Procesory DSP

Instrukcja laboratoryjna

Cyfrowa synteza sygnału - generator

Autorzy: dr inż. Paweł Dąbal

Ostatnia aktualizacja: 20.04.2022 r.

Wersja: 2022.0.2

Warszawa 2022

Spis treści

1. Wprowadzenie

Jedną z funkcjonalności udostępniania przez cyfrowe przetwarzanie sygnałów jest zdolność do cyfrowej syntezy sygnałów. W niniejszej instrukcji przedstawione zostaną najważniejsze zagadnienia związane z tym zadaniem i omówione na przykładowych konstrukcjach generatorów.

Niniejsza instrukcja ma na celu praktyczne zapoznanie ze sposobem tworzenia aplikacji do cyfrowej syntezy sygnału w paśmie akustycznym z użyciem płyty uruchomieniowej DSK6713. Uzyskane umiejętności pozwolą na kolejnych zajęciach na tworzenie rozbudowanych programów.

1.1. Cel ćwiczenia

Drugie ćwiczenie laboratoryjne ma na celu:

- dalsze poznawanie zintegrowanego środowiska projektowego *Code Composer Studio* w wersji v8.3.1 w zakresie tworzenia i zarządzania kodem źródłowym i konfiguracją projektu dla płyty uruchomieniowej *TMS320C6713DSK*;
- sposobem opisu cyfrowej syntezy sygnału w języku C;
- możliwościami debugowania oprogramowania (praca krokowa, obsługa pułapek, podgląd zmiennych);
- konfiguracją i użyciem liczników;
- konfiguracją i użyciem kodeka audio *TLV320AIC23B*;
- obsługą peryferii wejścia-wyjścia (LED, DIP);
- przypomnienie i usystematyzowanie wiedzy i umiejętności z zakresu posługiwania się językiem *C*,
- sposobem używania dołączonych bibliotek;
- praktyczne korzystanie z systemu kontroli wersji *Git* i serwisu *GitHub.com*.

1.2. Wymagania wstępne

Przed przystąpieniem do wykonywania ćwiczenia laboratoryjnego należy zapoznać się z:

- podstawowymi informacjami technicznymi dotyczącymi procesora sygnałowego *TMS320C6713*, a w szczególności z mapą pamięci (przygotować sobie tabelę *Memory Map Sumary*);
- schematem blokowym oraz ideowym płyty uruchomieniowej *TMS320C6713DSK*, a w szczególności zwrócić uwagę na informację niezbędne do obsłużenia diod LED oraz przełączników DIP-SWITCH;
- dokumentacją kodeka audio *TLV320AIC23B*, a w szczególności zwrócić uwagę na maksymalne, dopuszczalne poziomy napięć wejściowych, schemat blokowy;
- sposobem połączenia kodeka i procesora DSP, a w szczególności zapoznać się z dokumentacją użytego interfejsu;
- teorią cyfrowego przetwarzania sygnałów (próbkowanie, filtrowanie, transformata);
- elementami programowania w języku C (pętle, struktury, unie, funkcje, zmienne)
- utworzyć bezpłatne konto na platformie ***GitHub*** - posłuży ona do dokumentowania realizowanych prac.

1.3. Opis stanowiska laboratoryjnego

Stanowisko składa się z komputera PC z zainstalowanym oprogramowaniem *Code Composer Studio* w wersji 8.3.1, oscyloskopu cyfrowego RIGOL DS1052E, generatora arbitralnego RIGOL DG1022 i płyty uruchomieniowej *TMS320C6713DSK* z procesorem DSP *TMS320C6713* podłączonej do komputera za pomocą interfejsu USB2.0.

Oscyloskop i generator połączone są za pomocą kabla zakończonego z jednej strony końcówką *jack* 3,5 mm do płyty uruchomieniowej, a z drugiej strony gniazdem BNC do wyjścia CH1 generatora i wejścia CH1 oscyloskopu. Dodatkowo do wejścia CH2 oscyloskopu podłączona może być standardowa sonda pomiarowa. Generator może być źródłem sygnału arbitralnego, który będzie podlegał przetworzeniu. Możliwe jest skonfigurowanie połączeń tak aby komputer PC był źródłem/odbiornika sygnału. W laboratorium dostępne są również zestawy słuchawkowe z mikrofonem, którą umożliwią prace eksperymentalne na rzeczywistych próbkach głosowych i ewentualny odsłuch .

1.4. Uwagi odnośnie wykonania ćwiczenia laboratoryjnego

Z racji, że w trakcie ćwiczeń laboratoryjnych studenci będą korzystać z nieobudowanych układów elektronicznych (płyta uruchomieniowa), sond pomiarowych i skonfigurowanych połączeń należy zachować szczególne warunki porządku na stanowisku laboratoryjnym. Należy uważać, aby nie doprowadzić do zwarcia, ponieważ może to doprowadzić do nieodwracalnego uszkodzenia układów elektronicznych. Wszelkie nieprawidłowości w działaniu należy zgłaszać prowadzącemu. Nie należy zmieniać konfiguracji połączeń przy włączonym sygnale. W sposób szczególny należy pamiętać o dopuszczalnych parametrach napięciowych wejść płyty uruchomieniowej.

Bardzo ważne jest wykonywanie podanych w instrukcji poleceń w podanej kolejności. Niekiedy nie ma możliwości wykonania pominiętego kroku bez rozpoczynania procedury od nowa. Ważny jest również porządek i dbałość o czytelność i prawidłowe formatowanie kodu źródłowego.

1.5. Sposób rozliczenia ćwiczenia laboratoryjnego

Każde zajęcia kończą się oceną. W każdych zajęciach należy uczestniczyć. Ocena końcowa z laboratorium to średnia arytmetyczna ocen z każdego spotkania zaokrąglona zgodnie z aktualnym programem studiów do pełnej oceny. Zajęcia laboratoryjne składać mogą się z następujących elementów, z których każdy jest indywidualnie punktowany:

I. kolokwium na początek (*indywidualna pisemna odpowiedź na pytania*);
II. zadań podstawowe do wykonania (*przepisz / uruchom / sprawdź*);
III. zadania rozszerzające do wykonania (*stwórz kod* / *uruchom* / *sprawdź*).

