Synchronous languages

Lecture 1:
Embedded and real-time systems

ENSEEIHT 3A – parcours E&L
2021/2022

Frédéric Boniol (ONERA)
frederic.boniol@onera.fr

1. About embedded systems…

1.1. Some general definitions:
- What is an "embedded" system?
- What is a "real-time" system?

1.2. An exemple: the flight control system

1.3. Generalisation

1.4. Question: do we need specific languages for programming embedded software?

1. About embedded systems…

1.1. Some general definitions:

- What is an "embedded" system?
- What is a "real-time" system?

1.2. An exemple: the flight control system

1.3. Generalisation

1.4. Question: do we need specific languages for programming embedded software?

What is a digital embedded system?

- Information processing system embedded into a larger product [Pr. Marwedel, Dortmund Univ]

  – Composed of
    - Software components
    - Running on Hardware Components

  – Integrated with the physical process to be controlled

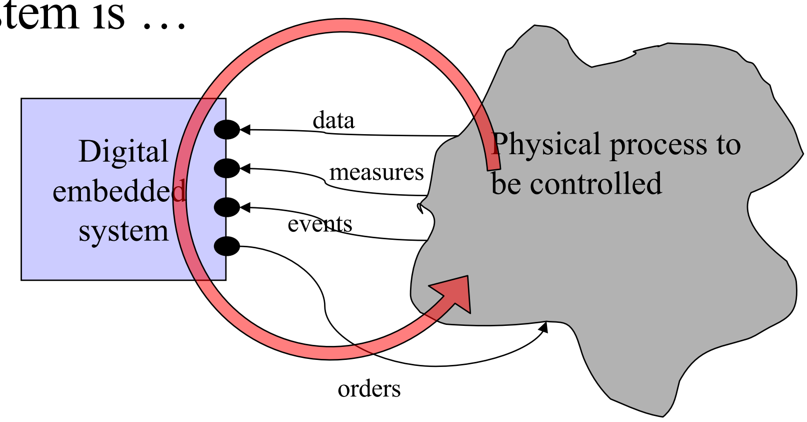
⇒ The main technical problem is

– managing time and concurrency

in the computational part of the embedded system.

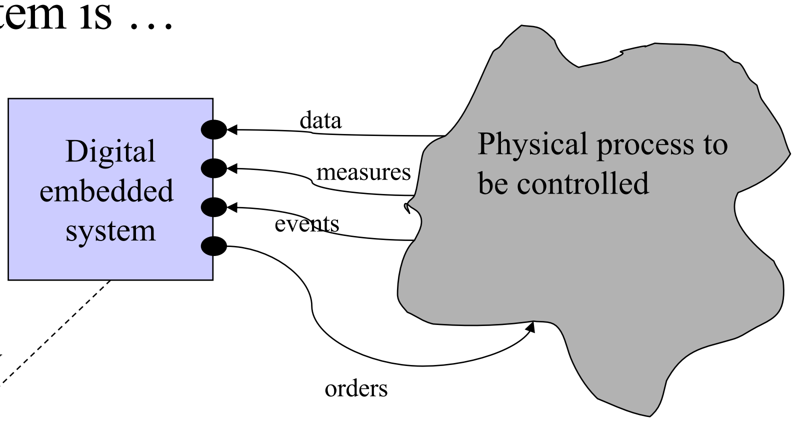
An embedded system is …

… strongly connected in a closed loop with the process to be controlled



An embedded system is …

… strongly connected in a closed loop with the process to be controlled

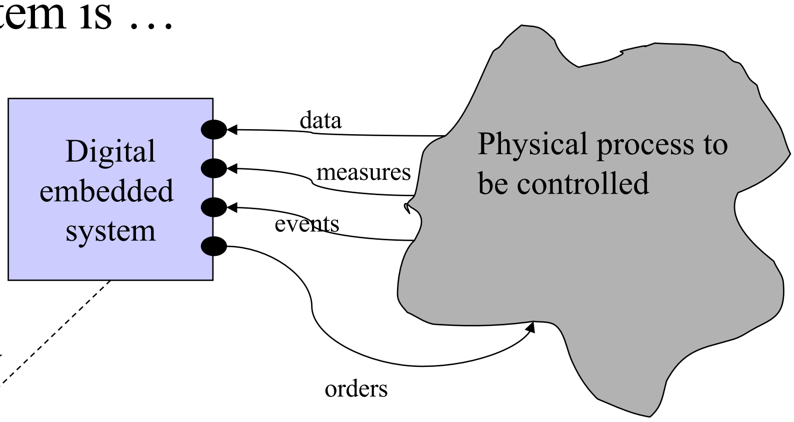


… may be implemented by

- an integrated circuit (ASIC, FPGA)
- a sequential SW on single-core processor
- **a multi-threaded SW on a single-core or multi-core processor**
- …

An embedded system is …

… strongly connected in a closed loop with the process to be controlled



… may be implemented by

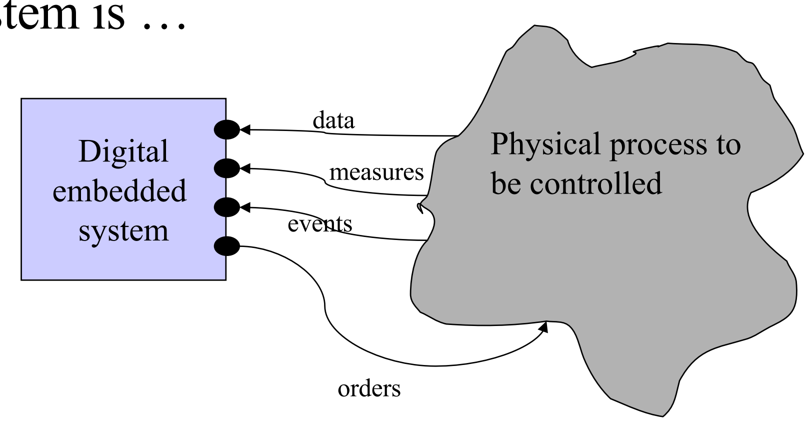
- an integrated circuit (ASIC, FPGA)
- a sequential SW on single-core processor
- a **multi-threaded SW on a single-core or multi-core processor**
- …

⇒ Characteristics of Embedded Systems

- Must be dependable
- Must be efficient (energy, weight, cost, etc.)
- **Must meet real-time constraints**

An embedded system is …

… a real time system



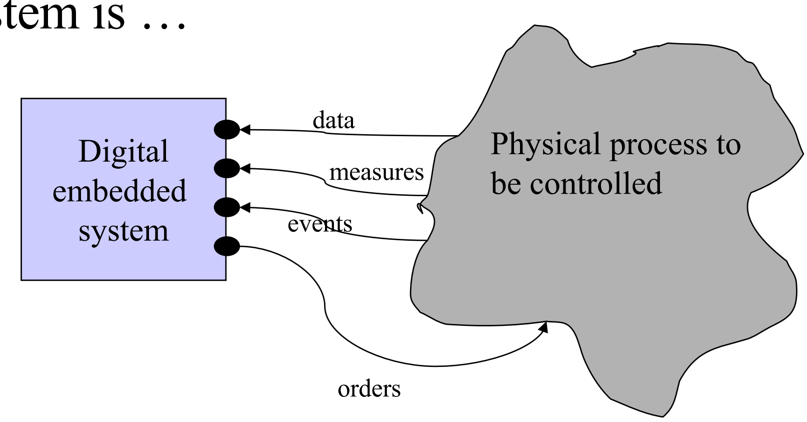
⇒ Characteristics of Embedded Systems

- Must be dependable
- Must be efficient (energy, weight, cost, etc.)
- **Must meet real-time constraints**

Must react to stimuli from the controlled process (or the operator) within the time interval dictated by the environment.

