

Technika Cyfrowa

Sprawozdanie - FPGA

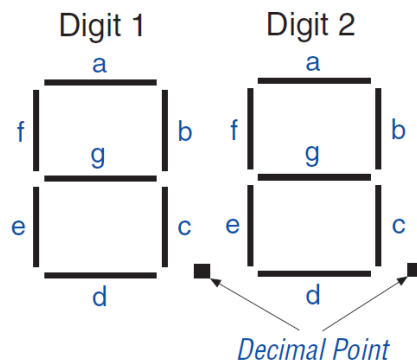
Maciej Trątnowiecki

AGH, Semestr Letni, 2020

1 Opis zrealizowanego projektu

W ramach zadania praktycznego przygotowałem implementację algorytmu wyświetlającego na dwóch wyświetlaczach siedmiosegmentowych animację węża. Algorytm zaimplementowałem w dwóch wersjach - w języku SystemVerilog i VHDL dla płytki rozwojowej Altera UP2 wyposażonej w układ FPGA FLEX 10KE EPF10K70RC240-4. Wykorzystałem środowisko Quartus II 9.0SP2, w którym przeprowadziłem symulację działania programu.

Sekwencję stanów wyjść wyświetlacza odpowiadającą za wyświetlenie zaplanowanej animacji wyznaczyłem na podstawie dokumentacji zestawu edukacyjnego Altera UP2 dostępnej pod adresem <http://eelinux.ee.usm.maine.edu/courses/ele373/upds.pdf>. Strony opisujące rozmieszczenie segmentów wyświetlacza led na płytce, oraz przypisane im piny układu FPGA zostały dołączone na końcu poniższego sprawozdania.



Poniżej zamieszczam przyjętą sekwencję stanów wysokich pozwalających uzyskać zaplanowaną animację.

1. Digit1 G; Digit1 E; Digit1 D
2. Digit1 E; Digit1 D; Digit2 D
3. Digit1 D; Digit2 D; Digit2 C
4. Digit2 D; Digit2 C; Digit2 G
5. Digit2 C; Digit2 G; Digit1 G
6. Digit2 G; Digit1 G; Digit1 F
7. Digit1 G; Digit1 F; Digit1 A
8. Digit1 F; Digit1 A; Digit2 A
9. Digit1 A; Digit2 A; Digit2 B
10. Digit2 A; Digit2 B; Digit2 G
11. Digit2 B; Digit2 G; Digit1 G
12. Digit2 G; Digit1 G; Digit1 E
13. Digit1 G; Digit1 E; Digit1 D

Stan opisany numerem 13 jest tożsamy ze stanem początkowym układu. Animacja ma zatem postać dwunastu ustawień wyświetlacza wyświetlanych w cyklu.

2 Implementacja w języku SystemVerilog

Jako wejście układu przyjęty został sygnał zegarowy (oznaczony clk). Jako wyjście układu zadeklarowano dwie jednowymiarowe macierze typu reg, każda o 7 wyjściach, które następnie zostaną zmapowane do odpowiednich pinów wyświetlacza. Algorytm definiuje też dwie stałe parametryczne znane na etapie kompilacji. Pierwszą z nich jest ITER_N będącą stałą wewnętrzną układu wskazującą ilość przyjętych stanów animacji. Drugą jest TICKS_PER_PHASE określającą co ile taktów zegara nastąpić ma zmiana stanu animacji na kolejny. W programie wykorzystano dwie zmienne lokalne przechowujące liczby naturalne.

Ustawione zostało sterowanie zdarzeniowe uruchamiające poniższą logikę za każdym rosnącym zboczem sygnału zegarowego. Jeśli zmienna lokalna odpowiedzialna za zliczanie zaobserwowanych taktów zegara wskazuje na zero, na wyświetlaczu wyświetlona zostaje sekwencja odpowiadająca aktualnemu stanowi symulacji. Następuje inkrementacja zmiennej wskazującej następny stan symulacji. Jeśli zmienna ta wskazuje na wartość równą stałej ITER_N, zostaje ona wyzerowana. Z każdym taktem procesora następuje inkrementacja zmiennej zliczającej takty. Jeśli zrówna się ona co do wartości ze stałą TICKS_PER_PHASE, zostaje wyzerowana.

