# Układy Cyfrowe i Systemy Wbudowane 2

Projekt

## Organy z pozytywką

Maja Bojarska, 241287

Damian Koper, 241292

10 maja 2020

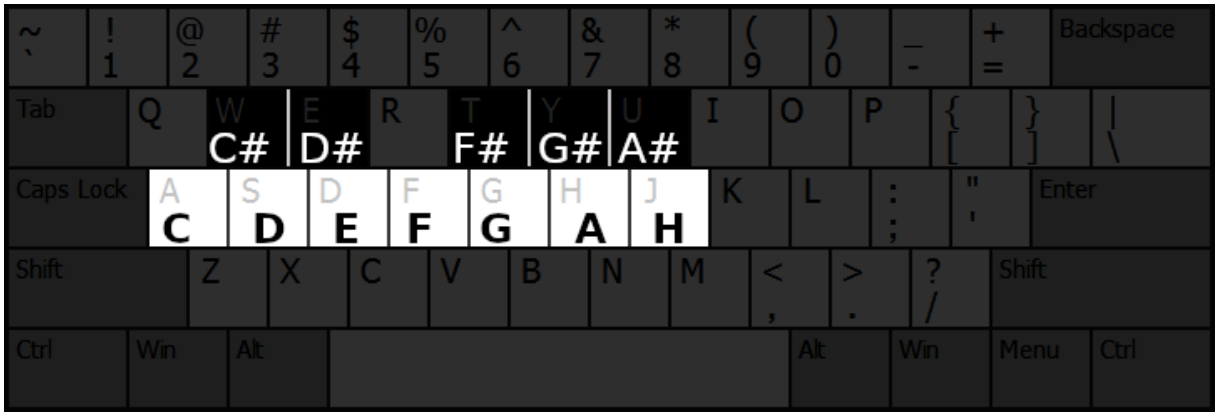
# Spis treści

# 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.



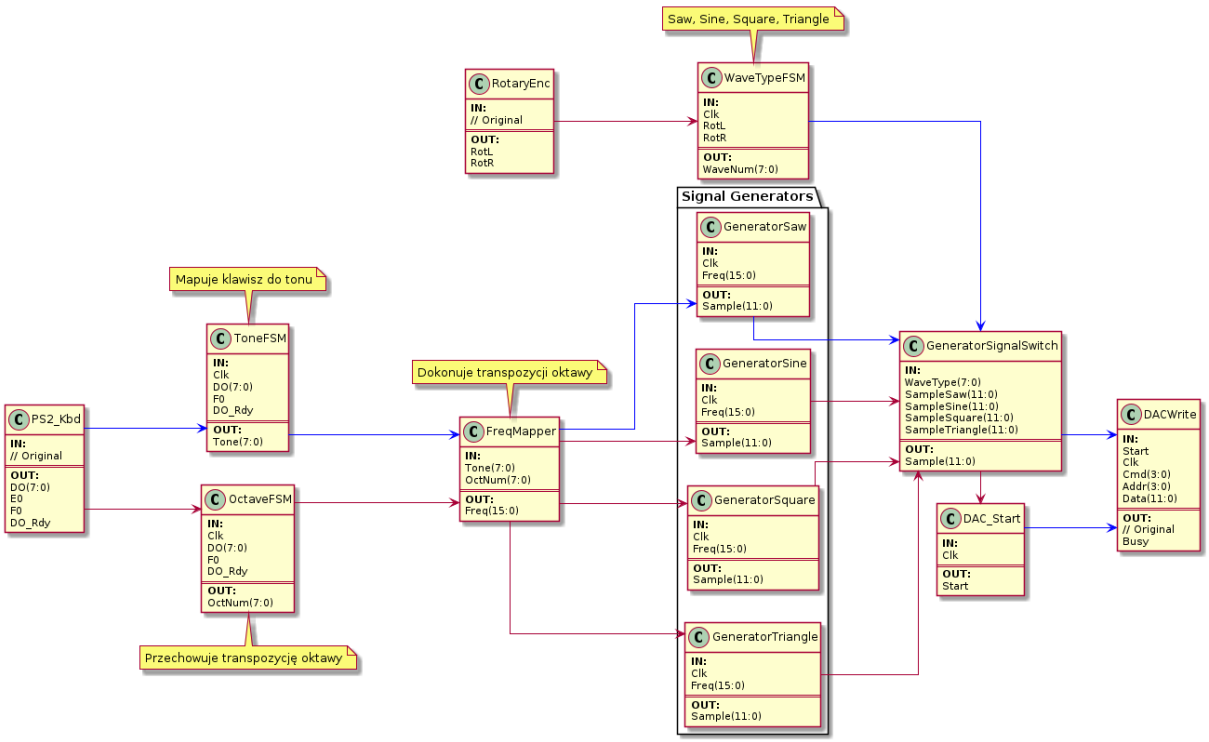

**Rysunek 1:** Przypisanie tonów oktawy do klawiszy klawiatury QWERTY. Poniżej każdej litery klawisza znajduje się odpowiadający mu ton.

