Laboratorium 4
Układ FPGA

Łukasz Kwinta, Kacper Kozubowski, Ida Ciepiela

maj 2024

Spis treści

1 Cel zadania

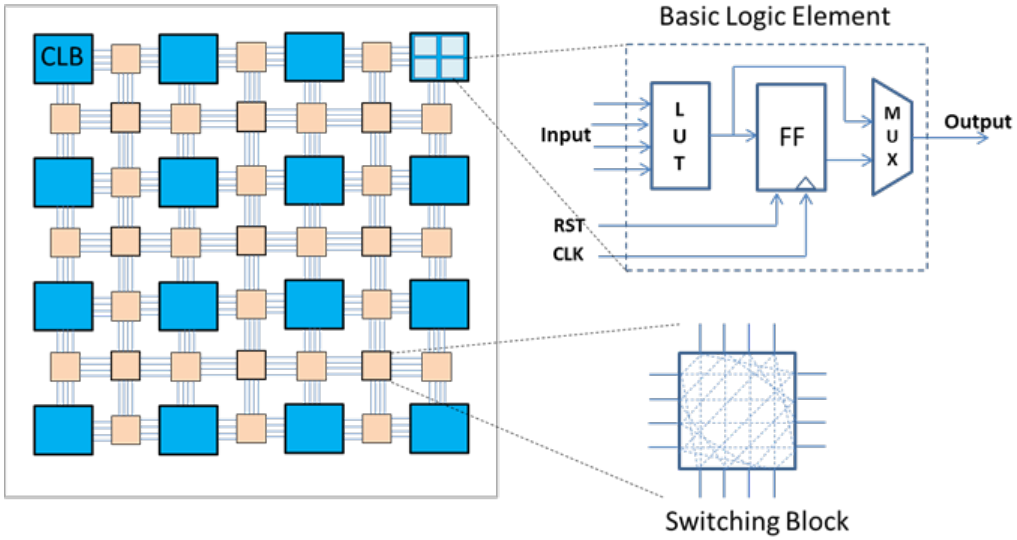
Celem laboratorium było zaprogramowanie układu FPGA tak aby wyświetlał animację poruszających się segmentów na obrzeżach wyświetlaczy 7 segmentowych. Należało również, zaimplementować prostą funkcjonalność obejmującą zmianę prędkości i kierunku ruchu świetlnego odcinka, przy pomocy znajdujących się na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

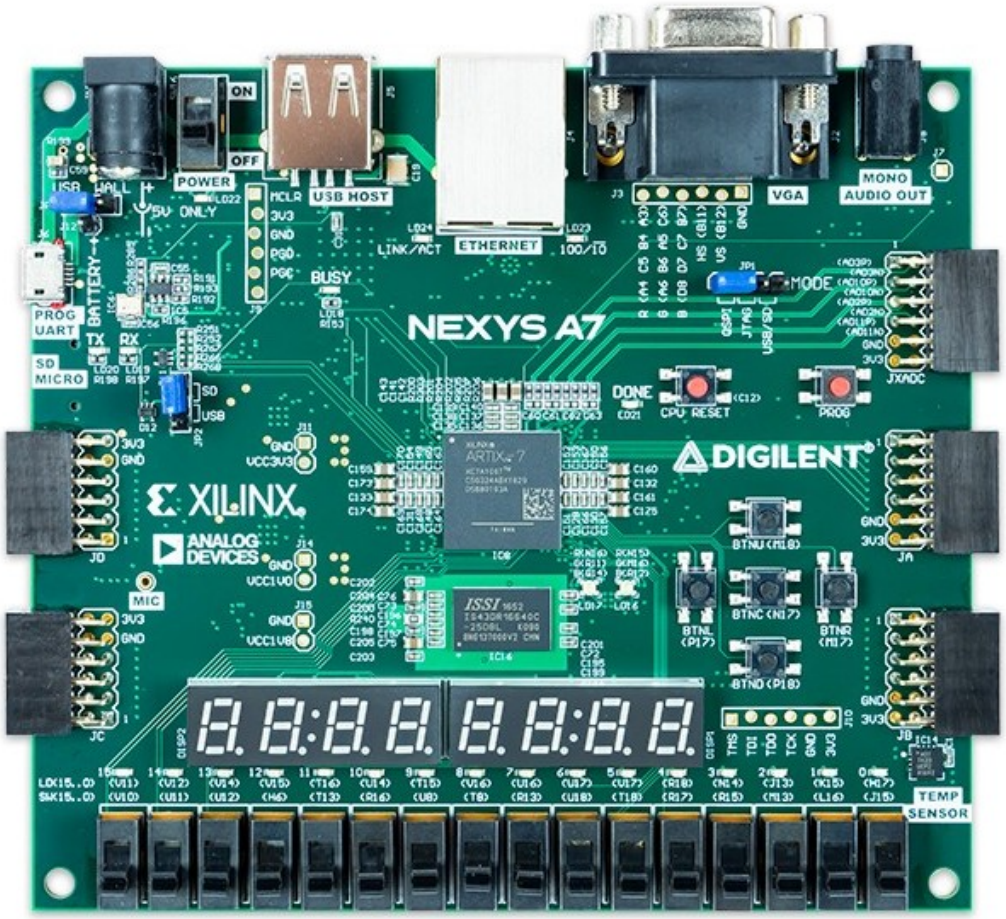
Dzięki takiej konstrukcji układy FPGA można dostosować do różnych zastosowań, co czyni je niezwykle wszechstronnymi w inżynierii cyfrowej.



