Przemysław Jedlikowski, 234927
Karol Kłusek, 235082 14.06.2019

Układy cyfrowe i systemy wbudowane 2

Projekt gry używającej VGA oraz RS232

Prowadzący: mgr inż. Antoni Sterna

Spis treści

1 Opis projektu

1.1 Cel projektu

Celem projektu było stworzenie gry typu *shooter*, w której gracz będzie strzelać do przeciwnika napierającego w jego kierunku. Gracz operował „paletką" umiejscowioną na dole ekranu, przeciwnik pojawiał się zaś na górze i systematycznie na niego spadał.

Projekt powinien implementować:

1. kontroler/sterownik VGA obsługujący rozdzielczość 1024×768,
2. moduł do transmisji przy użyciu RS232.

1.2 Opis gry

Gra jest renderowana w czasie rzeczywistym, co oznacza, że postępuje nawet bez interakcji gracza. Iluzję płynnego ruchu zapewnia się wysoką częstotliwością renderowania klatek - 30 na sekundę przy rozdzielczości 1024×768.

Gracz może wystrzelić pocisk (jeden w danym czasie - poprzednio wystrzelony musi zniknąć z planszy) po wciśnięciu klawisza. Pocisk wędruje z dołu ekranu do góry. Ostatecznie trafia w gracza lub „ucieka" z ekranu. **Prędkość pocisku: 9 px na klatkę.**

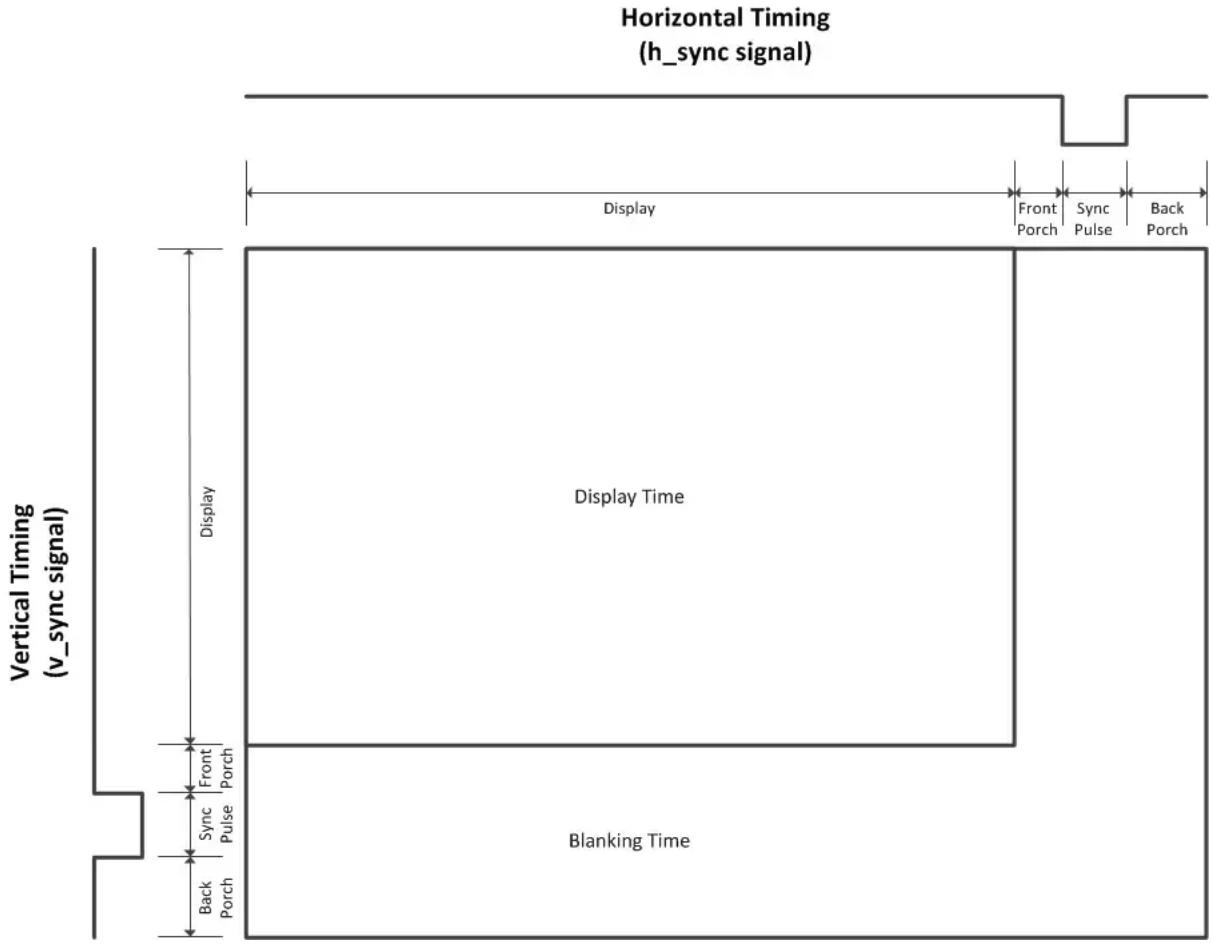
Gracz porusza się wzdłuż dolnej krawędzi ekranu. Jego model powinien powędrować z lewej do prawej krawędzi w ciągu 4 sekund, a zatem w 120 klatkach. Nie wolno mu wyjść poza obręb planszy. **Szybkość gracza: 7 px na klatkę.**

Przeciwnik jest jedyną przeszkodą w grze. Pojawia się na górnej krawędzi ekranu na różnej szerokości i opada. Na planszy jest jeden przeciwnik w tym samym czasie. Po dotknięciu gracza lub osiągnięciu dolnej krawędzi ekranu, gra się kończy. **Prędkość przeciwnika: 3 px na klatkę.**

