Data: 01.06.2021

Politechnika Wrocławska

Układy Cyfrowe i Systemy Wbudowane 2

Dokumentacja projektu:

Organy generujące fale piłokształtne z wyświetlaniem informacji na kanale tekstowym

---

Skład grupy:

Artur Sobolewski 248913

Przemysław Rychter 248820

Prowadzący:

Dr inż. Jarosław Sugier

1. Wprowadzenie

1.1. Cel projektu

Celem projektu było wykonanie układu realizującego organki generujące dźwięki w postaci fal piłokształtnych z zakresu jednej oktawy. Obsługa urządzenie miała odbywać się za pomocą klawiatury wysyłającej sygnały przez interfejs PS/2. Miało to pozwolić generować odpowiednie tony w zależności od naciśniętego klawisza. Za odtwarzanie dźwięków odpowiadałby podłączony do płyty głośniczek. Dodatkową formą komunikacji z użytkownikiem ma być wyświetlanie informacji dotyczącej wysokości i długości trwania dźwięku na kanale tekstowym VGA.

1.2. Wykorzystany sprzęt i narzędzia

Docelowym sprzętem do zaprogramowania była płyta FPGA Spartan-3E Starter Kit. Najważniejszą właściwością tej płyty, która ma wpływ na sposób generowania dźwięków o odpowiednich częstotliwościach jest oscylator zegarowy pracujący z częstotliwością 50 *MHz*, co daje długość okresu 20 *ns*. Kolejnymi elementami płyty Spartan-3E niezbędnymi do realizacji projektu są 2 porty PS/2 przeznaczone dla klawiatury lub myszy (wykorzystany zostanie tylko jeden z nich) oraz port VGA. Ponadto posiada ona też inne komponenty przydatne pod kątem tworzonego układu: moduł DAC – czterokanałowy konwerter sygnału cyfrowego na analogowy [1]. Wykorzystywanym środowiskiem było oprogramowanie Xilinx ISE, które pozwala na realizację kodu w języku opisu sprzętu VHDL oraz jego kompilację dostosowaną do podanej wcześniej płyty i końcowo wykonywać symulacje układu. Do projektu zostały przewidziane dodatkowe urządzenia: klawiatura wyposażona w interfejs PS2, głośniczek oraz wyświetlacz odbierający informacje z portu VGA.

1.3. Wstęp teoretyczny

W ramach projektu z portu PS/2 przy podłączonej klawiaturze zostaną użyte 2 magistrale tego interfejsu: PS2_DATA oraz PS2_CLK. Obie wykorzystują 11-bitowe słowa zwierające znaczniki startu, stopu, bit parzystości. Pierwsza z nich przesyła dane dotyczące naciskanych klawiszy: kod skanowania klawisza, kod czy klawisz jest wciśnięty, czy zwolniony [2].

Modułem, który posłużył do konwersji danych cyfrowych dźwięków na ich analogową reprezentację jest moduł DAC [3]. Zapewnia on 4 kanały wyjściowe do wyboru. Komunikacja z modułem odbywa się za pomocą szeregowego interfejsu peryferyjnego (SPI). W ramach projektu najważniejszym sygnałem jest SPI_MOSI, za pomocą, którego przekazywane są wystarczające informacje definiujące charakterystykę fali dźwiękowej jaka ma być wygenerowana. Jest to 32-bitowe słowo. 4 najniższych i 8 najwyższych bitów są nieistotne, natomiast ze wzrostem pozycji kolejnych bitów: 12-bitowy ciąg jest wartością, od której zależy chwilowa wartość napięcia składająca się na falę dźwiękową; 4-bitowy ciąg określający docelowy kanał wyjściowy modułu DAC; i końcowo 4-bitowy ciąg wskazujący tryb pracy

[1] Spis komponentów i funkcjonalności płyty Spartan-3E
XILINX. *Spartan-3E FPGA Starter Kit Board User Guide*.

[2] Opis protokołu interfejsu PS/2 z klawiaturą
XILINX. *Spartan-3E FPGA Starter Kit Board User Guide*.

[3] Opis modułu DAC
XILINX. *Spartan-3E FPGA Starter Kit Board User Guide*.

modułu – zazwyczaj jest to wartość „0011”, która oznacza automatyczne odświeżenie sygnału wyjściowego po otrzymaniu nowych danych.

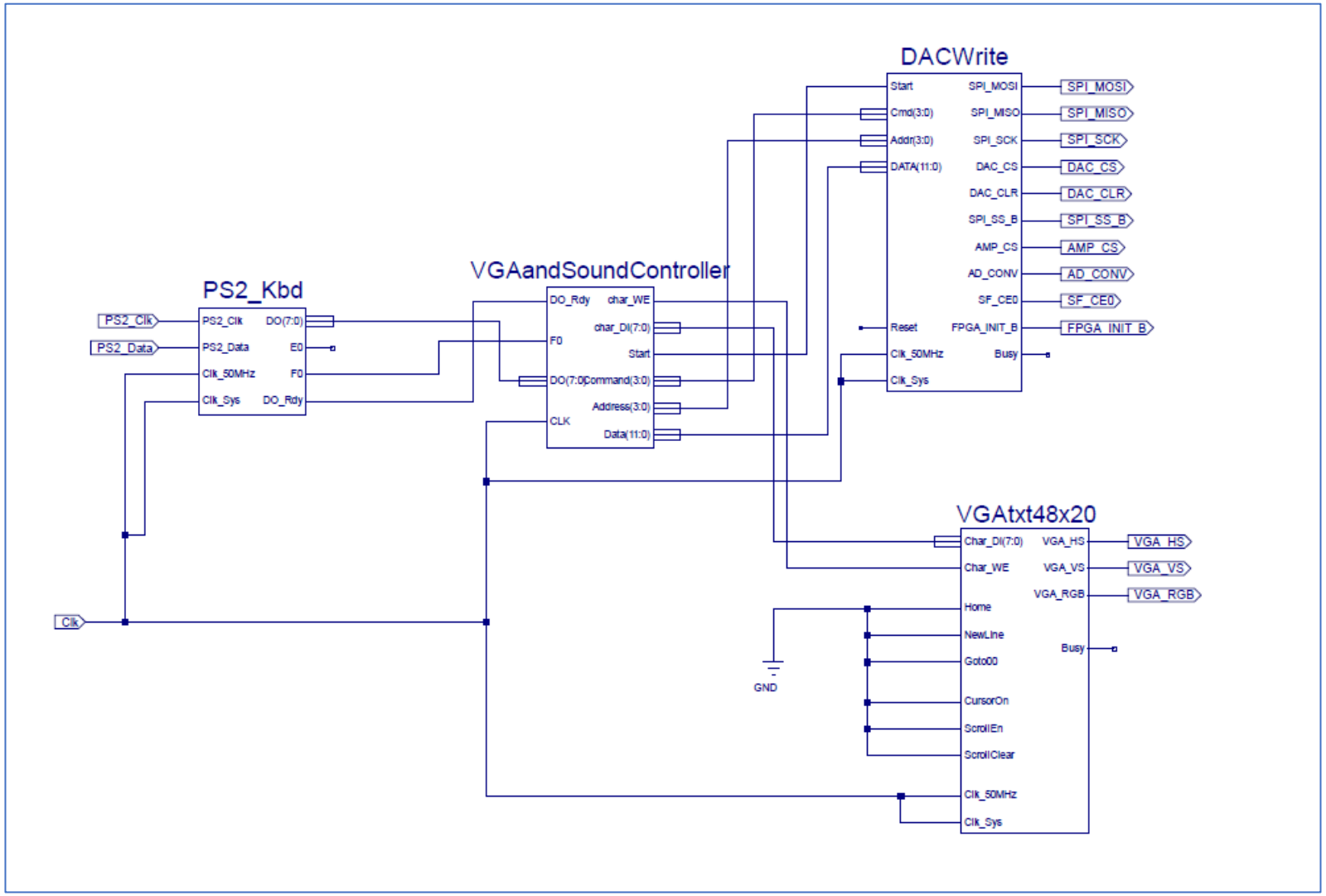
2. Przedstawienie układu

2.1. Struktura ogólna

Przedstawiona na poniższym rysunku struktura ogólna układu reprezentuje, główne funkcje projektu. Użytkownik naciska dany klawisz na klawiaturze, kod klawisza jest konwertowany na daną liczbę, która reprezentuje daną nutę. Liczba ta, czyli “kod nuty” jest następnie przesyłana do modułu, który obsługuje generacje dźwięku, oraz do modułu obsługującego wyświetlacz.

