Time

ESD_Cap4 --/++ (from Extra folder)

Overview

- Characterization of Time
- The role of Time in computer systems
- Active temporizations
- Implementations of Timers

Characterization of Time

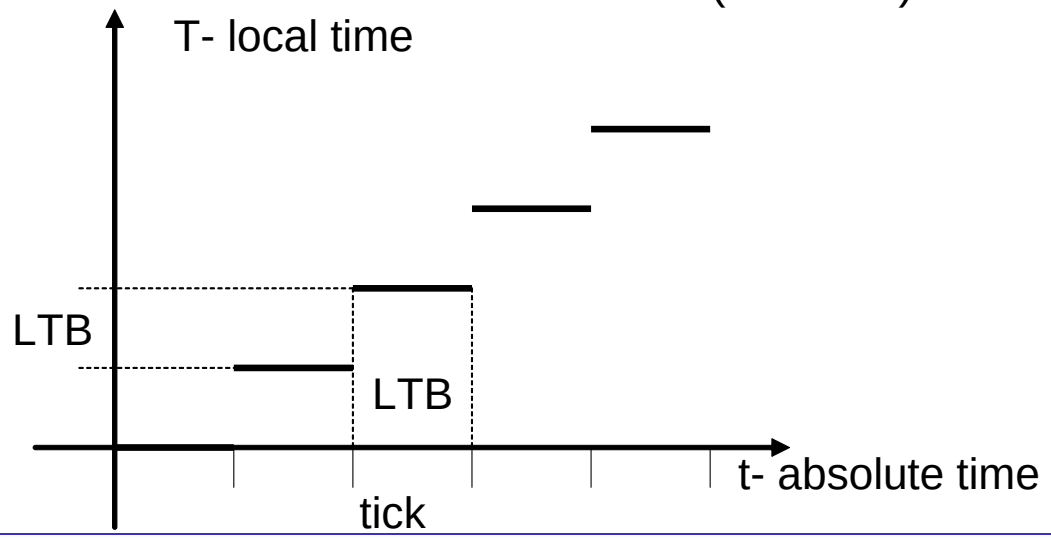
Characterization of Time

- Time model
  - Conceptual/continuous model: **t** (absolut time)
    - Time as a container of events (or produced by the events?)
  - Informatic/discrete model: **T** (local time)
    - It is associated to a physical phenomenon called **clock** that determines the **temporal resolution**
    - ltu $\Rightarrow$ local time unit [sec]
      - Local temporal resolution: it is a property of the local time representation (e.g. 1 msec resolution)
    - LTB = K*ltu $\Rightarrow$ Local Time Base [sec]
      - Updating period of a *watch variable* (time granularity)
        - » e.g. an update every 55 units of 1 msec, i.e. 18,2 times for a second

Characterization of Time

- For each CPU there could be one or more watch variables incremented of LTB every LTB in corrispondence of particolar events often called **tick**

- The behaviour of the watch variable is defined by a function in the absolut time domain T=f(t) defined as
  - **T = n ∗ LTB** (n∈N) for **n*k*ltu ≤ t < (n+1)*k*ltu**
    - Note: with a N bit watch variable T=(n mod 2^N)*LTB



Characterization of Time: errors

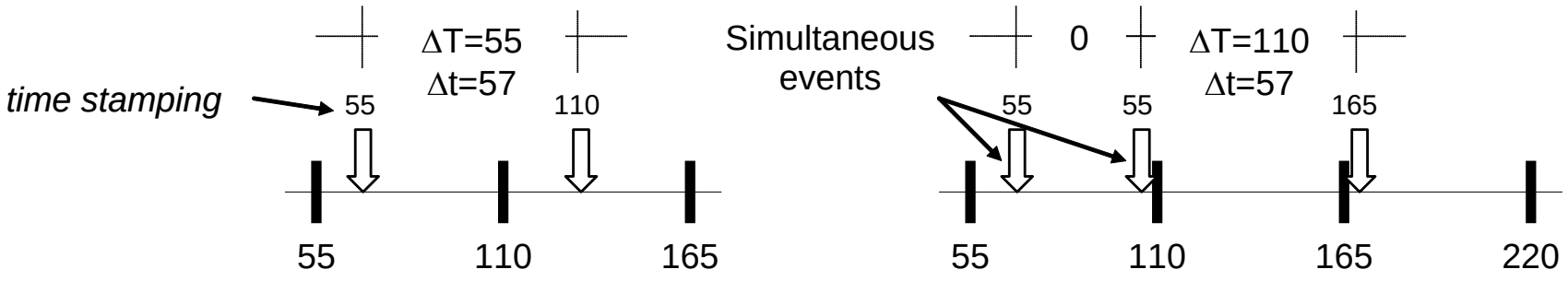
- The errors due to the adoption of a local time watch variable to misure the absolute time are decomposable in two contributions
  - Errors due to the physical phenomenon used as clock
    - Precision
  - Errors due to the granularity
    - Quantization