1.3 Wyświetlany obraz

Realizując projekt, zdecydowano się na skorzystanie z rozdzielczości wyświetlania 1024×768 px. Konfiguracja ta wymaga następujących zależności czasowych, zaprezentowanych na rysunku 1 [3]:

- Częstotliwość odświeżania (ang. *Refresh Rate*): 60 Hz,
- Częstotliwość zegara pikseli (ang. *Pixel Clock*): 65 MHz,
- Cykle zegara pikseli – poziomo:
  - Czas wyświetlania: 1024,
  - *Front Porch*: 24,
  - Puls synchronizacji: 136,
  - *Back Porch*: 160;
- Cykle zegara pikseli – pionowo:
  - Czas wyświetlania: 768,
  - *Front Porch*: 3,
  - Puls synchronizacji: 6,
  - *Back Porch*: 29;
- Polaryzacja pozioma (`h_sync`): n („0"),
- Polaryzacja pionowa (`v_sync`): n („0").



Rysunek 1: Diagram czasowy sygnału wyświetlania VGA. Źródło: [3].

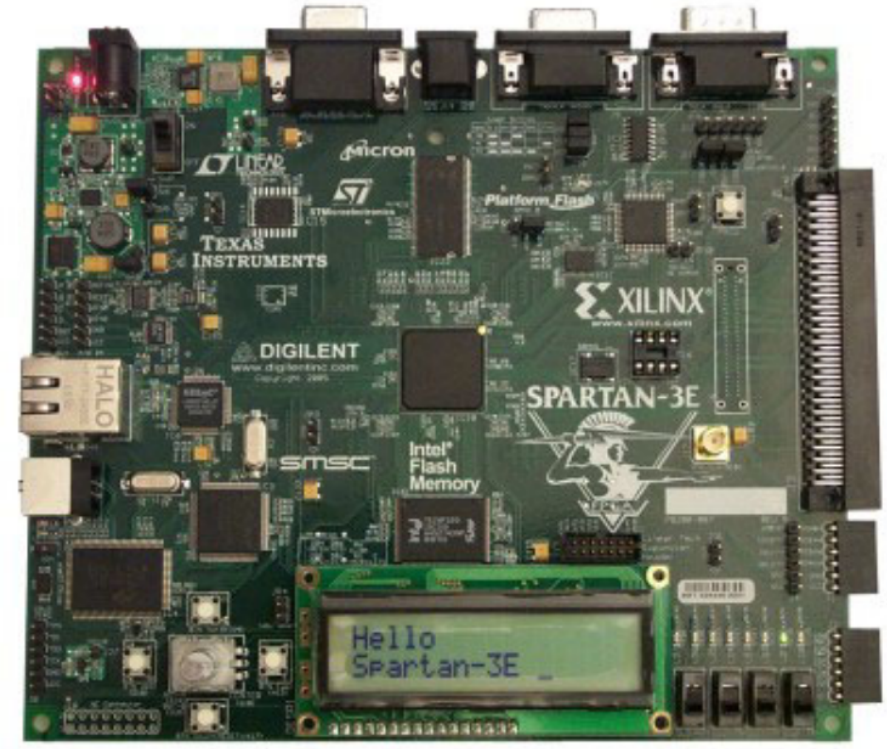
Wskazane czasy zostały zdefiniowane w module **vga_controller** opisanym w rozdziale 3.1.5.

2 Sprzęt

2.1 Opis płytki Spartan-3E

Spartan®-3E FPGA Starter Kit jest zestawem deweloperskim służącym do pracy z układami FPGA. Wyprodukowany przez firmę Xilinx w 2006 r. układ ma następujące cechy kluczowe [2]:

- Ponad 10 000 komórek logicznych,
- 4 Mb pamięci PROM,
- 64 MB (512 Mb) pamięci DDR SDRAM,
- 2-liniowy, 16-znakowy wyświetlacz LCD,
- porty PS/2 (mysz/klawiatura), VGA, Ethernet (10/100 Mb),
- dwa 9-pinowe porty RS-232 (DTE, DCE),
- zegar 50 MHz,
- 4-wyjściowy konwerter cyfrowo-analogowy (SPI),
- 2-wejściowy konwerter analogowo-cyfrowy (SPI) z programowalnym przedwzmacniaczem,
- 8 diod LED,
- 4 przyciski.



Rysunek 2: Zdjęcie poglądowe płytki Spartan-3e. Źródło: [2].

2.2 Wykorzystanie płytki w projekcie

W niniejszym projekcie wykorzystano następujące elementy zestawu:

- Port RS232 (DTE),
- Port VGA,
- Zegar 50 MHz,
- Moduł IP Core generujący szybszy zegar (patrz: rozdział 1.3).

Po przeprowadzeniu syntezy i implementacji projektu, środowisko Xilinx ISE WebPack (ISE Project Navigator) wygenerowało raport, z którego można odczytać, że wykorzystano następującą ilość dostępnych układów (m.in.):

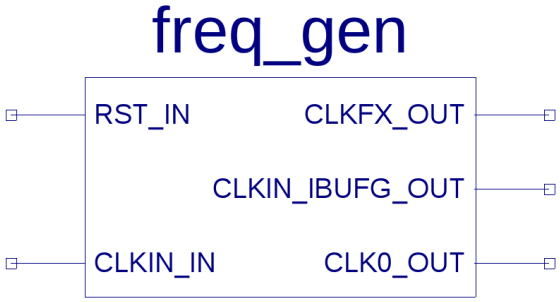
- 167 z 9312 (1%) przerzutników (*Slice Flip Flops*),
- 471 z 9312 (5%) 4-wejściowych „tablic tęczowych" / „tablic przeszukiwania" (*Look-Up Tables*, *LUTs*), w tym 401 do obsługi logiki, a 70 jako *route-thru*,
- 9 z 232 (3%) bloków wejściowo-wyjściowych (*Input-Output Blocks*, *IOBs*),
- 3 z 24(12%) multiplekserów zegarowych (*BUFGMUXs*),
- 1 z 4 (25%) cyfrowych zarządów zegara (ang. *Digital Clock Managers*, *DCMs*).

