

Laboratorium 4

Układ FPGA

Łukasz Kwinta, Kacper Kozubowski, Ida Ciepiela

maj 2024

Spis treści

1	Cel zadania	3
2	Czym są układy FPGA?	3
3	Realizacja	4
4	Rozwiązanie	5
4.1	Moduł dzielnika częstotliwości	5
4.2	Moduł filtrujący przyciski	6
4.3	Moduł sterujący wyświetlaczami	8
4.4	Właściwy moduł generujący animację	9
4.5	Plik ograniczeń fizycznych	15
5	Zastosowania układów FPGA	16
6	Wnioski	17

1 Cel zadania

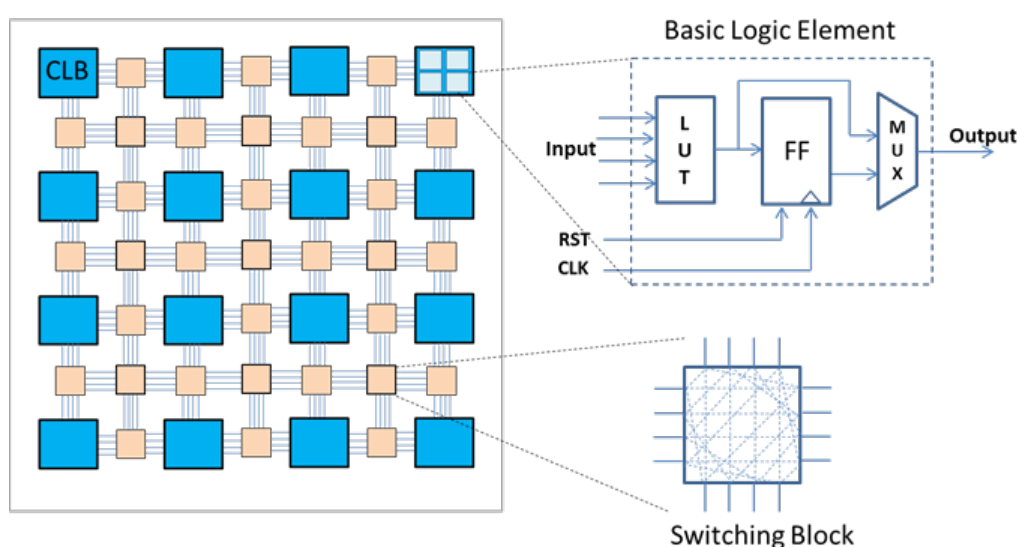
Celem laboratorium było zaprogramowanie układu FPGA tak aby wyświetlał animację poruszających się segmentów na obrzeżach wyświetlaczy 7 segmentowych. Należało również, zaimplementować prostą funkcjonalność obejmującą zmianę prędkości i kierunku ruchu świetlnego odcinka, przy pomocy znajdujących się na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

