

**Systemy mikroprocesorowe**

# **Instrukcja laboratoryjna**

## **Spotkanie 5**

**Cyfrowe przetwarzanie sygnału**

Autorzy: dr inż. Paweł Dąbal

Ostatnia aktualizacja: 17.12.2021 r.

Wersja: 1.0.0

## Spis treści

1.	Wprowadzenie.....	3
1.1.	Cel ćwiczenia .....	3
1.2.	Wymagania wstępne.....	3
1.3.	Opis stanowiska laboratoryjnego.....	3
1.4.	Wprowadzenie teoretyczne .....	3
1.4.1.	Cyfrowa synteza sygnału .....	4
1.4.2.	Cyfrowa filtracja sygnału .....	4
1.5.	Sposób realizacji ćwiczenia laboratoryjnego.....	4
2.	Zadania podstawowe do realizacji (12 pkt.) .....	6
2.1.	Zagadnienia wstępne .....	6
2.1.1.	Dołączenie do wirtualnej klasy i utworzenie repozytorium bazowego dla zadania .....	6
2.1.2.	Pobranie repozytorium i konfiguracja lokalna .....	6
2.1.3.	Konfiguracja środowiska STM32CubeIDE .....	7
2.1.4.	Opis projektu bazowego.....	8
2.2.	Cyfrowa synteza sygnału harmonicznego (6 pkt.).....	8
2.2.1.	Synteza sygnału harmonicznego o stałej amplitudzie i częstotliwości (3 pkt.) .....	8
2.2.2.	Synteza sygnału harmonicznego o stałej amplitudzie i zmiennej częstotliwości (3 pkt.).....	9
2.3.	Cyfrowa filtracja sygnału harmonicznego (6 pkt.).....	11
2.3.1.	Implementacja filtru FIR o średniej kroczącej (3 pkt.) .....	11
2.3.2.	Implementacja filtru FIR o kształtowanej charakterystyce (3 pkt.) .....	12
3.	Zadania rozszerzające do realizacji (14 pkt.).....	14
3.1.1.	Implementacja modulacji amplitudy sygnałem harmonicznym (3 pkt.) .....	14
3.1.2.	Modulacja częstotliwości sygnałem harmonicznym (4 pkt.) .....	14
3.1.3.	Projekt filtru pasmowo zaporowego na bazie filtru dolnoprzepustowego (7 pkt.).....	14

# 1. Wprowadzenie

Instrukcja ta zawiera zadania związane z tworzeniem programów realizujących przetwarzanie cyfrowego sygnału pochodzącego z akcelerometru i mikrofonu. W trakcie zajęć student będzie korzystał z środowiska programistycznego [STM32CubeIDE](#), rozbuduje dostarczony projekt aplikacji dla mikrokontrolera STM32L496ZGT6 umieszczonego na płycie uruchomieniowej *KAmLeon* o funkcjonalność umożliwiającą przetworzenie sygnału i jego przesłanie do komputera z użyciem interfejsu szeregowego. Ponadto będzie potrafił obsługiwać narzędzia wspomagające kontrolę wersji oprogramowania w celu dokumentowania własnych postępów.

## 1.1. Cel ćwiczenia

Ćwiczenie laboratoryjne ma na celu:

- nabycie umiejętności konfiguracji stanowiska pracy w oparciu o oprogramowanie [STM32CubeIDE 1.7.0](#);
- poznanie płyty uruchomieniowej [KAmLeon](#) z mikrokontrolerem [STM32L496ZGT6](#);
- przypomnienie i usystematyzowanie wiedzy i umiejętności z zakresu posługiwania się językiem C;
- przypomnienie wiedzy z zakresu podstaw cyfrowego przetwarzania sygnału;
- praktyczne korzystanie z systemu kontroli wersji [Git](#) i serwisu [GitHub](#).