3 Struktura projektu

3.1 Stworzone moduły

3.1.1 Moduł freq_gen

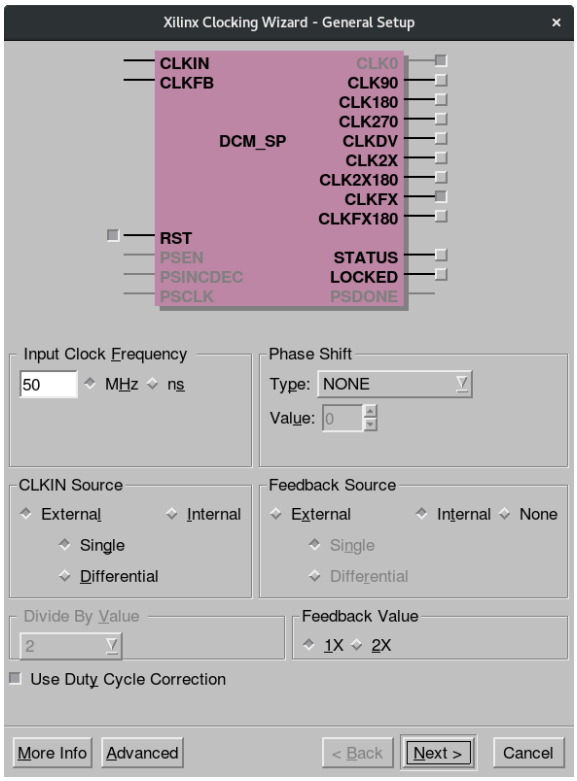
Moduł **freq_gen** jest jedynym modułem w projekcie, który został stworzony z zasobów udostępnianych przez płytkę Spartan-3E oraz środowisko Xilinx WebPack ISE. Jest to moduł *IP core*, którego zadaniem jest oversampling sygnału zegarowego. Do modułu podłączony zostanie dostępny na płytce zegar 50 MHz, moduł udostępni zaś ten sygnał oraz sygnał zegarowy 65 MHz, który zostanie dołączony do modułu **vga_controller**.



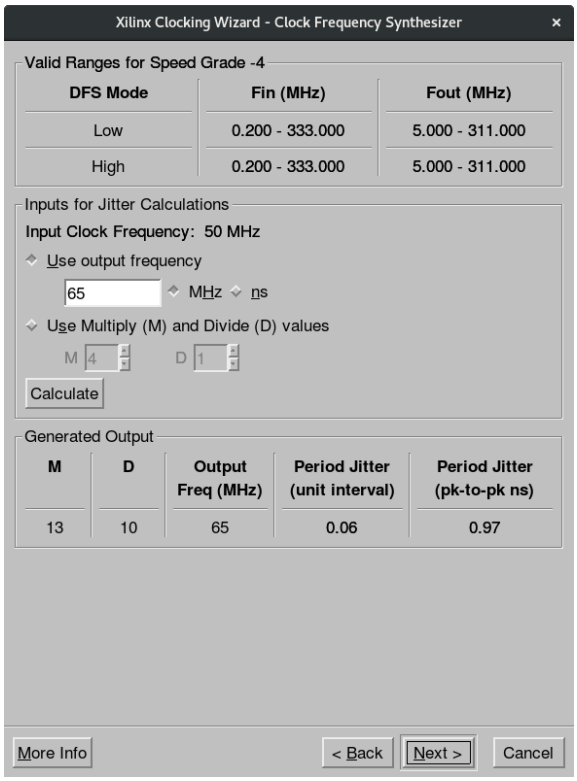
Rysunek 3: Symbol modułu **freq_gen**.

Moduł **freq_gen** został wygenerowany przy pomocy „kreatora zegarowego" (ang. *Clocking Wizard*). Na rysunku 4 zaprezentowano pierwszy ekran konfiguracyjny tego modułu, na którym oznaczono sygnały `RST` – asynchroniczny reset układu, `CLK0` – sygnał wejściowego zegara nieprzesunięty w fazie, oraz `CLKFX`, który dostarcza *w pełni cyfrowy, dedykowany syntezator częstotliwości* [1]. Wpisano także informację o zegarze wejściowym – 50 MHz, który zostanie dołączony do sygnału `CLKIN`.

Na rysunku 5 przedstawiono trzeci ekran konfiguracji, na którym podano oczekiwano częstotliwość wyjściową. Wyjaśnienie, dlaczego jest to właśnie 65 MHz, podano w rozdziale 1.3.



Rysunek 4: Pierwszy ekran konfiguracji nowego zegara.

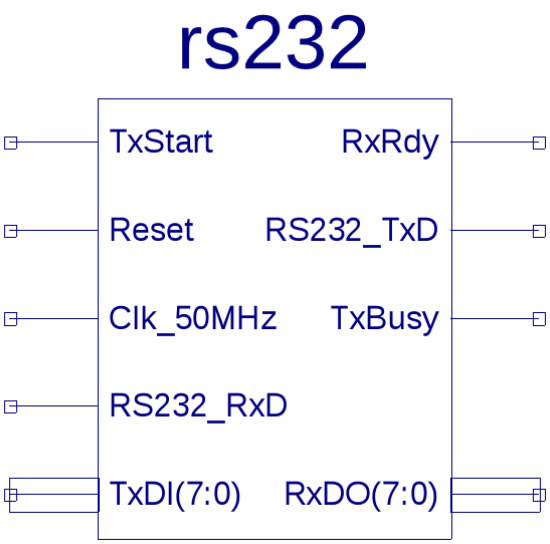


Rysunek 5: Trzeci ekran konfiguracji modułu.

