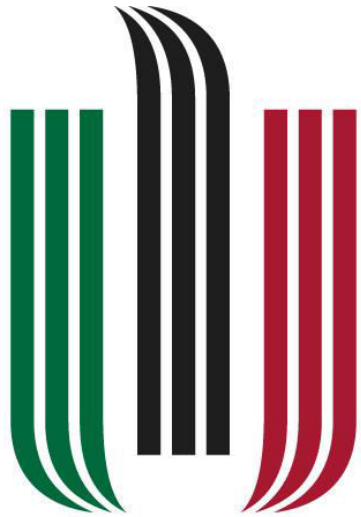


AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Dokumentacja projektu TRON

Złożone systemy cyfrowe

Krzysztof Blecharczyk, Hubert Romańczyk

Spis treści

Wprowadzenie

Cel projektu:

Celem projektu jest stworzenie na platformę FPGA gry podobnej do bardzo popularnych gier jakie można znaleźć w internecie, tj. **„Curve Fever”**. Plan pracy zakłada zgodność reguł stworzonej gry wraz z oryginalną produkcją.

WAŻNE: Projekt powstaje wyłącznie w celach edukacyjnych. Nigdy nie zostanie wykorzystany komercyjnie!

Podstawowe informacje o projekcie:

Zamysłem projektu jest usystematyzowanie oraz wykorzystanie umiejętności zdobytych na przedmiotach **„Podstawy elektroniki”**, oraz **„Technika cyfrowa”**, które należą do naszego toku studiów. Tematyka projektu została zaakceptowana przez prowadzącego zajęcia. Wybór nie jest przypadkowy - tworzenie gier komputerowych, poznanie ich mechaniki oraz możliwość dowolnej konfiguracji wszelkich opcji z nimi związanych od zawsze należało do obszaru zainteresowań osób biorących udział w projekcie. Ponadto jest to doskonała okazja, aby nauczyć się obsługi i programowania układów FPGA.

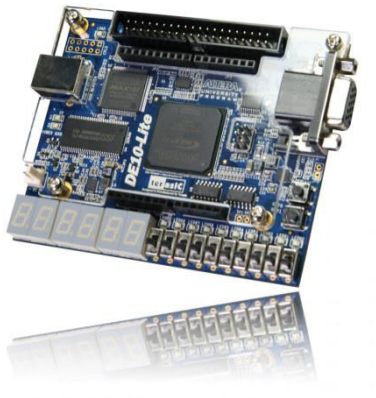
Sama nazwa projektu została zainspirowana filmem nakręconym w 1982 roku, o tytule „Tron”, reżyserii Steven Lisberger. Podobieństwo pomiędzy obiema produkcjami polega na śladzie, jaki zostawiają za sobą zarówno postacie w grze, jak i bohaterowie wyżej wspomnianego filmu.

Termin trwania projektu: 1.10.2019 r. - 16.02.2020 r.

Specyfikacja techniczna projektu:

Język w jakim program został napisany:	**SystemVerilog**
Układ cyfrowy wykorzystywany podczas prac nad projektem:	**Terasic DE10-Lite Board (P0466)**
Używane środowisko programistyczne:	**Quartus Prime Lite 19.1**

Specyfikacja układu cyfrowego:



Terasic DE10-Lite Board (P0466) to płytka deweloperska z układem FPGA Altera MAX 10.
Posiada ona:

- 50 tys. logicznych elementów programowalnych,
- 1638 Kbit pamięci M9K,
- 6 zintegrowanych układów ADC, z których każdy wspiera dokładnie jedno, dedykowane dla tego układu, wejście analogowe zgodne z **Arduino UNO R3**,
- 10 diod LED,
- 10 przełączników suwakowych,
- 2 przyciski,
- 6 wyświetlaczy, z których każdy jest 7-segmentowy
- wbudowany programator USB Blaster z gniazdem USB typu B,
- Ponadto wyjścia I/O:
    - 2x20 pinów GPIO
    - 4 bitowe wyjście VGA.

Terasic DE10-Lite Board korzysta z 64MB pamięci SDRAM na 16-bitowej szynie danych. Płytka zasilana jest prądem stałym o natężeniu 5V.

Link do pełnej specyfikacji układu: https://download.kamami.pl/p562820-DE10-Lite_User_Manual.pdf

Terminy

- **09.10.2019** - zgłoszenie tematu osobie prowadzącej zajęcia, uzyskanie akceptacji,
- **16.10.2019** - oddanie pierwszej wersji dokumentacji projektu,
- **30.10.2019** - zapoznanie się z podstawami tworzenia oprogramowania dla układów FPGA oraz językiem,
- **13.11.2019** - przygotowanie teoretyczne dla wyświetlania obrazu na monitorze przez złącze VGA,
- **20.11.2019** - stworzenie modułu pozwalającego wyświetlać obraz na monitorze poprzez złącze VGA,
- **27.11.2019** - rozszerzenie funkcjonalności płytki przez podłączenie złącza ps/2,
- **11.12.2019** - rozpoczęcie pracy nad logiką gry, zaplanowanie oraz omówienie planowanych funkcjonalności oraz sposobu ich realizacji,
- **30.12.2019** - praca nad mechaniką gry,
- **7.01.2019** - zakończenie prac nad mechaniką gry, wprowadzenie ostatnich poprawek, ukończenie dokumentacji.