Rysunek 2.1: Przykładowy schemat układu FPGA

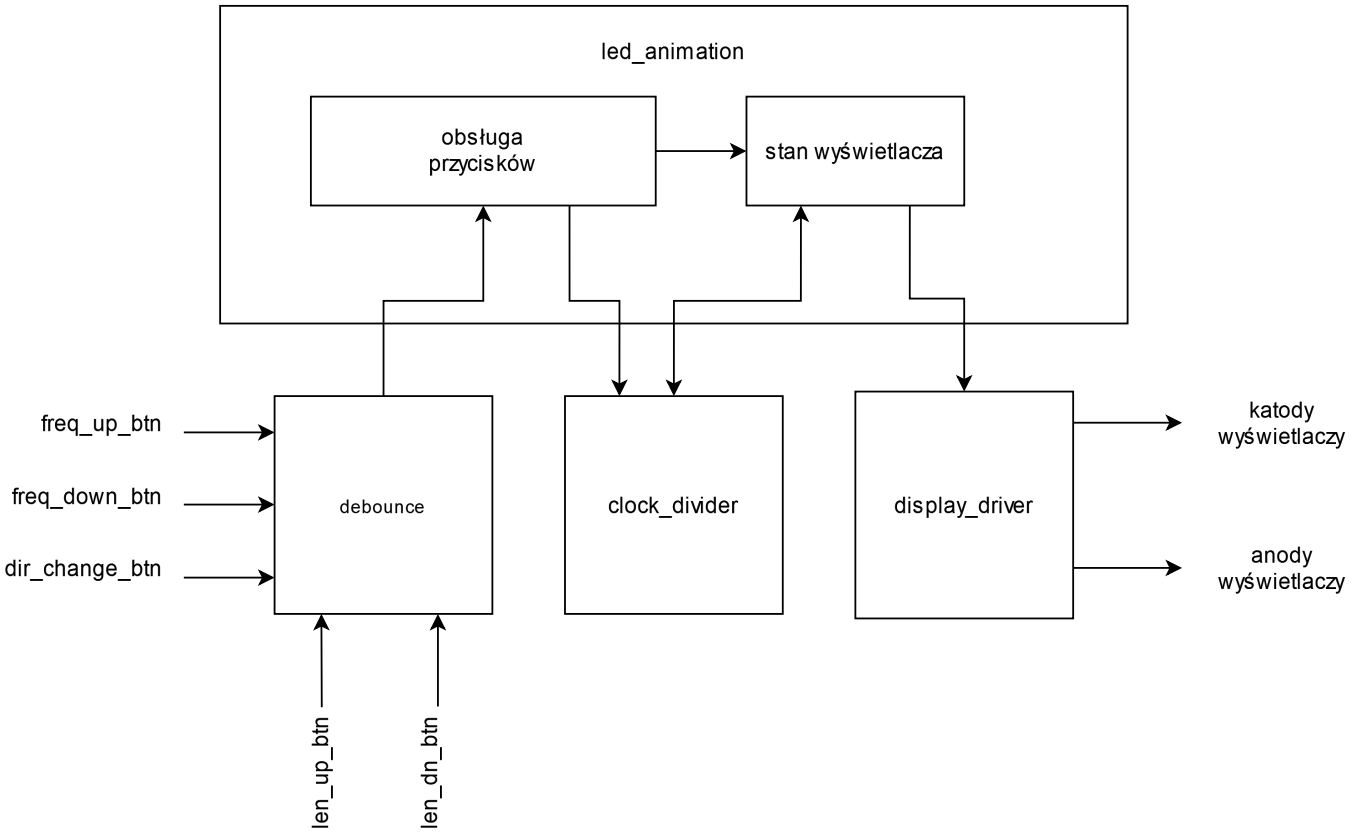
3 Realizacja

Do napisania programu na układ FPGA, spełniającego warunki zadania, wykorzystaliśmy język opisu sprzętu Verilog, a także oprogramowanie Vivado ML Edition od firmy Xilinx. 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

