Systemy Czasu Rzeczywistego

„FPGA”

laboratorium: 02

autor: mgr inż. Mateusz Baran

Data utworzenia: 01/10/14

1 Spis treści

2 Wiadomości wstępne

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące omawianego tematu. Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej części laboratorium.

2.1 Niezbędne wiadomości

Przed laboratorium należy zapoznać się z podstawowymi pojęciami dotyczącymi budowy układów FPGA:

- zasoby logiczne FPGA (CLB, slice, pamięć Block RAM, mnożarki),
- zasoby połączeniowe (programowalne połączenia wewnętrze, IOB),
- podstawy języka VHDL.

Podczas laboratorium będzie wykorzystywany układ Digilent Basys2 oparty o układ Xilinx Spartan3E-250.

2.2 Literatura:

[1] Podstawowy opis architektury układów FPGA:
http://www.csd.uoc.gr/~hy220/2009f/lectures/11_basic_fpga_arch.pdf

[2] Szczegóły budowy wykorzystywanych układów FPGA:
http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf

[3] Podstawy języka VHDL:
https://wiki.ittc.ku.edu/ittc/images/3/37/EECS_140_VHDL_Tutorial.pdf

3 Przebieg laboratorium

3.1 Zadanie 1. Na ocenę 3.0 (dst)

Na początku laboratorium wprowadzone zostaną nowe konstrukcje języka VHDL: proces oraz instrukcje warunkowe. Procesy są wyodrębnionymi fragmentami logiki wykonywanymi w sposób sekwencyjny:

```
my_process: process (input1, input2)
 begin
  output1 <= input1 and input2;
 end process;
```

W powyższym przykładzie proces o nazwie my_process jest wykonywany ilekroć jeden z sygnałów input1, input2 ulegnie zmianie. Jest to określane za pomocą tzw. *sensitivity list*. Typowo na tej liście umieszcza się sygnały, których wartości są wykorzystywane w ciele procesu.

Instrukcje warunkowe mają postać

```
if [condition] then
  statements;
 end if;
```

lub

```
if [condition] then
  statements;
```



```
else
  statements;
end if;
```

W miejscu warunku możemy umieścić na przykład testowanie, czy sygnał właśnie wzrósł: rising_edge(clock_signal), co jest przydatne przy tworzeniu elementów synchronizowanych sygnałem zegarowym.

Przejdźmy teraz do wstępnej struktury projektu:

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

entity Switches_LEDs is
   Port ( switches : in  STD_LOGIC_VECTOR(7 downto 0);
          LEDs     : out STD_LOGIC_VECTOR(7 downto 0);
          clk      : in STD_LOGIC
        );
end Switches_LEDs;

architecture Behavioral of Switches_LEDs is
   signal counter : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
begin

clk_proc: process(clk)
  begin
    if rising_edge(clk) then
      counter <= counter+1;
    end if;
  end process;

end Behavioral;
```

Encja Switches_LEDs z poprzedniego laboratorium została tak przerobiona, aby posiadać wewnętrzny licznik – sygnał o nazwie counter. Dodatkowo otrzymuje ona sygnał zegarony clk wykorzystywany do synchronicznego zwiększania licznika o 1 o procesie clk_proc. Sygnał jest magistralą o 8 wartościach inicjalizowanych sygnałami o wartości 0.

Aby móc wykorzystywać sygnał zegarowy, musimy dodać dwie linie do pliku ograniczeń:

```
NET "Clk" LOC = "B8";
NET "Clk" CLOCK_DEDICATED_ROUTE = FALSE;
```

Pierwszym zadaniem jest przerobienie licznika na 30-bitowy i wyświetlenie trzech najstarszych bitów na ośmiu diodach LED będących sygnałami wyjściowymi encji. Następnie zmień licznik tak, aby odliczał w dół. W ostatecznej wersji, oddawanej do oceny, wykorzystaj jeden z przełączników do określenia kierunku odliczania.

3.2 Zadanie 2. Na ocenę 4.0 (db)

Wykorzystane w poprzednim zadaniu dodawanie nie jest operacją prymitywną. Sprowadza się ono do odpowiednich operacji poszczególnych sygnałach. Niniejsze zadanie polega na napisaniu odpowiednich operacji na sygnałach tak, aby wynikiem był program dodający dwie liczby zakodowane przez położenia przełączników. Punktem wyjścia powinien być następujący program:

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

entity Switches_LEDs is
   Port ( switches : in  STD_LOGIC_VECTOR(7 downto 0);
          LEDs     : out STD_LOGIC_VECTOR(7 downto 0)
        );
end Switches_LEDs;

architecture Behavioral of Switches_LEDs is
   signal x      : STD_LOGIC_VECTOR(3 downto 0);
   signal y      : STD_LOGIC_VECTOR(3 downto 0);
   signal carry  : STD_LOGIC_VECTOR(3 downto 0);
   signal result : STD_LOGIC_VECTOR(4 downto 0);
begin
 LEDs <= "000" & result;
 x <= switches(3 downto 0);
 y <= switches(7 downto 4);

 result(0) <= x(0) XOR y(0);
 carry(0)  <= x(0) AND y(0);

 result(1) <= x(1) XOR y(1) XOR carry(0);
 carry(1)  <= (x(1) AND y(1)) OR (carry(0) AND X(1)) OR (carry(0) AND Y(1));

end Behavioral;
```

W programie tym mamy cztery sygnały pomocnicze: x i y są liczbami, które będą dodawane. Carry reprezentuje przeniesienie, zaś result jest wynikiem dodawania. Drugą częścią zadania jest przerobienie programu tak, aby dodawane były trzy liczby: switches(2 downto 0), switches(5 downto 3) i switches(7 downto 6).

3.3 Zadanie 3. Na ocenę 5.0 (bdb)

Zmodyfikuj program tak, aby wykorzystywał dwa liczniki: jeden do odliczania zboczy sygnału zegarowego do pełnych sekund (proszę pamiętać o wykorzystaniu jak najwęższego sygnału), oraz drugi, ośmiobitowy. Co jedną sekundę ośmiobitowy sygnał zwiększa się o wartość określoną stanami przełączników. Do zapisu liczb (na diodach

i przełącznikach) wykorzystywany jest kod znak-moduł, przy czym zero oznacza znak dodatni.

Podpowiedź: wykorzystaj bibliotekę IEEE.Numeric_STD. Sygnał zegarowy wykorzystywany w zadaniu na 3.0 ma częstotliwość 50 MHz.

