POLITECHNIKA WROCŁAWSKA
WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

Układy cyfrowe i systemy wbudowane 2

Instrument muzyczny
Sprawozdanie końcowe z projektu

Termin zajęć: PON, 8:00-11:00 TP

Autorzy (Grupa B):
Maciej Byczko, 252747
Bartosz Matysiak, 252757

Prowadzący zajęcia:
dr inż. Jacek Mazurkiewicz

Wrocław, 2022r.

Spis treści

1 Temat projektu

Projekt miał dotyczyć wykonania implementacji w sprzęcie instrumentu muzycznego typu "keyboard" przy użyciu płyty Spartan 3E.

2 Opis funkcjonalności

2.1 Założenia początkowe

W trakcie sporządzania początkowych założeń projektowych, stworzyliśmy następujące listy funkcjonalności:

2.1.1 Wersja podstawowa

- Możliwość użycia gamy dźwięków uprzednio nagranych na kartę SD.
- Wsparcie dla klawiatury komputerowej.
- Odtworzenie dźwięku.

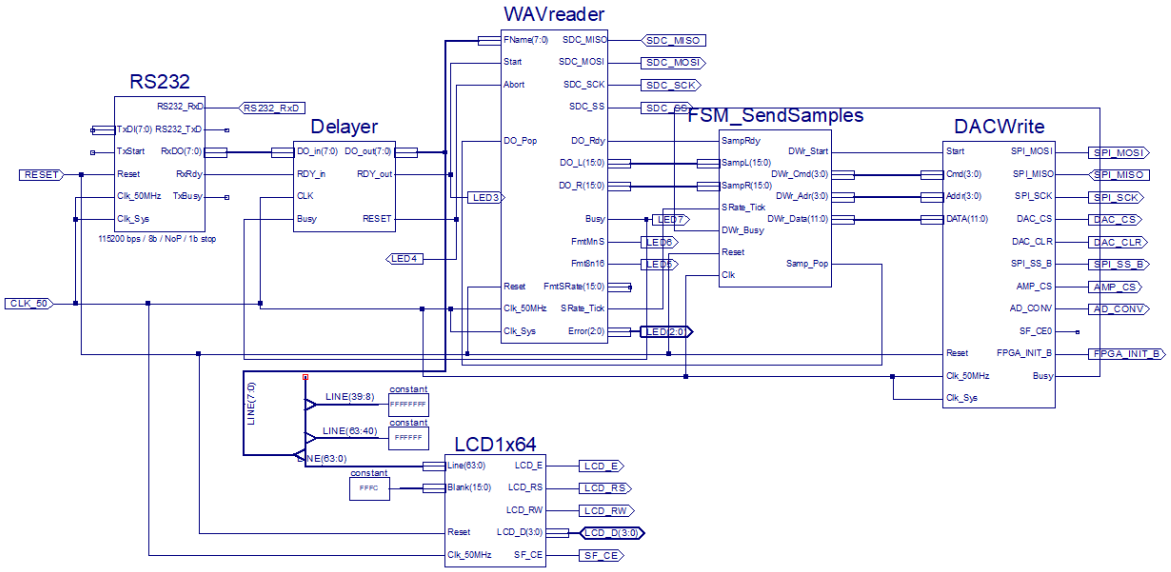
2.1.2 Możliwe rozszerzenia

- Możliwość prostej syntezy jednotonowej.
- Wsparcie dla launchpada.

2.2 Porównanie i nasza ocena projektu

Udało się zrealizować wszystkie założenia wymienione na liście funkcjonalności podstawowych; efekty tej części naszej pracy uważamy za zadowalające, zgodne z naszymi pierwotnymi wyobrażeniami co do funkcjonowania projektu. Ze względu na niespodziewane problemy nie zdążyliśmy przystąpić do prac nad rozszerzeniami. Z całą pewnością, dysponując większą ilością czasu, wykonalibyśmy wymienione rozszerzenia w pierwszej kolejności.

3 Schemat urządzenia



W projekcie wykorzystane zostały moduły własne, jak też te dostępne na stronie Zestawu Digilent S3E-Starter. Poza modułami ukazanymi na schemacie, projekt wykorzystuje także dodatkowe urządzenia: kartę SD z systemem plików FAT32, klawiaturę na złącze RS232 oraz głośnik. Dodatkowo, informacje pomocnicze dotyczące prawidłowego działania układu ukazywane są na diodach LED oraz wyświetlaczu LCD.

