VHDL			
Dominik Wróbel	**8 I 2018**	**Pn 9:30**	

Spis Treści

1. Zajęcia nr 1

1.1. Zapalanie LED przy zmianie stanu SWITCH

Opis działania programu

Diody połączone są z przyciskami, zmiana stanu przycisku powoduje zapalenie lub zgaszenie diody.

Kod źródłowy (najważniejsze fragmenty)

```
entity Diody is --Blok entity - deklaracja sygnałów wejściowych i wyjściowych
   Port (
                        SW : in STD_LOGIC_VECTOR (15 downto 0); --Deklaracja sygnału
wejściowege – switche jako typ logic_vector
                        LED : out STD_LOGIC_VECTOR (15 downto 0) -- Deklaracja sygnału
wyjściowego - diody jako typ logic_vector
                    );
end Diody;

architecture Behavioral of Diody is
begin
LED <= SW; -- Przypisanie sygnału ze switchy do odpowiadających LED
```

Wnioski

Instrukcje w architekturze są wykonywane równolegle, co odpowiada fizycznym połączeniem na płytce FPGA, dzięki temu zapalenie diody przy pomocy przycisków może być łatwo zrealizowane przy pomocy jednej instrukcji.

1.2. Statyczne wyświetlanie numeru stanowiska na wyświetlaczu 7-segmentowym, stanowisko numer 6

Opis działania programu

Program wyświetla na jednym z wyświetlaczy 7-segmentowych numer 6, jest to wyświetlanie statyczne, aby móc to wykonać zapoznać się należy najpierw architekturą płytki i połączeniami w wyświetlaczu – wspólna anoda oraz tranzystory PNP.

Kod źródłowy (najważniejsze fragmenty)

```
entity StaDisplay7SEG is -- deklaracja sygnałów wejściowych i wyjściowych
   Port (
                              SEG : inout STD_LOGIC_VECTOR (7 downto 0); -- Sygnał SEG
odpowiada za 8 LED znajdujacych się na jednym wyświetlaczu
          AN : inout STD_LOGIC_VECTOR (7 downto 0) -- Na płytce wyświetlacze są połączone w
układzie wspólnej anody więc wybór
                           ); -- wyświetlacza odbywa się przy pomocy anod
end StaDisplay7SEG;

architecture Behavioral of StaDisplay7SEG is -- wyświetla statycznie numer 6

begin
AN <= "11111110"; -- Anody są sterowane przy pomocy tranzystorów PNP wiec jeśli podamy stan
niski to otrzymamy stan wysoki na anodzie
-- Ta instrukcja aktywuje wiec jedna anodę
SEG <= "10000010";  -- Aby mógł płynąć prąd i dioda mogła świecić po ' drugiej stronie ' diody
musi być stan niski
 -- Ten zapis odpowiada po kolei diodom DP G F E D C B A wyświetlacza 7-segmentowego
end Behavioral;
```

Wnioski

Wyświetlanie statyczne nie wymaga tylko ustawienia wartości dwóch sygnałów, jednak zanim przystąpi się do ich ustawienia konieczne jest poznanie architektury płytki, a w szczególności połączenia LED (wspólna anoda) oraz ich sterowania (tranzystory PNP), na tej podstawie można stwierdzić czy wartość danego bitu ma wynosić 0 czy 1.

1.3. Programowa realizacja bramek logicznych

Opis działania programu

Program realizuje prostą logikę poprzez programowe bramki logiczne. Wejścia są podawane przy pomocy przełączników, a wyjścia można obserwować na LED.

Kod źródłowy (najważniejsze fragmenty)

```
architecture Behavioral of BramkiLog is -- blok odpowiedzialny za logikę bramek
-- wykorzystuje dostepne w jezyku VHDL funkcje logiczne takie jak NOT , OR , AND, XOR
begin
LED(0) <= SW(0) AND SW(1) ; -- Bramka AND
LED(1) <= NOT ( SW(0) AND SW(1) ); -- Bramka NAND
LED(2) <= SW(0) OR SW(1); -- Bramka OR
LED(3) <= NOT ( SW(0) OR SW(1) ); -- Bramka NOR
LED(4) <= SW(0) XOR SW(1); -- Bramka XOR
LED(5) <= NOT ( SW(0) XOR SW(1) ); -- Bramka XNOR
LED(6) <= NOT SW(1); -- Inwerter
-- Funkcje logiczne działają na typach logicznych, dlatego ich argumentami są sygnały typu LOGIC
end Behavioral;
```

Wnioski

Program działa poprawne, należy pamiętać, że instrukcje na FPGA wywołują się równolegle, dlatego zmiana jednego z przełączników może powodować równoległą zmianę wielu stanów LED, zależnie od logiki.