Reguły gry

Na planszy pokazują się dwie postacie. Najłatwiej zidentyfikować je jako węże. Kiedy gracz naciśnie dowolny przycisk związany ze sterowaniem swojej postaci, to jego postać zaczyna się poruszać, bez możliwości zatrzymania. Zabieg ten został zrobiony celowo – warto na początku odczekać parę chwil i zobaczyć w którą stronę uda się nasz przeciwnik. Każda z postaci początkowo wydłuża się, aż do momentu uzyskania odpowiedniej (zaprogramowanej) długości. Gracze sterują postaciami w dowolnym, wybranym przez siebie kierunku. Celem gry jest doprowadzenie do sytuacji, gdy drugi gracz wejdzie w naszego węża. Jeśli do tego doprowadzimy, gra kończy się a cały ekran pokrywa się kolorem zwycięskiego gracza.

UWAGA: Należy uważać na obramowanie planszy zaznaczone kolorem białym, oraz przeszkodę umieszczoną na środku! Jeśli w nią wejdziemy – przegrywamy.

Planowane bonusy

- przyspieszenie,
- zwolnienie,
- usunięcie dotychczasowych śladów innych graczy,
- usunięcie dotychczasowego śladu gracza zdobywającego bonus,
- czasowa nieśmiertelność,
- czasowy zanik ścian ograniczających pole gry,
- czasowa zmiana sterowania innym graczom,
- czasowa zmiana sterowania dla gracza zdobywającego bonus.

UWAGA – określenie planowane zostało użyte nie bez przyczyny. Niestety podczas robienia projektu zrezygnowaliśmy w nich.

Prace nad projektem

Obraz

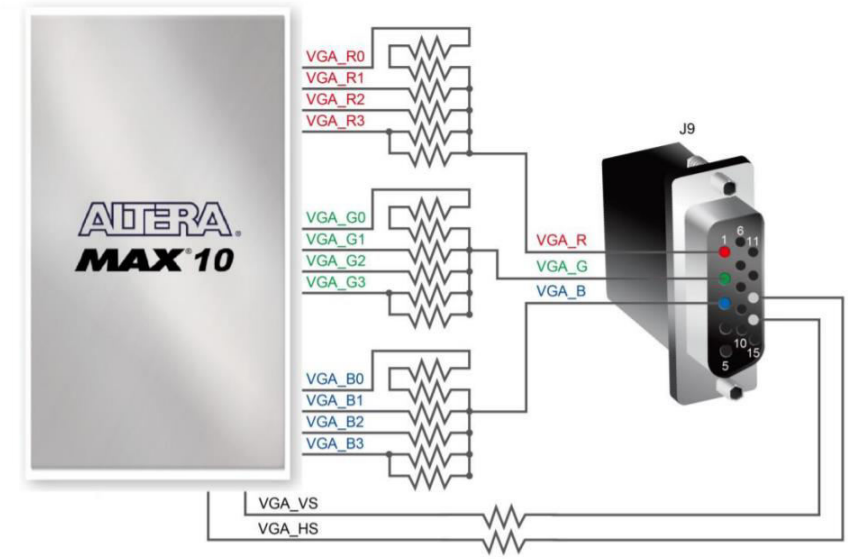
Początek pracy z projektem to, (nie licząc zapoznania się ze specyfikacją oraz instrukcją obsługi płytki, instalacją niezbędnego oprogramowania Quartus (**Quartus Prime Lite 19.1**)) zabraliśmy się za podłączenie płytki FPGA do monitora z wyjściem VGA.

Zagadnienie teoretyczne:

VGA to analogowy standard wideo wykorzystujący 15-stykowe złącze D-sub. Nie wymaga wysokich częstotliwości taktowania ani skomplikowanego kodowania.

VGA ma pięć głównych pinów sygnałowych odpowiednio po jednym dla każdego z kolorów: czerwonego, zielonego, niebieskiego oraz dwa do synchronizacji. Synchronizacja pozioma wyznacza linię. Synchronizacja pionowa wyznacza ekran, zwany także ramką.

