Zegar w VHDL na CPLD

Do skonstruowania opisywanego układu zegara zachęcił mnie projekt „Nixie Clock" prezentowany na łamach EP2-3/2003 autorstwa Piotra Zbysińskiego, który przedstawił implementację zegara przy użyciu języka opisu sprzętu VHDL. Postanowiłem i ja spróbować swoich sił w projektowaniu przy wykorzystaniu VHDL, jednak w swoim zegarze zamiast lamp Nixie, głównie ze względów konstrukcyjnych, zastosowałem zespół wyświetlaczy LED sterowanych multipleksowo. Dodatkowo, przy projektowaniu układu korzystałem z opisu projektu „Sterownik wyświetlacza multipleksowanego w VHDL" z EP 9/2002.

Zegar zbudowany jest z dwóch modułów: płytki bazowej oraz płytki wyświetlacza. Płytka bazowa zawiera: układ CPLD, generator zegarowy, elementy układu zerującego, zasilającego i filtrującego napięcie, złącze JTAG do programowania układu, oraz elementy układu klawiatury sterującej. Schemat ideowy układu pokazano na **rys. 1**.

W zegarze zastosowałem układ CPLD typu XC9572XL firmy Xilinx, zawierający 72 makrokomórki. Jest on przystosowany do zasilania napięciem 3,3 V, jednak istnieje możliwość podłączania do niego układów cyfrowych zasilanych napięciem 5 V. Układ zastosowany w zegarze ma obudowę PLCC44.

Ciekawym elementem jest generator częstotliwości wzorcowej, ponieważ na potrzeby projektu zastosowałem układ DS32kHz firmy Dallas-Maxim. Jest to generator częstotliwości 32,768 kHz z kompensacją temperaturową, o możliwości zasilania napięciem z przedziału 2,7...5,5 V, a więc świetnie nadającym się do zastosowania razem z układem CPLD zasilanym napięciem 3,3V.

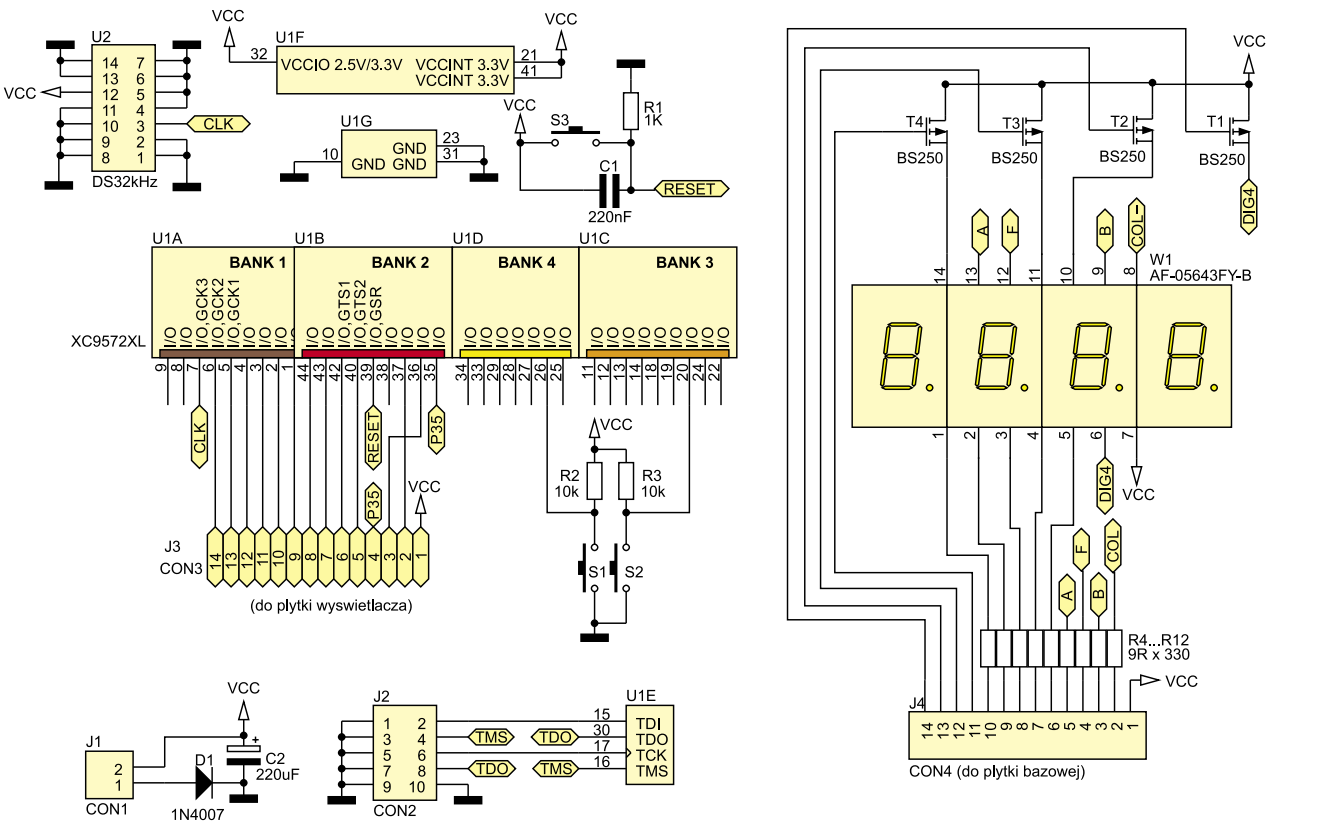
W zegarze zastosowałem 4-cyfrowy wyświetlacz LED ze wspólną anodą o konstrukcji przystosowanej do sterowania multipleksowanego. Jako tranzystory przełączające użyłem BS250. W celu ograniczania prądu segmentów sterowanych wyświetlaczy włączyłem rezystory o wartości 330 Ω. Płytki obu modułów są ze sobą połączone złączem goldpin, tworząc jednocześnie konstrukcyjną całość.