W ocenianiu prac praktycznych sprawdzane jest spełnienie następujących kryteriów:

A. przepisaniu podanego kodu programu w instrukcji do przygotowanego projektu dla zadań podstawowych oraz napisanie lub zmodyfikowanie kodu realizującego zadaną funkcjonalność;
B. skompilowaniu programu i usunięciu ewentualnych błędów związanych z przepisaniem;
C. sprawdzenie praktyczne programu za pomocą wcześniej przygotowanego i zweryfikowanego kodu;
D. przedstawienie efektów prac do oceny prowadzącemu;
E. wykonaniem migawki z podaną treścią komentarza.

Kolokwium na początek zajęć składającego się z 5 pytań/zadań, na którego napisanie jest 10 minut obejmującego swoim zakresem dotychczas realizowane zajęcia, za które łącznie można uzyskać 10 punktów.

Przy każdym zadaniu podano liczbę możliwych punktów do zdobycia. Łącznie za poprawne wykonanie zadań można uzyskać 50 punktów, jeżeli zrealizowane są wszystkie kryteria. W przypadku zadań uzupełniających premiowana jest **jakość** i **czas realizacji** zaprezentowanego rozwiązania. Dodatkowo za odpowiedź na

postawione pytanie można uzyskać punkty premiowe. Ocena końcowa wyznaczona jest następująco: **2,0** <0; 21), **3,0** <21; 26), **3,5** <26; 31), **4,0** <31; 36), **4,5** <36; 41), **5,0** <41; 50).

2. Zadania podstawowe do realizacji

W tej części instrukcji zamieszczone są treści, z którymi obowiązkowo należy się zapoznać i praktycznie przećwiczyć. Ważne jest, aby zapamiętać wykonywane przedstawione czynności, aby móc na kolejnych zajęciach wykonywać je na kolejnych zajęciach bez potrzeby sięgania do niniejszej instrukcji.

Uwaga: załączone wycinki z ekranu są poglądowe i pomagają jedynie w wskazaniu lokalizacji elementów interfejsu. Należy używać wartości podanych w tekście.

2.1. Zagadnienia wstępne

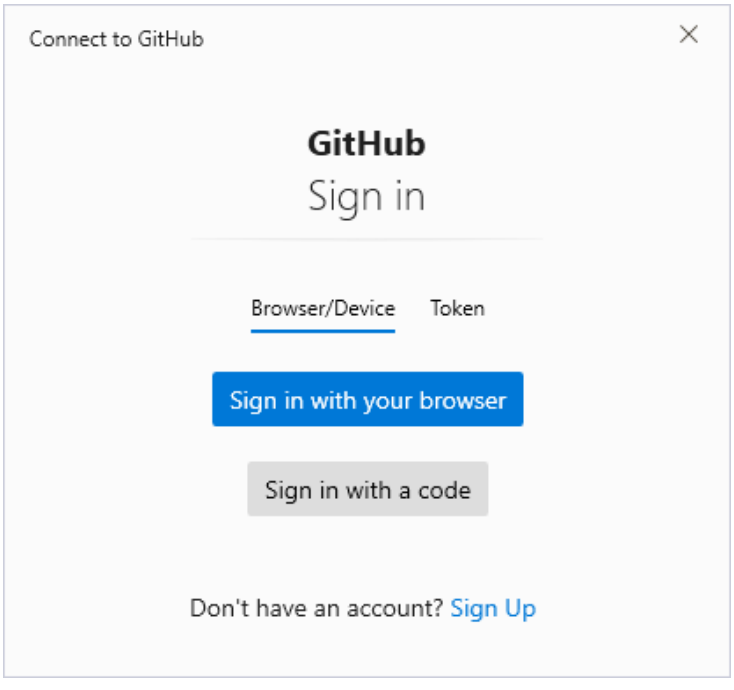
Na wstępie praktycznych zmagań z zagadnieniem programowania procesora sygnałowego opisane zostaną elementy związane z kontrolą wersji.

2.1.1. Dołączenie do wirtualnej klasy i utworzenie repozytorium bazowego dla zadania

Na zajęciach należy dołączyć do wirtualnej grupy w ramach *Classrom GitHub* za pomocą udostępnionego odnośnika przez prowadzącego zajęcia do zadania. Odnośnik należy użyć w nowej zakładce przeglądarki po otwarciu, którego tworzone jest nowe zadanie indywidualne. Z listy należy wybrać swój adres e-mail i potwierdzić przyjęcie zadania (ang. *assignment*). Automatycznie zostanie utworzone prywatne repozytorium indywidualnie dla każdego studenta na podstawie przygotowanego repozytorium-szablonu. W sytuacji, jeżeli student nie może odszukać się na liście (np. ktoś inny podłączył się pod daną osobę) proszę zgłosić to prowadzącemu zajęcia. W celu skorygowania nieprawidłowości. Zaakceptowanie zadania wiąże się również z dołączeniem do organizacji – wirtualnego konta organizacji w ramach którego tworzone są indywidualne repozytoria do zadań, z którego prowadzący zajęcia mają dostęp do wszystkich repozytoriów tworzonych w ramach zajęć.