Schemat połączeń prezentuje się następująco:



Impulsy VGA składają się z dwóch faz: rysowanie pikseli i interwał wygaszania. Sygnały synchronizacji występują w odstępach czasu wygaszania; oddzielone od rysowania pikseli przez ganek frontowy i ganek tylny. Kiedy opracowano VGA, monitory były oparte na lampach katodowych (CRT): interwał wygaszania daje czas na ustabilizowanie się poziomów napięcia i powrót działa elektronowego na początek linii lub ekranu.

Projekt zakłada stworzenie klasycznego wyświetlacza VGA **640 x 480 pikseli,** odświeżanego z częstotliwością **60 Hz**. Zegar pikselowy będzie taktowany na **25 MHz**, co stanowi prostą do podziału część zegara **50 MHz** używanej płytki. Kluczem do wytworzenia prawidłowego sygnału VGA jest odpowiednie ustawienie taktowania. Dla uproszczenia wykonamy obliczenia w pikselach i liniach. Każdy piksel jest tyknięciem zegara pikselowego **25 MHz** (**40 ns**). Linia to kompletny zestaw poziomych pikseli.

Wbrew pozorom, okno wyświetlane na monitorze VGA o rozmiarze 640x480 pikseli, jest dużo większe. Aby uzyskać poprawne działanie okna, należy zapewnić następujące ramki:

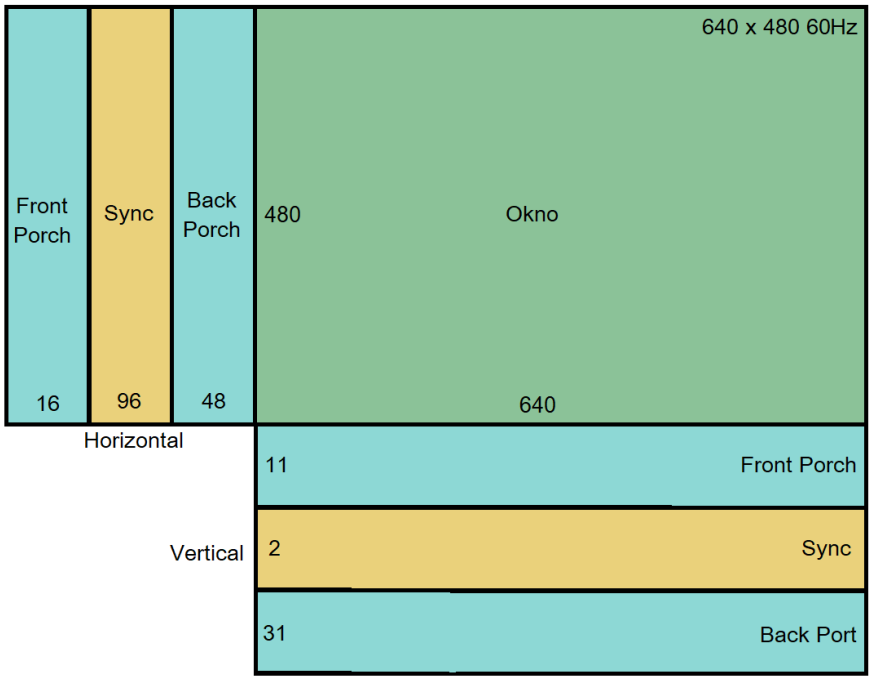
Horizontal Pixel Timings

- Front Porch: 16
- Sync Pulse: 96
- Back Porch: 48
- Okno: 640
- Suma:: 800

Vertical Line Timings

- Okno: 480
- Front Porch: 10
- Sync Pulse: 2
- Back Porch: 33
- Suma: 525

Podgląd:



Prezentacja kodu źródłowego:

UWAGA – zamieszczony tutaj kod źródłowy jest uproszczony i służy jedynie do pokazania sposobu realizacji danego zagadnienia