2. Zajęcia nr 2

2.1. Miganie diodą z częstotliwością 6 Hz z wykorzystaniem licznika

Opis działania programu

Program miga diodą z częstotliwością równą numerowi stanowiska, wykorzystujemy przy tym fakt, że określone wyjście licznika zmienia się co określony czas (im starszy bit tym wolniej), miganie jest zrealizowane poprzez dobór odpowiedniego wyjścia licznika, najbardziej zbliżonego do częstotliwości zadanej, obliczenia w komentarzach. W programie wykorzystywane jest sygnał zegarowy 100 MHz.

Kod źródłowy (najważniejsze fragmenty)

```
entity Counter is -- Sygnały wejściowe i wyjściowe
   Port (
                        CLK100MHZ : in STD_LOGIC; -- Zegar wejściowy
                        LED : out STD_LOGIC_VECTOR (15 downto 0) -- Diody do wizualizacji
działania licznika
         );
end Counter;


architecture Behavioral of Counter is

-- Na n-tym wyprowadzeniu częstotliwość migania będzie równa fn = f_clk / 2 ^ ( n+1 )
-- Po podstawieniu fn = 6 Hz oraz f_clk = 100 MHz otrzymano, że najbardziej zbliżonym do 6Hz
-- n będzie n = 26
signal count : STD_LOGIC_VECTOR ( 26 downto 0 ) := "00000000000000000000000000";
-- Sygnał który odpowiada za liczenie impulsów zegara
```

```
-- Jest to licznik 26-bitowy

begin

-- Ten proces odpowiada za liczenie impulsów rosnących, jest wywoływany zmianą stanu sygnału zegara
process(CLK100MHZ)

begin
   if( rising_edge(CLK100MHZ) )then
      count <= count + '1'; -- Dodajemy jedynkę do wektora logicznego więc należy wziąć ją w ' '
   end if;
end process;


LED <= count(26 downto 11); -- MSB będzie diodą migającą z częstotliwością 6Hz

end Behavioral;
```

Wnioski

Program wykorzystuje procesy, czyli fragmentu kodu, które są wykonywane gdy zmieni się sygnał zapisany na liście procesu. Determinowanie częstotliwości w sposób taki jak w programie jest bardzo wygodne, bo nie wymaga pisania dużej ilości kodu, ale nie jest bardzo precyzyjne, ma ograniczoną ilość możliwych częstotliwości.

2.2. Licznik zamieniający częstotliwość zegara na częstotliwość o numerze stanowiska, czyli 6 Hz, mruganiem diodą

Opis działania programu

Program zamienia częstotliwość 100 MHz na częstotliwość 6 Hz i z tą częstotliwością mruga diodą. Zadanie takie jest bardzo przydatne, ponieważ w wielu zastosowaniach częstotliwość 100 MHz może być zbyt duża dla danej aplikacji.

Kod źródłowy (najważniejsze fragmenty)

```
architecture Behavioral of Counter is
begin
-- Proces zamienia sygnał zegarowy na częstotliwość 6 Hz
-- mruga diodą z częstotliwością 6 Hz

process(CLK100MHZ) -- Proces wyzwalany zmianą sygnału zegarowego

variable count : integer := 0 ;


begin
   if( rising_edge(CLK100MHZ) )then -- Jeśli wykryje zbocze narastające
      if( count < 8333333 ) then -- f_clk / f_out = 100 MHz / 6 Hz = 16 666 666
         count := count + 1;      -- 16 666 666 / 2 = 8 333 333
         LED <= "0000000000000000"; -- Stan niski diody
      elsif ( count < 16666666 ) then
         count := count +1;
```

```
            LED <= "1111111111111111"; -- Stan wysoki diody
        else
            count := 0; -- Wyzerowanie licznika
    end if;
    end if;
end process;

end Behavioral;
```

Wnioski

W procesie wykorzystano zmienną, która może być używana tylko w tym procesie, nie jest widoczna poza nim. Zmienna w tym przypadku jest typu integer i zlicza ona zbocza narastające zegara (okresy). Wykorzystanie zegara do uzyskania określonej częstotliwości umożliwia uzyskanie precyzyjnej częstotliwości.