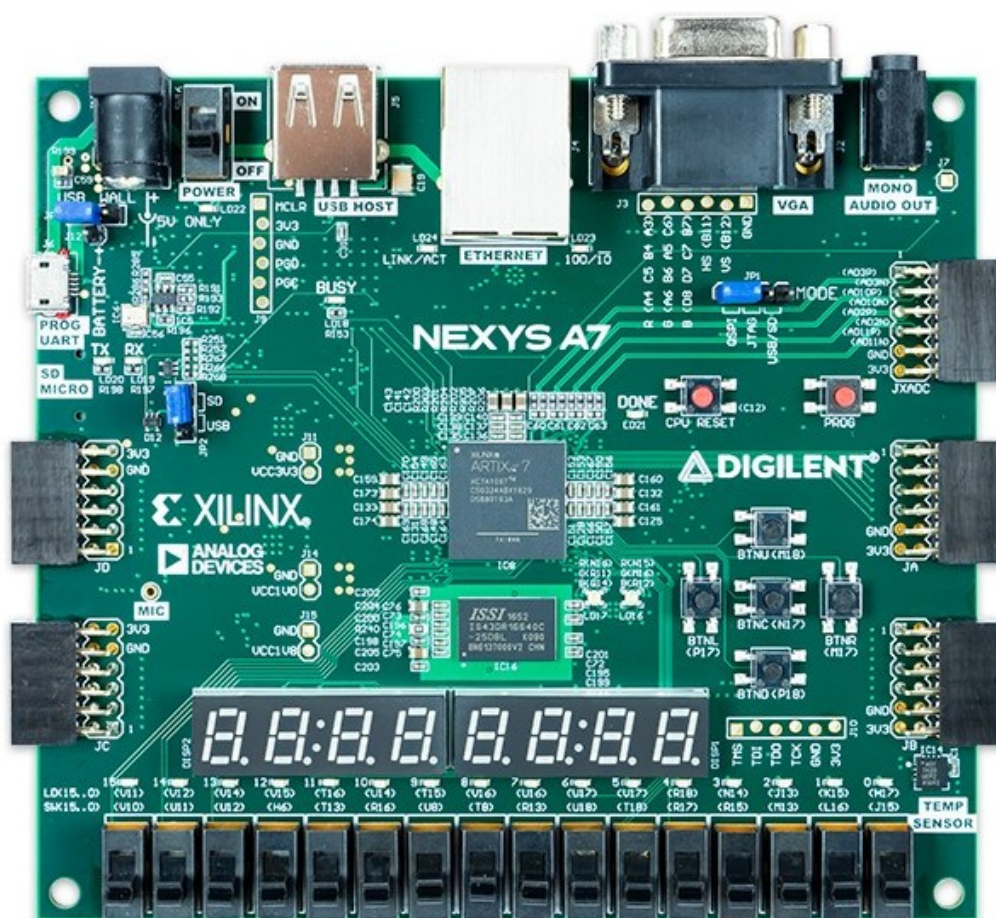
Dzięki takiej konstrukcji układy FPGA można dostosować do różnych zastosowań, co czyni je niezwykle wszechstronnymi w inżynierii cyfrowej.