An embedded system is …

… a real time system



Must execute at a pace compliant with the dynamic of the process.

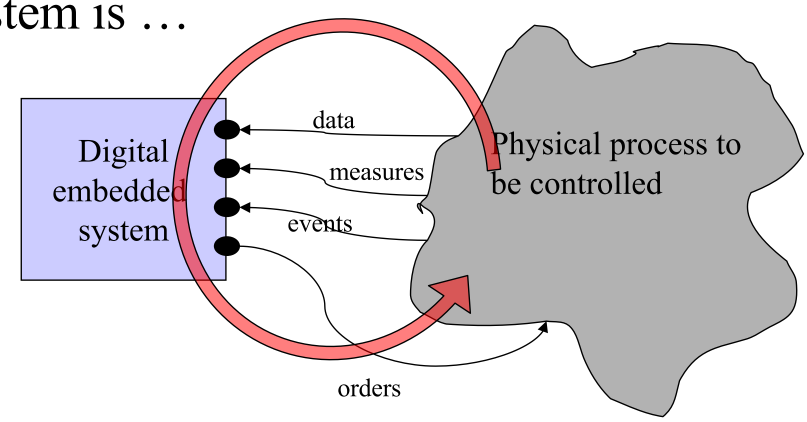
Must react to stimuli from the controlled process (or the operator) within the time interval dictated by the environment?

⇒ Characteristics of Embedded Systems

- Must be dependable
- Must be efficient (energy, weight, cost, etc.)
- **Must meet real-time constraints**

An embedded system is …

… a real time system



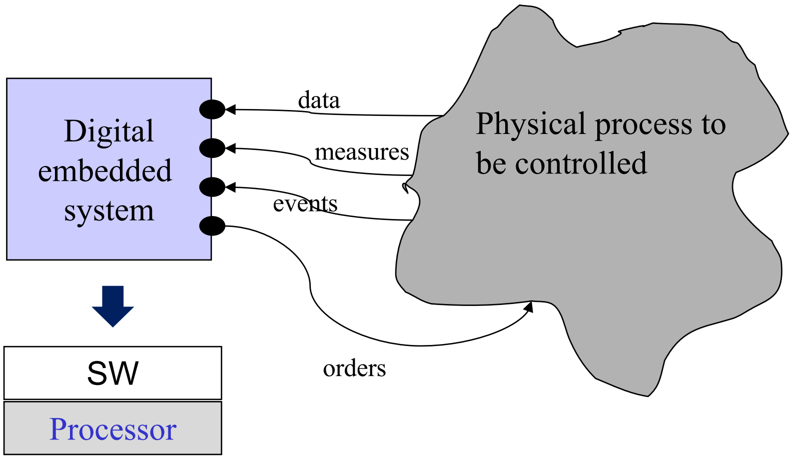
Two types of timing constraints:
- Periodicity of control loop.
- Latency of the control loop.

Must react to stimuli from the controlled process (or the operator) within the time interval dictated by the environment.

⇒ Characteristics of Embedded Systems

- Must be dependable
- Must be efficient (energy, weight, cost, etc.)
- **Must meet real-time constraints**

The question…



Implemented by a mutli-threaded SW on a single-core or multi-core processor.

Two types of timing constraints:
- Periodicity of control loop.
- Latency of the control loop.

Question: what kind of mechanisms do we need for

- Programming the SW?
- Mastering the execution of the SW?

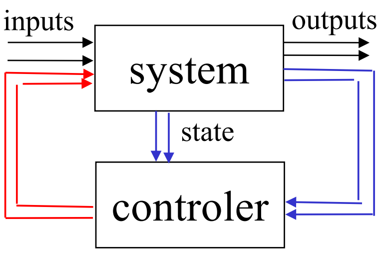
Example…

Command: Laws which govern the dynamical evolution of a system

- Command of actuators regarding the sensors
- In continuous time

Examples:

- Regulation of a liquid level between thresholds
- Command of flight surfaces



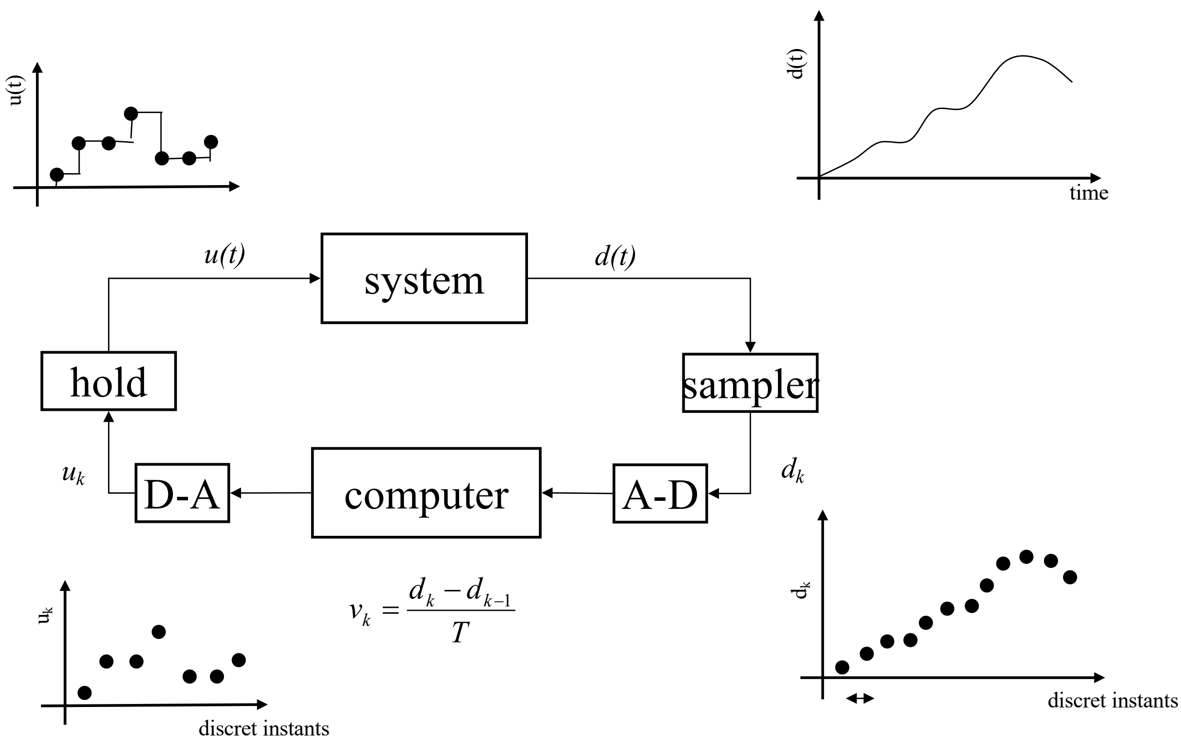
State equations

$$\begin{cases} dx = f(x,u) \\ y = h(x) \end{cases}$$

x internal state,
y output,
u input

Command a system = make the system evolve in order to reach a particular configuration or to follow a given trajectory

Example…



$$v_k = \frac{d_k - d_{k-1}}{T}$$

Application areas

- **Space systems**
  - Attitude and Orbit control (satellite)
  - Payload control (satellite)
  - Trajectory control (launcher)
  - Docking control (to the space station)
- **Avionic systems**
  - Flight control system
  - Flight guidance system
  - …
- **Automotive systems**
  - Engine control
  - …
- **Robotics**
- **Medical systems**
  - Pacemaker
  - …

