Technika Cyfrowa - sprawozdanie 4

Łukasz Jezapkowicz

24.05.2020

1 Korzystając z programu Quartus II firmy Intel (www.intel.com) , należy zrealizować praktycznie w układzie FPGA (np. na zestawie UP2 dostępnym w laboratorium) projekt wyświetlający na dwóch wyświetlaczach siedmiosegmentowych efekt węża.
Projekt należy wykonać dwoma sposobami:
- w języku SystemVerilog
- w języku VHDL

1.1 FPGA - czym jest, porównanie z mikrokontrolerem, wady i zalety

FPGA (Field-Programmable Gate Array - bezpośrednio programowalna macierz bramek) jest rodzajem programowalnego układu logicznego. W dużym uproszczeniu można FPGA określić jako zbiór bramek logicznych, między którymi możemy w dowolny sposób tworzyć połączenia. Układ FPGA się konfiguruje (nie tworzymy programu tylko opis funkcjonowania naszego projektu) używając języków sprzętu takich jak VHDL czy SystemVerilog.

FPGA

Programowalne układy logiczne (np. FPGA) róznią się znacząco od mikrokontrolerów:
- cena fizycznych chipów programowalnych układów logicznych jest kilkukrotnie droższa niż mikrokontrolerów (koszty opracowania i rozwoju aplikacji są wyższe)
- programowalne układy logiczne są znacznie szybsze w zadaniach, które powinny być wykonywane równolegle (np. w kartach graficznych)
- w przypadku mikrokontrolerów kod tłumaczony jest na instrukcje maszynowe, które są przechowywane w pamięci mikrokontrolera. W czasie pracy instrukcje są pobierane z pamięci i wykonywane przez ALU. W programowalnych układach logicznych nie wykonuje się kolejno instrukcji.

Wewnątrz tworzony jest układ sekwencyjny, który można opisać jako maszynę stanów (dzięki czemu możliwe jest wykonywanie wielu zadań równolegle).
Układy FPGA mają wiele plusów:
- można z nich zrobić praktycznie wszystko (na przykład mikrokontroler)
- są bardzo szybkie
- są bezpośrednio programowalne
- zadania mogą być wykonywane równolegle
Mają jednak równie wiele minusów:
- są kosztowne (mikrokontrolery są tańsze)
- potrzebują wysokiej mocy
- są skomplikowane
- posiadają mocno złożone narzędzia
W celu poszerzenia wiedzy obejrzałem również następujące materiały dotyczące tematyki FPGA:
Materiał 1
Materiał 2
Materiał 3

1.2 Wstęp do środowiska Quartus

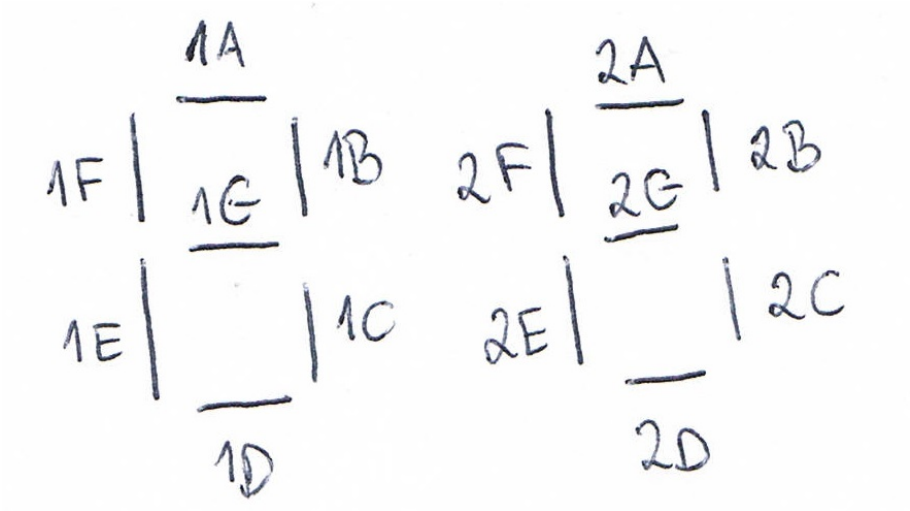
W celu praktycznej implementacji projektu założyłem konto Intel oraz pobrałem i zainstalowałem oprogramowanie Quartus II Web Edition v12.1 Service Pack 1. W celu instalacji oraz zapoznania się z tematyką Quartus'a skorzystałem z podanego poradnika:
http://home.agh.edu.pl/dlugopol/tcd/Lab_fpga.pdf

1.3 Języki VHDL oraz SystemVerilog

W celu zapoznania się z językami opisu sprzętu VHDL oraz SystemVerilog wykonałem następujące kursy dostarczane przez Intel:
Kurs VHDL
Kurs SystemVerilog
Języki VHDL oraz SystemVerilog pozwalają nam na "Behaviour Modeling" czyli opisywanie co układ powinien robić. Kompilator sam dobierze odpowiedni hardware, który umożliwi wykonanie opisanych przez nas działań.
Przejdźmy teraz do naszego projektu.