4 Podział projektu na moduły

Listy modułów użytych w projekcie są kolejnym elementem, który uległ zmianie względem pierwotnych założeń:

4.1 Wykorzystane moduły ze strony kursu

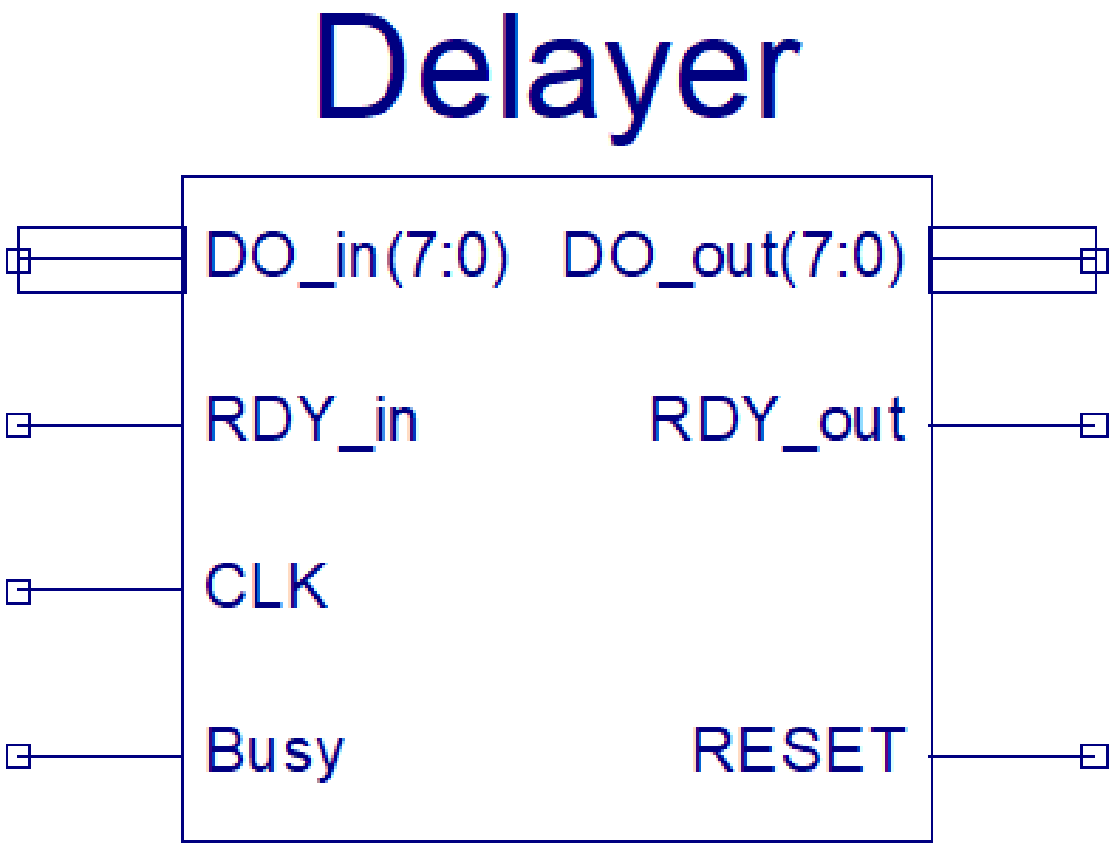
- **WAVreader** - obsługa karty SD
- **RS232** - obsługa klawiatury PS2
- **DACWrite** - obsługa głośnika
- **FSM_SendSamples** - pomocnicza maszyna stanów

4.2 Moduły własne, przygotowane na potrzeby projektu

- **Delayer** - moduł opóźniająco - przerywający do obsługi żądań odtwarzania dźwięków.