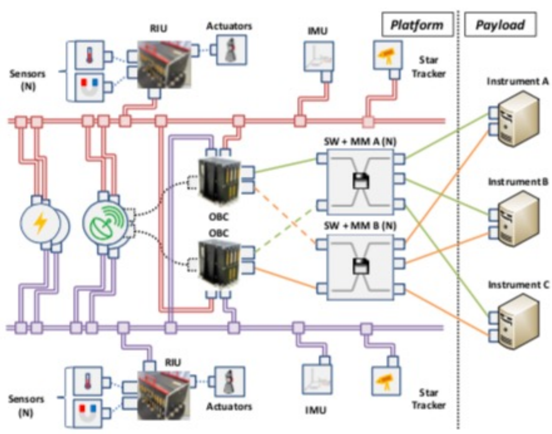
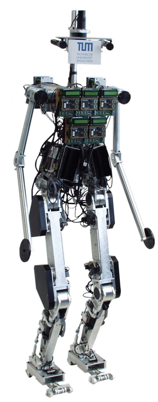


Figure 1. Traditional Satellite Network Topology



1. About embedded systems…

1.1. Some general definitions:
- What is an "embedded" system?
- What is a "real-time" system?

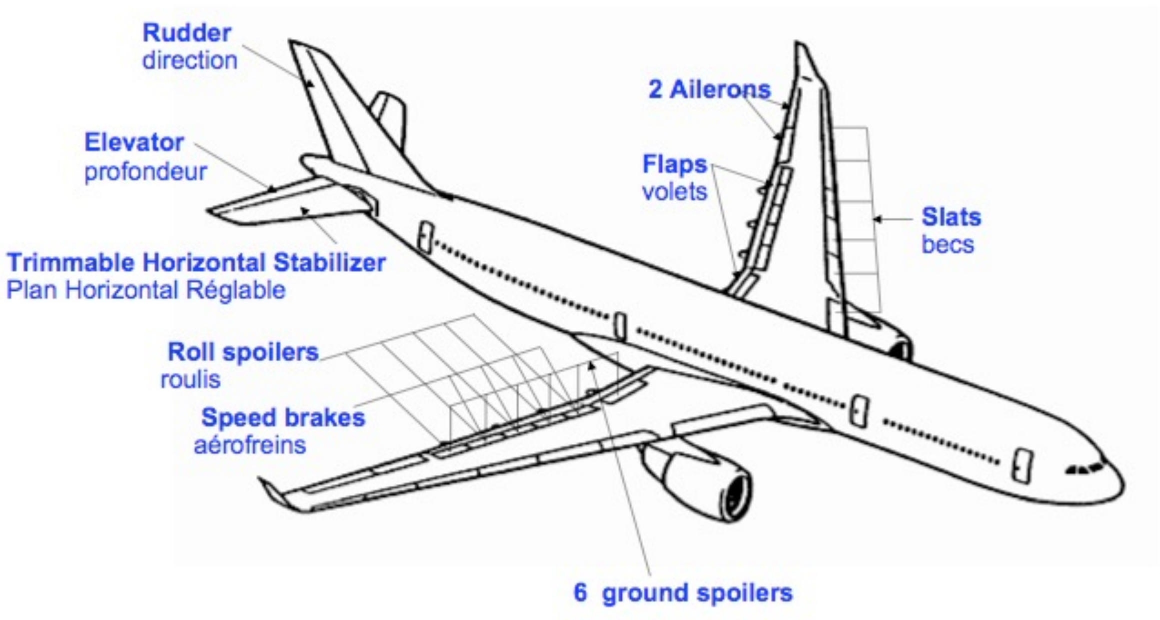
1.2. An exemple: the flight control system

1.3. Generalisation

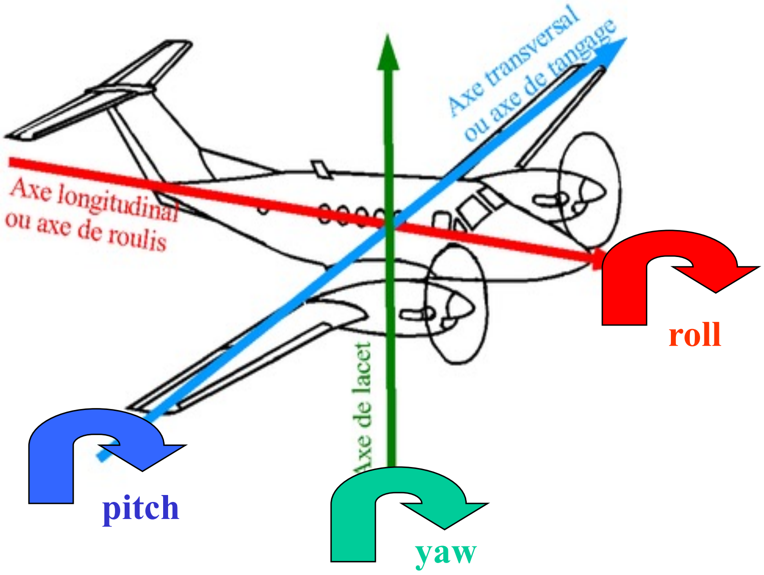
1.4. Question: do we need specific languages for programming embedded software?

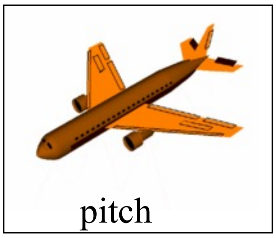
Embedded Systems: an example

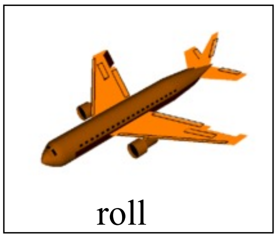
- The A330/A340 Flight control system

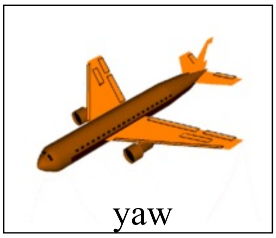


Flight control



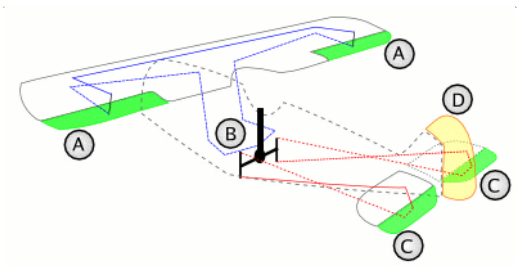






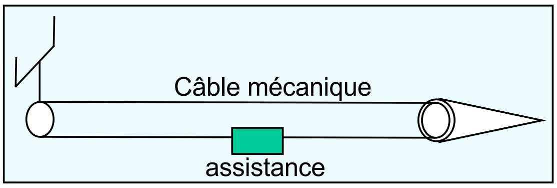
=> The Flight Control System – V1

- The **flight control system** is the set of elements between the stick and the surfaces which aim at controlling the attitude, the trajectory and the speed of the aircraft.