Rysunek 2.1: Przykładowy schemat układu FPGA

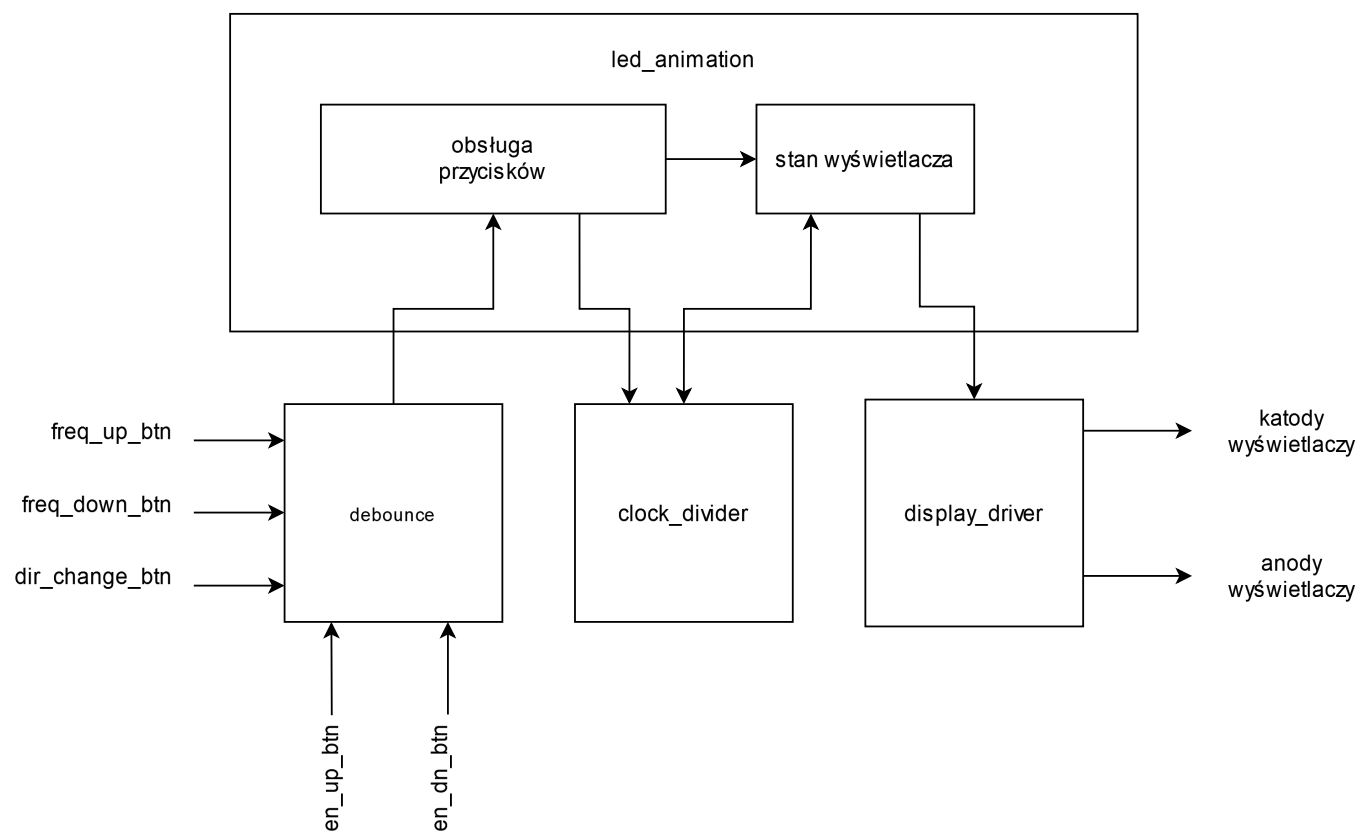
3 Realizacja

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 płytce 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

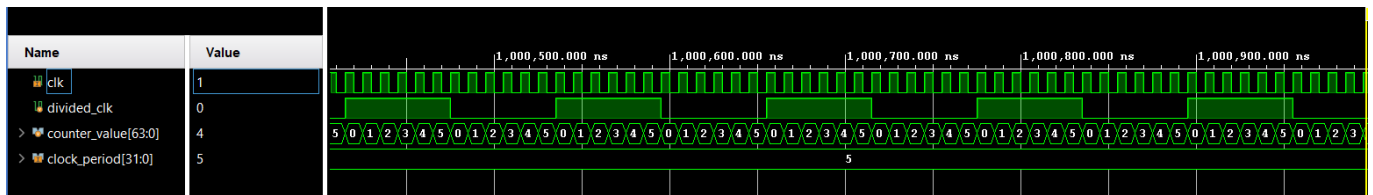
```

```

15 // zliczamy zadany okres zegara (ilość cykli zegara wejściowego), i gdy
16 // doliczymy do końca zmieniamy stan spowolnionego zegara na przeciwny
17 always@ (posedge clk)
18 begin
19     if (counter_value >= clock_period)
20     begin
21         divided_clock <= ~divided_clock;
22         counter_value <= 0;
23     end
24     else
25     begin
26         divided_clock <= divided_clock;
27         counter_value <= counter_value + 1;
28     end
29 end
30
31 endmodule

```

Poniżej przedstawiamy przykładową symulację działania tego modułu. Okres zegara został ustawiony na wartość 5 co oznacza, że zegar wyjściowy jest 5 razy wolniejszy od zegara systemowego



Rysunek 4.2: Symulacja działania modułu clock_divider

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.

```

1 // moduł filtrujący przyciski
2 module debounce #(parameter DEBOUNCE_TIME = 1000 * 100) (
3     input wire clk,
4     input wire button_physical,
5
6     output reg button_active
7 );
8
9 // ustawiamy początkowy stan przycisku na 0
10 initial
11     button_active = 0;

