Układy Cyfrowe i Systemy Wbudowane 2

Projekt

Organy z pozytywką

Maja Bojarska, 241287

Damian Koper, 241292

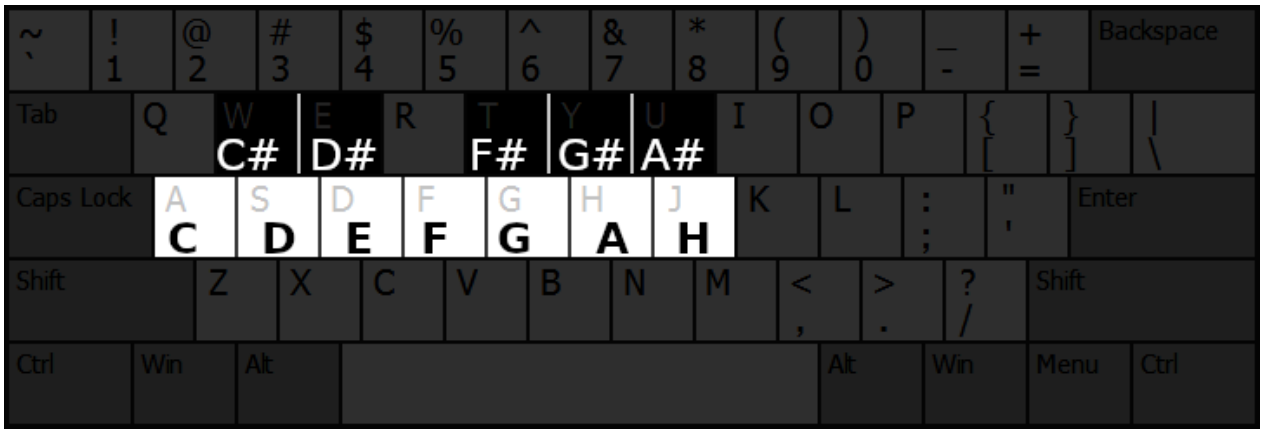
7 maja 2020

1 Cel projektu

Celem projektu było wykonanie układu realizującego działanie organów, sterowanych za pomocą klawiszy klawiatury, podłączonej poprzez interfejs PS2. Rozszerzeniem działania układu było odtwarzanie sekwencji dźwięków odczytanej z pliku tekstowego, zapisanego na karcie pamięci typu SD.

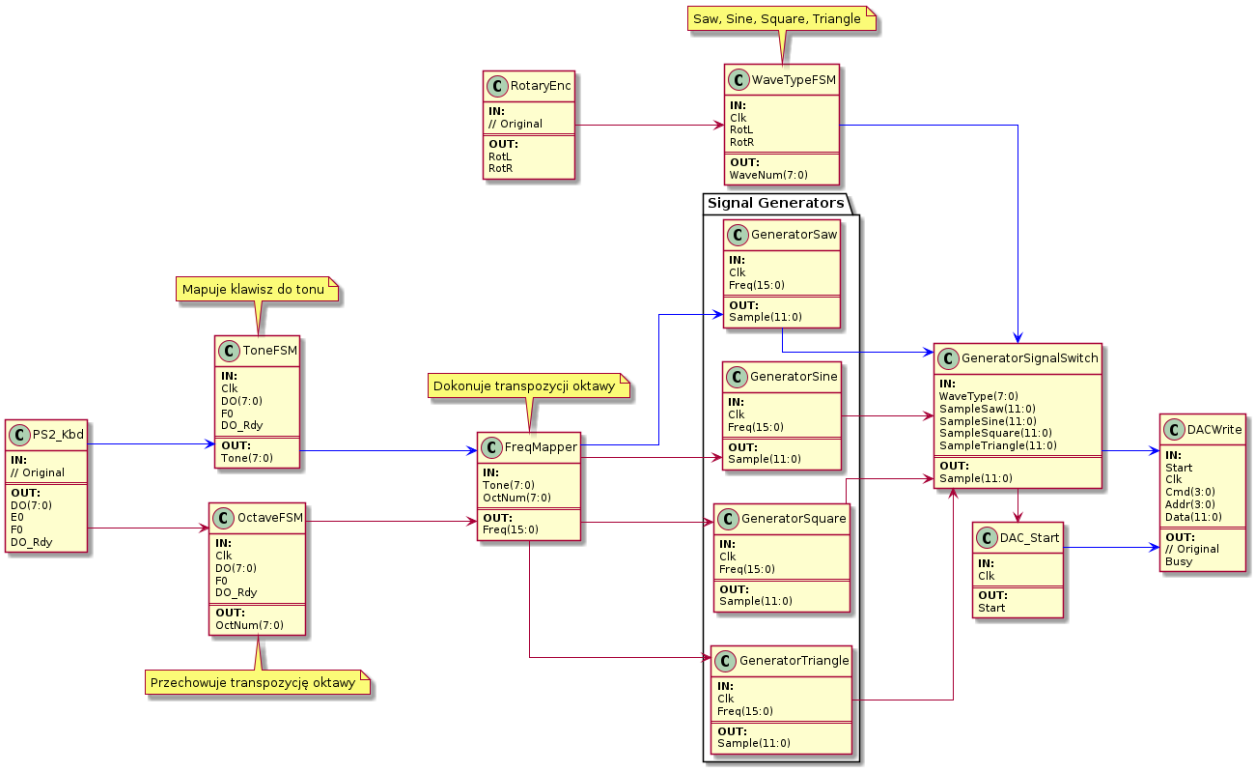
1.1 Założenia wstępne

Początkowy projekt układu zakładał podział kolejnych funkcjonalności na możliwie małe moduły, według zasady pojedynczej odpowiedzialności. Układ mapował klawisze klawiatury na odpowiadające im dźwięki. Klawisze były przypisane do dźwięków, zgodnie z układem przybliżonym do klawiszy jednej oktawy pianina. Oktawa zmieniana była za pomocą klawiszy strzałek w zakresie od 0 do 8.