## 1.2. Wymagania wstępne

Przed przystąpieniem do wykonywania ćwiczenia laboratoryjnego należy:

- zapoznać się z schematem płyty uruchomieniowej [KAmLeon](#) oraz dokumentacją mikrokontrolera STM32L496ZGT6 ([Product Specifications](#), [Reference Manuals](#));
- zapoznać się z składnią języka C – pojęcie zmiennej, stałej, funkcji, prototypu funkcji, parametru funkcji, wyrażenia warunkowego, pętli, wskaźnik, struktury i tablicy;
- zapoznanie się z sposobem implementacji operacji splotu;
- utworzyć konto na platformie [GitHub](#);
- zapoznać się z dokumentacją środowiska [STM32CubeIDE](#) i biblioteki [STM32CubeL4](#).

## 1.3. Opis stanowiska laboratoryjnego

Stanowisko do przeprowadzenia zajęć składa się z komputera PC z zainstalowanym oprogramowaniem koniecznym do realizacji zajęć: *STM32CubeIDE*, dedykowaną do układu biblioteką *STM32CubeL4* i systemem *Git* oraz płyty *KAmLeon* podłączonej do komputera za pomocą przewodu USB do złącza programatora *SWD PRG/DBG/vCOM* płyty uruchomieniowej. Połączenie to również odpowiada za zasilanie płyty. Ponadto w trakcie zajęć użyty może być oscyloskop cyfrowy.

## 1.4. Wprowadzenie teoretyczne

Współczesne mikrokontrolery charakteryzują się dużą wydajnością obliczeniową, która wynika z zaawansowanej architektury rdzeni obliczeniowych, np. z rdzeniem ARM Cortex-M4. Jedną z podstawowych operacji arytmetycznych związana z cyfrowym przetwarzaniem sygnału to mnożenie i dodawanie używane w implementacji operacji splotu. Obliczenia mogą być wykonywane wydajnie zarówno z użyciem liczb całkowitych oraz zmiennoprzecinkowych.

Jednym z sygnałów, który najczęściej towarzyszy człowiekowi jest sygnał akustyczny w postaci podłużnej fali mechanicznej. Z użyciem mikrofonów sygnał ten przetworzony jest na sygnał elektryczny analogowy lub cyfrowy. Dalej może być przetworzony pod kątem dodania efektów akustycznych bądź poprawieniem brzmienia zarejestrowanego sygnału. Znając charakter sygnałów akustycznych możemy je również przetwarzać na podstawie zapisu matematycznego budując w ten sposób elektroniczne instrumenty o wyjątkowym brzmieniu. Kolejnym sygnałem, który możemy mierzyć jest przyspieszenie, które towarzyszy nam cały czas. Pomiar przyspieszenia dzięki nowoczesnym czujnikom scalonym umożliwił powstanie wielu nowych urządzeń związanych branżą ochrony zdrowia i fitness. Analiza sygnału umożliwia wykrywanie aktywności osoby takich jak pozycja ciała, zliczanie wykonanych kroków, itd.

#### 1.4.1. Cyfrowa synteza sygnału

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,  $\varphi_0$  – faza początkowa,  $A$  – amplituda sygnału,  $A_0$  – składowa stała amplitudy. Wyznaczenie wartości funkcji sinus może być zrealizowane za pomocą rozwinięcia w szereg Taylora; 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.

#### 1.4.2. Cyfrowa filtracja sygnału

Proces filtracji sygnału polega na wyznaczeniu jego splotu z odpowiedzią impulsową filtru. W przypadku sygnałów cyfrowych jest to realizacja równania różnicowego o stałych współczynnikach składającego się z szeregu operacji mnożenia i dodawania:

$$y(n) = \sum_{k=0}^{K-1} \beta_k x(n-k) - \sum_{l=1}^{L-1} \alpha_l y(n-l)$$

gdzie:  $\alpha$  – współczynniki dodatniego sprzężenia zwrotnego,  $\beta$  – współczynniki odpowiedzi impulsowej. Jeżeli współczynniki  $\alpha$  są równe 0 wtedy mamy do czynienia z filtrem o skończonej odpowiedzi impulsowej FIR. W przeciwnym razie jest to filtr o nieskończonej odpowiedzi impulsowej IIR. Do wyznaczenia współczynników może posłużyć oprogramowanie MatLAB i jego narzędzie *filterDesigner*. Obliczenia możemy realizować z użyciem liczb całkowitych lub zmiennoprzecinkowych. W przypadku operacji na liczbach całkowitych możemy uzyskać dużą wydajność jednakże należy zwrócić szczególną uwagę na dokładność, która cierpi na niską dynamikę. Liczby zmiennoprzecinkowe pozwalają na dokładne obliczenia, kosztem jednak wydajności.