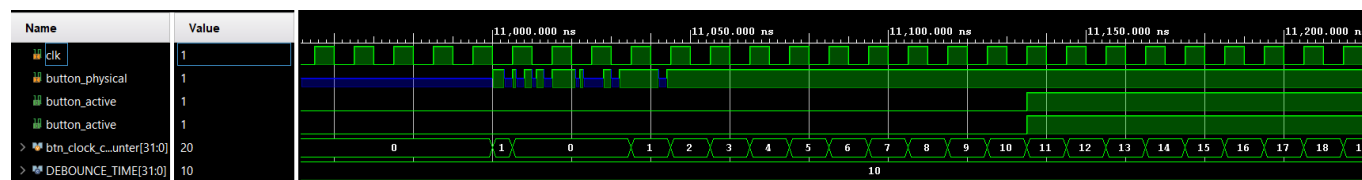
```

```

12
13 integer btn_clock_cycles_counter = 0;
14
15 // zliczamy zadaną ilość cykli zegara
16 // jeśli w którymś cyklu przycisk będzie w stanie niskim,
17 // resetujemy licznik wartości
18 always@ (posedge clk)
19 begin
20     if (button_physical == 1)
21     begin
22         btn_clock_cycles_counter <= btn_clock_cycles_counter + 1;
23         if (btn_clock_cycles_counter >= DEBOUNCE_TIME)
24             button_active <= 1;
25     end
26 else
27     begin
28         btn_clock_cycles_counter <= 0;
29         button_active <= 0;
30     end
31 end
32
33
34 endmodule

```

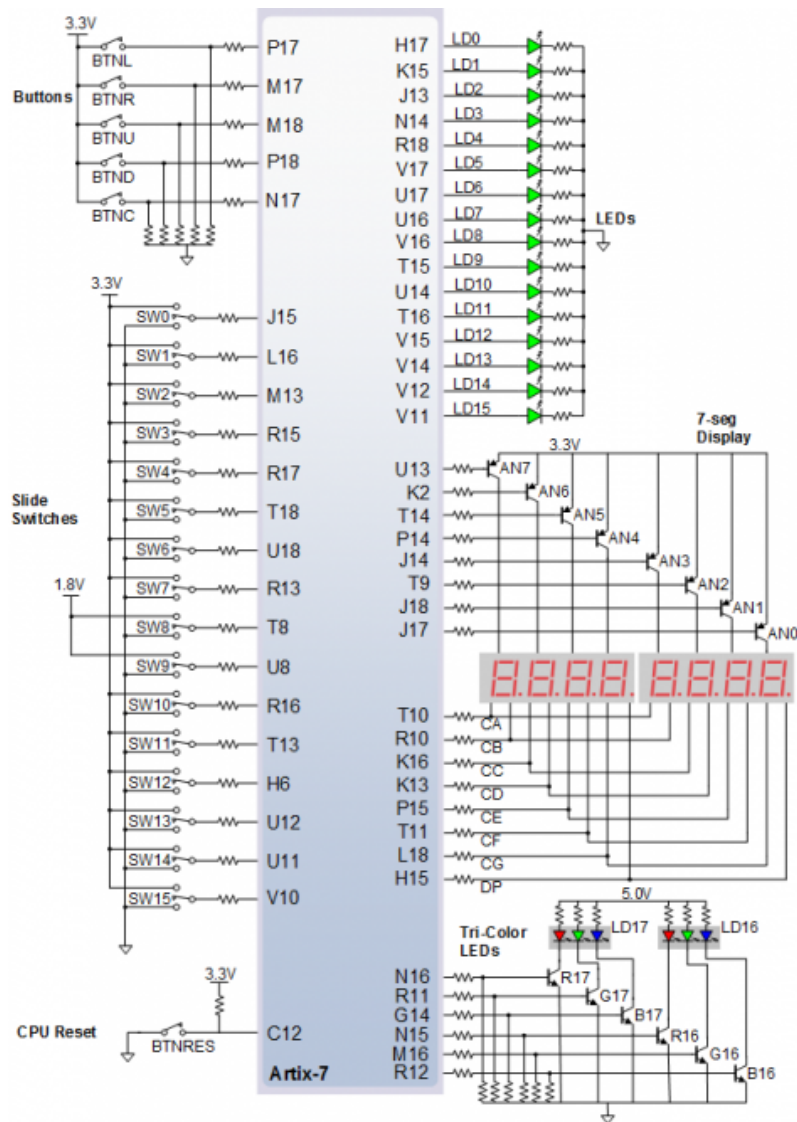
Poniżej przedstawiamy przykładową symulację działania tego modułu. Ilość wymaganych cykli zegara w których przycisk musi być w stanie wysokim została ustawiona na niewielką wartość 10 aby dobrze zilustrować działanie modułu. W rzeczywistości wartość ta musi być znacznie większa. Zasymulowany został również sztuczny szum jaki powstałby przy naciśnięciu fizycznego przycisku.



Rysunek 4.3: Symulacja działania modułu debounce

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 4.4: 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     // wyjścia sterujące wyświetlaczami
8     // stan niski na danym indeksie aktywuje dany wyświetlacz
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 wyświetlaczach
14     output reg [7:0] sseg_cathodes
15 );
16
17 initial
18 begin
19     sseg_anodes <= '1;
20     sseg_cathodes <= '1;
21 end
22
23 reg refresh_clk; //1 khz refresh clock
24 clock_divider clk_div (
25     .clock_period(REFERESH_PERIOD),
26     .clk(clk),
27     .divided_clock(refresh_clk)
28 );
29
30 reg [3:0] display_number = 0;
31
32 always@ (posedge refresh_clk)
33 begin
34     sseg_anodes = '1;
35     sseg_anodes[display_number] <= 0;
36     sseg_cathodes <= 8'(~display[display_number]);
37
38     display_number = display_number + 1;
39     if (display_number >= `DISPLAY_COUNT)
40         display_number <= 0;
41 end
42
43 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.

