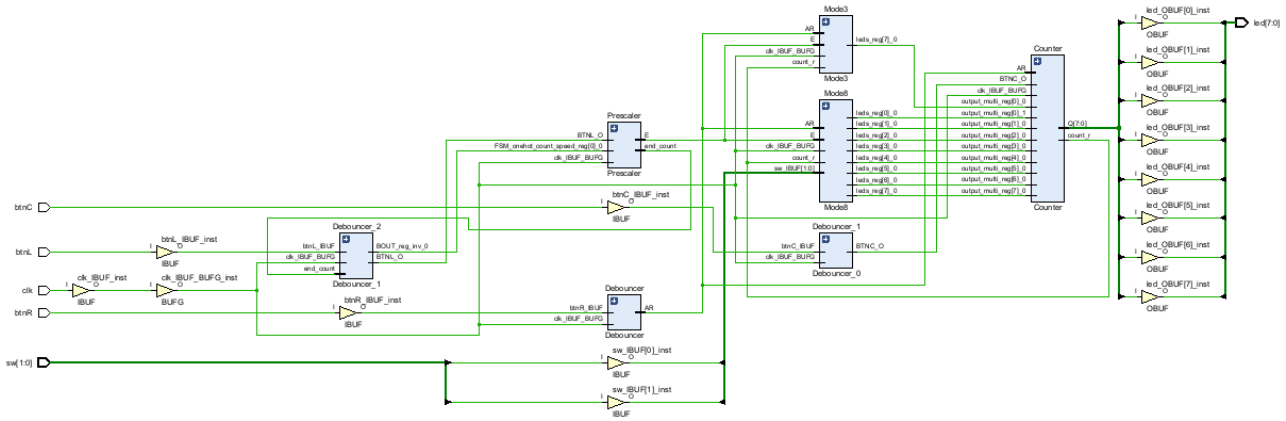
Data: 25.01.2024
Autor: Mikołaj Nastaj

Opis projektu:

Projekt ma na celu wykonywania czynności: po wciśnięciu przycisku użytkownik mógł przełączać się między wykonywanymi wariantami w dowolnym momencie, warianty to negacja zapalanych diod, oraz wędrujące diody, gdzie za pomocą switchów możemy kontrolować ilość zapalanych diod. Dodatkowo jednym z celów projektu jest wykonanie testbench'a, który ma umożliwiać symulację z różnymi wartościami prescaller'a.

Wynik syntezy:

Po wykonaniu symulacji i implementacji programu otrzymałem poniższy schemat blokowy:



Bloki Mode3 i Mode8 to bloki odpowiedzialne za wykonywanie poszczególnych wariantów projektu. Moduł Debouncer odpowiada za poprawną obsługę przycisków, aby zniwelować zakłócenia występujące po jego wciśnięciu. Moduł Counter sprawdza ilość wciśnięć przycisków right i c, które odpowiadają za symulację układu dla różnych czasów. Moduł Precaller odpowiada za przeliczenie jednej sekundy na podstawie ilości cykli, które wykona zegar.

Opracowany kod:

Główne moduły programu to Debouncer oraz Prescaler

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

entity Prescaler is
        Generic (
                prescalar : natural;
                speed : natural
        );
   Port ( INPUT : in STD_LOGIC;
        CLOCK_IN : in  STD_LOGIC;
        CLOCK_OUT : out  STD_LOGIC);
end Prescaler;

architecture Behavioral of Prescaler is
                signal count : natural := 0;
                signal count_speed : natural := 0;
                signal end_count : natural := prescalar;
                signal INTER_CLOCK : STD_LOGIC := '0';
begin
        CLOCK_OUT <= INTER_CLOCK;

        Prescaler_proc: process(CLOCK_IN) is
        begin
```

```
                if(rising_edge(CLOCK_IN)) then

        if(INPUT = '1') then
           if(count_speed = 0) then
              end_count <= speed;
              count_speed <= 1;
           else
              end_count <= prescalar;
              count_speed <= 0;
           end if;
        else
           if(count >= end_count - 1) then
              count <= 0;
              INTER_CLOCK <= '1';
           else
             INTER_CLOCK <= '0';
              count <= count + 1;
           end if;
                    end if;

                end if;
         end process;

end Behavioral;
```

Powyższy program opisujący Prescaler opisuje jego działanie. Moduł ten zlicza ilość wykonanych cykli zegara, na podstawie, której reszta układu możemy ustalić z jaką częstotliwością będą zapalać się diody, odpowiedzialne za to są dwie zmienne speed i prescalar, które określają naszą prędkość, po wciśnięciu przycisku btn_L możemy przełączać się między tymi prędkościami. Output prescalera, to pojedynczy cykl „nowego" zegara.

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


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Debouncer is
   Generic (
                COUNT_MAX : natural
         );
   port(  CLOCK_IN : in std_logic;
        BIN : in std_logic;
        BOUT : out std_logic
      );
end Debouncer;

architecture Behavioral of Debouncer is

constant BTN_ACTIVE : std_logic := '1';

signal count : integer := 0;
type state_type is (idle,wait_time);
signal state : state_type := idle;

begin

process(CLOCK_IN)
begin
```