Bloki funkcjonalne

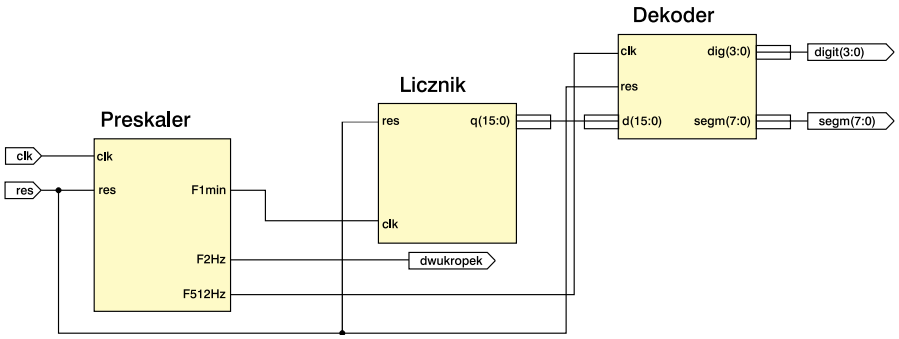
Do opisu zegara wykorzystałem język VHDL. Wielką zaletą tego języka opisu sprzętu jest jego uniwersalność i możliwość przenoszenia pomiędzy różnymi systemami do syntezy logicznej oraz układami programowalnymi wytwarzanymi przez różnych producentów. Gotowe bloki funkcjonalne zaimplementowane w tym projekcie, można z powodzeniem wykorzystać w swoich projektach, ewentualnie z niezbędną korektą, podyktowaną koniecznością dopasowania do potrzeb realizacji danego zadania.

Schemat blokowy zegara (**rys. 2**) składa się z trzech części: preskalera, licznika oraz dekodera. Preskaler dzieli częstotliwość wzorcową 32,768 kHz za pomocą zaimplementowanego w nim licznika na poszczególne częstotliwości taktujące pozostałe bloki funkcjonalne. I tak, częstotliwość *F512Hz* służy do sterowania poprzez dekoder, modułu wyświetlaczy multipleksowanych. Częstotliwość *F1min* taktuje moduł licznika, a częstotliwość *F2Hz* odpowiada za gaszenie i zapalanie dwukropka na wyświetlaczu LED. Moduł preskalera opisanego w języku VHDL umieszczono na **list. 1**.

Moduł licznika (**list. 2**) jest odpowiedzialny za zliczanie czasu. Zaimplementowano w nim cztery równoległe procesy (*jednmin*, *dziesmin*, *jedngodz*, *dziesgodz*). Każdy z tych procesów składa się z 4-bitowego licznika, sprawdza dość rozbudowane wyrażenia warunkowe oraz ma wpływ na pozostałe procesy. Sprawdzanie warunków odbywa się co 1 min, sterowane sygnałem taktującym moduł licznika, pochodzącym z preskalera. Moduł licznika na wyjściu udostępnia stany swoich czterech liczników binarnych z poszczególnych procesów odpowiedzialnych za zliczanie czasu. Z kolei