```

1  `timescale 1ns / 1ps // tylko dla symulacji
2
3  // Przydatne makra dla wyświetlaczow
4  `define SEGMENT_A_MASK 8'(1 << 0)
5  `define SEGMENT_B_MASK 8'(1 << 1)
6  `define SEGMENT_C_MASK 8'(1 << 2)
7  `define SEGMENT_D_MASK 8'(1 << 3)
8  `define SEGMENT_E_MASK 8'(1 << 4)
9  `define SEGMENT_F_MASK 8'(1 << 5)
10 `define SEGMENT_G_MASK 8'(1 << 6)
11 `define SEGMENT_DOT_MASK 8'(1 << 7)
12 `define DISPLAY_CLEAR '0
13 `define DISPLAY_ALL '1
14
15 `define DISPLAY_REFRESH_FREQUENCY (100 * 1000 / 8) // częstotliwość odświeżania ekranów = 0,125kHz
16 `define START_ANIMATION_PERIOD (1000*100*1000 / 4) // okres startowy = 0,25s
17 `define MAX_ANIMATION_PERIOD 1000*100*1000*100 // maks okres = 100s
18 `define MIN_ANIMATION_PERIOD 1
19
20 `define DISPLAY_COUNT 8 // liczba wykorzystywanych wyświetlaczy
21
22 `define MAX_SNAKE_LENGTH (`DISPLAY_COUNT*2 + 3)
23 `define START_SNAKE_LENGTH 1
24 `define MIN_SNAKE_LENGTH 1
25
26
27 /* Definicja modułu z wejściami i wyjściami zdefiniowanymi w pliku .xdc */
28 module led_animation(
29     // wejścia przycisków
30     input wire btn_freq_up,
31     input wire btn_freq_dn,
32     input wire btn_dir,
33     input wire len_up,
34     input wire len_dn,
35
36     input wire clk, // zegar systemowy 100Mhz
37
38     // wyjścia sterujące wyświetlaczami
39     // stan niski na danym indeksie aktywuje dany wyświetlacz
40     output reg [7:0] sseg_anodes,
41
42     // | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
43     // | DP | CG | CF | CE | CD | CC | CB | CA |
44     // stan niski na danym indeksie aktywuje dany segment na wszystkich aktywnych wyświetlaczach
45     output reg [7:0] sseg_cathodes,
46
47     // wyjście sterujące ledem
48     output wire clk_led
49 );
50
51 initial
52 begin
53     sseg_cathodes <= '1;
54     sseg_anodes <= '1;
55 end
56
57 // startowy okres spowolnionego zegara
58 longint clock_period = `START_ANIMATION_PERIOD;
59
60 // użycie modułu spowalniającego zegar

```