Za wyświetlenie odpowiedniego stanu animacji na wyświetlaczu odpowiedzialne są dwie instrukcje warunkowe *case*. Wartości zmiennych typu reg modyfikowane są poprzez przypisanie odpowiedniej liczby siedmiobitowej zapisanej binarnie.

Mechanizm zliczania taktów zegara został zaimplementowany w celu spowolnienia animacji. Jej prędkość zależy od częstotliwości sygnału generowanego przez wbudowany w płytke Altera UP2 układ zegarowy. Zbyt duża prędkość skutkować będzie niskimi walorami estetycznymi. Wielokrotnością spowolnienia sterować można za pomocą stałej parametrycznej TICKS_PER_PHASE. Może ona przyjąć wartość dowolnej liczby naturalnej z przedziału $< 1, \infty$.

Poniżej zamieszczam kod algorytmu zapisanego w języku SystemVerilog.

```
1000 module snake (
1001     input  wire  clk ,
1002     output reg [0:6] display1 ,
1003     output reg [0:6] display2
1004 );
1005 //parameter TICKS_PER_PHASE = 2717500;
1006 parameter TICKS_PER_PHASE = 10;
1007 parameter ITER_N = 12;
1008 int iteration = 0;
1009 int ticks_count = 0;
1010
1011 always_ff @ (posedge clk)
1012 begin
1013     if (ticks_count == 0)
1014     begin
1015         case (iteration)
1016             0: display1 = 7'b0001101;
1017             1: display1 = 7'b0001100;
1018             2: display1 = 7'b0001000;
1019             3: display1 = 7'b0000000;
1020             4: display1 = 7'b0000001;
1021             5: display1 = 7'b0000011;
1022             6: display1 = 7'b1000011;
1023             7: display1 = 7'b1000010;
1024             8: display1 = 7'b1000000;
1025             9: display1 = 7'b0000000;
1026             10: display1 = 7'b0000001;
1027             11: display1 = 7'b0000101;
1028             default : display1 = 7'b1111111;
1029         endcase
1030
1031         case (iteration)
1032             0: display2 = 7'b0000000;
1033             1: display2 = 7'b0001000;
1034             2: display2 = 7'b0011000;
1035             3: display2 = 7'b0011001;
1036             4: display2 = 7'b0010001;
1037             5: display2 = 7'b0000001;
1038             6: display2 = 7'b0000000;
1039             7: display2 = 7'b1000000;
1040             8: display2 = 7'b1100000;
1041             9: display2 = 7'b1100001;
```

```

1042     10: display2 = 7'b0100001;
1043     11: display2 = 7'b0000001;
1044     default : display2 = 7'b1111111;
1045 endcase
1046
1047     iteration = iteration + 1;
1048     if (iteration == ITER_N)
1049         iteration = 0;
1050 end
1051     ticks_count = ticks_count + 1;
1052     if (ticks_count == TICKS_PER_PHASE)
1053         ticks_count = 0;
1054 end
1055 endmodule

```

../simulations/FPGA/snake_verilog/snake.sv

Poniżej zamieszczam raport z kompilacji programu.

Flow Status	Successful - Sat May 30 22:08:14 2020
Quartus II Version	9.0 Build 235 06/17/2009 SP 2 SJ Web Edition
Revision Name	snake
Top-level Entity Name	snake
Family	FLEX10K
Device	EPF10K70RC240-4
Timing Models	Final
Met timing requirements	Yes
Total logic elements	197 / 3,744 (5 %)
Total pins	15 / 189 (8 %)
Total memory bits	0 / 18,432 (0 %)

3 Implementacja w języku VHDL

Jako wejście układu przyjęty został sygnał zegarowy (oznaczony *clk*). Jako wyjście układu zadeklarowano dwa wektory wartości logicznych o siedmiu elementach, które następnie zostaną zmapowane do odpowiednich pinów wyświetlacza. Program definiuje też dwie stałe parametryczne znane na etapie kompilacji. Pierwszą z nich jest *ITER_N* będącą stałą wewnętrzną układu wskazującą ilość przyjętych stanów animacji. Drugą jest *TICKS_PER_PHASE* określająca co ile taktów zegara nastąpić ma zmiana stanu animacji na kolejny. Może ona przyjmować wartości postaci liczb naturalnych większych lub równych jeden. W programie wykorzystano dwie zmienne lokalne przechowujące liczby naturalne.