Rys. 1. Schemat ideowy zegara



Rys. 2. Schemat blokowy zegara

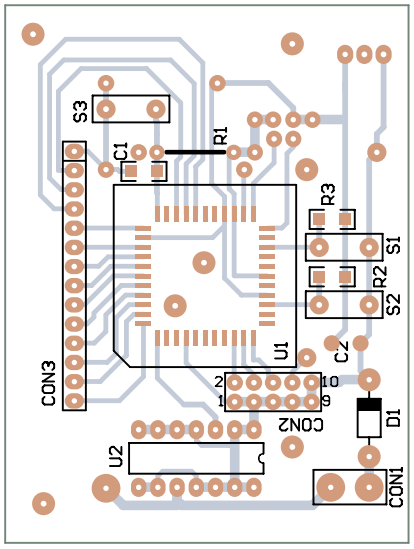
stany wyjściowe modułu licznika *q(15:0)* stanowią podstawę do działania modułu dekodera. W nim to odbywa się proces dekodowania i wyświetlania poszczególnych liczb na wyświetlaczu LED. Stąd też zaimplementowano w tym module dekoder

List. 1. Preskaler

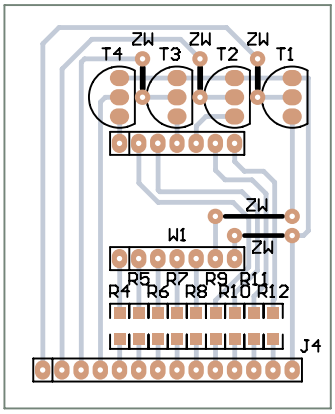
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Preskaler is
    Port ( clk,res : in std_logic;
   F512Hz,F1min,F2Hz: out std_logic);
end Preskaler;

architecture Behavioral of Preskaler is
signal q:std_logic_vector(20 downto 0);
begin
        process(clk,res)
begin
        if res='1' then
                q<="00000000000000000000";
                elsif clk'event and clk='1' then
                        q<=q+1;
                        if q="11110000000000000001" then
                                q<="00000000000000000000";
                        end if;
                        end if;
end process;
F512Hz<=q(6);
F1min<=q(20);
F2Hz<=q(15);
end Behavioral;
```



Rys. 3. Schemat montażowy płytki bazowej



Rys. 4. Schemat montażowy płytki wyświetlacza

Na CD: karty katalogowe i noty aplikacyjne elementów oznaczonych na wykazie elementów kolorem czerwonym

Wykaz elementów
Płytka bazowa
Rezystory:
R1: 1 kΩ
R2, R3: 10 kΩ
Kondensatory:
C1: 220 nF
C2: 220 µF/16 V
Półprzewodniki:
U1: XC9572XL PLCC44
U2: DS32kHz
D1: 1N4007
Inne:
S1, S2, S3: mikroswitch
CON1: złącze mini-jack (do zasilacza)
CON2: złącze żeńskie goldpin 2×5 (JTAG)
CON3: złącze żeńskie goldpin 1×14
Płytka wyświetlacza
R4...R12: 330 Ω
W1: AF-05643FY-B
T1...T4: BS250
CON4: wtyk kątowy goldpin 1×14

wyświetlacza 7-segmentowego LED, oraz procesy odpowiedzialne za sterowanie wyświetlaczem multipleksowanym (**list. 3**). Wyjścia dekodera sterują bezpośrednio płytkę wyświetlacza.

Schematy montażowe płytek drukowanych przedstawiono na **rys. 3** i **4**.

Podsumowanie

Opisywany projekt przygotowałem i skompilowałem przy pomocy bezpłatnego programu WebPack ISE firmy Xilinx. Wykorzystanie zasobów zastosowanego układu CPLD po skompilowaniu wyniosło 51 makrokomórek z 72 dostępnych (wersja programu 10.1).

Sam proces projektowania polegał na przygotowaniu (opisaniu w VHDL) poszczególnych bloków funkcjonalnych, utworzeniu z nich symboli graficznych, połączeniu symboli ze sobą w edytorze graficznym,

List. 2 Licznik

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

entity Licznik is
    Port (    res :       in  std_logic;
              clk :       in  std_logic;
                  q :     out std_logic_vector(15 downto 0));
end Licznik;

architecture Behavioral of Licznik is
signal jednmin:   std_logic_vector(3 downto 0);
signal dziesmin:  std_logic_vector(3 downto 0);
signal jedngodz:  std_logic_vector(3 downto 0);
signal dziesgodz: std_logic_vector(3 downto 0);

begin
jedn_cnt: process (res,clk) begin
         if res='1' then jednmin <="0001";
                  elsif (clk='0'and clk'event) then
                                    if jednmin="1001" then jednmin<="0000";
                                                else jednmin <= jednmin+1;
                                    end if;
                                    end if;
end process jedn_cnt;

dz_cnt: process (clk,res) begin
         if res='1' then dziesmin <="0001";
                  elsif clk='0' and clk'event then
                           if dziesmin="0101" and jednmin="1001" then dziesmin
<="0000";
                  elsif jednmin="1001" then dziesmin<=dziesmin+1;
                           end if;
                           end if;
end process dz_cnt;

set_cnt: process (clk, res) begin
         if res='1' then jedngodz<="0001";
                  elsif clk='0' and clk'event then
                           if (jedngodz="1001" and dziesmin="0101" and
jednmin="1001") or (dziesgodz="0010" and jedngodz="0011" and dziesmin="0101"
and jednmin="1001") then jedngodz<="0000";
                                    elsif dziesmin="0101" and jednmin="1001"
then jedngodz<=jedngodz+1;
                           end if;
                           end if;
end process set_cnt;

tys_cnt: process (clk, res) begin
         if res='1' then dziesgodz<="0010";
                  elsif clk='0' and clk'event then
                           if dziesgodz="0010" and jedngodz="0011" and
dziesmin="0101" and jednmin="1001"
                          then dziesgodz<="0000";
                                    elsif jedngodz="1001" and dziesmin="0101"
and jednmin="1001" then
                                      dziesgodz<=dziesgodz+1;
                           end if;
                           end if;
end process tys_cnt;

q<=dziesgodz & jedngodz & dziesmin & jednmin;

end Behavioral;
```