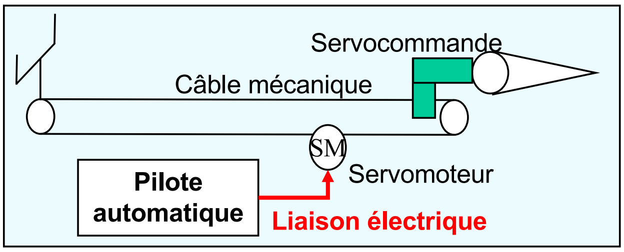


- <u>In the first generations </u>(before A320): the system is composed of:
  - piloting elements: stick, pedals, mechanical elements (cables…)
  - surfaces

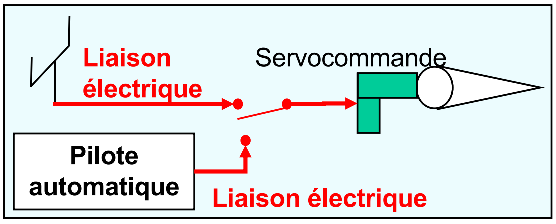
=> The Flight Control System - Evolution



V0: fully mechanical flight control system

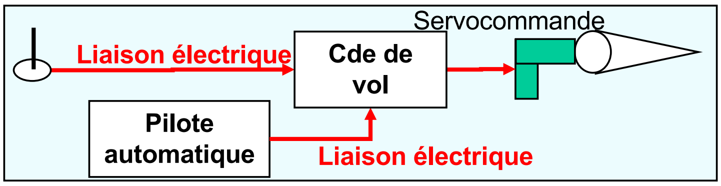


V1: mechnical flight control system + digital flight guidance (before A320)



V1.5 (never implemented)

> **Remark** :
> Concorde(1969) already had a digital fight control system

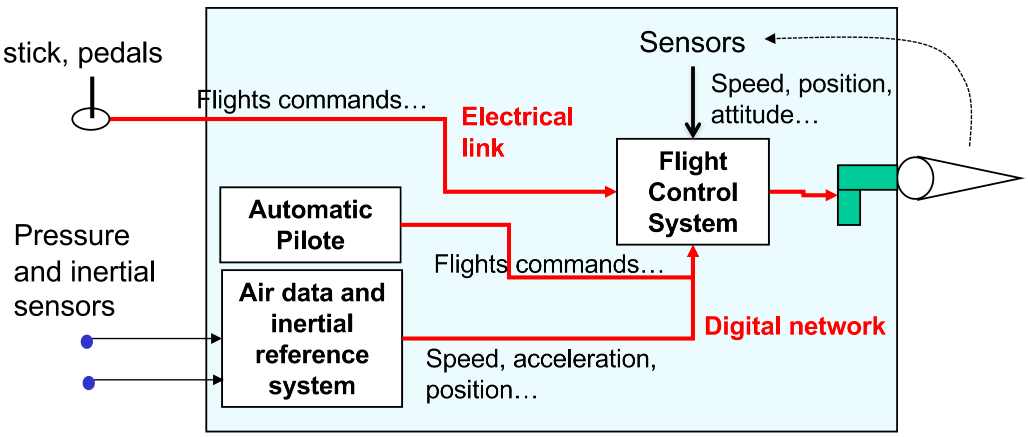


V2: digital flight control system (A320 …)

=> The Flight Control System – V2

<u>Second generation</u> (after A320): introduction of several digital embedded systems

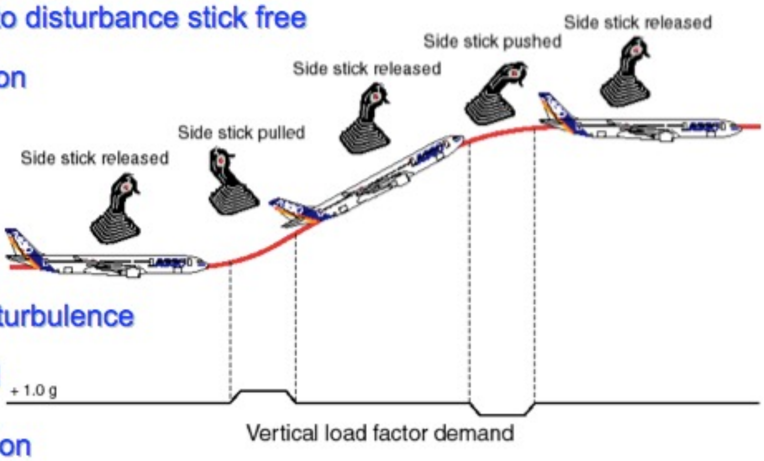
- To measure the movement of the aircraft
- To control its trajectory
- To control the flight surfaces



=> The Flight Control System – V2

Role of the avionic software

- ✓ Automatic pitch trim
- ✓ A/C response (almost) unaffected by speed, weight or centre of gravity location
- ✓ Bank angle resistance to disturbance stick free
- ✓ Efficient turn coordination
- ✓ Dutch roll damping
- ✓ Side-slip minimization
- ✓Accurate flying
- ✓ Passengers comfort in turbulence
- ✓ Reduced pilot workload
- ✓ Flight envelope protection



=> The Flight Control System – V2

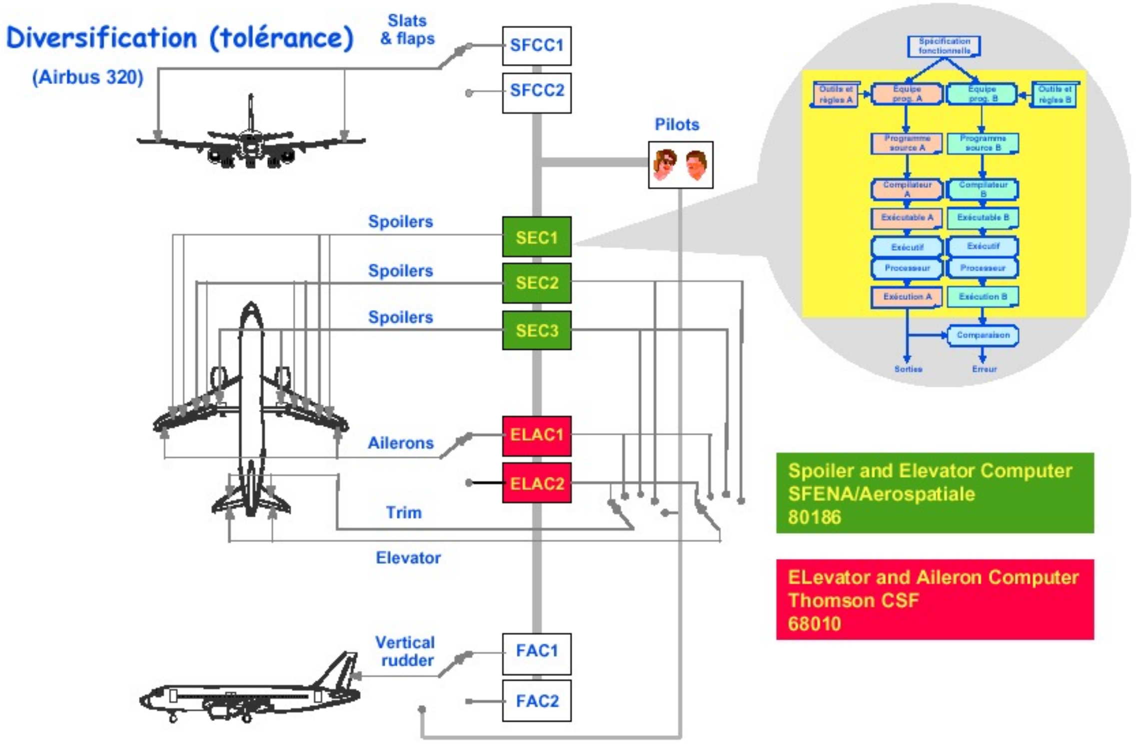
Role of the avionic software

- To monitor the health of the system,
- To reconfigure the system in case of abnormal behaviour

