REAL TIME SYSTEMS

2. The Real-Time kernel OSEK/VDX

The OSEK context

Context: the embedded "electronics" in vehicles

real-time **constraints (hard and soft)**

high safety

minimal hardware support (little RAM, 8- and 16-bit ECUs)

distributed architecture **around ≠ networks (CAN, VAN, LIN ...)**

cross-cutting functions (interoperability of subsystems)

flexibility of the architecture (addition of functions, portability **and** reusability **of software functions)**

OSEK / VDX: history

- Proposal of the OSEK group, consisting of:
  - manufacturers (BMW, DaimlerChrysler, Renault, PSA, etc.)
  - equipment manufacturers (Bosch, Siemens, etc.)
  - academics (Univ. Karlsruhe)
- Merger of the OSEK (German) and VDX (GIE PSA-Renault) projects
- Includes the European project MODISTARC (certification process of compliant implementations)
- Work started in 1995

The reference website: http://www.osek-vdx.org

Main OSEK OS services

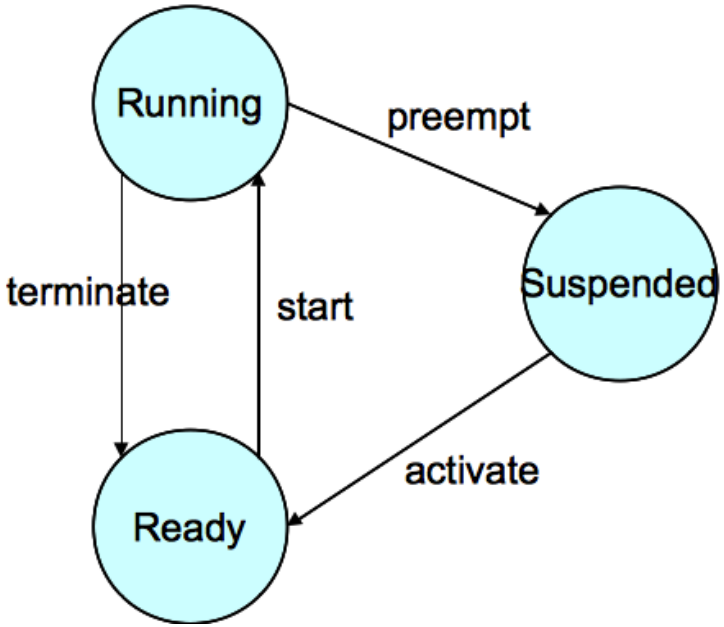
- Services for tasks
- Synchronization services (events)
- Mutual Exclusion Services
- Services for recurring phenomena (counters and alarms)
- Service for communication (in OSEK/VDX COM)
- Interruption management Services
- System services and Error management

=> All objects are static

Tasks

tâches basiques

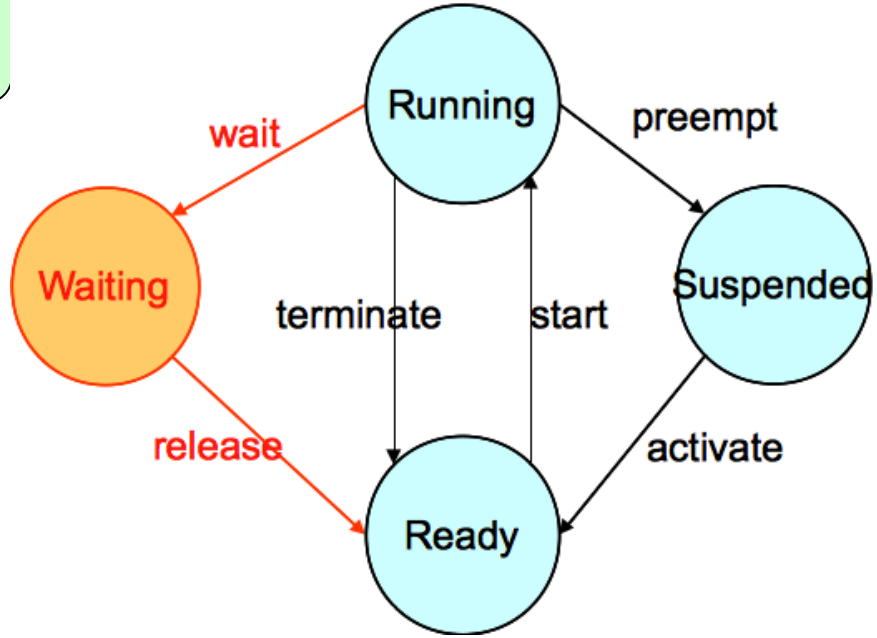
= tâche sans points bloquants



ActivateTask
TerminateTask
ChainTask
Schedule

tâches étendues

= tâche basique comportant des points bloquants



=> Possibility to memorize activation requests

Scheduling

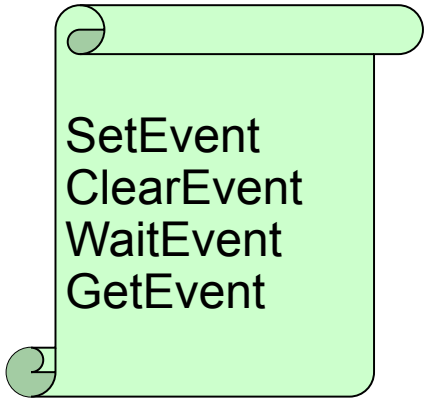
- Fixed priority that cannot be changed

- 3 scheduling modes:
  - preemptive
  - non-preemptive
  - mixed (preemptive for some tasks and non-preemptive for others)