2.3. Licznik modulo 6 z resetem asynchronicznym i częstotliwością 6 Hz.

Opis działania programu

Licznik modulo 6, pokazuje aktualną wartość na diodach w kodzie binarnym, zliczanie odbywa się z częstotliwością 6 Hz. Dodatkowo jako Reset został zaimplementowany switch, który zeruje licznik.

Kod źródłowy (najważniejsze fragmenty)

```
entity Counter is -- Deklaracja sygnałów wejściowych i wyjściowych
    Port (
                            CLK100MHZ : in STD_LOGIC; -- Sygnał zegarowy
                            LED : out STD_LOGIC_VECTOR ( 2 downto 0 ); -- Diody do wizualizacji
zliczania, potrzebne 3 bity, bo zliczanie modulo 6
                            SW : in STD_LOGIC_VECTOR ( 15 downto 0 ) -- Switch do reseteowania
asynchronicznego
        );
end Counter;

architecture Behavioral of Counter is

signal counter : STD_LOGIC_VECTOR ( 2 downto 0 ) := "000"; -- Licznik, max wartość równa 6

begin

process(CLK100MHZ) -- Proces wyzwalany zmianą stanu sygnału zegarowego

variable count : integer := 0 ;
variable count_out : integer := 0 ;

begin
    if( rising_edge(CLK100MHZ) )then
        if( count < 8333333 ) then -- Częstotliwość 6 Hz obliczona jak w programie poprzednim
            count := count + 1;
            count_out := 0;
```

```
      elsif ( count < 16666666 ) then
         count := count + 1;
         count_out := 1;
      else
         count := 0;
      end if;
   end if;

   if ( SW(0) = '1' ) then -- Jeżeli switch zostanie wcisnięty to wyzeruj licznik
      counter <= "0000000000000000";
   elsif( count_out = 1 AND count = 8333334 ) then -- Jeżeli zeminiła się zmienna count_out to
zwiększ licznik ( 6 Hz )
      counter <=  counter + '1';
   end if;

end process;

LED <= counter; -- Stan licznika na LEDy

end Behavioral;
```

Wnioski

W tym programie wykorzystano dodatkowy sygnał, który następnie jest przypisywany do LED. Należy pamiętać, że do sygnał może być zmieniamy tylko w jednym procesie, a jego wartość będzie taka jak ostatnio ustawiona w procesie (zmienia się po wykonaniu procesu), co pierwszych wersjach programu powodowało problemy z wyświetlaniem.

3. Zajęcia nr 3

3.1. Wyświetlanie na wyświetlaczach 7-segmentowych dynamicznie bez przekodowania

Opis działania programu

Program realizuje wyświetlanie dynamiczne na wyświetlaczach 7-segmentowych. Wyświetla na każdym wyświetlaczu inną cyfrę, tak, że wszystkie cyfry są wyświetlane jednocześnie. Program nie używa przekodowania więc wartości sygnałów są wpisywane bezpośrednio. Obliczenia w komentarzach.