Rysunek 1 Koncepcja działania układu

2.2. Schemat szczytowy

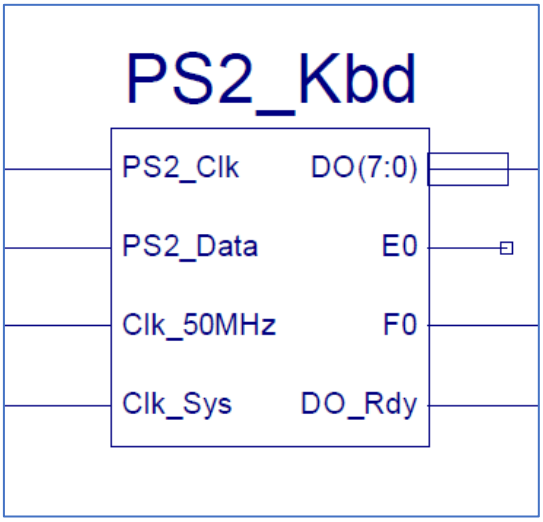


Rysunek 2 Schemat szczytowy układu

2.3. Moduły składowe

2.3.1. PS2_Kbd

Moduł został pobrany ze strony kursu [4] Jest to odbiornik kodów wysyłanych przez klawiaturę PS/2.



Rysunek 3 Symbol modułu PS2_Kbd

Szczegółowy opis modułu jest dostępny na stronie kursu

Wejścia:

- PS2_Clk – dane kontrolujące przesyłanie danych z klawiatury,
- PS2_Data – dane dotyczące naciskania klawiszy,
- Clk_50MHz, Clk_sys – sygnały zegarowe koordynujące pracę modułu.

Wyjścia:

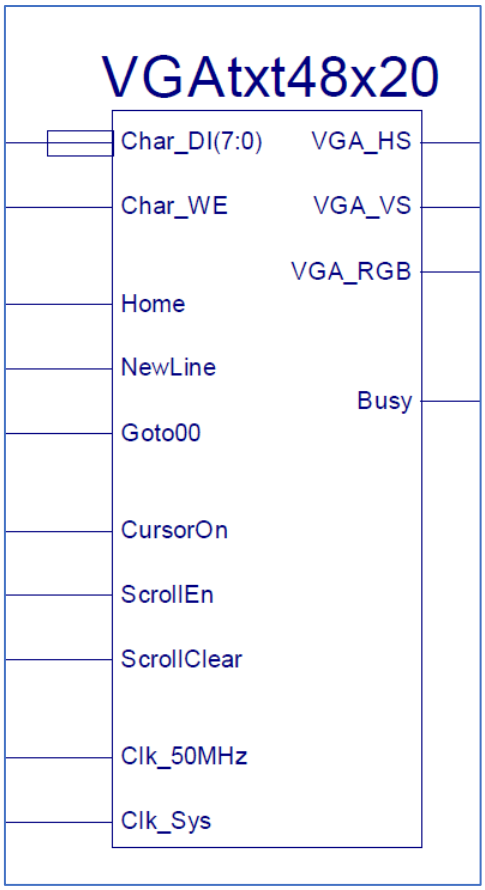
- DO(7:0) - bajt reprezentujący kod wciśniętego klawisza,
- F0 - sygnał reprezentujący zwolnienie klawisza,
- DO_Rdy - sygnał informujący o zakończeniu odbioru kodu klawisza.

2.3.2. VGAtxt48x20

Moduł został pobrany ze strony kursu [5]. Został użyty do obsługi wyświetlacza, za którego pomocą zostaje przestawiona jaka nuta była odegrana, oraz jak długo ona trwała.

[4] Moduł PS2_Kbd udostępniony przez prowadzącego Dr inż. J. Sugiera
Sugier, J. *Moduły pomocnicze do Spartan-3E Starter Kit*

[5] Moduł VGAtxt48x20 udostępniony przez prowadzącego Dr inż. J. Sugiera
Sugier, J. *Moduły pomocnicze do Spartan-3E Starter Kit*



Rysunek 4 Symbol modułu VGAtxt48x20

Wejścia:

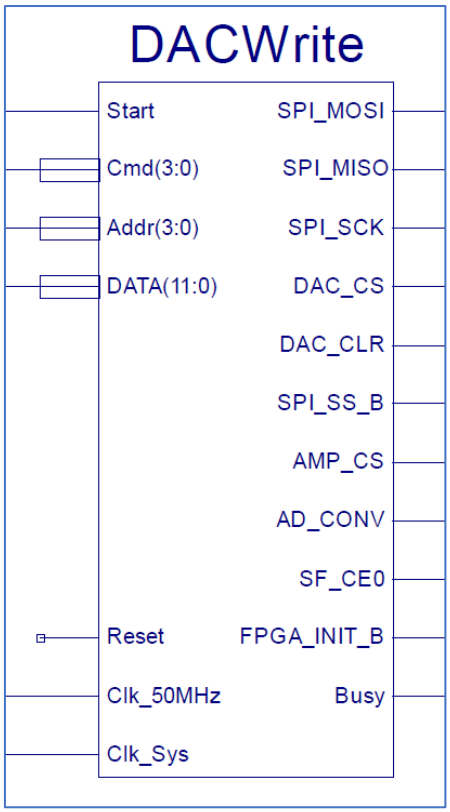
- Char_DI(7:0) - reprezentuje bajt znaku jaki ma być wyświetlony,
- Char_WE – informuje o tym, że w aktualnym takcie zegara, należy wyświetlić symbol o kodzie podanym na Char_DI(7:0).
- Clk_50MHz, Clk_Sys - sygnały zegarowe koordynujące pracę modułu.
- Pozostałe wejścia kontrolują sposób wyświetlania w zależności od otrzymanych wartości sygnału.

Wyjścia:
VGA_HS, VGA_VS, VGA_RGB – wyjścia sterujące modułem VGA.

2.3.3. DACWrite

Moduł został pobrany ze strony kursu [6]. Obsługuje wysyłanie danych do przetwornika cyfra/analog LTC2624. Interpretuje on sygnały podane na wejściu, które precyzują działanie modułu DAC, dostosowany do naszego projektu. Następnie przygotowywany jest ciąg sygnałów wyjściowych dostosowany do komunikacji za pomocą protokołu SPI. W ten sposób moduł DACWrite udostępnia interfejs do komunikacji pomiędzy naszym modułem VGAandSoundController a modułem DAC.

