

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Elektronika

SPECJALNOŚĆ: Aparatura elektroniczna

PRACA DYPLOMOWA
INŻYNIERSKA

Syntezaator muzyczny z procesorem
sygnałowym TMS320

Music synthesizer with TMS320 signal
processor

AUTOR:
Adam Matusiak

PROWADZĄCY PRACĘ:
dr hab. inż. Józef Borkowski, prof. PWr

OCENA PRACY:

WROCŁAW 10.12.2015

Cel pracy:

Opracowanie konstrukcji i wykonanie prototypu syntezy muzycznego opartego na procesorze sygnałowym stałoprzecinkowej rodziny C55x oraz wykonania dla niego oprogramowania realizującego syntezę addytywną dźwięku.

Zakres pracy:

- opracowanie cyfrowej metody addytywnej syntezy dźwięku,
- implementacja tej metody na stałoprzecinkowym procesorze sygnałowym,
- opracowanie konstrukcji interfejsu użytkownika umożliwiającej zmianę parametrów syntezy w czasie rzeczywistym oraz zawierającego klawiaturę,
- implementacja programu obsługi interfejsu na procesorze Cortex M0+,
- integracja układu interfejsu użytkownika oraz układu procesora sygnałowego,
- wykonanie prototypu syntezy muzycznego.

Teza pracy:

Rozwój techniki mikroprocesorowej, który umożliwia produkowanie coraz tańszych i jeszcze bardziej energooszczędnych układów scalonych przy zachowaniu wysokiej wydajności obliczeniowej umożliwia konstrukcję cyfrowych instrumentów muzycznych charakteryzujących się wysoką kompaktowością, przenośnością, energooszczędnością w porównaniu do klasycznych instrumentów cyfrowych nie ustępując im w możliwościach.

Spis treści

1. Wstęp	5
2. Przegląd zagadnień związanych z syntezą dźwięku	8
2.1. Syntezator muzyczny	8
2.2. Cyfrowa synteza dźwięku	8
2.2.1. Typy syntezy dźwięku	8
2.2.2. Algorytmy syntezy addytywnej	8
2.2.4. Obwiednia dźwięku	10
3. Projekt systemu	12
3.1. Opis systemu	13
3.1.1. Interfejs klawiatury	13
3.1.2. Interfejs nastaw parametrów syntezy	14
3.1.3. Kontroler interfejsu	15
3.1.4. Synteza dźwięku	16
3.1.4.1. Algorytm syntezy addytywnej	16
3.1.4.2. Generacja obwiedni ADSR	17
3.1.4.3. Polifonia	20
3.1.5. Komunikacja szeregową	20
3.1.6. Przetwarzanie cyfrowo-analogowe	21
4. Implementacja funkcjonalności i fizyczna konstrukcja	22
4.1. Konstrukcja klawiatury	22
4.2. Konstrukcja interfejsu nastaw potencjometrów	22
4.3. Płyta ewaluacyjna STM32L0 DISCOVERY	23
4.4. Płyta ewaluacyjna TI EZDSP5535	24
4.5. Oprogramowanie procesora STM32L053R8	25
4.5.1. Obsługa klawiatury	25
4.5.2. Obsługa interfejsu nastaw parametrów syntezy	26

4.5.3. Obsługa komunikacji szeregowej	27
4.6. Oprogramowanie procesora TMS320C5535.....	28
4.6.1. Konfiguracja kodeka audio AIC3204	28
4.6.2. Generacja obwiedni ADSR	31
4.6.3. Obsługa przycisków funkcyjnych za pomocą przetwornika ADC SAR	33
4.6.4. Implementacja algorytmu syntezy addytywnej.....	34
4.6.5. Obsługa komunikacji szeregowej	36
5. Testowanie systemu.....	38
5.1. Stwierdzenie poprawności działania urządzenia	38
5.2. Procesor STM32I053R8.....	38
5.2.1. Weryfikacja działania klawiatury.....	38
5.2.2. Weryfikacja działania odczytu parametrów nastaw interfejsu użytkownika	39
5.3. Procesor sygnałowy TMS320C5535 oraz układ kodeka audio TLV320AIC3204	40
5.3.1 Oscylogramy sygnału mierzonego na wyjściu kodeka audio	41
5.3.2 Subiektywne wrażenia osoby słuchającej przez słuchawki	43
6. Podsumowanie	45
7. Bibliografia	46

1. Wstęp

Niniejsza praca ma za zadanie przedstawić proces projektowania oraz opracowania prototypu elektronicznego instrumentu jakim jest syntezytor muzyczny. Wszechstronność zagadnienia jakim jest budowa syntezytora obejmująca rozległy zakres aspektów elektroniki, wydała się autorowi świetną motywacją do poprawienia swoich umiejętności praktycznych i uzupełnienia wiedzy teoretycznej. Cyfrowe przetwarzanie sygnałów, technika cyfrowa w zakresie procesorów sygnałowych i mikrokontrolerów oraz zagadnienia związane z akustyką – to lista takich aspektów, których szersze poznanie jest wymagane przy realizacji projektu cyfrowego syntezytora muzycznego. Uwzględniając szeroki zakres z którym wiąże się konstrukcja w pełni funkcjonalnego instrumentu do syntezy muzyki, ograniczony stan wiedzy autora oraz ograniczoną ilość czasu poświęconego na inżynierską w bieżącym semestrze przeznaczonym na pracę dyplomową autor pracy ograniczył się do całkowitych podstaw przy jego opracowaniu. Projekt zawiera jednak wszystkie aspekty dotyczące opracowania kompletnego syntezytora muzycznego – obsługi klawiszy, obsługi interfejsu do wprowadzania parametrów syntezy przez użytkownika oraz realizację algorytmu syntezy na stałoprzecinkowym procesorze sygnałowym, przetworzenia sygnału cyfrowego na wartościowy sygnał akustyczny na wyjściu syntezytora. Dodatkową ambicją projektu było ograniczenie zużycia energii konstruowanego urządzenia oraz jego potencjalna kompaktowość po uwzględnienie projektu dedykowanej płytki PCB. Istotną częścią niniejszej pracy było stworzenie prototypu konstrukcji opartego na gotowych zestawach ewaluacyjnych. Pozwala to na przeprowadzenie niezbędnych testów, które weryfikują poprawność opracowanego projektu oraz jego szybkość modyfikację.

Pomimo podstawowego charakteru tej pracy, podczas opracowywania założeń projektowych, uwzględniony został aspekt potencjalnego rozwoju projektu urządzenia ze względu na charakter obecnego stanu rynku mniej skomplikowanych syntezytorów muzycznych. Dlatego też rozpatrzono to czy istnieje potrzeba uzupełnienia oferty instrumentów o konstrukcję o przyjętych przez tą pracę cechach. Na rynku syntezytorów muzycznym dostępne są drogie analogowe syntezytory, które zapewniają szeroki wachlarz możliwości zmiany parametrów syntezy. Innym popularnym typem syntezytorów są instrumenty wykorzystujące technikę cyfrową do realizacji algorytmu syntezy dźwięku. Produkty wywodzące się z obu grup cechują się również dużymi rozmiarami – wbudowane są w nie szerokie klawiatury bez możliwości zamiany na klawiatury o mniejszej liczbie oktaf. Z tego powodu oraz uwzględniając to, że zwykle syntezytory posiadają swoje nagłośnienie oraz wzmacniacze akustyczne, projektowane są bez uwzględnienia energooszczędności. Syntezytory zasilane są więc zwykle bezpośrednio z sieci energetycznej. Wynikająca z tego względna trudność w przenoszeniu przysparza niekiedy problemów użytkownikom, którzy dla przykładu nie potrzebują nagłośnienia, korzystają bowiem ze słuchawek. Dodatkowo gdy w danym miejscu utrudniony jest dostęp do sieci energetycznej, prądożerne konstrukcje szybko zużywają energię ogniw bateryjnych zastosowanych do zasilania syntezytora. Szybki rozwój w dziedzinie techniki mikroprocesorowej umożliwia jednak konstrukcję syntezytora muzycznego pozbawionego negatywnych cech wspomnianych powyżej. Dlatego w tej pracy podjęto próbę przedstawienia opisu założeń i konstrukcji urządzenia pozostającego

stosunkowo tanim, energooszczędnym i potencjalnie posiadającym szerokie możliwości w dowolności kreowania dźwięku oraz w jak największym stopniu przenoszalne.

Dla sprostania takim wymaganiom, w niniejszej pracy, zaproponowano użycie energooszczędnych układów: procesora sygnałowego DSP (*digital signal processor*), kodeka audio oraz mikrokontrolera. Wykorzystano fakt, że na rynku układów półprzewodnikowych dostępna jest wielość tanich stałoprzecinkowych procesorów sygnałowych, które zapewniając bardzo małe zużycie energii. Przykładem takich układów jest rodzina TMS320C55x o najniższym współczynniku zużycia energii 0.15 mW/MHz w zastosowanym układzie. Mimo zminimalizowanego zużycia energii wymienione układy zapewniają wysoką wydajność obliczeniową sięgającą, w zastosowanym w tej pracy procesorze sygnałowym, do 200 MMACS (*millions MAC operations on second*, MAC – *multiply-accumulate operation*) przy maksymalnej osiąganey częstotliwości pracy rdzenia 100 MHz. Zastosowanie jednego z takich układów może zapewnić duże możliwości w zakresie syntezy dźwięku bez konieczności zwiększenia poboru mocy przez urządzenie. Z drugiej strony zastosowanie mikrokontrolera małej mocy komunikującego się z procesorem sygnałowym przy pomocy portu szeregowego zapewnia obsługę klawiatury dedykowanej oraz interfejsu nastaw z potencjometrami odciażając układ DSP zajmujący się syntezą dźwięku. Zastosowanie potencjometrów w fizycznym interfejsie pozwala użytkownikowi na łatwe wprowadzanie parametrów syntezy dźwięku umożliwiając płynne zmiany w barwie dźwięku w czasie rzeczywistym. Daje to intuicyjny interfejs do obsługi syntezy. Z tych właśnie względów zastosowano w konstrukcji 32-bitowy mikrokontroler z rdzeniem Cortex-M0+ firmy STMicroelectronics.

Rozdział drugi niniejszej pracy prezentuje w przekroju zagadnienia związane z budową i zasadą działania syntezyatorów muzycznych. Zadanie to zostało zrealizowane poprzez przegląd literaturowy oraz z uwzględnieniem dotychczasowej wiedzy autora pracy. Zdefiniowano pojęcie syntezyatora muzycznego w formie użytecznej dla dalszych rozważaniach. Przeprowadzony jest w tym rozdziale przegląd technik stosowanych do realizacji syntezy dźwięku ze szczególnym uwzględnieniem tych metod, które zaimplementowane zostały w opisywanym w tej pracy projekcie. Oprócz opisu algorytmów syntezy dokonano przeglądu metod realizowania generacji obwiedni dźwięku.

Rozdział trzeci zawiera opis projektu syntezyatora muzycznego. W tym miejscu opisane są rozwiązania techniczne – konstrukcje układów oraz algorytmy dla oprogramowania. Wyróżnione zostały bloki funkcjonalne urządzenia oraz połączenia między nimi. W tym miejscu umotywowano również dokonane wybory ze względu na czynniki takie jak zużycie energii elektrycznej oraz wydajność, ale także czasu reakcji układu dla uzyskania wrażenia zmiany parametrów syntezy w czasie rzeczywistym. Rozważania prowadzone są w nawiązaniu do użytych w systemie układów scalonych: procesora sygnałowego TMS320C5535, mikrokontrolera STM32L053R8 oraz układu kodeka TLV320AIC3204.

Rozdział czwarty opisuje sposoby implementacji rozwiązań zaprezentowanych w rozdziale trzecim. Przedstawiono w nim m.in fizyczną konstrukcję klawiatury, płytkę PCB zaprojektowaną na potrzeby budowy interfejsu nastaw parametrów syntezy oraz implementację najważniejszych fragmentów kodu napisanego w języku C dla układów

mikroprocesorowych. Opisano konfigurację poszczególnych układów peryferyjnych użytych w projekcie.

Rozdział piąty zawiera opis testów wykonanych na zbudowanym prototypie. W tym rozdziale opisane są zarówno wrażenia słuchowe testującego jak i przedstawione są testy obiektywne składające się na testy weryfikacji wartości kluczowych zmiennych napisanych programów poprzez debugowanie oraz analiza oscylogramów.

W rozdziale szóstym dokonano podsumowania wyników niniejszej pracy. Opisano co udało się zrealizować, co wymaga dalszej pracy oraz wymieniono możliwości rozbudowy urządzenia w przyszłości.

2. Przegląd zagadnień związanych z syntezą dźwięku

2.1. Syntezator muzyczny

Syntezator jest elektronicznym instrumentem muzycznym pozwalającym na generowanie przebiegów akustycznych zaprojektowanych przez konstruktora. Współczesna elektronika umożliwia tworzenie skomplikowanych sygnałów w pasmie akustycznym na wiele różnych sposobów. Zadaniem osoby tworzącej instrument jest zatem takie zaprojektowanie interfejsu, aby użytkownik w łatwy sposób mógł otrzymać pożądany tembr dźwięku. Najprostszym sposobem na osiągnięcie tego celu jest ograniczenie wyboru do zestawu wcześniej zdefiniowanych przez projektanta dźwięków. Podejście takie, choć prostsze, odbiera urządzeniu elastyczność w zakresie dynamicznej zmiany parametrów różnych aspektów dźwięku przez użytkownika.

Pierwsze syntezatory były konstrukcjami analogowymi przez co budowa instrumentów o dużych możliwościach z zakresu zmiany charakterystyk dźwięku była skomplikowana. Rozwój techniki cyfrowej umożliwił jednak konstrukcję urządzeń o tych cechach w bardziej kompaktowym wydaniu oraz dużo bardziej energooszczędnych.

Syntezator muzyczny wykorzystywany jest często do tworzenia imitacji sygnałów akustycznych wydawanych przez klasyczne instrumenty, bądź do generowania całkowicie nowych brzmień.

2.2. Cyfrowa synteza dźwięku

2.2.1. Typy syntezy dźwięku

Wśród wielu technik syntezy dźwięku wyróżnia się głównie dwie podstawowe: syntezę addytywną oraz syntezę subtraktywną dźwięku [1]. Synteza addytywna polega na budowaniu barwy dźwięku poprzez sumowanie przebiegów akustycznych, głównie związanych ze składowymi harmonicznymi w jeden dźwięk. Synteza subtraktywna polega zaś na tym, że bogaty w składowe harmoniczne sygnał zostaje poddany filtracji, tak aby poprzez usunięcie odpowiednich składowych dźwięku uzyskać pożądane brzmienie. Urządzenie opisywane w tej pracy korzysta z techniki syntezy addytywnej.

2.2.2. Algorytmy syntezy addytywnej

Syntezę addytywną można powiązać z analizą częstotliwościową sygnałów [2]. Posługując się narzędziem matematycznym jakim jest szereg Fouriera można rozłożyć funkcję okresową na sumę funkcji trygonometrycznych. Okresowy sygnał akustyczny wyrażony jako funkcja czasu może być zapisany poprzez szereg Fouriera jako [3]:

$$y(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} [a_k \cos(2\pi k f_0 t) - b_k \sin(2\pi k f_0 t)] \quad (1)$$

$$= a_0 + \sum_{k=1}^{\infty} r_k \cos(2\pi k f_0 t + \phi_k) \quad (2)$$

gdzie:

$f_0 = \frac{1}{T}$ - częstotliwość podstawowa,

$a_k = r_k \cos(\phi_k)$ dla $k \geq 0$,

$b_k = r_k \sin(\phi_k)$ dla $k \geq 1$,

$\phi_k = \text{atan}(b_k, a_k)$ - przesunięcie fazowe k -tej harmonicznej,

$r_k = \sqrt{a_k^2 + b_k^2}$ - amplituda k -tej harmonicznej.

Z powyższej zależności wynika to, że możliwe jest wytworzenie odpowiedniego dźwięku znając jego charakterystykę amplitudową poprzez dodanie do siebie kolejnych składowych harmonicznymi częstotliwości podstawowej, dobierając odpowiednio amplitudy i fazy tych składowych. Uwzględniając to, iż słuch ludzki nie odróżnia sygnałów akustycznych różniących się wyłącznie fazą oraz że nie jest w stanie usłyszeć składowej stałej powyższe wyrażenie można uprościć. Dodatkowo uwzględniając to, że synteza ma odbywać się na drodze cyfrowej otrzymujemy postać:

$$y(n) = \sum_{k=1}^{\infty} r_k \sin\left(\frac{2\pi kn}{N}\right) \quad (3)$$

gdzie:

n – numer próbki,

N – okres wyrażony w próbkach składowej podstawowej.

Ilość użytych składowych harmonicznymi do budowy dźwięku ograniczona jest przez częstotliwość próbkowania. Z twierdzenia o próbkowaniu wynika, że maksymalną częstotliwość składowych widmowych zawartych w sygnale spróbkowanym możliwych do odtworzenia wyrazić można przez częstotliwość Nyquista czyli odpowiadającą połowie częstotliwości próbkowania:

$$f_{maks} \leq \frac{f_s}{2} \quad (4)$$

gdzie:

f_{maks} - maksymalna częstotliwość użytej składowej harmonicznej,

f_s - użyta częstotliwość próbkowania.

2.2.4. Obwiednia dźwięku

Przez obwiednię sygnału rozumie się funkcję przyporządkowującą chwilową amplitudę sygnału w czasie. Obwiednia jest krzywą opisującą ekstrema przebiegu. W tej pracy rozumie się ją jako współczynnik z zakresu $[0,1]$ zależny od czasu, modyfikujący wartość chwilową sygnału zgodnie z zależnością dla danej próbki n :

$$y_{out} = \sum_{t=1}^T ob_t(n) * y_t(n) \quad (5)$$

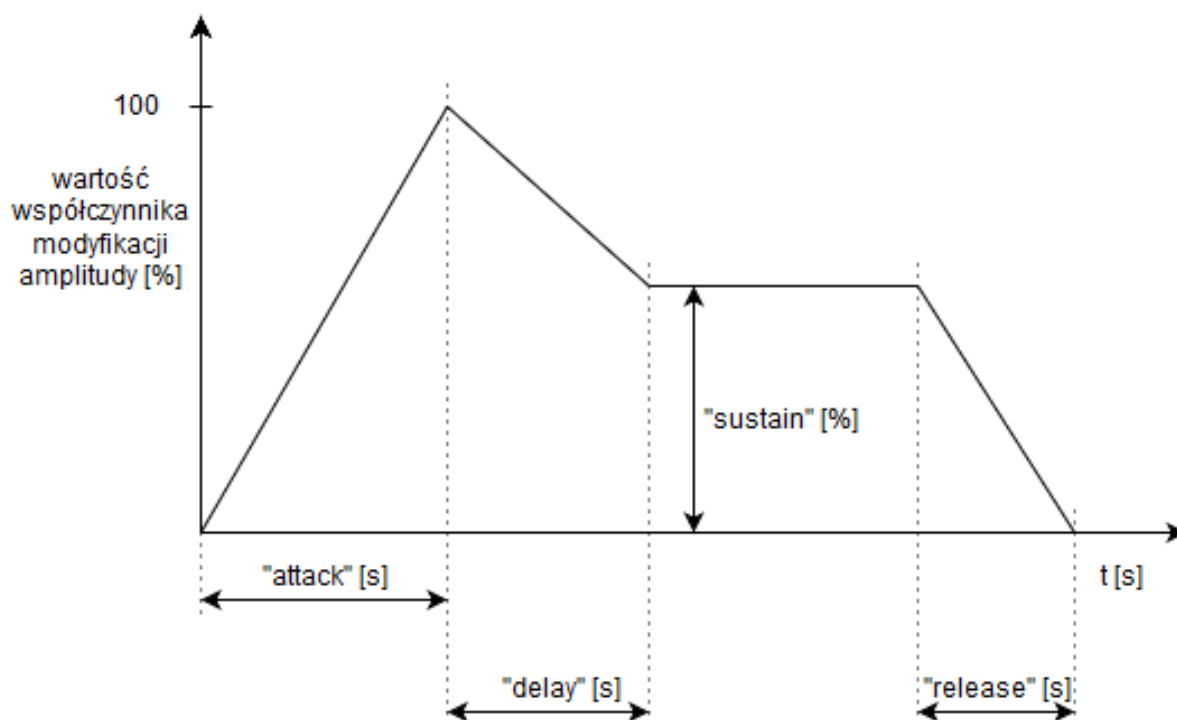
gdzie:

$y_{out}(n)$ - sygnał wyjściowy ze zmodulowaną obwiednią,
 $y_t(n)$ - sygnał powstały z syntezy addytywnej,
 $ob_t(n)$ - współczynnik obwiedni z zakresu [0,1].

W syntezatorach (głównie odnosi się to do modułowych syntezatorów analogowych) elementem modulującym obwiednię w żądany sposób jest generator obwiedni [1]. Stosowanie generatora obwiedni w syntezatorach ma na celu imitację efektów zmiany dynamiki dźwięku w czasie, występującego w instrumentach klasycznych, takich jak np. stopniowe wygaszanie się dźwięku następującym po uderzeniu w bęben. Możliwe jest ustawianie parametrów, tak aby tworzyć nowe, niewystępujące w klasycznych instrumentach efekty brzmieniowe.

W urządzeniu opisywanym w tej pracy zastosowano model obwiedni dźwięku ADSR („attack, decay, sustain, release”). Model ten opisuje obwiednie generowanego przez syntezator dźwięku poprzez cztery parametry (rys. 1):

- „attack” – parametr określający czas narostu głośności dźwięku od naciśnięcia klawisza, aż do maksymalnej głośności dźwięku,
- „delay” – parametr określający czas jaki upływa od osiągnięcia maksymalnej głośności aż do ustabilizowania się głośności dźwięku na ustalonym poziomie,
- „sustain” – parametr określający poziom głośności dźwięku po ustaniu pierwszych dwóch faz, dźwięk o takiej głośności trwa tak długo jak długo naciśnięty jest klawisz,
- „release” – parametr określający czas gaśnięcia dźwięku – które rozpoczyna się po zwolnienia przez grającego klawisza.



Rysunek 1 Wizualizacja parametrów obwiedni ADSR

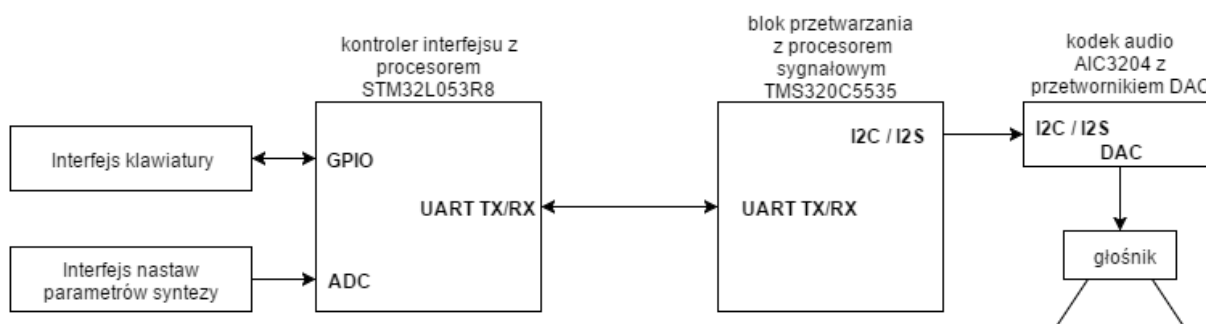
Parametry zdefiniowane w modelu obwiedni ADSR w zależności od syntezy mogą być modyfikowane przez grającego na instrumencie w szerokim zakresie, bądź też ograniczone do paru nastaw albo też raz tylko zdefiniowane. W projektowanym urządzeniu dano użytkownikowi możliwość dynamicznej zmiany tych parametrów co daje dodatkową elastyczność w zakresie kreowania dźwięku.

3. Projekt systemu

Projektowane urządzenie składa się z pięciu bloków funkcjonalnych. Na urządzenie składają się:

- moduł klawiatury,
- interfejs nastaw parametrów syntezy,
- kontroler interfejsu z 32-bitowym procesorem firmy STM z rdzeniem CortexM0+,
- jednostka przetwarzania z procesorem sygnałowym TMS320C5535,
- kodek audio z przetwornikiem cyfrowo-analogowym.

Interfejs użytkownika, na który składają się moduł klawiatury oraz blok interfejsu nastaw parametrów syntezy, połączony jest i sterowany przez energooszczędny mikrokontroler STM32L053C8 pełniący zadanie kontrolera interfejsu. Pomiędzy kontrolerem interfejsu, a blokiem przetwarzania DSP ustanowiono asynchroniczną komunikację szeregową służącą do przesyłania wartości parametrów syntezy oraz informacji o przyciśniętych klawiszach w danym momencie czasu do bloku syntezy. Komunikacja procesora sygnałowego z kodekiem audio odbywa się poprzez interfejsy I2C oraz I2S (rys. 2).

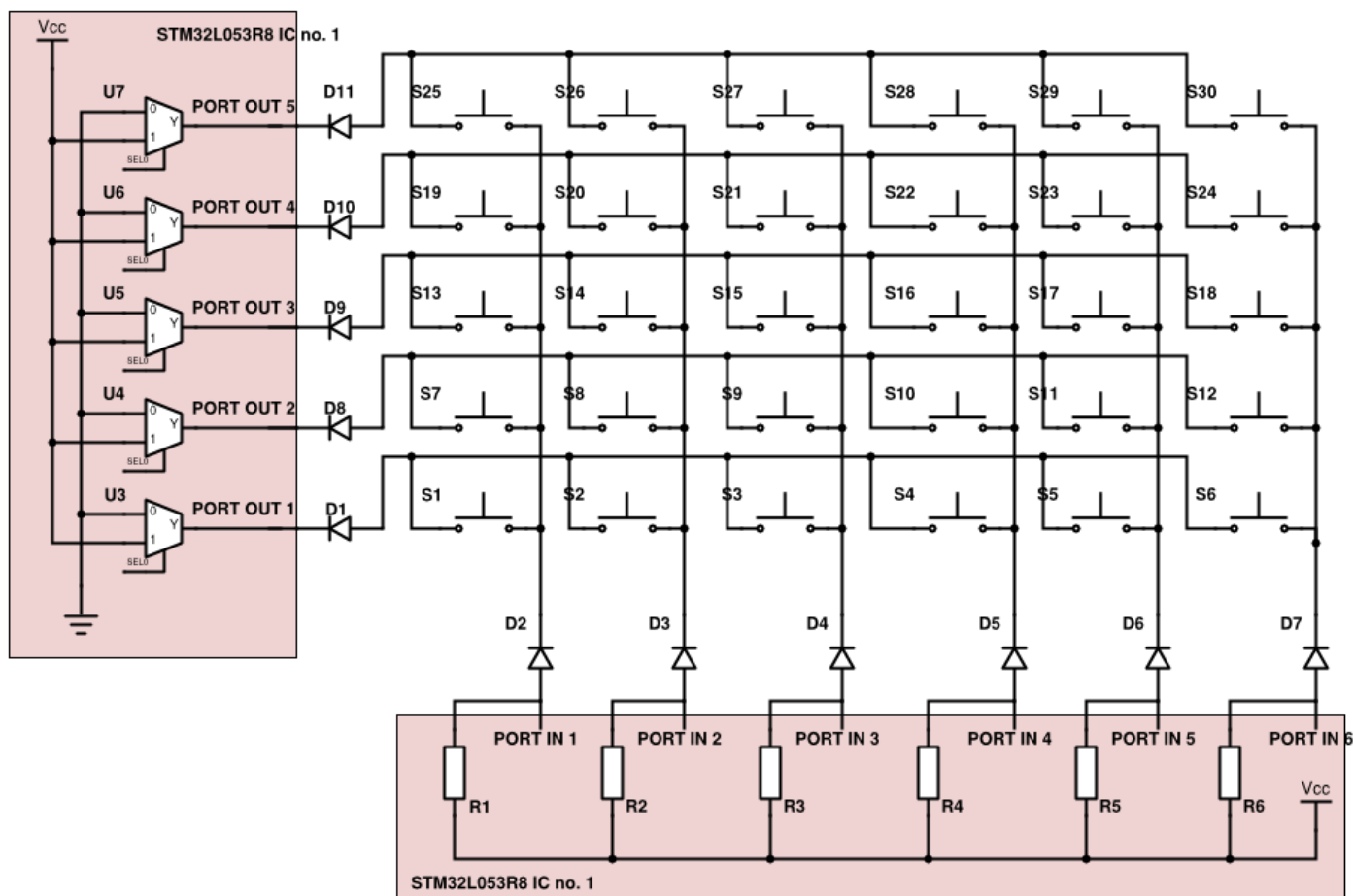


Rysunek 2 Ilustracja zależności pomiędzy blokami na który składa się projektowany syntezytor muzyczny

3.1. Opis systemu

3.1.1. Interfejs klawiatury

Moduł klawiatury został zaprojektowany jako matryca przycisków. Przewidziano możliwość obsługi 5-ciu zestawów po 6 przycisków, co sumarycznie daje liczbę 30-stu klawiszy. Stan naciśnięcia przycisków sprawdzany jest poprzez wejścia cyfrowe dla których przewidziano połączenie za diodami oznaczonymi D2 – D7 (rys. 3). Każdy zestaw przycisków (zestawy to przyciski oznaczone odpowiednio S1 – S6, S7 – S12, S13 – S18, S19 – S24, S25 – S30) aktywowany jest do odczytu poprzez wysterowanie odpowiednich wyjść cyfrowych dla których przewidziano połączenie za diodami D1 i D8 – D11 (rys. 3). Próba odczytu odpowiedniej linii musi być poprzedzona wpisaniem logicznego „0” na tej i tylko tej linii. Odczyt na cyfrowym wejściu wykazujący stan logicznego „0” świadczy o wciśniętym przycisku i odwrotnie: stan logicznej „1” świadczy o tym, że dany przycisk nie został wciśnięty. Rezystory podciągające linie do napięcia dodatniego zasilania ustawiane są programowo w procesorze STM32L053R8, tak samo sprawa wygląda z multiplekserami (rys.



3):

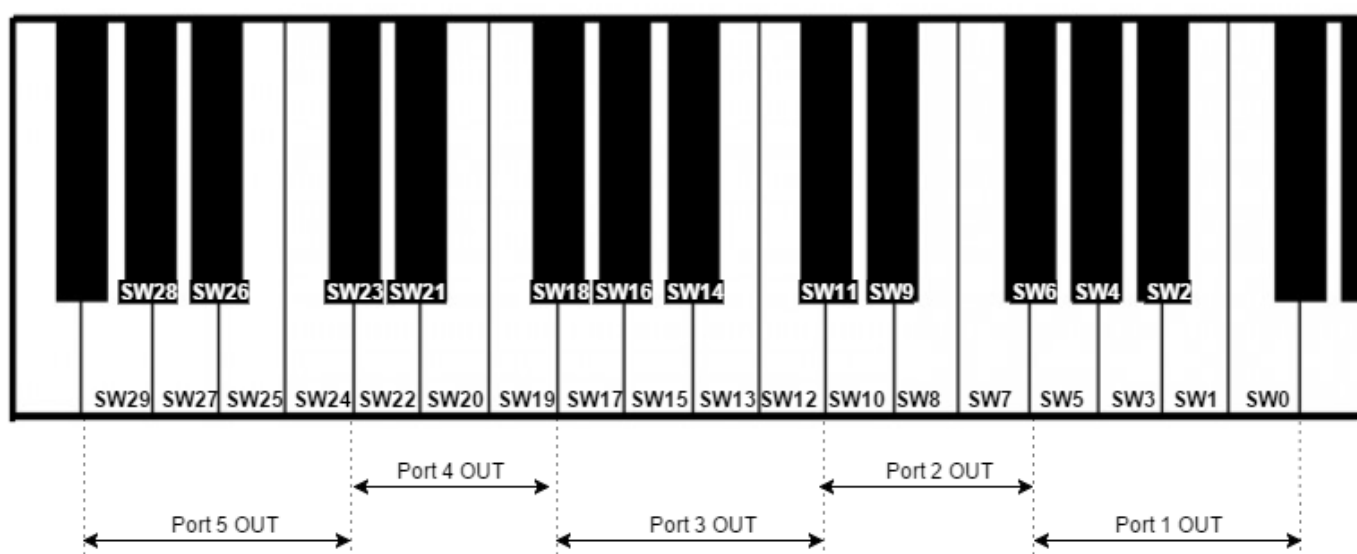
Rysunek 3 Schemat ideowy modułu klawiatury

Przyciski są przyporządkowane numerami do odpowiednich klawiszy od najwyższego tonu (najdalszy na lewo klawisz) co pół tonu (rys. 4). Tak zatem przycisk oznaczony jako S1

(rys. 3) odpowiada najwyższemu tonowi, zaś przycisk oznaczony jako S30 tonowi najniższemu. Konwencja przyjęta odpowiada numeracji bitów w 32-bitowym słowie jako odniesienie do fizycznej budowy klawiatury.

Podłączenie bloku klawiatury do kontrolera interfejsu poprzez wejścia oraz wyjścia cyfrowe umożliwia kolejno odczyt stanu zestawów poszczególnych przycisków. W równych odstępach czasu, co 10 ms, wysterowane są wyjścia kolejno jedną z kombinacji stanów logicznych : {„0111”, „1011”, „1101”, „1110”}, aktywując odpowiedni zestaw 6-ciu przycisków, co pozwala na odczyt ich stanu poprzez wejścia cyfrowe. Oprogramowanie mikrokontrolera dba o to, aby odczyt stanu przycisków odbywał się odpowiednio opóźniony względem momentu zmiany stanu wyjść. Dane rejestrowane są przez kontroler w 32 bitowym słowie, a informacja przesyłana jest dalej do procesora sygnałowego. Opis fizycznej realizacji klawiatury oraz oprogramowanie kontrolera interfejsu znajduje się w kolejnym rozdziale.

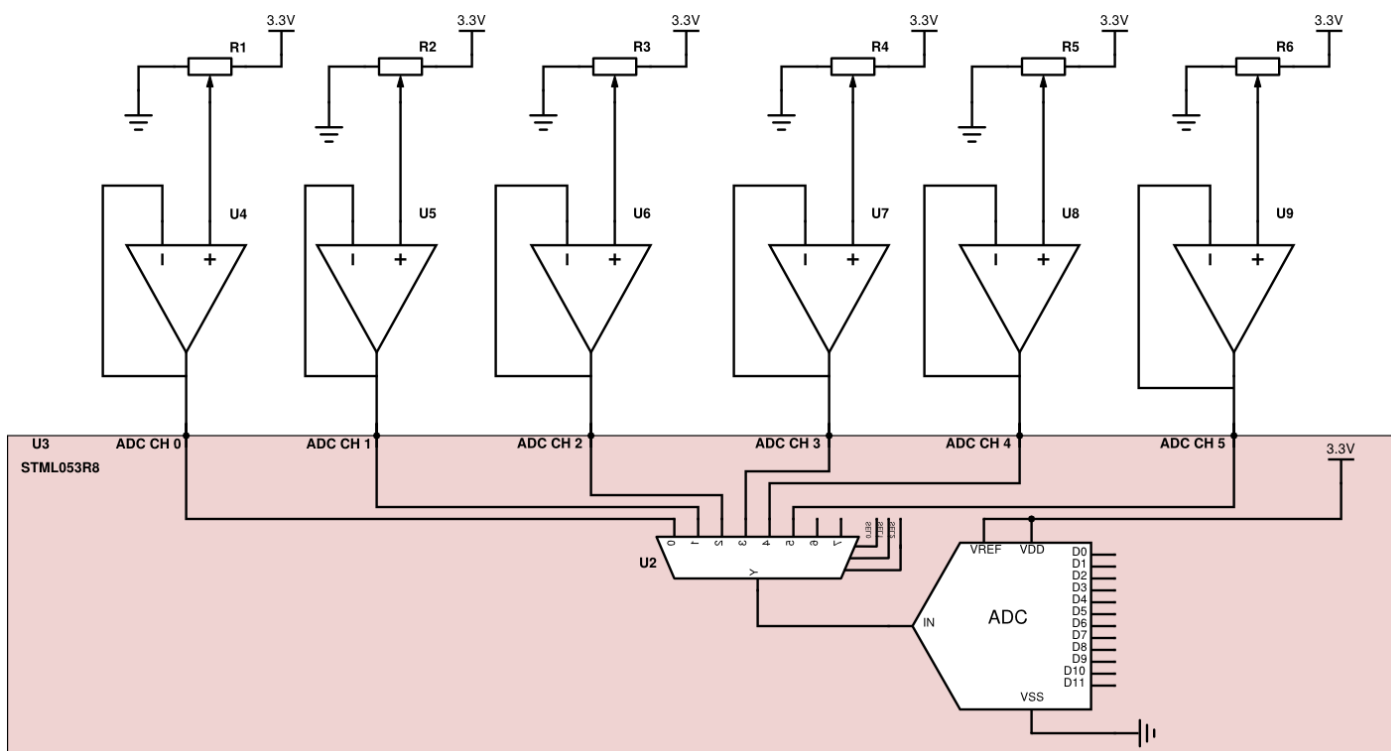
Rysunek 4 Wizualizacja interfejsu klawiatury wraz z oznaczeniami odniesionymi do schematu ideowego



3.1.2. Interfejs nastaw parametrów syntezy

Moduł interfejsu nastaw parametrów syntezy dźwięku zrealizowany jest poprzez użycie potencjometrów zasilanych przez napięcie zasilania procesora STM32L053R8 (rys. 5). Aby zredukować rezystancję wyjściową układu dzielnika napięcia zastosowano wzmacniacz operacyjny w układzie bufora. Napięcie pochodzące z jego wyjścia doprowadzone jest na wejście analogowe procesora STM32L053R8. Użytych zostało 6 potencjometrów o wartościach rezystancji po 1 MΩ dla zminimalizowania poboru prądu przez dzielnik oraz co za tym idzie - 6 kolejnych kanałów przetwornika ADC .

Próbkowanie doprowadzonych napięć do wejść analogowych odbywa się co stały odcinek czasu, kolejno od wejścia „ADC CH0” do „ADC CH5” gdzie przełączanie kanałów realizowane jest sprzętowo. Przetwornik jest skonfigurowany do odczytu z rozdzielczością 8 bitów. Wyniki próbkowania przechowywane są w dwóch 32-bitowych zmiennych i dalej za pomocą łącza szeregowego przesyłane do procesora sygnałowego.



Rysunek 5 Schemat ideowy kontrolera nastaw parametrów syntezy

Każdy z potencjometrów umożliwia użytkownikowi na zmianę jednego z parametrów syntezy. Informacja o stanie parametru przesyłana jest pośrednio przez kontroler interfejsu do modułu syntezy w czasie rzeczywistym tak aby użytkownik poprzez manipulację nastaw rezystorów mógł na bieżąco słyszeć uzyskany rezultat. Następujące parametry syntezy przyporządkowane są do potencjometrów:

- POT1 R1 – parametr „attack” obwiedni ADSR,
- POT2 R2 – parametr „decay” obwiedni ADSR,
- POT3 R3 – parametr „sustain” obwiedni ADSR,
- POT4 R4 – parametr „release” obwiedni ADSR,
- POT5 R5 – parametr kontrolujący amplitudę składowych harmonicznym nieparzystych generowanego dźwięku,
- POT6 R6 – parametr kontrolujący amplitudę składowych harmonicznym parzystych generowanego dźwięku.

3.1.3. Kontroler interfejsu

Do realizacji funkcji kontrolera interfejsu wykorzystano 32-bitowy mikrokontroler STM32L053R8 firmy STMicroelectronics z rdzeniem „Cortex M0+” firmy ARM. Głównym kryterium przy wyborze odpowiedniego procesora była minimalizacja zużycia energii w systemie. Użyty mikrokontroler cechuje się niskim zużyciem prądu na poziomie 139 $\mu\text{A}/\text{MHz}$ przy pracy aktywnej [10], dodatkowo oferując dużą elastyczność w zakresie dostosowania wydajności do potrzeb obliczeniowych co skutkuje wysoką efektywnością energetyczną. Kiedy nie jest potrzebna aktywność rdzenia procesora, możliwe jest wprowadzenie go w stan

uśpienia co skutkuje obniżeniem zużycia prądu do 0.27 μ A w stanie „standby” (czuwania). Producent układu implementuje energooszczędne układy peryferyjne co umożliwia dalsze ograniczenie konsumpcji energii elektrycznej oraz pozwala na wyłączenie peryferii nieużywanych. Na szczególną uwagę zasługuje układ LPUART (*low power universal asynchronous receiver/transmitter*) wykorzystywany do komunikacji w tym projekcie [9]. Mimo zastosowanych rozwiązań mikroprocesor zachowuje wysoką wydajnością charakterystyczną dla procesorów 32-bitowych sięgającą 0.95 MIPS/MHz (*milion instruction per second*), gdzie maksymalną częstotliwością pracy rdzenia jest 32 MHz.

Kontroler interfejsu ma za zadanie obsługę modułów interfejsu użytkownika, a więc modułu nastaw parametrów oraz klawiatury. Dodatkowym zadaniem kontrolera jest wysyłanie łączem szeregowym zestawu parametrów potrzebnym do zdefiniowania kształtu syntezy oraz informacji dotyczącej wciśnięcia (bądź braku wciśnięcia) klawiszy na klawiaturze. Innymi słowy kontroler interfejsu pełni funkcję sterownika modułów sterujących syntezą, z którym procesor sygnałowy komunikuje się za pomocą portu szeregowego.

Moduł kontrolera interfejsu połączony jest bezpośrednio (rys. 2 z interfejsem klawiatury za pomocą cyfrowych wejść i wyjść GPIO (*general purpose input output*) (rys. 3), interfejsem nastaw parametrów syntezy za pomocą wejść analogowych kanałów przetwornika analogowo-cyfrowego (rys. 5) oraz poprzez wyprowadzenia portu szeregowego TX i RX z procesorem sygnałowym TMS320C5535 (rys. 2).

3.1.4. Synteza dźwięku

Algorytmy syntezy muzyki w synteźatorze muzycznym implementowane są na układzie procesora TMS320C5535 firmy Texas Instruments. Jest to 16-bitowy, stałoprzecinkowy procesor sygnałowy wyposażony w wiele funkcjonalności sprzętowych ułatwiających i przyspieszających proces przetwarzania sygnałów. Tak jak w przypadku procesora STM32L053R8 głównym kryterium przy wyborze była minimalizacja zużycia energii w projektowanym systemie wbudowanym. Zastosowany procesor pochodzi z rodziny procesorów C55x cechującą się najniższym zużyciem energii wśród procesorów sygnałowych dostępnych na rynku. Procesor może być zasilany napięciem sięgającym jedynie 1,3 V przy częstotliwości pracy rdzenia 100 MHz. Zużycie energii procesora utrzymuje się wtedy na poziomie 0.15 mW/MHz. Przy wspomnianej wcześniej wartości częstotliwości pracy osiągnięta jest wydajność 200 MMACS (*Million Multiply Accumulate Cycles per Second*) [8].

Blok syntezy nazywany będzie w tej pracy również jako blok przetwarzania DSP.

3.1.4.1. Algorytm syntezy addytywnej

Synteza dźwięku w opisywanym urządzeniu realizowana jest zgodnie z założeniami metody syntezy addytywnej. Wybór tej techniki podyktowany został przez prostotę jej implementacji.

Parametry syntezy oraz informacja o przyciśnięciu klawiszy otrzymywane są poprzez komunikację szeregową od kontrolera interfejsu. Na ich podstawie oprogramowanie procesora sygnałowego dokonuje generacji sygnału.

Synteza pojedynczego dźwięku przeprowadzana jest zgodnie z równaniem:

$$y_{out}(n) = \text{pot5} * ob_{out}(n) * \sum_{k=0}^K \sin\left(\frac{2\pi(2k+1)n}{N}\right) + \text{pot6} * ob_{out}(n) * \sum_{j=1}^J \sin\left(\frac{2\pi(2j)n}{N}\right) \quad (6)$$

gdzie:

$y_{out}(n)$ – próbka wyjściowa na przetwornik,

pot5 – wartość z zakresu [0,1) modyfikująca maksymalną amplitudę składowych harmonicznym parzystych pochodząca z odczytu wskazania nastawy potencjometru POT5,

pot6 – wartość z zakresu [0,1) modyfikująca maksymalną amplitudę składowych harmonicznym nieparzystych pochodząca z odczytu wskazania nastawy potencjometru POT6,

$ob_{out}(n)$ – współczynnik modyfikacji amplitudy o wartości z zakresu [0,1) generujący obwiednie ADSR,

K – ilość składowych harmonicznym nieparzystych,

J – ilość składowych harmonicznym parzystych.

Parametry K i J są wprowadzone przez użytkownika za pomocą odpowiednich przycisków interfejsu.

Ze względu na 16-bitową, stałoprzecinkową architekturę procesora sygnałowego TMS320C5535 obliczenia przeprowadzane są w arytmetyce stałoprzecinkowej i1q15. Próbka wyjściowa również pozostaje w tym formacie, gdyż kodek do którego jest wysyłana, przeprowadza 16-bitowe przetwarzanie cyfrowo-analogowe.

Podstawowym sygnałem generacji jest funkcja sinus (równanie 6). Ze względu na łatwość w implementacji zastosowano metodę tablicową generacji sygnału sinusoidalnego. W programie przechowywane jest 12 tablic zawierających próbki jednego okresu sygnałów o 12 częstotliwościach które składają się na dźwięki najniższej oktawy. Dźwięki z oktaw wyższych uzyskiwane są poprzez odpowiedni skok po tablicy próbek sygnałów oktawy podstawowej.

3.1.4.2. Generacja obwiedni ADSR

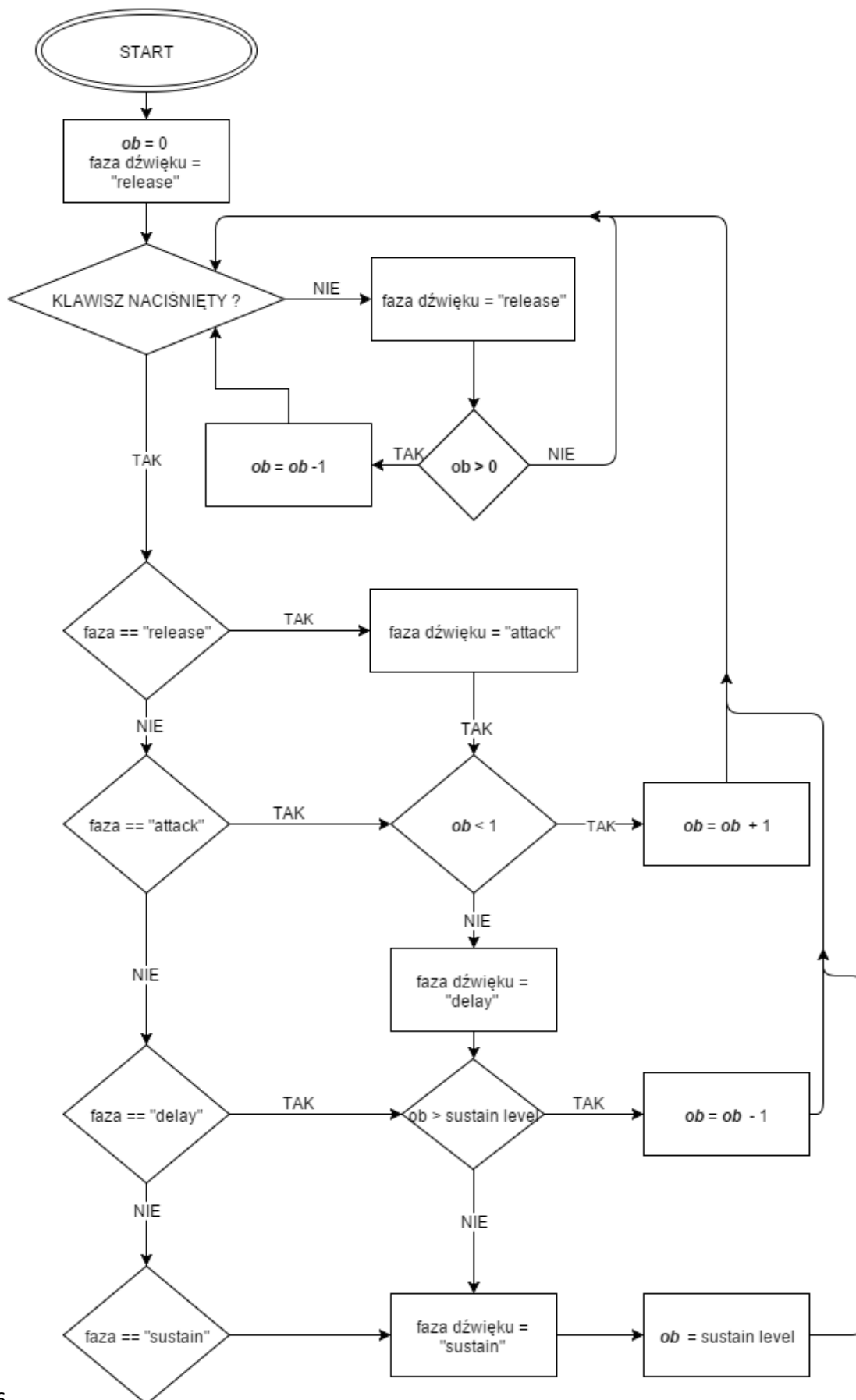
Cztery pierwsze parametry syntezy otrzymane od kontrolera interfejsu użytkownika dostarczają informacji o postaci obwiedni ADSR dźwięku. Na nią składają się cztery 8-bitowe wartości określające odpowiednio czas trwania faz: „attack”, „delay”, „release” oraz poziomu głośności dźwięku „sustain” po ustaniu fazy „delay”. Wykorzystuje się pełen zakres 8-bitowego parametru syntezy, który jest przekształcany liniowo w odpowiednią wielkość fizyczną charakteryzującą proces generacji obwiedni. Zakresy tych wielkości zostały określone jako:

- zakres parametru „attack” na od 0 ms do 2000 ms,
- zakres parametru „decay” na od 0 ms do 2000 ms,
- zakres parametru „sustain” na od 0% do 100%,

- zakres parametru „release” na od 0 ms do 2000 ms.

Generacja obwiedni realizowana jest zgodnie z (równanie 6). Zmienna **ob** reprezentująca współczynnik obwiedni zmienia się w zakresie od [0,1) mając wpływ na amplitudę sygnału w danej chwili. Algorytm przetwarzania (rys. 6) musi określić w jakiej fazie znajduje się dźwięk i odpowiednio modyfikować współczynnik generacji obwiedni. Ponieważ przyjęto model zmiany linowej amplitudy sygnału w czasie i uwzględniając, że system korzysta z arytmetyki stałoprzecinkowej i1q15, wystarczy, że zmienna **ob** jest albo inkrementowana lub dekrementowana w ustalonych odstępach czasu, albo (w fazie „sustain”) pozostaje bez zmian na ustalonym przez parametr poziomie. Główną zaletą zastosowania takiego algorytmu ze modyfikowanym w czasie współczynnikiem **ob** jest to, że nawet jeśli muzyk puści wciśnięty klawisz podczas trwania fazy „attack” lub „delay” dźwięk zamiast gwałtownie spadać do poziomu zadeklarowanego w parametrze „sustain”, zacznie zmniejszać swoją głośność stopniowo wchodząc w fazę „release” i ignorując wartość parametru „sustain”. Sytuacja wygląda analogicznie w przypadku kiedy wartość współczynnika **ob** nie zdąży spaść do poziomu zera w fazie „release”, użytkownik grający zdąży z powrotem wcisnąć klawisz. Nie występuje wówczas skok z aktualnego poziomu do poziomu 0 (który z definicji jest stanem początku fazy „attack” do której wchodzi dźwięk po naciśnięciu klawisza). Wady te występowałyby w algorytmie, który traktowałby fazy i związane z nimi współczynniki generacji obwiedni jako oddzielne niezależne wielkości. Każdemu klawiszowi przyporządkowany jest jeden współczynnik modulacji obwiedni, dlatego wybrzmiewanie różnych klawiszy jest od siebie niezależne.

Za transformację parametrów oraz realizowanie algorytmu generacji obwiedni odpowiada oprogramowanie bloku przetwarzania DSP.



6

Rysunek 6 Schemat przedstawiający algorytm generacji obwiedni ADSR generowanego sygnału dźwiękowego

3.1.4.3. Polifonia

Ze względu na prostotę implementacji i brak potrzeby zmiany dynamiki generowanych przez syntezytor dźwięków w czasie, zastosowano proste podzielenie zakresu dynamiki przetwornika cyfrowo-analogowego na równe części dla każdego generowanego dźwięku. Założono, że możliwe jest granie jednocześnie przez syntezytor 8-miu dźwięków. Mając na uwadze równanie dla generacji próbki pojedynczego dźwięku (równanie 6) oraz wymienione wyżej założenia polifonii otrzymujemy wzór wyrażający wartość próbki wyjściowej:

$$y_{out_{POLI}}(n) = \frac{1}{8} * \sum_{k=1}^8 y_{out_k}(n) \quad (7)$$

gdzie $y_{out_k}(n)$ – wartość próbki z równania 6, dla k-tego dźwięku dla n-tej próbki.

3.1.5. Komunikacja szeregową

Komunikacja między kontrolerem interfejsu, a blokiem przetwarzania DSP jest konieczna dla integracji systemu w pełni funkcjonalny instrument. Aby użytkownik syntezytora mógł wykreować pożądaną barwę dźwięku, wprowadzone przez niego parametry muszą być dostarczone do modułu w którym przeprowadzana zostaje synteza dźwięku.

Dla projektowanego systemu wybrano asynchroniczną komunikację przy pomocy dwóch układów UART wbudowanych jako układy peryferyjne bloku syntezy oraz kontrolera interfejsu. Ze względu na prostotę implementacji wykorzystane zostały dwie linie w zależności od układu odniesienia: linię Tx (transmisyjną) oraz Rx (odbiorczą). Cechy zastosowanej transmisji to:

- szybkość transmisji 19200 baudów (bitów na sekundę),
- słowo danych 8-bitowe,
- brak bitu parzystości,
- liczba bitów stopu równa 1,
- brak kontroli przepływu.

W projekcie zachodzi potrzeba wysłania 10-ciu bajtów danych (rys. 7) – 6 bajtów definiujących wartość parametrów syntezy oraz 4 bajtów zawierających informacje o wciśniętych w danej chwili klawiszach. Dodatkowo przed rozpoczęciem transmisji sekwencji danych wysyłany jest jeden bajt identyfikujący początek tej sekwencji. Bajt ten ma wartość w zapisie heksadecymalnym „0xAA” – czyli w zapisie binarnym „10101010”. W sumie w sekwencji wysyłanych jest 11 bajtów. Przy prędkości transmisji 19200 baudów oraz uwzględniając bity startu i stopu cała sekwencja przesyłana jest w czasie niecałych 6 ms.

1. Bajt inicjalizujący	BIT STARTU	10101010	BIT STOPU
2. Bajt danych	BIT STARTU	8 bitów danych z odczytu nastawy POT1	BIT STOPU
...			
7. Bajt danych	BIT STARTU	8 bitów danych z odczytu nastawy POT6	BIT STOPU
8. Bajt danych	BIT STARTU	pierwsze 8 bitów informacji o wciśniętych klawiszach	BIT STOPU
...			
11. Bajt danych	BIT STARTU	czwarte 8 bitów informacji o wciśniętych klawiszach	BIT STOPU

Rysunek 7 Struktura ramek pełnej sekwencji komunikacji szeregowej pomiędzy procesorem sygnałowym, mikrokontrolerem

3.1.6. Przetwarzanie cyfrowo-analogowe

Realizacja pełnego toru przetwarzania wymaga zainstalowania w systemie przetwornika cyfrowo-analogowego. Producent procesora sygnałowego użytego w projekcie, firma Texas Instruments, posiada w swojej ofercie układy kodeka audio małej mocy i małego napięcia pracy w zakresie takim jak procesor TMS320C5535. Ponieważ z płytą ewaluacyjną skojarzony jest układ tej rodziny nie było celowym zastosowanie przetwornika zewnętrznego. Układ kodeka to TLV320AIC3204. Układ ten charakteryzuje się 16-bitową rozdzielczością przetwornika DAC dla każdego z dwóch kanałów: lewego i prawego. Konfiguracja układu możliwa jest przy pomocy interfejsu I2C bądź interfejsu SPI. Przesył próbek dla przetwarzania odbywa się przy pomocy interfejsu I2S. Dodatkowym atutem wybranego kodeka audio jest to, iż posiada on dodatkowe bloki przetwarzania DSP, które mogą być wykorzystane dla odciążenia procesora sygnałowego – co daje możliwości dalszego rozwoju projektu w kierunku większej wydajności przetwarzania bloku syntezy [11].

W projektowanym systemie konfiguracja kodeka następuje poprzez interfejs I2C zaś przesyłanie próbek odbywa się przy pomocy interfejsu I2S (rysunek 6). Szybkość próbkowania jest stale ustawiona na wartość 48000 próbek na sekundę.

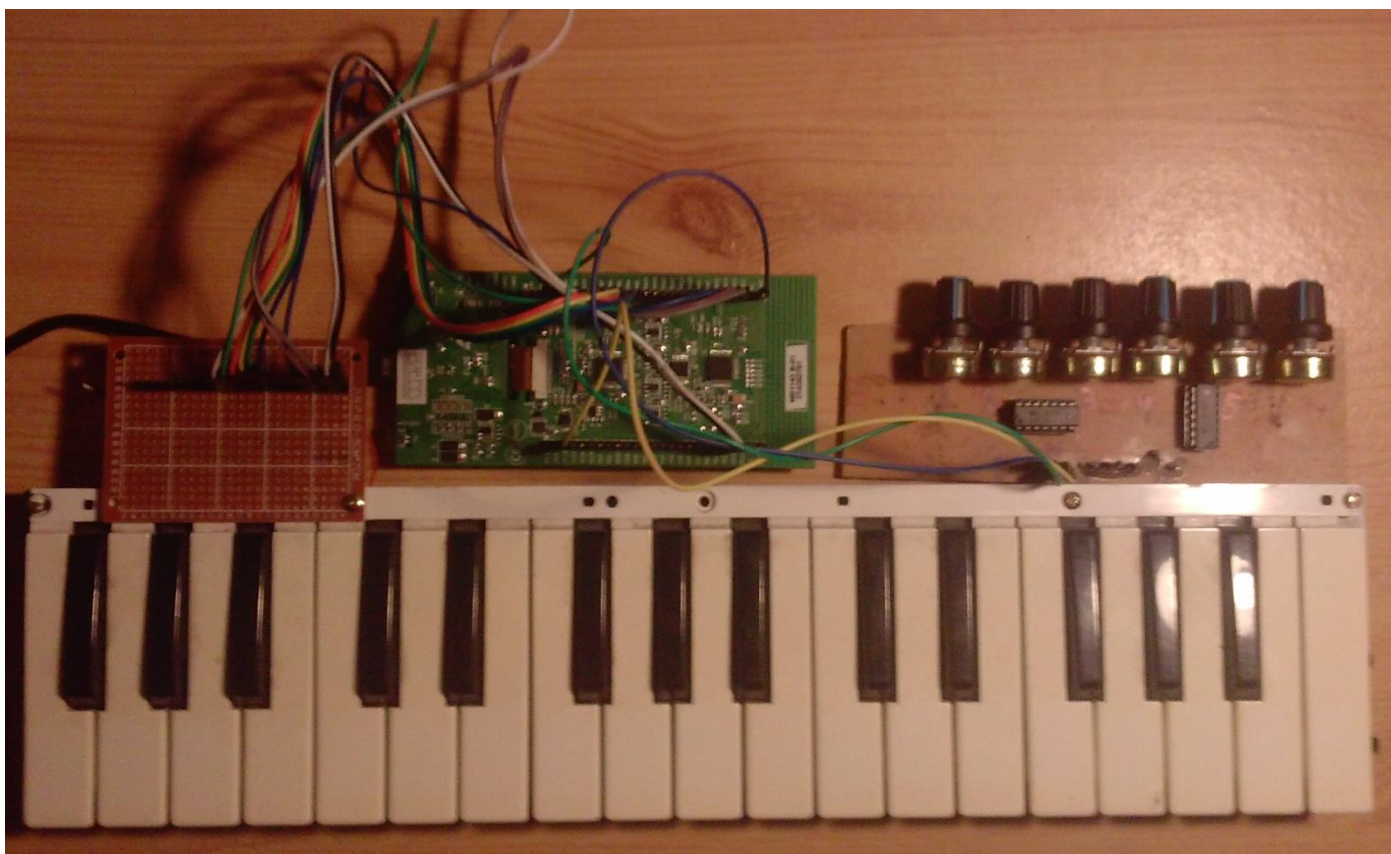
4. Implementacja funkcjonalności i fizyczna konstrukcja

W rozdziale tym opisano budowę prototypu projektowanego syntezyzatora muzycznego oraz oprogramowanie napisane na oba użyte układy procesorów implementujące zawarte w projekcie funkcjonalności.

4.1. Konstrukcja klawiatury

Do fizycznego wykonania bloku interfejsu klawiatury wykorzystano gotową klawiaturę pozyskaną z dekompozycji uszkodzonego urządzenia produkowanego seryjnie firmy Yamaha (zdjęcie 1). Struktura użytej klawiatury zgadza się z założeniami projektowymi.

Z racji, iż kontroler interfejsu to gotowa płytki ewaluacyjna, nie zaprojektowano dedykowanej płytki PCB dla tego modułu, a jedynie wykorzystano gotową płytkę uniwersalną - dla łatwiejszego dostępu do wejść i wyjść (zdjęcie 2).

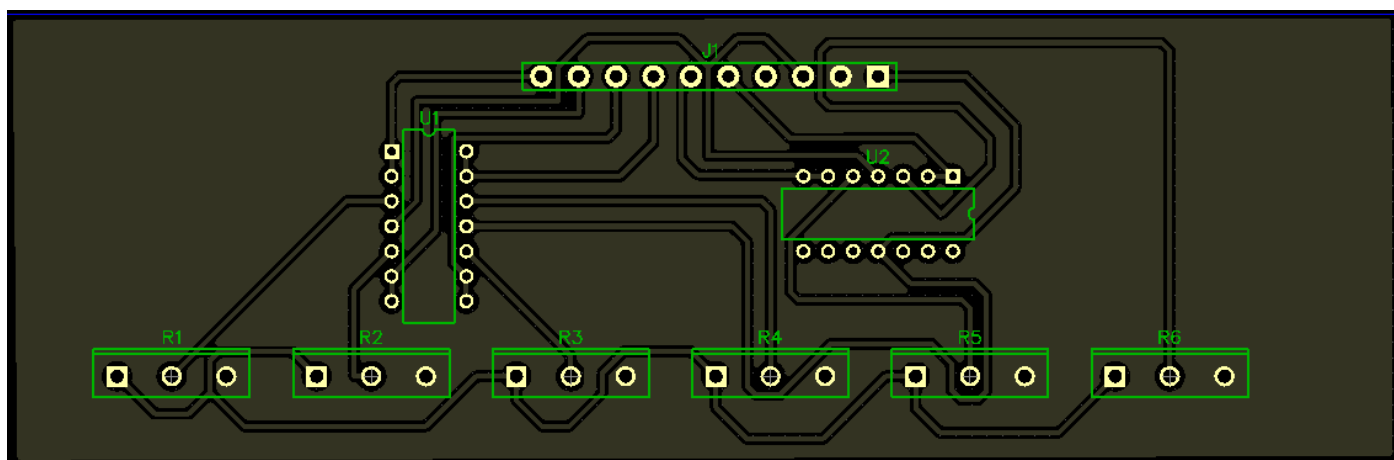


Zdjęcie 1 Na zdjęciu widoczny jest blok interfejsu użytkownika na który składa się klawiatura oraz interfejs nastaw parametrów syntezy oraz płytki ewaluacyjna pełniąca rolę kontrolera interfejsu

4.2. Konstrukcja interfejsu nastaw potencjometrów

Na blok interfejsu użytkownika składa się zaprojektowana na potrzeby konstrukcji prototypu płytki PCB (zdjęcie 2). Zamontowano na niej, zgodnie z projektem syntezyzatora, sześć potencjometrów o całkowitej rezystancji 1 M Ω każdy. Zgodnie ze schematem ideowym (rys. 5) każdy potencjometr podłączony jest do napięcia zasilania procesora STM32L053 czyli

3,3 V oraz do napięcia masy. Zastosowanie tak wysokiej rezystancji redukuje zużycie prądu do poziomu 3,3 μ A dla każdego dzielnika napięcia. Trzecie wyprowadzenie, wyjście układu dzielnika napięcia, podłączone jest do wejścia nieodwracającego bufora skonstruowanego na bazie jednego z czterech wzmacniaczy operacyjnych zawartych w użytym układzie scalonym LM324 firmy Texas Instruments dla zminimalizowania rezystancji wyjściowej układu do wartości pojedynczych omów. Zużycie prądu układu LM324 [12] jest na poziomie 700 μ A nie licząc prądu wyjściowego. Następnie wyjścia wzmacniaczy operacyjnych wyprowadzone są do złącz typu „goldpin” i dalej podłączone do odpowiednich wejść analogowych mikroprocesora użytego w konstrukcji bloku interfejsu. Płytkę PCB wykonano techniką termotransferu w technologii THT (*Through-Hole Technology*) – montażu przewlekane (zdjęcie 1).



Zdjęcie 2 Projekt dedykowanej płytki PCB dla interfejsu nastaw parametrów syntezy

4.3. Płytką ewaluacyjną STM32L0 DISCOVERY

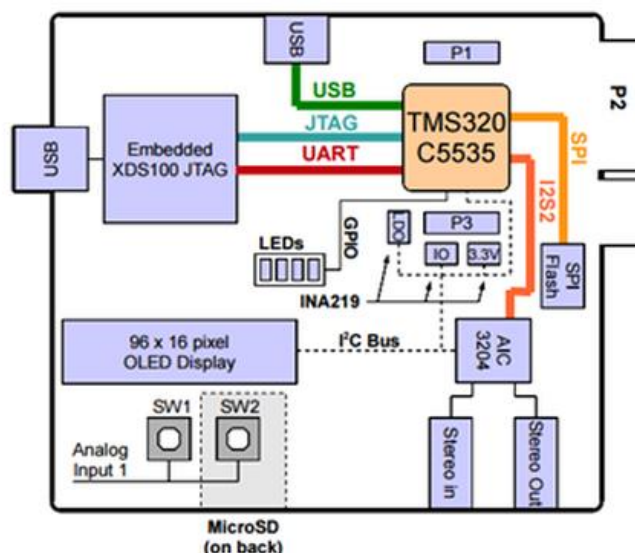
Platformą na której było rozwijane oprogramowanie dla mikrokontrolera STM32L053R8 była płytką ewaluacyjną STM32L0538DISCOVERY firmy STMicroelectronics (zdjęcie 3). Oprócz samego procesora na płycie znajduje się emulator ST-Link umożliwiający programowanie oraz debugowanie procesora poprzez złącze „micro USB”. Dodatkowym atutem płytki jest znajdujący się na niej kontroler portu szeregowego pozwalający na testowanie komunikacji szeregowej poprzez przekierowanie linii tx/rx układu UART mikrokontrolera na port USB, dzięki czemu przy użyciu odpowiedniego sterownika na komputerze PC można zobaczyć je jako wirtualny port COM. Kolejnym atutem płytki są wyprowadzenia pinów mikrokontrolera na złącza typu „goldpin” co umożliwia łatwe połączenie wyjść i wejść cyfrowych do modułu klawiatury oraz wejść analogowych do bloku nastaw syntezy. Tak samo możliwe jest wyprowadzenie linii TX/RX układu UART dla nawiązania komunikacji z procesorem sygnałowym. Platforma umożliwia dalszy rozwój projektu poprzez wykorzystanie znajdującego się na niej wyświetlacza typu „ePaper”, który może posłużyć jako wyświetlacz aktualnych parametrów syntezy oraz czujnika dotykowego pozwalającego na ich modyfikację [4].



Zdjęcie 3 Płytkę ewaluacyjną STM32L0DISCOVERY

4.4. Płytkę ewaluacyjną TI EZDSP5535

Platformą umożliwiającą rozwój oprogramowania dla stałoprzecinkowego procesora sygnałowego TMS320C5535 oraz zawierającą użyty w projekcie kodek audio TLV320AIC3204 jest płytkę ewaluacyjną o nazwie EZDSP5535 firmy Spectrum Digital (zdjęcie 4). Dzięki znajdującemu się na platformie emulatorowi USB XDS100v2, możliwe było programowanie i debugowanie układu procesora sygnałowego C5535. Na płytce znajduje się układ firmy FTDI o nazwie FT2232 umożliwiający nawiązanie komunikacji z komputerem PC poprzez układ peryferyjny UART procesora DSP. Umożliwia to testowanie komunikacji szeregowej za pomocą oprogramowania znajdującego się na komputerze PC. Układ kodeka komunikuje się z procesorem sygnałowym poprzez magistralę szeregową I2C co pozwala na konfigurację układu kodeka audio. Przesyłanie próbek wygenerowanych w procesorze sygnałowym odbywa się za pomocą magistrali I2S (zdjęcie 4). Platforma zawiera również dwa przyciski wykorzystane przy konstrukcji prototypu projektowanego syntezy. Oba przyciski podłączone są do wejścia analogowego procesora oraz podciągnięte do napięcia zasilania różnej wartości rezystancjami i na podstawie odczytu przetwornika ADC można ustalić które z podłączonych przycisków są aktualnie wciśnięte. Dużym udogodnieniem w testowaniu jest obecność wyjścia audio typu „mini-jack” (zdjęcie 4). Ze względu na brak wewnętrznej pamięci flash wewnątrz procesora sygnałowego, na płytce ewaluacyjnej znajduje się układ pamięci flash o pojemności 8 megabajtów dostępnej poprzez interfejs SPI (zdjęcie 4). Niewykorzystany w projekcie wyświetlacz „OLED” może stanowić o możliwości dalszego rozwoju projektu przykładowo poprzez wyświetlanie informacji o syntezie, bądź to do wyświetlania wyników szybkiej transformaty Fouriera z generowanego dźwięku w postaci pasków co wydaje się szczególnie naturalne ze względu na obecność koprocatora FFT w układzie procesora sygnałowego [5].



Zdjęcie 4 Płytki ewaluacyjnej ezdsp5535 oraz schemat jej wewnętrznych połączeń

4.5. Oprogramowanie procesora STM32L053R8

Oprogramowanie dla mikrokontrolera STM32L053R8 zostało napisane w języku C. Korzystano z interfejsu programistycznego CMSIS (Cortex Microcontroller Software Interface Standard) udostępnionego przez firmy ARM i STMicroelectronics oraz konfigurującego dostęp do ustawianych rejestrów układów peryferyjnych podczas ich inicjalizacji. Do pisania kodu, kompilacji oraz debugowania programu korzystano ze zintegrowanego środowiska programistycznego „Keil” równocześnie używając wbudowanego emulatora na płytce ewaluacyjnej ST-Link.

4.5.1. Obsługa klawiatury

Oprogramowanie realizuje algorytm opisany w punkcie 3.1.1.

Do obsługi cyfrowych wejść i wyjść wykorzystano port B mikrokontrolera. Inicjalizacja tego układu peryferyjnego polegała kolejno na:

- aktywowaniu sygnału zegara dla portu w module zarządzania zegarami układu mikrokontrolera - RCC (*reset and clock control*),
- ustawieniu używanych pinów w trybie wyjść/wejść cyfrowych,
- ustaleniu kierunku pracy pinów cyfrowych,
- określeniu szybkości zmiany wyjść cyfrowych,
- podłączeniu rezystorów podciągających do dodatniego napięcia zasilania dla wejść cyfrowych.

Piny 3 do 8 portu B mikrokontrolera działają jako wejścia cyfrowe, zaś piny od 10 do 14 działają jako cyfrowe wyjścia.

Do przechowywania informacji o wciśniętych klawiszach wykorzystano zmienną 32-bitową *keys* w której 30 bitów odzwierciedla stan każdego z 30 klawiszy. Jeśli odpowiadający klawisz nie jest wciśnięty bit ma wartość „1” i odwrotnie wciśnięciu klawisza odpowiada wartość „0”. Dlatego zmienną inicjalizuje się wartością 0x3FFFFFF mówiącą że żaden klawisz nie jest wciśnięty.

Kolejne odczyty aktualnego stanu zestawu przycisków odbywa się co 10 ms. poprzez obsługę przerwania generowanego przez układ peryferyjny SysTick Timer. Poniżej pokazano kluczowy fragment kodu przerwania:

```
1 void SysTick_Handler (void) {  
2     keys &= ~(0x3F << ((NUM_OF_INPUT_PINS) * octaveIndex));  
3     keys |= (((GPIOB -> IDR) & INPUT_MASK) >> FIRST_INPUT_PIN) << (NUM_OF_INPUT_PINS *  
4     GPIOB -> ODR = OUTPUT_MASK & ~(1UL << (FIRST_OUTPUT_PIN + (octaveIndex + 1) % NUM_
```

Listing 1 Procedura obsługi przerwania dla sysTickTimera

Linia oznaczona jako (2) na listingu 1 przygotowuje zmienną do odczytu wpisując wartości „0” na bitach reprezentujących stan klawiszy, których odczyt nastąpi w tym przerwaniu. Zmienna „octaveIndex” wskazuje, który zestaw jest odczytywany, makro „NUM_OF_INPUT_PINS” reprezentuje liczbę wejść cyfrowych. Linia oznaczona jako (3) na listingu 1 wpisuje wartości „1” na bitach reprezentujących niewciśnięty klawisz. Odczyt stanu portu następuje poprzez odczyt wartości rejestru zdefiniowanego jako „IDR” z odpowiednich bitów reprezentowanych przez makro „INPUT_MASK”. Odczytana wartość umieszczana jest na odpowiednim miejscu poprzez dwa przesunięcia bitowe reprezentowane operatorami „>>” i „<<”. Linia oznaczona jako (4) na listingu 1 aktywuje kolejny zestaw przycisków poprzez odpowiednie wysterowanie wyjść cyfrowych zgodnie z algorytmem opisanym w punkcie 3.1.1.. Wpisanie w rejestr „GPIOB -> ODR” wartości skutkuje zmianą wysterowania wyjść cyfrowych. Aktywowanie kolejnego zestawu przycisków realizowane jest poprzez ustawienia tylko jednego portu w stan „0” – w tej linii odbywa się to poprzez przesunięcie logiczne wartości „1UL” do bitu reprezentującego aktywowane wyjście cyfrowe i negacji bitowej tej wartości.

Po wykonaniu tych operacji, program obsługi przerwania aktualizuje informacje o aktualnie sprawdzanym zestawie przycisków.

Wysterowanie wyjść dla kolejnego odczytu odbywa się od razu po wcześniejszym odczycie tak aby odczyt następował dopiero po ustabilizowaniu się wyjść cyfrowych.

4.5.2. Obsługa interfejsu nastaw parametrów syntezy

Odczytywanie wartości nastaw potencjometrów realizowane jest poprzez użycie wbudowanego w układ mikrokontrolera - przetwornika analogowo-cyfrowego. Zgodnie z projektem (podrozdział 3.1.2) wykorzystano sześć kanałów przetwornika połączonych z sześcioma wyjściami analogowymi. Konfiguracja układu przetwornika polegała kolejno na:

- aktywowaniu sygnału zegara dla przetornika, w module zarządzania zegarami układu mikrokontrolera - RCC (*reset and clock control*),
- ustawieniu używanych pinów (PA2, PA3, PA6, PA7, PB0, PB1) jako wyjść analogowych,
- ustawieniu rozdzielczości przetwornika na 8-bitową,
- połączeniu wewnętrznym kanałów przetwornika z pinami,
- uaktywnienie używanych kanałów, tak aby kolejno wykonywała się na nich konwersja,
- pozwolenie na obsługę żądania obsługi przerwania generowanego po każdej skończonej konwersji.

Układ peryferyjny przetwornika ADC wykonuje sekwencję sześciu pomiarów po jednym na każdy kanał. Po każdej skończonej konwersji wykonywana jest procedura obsługi przerwania odczytująca wynik (listing 2). Z rejestru „DR” (data register) układu wczytywane jest 8 bitów do odpowiedniego elementu tablicy o nazwie *ADCData*. Informację o tym z jakiego numeru potencjometru odczytywana jest wartość napięcia przechowuje zmienna *adcIndex*. Dalsze przetwarzanie oraz interpretacja wykonywane są w pętli głównej programu. Po zakończeniu procedury obsługi przerwania, jej kod, poprzez funkcję *startADC*, informuje przetwornik o żądaniu startu kolejnej konwersji kolejnego kanału.

```

1 void ADC1_COMP_IRQHandler (void) {
2     if(ADC1 -> ISR & ADC_ISR_EOC){
3         ADCdata[adcIndex] = ADC1 -> DR;
4         ADC1 -> ISR |= ADC_ISR_EOSEQ;
5     }
6     adcIndex = (adcIndex+1)%6;
7     startADC();
8 }

```

Listing 2 Procedura obsługi przerwania dla przetwornika ADC

4.5.3. Obsługa komunikacji szeregowej

Aby możliwe było przekazanie wprowadzonych przez użytkownika parametrów syntezy do bloku przetwarzania DSP konieczna jest konfiguracja układu peryferyjnego UART dla zainicjalizowania transmisji. Zgodnie z założeniami projektowymi opisanymi w podrozdziale 3.1.5. układ ustawiony jest tak aby:

- szybkość transmisji wynosiła 19200 baudów,
- wysyłane było 8-bitowe słowo,
- transmisja nie miała bitu parzystości,
- liczba bitów wynosiła 1,
- umożliwić obsługę transmisji przerwaniami.

Konfiguracja układu peryferyjnego polegała na ustawieniu odpowiednich wartości rejestrów kontrolnych.

Transmisja danych polega na wysyłaniu kolejnych bitów zapisanych w tablicy 8-bitowych zmiennych. Procedura obsługi przerwania (listing 3) kolejno wpisuje wartości z

przygotowanej w innym miejscu programu tablicy *dataToSend*. Przesył znaku zainicjowany jest poprzez wpis 8 bitów do rejestru o nazwie *TDR*.

```
1 void USART1_IRQHandler(void)
2 {
3     USART1->ICR |= USART_ICR_TCCF;
4
5     if(usartIndex >= NUM_OF_BYTES_TO_SEND)
6         uartIndex = 0;
7     else
8         USART1->TDR = dataToSend[usartIndex++];
9 }
```

Listing 3 Procedura obsługi przerwania dla układu UART

Przygotowanie danych do wysłania odbywa się w pętli głównej programu. Zmienna globalna przechowująca informacje o wciśniętych klawiszach (listing 1) o nazwie *keys* użyta jest do wypełnienia czterech pierwszych bajtów tablicy *dataToSend*. Sześć kolejnych bajtów przepisywane jest bezpośrednio z tablicy *ADCDData* (listing 2).

4.6. Oprogramowanie procesora TMS320C5535

Oprogramowanie dla procesora sygnałowego TMS320C5535 pisane było w języku C. Kod programu był pisany i kompilowany w zintegrowanym środowisku programistycznym CCS (*code composer studio*) w wersji 5.5. Wykorzystany został interfejs programistyczny CSL (*chip support library*) dostarczony przez producenta, firmę Texas Instruments. Biblioteka ta umożliwia na dwojaki sposób pisanie oprogramowania: dostarcza pliki nagłówkowe opisujące wszystkie rejestry procesora co daje możliwość niskopoziomowego programowania. Z drugiej strony biblioteka CSL dostarcza interfejsu typu HAL (*hardware layer abstraction*) oddzielający programistę od warstwy sprzętowej. W tym przypadku nie jest wymagane tak szczegółowe poznanie peryferii procesora. W oprogramowaniu pisanym na potrzeby konstrukcji opisywanego urządzenia korzystano z obu sposobów. Do wgrywania kodu oraz debugowania użyty został wbudowany w platformę ezdsp5535 emulator XDS100v2.

4.6.1. Konfiguracja kodeka audio AIC3204

Do rozpoczęcia wysyłania próbek wygenerowanego sygnału konieczne jest skonfigurowanie kodeka audio. Ponieważ szyna kontrolna połączenia procesora sygnałowego TMS320C5535 a kodekiem TLV320AIC3204 projektowana jest jako magistrala interfejsu I2C, dlatego też pierwszym krokiem do konfiguracji kodeka jest obsługa modułu interfejsu I2C znajdującego się wewnątrz procesora sygnałowego w postaci układu peryferyjnego. Układ obsługi interfejsu skonfigurowano następująco:

- moduł ustawiony jest w trybie „master”,
- taktowany zegarem systemowym 100 MHz zredukowanym o czynnik 20, którego wartość wpisana jest do odpowiednich rejestrów, czyli 5 MHz, natomiast zegar SCL wyprowadzony dla taktowania magistrali zredukowany jest dalej o czynnik 40, tzn. zegar generuje przebieg o częstotliwości 125 kHz,

- zablokowano generowanie przerwań – obsługując komunikację metodą „pollingu”.

```

1      /* Konfiguracja kodeku audio TLV320AIC3204 */
2      AIC3204_registerSet( 0, 0x00 ); // wybór strony pamięci 0x00 | "Page Select Register"
3      AIC3204_registerSet( 1, 0x01 ); // reset kodeka | "Software Reset Register"
4      EZDSP5535_waitusec ( 1000 ); // czekaj minimum 1ms po resecie
5      AIC3204_registerSet( 0, 0x01 ); // wybór strony pamięci 0x01 | "Page Select Register"
6      AIC3204_registerSet( 1, 0x08 ); // przerwij wew. połączenie pinów zasilania analogowego
AVdd od cyfrowego DVdd | "Power Configuration Register"
7      AIC3204_registerSet( 2, 0x01 ); // włącz bloki analogowe kodeka, zasil z LDO = 1,75V (low
drop-out) | "LDO Control Register"
8      AIC3204_registerSet( 123, 0x05 ); // wymuś włączenie nap. referencyjnego w czasie 40 ms |
"Reference Power-up Configuration Register"
9      waitusec ( 50000 ); // poczekaj więcej niż 40ms
10     AIC3204_registerSet( 0, 0x00 ); // wybierz stronę pamięci 0x00 | "Page Select Register"
11
12     /* konfiguracja pętli PLL, zegarów oraz włączenie jej zasilania */
13     AIC3204_registerSet( 27, 0x0d ); // ustaw interfejs audio jako I2S, ustal długość słowa na
16 bitów,
// wyprowadz syg. zegarowe BCLK (bit clock line) i WCLK
(word clock line) na zew. piny (dla konfiguracji I2S - kodek pracuje jako "master") | "Audio
Interface Setting Register 1"
14     AIC3204_registerSet( 28, 0x00 ); // ustaw opóźnienie danych na 0 cykli BCLK | "Audio
Interface Setting Register 2"
15     AIC3204_registerSet( 4, 0x03 ); // ustaw taktowanie PLL z MCLK = 12Mhz, sygnał
CODEC_CLKIN taktowany z wyjścia PLL CLK | "Clock Setting Register 2"
16     AIC3204_registerSet( 6, 0x07 ); // ustaw współczynnik PLL: J=7 | "Clock Setting Register
3"
17     AIC3204_registerSet( 7, 0x06 ); // ustaw współczynnik PLL: bajt wysoki (D=1680) | "Clock
Setting Register 4"
18     AIC3204_registerSet( 8, 0x90 ); // ustaw współczynnik PLL: bajt niski (D=1680) | "Clock
Setting Register 5"
19     AIC3204_registerSet( 30, 0x88 ); // zasil dzielnik BCLK i ustaw N = 8 | "Clock Setting
Register 12, BCLK N Divider"
20     // BCLK=DAC_CLK/N =(12 288 000/8) = 1.536MHz = 32*fs = 32*48kHz
21     AIC3204_registerSet( 5, 0x91 ); // włączenie zasilania PLL, ustaw współczynniki PLL na
P=1 and R=1 | "Clock Setting Register 2, PLL P and R Values"
22     waitusec ( 10000 ); // czekaj aż układ PLL włączy się
23     AIC3204_registerSet( 13, 0x00 ); // ustaw wysoki bajt dzielnika DOSR (DOSR ustaw na 128,
ustaw 48KHz taktowanie DAC) | "DAC OSR Setting Register 1, MSB Value"
24     AIC3204_registerSet( 14, 0x80 ); // ustaw niski bajt dzielnika DOSR, (DOSR ustaw na 128,
ustaw 48KHz taktowanie DAC) | "DAC OSR Setting Register 2, LSB Value"
25     AIC3204_registerSet( 20, 0x80 ); // tak samo jak w przypadku DAC, AOSR = 128 | "ADC
Oversampling (AOSR) Register"
26     AIC3204_registerSet( 11, 0x82 ); // zasil dzielnik NDAC i ustaw NDAC na 2
27     AIC3204_registerSet( 12, 0x87 ); // zasil dzielnik MDAC i ustaw MDAC na 7
28     AIC3204_registerSet( 18, 0x87 ); // zasil dzielnik NADC i ustaw NADC na 7
29     AIC3204_registerSet( 19, 0x82 ); // zasil dzielnik MADC i ustaw MADC na 2
30
31     /* konfiguracja DAC i włączenie zasilania */
32     AIC3204_registerSet( 0, 0x01 ); // wybór strony pamięci 0x01 | "Page Select Register"
33     AIC3204_registerSet( 12, 0x08 ); // sygnał LDAC AFIR przekierowany na HPL (Left Headphone
Driver) | "HPL Routing Selection Register"
34     AIC3204_registerSet( 13, 0x08 ); // sygnał RDAC AFIR przekierowany na HPR (Right Headphone
Driver) | "HPR Routing Selection Register"
35     AIC3204_registerSet( 0, 0x00 ); // wybór strony pamięci 0x00 | "Page Select Register"
36     AIC3204_registerSet( 64, 0x02 ); // wyrównaj głośności kanału lewego z prawym (sterowanie
z kanału lewego) | "DAC Channel Setup Register 2"
37     AIC3204_registerSet( 65, 0x00 ); // ustaw wzmocnienie na 0dB | "Left DAC Channel Digital
Volume Control Register"
38     AIC3204_registerSet( 63, 0xd4 ); // zasil oba kanały DAC and set channel | ": DAC Channel
Setup Register 1"
39     AIC3204_registerSet( 0, 0x01 ); // wybór strony pamięci 0x01 | "Page Select Register"
40     AIC3204_registerSet( 16, 0x00 ); // uaktywnij HPL , 0dB wzmocnienia
41     AIC3204_registerSet( 17, 0x00 ); // uaktywnij HPR , 0dB wzmocnienia
42     AIC3204_registerSet( 9, 0x30 ); // Power up HPL,HPR
43     EZDSP5535_waitusec ( 100 ); // czekaj 100 us

```

Do konfiguracji układu peryferyjnego użyte zostały dedykowane funkcje z biblioteki CSL.

Zadanie konfiguracji kodeka audio polegało na modyfikacji wartości 8-bitowych rejestrów kontrolnych znajdujących się w układzie kodeka. Oprogramowanie realizuje zapis danych do konfigurowanych rejestrów wewnętrznych poprzez wysłanie kolejno adresu modyfikowanego rejestru, a następnie jego wartości poprzez interfejs I2C. Działanie to realizowane jest poprzez funkcję o nazwie *AIC3204_registerSet* (listing 4). W części konfiguracyjnej kodu napisanego dla procesora sygnałowego TMS320C5535 wykorzystywana jest ona wielokrotnie, kolejno inicjalizując rejestry wewnętrzne układu kodeka. Do wykonywania swojego zadania wykorzystuje ona funkcję biblioteczną CSL *I2C_write* dysponując przy tym informacją o adresie standardu I2C typu *slave* przypisanym układowi komunikacji kodeka.

Pierwsza część kodu konfiguracji (od linii nr. 2 w listingu 4) ustawia podstawowe parametry niezbędne do prawidłowego działania kodeka oraz przygotowuje go do dalszej konfiguracji. Zrealizowane są następujące zadania: uruchamiane zostaje zasilanie dla bloków analogowych układu kodeka oraz aktywowane jest napięcie referencyjne dla układów przetwornika cyfrowo-analogowego (DAC) oraz przetwornika analogowo-cyfrowego (ADC).

Głównym zadaniem drugiego fragmentu kodu odpowiedzialnego za obsługę kodeka (od linii nr. 13 w listingu 4) jest konfiguracja układu pętli synchronizacji fazowej PLL, konfiguracja sygnałów zegarowych wewnątrz kodeka oraz konfiguracja kontrolera interfejsu I2S jako interfejsu audio dla wymiany danych z procesorem sygnałowym. W szczególności ustala się w tym miejscu długość słowa na 16 bitów oraz częstotliwość próbkowania przetwornika DAC na 48 kHz.

Do pinów wejściowych MCLK (*master clock*) układu kodeka doprowadzony jest oscylator kwarcowy o częstotliwości pracy 12 MHz. Wybrany jest on jako wejściowy sygnał (PLL_{CLKIN}) układu pętli PLL. Ponieważ znana jest zależność charakteryzująca zmianę częstotliwości wyjścia pętli PLL w stosunku do jej wejścia [11], wprowadzono odpowiednie parametry: $J = 7$, $D = 1680$, $P = 1$, $R = 1$ (listing 4, linie 16 – 21) do rejestrów wewnętrznych układu kodeka. Zależność ta, z uwzględnieniem wprowadzonych wyżej oznaczeń, wyraża się następująco:

$$PLL_{CLK} = \frac{PLL_{CLKIN} * R * J.D}{P} \quad (8)$$

gdzie współczynniki 'R', 'J', 'D' oraz 'P' są dodatnimi liczbami całkowitymi, zaś w zapisie 'J.D' znak '.' oznacza separator dziesiętny. Ze wzoru wynika, że generowany jest sygnał o częstotliwości 86,016 MHz. Wyjście to poprzez dzielnik MDAC dostarcza sygnału taktowania układowi DAC. Wartość MDAC ustawiona jest na 7, tzn. sygnał DAC_{CLK} ma częstotliwość 12,288 MHz. Sygnał BCLK, wyprowadzony jako zegar główny interfejsu I2S tworzony jest poprzez podanie sygnału DAC_{CLK} na dzielnik N, który w tym przypadku dzieli przez wartość 8. Z tego wynika, że $BCLK = 1,536$ MHz, co jest równoważne temu, że $BCLK = 32 * 48\text{kHz}$ co odpowiada 32 bitom w dwóch 16-bitowych słowach reprezentujących próbki kanału lewego i prawego. Jednocześnie sygnał decydujący o tempie próbkowania wyrażony jest jako $DAC_{CLK} / NDAC$, gdzie w rozpatrywanym przypadku wartość NDAC ustawiona jest na 2, co daje w wyniku 6,144 MHz. Sygnał wyjściowy jest zatem

nadpróbkowany o czynnik 128, który to musi być wpisany jako informacja do rejestru „DAC OSR Setting Register”.

Trzecia część kodu konfiguracji (od linii nr. 31 w listingu 4) odpowiedzialna jest zaś za wyprowadzenie sygnału z przetwornika cyfrowo-analogowego w taki sposób na wyjścia słuchawkowe HPL i HPR oraz konfigurację części analogowej układu odpowiedzialnej za elektryczne wzmocnienie wyjściowego sygnału elektrycznego.

4.6.2. Generacja obwiedni ADSR

Oprogramowanie napisane dla procesora sygnałowego TMS320C5535 realizuje generację obwiedni dla kreowanego przez syntezytor - dźwięku, poprzez implementację algorytmu opisanego w podrozdziale 3.1.4.2.

Parametry charakteryzujące kształt obwiedni otrzymywane są poprzez łącze szeregowe od kontrolera interfejsu z mikroprocesorem STM32L053R8 i przechowywane są w czterech 16-bitowych bezznakowych zmiennych całkowitych o nazwach odniesionych odpowiednio do określonego przez nie parametru:

- *attackParam* – określający czas w milisekundach trwania fazy „attack”,
- *decayParam* – określający czas w milisekundach trwania fazy „decay”,
- *sustainParam* – określający poziom z zakresu od 0x0000 do 0x7FFFF,
- *releaseParam* – określający czas w milisekundach trwania fazy „release”.

Implementacja rozważanego (rys. 7) algorytmu generacji obwiedni, który opiera się na zmianie parametru **ob** w czasie, musi realizować tę funkcjonalność w odniesieniu do ustalonego wzorca czasu. Sygnał, który to umożliwia generowany jest przez układ peryferyjny procesora sygnałowego jakim jest układ 32-bitowego timera ogólnego przeznaczenia. Skonfigurowano ten układ w taki sposób aby zgłaszał żądanie przerwania co 1 ms. Wybór takiego odcinka czasu podyktowany był tym, aby słuchający generowanego dźwięku nie był w stanie odróżnić kwantowej zmiany amplitudy, a miał wrażenie płynności tej zmiany. Taki okres czasu został wyznaczony eksperymentalnie.

Parametr zmiany amplitudy **ob** został zaimplementowany jako całkowita, 16-bitowa zmienna bez znaku o nazwie *envelopeScaler* i odzwierciedla poziom głośności w zakresie od 0x0000 do 0x7FFF co odpowiada wartościom z zakresu [0,1) w arytmetyce i1q15 w odpowiedni sposób skalując amplitudę generowanego sygnału.

Implementacja w języku C algorytmu generacji obwiedni na procesorze sygnałowym zakłada utworzenie instrukcji wyboru *switch case* modyfikującego zmienną *envelopeScaler* w zależności od fazy w której znajduje się generowany dźwięk. Faza w której aktualnie znajduje się dźwięk reprezentowana jest poprzez zmienną *adsrIndex*, przyjmującą wartości typu enum: {*attack*, *decay*, *sustain*, *release*}. Odpowiednia modyfikacja zmiennej reprezentującej współczynnik **ob** realizowana jest w oparciu o opis algorytmu (rys. 6). Wartość o którą modyfikowana jest zmienna *envelopeScaler* obliczana jest poprzez podzielenie pełnego zakresu wartości amplitud, specyficznego dla danej fazy, przez wartość określającą ilość milisekund przypadającą na daną fazę. Zrealizowane jest to w liniach: 15, 22 oraz 31 listingu nr. 5. Przejście do fazy kolejnej uwarunkowane jest przekroczeniem pewnego

poziomu amplitudy. Sprawdzanie tego przekroczenia realizowane jest poprzez instrukcje warunkowe w liniach: 16, 23 oraz 32 listingu nr. 5.

```
1 interrupt void gptIsr(void){
2     if(keyFlag)
3     {
4         if(!previousKeyFlagState)
5             adsrIndex = attack;
6     }
7     else
8     {
9         adsrIndex = release;
10    }
11
12    switch(adsrIndex)
13    {
14        case attack:
15            envelopeScaller += MAX_SCALLER/attackParam;
16            if(envelopeScaller > MAX_SCALLER){
17                envelopeScaller = MAX_SCALLER;
18                adsrIndex++;
19            }
20            break;
21        case decay:
22            envelopeScaller -= (MAX_SCALLER - sustainParam)/decayParam;
23            if(envelopeScaller < sustainParam){
24                envelopeScaller = sustainParam;
25                adsrIndex++;
26            }
27            break;
28        case sustain:
29            break;
30        case release:
31            envelopeScaller -= sustainParam/releaseParam;
32            if(envelopeScaller > MAX_SCALLER){
33                envelopeScaller = MIN_SCALLER;
34            }
35    }
36
37    previousKeyFlagState = keyFlag;
38    IRQ_clear(TINT_EVENT);
39    CSL_SYSCTRL_REGS->TIAFR = 0x01;
40}
```

Listing 5 Procedura obsługi przerwania dla timera ogólnego przeznaczenia

W procedurze obsługi przerwania użytego timera (listing 5) dodatkowo sprawdza się czy klawisz, dla którego generowana jest obwiednia został wciśnięty. Determinowane jest to poprzez sprawdzenie zmiennych odzwierciedlających stan danego klawisza podczas wcześniejszego przerwania. Jeżeli taka sytuacja miała miejsce, faza w której znajduje się dźwięk, zmienia się na fazę „attack” tzn. amplituda dźwięku, o ile to możliwe, zaczyna wzrastać tak jak ma to miejsce przy pierwszym wciśnięciu – nawet jeśli zmienna *envelopeScaller* miała wartość różną od zera. Pod koniec procedury zmienna *previousKeyFlagState* aktualizowana jest do bieżącego stanu wciśniętego klawisza. Zmienna

keyFlag modyfikowana jest w innym miejscu programu, który odpowiedzialny jest za odbiór danych z kontrolera interfejsu.

Zaprezentowana procedura (listing 5) powtarzana jest dla każdego wciśniętego klawisza, aż do ośmiu klawiszy wciśniętych jednocześnie.

4.6.3. Obsługa przycisków funkcyjnych za pomocą przetwornika ADC SAR

Aby umożliwić większą elastyczność w zakresie możliwości uzyskiwania żądanej przez użytkownika charakterystyki syntezy dźwięku, umożliwiono mu zmianę ilości składowych harmoniczných w generowanym sygnale. Jeden z przycisków (SW2 na zdjęciu 4) po przyciśnięciu przez użytkownika, zwiększa o jeden liczbę składowych harmoniczných zarówno dla harmoniczných parzystych jak i nieparzystych.

Drugi z przycisków (SW1 na zdjęciu 4), do celów testowania, generuje dźwięk o częstotliwości 1kHz przy tych samych parametrach syntezy co dźwięk generowany przez klawisze.

Wciśnięcie obu przycisków jednocześnie skutkuje zakończeniem działania programu na procesorze DSP.

Stwierdzenie wciśnięcia przycisków odbywa się poprzez odczyt napięcia na wejściu analogowym 10-bitowego przetwornika analogowo-cyfrowego typu SAR (z kompensacją równoległą). Odbywa się to poprzez zgłoszenie żądania obsługi przerwania przez układ przetwornika. Odczyt następuje poprzez procedurę obsługi przerwania (listing 6). Do zmiennej *readData* poprzez funkcję biblioteczną CSL, wczytywana jest wartość zmierzona przez przetwornik.

```
1 interrupt void sarISR(void){
2     *SARstatus = SAR_readData(hSar, &readData);
3 }
```

Listing 6 Procedura obsługi przerwania przetwornika ADC typu SAR

W zależności od odczytanej wartości wyniku przetwarzania przetwornika zostaje określone, która z czterech możliwych kombinacji wciśnięcia dwóch funkcyjnych przycisków jest aktualnie aktywna. Odpowiada za to fragment pętli głównej programu (listing 6). Odczytaną wartość przetwarzania, zachowaną w zmiennej *readData* porównuje się z wartościami zdefiniowanymi w makrach, które odpowiadają napięciom panującym na wejściu analogowym przetwornika odpowiednio:

- SW12 – teoretyczna wartość odczytu przetwornika, gdy oba przyciski są wciśnięte,
- SW1 – teoretyczna wartość odczytu przetwornika, gdy wyłącznie przycisk SW1 jest wciśnięty,
- SW2 – teoretyczna wartość odczytu przetwornika, gdy wyłącznie przycisk SW2 jest wciśnięty.

Przyjmuje się, że odczyt może różnić się od tej teoretycznej wartości o 12.5% co uwzględnia kod (listing 7). Użyta w realizacji tej funkcjonalności funkcja *sw2TransitionTest*

sprawdza, czy zaistniało przejście ze stanu braku wciśnięcia do stanu z wciśnięciem przycisku, co rozpoczyna generację dźwięku o częstotliwości 1kHz.

```
1  if(readData > 0.875*SW12 && readData < 1.125*SW12)
2  {
3      keyFlag = 1;
4      sw2TransitionTest();
5      sw2State = 1;
6  }
7  else if(readData > 0.875*SW1 && readData < 1.125*SW1)
8  {
9      keyFlag = 1;
10 }
11 else
12 {
13     keyFlag = 0;
14     if(readData > 0.875*SW2 && readData < 1.125*SW2)
15     {
16         sw2TransitionTest();
17         sw2State = 1;
18     }
19     else
20     {
21         sw2State = 0;
22     }
23 }
```

Listing 7 Fragment pętli głównej programu odpowiadający za interpretację wciśniętych przycisków

Przetwornik analogowo-cyfrowy skonfigurowano tak aby:

- za wartość napięcia odniesienia było przyjmowane napięcie wejściowe pochodzące z układu stabilizatora napięcia LDO (*low drop-out*) znajdującego się na płycie ewaluacyjnej ezdsp5535, ustawionego na poziomie 3,3 V,
- taktowanie przetwornika było mniejsze od 2 MHz która jest wartością maksymalną. Dlatego doprowadzony zegar systemowy o częstotliwości 100 MHz podzielony został o współczynnik 100, tzn. do przetwornika doprowadzony został sygnał o częstotliwości 1MHz, a ponieważ próbkowanie trwa 32 cykle tego zegara, próbkowanie przeprowadzone jest z częstotliwością 31,25 kHz,
- wysyłał żądanie obsługi przerwania po zakończeniu przetwarzania.

Wspomniana wyżej konfiguracja została przeprowadzona przy użyciu funkcji bibliotecznych CSL.

4.6.4. Implementacja algorytmu syntezy addytywnej

Aby umożliwić przesył wygenerowanych w bloku syntezy wartości próbek do przetwornika DAC znajdującego się w układzie kodeka audio, konieczne jest skonfigurowanie układu peryferyjnego procesora sygnałowego - kontrolera interfejsu I2S. Dla komunikacji z układem kodeka TLV320AIC3204, tak jak przedstawiono w podrozdziale 4.6.1., ustawiono ten układ w trybie „master”, a więc kontroler interfejsu I2S w procesorze sygnałowym musi pracować w trybie „slave”. Sygnały zegarowe BCLK (*bit clock*) oraz (word clock) dostarczone

są od strony kodeka. Oprogramowanie inicjalizacji układu kontrolera charakteryzuje się m. in. :

- transmisją dźwięku stereofonicznego – na każdy okres częstotliwości próbkowania wysyłana do kodeka jest próbka dla kanału lewego oraz próbka dla kanału prawego,
- obsługą kontrolera poprzez przerwania – układ peryferyjny generuje żądanie obsługi przerwania po zmianie stanu na linii sygnału zegara WCLK.

W trakcie trwania procedury obsługi przerwania (listing 8) generowane są wynikowe próbki syntezy dźwięku zgodnie z założonym algorytmem (równanie 6).

Główna pętla w procedurze obsługi przerwania (linia 11 listing 8) odpowiada za generację każdego z ośmiu dźwięków przewidzianych w projekcie polifonii syntezy muzycznego. Każdy przebieg tej pętli generuje oddzielnie grupę składowych harmonicznym nieparzystych (linia 13 listing 8) oraz grupę składowych harmonicznym (linia 19 listing 8). Generowanie przebiegu tonu sinusoidalnego odbywa się poprzez odczyt zapisanych próbek we wcześniej zdefiniowanej tablicy częstotliwości bazowej, której adres początkowy (inicjalizowany w innej części programu) wskazuje odpowiednia zmienna z tablicy wskaźników o nazwie *sineTable*. Za każdym przebiegiem pętli głównej jeden z ośmiu wskaźników z tej tablicy pozwala na dostęp do odpowiedniej tablicy próbek tonu podstawowego. Zmienne *pot5* oraz *pot6* odpowiedzialne są za modyfikację amplitudy grup składowych harmonicznym nieparzystych (linia 17, listing 8) oraz amplitudy grup składowych harmonicznym parzystych (linia 18, listing 8). W trakcie ich generacji uwzględniony zostaje zakres dynamiczny cyfrowego wejścia przetwornika DAC – wygenerowanemu sygnałowi zostaje przydzielony pewien zakres głośności poprzez dzielenie wartości uzyskanej próbki przez odpowiedni współczynnik związany z ilością generowanych dźwięków w polifonii oraz ilością składowych harmonicznym zawartych w pojedynczym dźwięku. Po zsumowaniu wartości powstałych po generacji składowych harmonicznym i nieharmonicznym do zmiennej *sample*, kod implementuje funkcjonalność jaką jest generowanie obwiedni poprzez przeprowadzenie odpowiedniego mnożenia w arytmetyce i1q15. Wskaźnik *envelopeScaller* (którego modyfikację opisuje rozdział 4.6.2.) wskazuje na wartość odpowiadającą parametrowi **ob** z równania 6, realizując funkcję współczynnika skalowania w tym mnożeniu. Po każdym przebiegu pętli, wskaźnik *envelopeScaller* jest inkrementowany, aby wskazywał na odpowiedni współczynnik reprezentujący właściwy parametr **ob**. Tego samego typu mnożenia przeprowadzane są przy modyfikacji amplitudy grup harmonicznym reprezentowanych przez zmienne *pot5* oraz *pot6*, które to wprowadzane są przez użytkownika jako parametry syntezy poprzez blok interfejsu użytkownika. Wysłanie 16-bitowej wartości próbki następuje poprzez wpisanie jej do rejestru o nazwie I2STXLT1 (linia 33, listing 8). Po wpisaniu wartości do tego rejestru następuje automatyczne przeniesienie jej do rejestru szeregowego wysyłającego kolejno bity poprzez interfejs I2S. Dlatego też możliwe jest ponowne wpisanie wartości do rejestru I2STXLT1, co czynione jest z wartością próbki kanału prawego, tuż po wpisaniu tam wartości próbki kanału lewego syntezy dźwięków. Ostatnia część kodu procedury obsługi przerwania odpowiada za wyliczenie indeksu tablicy dla wygenerowania kolejnej próbki każdego z generowanych dźwięków. Każda harmoniczna każdego z ośmiu możliwych, generowanych w tym samym czasie

dźwięków, posiada przypisaną zmienną zawierającą aktualny indeks w tablicy reprezentującej sygnał podstawowy generowanego dźwięku.

```
1  interrupt void i2s_txIsr(void)
2  {
3      Int16 sample = 0;
4      Int16 poliSample = 0;
5      Int16 evenHarmonicSample = 0;
6      Int16 oddHarmonicSample = 0;
7      Int16 i = 0;
8      Int16 poliIndex = 0;
9      (void*)regs->I2SINTFL;
10
11     for(poliIndex = 0; poliIndex < POLI_NUM; poliIndex++)
12     {
13         for(i = 0; i < numberOfHarmonics; i=i+2) //even harmonics
14         {
15             evenHarmonicSample += sinetable[poliIndex][sampleIndex[poliIndex][i]]/(8*(i+1));
16         }
17         evenHarmonicSample = (Int16)((Int32)pot5 * ( sample ) >> 15) ;
18
19         for(i = 1; i < numberOfHarmonics; i=i+2) //odd harmonics
20         {
21             oddHarmonicSample += sinetable[poliIndex][sampleIndex[poliIndex][i]]/(8*(i+1));
22         }
23         oddHarmonicSample = (Int16)((Int32)pot6 * ( sample ) >> 15) ;
24
25         sample = oddHarmonicSample + evenHarmonicSample;
26
27         sample = (Int16)((Int32)*envelopeScaller * ( sample ) >> 15) ;
28
29         poliSample+=sample;
30         envelopeScaller++;
31     }
32
33     regs->I2STXLT1 = poliSample;
34     regs->I2STXRT1 = poliSample;
35
36     for(poliIndex = 0; poliIndex < POLI_NUM; poliIndex++)
37     {
38         for(i = 0; i < numberOfHarmonics; i++)
39         {
40             sampleIndex[poliIndex][i]+=2*i+1;
41             sampleIndex[poliIndex][i] = sampleIndex[poliIndex][i] % NUM_OF_SAMPLES_IN_ARRAY;
42         }
43     }
44     envelopeScaller = 0;
45 }
```

Listing 8 Procedura obsługi przerwania dla transmisji interfejsu I2S

4.6.5. Obsługa komunikacji szeregowej

Aby możliwy był odczyt parametrów syntezy wysyłanych poprzez łącze szeregowe przez kontroler interfejsu zawierający procesor STM32L053R8, zgodnie z projektem (podrozdział 3.1.5.) skonfigurowano moduł interfejsu szeregowego UART procesora sygnałowego odpowiednio na:

- szybkość transmisji 19200 baudów, poprzez odpowiedni podział częstotliwości doprowadzonego do układu sygnału zegara głównego 100MHz,
- słowo danych ustawiono na 8-bitowe,
- brak bitu parzystości,
- liczba bitów stopu to 1,
- brak kontroli przepływu,
- umożliwienie generacji żądań obsługi procedury przerwania do obsługi nadchodzących danych.

Konfiguracja układu peryferyjnego polegała na użyciu odpowiednich funkcji biblioteki CSL.

Zgłoszenie żądania obsługi przerwania (listing 9) następuje po tym gdy odebrany bajt danych znajdzie się w rejestrze RBR układu peryferyjnego UART. Procedura obsługi przerwania skojarzona z tym żądaniem sprawdza czy przyjęty bajt jest bajtem inicjującym, jeśli tak to zeruje zmienną *uartIndex* przechowującą informację o numerze bajtu. Jeśli nie, to pod warunkiem, że *uartIndex* znajduje się w spodziewanym zakresie, odczytuje otrzymany bajt na odpowiednim miejscu w 10-bajtowej tablicy *uartData*. W innym miejscu programu – w pętli głównej programu - otrzymane dane są interpretowane jako parametry syntezy.

```

1  interrupt void uart_txIsr(void)
2  {
3      Uint16 bufferUart = 0;
4      bufferUart = hUart -> uartRegs -> RBR;
5      if(uartIndex >= NUM_OF_BYTES_IN_UART_DATA)
6          if(bufferUart == 0xAA)
7              uartIndex = 0;
9      else
10     {
11         uartData[uartIndex] = bufferUart;
12         uartIndex++;
13     }
14     (void*) hUart -> uartRegs -> IIR;
15 }
16

```

Listing 9 Procedura obsługi przerwania dla układu UART

5. Testowanie systemu

Testowanie realizowane jest na trzech poziomach:

- odsłuchu wydawanych przez podłączone słuchawki do kodeka audio dźwięków,
- odczytu oscylogramów przez podłączony oscyloskop do kodeka audio oraz do linii komunikacji szeregowej, badaniu wewnętrznych sygnałów w systemie,
- odczytu wartości zmiennych podczas debugowania programów.

W tym rozdziale skorzystano ze wszystkich trzech możliwości – wykluczając jednak użycie oscyloskopu, w zamian wykorzystano wejście karty dźwiękowej komputera PC oraz programu umożliwiającego nagrywanie – Audacity.

5.1. Stwierdzenie poprawności działania urządzenia

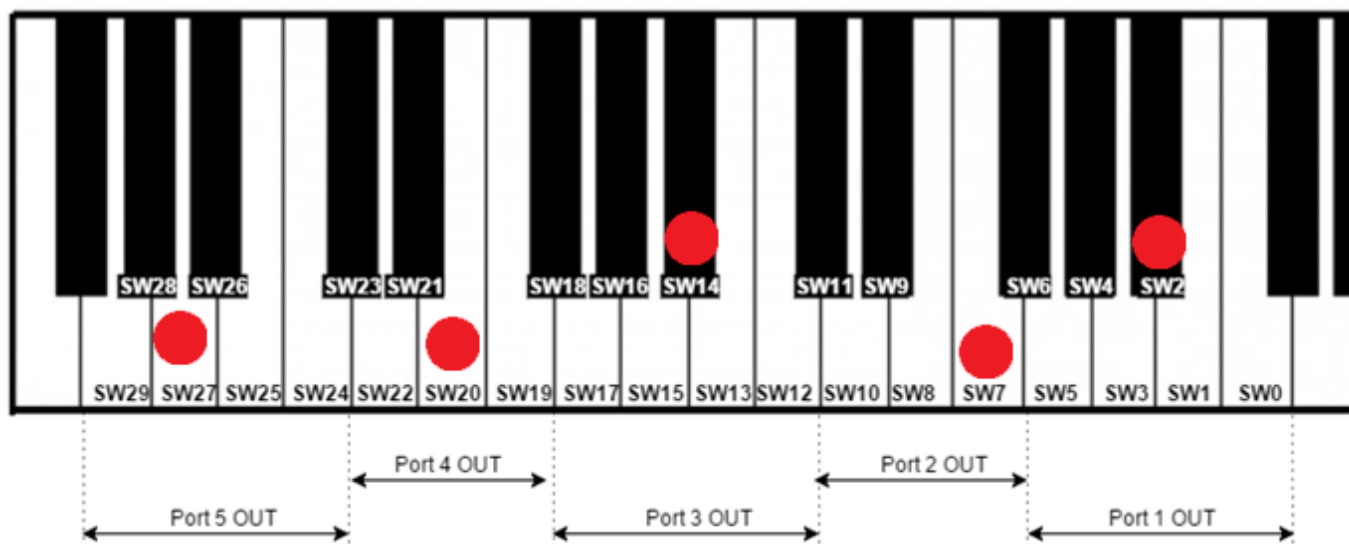
Działanie prototypu syntezy muzycznej opisywanego w tej pracy zostało stwierdzone poprzez podłączenie do wyjścia audio płytki ezdsp5535 słuchawek oraz równoległe połączenie wyjścia do oscyloskopu. Po naciśnięciu klawisza został wygenerowany dźwięk o stałym tonie, co stwierdzono poprzez odsłuch i obserwację przebiegu na ekranie oscyloskopu. Szczegółowe przedstawienie badania poprawności przeprowadzonej przez urządzenie syntezy ze szczególnym uwzględnieniem parametrów wprowadzonych przez użytkownika opisane jest niżej w tym rozdziale.

5.2. Procesor STM32I053R8

W używanym do debugowania środowisku Keil wykorzystano narzędzie jakim jest „watch window” do rejestracji wartości zmiennej. Wykorzystano również możliwości użycia „breakpointów” oraz zatrzymywania przebiegu programu które oferuje mikrokontroler.

5.2.1. Weryfikacja działania klawiatury

Dla potrzeb przedstawienia testowania działania interfejsu klawiatury, arbitralnie wybrano po jednym klawiszu w każdym z 5-ciu zestawów przycisków. Uruchomiono proces debugowania. Wybrano klawisze oznaczone jako: (SW1, SW7, SW14, SW20, SW27) (rysunek 8), wciśnięto je jednocześnie, a następnie zatrzymano wykonywanie programu.



Rysunek 8 Klawisze które zostały wciśnięte jednocześnie na potrzeby testu

Korzystając z narzędzia odczytu wartości zmiennych w trakcie działania programu – poprzez narzędzie „watch window” odczytano wartość 0x37EFBF7B (zdjęcie 5).

Watch 1		
Name	Value	Type
keys	0x37EFBF7B	unsigned int

Zdjęcie 5 Zrzut ekranu z debugera pokazujące informację o wciśniętych klawiszach

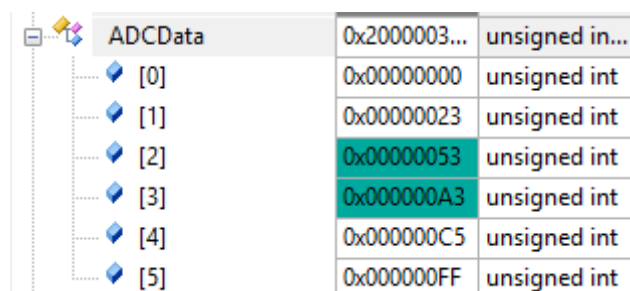
Po rozwinięciu tej wartości do notacji binarnej otrzymano zapis: 0011 0111 1110 1111 1011 1111 0111 1011. Zera występują na pozycjach bitów: 2, 7, 14, 20, 27. Porównując to z wciśniętymi klawiszami: SW1, SW7, SW14, SW20, SW27, stwierdzono poprawność odwzorowania tej kombinacji w programie kontrolera interfejsu.

Zgodnie z zamieszczoną powyżej metodą sprawdzono wszystkie klawisze wciskając je pojedynczo oraz kilka innych kombinacji. Stwierdzono więc poprawność działania bloku interfejsu klawiatury oraz oprogramowania i integracji z blokiem kontrolera interfejsu.

5.2.2. Weryfikacja działania odczytu parametrów nastaw interfejsu użytkownika

Aby stwierdzić poprawność działania interfejsu nastaw potencjometrów opracowano procedurę porównującą odczyt napięcia na odpowiednim kanale przetwornika ADC mikrokontrolera STM32L053R8 ze zmierzonym przez woltomierz faktycznym napięciem na rozważanym potencjometrze. Odczytana poprzez debugger wartość o zidentyfikowanym numerze przeliczana jest na napięcie poprzez znajomość napięcia referencyjnego przetwornika. Następnie mierzy się napięcie na wyjściu dzielnika napięcia tworzonego przez potencjometr oraz notuje się stan mechaniczny obrotu potencjometra. Te trzy dane dla każdego z potencjometrów są porównywane i sprawdzana jest różnica w odczycie. Jeżeli błąd jest na tyle mały, żeby go zaakceptować – test określa się jako pozytywny.

Na potrzeby ilustracji arbitralnie przyjęto zestaw nastaw dla interfejsu i wykonano procedurę opisaną powyżej. Po ustawieniu gałek potencjometrów w odpowiedni sposób odczytano poprzez debugowanie wartości zawarte w tablicy *ADCData* (zdjęcie 6).



ADCData	0x2000003...	unsigned in...
[0]	0x00000000	unsigned int
[1]	0x00000023	unsigned int
[2]	0x00000053	unsigned int
[3]	0x000000A3	unsigned int
[4]	0x000000C5	unsigned int
[5]	0x000000FF	unsigned int

Zdjęcie 6 Zrzut ekranu pokazujący wartości odczytane z przetwornika ADC

Następnie odczytane wartości przeliczono na napięcie i porównano z napięciem zmierzonym. Przy okazji określono poprawność określenia kolejności potencjometrów w programie (rysunek 9). Maksymalny otrzymany błąd o wartości 0,1 V jest w zupełności satysfakcjonujący, zwłaszcza uwzględniając to, że przetwornik działał z rozdzielczością jedynie 8-bitową.



Wartość odczytana z debuggera	0x00	0x23	0x53	0xA3	0xC5	0xFF
Wartość obliczona [V]	0	0.45	1.07	2.10	2.54	3.29
Wartość zmierzona [V]	0.00	0.45	1.05	2.00	2.44	3.30

Rysunek 9 Podsumowanie testu sprawdzającego poprawność działania interfejsu nastaw syntezy

5.3. Procesor sygnałowy TMS320C5535 oraz układ kodeka audio TLV320AIC3204

Test poprawności działania oprogramowania procesora sygnałowego oraz poprawności działania układu kodeka audio oraz jego konfiguracji polegała na odsłuchu poprzez podłączone do wyjścia audio układu kodeka słuchawki przez osobę słuchającą. Drugim testem było podłączenie wyjścia audio kodeka do wejścia karty dźwiękowej komputera PC i nagraniu dźwięku poprzez program do obróbki dźwięku *Audacity*. Sygnałem testowanym, generowanym przez procesor sygnałowy jest sygnał o stałej częstotliwości podstawowej 1 kHz. Wywołanie dźwięku następuje poprzez wciśnięcie jednego z przycisków obsługiwanych przez przetwornik ADC procesora sygnałowego (podrozdział 4.6.3). Drugi z tych przycisków modyfikuje ilość harmonicznnych znajdujących się w sygnale. Podczas testu współczynnik *pot5* zmniejsza się jak 1/k dla kolejnych harmonicznnych oraz *pot6* ustawiony jest na 0, aby uzyskać dźwięk charakterystyczny dla sygnału prostokątnego. Test nie bada

działania polifonii. Te ograniczenia wprowadzone są ze względu na nierozwiązane problemy dotyczące komunikacji szeregowej pomiędzy procesorem sygnałowym, a mikrokontrolerem obsługującym interfejs użytkownika.

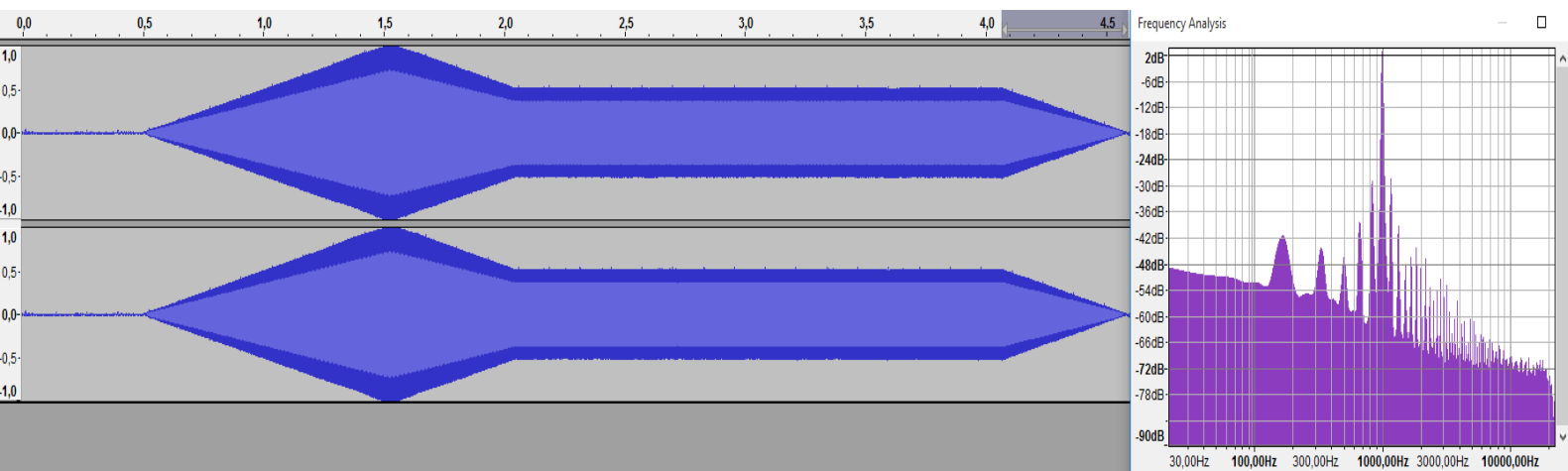
5.3.1 Oscylogramy sygnału mierzonego na wyjściu kodeka audio

Sygnał opisany wcześniej w tym rozdziale nagrano przez kartę dźwiękową. Parametry generacji obwiedni ustawiono (listing 5) odpowiednio na:

- „attack” na 1000 ms,
- „decay” na 500 ms,
- „sustain” na 50% (czyli połowę zakresu – wartość 0x3FFF),
- „release” na 500 ms.

Wygenerowano tylko pojedynczą składową sinusoidalną ($k = 1$). Nagrano cały okres trwania dźwięku i poddano analizie (zdjęcie 7). Stwierdzono, że obwiednia sygnału jest zgodna z oczekiwaniem. Dodatkowo przeprowadzono analizę widmową uzyskanego dźwięku, stwierdzając, że jedyna znacząca składowa dźwięku to 1kHz. Poprzez graficzny pomiar (względnie niedokładny) w programie stwierdzono odpowiednio, że:

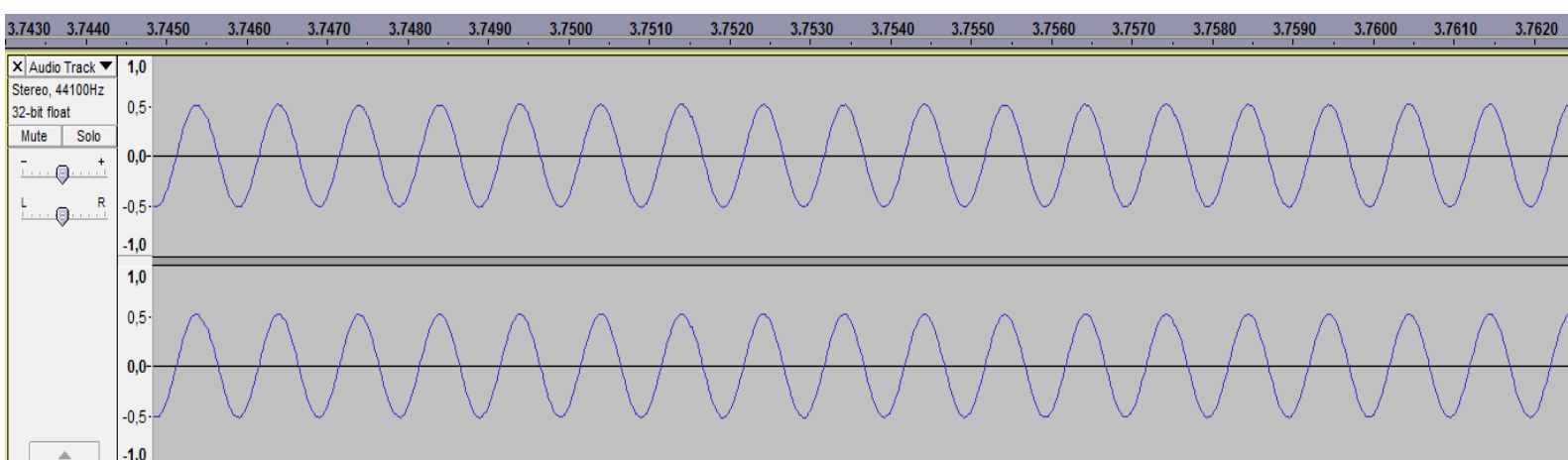
- czas trwania fazy „attack” to 1,034 s. ,
- czas trwania fazy „decay” to 0,515 s. ,
- poziom dźwięku w fazie „sustain” jest na poziomie ok. 0.5 wartości maksymalnej,
- czas trwania fazy „relay” to 0,500 s. .



Zdjęcie 7 Zrzut ekranu z programu Audacity pokazujący pierwszy nagrany dźwięk oraz analizę widmową tego zarejestrowanego sygnału

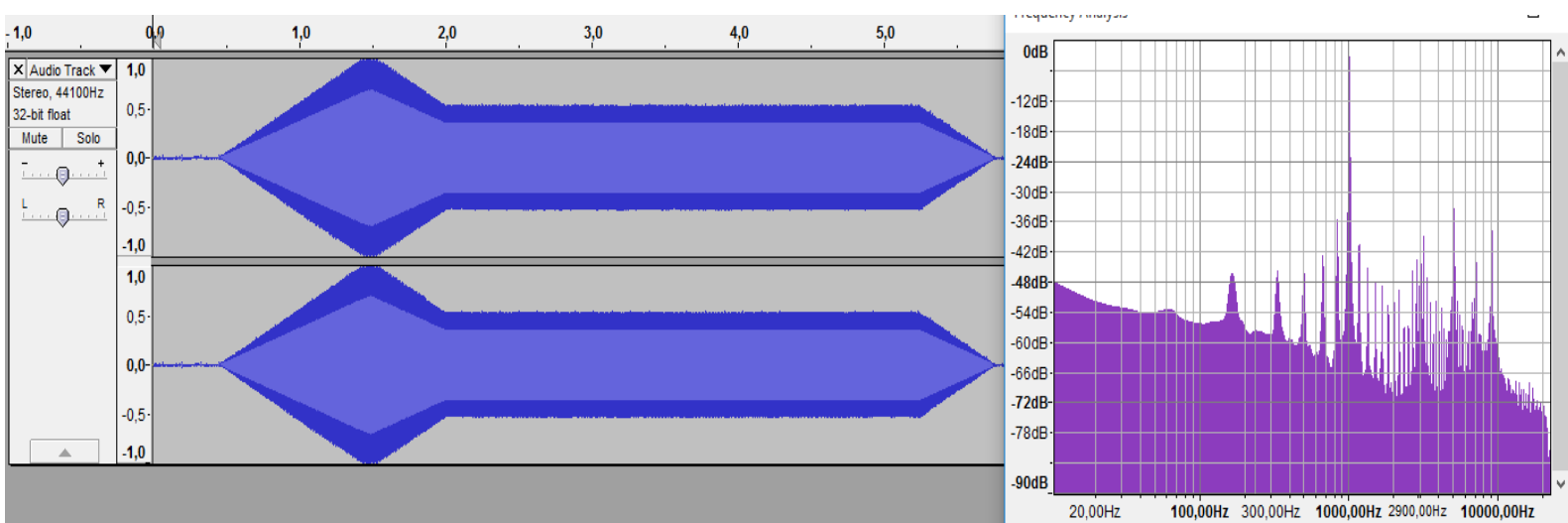
Mając na uwadze, że pomiar graficzny w programie może być niedokładny – otrzymane wyniki są w pełni zadowalające i wskazują, że wszystko działa tak jak powinno.

Po zmianie skali dziedziny czasu (zdjęcie 8) otrzymanego oscylogramu otrzymano obraz kształtu sygnału. Tak jak się spodziewano był to sygnał sinusoidalny o częstotliwości 1kHz (po pomiarze metodą graficzną).



Zdjęcie 8 Zrzut ekranu z programu pokazujący w przybliżeniu pierwszy z nagranych dźwięków

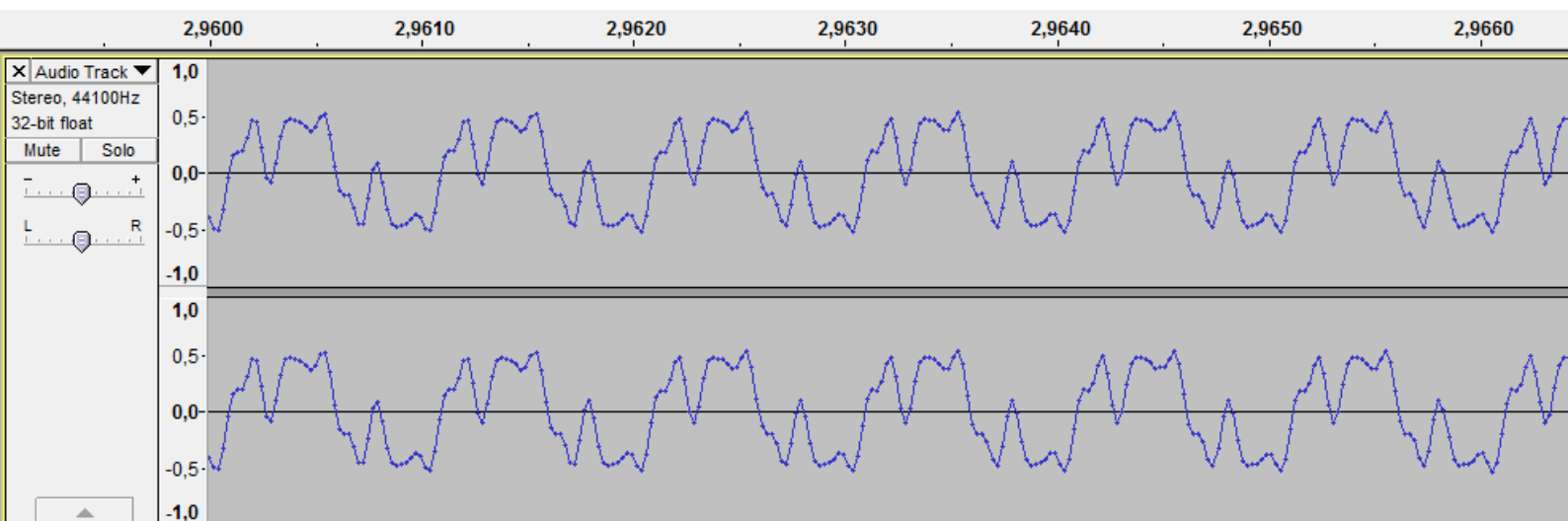
Drugi test odbył się z udziałem wygenerowanego dźwięku o identycznych parametrach jak w teście poprzednim. Wciśnięto jednak czterokrotnie przycisk odpowiedzialny za dodanie kolejnych harmonicznnych. Nagrano pełen okres wybrzmiewania wygenerowanego dźwięku poprzez kartę dźwiękową tak jak poprzednio (zdjęcie 9). Dodatkowo dokonano analizy częstotliwościowej sygnału.



Zdjęcie 9 Zrzut ekranu z programu Audacity pokazujący drugi nagrany dźwięk oraz analizę widmową tego zarejestrowanego sygnału

Obwiednia sygnału jest identyczna jak w poprzednim przypadku. Różnicę widać jedynie na wykresie częstotliwości. Widoczne są na nim nieparzyste składowe harmoniczne o częstotliwościach kolejno 3 kHz, 5kHz, 7 kHz, 10 kHz. Amplitudy tych harmonicznnych nie zgadzają się jednak z założeniami co prawdopodobnie związane jest z ograniczoną dokładnością arytmetyki stałoprzecinkowej w operacji dzielenia implementowanej przez kompilator języka C.

Po zmianie skali dziedziny czasu (zdjęcie 10) otrzymanego oscylogramu otrzymano obraz kształtu sygnału. Ponieważ faza kolejnych harmoniczných jest generowana losowo otrzymano sygnał o nieregularnych kształtach.



Zdjęcie 10 Zrzut ekranu z programu pokazujący w przybliżeniu drugi z nagranych dźwięków

5.3.2 Subiektywne wrażenia osoby słuchającej przez słuchawki

Podczas odsłuchu, przy pewnych ustalonych parametrach syntezy (takich jak w przypadku oscylogramu 7) stwierdzono jednoznacznie, że dźwięk jest wysoki, głośny i czysty co jest oczywiście zgodne z powszechnym i subiektywnym odczuciem typowego człowieka na sygnał sinusoidalny o częstotliwości 1kHz. Wyraźnie też słychać wszystkie fazy generacji obwiedni. Słychać narost głośności dźwięku tuż po naciśnięciu przycisku co odpowiada fazie „attack”. W taki sam sposób stwierdzono poprawność działania innych faz generacji obwiedni. Po dodaniu kolejnych harmoniczných stwierdzono, że dźwięk zmienia swoją barwę wydając się bardziej „ostry”. Po odsłuchu sygnału prostokątnego przefiltrowanego, tak aby ilość składowych harmoniczných nieparzystych zgadzała się z ilością nieparzystych harmoniczných, wygenerowanego przez skrypt napisany w programie *MATLAB*, słuchający miał wrażenie, że oba porównywane dźwięki są podobne.

W drugiej fazie testu kolejno wzbogacano generowany sygnał o kolejne składowe nieharmoniczne o malejącej amplitudzie (podrozdział 5.3.1). Osoba odsłuchująca porównywała barwę tego dźwięku z dźwiękiem wygenerowanym przez skrypt napisany w programie *MATLAB*. Sygnał ten tworzony był poprzez odpowiednie przefiltrowanie sygnału prostokątnego przez filtr dolnoprzepustowy w taki sposób, aby w sygnale pozostawały składowe harmoniczne takie same w sygnale, który wydobyty został przez syntezytor. Osoba

słuchająca określiła w skali od 1 - 10 punktów podobieństwo do sygnału wzorcowego. Otrzymane wyniki zgromadzono w tabeli 1.

Tabela 1 Wyniki oceny subiektywnej podobieństwa dwóch sygnałów

Ilość kolejnych nieparzystych składowych harmonicznych	Obiektywna ocena podobieństwa sygnału w skali 1 - 10
0	10
1	8
2	8
3	5
4	5

Wyniki drugiej fazy testu nie są satysfakcjonujące, dlatego jeżeli planowany byłby rozwój projektu należy wykonać rewizję implementacji algorytmu syntezy addytywnej wykonywanej w języku C.

6. Podsumowanie

W niniejszej pracy opracowano kompletny projekt budowy syntezy muzycznego opartego na procesorze sygnałowym TMS320C5535 oraz na tej podstawie skonstruowano prototyp konstrukcji takiego syntezy. W zakresie obsługi klawiatury oraz interfejsu nastaw parametrów osiągnięto zadowalający wynik co stwierdzono na podstawie przeprowadzonych testów. Pomyślnie również przebiegła konfiguracja układu kodeka audio oraz procesora sygnałowego. Uzyskane podczas testów systemu oscylogramy potwierdzają poprawność działania części oprogramowania generującego obwiednie typu ADSR. W rozdziale piątym tej pracy zostały przedstawione testy działania algorytmu syntezy addytywnej, które pokazały, że oczekiwane wartości amplitud kolejnych harmoniczných generowanych sygnałów różnią się od wartości amplitud faktycznych, a więc dalszy rozwój projektu wymaga weryfikacji implementacji kodu algorytmu wykonywanym na procesorze sygnałowym. Niemniej jednak nadal możliwe jest tworzenie różnych barw dźwięków poprzez zmiany wielkości parametrów syntezy. Powodem dla którego testy nie były przeprowadzone dla weryfikacji polifonii były problemy z nawiązaniem komunikacji szeregowej pomiędzy procesorem sygnałowym, a mikrokontrolerem obsługującym interfejs nastaw parametrów syntezy. Z tego względu komfortowa gra na zbudowanym prototypie nie jest możliwa. Dlatego priorytetem w dalszym rozwoju projektu jest rozwiązanie tego problemu.

Istnieje możliwość ulepszenia projektu opracowanego w tej pracy w szerokim zakresie. Dla otrzymania wspomnianej w rozdziale pierwszym - kompaktowości instrumentu - możliwe jest opracowanie płytki PCB zawierającej wszystkie potrzebne układy: procesora sygnałowego, mikrokontrolera oraz układu kodeka. Natomiast dalsze ograniczenie zużycia energii elektrycznej przez system powinno być zrealizowane poprzez wykorzystanie wbudowanych funkcjonalności zastosowanych układów mikroprocesorowych, takich jak stany uśpienia czy wyłączenie taktowania nieużywanych układów peryferyjnych. Wysoka mobilność układu możliwa jest do uzyskania poprzez integrację dodatkowego modułu zasilania. Ponieważ zastosowano jedynie dwa potencjometry zmieniające parametry syntezywanego sygnału (kolejne cztery służyły do zmiany parametrów generacji obwiedni) kolejnym krokiem do ubogacenia możliwości sterowania syntezy przez użytkownika jest opracowanie syntezy grupowej z wykorzystaniem większej ilości wskaźników zmiany parametrów syntezy. Ponieważ obsługa większej ilości potencjometrów może jednak być kłopotliwa, dlatego też warto rozważyć zamianę ich na enkodery.

Chociaż nie wszystkie założenia projektowe zostały spełnione, opisywane urządzenie spełnia swoje podstawowe zadanie i daje bazę do konstrukcji w pełni funkcjonalnego instrumentu muzycznego cechującego się szerokimi możliwościami wpływania na charakter syntezy, energooszczędnością, mobilnością oraz kompaktowością.

7. Bibliografia

- [1] Gordon Reid, Sound On Sound, Synth Secrets, Part 1: Analogue Oscillators, Filters & LFOs <http://www.soundonsound.com/sos/jun00/articles/synthsec.htm> [dostęp: 3 grudnia 2015]
- [2] Gordon Reid, Sound On Sound, Synth Secrets, Part 14: An Introduction To Additive Synthesis <http://www.soundonsound.com/sos/jun00/articles/synthsec.htm> [dostęp: 3 grudnia 2015]
- [3] Julius O. Smith III, Spectral Audio Signal Processing", W3K Publishing, 2011, ISBN 978-0-9745607-3-1.
- [4] Nota katalogowa płytki ewaluacyjnej STM32L053DISCOVERY http://www.st.com/st-web-ui/static/active/en/resource/technical/document/data_brief/DM00122138.pdf [dostęp: 3 grudnia 2015]
- [5] Nota katalogowa płytki ewaluacyjnej ezdsp5535 http://support.spectrumdigital.com/boards/ezdsp5535/revc/files/ezdsp5535_TechRef_RevC.pdf [dostęp: 03.12.2015]
- [6] Nota katalogowa rodziny układów TMS320C55x <http://www.ti.com/lit/ug/spru393/spru393.pdf> [dostęp: 03.12.2015]
- [7] Nota katalogowa rdzenia CPU rodziny układów TMS320C55x <http://www.ti.com/lit/ug/spru371f/spru371f.pdf> [dostęp: 03.12.2015]
- [8] Nota katalogowa procesora sygnałowego TMS320C5535 <http://www.ti.com/lit/ds/symlink/tms320c5532.pdf> [dostęp: 03.12.2015]
- [9] Nota katalogowa rodziny układów STM32L0 http://www.st.com/web/en/resource/technical/document/reference_manual/DM00095744.pdf [dostęp: 3 grudnia 2015]
- [10] Nota katalogowa mikrokontrolera STM32L053R8 <http://www.st.com/web/en/resource/technical/document/datasheet/DM00105960.pdf> [dostęp: 3 grudnia 2015]
- [11] Nota katalogowa układu kodeka audio TLV320AIC3204 <http://www.ti.com.cn/general/cn/docs/lit/getliterature.tsp?genericPartNumber=tlv320aic3204&fileType=pdf> [dostęp: 3 grudnia 2015]
- [12] Nota katalogowa układu wzmacniaczy operacyjnych LM324 <http://www.ti.com/lit/ds/symlink/lm124-n.pdf> [dostęp: 3 grudnia 2015]