Program korzysta z biblioteki IEEE, importując moduły odpowiedzialne za standardową logikę i operacje numeryczne. Definiuje także element *snake*, oraz związaną z nim architekturę *arch1*. Wspomniana architektura definiuje procedurę *display_phase* przyjmującą jako argument liczbę naturalną. Odpowiedzialna jest ona za wyświetlenie na wyświetlaczu stanu animacji związanego z liczbą przyjętą jako argument. Wykorzystuje do tego instrukcję warunkową *case*, za pomocą której wypełnia wektory odpowiednimi stanami wyjść. Warto zauważyć, że wektory indeksowane są od prawej strony. Stanowi to pewien kontrast do poprzedniego programu, w którym wartości w jednowymiarowej macierzy indeksowane były od lewej strony. Z tego powodu w celu zachowania tej samej kolejności pinów na wyjściu (tzn. kompatybilności z tym samym mapowaniem do pinów wyświetlacza) konieczne jest odbicie lustrzane przypisywanych w wektorach wartości.

W architekturze *arch1* zarejestrowany jest proces *tick* uruchamiany sygnałem *clk*. Z każdym rosnącym zboczem sygnału zegarowego, przeprowadzana jest analiza logiczna analogiczna do tej zapisanej w języku SystemVerilog. Jeśli licznik taktów wskazuje zero, następuje zmiana stanu animacji za pomocą wcześniej zdefiniowanej procedury, oraz inkrementacja zmiennej identyfikującej następny stan animacji. Jest ona zerowana, gdy zrówna się co do wartości ze stałą *ITER_N*. Z każdym taktem zegara następuje inkrementacja licznika taktów, który jest zerowany gdy zrówna się co do wartości ze stałą *TICKS_PER_PHASE*.

Algorytm implementuje opóźnienie animacji względem zegara w sposób analogiczny do opisanego w poprzednim punkcie.

Poniżej zamieszczam kod algorytmu zapisanego w języku VHDL.

```

1000 library IEEE;
1001 use IEEE.STD_LOGIC_1164.ALL;
1002 use IEEE.NUMERIC_STD.ALL;

1004 entity snake is
1005     port(
1006         clk:        in  std_logic;

1008         display1:   out std_logic_vector(6 downto 0) := "1111111";
1009         display2:   out std_logic_vector(6 downto 0) := "1111111"
1010     );
1011 end snake;

1012 architecture arch1 of snake is
1013     procedure display_phase(phase : in natural) is
1014     begin
1015         case phase is
1016             -- SEG NUMBERS          6543210
1017             when 0 => display1 <= "1011000";
1018             when 1 => display1 <= "0011000";
1019             when 2 => display1 <= "0001000";
1020             when 3 => display1 <= "0000000";
1021             when 4 => display1 <= "1000000";
1022             when 5 => display1 <= "1100000";
1023             when 6 => display1 <= "1100001";
1024             when 7 => display1 <= "0100001";
1025             when 8 => display1 <= "0000001";
1026             when 9 => display1 <= "0000000";
1027             when 10 => display1 <= "1000000";
1028             when 11 => display1 <= "1010000";
1029             when others => display1 <= "1111111";
1030         end case;
1031         case phase is
1032             -- SEG NUMBERS          6543210
1033             when 0 => display2 <= "0000000";
1034             when 1 => display2 <= "0001000";
1035             when 2 => display2 <= "0001100";
1036             when 3 => display2 <= "1001100";
1037             when 4 => display2 <= "1000100";
1038             when 5 => display2 <= "1000000";
1039             when 6 => display2 <= "0000000";
1040             when 7 => display2 <= "0000001";
1041             when 8 => display2 <= "0000011";
1042             when 9 => display2 <= "1000011";
1043             when 10 => display2 <= "1000010";
1044             when 11 => display2 <= "1000000";
1045             when others => display2 <= "1111111";
1046         end case;
1047     end procedure display_phase;

1048 begin
1049     tick : process(clk) is
1050     constant ITER_N : natural := 12;
1051     variable iteration : natural := 0;

1052     --constant TICKS_PER_PHASE : natural := 2717500;
1053     constant TICKS_PER_PHASE : natural := 10;
1054     variable ticks_count : natural := 0;
1055     begin
1056         if rising_edge(clk) then
1057             if ticks_count = 0 then
1058                 display_phase(iteration);
1059                 iteration := iteration + 1;
1060                 if iteration = ITER_N then
1061                     iteration:= 0;
1062                 end if;
1063             end if;
1064             ticks_count := ticks_count + 1;
1065             if ticks_count = TICKS_PER_PHASE then
1066                 ticks_count := 0;
1067             end if;
1068         end if;
1069     end process;
1070 end arch1;

```

```

1074 | end process tick;
      | end arch1;

```

../simulations/FPGA/snake_vhdl/snake.vhd

Poniżej zamieszczam raport z kompilacji programu.