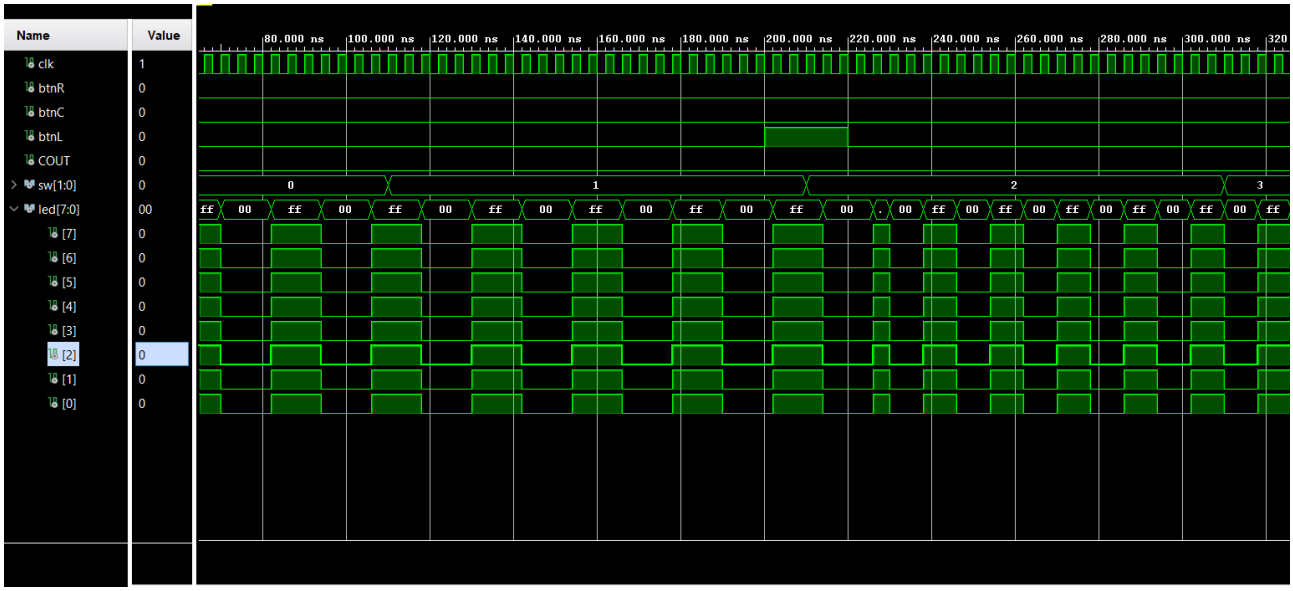
```
    if(rising_edge(CLOCK_IN)) then
        case (state) is
            when idle =>
                if(BIN = BTN_ACTIVE) then
                    state <= wait_time;
                else
                    state <= idle;
                end if;
                BOUT <= '0';
            when wait_time =>
                if(count = COUNT_MAX) then
                    count <= 0;
                    if(BIN = BTN_ACTIVE) then
                        BOUT <= '1';
                    end if;
                    state <= idle;
                else
                    count <= count + 1;
                end if;
        end case;
    end if;
end process;

end architecture Behavioral;
```

Powyżej znajduje się kod opisujący działanie Debouncera. Poprzez moduł TOP przekazujemy do Debouncera sygnały z przycisków, jako że korzystamy z trzech przycisków to też z trzech modułów korzystamy, do każdego z nich podajemy dodatkowo sygnał z zegara systemowego oraz wykorzystujemy stałą COUNT_MAX, dzięki której określamy opóźnienie naszego sygnału, tak aby uniknąć zakłóceń wywołanych przez wciśnięcie przycisku.

