Politechnika Wrocławska
Katedra Informatyki Technicznej
Zakład Systemów Komputerowych i Dyskretnych

Układy cyfrowe i systemy wbudowane 2
Sprawozdanie projektowe

Implementacja pianino w języku VHDL

Autorzy:

Radosław Lis xxxxxx
Ernest Kot xxxxxx
PT-TN-12

prowadzący - dr inż. Jarosław Sugier

Wrocław, dn. 29 maja 2020

Spis treści

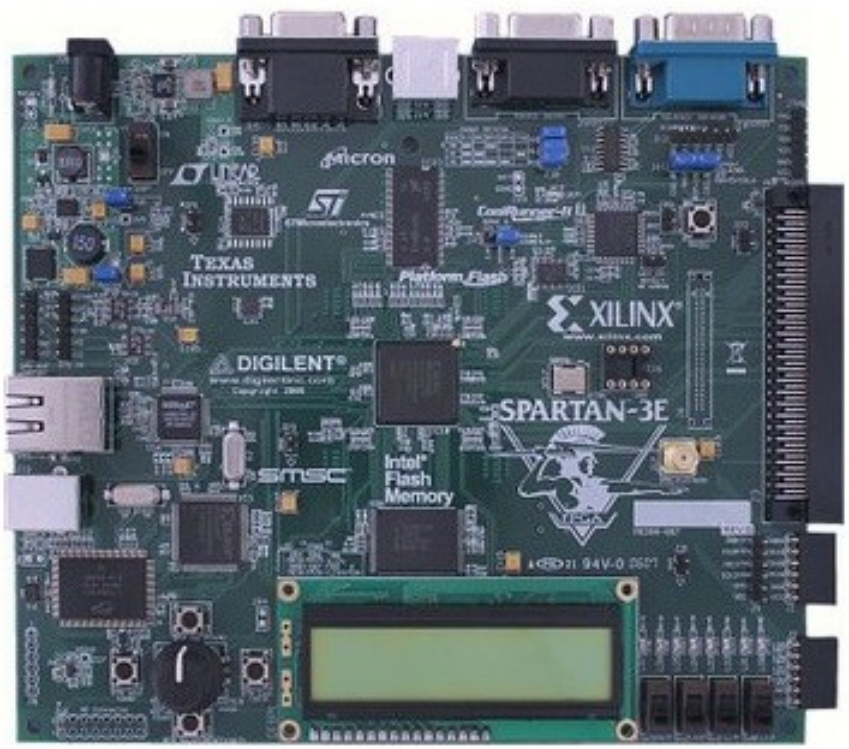
1 Wprowadzenie

1.1 Cel i zakres projektu

Celem projektu było stworzenie jednooktawowego pianino, wydającego dźwięki za pomocą głośnika komunikującego się z układem Spartan-3E. Granie na pianino miało zostać zrealizowane za pomocą klawiatury PS/2, a wizualizacja granych nut oraz okresu wydawania dźwięku miała być zaimplementowana z myślą o zaprezentowaniu jej na monitorach obsługujących VGA (*Video Graphics Array*).

1.2 Opis sprzętu

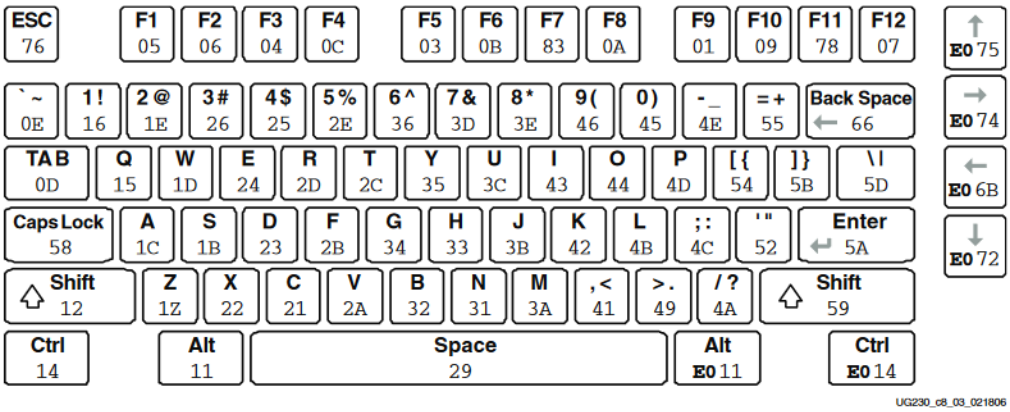
- **Układ FPGA Spartan 3-E** - zapewniający zaprogramowanie układu i komunikację reszty sprzetów



Rysunek 1: Układ Spartan 3-E

- **Monitor ze złączem analogowym VGA** - w celu wizualizacji granych dźwięków
- **Głośnik** - w celu empirycznego doświadczenia granych dźwięków

- **Klawiatura z portem PS/2** - w celu umożliwienia grania na pianino



Rysunek 2: Schemat klawiatury PS/2 wraz z kodami klawiszy

1.3 Teoria

1.3.1 Generowanie dźwięku

Celem było zrealizowanie oktawy trzykreślnej z następującymi wartościami częstotliwości przypisanymi do danego tonu [7]:

ton	częstotliwość [Hz]
c^3	1046,5
cis^3	1108,7
d^3	1174,7
dis^3	1244,5
e^3	1318,5
f^3	1396,9
fis^3	1480,0
g^3	1568,0
gis^3	1661,2
a^3	1760,0
b^3	1865,0
h^3	1975,5

Tabela 1: Tabela częstotliwości wykorzystanych tonów

Do wygenerowania pojedynczego dźwięku, czyli pojedynczej nutki z oktawy trzykreślnej najbardziej nadawało się generowanie sygnału piłokształtnego, które zostało zrealizowane w projekcie. Sygnał piłokształtny powstawał dzięki inkrementacji licznika liczb od 0 do 31 podczas zbocza narastającego ($rising\_edge$) zegara. Licznik się przekręca i działa od nowa. Jeżeli chce się generować sygnał w ten sposób, to należy wziąć pod uwagę taktowanie zegara, które wynosi 50 MHz. Chcąc wydobyć dźwięk o odpowiednim tonie - dopasowanym do wybranej oktawy - trzeba taką liczbę podzielić przez zakres generowanego sygnału, czyli 32 oraz przez częstotliwość wzorcową dźwięku danej oktawy [5].

Przykładowo jeżeli chce się uzyskać dźwięk z oktawy trzykreślnej f^3 o częstotliwości wzorcowej 1396,9 Hz nalezy wykonać poniższe działanie:

$$\frac{50 \cdot 10^6}{32 \cdot 1396.9} \approx 1119$$

Z tego wniosek, że 5-bitowy sygnał piłokształtny musi zostać wygenerowany o częstotliwości 1119 Hz. Z tego powodu zewnętrzny licznik tworzy sygnał 1119 razy w trakcie jednego taktu, po czym licznik był resetowany. Przed końcowym wysłaniem informacji do głośnika, sygnał jest jeszcze przepuszczany przez przetwornik cyfrowo-analogowy.

1.3.2 Konwersja kodów klawiszy PS/2

W celu usprawnienia i ujednolicenia działania reszty modułów klawisze na podstawie swoich kodów - zgodnych ze specyfikacją PS/2 [4] - są konwertowane na wektory z zakresu 0000_b do 1100_b.

1.3.3 Licznik decymalny

Do wypisania czasu wciśnięcia klawisza na ekran za pomocą gotowego modułu *VGAtxt48x20* wymagana jest implementacja licznika decymalnego, który będzie wyświetlał czas z dokładnością do części dziesiętnych sekundy. Implementacja licznika decymalnego podzielona jest na dwa mniejsze liczniki. Jeden odpowiedzialny za zliczanie jedności sekund oraz jeden za zliczanie części dziesiątych sekundy. Jeżeli licznik odpowiedzialny za części dziesiąte zostanie zinkrementowany do liczby 10, wtedy jest resetowany do zera, a licznik odpowiedzialny za jedności sekundy zostaje zinkrementowany [3]. Wynik wyświetlany jest w postaci: **cyfra.cyfra**.