Kod źródłowy (najważniejsze fragmenty)

```
entity DynamicSEG is -- Deklaracja sygnałów wejściowych i wyjściowych
   Port (
                        CLK100MHZ: in STD_LOGIC;
                        SEG: inout STD_LOGIC_VECTOR (7 downto 0);
                        AN : inout STD_LOGIC_VECTOR (7 downto 0)
                        );

end DynamicSEG;
```

```
architecture behavioral of DynamicSEG is

variable count_clk : integer := 0; -- Zmienna do liczenia

begin

-- Proces ten ma wyświetlać różne konfiguracje świecących LED na różnych
-- wyświetlaczach, aby to osiągnać zmienia się anody z odpowiednią częstotliwością
-- tak aby oko nie zauważyło zmian ( min. 50 Hz )
   process (CLK100MHZ)
 -- Obliczamy, ile impulsów trzeba zliczyć, aby miec częstotliwość 100 Hz
-- f_out / f_in = 100 MHz / 100 Hz = 1 000 000
-- 1 000 000 / 2 = 500 000
-- Dzielimy 500 000 impulsów pomiędzy 4 wyświetlacze, wychodzi więc 125 000 impulsów na
wyświetlacz
   begin

      if (rising_edge(CLK100MHZ)) then

                        if (count_clk < 125000) then -- Przez daną część czasu
            count_clk := count_clk + 1; -- Zwiększamy licznik
            SEG <= "10101010";  -- Ustawiamy dane diody na wyświetlaczu nr 1
            AN <= "11111110";
         elsif (counter < 250000) then
            count_clk := count_clk + 1;
            SEG <= "01010101";
            AN <= "11111101"; -- Następnie na wyświetlaczu nr 2 itd.
         elsif (counter < 375000) then -- Częstotliwość zmian jest na tyle szybka, że oko nie widzi
różnicy w zmianach
            count_clk := count_clk + 1; -- które zachodzą pomiędzy wyświetlaczami i wszystko
widać jakby
            SEG <= "11110000"; -- było wyświetlane statycznie
            AN <= "11111011";
         elsif (counter < 500000) then
            count_clk := count_clk + 1;
            SEG <= "00001111";
            AN <= "11110111";
         else
            count_clk := 0;
         end if;
      end if;
   end process;


end behavioral;
```

Wnioski

Krytyczne znaczenie w tym programie ma odpowiednie przeskalowanie częstotliwości na taką, która pozwoli na zmienianie anod tak szybko, aby nie było widać zmian. Instrukcja elsif pozwala na

wybór przedziału (anody) w taki sposób, że sygnały nie są nadpisywane przez ostatnie przypisanie.

3.2. Wyświetlanie na wyświetlaczach 7-segmentowych dynamicznie z przekodowaniem

Opis działania programu

Program ma działać tak jak poprzedni, ale tym razem jest realizowany programowo w inny sposób, ponieważ wykorzystuję procedurę służącą do przekodowania kodu BCD na kod wyświetlacza 7-segmentowego.