W głównym module Tron zostały zadeklarowane następujące pola:

```
1  module Tron(
2
3      ...
4
5      //do wyświetlacza
6      output wire [9:0] xCount, yCount,     //liczniki pikseli
7      output reg displayArea,                //wyświetlany obszar
8      output VGA_hSync, VGA_vSync,           //odpowiednio horyzontalna, oraz wertykalna synchronizacja
9      output wire [3:0] Red, Green, Blue     //kolory - czerwony, zielony, niebieski
10     );
11
12     wire VGA_clk;
13     wire R;                 //obszary, który zaznaczymy na czerwono
14     wire G;                 //obszary, który zaznaczymy na zielono
15     wire B;                 //obszary, który zaznaczymy na niebiesko
16
17     ...
18
19     clk_reduce reduce1(clk, VGA_clk);       //redukcja zegara do VGA (moduł napisany niżej)
20     VGA_gen gen1(VGA_clk, xCount, yCount,   //generowanie VGA
21                  displayArea, VGA_hSync, VGA_vSync);
22
23     ...
24
25     always @(posedge VGA_clk)              //generowanie ramki i areny do gry
26     begin
27             border <= (((xCount >= 0) && (xCount < 20) || (xCount >= 630) && (xCount < 641)) ||
28                        ((yCount >= 0) && (yCount < 11) || (yCount >= 470) && (yCount < 481))) ||
29                        (((xCount >= 160 && xCount < 170) || (xCount >= 480 && xCount < 490)) &&
```

```
30              ((yCount >= 160 && yCount < 230) || (yCount >= 250 && yCount < 320))) ||
31                          ((xCount >= 160 && xCount < 480) && ((yCount >= 160 && yCount < 170) ||
32                          (yCount >= 310 && yCount < 320)));
33      end
34
35      ...
36
37      //obsluga wyswietlania
38
39      //przypisanie obszarów do zaznaczenia
40      assign R = (displayArea && (border || snakeHead2 || endGameR));
41      assign G = (displayArea && (border || snakeHead1 || endGameG));
42      assign B = (displayArea && (border || snakeBody1 || snakeBody2) && (~endGameR && ~endGameG));
43      always@(posedge VGA_clk)
44      begin
45              Red = {3{R}};   //kolorwanie
46              Green = {3{G}};
47              Blue = {3{B}};
48      end
49  end module
50
51  module clk_reduce(clk, VGA_clk);            //redukcja zegara
52
53      input clk;                      //50MHz clock
54      output reg VGA_clk;             //25MHz clock
55      always@(posedge clk)
56      begin
57              VGA_clk=~VGA_clk;
58      end
59
60  endmodule
61
62  module VGA_gen(VGA_clk, xCount, yCount, displayArea, VGA_hSync, VGA_vSync);
63
64      input VGA_clk;
65      output reg [9:0]xCount, yCount;
66      output reg displayArea;
67      output VGA_hSync, VGA_vSync;
68
69      reg p_hSync, p_vSync;
70                              //przepraszam za angielskie komentarze, ale tak najlepiej to opisać
71      integer porchHF = 640;          //start of horizntal front porch
72      integer syncH = 655;            //start of horizontal sync
73      integer porchHB = 747;          //start of horizontal back porch
74      integer maxH = 793;             //total length of line
75
76      integer porchVF = 480;          //start of vertical front porch
77      integer syncV = 490;            //start of vertical sync
78      integer porchVB = 492;          //start of vertical back porch
79      integer maxV = 525;             //total rows
80
81      always@(posedge VGA_clk)        //generowanie xCount
82      begin
83              if(xCount === maxH)
84                      xCount <= 0;
85              else
86                      xCount <= xCount + 1;
87      end
88
89      always@(posedge VGA_clk)        //generowanie yCount
90      begin
91              if(xCount === maxH)
92              begin
93                      if(yCount === maxV)
94                              yCount <= 0;
95                      else
96                      yCount <= yCount + 1;
97              end
98      end
99
100     always@(posedge VGA_clk)        //wyświetlany obszar
101     begin
102             displayArea <= ((xCount < porchHF) && (yCount < porchVF));
103     end
104
105     always@(posedge VGA_clk)        //synchronizacja
106     begin
```

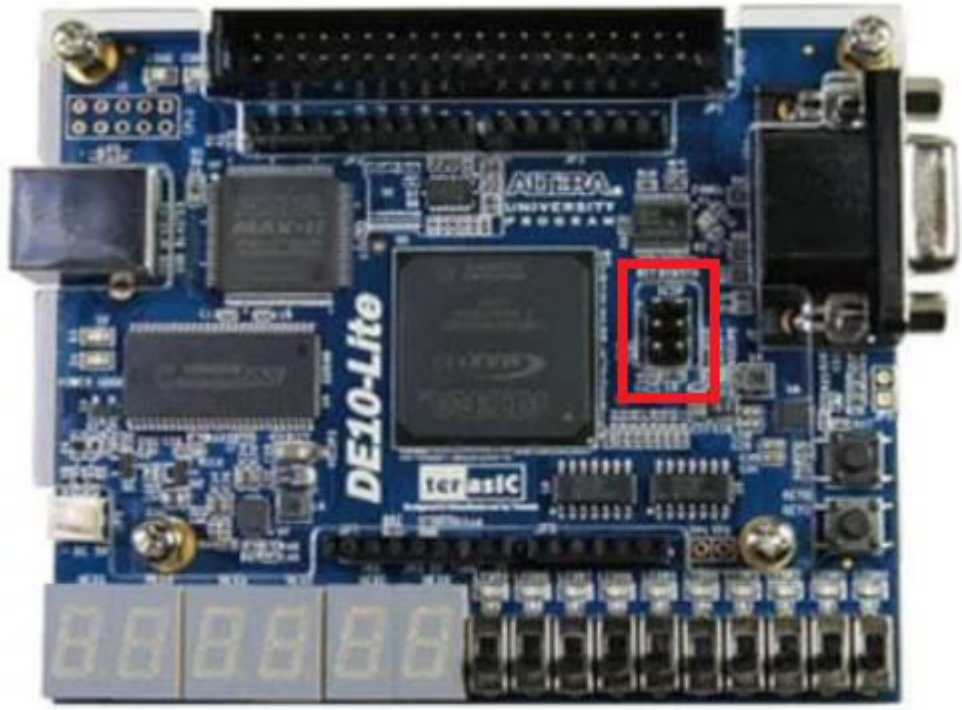
```
107             p_hSync <= ((xCount >= syncH) && (xCount < porchHB));
108             p_vSync <= ((yCount >= syncV) && (yCount < porchVB));
109     end
110
111     assign VGA_vSync = ~p_vSync;
112     assign VGA_hSync = ~p_hSync;
113 endmodule
```