Skala organów opiera się na dwunastotonowym systemie równomiernie temperowanym. Zakłada on podział oktawy na 12 równych części. Skutkuje to stosunkiem częstotliwości dwóch kolejnych dźwięków wynoszącym $\sqrt[12]{2}$. Poniżej przedstawiona została relacja pomiędzy użytymi identyfikatorami dźwięków (id), a odpowiadającą im częstotliwością fali (f).

| id | f [Hz] | id | f [Hz] | id | f [Hz] | id | f [Hz] | id | f [Hz] |
|---|---|---|---|---|---|---|---|---|---|
| C0 | 16.35 | C2 | 65.41 | C4 | 261.63 | C6 | 1046.5 | C8 | 4186.01 |
| C#0 | 17.32 | C#2 | 69.3 | C#4 | 277.18 | C#6 | 1108.73 | C#8 | 4434.92 |
| D0 | 18.35 | D2 | 73.42 | D4 | 293.66 | D6 | 1174.66 | D8 | 4698.63 |
| D#0 | 19.45 | D#2 | 77.78 | D#4 | 311.13 | D#6 | 1244.51 | D#8 | 4978.03 |
| E0 | 20.6 | E2 | 82.41 | E4 | 329.63 | E6 | 1318.51 | E8 | 5274.04 |
| F0 | 21.83 | F2 | 87.31 | F4 | 349.23 | F6 | 1396.91 | F8 | 5587.65 |
| F#0 | 23.12 | F#2 | 92.5 | F#4 | 369.99 | F#6 | 1479.98 | F#8 | 5919.91 |
| G0 | 24.5 | G2 | 98 | G4 | 392 | G6 | 1567.98 | G8 | 6271.93 |
| G#0 | 25.96 | G#2 | 103.83 | G#4 | 415.3 | G#6 | 1661.22 | G#8 | 6644.88 |
| A0 | 27.5 | A2 | 110 | A4 | 440 | A6 | 1760 | A8 | 7040 |
| A#0 | 29.14 | A#2 | 116.54 | A#4 | 466.16 | A#6 | 1864.66 | A#8 | 7458.62 |
| H0 | 30.87 | H2 | 123.47 | H4 | 493.88 | H6 | 1975.53 | H8 | 7902.13 |
| C1 | 32.7 | C3 | 130.81 | C5 | 523.25 | C7 | 2093 | | |
| C#1 | 34.65 | C#3 | 138.59 | C#5 | 554.37 | C#7 | 2217.46 | | |
| D1 | 36.71 | D3 | 146.83 | D5 | 587.33 | D7 | 2349.32 | | |
| D#1 | 38.89 | D#3 | 155.56 | D#5 | 622.25 | D#7 | 2489.02 | | |
| E1 | 41.2 | E3 | 164.81 | E5 | 659.25 | E7 | 2637.02 | | |
| F1 | 43.65 | F3 | 174.61 | F5 | 698.46 | F7 | 2793.83 | | |
| F#1 | 46.25 | F#3 | 185 | F#5 | 739.99 | F#7 | 2959.96 | | |
| G1 | 49 | G3 | 196 | G5 | 783.99 | G7 | 3135.96 | | |
| G#1 | 51.91 | G#3 | 207.65 | G#5 | 830.61 | G#7 | 3322.44 | | |
| A1 | 55 | A3 | 220 | A5 | 880 | A7 | 3520 | | |
| A#1 | 58.27 | A#3 | 233.08 | A#5 | 932.33 | A#7 | 3729.31 | | |
| H1 | 61.74 | H3 | 246.94 | H5 | 987.77 | H7 | 3951.07 | | |

**Tablica 1:** Tabela częstotliwości dźwięków w dwunastotonowym systemie równomiernie temperowanym, A4 strojone pod 440Hz[6].

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*[5]. Przykład zapisu dźwięku w pliku tekstowym:

$$a401$$

gdzie:

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

$$4 - Numer\ oktawy$$

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

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 $10ms$.

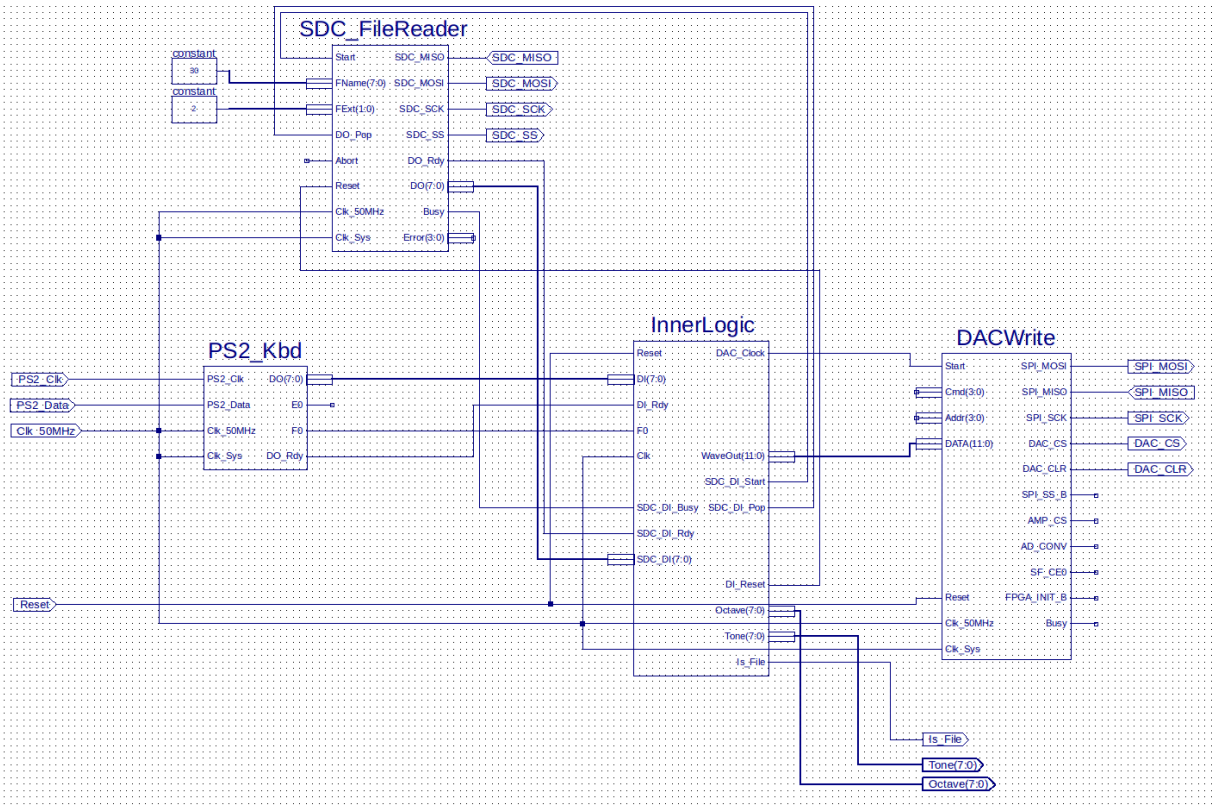
# 2 Struktura układu

## 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[5]*
- *PS2_Kbd[4]*
- *DACWrite[3]*

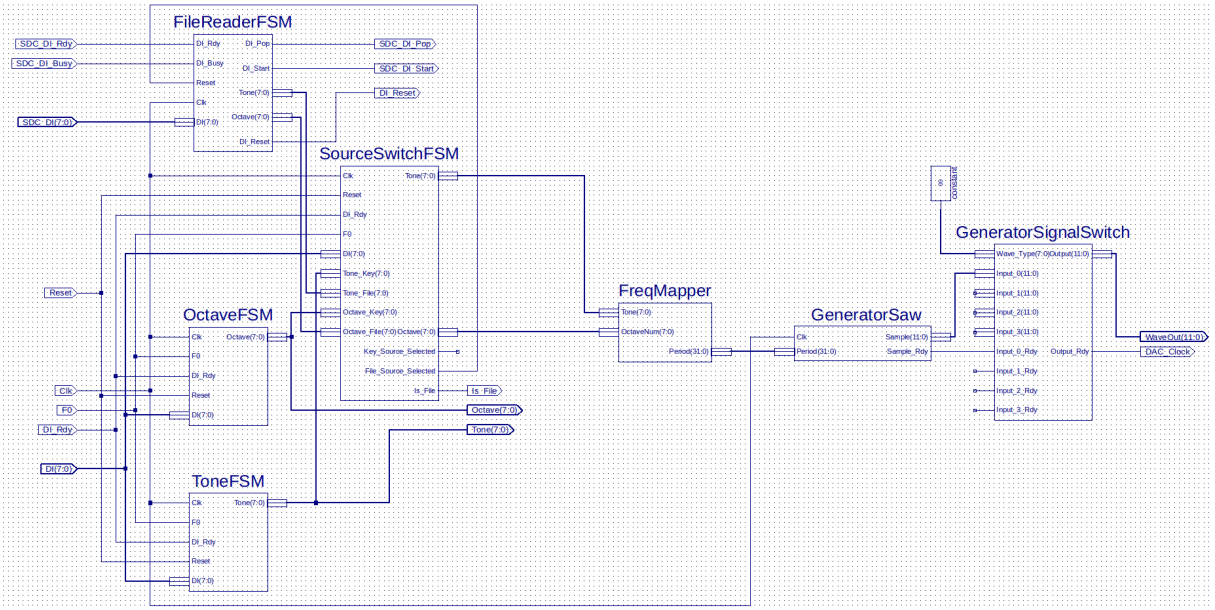
Wszystkie moduły komunikują się z modułem *InnerLogic*.




**Rysunek 3:** Schemat najwyższego poziomu.

## 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.




**Rysunek 4:** Schemat InnerLogic.

W celach debugowania jednymi z wyjść modułu InnerLogic są numery aktualnie odtwarzanej oktawy i źródła dźwięku. Zostały one podłączone do diod LED układu, gdzie pierwsza z nich (7) pokazuje źródło dźwięku (zapalona oznacza wejście z karty pamięci), a ostatnie trzy (2-0) kod oktawy.

## 2.3 FileReaderFSM

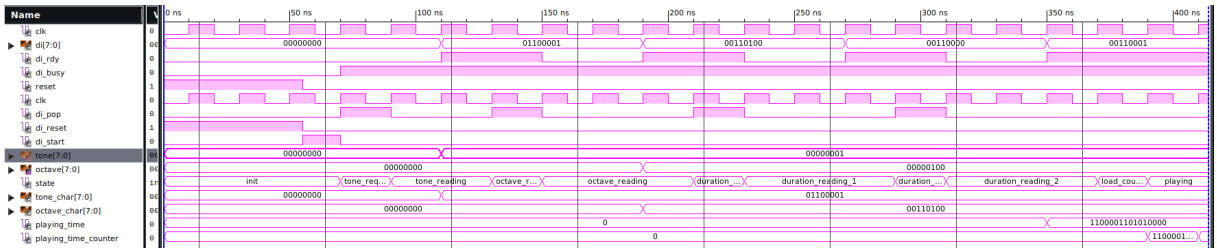
Moduł FileReaderFSM realizuje maszynę stanów, która jest odpowiedzialna za interakcje z modułem *SDC_FileReader*. Odpowiada on za dostarczanie numeru tonu i oktawy przez określony czas, gdzie wszystkie te dane odczytywane są z karty SD.

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



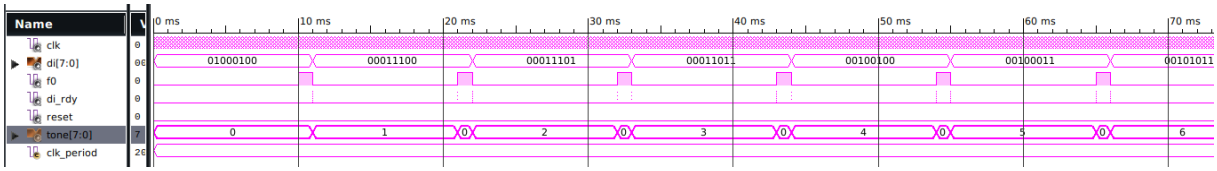

**Rysunek 5:** Symulacja modułu FileReaderFSM.

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 *<subject>_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 liczników, które odliczają czas trwania dźwięku. Zastosowano dwa liczniki. Jeden, który odlicza okresy $10ms$ i drugi, który odlicza cykle zegara, które wystąpią w tym okresie. W procesie optymalizacji maksymalnej częstotliwości zrezygonowano z pojedynczego licznika, ponieważ wymagałoby to wykonania mnożenia, które przez swoją złożoność angażowałoby użycie dwustopniowego mnożenia (rejestry AREG i BREG[1]), co drastycznie zwiększa czas Clock-to-Output.

## 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. Warto również zauważyć, że po wciśnięciu innego klawisza nie

jest zmieniany stan.

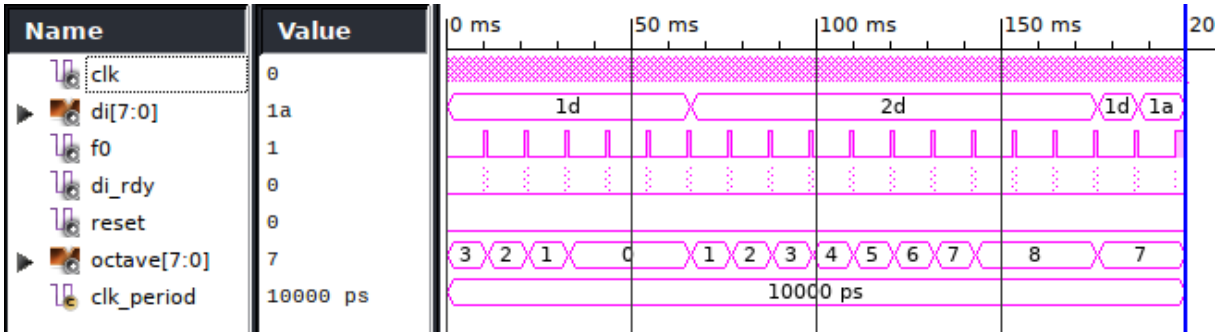



**Rysunek 6:** Symulacja modułu ToneFSM.

## 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 (←, kod *X"1d"*) lub prawej (→, kod *X"2d"*). 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 nr 0-8 lub inaczej, dźwiękom C0-H8.

Reakcję modułu na wciśnięcia przycisków przedstawia symulacja na rysunku 7. W chwili 0ms, numer oktawy jest równy wartości domyślnej modułu, czyli 3. Następnie, w odstępach 11ms wciskana jest lewa strzałka, do chwili 55ms. Widać, że numer oktawy początkowo maleje, jednak od chwili 33ms pozostaje na wartości 0. Spowodowane jest to ograniczeniem zakresu oktaw. W zakresie czasu 66-165ms, wielokrotnie wciskany jest klawisz strzałki prawej, co skutkuje zwiększaniem numeru oktawy, jednak ponownie występuje ograniczenie, tym razem górne, w chwili 154ms.



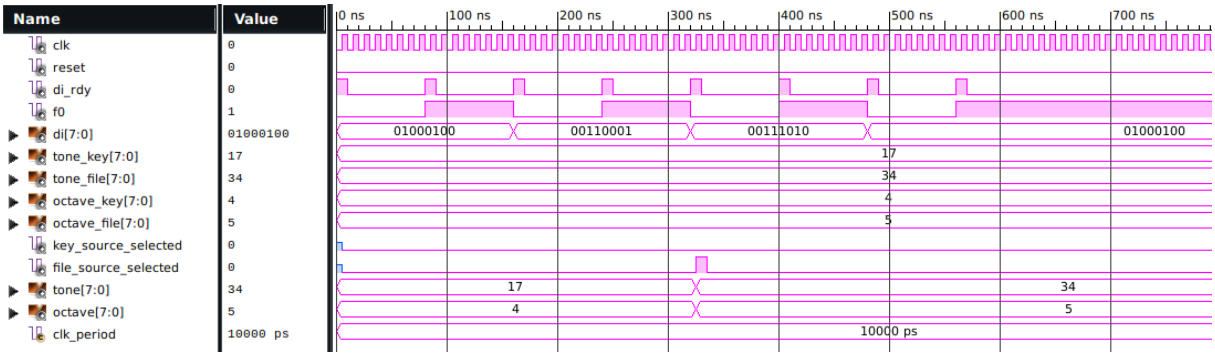

**Rysunek 7:** Symulacja modułu OctaveFSM.

Istotnym procesem tego modułu jest "NEXT_MOVE_DECODE", który interpretuje

naciśnięcia klawiszy i odpowiednio zmienia numer oktawy. Opis architektury przedstawia listing 3.

## 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.

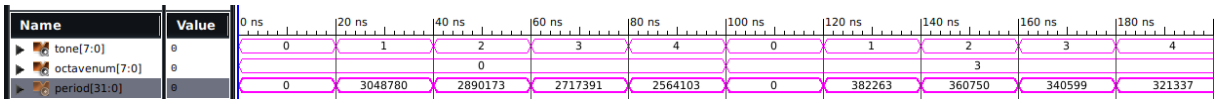
Działanie omawianego modułu zrealizowane jest za pomocą trzech procesów. Proces SYNC_PROC odpowiada za przejścia pomiędzy stanami maszyny stanów oraz za wysyłanie jednotaktowych impulsów informujących o zmiania stanu.

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.

$$f_{clk} = 50MHz$$

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

$$P_{C3} = 382263$$

$$T_{C3} = P_{C3} \cdot T_{clk} = 7645260ns$$

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

## 2.8 GeneratorSaw

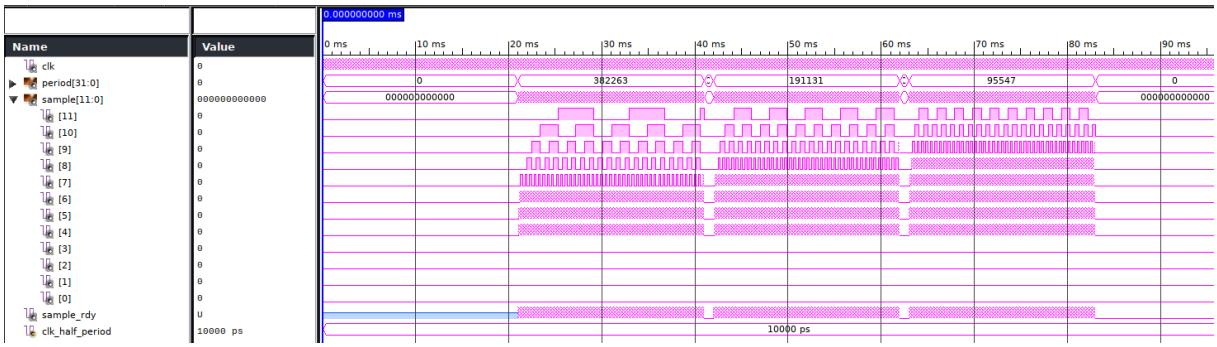
Moduł *GeneratorSaw* generuje falę piłokształtną dla zadanego okresu, dostarczonego z modułu *FreqMapper*. Skuteczna rozdzielczość generatora wynosi 8 bitów, jednak próbki na wyjściu są przesuwane o 4 bity w lewo, aby osiągnąć zgodność z rozmiarem 12-bitowej próbki na wejściu modułu *DACWrite*.

Kolejne próbki fali generowane są na podstawie zadanego okresu, 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. Podanie wartości 0 na wejście *Period* skutkuje ciszą. Próbka wyjściowa ma wtedy wartość 0, dopóki okres fali nie zostanie zmieniony na wartość niezerową.

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. Warto zauważyć, że impuls Sample_Rdy, jest generowany wraz z wyznaczaniem kolejnych nowych próbek. Dzięki temu każda z nich wczytywana jest przez moduł DACWrite[3].




**Rysunek 10:** Symulacja modułu GeneratorSaw.

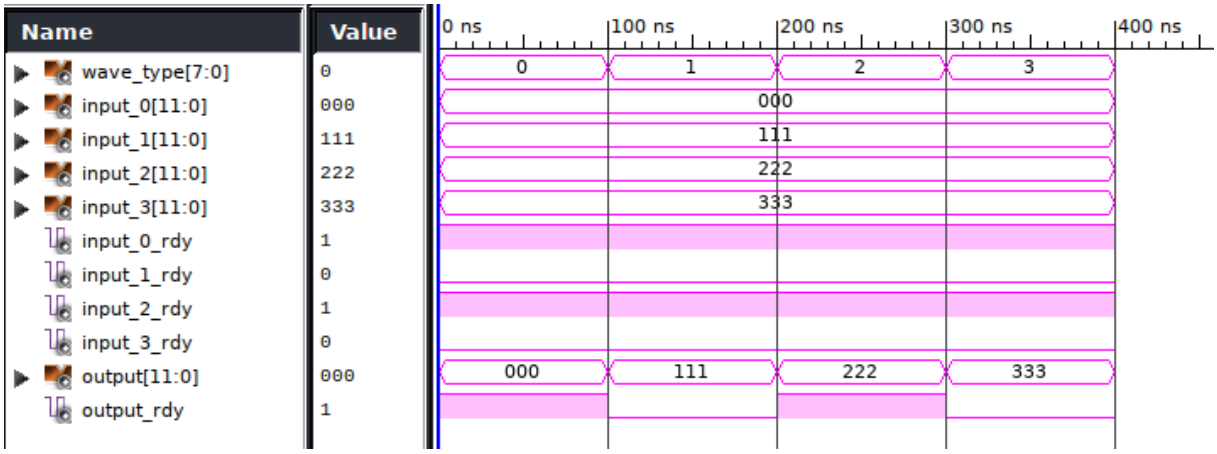
Moduł GeneratorSaw zawiera jeden proces, wyzwalany dla wschodzącego zbocza zegara 50MHz. Wartość licznika *low-freq* przechowywana jest w sygnale *Counter_A*, natomiast *high-freq* w sygnale *Counter_B*. Zmiana stanu *Counter_A*, powoduje wyzerowanie licznika *Counter_B*. Podanie nowej wartości okresu fali na wejscie *Period* powoduje wyzerowanie obu liczników, wyzerowanie próbki oraz rozpoczecie przebiegu fali od początku, z odpowiednio zmienionymi liczbami cykli dla każdego licznika. Opis architektury przedstawia listing 4.

## 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.

Moduł GeneratorSignalSwitch nie zawiera procesów. W opisie behawioralnym, za pomocą klauzuli "with ... select", wybierane są wartości wyjściowe *Output* oraz *Output_Rdy*, na podstawie wartości na wejściu *Wave_Type*.

# 3 Symulacja InnerLogic

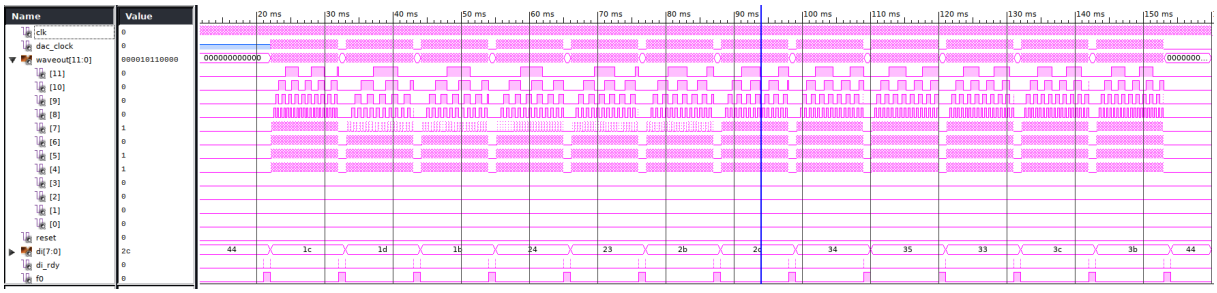
Oba warianty wejść zostały odpowiednio przetestowane w symulacji. Rysunki 12 i 13 przedstawiają symulacje działania modułu InnerLogic i falę generowaną przez ten moduł.

## 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.

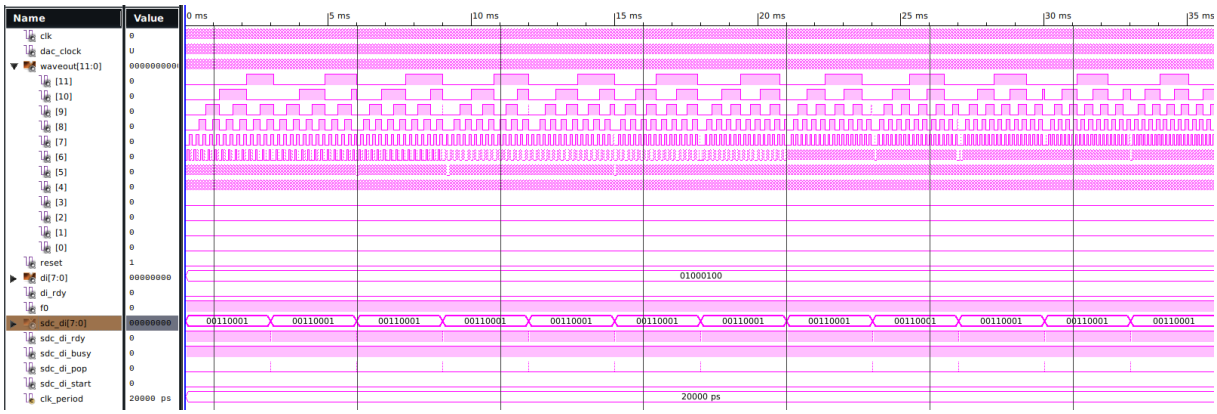
## 3.2 Wejście z karty pamięci

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




**Rysunek 12:** Symulacja modułu InnerLogic. Wejście z klawiatury.

```
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

Zrzut ekranu pokazany na rysunku 14 dostarcza informacji o użyciu zasobów sprzętowych przez projekt co jego implementacji przez narzędzie ISE. Widzimy, że łącznie użyto 1268 plastrów, 734 rejestry i 2113 generatory look-up-table. Mniej niż połowę zasobów zużywa sama logika projektu bez obsługi urządzeń WE/WY. Najwięcej generatorów look-up-table używa moduł *FreqMapper*. Narzędzie ISE zaimplementowało mapowanie kodów tonu i oktawy na okres fali jako funkcję kombinatoryczną.

| 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.

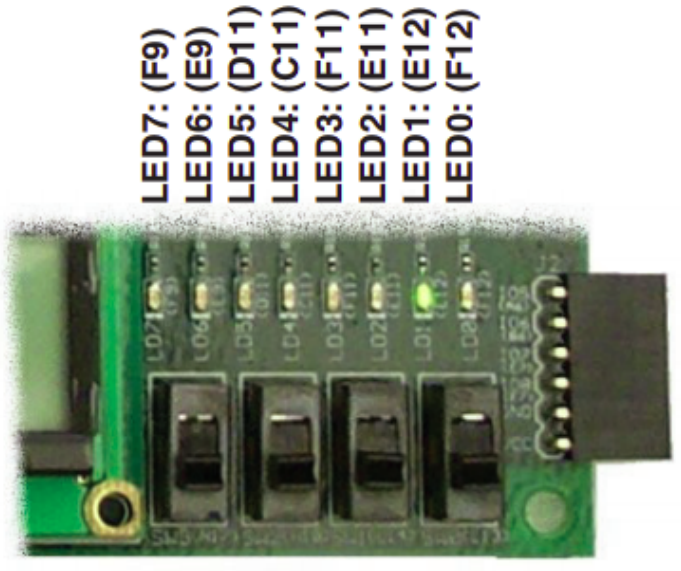
# 5 Podręcznik użytkowania

Urządzenie jest prostym instrumentem muzycznym, zbliżonym w działaniu do elektronicznych organów z klawiaturą fortepianową.

Oferowane są dwa tryby działania:

- tryb organów, w którym użytkownik odgrywa dźwięki z wykorzystaniem klawiatury komputerowej,
- tryb pozytywki, w którym urządzenie odgrywa melodię umieszczoną na karcie pamięci typu SD.

Szczegółowy opis działania powyższych trybów został omówiony w kolejnych podrozdziałach podręcznika.




**Rysunek 15:** Diody na płytce UG230[2].

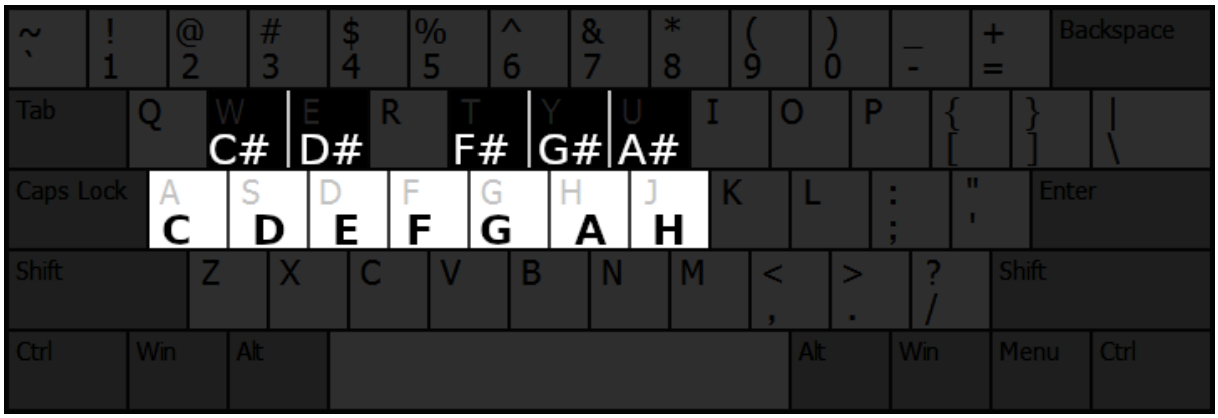
Stan urządzenia sygnalizowany jest za pomocą diod. Numer oktawy wyświetlany jest na diodach LED2-LED0, zakodowany w systemie dwójkowym, przy czym LED0 jest najmłodszym bitem. Dioda LED7 sygnalizuje tryb, zapalona oznacza tryb pozytywki, natomiast zgaszona, tryb organów.

Tryb urządzenia może zostać zmieniony wciskając odpowiedni klawisz klawiatury:

- klawisz 'M' - tryb organów,
- klawisz 'N' - tryb pozytywki.

## 5.1 Tryb organów

Klawiatura umożliwia granie dźwięków w zakresie jednej oktawy. Przypisanie klawiszy do dźwięków jest zbliżone do ułożenia klawiszy jednej oktawy fortepianu, rozpoczynającej się od dźwięku C.




**Rysunek 16:** Przypisanie dźwięków oktawy do klawiszy klawiatury QWERTY. Poniżej każdej litery klawisza znajduje się odpowiadający mu dźwięk.

Numer obecnej oktawy można zmienić strzałkami na klawiaturze. Strzałka lewa (←) zmniejsza numer oktawy o jeden, natomiast strzałka prawa (→), podwyższa go o jeden. Zakres dostępnych oktaw to 0-8, co w kontekście dźwięków, oznacza zakres od C0 (16.35Hz) do H8 (7902.13Hz).

## 5.2 Tryb pozytywki

W trybie pozytywki, urządzenie odtwarza melodię zapisaną na karcie pamięci. Plik zawerający melodię powinien mieć nazwę "0.txt", a jego zawartość musi być zgodna z przewidzianym formatem, który określa sposób zapisu melodii.

Pojedynczy dźwięk melodii zapisany jest na 4 znakach ASCII i składa się z następujących informacji:

$$a401$$

gdzie:

$$a - Klawisz\ na\ klawiaturze\ QWERTY$$
$$4 - Numer\ oktawy$$
$$01 - Czas\ trwania\ [x * 10ms]$$

Pauza (cisza) realizowana jest poprzez określenie litery dźwięku na klawiaturze, która nie jest przypisana do żadnego dźwięku w trybie organów (rysunek 16). Numer oktawy w tym przypadku nie jest istotny, jednak wciąż konieczny. Przykładowo, "x050" określa pauzę o długości 500ms.

Melodia składa się z wielu dźwięków, opisanych kolejno w powyższy sposób, bez separatorów. Liczba dźwięków w melodii ograniczona jest maksymalnym rozmiarem pliku, dopuszczalnym na karcie pamięci. Przykładowa melodia, składająca się z 8 dźwięków, w tym jednosekundowej pauzy:

$$a475d450g425x050x050g475d450a425$$

# 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 polifonii, czyli odgrywania wielu dźwięków jednocześnie.

# 7 Listingi

```
1
2  ARCHITECTURE Behavioral OF SourceSwitchFSM IS
3      TYPE state_type IS (Key_Source, File_Source);
4      SIGNAL state, next_state : state_type;
5      SIGNAL Tone_DUMMY : STD_LOGIC_VECTOR(7 DOWNTO 0) := X"00";
6      SIGNAL Octave_DUMMY : STD_LOGIC_VECTOR(7 DOWNTO 0) := X"00";
7  BEGIN
8
9      SYNC_PROC : PROCESS (clk)
10     BEGIN
11         IF rising_edge(Clk) THEN
12             IF (Reset = '1') THEN
13                 state <= Key_Source;
14             ELSIF DI_Rdy = '1' THEN
15                 state <= next_state;
16             END IF;
17             IF state = Key_Source AND next_state = File_Source THEN
18                 File_Source_Selected <= '1';
19             ELSIF state = File_Source AND next_state = Key_Source THEN
20                 Key_Source_Selected <= '1';
21             ELSE
22                 File_Source_Selected <= '0';
23                 Key_Source_Selected <= '0';
24             END IF;
25         END IF;
26     END PROCESS;
27
28     NEXT_STATE_DECODE : PROCESS (state, DI, F0, DI_Rdy)
29     BEGIN
30         next_state <= state;
31         IF F0 = '0' THEN
32             CASE DI IS
33                 WHEN X"31" => next_state <= Key_Source; -- N
34                 WHEN X"3A" => next_state <= File_Source; -- M
35                 WHEN OTHERS =>
36             END CASE;
37         END IF;
```

```
38      END PROCESS;
39
40      OUTPUT_ENCODE : PROCESS (state, DI_Rdy, Tone_Key, Octave_Key,
    ↪ Tone_File, Octave_File)
41      BEGIN
42          CASE state IS
43              WHEN Key_Source =>
44                  Tone_DUMMY <= Tone_Key;
45                  Octave_DUMMY <= Octave_Key;
46              WHEN File_Source =>
47                  Tone_DUMMY <= Tone_File;
48                  Octave_DUMMY <= Octave_File;
49          END CASE;
50      END PROCESS;
51
52      Tone <= Tone_DUMMY;
53      Octave <= Octave_DUMMY;
```

**Listing 1:** Opis architektury modułu SourceSwitchFSM

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

**Listing 2:** Opis architektury modułu FreqMapper

```
1 begin
2   SYNC_PROC : PROCESS (clk)
3   BEGIN
4         IF rising_edge(Clk) THEN
5             IF (Reset = '1') THEN
6                 Current_Oct_Num <= Default_Oct_Num;
7             ELSIF DI_Rdy = '1' THEN
8                 Current_Oct_Num <= Next_Oct_Num;
9             END IF;
10        END IF;
11    END PROCESS;
12
13  NEXT_MOVE_DECODE : PROCESS (DI, F0)
```

```
14      BEGIN
15      Next_Oct_Num <= Current_Oct_Num;
16          IF F0 = '0' THEN
17        -- Left arrow: 0x1D
18              IF (DI = X"1D" AND Current_Oct_Num > 0) THEN
19          Next_Oct_Num <= Current_Oct_Num - 1;
20        -- Right arrow: 0x1D
21        ELSIF (DI = X"2D" AND Current_Oct_Num < 8) THEN
22          Next_Oct_Num <= Current_Oct_Num + 1;
23        ELSE
24          Next_Oct_Num <= Current_Oct_Num;
25        END IF;
26      END IF;
27      END PROCESS;
28
29    Octave <= std_logic_vector(to_unsigned(Current_Oct_Num, 8));
30  end Behavioral;
```

**Listing 3:** Opis architektury modułu OctaveFSM

```
1   BEGIN
2     Cycles_Per_Period_Counter_A <= to_integer(unsigned(Period)) / 2 ** (
       ↪ Effective_Wave_Resolution - Counter_B_To_A_Resolution_Ratio);
3     Cycles_Per_Period_Counter_B <= to_integer(unsigned(Period)) / 2 ** (
       ↪ Effective_Wave_Resolution);
4
5     PROCESS (Clk) BEGIN
6       -- New Period value on input, clear counters and samples.
7       IF (rising_edge(Clk)) THEN
8         IF (Period /= Last_Period) THEN
9           Last_Period <= Period;
10          Next_8b_Sample <= x"00";
11          Counter_A <= 1;
12          Counter_B <= 1;
13          Next_8b_Sample_A <= x"00";
14          Next_8b_Sample_B <= x"00";
15          Sample_Rdy <= '0';
16        END IF;
17        -- Calculate next sample only if the target period is > 0 (tone
       ↪ not silent).
```

```
18        IF (Period /= x"00000000") THEN
19          Counter_A <= Counter_A + 1;
20          Counter_B <= Counter_B + 1;
21          IF (Counter_A > Cycles_Per_Period_Counter_A) THEN
22            -- Low frequency counter (A) rolled over. Takes priority over
    ↪ high freq counter (counter B).
23            Next_8b_Sample_A <= Next_8b_Sample_A + 2 **
    ↪ Counter_B_To_A_Resolution_Ratio;
24            Next_8b_Sample_B <= Next_8b_Sample_A;
25            Next_8b_Sample <= Next_8b_Sample_A;
26            Counter_A <= 1;
27            Counter_B <= 1;
28            Sample_Rdy <= '1';
29          ELSIF (Counter_B > Cycles_Per_Period_Counter_B) THEN
30            -- High frequency counter (B) rolled over
31            Next_8b_Sample_B <= Next_8b_Sample_B + 1;
32            Next_8b_Sample <= Next_8b_Sample_B;
33            Counter_B <= 1;
34            Sample_Rdy <= '1';
35          ELSE
36            Sample_Rdy <= '0';
37          END IF;
38        END IF;
39      END IF;
40    END PROCESS;
41
42    Sample <= std_logic_vector(Next_8b_Sample) & x"0";
43  END Behavioral;
```

**Listing 4:** Opis architektury modułu GeneratorSaw

# Bibliografia


[1] *Spartan-3E FPGA Family - Data Sheet*. URL: https://www.xilinx.com/support/documentation/data_sheets/ds312.pdf.

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

[3] Jarosław Sugier. *Moduły S3EStarter - DACWrite*. URL: http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/#_Toc479592717.

[4] Jarosław Sugier. *Moduły S3EStarter - PS2_Kbd*. URL: http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/#_Toc479592711.

[5] Jarosław Sugier. *Moduły S3EStarter - SDC_FileReader*. URL: http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/#_Toc479592721.

[6] Michigan Technological University. *Frequencies for equal-tempered scale, A4 = 440 Hz*. URL: https://pages.mtu.edu/~suits/notefreqs.html.