- Management of shared resources with Priority Ceiling Protocol
  (priority inheritance + avoidance of inter-blocking by assigning priority to resources and resource allocation rule)

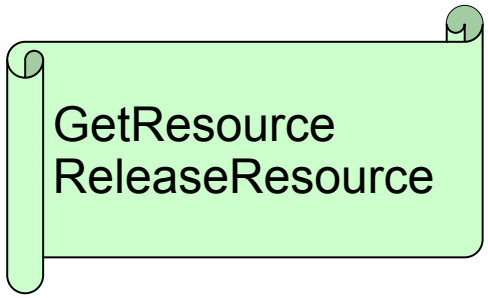
Event-driven synchronization

- Private events: 1 event is the property of a task (extended)
- Model n producers / 1 consumer
- Explicit consumer expectation (synchronous)
- Memorized events, explicit deletion
- Waiting possible in OR on a list of events



Mutual exclusion

- Services for explicitly taking and releasing a resource
- Using the PCP protocol to avoid:
  - priority inversion
  - inter-task blocking
- A standard resource: Res_scheduler (switch to non-preemptive mode)



Alarms and counters

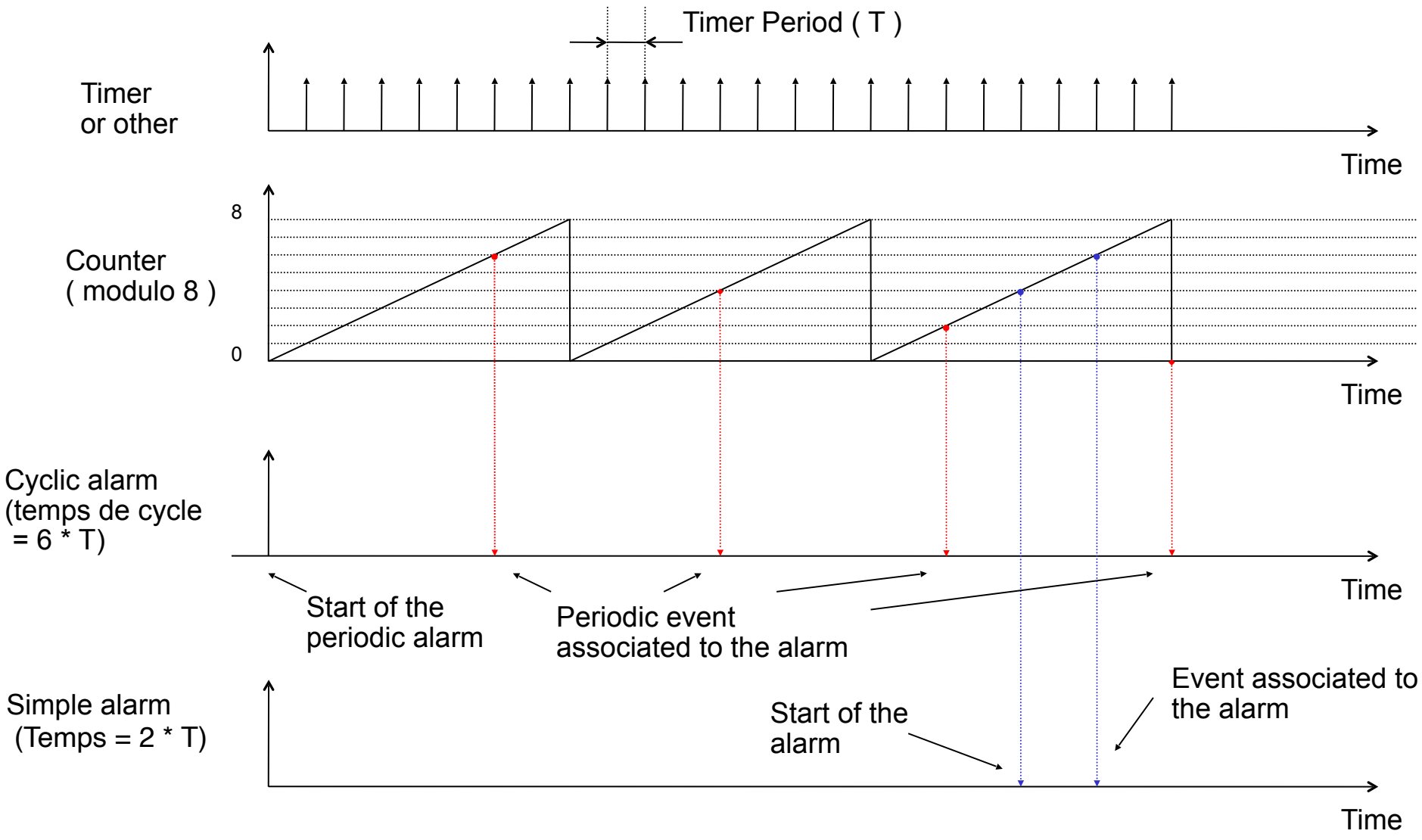
Counters

- recording of external ticks, possibly pre-divisional tick
- finished counter with automatic reset