Errors due to the physical phenomenon

- Precise watch variable
  - **t(tick[n]) - t(tick[n-1]) = LTB + e[n]** with **e[n]=0**
  - If e[n] not always = 0
    - e[n] characterization
      - Error with average value ≠ 0
        - » Accuracy problem: the physical phenomenon has a wrong frequency
      - Error with average = 0
        - » Repeatability problem: compensating irregularity (**jitter**)
- Correct watch variable (at an absolute time t)
  - Abs(T – t) < LTB

Errors due to the granularity

- Ordering and simultaneity
  - Events that happen between two consecutive ticks are not temporally orderable and so they are considered as simultaneous in the local time also if they are distant by an absolute time $\Delta t \leq LTB$
- Interval measurement
  - Events that are distant Δt in the absolute time are considered distant ΔT in the local time
    - $\lfloor \Delta t/LTB \rfloor * LTB \leq \Delta T \leq \lceil \Delta t/LTB \rceil * LTB$



The role of Time in computer systems

The role of Time in computer systems

- Events generator (Time events flow)
  - Time as source of implicit stimulus (not produced by external world phenomenon)
    - It individuates instants to activate
      - Sampling of states
      - Information emissions
      - Update of watch variable and calendar
  - Active role
    - Use of interrupts to notify time events and to activate related actions
      - Coarse-grain granularity: 0.5 .. 100 ms

The role of Time in computer systems

- State value
  - Value to be read in specific moments
    - To evaluate temporal intervals between two events
    - To associate to an event the information related to the current time
      - e.g. *Time-stamping*
  - Passive role
    - As a variable to be read in a given moment
      - Fine-grain granularity: 1…100 μs

The role of Time in computer systems

- Temporization for HW devices
  - Events generator for HW devices that can produce other events with respect to the computer system
  - Active role
    - It produces actions on HW devices that can then generate interrupts
      - Coarse/Fine grain granularity: it depends on the HW device

