Politechnika Wrocławska

Wydział Elektroniki
Kierunek Informatyka

Układy cyfrowe i systemy wbudowane 2

Mariusz Makuch 241364
Maksim Birszel 241353

Prowadzący:
Dr inż. Jacek Mazurkiewicz

Wrocław 2020

Spis treści

Rozdział 1

Wprowadzenie

1.1 Cel i zakres projektu

Celem projektu było stworzenie imitacji keyboarda składającego się z jedej oktawy. Dzwięk zostaje wydobyty z podpiętego głośnika do układu Spartan-3E, który jest odtwarzany podczas interakcji użytkownika z klawiaturą.

1.2 Sprzęt oraz opis użytkowania

Projekt został napisany w środowisku Xilinx ISE przy użyciu języka VHDL. Niezbędny sprzęt do zrealizowania projektu:

- Spartan 3E Starter - układ dostępny w sali labolatoryjej. Wyposażony jest w port PS/2 oraz przetwornik cyfrowo-analogowy (DAC LTC2624) do których zostaną podłączone poniższe urządzenia.
- Głośnik - został on podłączony do przetwornika cyfrowo-analogowego układu w celu wytworzenia fali akustycznej o odpowiedniej częstotliwości.
- Klawiatura z portem PS/2 - umożliwia komunikacje użytkownika z układem i odegranie właściwego dzwięku. Na poniższym obrazku możemy zauważyć który klawisz odpowiada za dany dzwięk.

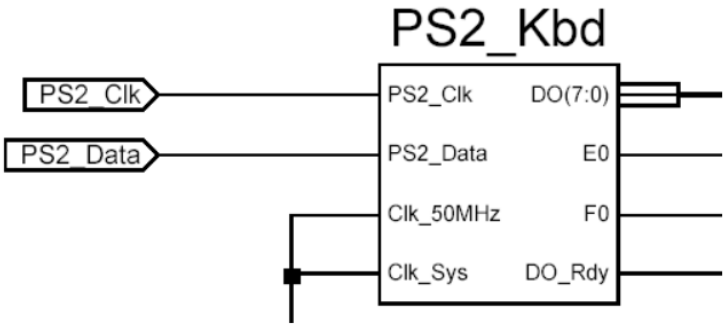
Rozdział 2

Implementacja

2.1 Wykorzystane moduły

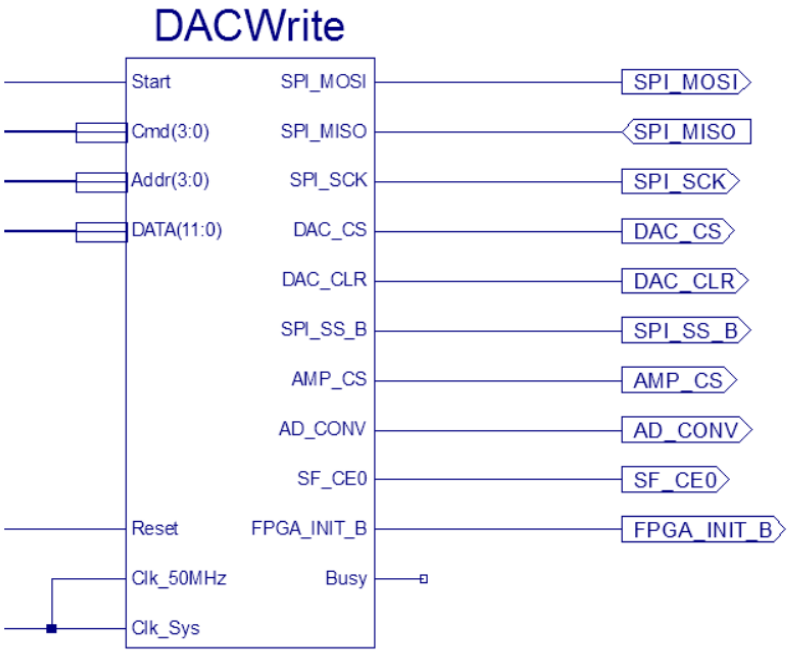
2.1.1 PS2_Kbd

Pierwszym modułem który zostały wykorzystany w projekcie to odbiornik kodów wysyłanych przez klawiaturę PS/2. Sygnały PS2_Clk oraz PS2_Data to sygnały wejściowe które należy dołączyć do wyprowadzeń zewnętrznych. Sygnał DO_Rdy jest odpowiedzialne za sygnalizacje zakończenie odbioru kodu. Na wyjściu DO (7:0) podawa jest wówczas jego wartość, natomiast wyjścia EO i FO sygnalizują czy był on poprzedzony kodem rozszerzony lub kodem zwolnienia klawisza.