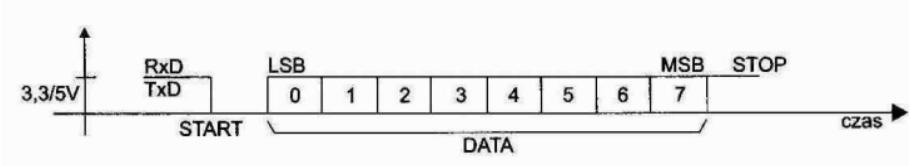
3.1.2 Moduł rs232

Moduł **RS232** zapewnia jeden z najprostszych sposobów komunikacji szeregowej między komputerami a urządzeniami zewnętrznymi. Typowy przebieg transmisji przedstawiono na rysunku 7. W stanie spoczynku linia pozostaje w stanie wysokim; wysyłanie rozpoczyna się przez wysłanie *bitu startu* – „0”. Następnie nadawane są kolejno bity od najmłodszego do najstarszego (ang. *Least Significant Bit first*). Transmisję kończy *bit stopu* – „1” [4, str. 151].

Stworzony moduł pozwala na jednoczesne nadawanie bajtów i ich odbieranie. Jego konfiguracja pozwala na transmisję w prędkości 115200 *bps*, 8 bitów danych, bez bitów parzystości i z 1 bitem stopu.



Rysunek 6: Symbol modułu **rs232**.



Rysunek 7: Schemat transmisji RS232. Źródło: [4]

Listing 1: Definicja modułu **rs232**.

```
5  entity rs232 is
6      Generic (cykle_na_bit: INTEGER := 434);
7      Port (TxDI:         in   STD_LOGIC_VECTOR (7 downto 0);
8            TxStart:      in   STD_LOGIC;
9            Reset:        in   STD_LOGIC;
10           Clk_50MHz:    in   STD_LOGIC;
11           RS232_RxD:    in   STD_LOGIC;
12           RxRdy:        out  STD_LOGIC                    := '1';
13           RS232_TxD:    out  STD_LOGIC                    := '1';
14           RxDO:         out  STD_LOGIC_VECTOR (7 downto 0) :=
                (others => '0');
15           TxBusy:       out  STD_LOGIC                    :=
                '0');
16 end rs232;
```

Wejścia i wyjścia tego modułu oznaczają:

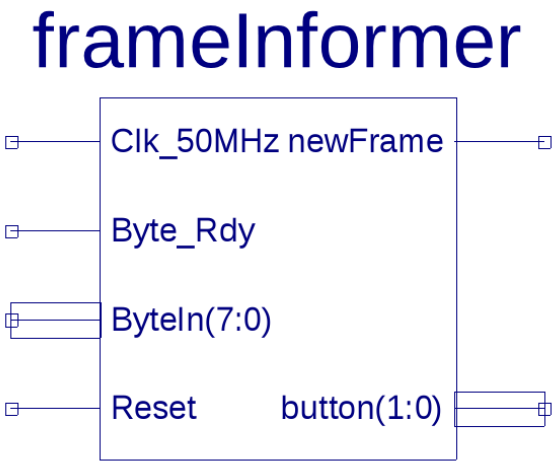
- `TxDI` – bajt do wysłania,
- `TxStart` – jednotaktowy impuls rozpoczynający transmisję bajtu do odbiornika,
- `Reset` – sygnał asynchronicznego resetu układu,

- `Clk_50MHz` – wejście dla zegara o częstotliwości 50 MHz,
- `RS232_RxD` – linia odbierania sygnału,
- `RxRdy` – jednotaktowy impuls służący do poinformowania, że odebrano bajt,
- `RS232_TxD` – linia nadawania sygnału,
- `RxDO` – odebrany bajt (do odczytu po impulsie `RxRdy`),
- `TxBusy` – informacja o tym, że moduł jest zajęty i nie może przyjąć bajtu do nadania („1” – zajęty, „0” – gotowy do nadawania).

Moduł składa się z dwóch procesów, nazwanych odpowiednio `nadawanie` oraz `odbieranie`. Są to maszyny stanów, które przechodząc przez stany `st_gotowy`, `st_bit_startu`, `st_dane` oraz `st_bit_stopu` odpowiednio nadają lub odbierają bajt danych.

3.1.3 Moduł frameInformer

Moduł **frameInformer** jest pomocnikiem modułu **obrazek**. Przyjmuje odebrane bajty z modułu **rs232**, przetwarza je na uproszczoną informację i wysyła do rzeczonego modułu. Dodatkowo generuje jednotaktowy impuls co $\frac{1}{30}$ sekundy, aby poinformować moduł o konieczności przeliczenia pozycji obiektów na ekranie.



Rysunek 8: Symbol modułu **frameInformer**.

Listing 2: Definicja modułu **frameInformer**.

```
5  entity frameInformer is
6     Generic ( cykle_na_ramke: integer := 1666667 );
7
8     Port (Clk_50MHz: in  STD_LOGIC;
9           Byte_Rdy:  in  STD_LOGIC;
10          ByteIn:    in  STD_LOGIC_VECTOR(7 downto 0);
11          Reset:     in  STD_LOGIC;
12          newFrame:  out STD_LOGIC                    := '0';
13          button:    out STD_LOGIC_VECTOR(1 downto 0) := "00");
14 end frameInformer;
```