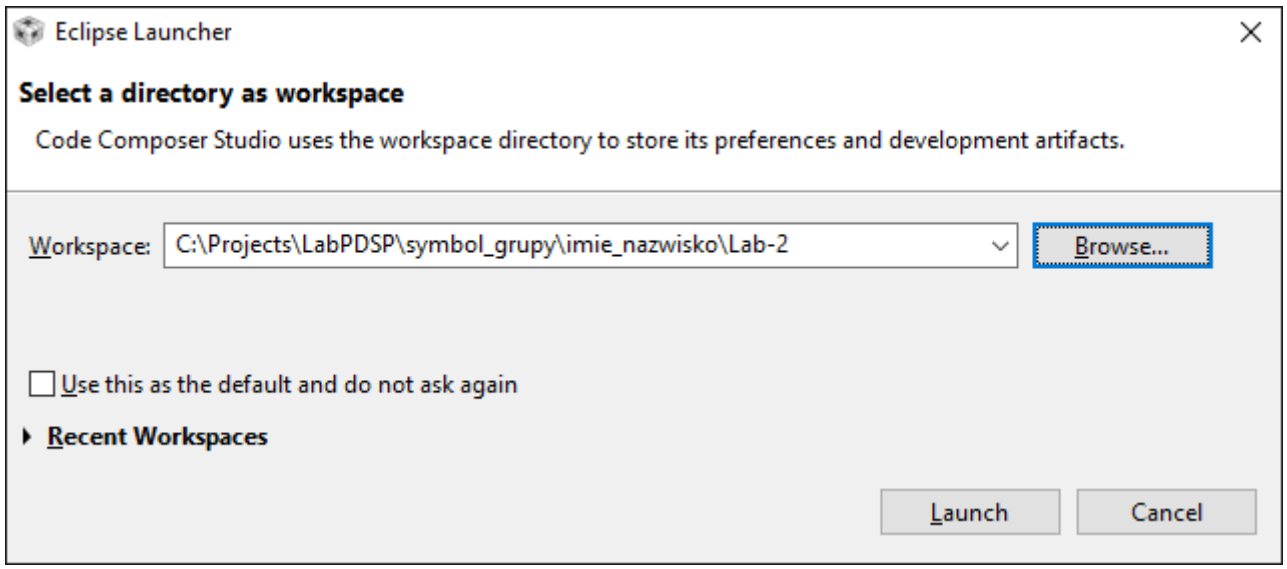
2.1.2. Pobranie repozytorium i konfiguracja lokalna

W celu pobrania repozytorium należy odszukać na pulpicie skrót o nazwie *LabGitConfig_205*, który uruchomi skrypt wiersza poleceń, w którym należy podać: 1) swoje imię i nazwisko, 2) adres e-mail, 3) symbol grupy, 4) nazwę ćwiczenia „*LabPDSP*” 5) numer ćwiczenia, 6) adres indywidualnego repozytorium uzyskany w wcześniejszym punkcie. Po wykonaniu punktu 6) pojawi się okno logowania do serwisu *GitHub* podobne do zamieszczonego obok. W celu połączenia należy wybrać przycisk *Sign in with your browser* co spowoduje otworzenie nowej karty przeglądarki w którym należy udzielić dostępu do konta. Po uzyskaniu zgody nastąpi sklonowanie projektu z serwera do lokalizacji wynikającej z symbolu grupy i nazwy użytkownika. **Nie należy zamykać** okna wiersza poleceń.



Przed utworzeniem pierwszego projektu aplikacji dla mikrokontrolera należy otworzyć plik ***READMY.md*** znajdujący się w sklonowanym projekcie na komputerze i uzupełnić bądź zmienić na

właściwe wskazane w nim pozycje dotyczące studenta (datę, symbol grupy, imię i nazwisko). Pliki z rozszerzeniem **.md* są często elementami projektów. W odróżnieniu od plików tekstowych umożliwiają wprowadzenie elementów formatowania tekstu, który jest interpretowany przez serwery *Git* jak i samo środowisko *VS Code.* Po uzupełnieniu pliku można wykonać pierwszą migawkę (ang. *commit*) projektu i wysłać ją na serwer. Do obsługi systemu *Git* można skorzystać z środowiska *VS Code* otwierając w nim katalog sklonowanego repozytorium (*C:\Projects\LabPDSP\symbol_grupy\imie_nazwisko\Lab-2*). Jako komentarz do migawki należy podać „***Zadanie 2.1.2 - aktualizacja pliku READMY***”, a następnie zatwierdzić przyciskiem ***Commit*** (✓). W celu przesłania zmian w repozytorium na serwer należy wypchnąć (ang. *push*) wykonane zmiany (migawki) na serwer. Można to zrobić na dwa sposoby. Pierwszy to z panelu *Source Control* wybrać przycisk „…” (*More Actions*…) i z rozwijanej listy wybrać pozycję ***Push***. Po wykonaniu tych operacji można przejść do okna przeglądarki i odświeżyć widok projektu w celu zweryfikowanie poprawności przesłanych zmian. Alternatywny sposób wypchnięcia zmian to wydanie polecenia ***Git: Push*** w wierszu poleceń (*F1*).



