-- Low-level I/O driver for the AVR.

with Interfaces; use Interfaces;

package NXT.AVR_IO is

----------
-- Outgoing --
----------

procedure Send_Initailization;
-- send the mandatory initialization message to the AVR

type Outgoing_AVR_Message is private;
-- Represents all messages sent to the AVR

Null_Outgoing_Msg : constant Outgoing_AVR_Message;

function Power_Down_Message return Outgoing_AVR_Message;

function Set_Sensor_Power_Message
(Port : Sensor_Id;
 Power_Type : Sensor_Power)
return Outgoing_AVR_Message;

function Set_Motor_Message
(Motor : Motor_Id;
 Power : PWM_Value;
 Brake : Boolean)
return Outgoing_AVR_Message;

procedure Send_To_AVR (This : Outgoing_AVR_Message);
-- computes and appends checksum, sends to AVR
type Incoming_AVR_Message is private;
   -- Represents all messages received from the AVR

type Raw_ADC_Inputs is array (Sensor_Id) of Unsigned_16;
pragma Atomic_Components (Raw_ADC_Inputs);
   -- A/D converter values, each with a range of 0 .. 1023

function Raw_Inputs (From : Incoming_AVR_Message) return Raw_ADC_Inputs;
   -- Returns the raw input readings from the message. Does no actual
   -- "decoding", just provides access.
pragma Inline_Always (Raw_Inputs);  -- just an accessor function

function Buttons_Sample (From : Incoming_AVR_Message) return Unsigned_16;
   -- Returns the raw buttons reading contained within From
pragma Inline_Always (Buttons_Sample);  -- just an accessor function

function Battery_Reading (From : Incoming_AVR_Message) return Unsigned_16;
   -- Returns the battery reading contained within From
pragma Inline_Always (Battery_Reading);  -- just an accessor function

procedure Receive_From_AVR
   (This : out Incoming_AVR_Message;
    Valid : out Boolean);
   -- Receives the next message from the AVR and places into This iff the
   -- checksum in the received message is correct. If the checksum is not
   -- correct, no update to This takes place.
   -- On return, Valid will be True if checksum in received message is
   -- correct, otherwise it will be False.

private
   -- outgoing message representation, sent from ARM to AVR

type PWM_Output_Values is array (0 .. 3) of PWM_Value;
for PWM_Output_Values'Component_Size use 8;
for PWM_Output_Values'Size use 32; -- confirming

type Port_Bits is array (Sensor_Id) of Boolean;
for Port_Bits'Component_Size use 1;
for Port_Bits'Size use 4; -- confirming

type Sensor_Power_Control is
  record
    Pulsed_9V : Port_Bits; -- "active"
    Constant_9V : Port_Bits;
  end record;

for Sensor_Power_Control use
  record
    Pulsed_9V at 0 range 0 .. 3;
    Constant_9V at 0 range 4 .. 7;
  end record;

for Sensor_Power_Control'Size use 8; -- confirming

type Motor_Braking_Control is (No_Braking, Braking);
for Motor_Braking_Control use (No_Braking => 0, Braking => 1);
-- the above is confirming, but the values are essential so we make it
-- explicit to prevent accidental reordering later

type Motor_Braking_Modes is array (0 .. 7) of Motor_Braking_Control;
-- bit 0 is Motor A, 1 is Motor B, and 2 is Motor C; others are unused
for Motor_Braking_Modes'Component_Size use 1;
for Motor_Braking_Modes'Size use 8; -- confirming

type Outgoing_AVR_Message is
  record
    Power_Command : Unsigned_8;
    PWM_Frequency : Unsigned_8; -- in KHz units, range 1 .. 32
    PWM_Values : PWM_Output_Values;
    Output_Mode : Motor_Braking_Modes;
    Input_Power : Sensor_Power_Control;
    Checksum : Unsigned_8;
  end record;
for Outgoing_AVR_Message use
    record
        Power_Command at 0 range 0 .. 7;
        PWM_Frequency at 1 range 0 .. 7;
        PWM_Values at 2 range 0 .. 31;
        Output_Mode at 6 range 0 .. 7;
        Input_Power at 7 range 0 .. 7;
        Checksum at 8 range 0 .. 7;
    end record;

Null_Outgoing_Msg : constant Outgoing_AVR_Message :=
    (Power_Command => 0,  
        PWM_Frequency => 0,  
        PWM_Values => (0, 0, 0, 0),  
        Output_Mode => (others => No_Braking),  
        Input_Power => ((others => False), (others => False)),  
        Checksum => 0);

-- incoming message representation, received from AVR

type Incoming_AVR_Message is
    record
        Inputs : Raw_ADC_Inputs; -- Raw a/d converter values [0..1023]
        Buttons : Unsigned_16; -- Raw a/d converter values [0..1023]
        Battery : Unsigned_16; -- Raw a/d converter values [0..1023]
        Checksum : Unsigned_8;
    end record;

for Incoming_AVR_Message use
    record
        Inputs at 0 range 0 .. 63;
        Buttons at 8 range 0 .. 15;
        Battery at 10 range 0 .. 15; -- also contains firmware info
        Checksum at 12 range 0 .. 7;
    end record;

end NXT.AVR_IO;