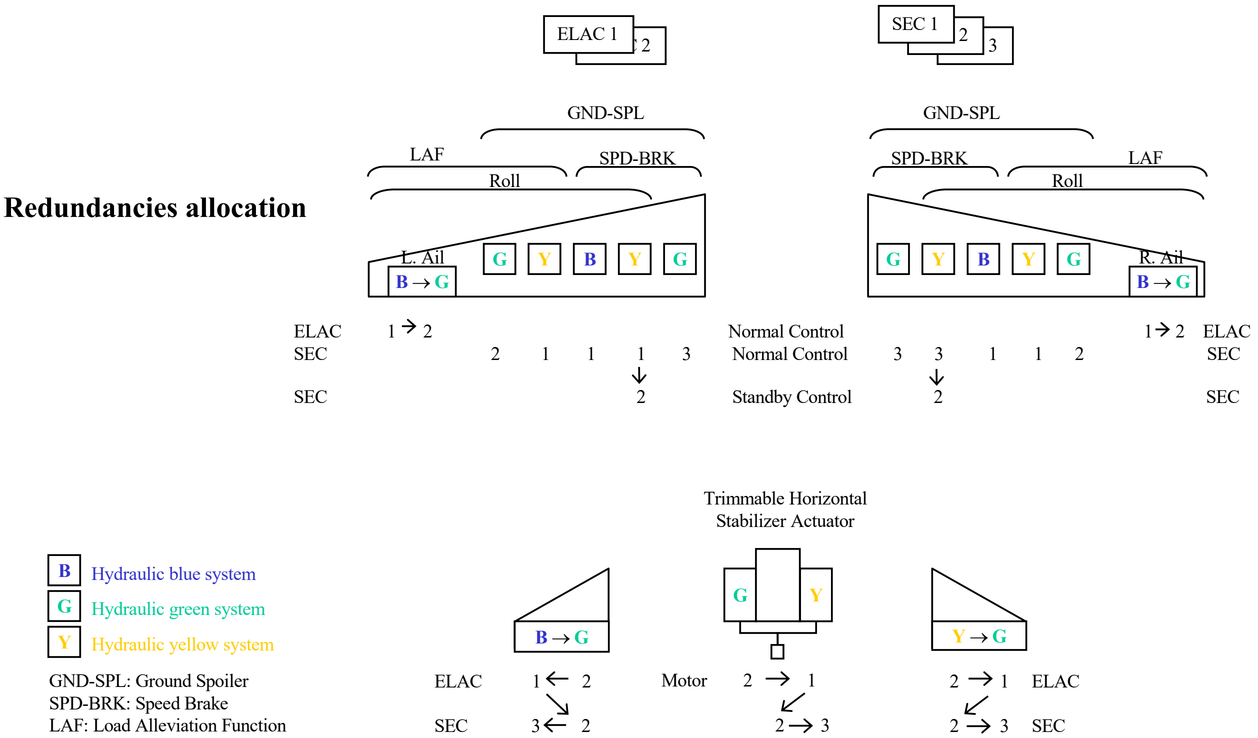
=> Example: A320 Flight control system

=> The Flight Control System – A320

- The A320 Flight control system

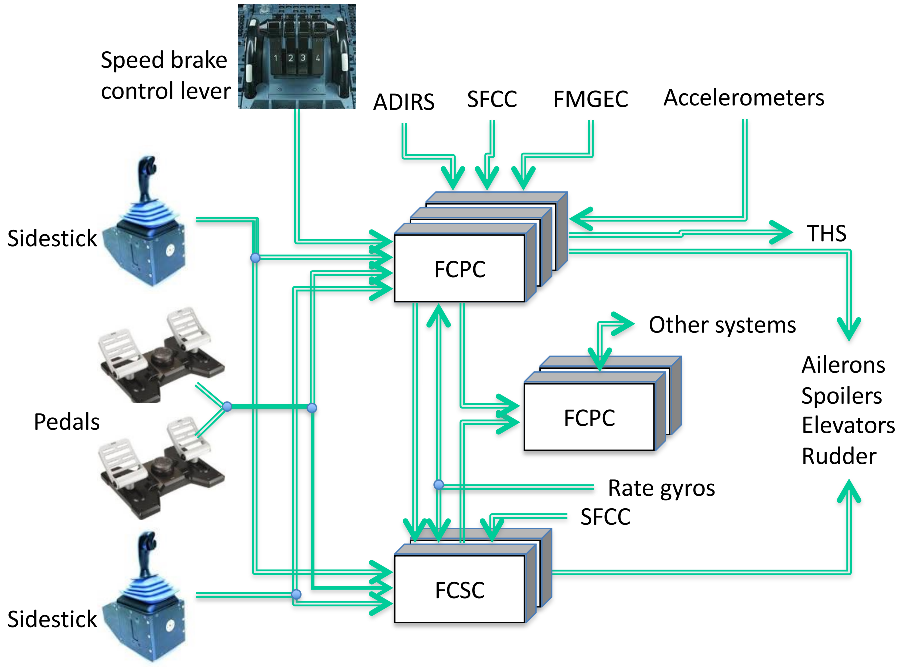


=> The Flight Control System – A320



=> The Flight Control System – A340

- The A330/A340 Flight control system



1. About embedded systems…

1.1. Some general definitions:
- What is an "embedded" system?
- What is a "real-time" system?

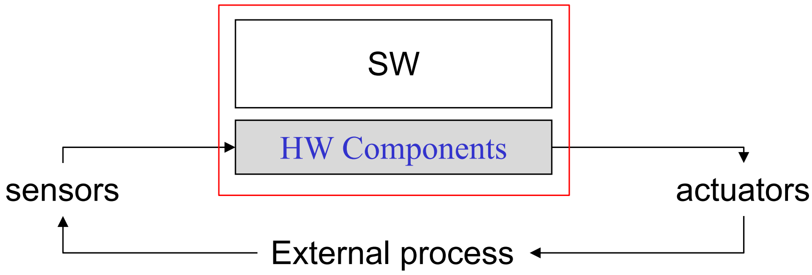
1.2. An exemple: the flight control system

1.3. Generalisation

1.4. Question: do we need specific languages for programming embedded software?

Generalisation

- Embedded systems are composed of



- Embedded systems = real-time systems

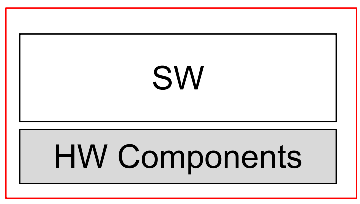
- Two types of real time systems
  – Soft real-time: can miss some timing constraints (but not too often)
  – Hard real-time: missing a deadline is considered as a catastrophic failure (e.g., braking system)

=> Flight control system, Attitude and Orbit Control System… are hard real-time

Generalisation

- Role of software
  – To implement the functional part of the system
    - Control laws
    - Reconfiguration rules…

    => Functional SW



  – To provide a way for the functional SW to (easily) access the HW components
    - IO drivers
    - Memory controllers…

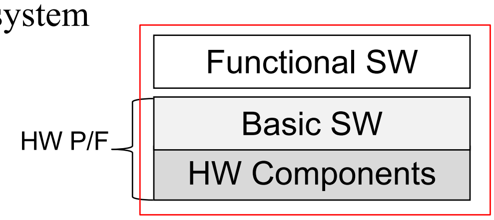
    ⇒Basic software

⇒ An embedded system is a mixture functional and basic software.

Generalisation

- Role of software
  - To implement the functional part of the system
    - Control laws
    - Reconfiguration rules…

    => Functional SW



  - To provide a way for the functional SW to (easily) access the HW components
    - IO drivers
    - Memory controllers…

    ⇒Basic software

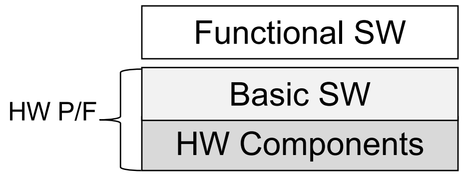
⇒ An embedded system is a mixture functional and basic software.

1. About embedded systems…

Generalisation

⇒ Question:

- **Do we need any specific languages** for programming functional SW?