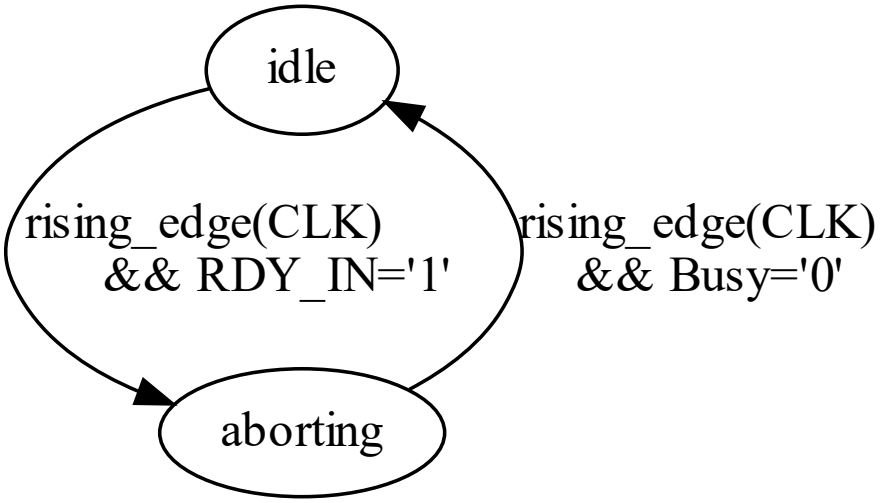
4.3 Kody źródłowe i opisy modułów

4.3.1 Delayer



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Delayer is
    Port ( DO_in : in  STD_LOGIC_VECTOR (7 downto 0);
           RDY_in : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           Busy : in  STD_LOGIC;
           DO_out : out  STD_LOGIC_VECTOR (7 downto 0);
           RDY_out : out  STD_LOGIC;
           RESET : out  STD_LOGIC);
end Delayer;

architecture Behavioral of Delayer is

constant CYCLE: time := 20 ns;

type state_type is (idle, aborting);
signal state, next_state : state_type;

begin

  process1: process( CLK )
  begin
    if rising_edge(CLK) then
      state <= next_state;
    end if;
  end process process1;

  process2: process( state, RDY_IN, Busy )
  begin
    next_state <= state;
    case state is
      when idle =>
        if RDY_in = '1' then
          RESET <= '1';
          next_state <= aborting;
        end if;
      when aborting =>
        if Busy = '0' then
          RESET <= '0';
          DO_out <= DO_in;
          RDY_out <= '1';
          next_state <= idle;
        end if;
    end case;
  end process process2;

end Behavioral;
```

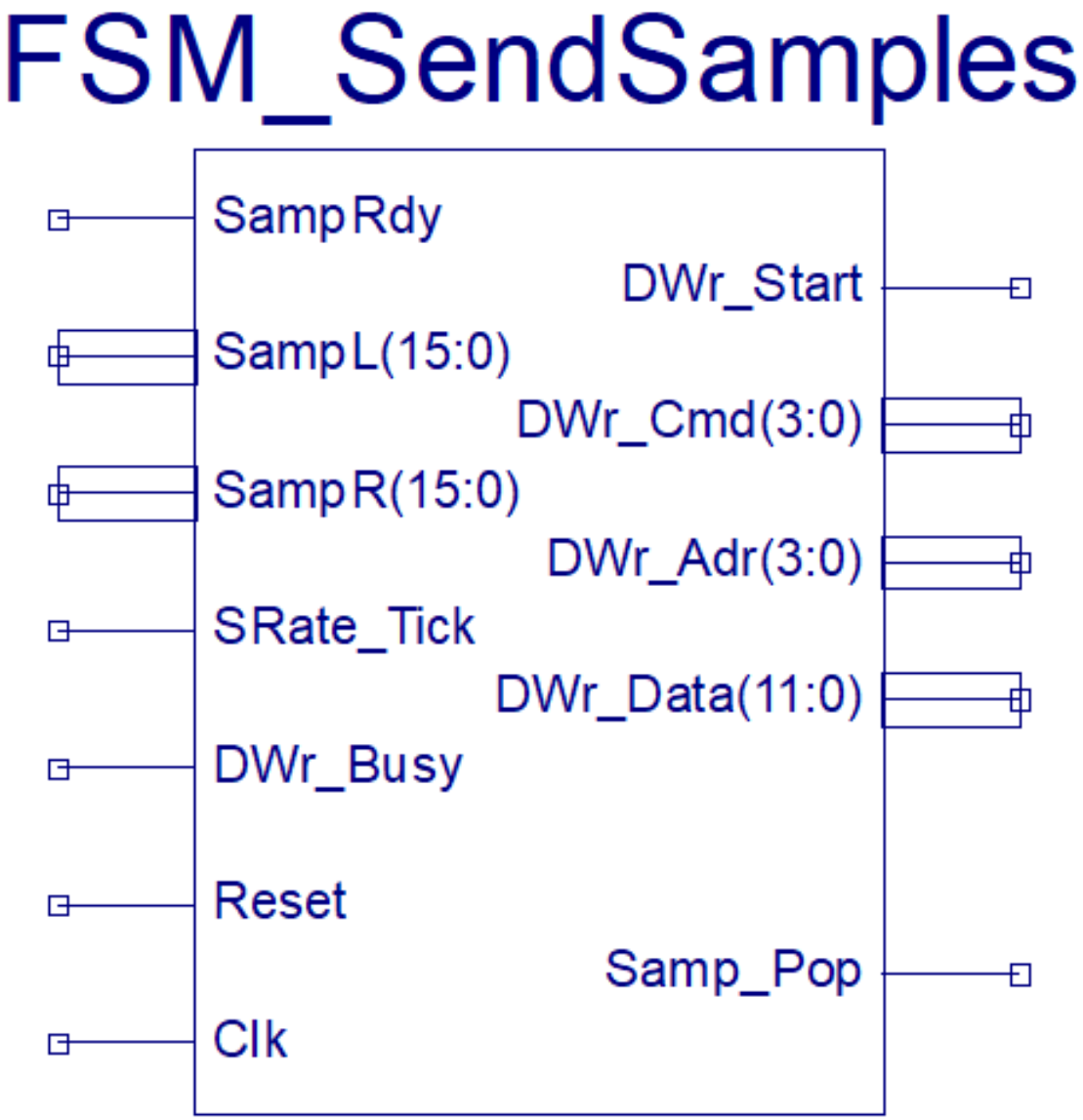
Moduł służy do opóźniania sygnału **RxDO(7:0)** i został zrealizowany jako poniższa maszyna stanów:



Logiczna jedynka na wejściu **RDY_IN** jest sygnałem powodującym przejście układu ze stanu *idle* do stanu *aborting* oraz ustawienie wyjścia **RESET**. Powrót do stanu *idle* następuje dopiero, gdy nastąpi zmiana na logiczne zero sygnału **Busy**, podanego jako sprzężenie zwrotne z układu **WAVreader**. Ma miejsce wtedy też wyzerowanie sygnału **RESET** i dopiero wtedy następuje podanie nowych wartości sygnałów **DO_out(7:0)** i **RDY_out**.

Sensem użycia modułu **Delayer** jest wspomożenie działania modułu **WAVreader**, który umożliwia przerwanie odtwarzania pliku, jednak powrót do stanu gotowości może i zazwyczaj nie jest natychmiastowy.

4.3.2 FSM_SendSamples