### 1.5. Sposób realizacji ćwiczenia laboratoryjnego

Każde zajęcia składać będą się z następujących elementów:

- zadań obowiązkowych - do wykonania krok po kroku na podstawie instrukcji – do uzyskania od 0 do 12 pkt.;
- zadań uzupełniających – do samodzielnego zbudowania i napisania programu – do uzyskania od 0 do 14 pkt;
- pytań sprawdzających rozumienie działania programu – zadawane przez prowadzącego przyjmującego wykonanie zadania – odpowiedź wpływa na punktację z zadań obowiązkowych i uzupełniających, tzn. czy przyznać maksymalną możliwą liczbę punktów za zadanie czy tylko część przy ewidentnym braku zrozumienia problemu.

Po zrealizowaniu każdego z zadań należy poprosić prowadzącego o sprawdzenie i przyznanie punktów. Na koniec zajęć wystawiana jest ocena na podstawie sumy uzyskanych punktów: **2,0** <0; 10), **3,0** <10; 13), **3,5** <13; 16), **4,0** <16; 19), **4,5** <19; 23), **5,0** <23; 26>. W każdych zajęciach należy uczestniczyć. Przy każdym zadaniu określona jest liczba punktów jakie można uzyskać. W przypadku zadań uzupełniających premiowana jest **jakość** i **czas realizacji** zaprezentowanego rozwiązania. Ocena końcowa z laboratorium to średnia arytmetyczna ocen z każdego spotkania.

## 2. Zadania podstawowe do realizacji (12 pkt.)

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

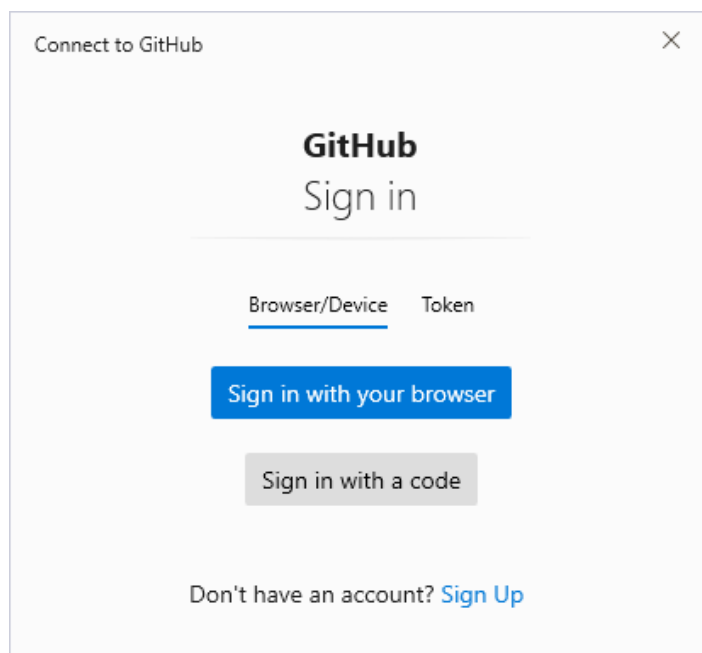
Przed przystąpieniem do pracy należy skonfigurować zainstalowane oprogramowanie i uzyskać dostęp do repozytorium, tzn. system kontroli wersji *Git*, instalacji biblioteki do obsługi mikrokontrolerów rodziny STM32L4 w środowisku *STM32CubeIDE* oraz sklonować repozytorium dla zajęć.