Kod źródłowy (najważniejsze fragmenty)

```
architecture behavioral of DynamicSEG is

variable count_clk : integer := 0; -- Zmienna do liczenia
shared variable seg_v : STD_LOGIC_VECTOR( 7 downto 0) := "00000000"; -- Zmienna
przechowująca wyjście z funkcji
shared variable bcd_1 : STD_LOGIC_VECTOR( 3 downto 0) := "0001"; -- Zmienna określająca
liczbę na wyświetlaczu 1
shared variable bcd_2 : STD_LOGIC_VECTOR ( 3 downto 0) := "0010"; -- j.w. na 2
shared variable bcd_3 : STD_LOGIC_VECTOR ( 3 downto 0) := "0011"; -- j.w. na 3
shared variable bcd_4 : STD_LOGIC_VECTOR ( 3 downto 0) := "0100"; -- j.w. na 4

-- Procedura konwertująca kod BCD na kod wyświetlacza 7-SEG
procedure CONV_BCD_7SEG( variable BCD_NUM : in STD_LOGIC_VECTOR(3 downto 0);
                         variable SEG_NUM : out STD_LOGIC_VECTOR(7 downto 0)
                         ) is
begin     -- BCD CODE         -- 7 SEG CODE

case BCD_NUM is
  when "0000" => SEG_NUM := "11000000";    --0    7 SEG DISPLAY VECTOR IS " DP
GFEDCBA "
  when "0001" => SEG_NUM := "11111001";    --1          A
  when "0010" => SEG_NUM := "10100100";    --2       -------
  when "0011" => SEG_NUM := "10110000";    --3     F |     | B
  when "0100" => SEG_NUM := "10011001";    --4       |  G  |
  when "0101" => SEG_NUM := "10010010";    --5       -------
  when "0110" => SEG_NUM := "10000010";    --6     E |     | C
  when "0111" => SEG_NUM := "11111000";    --7       |     |   --
  when "1000" => SEG_NUM := "10000000";    --8       -------   -- DP
  when "1001" => SEG_NUM := "10010000";    --9          D
  when others =>  SEG_NUM := "01110111"; -- DOT
  end case;

end CONV_BCD_7SEG;

begin

-- Proces ten ma wyświetlać różne konfiguracje świecących LED na różnych
```

```
-- wyświetlaczach, aby to osiągnać zmienia się anody z odpowiednią częstotliwością
-- tak aby oko nie zauważyło zmian
   process (CLK100MHZ)
 -- Obliczamy, ile impulsów trzeba zliczyć, aby miec częstotliwość 100 Hz
-- f_out / f_in = 100 MHz / 100 Hz = 1 000 000
-- 1 000 000 / 2 = 500 000
-- Dzielimy 500 000 impulsów pomiędzy 4 wyświetlacze, wychodzi więc 125 000 impulsów na
wyświetlacz
   begin

      if (rising_edge(CLK100MHZ)) then

           if (count_clk < 125000) then -- Przez daną część czasu
            count_clk := count_clk + 1; -- Zwiększamy licznik
              CONV_BCD_7SEG(bcd_1, seg_v); -- Konwersja z BCD na SEG
            SEG <= seg_v;  -- Ustawiamy dane diody na wyświetlaczu nr 1
           AN <= "11111110";
         elsif (counter < 250000) then
           count_clk := count_clk + 1;
                CONV_BCD_7SEG(bcd_2, seg_v);
           SEG <= seg_v;
           AN <= "11111101"; -- Następnie na wyświetlaczu nr 2 itd.
         elsif (counter < 375000) then -- Częstotliwośc zmian jest na tyle szybko, że oko nie widzi
różnicy w zmianach
           count_clk := count_clk + 1; -- które zachodzą pomiędzy wyświetlaczami i wszystko
widać jakby
             CONV_BCD_7SEG(bcd_3, seg_v);
           SEG <= seg_v; -- było wyświetlane statycznie
           AN <= "11111011";
         elsif (counter < 500000) then
           count_clk := count_clk + 1;
             CONV_BCD_7SEG(bcd_4, seg_v);
           SEG <= seg_v
           AN <= "11110111";
         else
           count_clk := 0;
         end if;
      end if;
   end process;

end behavioral;
```

Wnioski

Wykorzystanie procedur w programie pozwala na zautomatyzowanie czynności wykonywanych często w programie, co sprawia, że program staje się krótszy i łatwiejszy do napisania. Procedura konwertująca wpływa w dużej mierze na wygodę pisania programu. Procedura w odróżnieniu od procesu musi zostać wywołana w kodzie.

3.3. Licznik z wyświetlaczem 7-segmentowym

Opis działania programu

Program ten liczy w górę. Do poprzednio realizowanego wyświetlania dynamicznego z przekodowaniem jest dołożona zmiana cyfr na wyświetlaczach, na każdym następnym zmiana jest 4 razy wolniejsza.