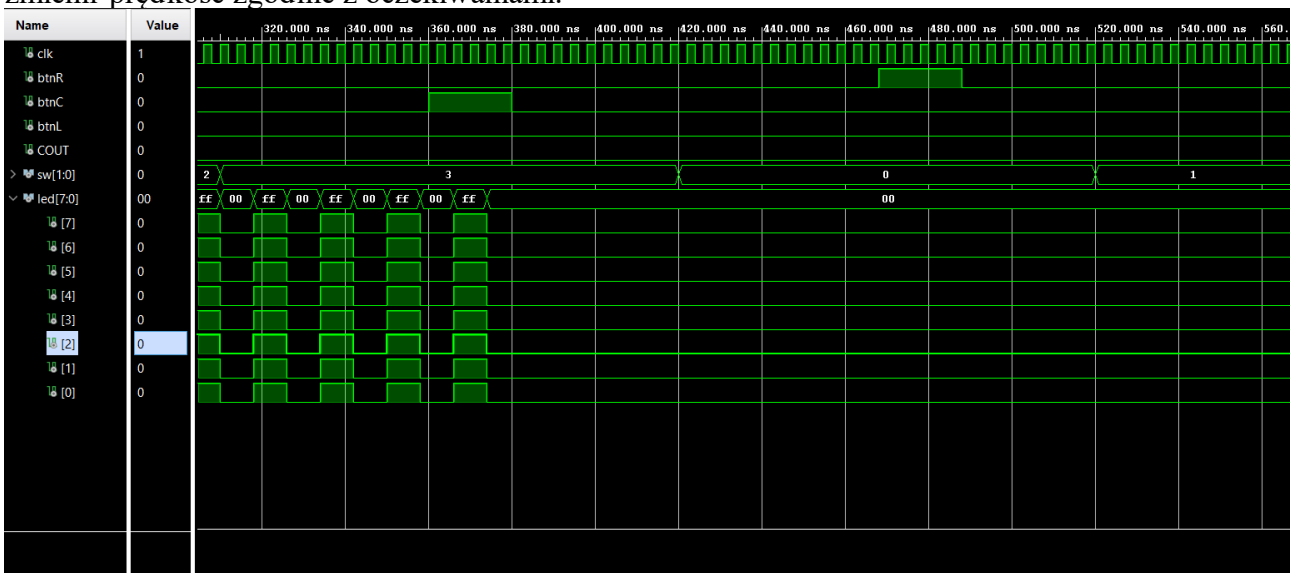
Testbench i symulacje:

Po wykonaniu programu wykonałem moduł testbench, który pozwolił na przeprowadzenie symulacji. Przyciski odpowiadają za różne funkcje, btnR – zmiana wariantu, btnC – wyłączenie/włączenie ledów, btnL – zmiana prędkości, dodatkowo wykorzystujemy switche, aby określić liczbę zapalonych ledów.

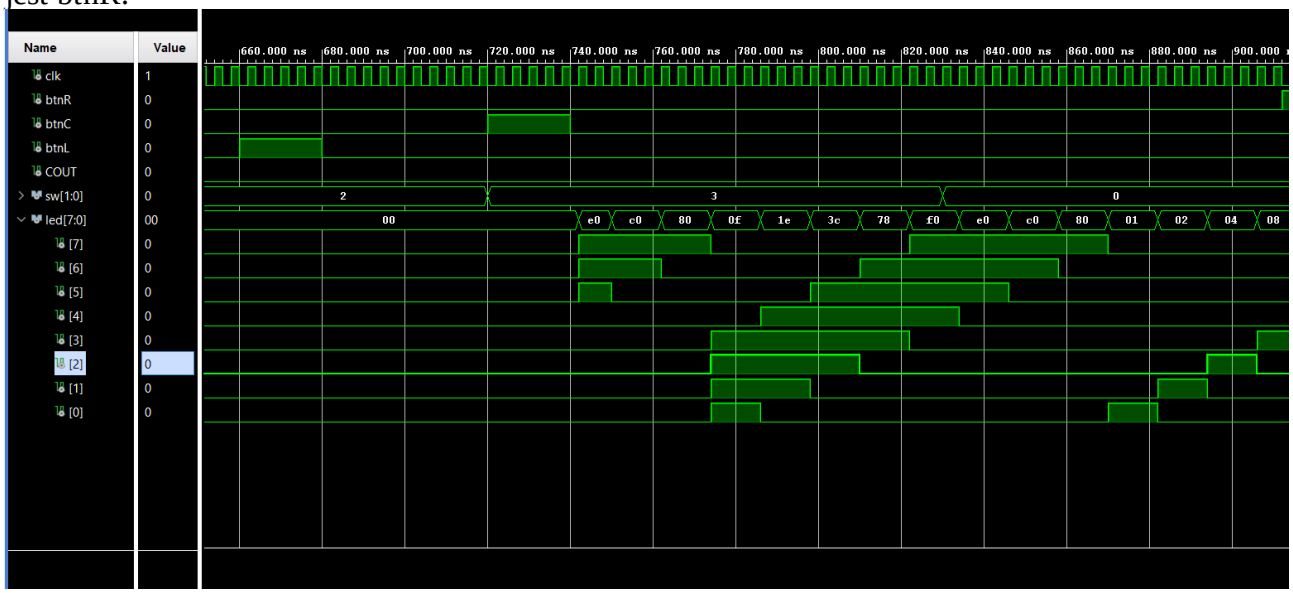
W testbench'u znajduje się 6 procesów:
CLOCK_IN_process - określający działanie zegara
Predkosc – określająca zmianę prędkości za pomocą przycisków
OFF_ON – wyłączający/włączający ledy
Mode – zmiana wariantu pracy urządzenia
Switch – zmieniający stan switchy, które określają liczbę wędrujących diod
Symulacja:



Włączono urządzenie, domyślnym wariantem działania był wariant trzeci, czyli negacja diod; btnL zmienił prędkość zgodnie z oczekiwaniami.

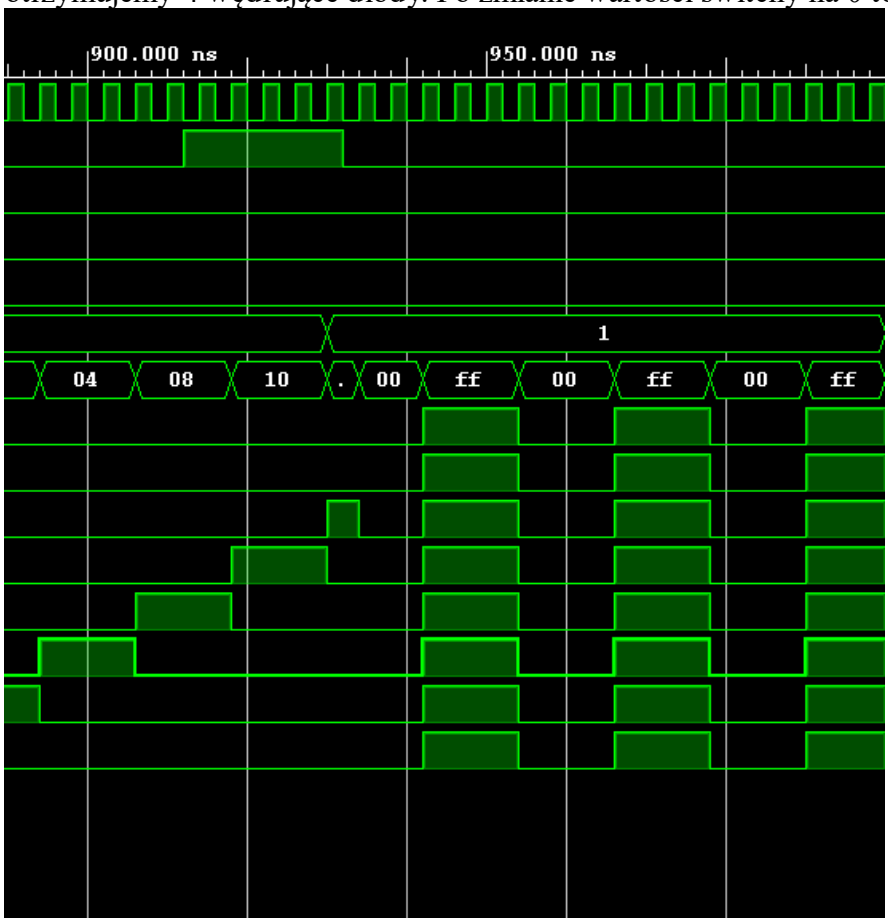


BtnC wyłączył ledy , w tym momencie nie ma reakcji na żaden inny przycisk, czego przykładem jest btnR.



Po wciśnięciu przycisku btnC włączamy ledy, urządzenie zarejestrowało zmianę wariantu pracy i

po włączeniu otrzymujemy wędrujące diodu, jako że switche ustawione są na wartość 3, to otrzymujemy 4 wędrujące diody. Po zmianie wartości switchy na 0 to liczba wędrujących diod to 1.



Po wciśnięciu przycisku btnR zmieniamy tryb i na pozytywnym zboczu zegara zmienia się wariant pracy urządzenia.