1.4 Problem

Problemem, który chcemy rozwiązać jest implementacja "węża" na dwóch wyświetlaczach siedmiosegmentowych. W każdej chwili wąż powinien zajmować trzy segmenty i poruszać się dookoła tych wyświetlaczy. Jego działanie pokazane jest na poniższym filmiku:
Wąż
W celu rozpracowania rozwiązania problemu na początku oznaczę odpowiednie segmenty symbolami:



Symbolika

Warto tutaj zaznaczyć kilka faktów:
- segmenty $1B$, $1C$, $2F$ oraz $2E$ nie będą nigdy używane.
- wszystkie inne segmenty poza $1G$ oraz $2G$ w każdym pełnym przejściu węża używane są tylko raz. Segmenty $1G$ oraz $2G$ używane są dwukrotnie.
- razem jest 12 różnych stanów, w których może być wąż.

Łatwo stwierdzić, że to czy segment będzie w następnym kroku aktywny zależy od wartości innego segmentu w danym kroku (dla $2G$ zależy od dwóch segmentów). Te przemyślenia można zebrać w następujące zależności:
$[1F]_{n+1} = [1G]_n$
$[1A]_{n+1} = [1F]_n$
$[2A]_{n+1} = [1A]_n$
$[2B]_{n+1} = [2A]_n$
$[2G]_{n+1} = [2B]_n$ OR $[2C]_n$
$[1G]_{n+1} = [2G]_n$
$[1E]_{n+1} = [1G]_n$
$[1D]_{n+1} = [1E]_n$
$[2D]_{n+1} = [1D]_n$
$[2C]_{n+1} = [2D]_n$
Spróbujmy teraz w językach opisu sprzętu VHDL oraz SystemVerilog opisać powyższy problem.