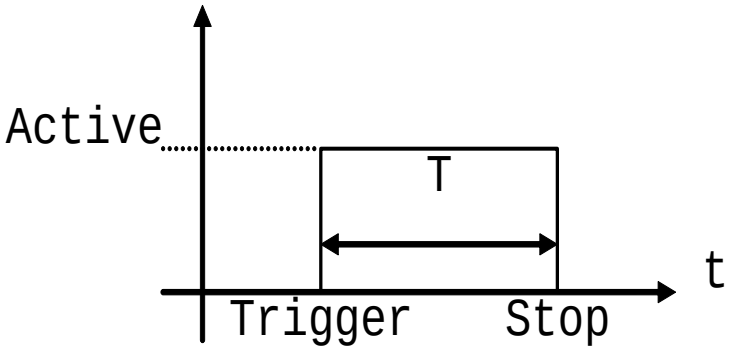
Active temporizations

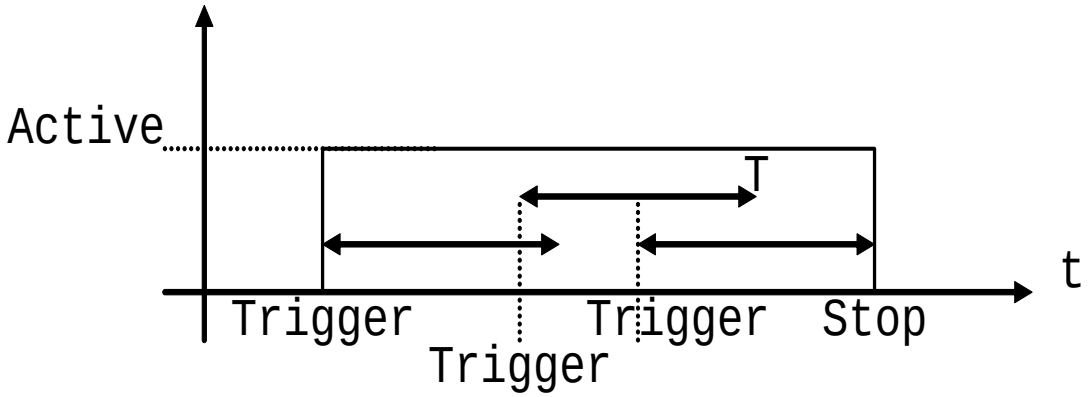
Active temporizations

- A device that provides temporization is called Timer
  - Typical active timer classification
    - *Duration Timer*
    - *Delay Timer*
    - *Cyclical Timer*

Active temporizations

- *Duration Timer*
  - It is a monostable timer activated by a *trigger* event and deactivated by a stop event after a specific amount of time T
    - It is called *re-triggerable* if the count of the time re-start from 0 when a new trigger event is provided prior to the stop event

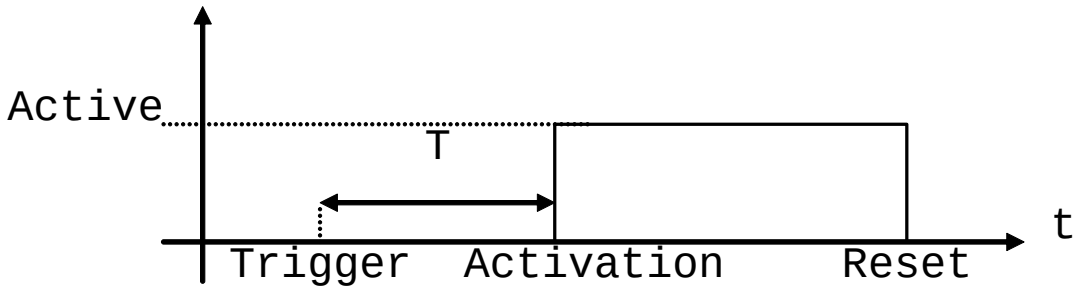




T = Timer value

Active temporizations

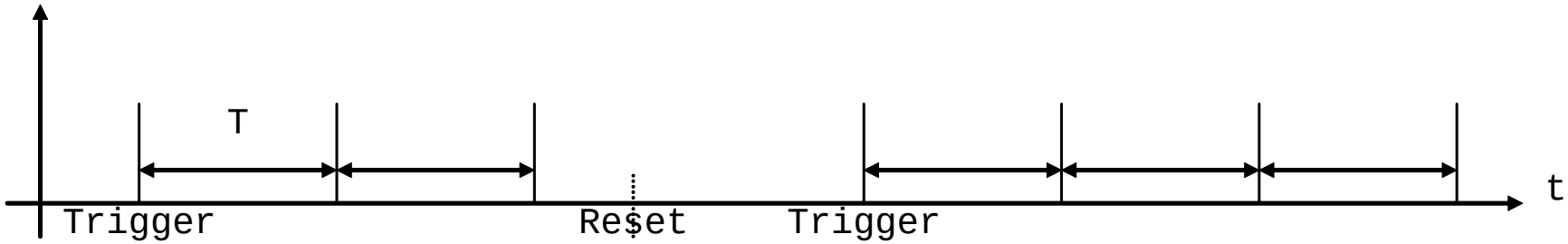
- *Delay Timer*
  - The trigger event starts the count of a specific amount of time, at the end of this time it is then generated the *activation* event
    - Normally it exists a *reset* command for the deactivation
      - If the rest command is provided prior to the generation of the activation event (i.e. before T is elapsed) the activation event is not generated



T = Timer value

Active temporizations

- *Cyclical Timer*
  - After a *trigger* event they are produced events with a period T
  - A *reset* event stops the Timer
    - It restarts after a new trigger event