```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity FSM_SendSamples is
5      Port ( Clk, Reset : in  STD_LOGIC;
6             -- WAVreader:
7             SRate_Tick : in  STD_LOGIC;
8             SampRdy : in  STD_LOGIC;
9             SampL, SampR : in  STD_LOGIC_VECTOR (15 downto 0);
10         Samp_Pop : out STD_LOGIC;
11         -- DACWrite
12         DWr_Busy : in STD_LOGIC;
13         DWr_Start : out STD_LOGIC;
14         DWr_Cmd, DWr_Adr : out STD_LOGIC_VECTOR( 3 downto 0 );
15         DWr_Data : out STD_LOGIC_VECTOR( 11 downto 0 )
```

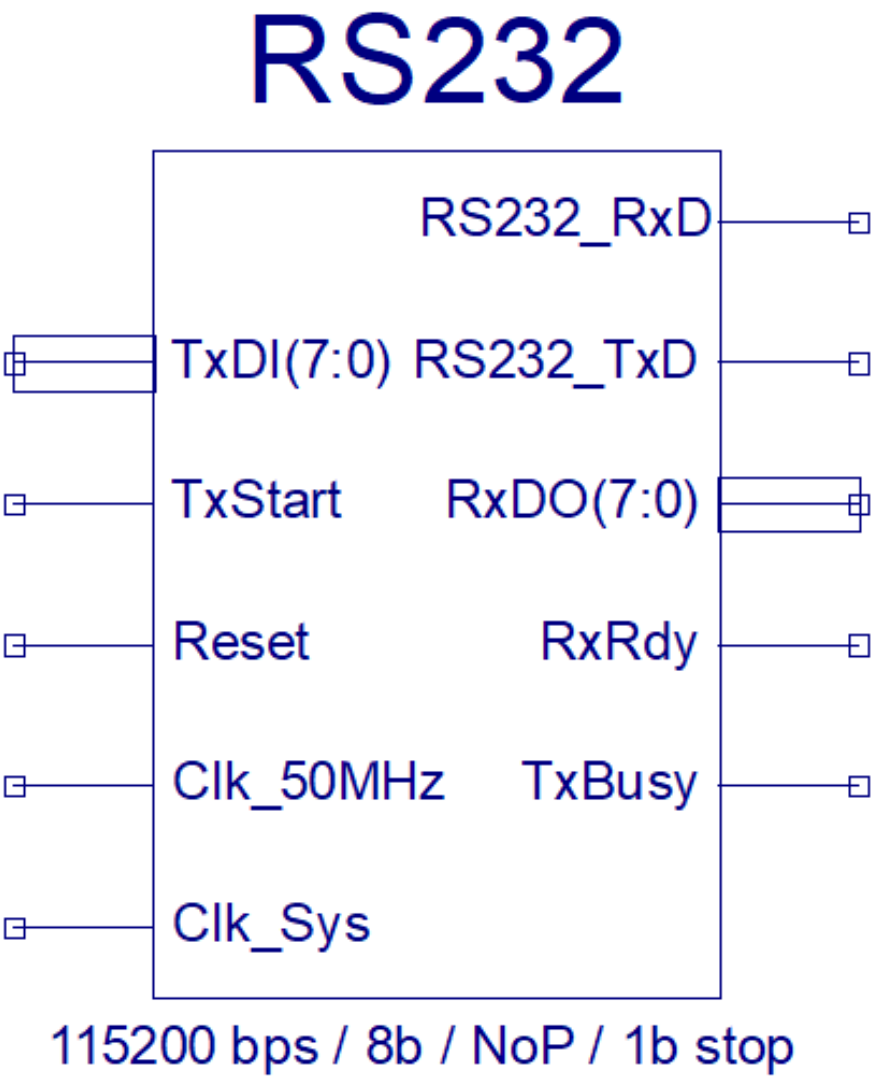
```
      );
end FSM_SendSamples;