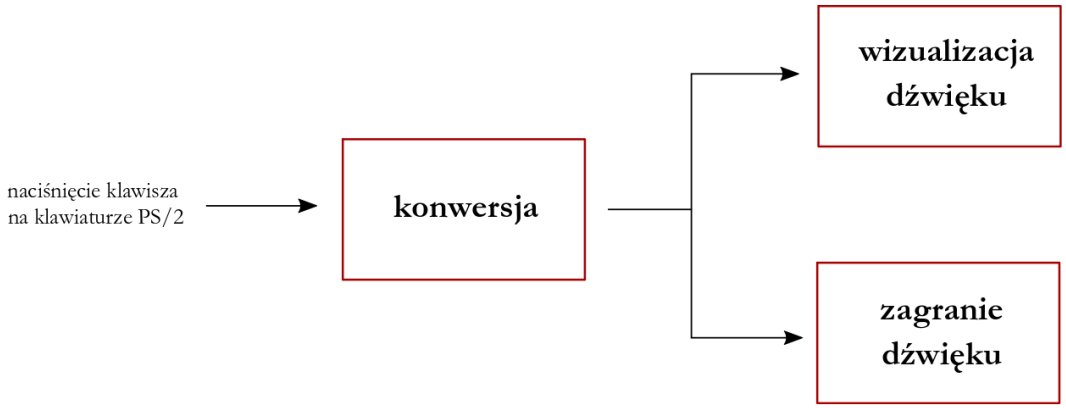
1.3.4 Wyświetlanie na ekranie

Wyświetlanie na ekranie zostało zrealizowane poprzez przechwytywanie przekonwertowanych kodów klawiszy PS/2 by zgodnie z maszyną stanów wyświetlić na ekranie znak odpowiadający danemu kodowi klawisza. W tym celu zostały przypisane poszczególnym tonom 4-bitowe kody od 0000_b do 1100_b. Zgodnie ze specyfikacją modułu *VGAtxt48x20* [10] należało ten kod przekształcić w odpowiadający klawiszowi kod ASCII zgodnie z tabelą kodów ASCII [2]. Dla cyfr zostało zrealizowane to za pomocą prostego doklejenia cyfry do liczby heksadecymalnej 30_{16} [12].

2 Przedstawienie układu

2.1 Struktura ogólna

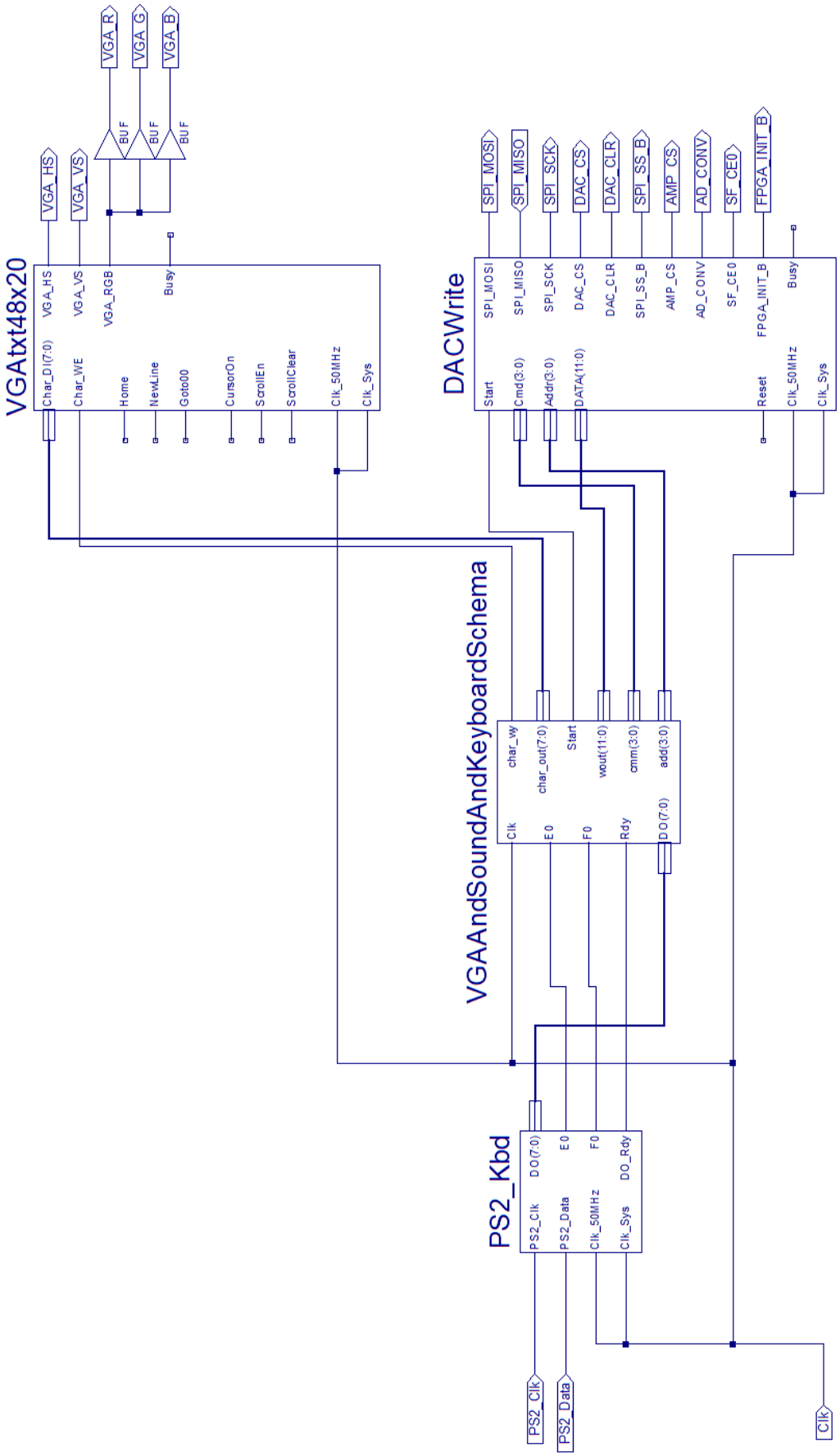
Zgodnie z tym co jest przedstawione na rysunku 3 użykownik miał naciskać klawisz, którego kod miał być za pomocą odpowiedniego modułu przekonwertowany i ta nowa wartość miała zostać wysłana do modułów odpowiadających za wizualizację oraz zagranie dźwięku. Więcej szczegółów co do modułów znajduje się w dalszej części sprawozdania.



Rysunek 3: Uproszczony schemat logiczny projektu

2.2 Schemat szczytowy

Końcowy schemat składający się z wszystkich modułów przedstawiony jest na rysunku 4.



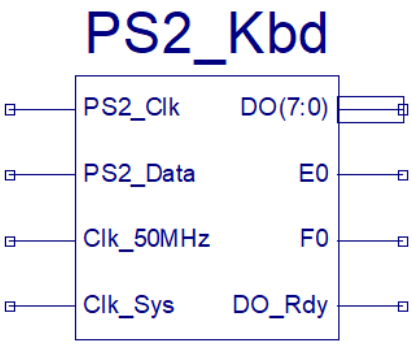
Rysunek 4: Schemat szczytowy projektu

2.3 Moduł PS2_Kbd

2.3.1 Funkcja

Moduł został pobrany ze strony kursu [11]. Wykorzystany został jako odbiornik kodów wysyłanych przez klawiaturę PS/2.

2.3.2 Symbol



Rysunek 5: Symbol modułu

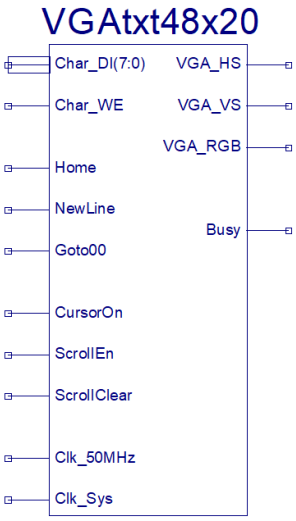
Specyfikacja modułu jest do przejrzenia na stronie kursu [9].

2.4 Moduł VGAtxt48x20

2.4.1 Funkcja

Moduł został pobrany ze strony kursu [11]. Wykorzystany został do wyświetlenia nut oraz okresu w jakim były grane na ekranie.

2.4.2 Symbol



Rysunek 6: Symbol modułu

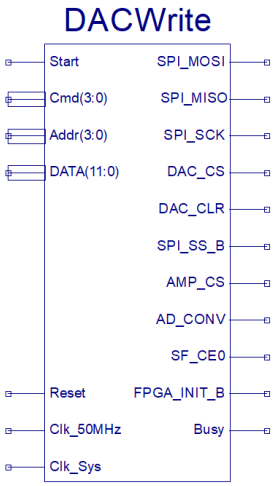
Specyfikacja modułu jest do przejrzenia na stronie kursu [10].

