UCiSW 2
Odtwarzacz .wav

23.05.2018

Informacja:	XX
Prowadzący:	Dr inż. Jarosław Sugier

Spis treści

1 Wprowadzenie

1.1 Cel projektu

Celem projektu było stworzenie programu wczytującego plik .wav z karty pamięci, pobranie metadanych i odtworzenie dźwięku.

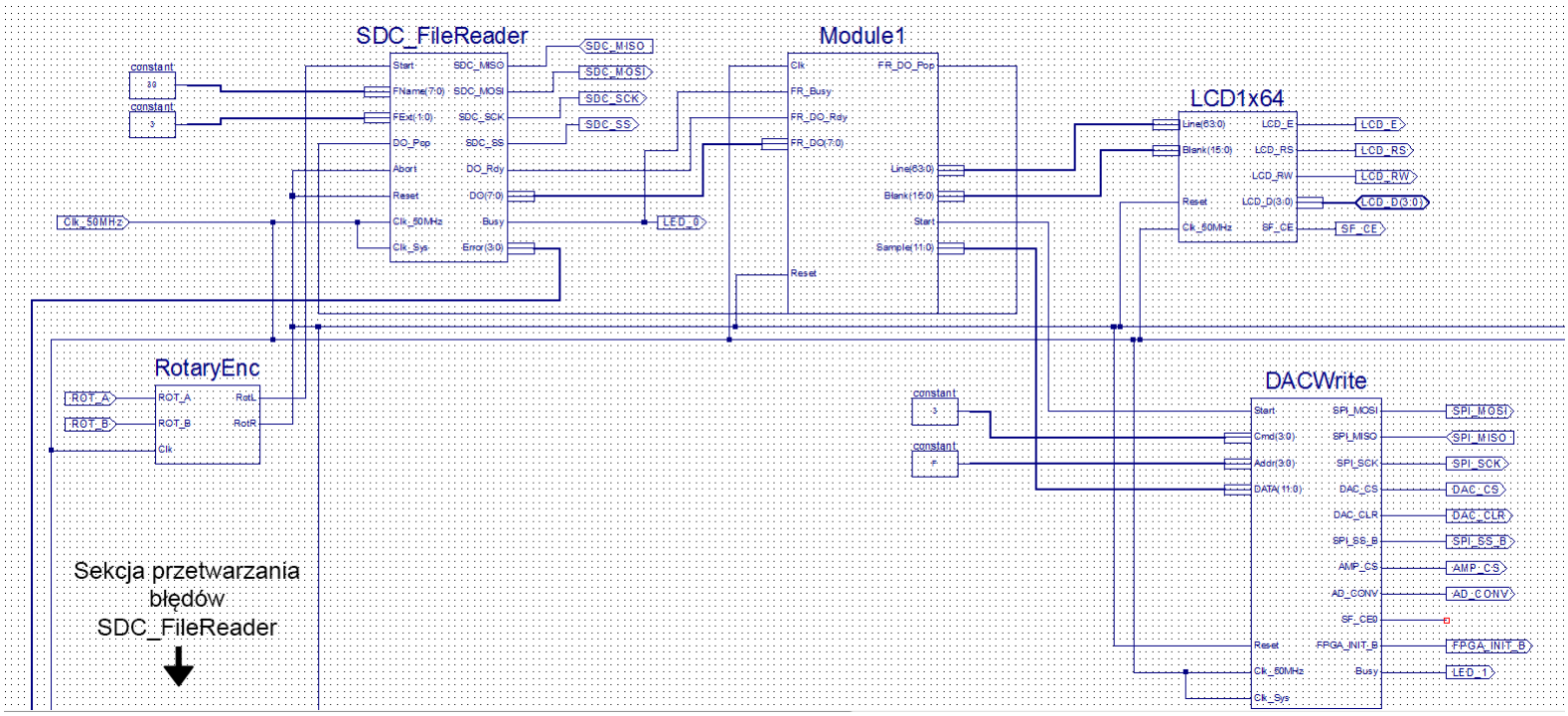
1.2 Sprzęt

- Spartan-3E (XC3S500E)
- Karta SD
- Głośnik

1.3 Teoria

2 Projekt

2.1 Schemat oraz wykorzystane moduły



Wykorzystane moduły pochodzące ze strony zsk.ict.pwr.wroc.pl/zsk_ftp/fpga:

- RotaryEnc: Pozwala na użycie enkodera przyrostowego
  Obrót w lewo: Start (SDC_FileReader)
  Obrót w prawo: Reset (wszystkie moduły)
- SDC_FileReader Pozwala odczytywać dane z karty SD
  W projekcie odczytuje on pliki .wav ("11"=> FExt)
  Nazwa pliku: kod ASCII => FName
- LCD1x64: Wyświetla dane na ekranie Spartan 3E - w tym przypadku metadane pliku .wav
- DACWrite: Wysyła dane do przetwornika DAC
  Sygnał pojawia się na wszystkich pinach ("1111"=> Addr)
  Sygnał wysyłany jest natychmiastowo ("0011"=> Cmd)

2.2 Moduł Module1

Moduł ten ma za zadanie odpowiednio sterować SDC_FileReader by pobrać bajty metadanych oraz dźwięku z pliku .wav. Moduł musi również wysyłać próbki dźwięku w odpowiedniej częstotliwości do DACWrite.

2.2.1 WE/WY

```
Port (
        Clk : in STD_LOGIC;
        Reset: in STD_LOGIC;
        -- Komunikacja z SDC_FileReader
        FR_Busy : in  STD_LOGIC;
        FR_DO : in  STD_LOGIC_VECTOR (7 downto 0);
        FR_DO_Rdy : in  STD_LOGIC;
        FR_DO_Pop : out  STD_LOGIC;
        -- Zawartosc wyswietlacza
        Line : out STD_LOGIC_VECTOR (63 downto 0) := (others => '0');
        Blank : out STD_LOGIC_VECTOR (15 downto 0);
        -- Probka oraz sygnal startu odczytu dla DACWrite
        Sample : out STD_LOGIC_VECTOR (11 downto 0) := (others => '0');
        Start : out STD_LOGIC);
```

2.2.2 Sygnały

```
signal state, nextState : stateType;
-- Licznik odczytanych bajtow
signal counter : signed (63 downto 0) := (others => '0');
-- Licznik dlugosci przerwy miedzy wysylaniem probek
signal counterSampleRate : unsigned (15 downto 0) := (others => '0');
-- Metadane
signal numChannels : STD_LOGIC_VECTOR (15 downto 0);
signal sampleRate : STD_LOGIC_VECTOR (31 downto 0);
signal bitsPerSample : STD_LOGIC_VECTOR (15 downto 0);
```

2.2.3 Proces 1 - przejście między stanami

Proces również wykrywa sygnał Reset - przechodzi wtedy do odpowiedniego stanu (Q0R).

```
process1 : process (Clk, state, Reset)
begin
        if Reset = '1' then
                state <= Q0R;
        elsif rising_edge(Clk) then
                state <= nextState;
        end if;
end process process1;
```

2.2.4 Proces 2 - wybór następnego stanu

```
process2 : process(state, FR_DO, FR_DO_Rdy, FR_Busy, counter, counterSampleRate)
begin
        nextState <= state;
        case state is

        when Q0 =>
                if FR_Busy = '1' then
                        nextState <= Q1;
                else
                        nextState <= Q0;
                end if;

        when Q0R =>
                nextState <= Q0;
```

Stan Q0 trwa do momentu pobudzenia modułu SDC_FileReader przez jednotaktowy sygnał z enkodera - sygnał FR_Busy o wartości 1 wskazuje na to że moduł SDC_FileReader pracuje.

Stan Q0R to stan Reset.

```
        when Q1 =>
                if FR_DO_Rdy = '1' then
                        nextState <= Q2;
                else
                        nextState <= Q1;
                end if;
```

Stan Q1 oczekuje na bajt - SDC_FileReader sygnalizuje koniec ładowania bajtu przy pomocy sygnału FR_DO_Rdy o wartości 1.