1.5 VHDL oraz SystemVerilog

1.5.1 VHDL

Opiszę teraz działanie kodu stworzonego przeze mnie.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL; -- rising_edge
3  use IEEE.NUMERIC_STD.ALL; -- operacje arytmetyczne
4
5  -- program odwzorowujący "węża"
6
7  entity snake is
8      port (
9          clock : in std_logic;
10         F1,A1,A2,B2,G2,G1,E1,D1,D2,C2 : out std_logic -- nazwa zaczyna sie od litery bo nie można od cyfry
11     );
12 end snake;
13
14 architecture logic of snake is
15     signal board : std_logic_vector(11 downto 0) := "111000000000"; -- F1|A1|A2|B2|G2|G1|E1|D1|D2|C2|G2|G1
16 begin
17     P: process(clock)
18         begin
19             if rising_edge(clock) then
20                 -- zmiana wartości w segmentach
21                 F1 <= board(0);
22                 A1 <= board(11);
23                 A2 <= board(10);
24                 B2 <= board(9);
25                 G2 <= board(8) or board(2);
26                 G1 <= board(7) or board(1); -- ponieważ G2 pojawia się również w board(1)
27                 E1 <= board(6);
28                 D1 <= board(5);
29                 D2 <= board(4);
30                 C2 <= board(3);
31
32                 -- przesunięcie węża
33                 for index in board'low to board'high loop  -- "111000000000" => "011100000000"
34                     board(index) <= board((index+1) mod board'length);
35                 end loop;
36             end if;
37         end process;
38
39 end architecture;
```

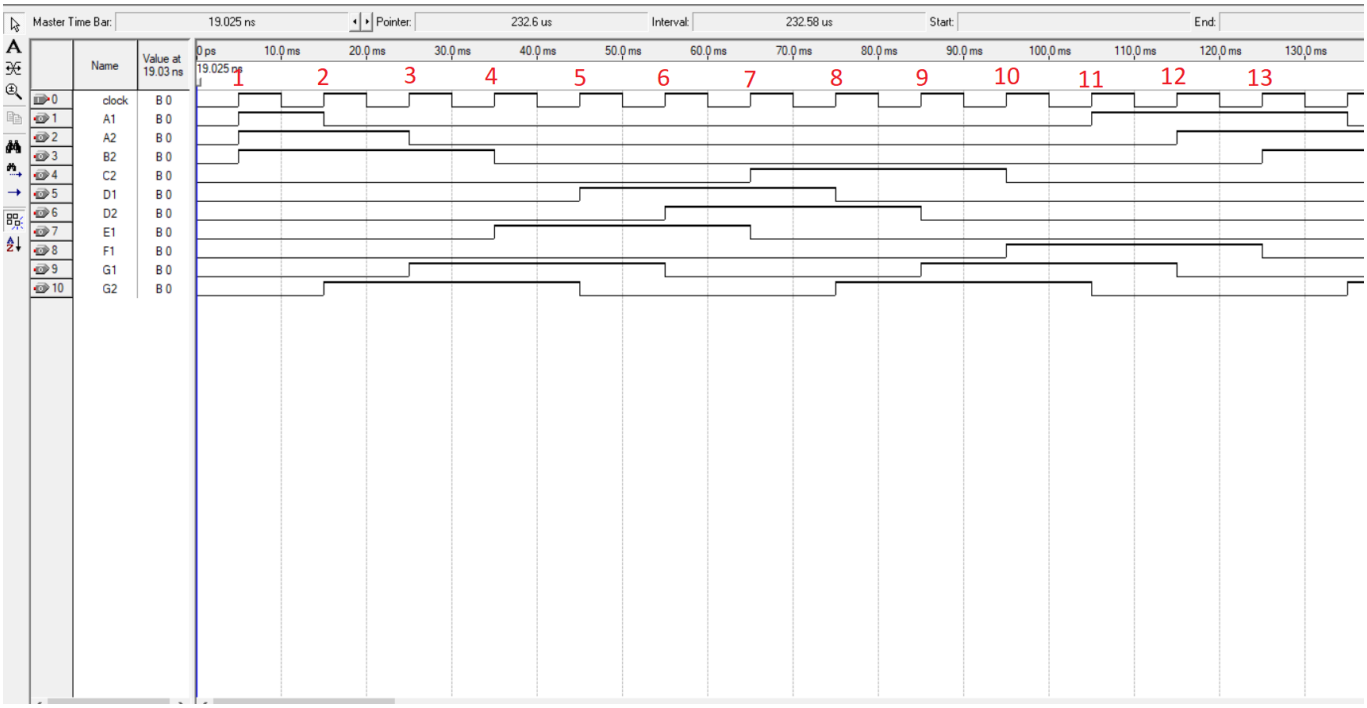
Kod źródłowy w VHDL

Na początku załączam bibliotekę $IEEE$ i pobieram z niej odpowiednie pliki źródłowe potrzebne w mojej implementacji. W VHDL należy zaimplementować dwa bloki kodu: *entity* oraz *architecture*. Blok *entity* określa wejścia i wyjścia modelu, który projektujemy. W tym przypadku na wejściu otrzymuje jedynie sygnał zegarowy *clock*. Na wyjściu pojawi się 10 wartości logicznych $F1$,$A1$,... symbolizujących 10 segmentów, które są w zadaniu używane (wcześniej opisałem 4 pomijane segmenty). Blok *architecture* związany jest z danym *entity* i opisuje jego funkcjonalność. Tutaj funkcjonalność wygląda następująco:

- Tworzę zmienną globalną *board*. Zmienna *board* jest wektorem 12-elementowym i symbolizuje przesuwającego się węża (w każdym momencie są jedynie 3 elementy o wartości 1 w wektorze). Wektor wyznacza kolejne segmenty w kolejności zgodnej z ruchem węża czyli $1F \rightarrow 1A \rightarrow 2A \rightarrow 2B \rightarrow 2G \rightarrow 1G \rightarrow 1E \rightarrow 1D \rightarrow 2D \rightarrow 2C \rightarrow 2G \rightarrow 1G \rightarrow$.

