

Real Time Programming with Ada

Part 2: Real time facilities

Real Time Programming: we need support for

- Concurrency (Ada tasking)
- Communication & synchronization (Ada Rendezvous)
- Consistency in data sharing (Ada protected data type)
- **Real time facilities (Ada real time packages and delay statements)**
 - accessing system time so that the passage of time can be measured
 - delaying activities until future time points
 - Timeouts: waiting for or running some action for a given time period
 - (Scheduling :timely execution of tasks by a Run-time system)

System Time

A **timer circuit** programmed to interrupt the processor at fixed rate.

- To approximate the universal time

Each time interrupt is called a system **tick** (time resolution):

- Normally, the tick can vary 1-50ms, even microseconds in RTOS
 - Linux 2.4, 10ms (100HZ), Linux 2.6, 1ms (1000HZ)
- The tick may be selected by the user
- All time parameters for tasks should be the multiple of the tick
- System time = 32 bits
 - One tick = 1ms: your system can run 50 days
 - One tick = 20ms: your system can run 1000 days = 2.5 years
 - One tick = 50ms: your system can run 2500 days= 7 years
- In **Ada95**, it is required that the system time should last at least **50 years**

Real-Time Support in Ada

- Two pre-defined packages to access the system clock
 - `Ada.Calendar` and `Ada.Real_Time`
 - Both based on the same hardware clock
- There are two delay-statements
 - `Delay` *time_expression* (in seconds)
 - `Delay until` *time_expression*
- The `delay` statements can be used together with `select` to program timeouts, timed entry etc.

Package calendar in Ada: specification

```
package Ada.Calendar is
  type Time is private;
    --- time is pre-defined based on the system clock
  subtype Year_Number is Integer range 1901 .. 2099;
  subtype Month_Number is Integer range 1 .. 12;
  subtype Day_Number is Integer range 1 .. 31;
  subtype Day_Duration is Duration range 0.0 .. 86_400.0;
    --- Duration is pre-defined type (length of interval,
        expressed in sec's) declared in the package: Standard
  function Clock return Time;
  function Year (Date : Time) return Year_Number;
  function Month (Date : Time) return Month_Number;
  function Day (Date : Time) return Day_Number;
  function Seconds (Date : Time) return Day_Duration;
  procedure Split (Date : in Time;
                  Year : out Year_Number;
                  Month : out Month_Number;
                  Day : out Day_Number;
                  Seconds : out Day_Duration);
```

Package calendar in Ada: specification (ctn.)

```
function Time_Of (Year      : Year_Number;  
                  Month     : Month_Number;  
                  Day       : Day_Number;  
                  Seconds   : Day_Duration := 0.0)
```

```
return Time;
```

```
function "+" (Left : Time;    Right : Duration) return Time;
```

```
function "+" (Left : Duration; Right : Time) return Time;
```

```
function "-" (Left : Time;    Right : Duration) return Time;
```

```
function "-" (Left : Time;    Right : Time) return Duration;
```

```
function "<" (Left, Right : Time) return Boolean;
```

```
function "<=" (Left, Right : Time) return Boolean;
```

```
function ">" (Left, Right : Time) return Boolean;
```

```
function ">=" (Left, Right : Time) return Boolean;
```

```
Time_Error : exception;
```

```
private
```

```
    -- not specified by the language
```

```
    -- implementation dependent
```

```
end Ada.Calendar;
```

Package Real_Time in Ada: specification

```
package Ada.Real_Time is
  type Time is private;
  Time_First : constant Time;
  Time_Last  : constant Time;
  Time_Unit  : constant := implementation-defined-real-number;
  type Time_Span is private;
      --- as Duration, a Time_Span value M representing
      the length of an interval, corresponding to
      the real time duration M*Time_Unit.
  Time_Span_First : constant Time_Span;
  Time_Span_Last  : constant Time_Span;
  Time_Span_Zero  : constant Time_Span;
  Time_Span_Unit  : constant Time_Span;
  Tick : constant Time_Span;
  function Clock return Time;
  function "+" (Left : Time; Right : Time_Span) return Time;
  function "+" (Left : Time_Span; Right : Time) return Time;
  function "-" (Left : Time; Right : Time_Span) return Time;
  function "-" (Left : Time; Right : Time) return Time_Span;
  function "<" (Left, Right : Time) return Boolean;
  function "<=" (Left, Right : Time) return Boolean;
  function ">" (Left, Right : Time) return Boolean;
  function ">=" (Left, Right : Time) return Boolean;
```