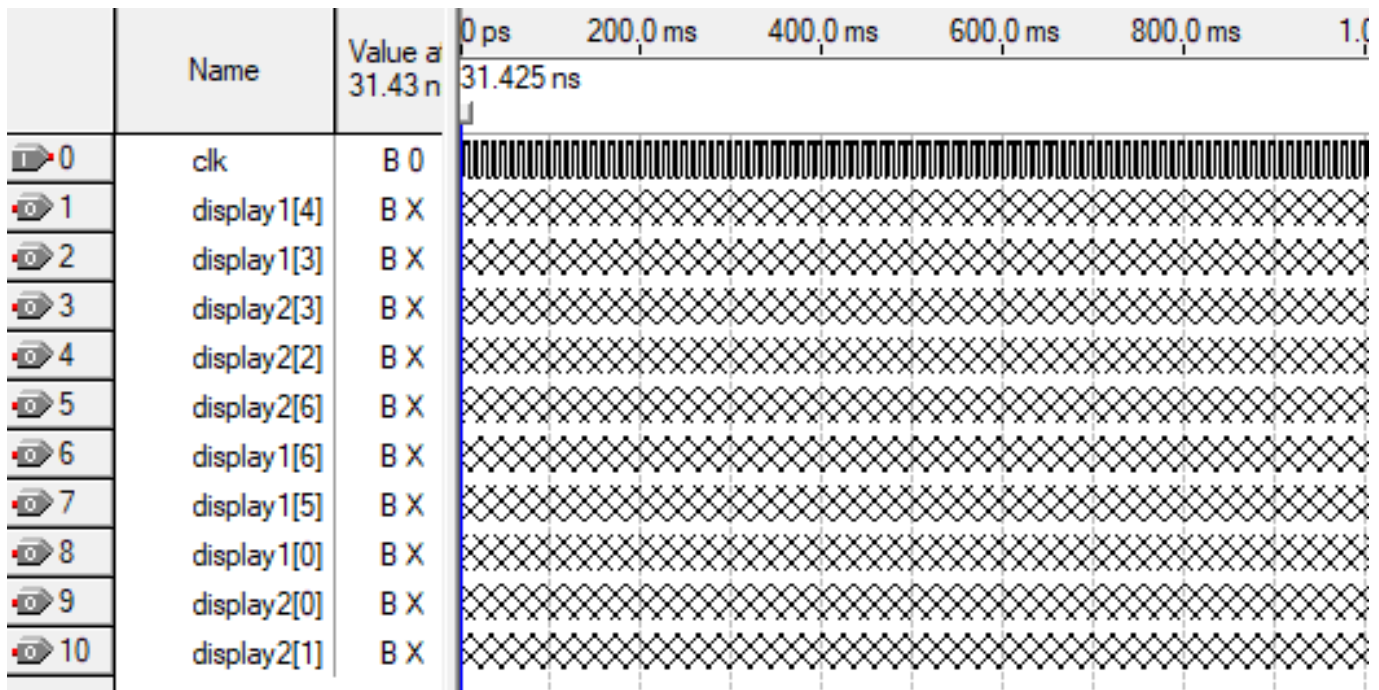
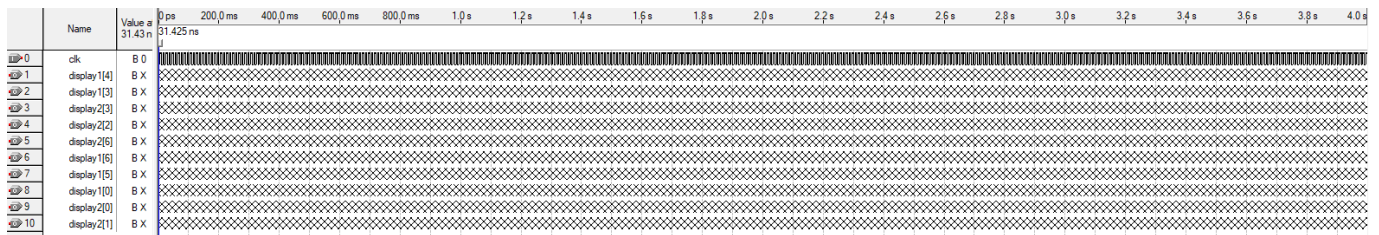
Flow Status	Successful - Sat May 30 22:07:01 2020
Quartus II Version	9.0 Build 235 06/17/2009 SP 2 SJ Web Edition
Revision Name	snake
Top-level Entity Name	snake
Family	FLEX10K
Device	EPF10K70RC240-4
Timing Models	Final
Met timing requirements	Yes
Total logic elements	192 / 3,744 (5 %)
Total pins	15 / 189 (8 %)
Total memory bits	0 / 18,432 (0 %)