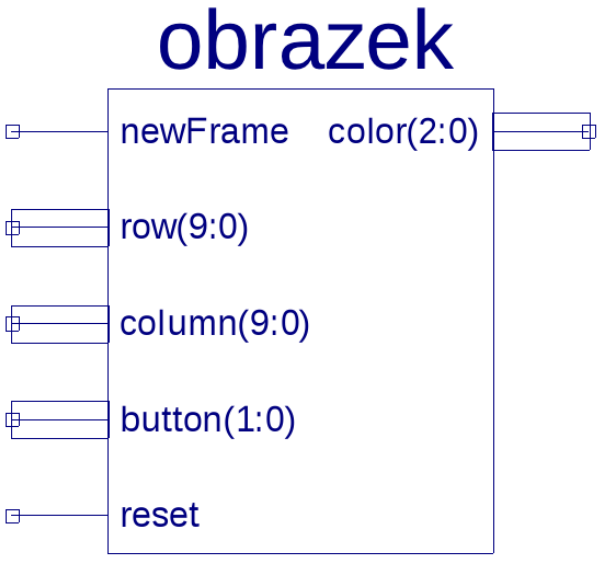
Wejścia i wyjścia tego modułu oznaczają:

- `Clk_50MHz` – podłączenie zegara 50 MHz,
- `Byte_Rdy` – podłączenie do wyjścia `Byte_Rdy` modułu **rs232**,
- `ByteIn` – podłączenie do wyjścia `RxDO` modułu **rs232**,
- `Reset` – podłączenie do linii resetowania układu,
- `newFrame` – jednotaktowy impuls, nadawany co $\frac{1}{30}$ s, informujący o konieczności przeliczenia pozycji elementów na obrazie,
- `button` – wyjście przekształcające odebrane bajty na informację o wciśniętym klawiszu na klawiaturze RS232 według reguły:
  - bity `00` – brak wciśniętego klawisza,
  - bity `01` – przycisk „w prawo”,
  - bity `10` – przycisk „w lewo”,
  - bity `11` – przycisk „strzał”.

Moduł składa się z dwóch procesów: `nowyZnak`, który co takt zegara sprawdza, czy moduł **rs232** odebrał bajt, i jeśli tak, przetwarza go i wystawia według wskazanej wyżej reguły, oraz `liczenie`, który odlicza odpowiednią ilość taktów zegara 50 MHz (dokładnie 16666667 taktów) i wysyła jednotaktowy impuls na wyjście `newFrame`.

3.1.4 Moduł obrazek

Moduł **obrazek** jest głównym modułem obsługującym logikę omówioną w rozdziale 1.2.



Rysunek 9: Symbol modułu **obrazek**.

Listing 3: Definicja modułu **obrazek**.

```
5  entity obrazek is
6      Port (reset:    in  STD_LOGIC;
```

```
7             row:       in  STD_LOGIC_VECTOR(9 downto 0);
8             column:    in  STD_LOGIC_VECTOR(9 downto 0);
9             newFrame:  in  STD_LOGIC;
10            button:    in  STD_LOGIC_VECTOR(1 downto 0);
11            color:     out STD_LOGIC_VECTOR(2 downto 0));
12 end obrazek;
```

Wejścia i wyjścia tego modułu oznaczają:

- `newFrame` – wejście impulsu informującego o konieczności przeliczenia pozycji obiektów na ekranie (wyjście `newFrame` modułu **frameInformer**),
- `row`, `column` – wejścia informujące o pozycji aktualnie wyświetlanego na ekranie piksela (odcięta, rzędna; wyjścia `row`, `column` modułu **vga_controller**),
- `button` – wejście informujące o wciśniętym na klawiszu,
- `reset` – wejście linii asynchronicznego resetu układu,
- `color` – wyjście podające kolor do wyświetlenia w danym pikselu ekranu (barwy: czerwona, zielona, niebieska; składowa koloru używana („1”) lub nie („0”)).

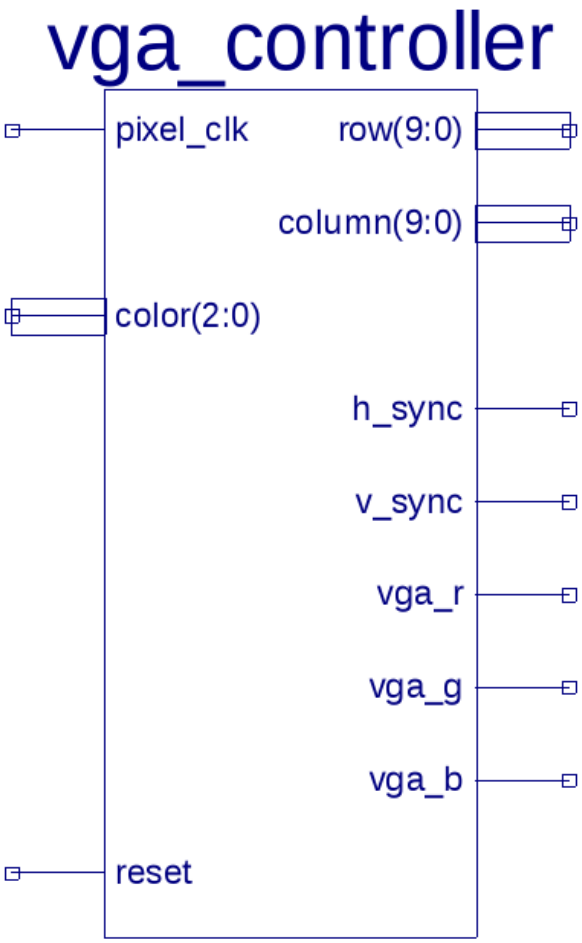
Kolory możliwe do wyświetlenia (wartość wyjścia `color`) to:

- `"000"` – czarny,
- `"001"` – niebieski,
- `"010"` – zielony,
- `"011"` – cyjan,
- `"100"` – czerwony,
- `"101"` – karmazynowy (magenta),
- `"110"` – żółty,
- `"111"` – biały.

Na moduł składa się dwa procesy: `recalculate`, którego zadaniem jest przeliczanie pozycji obiektów (gracza, kuli, przeciwnika) na ekranie według reguł mechaniki gry, oraz `output`, który podaje informacje o kolorze, który należy wyświetlić w danym miejscu ekranu, na podstawie zapamiętanych w module położeń obiektów na planszy.