Package Real_Time in Ada: specification (cnt.)

```
function "+" (Left, Right : Time_Span) return Time_Span;
function "-" (Left, Right : Time_Span) return Time_Span;
function "-" (Right : Time_Span) return Time_Span;
function "*" (Left : Time_Span; Right : Integer) return Time_Span;
function "*" (Left : Integer; Right : Time_Span) return Time_Span;
function "/" (Left, Right : Time_Span) return Integer;
function "/" (Left : Time_Span; Right : Integer) return Time_Span;
function "abs"(Right : Time_Span) return Time_Span;
function "<" (Left, Right : Time_Span) return Boolean;
function "<=" (Left, Right : Time_Span) return Boolean;
function ">" (Left, Right : Time_Span) return Boolean;
function ">=" (Left, Right : Time_Span) return Boolean;
function To_Duration (TS : Time_Span) return Duration;
function To_Time_Span (D : Duration) return Time_Span;
function Nanoseconds (NS : Integer) return Time_Span;
function Microseconds (US : Integer) return Time_Span;
function Milliseconds (MS : Integer) return Time_Span;
type Seconds_Count is range implementation-defined;
procedure Split(T : in Time; SC : out Seconds_Count;
                TS : out Time_Span);
function Time_Of(SC : Seconds_Count; TS : Time_Span) return Time;
private
  ... -- not specified by the language
end Ada.Real_Time;
```


Programming Delays

Relative Delays

- Delay the execution of a task for a given period

```
delay 10.0
```

- Alternatively by accessing the system clock:

```
Start := Clock;
```

```
loop
```

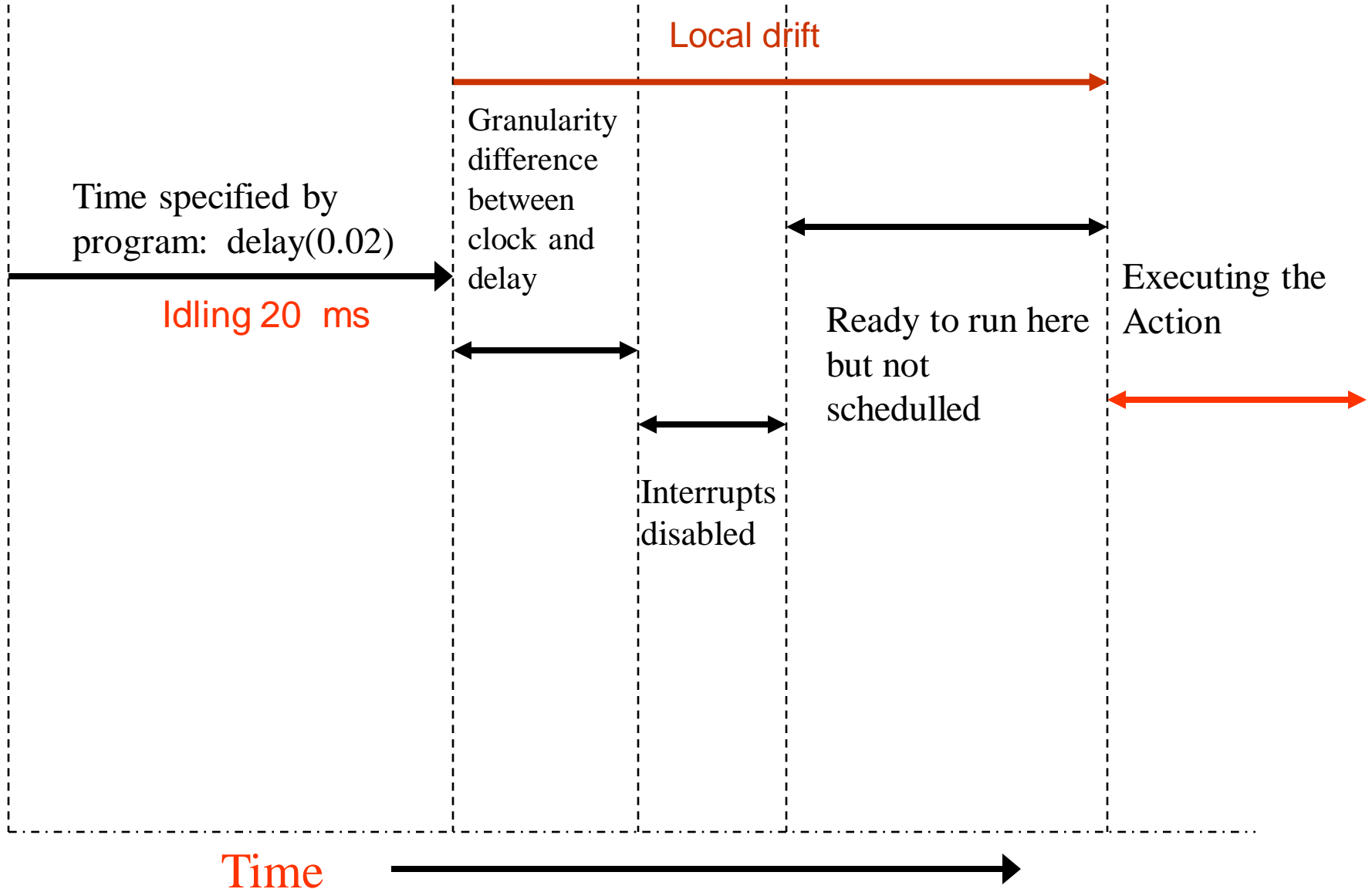
```
    exit when (Clock - Start) > 10.0;    -- busy waiting
```

```
end loop;
```

```
ACTION;
```

(access “clock” all time every tick!)