4 Symulacja działania programu

Długość symulacji ustaliłem na cztery sekundy. Symulację oparłem o plik określający kształt fali. Z dokumentacji płytki rozwojowej Altera UP2 odczytać możemy częstotliwość sygnału generowanego przez zegar jako 25.175MHz . Oznacza, to że zegar generuje sygnał o okresie 39.7ns (jako że okres jest odwrotnością częstotliwości). Jeśli chcielibyśmy otrzymać animację z dziesięcioma zmianami stanu na sekundę, odpowiadałoby to ustawieniu stałej TICKS_PER_PHASE na wartość 2717500. W praktyce symulacja o takiej dokładności wymaga sporej mocy obliczeniowej, dlatego na jej potrzeby ustawiłem stałą na 10. Do pliku symulacji dodałem sygnał wejściowy układu zegara o okresie 10 ms. Takie uproszczenie symulacji nie ma wpływu na możliwość pokazania poprawności działania algorytmu.

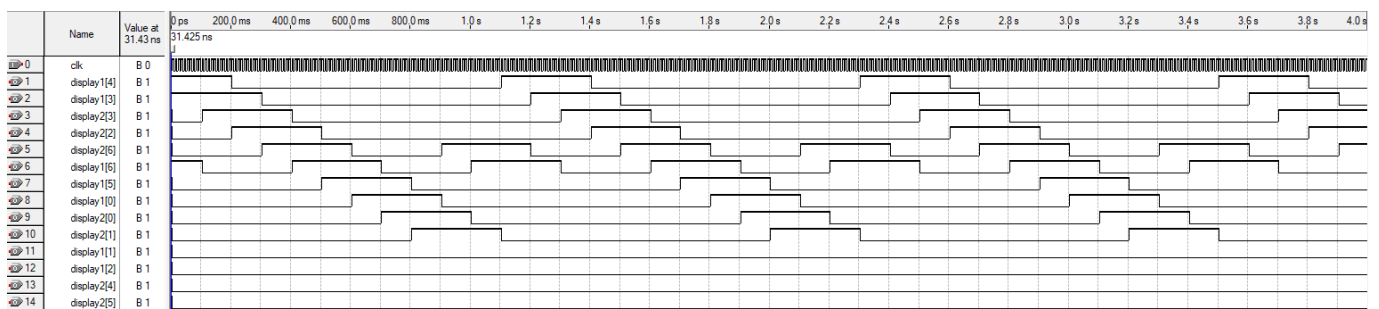
The image shows a 'Clock' configuration dialog box. It has two main sections: 'Time range' and 'Base waveform on'. In 'Time range', 'Start time' is set to 0 ps and 'End time' is set to 4.0 s. In 'Base waveform on', the 'Clock settings:' radio button is selected. Below it, there is a dropdown menu. The 'Time period:' radio button is also selected. Under 'Time period:', 'Period' is set to 10.0 ms, 'Offset' is set to 0.0 ms, and 'Duty cycle (%)' is set to 50. At the bottom are 'OK' and 'Cancel' buttons.