Sterowanie

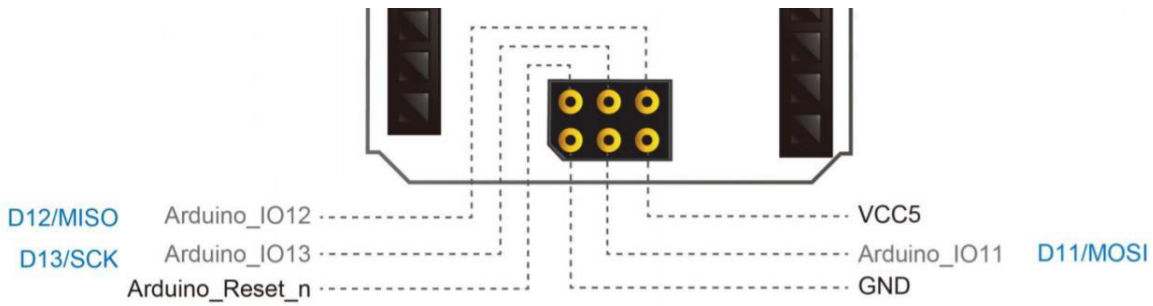
Gracze będą mogli sterować swoimi postaciami w grze za pomocą klawiatury.

Rozszerzenie funkcjonalności płytki FPGA

Płytka FPGA (**Terasic DE10-Lite Board (P0466)**) nie była fabrycznie wyposażona w złącze ps2, aby podpiąć do niej klawiaturę. Płytka miała jednak złącza określone w instrukcji jako Arduino Connectors. Zostały one zaznaczone na czerwono na rysunku:

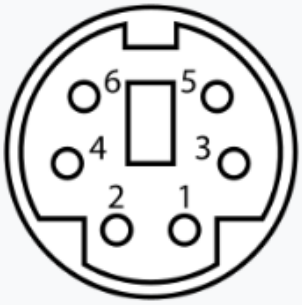


Jak widzimy mamy dostępne 6 pinów. Ich specyfikacja w instrukcji wygląda następująco:



Taki układ jest dla nas idealny! Możemy wykorzystać go do naszych celów. Aby podłączyć klawiaturę, musieliśmy mieć żeńskie wejście ps2. W tym celu zakupiliśmy kabel:

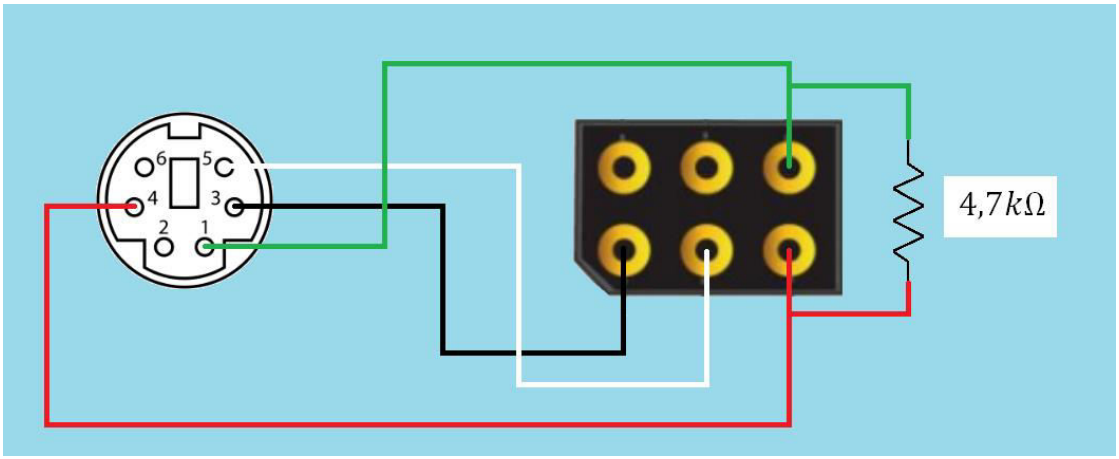
Zielona końcówka została odcięta w celu zaadoptowania jej do płytki. Złącze ps2 przesyła dane w następujący sposób:



Female connector from the front

Pin 1	+DATA	Data
Pin 2		Not connected[b]
Pin 3	GND	Ground
Pin 4	Vcc	+5 V DC at 275 mA
Pin 5	+CLK	Clock
Pin 6		Not connected[c]

Jak widzimy ze schematu, mimo, że jest 6 wejść w złączu, wykorzystanych zostało tylko 4. W ten oto sposób połączyliśmy złącze do płytki w następujący sposób:



Złącze PS/2	Złącze na płytce
Pin 1 DATA	Arduino_IO12
Pin 3 GND	GND
Pin 4 VCC	VCC5
Pin 5 CLK	Arduino_IO11

Pomiędzy kablami czerwonym a zielonym został wpięty opornik 4,7 kΩ. Jest on niezbędny do prawidłowego działania układu.

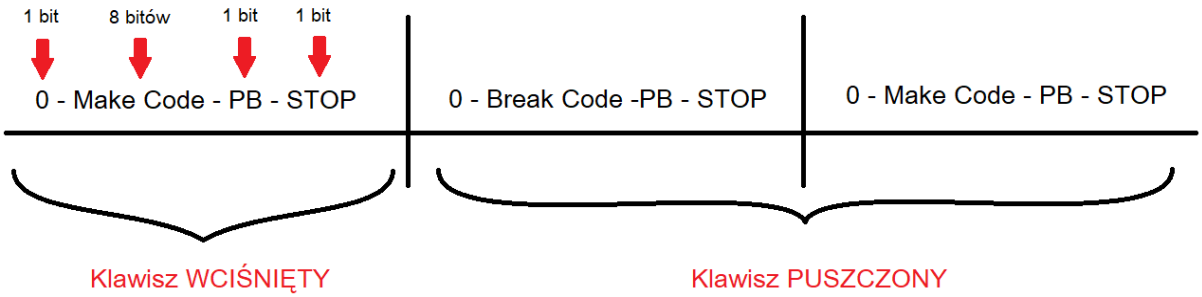
Klawiatura

Współpraca naszej płytki z klawiaturą nie jest zagadnieniem bardzo skomplikowanym. Jest kilka podstawowych rzeczy, które trzeba wiedzieć:

1. Kiedy klawiatura nie wysyła danych, wysyła stale logiczną 1 (stan wysoki), tak samo jak zegar.
2. Wciśnięcie jednego klawisza powoduje wysłanie co najmniej 33 bitów danych wejściowych z klawiatury do płytki.
3. Kiedy pierwszy raz wciskamy przycisk wysyła on 11 bitów danych w postaci tzw. kodu make. Kiedy zwolnimy przycisk wysyła on 11 bitów tzw. break kodu, następnie ponownie ten sam kod make. (Od tej zasady istnieją wyjątki).
4. Klawiatura wysyła w kółko ten sam kod make (te 11 bitów) tak długo, jak trzymamy wciśnięty określony klawisz.
5. Klawiatura wysyła dane na ujemne zbocze swojego zegara.
6. Klawiatura posiada swój własny zegar, który możemy odczytać. Nie ma potrzeby definiowania własnego zegara aby współpracować z klawiaturą.

Adnotacja dla punktów 2 i 3:
Pierwszym wysłanym przez klawiaturę bitem jest zero (aby rozpocząć), następnie 8 bitów danych specyficznych dla naciśniętego klawisza (kod make), bit parzystości (PB), a na koniec jeden do zatrzymania. Bit parzystości ma wartość jeden lub zero, w zależności od liczby wystąpień 1 w kodzie make. Jeśli liczba wystąpień 1 jest parzysta, będzie to 1, a jeśli liczba wystąpień 1 jest nieparzysta, bit parzystości przyjmie wartość 0. Ten bit gwarantuje, że liczba jedynek w całej sekwencji jest nieparzysta.