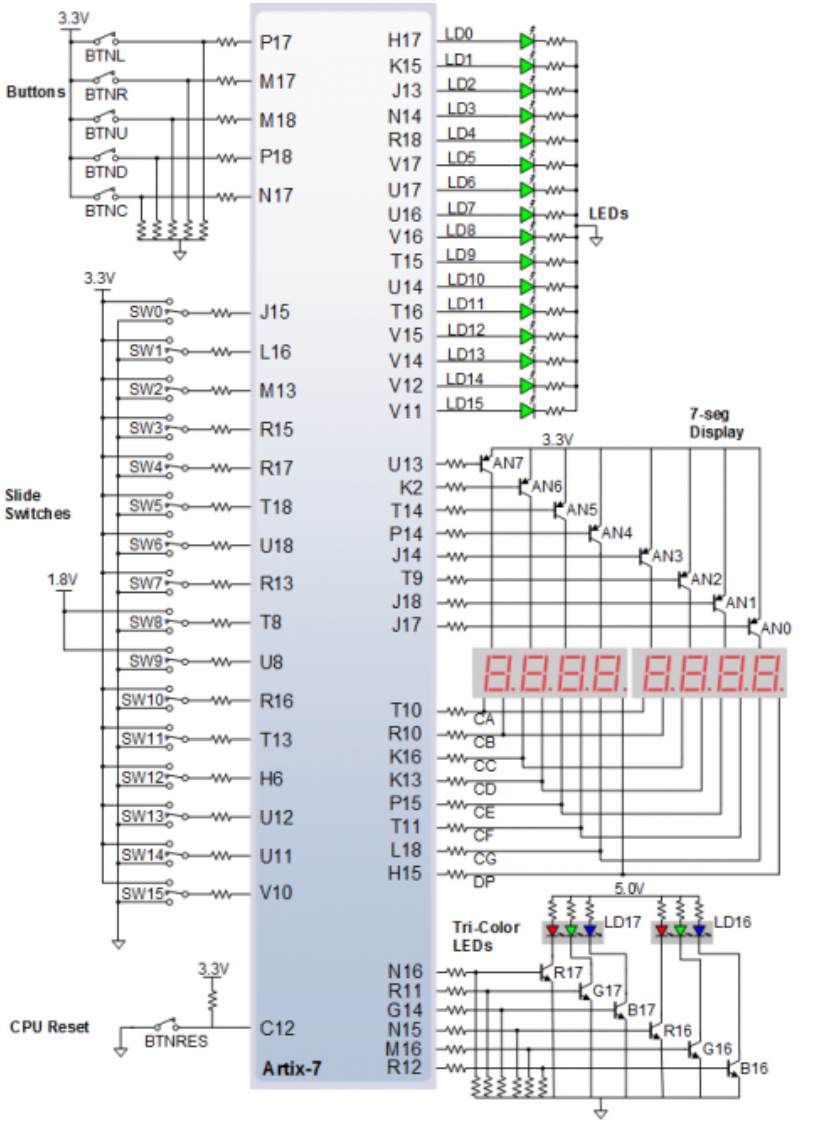
Aby uniknąć efektu drgania styków przycisków, zdecydowaliśmy się na zastosowanie modułu filtrującego wejście przycisków. Działa on na bardzo prostej zasadzie - zlicza on ilość cykli zegara systemowego w których przycisk jest w stanie wysokim - wciśnięty. Gdy ilość cykli przekroczy zadaną wartość, przycisk 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
```

```
    // rejestr wejściowy określający stan wszystkich wyświetlaczy
    input reg [7:0] display [7:0],

    // wyjscia steujace wyswietlaczami
    // stan niski na danym indeksie aktywuje dany wyswietlacz
    output reg [7:0] sseg_anodes,

    // |   7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
    // |  DP   |   CG  |  CF   |  CE   |  CD   |  CC   |  CB   |  CA   |
    // 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

Główny moduł całego programu. Definiuje użyteczne makra, obsługuje wszystkie inne moduły przekazując im odpowiednie argumenty, przyjmuje i obsługuje sygnały wejściowe z przycisków oraz kontroluje wyświetlanie animacji. Dzięki odpowiednio zaimplementowanej logice możliwe jest wywołanie animacji na dowolnej liczbie liniowo ułożonych wyświetlaczy, po zmianie jednego parametru.

```
`timescale 1ns / 1ps // tylko dla symulacji

// Przydatne makra dla wyswietlaczow
`define SEGMENT_A_MASK 8'(1 << 0)
`define SEGMENT_B_MASK 8'(1 << 1)
```

```
`define SEGMENT_C_MASK 8'(1 << 2)
`define SEGMENT_D_MASK 8'(1 << 3)
`define SEGMENT_E_MASK 8'(1 << 4)
`define SEGMENT_F_MASK 8'(1 << 5)
`define SEGMENT_G_MASK 8'(1 << 6)
`define SEGMENT_DOT_MASK 8'(1 << 7)
`define DISPLAY_CLEAR '0
`define DISPLAY_ALL '1

`define DISPLAY_REFRESH_FREQUENCY (100 * 1000 / 8) // częstotliwość odświeżania ekranów = 0,125kHz
`define START_ANIMATION_PERIOD (1000*100*1000 / 4) // okres startowy = 0,25s
`define MAX_ANIMATION_PERIOD 1000*100*1000*100 // maks okres = 100s
`define MIN_ANIMATION_PERIOD 1

`define DISPLAY_COUNT 8 // liczba wykorzystywanych wyświetlaczy

`define MAX_SNAKE_LENGTH (`DISPLAY_COUNT*2 + 3)
`define START_SNAKE_LENGTH 1
`define MIN_SNAKE_LENGTH 1


/* Definicja modułu z wejsciami i wyjsciami zdefiniowanymi w pliku .xdc */
module led_animation(
    // wejscia przycisków
    input wire btn_freq_up,
    input wire btn_freq_dn,
    input wire btn_dir,
    input wire len_up,
    input wire len_dn,

    input wire clk, // zegar systemowy 100Mhz

    // wyjscia steujace wyswietlaczami
    // stan niski na danym indeksie aktywuje dany wyswietlacz
    output reg [7:0] sseg_anodes,

    // |  7  |  6  |  5  |  4  |  3  |  2  |  1  |  0  |
    // | DP  |  CG | CF  | CE  | CD  | CC  | CB  | CA  |
    // stan niski na danym indeksie aktywuje dany segment na wszystkich aktywnych wyswietlaczach
    output reg [7:0] sseg_cathodes,

    // wyjscie sterujace ledem
    output wire clk_led
);

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

// startowy okres spowolnionego zegara
longint clock_period = `START_ANIMATION_PERIOD;

// użycie modułu spowalniającego zegar
reg divided_clk;
assign clk_led = divided_clk;
clock_divider clk_div (
    .clock_period(clock_period),
    .clk(clk),
    .divided_clock(divided_clk)
```

```
67  );
68
69  reg [7:0] display [7:0];
70  displays_driver display_driver (
71      .clk(clk),
72      .display(display),
73      .sseg_anodes(sseg_anodes),
74      .sseg_cathodes(sseg_cathodes)
75  );
76  defparam display_driver.REFERESH_PERIOD = `DISPLAY_REFRESH_FREQUENCY;
77
78  // definicja kierunku poruszania sie segmentu
79  enum {LEFT, RIGHT} dir = RIGHT;
80
81  localparam num_of_segments = (4 + `DISPLAY_COUNT * 2);
82  integer curr_state = 0;
83  integer curr_display = 0;
84
85  // robimy cykliczną kolejkę do przechowywania informacji o zaświeconych segmentach
86  integer snake[0:num_of_segments][0:1];
87  integer head = `START_SNAKE_LENGTH - 1;
88  integer tail = 0;
89
90  integer length = `START_SNAKE_LENGTH;
91  integer old_length = `START_SNAKE_LENGTH;
92
93  integer i;
94  initial begin
95      for (i = 0; i < num_of_segments; i = i + 1)
96          begin
97              snake[i][0] = -1;
98              snake[i][1] = 0;
99          end
100     for (i = 0; i < `DISPLAY_COUNT; i = i + 1)
101         display[i] = `DISPLAY_CLEAR;
102 end
103
104 // przejście na kolejny segment
105 always@ (posedge divided_clk)
106 begin
107     if (dir == RIGHT)
108         curr_state = (curr_state + 1);
109     else
110         curr_state = (curr_state - 1);
111
112     if (curr_state < 0)
113         curr_state = num_of_segments + curr_state;
114
115     else if (curr_state > num_of_segments - 1)
116         curr_state = curr_state - num_of_segments;
117 end
118
119 always@ (posedge divided_clk)
120 begin
121     if (old_length > length)
122         begin
123             display[snake[tail][0]] = display[snake[tail][0]] - 2**(snake[tail][1]);
124
125             tail = tail + 1;
126             old_length = old_length - 1;
127             if (tail == num_of_segments)
```

```
                tail = 0;
            display[snake[tail][0]] = display[snake[tail][0]] - 2**(snake[tail][1]);
        end
    else if (old_length < length)
        begin
            tail = tail - 1;
            old_length = old_length + 1;
            if (tail < 0)
                tail = num_of_segments - 1;
        end

    else if (snake[tail][0] != -1) // gasimy ogon
        begin
            display[snake[tail][0]] = display[snake[tail][0]] - 2**(snake[tail][1]);
        end

    // uaktualnienie wskaźników
    tail = tail + 1;
    head = head + 1;
    if (head == num_of_segments)
        head = 0;
    if (tail == num_of_segments)
        tail = 0;

    snake[head][0] = curr_display;

    if (curr_display == 0) // obsługa prawego wyświetlacza
        begin
            snake[head][1] = curr_state;
            display[0] = display[0] + 2**curr_state;

            if ((dir == LEFT && curr_state == 0) || (dir == RIGHT && curr_state == 3))
                curr_display = curr_display + 1;
        end

    else if (curr_display == (`DISPLAY_COUNT - 1)) // obsługa lewego wyświetlacza
        begin
            snake[head][1] = (curr_state - (`DISPLAY_COUNT - 1) < 6 ? curr_state - (`DISPLAY_COUNT - 1) : 0);
            display[`DISPLAY_COUNT - 1] = display[`DISPLAY_COUNT - 1] + 2**(curr_state - (`DISPLAY_COUNT - 1) < 6
                                                                            ? curr_state - (`DISPLAY_COUNT - 1) : 0);

            if ((dir == LEFT && curr_state == (`DISPLAY_COUNT + 2))
                 || (dir == RIGHT && curr_state == (`DISPLAY_COUNT + 5)))
                curr_display = curr_display - 1;
        end

    else if (curr_state > 3 && curr_state < (`DISPLAY_COUNT + 2)) // obsługa dolnych segmentów
        begin
            snake[head][1] = 3;
            display[curr_display] = display[curr_display] + 2**3;

            if (dir == RIGHT)
                curr_display = (curr_display + 1) & (`DISPLAY_COUNT - 1);

            else
                curr_display = (curr_display - 1) & (`DISPLAY_COUNT - 1);
        end
    else // obsługa górnych segmentów
        begin
            snake[head][1] = 0;
            display[curr_display] = display[curr_display] + 2**0;
```

```
189
190             if (dir == RIGHT)
191                 curr_display = (curr_display - 1) & (`DISPLAY_COUNT - 1);
192
193             else
194                 curr_display = (curr_display + 1) & (`DISPLAY_COUNT - 1);
195         end
196 end
197
198
199
200 // użycie modułu filtrującego zakłócenia przycisków
201 reg button_dir_active, button_freq_up_active, button_freq_dn_active, button_len_dn_active, button_len_up_active;
202 debounce dir_debounce (
203     .clk(clk),
204     .button_physical(btn_dir),
205     .button_active(button_dir_active)
206 );
207
208 debounce freq_up_debounce (
209     .clk(clk),
210     .button_physical(btn_freq_up),
211     .button_active(button_freq_up_active)
212 );
213
214 debounce freq_dn_debounce (
215     .clk(clk),
216     .button_physical(btn_freq_dn),
217     .button_active(button_freq_dn_active)
218 );
219
220 debounce len_dn_debounce (
221     .clk(clk),
222     .button_physical(len_dn),
223     .button_active(button_len_dn_active)
224 );
225
226 debounce len_up_debounce (
227     .clk(clk),
228     .button_physical(len_up),
229     .button_active(button_len_up_active)
230 );
231
232 reg button_freq_up_old = 0;
233 reg button_freq_dn_old = 0;
234 reg button_dir_old = 0;
235 reg button_len_up_old = 0;
236 reg button_len_dn_old = 0;
237
238 reg button_freq_up_raise = 0;
239 reg button_freq_dn_raise = 0;
240 reg button_dir_raise = 0;
241 reg button_len_up_raise = 0;
242 reg button_len_dn_raise = 0;
243
244 // blok wykrywający narastające stanu przyciku poprzez porównanie starej wartosci z nową wartością
245 always@ (posedge clk)
246 begin
247     if (button_freq_up_active != button_freq_up_old && button_freq_up_active == 1)
248         button_freq_up_raise <= 1;
249
```

```
250     if (button_freq_dn_active != button_freq_dn_old && button_freq_dn_active == 1)
251         button_freq_dn_raise <= 1;
252 
253     if (button_dir_active != button_dir_old && button_dir_active == 1)
254         button_dir_raise <= 1;
255 
256     if (button_len_up_active != button_len_up_old && button_len_up_active == 1)
257         button_len_up_raise <= 1;
258 
259     if (button_len_dn_active != button_len_dn_old && button_len_dn_active == 1)
260         button_len_dn_raise <= 1;
261 
262     if (button_freq_up_raise == 1)
263         if (clock_period > `MIN_ANIMATION_PERIOD)
264             clock_period <= clock_period >> 1;
265 
266     if (button_freq_dn_raise == 1)
267         if (clock_period < `MAX_ANIMATION_PERIOD)
268             clock_period <= clock_period << 1;
269 
270     if (button_dir_raise == 1)
271         if(dir == LEFT) dir <= RIGHT;
272         else            dir <= LEFT;
273 
274     if (button_len_up_raise == 1)
275         if (length < `MAX_SNAKE_LENGTH)
276             length <= length + 1;
277 
278     if (button_len_dn_raise == 1)
279         if (length > `MIN_SNAKE_LENGTH)
280              length <= length - 1;
281 
282     button_freq_dn_old <= button_freq_dn_active;
283     button_freq_up_old <= button_freq_up_active;
284     button_dir_old <= button_dir_active;
285     button_len_up_old <= button_len_up_active;
286     button_len_dn_old <= button_len_dn_active;
287 
288     button_freq_up_raise = 0;
289     button_freq_dn_raise = 0;
290     button_dir_raise = 0;
291     button_len_up_raise = 0;
292     button_len_dn_raise = 0;
293 end
294 
295 endmodule
```

5 Zastosowania układów FPGA

Układy FPGA znajdują szerokie zastosowanie w wielu dziedzinach, dzięki swojej elastyczności, wydajności i możliwości szybkiej rekonfiguracji. Poniżej przedstawiamy kilka przykładów:

- Są wykorzystywane w routerach, switchach i bramkach sieciowych do szybkiego przetwarzania pakietów danych.
- Wykorzystywane są w urządzeniach audio, sprzęcie muzycznym, systemach obróbki obrazu oraz w systemach medycznych, takich jak USG czy MRI
- Układy FPGA są używane do sterowania silnikami, czujnikami, systemami wizyjnymi oraz w systemach kontroli jakości w procesach produkcyjnych.
- FPGA są wykorzystywane w systemach ADAS (Advanced Driver Assistance Systems) do przetwarzania danych z czujników, takich jak kamery, radar czy lidar, oraz do szybkiego podejmowania decyzji na podstawie analizy otoczenia pojazdu.
- W samochodach autonomicznych FPGA są również stosowane do bezpiecznego sterowania systemami napędowymi i hamulcowymi.

Jak widać Zastosowania układów FPGA są niezwykle różnorodne i stale się rozwijają dzięki zmniejszającym się kosztom produkcji i dostęp do tej technologi.

6 Wnioski

Podczas realizacji laboratorium z układów FPGA zdobyliśmy praktyczne doświadczenie w programowaniu i konfigurowaniu tego typu układów. Wykorzystanie języka Verilog oraz narzędzia Vivado ML Edition od Xilinx dało nam cenną wiedzę w obszarze programowania sprzętu.

Praca nad projektem pozwoliła nam zrozumieć układy FPGA oraz ich elastyczność w zakresie programowania. Ponadto, zdaliśmy sobie sprawę z ich licznych praktycznych zastosowań w otaczającym nas świecie i zrozumieliśmy jak wielki potencjał skrywa ta niepozorna technologia.