Technika Cyfrowa - ćwiczenie 4

Fortuna Wojciech, Ramut Michał, Stylski Bartłomiej, Tendaj Konrad

Akademia Górniczo-Hutnicza w Krakowie

18 Czerwca 2024

1 Układ FPGA <– co to?

FPGA (od ang. field-programmable gate array) - to zaawansowane układy scalone, które można programować do wykonywania różnorodnych funkcji logicznych. W przeciwieństwie do tradycyjnych układów ASIC (Application-Specific Integrated Circuits), które są projektowane i produkowane z myślą o konkretnym zadaniu, FPGA można programować i przeprogramowywać wielokrotnie, co daje im ogromną wszechstronność i elastyczność.

2 Budowa i działanie FPGA

2.1 Logic Array Blocks (LAB)

Logic Array Blocks (LAB) są kluczowymi elementami strukturalnymi w architekturze FPGA. LAB są podstawowymi jednostkami logicznymi, z których składają się układy FPGA. Każdy LAB zawiera zestaw elementów logicznych oraz struktury umożliwiające programowalne połączenia między tymi elementami. Są one zrealizowane przy pomocy Look-Up Tables (LUT) podłączonych do szybkiej pamięci RAM. Pozwala to poprzez zmienianie zawartości pamięci RAM programować dowolne funkcje logiczne. Dzięki temu LAB umożliwiają elastyczną i efektywną realizację złożonych operacji logicznych wewnątrz FPGA.

2.2 Embedded Array Blocks (EAB)

Embedded Array Blocks (EAB) to specjalistyczne układy w architekturze FPGA, które mogą mieć różne zastosowania, takie jak:

- **Pamięć RAM:**

  EAB mogą być skonfigurowane jako pamięć RAM (Random Access Memory), służącą do przechowywania danych tymczasowych, buforowania danych, oraz szybkiego dostępu do informacji w trakcie działania układu FPGA.

- **Pamięć ROM:**

  EAB mogą działać jako pamięć ROM (Read-Only Memory), przechowującą stałe dane, takie jak tablice prawdy, stałe współczynniki lub inne informacje konfiguracyjne.

- **Bloki DSP:**

  EAB mogą być wykorzystywane jako bloki DSP (Digital Signal Processing), wykonujące zaawansowane obliczenia matematyczne, takie jak mnożenie, sumowanie, filtrowanie sygnałów, itp. Dzięki temu, FPGA mogą realizować złożone zadania przetwarzania sygnałów cyfrowych.

2.3 Magistrala połączeń

Magistrala połączeń w FPGA to sieć przewodów, które łączą różne bloki logiczne oraz inne komponenty układu. Jest to kluczowy element architektury FPGA, umożliwiający przepływ sygnałów między różnymi częściami układu. Poniżej przedstawione są główne cechy i funkcje magistrali połączeń w FPGA:

- **Programowalność:**

  Magistrala połączeń w FPGA jest w pełni programowalna, co oznacza, że użytkownik może konfigurować połączenia między blokami logicznymi zgodnie z wymaganiami swojego projektu.

- **Optymalizacja Ścieżek Sygnałowych:**

  Narzędzia do projektowania FPGA automatycznie optymalizują ścieżki sygnałowe, aby minimalizować opóźnienia i maksymalizować wydajność. Dzięki temu magistrala połączeń może efektywnie łączyć różne elementy projektu, zapewniając szybki i niezawodny przepływ danych.

- **Wysoka Przepustowość:**

  Magistrala połączeń w FPGA jest zaprojektowana tak, aby obsługiwać wysoką przepustowość danych, co jest kluczowe dla aplikacji wymagających dużych ilości przesyłanych danych, takich jak przetwarzanie sygnałów czy obrazów.