Ilustracja:



Klawisz	Scan Code Make (break)
W	1D (F01D)
A	1C (F01C)
S	1B (F01B)
D	23 (F023)
I	43 (F043)
J	3B (F03B)
K	42 (F042)
L	4B (F04B)

https://techdocs.altium.com/display/FPGA/PS2+Keyboard+Scan+Codes

Prezentacja kodu źródłowego:

UWAGA – zamieszczony tutaj kod źródłowy jest uproszczony i służy jedynie do pokazania sposobu realizacji danego zagadnienia

W głównym module Tron zostały zadeklarowane następujące pola:

```
module Tron(

        ...

        //do klawiatury
        input keyboardData, keyboardCLK,

        ...
        );

        ...

        kbInput kbIn(keyboardCLK, keyboardData, direction1, direction2); //obsługa klawiatury

        ...

end module

        ...

module kbInput(keyboardCLK, keyboardData, direction1, direction2);

        input keyboardCLK, keyboardData;
        output reg [4:0] direction1, direction2;
        reg [7:0] code;
        reg [10:0]keyCode;
        reg recordNext = 0;
        integer count = 0;

always@(negedge keyboardCLK)          //liczenie kodów klawiatury w celu uniknięcia przesunięcia fazowego
        begin
                keyCode[count] = keyboardData;
                count = count + 1;
                if(count == 11)
                begin
                        code = keyCode[8:1];
                        count = 0;
                end
        end

        always@(code)     //obsługa sterowania wężami, skręcanie
        begin
                if(code == 8'h1D)
                begin
                        direction1 <= 5'b00010;
                        direction2 <= direction2;
                end
                else if(code == 8'h1C)
                begin
                        direction1 <= 5'b00100;
                        direction2 <= direction2;
                end
                else if(code == 8'h1B)
                begin
                        direction1 <= 5'b01000;
                        direction2 <= direction2;
                end
                else if(code == 8'h23)
                begin
                        direction1 <= 5'b10000;
                        direction2 <= direction2;
                end
                else if(code == 8'h43)
                begin
                        direction2 <= 5'b00010;
                        direction1 <= direction1;
                end
```

```
68                else if(code == 8'h3B)
69                begin
70                        direction2 <= 5'b00100;
71                        direction1 <= direction1;
72                end
73                else if(code == 8'h42)
74                begin
75                        direction2 <= 5'b01000;
76                        direction1 <= direction1;
77                end
78                else if(code == 8'h4B)
79                begin
80                        direction2 <= 5'b10000;
81                        direction1 <= direction1;
82                end
83                else
84                begin
85                        direction2 <= direction2;
86                        direction1 <= direction1;
87                end
88        end
89 endmodule
```

Mechanika gry

Prezentacja kodu źródłowego:

UWAGA – zamieszczony tutaj kod źródłowy jest uproszczony i służy jedynie do pokazania sposobu realizacji danego zagadnienia

Poniższy kod źródłowy pokazuje sposób w jaki wąż się porusza.

W głównym module Tron zostały zadeklarowane następujące pola:

```
 1 module Tron(
 2
 3         ...
 4
 5         wire [4:0] direction1, direction2;
 6         reg [6:0] size = 90;
 7         reg [9:0] snakeX1[0:127];
 8         reg [8:0] snakeY1[0:127];
 9         reg [9:0] snakeHeadX1;
10         reg [9:0] snakeHeadY1;
11         reg [9:0] snakeX2[0:127];
12         reg [8:0] snakeY2[0:127];
13         reg [9:0] snakeHeadX2;
14         reg [9:0] snakeHeadY2;
15         reg snakeHead1;
16         reg snakeBody1;
17         reg snakeHead2;
18         reg snakeBody2;
19         reg endGameR, endGameG;
20         reg border;
21         reg found;
22         wire update, reset;
23         integer count1, count2, count3;
24
25         ...
26
27         updateClk UPDATE(clk, update); //redukcja clocka do poruszania
28
29         ...
30         //UWAGA w kodzie źródłowym znajdują sie identyczne funkcje do sterowania drugim wężem
31         always@(posedge update) //poruszanie weza 1
32         begin
33                 if(start)
34                 begin
35                         for(count1 = 127; count1 > 0; count1 = count1 - 1)
```

```
                                begin
                                        if(count1 <= size - 1)
                                        begin
                                                snakeX1[count1] = snakeX1[count1 - 1];
                                                snakeY1[count1] = snakeY1[count1 - 1];
                                        end
                                end
                        case(direction1)
                                5'b00010: snakeY1[0] <= (snakeY1[0] - 10);
                                5'b00100: snakeX1[0] <= (snakeX1[0] - 10);
                                5'b01000: snakeY1[0] <= (snakeY1[0] + 10);
                                5'b10000: snakeX1[0] <= (snakeX1[0] + 10);
                        endcase
                end
                else if(~start)
                begin
                        for(count3 = 1; count3 < size; count3 = count3+1)
                                begin
                                        snakeX1[count3] = 100;
                                        snakeY1[count3] = 100;
                                end
                        snakeX1[0] = 110;
                        snakeY1[0] = 100;
                end

        end


        always@(posedge VGA_clk) //generowanie ciala weza 1
        begin
                found = 0;

                for(count2 = 1; count2 < size; count2 = count2 + 1)
                begin
                        if(~found)
                        begin
                                snakeBody1 = ((xCount > snakeX1[count2] && xCount < snakeX1[count2]+10)
                                             && (yCount > snakeY1[count2] && yCount < snakeY1[count2]+10));
                                found = snakeBody1;
                        end
                end
        end

        always@(posedge VGA_clk) //generowanie glowy weza 1
        begin
                snakeHead1 = (xCount > snakeX1[0] && xCount < (snakeX1[0]+10)) &&
                             (yCount > snakeY1[0] && yCount < (snakeY1[0]+10));
        end

        ...

end module

        ...

module updateClk(clk, update);
        input clk;
        output reg update;
        reg [21:0]count;

        always@(posedge clk)
        begin
                count <= count + 1;
                if(count == 1777777)
                begin
                        update <= ~update;
                        count <= 0;
                end
        end
endmodule
```

