Beispiele zum Anhang: Maschinennahes Programmieren in Ada

Aus: Alan Burns, Andy Wellings: *Real-Time Systems and Programming Languages. Ada, Real-Time Java and C/Real-Time POSIX*. Addison Wesley, 2009 (Kapitel 14.3).

Das Package `Ada.Interrupts`

```
package Ada.Interrupts is
  type Interrupt_Id is implementation_defined; -- must be discrete
  type Parameterless_Handler is access protected procedure;

  function Is_Reserved(Interrupt : Interrupt_Id) return Boolean;
    -- Returns True if the interrupt is reserved,
    -- returns False otherwise.
  function Is_Attached(Interrupt : Interrupt_Id) return Boolean;
    -- Returns True if the interrupt is attached to a
    -- handler, returns False otherwise.
    -- Raises Program_Error if the interrupt is reserved.
  function Current_Handler(Interrupt : Interrupt_Id)
                          return Parameterless_Handler;
    -- Returns an access variable to the current handler for
    -- the interrupt. If no user handler has been attached, a
    -- value is returned which represents the default handler.
    -- Raises Program_Error if the interrupt is reserved.
  procedure Attach_Handler(New_Handler : Parameterless_Handler;
                          Interrupt : Interrupt_Id);
    -- Assigns New_Handler as the current handler.
    -- If New_Handler is null, the default handler is restored.
    -- Raises Program_Error:
    --     (a) if the protected object associated with the
    --         New_Handler has not been identified with a
    --         pragma Interrupt_Handler,
    --     (b) if the interrupt is reserved,
    --     (c) if the current handler was attached statically
    --         using pragma Attach_Handler.
  procedure Exchange_Handler(
            Old_Handler : out Parameterless_Handler;
            New_Handler : Parameterless_Handler;
            Interrupt : Interrupt_Id);
    -- Assigns New_Handler as the current handler for the
    -- Interrupt and returns the previous handler in
    -- Old_Handler.
    -- If New_Handler is null, the default handler is restored.
    -- Raises Program_Error:
    --     (a) if the protected object associated with the
    --         New_Handler has not been identified with a
    --         pragma Interrupt_Handler,
    --     (b) if the interrupt is reserved,
    --     (c) if the current handler was attached statically
    --         using pragma Attach_Handler.
  procedure Detach_Handler(Interrupt : Interrupt_Id);
    -- Restores the default handler for the specified interrupt.
    -- Raises Program_Error:
    --     (a) if the interrupt is reserved,
    --     (b) if the current handler was attached statically
    --         using pragma Attach_Handler.
  function Reference(Interrupt : Interrupt_Id)
          return System.Address;
    -- Returns an Address which can be used to attach
    -- a task entry to an interrupt via an address
    -- clause on an entry.
    -- Raises Program_Error if the interrupt cannot be
    -- attached in this way.
private
  ... -- not specified by the language
end Ada.Interrupts;
```

Das Package `Ada.Interrupts.Names`

Das Package kann durch eine Ada-Implementierung definiert werden.

Dadurch ist es möglich, Namen als Interrupt-Bezeichner zu werwenden (z.B. `Adc`, siehe Beispiel S. A-16 und unten)

```
package Ada.Interrupts.Names is
  implementation_defined : constant Interrupt_Id :=
                                    implementation_defined;
  ...
  implementation_defined : constant Interrupt_Id :=
                                    implementation_defined;
private
  ... -- not specified by the language
end Ada.Interrupts.Names;
```

Beispiel: Ada Code für einen Gerätetreiber (A/D-Konverter, vgl. S. A-16)

```
package Adc_Device_Driver is
  Max_Measure : constant := (2**16)-1;
  type Channel is range 0..63;
  subtype Measurement is Integer range 0..Max_Measure;
  procedure Read(Ch: Channel; M : out Measurement);
    -- potentially blocking
  Conversion_Error : exception;
private
  for Channel'Size use 6;
  -- indicates that six bits only must be used
end Adc_Device_Driver;

-----

with Ada.Interrupts.Names; use Ada.Interrupts;
with System; use System;
with System.Storage_Elements; use System.Storage_Elements;
package body Adc_Device_Driver is

  Bits_In_Word : constant := 16;
  Word : constant := 2; -- bytes in word
  type Flag is (Down, Set);

  type Control_Register is
  record
    Ad_Start : Flag;
    Ienable  : Flag;
    Done     : Flag;
    Ch       : Channel;
    Error    : Flag;
  end record;
```

```
for Control_Register use
  -- specifies the layout of the control register
record
    Ad_Start at 0*Word range 0..0;
    Ienable  at 0*Word range 6..6;
    Done     at 0*Word range 7..7;
    Ch       at 0*Word range 8..13;
    Error    at 0*Word range 15..15;
end record;

for Control_Register'Size use Bits_In_Word;
  -- the register is 16-bits long
for Control_Register'Alignment use Word;
  -- on a word boundary
for Control_Register'Bit_order use Low_Order_First;

type Data_Register is range 0 .. Max_Measure;
for Data_Register'Size use Bits_In_Word;
  -- the register is 16-bits long

Contr_Reg_Addr : constant Address := To_Address(8#150002#);
Data_Reg_Addr : constant Address  := To_Address(8#150000#);
Adc_Priority : constant Interrupt_Priority := 63;

Control_Reg : aliased Control_Register;
    -- aliased indicates that pointers are used to access it
for Control_Reg'Address use Contr_Reg_Addr;
    -- specifies the address of the control register

Data_Reg : aliased Data_Register;
for Data_Reg'Address use Data_Reg_Addr;
    -- specifies the address of the data register


protected type Interrupt_Interface(Int_Id : Interrupt_Id;
               Cr : access Control_Register;
               Dr : access Data_Register) is
  entry Read(Chan : Channel; M : out Measurement);
private
  entry Done(Chan : Channel; M : out Measurement);
  procedure Handler;
  pragma Attach_Handler(Handler, Int_Id);
  pragma Interrupt_Priority(Adc_Priority);
    -- see Section @\ref{ceiling}~ for discussion on priorities
  Interrupt_Occurred : Boolean := False;
  Next_Request : Boolean := True;
end Interrupt_Interface;

Adc_Interface : Interrupt_Interface(Names.Adc,
                Control_Reg'Access,
                Data_Reg'Access);
  -- this assumes that `Adc' is registered as an
  -- Interrupt_Id in Ada.Interrupts.Names
  -- 'Access gives the address of the object
  protected body Interrupt_Interface is

    entry Read(Chan : Channel; M : out Measurement)
          when Next_Request is
      Shadow_Register : Control_Register;
    begin
      Shadow_Register := (Ad_Start => Set, Ienable => Set,
            Done => Down, Ch   => Chan, Error => Down);
      Cr.all := Shadow_Register;
      Interrupt_Occurred := False;
      Next_Request := False;
      requeue Done;
    end Read;

    procedure Handler is
    begin
      Interrupt_Occurred := True;
    end Handler;

    entry Done(Chan : Channel; M : out Measurement)
                             when Interrupt_Occurred is
    begin
      Next_Request := True;
      if Cr.Done = Set and Cr. Error = Down then
            M := Measurement(Dr.all);
      else
        raise Conversion_Error;
     end if;
    end Done;
  end Interrupt_Interface;

  procedure Read(Ch : Channel; M : out Measurement) is
  begin
    for I in 1..3 loop
      begin
        Adc_Interface.Read(Ch,M);
        return;
      exception
        when Conversion_Error => null;
      end;
    end loop;
    raise Conversion_Error;
  end Read;

end Adc_Device_Driver;
```