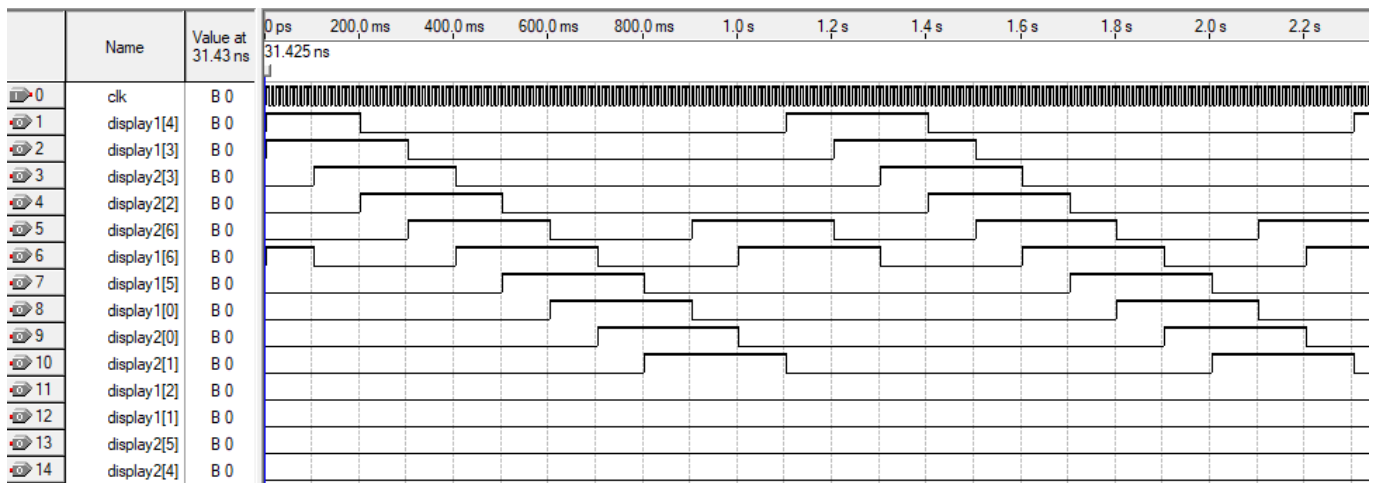
Przygotowany plik wejściowy symulacji ma postać.



Widzimy, że na każdy 100ms okres symulacji przypada 10 taktów procesora.

Tak przygotowaną symulację przeprowadziłem dla obu przygotowanych wcześniej programów. Otrzymałem pokrywające się sygnały wyjściowe. Ich przebieg ilustruje poniższy wykres.





Zauważmy, że otrzymany wykres ma przebieg cykliczny. Cztery złącza (display1[1], display1[2], display2[4], display2[5]) nie są włączane w żadnym momencie symulacji. Odpowiadają one za pionowe segmenty ze środka wyświetlacza po których wąż się nie porusza. Dwa złącza (display1[6] i display2[6]) występują częściej niż pozostałe. Odpowiadają one za poziome segmenty ze środka wyświetlacza. Każdy z segmentów zapalany jest zawsze na trzy kolejne stany animacji (odpowiada to długości węża).

Wiem, że:

Wyjścia display1[0 - 6] odpowiadają segmentom Digit1 A-G.

Wyjścia display2[0 - 6] odpowiadają segmentom Digit2 A-G.
















Tłumacząc opis sekwencji z pierwszej sekcji tego sprawozdania otrzymujemy:

1. display1 6; display1 4; display1 3
2. display1 4; display1 3; display2 3
3. display1 3; display2 3; display2 2
4. display2 3; display2 2; display2 6
5. display2 2; display2 6; display1 6
6. display2 6; display1 6; display1 5
7. display1 6; display1 5; display1 0
8. display1 5; display1 0; display2 0
9. display1 0; display2 0; display2 1
10. display2 0; display2 1; display2 6
11. display2 1; display2 6; display1 6
12. display2 6; display1 6; display1 4
13. display1 6; display1 4; display1 3

Porównując zapisane stany animacji z otrzymanymi wynikami symulacji stwierdzić możemy poprawność zaproponowanych algorytmów.

5 Przydział pinów

W środowisku Quartus II wybrałem układ FLEX 10KE EPF10K70RC240-4 jako urządzenie docelowe dla napisanego programu. Następnie korzystając z funkcji *Pin Planner*, oraz dokumentacji płytki Altera UP2 przygotowałem mapowanie portów algorytmu na odpowiednie piny układu. Pin zegarowy odczytałem z dokumentacji układu EPF10K70RC240-4 dostępnej pod adresem <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/dp/flex10k/archives/epf10k70.pdf>. Otrzymałem poniższy układ.