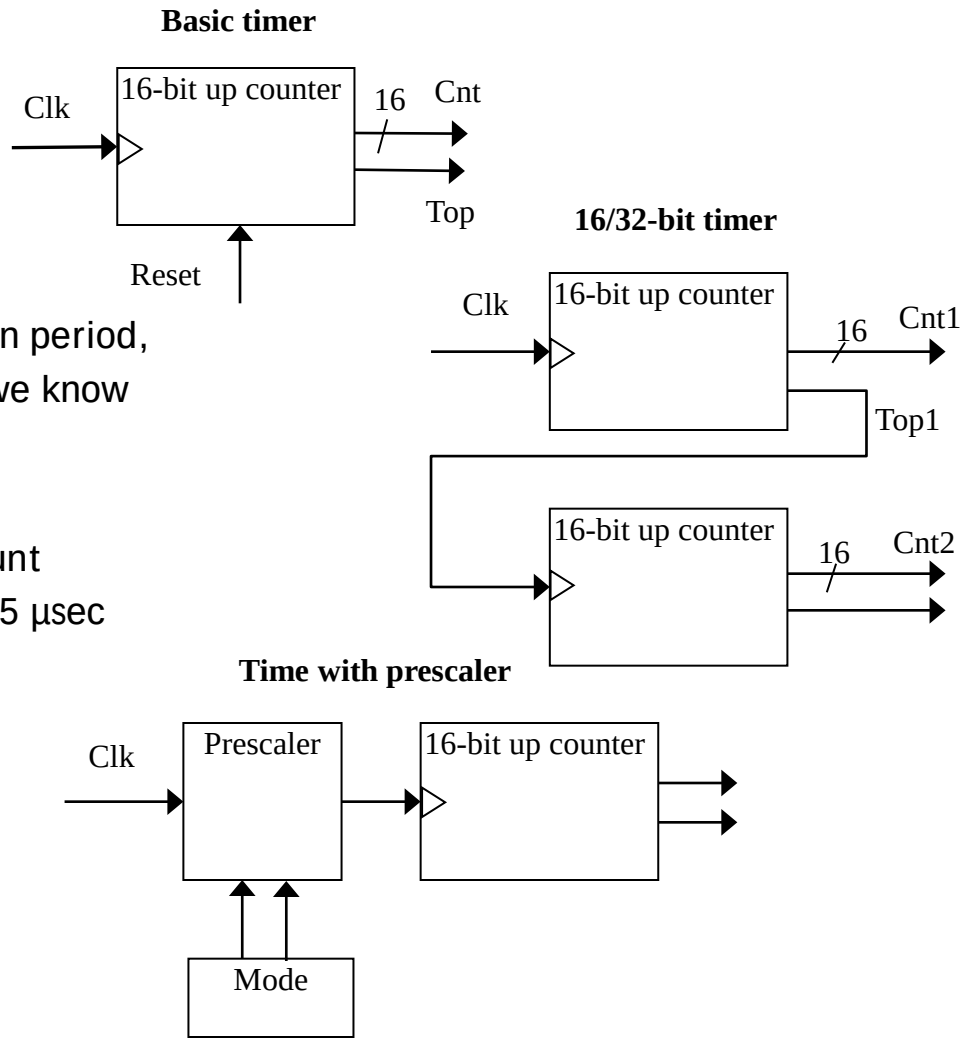


T = Timer value

Implementation of Timers

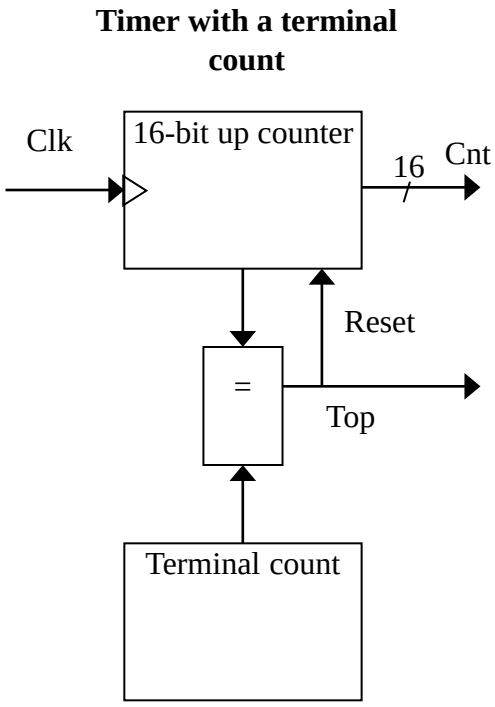
Implementation of Timers

- Basic HW components
  - *Timer Counter*
    - It measure time intervals
      - Based on impulse counting
        - » e.g. given a Clk with known period, by counting the impulses we know how much time is elapsed
      - Es. Clk 10 ns
        - » a 16-bit counter would count up to 65,535*10 ns = 655.35 µsec
        - » Top: it signals overflow
  - *Prescaler*
    - Divides the clock
    - Increments the range
    - Reduce the resolution