```

61  reg divided_clk;
62  assign clk_led = divided_clk;
63  clock_divider clk_div (
64      .clock_period(clock_period),
65      .clk(clk),
66      .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)
128             tail = 0;
129         display[snake[tail][0]] = display[snake[tail][0]] - 2**(snake[tail][1]);
130     end
131 else if (old_length < length)
132     begin
133         tail = tail - 1;
134         old_length = old_length + 1;
135         if (tail < 0)
136             tail = num_of_segments - 1;
137     end
138
139 else if (snake[tail][0] != -1) // gasimy ogon
140     begin
141         display[snake[tail][0]] = display[snake[tail][0]] - 2**(snake[tail][1]);
142     end
143
144 // uaktualnienie wskaźników
145 tail = tail + 1;
146 head = head + 1;
147 if (head == num_of_segments)
148     head = 0;
149 if (tail == num_of_segments)
150     tail = 0;
151
152 snake[head][0] = curr_display;
153
154 if (curr_display == 0) // obsługa prawego wyświetlacza
155     begin
156         snake[head][1] = curr_state;
157         display[0] = display[0] + 2**curr_state;
158
159         if ((dir == LEFT && curr_state == 0) || (dir == RIGHT && curr_state == 3))
160             curr_display = curr_display + 1;
161     end
162
163 else if (curr_display == (`DISPLAY_COUNT - 1)) // obsługa lewego wyświetlacza
164     begin
165         snake[head][1] = (curr_state - (`DISPLAY_COUNT - 1) < 6 ? curr_state - (`DISPLAY_COUNT - 1) : 0);
166         display[`DISPLAY_COUNT - 1] = display[`DISPLAY_COUNT - 1] + 2**(curr_state - (`DISPLAY_COUNT - 1) < 6
167                                     ? curr_state - (`DISPLAY_COUNT - 1) : 0);
168
169         if ((dir == LEFT && curr_state == (`DISPLAY_COUNT + 2))
170             || (dir == RIGHT && curr_state == (`DISPLAY_COUNT + 5)))
171             curr_display = curr_display - 1;
172     end
173
174 else if (curr_state > 3 && curr_state < (`DISPLAY_COUNT + 2)) // obsługa dolnych segmentów
175     begin
176         snake[head][1] = 3;
177         display[curr_display] = display[curr_display] + 2**3;
178
179         if (dir == RIGHT)
180             curr_display = (curr_display + 1) & (`DISPLAY_COUNT - 1);
181
182     else

```

```

183         curr_display = (curr_display - 1) & (`DISPLAY_COUNT - 1);
184     end
185     else // obsługa górnych segmentów
186     begin
187         snake[head][1] = 0;
188         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
196     end
197 end
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 przycisku poprzez porównanie starej wartości 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

```

4.5 Plik ograniczeń fizycznych

Aby przygotowany przez nas program działał poprawnie, musieliśmy przypisać wejścia i wyjścia do określonych fizycznych lokalizacji na wykorzystywanym układzie FPGA. W tym celu utworzyliśmy plik zawierający wszystkie konieczne mapowania. Kody potrzebnych nam pinów pozyskaliśmy z instrukcji zapewnionej przez producenta płytki Nexys-a7.