[6] Moduł DACWrite, udostępniony przez prowadzącego Dr inż. J. Sugiera
Sugier, J. *Moduły pomocnicze do Spartan-3E Starter Kit*



Rysunek 5 Symbol modułu DACWrite

Wejścia:

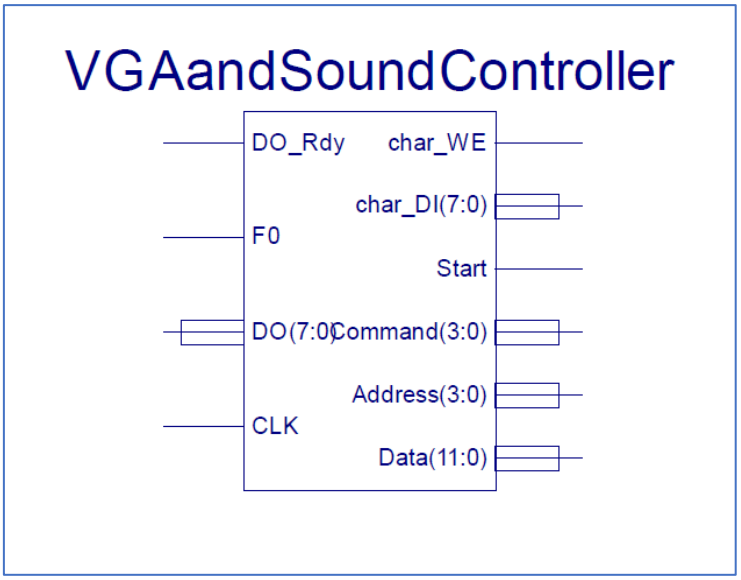
- Start – impuls jednotaktowy, który powoduje pobrania danych podanych na wejścia *Cmd*, *Addr* i *DATA*. Wejścia te zostały szerzej opisane we wstępie,
- Cmd – Koordynacja trybem pracy modułu DAC,
- Addr – określenie kanału wyjściowego modułu DAC,
- DATA – słowo binarne określające wartość sygnału analogowego.

Wyjścia:

Sygnały komunikujące się z modułem DAC.

2.3.4. VGAandSoundController

Jest to główny moduł naszego projektu, który składa się z 3 podmodułów. W następnych punktach opiszemy poszczególne podmoduły. Odpowiada on za przetworzenie sygnałów wejściowych, które pochodzą z modułu PS2_Kbd i dostarczają informacje na temat akcji wykonywanej na dołączonej klawiaturze. Na ich podstawie sprawdzane jest jaki klawisz został naciśnięty i jakiemu dźwiękowi on odpowiada, po czym generowana jest cyfrowa reprezentacja fali dźwiękowej do wygenerowania oraz znak reprezentujące generowany ton i ostateczne sekwencję znaków informująca o długości trwania dźwięku lub ciszy.



Rysunek 6 Symbol modułu VGAandSoundController

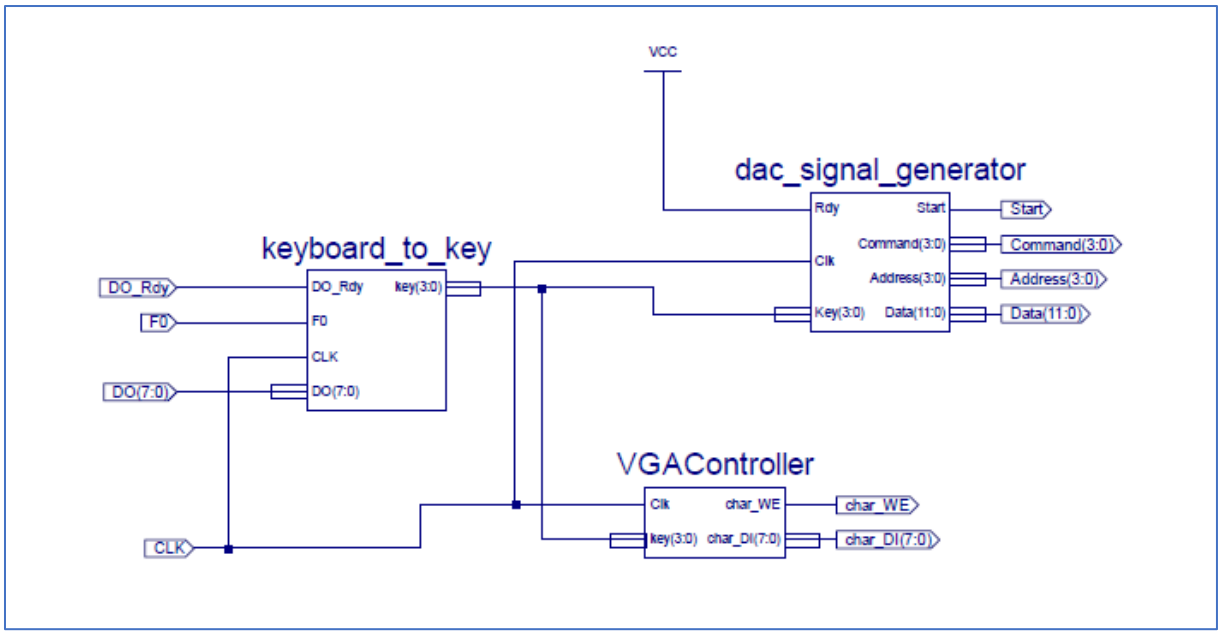
Wejścia:

- DO_Rdy – impuls jednotaktowy, powodujący pobranie danych z pozostałych wejść,
- F0 – sygnał informujący o wciśnięciu i zwolnieniu klawisza,
- DO – bajt reprezentujący naciśnięty klawisz w kodzie hexadecymanym zapisanym binarnie,
- CLK – globalny sygnał zegarowy.

Wyjścia:

char_WE, char_DI, Start – sygnały przekazywane do modułu VGAtxt48x20,

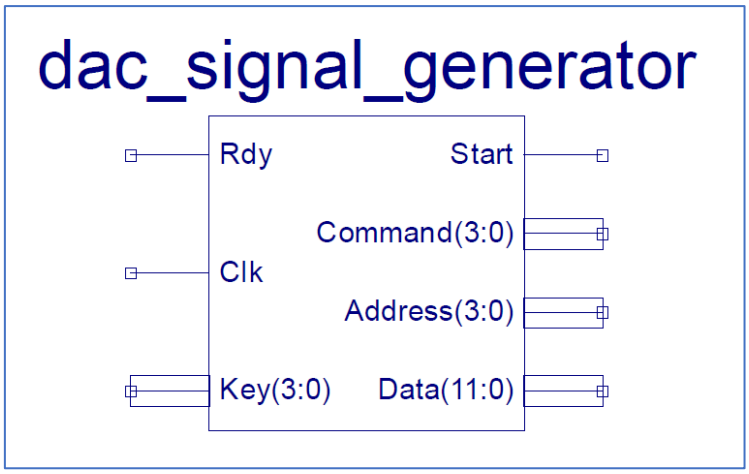
Command, Address, Data – sygnały przekazywane do modułu DACWrite.



Rysunek 7 Schemat modułu VGAandSoundController

2.3.5. dac_signal_generator

Moduł dac_signal_generator odpowiada za generowanie fali piłokształtnej i sygnałów sterujących, w formacie dostosowanym do danych jakie przyjmuje modułu DACWrite (fala - wartości liczb całkowitych w postaci wektora 12-bitoweg). Wartości przyjmowane na wektorze Key określają częstotliwość fali, jaka ma być generowana. Praca tego modułu dostosowana jest do częstotliwości z jaką pracuje wykorzystana płyta Spartan – 50 MHz – w oparciu o nią dobierane są opóźnienia w zmianie wartości wychodzącej, dzięki czemu otrzymujemy określone częstotliwości fal.



Rysunek 8 Symbol modułu generującego cyfrowe fale piłokształtne

Wejścia:

- Rdy – pełni rolę sygnału sterującego, które powoduje pracę lub spoczynek modułu,
- Clk – sygnał zegarowy,
- Key(3:0) – wartości określające częstotliwość fali do wygenerowania,

Wyjścia:

- Start – impuls wysyłany do modułu DACWrite – informujący by ten przyjął przesyłane dane,
- Command(3:0) – wektor określający pracę modułu DACWrite,
- Address(3:0) – wektor określający, na który port wyjściowy ma być wysyłany sygnał analogowy z modułu DACWrite,
- Data(11:0) – wektor, którego określa wartość sygnału w danym czasie.

