Laboratorium 4
Układ FPGA

Łukasz Kwinta, Kacper Kozubowski, Ida Ciepiela

maj 2024

Spis treści

1 Cel zadania

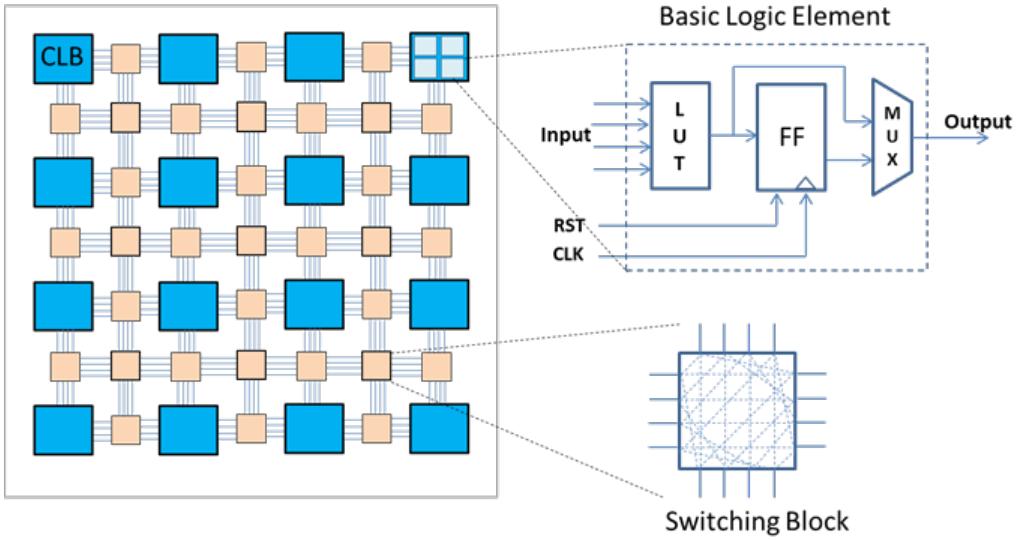
Celem laboratorium było zaprogramowanie układu FPGA tak aby wyświetlał animację poruszających się segmentów na obrzeżach wyświetlaczy 7 segmentowych. Należało również, zaimplementować prostą funkcjonalność obejmującą zmianę prędkości i kierunku ruchu świetlnego odcinka, przy pomocy znajdujących się na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

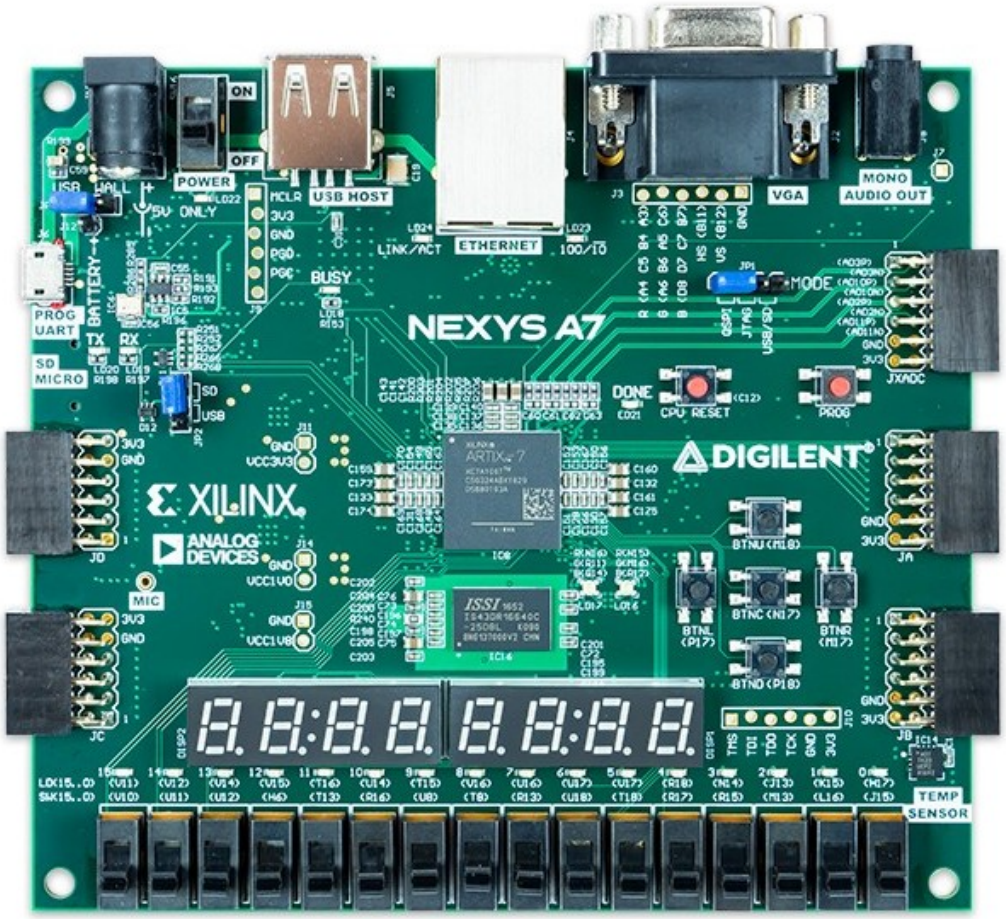
Dzięki takiej konstrukcji układy FPGA można dostosować do różnych zastosowań, co czyni je niezwykle wszechstronnymi w inżynierii cyfrowej.