Kod źródłowy (najważniejsze fragmenty)

```
architecture behavioral of DynamicSEG is

        -- 4 NIEZALEŻNE LICZNIKI LICZĄCE CZAS ZMIAN JEDNOŚCI, DZIESIĄTEK,
SETEK I TYSIĘCY
        shared variable co_1: Integer := 0; -- Zmienna liczaca czas jednosci
    shared variable co_10: Integer := 0; -- czas na dzisiatki
    shared variable co_100: Integer := 0; -- czas na setki
    shared variable co_1000: Integer := 0; -- czas na tysiace

        shared variable count: Integer := 0; -- Licznik liczacy czas na wyswietlanie dla jednego
wyswietlacza

        -- 4 ZMIENNE OKREŚAJĄCE WARTOŚĆ LICZBY DO WYŚWIETLENIA W KODZIE
BCD
        shared variable bcd_1: STD_LOGIC_VECTOR(3 downto 0) := "0000"; --
        shared variable bcd_10: STD_LOGIC_VECTOR(3 downto 0) := "0000";
        shared variable bcd_100: STD_LOGIC_VECTOR(3 downto 0) := "0000";
        shared variable bcd_1000: STD_LOGIC_VECTOR(3 downto 0) := "0000";

        -- zmienna do której zapisywane jest wyjście procedury konwertującej
        shared variable seg_v : STD_LOGIC_VECTOR( 7 downto 0) := "00000000";

-- procedura konwertujaca kod BCD na kod wyswietlacza 7-segmentowego ( jak w programie
poprzednim )


begin

    process (CLK100MHZ)

    begin

        if (rising_edge(CLK100MHZ)) then

            if (count < 125000) then
                count := count + 1; -- Przypisania i warunki jak w programie poprzednim
                CONV_BCD_7SEG(bcd_1, seg_v);
                    SEG <= seg_v;
                AN <= "11111110";
            elsif (count < 250000) then
                count := count + 1;
```

```
            CONV_BCD_7SEG(bcd_10, seg_v);
                SEG <= seg_v;
            AN <= "11111101";
        elsif (counter < 375000) then
             count := count + 1;
            CONV_BCD_7SEG(bcd_100, seg_v);
                SEG <= seg_v;
            AN <= "11111011";
        elsif (counter < 500000 ) then
            count := count + 1;
            CONV_BCD_7SEG(bcd_1000, seg_v);
                SEG <= seg_v;
            AN <= "11110111";
        else
            count := 0;
        end if;

    end if;
end process;


        process (CLK100MHZ)

        -- Proces ma na celu obliczanie i zmienianie długości trwania jedności
        -- dziesiątek setek i tysięcy, znajdują się tutaj 4 niezależne liczniki
        -- każdy następny licznik zlicza dziesięć razy wolniej od poprzedniego
        -- Proces zapisuje aktualną wartość jedności, dzisiętek, setek i
        -- tysięcy do zmiennych w kodzie bcd

    begin

        if (rising_edge(CLK100MHZ)) then

            if (co_1 < 2 000 000) then -- Czestotliwość zmiany jedności to 25 Hz
                co_1 := co_1 + 1; -- bo 100 Mhz / 25 Hz = 4 000 000
            else -- 4 000 000 / 2 = 2 000 000
                co_1 := 0;
                                    if (bcd_1 < "1010") then -- Dopoki jednosci sie nie przepełniają to je
zwiększamy
                                        bcd_1 := bcd_1 + 1;
                                    else
                                        bcd_1 := "0000"; -- Gdy się przepełniają, to je zerujemy
                                    end if;
            end if;

                            if (co_10 < 20 000 000) then -- Częstotliwość dzisiątek to 2,5 Hz
                co_10 := co_10 + 1;
            else
                co_10 := 0;
                                    if (bcd_10 < "1010") then -- analogicznie jak w warunku wyżej, ale
dla dzisiątek
                                        bcd_10 := bcd_10 + 1;
```

```
                        else
                                bcd_10 := "0000";
                        end if;
        end if;

                        if (co_100 < 200 000 000) then -- 0,25 Hz
           co_100 := co_100 + 1;
        else
           co_100 := 0;
                        if (bcd_100 < "1010") then -- analogicznie dla setek
                                bcd_100 := bcd_100 + 1;
                        else
                                bcd_100 := "0000";
                        end if;
        end if;

                        if (co_1000 < 20 000 000 000) then -- 0,025 Hz
           co_1000 := co_1000 + 1;
        else
           co_1000 := 0;
                        if (bcd_1000 < "1010") then -- analogicznie dla tysięcy
                                bcd_1000 := bcd_1000 + 1;
                        else
                                bcd_1000 := "0000";
                        end if;
        end if;

      end if;

   end process;

end behavioral;
```

Wnioski

Ciekawym elementem programu jest wykorzystanie zmiennych ‘shared’, zmienne te mogą być wykorzystywane przez wszystkie procesy, są współdzielone. Jeden z procesów realizuje zliczanie, a drugi wyświetlanie, takie rozbicie programu na procesy powoduje wygodę pisania. Warto też zauważyć, że jeżeli program ma zmieniać jakiś sygnał, to często lepszym rozwiązaniem jest wykonanie operacji na zmiennej, a następnie przypisanie tej zmiennej do sygnału, tak, że sygnał jest zmieniony w procesie tylko raz.