Opis realizacji modułu:

Na samym początku architektura modułu dac_signal_generator definiuje zmienne:

- clks_delay – liczba określające ile taktów zegara należy odczekać pomiędzy kolejnymi zmianami wartości sygnału (bazowa wartość 999 – pozwala uzyskać częstotliwość równą 1 kHz). Konkretne częstotliwości są możliwe do uzyskania dzięki kontrolowaniu częstotliwości inkrementacji wartości przechowywanej w zmiennej var – przed każdą inkrementacją proces odczekuje zadaną liczbę taktów, których ilość jest zależna od wartości na wejściu Key. Zależność ta została stabelaryzowana w instrukcji with-select.
- clk_counter – zmienna, która realizuje zliczanie taktów
- var – zmienna inkrementowana, której wartość przekazywana jest na wyjście Data.

W procesie został zaimplementowany licznik modulo działający w maksymalnym zakresie liczb całkowitych 12-bitowych. Wartość zwiększana jest o dziesiętną wartość 82, po zliczeniu wymaganej liczby okresów. Zależność liczby taktów zegara do odczekania od częstotliwości fali, jaka ma zostać wygenerowana wyznaczana była następująco:

Jeśli długość taktu sygnału zegarowego płyty: $T_p = 20\ [ns] = 2 * 10^{-8}\ [s]$, liczba kolejnych wartości składających się na jeden okres fali piłokształtnej: $L_T = 50$, to długość okresu fali piłokształtnej:

$$T = L_T * T_p * (x + 1),$$

Gdzie:

T – długość okresu,

x – liczba taktów zegara, jaką należy odczekać (we wzorze występuje + 1, ponieważ jest to dodatkowy takt potrzebny na inkrementację zmiennej)

Następnie jeśli: $f = \frac{1}{T}$ $(zależność\ częstotliwości\ od\ długości\ okresu)$ to po przekształceniach otrzymujemy, zależność od częstotliwości:

$$x = \frac{1 - f L_T T_p}{f L_T T_p}$$

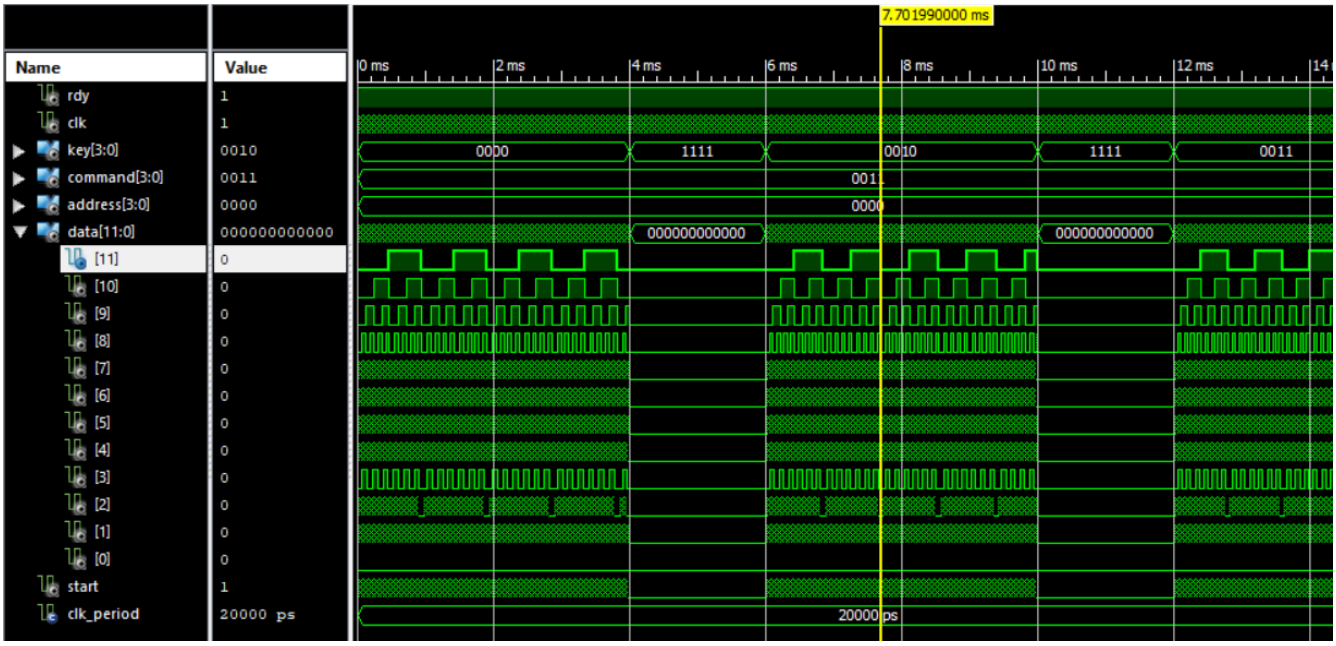
```
entity dac_signal_generator is

    Port ( Rdy : in  STD_LOGIC;
           Clk : in  STD_LOGIC;
              Key : in STD_LOGIC_VECTOR (3 downto 0);
           Command : out  STD_LOGIC_VECTOR (3 downto 0);
           Address : out  STD_LOGIC_VECTOR (3 downto 0);
           Data : out  STD_LOGIC_VECTOR (11 downto 0);
           Start : out  STD_LOGIC);
end dac_signal_generator;

architecture Behavioral of dac_signal_generator is

    signal clks_delay : integer := 999;
    signal clk_counter : integer := 0;
    signal var : unsigned (11 downto 0) := X"000";
begin

    with Key select clks_delay <=
        955 when "0000",     -- 1046.50 c         [ HZ ]
        901 when "0001",     -- 1108.73 cis
        850 when "0010",     -- 1174.66 d
        802 when "0011",     -- 1244.51 dis
        758 when "0100",     -- 1318.51 e
        715 when "0101",     -- 1396.91 f
        675 when "0110",     -- 1479.98 fis
        637 when "0111",     -- 1567.98 g
        601 when "1000",     -- 1661.22 gis
        567 when "1001",     -- 1760.00 a
        535 when "1010",     -- 1864.66 b
        505 when "1011",     -- 1975.53 h
        0    when others;

    Command <= "0011";
    Address <= "0000";

    process(Clk)
    begin

        if rising_edge(Clk) and Rdy = '1' then
            if clks_delay /= 0 then --
                if clk_counter < clks_delay then
                    clk_counter <= clk_counter + 1;
                    Start <= '0';
                else
                    Start <= '1';
                    if var < X"FAD" then
                        var <=  var + 82;
                    else
                        var <= X"000";
                    end if;
                    clk_counter <= 0;
                end if;
            else
                var <= X"000";
                clk_counter <= 0;
            end if;
        end if;

    end process;

    Data <= STD_LOGIC_VECTOR(var);

end Behavioral;
```

Fragment 1 Kod VHDL modułu dac_signal_generator

Symulacja:

Testbench w pętli podaje się na wejście *key* kolejne wartości kodu odpowiadające generowanemu tonowi na przemian z kodem ciszy. Na symulacji widać, że moduł w momencie wystąpienia kodu różnego od „1111” stopniowo zmieniają się wartości *data* do pewnej wartości, po czym inkrementacja odbywa się od zera. Został przedstawiony tylko fragment przedstawiający generowanie 3 dźwięków na przemian z ciszą. By zmierzyć okres fali każdego dźwięku ustawiano wskaźnik w momencie maksymalnej wartości dźwięku i w najbliższym takim momencie. Następnie obliczone zostały odwrotności różnic czasów, otrzymując realną częstotliwość dźwięku jaki został wygenerowany. Przykładowe obliczania:

$$f = \frac{1}{8{,}55299 - 7{,}70199} * 10^3 = 1175\ [Hz]$$



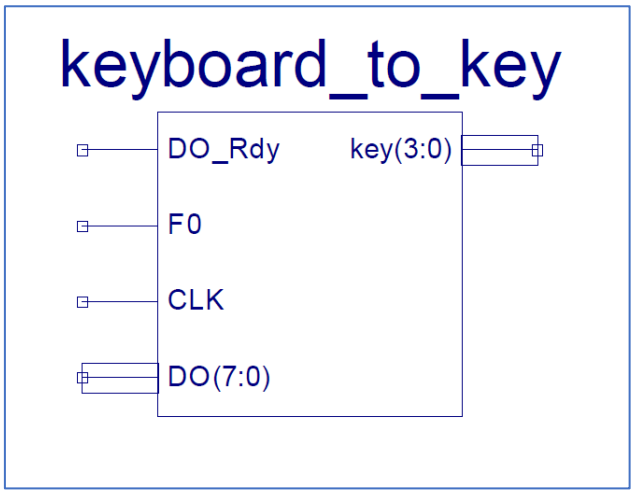
Rysunek 10 Przebieg czasowy symulacji modułu dac_signal_generator

```
    -- Stimulus process
    stim_proc: process
          type typeByteArray is array ( NATURAL range <> ) of STD_LOGIC_VECTOR( 3
downto 0 );
           variable arrKeys : typeByteArray( 0 to 11 ):= ( "0000", "0010", "0011",
           "0100", "0101", "0110", "0111",
          "1000", "1001", "1010", "1011", "1111");
     begin
          Rdy <= '1';
          for i in arrKeys'RANGE loop
              Key <= arrKeys(i);
              wait for 4 ms;
              Key <= "1111";
              wait for 2 ms;
          end loop;

        wait;
    end process;
```

Rysunek 9 Proces testbencha modułu dac_signal_generator

2.3.6. keyboard_to_key

Moduł otrzymuje na wejście bajt DO(7:0), reprezentujący kod klawisza z klawiatury PS/2 (odebrany za pomocą modułu PS2_Kbd) i konwertuje kod na określoną nutę (kod nuty) lub ciszę (kod ciszy). Kod ten podawany jest na wyjście key(3:0).



Rysunek 11 Symbol modułu keyboard_to_key

Wejścia:

- CLK – globalny sygnał zegarowy,
- F0 - sygnał informujący czy kod klawisza był poprzedzony bajtami X"F0", czyli kodem zwolnienia klawisza,
- DO_Rdy - sygnał (impuls jednotaktowy) informujący o zakończeniu odbioru kodu
- DO(7:0) - bajt zawierający kod klawisza,
- Key(3:0) - 4 bity reprezentujące kod danej nuty lub ciszę.

Poniższy fragment kodu przedstawia zmianę kodu z klawiatury na kod nuty (lub ciszy). Kod nuty podawany jest na wyjście tylko w tych taktach zegara w których DO_Rdy = 1 oraz F0 = 0 oraz dany klawisz został wybrany aby reprezentował jakąkolwiek nutę. W innych przypadkach na wyjście podawanych jest kod ciszy.

```
architecture Behavioral of keyboard_to_key is
-- 7 klawiszy bialych ASDFGHJ - cdefgah
-- 5 klawiszy czarnych WE TYU - cis dis fis gis b
begin

  process( CLK, DO_Rdy, F0, DO )
  begin
    if( rising_edge(CLK) ) then

     case DO_Rdy & F0 & DO is

      when "10" & X"1C" => -- A
            key <= "0000";
      when "10" & X"1B" => -- S
            key <= "0010";
      when "10" & X"23" => -- D
            key <= "0100";
      when "10" & X"2B" => -- F
            key <= "0101";
        when "10" & X"34" => -- G
            key <= "0111";
        when "10" & X"33" => -- H
            key <= "1001";
        when "10" & X"3B" => -- J
            key <= "1011";
        when "10" & X"1D" => -- W
            key <= "0001";
        when "10" & X"24" => -- E
            key <= "0011";
        when "10" & X"2C" => -- T
            key <= "0110";
        when "10" & X"35" => -- Y
            key <= "1000";
        when "10" & X"3C" => -- U
            key <= "1010";
        when "00" =>
        when "01" =>
      when others =>
            key <= "1111";

    end case;

     if DO_Rdy = '1' and F0 = '1' then
        key <= "1111";
     end if;
    end if;

  end process;
```

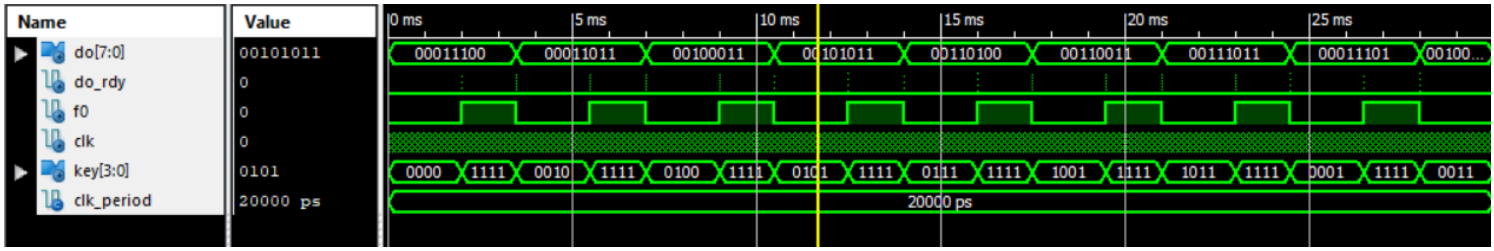
Rysunek 12 Kod VHDL definiujący pracę modułu keyboard_to_key

Symulacja:

Testbench w pętli przypisuje kolejne wartości (klawisze reprezentujące poszczególne nuty) na wejście DO(7:0). Na symulacji widać że moduł poprawie konwertuje klawisze na kody nut. W momencie kiedy jest impuls Do_Rdy = 1 oraz poprzedni klawisz został zwolniony czyli F0 = 0, do wektora key jest przypisywany kod nuty lub cisza, jeżeli klawisz nie reprezentuje żadnej nuty. Jeżeli klawisz zostanie puszczony – F0 = 1, moduł na wyjście podaje "1111" czyli jak wcześniej kod ciszy.

```
   tb: process
        type typeByteArray is array ( NATURAL range <> ) of
STD_LOGIC_VECTOR( 7 downto 0 );
        variable arrBytes : typeByteArray( 0 to 12 ):= ( X"1C", X"1B",
X"23", X"2B", X"34", X"33", X"3B", X"1D", X"24", X"2C", X"35", X"3C",
X"3A");
    begin
        for i in arrBytes'RANGE loop
            DO <= arrBytes(i);
            DO_Rdy <= '1';
            F0 <= '0';
            wait for Clk_period;
            DO_Rdy <= '0';
            wait for 2 ms - Clk_period;
            DO_Rdy <= '1';
            F0 <= '1';
            wait for Clk_period;
            DO_Rdy <= '0';
            wait for 1.5 ms - Clk_period;
        end loop;

        wait;
    end process;
```

Rysunek 13 Proces stworzony do symulcaji modułu keyboard_to_key

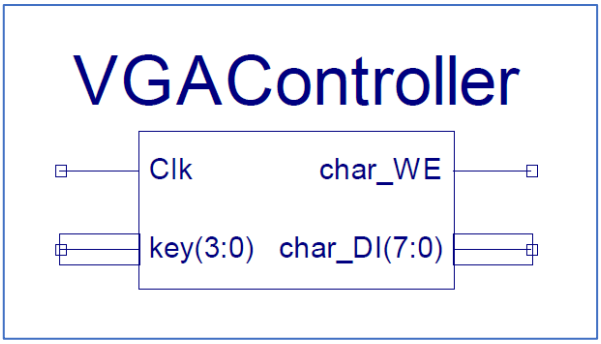


Rysunek 14 Efekt symulacji modułu keyboard_to_key

Na powyższym rysunku przedstawiającym przebiegi czasowe symulacji można zaważyć, że gdy klawisz był naciśnięty na wyjściu *key* pojawiały się wartości kodu przypisane do konkretnych dźwięków lub ciszy.

