```
--=====================================================================
--  This document is the property of Aliathon. The document (or any   --
--  part of it) must not be copied without permission from Aliathon.  --
--  Any authorised copies of this document must include this header.  --
--                   (c) Copyright Aliathon Ltd                       --
--=====================================================================
--  File        : crc_pkg.vhd
--
--  Author      : Steve McDonald
--
--  Description : This module provides some standard CRC functions...
--=====================================================================
--  Notes : This file is provided for information only.  It does not
--          imply any warranty or liability and must be used entirely
--          at your own risk.
--=====================================================================

library ieee;
use ieee.std_logic_1164.all;

package crc_pkg is

  function  ccitt_crc_16(d, c : std_logic_vector) return std_logic_vector;
  function  ccitt_crc_32(d, c : std_logic_vector) return std_logic_vector;
  function   atm_hec_crc(d    : std_logic_vector) return std_logic_vector;

end crc_pkg;


package body crc_pkg is

  --                                                 data in >-----
  --                                                              |
  --      ----         ----         ----             ------      |
  --   ->| c0 |--(+)->| c1 |--(+)->| c2 |-..--(+)->| c(n) |--->(+)
  --  |   ----    ^    ----    ^    ----       ^    ------      |
  --  |           |            |               |                |
  --   ---------------------------------------------------------
  --   0           1            2               (n)
  --  x (1)       x            x               x     <- CRC Poly
  --
  -- The CRC is implemented as shown above...
  -- The incoming data is fed into the shift register.  If the bit
  -- in the CRC Polynomial is set, the corresponding tap is added.
  -- Note: Careful with the bit order convention as some CRC's
  --       (e.g. G.704) have the bit numbers reversed...

  function crc(p,d,c : std_logic_vector) return std_logic_vector is
    variable r : std_logic_vector(c'length downto 1) := c;
  begin
    for i in d'range loop
      if ((r(r'high) xor d(i)) = '1') then
        r := (r(r'high-1 downto 1) xor p(r'high-1 downto 1)) & '1';
      else
        r :=  r(r'high-1 downto 1)                           & '0';
      end if;
    end loop;
    return (r);
  end;

  --                          16  12  5
  -- CCITT CRC-16 Poly is:  x + x + x + 1

  constant ccitt_crc_16_poly : std_logic_vector(16 downto 0)
                             := "10001000000100001";

  function ccitt_crc_16(d,c : std_logic_vector) return std_logic_vector is
  begin
    return crc(ccitt_crc_16_poly,d,c);
  end;

  --                          32  26  23  22  16  12  11  10  8   7   5   4   2
  -- CCITT CRC-32 Poly is:  x + x + x + x + x + x + x + x + x + x + x + x + x + x + 1

  constant ccitt_crc_32_poly : std_logic_vector(32 downto 0)
                             := "100000100110000010001110110110111";

  -- Note that in this implementation, it is assumed that the MSB of the data is the
  -- first to be received.  In some interfaces (e.g. UARTs) this may not be the case
  -- and it may be necessary to flip the bit (or even byte) order to get the correct
  -- result...

  function ccitt_crc_32(d,c : std_logic_vector) return std_logic_vector is
  begin
    return crc(ccitt_crc_32_poly,d,c);
  end;

  --                     8   2
  -- ATM HEC Poly is:  x + x + x + 1

  constant atm_hec_poly : std_logic_vector(8 downto 0)
                        := "100000111";

  -- Note that for ATM, the CRC is initialised to zero and 0x55 is added to the result...
  -- It is also assumed that the data input is 32-bits wide (i.e. ATM Header)...

  function atm_hec_crc(d : std_logic_vector) return std_logic_vector is
  begin
    return crc(atm_hec_poly,d,x"00") xor x"55";
  end;

end crc_pkg;
```