architecture Behavioral of FSM_SendSamples is

  type state_type is ( sReset, sReady, sWaitL, sSendL, sWaitR, sSendR );
  signal State, NextState : state_type;

begin

  -- State register (with asynchronous reset) = process1
  process ( Clk, Reset )
  begin
    if Reset = '1' then
      State <= sReset;
    elsif rising_edge( Clk ) then
      State <= NextState;
    end if;
  end process;

  -- Next state decoding = process2
  process ( State, SampRdy, SRate_Tick, DWr_Busy )
  begin

    NextState <= State; -- default

    case State is
      when sReset =>
        NextState <= sReady;

      when sReady =>
        if SampRdy = '1' and SRate_Tick = '1' then
          NextState <= sWaitL;
        end if;

      -- Wait until DAC is ready
    when sWaitL =>
        if DWr_Busy = '0' then
          NextState <= sSendL;
        end if;

      -- Send left channel sample
    when sSendL =>
        NextState <= sWaitR;

      -- Wait until DAC is ready
    when sWaitR =>
        if DWr_Busy = '0' then
          NextState <= sSendR;
        end if;

      -- Send right channel sample
      when sSendR =>
        NextState <= sReady;

    end case;
```

```
72    end process;
73
74    -- Outputs
75    -- Pop FIFO with samples when sending the second (right) sample