3.1.5 Moduł vga_controller

Moduł **vga_controller** jest sterownikiem odpowiedzialnym za poprawne wyświetlanie obrazu na ekranie.



Rysunek 10: Symbol modułu **vga_controller**.

Listing 4: Definicja modułu **vga_controller**.

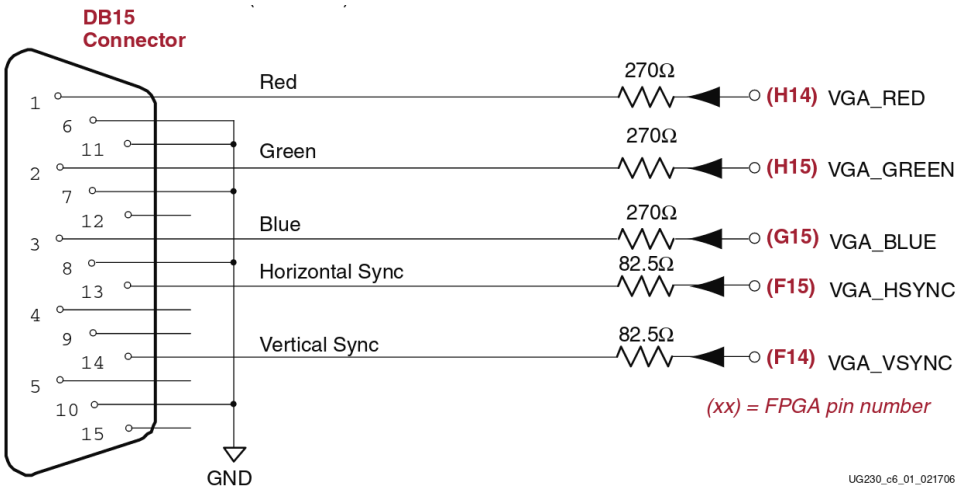
```
5  entity vga_controller is
6     Generic(
7        h_display:  INTEGER   := 1024;
8        h_fp:       INTEGER   := 24;
9        h_pulse:    INTEGER   := 136;
10       h_bp:       INTEGER   := 160;
11       h_polarity: STD_LOGIC := '0';
12
13       v_display:  INTEGER   := 768;
14       v_fp:       INTEGER   := 3;
15       v_pulse:    INTEGER   := 6;
16       v_bp:       INTEGER   := 29;
17       v_polarity: STD_LOGIC := '0'
18    );
19    Port(
20       reset:      in STD_LOGIC;
21       color:      in STD_LOGIC_VECTOR(2 downto 0);
22       pixel_clk: in STD_LOGIC;
23
24       row:        out STD_LOGIC_VECTOR(9 downto 0) := (others =>
```

```
            '0');
25        column:     out STD_LOGIC_VECTOR(9 downto 0) := (others =>
            '0');
26        h_sync:     out STD_LOGIC                    := h_polarity;
27        v_sync:     out STD_LOGIC                    := v_polarity;
28        vga_r:      out STD_LOGIC                    := '0';
29        vga_g:      out STD_LOGIC                    := '0';
30        vga_b:      out STD_LOGIC                    := '0'
31    );
32  end vga_controller;
```

Wejścia i wyjścia tego modułu oznaczają:

- `pixel_clk` – zegar pikseli (patrz: rozdział 1.3),
- `color` – wartość koloru do wyświetlenia w podawanym miejscu ekranu,
- `reset` – linia asynchronicznego resetu układu,
- `row`, `column` – magistrale podające informację o aktualnie wyświetlanym pikselu ekranu, aby możliwe było odebranie odpowiedniego koloru do wyświetlenia na ekranie,
- `h_sync` – linia synchronizacji poziomej,
- `v_sync` – linia synchronizacji pionowej,
- `vga_r` – linia dla składowej czerwonej obrazu,
- `vga_g` – linia dla składowej zielonej obrazu,
- `vga_b` – linia dla składowej niebieskiej obrazu.

Spartan-3E posiada port VGA, przez który dostarcza sygnał dla monitora przez rezystory: 270 Ω dla linii koloru, 82.5 Ω dla linii synchronizacji. Konfigurację tę przedstawia rysunek 11.

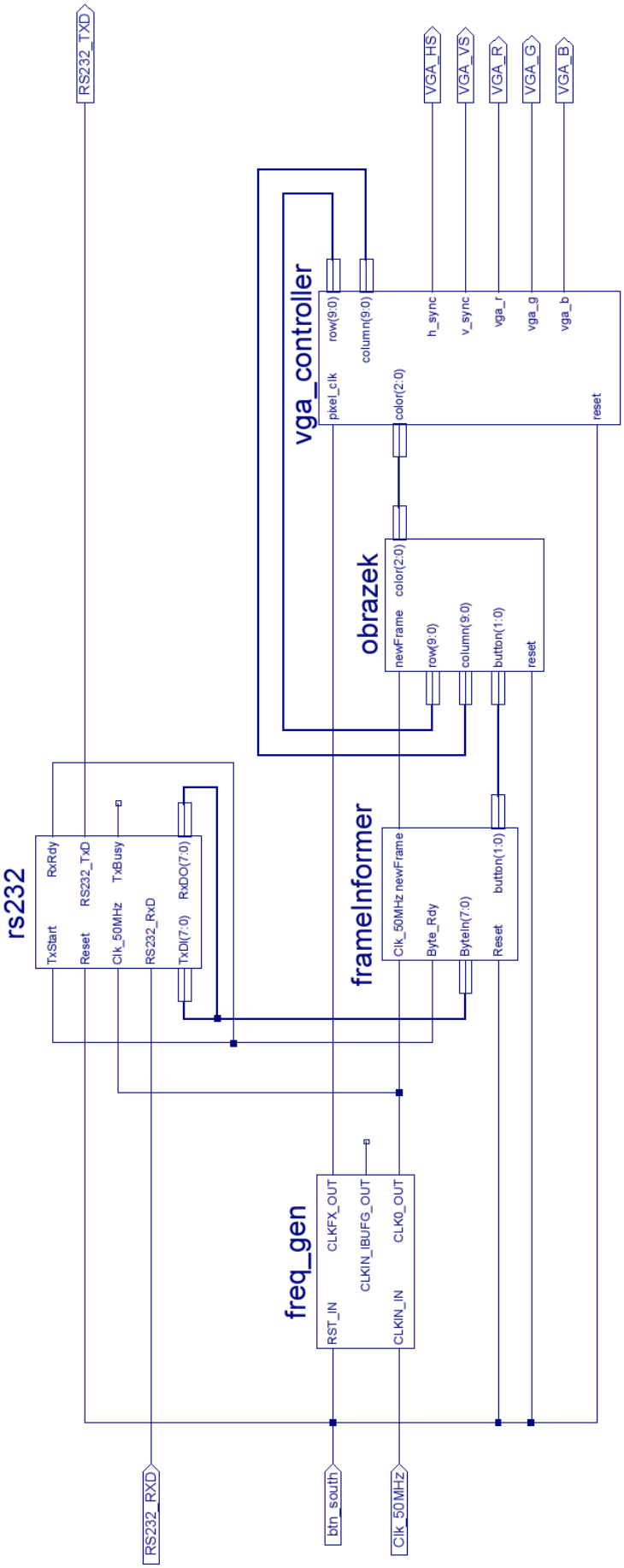


Rysunek 11: Wyprowadzenia VGA w płytce Spartan-3e. Źródło: [2].

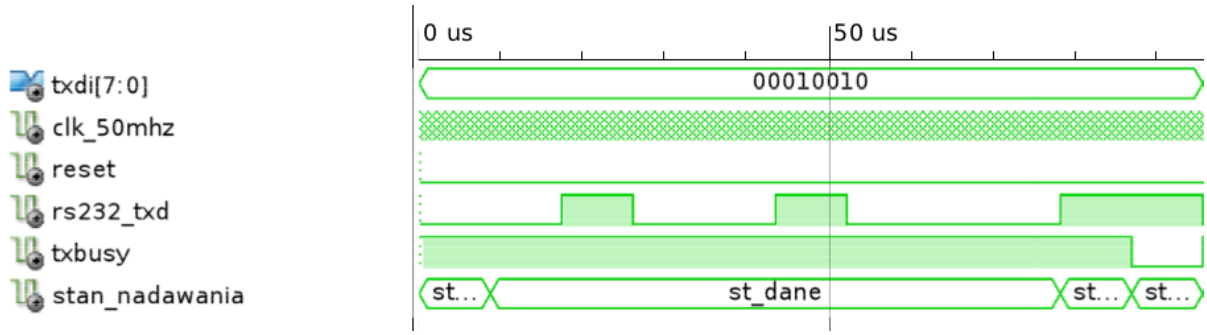
Moduł ten składa się z jednego, nienazwanego procesu, implementującego proces wyświetlania obrazu na ekranie monitora, na podstawie schematu przedstawionego na rysunku 1.

3.2 Schemat projektu

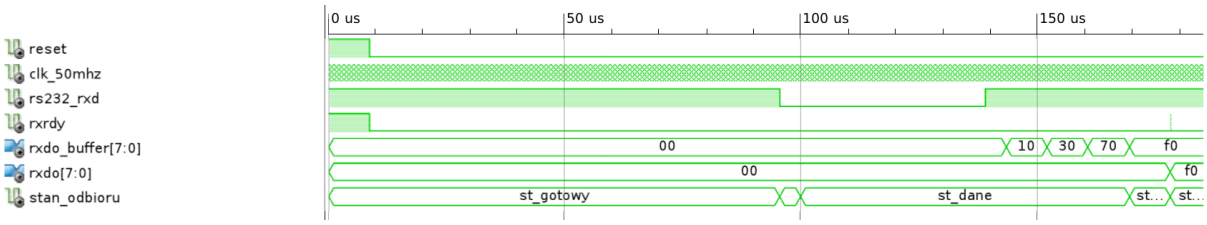
Rysunek 12 przedstawia schemat niniejszego projektu.



Rysunek 12: Schemat projektu.



Rysunek 13: Fragment symulacji nadawania RS232.



Rysunek 14: Fragment symulacji odbierania RS232.

4 Testowanie

4.1 Symulacja w ISim

W projekcie przygotowano dwie symulacje dla modułu **rs232**: nadawania i odbierania.

4.1.1 RS232: nadawanie

Na rysunku 13 przedstawiono fragment symulacji nadawania bajtu `0x12`. Opis sygnałów widocznych w symulacji opisano w rozdziale 3.1.2.

Nadawanie sygnału polega na wysłaniu ich na linii `RS232_TxD` w kolejności od najmłodszego do najstarszego, poprzedzając je bitem startu, oraz wysyłając po nich bit stopu. Na czas transmisji linia `TxBusy` wchodzi w stan wysoki, po zakończeniu – w stan niski.

4.1.2 RS232: odbieranie

Na rysunku 14 przedstawiono fragment symulacji odbierania bajtu `0xF0`. Opis sygnałów widocznych w symulacji opisano w rozdziale 3.1.2.

Odbieranie rozpoczyna się po wykryciu zbocza opadającego na linii `RS232_RxD` – układ przechodzi wówczas do stanu `st_bit_startu`. W nim, po czasie połowy cykli transmisji bitu sprawdzane jest, czy sygnał na pewno jest nadawany, ponownie weryfikując niski stan linii, a następnie rozpoczyna cykle oczekiwania i próbkowania odbieranego sygnału. Spróbkowane wartości są przechowywane w sygnale wewnętrznym `RxDO_buffer`, którego zawartość jest wystawiana w stanie `st_bit_stopu` na magistralę `RxDO` wraz z jednotaktowym impulsem na linii `RxRdy` oznaczającym odebranie bajtu i gotowość do jego odczytu.

5 Instrukcja użytkownika

Po zaprogramowaniu płytki Spartan-3e przez wykwalifikowany personel, można korzystać z gry. Jej obsługa odbywa się przy użyciu klawiatury podłączonej do portu RS232. Jeśli użytkownik nie dysponuje taką klawiaturą, jak ma to miejsce w laboratorium, należy użyć aplikacji terminala portu szeregowego, która będzie nadawać wpisywane na klawiaturze komputera znaki i odbierać znaki wysyłane przez układ.

Aby poruszyć graczem w lewo, należy nacisnąć klawisz **a**. Aby poruszyć graczem w prawo – klawisz **d**. Strzał można oddać klawiszem **spacji**.

Rysunek 15: Rysunek poglądowy przycisków sterowania na klawiaturze.

6 Podsumowanie i wnioski

Z pewnością można stwierdzić, że programowanie układów w języku opisu sprzętu, jakim jest VHDL, jest zadaniem niebanalnym, ponieważ nie możemy korzystać jedynie z abstrakcji znanych z języków takich jak C++ czy Rust, lecz musimy zdawać sobie sprawę z tego, jaki układ powstanie z syntezy tworzonego kodu.

Implementacja modułu RS232 okazała się ostatecznie nieskomplikowana po zwróceniu uwagi przez prowadzącego na to, że po weryfikacji bitu startu wystarczy odczekiwać tyle cykli zegara, ile ma trwać transmisja jednego bitu, a następnie próbkować sygnał na linii transmisyjnej. W ten sposób zawsze trafiamy „w środek" transmisji danego bitu i ograniczamy do minimum szansę spróbkowania złego bitu.

Ważnym aspektem okazało się takie stworzenie układu RS232, aby wyeliminować szansę spróbkowania niewłaściwego bitu, zapewniając stabilność układu. W tym celu wprowadzono dodatkowy sygnał, który zawierał stan linii transmisyjnej, który następnie przypisywano do sygnału testowanego (patrz: listing 5). W ten sposób, choć opóźniliśmy o jeden takt zegara odczyt linii, upewniliśmy się, że nie dojdzie do przesłuchów.

Listing 5: Fragment implementacji modułu **rs232**.

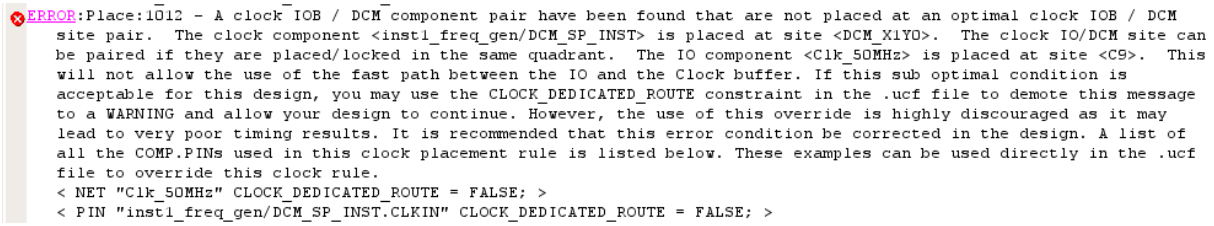
```
98    odbieranie: process(Clk_50MHz, Reset)
99    begin
100      if(Reset = '1') then
101         stan_odbioru          <= st_gotowy;
102         RxRDY                 <= '1';
103         licznik_taktow_odbior <= 0;
104         licznik_bitow_odbior  <= 0;
105         RxDO                  <= (others => '0');
106         RxDO_buffer           <= (others => '0');
107
108      elsif(rising_edge(Clk_50MHz)) then
109         RS232_RxD_stable     <= RS232_RxD;
110         RS232_RxD_stable_pre <= RS232_RxD_stable;
111
112         case stan_odbioru is
```

6.1 Propozycje rozbudowy układu

Ponieważ nie udało się wprowadzić losowania wartości odciętej przeciwnika, warto stworzyć moduł lub rozbudować jeden z istniejących, aby umożliwić ten zabieg. Autorzy niniejszej pracy próbowali wprowadzić takie rozwiązanie na dwa sposoby:

- wprowadzić proces liczący od 0 do wartości szerokości ekranu (-1), który byłby zwiększany z każdym taktem zegara,
- wprowadzić rejestr o sprzężeniu liniowym (na podstawie [4, str. 71]).

Niestety, układ ten, o ile się syntezował, wywoływał błąd procesu *Place & Route* (patrz: rysunek 16), lub proces ten generował ostrzeżenie o niespełnieniu wymagania synchronizacji zegarowej.



Rysunek 16: Błąd niespełnionej zależności zegarowej i braku „szybkiej ścieżki” zegara.

Nie udało się także wprowadzić poprawnej reakcji na kolizje przeciwnika z pociskiem ani przeciwnika z graczem. Należy wprowadzić odpowiednie zmiany w module **obrazek**.

Bibliografia

[1] Xilinx Inc. *Clocking Wizard*. URL: https://www.xilinx.com/products/intellectual-property/clocking_wizard.html (visited on 06/13/2019).

[2] Xilinx Inc. *Xiling UG230 Spartan-3E FPGA Starter Kit Board User Guide, v1.2*. Jan. 20, 2011. URL: https://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf (visited on 06/13/2019).

[3] Digi-Key Corp. Scott Larson. *VGA Controller (VHDL) - Logic - eewiki*. Mar. 7, 2018. URL: https://www.digikey.com/eewiki/pages/viewpage.action?pageId=15925278 (visited on 06/13/2019).

[4] Jacek Majewski; Piotr Zbysiński. *Układy FPGA w przykładach*. 1st ed. Wydawnictwo BTC, 2007.

Spis rysunków