- Podczas każdego wyzwolenia zboczem narastającym sygnału zegarowego wykonuje następujące operacje:
  - Wypełniam zmienne wyjściowe nowymi wartościami bazując na zależnościach z poprzedniego rozdziału. Warto zauważyć, że w $G1$ pojawił się OR, ponieważ segment ten zależy od $G2$, które pojawia się dwukrotnie w naszym wektorze *board*
  - Przesuwam węża o 1 pozycję w prawo (idąc indeksami od 0 jedną pozycję w lewo).

Jak się okazuje taki zabieg jest wystarczający by otrzymać pożądany efekt.
Przy pomocy narzędzia $Vector\ Waveform\ File$ przeprowadziłem symulacje dla zbudowanego modelu. Poniżej widoczne wyniki symulacji:



Symulacja dla sygnału zegarowego o długości $1s$ i okresie $10ms$

Na powyższym rysunku zaznaczyłem kolejne wyzwolenia zboczem narastającym. Jak widać po 13 wyzwoleniu stan na wyjściu jest identyczny jak po 1. Widać więc, że pojawił się cykl 12-elementowy - dokładnie tak jak chcieliśmy. W kolejnych stanach wartość logiczną 1 mają segmenty:
$1A, 2A, 2B$
$2A, 2B, 2G$
$2B, 1G, 2G$
$1E, 1G, 2G$
$1D, 1E, 1G$
$1D, 2D, 1E$
$2C, 1D, 2D$
$2C, 2D, 2G$
$2C, 1G, 2G$
$1F, 1G, 2G$
$1A, 1F, 1G$

$1A, 2A, 1F$
Po porównaniu z wcześniej określoną symboliką stwierdzam, iż wąż porusza się prawidłowo i zatacza cykl.

1.5.2 SystemVerilog

Opiszę teraz działanie kodu stworzonego przeze mnie.

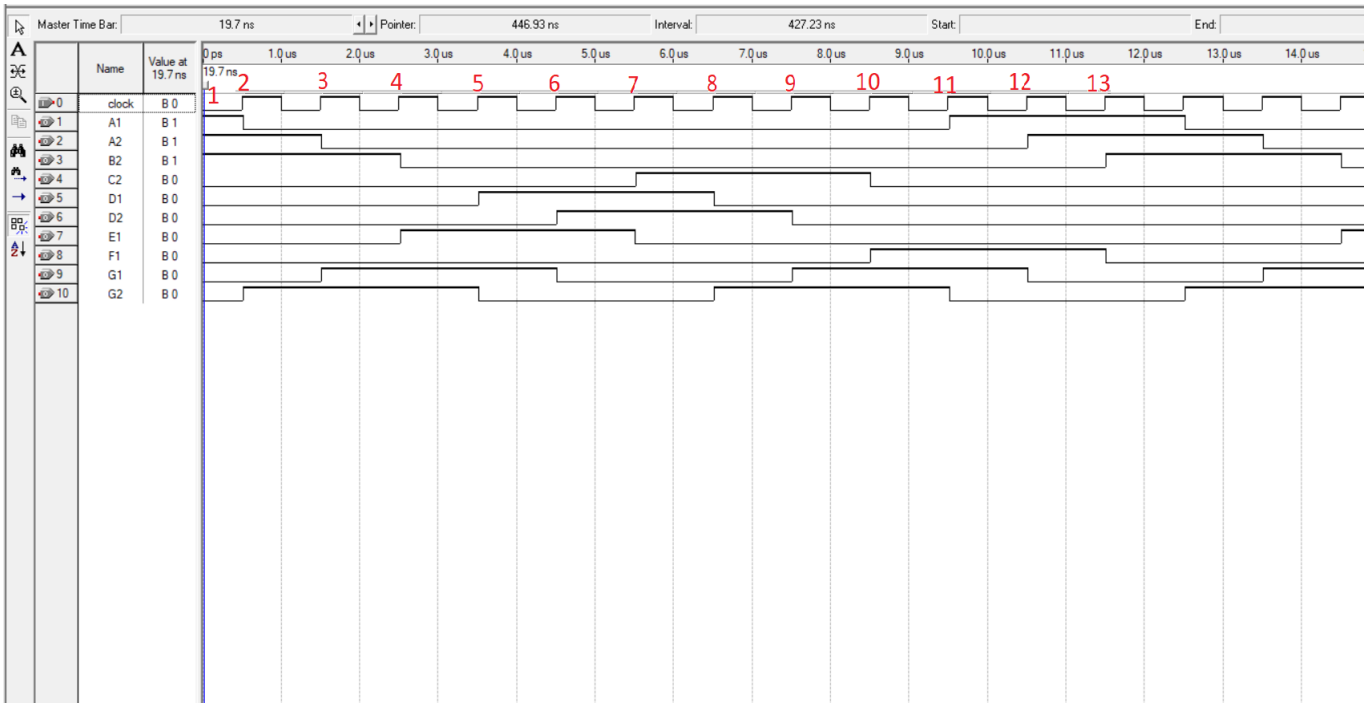
```
1  // program odwzorowujący "węża"
2
3  module snake (
4      input clock,
5      output F1,A1,A2,B2,G2,G1,E1,D1,D2,C2 // nazwa zaczyna sie od litery bo nie można od cyfry
6  );
7
8  reg [11:0] board = 12'b111000000000; // F1|A1|A2|B2|G2|G1|E1|D1|D2|C2|G2|G1
9
10 // zmiana wartości w segmentach
11 assign F1 = board[0];
12 assign A1 = board[11];
13 assign A2 = board[10];
14 assign B2 = board[9];
15 assign G2 = board[8] || board[2];
16 assign G1 = board[7] || board[1]; // ponieważ G2 pojawia się również w board[1]
17 assign E1 = board[6];
18 assign D1 = board[5];
19 assign D2 = board[4];
20 assign C2 = board[3];
21
22 assign tmp = board[0];
23 always @(posedge clock)
24 begin
25     // przesunięcie węża
26     board = board >> 1;
27     board[$size(board)-1] = tmp;
28 end
29
30 endmodule
```

Kod źródłowy w System Verilog

Działanie programu jest identyczne do programu z poprzedniego rozdziału.
Tworzę moduł *snake*, który przyjmuje na wejściu sygnał zegarowy *clock* a na wyjście zwraca $F1$,$A1$,... . Kolejnymi krokami programu są:

- Tworzę globalną tablice *board* identyczną do poprzedniej
- Wypełniam zmienne wyjściowe nowymi wartościami bazując na zależnościach z poprzednich rozdziałów. Warto zauważyć, że w $G1$ pojawił się OR, ponieważ segment ten zależy od $G2$, które pojawia się dwukrotnie w naszym wektorze *board*.