Pobieranie metadanych wygląda następująco:

```
process3: process(Clk, state, FR_DO_Rdy, FR_DO)
        begin
        if rising_edge(Clk) and state = Q1 and FR_DO_Rdy = '1' then
                if counter = X"16" then
                        numChannels(7 downto 0) <= FR_DO;
                        Line(7 downto 0) <= FR_DO;
                elsif counter = X"17" then
                        numChannels(15 downto 8) <= FR_DO;
                        Line(15 downto 8) <= FR_DO;
                elsif counter = X"18" then
                        sampleRate(7 downto 0) <= FR_DO;
                        Line(23 downto 16) <= FR_DO;
                elsif counter = X"19" then
                        sampleRate(15 downto 8) <= FR_DO;
                        Line(31 downto 24) <= FR_DO;
                elsif counter = X"1A" then
                        sampleRate(23 downto 16) <= FR_DO;
                        Line(39 downto 32) <= FR_DO;
                elsif counter = X"1B" then
                        sampleRate(31 downto 24) <= FR_DO;
                        Line(47 downto 40) <= FR_DO;
                elsif counter = X"22" then
                        bitsPerSample(7 downto 0) <= FR_DO;
                        Line(55 downto 48) <= FR_DO;
                elsif counter = X"23" then
                        bitsPerSample(15 downto 8) <= FR_DO;
                        Line(63 downto 56) <= FR_DO;
```

```
                end if;
        end if;
end process process3;
```

Program sprawdza obecnie wczytywany bajt - jeżeli jest to jeden z bajtów zawierających interesujące nas metadane, zostaje przekazany do odpowiedniego sygnału.

```
        when Q2 =>
                if counter >= X"4D" then
                        nextState <= Q3;
                else
                        nextState <= Q1;
                end if;
```

Stan Q2 sprawdza obecnie wczytywany bajt - jeżeli wczytano wszystkie bajty metadanych (bajty do 77) program przechodzi do dalszych stanów odpowiedzialnych za wczytywanie bajtów dźwięku; w innym przypadku następuje powrót do Q1.

```
        when Q3 =>
                if FR_DO_Rdy = '1' then
                        nextState <= Q4;
                else
                        nextState <= Q3;
                end if;
```

Stan Q3 działa analogicznie do Q1 - oczekuje na bajt.

Pobieranie dźwięku wygląda następująco:

```
process4: process(Clk, state, FR_DO_Rdy, FR_DO)
        begin
        if rising_edge(Clk) and state = Q3 and FR_DO_Rdy = '1' then
                sample(11 downto 4) <= FR_DO;
        end if;
end process process4;
```

Przed stanem Q3 występuje stan Q3B który nadaje odpowiednie tempo odczytywania (oraz w konsekwencji - wysyłania) dźwięku. Odpowiedni sygnał jest inkrementowany oraz sprawdzany - gdy przekroczy odpowiednią wartość stan Q3B przechodzi do Q3.

Ze względu na brak czasu projekt jest przystosowany pod pliki .wav o częstotliwości 8000 KHz - wartość ta powinna być porównywana z sygnałem sampleRate umożliwiając różne wartości odczytane z metadanych.

```
when Q3B =>
        if counterSampleRate >= X"186A" then
                nextState <= Q3;
        else
                nextState <= Q3B;
end if;
```

Proces odpowiedzialny za inkrementacje oraz zerowanie sygnału counterSampleRate:

```
process4b: process(Clk, state, counterSampleRate)
        begin
        if rising_edge(Clk) then
                if state = Q3B then
                        counterSampleRate <= counterSampleRate + 1;
                elsif state = Q4 then
                        counterSampleRate <= X"0000";
                end if;
        end if;
end process process4b;
```

```
        when Q4 =>
                if FR_Busy = '0' then
                        nextState <= Q5;
                else
                        nextState <= Q3B;
                end if;
```

Stan Q4 weryfikuje to czy SDC_FileReader pracuje - FR_Busy o wartości 0 sygnalizuje że wszystkie bajty zostały odczytane; program przechodzi wtedy do stanu końcowego Q5. W przeciwnym przypadku następuje powrót do odczytywania bajtów w Q3 (przez Q3B).