⇒ It depends on the complexity of the SW
  - ⇒ Yes for today (complex) systems
  - ⇒ No for previous (simple) systems

⇒ Yes for hard real-time systems:
  - ⇒ In order to manage time and concurrency in an accurate way

⇒ What kind of languages?

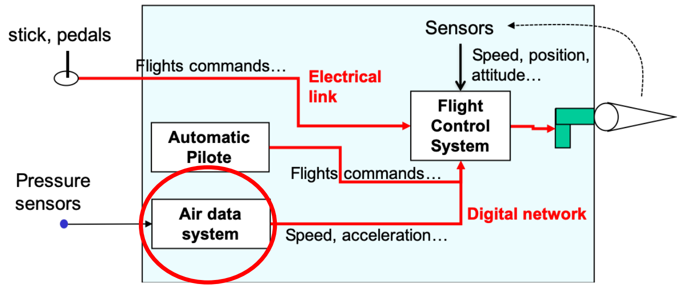
2. Question: do we need specific languages for programming functional SW

- The single-threaded case
- The multi-threaded case

Example 1: single-period system

Example 1: AD system (Air Data)

- Computes the Mach number and the altitude of an aircraft from pressure information

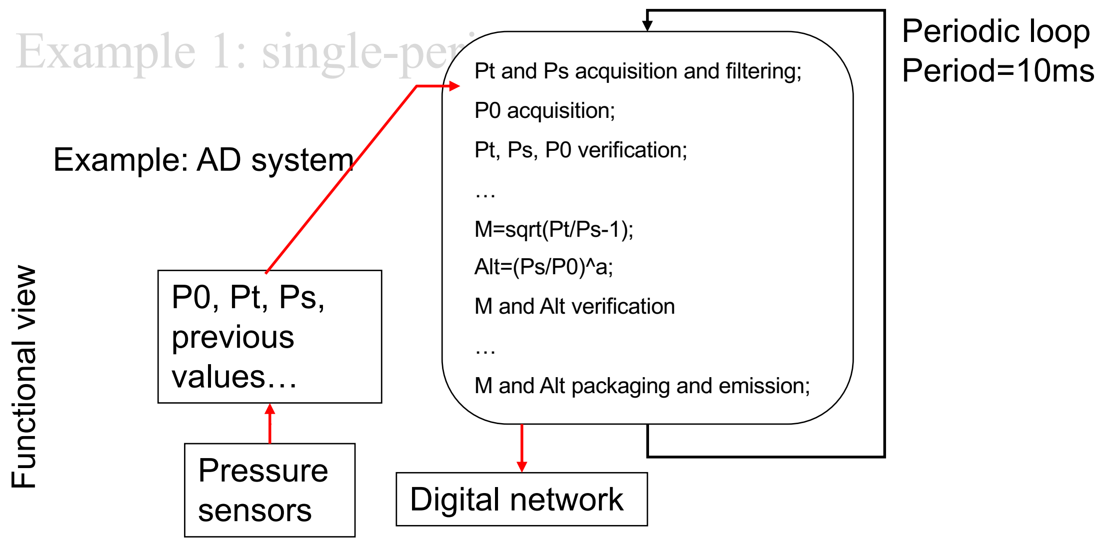


⇒ Simple system (single-period system)

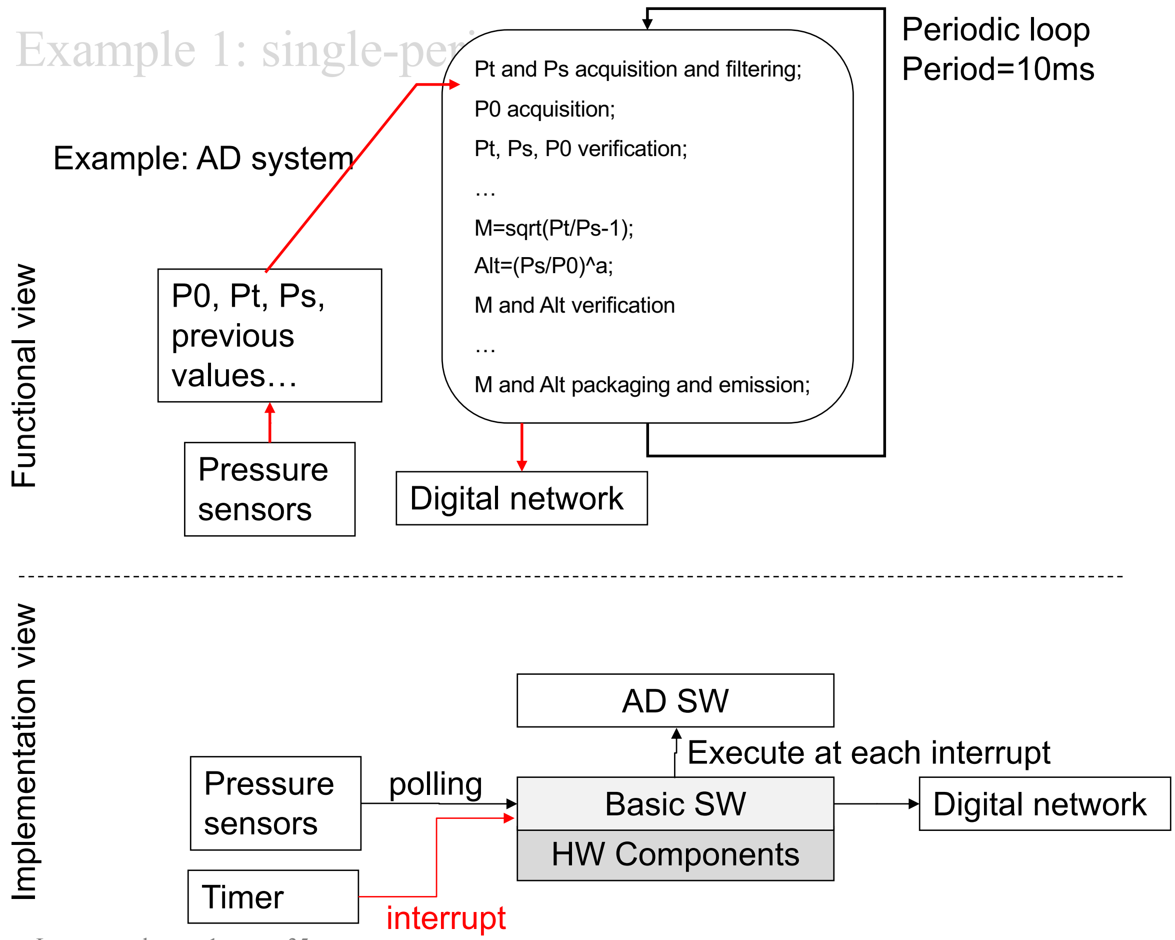
- composed of only one sequential code
- periodically executed
- only one period (10 ms)
- on a single-core HW platform

(Case of the first generation of avionic systems (up to A320))

