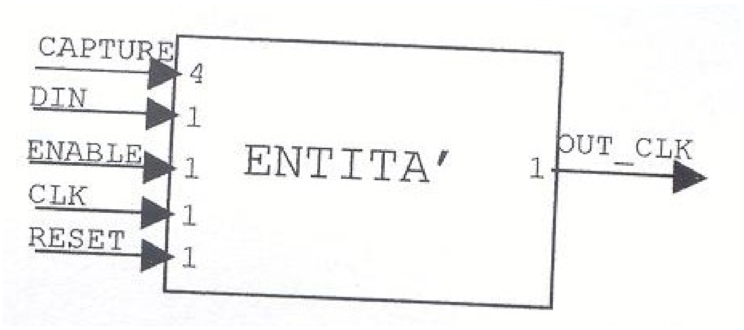
18 April 2008

Describe a circuit to generate a clock signal (**`OUT_CLK`**) whose period is described by the input signals as follows:

Starting from a reset, the 4-bits signal **`CAPTURE`** is sampled for 4 clock cycles, this signal provides the BCD-coded positions of the bits to read from the input signal **`DIN`** and the 4 values obtained form the division factor (still in BCD format) for the clock signal to determine **`OUT_CLK`**.

`OUT_CLK` must start after 16 clock cycles from the reset and must be kept until a new reset.

When the activation signal **`ENABLE`** is equal to 0, the output must be set to high impedance without changing the function of the circuit!



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

entity clk_generator is
   port(CLK, EN, RST, DIN: in std_logic;
        CAPTURE: in std_logic_vector (3 downto 0);
        OUT_CLK: out std_logic);
end clk_generator;

architecture Behavioral of clk_generator is
   --regIN: to store serial input.
   --div: for saving the division factor.
   --W1, W2, W3, W4: for saving the bits position in regIn.
   signal regIn: std_logic_vector (15 downto 0):=(others => '0');
   signal div: std_logic_vector (3 downto 0):="0000";
   signal W1, W2, W3, W4: integer range 0 to 20:=0;

   --cnt: for uploading "regIn", "div" and "W" signals.
   --cnt_out, limit: for the output timing.
   signal cnt: integer range 0 to 20:=0;
   signal cnt_out: integer range 0 to 10:=1;
   signal limit: natural:=0;
   begin

      process (CLK, RST, EN)
      begin

         if (RST = '1') then
            --General reset.
            OUT_CLK <= 'Z';
            regIn <= (others => '0');
            div <= (others => '0');
            cnt <= 0;
            cnt_out <= 1;   --It will be seen later the reason why it is initialized to 1.

         elsif rising_edge(CLK) then
            if (EN = '0') then      --When the entity is disabled, the output is kept in
               OUT_CLK <= 'Z';      --high impedance status.

            elsif (EN = '1') then
               if (cnt <= 16) then
                  --16 values are uploaded; for the first 4 clock pulses,
                  --4 occurrence of "CAPTURE" are stored also.
                  case cnt is
                     when 0 => regIn <= regIn(14 downto 0) & DIN;
                               W1 <= conv_integer(CAPTURE);
                               cnt <= cnt + 1;

                     when 1 => W2 <= conv_integer(CAPTURE);
                               regIn <= regIn (14 downto 0) & DIN;
                               cnt <= cnt + 1;

                     when 2 => W3 <= conv_integer(CAPTURE);
                            regIn <= regIn (14 downto 0) & DIN;
                            cnt <= cnt + 1;

                     when 3 => W4 <= conv_integer(CAPTURE);
                               regIn <= regIn (14 downto 0) & DIN;
                               cnt <= cnt + 1;

                     --For the rest of the clock pulses the only thing that
                     --is done is the the storage of serial input.
                     when 4 to 15 => regIn <= regIn(14 downto 0) & DIN;
                                     cnt <= cnt + 1;
                     --Division factor computation:
                     --once "regIn" is filled, "div" is computed.
                     --This cannot be done before the 17th clock pulse.
                     --NOTE: Since the output must be computed after 16 clock pulses from the
                     --last reset, the first output bit is computed in this context.
                     --If this is not done one clock pulse delay occurs.
                     --That's why "cnt_one" has been initialized to 1.
                     when 16 => div(0) <= regIn (w1);
                                div(1) <= regIn (w2);
                                div(2) <= regIn (w3);
                                div(3) <= regIn (w4);
                                OUT_CLK <= '0';
                                cnt <= cnt + 1;


                     when others => null;
                  end case;

               elsif (cnt > 16) then
                  --Output bitstrem generation:
                  if (cnt_out < limit) then
                     OUT_CLK <= '0';
                     cnt_out <= cnt_out + 1;

                  elsif ((cnt_out >= limit) and (cnt_out < ((2*limit)-1)) )then
                     OUT_CLK <= '1';
                     cnt_out <= cnt_out + 1;

                  --At the end, "cnt_out" is set to 0 again so that every instruction above
                  --(for the Output bitstream generation) can be repeated and
                  --the output bitstream can be generated until a general reset:
                  elsif (cnt_out = ((2*limit)-1) ) then
                     cnt_out <= 0;

                  end if;

               end if;
            end if;
         end if;

      end process;