Rysunek 2.1: Przykładowy schemat układu FPGA

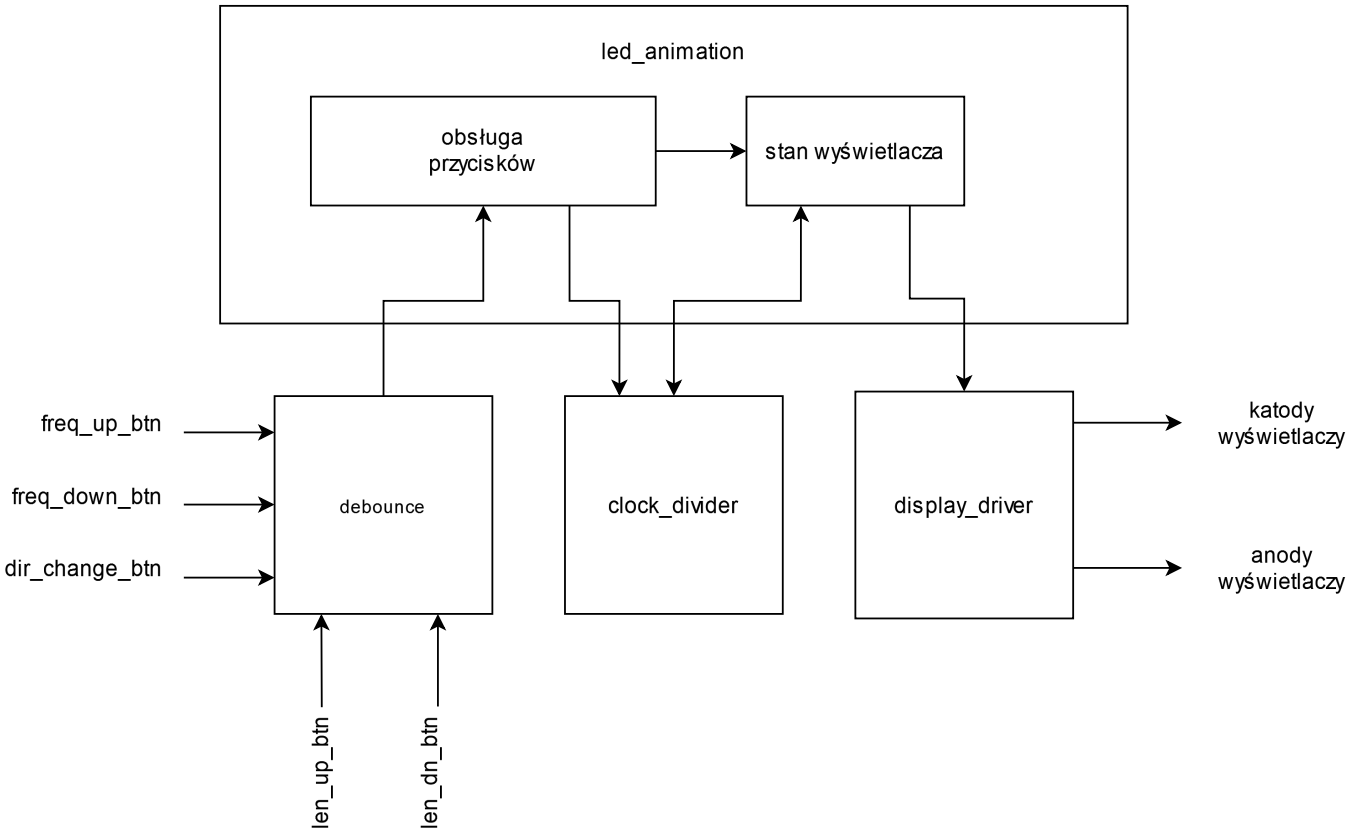
3 Realizacja

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

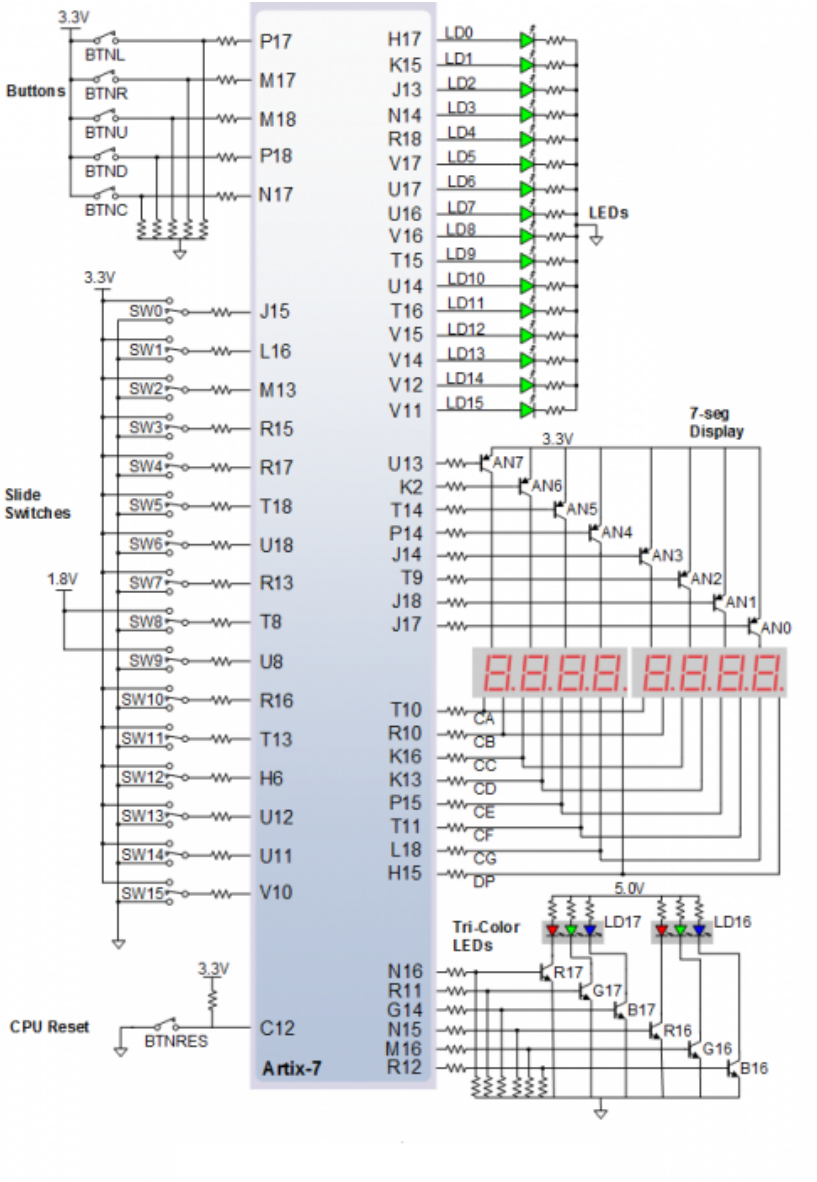
Aby uniknąć efektu drgania styków przycisków, zdecydowaliśmy się na zastosowanie modułu filtrującego wejście przycisków. Działa on na bardzo prostej zasadzie - zlicza on ilość cykli zegara systemowego w których przycisk jest w stanie wysokim - wciśnięty. Gdy ilość cykli przekroczy zadaną wartość, przycisk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