#### 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 prowadzącego do zadania, po otwarciu którego tworzone jest zadanie – indywidualne repozytorium. 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. Na pierwszych zajęciach jest to wersja minimalna, a na kolejnych będzie zawierała już wstępnie skonfigurowane projekty. 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*, który uruchomi skrypt wiersza poleceń, w którym należy podać: 1) swoje imię i nazwisko, 2) adres e-mail, 3) symbol grupy, 4) numer ćwiczenia, 5) adres indywidualnego repozytorium uzyskany w wcześniejszym punkcie. Po wykonaniu punktu 5) 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. W kolejnym pytaniu można uruchomić środowisko programistyczne *STM32CubeIDE*. Po jego uruchomieniu nie należy zamykać okna wiersza poleceń. Przykładowy przebieg wykonania skryptu zamieszczony został poniżej.



```

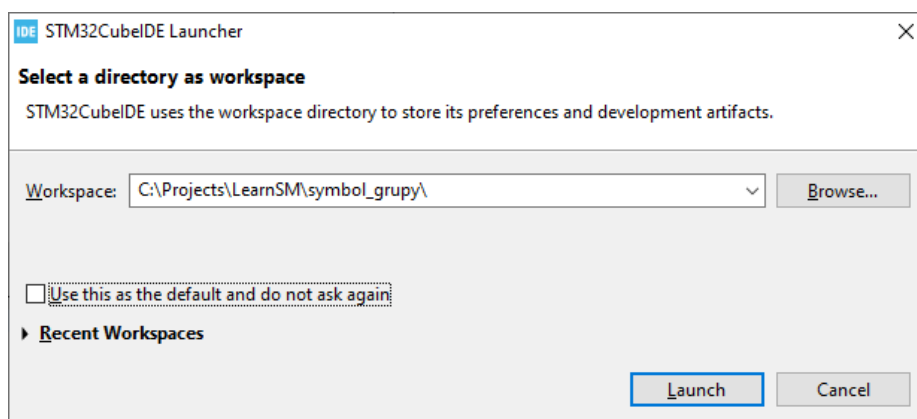
Windows PowerShell
Konfiguracja systemu Git dla zajęć laboratoryjnych Systemy Mikroprocesorowe
Proszę wprowadzić imię i nazwisko: Paweł Dabał
Proszę wprowadzić adres e-mail (@student.wat.edu.pl): pawel.dabal@student.wat.edu.pl
Proszę podać pełny symbol grupy (np. WEL19EL1S0): WEL19ExySz
Podaj numer ćwiczenia: 1
Podaj adres indywidualnego repozytorium: https://github.com/ztc-wel-wat/sm-lab-1-gpio-exti-pdabal

CMDKEY: Nie można odnaleźć elementu.
Cloning into 'C:\Projects\LabSM\WEL19EXYSZ\Lab-1'...
info: please complete authentication in your browser...
remote: Enumerating objects: 32, done.
remote: Counting objects: 100% (32/32), done.
remote: Compressing objects: 100% (27/27), done.
remote: Total 32 (delta 4), reused 28 (delta 1), pack-reused 0
Receiving objects: 100% (32/32), 215.44 KiB | 1.89 MiB/s, done.
Resolving deltas: 100% (4/4), done.
Ustawiona nazwa użytkownika:
Paweł Dabał
Ustawiony adres e-mail:
pawel.dabal@student.wat.edu.pl
Aby uruchomić STM32CubeIDE naciśnij Enter:

```

### 2.1.3. Konfiguracja środowiska STM32CubeIDE

Po uruchomieniu środowiska programistycznego *STM32CubeIDE* pojawi się pytanie o wskazanie katalogu, który będzie pełnił rolę przestrzeni roboczej. Należy wskazać na katalog skopiowanego w poprzednim kroku repozytorium (w przykładzie: *C:\Projects\LabSM\symbol\_grupy*). Wybór katalogu zatwierdzamy przyciskiem *Launch*. Można używać wielu różnych przestrzeni roboczych w przypadku pracy z różnymi projektami. Po załadowaniu środowiska należy zamknąć zakładkę *Information Center*.



W kolejnym kroku należy zaimportować projekt do przestrzeni roboczej poprzez wybranie z menu *File* → *Import*, a następnie w oknie, które się pojawi wybrać *General* → *Existing Projects into Workspace*. Zatwierdzić przyciskiem *Next*. Za pomocą przycisku *Browse...* wybrać katalog repozytorium (np. *C:\Projects\LabSM\symbol\_grupy*). Zaktualizuje się lista dostępnych projektów i na niej należy wybrać *BasicTimersAndConverters* i zatwierdzić przyciskiem *Finish*.