3 Praktyczne zastosowania układów FPGA

3.1 Specjalistyczny sprzęt medyczny

źródło: https://www.intel.com/content/www/us/en/healthcare-it/products/programmable/overview.html

- FPGA mogą być aktualizowane bezpośrednio u użytkownika końcowego, co jest istotne, gdy standardy medyczne ewoluują lub gdy zmieniają się wymagania sprzętu. To zapewnia długoterminową użyteczność i zgodność ze zmieniającymi się normami.

- FPGA nie wymagają minimalnych ilości zamówień, co jest powszechnie wymagane w przypadku ASIC. Jest to szczególnie ważne w przypadku wysoce wyspecjalizowanych przyrządów, które nie są produkowane w dużych ilościach.

3.2 Kryptografia

źródło: https://xiphera.com/benefits-of-fpgas-as-implementation-platforms-for-cryptosystems/

- FPGA pozwalają na pełną izolację krytycznych obliczeń kryptograficznych oraz kluczy od reszty systemu, który może działać na potencjalnie wadliwym oprogramowaniu. Dzięki implementacji kryptografii i przechowywania kluczy w FPGA, system może być zaprojektowany tak, aby oprogramowanie nigdy nie miało dostępu do kluczy kryptograficznych ani innych informacji krytycznych z punktu widzenia bezpieczeństwa.

- Jedną z kluczowych zalet FPGA jest możliwość aktualizacji algorytmów kryptograficznych w systemie już wdrożonym. Jest to szczególnie ważne w kontekście zagrożeń związanych z algorytmami kwantowymi, które wymagają nowych algorytmów kryptograficznych.

3.3 Uczenie maszynowe

źródło: https://www.run.ai/guides/gpu-deep-learning/fpga-for-deep-learning

- Reprogramowalność FPGA jest ich największą zaletą w kontekście uczenia maszynowego. Umożliwia programowanie poszczególnych bloków lub całego obwodu, aby dostosować je do specyficznych wymagań algorytmu. Jeśli oprogramowanie nie spełnia oczekiwań, można je łatwo zmodyfikować, co daje ogromną elastyczność operacyjną.

- Według badań Microsoftu, FPGA mogą być nawet 10 razy bardziej efektywne pod względem zużycia energii niż GPU, co pomaga zmniejszyć całkowite zużycie energii w implementacjach uczenia maszynowego i głębokiego uczenia. Może to obniżyć koszty treningu modeli.

4 Omówienie kodu projektu

4.1 Treść zadania

- Zrealizować projekt efektu świetlnego wyświetlanego na dwóch wyświetlaczach siedmiosegmentowych, polegającego na przemieszczaniu się po obrzeżach tych wyświetlaczy w określonym kierunku odcinka świetlnego składającego się z dwóch segmentów.
- Po chwilowym krótkim wciśnięciu pierwszego z dwóch przycisków (lewego), układ powinien zmienić prędkość poruszania się odcinka świetlnego.
- Po chwilowym krótkim wciśnięciu drugiego z dwóch przycisków (prawego), układ powinien zmienić kierunek poruszania się odcinka.

4.2 Schemat projektu

Nasz projekt postanowiliśmy zrealizować dla układu EPF10K70RC240-4 UP2 firmy Altera. Korzystając z programu Quartus II v9 z wykorzystaniem języka VHDL.

```
entity light_effect is
    port (
        clk       : in std_logic;
        speed_btn : in std_logic;
        dir_btn   : in std_logic;
        segments1 : out std_logic_vector(6 downto 0);
        segments2 : out std_logic_vector(6 downto 0)
    );
end light_effect;
```

Rysunek 1: Blok kodu w języku VHDL - opis interfejsu

- clk – wejście - podpięte do zegara o częstotliwości 25.175 MHz (pin 91)
- speed_btn – wejście - podpięte do lewego przycisku (FLEX_PB1) – aktywnego zerem (pin 28)
- dri_btn – wejście - podpięte do prawego przycisku (LEX_PB2) – aktywnego zerem (pin 29)
- segments1, segments2 – wyjścia - siedmioelementowe wektory podpięte odpowiednio do lewego i prawego wyświetlacza siedmiosegmentowego (pierwszy element do elementu A, ostatni do G)

4.3 Zmienne użyte w projekcie

```
architecture behavioral of light_effect is
    signal segment_pattern : std_logic_vector(13 downto 0); -- 2 x 7-
        segmentowe wyświetlacze
    signal current_pos     : integer range 0 to 7 := 0;
    signal direction       : std_logic := '1';
    signal speed           : std_logic := '0';
    signal speed_btn_last, dir_btn_last : std_logic := '1';

    signal speed_btn_shift : std_logic_vector(7 downto 0) := (others
        => '1');
    signal dir_btn_shift   : std_logic_vector(7 downto 0) := (others
        => '1');
begin
```

Rysunek 2: Blok kodu w języku VHDL wraz z komentarzami obsłuującymi zmienne

4.4 Obsługa przycisków

```
1   -- obsługa lewego przycisku (sterującego prędkością)
2       process(clk)
3       begin
4           if rising_edge(clk) then
5               speed_btn_shift <= speed_btn_shift(6 downto 0) & speed_btn
                    ;
6               if speed_btn_shift = "00000000" and speed_btn_last = '1'
                    then
7                   speed <= not speed; -- jeśli przycisk jest wciśnięty
                        zmień wartość na przeciwną
8                   speed_btn_last <= '0';
9               end if;
10              if speed_btn_shift = "11111111" then
11                  speed_btn_last <= '1';
12              end if;
13          end if;
14      end process;
15
16      --- obsługa prawego przycisku (sterującego kierunkiem)
17      process(clk)
18      begin
19          if rising_edge(clk) then
20              dir_btn_shift <= dir_btn_shift(6 downto 0) & dir_btn;
21              if dir_btn_shift = "00000000" and dir_btn_last = '1' then
22                  direction <= not direction; -- jeśli przycisk jest wci
                        śnięty zmień kierunek
23                  dir_btn_last <= '0';
24              end if;
25              if dir_btn_shift = "11111111" then
26                  dir_btn_last <= '1';
27              end if;
28          end if;
29      end process;
```

Rysunek 3: Blok kodu w języku VHDL – obsługujących dane wejściowe z fizycznych przycisków.

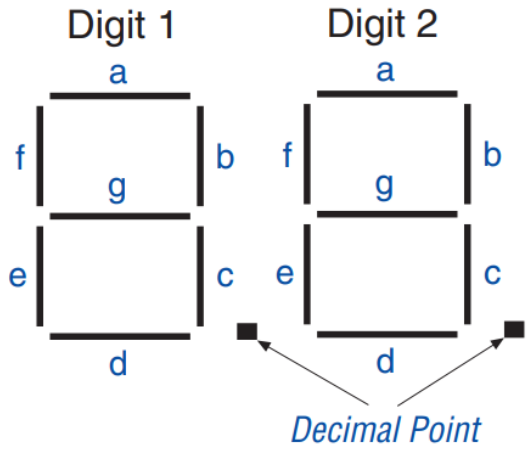
Żeby być w stanie odczytać faktyczną wartość z fizycznych przycisków stosujemy debouncing. Używamy do tego dwóch przesuwających się rejestrów dir_btn_shift i speed_btn_shift. Uznajemy, że przycisk jest faktycznie wciśnięty wtedy, gdy wszystkie bity w rejestrze przyjmują wartość '0', taka sytuacja zachodzi tylko wtedy, gdy przez osiem kolejnych cykli zegara przycisk utrzymuje wartość '0'. Dodatkowo wykorzystujemy zmienne speed_btn_last i dir_btn_last w celu zapewnienia, że przy jednym wciśnięciu przycisku wartość zmieni się tylko raz. Wczytywanie wartości z obu przycisków działa tak samo.

4.5 Obsługa przemieszczania się efektu świetlnego

Zegar w wybranym przez nas układzie ma częstotliwość 25.175 MHz – więc w wolniejszym trybie zmienna current_pos – śledząca pozycję efektu świetlnego będzie zmieniać mniej więcej raz na sekundę, a w szybszym pięć razy. Zależnie od wartości zmiennej dir – określającej kierunek zwiększamy lub zmiejszamy wartość zmiennej current_pos o jeden. Zmienna current_pos cyklicznie przyjmuje wartości z zakresu od 0 do 6.

```vhdl
-- Obsługa przemieszczania się efektu świetlnego
process(clk)
variable counter : integer range 0 to 30_000_000;
begin
    if rising_edge(clk) then
        counter := counter + 1;
        if (speed = '0' and counter >= 25_000_000) or (speed = '1'
             and counter >= 5_000_000) then
            counter := 0;
            if direction = '1' then
                if current_pos < 7 then
                    current_pos <= current_pos + 1;
                else
                    current_pos <= 0;
                end if;
            else
                if current_pos > 0 then
                    current_pos <= current_pos - 1;
                else
                    current_pos <= 7;
                end if;
            end if;
        end if;
    end if;
end process;
```

Rysunek 4: Blok kodu w języku VHDL – realizującego przemieszczenie efektu w zależności od zegara oraz zmiennych.



Rysunek 5: schemat wyświetlaczy 7-segmentowych

4.6 Ustawianie wyświetlaczy 7-segmentowych

```
    -- Ustawianie odpowiednich wartości wyświetlaczy 7-segmentowych
    process(current_pos)
    begin
        case current_pos is
        when 0 => segment_pattern <= "11111101111110"; -- A1 A2
        when 1 => segment_pattern <= "11111111111100"; -- A2 B2
        when 2 => segment_pattern <= "11111111111001"; -- B2 C2
        when 3 => segment_pattern <= "11111111110011"; -- C2 D2
        when 4 => segment_pattern <= "11101111110111"; -- D1 D2
        when 5 => segment_pattern <= "11001111111111"; -- D1 E1
        when 6 => segment_pattern <= "10011111111111"; -- E1 F1
        when 7 => segment_pattern <= "10111101111111"; -- A1 F1
        when others => segment_pattern <= (others => 'Z');
        end case;
    end process;

    segments1 <= segment_pattern(13 downto 7);
    segments2 <= segment_pattern(6 downto 0);
```

Rysunek 6: Blok kodu w języku VHDL – realizującego aktywację odpowiednich segmentów wyświetlacza,

Żeby odpowiednio zrealizować przemieszczanie się efektu świetlnego po wyświetlaczach – przypisujemy wektorowi segment_pattern odpowiednie wartości w zależności od wartości zmiennej current_pos. Następnie pierwszą część zmiennej segment_pattern przypisujemy do wyjścia podpiętego do prawego wyświetlacza, a drugą część do wyjścia podpiętego do lewego wyświetlacza.

5 Wnioski

5.1 Co można było zrobić inaczej:

- Realizując ten projekt skupiliśmy się na jednym układzie i jednym języku. Projekt pewnie wyglądałby znacząco inaczej, jeśli postanowilibyśmy pisać w języku SytemVerilog.
- Wprowadzić bardziej złożone efekty świetlne, takie jak dynamiczne zmiany wzorców, różne tryby wyświetlania.
- Na bieżąco weryfikować działanie kodu na fizycznej płytce. Ułatwiłoby to znacząco proces testowania poprawności pisanego kodu.

5.2 Zastosowania:

- Wyświetlacz na kuchence z licznikiem (wykonuje sygnał świetlny z programu, gdy jedzenie jest gotowe):

Rysunek 7: kuchenka

- Wyświetlacz boisku (wykonuje sygnał świetlny z programu, gdy został strzelony gol):

Rysunek 8: boisko