20 June 2008

Create a VHDL entity to implement a chronometer with tenth of a second precision.

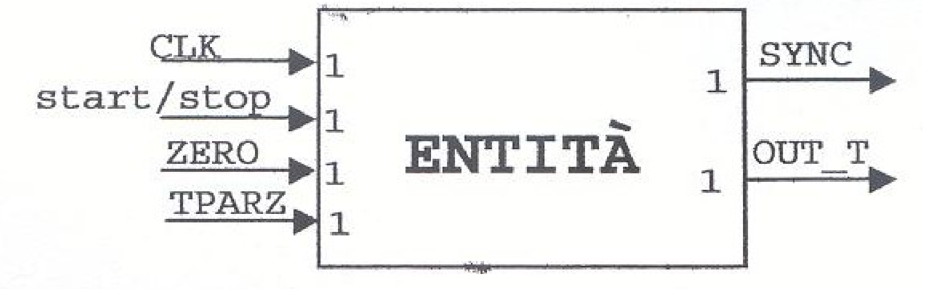
The format is mm:ss:d – The chronometer will be able to count from 0 to 59 minutes, 59 seconds and 9 tenths of a second.

An input (**ZERO**) reset the count. Another input (**start/stop**) is used to stop and restart the count.

An additional input (**TPARZ**) is used to memorize up to 4 partial times during counting; when the counting has been stop (by **start/stop**), **TPARZ** enables the transfer of the total time counted to a serial output (**OUT_T**) followed by the 4 partial times saved (if any is saved). A synchronism signal (**SYNC**) emits 1 pulse at the start of each time value transferred to **OUT_T**.

OUT_T values are 4-bits, BCD-coded, accounting to a total of 20 bits for each chronometer's value.

Finally, **OUT_T** must be at high impedance in any other case.



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Chrono is
   port (CLK, START_STOP, ZERO, TPARZ: in std_logic;
         SYNC, OUT_T: out std_logic);
end Chrono;

architecture Behavioral of Chrono is

   --TIMES: to store the current time count and the partial time counts;
   --time count is stored in the MSBs (20 bits);
   --partial time counts are stored in the bits which come after.
   signal TIMES: std_logic_vector (99 downto 0):= (others => '0');

   --to count the number of sampled partial times
   signal cntPT: integer range 0 to 5:=0;
   signal state: std_logic:='0';
   signal cnt: integer range 0 to 100:=0;

   signal TENTHS: integer range 0 to 11:=0;          --Tenths of seconds.
   signal M1, M2, S1, S2: integer range 0 to 11:=0;  --Minutes and seconds.

   begin

      process (CLK, START_STOP, ZERO, TPARZ)
      begin
         if (ZERO = '1') then
            --General reset:
            TIMES <= (others => '0');
            cntPT <= 0;
            cnt <= 0;
            TENTHS <= 0;
            M1 <= 0;
            M2 <= 0;
            S1 <= 0;
            S2 <= 0;

         elsif (rising_edge (CLK)) then
            if ((START_STOP = '0') and (state = '0')) then
               OUT_T <= 'Z';
               SYNC <= 'Z';

            elsif (START_STOP = '1') then
               state <= '1';  --At least one time is counted.
               OUT_T <= 'Z';  --While the entity is counting, the output must be set to
               SYNC <= 'Z';   --high impedance.

               ------------------------------------------------------------------
               --Count (IT WORKS BUT I DON'T LIKE IT!):
               if (M2 < 6) then
                  if (M1 < 10) then
                     if (S2 < 6) then
                        if (S1 < 10) then
                           if (TENTHS < 9) then TENTHS <= TENTHS + 1; end if;
                           if (TENTHS = 9) then
                              if (M2 /= 5) then TENTHS <= 0; end if;
                              if (S1 < 9) then S1 <= S1 + 1; end if;
                           end if;
                        end if;
                        if (S1 = 9 and TENTHS = 9) then
                           if (M2 /= 5) then S1 <= 0; end if;
                           if (S2 < 5) then S2 <= S2 + 1; end if;
                        end if;
                     end if;
                     if (S2 = 5 and S1 = 9) then
                        if (M2 /= 5) then S2 <= 0; end if;
                        if (M1 < 9) then M1 <= M1 + 1; end if;
                     end if;
                  elsif (M1 = 9 and S2 = 5) then
                     if (M2 /= 5) then M1 <= 0; end if;
                     if (M2 < 5) then M2 <= M2 + 1; end if;
                  end if;
               end if;

               ---------------------------------------------------------------------------
               --Partial times sampling:
               if (TPARZ = '1') then
                  case cntPT is
                     --Partial time #1.
                     when 0 => TIMES(79 downto 60) <= conv_std_logic_vector(M2, 4) &
                                                      conv_std_logic_vector(M1, 4) &
                                                      conv_std_logic_vector(S2, 4) &
                                                      conv_std_logic_vector(S1, 4)  &
                                                      conv_std_logic_vector(TENTHS, 4);
                                                      cntPT <= cntPT + 1;

                     --Partial time #2.
                     when 1 => TIMES(59 downto 40) <= conv_std_logic_vector(M2, 4) &
                                                      conv_std_logic_vector(M1, 4) &
                                                      conv_std_logic_vector(S2, 4) &
                                                      conv_std_logic_vector(S1, 4)  &
                                                      conv_std_logic_vector(TENTHS, 4);
                                                      cntPT <= cntPT + 1;

                     --Partial time #3.
                     when 2 => TIMES(39 downto 20) <= conv_std_logic_vector(M2, 4) &
                                                      conv_std_logic_vector(M1, 4) &
                                                      conv_std_logic_vector(S2, 4) &
                                                      conv_std_logic_vector(S1, 4)  &
                                                      conv_std_logic_vector(TENTHS, 4);
                                                      cntPT <= cntPT + 1;

                     --Partial time #4.
                     when 3 => TIMES(19 downto 0) <=  conv_std_logic_vector(M2, 4) &
                                                      conv_std_logic_vector(M1, 4) &
                                                      conv_std_logic_vector(S2, 4) &
                                                      conv_std_logic_vector(S1, 4)  &
                                                      conv_std_logic_vector(TENTHS, 4);
                                                      cntPT <= cntPT + 1;
                     when others => null;
                  end case;
               end if;
               -------------------------------------------------------------------------

               elsif ((START_STOP = '0') and (state = '1')) then
               --Output bitstream generation:
               if (cnt = 0) then
                  TIMES(99 downto 96) <= conv_std_logic_vector(M2, 4);
                  TIMES(95 downto 92) <= conv_std_logic_vector(M1, 4);
                  TIMES(91 downto 88) <= conv_std_logic_vector(S2, 4);
                  TIMES(87 downto 84) <= conv_std_logic_vector(S1, 4);
                  TIMES(83 downto 80) <= conv_std_logic_vector(TENTHS, 4);
                  cnt <= cnt + 1;
               elsif ((cnt > 0) and (cnt <= 20+20*cntPT)) then
                  OUT_T <= TIMES (99);
                  TIMES <= TIMES (98 downto 0) & 'Z';
                  cnt <= cnt + 1;
                  case cnt is
                     when 1 => SYNC <= '1';
                     when 81 => SYNC <= '1';
                     when 61 => SYNC <= '1';
                     when 41 => SYNC <= '1';
                     when 21 => SYNC <= '1';
                     when others  => SYNC <= '0';
                  end case;
               elsif (cnt > 20+20*cntPT) then
                  OUT_T <= 'Z';
                  SYNC <= 'Z';
               end if;

            end if;

         end if;
      end process;

end Behavioral;
```