	Node Name	Direction	Location
	clk	Input	PIN_91
	display1[6]	Output	PIN_13
	display1[5]	Output	PIN_12
	display1[4]	Output	PIN_11
	display1[3]	Output	PIN_9
	display1[2]	Output	PIN_8
	display1[1]	Output	PIN_7
	display1[0]	Output	PIN_6
	display2[6]	Output	PIN_24
	display2[5]	Output	PIN_23
	display2[4]	Output	PIN_21
	display2[3]	Output	PIN_20
	display2[2]	Output	PIN_19
	display2[1]	Output	PIN_18
	display2[0]	Output	PIN_17

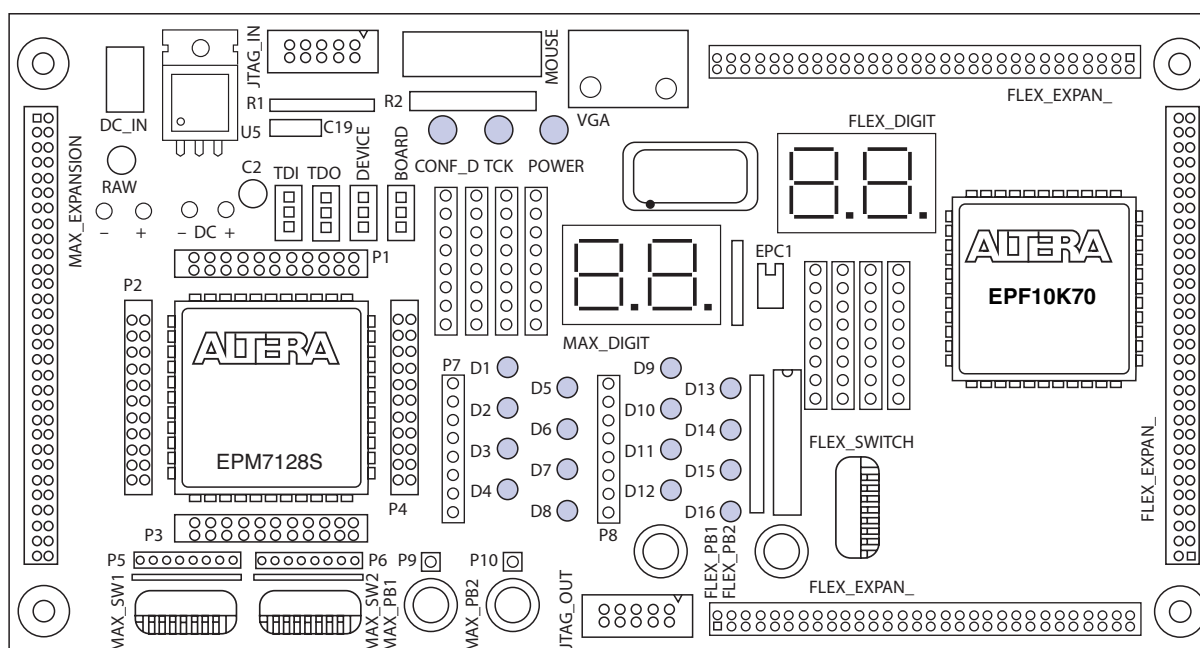
Tak przygotowane mapowanie pinów układu pozwoli wyświetlić animację na wyświetlaczu dwóch cyfr FLEX_DIGIT znajdującym się w prawym górnym rogu płytki rozwojowej.

Education Board. Because design changes are downloaded directly to the devices on the board, prototyping is easy and multiple design iterations can be accomplished in quick succession.

UP2 Education Board Description

The UP2 Education Board, shown in Figure 1, contains the features described in this section.

Figure 1. UP2 Education Board Block Diagram



DC_IN & RAW Power Input

The DC_IN power input accepts a 2.5-mm × 5.55-mm female connector. The acceptable DC input is 7 to 9 V at a minimum of 350 mA. The RAW power input consists of two holes for connecting an unregulated power source. The hole marked with a plus sign (+) is the positive input; the hole marked with a minus sign (–) is board-common.

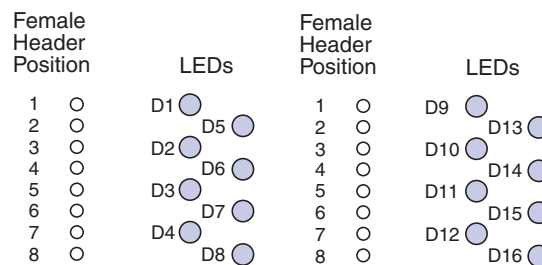
Oscillator

The UP2 Education Board contains a 25.175-MHz crystal oscillator. The output of the oscillator drives a global clock input on the EPM7128S device (pin 83) and a global clock input on the FLEX 10K device (pin 91).