Implementation of Timers

- Basic HW components
  - *Interval Timer*
    - They indicates when it is elapsed a given time interval
      - *Number of clock cycles = Desired time interval / Clock period*
  - *Watchdog Timer*
    - The system shall reset the timer each X time units otherwise it signals an “alarm”
      - Common uses
        - » Detect errors and faults
        - » Self-reset
        - » Timeout
        - » Energy saving



Implementation of Timers

- HW: *Calendar/Clock Circuits*

  (WARNING: also called Real-Time Clocks – see Extra)

  - ICs that offer watch and calendar functions
    - Not used as Timers due to their low resolution/granularity (1 sec)
      - Normally based on 32768 Hz oscillators
        - » ½ pre-scaler and 16 bit counter -> (1/32768 [Hz])/2 * (2^16) = 1 [sec]
  - They contain all the counters needed for seconds, minutes, hours, date of the day (*dd/mm/aaa*)
  - Normally data are read out serially to reduce the pin-out
    - Quite slow operation (hundred of microseconds)
  - Normally battery-powered to back-up data
    - Back-up battery

Implementation of Timers

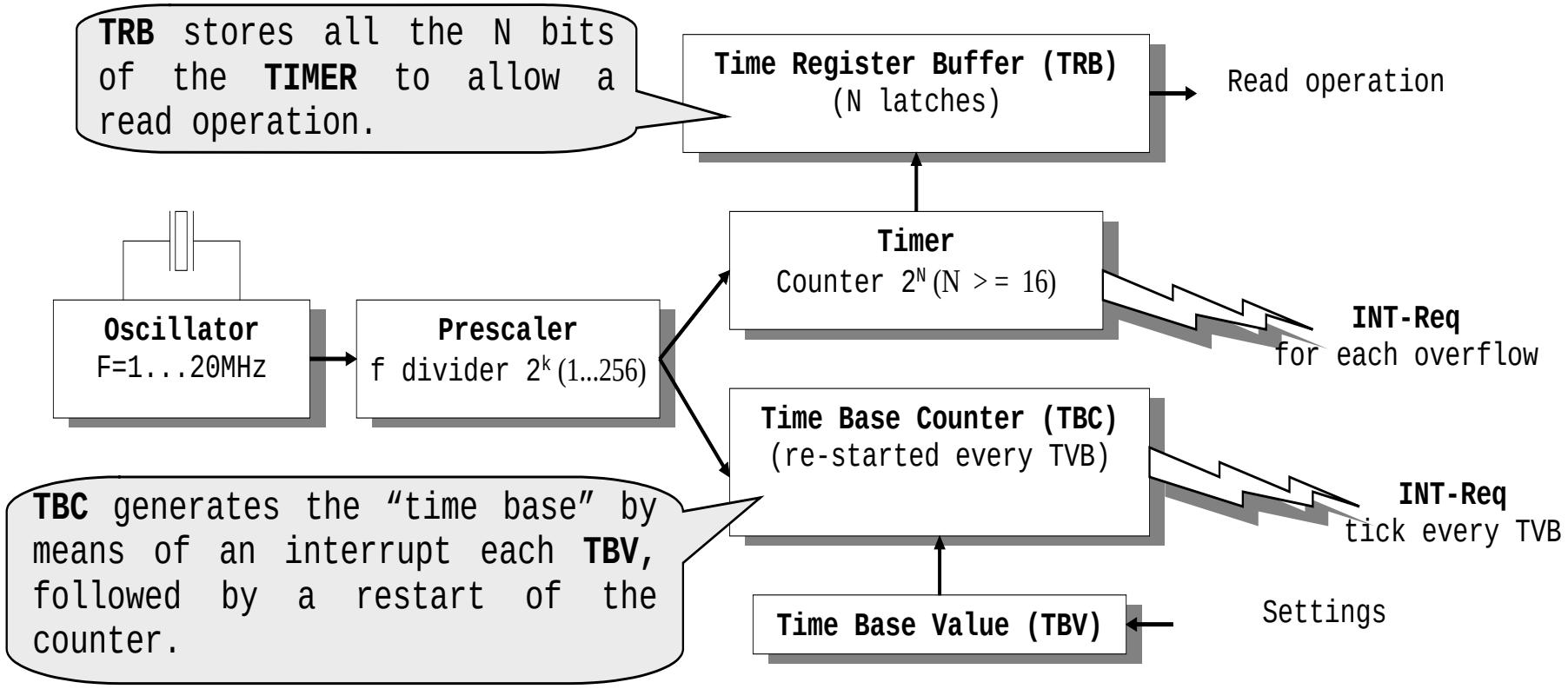
- Minimal HW/SW implementations
  - Simple clock circuits that generate periodical signals (*ticks*) to the interrupt generation circuit
    - **Real-Time Clock** (**RTC**, also called ***System Clock***)
      - Quartz oscillator with *prescaler*
        - » If backed-up it can also provide calendar/clock circuit functions
  - All the time-related system functions are implemented by means of the "*RTC tick ISR*"
    - ISR updates watch and calendar variables
      - Used both for time-stamping and interval measurements
    - ISR also manages processes state transitions