76    Samp_Pop <= '1' when State = sSendR else '0';
77
78    -- DACWrite start: when sending either left or right channel sample
79    DWr_Start <= '1' when State = sSendL or State = sSendR else '0';
80    -- command: "write" when sending left sample, else "write&update"
81    DWr_Cmd <= "0000" when State = sSendL or State = sWaitR else "0010";
82    -- address: DAC C when sending left sample, else DAC D
83    DWr_Adr <= "0010" when State = sSendL or State = sWaitR else "0011";
84    -- data: left or right sample
85    DWr_Data <= SampL( 15 downto 4 ) when State = sSendL or State = sWaitR
          else SampR( 15 downto 4 );
86
87  end Behavioral;
```

Moduł pomocniczy ze strony kursu, sześciostanowa maszyna stanów służąca jako moduł wspomagający wysyłanie próbek do modułu **DACWrite**.

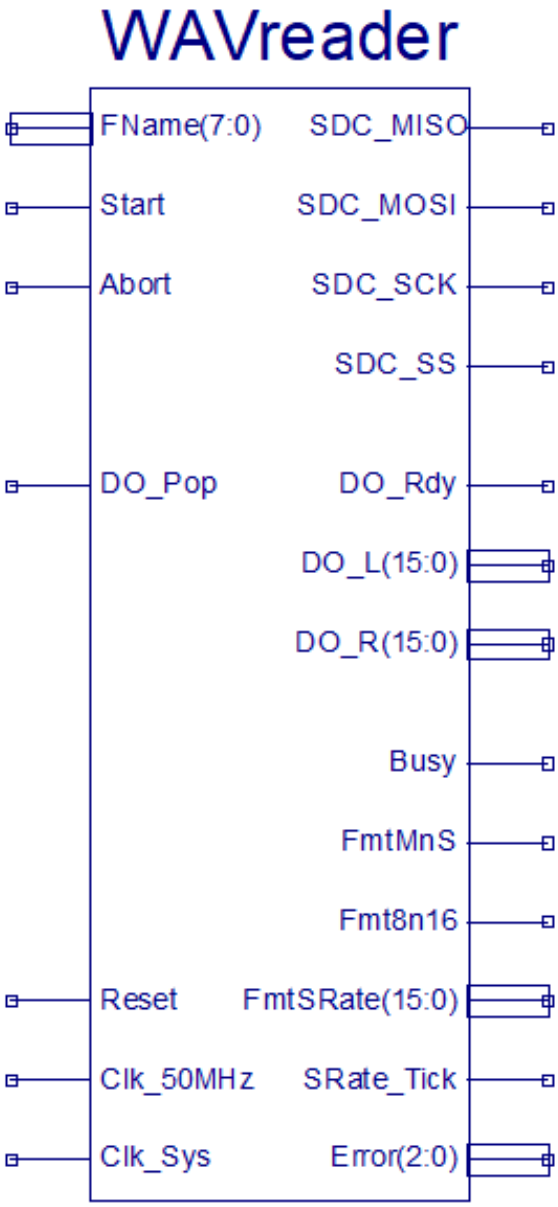
4.4 Omówienie modułów - czarnych skrzynek

4.4.1 RS232



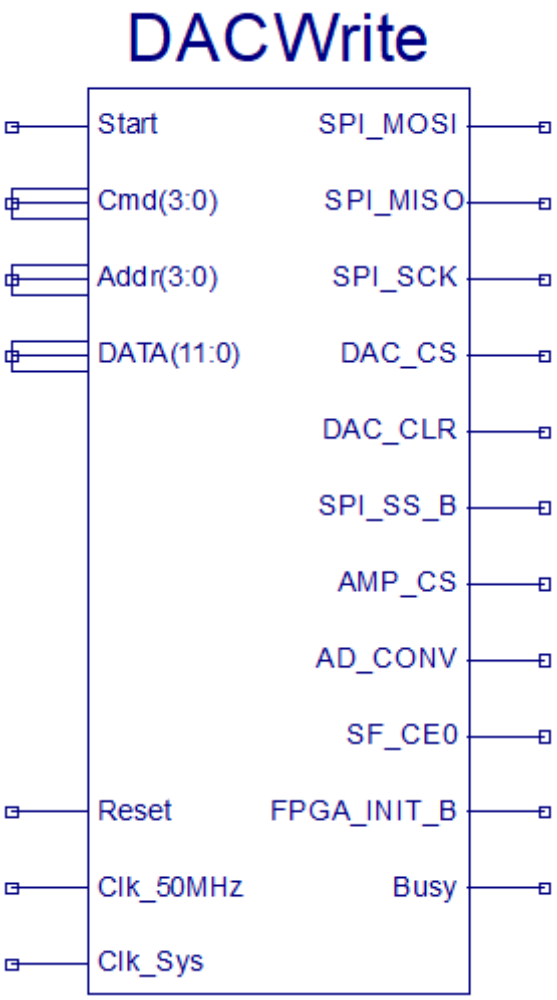
Moduł umożliwiający obsługę urządzeń podłączonych przez interfejs RS232. W projekcie wykorzystujemy go do odbioru bajtów, których źródłem jest klawiatura. Wyprowadzenia odpowiedzialne za wysył danych pozostają niewykorzystane.

4.4.2 WAVreader



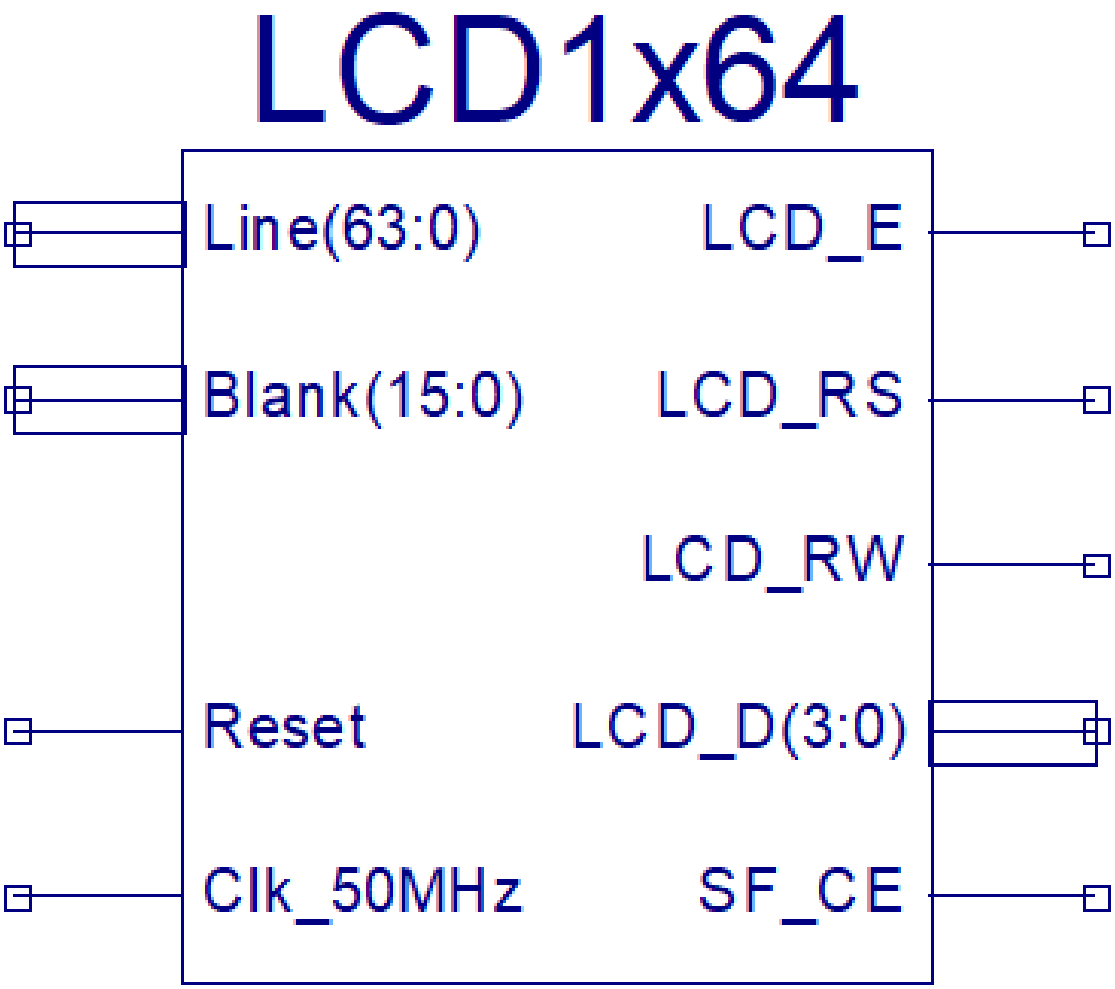
Moduł służący do obsługi karty SD; w momencie ustawienia jedynki na wejściu **Start** odczytywany jest kod ascii znaku z wejścia **FName**. Następnie plik o rozszerzeniu **.wav* i nazwie zadanej poprzez odczytany znak jest sczytywany z karty SD, a jego rozszerzone do 16 bitów próbki wysyłane na wyjścia **DO_L** i **DO_R**. W projekcie wykorzystujemy także funkcjonalność jaką daje wejście **Abort** - umożliwia ono przerwanie odtwarzania danego pliku. O zakończeniu procesu przerywania i powrocie modułu do stanu podstawowego informuje nas zero logiczne na wyjściu **Busy**.

4.4.3 DACWrite



Moduł obsługujący wysyłanie danych do przetwornika DAC LTC2624. W naszym projekcie wykorzystujemy go do obsługi głośnika.

4.4.4 LCD1x64



Moduł wspomagający testowanie urządzenia i jego testowanie; nie pełni żadnej innej istotnej roli w projekcie.

5 Instrukcja obsługi urządzenia

Do uruchomienia naszego urządzenia będą potrzebne:

- Płytka rozwojowa Spartan 3E Starter Board wyposażona w slot do odczytu karty SD;
- Plik **.bit* z konfiguracją naszego urządzenia;
- Program *jrs_term.exe*;

a także urządzenia zewnętrzne:

- Klawiatura obsługująca interfejs RS232;
- Głośnik;
- Karta SD z plikami ***.wav**;

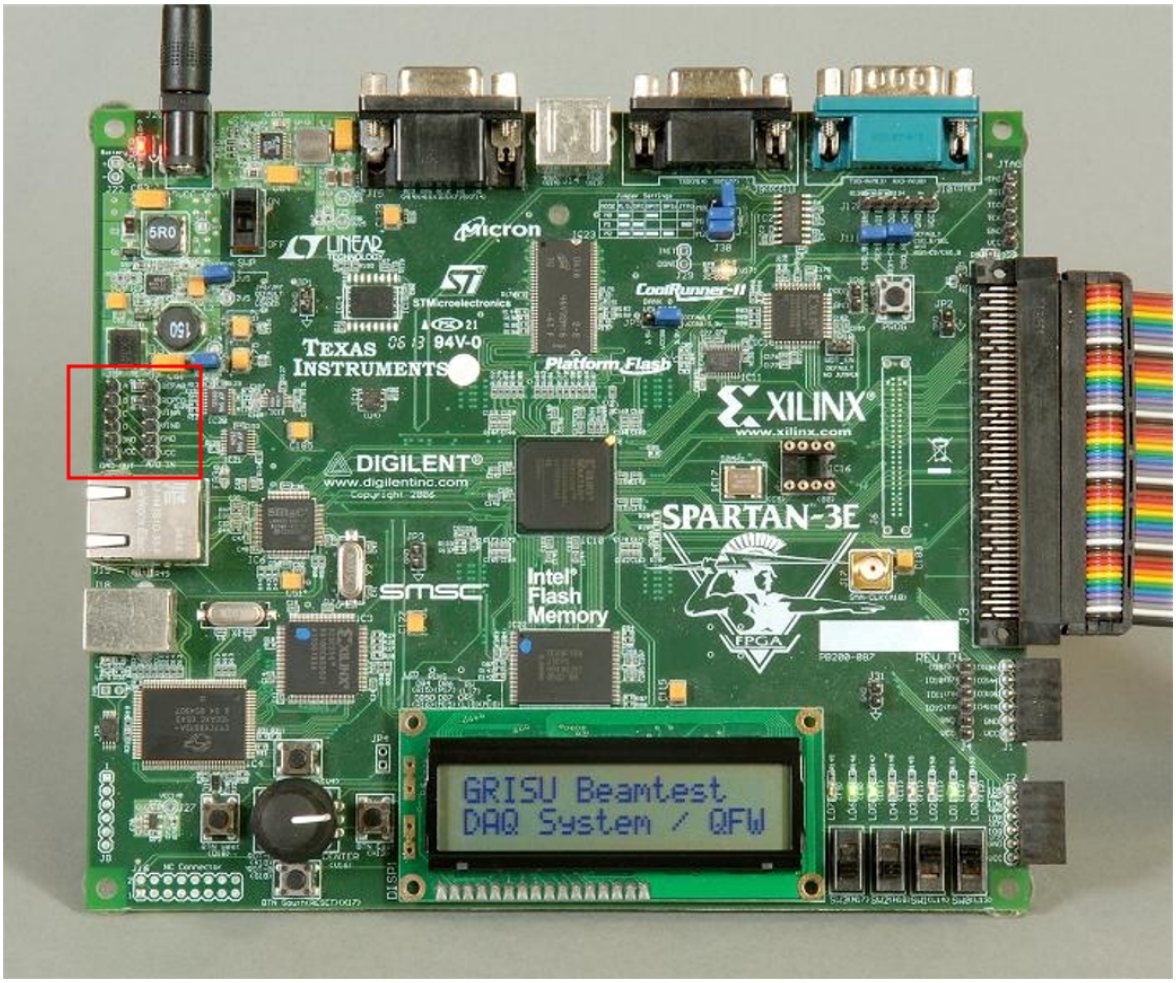
Lista kroków:

1. Wszystkie wymienione urządzenia należy podłączyć pod odpowiednie dla nich porty.
    - W przypadku głośnika jest to podłączenie pod piny C oraz GND z lewej strony płytki.
    - Kartę SD należy włożyć do odpowiedniego slotu; Należy wcześniej zadbać, aby była ona sformatowana w systemie FAT32; Odtwarzane z niej pliki **.wav* powinny mieć nazwę składającą się z jednej litery - jest to jednocześnie nazwa klawisza, który posłuży do jego odtworzenia.
    - Klawiaturę należy podłączyć pod interfejs RS232.
2. Gdy wszystkie urządzenia są podłączone, należy wgrać plik konfiguracyjny *instrument.bit* na płytkę.
3. Naciśnięcie klawiszy na klawiaturze powinno spowodować odtworzenie odpowiedniego dźwięku.

W testowanej przez nas konfiguracji przejęliśmy wykorzystanie następujących zakresów znaków:

- [12345678]- skrzypce;
- [qwertyui]- pianino;
- [asdfghjk]- syntezator;

Użytkownik może jednak całkowicie to zignorować i nagrać własne próbki dźwiękowe.



Rysunek 1: Miejsce podłączenia głośnika

6 Literatura

- The Programmable Logic Data Book. Xilinx, Inc.
- Libraries Guide. Xilinx, Inc.
- http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/
- http://gmvhdl.com/delay.htm
- https://vhdlwhiz.com/std_logic/