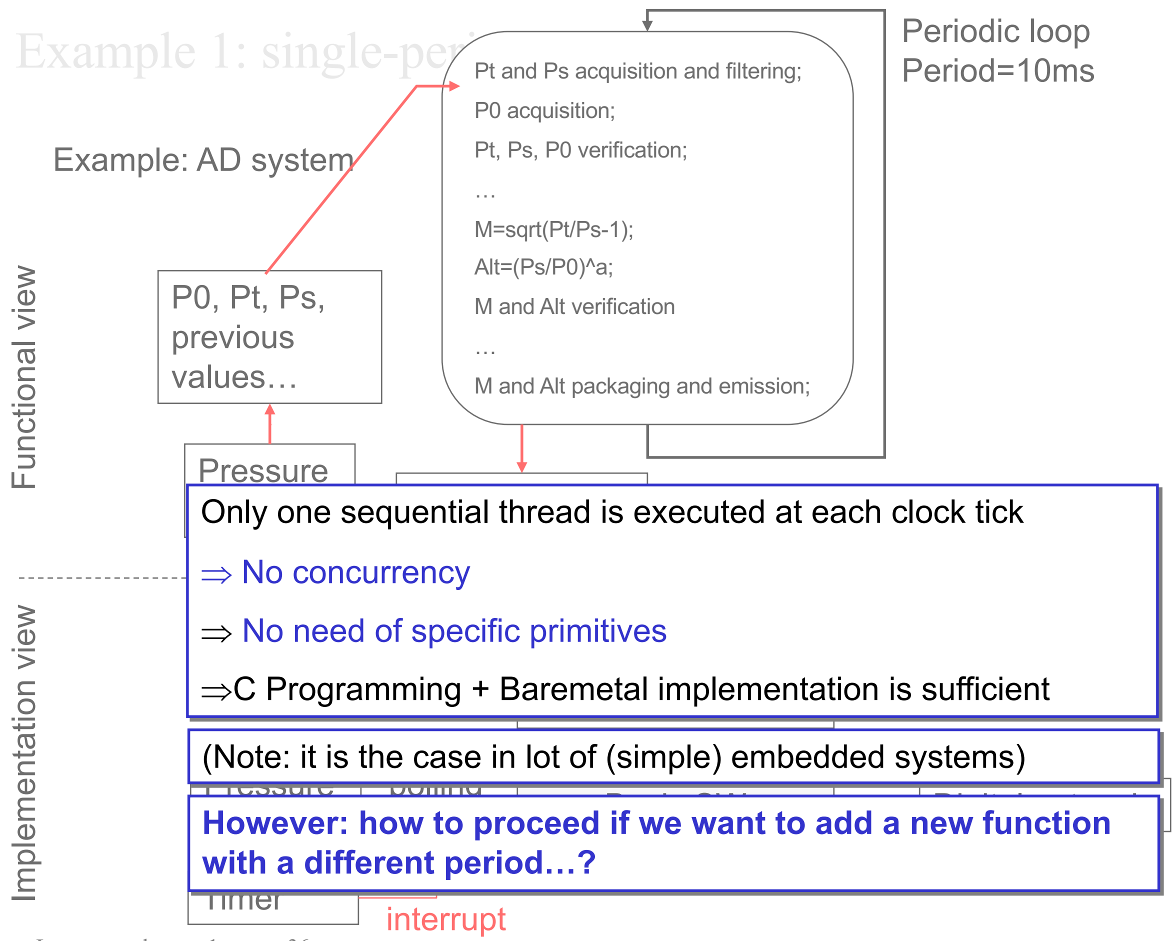
Example 1: single-peri



Example 1: single-peri



Example 1: single-peri



Only one sequential thread is executed at each clock tick

⇒ No concurrency

⇒ No need of specific primitives

⇒C Programming + Baremetal implementation is sufficient

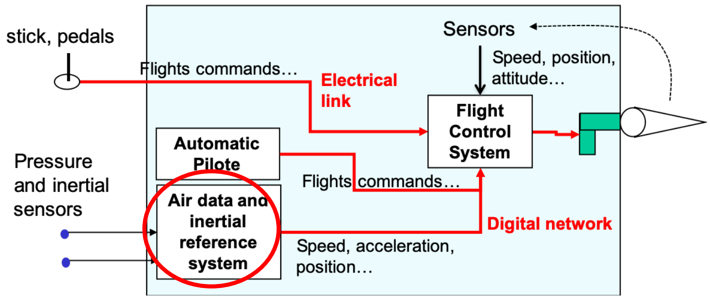
(Note: it is the case in lot of (simple) embedded systems)

However: how to proceed if we want to add a new function with a different period…?

Example 2: multi-period system

Example 2: ADIR system (Air Data and Inertial)

- Computes the Mach number and the altitude of an aircraft from pressure information
- Computes the position of the aircraft from gyroscope information
- Check correctness



⇒ Multi-threaded system

- composed of several functional (sequential) activities
- each activity is periodic, but they may have different period

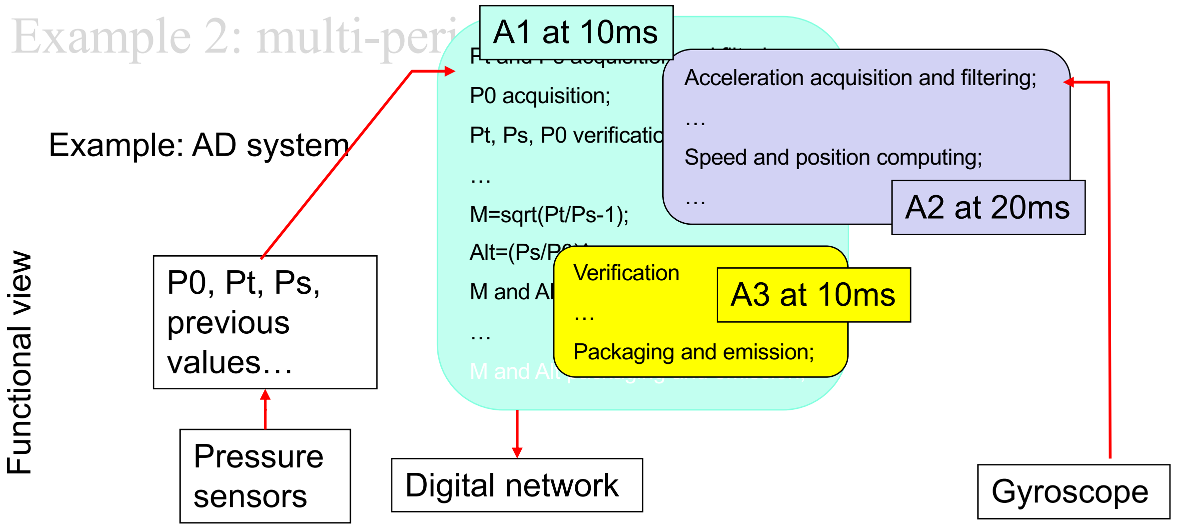
=> they can ask for processor at the same time

=> Concurrency problem

– how to manage execution of concurrent activities?

(Case of the second generation of avionic systems (after A340))

Example 2: multi-peri



Example 2: multi-peri

A1 at 10ms

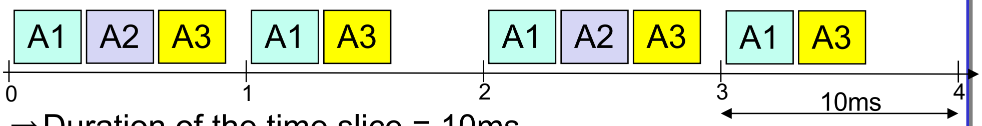
P0 acquisition;

Acceleration acquisition and filtering;

at 20ms

Concurency between three activities!

⇒ First solution: to define manually at design time a static sequence of time slices



⇒Duration of the time slice = 10ms

⇒At execution time, only one sequential activity is executed

⇒No more concurency at execution time

```
Every clock tick
  run A1;
  If (tick==0 mod 2)
     run A2;
  run A3;
End every;
```

⇒C Programming + Baremetal implementation is sufficient

However: not suitable for complex software

Ex: A380 flight control sowtware = 1M lines of code

Toward more complex avionic software

- Today generation of embedded software (A350, military aircraft, space vehicles…)
  - composed of
    - several periodic activities (triggered by periodic clocks)
    - several aperiodic activities (triggered by external events)

⇒Question:
- how to specify / program these activities?