Rysunek 1: Przypisanie tonów oktawy do klawiszy klawiatury QWERTY. Poniżej każdej litery klawisza znajduje się odpowiadający mu ton.

Pierwotny projekt zakładał również podział na wiele rodzajów fal. Rodzaj fali miał być wybierany poprzez obracanie enkodera cyfrowego w lewo (poprzedni) lub prawo (następny). Diagram przepływu danych wstępnej wersji projektu, został przedstawiony na rysunku 2.



Rysunek 2: Wstępny projekt organów. Niebieskimi strzałkami oznaczono podstawową i wykonaną jako pierwszą funkcjonalność.

1.2 Założenia rozszerzone

Rozszerzeniem funkcjonalności organów sterowanych za pomocą klawiatury było uzyskanie możliwości odtwarzania wcześniej zapisanej sekwencji dźwięków o zmiennej długości. Wykorzystano do tego możliwość wczytania danych z karty pamięci i moduł *SDC_FileReader*. Przykład zapisu dźwięku w pliku tekstowym:

$$a401$$

gdzie:

$$a - Klawisz\ na\ klawiaturze\ QWERTY$$

$$4 - Oktawa$$

$$01 - Czas\ trwania\ [x * 10ms]$$

Jeden dźwięk jest zawsze definiowany z wykorzystaniem 4 znaków, więc zapis nie wymaga stosowania separatorów. Czas trwania dźwięku zapisany jest z wykorzystaniem liczby dziesiętnej zapisanej tekstowo na 2 znakach. Definiuje ona czas trwania dźwięku, jako wielokrotność $10ms$.

2 Struktura układu

2.1 Schemat najwyższego poziomu

Schemat najwyższego poziomu, przedstawiony na rysunku 3, zawiera wszystkie zewnętrzne moduły odpowiedzialne za komunikację z urządzeniami peryferyjnymi. Są to:

- *SDC_FileReader*
- *PS2_Kbd*
- *DACWrite*

Wszystkie moduły komunikują się z modułem *InnerLogic*.

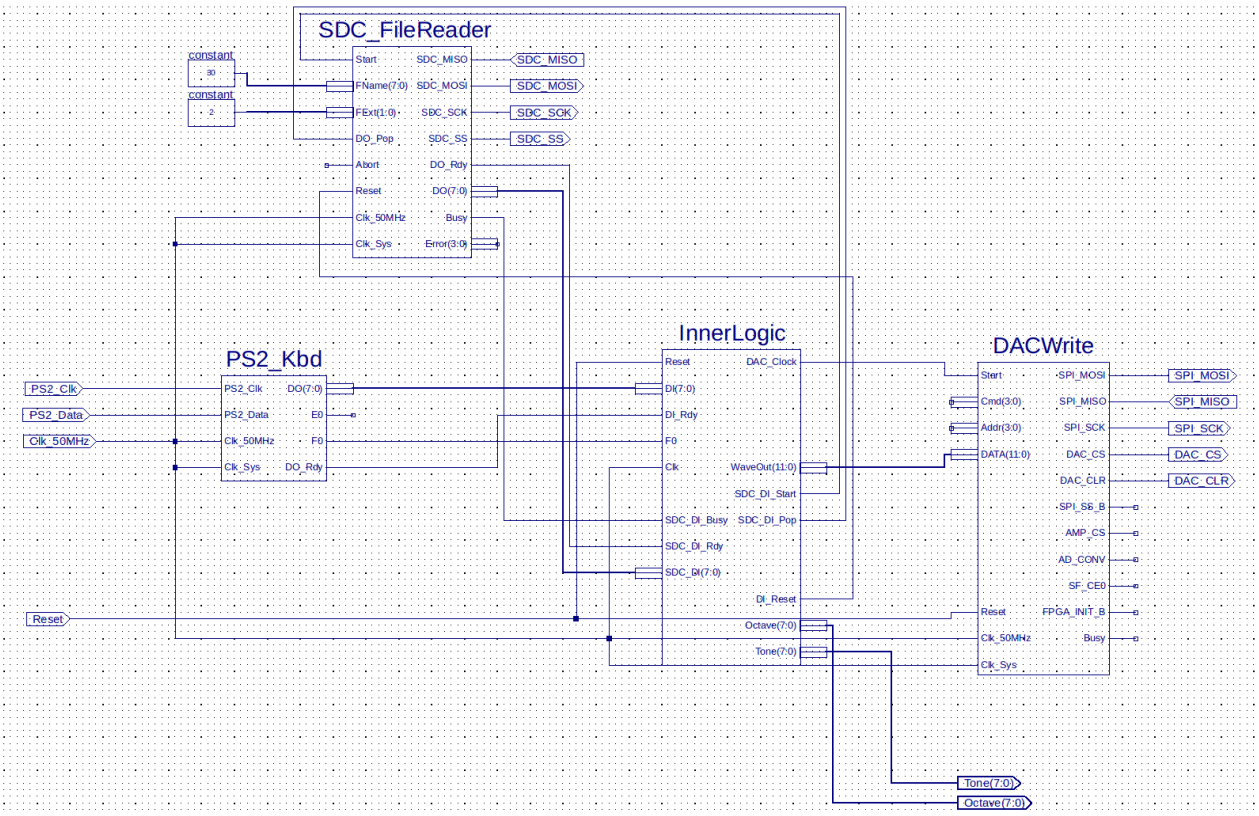
2.2 InnerLogic

Schemat InnerLogic, przedstawiony na rysunku 4, zawiera wszystkie moduły odpowiedzialne za generowanie sygnału wyjściowego, na podstawie danych wejściowych zebranych z klawiatury i karty SD. Przedstawione tutaj moduły mają swoje częściowe odwzorowanie we wstępnym projekcie, przedstawionym na rysunku 2.

W celach debugowania jednymi z wyjść modułu InnerLogic są numery aktualnie odtwarzanego tonu oraz oktawy. Zostały one podłączone do diod LED układu, gdzie pierwsze cztery (7-4) pokazują zakodowany binarnie numer tonu, a ostatnie trzy (2-0) kod oktawy.

2.3 FileReaderFSM

Moduł FileReaderFSM realizuje maszynę stanów, która jest odpowiedzialna za interakcjem z modułem *SDC_FileReader*. Odpowiada on za dostarczanie numeru tonu i oktawy przez określony czas, gdzie wszystkie te dane odczytywane są z karty SD.



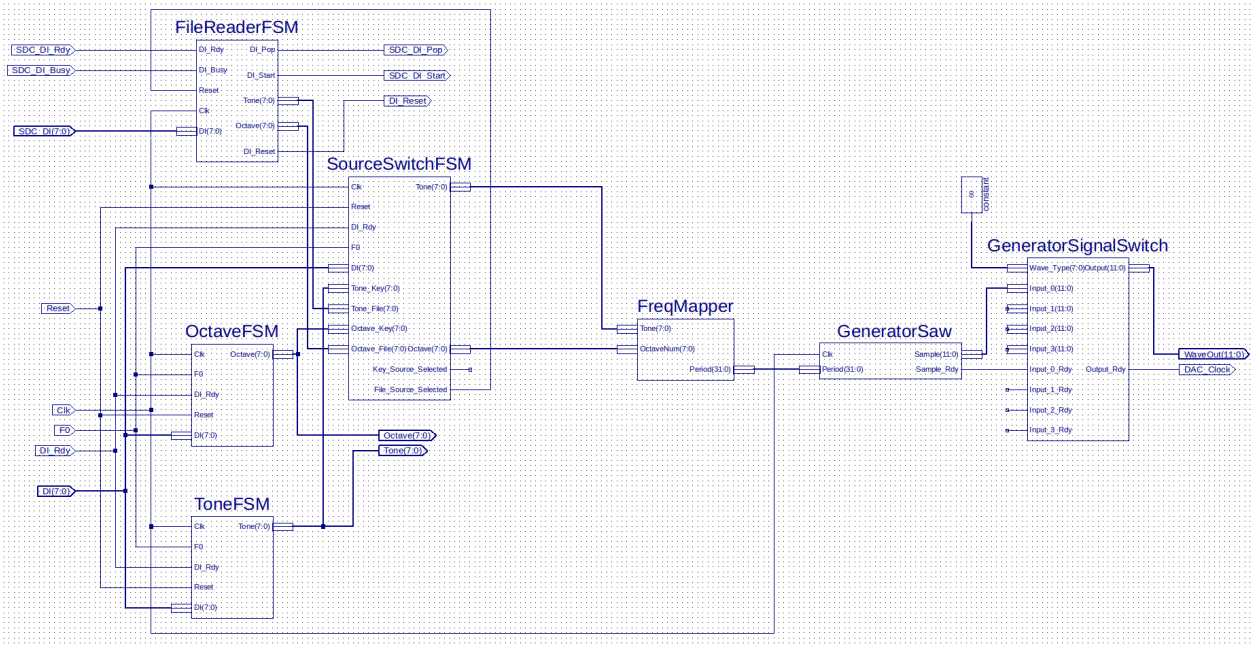
Rysunek 3: Schemat najwyższego poziomu.

Moduł *SDC_FileReader* umożliwia odczyt wartości z pliku podobnie do kolejki FIFO. Moduł *FileReaderFSM* w procesie odczytu danych jednego dźwięku najpierw odczytuje znak tonu, potem oktawy, a następnie czas jego trwania, wpisując tę wartość do licznika. Po zakończonym odczycie 4 znaków, licznik jest uruchamiany, a wartości zmapowanych kodów tonu i oktawy są obecne na wyjściach modułu, dopóki licznik się nie wyzeruje. Wczytanie tonu 1 i oktawy 4 przedstawia symulacja na rysunku 5.

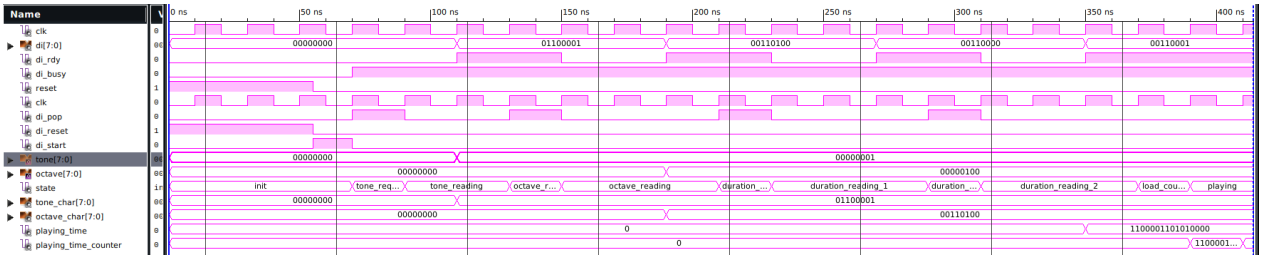
Ton i oktawa podawane są na wyjście zaraz po ich odczytaniu, a licznik uruchamiany jest po wczytaniu całego słowa określającego długość dźwięku. Skutkuje to pomijalnie małym wydłużeniem czasu trwania dźwięku.

2.3.1 Symulacja

Na rysunku 5 w chwili $t = 55ns$ widzimy zwolnienie sygnału reset i podanie sygnału start do modułu *SDC_FileReader*. Następnie moduł przechodzi przez stany na zmianę <*sub*-



Rysunek 4: Schemat InnerLogic.



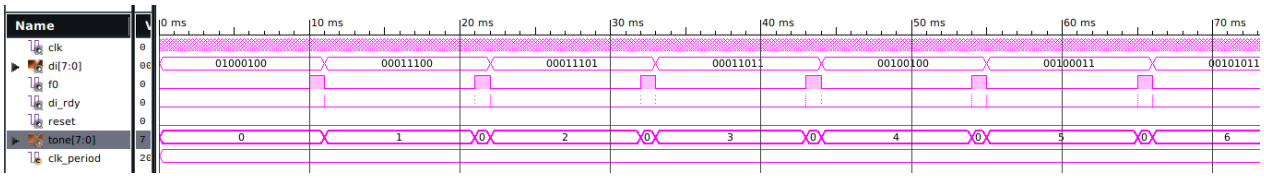
Rysunek 5: Symulacja modułu FileReaderFSM.

ject>_*Request* i <*subject*>_*Reading*, w których kolejno popycha kolejkę odczytu z karty pamięci i zapisuje dane dostarczone przez sygnał *DI*. Na końcu, w chwili $t = 380ns$ moduł przechodzi do procesu dekrementowania licznika, który odlicza czas trwania dźwięku.

2.4 ToneFSM

ToneFSM realizuje maszynę stanów, której stan określa aktualnie odtwarzany ton. Kody od 1 do 12 odpowiadają wszystkim tonom jednej oktawy. Kod 0 odpowiada ciszy, czyli stanowi, kiedy żaden przycisk nie jest wciśnięty. Stan maszyny zmieniany jest w momencie naciśnięcia lub puszczenia przycisku na klawiaturze i stan ten jest następnie mapowany na odpowiedni kod tonu. Kolejne wciśnięcia przycisków (A, W, S, E, D, R, F) przedstawia symulacja na

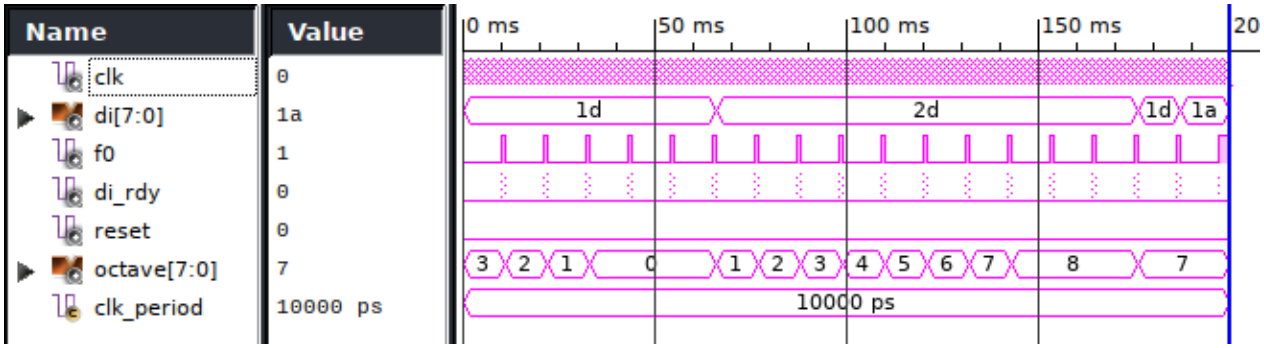
rysunku 6. Kod lewej strzałki to *X"1d"*, natomiast prawej *X"2d"*. Warto również zauważyć, że po wciśnięciu innego klawisza nie jest zmieniany stan.



Rysunek 6: Symulacja modułu ToneFSM.

2.5 OctaveFSM

OctaveFSM realizuje maszynę stanów, której stan określa aktualną oktawę, względem której określane są okresy fali, dla aktualnego tonu (przez ton rozumiane są tutaj wartości całkowite, z przedziału 0-12). Stan maszyny zmieniany jest w momencie naciśnięcia strzałki lewej ($\leftarrow$) lub prawej ($\rightarrow$). Strzałka lewa zmniejsza numer oktawy o jeden, a strzałka prawa podwyższa o jeden. Zakres wartości wyjścia *OctaveNum* ograniczony jest do przedziału 0-8, co odpowiada oktawom 1-9 lub inaczej, dźwiękom C1-H9. Reakcję modułu na wciśnięcia przycisków przedstawia rysunek 7

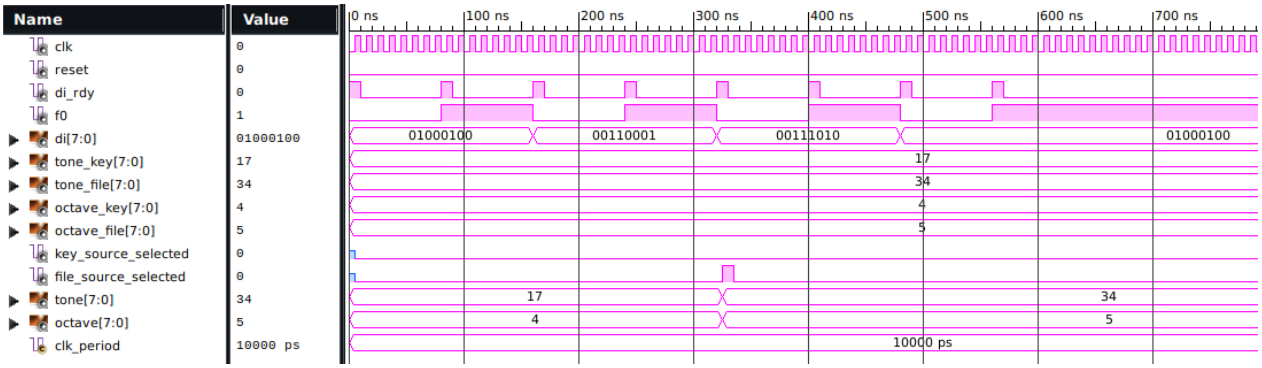


Rysunek 7: Symulacja modułu OctaveFSM.

2.6 SourceSwitchFSM

Moduł SourceSwitchFSM odpowiedzialny jest za wybór źródła dźwięku i za restartowanie odczytu dźwięków z karty pamięci w przypadku wyboru tego źródła. Klawisz M zmienia źródło na klawiaturę, a klawisz N na kartę pamięci. Na symulacji z rysunku 8 w chwili $t = 320ns$

widać zmianę domyślnego źródła klawiatury na kartę pamięci i wysłanie impulsu wystąpienia zdarzenia zmiany źródła. Zdarzenie to obsługiwane jest przez moduł *FileReaderFSM*, który restartuje proces odczytu.

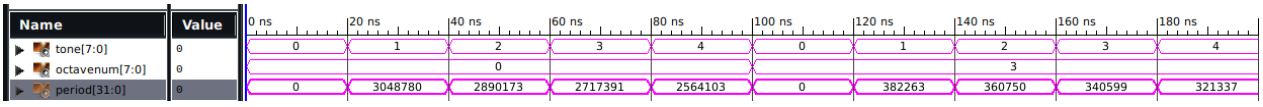


Rysunek 8: Symulacja modułu SourceSwitchFSM.

Działanie omawianego modułu zrealizowane jest za pomocą trzech procesów. Proces SYNC_PROC odpowiada za przejścia pomiędzy stanami maszyny stanów oraz za wysyłanie jednotaktowych impulsów informujących o zmiania stanu. Proces NEXT_STATE_DECODE odpowiada za dekodowanie następnego stanu na podstawie wciśniętego klawisza, a proces OUTPUT_ENCODE steruje sygnałami wyjściowymi przekierowyjąc odpowiednie źródło w zależności od stanu. Opis architektury przedstawia listing 1.

2.7 FreqMapper

FreqMapper obsługuje proces mapowania oktawy i tonu na liczbę cykli zegara o częstotliwości 50MHz, która odpowiada okresowi fali danego dźwięku. Ton 0 mapowany jest zawsze na wartość 0, co w dalszym procesie generowania sygnału oznacza ciszę. Symulację dla oktawy 0 oraz 3 przedstawia rysunek 9. Sam proces mapowania opisuje kod z listingu 2 i sprowadza się on do odczytania wartości z dwuwymiarowej tablicy `Cycles_Per_Wave_Period_Table`.



Rysunek 9: Symulacja modułu FreqMapper.

Zależności pomiędzy liczbą okresów P_{C3}, a okresem oraz częstotliwością tonu C3, opisują poniższe równania.

$$f_{clk} = 50MHz$$

$$T_{clk} = \frac{1}{50MHz} = 20ns$$

$$P_{C3} = 382263$$

$$T_{C3} = P_{C3} * T_{clk} = 7645260ns$$

$$f_{C3} = \frac{1}{T_{C3}} \approx 130.81Hz$$

2.8 GeneratorSaw

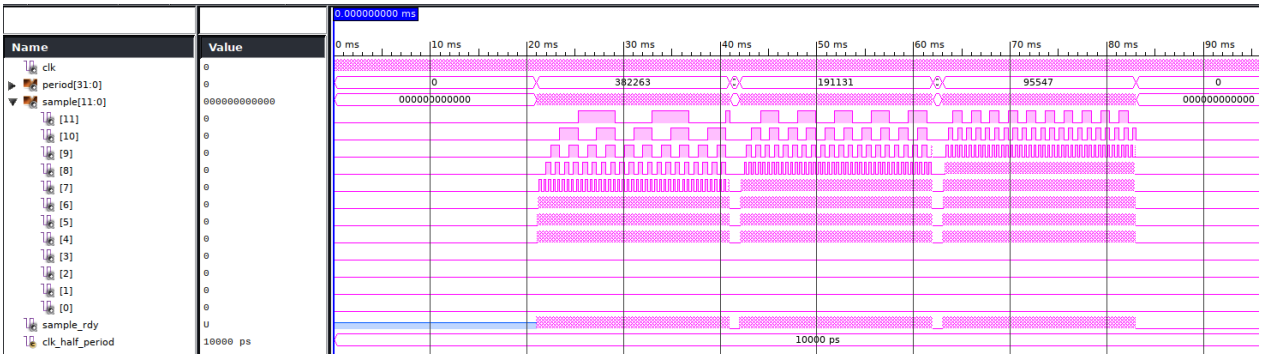
Moduł *GeneratorSaw* generuje falę piłokształtną dla zadanego okresu dostarczonego z modułu *FreqMapper*. Skuteczna rozdzielczość generatora wynosi 8 bitów, jednak próbki na wyjściu są przesuwane o 4 bity w lewo, aby osiągnąć zgodność z rozmiarem próbki na wejściu modułu *DACWrite*.

Kolejne próbki fali generowane są w oparciu o dwa liczniki. Jeden niskiej częstotliwości (*low-freq*), który odpowiada za bity 0-3, oraz drugi, o okresie 16-krotnie krótszym (*high-freq*), który odpowiada za bity 4-7 próbki wynikowej. Bity 8-11 pozostają zawsze wyzerowane. Zastosowanie dwóch liczników ma na celu osiągnięcie wyższej rozdzielczości generatora, przy zachowaniu niewielkiego błędu okresu fali, który wynika z niedokładności dzielenia. Licznik *high-freq* podlega pod licznik *low-freq*, tzn. wyzerowanie licznika *low-freq* skutkuje wyzerowaniem licznika *high-freq* oraz bitów 4-7. Dzięki temu błąd czasu trwania okresu fali zostaje zredukowany.

Symulację dla modułu GeneratorSaw przedstawia rysunek 10. Na wejście *Period* podano następujące wartości:

$$0(cisza) \rightarrow 382263(C3) \rightarrow 191131(C4) \rightarrow 95547(C5) \rightarrow 0(cisza). \quad (1)$$

Dla każdej z powyższych wartości była generowana fala w symulacji przez 20ms, z odstępami ciszy trwającymi 1ms.



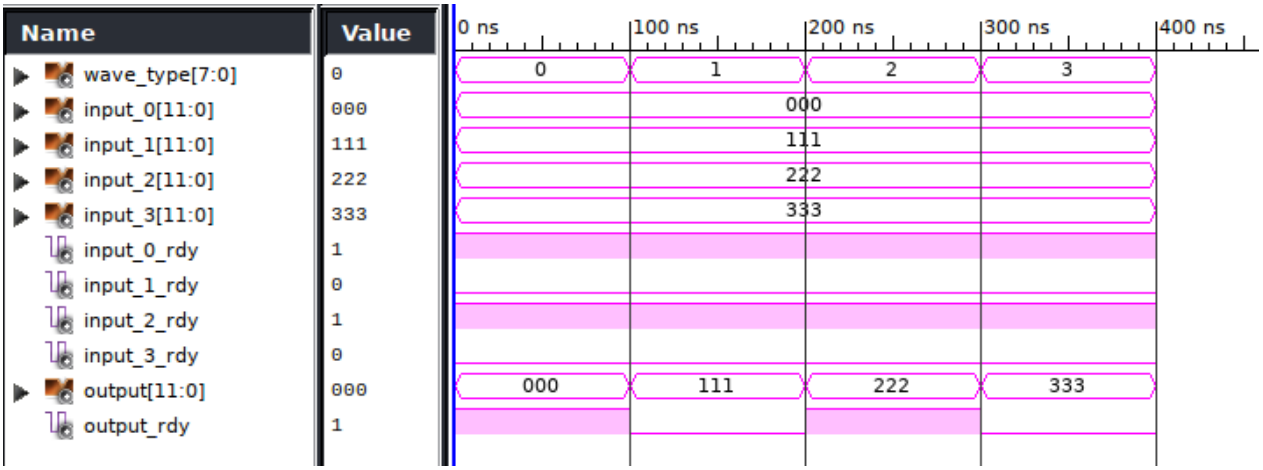
Rysunek 10: Symulacja modułu GeneratorSaw.

2.9 GeneratorSignalSwitch

Moduł GeneratorSignalSwitch jest multiplekserem sygnałów przychodzących z różnych generatorów fali. Rodzaj fali przekazywanej na wyjście określany jest sygnałem *wave_type*. Moduł ten przekazuje zarówno próbkę na wyjściu generatora, jak i impuls *sample_rdy*, generowany przy każdej zmianie wartości próbki. Wybór rodzaju fali X, gdzie X jest liczbą z przedziału 0-3, powoduje następujące zależności:

$$output <= input\_X$$

$$output\_rdy <= input\_X\_rdy$$



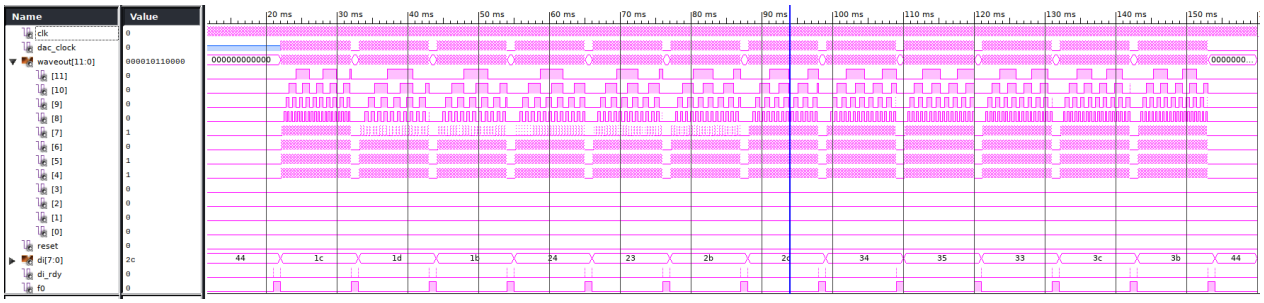
Rysunek 11: Symulacja modułu GeneratorSignalSwitch.

3 Symulacja InnerLogic

Oba warianty wejść zostały odpowiednio przetestowane w symulacji. Rysuneki 12 i 13 przedstawiają symulacje działania modułu InnerLogic i falę generowaną przez ten moduł.

3.1 Wejście z klawiatury

Rysunek 12 przedstawia symulację działania modułu InnerLogic i falę generowaną przez ten moduł. Symulacja obejmuje odtworzenie wszystkich tonów w jednej oktawie, poprzez wciśnięcie odpowiadających im klawiszy. Przeplatane jest to chwilą ciszy, co widać na symulacji.

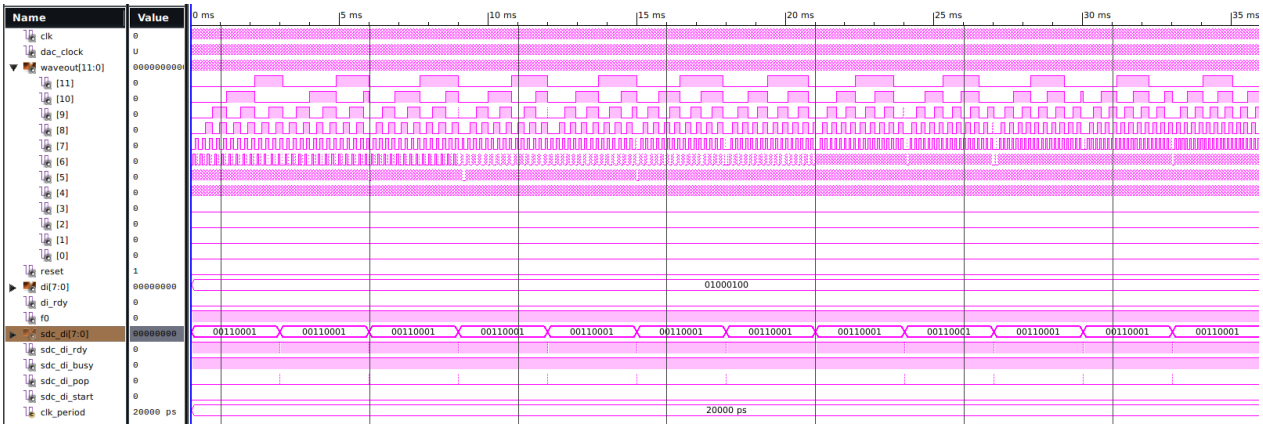


Rysunek 12: Symulacja modułu InnerLogic. Wejście z klawiatury.

3.2 Wejście z karty pamięci

Symulacja obejmuje odtworzenie melodii zdefiniowanej w pliku na karcie pamięci. Dźwięki opisane są następującym ciągiem:

```
a403w403s403e403d403f403t403g403y403h403u403j403
```



Rysunek 13: Symulacja modułu InnerLogic. Wejście z karty pamięci.

4 Implementacja

4.1 Analiza czasów

Narzędzie ISE w wygenerowanym raporcie z procesu implementacji zapewnił, że wymagania czasowe związane z częstotliwością taktowania zegara 50MHz zostaną spełnione.

```
Timing summary:
---------------
Timing errors: 0  Score: 0  (Setup/Max: 0, Hold: 0)
Constraints cover 10400333 paths, 0 nets, and 8116 connections
Design statistics:
    Minimum period:  18.392ns{1}   (Maximum frequency:  54.371MHz)
```

4.2 Analiza zajętości zasobów

Module Name	Partition	Slices	Slice Reg	LUTs	LUTRAM	BRAM	MAP_MULT18X18	BUFG	DCM
main		0/1268	0/734	0/2113	0/0	0/1	0/0	1/1	0/0
InnerLogic_1		0/616	0/277	0/1086	0/0	0/0	0/0	0/0	0/0
FileReaderFSM_1		206/...	124/124	345/345	0/0	0/0	0/0	0/0	0/0
FreqMapper_1		194/...	0/0	368/368	0/0	0/0	0/0	0/0	0/0
GeneratorSaw_1		150/...	107/107	255/255	0/0	0/0	0/0	0/0	0/0
OctaveFSM_1		29/29	32/32	54/54	0/0	0/0	0/0	0/0	0/0
SourceSwitchFSM_1		23/23	2/2	38/38	0/0	0/0	0/0	0/0	0/0
ToneFSM_1		14/14	12/12	26/26	0/0	0/0	0/0	0/0	0/0
XLXI_27		1/592	0/383	1/957	0/0	0/1	0/0	0/0	0/0
XLXI_86		0/119	0/108	0/151	0/0	0/0	0/0	0/0	0/0
XLXI_1		66/66	72/72	71/71	0/0	0/0	0/0	0/0	0/0
XLXI_2		53/53	36/36	80/80	0/0	0/0	0/0	0/0	0/0
XLXI_89		28/28	36/36	52/52	0/0	1/1	0/0	0/0	0/0
XLXI_90		430/...	225/239	749/753	0/0	0/0	0/0	0/0	0/0
Res_Busy		5/5	4/4	1/1	0/0	0/0	0/0	0/0	0/0
Res_DO_CE		4/4	5/5	1/1	0/0	0/0	0/0	0/0	0/0
Res_Go		5/5	5/5	2/2	0/0	0/0	0/0	0/0	0/0
XLXI_33		23/27	34/39	34/35	0/0	0/0	0/0	0/0	0/0
ResStart		4/4	5/5	1/1	0/0	0/0	0/0	0/0	0/0
XLXI_34		29/33	32/35	34/35	0/0	0/0	0/0	0/0	0/0
ResDORdy		4/4	3/3	1/1	0/0	0/0	0/0	0/0	0/0

Rysunek 14: Raport zajętości zasobów.

5 Podręcznik użytkowania

6 Podsumowanie

6.1 Zrealizowane założenia

Działanie poparte poprawnymi efektami symulacji pozwala sądzić, iż projekt został wykonany poprawnie, zgodnie ze wstępnymi założeniami. Zrezygnowano jednak z pozostałych generatorów typów fal, pozostając tylko przy fali piłokształtnej. Proces generowania fali w pozostałych generatorach odbywałby się podobnie, poprzez użycie innego zachowania liczników lub podawanie na wyjście stablicowanych wartości.

6.2 Możliwości dalszego rozwoju

Kolejnym etapem rozwoju projektu może być dodanie innych typów generatorów fal, oraz możliwości przełączania się pomiędzy nimi z użyciem sygnałów wejściowych, czy to z klawiatury, czy na przykład z enkodera. W następnej kolejności zrealizować można obsługę więcej niż jednego przycisku w tym samym czasie.

7 Listingi

```
ARCHITECTURE Behavioral OF SourceSwitchFSM IS
    TYPE state_type IS (Key_Source, File_Source);
    SIGNAL state, next_state : state_type;
    SIGNAL Tone_DUMMY : STD_LOGIC_VECTOR(7 DOWNTO 0) := X"00";
    SIGNAL Octave_DUMMY : STD_LOGIC_VECTOR(7 DOWNTO 0) := X"00";
BEGIN

    SYNC_PROC : PROCESS (clk)
    BEGIN
        IF rising_edge(Clk) THEN
            IF (Reset = '1') THEN
                state <= Key_Source;
            ELSIF DI_Rdy = '1' THEN
                state <= next_state;
            END IF;
            IF state = Key_Source AND next_state = File_Source THEN
                File_Source_Selected <= '1';
            ELSIF state = File_Source AND next_state = Key_Source THEN
                Key_Source_Selected <= '1';
            ELSE
                File_Source_Selected <= '0';
                Key_Source_Selected <= '0';
            END IF;
        END IF;
    END PROCESS;

    NEXT_STATE_DECODE : PROCESS (state, DI, F0, DI_Rdy)
    BEGIN
        next_state <= state;
        IF F0 = '0' THEN
            CASE DI IS
                WHEN X"31" => next_state <= Key_Source; -- N
                WHEN X"3A" => next_state <= File_Source; -- M
                WHEN OTHERS =>
```

```
            END CASE;
        END IF;
    END PROCESS;

    OUTPUT_ENCODE : PROCESS (state, DI_Rdy, Tone_Key, Octave_Key,
    ↪ Tone_File, Octave_File)
    BEGIN
        CASE state IS
            WHEN Key_Source =>
                Tone_DUMMY <= Tone_Key;
                Octave_DUMMY <= Octave_Key;
            WHEN File_Source =>
                Tone_DUMMY <= Tone_File;
                Octave_DUMMY <= Octave_File;
        END CASE;
    END PROCESS;

    Tone <= Tone_DUMMY;
    Octave <= Octave_DUMMY;
END Behavioral;
```

Listing 1: Opis architektury modułu SourceSwitchFSM

```
BEGIN
    Period <= STD_LOGIC_VECTOR(
            to_unsigned(
              INTEGER(
                Cycles_Per_Wave_Period_Table(
                  to_integer(unsigned(OctaveNum)))(to_integer(unsigned(
    ↪ Tone))
                )
              ),
              32
            )
          );
END Behavioral;
```

Listing 2: Opis architektury modułu FreqMapper