2.4. VGAController

Funkcją jaką sprawuje ten moduł jest odbieranie na wejściu *key* kodu świadczącego o tym jaki klawisz jest wciśnięty (lub czy żaden nie jest), wysyłanego z modułu *keyboard_to_key*. Są to słowa 4-bitowe, gdzie poszczególnym nutom przypisano konkretne wartości. Działanie modułu jest oparte o maszynę stanów, w której wykrycie zmiany odbieranego wejścia *key* rozpoczynane jest sekwencyjne przechodzenie do kolejnych stanów, czego efektem jest wydruk na wyświetlaczy VGA.



Rysunek 15 Symbol modułu VGAController

Wejścia:

- Clk – globalny sygnał zegarowy
- key – wartość informująca o tym jaki klawisz jest aktualnie naciśnięty. Zależność ta jest opisana przy module *keyboard_to_key*. Każdy klawisz jaki nasze organki mają obsługiwać mają swoją przypisaną wartość zapisaną jako 4-bitowy wektor.

Wyjścia:

- char_WE – jednotaktowy impuls powodujący przekazanie wartości z linii char_DI do modułu VGAtxt48x20,
- char_DI – bajt reprezentujący znak jaki ma być wypisany.

Kod modułu:

```
ARCHITECTURE Behavioral OF VGAController IS
    TYPE state_type IS (Silence, PrintKey, WaitForOther, Space, SecondsS,
Dot, FractionsOfSecond, Space2, SecondsSilence, SpaceSilence, DotSilence,
FractionsOfSecondSilence, Space2Silence, DrawSilence, Space3);


    SIGNAL state : state_type := DrawSilence;
    SIGNAL next_state : state_type;
    SIGNAL lastKey : STD_LOGIC_VECTOR(3 DOWNTO 0);
    SIGNAL counter, tensOfASecond, seconds : integer := 0;
    SIGNAL tensOfASecondToDisplay, secondsToDisplay : STD_LOGIC_VECTOR(3
DOWNTO 0);
```

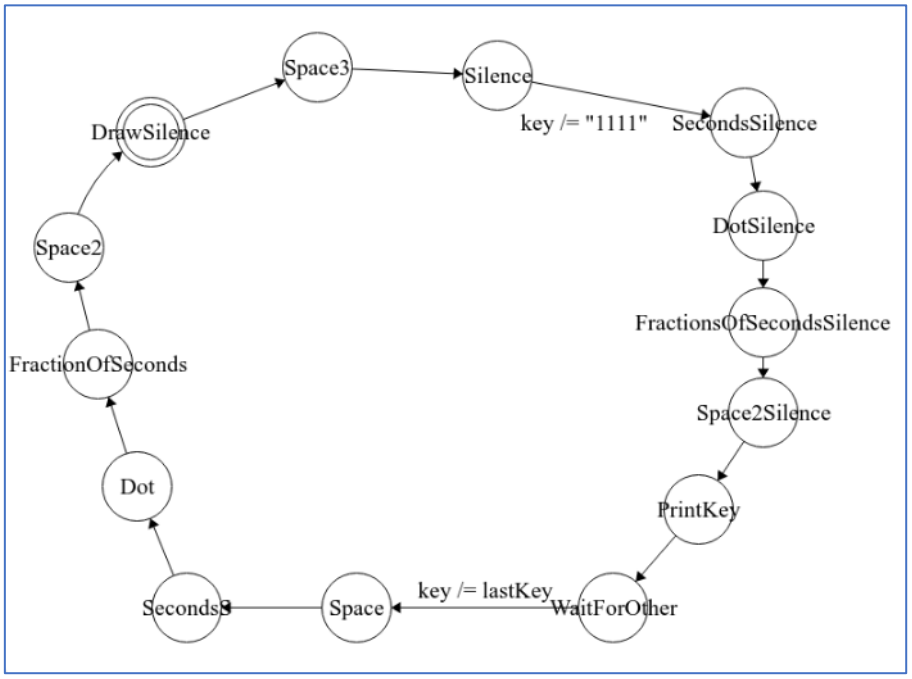
Rysunek 16 Kod VHDL modułu VGAController (cz. 1)

Pierwszy fragment kodu definiuje sygnały wewnętrzne modułu oraz stany, na których będzie opierała się praca modułu jako maszyna stanów. Kolejne zadeklarowane stany odpowiadają za wysyłanie odpowiednich znaków do wypisania na ekranie. Ich nazwy odpowiadają konkretnym znakom jakie mają być wyświetlone. Najważniejszymi stanami, które wykonują coś więcej niż tylko przekazanie znaku do wypisania są: Silence oraz WaitForOther. Ponadto wprowadzone zostały sygnały, które będą zapamiętywały aktualny stan oraz kolejny stan na jaki zostanie przełączony. Kolejne zmienne odpowiadają za: *lastKey* – kod jaki był pobrany w poprzednim takcie, *counter* – zliczanie taktów, *tensOfASecond* – zliczanie dziesiętnych części milisekundy, *seconds* – zliczanie milisekund, *tensOfASecondToDisplay* – liczba dziesiętnych części milisekundy do wypisania, *secondsToDisplay* – liczba milisekund to wypisania.

```
BEGIN

    nextStateProcess : PROCESS (Clk)
    BEGIN
        IF rising_edge(Clk) THEN
                state <= next_state;
        END IF;
    END PROCESS;

    checkLastKeyProcess : PROCESS (Clk)
    BEGIN
        IF rising_edge(Clk) THEN
                lastKey <= key;
        END IF;
    END PROCESS;

    logicProcess : PROCESS (state, key, lastKey)
        BEGIN
        next_state <= state; --default
        CASE state IS
            WHEN Silence
                    IF key /= "1111"    THEN next_state <= SecondsSilence;    -- PrintKey;
                    ELSE next_state <= Silence;
                    END IF;

            WHEN SecondsSilence             => next_state <= DotSilence;
            WHEN DotSilence              => next_state <= FractionsOfSecondSilence;
            WHEN FractionsOfSecondSilence => next_state <= Space2Silence;
            WHEN Space2Silence              => next_state <= PrintKey;
            WHEN PrintKey                       => next_state <= WaitForOther;
            WHEN WaitForOther               =>
                IF key /= lastKey THEN
                    next_state <= Space;
                END IF;
            WHEN Space                          => next_state <= SecondsS;
            WHEN SecondsS                       => next_state <= Dot;
            WHEN Dot                                => next_state <= FractionsOfSecond;
            WHEN FractionsOfSecond          => next_state <= Space2;
            WHEN Space2                         => next_state <= DrawSilence;
            WHEN DrawSilence                    => next_state <= Space3;
            WHEN Space3                         => next_state <= Silence;
            WHEN OTHERS                         => next_state <= Silence;
        END CASE;
    END PROCESS;
```

Rysunek 17 Kod VHDL modułu VGAController (cz. 2)

W powyższej części kodu zdefiniowane zostały procesy, które odpowiadają za przechodzenie pomiędzy stanami. Proces *nextStateProcess* przełącza stany, *checkLastKeyProcess* przypisuje kod klawisza jaki otrzymano w poprzednim takcie, *logicProcess* definiuje do jakiego stanu maszyna ma przejść. W stanie Silence odbywa się sprawdzenie czy wciśnięto klawisz, gdy wcześniej żaden nie był naciśnięty. Jeśli wciśnięto nowy klawisz przechodzi do stanu, od którego zaczyna się wypisanie czasu trwania ciszy, po czym wydrukuje znakową reprezentację nuty i będzie utrzymywała się w stanie *WaitForOther*. Maszyna przejdzie do kolejnych stanów odpowiedzialnych za wypisanie długości trwania nuty, gdy klawisz zostanie zwolniony. Końcowo maszyna powróci do stanu zliczającego czas trwania ciszy. Opisany proces przedstawia poniższy graf oraz fragment kodu z rysunku 15.