2.1.2 DACWrite

Moduł obsługuje wysyłanie danych do przetwornika DAC LTC2624. Impuls Start = '1' zatrzaskuje dane na wejściach Cmd, Addr i DATA oraz rozpoczyna ich szeregowe wysłanie do układu.



2.2 convertModule

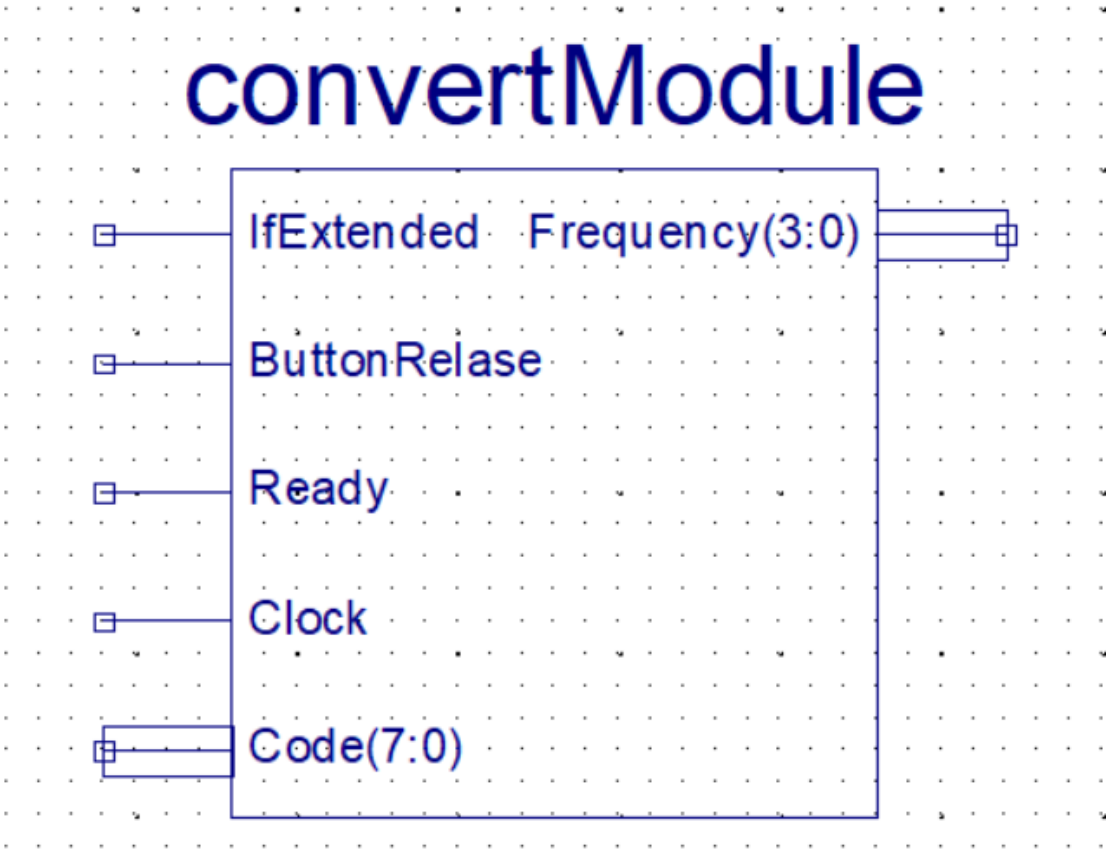
2.2.1 Opis

Wejścia:

- IfExtended - sygnalizuje czy kod z klawiatury był poprzedzony kodem rozszerzonym
- ButonRelase - sygnalizuje zwolnienie klawisza
- Ready - sygnalizuje zakonczenie wysyłania kodu
- Clock - zegar 50MHz
- Code (7:0) - ośmiobitowy kod klawisza

Wyjścia:

- Frequency (3:0) - wartość przypisanan kodowi klawisza



2.2.2 Kod

```
entity convertModule is
    Port ( Code : in  STD_LOGIC_VECTOR (7 downto 0);
           IfExtended : in  STD_LOGIC;
           ButtonRelase : in  STD_LOGIC;
           Ready : in  STD_LOGIC;
           Clock : in  STD_LOGIC;
  Frequency : out  STD_LOGIC_VECTOR (3 downto 0));
end convertModule;

architecture Behavioral of convertModule is

begin
```

```
convert_process: process ( Code, ButtonRelase, IfExtended, Ready, Clock )
begin

      if rising_edge( Clock ) and Ready = '1' then

case (IfExtended & ButtonRelase & Code) is
when ( "00" & X"1C" ) => Frequency <="0001"; -- C
when ( "00" & X"1D" ) => Frequency <="0010"; -- C#
when ( "00" & X"1B" ) => Frequency <="0011"; -- D
when ( "00" & X"24" ) => Frequency <="0100"; -- D#
when ( "00" & X"23" ) => Frequency <="0101"; -- E
when ( "00" & X"2B" ) => Frequency <="0110"; -- F
when ( "00" & X"2C" ) => Frequency <="0111"; -- F#
when ( "00" & X"34" ) => Frequency <="1000"; -- G
when ( "00" & X"35" ) => Frequency <="1001"; -- G#
when ( "00" & X"33" ) => Frequency <="1010"; -- A
when ( "00" & X"3C" ) => Frequency <="1011"; -- A#
when ( "00" & X"3B" ) => Frequency <="1100"; -- B
when ( "00" & X"42" ) => Frequency <="1101"; -- C
when others => Frequency <="0000";
end case;

      end if;

end process;

end Behavioral;
```

2.3 soundGenerateModule

2.3.1 Opis

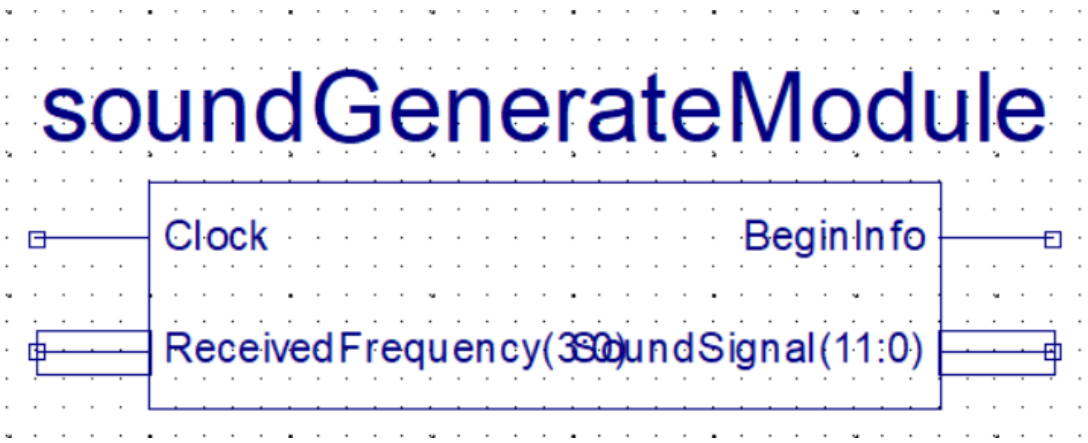
Wejścia:

- Clock - zegar 50MHz
- ReceivedFrequency (3:0) - wartość kodu wciśniętego klawisza

Wyjścia:

- BeginInfo - sygnał odpowiedzialny za wysłanie informacji do DACWrite o zatrząsnięciu danych

- SoundSignal (11:0) - wygenerowany sygnał przekazwany do modulu DACWrite



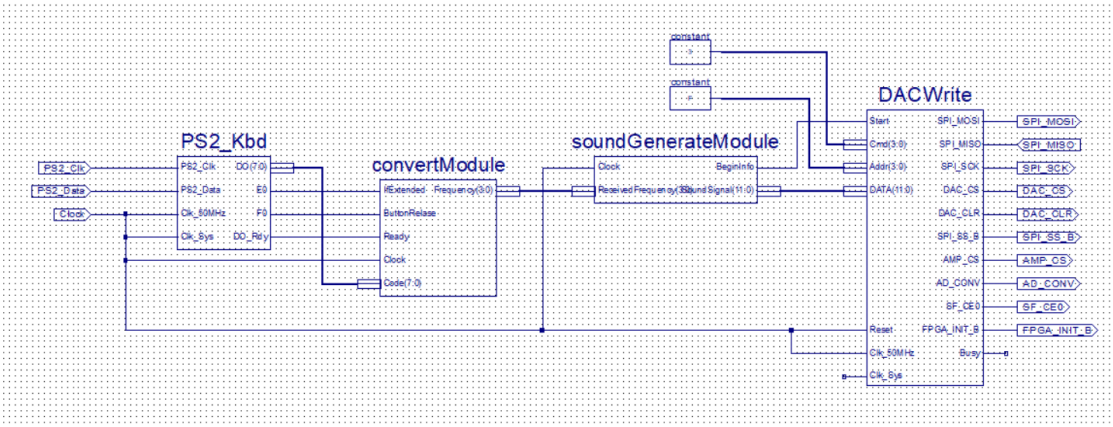
2.3.2 Kod

```
entity soundGenerateModule is
    Port ( Clock : in  STD_LOGIC;
           ReceivedFrequency : in STD_LOGIC_VECTOR (3 downto 0);
           BeginInfo : out  STD_LOGIC;
           SoundSignal : out  STD_LOGIC_VECTOR (11 downto 0));
end soundGenerateModule;

architecture Behavioral of soundGenerateModule is

signal FrequencyValue : integer;
signal Cnt : STD_LOGIC;
signal Data : UNSIGNED (4 downto 0) := X"0"&'0';
signal Counter : UNSIGNED (11 downto 0) := X"000";

begin

soundAssign: process(ReceivedFrequency, FrequencyValue)
begin
case ReceivedFrequency is
when "0001" => FrequencyValue <=1494;
when "0010" => FrequencyValue <=1409;
when "0011" => FrequencyValue <=1330;
```

```
when "0100" => FrequencyValue <=1255;
when "0101" => FrequencyValue <=1184;
when "0110" => FrequencyValue <=1118;
when "0111" => FrequencyValue <=1055;
when "1000" => FrequencyValue <=996;
when "1001" => FrequencyValue <=940;
when "1010" => FrequencyValue <=888;
when "1011" => FrequencyValue <=838;
when "1100" => FrequencyValue <=791;
when "1101" => FrequencyValue <=747;
when others => FrequencyValue <=0;
end case;
end process;

counterValue: process(Clock, Data)
variable i : natural range 0 to 1494;
begin
i := 0;
if(rising_edge(Clock)) then
Counter <= Counter + 1;
BeginInfo <= '0';
if STD_LOGIC_VECTOR(Counter) = FrequencyValue then
Counter <= X"000";
Data <= Data + 1;
BeginInfo <= '1';
end if;
end if;
SoundSignal <= STD_LOGIC_VECTOR(Data)&"0000000";
end process;

soundGenerator: process(Cnt)
variable i : integer := 0;
begin
i := 0;
if(rising_edge(Cnt)) then
if(counter = 31) then
i := 0;
else
i := i + 1;
end if;
```

```
end if;
end process;

end Behavioral;
```

2.4 Schemat szczytowy



Rozdział 3

Symulacja

3.1 Symulacja convertModule

3.1.1 Opis

Na wejściu zostały podane przykładowe kody odpowiadające klawiszom przypisanym do konkretnych, wcześniej ustalonych dźwięków.

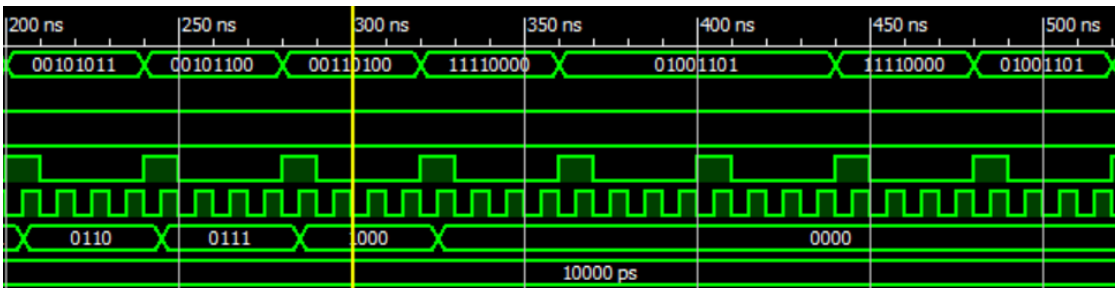
3.1.2 Kod

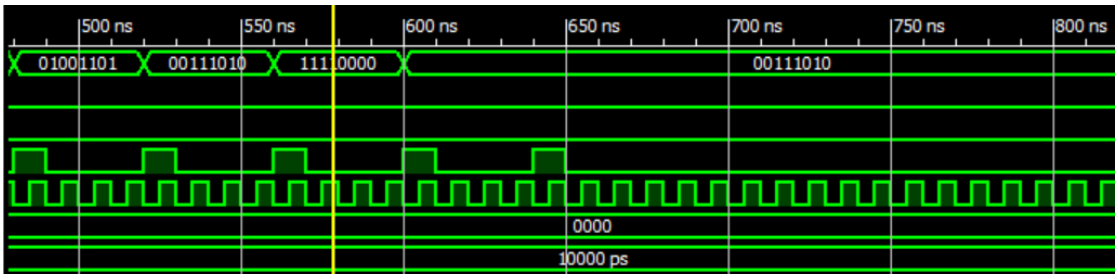
```
    ENTITY convertModule_test IS
END convertModule_test;

ARCHITECTURE behavior OF convertModule_test IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT convertModule
    PORT(
         Code : IN  std_logic_vector(7 downto 0);
         IfExtended : IN  std_logic;
         ButtonRelase : IN  std_logic;
         Ready : IN  std_logic;
         Clock : IN  std_logic;
         Frequency : OUT  std_logic_vector(3 downto 0)
        );
    END COMPONENT;

   --Inputs
```

```
   signal Code : std_logic_vector(7 downto 0) := (others => '0');
   signal IfExtended : std_logic := '0';
   signal ButtonRelase : std_logic := '0';
   signal Ready : std_logic := '0';
   signal Clock : std_logic := '0';

  --Outputs
   signal Frequency : std_logic_vector(3 downto 0);

   -- Clock period definitions
   constant Clock_period : time := 10 ns;

BEGIN

-- Instantiate the Unit Under Test (UUT)
   uut: convertModule PORT MAP (
          Code => Code,
          IfExtended => IfExtended,
          ButtonRelase => ButtonRelase,
          Ready => Ready,
          Clock => Clock,
          Frequency => Frequency
        );

   -- Clock process definitions
   Clock_process :process
   begin
Clock <= '0';
wait for Clock_period/2;
Clock <= '1';
wait for Clock_period/2;
   end process;

   -- Stimulus process
   stim_proc: process
type hexarray is array (0 to 16) of STD_LOGIC_VECTOR(7 downto 0);
variable arrayBytes: hexarray := (X"1C", X"1D", X"1B", X"F0", X"23", X"2B", X"2C",
begin
for i in arrayBytes'RANGE loop
Code <= arrayBytes(i);
```

```
wait for 4 * Clock_period;
end loop;
wait;
   end process;

rdy_process: process
begin
 for i in 0 to 16 loop
 Ready <= '1';
 wait for Clock_period;
 Ready <= '0';
 wait for Clock_period * 3;
end loop;
  wait;
end process;

END;
```

3.1.3 Symulacja czasowa




500 ns
550 ns
600 ns
650 ns
700 ns
750 ns
800 ns
01001101
00111010
11110000
00111010
0000
10000 ps

Rozdział 4

Wnioski

Projekt spełnia początkowe wymagania, a symulacja przebiega zgodnie z poczynionymi założeniami.
Ze względu na brak możliwości rzeczywistego testu oprogramowania na płytce, nie możemy jednoznacznie określić, czy działa w pożądany przez nas sposób.
Największym problemem (jako można było na początku założyć) okazała się praca zdalna, która uniemożliwiała testowanie poszczególnych modułów na bieżąco i rozwiązywanie błędów zmieniając pojedyncze fragmenty kodu i testowanie ich na płytce. Zamiast tego musieliśmy pisać kod całego modułu, następnie symulacje, a w końcowej fazie szukać błędów w kodzie programu oraz w samej symulacji, co strasznie utrudniało cały proces i potęgowało błędy,