2.5 Moduł DACWrite

2.5.1 Funkcja

Moduł został pobrany ze strony kursu [11]. Wykorzystany został do wysyłania danych do przetwarzania dźwięku na daną czestotliwość.

2.5.2 Symbol

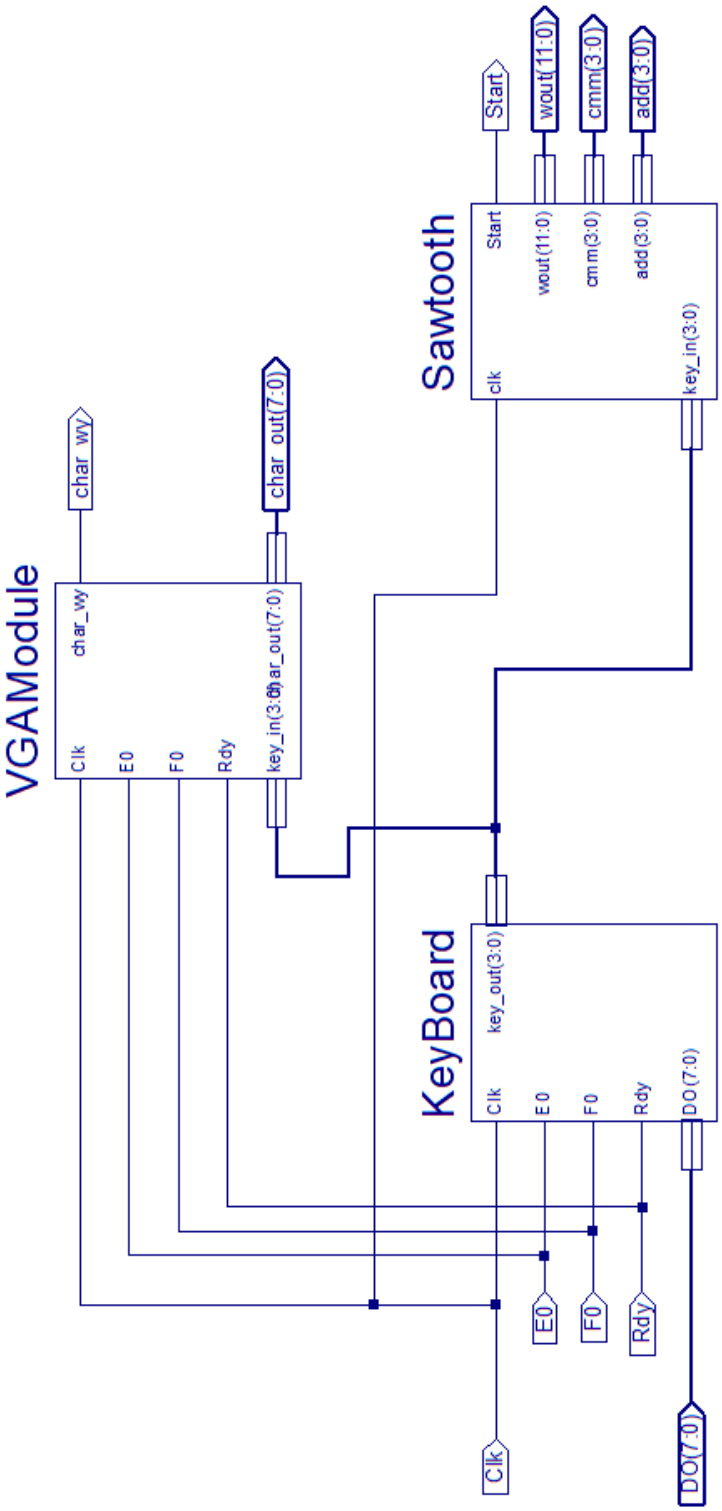


Rysunek 7: Symbol modułu

Specyfikacja modułu jest do przejrzenia na stronie kursu [8].

2.6 Moduł VGAAndSoundAndKeyboardSchema

Jest to główny moduł projektu, składający się z trzech mniejszych podmodułów, tak jak to widać na rysunku 8.



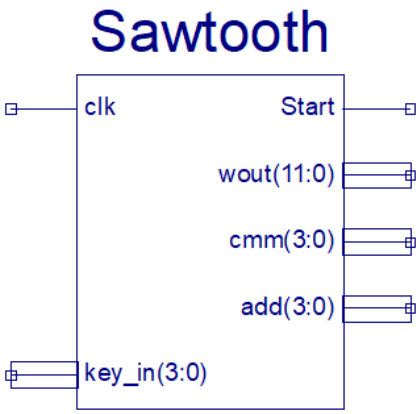
Rysunek 8: Podmoduły składające się na moduł VGAAndSoundAndKeyboardSchema

2.6.1 Moduł Sawtooth

2.6.1.1 Funkcja

Przelicza uzyskany 4-bitowy kod tonu na częstotliwość i odpowiednio kontrolując ją za pomocą licznika zapewnia generację dźwięku w czasie naciskania klawisza na klawiaturze PS/2, mając cały czas na wyjściu odpowiednią częstotliwość. Częstotliwość wysyłana jest - wraz z pozostałymi wyjściami - do modułu *DACWrite*.

2.6.1.2 Symbol



Rysunek 9: Symbol modułu

2.6.1.3 Lista wejść i wyjść

2.6.1.3.1 Wejścia

- *clk* - zegar z taktowaniem 50MHz
- *key_ in(3:0)* - 4-bitowy wektor jednoznacznie identyfikujący dany ton

2.6.1.3.2 Wyjścia

- *Start* - sygnał, który gdy jest równy 1 to zatrzaskuje pozostałe wejścia
- *wout(11:0)* - sygnał piłokształtny o danej częstotliwości
- *cmm(3:0)* - zahardkodowana, stała wartość 1111_b
- *add(3:0)* - zahardkodowana, stała wartość 0011_b

2.6.1.4 Kod

Generowanie częstotliwości odbywało się poprzez inkrementację licznika wraz z każdym *rising_ edge* aż do momentu gdy będzie odpowiadał liczbie cykli procesora na jeden z 32-ch kroków przypisanej do danego tonu na podstawie częstotliwości.

Kod źródłowy 1: Proces generowania częstotliwości

```
1  oneStepIteration : process(clk, data)
2  begin
3      if(rising_edge(clk)) then
4          tempIt <= tempIt + 1;
5          Start <= '0'; --zresetowanie sygnału
6          if(tempIt = numberCyclesProcessor and numberCyclesProcessor /=0 ) then
7              tempIt <= "000000000000";
8              data <= data + 1;
9              Start <= '1'; --zasygnalizowanie potrzeby wygenerowania dźwięku
10         end if;
11     end if;
12     wout <= STD_LOGIC_VECTOR (data)&"0000000"; --sklejenie, aby utworzyć 12-bitowy wektor
13     add<="1111";
14     cmm<="0011";
15 end process;
```

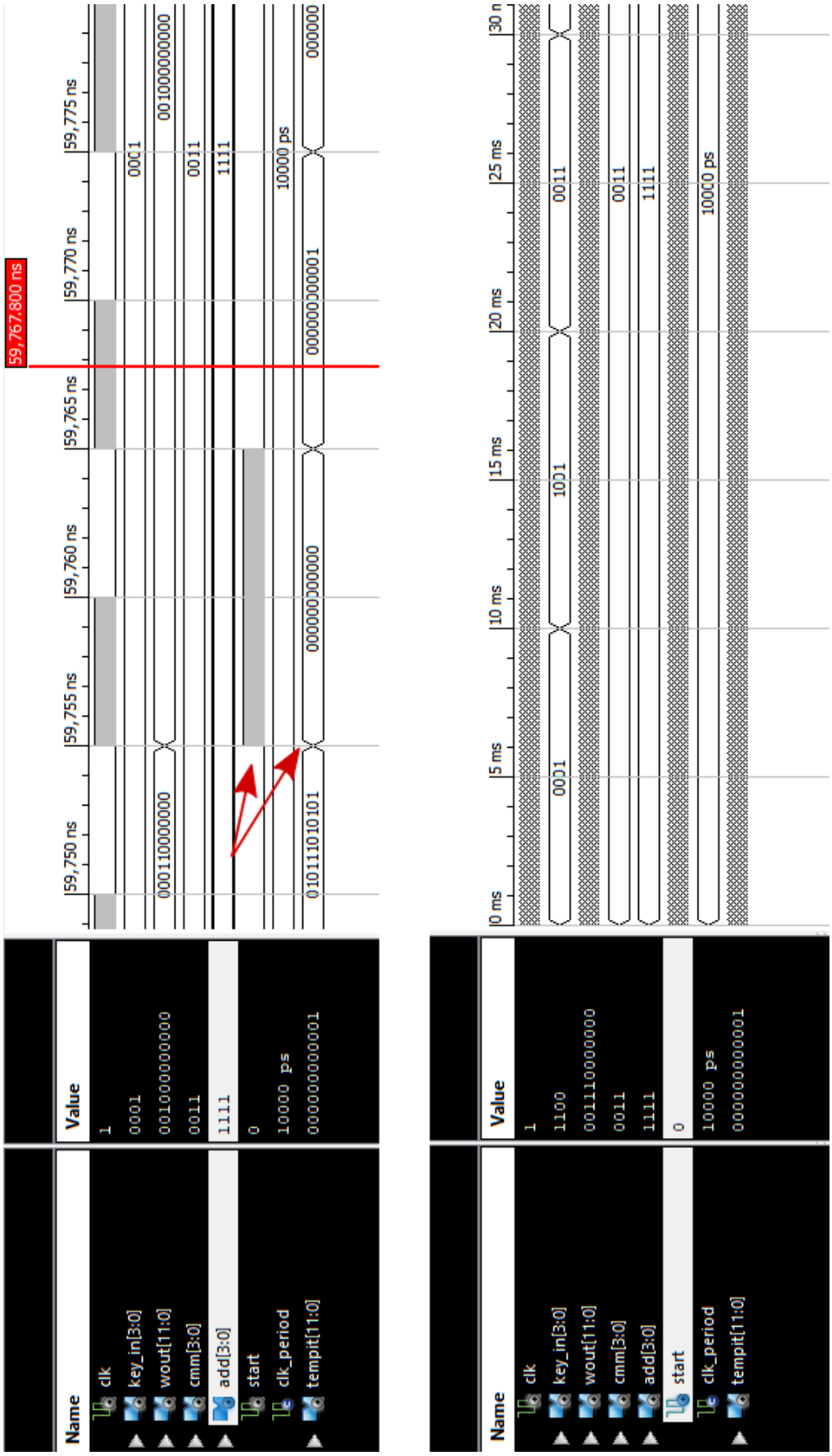
Poniżej jest wylistowany proces, który przypisuje do liczby potrzebnych cykli do wykonania na jeden krok odpowiednią wartość przypisaną do klawisza. W komentarzach widać sposób wyliczenia tejże wartości, tak jak to widać przykładowo w linii 4.

Kod źródłowy 2: Przypisanie wartości liczników na podstawie częstotliwości danego tonu

```
1  frequency_out : process(key_in, numberCyclesProcessor)
2  begin
3      if key_in = "0001" then
4          numberCyclesProcessor <= 1493; --c^3 50 000 000/32/1046,5 -- przypisanie
5      elsif key_in = "0010" then
6          numberCyclesProcessor <= 1409; --cis^3 50 000 000/32/1108,7
7      elsif key_in = "0011" then
8          numberCyclesProcessor <= 1330; --d^3 50 000 000/32/1174,7
9      ...
10     else
11         numberCyclesProcessor <= 0;
12
13     end if;
14 end process;
```

2.6.1.5 Symulacja

Działanie modułu zostało przetestowane i zasymulowane za pomocą napisanego *test bench'a*, który znajduje się w pliku *sawToothTestBench.vhd*. Na rysunku 10 widać 2 screenshot'y z symulacji. Na pierwszym - ze strzałkami - widać, że w momencie gdy licznik *temp* - który liczy cykle procesora - zostaje wyzerowany, zostaje też wrzucona jedynka na wyjście *Start* i rozpoczyna się przesył danych do układu. Na drugim widać ogólne działanie modułu - w *test bench'u* są symulowane po kolei wciśnięcia wybranych klawiszy i widać jak pracują licznik *temp* oraz wektor *wout* z częstotliwością.



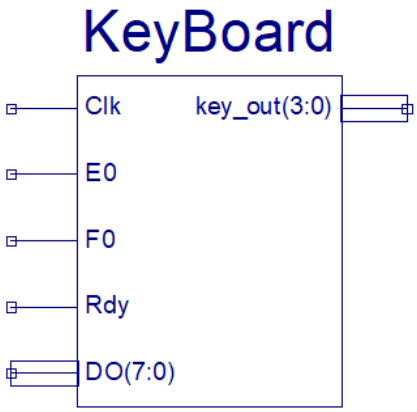
Rysunek 10: Symulacja modułu *Sawtooth*

2.6.2 Moduł Keyboard

2.6.2.1 Funkcja

Moduł dostaje na wejście 1-bajtowy kod klawisza z klawiatury PS/2 (wyjście z modułu *PS2_Kbd*) i następnie konwertuje go za pomocą maszyny stanów - zgodnie z przypisanymi wartościami - na 4-bitowy wektor, który może zostać odczytany przez moduł *Sawtooth*.

2.6.2.2 Symbol



Rysunek 11: Symbol modułu

2.6.2.3 Lista wejść i wyjść

2.6.2.3.1 Wejścia

- *Clk* - zegar z taktowaniem 50MHz
- *E0* - sygnał informujący czy kod z klawiatury był poprzedzony bajtami X"E0", tzw. kod rozszerzony
- *F0* - sygnał informujący czy kod klawiatury był poprzedzony bajtami X"F0", tzw. kod zwolnienia klawisza
- *Rdy* - sygnał (impuls jednotaktowy) informujący o zakończeniu odbioru kodu
- *DO(7:0)* - wektor zawierający kod klawisza, który jest do niego ładowany w momencie gdy $Rdy = 1$

2.6.2.3.2 Wyjścia

- *key_out(3:0)* - 4-bitowy wektor uzyskany na podstawie konwersji kodu klawisza w celu stworzenia wektora, który może być jednoznacznie zidentyfikowany przez moduł *Sawtooth*

2.6.2.4 Kod

Zamiana kodów z klawiatury na PS/2 na możliwe do odczytania przez moduł *Sawtooth* wymagała implementacji maszyny stanów. Każdy klawisz posiada swój własny stan, a dodatkowo został wprowadzony stan ciszy S. Ze względu na poziom skomplikowania nie został zamieszczony graf maszyny stanów (znajdowałoby się tam ponad 200 strzałek), tylko wylistowany został poniżej kawałek jej kodu.

Kod źródłowy 3: Maszyna stanów

```
process3 : process( state, DO, F0 )
begin next_state <= state;
    case state is
        when S => if DO = X"15" and F0 = '0' then next_state <= A; -- Q
        elsif DO = X"1E" and F0 = '0' then next_state <= B; -- 2
        elsif DO = X"1D" and F0 = '0' then next_state <= C; -- W
        elsif DO = X"26" and F0 = '0' then next_state <= D; -- 3
        elsif DO = X"24" and F0 = '0' then next_state <= E; -- E
        elsif DO = X"2D" and F0 = '0' then next_state <= F; -- R
        elsif DO = X"2E" and F0 = '0' then next_state <= G; -- 5
        elsif DO = X"2C" and F0 = '0'  then next_state <= H; -- T
        elsif DO = X"36" and F0 = '0'  then next_state <= I; -- 6
        elsif DO = X"35" and F0 = '0'  then next_state <= J; -- Y
        elsif DO = X"3C" and F0 = '0'  then next_state <= K; -- U
        elsif DO = X"43" and F0 = '0'  then next_state <= L; -- I
        else next_state <= S; end if;
        when A => if DO = X"15" and F0 = '0'  then next_state <= A; -- Q
        ...
        when B => if DO = X"15" and F0 = '0'  then next_state <= A; -- Q
        ...
        when C => if DO = X"15" and F0 = '0'  then next_state <= A; -- Q
        ...
        when D => if DO = X"15" and F0 = '0'  then next_state <= A; -- Q
        ...
        when E => if DO = X"15" and F0 = '0'  then next_state <= A; -- Q
        ...
        when F => if DO = X"15" and  F0 = '0'  then next_state <= A; -- Q
        ...
        when G => if DO = X"15"  and F0 = '0' then next_state <= A; -- Q
        ...
        when H => if DO = X"15" and  F0 = '0'  then next_state <= A; -- Q
        ...
        when I => if DO = X"15"  and  F0 = '0' then next_state <= A; -- Q
        ...
        when J => if DO = X"15"  and F0 = '0' then next_state <= A; -- Q
        ...
        when K => if DO = X"15"  and  F0 = '0' then next_state <= A; -- Q
        ...
        when L => if DO = X"15"  and  F0 = '0' then next_state <= A; -- Q
        ...
        when others => next_state <= S;
    end case;
end process process3;
```