Rysunek 18 Graf reprezentujący maszynę stanów.

```
    outputProcess : PROCESS (state, secondsToDisplay, tensOfASecondToDisplay, key)
    BEGIN
        CASE state IS
            WHEN Silence =>
                char_DI <= "00000000"; -- Avoiding latch
                char_WE <= '0';
            WHEN SecondsSilence =>
                char_DI <= std_logic_vector(X"3" & secondsToDisplay(3 DOWNTO 0));
                char_WE <= '1';
            WHEN DotSilence =>
                char_DI <= "00101110"; -- Dot
                char_WE <= '1';
            WHEN FractionsOfSecondSilence =>
                char_DI <= std_logic_vector(X"3" & tensOfASecondToDisplay(3 DOWNTO 0));
                char_WE <= '1';
            WHEN Space2Silence =>
                char_DI <= "00100000"; -- Space
                char_WE <= '1';
            WHEN PrintKey =>
                                                    -- znak na ekranie          Nuta
                IF  key = "0000" THEN char_DI <= x"63";   -- c                    - C
                ELSIF key = "0001" THEN char_DI <= X"43"; -- C                    - C#
                ELSIF key = "0010" THEN char_DI <= X"64"; -- d                    - D
                ELSIF key = "0011" THEN char_DI <= X"44"; -- D                    - D#
                ELSIF key = "0100" THEN char_DI <= X"65"; -- e                    - E
                ELSIF key = "0101" THEN char_DI <= X"66"; -- f                    - F
                ELSIF key = "0110" THEN char_DI <= X"46"; -- F                    - F#
                ELSIF key = "0111" THEN char_DI <= X"67"; -- g                    - G
                ELSIF key = "1000" THEN char_DI <= X"47"; -- G                    - G#
                ELSIF key = "1001" THEN char_DI <= X"61"; -- a                    - A
                ELSIF key = "1010" THEN char_DI <= X"62"; -- b                    - B
                ELSIF key = "1011" THEN char_DI <= X"68"; -- h                    - H
                ELSIF key = "1111" THEN char_DI <= X"23"; -- #                    - CISZA
                ELSE  char_DI <= X"00";
                END IF;
                char_WE <= '1';
            WHEN WaitForOther =>
                char_DI <= "00000000"; -- Avoiding latch
                char_WE <= '0';
            WHEN Space              =>
                char_WE <= '1';
                char_DI <= "00100000"; -- Space
            WHEN SecondsS           =>
                char_DI <= std_logic_vector(X"3" & secondsToDisplay(3 DOWNTO 0));
                char_WE <= '1';
            WHEN Dot                =>
                char_DI <= "00101110"; -- Dot
                char_WE <= '1';
            WHEN FractionsOfSecond =>
                char_DI <= std_logic_vector(X"3" & tensOfASecondToDisplay(3 DOWNTO 0));
                char_WE <= '1';
            WHEN Space2             =>
                char_DI <= "00100000"; -- Space
                char_WE <= '1';
            WHEN Space3             =>
                char_DI <= "00100000"; -- Space
                char_WE <= '1';
            WHEN DrawSilence        =>
                char_DI <= X"23";
                char_WE <= '1';
            WHEN OTHERS =>
                char_DI <= "00000000"; -- Avoiding latch
                char_WE <= '0';
        END CASE;
    END PROCESS;
```

Rysunek 19 Kod VHDL modułu VGAController (cz. 3)

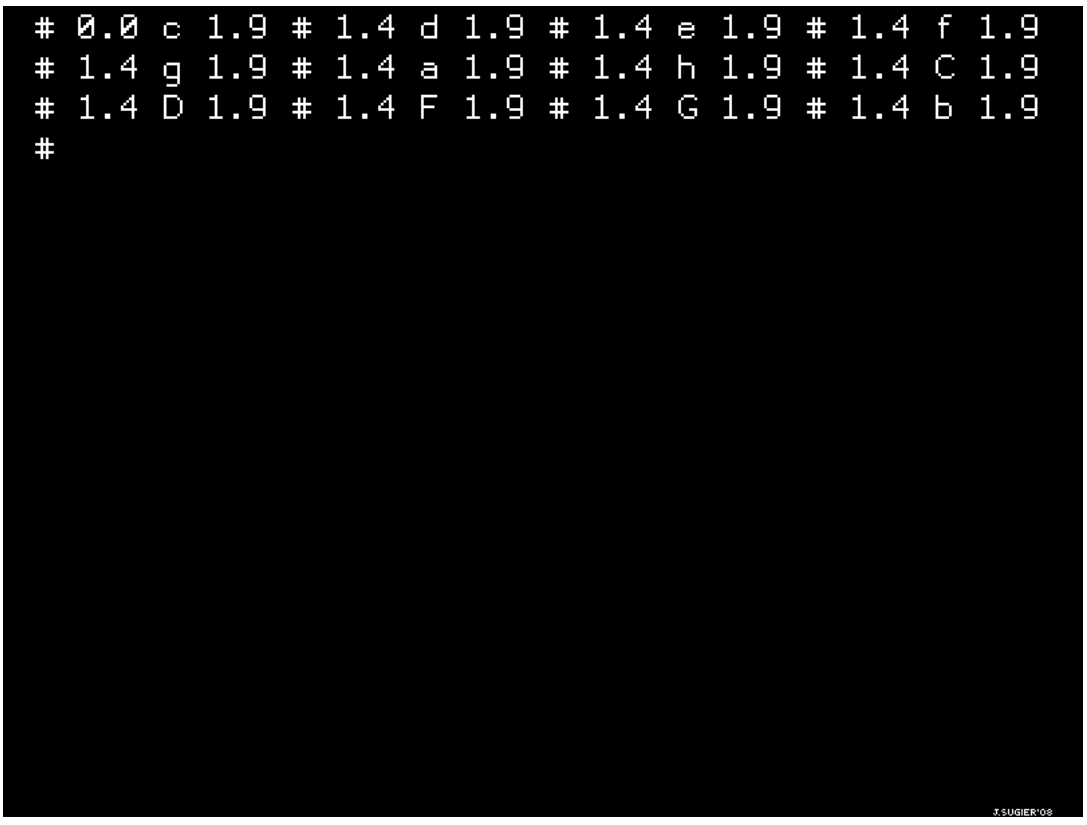
```
     --CLK freq = 50 Mhz T = 1/(50*10^6) s

    -- 1/10s = T * 5 * 10^6
    -- 1/10ms = T * 5 * 10^3

    countToTenthOfAMiliSecond : PROCESS (Clk)
    BEGIN
        IF (rising_edge(Clk)) THEN
            IF counter = 5000 THEN -- czy odliczono 1 ms; 5000000 <- 0,1 s
                counter <= 0;
            ELSE
                counter <= counter + 1;
            END IF;

            IF key /= lastKey THEN
                counter <= 0;
            END IF;
        END IF;
    END PROCESS ;

    countTensOfAMiliSecond : PROCESS (Clk)
    BEGIN
        IF (rising_edge(Clk)) THEN
            IF counter = 5000 THEN -- jeśli policzyło 1/10 ms
                IF tensOfASecond = 9 THEN
                    tensOfASecond <= 0;
                ELSE
                    tensOfASecond <= tensOfASecond + 1;
                END IF;
            END IF;

            IF key /= lastKey THEN
                tensOfASecondToDisplay <=
std_logic_vector(to_unsigned(tensOfASecond, tensOfASecondToDisplay'length));
                tensOfASecond <= 0;
            END IF;
        END IF;
    END PROCESS;

    countMiliSecondsS : PROCESS (Clk)
    BEGIN
        IF (rising_edge(Clk)) THEN
            IF tensOfASecond = 9 THEN
                IF seconds = 9 THEN
                    seconds <= 0 ;
                ELSE
                    seconds <= seconds + 1;
                END IF;
            END IF;
            IF key /= lastKey THEN
                secondsToDisplay  <= std_logic_vector(to_unsigned(seconds,
secondsToDisplay'length));
                seconds <= 0;
            END IF;
        END IF;
    END PROCESS;

END Behavioral;
```

