RTL Code for Sinc^3 Decimation Filter in VHDL

(written and compiled in Altera Quartus II 9.1 sp1 Web Edition)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;


entity AD7401 is
port
       (
       clk20,clk5,reset,mdata,WRinit     : in std_logic;
       DATA                              : out std_logic_vector(15 downto 0);
              -- ADC's final 16-bit samples (2's complement)

       WRram                             : out std_logic_vector(7 downto 0);      -- Write Address (RAM)
       wren                              : out std_logic                          -- WriteEnable (RAM)
       );
end AD7401;


architecture RTL of AD7401 is

signal ACC1, ACC2, ACC3, ACC3_d                : std_logic_vector(23 downto 0)   := (others => '0');
       -- Accumulators

signal DIFF1, DIFF2, DIFF3, DIFF1_d, DIFF2_d   : std_logic_vector(23 downto 0)   := (others => '0');
       -- Differentiators

signal delta                                   : std_logic_vector(23 downto 0)   := (others => '0');
       -- Interface

signal write_count                             : std_logic_vector(9 downto 0)    := (others => '0');
       -- WriteEnable vector

signal word_count                              : std_logic_vector(7 downto 0)    := (others => '0');
       -- Decimation rate vector

signal WRcount                                 : std_logic_vector(7 downto 0)    := (others => '0');
       -- RAM Write Address counter

signal word_clk, WriteEnable                   : std_logic                       := '0';
       -- clock_enable for the decimation process + write_enable (RAM)

begin

process(clk5,reset)                                        -- Serial interface (delta modulation)
begin
  if reset = '0' then
    delta <= (others => '0');
  elsif clk5'event AND clk5 = '1' then
    if mdata = '0' then
      delta <= std_logic_vector(to_unsigned(0,24));   -- equal to: <= "000000000000000000000000";
    else
      delta <= std_logic_vector(to_unsigned(1,24));   -- equal to: <= "000000000000000000000001";
    end if;
  else
    NULL;
  end if;
end process;

process(clk5,reset)                                        -- Accumulators (IIR)
begin
  if reset = '0' then
    ACC1 <= (others => '0');
    ACC2 <= (others => '0');
    ACC3 <= (others => '0');
  elsif clk5'event AND clk5 = '1' then
```

```
    ACC1 <= ACC1 + delta;
    ACC2 <= ACC2 + ACC1;
    ACC3 <= ACC3 + ACC2;
  else
    NULL;
  end if;
end process;

process(clk5,reset)
begin
  if reset ='0' then
    word_count <= (others => '0');
  elsif clk5'event AND clk5 = '0' then
    word_count <= word_count + '1';                --  word_count is an 8-bit reg for a decimation ratio of 2^8= 256
  else
    NULL;
  end if;
end process;

process(word_count)                                -- ClockEnable process (DECIMATION)
begin
  if word_count = x"FF" then
    word_clk <= '1';
  else
    word_clk <= '0';
  end if;
end process;

process(clk5,reset)                                -- Differentiators (FIR)
begin
  if reset = '0' then
    DIFF1  <= (others => '0');
    DIFF2  <= (others => '0');
    DIFF3  <= (others => '0');
    ACC3_d <= (others => '0');
    DIFF1_d <= (others => '0');
    DIFF2_d <= (others => '0');
  elsif clk5'event AND clk5 = '1' then
    if word_clk = '1' then
      DIFF1   <= ACC3 - ACC3_d;
      DIFF2   <= DIFF1 - DIFF1_d;
      DIFF3   <= DIFF2 - DIFF2_d;
      ACC3_d  <= ACC3;
      DIFF1_d <= DIFF1;
      DIFF2_d <= DIFF2;
    else
      NULL;
    end if;
  else
    NULL;
  end if;
end process;

process(clk5,reset)                                -- DATA assignment process
begin
  if reset = '0' then
    DATA <= (others => '0');
  elsif clk5'event AND clk5 = '1' then
    if word_clk = '1' then                         -- extracting the Most Significant Bits - MSB

--     DATA(15 downto 8) <= CN5(15 downto 8);      -- RAM outputs DATA inversely
--     DATA(7 downto 0)  <= CN5(23 downto 16);     -- would look like this if we needed normal output

       DATA <= (DIFF3(23 downto 8) - x"8000");

       -- WAV at the PC needs LSByte first, so we keep the inverse MSByte/LSByte RAM output !!
       -- WAV needs 2's complement implementation, achieved by XORing decimator's output with 0x8000

    else
      NULL;
    end if;
  else
    NULL;
  end if;
end process;


-- RAM writing process

wren  <= WriteEnable;
WRram <= WRcount;

process(clk20,reset)
begin
  if reset ='0' then
    write_count <= (others => '0');
  elsif clk20'event AND clk20 = '0' then
    write_count <= write_count + '1';   -- write_count is a 10-bit reg for sync of RAM writing process
  else
    NULL;
  end if;
end process;

process(write_count)                    -- WriteEnable process (RAM)
begin
  if write_count = x"2FF" then
    WriteEnable <= '1';
  else
    WriteEnable <= '0';
  end if;
end process;

process(clk20,reset)
begin
  if reset = '0' then
       WRcount <= (others => '0');
  elsif clk20'event AND clk20 = '1' then

    if WriteEnable = '1' then
         WRcount <= WRcount + '1';
       else
         NULL;
       end if;

       if WRinit = '1' then             -- signal sent from TxD block to initialize RAM writing process
         WRcount <= (others => '0');
       else
         NULL;
       end if;

  else
       NULL;
  end if;
end process;

end RTL;
```