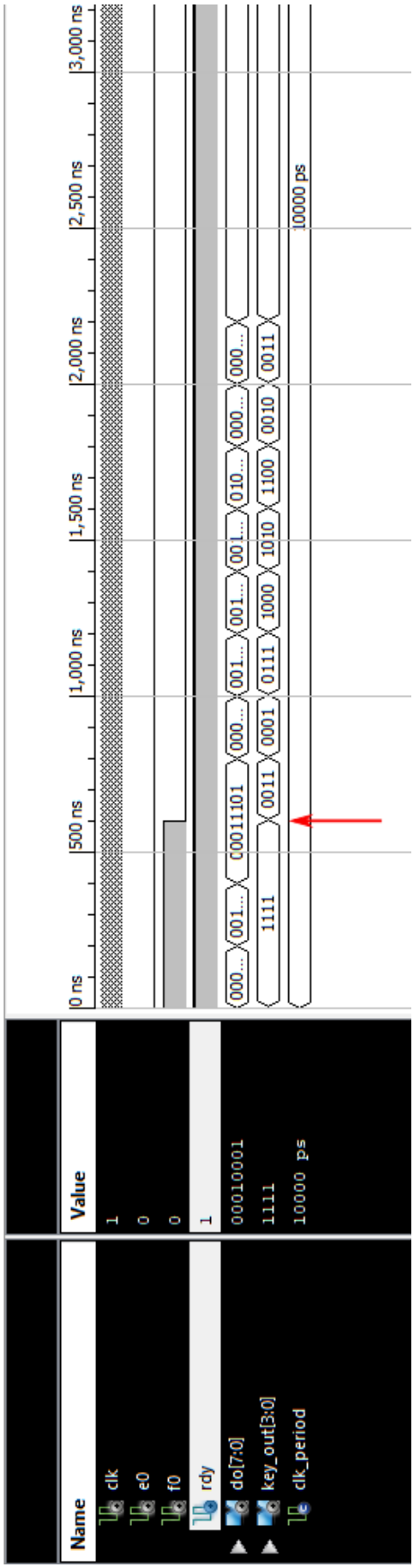
Drugi proces - w zależności od obecnego stanu - odpowiednio modyfikuje wyjście modułu. Do sygnału wyjściowego *key_out* przypisywany jest wektor z zakresu od 0000_b do 1100_b zgodnie z tym co zostało napisane we wstępie teoretycznym.

Kod źródłowy 4: Przypisywanie kodów w zależności od stanu

```
process4 : process( state )
begin
    case state is
        when S =>
            key_out <= "1111"; -- nic
        when A =>
            key_out <= "0001"; -- c^3 Q
        when B =>
            key_out <= "0010"; -- cis^3 2
        when C =>
            key_out <= "0011"; -- d^3 W
        when D =>
            key_out <= "0100"; -- dis^3  3
        when E =>
            key_out <= "0101"; -- e^3 E
        when F =>
            key_out <= "0110"; -- f^3 R
        when G =>
            key_out <= "0111"; -- fis^3 5
        when H =>
            key_out <= "1000"; -- g^3 T
        when I =>
            key_out <= "1001"; -- gis^3 6
        when J =>
            key_out <= "1010"; -- a^3 Y
        when K =>
            key_out <= "1011"; -- b^3 U
        when L =>
            key_out <= "1100"; -- h^3 I
        when others => key_out <= "1111"; -- nic
    end case;
end process process4;
```

2.6.2.5 Symulacja

Działanie modułu zostało przetestowane i zasymulowane za pomocą napisanego *test bench'a*, który znajduje się w pliku $KeyBoardTestBench.vhd$. Na rysunku 12 widać screenshot z przebiegu symulacji. W *test bench'u* na wejście $do(7:0)$ są co $200ns$ podawane nowe kody bajtowe klawiszy, lecz na początku $F0$ jest cały czas równe 1, więc moduł - prawidłowo - tego nie wychwytuje i *key_ out* jest cały czas równy 1111. Dopiero w $600ns$ sygnał $F0$ przyjmuje wartość 0 i kody klawiszy są konwertowane na odpowiednie wektory i przypisywane do *key_ out*, co widać w newralgicznym miejscu oznaczonym czerwoną strzałką.



Rysunek 12: Symulacja modułu *Keyboard*

2.6.3 Moduł VGAModule

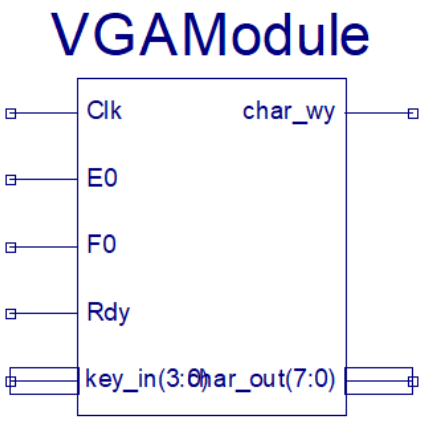
2.6.3.1 Funkcja

Moduł swoją listą wejść jest niemalże identyczny z modułem *Keyboard*, lecz jego funkcja jest zupełnie inna. Czeka na zmianę wartości sygnału *Rdy* i zgodnie ze swoją maszyną stanów wysyła po kolei - znak tonu, spację, czas wykonywania oraz kolejną spację. Wyjścia są przypisane do odpowiednich wejść w module *VGAtxt48x20*. Czyli na ekranie wygląda to mniej więcej tak:

... c 1.2 a 1.8 F 0.3 ...

Wysłanie każdego klawisza jest sygnalizowane za pomocą wyjścia *char_ wy*.

2.6.3.2 Symbol



Rysunek 13: Symbol modułu

2.6.3.3 Lista wejść i wyjść

2.6.3.3.1 Wejścia

- *Clk* - zegar z taktowaniem 50MHz
- *E0* - sygnał informujący czy kod z klawiatury był poprzedzony bajtami X"E0", tzw. kod rozszerzony
- *F0* - sygnał informujący czy kod klawiatury był poprzedzony bajtami X"F0", tzw. kod zwolnienia klawisza
- *Rdy* - sygnał (impuls jednotaktowy) informujący o zakończeniu odbioru kodu
- *key_ in(3:0)* - 4-bitowy wektor jednoznacznie identyfikujący dany ton

2.6.3.3.2 Wyjścia

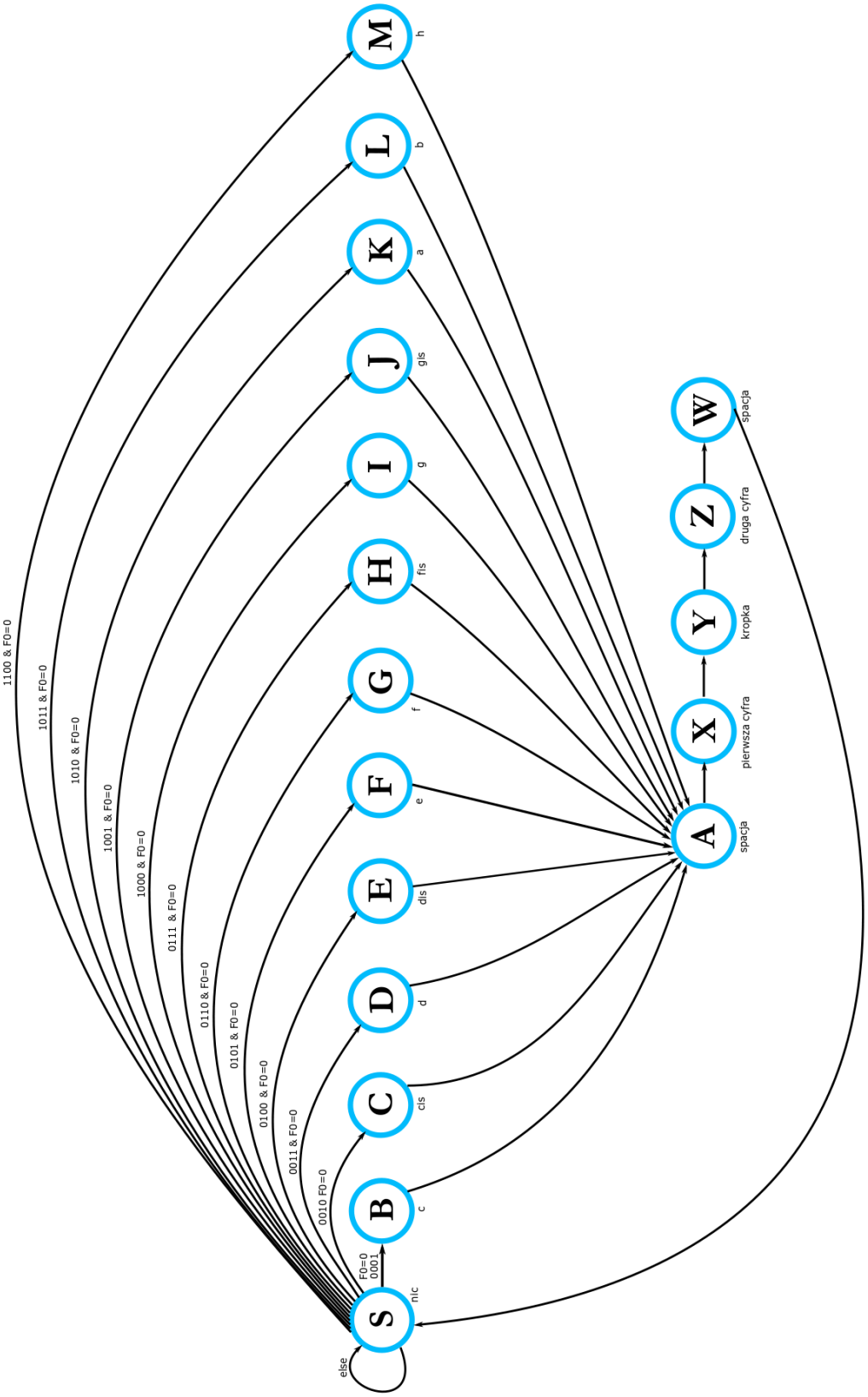
- *char_ wy* - sygnalizuje wysłanie znaku na ekran
- *char_ out* - 8-bitowy wektor reprezentujący kod ASCII znaku, który jest przypisany do tonu

2.6.3.4 Kod

Na potrzeby modułu została skonstruowana maszyna stanów 14, która w zależności od wartości sygnału wejściowego *key_in* zmienia swoje stany. Ważnym elementem jest zdefiniowanie drugiego warunku $F0 =' 0'$ w celu poprawnego obioru kodu.

Kod źródłowy 5: Maszyna stanów modułu VGAModule

```
process2 : process( state, key_in, F0 )
begin next_state <= state;
case state is
    when S => if key_in = "0001" and  F0 = '0' then next_state <= B;
        elsif key_in = "0010" and  F0 = '0' then next_state <= C;
        elsif key_in = "0011" and  F0 = '0' then next_state <= D;
        elsif key_in = "0100" and  F0 = '0' then next_state <= E;
        elsif key_in = "0101" and  F0 = '0' then next_state <= F;
        elsif key_in = "0110" and  F0 = '0' then next_state <= G;
        elsif key_in = "0111" and  F0 = '0' then next_state <= H;
        elsif key_in = "1000" and  F0 = '0' then next_state <= I;
        elsif key_in = "1001" and  F0 = '0' then next_state <= J;
        elsif key_in = "1010" and  F0 = '0' then next_state <= K;
        elsif key_in = "1011" and  F0 = '0' then next_state <= L;
        elsif key_in = "1100" and  F0 = '0' then next_state <= M;
        else next_state <= S; end if;
    when A => next_state <= X;
    when B => next_state <= A;
    when C => next_state <= A;
    when D => next_state <= A;
    when E => next_state <= A;
    when F => next_state <= A;
    when G => next_state <= A;
    when H => next_state <= A;
    when I => next_state <= A;
    when J => next_state <= A;
    when K => next_state <= A;
    when L => next_state <= A;
    when M => next_state <= A;
    when X => next_state <= Y;
    when Y => next_state <= Z;
    when Z => next_state <= W;
    when W => next_state <= S;
    when others => next_state <= S;
end case;
end process process2;
```



Rysunek 14: Graf maszyny stanów modułu *VGAModule*

Zgodnie z tym co zostało napisane we wstępie teoretycznym do 8-bitowego sygnału wyjściowego *char_out* zostaje przypisany kod ASCII odpowiadający danemu tonowi. W przypadku wyrzucania na ekran cyfr dzieje się to za pomocą prostego doklejenia - *X"3" unityCounterToDisplay(3 downto 0)* w linii 33.

Kod źródłowy 6: Modyfikacja znaku wyjściowego w zależności od stanu

```
1  process3 : process( state, decimalCounterToDisplay, unityCounterToDisplay )
2  begin
3      case state is
4          when S =>
5              char_out <= "00000000"; -- nic
6          when A =>
7              char_out <= "00100000"; -- spacja
8          when B =>
9              char_out <= "01100011"; -- c
10         when C =>
11             char_out <= "01000011"; -- cis DUZE C
12         when D =>
13             char_out <= "01100100"; -- d
14         when E =>
15             char_out <= "01000100"; -- dis DUZE D
16         when F =>
17             char_out <= "01100101"; -- e
18         when G =>
19             char_out <= "01100110"; -- f
20         when H =>
21             char_out <= "01000110"; -- fis DUZE F
22         when I =>
23             char_out <= "01100111"; -- g
24         when J =>
25             char_out <= "01000111"; -- gis DUZE G
26         when K =>
27             char_out <= "01100001"; -- a
28         when L =>
29             char_out <= "01100010"; -- b
30         when M =>
31             char_out <= "01101000"; -- h
32         when X =>
33             char_out <= std_logic_vector(X"3" & unityCounterToDisplay(3 downto 0)); -- doklejenie
34         when Y =>
35             char_out <= "00101110"; -- kropka pomiedzy cyframi
36         when Z =>
37             char_out <= std_logic_vector(X"3" & decimalCounterToDisplay(3 downto 0));
38         when W =>
39             char_out <= "00100000"; -- spacja
40         when others => char_out <= "00000000"; -- nic
41     end case;
42 end process process3;
```

W poniższym kodzie źrodłowym nr 7 widać dwa procesy implementujące działanie licznika decymalnego potrzebnego do prawidłowego zliczania czasu wciśnięcia klawisza. Chcąc zaimplementować licznik decymalny, zaimplementowano dwa osobne liczniki, przedstawione w poniższych procesach.

Pierwszy proces - **count**, odpowiedzialny jest za zliczanie części dzięsiatych sekundy. Zgodnie z przedstawionym kodem, działa on w ten sposób, że z każdym przekręceniem zewnętrznego 5-milionowego licznika (sygnał **temp**) inkrementowany jest licznik części dziesiętnych, który ponadto jest resetowany po tym jak osiąnie wartość 10. Zostaje przypisany do sygnału **decimalCounterToDisplay** i wyświetlony na ekran. Licznik ten jest resetowany jeżeli użytkownik nie wciska żadnego klawisza.

Drugi proces - **count2**, odpowiedzialny jest za zliczanie jedności sekundy. Zgodnie z przedstawionym kodem, działa on w ten sposób, że z każdym przekręceniem licznika części dziesiątych sekundy zostaje inkrementowany oraz jego wartość zostaje wysyłana do modułu *VGATxt48x20*, a następnie jest ona wypisywana na ekran. Jeżeli nie jest wciskany żaden klawisz to sygnał jest resetowany do wartości 0.

Kod źródłowy 7: Procesy liczenia okresu naciskania klawisza

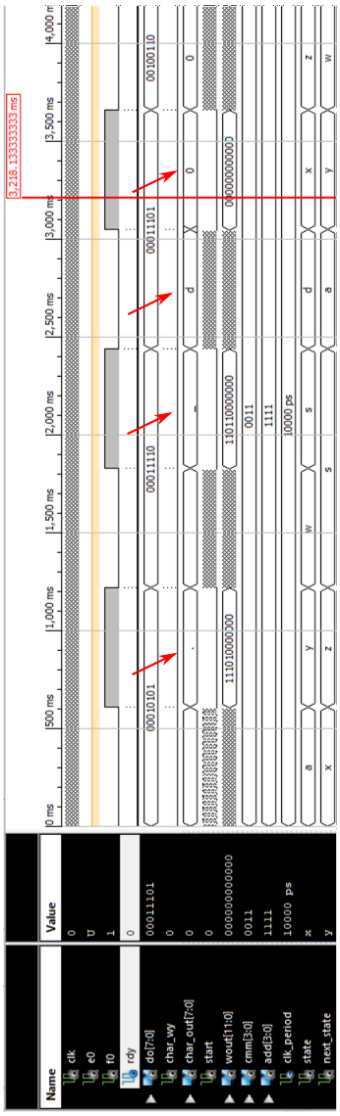
```
count : process (Clk, temp, F0, Rdy)
begin
    if(rising_edge(Clk)) then
        if temp = "01001100010010110100000" then
            if decimalCounter = "1001" then
                decimalCounter <= "0000";
            else
            decimalCounter <= decimalCounter + 1;
            end if;
        elsif temp = "00000000000000000000000" and F0 = '1' then
            if (decimalCounterToDisplay = "0000") then
                decimalCounterToDisplay <= decimalCounter;
            end if;
            decimalCounter <= "0000";
        elsif temp /= "00000000000000000000000" and F0 = '0' and Rdy = '0' then
            decimalCounterToDisplay <= "0000";
        end if;
    end if;
end process count;

count2 : process (Clk, decimalCounter, F0, Rdy)
begin
    if(rising_edge(Clk)) then
        if (decimalCounter="1001" and decimalCounterOld /= decimalCounter) then
            unityCounter <= unityCounter + 1;
            unityCounterToDisplay <= unityCounterToDisplay + 1;
            decimalCounterOld <= decimalCounter;
        elsif temp = "00000000000000000000000" and F0 = '1' and Rdy = '0' then
            unityCounterToDisplay <= "0000";
            decimalCounterOld <= "0000";
        elsif temp = "00000000000000000000000" and F0 = '1' then
            if (unityCounterToDisplay = "0000") then
                unityCounterToDisplay <= unityCounter;
            end if;
            unityCounter <= "0000";
        end if;
    end if;
end process count2;
```

2.6.3.5 Symulacja

Moduł - zgodnie z poleceniem opiekuna projektu - nie został osobno przetestowany, lecz dopiero gdy został złożony cały schemat.

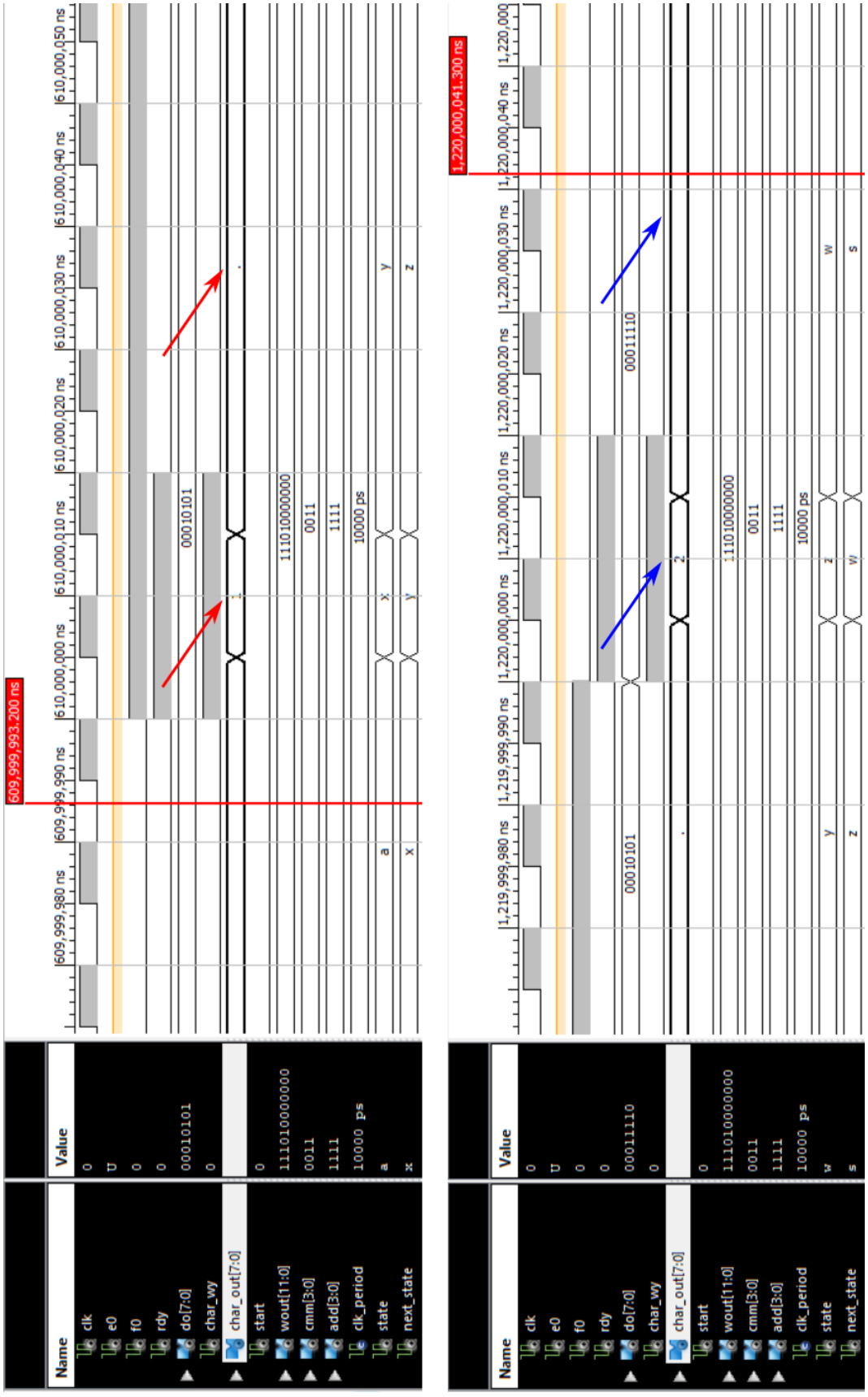
2.6.4 Symulacja

Działanie projektu zostało przetestowane i zasymulowane za pomocą napisanego *test bench'a*, który znajduje się w pliku *VGAAndSoundAndKeyboar.vhd*. Na rysunku 15 widać przebieg symulacji dla $4000ms$. Strzałkami oznaczono znaki, które są wyświetlane na ekranie.



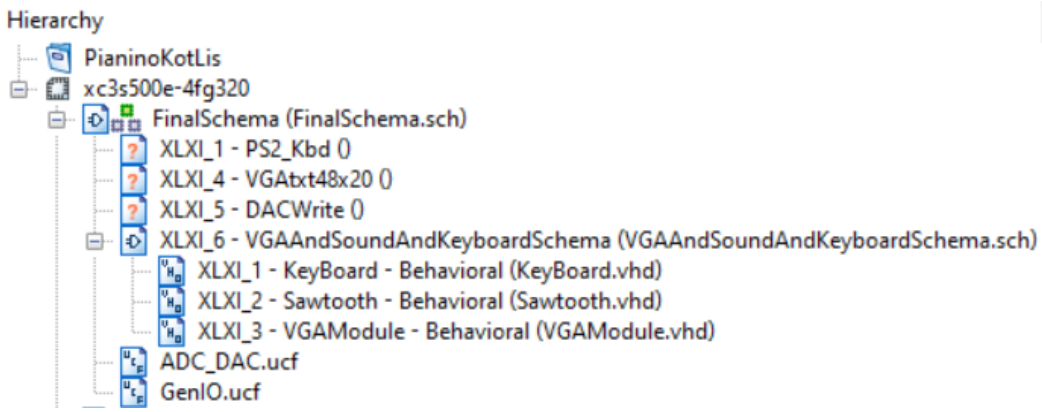
Rysunek 15: Symulacja ostatecznej wersji projektu

Na rysunku nr 16 widać szczegółowo wywołanie impulsu *char_wy* i wszystkie znaki, które są wyświetlane na ekran w danym momencie. Jest to część symulacji, w której na ekranie wyświetlany jest czas, przez jaki był wciskany klawisz. Na pierwszym screenshocie - z czerwonymi strzałkami - widać 1 oraz kropkę, a na drugim - z niebieskimi strzałkami - 2 i spację, co razem składa się w „1.2 ”. Taki czas został odmierzony w $610ms$. Odmierzone zostało dwukrotnie więcej z powodu ustawienia taktowania zegara na $10ns$, jeśli zostałby ustawiony na $20ns$ wszystko byłoby poprawnie.



Rysunek 16: Symulacja ostatecznej wersji projektu

2.6.5 Końcowa hierarchia modułów



Rysunek 17: Hierarchia w projekcie

3 Implementacja

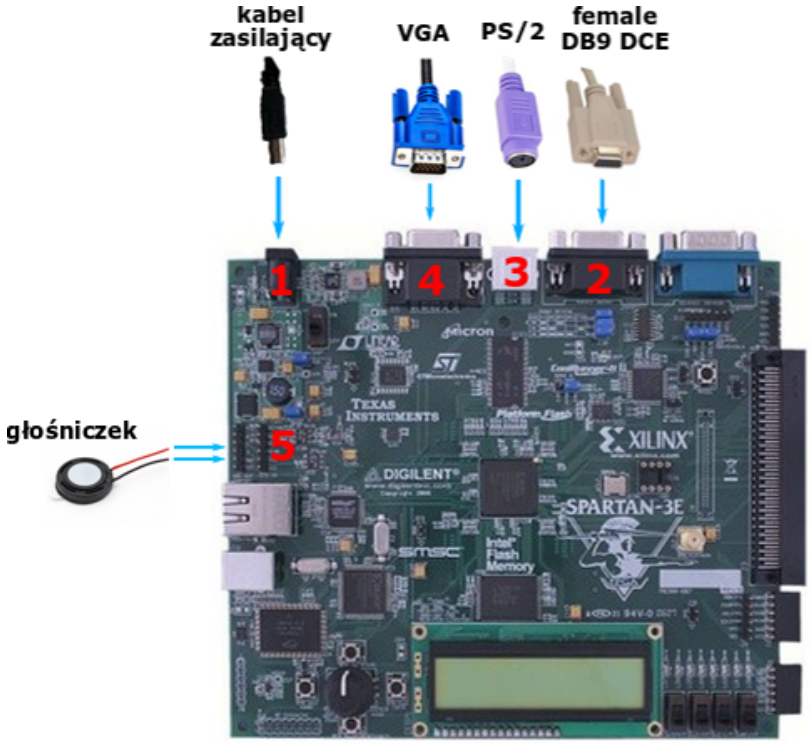
3.1 Raport

Device Utilization Summary			
Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	251	9,312	2%
Number of 4 input LUTs	346	9,312	3%
Number of occupied Slices	265	4,656	5%
Number of Slices containing only related logic	265	265	100%
Number of Slices containing unrelated logic	0	265	0%
Total Number of 4 input LUTs	435	9,312	4%
Number used as logic	343		
Number used as a route-thru	89		
Number used as Shift registers	3		
Number of bonded IOBs	18	232	7%
Number of BUFGMUXs	1	24	4%
Number of Slices	265	4656	5%
Number of SLICEMs	3	2328	1%
Average Fanout of Non-Clock Nets	3.17		

Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
NET "Clk_BUFGP/IBUFG" PERIOD = 20 ns HIGH 50%	SETUP	8.773ns	11.227ns	0	0
	HOLD	1.013ns		0	0

Całkowity rozmiar programu: 732KB

3.2 Uproszczona instrukcja obsługi



Rysunek 18: Rozmieszczenie wykorzystywanych portów układu Spartan 3-E

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. Odgrywać dźwięki za pomocą tabeli 2 - zgodnie z wierszem *Klawisz PS/2* - i obserwować efekt na ekranie. Pojawiające się znaki powinny być zgodne z wierszem *Znak na ekranie*.

Ton	c^3	cis^3	d^3	dis^3	e^3	f^3	fis^3	g^3	gis^3	a^3	b^3	h^3
Klawisz na PS/2	q	2	w	3	e	r	5	t	6	y	u	i
Znak na ekranie	c	C	d	D	e	f	F	g	G	a	b	h

Tabela 2: Dźwięki wraz z odpowiadającymi im kodami klawiszy oraz znaków na ekranie

4 Podsumowanie

4.1 Ocena krytyczna efektu

Założenia projektowe zostały w całości wykonane, choć sam projekt nie został przetestowany na płytce. Zastrzeżenia może budzić implementacja niektórych fragmentów kodu, która - mimo że jest zrobiona prawidłowo i spełnia swoje wymagane funkcje - to mogłaby być zoptymalizowana, m.in. warto by zredukować maszynę stanów, która w module *Keyboard* kontroluje wartość kodów.

4.2 Ocena pracy

Sposród wszystkich implementacji najtrudniejsza okazała się ta odpowiadająca za generację dźwięku, czyli moduł *Sawtooth*. Odpowiada za to lekkie skomplikowanie związane z samym wyliczaniem odpowiednich wartości, liczby cykli procesora by wygenerować daną częstotliwość. Niemało problemów przysporzyła też implementacja modułu *VGAModule* - zaprojektowanie licznika oraz samo wyrzucanie na ekran znaków wymagało trochę myślenia oraz praktycznej znajomości języka VHDL.

Warto również wspomnieć, że praca z modułem *VGAtxt48x20* była utrudniona, ponieważ ze względu na obecną sytuację i brak fizycznego dostępu do płyty Spartan-3E nie można było przetestować czy moduł został rzeczywiście dobrze wykorzystany. Brak empirycznego doświadczenia efektów pracy znacząco spowolnił implementację związaną z jego obsługą.

Względnie prosta okazała się implementacja modułu odpowiadającego za odbiór bajtów z klawiatury PS/2, oprócz tego, że maszyna stanów jest w obecnym stanie ogromna.

4.3 Możliwe kierunki rozbudowy układu

Układ może zostać rozbudowany o kilka ciekawych funkcjonalności:

1. Dodanie więcej oktaw i wykorzystanie większej ilości klawiszy PS/2
2. Dokładniejsze liczenie czasu naciskania klawiszy, np. do setnych części sekundy
3. Odgrywanie wybranej melodii po wykonaniu danej kombinacji klawiszy

Bibliografia

[1] Interwały, częstotliwości dźwięków muzycznych. http://www.fizykon.org/muzyka/muzyka_interwaly.htm? [Online; accessed 29-May-2020].

[2] Kody ASCII. http://www.algorytm.edu.pl/wstp-do-c/ascii.html. [Online; accessed 29-May-2020].

[3] Licznik decymalny. https://surf-vhdl.com/how-to-implement-a-bcd-counter-in-vhdl/. [Online; accessed 29-May-2020].

[4] PS2 Keyboard Scan Codes. https://techdocs.altium.com/display/FPGA/PS2+Keyboard+Scan+Codes. [Online; accessed 29-May-2020].

[5] Sawtooth wave generator in VHDL. https://vhdlguru.blogspot.com/2015/04/sawtooth-wave-generator-in-vhdl.html?fbclid=IwAR0_fKOmRG7ULpcjE7FZFT-QZr7EswhihjnfxQcP8KH6Y-hTUb3n130InAc. [Online; accessed 29-May-2020].

[6] Spartan-3E FPGA Starter Kit Board User Guide. https://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf. [Online; accessed 29-May-2020].

[7] Tabela częstotliwości tonów. http://www.fizykon.org/muzyka/muzyka_tabela_czestotliwosci_tonow.htm. [Online; accessed 29-May-2020].

[8] dr inż. Jarosław Sugier. Specyfikacja modułu DACWrite. http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/#_Toc479592717. [Online; accessed 29-May-2020].

[9] dr inż. Jarosław Sugier. Specyfikacja modułu PS2Kbd. http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/#_Toc479592711. [Online; accessed 29-May-2020].

[10] dr inż. Jarosław Sugier. Specyfikacja modułu VGAtxt48x20. http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/#_Toc479592716. [Online; accessed 29-May-2020].

[11] dr inż. Jarosław Sugier. Zestawy Digilent S3E-Starter. http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga. [Online; accessed 29-May-2020].

[12] Julio Sanchez, Maria P. Canton. Microcontrollers: High-Performance Systems and Programming. https://books.google.pl/books?id=O4XNBQAAQBAJ&pg=PA317&lpg=PA317&. [Online; accessed 29-May-2020].