W stanie Q4 następuje również zawiadomienie modułu DACWrite tak by ten rozpoczął ładowanie przesłanej próbki:

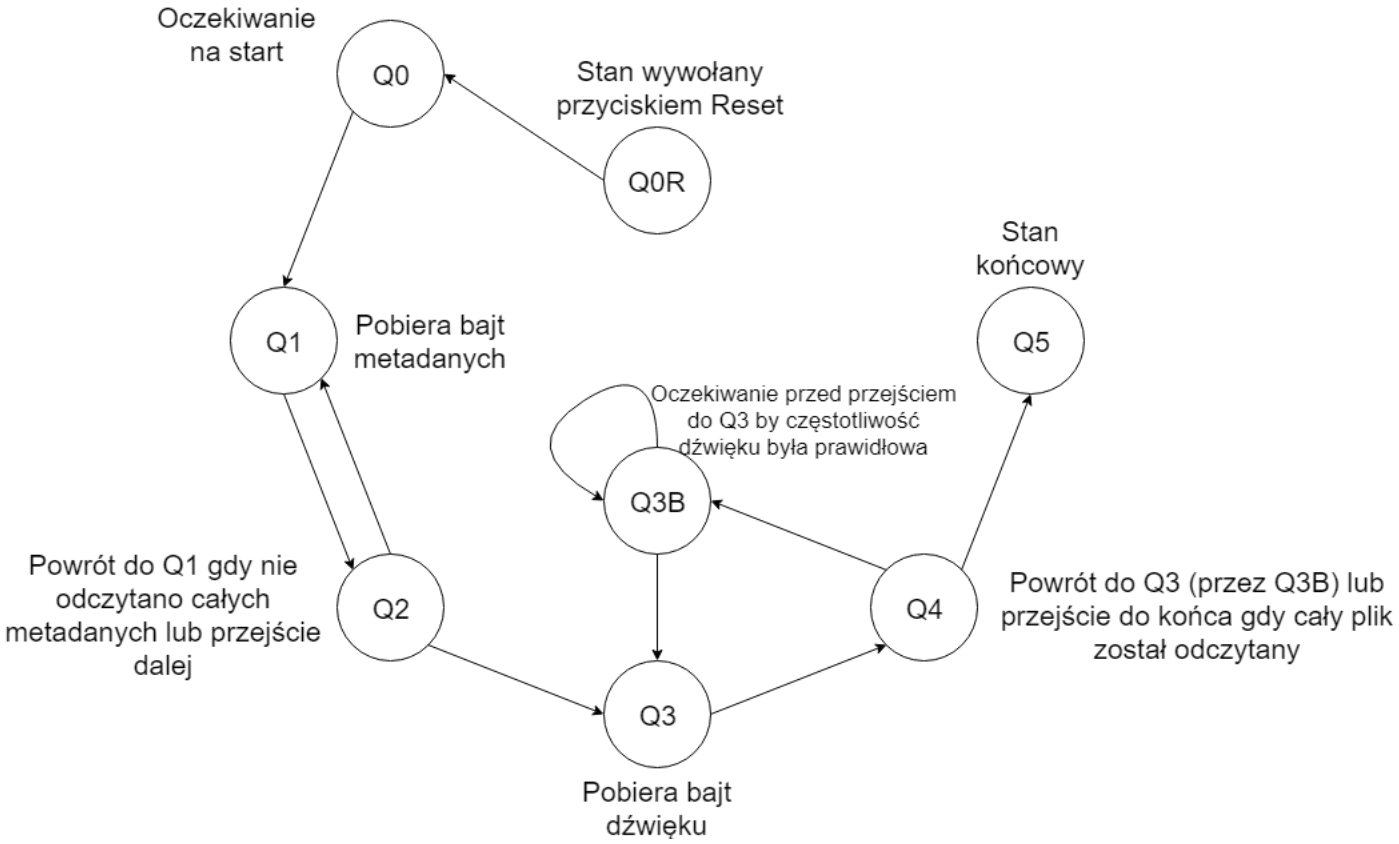
```
Start <= '1' when state = Q4
        else '0';
```

Ostatnim stanem jest Q5 - pętla końcowa:

```
        when Q5 =>
                nextState <= Q5;

        end case;
end process process2;
```

2.2.5 Diagram stanów



2.3 Symulacja

3 Implementacja

4 Podsumowanie

4.1 Ocena krytyczna

4.2 Wnioski

TEST [3] [2] [1] test

Literatura

[1] dr inż Jarosław Sugier. Strona zestawu spartan-3e. zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/.

[2] C. Stuart. Microsoft wave soundfile format. soundfile.sapp.org/doc/WaveFormat/.

[3] Xilinx. *Spartan-3E FPGA Starter Kit Board User Guide*. xilinx.com/support/documentation/boards_and_kits/ug230.pdf