Rysunek 20 Kod VHDL modułu VGAController (cz. 4)

Procesy przestawione w powyższym fragmencie kodu odliczają liczbę taktów zegara, których czas trwania składa się odpowiednio jednej dziesiątej milisekundy (5000 taktów), po czym w zmiennej `tensOfASecond` inkrementuje się jej wartość. W innym procesie sprawdza się czy zliczono 10 takich części. W takim przypadku inkrementuje się wartość zmiennej `seconds`. W każdym procesie sprawdza się też, czy nie został zmieniony kod określający generowany dźwięk, wówczas wartości są zerowane oraz przekazywane są kody znaków reprezentujących odmierzony czas do wyświetlenia.

Symulacja:

Testowanie modułu *VGAController* odbywało się już w utworzonym podmodule układu głównego (*VGAandSoundGenerator*) wraz z dołączonym modułem *VGAtxt48x20*. W tym przypadku testbech napisany w tym celu wyglądał tak samo jak ten w przypadku modułu *keyboard_to_key*. Z dodanymi okresami ciszy. Efektem symulacji były przebiegi czasowe wartości cyfrowych generowanego dźwięku oraz plik bitmapowy przedstawiający wyświetlane znaki na ekranie.



Rysunek 21 Plik wyjściowy symulacji modułu VGAandSoundGenerator

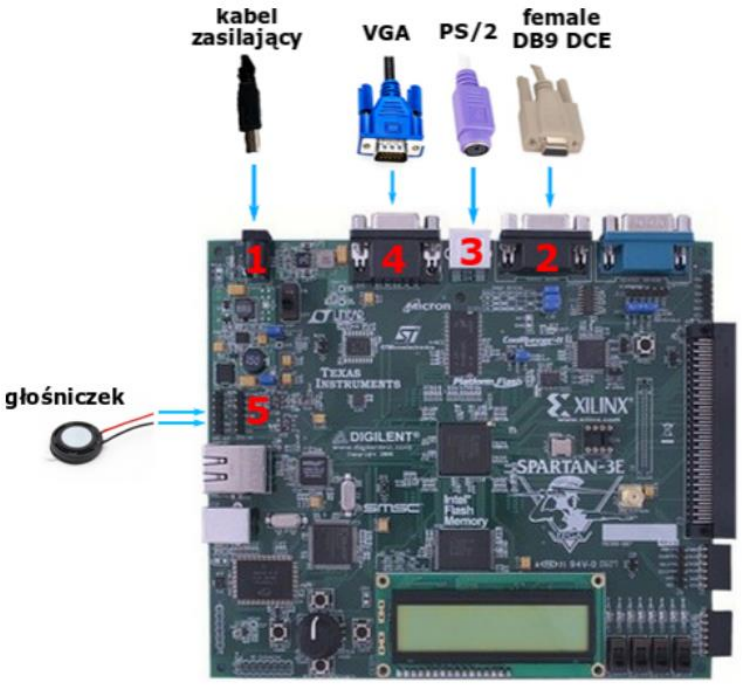
3. Implementacja

3.1. Raport

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	343	9,312	3%
Number of 4 input LUTs	337	9,312	3%
Number of occupied Slices	324	4,656	6%
Number of Slices containing only related logic	324	324	100%
Number of Slices containing unrelated logic	0	324	0%
Total Number of 4 input LUTs	526	9,312	5%
Number used as logic	334		
Number used as a route-thru	189		
Number used as Shift registers	3		
Number of bonded IOBs	16	232	6%
Number of RAMB16s	2	20	10%
Number of BUFGMUXs	1	24	4%
Average Fanout of Non-Clock Nets	2.86		

Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
Autotimespec constraint for clock net Clk_BUFGP	SETUPHOLD	0.842ns	8.007ns	0	00

3.2. Podręcznik użytkowania narzędzia



Rysunek 22 Instrukcja podłączenia płyty

1. Podłączyć kabel zasilający płytkę do portu oznaczonego numerem 1
2. Podłączyć kabel DCE (od komputera) do portu oznaczonego numerem 2.
3. Podłączyć kabel VGA do portu DB15 VGA Connector oznaczonego numerem 4.
4. Podłączyć kabel od klawiatury typu PS/2 do portu oznaczonego numerem 3.
5. Podłączyć kabelki od głośnika do dowolnych pinów przetwornika LTC2624 oznaczonego numerem 5.
6. Zaprogramować płytę w środowisku ISE Design Suite 14.7.
7. Naciskając odpowiednie klawisze (tabelka poniżej) odgrywać nuty i obserwować ich symbole na wyświetlaczu.

Nuta	c	cis	d	dis	e	f	fis	g	gis	a	b	h	Cisza lub klawisz nieodpowiadający żadnej nucie
Klawisz	a	w	s	e	d	f	t	g	y	h	u	j	Brak lub klawisz inny niż w poprzednich kolumnach
Znak na ekranie	c	C	d	D	e	f	F	g	G	a	b	h	#

4. Podsumowanie

4.1. Ocena krytyczna efektu

Założenia projektowe zostały zrealizowane – odegranie oraz wizualizacja dźwięku. Ze względu na brak większego doświadczenia, w projektowaniu układów FPGA rozwiązania zaproponowane mogą nie być optymalne, aczkolwiek nie implementowaliśmy skomplikowanych algorytmów a jedynie funkcjonalne przypadki użycia, więc nie wydaje się abyśmy mogli zrobić to źle, nieefektywnie.

Częstotliwość aktualizacji wartości składowej pojedynczej fali dźwiękowej wynosi 50 zmian na okres fali. Taka dokładność jest przesadzona i ludzkie ucho nie wychwyciło by różnicy pomiędzy dźwiękiem z taką częstotliwością odświeżania a częstotliwością na poziomie 32-36.

4.2. Ocena pracy

4.3. Możliwe kierunki rozbudowy układu

1. Dodanie więcej oktaw – albo poprzez wykorzystanie większej ilości klawiszy jak nut, albo ustawienie klawiszy przełączających do niższej/wyższej oktawy
2. Zapamiętywanie melodii i odgrywanie jej
3. Zapamiętanie wzorcowej melodii, i sprawdzanie poprawności ogrywania jej przez użytkownika

5. Literatura

Sugier, J. (01.06.2020). *Moduły pomocnicze do Spartan-3E Starter Kit - DACWrite*. Pobrano z lokalizacji http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/#_Toc59028436

Sugier, J. (01.06.2020). *Moduły pomocnicze do Spartan-3E Starter Kit - PS2_Kbd*. Pobrano z lokalizacji http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/#_Toc59028429

Sugier, J. (01.06.2020). *Moduły pomocnicze do Spartan-3E Starter Kit - VGAtxt48x20*. Pobrano z lokalizacji http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/#_Toc59028435

XILINX. (01.06.2020). *Spartan-3E FPGA Starter Kit Board User Guide - str. 64-65*. Pobrano z lokalizacji https://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf

XILINX. (01.06.2020). *Spartan-3E FPGA Starter Kit Board User Guide - str. 69-73*. Pobrano z lokalizacji https://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf

XILINX. (01.06.2020). *Spartan-3E FPGA Starter Kit Board User Guide - str. 12*. Pobrano z lokalizacji https://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf