Laboratorium 4
Układ FPGA

Łukasz Kwinta, Kacper Kozubowski, Ida Ciepiela

maj 2024

Spis treści

1 Cel zadania

Celem laboratorium było zaprogramowanie układu FPGA tak aby wyświetlał animację poruszających się segmentów na obrzeżach wyświetlaczy 7 segmentowych. Należało również, zaimplementować prostą funkcjonalność obejmującą zmianę prędkości i kierunku ruchu, świetlnego odcinka, przy pomocy znajdujących się na płytce przycisków.

2 Czym są układy FPGA?

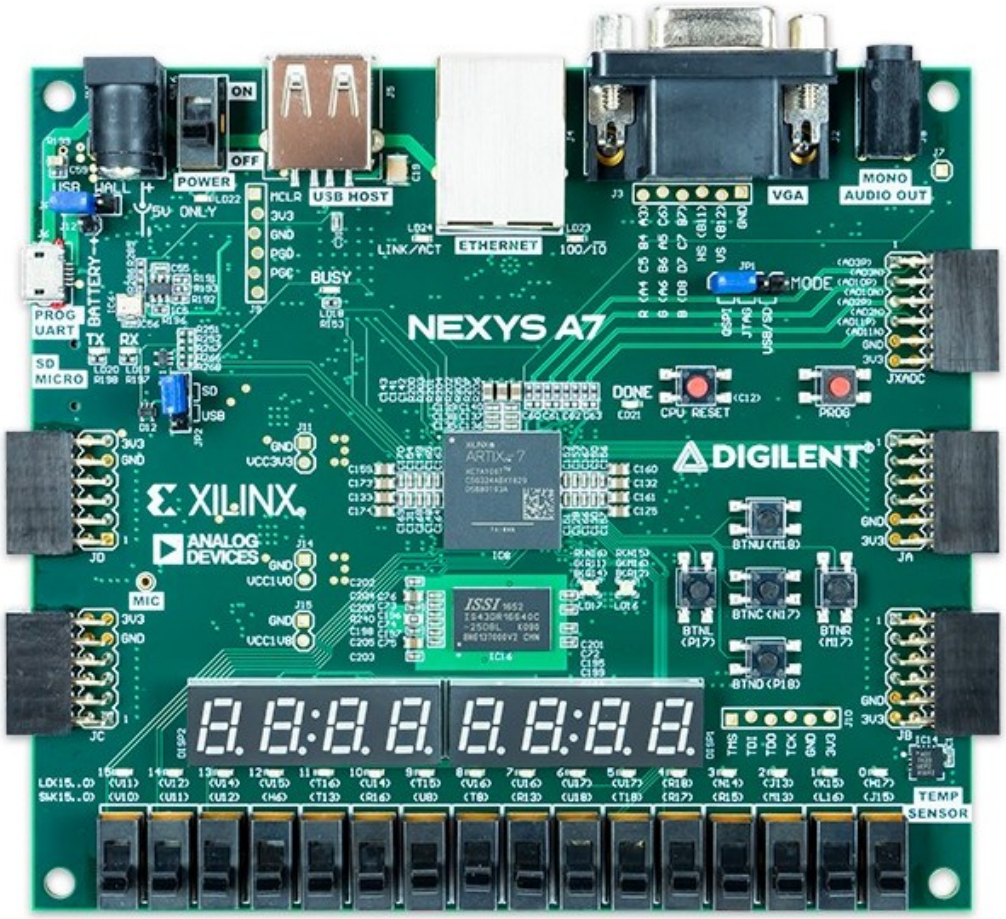
FPGA (Field-Programmable Gate Array) to rodzaj układu logicznego, który można programować po jego wyprodukowaniu. W przeciwieństwie do tradycyjnych układów ASIC (Application-Specific Integrated Circuit), które są zaprojektowane do wykonywania jednego konkretnego zadania i nie mogą być zmieniane po wyprodukowaniu, FPGA oferują elastyczność i możliwość wielokrotnego programowania. Kluczowym aspektem takiego układu jest matryca programowalnych bloków logicznych i konfigurowalnych połączeń.