⇒Two ways:

⇒Asynchronous programming =
- C programming
- Real-Time Operating System

⇒Synchronous programming =
- Specific and simpler languages
- Compiler to C implantation

3. Synchronous versus Asynchronous programming

- Challenges of asynchronous programming
- Principle of synchronous programming

Challenges of asynchronous programming

What is « asynchronism »

Asynchronism =

The threads do not share the same « time »

=> No global time

=> Durations of instructions are not defined in the semantics of the languages

=> Durations are undefined

Benefit:

Fit well with the concrete behaviour of the HW architectures.

Problem:

Concurrency is not deterministic

⇒A program can behave differently with the same inputs

⇒Increases the difficulty in mastering the behaviour of the programs

Challenges of asynchronous programming

Example:

```
Signal X : integer ;

X <- 0;
[
   X <- 1;
   X <- 2;
||
   Y <- X+1;
]
```

Asynchronism
=> Each branch run in parallel at its own pace

Asynchronous semantics => ,
several interleaving behaviour
=> several results for Y: 1, 2 or 3

⇒ Non-deterministic execution

Challenges of asynchronous programming

Conclusion:

Asynchronism is source of complexity and difficulty.
It requires to:

→ Add watchdogs in the code to control the real-time durations of the threads,

→ Add control mecanisms in the code to prevent non-determinisic behaviours

⇒ Increase in the complexity of programming languages and of the programs!

⇒ The problem comes from time

The synchronous « idea »

Idea:

Simplification of the programming model

=> Complicated details are « ignored »

=> <u>Strong synchronous assumption</u>

- All the threads share the same « time »
- **Duration of instructions = 0**

⇒Two threads beginning at the same time end at the same time

⇒They are « synchronous »

=> <u>Benefit</u>:

- determinism
- simplification of the programming model

⇒ <u>However</u>: does not fit the the concret execution

⇒hypothesis to be confirmed by comparison with the concrete world hypothèse à confirmer par confrontation avec le monde concret

Remark:

Similar to physicists' approaches when modeling complexe phenomena

=> approach by simplification (i.e., to ignore unnecessary details)

=> Here: abstraction of time!

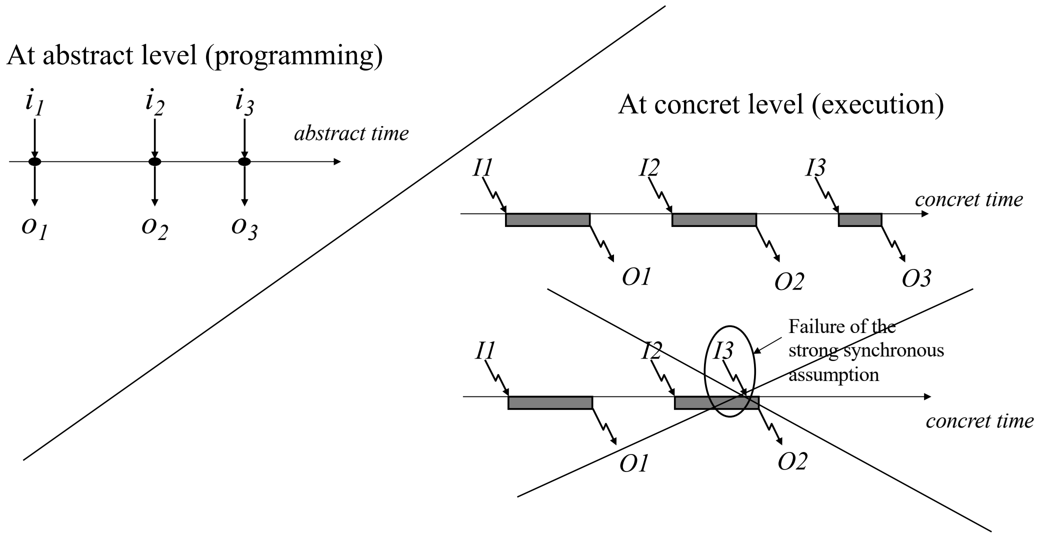
The synchronous « idea »

Strong synchronous assumption: *execution time = 0*

⇒ **Abstraction**

– At programming level, the program behaves « as if » all the durations are zéro.

– At concret level, one has to check that the real execution time is negligible, ie., smaller than the period of the threads.



The synchronous « idea »

Example:

```
Signal X : integer combine with + ;

X <- 0;
[
   X <- 1;
   X <- 2;
||
   Y <- X+1;
]
```

Synchronism
=> All the branches run simultaneously and instantaneously

Synchronous semantics => ,

Only one interleaving behaviour
=> Only one result for Y = 4
=> Single assignation
=> Deterministic semantics

The synchronous « idea »

Benefit example:

```
Signal X : integer combine with + ;
X <- 1;
X <- 2;
```

congruent to

```
Var X : integer combine with + ;
X <- 3;
```

=> It allows code optimisation at compile time

Two main synchronous languages

- LUSTRE:
  - Data flow programming
  - Equational style
- ESTEREL:
  - Event flow programming
  - Imperative style

LUSTRE in a nutshell

LUSTRE programm =

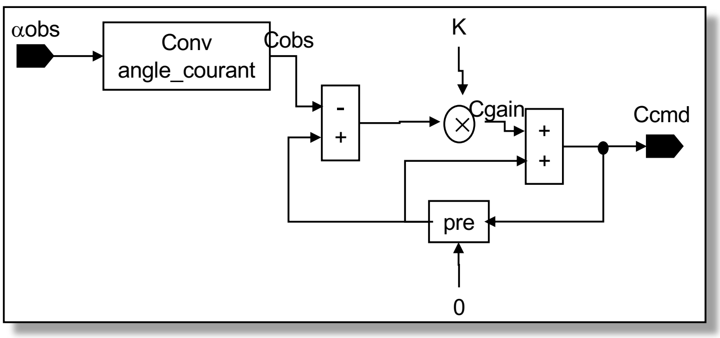
set of equations betweens « data flows »

- Academic origin: Vérimag (Grenoble)
- Industrial version: SCADE - Esterel Technologie
- Industrial users: Airbus, Dassault, Continental…

Lustre

```
Node asserv (αobs : real)
returns (Ccmd : real);
var Cobs, Cgain : real;
let
   Ccmd = (0 → pre(Ccmd)) + Cgain ;
   Cgain = K * ((0 → pre(Ccmd)) - Cobs;
   Cobs = Conv_angle_courant(αobs);
tel.
```

SCADE

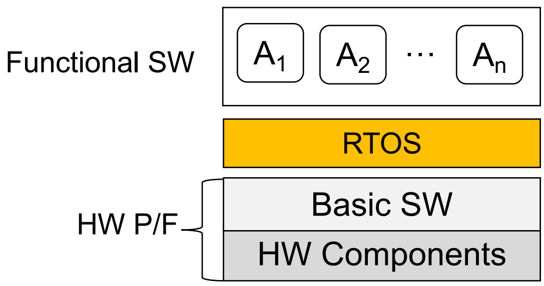


4. Summary

- What are we talking about
- The synchronous idea

What are we talking about

Embedded system =



⇒ Question: how to program A_1, A_2… A_n ?

⇒ Two ways

- Asynchronous way:
  A1, … An = C programs + RTOS
- Synchronous way
  A1, … An = one Lustre model

The synchronous idea

- Idea:
  - Take into account the time in the semantics of the programming language
  - abstraction of the time to simplify the programs

  ⇒Duration = 0

  ⇒LUSTRE

End of lecture 1

⇒Next lecture: LUSTRE (simplified version)