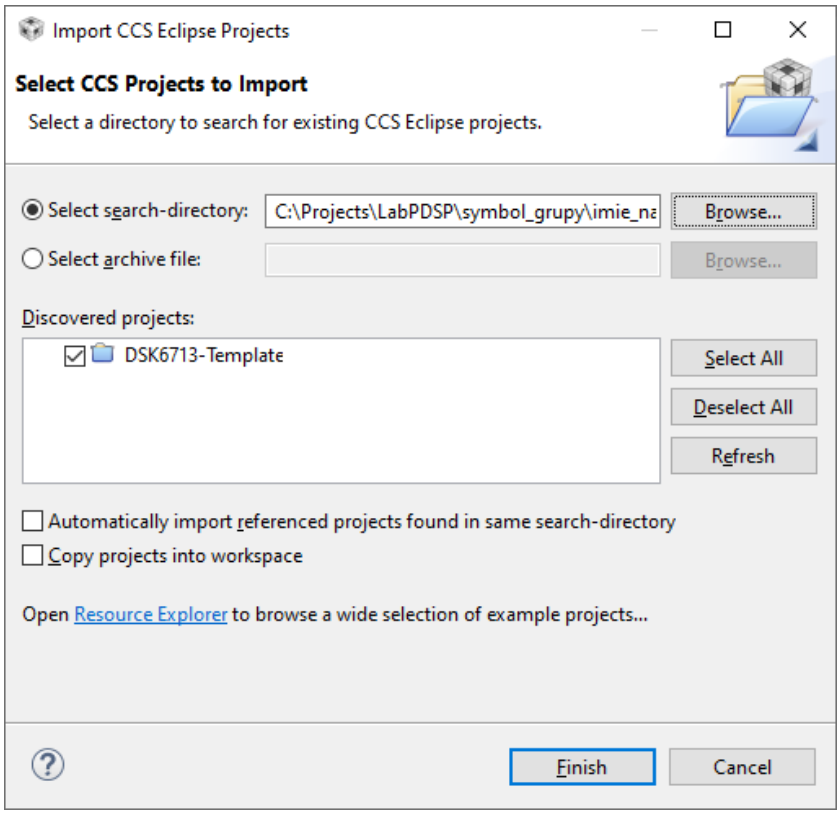
Mechanizm migawek pozwala na zapisywanie chwilowego stanu projektu. Można cofnąć się do wcześniejszej fazy projektu, tworzyć rozgałęzienia, łączyć rozgałęzienia. W trakcie zajęć ograniczymy się do prostej liniowej struktury migawek wykonywanych po zakończeniu każdego z zadań instrukcji opatrzonych stosownym komentarzem. W dalszej części instrukcji będą podane treści komentarzy jakimi należy opatrzyć realizowane migawki. Powstanie zatem coś w rodzaju sprawozdania z zajęć, które będzie przechowywane w serwisie *GitHub*.

2.2. Import projektu aplikacji bazowej i jej uruchomienie (2 pkt.)

Należy uruchomić środowisko *Code Composer Studio 8.3.1* i w pierwszym oknie wyboru katalogu przestrzeni roboczej wskazać katalog sklonowanego repozytorium: *C:\Projects\LabPDSP\symbol_grupy\imie_nazwisko\Lab-2*. Odpowiednio na kolejnych zajęciach będzie wskazywany kolejny katalog. Uruchamia się właściwa aplikacja z otwartą zakładką powitalną *Getting Started*, którą należy zamknąć.

2.2.1. Import projektu bazowego w środowisku i jego kompilacja

W celu zaimportowania projektu bazowego należy wybrać z menu ***File*** pozycję ***Import***, a następnie rozwinąć ***C/C++*** i wybrać ***CCS Projects***, wybór zatwierdzić przyciskiem ***Next***. W kolejnym oknie w polu ***Select root directory*** należy podać ścieżkę do katalogu z repozytorium (można użyć przycisku ***Browse*** w celu wskazania katalogu). W części *Discovered projects* zostanie automatycznie dodany i zaznaczony projekt ***DSK6713-Template***, który należy zaimportować naciskając przycisk ***Finish***.



Rysunek 2.1 Okno kreatora tworzenia nowego projektu (New CCS Project)

Aby skompilować utworzony projekt należy wybrać z menu ***Project->Build Projcet***. Proces ten w przypadku pierwszego uruchomienia może zająć około minuty z racji, że kompilowane są w tedy niejawnie dołączone biblioteki standardowe. Procedurę uruchamiania kompilacji można zautomatyzować w taki sposób, że po każdym zapisaniu pliku źródłowego będzie uruchamiana. Aby to zrobić należy wybrać w menu ***Project->Build Automatically***. Na dole okna pojawi się zakładka ***Console***, do której przekierowane są wszystkie komunikaty związane z przebiegiem procesu kompilacji.

2.2.2. Uruchomienie przygotowanego programu

W celu zweryfikowania możliwości programowania płyty uruchomieniowej należy podłączyć do niej zasilanie oraz przewód USB do komunikacji z komputerem, a następnie wybrać w menu ***Run->Debug*** (F11). Jeżeli procedura ładowania programu przebiegnie pomyślnie to okno główne zmieni swój widok na tak zwaną perspektywę (ang. *perspective*) *debugera*. Do zmiany perspektywy służą przyciski umieszczone w prawym górnym rogu okna głównego oraz pozycje w menu ***Window->Perspective***. Aby uruchomić program należy wybrać w menu ***Run->Resume*** (*F8*). W celu zatrzymania wykonywania programu należy wybrać w menu ***Run->Suspend*** (*Alt + F8*). Aby zakończyć pracę debugera i zamknąć połączenie z płytą uruchomieniową należy wybrać w menu ***Run->Terminate*** (*Ctrl + F2*). Należy zakończyć tryb pracy debuggera (*terminate*).

2.3. Cyfrowa synteza sygnału harmonicznego

Podstawowym sygnałem w telekomunikacji jest okresowy sygnał harmoniczny o przebiegu sinusoidalnym. Reprezentuje on zarówno ton sygnału akustycznego jak i sygnał nośny, który po zmodulowaniu może być transmitowany na duże odległości. Do cyfrowego generowania sygnału o określonych parametrach potrzebny jest jego opis matematyczny w dziedzinie czasu lub częstotliwości w sposób cyfrowy. W przypadku sygnału harmonicznego możemy zapisać:

$$y(n) = A_0 + A \cdot \sin\left(\frac{n \cdot 2 \cdot \pi \cdot f}{f_s} + \varphi_0\right)$$

gdzie: *n* – numer próbki, *f* -częstotliwość sygnału harmonicznego, f_s – częstotliwość próbkowania, φ_0 – faza początkowa, *A* – amplituda sygnału, A_0 – składowa stała amplitudy. Wyznaczenie wartości funkcji sinus może być zrealizowane za pomocą rozwinięcia w szereg Tylora; z użyciem dostatecznie dużej tablicy zawierającej kolejne wartości funkcji dla jednej ćwiartki okresu, a dalsze poprawienie dokładności odbywa się z użyciem aproksymacji liniowej; równania różnicowego. Sygnały okresowe takie jak piłokształtne, prostokątne, trapezowe realizowane są przez kształtowanie amplitudy sygnału w funkcji numeru próbki w okresie tego sygnału. Bardziej złożone przebiegi zapamiętane są w postaci tablic.

Przed przystąpieniem do dalszych prac należy utworzyć parę plików o nazwie ***generator.c*** i ***generator.h*** oraz umieścić je odpowiednio w katalogach ***source*** i ***include***. Będą one zawierać funkcjonalność generatora w postaci dedykowanych funkcji. Testowanie działania realizowane będzie w funkcji głównej *main*.

2.3.1. Synteza sygnału harmonicznego o stałej amplitudzie i częstotliwości (6 pkt.)

Cyfrowy oscylator wymaga przechowywania minimum informacji o amplitudzie, fazie i kroku fazy, który wynika z częstotliwości generowanego sygnału. W związku z czym do przechowywania tych informacji należy zdefiniować strukturę ***OSC_Cfg_t*** w pliku *generator.h* jak to przedstawione poniżej:

```
36 typedef struct {
37     float amplitude;    // Real value in V in range from 0 to 1.650
38     float phase;        // Real value in radians
39     float phaseStep;    // Real value in radians
40 } OSC_Cfg_t;
```

Konfiguracja pól struktury powinna zostać zrealizowana za pośrednictwem dedykowanej funkcji (***OSC_Init***), która sprawdzi parametry i wyznaczy krok fazy (***phaseStep***) na podstawie częstotliwości (***frequency***) podanej w parametrze. Należy dodać funkcję konfiguracji oscylatora w pliku *generator.c* jak to przedstawione poniżej.

```
 8 #include "pdsp.h"
 9 #include "generator.h"
10
11 void OSC_Init(OSC_Cfg_t * hOscCfg, float amplitude, float frequency){
12     frequency = (frequency < (CODEC_fs / 2.0f)) ? frequency : (CODEC_fs / 2.0f);
13     hOscCfg->amplitude = (amplitude < (CODEC_Vpp * 0.5f)) ? amplitude : (CODEC_Vpp * 0.5f);
14     hOscCfg->phaseStep = (frequency * PDSP_2PI_DIV_FS);
15     hOscCfg->phase = 0.0f;
16 }
```

Wartość nowej próbki należy wyznaczyć korzystając z zależności $y(n) = A \cdot \sin\left(\frac{n \cdot 2 \cdot \pi \cdot f}{f_s}\right)$, gdzie *A* – amplituda, $\frac{2 \cdot \pi \cdot f}{f_s}$ – krok fazy. Każdorazowo po wyznaczeniu próbki należy zaktualizować fazę (***phase***) przez zwiększenie o krok fazy i dokonać ewentualnej korekty wynikającej z okresowości funkcji trygonometrycznej (***OSC_GetValue***). Dodatkowo pomocne mogą być funkcję pozwalające na zmianę częstotliwości ***OSC_SetFrequency*** oraz amplitudy ***OSC_SetAmplitude*** syntezowanego sygnału. W pliku *generator.c* należy dodać kod wspomnianych funkcji jak to przedstawione poniżej.

```
18 float OSC_GetValue(OSC_Cfg_t * hOscCfg){
19     float wave;
20 
21     wave = hOscCfg->amplitude * sinf(hOscCfg->phase);
22     hOscCfg->phase += hOscCfg->phaseStep;
23     hOscCfg->phase = fmodf(hOscCfg->phase, PDSP_2PI);
24 
25     return wave;
26 }
27 
28 void OSC_SetFrequency(OSC_Cfg_t * hOscCfg, float frequency){
29     frequency = (frequency < (CODEC_fs / 2.0f)) ? frequency : (CODEC_fs / 2.0f);
30     hOscCfg->phaseStep = (frequency * PDSP_2PI_DIV_FS);
31 }
32 
33 void OSC_SetAmplitude(OSC_Cfg_t * hOscCfg, float amplitude){
34     hOscCfg->amplitude = (amplitude < (CODEC_Vpp * 0.5f)) ? amplitude : (CODEC_Vpp * 0.5f);
35 }
```

Aby możliwe było użycie przygotowanych funkcji w pliku *main.c* należy dodać prototypy przygotowanych funkcji do pliku *generator.h* jak to podano poniżej.

```
18 void OSC_Init(OSC_Cfg_t * hOscCfg, float amplitude, float frequency);
19 float OSC_GetValue(OSC_Cfg_t * hOscCfg);
20 void OSC_SetFrequency(OSC_Cfg_t * hOscCfg, float frequency);
21 void OSC_SetAmplitude(OSC_Cfg_t * hOscCfg, float amplitude);
```

W celu przetestowania funkcjonalności generatora należy zmodyfikować funkcję główną w pliku *main.c*. Najpierw należy dodać plik nagłówkowy *generator.h*, po dyrektywie dołączenia pliku nagłówkowego *pdsp.h*. Następnie należy w funkcji *main* zdefiniować nową zmienną typu *OSC_Cfg_t* o nazwie ***oscylator***, a następnie zainicjować ją przez wywołanie funkcji *OSC_Init* podając jako wartość *amplitudy* ***1.0f*** i *częstotliwości* ***1000.0f***. W pętli głównej *while* należy zastąpić przepisywanie próbki z wejścia kodeka na wyjście operacją zapisu do kodeka nowej wartości próbki z oscylatora pomnożona przez rozdzielczość napięciową zgodnie z poniższym kodem.

```
19 void main(){
20     PDSP_Init();
21     PDSP_CODEC_Init();
22     PDSP_INT_Init();
23 
24     OSC_Cfg_t oscylator;
25     OSC_Init(&oscylator, 1.0f, 1000.0f);
26     while (1){
27 #if PDSP_MODE == PDSP_MODE_POLL
28         DataIn = CODEC_GetSampleStereo();    // Odczytanie nowej próbki od kodeka
29         CODEC_SetSampleStereo(DataOut);      // Wysłanie próbki do kodeka
30         SampleNumber++;
31         // Przetwarzanie
32         DataOut = DataIn;
33         Int16 value = (Int16) (OSC_GetValue(&oscylator) * CODEC_V_TO_BIT);
34         DataOut.channel[0] = value;
35         DataOut.channel[1] = value;
36 #elif PDSP_MODE == PDSP_MODE_INT
37         if (NewData == true)        {
38             NewData = false;
39             DataOut = DataIn;
40             Int16 value = (Int16)(OSC_GetValue(&oscylator) * CODEC_V_TO_BIT);
41             DataOut.channel[0] = value;
42             DataOut.channel[1] = value;
43         }
44 #endif
45     }
46 }
```

Program należy skompilować i zweryfikować jego działanie korzystając z oscyloskopu. Jeżeli wszystko jest w porządku należy zgłosić prowadzącemu i po zatwierdzeniu wykonać migawkę z komentarzem „***Zadanie 2.3.1 – synteza sygnału harmonicznego***”.

2.3.2. Synteza sygnału cyfrowego w postaci oscylatora ze sprzężeniem zwrotnym (6 pkt.)

Cyfrowy oscylator ze sprzężeniem zwrotnym wymaga przechowywania informacji o amplitudzie, częstotliwości, współczynnikach oscylacji *cosTheta* i *sinTheta* oraz historii trzech ostatnich wartości. W związku z czym do przechowywania tych informacji należy zdefiniować strukturę ***SIN_Cfg_t*** w pliku *generator.h* jak to przedstawione poniżej.

```
typedef struct {
    float amplitude;
    float frequency;
    float cos_theta;
    float sin_theta;
    float y[3];
}SIN_Cfg_t;
```

Konfiguracja pól struktury powinna zostać zrealizowana za pośrednictwem dedykowanej funkcji (***SIN_Init***), która sprawdzi parametry i wyznaczy współczynniki oscylacji (***cosTheta***, ***sinTheta***) na podstawie częstotliwości (***frequency***) podanej w parametrze oraz inicjuje historię próbek. Należy dodać funkcję konfiguracji oscylatora w pliku *generator.c* jak to przedstawione poniżej.

```
void SIN_Init(SIN_Cfg_t * hSinOsc, float amplitude, float frequency){
    hSinOsc->amplitude = (amplitude < (CODEC_Vpp * 0.5f)) ? amplitude : (CODEC_Vpp * 0.5f);
    hSinOsc->frequency = (frequency < (CODEC_fs / 2.0f)) ? frequency : (CODEC_fs / 2.0f);
    hSinOsc->cos_theta = (cosf(PDSP_2PI_DIV_FS * hSinOsc->frequency) * 2);
    hSinOsc->sin_theta = (sinf(PDSP_2PI_DIV_FS * hSinOsc->frequency) * hSinOsc->amplitude);
    hSinOsc->y[0] = 0;
    hSinOsc->y[1] = 1;
    hSinOsc->y[2] = 0;
}
```

Wartość nowej próbki należy wyznaczyć wykonując kolejne operacje na historii wartości oraz przez przesunięcie próbek w historii (***SIN_GetValue***). Dodatkowo pomocne mogą być funkcję pozwalające na zmianę częstotliwości ***SIN_SetFrequency*** oraz amplitudy ***SIN_SetAmplitude*** syntezowanego sygnału. W pliku *generator.c* należy dodać kod wspomnianych funkcji jak to przedstawione poniżej.

```
float SIN_GetValue(SIN_Cfg_t * hSinOsc){
    hSinOsc->y[0] = (hSinOsc->cos_theta * hSinOsc->y[1]) - hSinOsc->y[2];
    hSinOsc->y[2] = hSinOsc->y[1];
    hSinOsc->y[1] = hSinOsc->y[0];
    return (hSinOsc->sin_theta * hSinOsc->y[0]);
}

void SIN_SetFrequency(SIN_Cfg_t * hSinOsc, float frequency){
    hSinOsc->frequency = (frequency < (CODEC_fs / 2.0f)) ? frequency : (CODEC_fs / 2.0f);
    hSinOsc->cos_theta = (cosf(PDSP_2PI_DIV_FS * hSinOsc->frequency) * 2);
    hSinOsc->sin_theta = (sinf(PDSP_2PI_DIV_FS * hSinOsc->frequency) * hSinOsc->amplitude);
}

void SIN_SetAmplitude(SIN_Cfg_t * hSinOsc, float amplitude){
    hSinOsc->amplitude = (amplitude < (CODEC_Vpp * 0.5f)) ? amplitude : (CODEC_Vpp * 0.5f);
    hSinOsc->cos_theta = (cosf(PDSP_2PI_DIV_FS * hSinOsc->frequency) * 2);
    hSinOsc->sin_theta = (sinf(PDSP_2PI_DIV_FS * hSinOsc->frequency) * hSinOsc->amplitude);
}
```

Aby możliwe było użycie przygotowanych funkcji w pliku *main.c* należy dodać prototypy przygotowanych funkcji do pliku *generator.h* jak to podano poniżej.

```
void SIN_Init(SIN_Cfg_t * hSinOsc, float amplitude, float frequency);
float SIN_GetValue(SIN_Cfg_t * hSinOsc);
void SIN_SetFrequency(SIN_Cfg_t * hSinOsc, float frequency);
void SIN_SetAmplitude(SIN_Cfg_t * hSinOsc, float amplitude);
```

W celu przetestowania funkcjonalności generatora należy zmodyfikować funkcję główną w pliku *main.c*. Należy w funkcji *main* zdefiniować nową zmienną typu *SIN_Cfg_t* o nazwie ***oscylator***, a następnie zainicjować ją przez wywołanie funkcji *SIN_Init* podając jako wartość *amplitudy* ***1.0f*** i *częstotliwości* ***1000.0f***. W pętli głównej *while* należy zastąpić przepisywanie próbki z wejścia kodeka na wyjście operacją zapisu do kodeka nowej wartości próbki z oscylatora pomnożona przez rozdzielczość napięciową zgodnie z poniższym kodem.

```
void main(){
    PDSP_Init();
    PDSP_CODEC_Init();
    PDSP_INT_Init();

    SIN_Cfg_t oscylator;
    SIN_Init(&oscylator, 1.0f, 1000.0f);
    while (1){
#if PDSP_MODE == PDSP_MODE_POLL
        DataIn = CODEC_GetSampleStereo();    // Odczytanie nowej próbki od kodeka
        CODEC_SetSampleStereo(DataOut);      // Wysłanie próbki do kodeka
        SampleNumber++;
        // Przetwarzanie
        DataOut = DataIn;
        Int16 value = (Int16)(SIN_GetValue(&oscylator) * CODEC_V_TO_BIT);
        DataOut.channel[0] = value;
        DataOut.channel[1] = value;
#elif PDSP_MODE == PDSP_MODE_INT
        if (NewData == true){
            NewData = false;
            DataOut = DataIn;
            Int16 value = (Int16)(SIN_GetValue(&oscylator) * CODEC_V_TO_BIT);
            DataOut.channel[0] = value;
            DataOut.channel[1] = value;
        }
#endif
    }
}
```

Program należy skompilować i zweryfikować jego działanie korzystając z oscyloskopu. Jeżeli wszystko jest w porządku należy zgłosić prowadzącemu i po zatwierdzeniu wykonać migawkę z komentarzem „***Zadanie 2.3.2 – synteza sygnału harmonicznego w postaci oscylatora ze sprzężeniem zwrotnym***”.

2.3.3. Synteza sygnału cyfrowego z użyciem tablicy (8 pkt.)

Synteza sygnału z użyciem akumulatora fazy i tablicy polega na wyznaczaniu nowej wartości dyskretnej fazy, a następnie na wskazywaniu odpowiedniej wartości próbki dla dyskretnej fazy sygnału. Wartości próbek mogą zostać wpierw wyznaczone i zapisane w postaci tablicy stałych lub mogą być wyznaczane przed pierwszym użyciem. Podejście takie wymaga przechowywania informacji o amplitudzie, indeksie fazy, długości tablicy oraz samych wartości próbek funkcji trygonometrycznej. W związku z czym do przechowywania tych informacji należy zdefiniować strukturę ***TAB_Cfg_t*** w pliku *generator.h* jak to przedstawione poniżej.

```
typedef struct {
    float amplitude;
    uint16_t phaseIndex;
    uint16_t length;
    float * samples;
}TAB_Cfg_t;
```

Konfiguracja pól struktury powinna zostać zrealizowana za pośrednictwem dedykowanej funkcji (***TAB_Init***), która sprawdzi parametry i wyznaczy wartości próbek dla całego okresu funkcji sinus z krokiem równym

ilorazowi częstotliwości próbkowania i zadanej częstotliwości (***frequency***) podanej w parametrze. Należy dodać funkcję konfiguracji oscylatora w pliku *generator.c* jak to przedstawione poniżej.

```
66 void TAB_Init(TAB_Cfg_t * hSinTab, float amplitude, float frequency){
67     frequency = (frequency < (CODEC_fs / 2.0f)) ? frequency : (CODEC_fs / 2.0f);
68     hSinTab->amplitude = (amplitude < (CODEC_Vpp * 0.5f)) ? amplitude : (CODEC_Vpp * 0.5f);
69     hSinTab->phaseIndex = 0;
70     hSinTab->length = (int16_t)(CODEC_fs / frequency);
71     hSinTab->samples = (float *) malloc(sizeof(float) * hSinTab->length);
72 
73     if (hSinTab->samples == NULL)
74         return;
75 
76     uint16_t i;
77     float phase = 0.0f;
78     float phaseStep = (PDSP_2PI / hSinTab->length);
79     float * pValue = hSinTab->samples;
80     for (i = 0; i < hSinTab->length; i++){
81         *pValue = sinf(phase);
82         phase += phaseStep;
83         pValue++;
84     }
85 }
```

Wartość nowej próbki należy wyznaczyć wykonując kolejne operacje na historii wartości oraz przez przesunięcie próbek w historii (***TAB_GetValue***). Dodatkowo pomocne mogą być funkcję pozwalające na zmianę częstotliwości ***TAB_SetFrequency*** oraz amplitudy ***TAB_SetAmplitude*** syntezowanego sygnału. W pliku *generator.c* należy dodać kod wspomnianych funkcji jak to przedstawione poniżej.

```
87 float TAB_GetValue(TAB_Cfg_t * hSinTab){
88     float wave;
89 
90     wave = (hSinTab->samples[hSinTab->phaseIndex] * hSinTab->amplitude);
91     hSinTab->phaseIndex++;
92     hSinTab->phaseIndex %= hSinTab->length;
93 
94     return wave;
95 }
96 
97 void TAB_SetFrequency(TAB_Cfg_t * hSinTab, float frequency){
98     free(hSinTab->samples);
99 
100     frequency = (frequency < (CODEC_fs / 2.0f)) ? frequency : (CODEC_fs / 2.0f);
101     hSinTab->length = (int16_t)(CODEC_fs / frequency);
102     hSinTab->phaseIndex = 0;
103     hSinTab->samples = (float *) malloc(sizeof(float) * hSinTab->length);
104     if (hSinTab->samples == NULL)
105         return;
106 
107     uint16_t i;
108     float phase = 0.0f;
109     float phaseStep = (PDSP_2PI / hSinTab->length);
110     float * pValue = hSinTab->samples;
111     for (i = 0; i < hSinTab->length; i++){
112         *pValue = sinf(phase);
113         phase += phaseStep;
114         pValue++;
115     }
116 }
117 
118 void TAB_SetAmplitude(TAB_Cfg_t * hSinTab, float amplitude){
119     hSinTab->amplitude = (amplitude < (CODEC_Vpp * 0.5f)) ? amplitude : (CODEC_Vpp * 0.5f);
120 }
```

Aby możliwe było użycie przygotowanych funkcji w pliku *main.c* należy dodać prototypy przygotowanych funkcji do pliku *generator.h* jak to podano poniżej.

```
42 void TAB_Init(TAB_Cfg_t * hSinTab, float amplitude, float frequency);
43 float TAB_GetValue(TAB_Cfg_t * hSinTab);
44 void TAB_SetFrequency(TAB_Cfg_t * hSinTab, float frequency);
45 void TAB_SetAmplitude(TAB_Cfg_t * hSinTab, float amplitude);
```

W celu przetestowania funkcjonalności generatora należy zmodyfikować funkcję główną w pliku *main.c*. Należy w funkcji *main* zdefiniować nową zmienną typu *TAB_Cfg_t* o nazwie ***oscylator***, a następnie zainicjować ją przez wywołanie funkcji *TAB_Init* podając jako wartość *amplitudy* ***1.0f*** i *częstotliwości* ***1000.0f***. W pętli głównej *while* należy zastąpić przepisywanie próbki z wejścia kodeka na wyjście operacją zapisu do kodeka nowej wartości próbki z oscylatora pomnożona przez rozdzielczość napięciową zgodnie z poniższym kodem.

```
19 void main(){
20     PDSP_Init();
21     PDSP_CODEC_Init();
22     PDSP_INT_Init();
23 
24     TAB_Cfg_t oscylator;
25     TAB_Init(&oscylator, 1.0f, 1000.0f);
26     while (1){
27 #if PDSP_MODE == PDSP_MODE_POLL
28         DataIn = CODEC_GetSampleStereo();    // Odczytanie nowej próbki od kodeka
29         CODEC_SetSampleStereo(DataOut);      // Wysłanie próbki do kodeka
30         SampleNumber++;
31         // Przetwarzanie
32         DataOut = DataIn;
33         Int16 value = (Int16)(TAB_GetValue(&oscylator) * CODEC_V_TO_BIT);
34         DataOut.channel[0] = value;
35         DataOut.channel[1] = value;
36 #elif PDSP_MODE == PDSP_MODE_INT
37         if (NewData == true){
38             NewData = false;
39             DataOut = DataIn;
40             Int16 value = (Int16)(TAB_GetValue(&oscylator) * CODEC_V_TO_BIT);
41             DataOut.channel[0] = value;
42             DataOut.channel[1] = value;
43         }
44 #endif
45     }
46 }
```

Program należy skompilować i zweryfikować jego działanie korzystając z oscyloskopu. Jeżeli wszystko jest w porządku należy zgłosić prowadzącemu i po zatwierdzeniu wykonać migawkę z komentarzem „***Zadanie 2.3.3 – synteza sygnału cyfrowego z użyciem tablicy***”.

3. Zadania dodatkowe z zakresu cyfrowej syntezy sygnału

Zaproponować funkcję która będzie przeliczać wartość amplitudy wyrażoną w

3.1. Synteza sygnału prostokątnego

3.1.1. Sygnał prostokątny o określonej częstotliwości i amplitudzie (2 pkt.)

Zaproponować zbiór funkcjonalności analogiczny jak dla generatora funkcji harmonicznych, który będzie generował sygnał prostokątny o stałym wypełnianiu równym 50 %.

Należy skompilować, przetestować i przedstawić prowadzącemu do oceny. Na zakończenie należy wykonać migawkę i opatrzyć ją komentarzem o następującej treści „***Zadanie 3.1.1 – sygnał prostokątny o określonej częstotliwości i amplitudzie***".

3.1.2. Sygnał prostokątny o określonej częstotliwości, amplitudzie i wypełnieniu (4 pkt.)

Rozszerzyć funkcjonalność poprzedzającego zadnia tak aby uwzględnić możliwość konfiguracji wypełnienia w zakresie od <0 do 1>.

Należy skompilować, przetestować i przedstawić prowadzącemu do oceny. Na zakończenie należy wykonać migawkę i opatrzyć ją komentarzem o następującej treści „***Zadanie 3.1.2 – sygnał prostokątny o określonej częstotliwości, amplitudzie i wypełnieniu***".

3.2. Synteza sygnału trójkątnego

3.2.1. Sygnał prostokątny o określonej częstotliwości i amplitudzie (2 pkt.)

Zaproponować zbiór funkcjonalności analogiczny jak dla generatora funkcji harmonicznych, który będzie generował sygnał trójkątny o równomiernej odległości między wierzchołkami.

Należy skompilować, przetestować i przedstawić prowadzącemu do oceny. Na zakończenie należy wykonać migawkę i opatrzyć ją komentarzem o następującej treści „***Zadanie 3.2.1 – sygnał trójkątny o określonej częstotliwości i amplitudzie***".

3.2.2. Sygnał trójkątny o określonej częstotliwości, amplitudzie i odległości między wierzchołkami (4 pkt.)

Rozszerzyć funkcjonalność poprzedzającego zadnia tak aby uwzględnić możliwość konfiguracji odległości między wierzchołkami w zakresie od <0 do 1> okresu sygnału.

Należy skompilować, przetestować i przedstawić prowadzącemu do oceny. Na zakończenie należy wykonać migawkę i opatrzyć ją komentarzem o następującej treści „***Zadanie 3.2.2 – sygnał trójkątny o określonej częstotliwości, amplitudzie i wypełnieniu***".

3.3. Synteza sygnału harmonicznego z użyciem tabeli z redukcją pamięci (8 pkt.)

W oparciu o rozwiązanie przedstawione w zadaniu 2.3.3 zaproponować modyfikację przygotowanych funkcji tak aby korzystać z pierwszej ćwiartki okresu wartości funkcji sinus, tj. dla kąta <0; $\pi/2$>.

Należy skompilować, przetestować i przedstawić prowadzącemu do oceny. Na zakończenie należy wykonać migawkę i opatrzyć ją komentarzem o następującej treści „***Zadanie 3.3 – sygnał sygnału harmonicznego z użyciem tabeli z redukcją pamięci***”.