Bloki logiczne to podstawowe jednostki wykonujące logikę i arytmetykę. Każdy blok zawiera programowalne elementy, takie jak bramki logiczne, multiplexery, oraz przerzutniki, które można konfigurować do wykonywania różnych funkcji. Natomiast, sieć połączeń umożliwia łączenie tych bloków w dowolny sposób.

Dzięki takiej konstrukcji układy FPGA można dostosować do różnych zastosowań, co czyni je niezwykle wszechstronnymi w inżynierii cyfrowej.

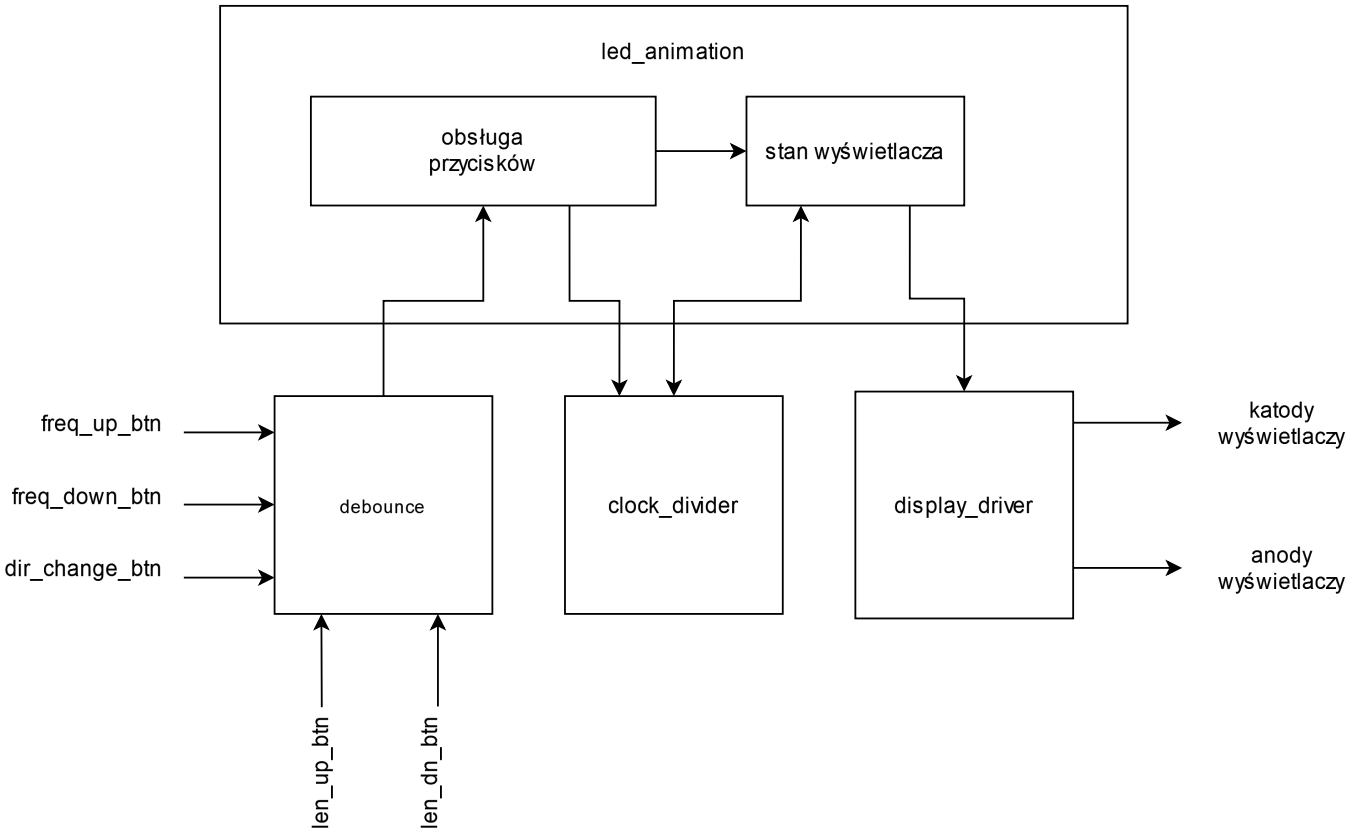
3 Realizacja

Do napisania programu na układ FPGA, spełniającego warunki zadania, wykorzystaliśmy język opisu sprzętu Verilog, a także oprogramowanie Vivado ML Edition od firmy Xilin. Dostarczone przez nas rozwiązanie zostało przygotowane na płytkę Nexys-A7 50T.



Rysunek 3.1: Wykorzystana płytka z układem FPGA

4 Rozwiązanie



Rysunek 4.1: Schemat blokowy rozwiązania

4.1 Moduł dzielnika częstotliwości

Aby w łatwy sposób zmieniać prędkość animacji, zdecydowaliśmy się na zastosowanie modułu dzielnika częstotliwości. Moduł przyjmuje na wejściu zegar systemowy oraz rejestr oznaczający obecny okres zegara wyjściowego. Następnie zlicza on takty zegara systemowego i gdy licznik dojdzie do zadanej wartości, zmienia stan zegara wyjściowego na przeciwny.

```
1  // moduł dzielący zegar
2  module clock_divider(
3      input integer clock_period,
4      input wire clk,
5
6      output reg divided_clock
7  );
8
9
10 initial
11     divided_clock <= 0;
12
13 longint counter_value = 0;
14
```

```
// zliczamy zadany okres zegara (ilość cykli zegara wejściowego), i gdy
// doliczymy do konca zmieniamy stan spowolnionego zegara na przeciwny
always@ (posedge clk)
begin
    if (counter_value >= clock_period)
        begin
            divided_clock <= ~divided_clock;
            counter_value <= 0;
        end
    else
        begin
            divided_clock <= divided_clock;
            counter_value <= counter_value + 1;
        end
end

endmodule
```

4.2 Moduł filtrujący przyciski

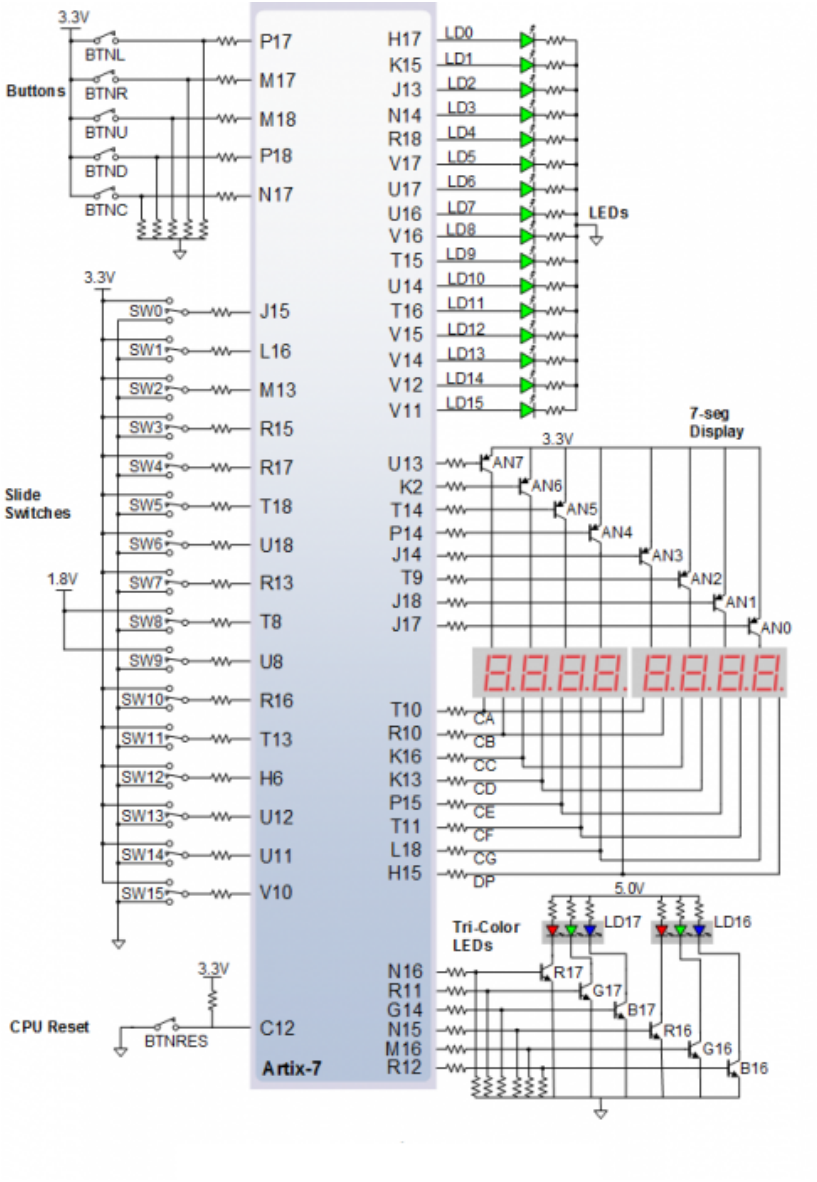
Aby uniknąć efektu drgania styków przycisków, zdecydowaliśmy się na zastosowanie modułu filtrującego wejście przycisków. Działa on na bardzo prostej zasadzie - zlicza on ilość cykli zegara systemowego w których przycisk jest w stanie wysokim - wciśnięty. Gdy ilość cykli przekroczy zadaną wartość, przycisk uznawany jest za wciśnięty. Każdy stan niski pomiędzy kolejnymi resetuje licznik. Długość odliczania można ustawić poprzez parametr DEBOUNCE_TIME przy instancjonowaniu modułu.

```
// moduł filtrujący przyciski
module debounce #(parameter DEBOUNCE_TIME = 1000 * 100) (
    input wire clk,
    input wire button_physical,

    output reg button_active
);

// ustawiamy początkowy stan przycisku na 0
initial
    button_active = 0;

integer btn_clock_cycles_counter = 0;

// zliczamy zadaną ilość cykli zegara
// jeśli w którymś cyklu przycisk będzie w stanie niskim,
// resetujemy licznik wartości
always@ (posedge clk)
begin
   if (button_physical == 1)
       begin
            btn_clock_cycles_counter <= btn_clock_cycles_counter + 1;
            if (btn_clock_cycles_counter >= DEBOUNCE_TIME)
                button_active <= 1;
       end
```

```
26      else
27          begin
28              btn_clock_cycles_counter <= 0;
29              button_active <= 0;
30          end
31
32  end
33
34  endmodule
```

4.3 Moduł sterujący wyświetlaczami

Aby wyświetlać wiele segmentów wielu wyświetlaczach 7 segmentowych na raz, musieliśmy zaimplementować moduł sterujący wyświetlaczami. Moduł ten odświeża wyświetlacze z zadaną częstotliwością po kolei, tak aby stworzyć wrażenie, że wiele wyświetlaczy aktywnych jest w jednym czasie.



Rysunek 4.2: Schemat podpięcia wyświetlaczy do układu FPGA

Zabieg ten musieliśmy zastosować gdyż wszystkie wyświetlacze mają wspólne katody segmentów co oznacza, że przy aktywacji anody wielu wyświetlaczy w jednym czasie, będą one wyświetlać te same segmenty.

```
1  module displays_driver #(parameter REFERESH_PERIOD = 100 * 1000)(
2      input wire clk,
3
4      // rejestr wejściowy określający stan wszystkich wyświetlaczy
5      input reg [7:0] display [7:0],
6
7      // wyjscia steujace wyswietlaczami
8      // stan niski na danym indeksie aktywuje dany wyswietlacz
9      output reg [7:0] sseg_anodes,
10
11     // |  7  |  6  |  5  |  4  |  3  |  2  |  1  |  0  |
12     // | DP  |  CG | CF  | CE  | CD  | CC  | CB  | CA  |
13     // stan niski na danym indeksie aktywuje dany segment na wszystkich aktywnych wyswietlaczach
```

```
    output reg [7:0] sseg_cathodes
);

initial
begin
    sseg_anodes <= '1;
    sseg_cathodes <= '1;
end

reg refresh_clk; //1 khz refresh clock
clock_divider clk_div (
    .clock_period(REFERESH_PERIOD),
    .clk(clk),
    .divided_clock(refresh_clk)
);

reg [3:0] display_number = 0;

always@ (posedge refresh_clk)
begin
    sseg_anodes = '1;
    sseg_anodes[display_number] <= 0;
    sseg_cathodes <= 8'(~display[display_number]);

    display_number = display_number + 1;
    if (display_number >= `DISPLAY_COUNT)
        display_number <= 0;
end

endmodule
```

4.4 Właściwy moduł generujący animację

```

```

5 Zastosowania układów FPGA

6 Wnioski