List. 3 Dekoder

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

entity Dekoder is Port (
        d: in std_logic_vector(15 downto 0);
        clk, res: in std_logic;
        dig: out std_logic_vector(3 downto 0);
        segm: out std_logic_vector(7 downto 0)
         );
end Dekoder;

architecture reg_dek of Dekoder is
signal adr: std_logic_vector(1 downto 0); -- linie adresowe MUX-a
signal bcd: std_logic_vector(3 downto 0); -- wyj. MUX-a/wej. dekodera

begin
-- dekoder wyswietlanych cyfr
with adr select
    dig <= „1110” when „00”,    -- cyfra 0 (jedn.)
           “1101” when “01”,    -- cyfra 1 (dzies.)
           “1011” when “10”,    -- cyfra 2 (setki)
           “0111” when “11”,    -- cyfra 3 (tys.)
           “1111” when others;   -- wygaszenie cyfr


-- MUX danych do wyswietlania
with adr select
   bcd <= d(15 downto 12) when „11”,
          d(11 downto 8)  when „10”,
            d(7 downto 4)   when „01”,
          d(3 downto 0)   when „00”,
          “1111” when others;

-- licznik adresu wyswietlanej cyfry
licz_adr: PROCESS (clk, res) BEGIN
        IF (res=’1’) THEN adr  <= „00”;
                ELSIF (clk=’1’ AND clk’EVENT) THEN
                        adr <= adr + 1;
        END IF;
END PROCESS licz_adr;


-- dekoder 8-segmentowy
with bcd select
--       g f e d c b a
   segm <= „11000000” when „0000”,    -- 0
           „11111001” when „0001”,    -- 1
           „10100100” when „0010”,    -- 2
           „10110000” when „0011”,    -- 3
           „10011001” when „0100”,    -- 4
           „10010010” when „0101”,    -- 5
           „10000010” when „0110”,    -- 6
           „11111000” when „0111”,    -- 7
           „10000000” when „1000”,    -- 8
           „10010000” when „1001”,    -- 9
           „11111111” when others;    -- wygaszenie

end reg_dek;
```

oraz skompilowaniu tak przygotowanego projektu. Do zaprogramowania układu wykorzystałem prosty programator podłączany do komputera poprzez złącze drukarki.

Elementem, którego nie udało mi się zrealizować w przedstawionym projekcie, była obsługa klawiatury sterującej ustawianiem czasu, poniekąd pozostałe zasoby logiczne (21 makrokomórek) powinny taką implementację bezproblemowo umożliwić.

Na zakończenie chciałbym zachęcić wszystkich kolegów elektroników do zmierzenia się z projektowaniem urządzeń przy użyciu układów programowalnych CPLD i języka opisu sprzętu VHDL. Możliwość korzystania z przygotowanych wcześniej gotowych (opisanych) bloków funkcjonalnych i ich bezproblemowe użycie jako symbole graficzne w kolejnych projektach stanowią naprawdę duże udogodnienie.

Norbert Szyrej
n.szyrej@gmail.com