Podsumowanie projektu:

Prace nad projektem zostały zaplanowane na cały semestr, a osoby biorące w nim udział starały się trzymać ustalonych wcześniej terminów. Niestety nie wszystko udało się zrealizować – podczas wykonywania projektu została podjęta decyzja o zrezygnowaniu z bonusów. Wynika to z faktu braku czasu, zwłaszcza w okolicy końca semestru – trzeci rok studiów nie bez przyczyny uznawany jest za jeden z najcięższych. Póki co ta zasada się potwierdza.

Warto wylistować, oraz pokrótce opisać nabyte umiejętności:

- Stworzenie wyświetlania obrazu na monitorze – była to dla nas nowość! Podczas naszego pierwszego projektu związanego z układami FPGA korzystaliśmy jedynie z siedmiosegmentowego wyświetlacza na płytce. W tym projekcie nauczyliśmy się działania oraz wyświetlania obrazu za pomocą kabla VGA. Okazało się, że nie jest to zagadnienie bardzo skomplikowane – wbrew naszym początkowym obawom.
- Rozszerzenie funkcjonalności płytki – nasz projekt wymagał sterowania postaciami, a więc potrzebowaliśmy klawiatury! Fabrycznie nasza płytka (**DTE 10Lite**) nie jest wyposażona ani w złącze PS/2, ani w USB. Byliśmy zmuszeni do rozszerzenia funkcjonalności naszej płytki, poprzez zlutowanie żeńskiego złącza do nakładki na piny w naszej płytce. Nabyliśmy umiejętność rozszerzania funkcjonalności układów FPGA, oraz działania klawiatury.
- Stworzenie spójnej i działającej mechaniki gry. Umiejętność, która mamy nadzieję zaowocuje w przyszłości!

Obie osoby są zadowolone oraz bardzo usatysfakcjonowane efektami własnej pracy. Uważamy, że wybrany temat bardzo nas rozwinął, oraz okazał się bardziej ciekawy niż wydawało się to na początku! Nauczyliśmy się zupełnie innego stylu programowania. Zostaliśmy zmuszeni do sprzętowego myślenia na poziomie bramek i rejestrów i porzucenia całkowicie programistycznego myślenia jakiego nauczyliśmy się wraz z językami C oraz Java. Warto nadmienić, że nikt z uczestników nie miał większej wiedzy ani doświadczenia w wytwarzaniu oprogramowania w taki sposób.

Wnioski

- Układy FPGA dają bardzo szerokie spektrum możliwości – my stworzyliśmy grę, ale to tylko jedna z kilkudziesięciu możliwości wykorzystania naszej płytki.
- Sposób myślenia – inny niż w wysokopoziomowych językach jest naszym zdaniem bardzo istotną rzeczą. W perspektywie kilkuletniej programiści – specjaliści w jakiejś wąskiej dziedzinie, powinniśmy mieć szerszy obraz na Informatykę jako całość. Ten projekt zdecydowanie poszerzył nasze horyzonty.
- Układy FPGA, choć momentami trudne są bardzo interesujące i przyszłościowe – niedawno została wydana książka o tematyce wytwarzania gier w Verilogu. Link: https://www.amazon.com/Designing-Video-Game-Hardware-Verilog/dp/1728619440

Literatura:

Specyfikacja płytki:	https://download.kamami.pl/p562820-DE10-Lite_User_Manual.pdf
Podstawy VGA:	https://timetoexplore.net/blog/arty-fpga-vga-verilog-01
Przykładowa gra w na układ FPGA:	https://www.fpga4fun.com/PongGame.html
Filmik o vga i fpga:	https://www.youtube.com/watch?v=4enWoVHCykI&t=12s
Klawiatura w układach FPGA:	https://www.youtube.com/watch?v=EtJBqvk1ZZw
Kurs FPGA:	https://forbot.pl/blog/kurs-fpga-podstawy-vhdl-w-praktyce-spis-tresci-id22265