--This is done outside from the process because if this is done within the process
--a clock pulse delay occurs (before computing "limit", "div" must be updated and this
--cannot be done in just one clock pulse, within the process)
limit <= conv_integer(div) when (cnt > 16) else limit;

end Behavioral;
```

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

--one serial input, DIN: 16 bit should be stored.
--one 4 bits parallel input, CAP: 4 occurrences should be stored along with
--the first 4 bits of DIN (during the first 4 clock periods of time):
--4 BDC codes which represent the position of 4 bits in DIN bitstream
--this latter bits represent how long OUT_CLK semi-period should be, for
--instance: if this 4 bits encode the number N, OUT_CLK period should be
--2N clock periods long.

--Problems: 1) BCD is used to encode decimal numbers (0 to 9) with 4 bits, therefore
--it doesn't make any sense to store 16 bits from DIN... It follows that 16 decimal
--values must be encoded (it's not BCD);
--2) if clock frequency is not halved at least, it is impossible to produce the required
--output signal after 16 clock periods of time...

entity CLKGEN is
   port (CAP: in std_logic_vector(3 downto 0);
         CLK, DIN, EN, RST: in std_logic;
         OUT_CLK: out std_logic);
end CLKGEN;

architecture Behavioral of CLKGEN is

signal reg: std_logic_vector (15 downto 0):=(others => 'Z');
signal cnt: integer range 0 to 16:=0;
signal b0, b1, b2, b3: integer range 0 to 16:=0;
signal buff: std_logic:='Z';
signal DIV: std_logic_vector (3 downto 0):="0000";
signal cntOut, LIM: std_logic_vector(4 downto 0):="00000";
begin

process (CLK, RST)
begin
--1) Reset:
if (RST = '1') then
   reg <= (others => 'Z');
   cnt <= 0;
   b0 <= 0;
   b1 <= 0;
   b2 <= 0;
   b3 <= 0;
   buff <= 'Z';
   DIV <= "0000";
   cntOut <= "00000";
   LIM <= "00000";
elsif (rising_edge(CLK)) then
   --2) first 4 DIN bits and 4 CAP occurrences are stored
   --and OUT_CLK (buff) is kept in high impedance state(*):
   if (cnt = 0) then
      b0 <= CONV_INTEGER(CAP);
      reg <= reg (14 downto 0) & DIN;
      cnt <= cnt + 1;
      buff <= 'Z';
   --(*)
   elsif (cnt = 1) then
      b1 <= CONV_INTEGER(CAP);
      reg <= reg (14 downto 0) & DIN;
      cnt <= cnt + 1;
      buff <= 'Z';
   --(*)
   elsif (cnt = 2) then
      b2 <= CONV_INTEGER(CAP);
      reg <= reg (14 downto 0) & DIN;
      cnt <= cnt + 1;
      buff <= 'Z';
   --(*)
   elsif (cnt = 3) then
      b3 <= CONV_INTEGER(CAP);
      reg <= reg (14 downto 0) & DIN;
      cnt <= cnt + 1;
      buff <= 'Z';
   --3) 11 more DIN bits are stored (buff='Z'):
   elsif (cnt > 3 and cnt < 15) then
      reg <= reg (14 downto 0) & DIN;
      cnt <= cnt + 1;
      buff <= 'Z';
   --4) as the last DIN bit is stored, the entity begins to produce the output:
   --buff is set to '1';
   --DIV is not ready yet, one last bit misses and it will be in reg(0): this means that
   --this moment this bit is not available from reg, but it is available from DIN;
   --furthermore, the entity should be able to extract the bits in the positions b_i
   --(previously computed) and these bits are currently in the positions (b_i - 1), in
   --fact, after the last bit comes from DIN, these bits are shifting in the b_i positions.
   --NB: b_i are computed assuming that all of the 16 bits are available.
   elsif (cnt = 15 ) then
      reg <= reg (14 downto 0) & DIN;
      buff <= '1';
      --------------------
      --LIM=2DIV (left-shift of every bit and LSB=0)
      if (b3 = 0) then
         DIV(3) <= DIN;
         LIM(4) <= DIN;
      else
         DIV(3) <= reg(b3-1);
         LIM(4) <= reg(b3-1);
      end if;
      if (b2 = 0) then
         DIV(2) <= DIN;
         LIM(3) <= DIN;
      else
         DIV(2) <= reg(b2-1);
         LIM(3) <= reg(b2-1);
      end if;
      if (b1 = 0) then
         DIV(1) <= DIN;
         LIM(2) <= DIN;
      else
         DIV(1) <= reg(b1-1);
         LIM(2) <= reg(b1-1);
      end if;
      if (b0 = 0) then
         DIV(0) <= DIN;
         LIM(1) <= DIN;
      else
         DIV(0) <= reg(b0-1);
         LIM(1) <= reg(b0-1);
      end if;
      --------------------
      LIM(0) <= '0';
      cntOut <= cntOut + 1;
      cnt <= cnt + 1;
   --once all of the 16 DIN bits are stored (cnt>15), output is processed:
   elsif (cnt > 15) then
      --special case: OUT_CLK='1' indefinitely (cntOut is not incremented):
      if (DIV = "0000") then
         buff <= '1';
      --first semi-period:
      elsif (cntOut < DIV) then
         buff <= '1';
         cntOut <= cntOut + 1;
      --second semi-period (**):
      elsif (cntOut >= DIV and cntOut < (LIM - 1)) then
         buff <= '0';
         cntOut <= cntOut + 1;
      --(**) at the end of the OUT_CLK period cntOut is set to zero
      --so that a new period of OUT_CLK can be produced
      --(cntOut is processed as unsigned since the "unsigned" package is used)
      elsif (cntOut = LIM-1) then
         buff <= '0';
         cntOut <= "00000";
      end if;
   end if;
end if;
end process;

OUT_CLK <= buff when (EN = '1') else 'Z';

end Behavioral;
```