D1 through D16 LEDs

The UP2 Education Board contains 16 LEDs that are pulled-up with a 330- Ω resistor. An LED is illuminated when a logic 0 is applied to the female header associated with the LED. LEDs D1 through D8 are connected in the same sequence to the female headers (i.e., D1 is connected to position 1, and D2 is connected to position 2, etc.). LEDs D9 through D16 are connected in the same sequence to the female headers (i.e., D9 is connected to position 1, and D10 is connected to position 2, etc.). See Figure 3.

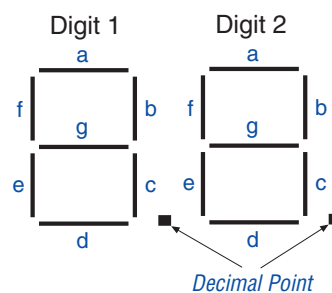
Figure 3. LED Positions



MAX_DIGIT Display

MAX_DIGIT is a dual-digit, seven-segment display connected directly to the EPM7128S device. Each LED segment of the display can be illuminated by driving the connected EPM7128S device I/O pin with a logic 0. Figure 4 shows the name of each segment.

Figure 4. Display Segment Name



FLEX 10K Device

The UP2 Education Board provides the following resources for the FLEX 10K device. The pins from the FLEX 10K device are pre-assigned to switches and LEDs on the board.

- JTAG chain connection for the ByteBlaster II cable
- Socket for an EPC1 configuration device
- Two momentary push button switches
- One octal DIP switch
- Dual-digit seven-segment display
- On-board oscillator (25.175 MHz)
- VGA port
- Mouse port
- Three expansion ports, each with 42 I/O pins and seven global pins

FLEX_PB1 & FLEX_PB2 Push Buttons

FLEX_PB1 and FLEX_PB2 are two push buttons that provide active-low signals to two general-purpose I/O pins on the FLEX 10K device. FLEX_PB1 connects to pin 28, and FLEX_PB2 connects to pin 29. Each push button is pulled-up through a 10-K Ω resistor.

FLEX_SW1 Switches

FLEX_SW1 contains eight switches that provide logic-level signals to eight general-purpose I/O pins on the FLEX 10K device. An input pin is set to logic 1 when the switch is open and set to logic 0 when the switch is closed. Table 6 lists the pin assignment for each switch.

Table 6. FLEX_SW1 Pin Assignments	
Switch	FLEX 10K Pin
FLEX_SWITCH-1	41
FLEX_SWITCH-2	40
FLEX_SWITCH-3	39
FLEX_SWITCH-4	38
FLEX_SWITCH-5	36
FLEX_SWITCH-6	35
FLEX_SWITCH-7	34
FLEX_SWITCH-8	33

FLEX_DIGIT Display

FLEX_DIGIT is a dual-digit, seven-segment display connected directly to the FLEX 10K device. Each LED segment on the display can be illuminated by driving the connected FLEX 10K device I/O pin with a logic 0. See Figure 4 on page 8 for the name of each segment. Table 7 lists the pin assignment for each segment.

Table 7. FLEX_DIGIT Segment I/O Connections		
Display Segment	Pin for Digit 1	Pin for Digit 2
a	6	17
b	7	18
c	8	19
d	9	20
e	11	21
f	12	23
g	13	24
Decimal point	14	25

VGA Interface

The VGA interface allows the FLEX 10K device to control an external video monitor. This interface is composed of a simple diode-resistor network and a 15-pin D-sub connector (labeled VGA), where the monitor can plug into the boards. The diode-resistor network and D-sub connector are designed to generate voltages that conform to the VGA standard.

Information about the color, row, and column indexing of the screen is sent from the FLEX 10K device to the monitor via five signals. Three VGA signals are red, green, and blue, while the other two signals are horizontal and vertical synchronization. Manipulating these signals allows images to be written to the monitor's screen.



See "VGA Driver Operation" on page 25 for details on how the VGA interface operates.

Table 8 lists the D-sub connector and the FLEX 10K device connections.