- Podczas każdego wyzwolenia zboczem narastającym przesuwam węża o 1 pozycję w prawo (idąc indeksami od 0 jedną pozycję w lewo).

Jak się okazuje taki zabieg jest wystarczający by otrzymać pożądany efekt.
Przy pomocy narzędzia $Vector\ Waveform\ File$ przeprowadziłem symulacje dla zbudowanego modelu. Poniżej widoczne wyniki symulacji:



Symulacja dla sygnału zegarowego o długości $16\mu s$ i okresie $1\mu s$

Na powyższym rysunku zaznaczyłem kolejne wyzwolenia zboczem narastającym. Jak widać po 13 wyzwoleniu stan na wyjściu jest identyczny jak po 1. Widać więc, że pojawił się cykl 12-elementowy - dokładnie tak jak chcieliśmy. W kolejnych stanach wartość logiczną 1 mają segmenty wymienione w poprzednim rozdziale (w tej samej kolejności).
Cel naszego zadania został osiągniety - działający model w dwóch jężykach opisu sprzętu (VHDL oraz System Verilog).

1.6 Wnioski

1.6.1 Czy układy działają poprawnie?

Przedstawione przeze mnie rozumowanie pozwoliło mi dojść do poprawnego wyniku końcowego, którym są poprawnie zaprojektowane i zaimplementowane modele węża na dwóch siedmiosegmentowych wyświetlaczach (zaimplementowane w językach opisu sprzętu VHDL oraz System Verilog). W dojściu do poprawnego rozwiązania poprowadziły mnie dobrze przygotowane poradniki wspomniane wcześniej. Działania układów FPGA oraz pisania w językach opisu sprzętu można nauczyć się samodzielnie dysponując odpowiednim zaangażowaniem. Stworzone przeze mnie układy wypełniają swoje zadanie, stwierdzam więc, że zostały wykonane poprawnie.

1.6.2 Co można było zrobić inaczej?

- Zastosować inny język opisu sprzętu
- Zbudować bardziej ambitny projekt (niestety niedobór czasu daje się we znaki)
- Przetestować program w praktyce, na rzeczywistym układzie FPGA
- Zastosować wyzwalanie zboczem opadającym
- Stworzyć większego węża (na większej ilości segmentów)
- Stworzyć węża, którym można sterować

1.6.3 Gdzie to można zastosować?

- Tak jak wcześniej wspomniane, można stworzyć większy projekt węża lub udostępnić sterowanie węża przez użytkownika
- Wiedza na temat układów FPGA może się w przyszłości przydać (może przejmą rynek?)
- Umiejętność pisania w nowych językach programowania zawsze się przydaje
- W układach FPGA można stworzyć praktycznie wszystko, od węża do mikrokontrolerów. Przydatna jest tutaj wiedza na temat wyżej wymienionych języków opisu sprzętu
- Układy FPGA pozwalają osiągać lepsze wyniki (wydajnościowo) niż klasyczne procesory
- Przetwarzanie obrazów i wideo (karty graficzne)
- Układy FPGA znajdują zastosowanie w bardzo wielu dziedzinach takich jak Elektronika Użytkowa czy Kryptowaluty