```

1  ###
2  ### Plik zawierający informacje wiążące fizyczne wyprowadzenia układu
3  ### z nazwami używanymi w kodzie
4  ###
5
6  ## Przypisanie zegara systemowego o częstotliwości 100Mhz (okres 10.00 ns)
7  set_property -dict { PACKAGE_PIN E3      IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
8  create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports { clk }];
9
10 ## Przypisanie przycisków których będziemy używać
11 set_property -dict { PACKAGE_PIN M18      IOSTANDARD LVCMOS33 } [get_ports { btn_freq_up }]; #IO_L4N_T0_D05_14 Sch=btnc
12 set_property -dict { PACKAGE_PIN P18      IOSTANDARD LVCMOS33 } [get_ports { btn_freq_dn }]; #IO_L9N_T1_DQS_D13_14 Sch=btnd
13 set_property -dict { PACKAGE_PIN N17      IOSTANDARD LVCMOS33 } [get_ports { btn_dir }]; #IO_L9P_T1_DQS_14 Sch=btnc
14 set_property -dict { PACKAGE_PIN M17      IOSTANDARD LVCMOS33 } [get_ports { len_up }]; #IO_L9P_T1_DQS_14 Sch=btnc
15 set_property -dict { PACKAGE_PIN P17      IOSTANDARD LVCMOS33 } [get_ports { len_dn }]; #IO_L9P_T1_DQS_14 Sch=btnc
16
17 ## Przypisanie wyprowadzeń do wyświetlacza 7 segmentowego - jako, że segmenty są aktywne w stanie niskim
18 ## musimyysterować wszystkie nawet nieużywane segmenty aby ustawić stan wysoki na segmentach które nie powinny
19 ## się świecić
20
21 # Segment A
22 set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { sseg_cathodes[0] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
23 # Segment B
24 set_property -dict { PACKAGE_PIN R10 IOSTANDARD LVCMOS33 } [get_ports { sseg_cathodes[1] }]; #IO_25_14 Sch=cb
25 # Segment C
26 set_property -dict { PACKAGE_PIN K16 IOSTANDARD LVCMOS33 } [get_ports { sseg_cathodes[2] }]; #IO_25_15 Sch=cc
27 # Segment D
28 set_property -dict { PACKAGE_PIN K13 IOSTANDARD LVCMOS33 } [get_ports { sseg_cathodes[3] }]; #IO_L17P_T2_A26_15 Sch=cd
29 # Segment E
30 set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMOS33 } [get_ports { sseg_cathodes[4] }]; #IO_L13P_T2_MRCC_14 Sch=ce
31 # Segment F
32 set_property -dict { PACKAGE_PIN T11 IOSTANDARD LVCMOS33 } [get_ports { sseg_cathodes[5] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
33 # Segment G
34 set_property -dict { PACKAGE_PIN L18 IOSTANDARD LVCMOS33 } [get_ports { sseg_cathodes[6] }]; #IO_L4P_T0_D04_14 Sch=cg
35 # Kropka
36 set_property -dict { PACKAGE_PIN H15 IOSTANDARD LVCMOS33 } [get_ports { sseg_cathodes[7] }]; #IO_L19N_T3_A21_VREF_15 Sch=dp
37
38 # Aktywacja 1 wyświetlacz 1 (1 od prawej)
39 set_property -dict { PACKAGE_PIN J17 IOSTANDARD LVCMOS33 } [get_ports { sseg_anodes[0] }]; #IO_L23P_T3_F0E_B_15 Sch=an[0]
40 # Aktywacja 2 wyświetlacza
41 set_property -dict { PACKAGE_PIN J18 IOSTANDARD LVCMOS33 } [get_ports { sseg_anodes[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
42 # Aktywacja 3 wyświetlacza
43 set_property -dict { PACKAGE_PIN T9 IOSTANDARD LVCMOS33 } [get_ports { sseg_anodes[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
44 # Aktywacja 4 wyświetlacza
45 set_property -dict { PACKAGE_PIN J14 IOSTANDARD LVCMOS33 } [get_ports { sseg_anodes[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
46 # Aktywacja 5 wyświetlacza
47 set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVCMOS33 } [get_ports { sseg_anodes[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
48 # Aktywacja 6 wyświetlacza
49 set_property -dict { PACKAGE_PIN T14 IOSTANDARD LVCMOS33 } [get_ports { sseg_anodes[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]

```

```
50 # Aktywacja 7 wyświetlacza
51 set_property -dict { PACKAGE_PIN K2 IOSTANDARD LVCMOS33 } [get_ports { sseg_anodes[6] }]; #IO_L23P_T3_35 Sch=an[6]
52 # Aktywacja 8 wyświetlacza
53 set_property -dict { PACKAGE_PIN U13 IOSTANDARD LVCMOS33 } [get_ports { sseg_anodes[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]
54
55 ## Przypisanie leda wizualizującego obecną częstotliwość spowolnionego zegara
56 set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { clk_led }]; #IO_L18P_T2_A24_15 Sch=led[0]
57
```

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 technologii.

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.