Implementation of Timers

- Minimal HW/SW implementations
  - With this approach resolution and granularity of *perception* and *mesurement* are equal and correspond to the ticks period (*Tck*)
  - Ticks frequency shall be a compromise between a fine-grain granularity and an acceptable overhead for the system
    - The overhead is the average "*RTC tick ISR*" execution time
      - It should be no more than 1/10 of Tck
    - Typical Tck values
      - 1..10 msec for real-time systems
      - 100 msec for standard applications

Implementation of Timers

- Typical HW/SW implementations
  - Main goal: different granularities for different purposes



Implementation of Timers

- Typical HW/SW implementations
  - The precision is the same of the oscillator
  - Timer
    - Resolution (TIR) = Granularity (TIG)
      - 2^k / FreqOsc [sec] (typically 1…100 μsec)
    - Period
      - TIP = 2^N ∗ TIG [sec] (typically form 64 msec to 16 sec)
        - » Itu=TIG e LTB=1
  - Time Base
    - Resolution (TBR)
      - 2^k / FreqOsc [sec] (typically 1…100 μsec)
    - Granularity (TBG) = Period (TBP)
      - TBV ∗ TBR [sec] (typically 1…50 msec)
        - » Itu=TIG e LTB=TBV

Implementation of Timers

- Typical HW/SW implementations
  - Conclusions
    - Processes time management is perfomed by the “TBC tick ISR”
    - Update of watch and calendar variables could be done, for coarse-grain usage, by “TBC tick ISR” or “TIMER overflow ISR”
    - Fine-grain interval measurements and fine-grain time-stampings will use values read directly from TIMER
      - Since TIMER has a period TIP, to measure time intervals long than TIP it is needed ti proper manage TIMER overflows

Implementation of Timers

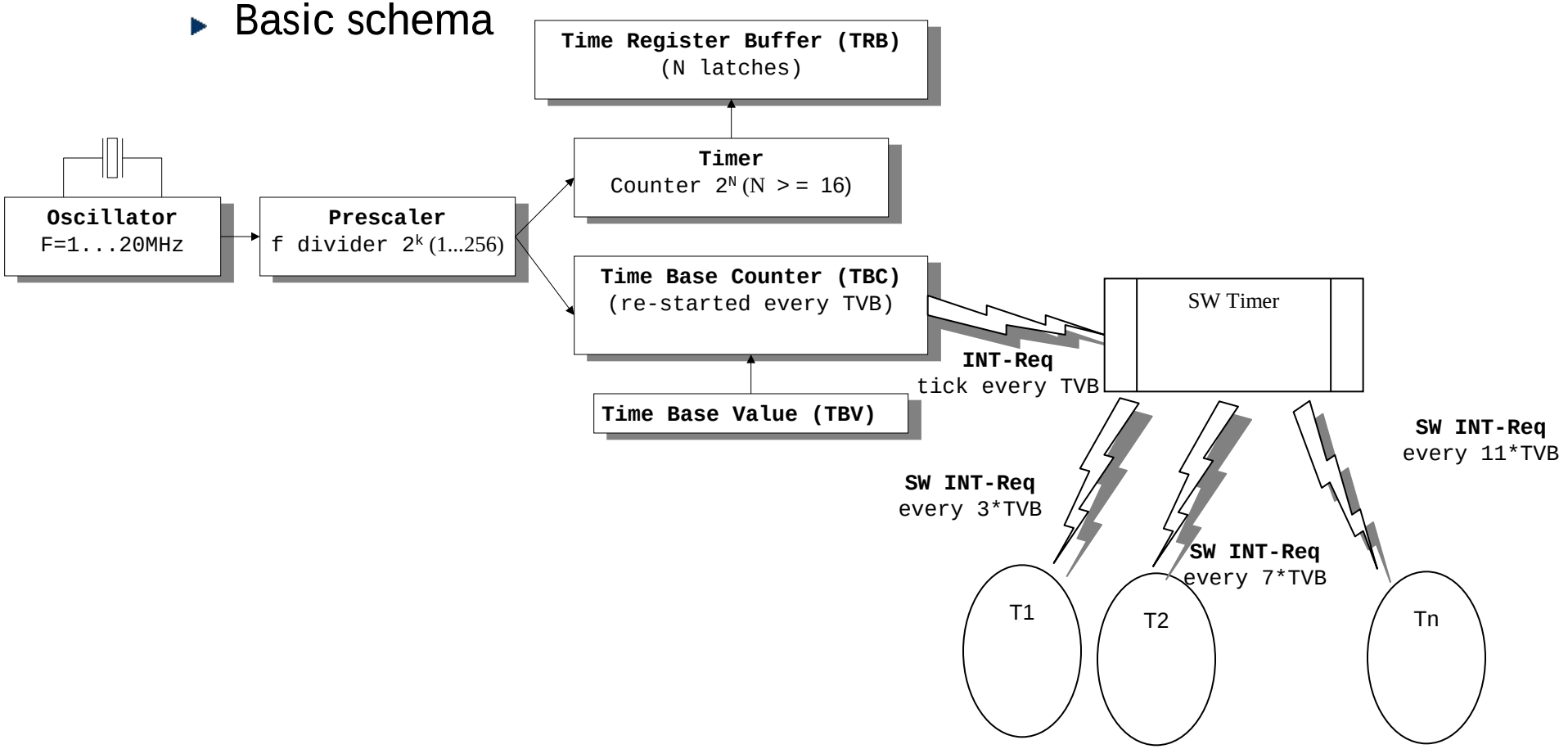
- Complex implementations
  - Special Processors
    - Processors designed to support hard real-time applications often includes HW mechanisms to perform time-related operations with fine-grain granularity and near zero overheads
      - e.g. Microcontroller Intel 80C196KB
        - » HSI/HSO
  - Temporization Co-Processors
    - In order to provide high resolutions and fine-grain granularity with near zero overheads it is also possible to use a co-processor that helps the main one to manage time-related operations
      - e.g.
        - » Such co-processors are very useful to exploit general purpose processors also for hard real-time applications

Implementation of Timers

- SW implementations
  - Each application could have its time-related requirements
    - They could be different and also concurrent
  - So, it could be not possible to use an HW Timer for each application but it is needed a SW implementation that "virtualize" a single HW Timer
    - For example, it is possible to manage multiple SW Timers by means of a single HW Timer ISR
      - This approach is used in almost all the operating systems to provide time-services to different (and often concurrent) processes
        - » Processes ask time-services to the OS by means of a proper API

Implementation of Timers

- SW implementations
  - Basic schema



Implementation of Timers

- SW implementations
  - Basic Linux example

```
////////////////////////////////////////

#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>
#include <sys/types.h>

////////////////////////////////////////

// Timed actions
void todo()
{
    printf("Hello1!\n");
}

// Timeout management
void timeout(int s)
{
  todo();
  alarm(1);
}
```

```
////////////////////////////////////////

int main()
{
  signal(SIGALRM, timeout);
  alarm(1);

  while(1);

  return 0;

}
```