Alarms

- attached to 1 counter and 1 task
- single or cyclic, absolute or relative triggering

Alarms and counters



Alarms and counters

Applications:

- activation of periodic or non-periodic tasks
- reporting of periodic or non-periodic event occurrences
- watchdog

SetRelAlarm
SetAbsAlarm
GetAlarm
CancelAlarm

Interruptions

Interruption categories

Category 1

ISR (isr_name)
{

code **without calls**
to services of OS

}

Category 2

ISR (isr_name)

{

code **with calls**
to service of OS

}

Inter-process communication

- Communication through messages
  - 1 message = 1 name + 1 data type + attributes

- Asynchronous communication model:
  - `SendMessage` : writing a new value
  - `ReceiveMessage` : reading of the current value

- Effective transmission according to attributes

➡ Resynchronization required :
  - polling of the state of the message, activation of the task or occurrence of event on end of sending or reception...

Inter-process communication

Some attributes of the messages:

- UnqueuedMessage : management type "blackboard"
- QueuedMessage : "mailbox" type management
- On-demand transmission
- Periodic transmission
- Mixed transmission
  ➡ periodical + communication in case of change

Inter-process communication

3 communication types:

- 1 : 1 (1 static receiver)
- 1 : 1 among n (dynamic selection from a static list)
- 1 : n (distribution to n recipients)

2 protocols (unacknowledged) :

- UUDT : non-segmented
- USDT : segmented

OSEK Task configuration: OIL

- OIL language: OSEK Implementation Language
- Hook Routines to temporary control the system
- Task types: conformity classes
  - BCC 1 : Only basic tasks
    --- 1 active task and 1 task per priority level
  - BCC 2 : Only basic tasks
    --- several active tasks + several tasks per priority level
  - ECC 1 : BCC 1 + extended tasks
  - ECC 2 : BCC 2 + extended tasks

OSEK Task configuration: OIL

- Can be split in several OIL file

  Ex: #include « implementation.oil »

- Objects:
  - define the configuration parameters
  - one main object: CPU

```
CPU ATMEL_AT91SAM7S256
{
    OS TRAMPOLINE {
    …
    };
    APPMODE appmode1;
    TASK task1 {
    …
    };
  …
}
```

Some objects

- Object OS
  - Only one OS object per CPU
  - Defines the OSEK configuration(hook routines, scheduler)

```
OS TRAMPOLINE {
    STATUS               = EXTENDED;
    STARTUPHOOK          = FALSE;
    ERRORHOOK            = FALSE;
    SHUTDOWNHOOK         = FALSE;
    PRETASKHOOK          = FALSE;
    POSTTASKHOOK         = FALSE;
    USEGETSERVICEID      = FALSE;
    USEPARAMETERACCESS   = FALSE;
    USERESSCHEDULER      = FALSE;
    };
```

- Object APPMODE (= `appmode1 {};`)
  - Groups task in a same set

Task definition

- Object Task

```
TASK taskname {
    AUTOSTART = FALSE /* task waits until alarm to
                         start */
    PRIORITY = 3; /* task priority,
                     1 = lowest priority */
    ACTIVATION = 1; /* number of occurrence of
                       the task in the ready
                       list  */
    SCHEDULE = FULL; /* NONE = no preemption
                        FULL = possible preemption*/
    STACKSIZE = 512; /* Size of the stack, ie 512*/
    };
```

- Autres champs
  - RESSOURCE = ressource1 (work with ressource n°1)
  - EVENT = event1 (waits event 1)
  - MESSAGE = message1 (synchronization using message1)

Task definition

- Tick definition : object COUNTER

```
COUNTER SysTimerCnt {
    MINCYCLE = 1; /* tick period */
    MAXALLOWEDVALUE = 100; /* max value of the cpt*/
    TICKPERBASE = 1; /* cpt step */
    };
```

- Periodic activation of tasks:
  - One alarm per task
  - One counter associated to an alarm

```
ALARM cyclic_alarm1 {
    ACTION = ACTIVATETASK {
        TASK = taskname;
    };
    AUTOSTART = TRUE {
        ALARMTIME = 1; /* 1st alarm instance */
        CYCLETIME = 1; /* period, here 1 tick */
        APPMODE = appmode1;
    };
};
```

Conclusion on OSEK

- **A mature and complete proposal**
- **Industrial products**
- **An important step towards application portability, reuse of components in ECUs ...**
- **Variants : OSEKtime OS**
  - ➡TT (time triggered) tasks for critical applications
  - ➡joins the APEX model for the partition level