Semantics of `Delay(0.02); Action`



Absolute Delays

- To delay the execution of a task to a specified time point

```
START := Clock;  
FIRST_ACTION;  
delay until START + 10.0; (this is by interrupt)  
SECOND_ACTION;
```

- Alternatively by accessing the system clock

```
Start := Clock;  
FIRST_ACTION;  
loop  
  exit when Clock > Start+10.0; -- busy waiting  
end loop;  
SECOND_ACTION;
```

(access “clock” all time every tick!)

- As with **delay**, **delay until** is accurate only in **its lower bound**

Periodic Task

```
with Ada.Real_Time; use Ada.Real_Time;
```

```
task Periodic_T;
```

```
task body Periodic_T is
```

```
    Next_Release : Time;
```

```
    ReleaseInterval : Duration := 10
```

```
begin
```

```
    Next_Release := Clock + ReleaseInterval;
```

```
    loop
```

```
        -- Action
```

```
        delay until Next_Release;
```

```
        Next_Release := Next_Release + ReleaseInterval;
```

```
    end loop;
```

```
end Periodic_T;
```

If Action takes 11 seconds, the delay statement will have no effect

Will run on average every 10 seconds

local drift only

Example (ticket office)

```
with Ada.calendar; use Ada.calendar;  
procedure Ticket_office is
```

all tasks

```
task Ticket_Agent is  
  entry Registration(...);  
end Ticket_Agent;
```

```
task body Ticket_Agent is  
  -- declarations  
  Shop_Open : Boolean := True;  
Begin  
  Closing_Time := clock + hours(8);  
  while Shop_Open loop  
    select  
      accept Registration(...) do  
        -- log details  
      end Registration;  
    or  
      delay until Closing_Time;  
      Shop_Open := False;  
    end select;  
    -- process registrations  
  end loop;  
end Ticket_Agent;
```

```
Begin
```

```
...
```

```
end Ticket_office
```

Example (Controller 1)

```
with Ada.Real_Time; use Ada.Real_Time;  
with Data_Types; use Data_Types;  
with IO; use IO;  
with Control_Procedures; use Control_Procedures;
```

```
procedure Controller is  
task Temp_Controller;  
task Pressure_Controller;
```

Example (Controller 2)

```
task body Temp_Controller is  
  TR : Temp_Reading; HS : Heater_Setting;  
  Next : Time;  
  Interval : Time_Span := Milliseconds(30);  
begin  
  Next := Clock;  -- start time  
  loop  
    Read(TR);  
    Temp_Convert(TR, HS);  
    Write(HS);  
    Write(TR);  
    Next := Next + Interval;  
    delay until Next;  
  end loop;  
end Temp_Controller;
```


Example (Controller 3)

```
task body Pressure_Controller is
  PR : Pressure_Reading; PS : Pressure_Setting;
  Next : Time;
  Interval : Time_Span := Milliseconds(70);
begin
  Next := Clock;  -- start time
  loop
    Read(PR);
    Pressure_Convert(PR, PS);
    Write(PS);
    Write(PR);
    Next := Next + Interval;
    delay until Next;
  end loop;
end Pressure_Controller;
begin
  null;
end Controller;
```

Here Temp_Controller & Pressure_Controller start concurrently

Programming Timeouts

Timeout on "Entries"

```
loop
  select
    accept Call(T : temperature) do
      New_temp:=T;
    end Call;
  or
    delay 10.0;
      --action for timeout
    end select;
  --other actions
end loop;
```

Timeout (by server)

```
task Server is
  entry Call(T : in Temperature);
  -- other entries
end Server;

task body Server is
  -- declarations
begin
  loop
    select
      accept Call(T : in Temperature) do
        New_Temp := T;
      end Call;
    or
      delay 10.0;
      -- action for timeout
    end select;
    -- other actions
  end loop;
end Server;
```

Timeout (by client)

```
loop
  -- get new temperature T
  Server.Call(T);
end loop;
```

```
loop
  -- get new temperature T
  select
    Server.Call(T);
  or
    delay 0.5;
    -- other actions
  end select;
end loop;
```

SUMMARY: Language support for RT Programming

- Concurrency: multi-tasking
- Communication & synchronization
- Consistency in data sharing /protected data types
- Real time facilities
 - Access to system clock/time
 - Delay constructs: Delay(10) and Delay until next-time
 - Timeouts: select and delay
 - Timely execution of tasks (run-time system)