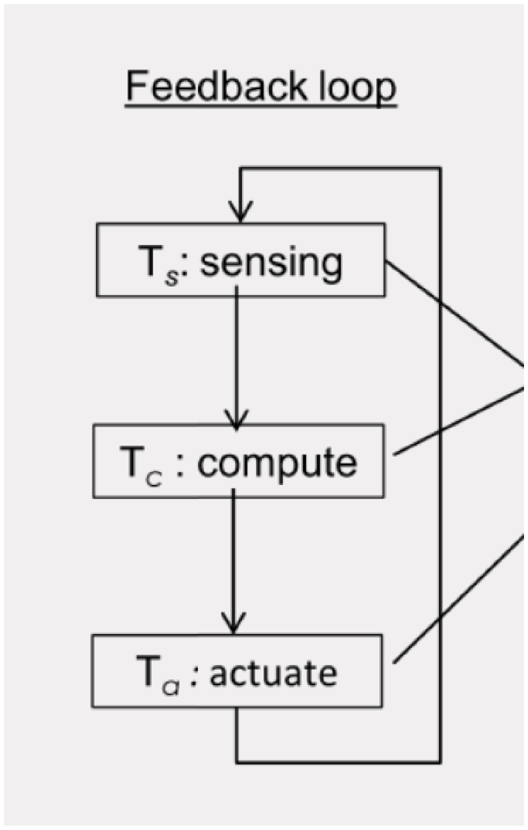
- If we can show that the task set **is schedulable in Critical Instant**, it will also be schedulable in other situations
- If we can show that the task set **is schedulable for the worst-case execution times (WCET)**, then the task set will also be schedulable if the actual execution times are shorter

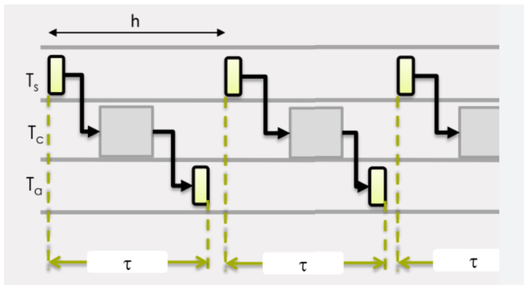
3.3. Model of Control Tasks

- A control application performs sensing, computing and actuating operations **in sequence periodically**
- T_s : **sensor task** to perform **measure** $x[k]$ using sensors. Typically, A/D conversion and filtering are performed in this task
- T_c : **controller task** to perform **compute input signal** $u[k]$. The execution time of this task depends on the complexity of the control algorithm
- T_a : **actuator task** to **perform actuate** the system by $u[k]$. Typically, D/A conversion and post-processing is done in this task
- The interval between two consecutive activation of T_s is called **sampling period** h
- The interval between the **start** of sensor task T_s and the **end** of the corresponding actuation task T_a is called the **sensor-**

to-actuator delay τ, and it was always specified before design

- Sensor-to-actuator delay = τ = $e_s + D_c + e_a$





3.4. Control Task with Constant Delay

- T_s : $\{0, e_s, h\}$
- T_a : $\{D_c + e_s, e_a, h\}$
- T_c is executed in between T_s and T_a, deadline $D_c = \tau - e_s - e_a$, period= h,WCET= e_c
- Response time of T_c is R_c ; all tasks including T_c should be scheduled such that the worst-case $R_c \leq D_c$

3.5. Overall of Co-design Problem

Co-design problem

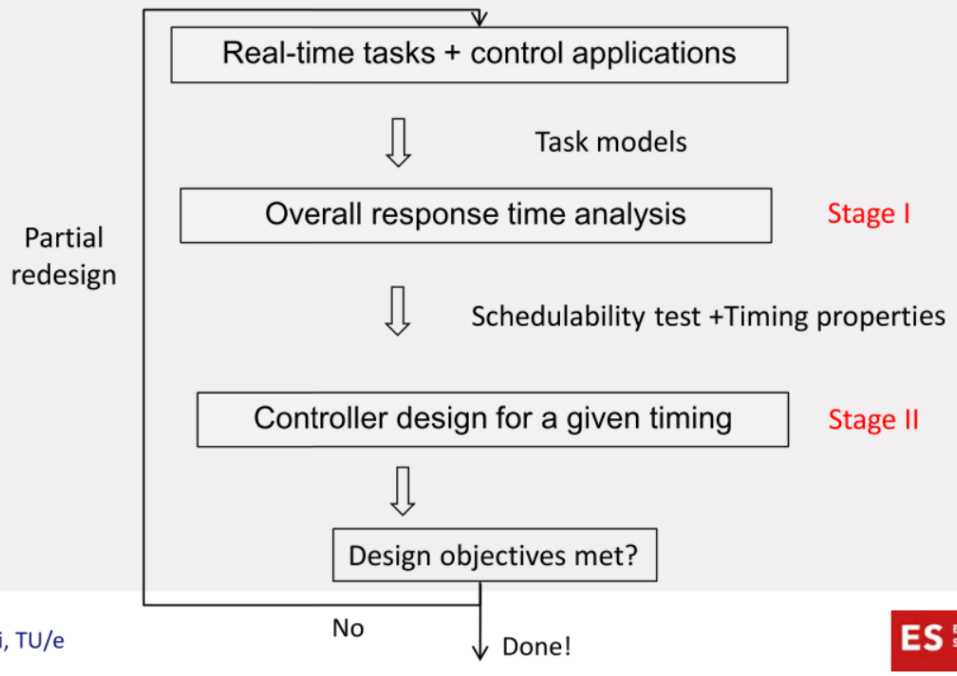
We have the following set of tasks running on a processor

- A set of *n* real-time tasks which are characterized by T_i:{p_i, D_i, e_i}, where i=1,2…n;
- A control application, which is partitioned into T_s, T_c and T_a, is responsible to control a dynamical system

$$\dot{x} = Ax + Bu;\ y = Cx$$

- T_s and T_a are time-triggered and have negligible execution time; offset between them is D_c;
- T_c:{h, D_c, e_c}, $D_c \leq h$ which computes the control algorithm
- Design goals are
  - Schedule tasks such that (i) $R_i \leq D_i$ for i=1,2…n (ii) $R_c \leq D_c$
  - Design feedback and feed forward gains K and F in the following control law $u = Kx + Fr$ such that y → r as time → ∞

Co-design steps



Real-time tasks + control applications
Task models
Overall response time analysis
Stage I
Partial redesign
Schedulability test +Timing properties
Controller design for a given timing
Stage II
Design objectives met?
No
Done!

47 Dip Goswami, TU/e