Wrocław University
of Science and Technology

Układy cyfrowe i systemy wbudowane 2 - Projekt

Organy wykonane na *Spartanie-3E*

Wydział Elektroniki
Kierunek: Informatyka

Skład grupy projektowej:
Jakub Sokołowski 226080
Paweł Szynal 226026

Prowadzący: dr inż. Jarosław Sugier

Spis treści

Spis rysunków

Listings

1.1 Wprowadzenie

1.1.1 Temat Pracy

Organy wykonane na *Spartanie-3E*

1.1.2 Cel projektu

Założeniem projektu było wykonanie prostych organów z wykorzystaniem układu Spartan3E w celach dydaktycznych. Jako grupa zdecydowaliśmy się na próbę implementacji jedno-gamowego piania obsługiwanego przez klawiaturę podłączaną do układu, z możliwością zapisywania i odtwarzania danego fragmentu na pamięci RAM urządzenia. Do odtwarzania naszej muzyki wykorzystaliśmy przetwornik DAC i podłączony do niego głośnik.

Prace nad projektem podzieliliśmy na pomniejsze etapy, tj.:

- Generacja sygnału 1,5 kHz,
- Wyprowadzenie sygnału z modułu Podzielnika Częstotliwości,
- Obsłużenie przetwornika DAC,
- Obsługa klawiatury jako wejścia,
- Implementacja pamięci RAM,
- Odtwarzanie z pamięci,
- Nagrywanie, odgrywanie.

1.1.3 Opis Sprzętu i Narzędzi

Wykorzystywany sprzęt:

- Spartan-3E FPGA Starter Jest to w pełni funkcjonalny układ FPGA firmy Digilent. Wyposażony jest w 32-bitowy procesor RISC i interfejsy DDR. Płyta posiada również interfejsy programowania równoległego Xilinx Platform. Płyta jest w pełni kompatybilna ze narzędziami, Xilinx ISE® i ModelSim XE.
- Klawiatury komputerowej Podłączona do zestawu na porcie PS2
- Głośnik firmy Tonsil Połączony do pinów GND i DAC-C

Wykorzystywane narzędzia:

- Xilinx ISE® Jest to zintegrowane środowisko programistyczne służące do wykonywania wszystkich operacji związanych z przygotowaniem projektu w układzie FPGA
- ModelSim XE Symulator, który oblicza i przedstawia graficznie wygenerowany przez układ odpowiedzi na portach wyjściocwych.

1.1.4 Opis interfejsów i używanych protokołów

Jak już zostało wspomniane powyżej do interakcji z Naszym układem wymagana jest klawiatura komputerowa z wtyczką PS2 oraz głośnik. Parametry klawiatury czy głośnika, jak i ich właściwości nie mają dla poprawnej pracy układu żadnego znaczenia. W przypadku klawiatury wynika to z standardu jakim jest PS2, a w przypadku głośnika, jest to standard przetwornika DAC. Tym samym nie ma potrzeba zamieszczania danych katalogowych opisywanych elementów.

1.2 Realizacja Projektu

1.2.1 Zajęcia nr 1

Omówienie pracy i BHP

1.2.2 Zajęcia nr 2

Pierwszym celem naszego projektu było zaznajomienie się z modułem DAC i Sawtooth aby za pomocą ich wygenerować falę piło-kształtna o zadanej częstotliwości.

Sygnały zegarowe

Głównym i zarazem podstawowym wykorzystywanym układ jest generator częstotliwości służącej taktowaniu układu. W naszym projekcie korzystaliśmy z podstawowego sygnału zegarowego o częstotliwości 50 MHz.

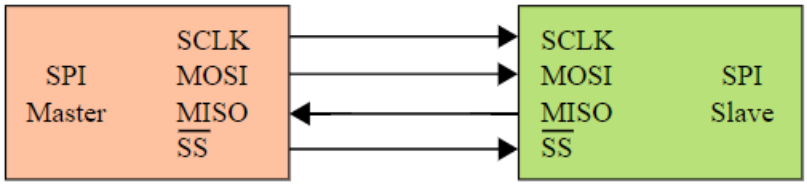
W oparciu o podstawowy zegarowy sygnał wejściowy i częstotliwość dźwięku, tworzony był kolejny sygnał zegarowy.

Przetwornik i Moduł DAC

W ramach przygotowania do projektu zaznajomiliśmy się z modułem DAC. Jak sama nazwa wskazuje jest to przetwornik cyfrowo-analogowy (z ang. Digital to Analog Converter, DAC) Innymi słowy jest to układ przetwarzający dyskretny sygnał cyfrowy na równoważny mu sygnał analogowy. Spartan-3E zawiera kompatybilny czterokanałowy, szeregowy konwerter cyfrowo-analogowy. Komunikacja SPI:

FPGA wykorzystuje interfejs komunikacji szeregowej (SPI -szeregowy interfejs urządzeń peryferyjnych) do czterech kanałów DAC. Magistrala SPI jest w pełni dupleksowa, synchroniczna. Nasz przetwornik ma 12-bit rozdzielczość i korzysta z SPI jako czterokanałowego sposobu przesyłu danych.

Zobrazowanie SPI:



Rysunek 1.1: SPI

- MOSI (ang. Master Output Slave Input) – dane dla układu peryferyjnego,
- MISO (ang. Master Input Slave Output) – dane z układu peryferyjnego,
- SClk (ang. Serial CLock) – sygnał zegarowy (taktujący),
- SS (ang. Slave Select) - wybór układu podrzędnego

W naszym wypadku to SPI działa troszkę inaczej. Dodane są linie DAC_CS i DAC_CLR. CS oznacza początek przesyłu(dla stanu wysokiego), a CLR to typowy Clr. SPI przesyła 12-bitowe dane, potem 4 bity określający kanał przetwornika DAC na który ma to trafić.

Listing 1.1: UCF Location Constraints

```
NET "SPI_MISO" LOC = "N10" | IOSTANDARD = LVCMOS33 ;
NET "SPI_MOSI" LOC = "T4" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "SPI_SCK" LOC = "U16" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "DAC_CS" LOC = "N8" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "DAC_CLR" LOC = "P8" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
```

1.2.3 Zajęcia nr 3

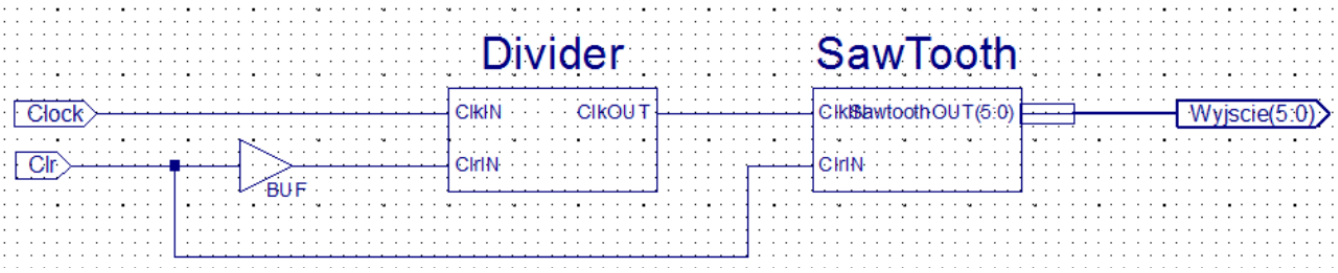
Sawtooth Wave

Fala piło-kształtna jest rodzajem fali niesinusoidalne zaś jest tak nazwana w oparciu o jego podobieństwo do zębów piły. Moduł przesyła odpowiednio przygotowany sygnał piło-kształtny, który następnie poprzez działanie modułu DAC wygrywany jest na głośniku dlatego można powiedzieć że dla generacji dźwięku jest to najważniejszy moduł ze wszystkich.

Wykorzystywany przez nas wzór do wygenerowania piły:

$$x(t) = t - \underbrace{\lfloor t \rfloor}$$

Schemat



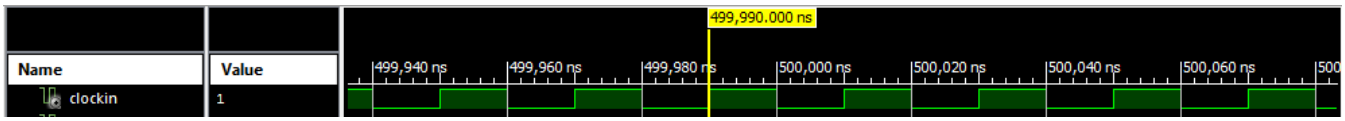
Rysunek 1.2: Schemat testowego sawTooth

Devider jest modułem testowym przed wykonaniem organków w który odpowiadał za opdzielenie 50MHz na inne częsttliwości. Idea dość prosta i sprowadzająca się do prostego rachunku. 50MHz daje nam 20ns okresu zegarowego.

Listing 1.2: Clock_processs w module Divider

```
Clock_processs: process
  begin
          Clock <= '0';
          wait for Clock_period/2;
          Clock <= '1';
          wait for Clock_period/2;
  end process;

  % Clock_period zostalo zadeklarowane przed tym jako stala
  % constant Clock_period : time := 20ns;
```

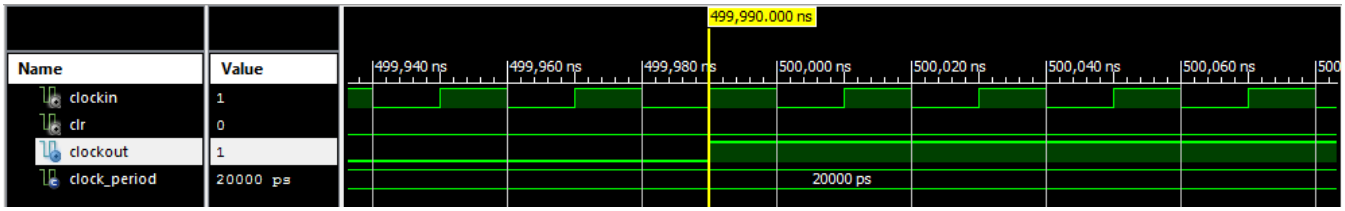


Rysunek 1.3: Clock_period

Dzielenie częstotliwości

Listing 1.3: Dzielenie częstotliwości

```
process( ClkIN, ClrIN, temp )
begin
        if(ClrIN = '1') then
                iterator <= 1;
                temp <= '0';
        elsif rising_edge(ClkIN)  then
                iterator <= iterator + 1;
                if (iterator = frequency) then
                        temp <= NOT temp;
                        iterator <= 1;
                end if;
        end if;
ClkOUT <= temp;
end process;
```

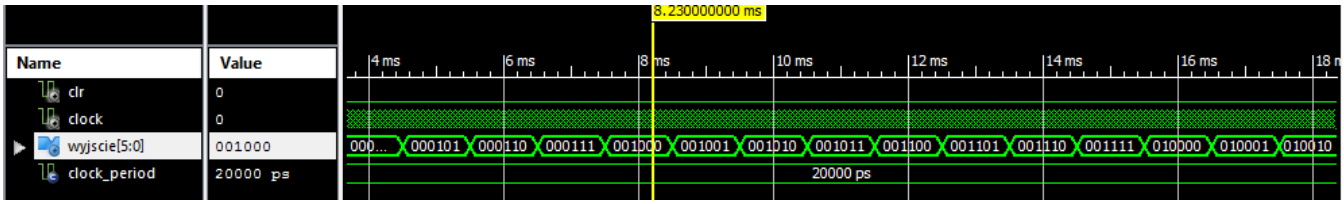


Rysunek 1.4: Test dzielenia częstotliwości

Generacja piły

Listing 1.4: Generacja piły

```
process(ClockIN, ClrIN)
begin
    if(ClrIN = '1') then
        iteratorProbek <= 0;
    elsif(rising_edge(ClockIN)) then
        if(iteratorProbek = 63) then
            iteratorProbek <= 0;
        else
            iteratorProbek <= iteratorProbek + 1;
        end if;
    end if;
end process;
```



Rysunek 1.5: Generacja piły

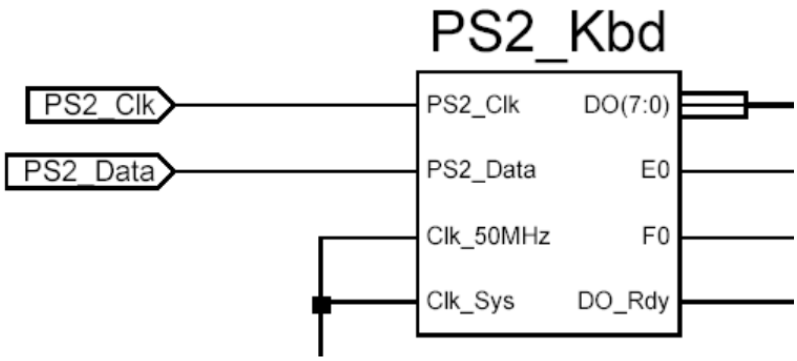
1.2.4 Zajęcia nr 4

Kolejnym zadaniem naszego projektu było zaimplementowanie modułów potrzebnych do obsługi organków. Potrzebowaliśmy do tego zadania następujących modułów:

- Moduł będącego odbiornikiem kodów wysyłanych przez klawiaturę PS/2,
- Moduł obsługujący jedną gamę na podstawie klawiszy,
- Moduł generujący dźwięki,
- Moduł przeznaczony jest do wyprowadzeń zewnętrznych FPGA-naszego głośnika.

Moduł PS2_Kbd

Moduł ten jest odbiornikiem kodów wysyłanych przez klawiaturę PS/2. Został on pobrany ze strony http://www.zsk.ict.pwr.wroc.pl



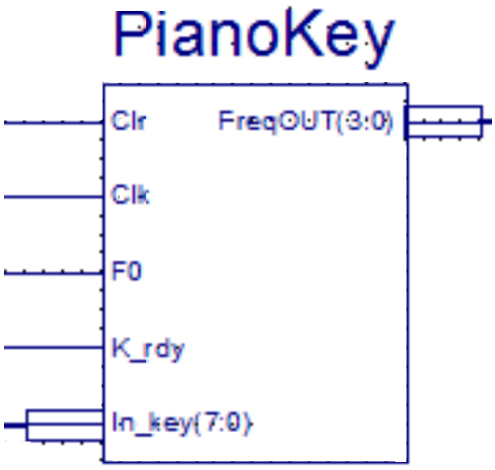
Rysunek 1.6: Moduł Ps2

- DO(7:0) - Przechowuje wartość odbieranego kodu.
- E0 - Flaga sygnalizująca czy wchodzący sygnał był poprzedzony bajtami X"E0" (tzw. kod rozszerzony).
- F0 - Flaga sygnalizująca czy nastąpiło zwolnienie naciśniętego klawisza.
- Sygnał DO_Rdy - Sygnalizuje zakończenie odbioru kodu (impuls jednotaktowy).

Moduł PianoKey

Zadaniem tego modułu jest obsługa maszyny stanów klawiszy pianina. Po podaniu zbocza rosnącego na wejście *K_rdy* zaczytywany jest kod skaningowy klawiatury. W zależności od wciśniętego klawisza, ustawiany jest odpowiedni stan. Oprogramowana została jedna oktawa tzn. 12 klawiszy co daje nam 12 stanów odpowiadających wciśniętym klawiszom, oraz jeden stan odpowiadający braku wciśniętego klawisza - stan spoczynku. Klawisze klawiatury które odpowiadają dźwiękom pianina to (A, W, S, E, D, F, T, G, Y, H ,U , J). Nazwy stanów odpowiadają wciśniętym klawiszom. Po wciśnięciu klawisza, którego należy do obsługiwanej oktawy następuje przejście do stanu (*next_state* w kodzie) odpowiadającego klawiszowi. Jeśli wciśnięty zostanie nieobsługiwany klawisz, maszyna pozostaje w stanie spoczynku (oznaczonym jako *DEF*). Przejście w stan w spoczynku odbywa się również po puszczeniu klawisza. Puszczenie klawisza sygnalizuje wejście *F0*. Zarządzanie stanami wykonuje proces *StMch*, którego kod znajduje się na listingu 1.5.

Ponieważ po stanie wciśnięciu klawisza zawsze występuje stan spoczynku, na takim pianinie nie jest możliwe granie akordów - naciśnięcie kilku klawiszy jednocześnie.



Rysunek 1.7: Moduł PianoKey

Wejścia:

- Clr - reset,
- Clk – systemowy zegar,
- K_rdy – odebranie sygnału o odczytaniu klawisza z klawiatury na porcie PS2,
- IN_klaw – kod skaningowy klawisza odczytanego z klawiatury,
- F0 – kod zwolnienia klawisza.

Wyjścia:

- FreqOUT – wyjście, będące tłumaczeniem wciśniętego klawisza na konwencję używaną dalej.

Listing 1.5: Fragment procesu StMch

```
StMch : process( state, In_key, K_rdy, tmp )
   begin
      next_state <= state;
      case state is
         when Def =>
            if K_rdy = '1' and In_key = X"1C" then
               next_state <= A;
            .....

            elsif K_rdy = '1' then
               next_state <= DEF;
            end if;
         when A =>
            if F0 = '1' then
               next_state <= DEF;
            end if;
            .....
         when K =>
            if F0 = '1' then
               next_state <= DEF;
            end if;
      end case;
end process StMch;
```

W zależności od obecnego stanu, należy wygenerować sygnał wyjściowy *FreqOut* odpowiadający wciśniętemu klawiszowi. Wartość 0 oznacza stan spoczynku - ciszę, a wartości 1-12 kolejne klawisze - dźwięki. Do zakodowania 13 różnych stanów potrzeba co najmniej 4 bitów, więc sygnał wyjściowy *FreqOut* jest 4-bitowym wektorem. Proces konwertujący stan na sygnał jest przedstawiony na listingu 1.6.

Listing 1.6: Fragment procesu OutPr

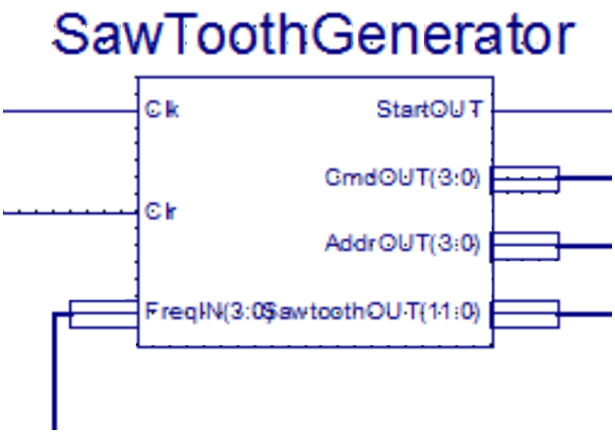
```
OutPr : process ( state )
   begin
      case state is
         when A =>
            FreqOUT <= conv_std_logic_vector(1,4);
         when W =>
            FreqOUT <= conv_std_logic_vector(2,4);

        ...

         when DEF =>
            FreqOUT <= conv_std_logic_vector(0,4);
         end case;
   end process OutPr;
```

SawToothGenerator

Jest to najważniejszy moduł ze wszystkich. Odpowiedzialny jest jest za generowanie dźwięków za pomocą wygenerowanego wektora sygnału piło-kształtnego.



Rysunek 1.8: Moduł SawToothGenerator

Wejścia:

- Clk – wejście zegara systemowego,
- Clr - reset,
- FreqIN – wejście kodu oznaczające częstotliwość dźwięku.

Wyjścia:

- AddrOUT – wyjście addresowe,
- CmdOUT – wyjście rozkazu,
- StartOUT – wyjście Start,
- SawtoothOUT – wektor sygnału piłokształtnego.

Powyższy moduł składa się z trzech procesów:

- SetFreq,
- FreqDiv,
- SawToothGen,

SetFreq Pierwszy zaimplementowany proces-SetFreq jest szeregiem operacji warunkowych-elsif odpowiedzialnych za przypisanie częstotliwości w zależności od wciśniętego przycisku na klawiaturze. Wejściowy FreqIN przechowuje wartość klawisza zaś Freq ma przypisaną wartość oznaczającą częstotliwość konkretnego dźwięku, zgodną z oktawą na Pianie. [4] Na listingu 1.7 znajduje się proces który przekształca kod dźwięku na jego częstotliwość. Częstotliwość dźwięku generowana poprzez sumowanie impulsu piły o częstotliwości 1,5 Hz. Przykładowo, jeśli chcemy uzyskać za pomocą sygnału 1.5 hz dźwięk *C2* który ma częstotliwość 131 Hz należy dodać 1.5 Hz 87 razy. Analogicznie, w najwyższym dźwięku oktawy - *C3* o częstotliwości 262 Hz znajdą się 173 składowe 1.5 Hz ($262/1.5 \approx 173$). Liczba ile razy należy dodać 1.5 Hz by otrzymać dany dźwięk jest w kodzie oznaczana jako *Freq*. W tym projekcie oprogramowano klawisze od *C2* do *C3* o częstotliwościach od 131 do 262 Hz - odpowiadający im zakres *Freq* to 87-173.

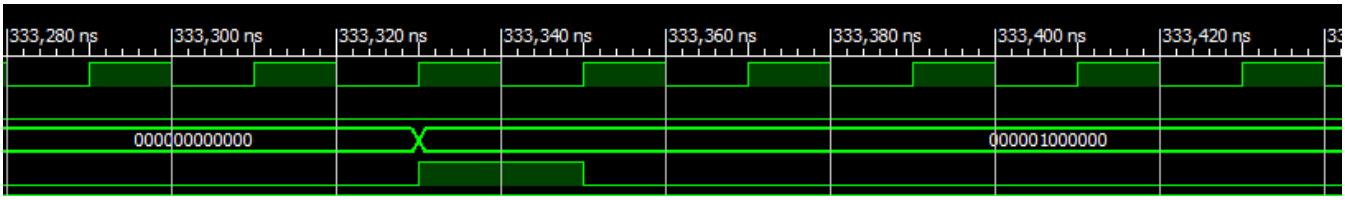
Listing 1.7: Proces SetFreq

```
SetFreq: process(FreqIN, Freq)
begin
    if(FreqIN = "0001") then
            Freq <= 173;
    elsif(FreqIN = "0010") then
            Freq <= 164;
    elsif(FreqIN = "0011") then
            Freq <= 154;
    elsif(FreqIN = "0100") then
            Freq <= 146;
    elsif(FreqIN = "0101") then
            Freq <= 137;
    elsif(FreqIN = "0110") then
            Freq <= 130;
    elsif(FreqIN = "0111") then
            Freq <= 123;
    elsif(FreqIN = "1000") then
            Freq <= 116;
    elsif(FreqIN = "1001") then
            Freq <= 109;
    elsif(FreqIN = "1010") then
            Freq <= 103;
    elsif(FreqIN = "1011") then
            Freq <= 97;
    elsif(FreqIN = "1100") then
            Freq <= 92;
    elsif(FreqIN = "1101") then
            Freq <= 87;
    else
            Freq <= 0;
    end if;
end process;
```

FreqDiv Zadaniem procesu FreqDiv jest w oparciu o ustaloną częstotliwość, i zegarowy sygnał wejściowy utworzenie sygnału zegarowego o częstotliwości zgodnej z parametrem Freq będący wyjściem procesu SetFreq. Dodatkowo w tym samym procesie, tj. FreqDiv opisywany jest sygnał StartOUT, którego obecność wynika z modułu DACWrite. StartOUT jest sygnałem jednotaktowym dla zegara systemowego wygenerowanym za każdym razem jak zegar o częstotliwości wejściowej FreqIN zmieni swój stan.

Listing 1.8: Proces FreqDiv

```
FreqDiv: process( Clk, Clr, tmpFreqDiv )
begin
    if(Clr = '1' ) or (Freq = 0) then
        iFreqDiv <= 1;
        tmpFreqDiv <= '0';
        StartOUT <= '0';
    elsif rising_edge(Clk)  then
        if (iFreqDiv = Freq) then
            tmpFreqDiv <= NOT tmpFreqDiv;
            iFreqDiv <= 1;
            if (stCnt mod 2 = 0) then
                StartOUT <= '1';
                stCnt <= stCnt + 1;
            else
                stCnt <= stCnt + 1;
            end if;
        else
            iFreqDiv <= iFreqDiv + 1;
            StartOUT <= '0';
        end if;
    end if;
ClkSawTooth <= tmpFreqDiv;
end process;
```



Rysunek 1.9: Test dzielenia częstotliwości

SawToothGen Jest to ostatni proces w naszym module SawToothGenerator. Zadaniem tego procesu jest utworzonie sygnału piłokształtnego zgodnego z sygnałem zegarowym z poprzedniego procesu-FreqDiv. Sygnał piłokształtny generowany jest na zasadzie inkrementowanego wektora, który po 64 okresie się resetuje z powrotem.

Listing 1.9: Process: FreqDiv

```
SawToothGen: process( ClkSawTooth, Clr )
begin
    if(Clr = '1') then
        iSawGen <= 0;
    elsif(rising_edge(ClkSawTooth)) then
        if(iSawGen = 63) then
            iSawGen <= 0;
        else
            iSawGen <= iSawGen + 1;
        end if;
    end if;
end process;
```

Sygnały wyściowe W ostatniej części tego modułu jest wysyłanie sygnałów wyjściowych CmdOUT i AddrOUT. Sygnały te są potrzebne dla modułu DACWrite. Tutaj także wysyłany jest sygnał SawToothOUT będący sygnały piłokształtnym z poprzedniego procesu po operacji konkatenacji z sześcioma zerami umieszczonym na najmniej ważnych pozycjach. Operacja konkatenacji wynika ponownie z potrzeby dostarczenia tego typu sygnału do modułu DACWrite.

Listing 1.10: Sygnały wyściowe

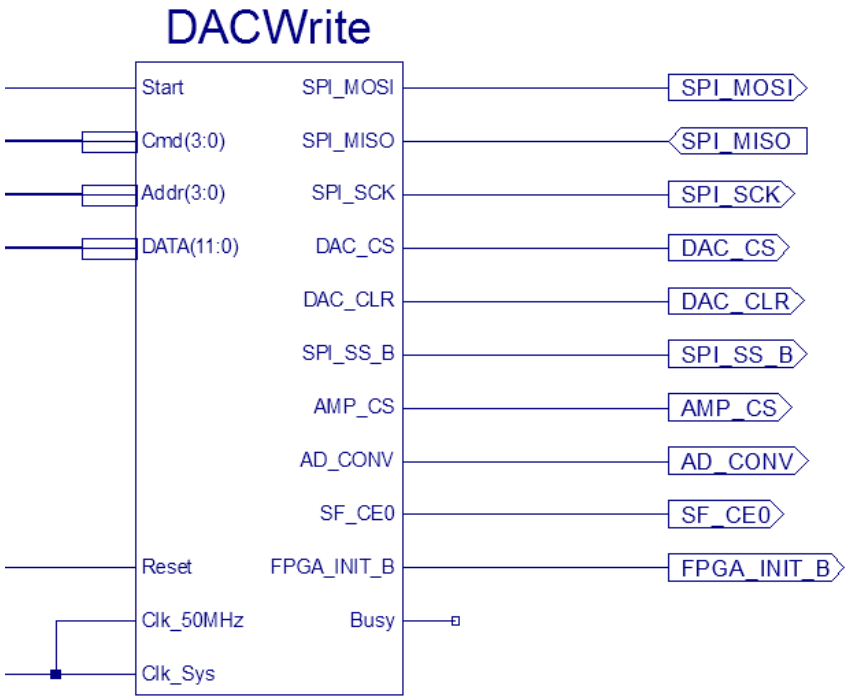
```
CmdOUT <= conv_std_logic_vector(Cmd,4);
AddrOUT <= conv_std_logic_vector(Addr,4);
SawtoothOUT <= conv_std_logic_vector(iSawGen,6) & "000000";
```

Moduł DAC

Niezbędny do realizacji naszego projektu było zaznajomienie się z przetwornikiem cyfrowo-analogowym (DAC), który konwertuje sygnał cyfrowy na sygnał analogowy. Był on kluczowy dla naszych organów ponieważ przetwornik DAC jest powszechnie używany w odtwarzaczach muzycznych do konwersji cyfrowych strumieni danych na analogowe sygnały audio. Obsługa przetwornika DAC odbywa się za pomocą magistrali SPI.

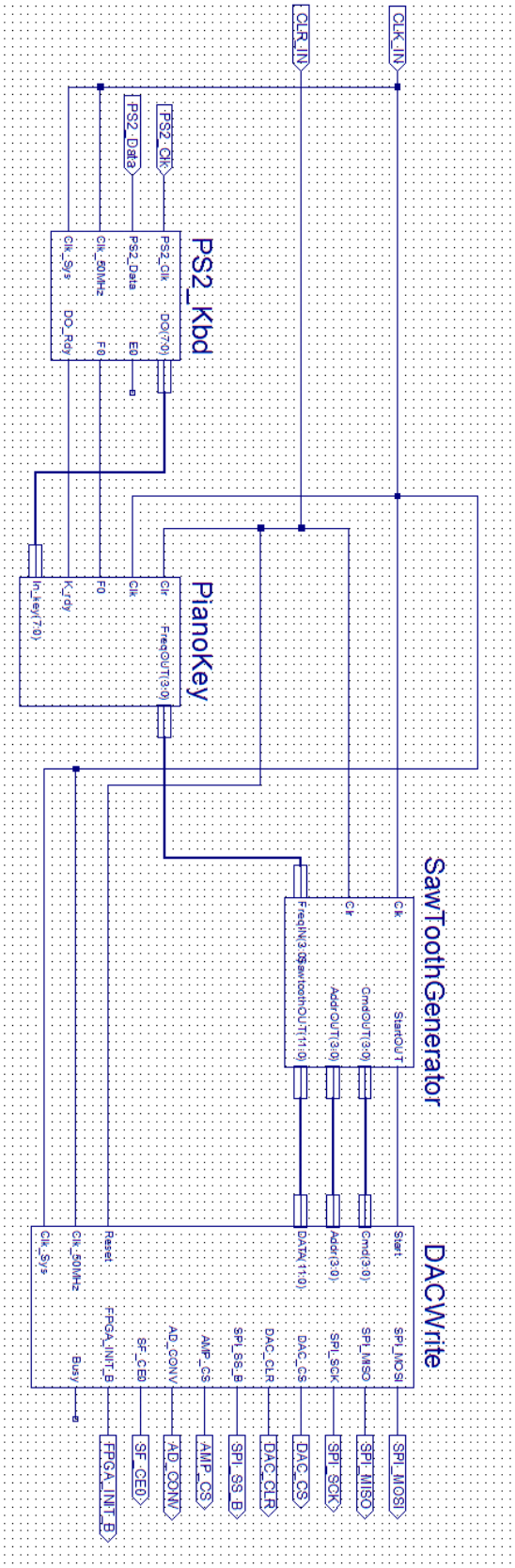
Moduł DacWrite został pobrany ze strony Zespołu Systemów Komputerowych i był odpowiedzialny za komunikacje z przetwornikiem cyfra/analog LTC264 w naszym układzie. W zależności od stanu wejścia *Start*, dane z wejść Cmd, Addr i Data są zatrzaskiwane i przesyłane szeregowo.

- Start - zatrzaskuje wartości na pozostałych wejściach
- Cmd(3:0) - polecenie jakie można wydać przetwornikowi, w tym wypadku będzie to polecenie '0011' aktualizuje wybrane wyjście DAC zawartością DATA
- Addr(3:0) - adres pinu na który wysyłany jest sygnał. Na to wejście również podawana jest stała wartość '0010' która oznacza pin DAC-C, do którego podłączony jest głośnik
- DATA(11:0) - dane przekazywane na pin, w tym wypadku wysokość generowanego dźwięku



Rysunek 1.10: Moduł DACWrite

1.3 Schemat


CLR_IN
CLK_IN
PS2_Data
PS2_Clk
PS2_Kbd
PianoKey
SawToothGenerator
DACWrite
SPI_MOSI
SPI_MISO
SPI_SCK
DAC_CS
DAC_CLR
SPI_SS_B
AMP_CS
AD_CONV
SF_CE0
FPGA_INIT_B

1.4 Podsumowanie

SawToothSchematic Project Status (04/01/2019 - 09:15:38)			
Project File:	ucisw2-organki.xise	**Parser Errors:**	No Errors
Module Name:	SawToothSchematic	**Implementation State:**	Programming File Generated
Target Device:	xc3s500e-4fg320	• **Errors:**	No Errors
Product Version:	ISE 14.7	• **Warnings:**	44 Warnings (2 new)
Design Goal:	Balanced	• **Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	• **Timing Constraints:**	All Constraints Met
Environment:	System Settings	• **Final Timing Score:**	0 (Timing Report)

Rysunek 1.11: Stan projektu

Device Utilization Summary

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	140	9,312	1%
Number of 4 input LUTs	187	9,312	2%
Number of occupied Slices	142	4,656	3%
Number of Slices containing only related logic	142	142	100%
Number of Slices containing unrelated logic	0	142	0%
Total Number of 4 input LUTs	226	9,312	2%
Number used as logic	187		
Number used as a route-thru	39		
Number of bonded IOBs	14	232	6%
Number of BUFGMUXs	2	24	8%
Average Fanout of Non-Clock Nets	2.74		

Tablica 1.1

1.5 Wnioski

Udało nam się wykonać poprawnie główną część projektu czyli w pełni funkcjonalne jedno-gamowe organki. Część ta wykonana została stosunkowo szybko - organki zagrały pierwszy raz już na czwartych zajęciach. Niestety, po zabraniu się za tworzenie pozytywki, natrafiliśmy na wiele problemów z poprawnym przekierowaniem sygnałów odczytanych z ROM. Funkcja ta wymagała obejścia obecnych modułów i nie mogliśmy sobie z tym poradzić, bez znaczącej degradacji istniejącego kodu. Duży problem też stwarzało to, że praca nad projektem na płytce była możliwa tylko w czasie zajęć - pomysły i rozwiązania na które wpadaliśmy po zakończeniu zajęć mogły być zweryfikowane dopiero dwa tygodnie później.

Bibliografia

[1] Spartan-3E FPGA Starter Kit Board User Guide,
https://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf

[2] LTC2624 Manual,
http://cds.linear.com/docs/en/datasheet/2604fd.pdf

[3] Moduł PS2_Kbd, DACWrite,
http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga

[4] Tablica częstotliwości dźwięków pierwszej oktawy gamy C-dur,
https://www.liutaiomottola.com/formulae/freqtab.htm