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 SET_ACTIVE(mask) 8'(~mask)

`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
```

```
16
17  `define START_ANIMATION_PERIOD (100 * 1000 / 8) // maks okres = 0,25s
18  `define MAX_ANIMATION_PERIOD 1000*100*1000*100 // maks okres = 100s
19  `define MIN_ANIMATION_PERIOD 1
20
21  `define DISPLAY_COUNT 8 // liczba wykorzystywanych wyœwietlaczy
22
23  `define MAX_SNAKE_LENGTH (`DISPLAY_COUNT*2 + 3)
24  `define START_SNAKE_LENGTH 1
25  `define MIN_SNAKE_LENGTH 1
26
27
28  /* Definicja modu³u z wejsciami i wyjsciami zdefiniowanymi w pliku .xdc */
29  module led_animation(
30      // wejscia przycisków
31      input wire btn_freq_up,
32      input wire btn_freq_dn,
33      input wire btn_dir,
34      input wire len_up,
35      input wire len_dn,
36
37      input wire clk, // zegar systemowy 100Mhz
38
39      // wyjscia steujace wyswietlaczami
40      // stan niski na danym indeksie aktywuje dany wyswietlacz
41      output reg [7:0] sseg_anodes,
42
43      // |   7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
44      // |  DP   |   CG  |  CF   |  CE   |  CD   |  CC   |  CB   |  CA   |
45      // stan niski na danym indeksie aktywuje dany segment na wszystkich aktywnych wyswietlaczach
46      output reg [7:0] sseg_cathodes,
47
48      // wyjscie sterujace ledem
49      output wire clk_led
50  );
51
52  initial
53  begin
54      sseg_cathodes <= '1;
55      sseg_anodes <= '1;
56  end
57
58  // startowy okres spowolnionego zegara
59  longint clock_period = 1000*100*1000 / 4;
60
61  // u¿ycie modu³u spowalniaj¹cego zegar
62  reg divided_clk;
63  assign clk_led = divided_clk;
64  clock_divider clk_div (
65      .clock_period(clock_period),
66      .clk(clk),
67      .divided_clock(divided_clk)
68  );
69
70  reg [7:0] display [7:0];
71  displays_driver display_driver (
72      .clk(clk),
73      .display(display),
74      .sseg_anodes(sseg_anodes),
75      .sseg_cathodes(sseg_cathodes)
76  );
```

```
77  defparam display_driver.REFERESH_PERIOD = `START_ANIMATION_PERIOD;
78
79  // definicja kierunku poruszania sie segmentu
80  enum {LEFT, RIGHT} dir = RIGHT;
81
82  localparam num_of_segments = (4 + `DISPLAY_COUNT * 2);
83  integer curr_state = 0;
84  integer curr_display = 0;
85
86  // robimy cykliczn¹ kolejkê do przechowywania informacji o zaœwieconych segmentach
87  integer snake[0:num_of_segments][0:1];
88  integer head = `START_SNAKE_LENGTH - 1;
89  integer tail = 0;
90
91  integer length = `START_SNAKE_LENGTH;
92  integer old_length = `START_SNAKE_LENGTH;
93
94  integer i;
95  initial begin
96      for (i = 0; i < num_of_segments; i = i + 1)
97          begin
98              snake[i][0] = -1;
99              snake[i][1] = 0;
100         end
101     for (i = 0; i < `DISPLAY_COUNT; i = i + 1)
102         display[i] = `DISPLAY_CLEAR;
103 end
104
105 // przejœcie na kolejny segment
106 always@ (posedge divided_clk)
107 begin
108     if (dir == RIGHT)
109         curr_state = (curr_state + 1);
110     else
111         curr_state = (curr_state - 1);
112
113     if (curr_state < 0)
114         curr_state = num_of_segments + curr_state;
115
116     else if (curr_state > num_of_segments - 1)
117         curr_state = curr_state - num_of_segments;
118 end
119
120 always@ (posedge divided_clk)
121 begin
122     if (old_length > length)
123         begin
124             display[snake[tail][0]] = display[snake[tail][0]] - 2**(snake[tail][1]);
125
126             tail = tail + 1;
127             old_length = old_length - 1;
128             if (tail == num_of_segments)
129                 tail = 0;
130             display[snake[tail][0]] = display[snake[tail][0]] - 2**(snake[tail][1]);
131         end
132     else if (old_length < length)
133         begin
134             tail = tail - 1;
135             old_length = old_length + 1;
136             if (tail < 0)
137                 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 wskaŸników
    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) // obs³uga prawego wyœwietlacza
        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)) // obs³uga lewego wyœwietlacza
        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)) // obs³uga 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 // obs³uga górnych segmentów
        begin
            snake[head][1] = 0;
            display[curr_display] = display[curr_display] + 2**0;

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

            else
                curr_display = (curr_display + 1) & (`DISPLAY_COUNT - 1);
        end
end

```

```
199
200
201 // u¿ycie modu³u filtruj¹cego zak³ócenia przycisków
202 reg button_dir_active, button_freq_up_active, button_freq_dn_active, button_len_dn_active, button_len_up_active;
203 debounce dir_debounce (
204     .clk(clk),
205     .button_physical(btn_dir),
206     .button_active(button_dir_active)
207 );
208
209 debounce freq_up_debounce (
210     .clk(clk),
211     .button_physical(btn_freq_up),
212     .button_active(button_freq_up_active)
213 );
214
215 debounce freq_dn_debounce (
216     .clk(clk),
217     .button_physical(btn_freq_dn),
218     .button_active(button_freq_dn_active)
219 );
220
221 debounce len_dn_debounce (
222     .clk(clk),
223     .button_physical(len_dn),
224     .button_active(button_len_dn_active)
225 );
226
227 debounce len_up_debounce (
228     .clk(clk),
229     .button_physical(len_up),
230     .button_active(button_len_up_active)
231 );
232
233 reg button_freq_up_old = 0;
234 reg button_freq_dn_old = 0;
235 reg button_dir_old = 0;
236 reg button_len_up_old = 0;
237 reg button_len_dn_old = 0;
238
239 reg button_freq_up_raise = 0;
240 reg button_freq_dn_raise = 0;
241 reg button_dir_raise = 0;
242 reg button_len_up_raise = 0;
243 reg button_len_dn_raise = 0;
244
245 // blok wykrywaj¹cy narastaj¹ce stanu przyciku poprzez porównanie starej wartosci z now¹ wartoœci¹
246 always@ (posedge clk)
247 begin
248     if (button_freq_up_active != button_freq_up_old && button_freq_up_active == 1)
249         button_freq_up_raise <= 1;
250
251     if (button_freq_dn_active != button_freq_dn_old && button_freq_dn_active == 1)
252         button_freq_dn_raise <= 1;
253
254     if (button_dir_active != button_dir_old && button_dir_active == 1)
255         button_dir_raise <= 1;
256
257     if (button_len_up_active != button_len_up_old && button_len_up_active == 1)
258         button_len_up_raise <= 1;
259
```

```
260     if (button_len_dn_active != button_len_dn_old && button_len_dn_active == 1)
261         button_len_dn_raise <= 1;
262 
263     if (button_freq_up_raise == 1)
264         if (clock_period > `MIN_ANIMATION_PERIOD)
265             clock_period <= clock_period >> 1;
266 
267     if (button_freq_dn_raise == 1)
268         if (clock_period < `MAX_ANIMATION_PERIOD)
269             clock_period <= clock_period << 1;
270 
271     if (button_dir_raise == 1)
272         if(dir == LEFT) dir <= RIGHT;
273         else            dir <= LEFT;
274 
275     if (button_len_up_raise == 1)
276         if (length < `MAX_SNAKE_LENGTH)
277             length <= length + 1;
278 
279     if (button_len_dn_raise == 1)
280         if (length > `MIN_SNAKE_LENGTH)
281              length <= length - 1;
282 
283     button_freq_dn_old <= button_freq_dn_active;
284     button_freq_up_old <= button_freq_up_active;
285     button_dir_old <= button_dir_active;
286     button_len_up_old <= button_len_up_active;
287     button_len_dn_old <= button_len_dn_active;
288 
289     button_freq_up_raise = 0;
290     button_freq_dn_raise = 0;
291     button_dir_raise = 0;
292     button_len_up_raise = 0;
293     button_len_dn_raise = 0;
294 end
295 
296 endmodule
297 
```

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.