W celu zarządzania repozytorium dla zadania należy w środowisku *STM32CubeIDE* należy otworzyć widok (ang. *perspective*) zarządzania repozytorium. W tym celu z menu wybieramy: *Window* → *Perspective* → *Open Perspective* → *Other...*, gdzie z listy należy wybrać *Git*. Nastąpi przełączenie widoku i po lewej stronie pojawi się zakładka *Git Repositories* z listą dostępnych repozytoriów. Proszę zweryfikować obecność pobranego repozytorium, a w przypadku braku zgłosić prowadzącemu zajęcia.

W trakcie zajęć ograniczymy się do prostej liniowej struktury migawek wykonywanych po zakończeniu każdego z zadań instrukcji opatrzonej 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*.

### 2.1.4. Opis projektu bazowego

W sklonowanym repozytorium znajduje się projekt bazowy korzystający z biblioteki HAL o nazwie *BasicDSP*. Mikrokontroler został skonfigurowany tak, że taktowany jest częstotliwością 80 MHz. Wyprowadzenia mikrokontrolera do których podłączone są komponenty zamieszczone na płycie *KAMeLeon* zostały skonfigurowane zgodnie z ich przeznaczeniem (diody LED0..7, dioda LED\_RGB, joystick, wyświetlacz tekstowy LCD oraz 7-segmentowy, sensory LM75 i LSM303C wraz z konfiguracją magistrali, przetworniki ADC i DAC).

Posiada on skonfigurowane wyprowadzenia mikrokontrolera do których dołączone są diody LED (*LED0 ... LED7*), pięć styków joystick w trybie cyfrowego wejścia (*SW\_RIGHT, SW\_LEFT, SW\_DOWN, SW\_UP, SW\_OK*), wyświetlacza 7-segmentowego, wyświetlacza tekstowego, interfejsy *LPUART1, I2C1* dla czujnika temperatury *LM75* i *I2C3* dla czujnika przyspieszenia *LSM303C*. W pliku *gpio.c* zdefiniowane zostały podstawowe funkcje do obsługi diod LED oraz odczytu stanu joysticka. Obsługa wyświetlacza 7-segmentowego oraz tekstowego LCD zamieszczona została w plikach *display.c, lcd.c*, a sensorów w pliku *sensor.c*. Ponadto w pliku *dsp.c* zamieszczone zostały podstawowe funkcje imitujące komunikację z kodekiem audio: *DSP\_GetSample* oraz *DSP\_SetSample*.

## 2.2. Cyfrowa synteza sygnału harmonicznego (6 pkt.)

W celu zobrazowania syntezywanego sygnału należy użyć programu *SerialPlot* oraz przygotowanej konfiguracji z pliku *Lab-5.ini* znajdującego się na dysku sieciowym oraz w katalogu repozytorium. Mikrokontroler przesyła z częstotliwością 8 kHz po dwie liczby 8-bitowe ze znakiem, które reprezentują sygnał stereofoniczny.

### 2.2.1. Synteza sygnału harmonicznego o stałej amplitudzie i częstotliwości (3 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 sekcji kodu pomiędzy znacznikami */\* USER CODE BEGIN PTD \*/* a */\* USER CODE END PTD \*/* jak to przedstawione poniżej:

```
35 /* USER CODE BEGIN PTD */
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 pomiędzy znaczniki */\* USER CODE BEGIN 0 \*/* a */\* USER CODE END 0 \*/* jak to przedstawione poniżej.

```
90 /* USER CODE BEGIN 0 */
91 void OSC_Init(OSC_Cfg_t *hOscCfg, float amplitude, float frequency) {
92     frequency = fmodf(frequency, (DSP_CODEC_Fs / 2.0f));
93     hOscCfg->amplitude = fmodf(amplitude, (DSP_CODEC_Vpp * 0.5f));
94     hOscCfg->phaseStep = (frequency * DSP_2PI_DIV_FS);
95     hOscCfg->phase = 0.0f;
96 }
```

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. Pomędzy znaczniki `/* USER CODE BEGIN 0 */` a `/* USER CODE END 0 */` należy dodać kod wspomnianych funkcji jak to przedstawione poniżej.

```

98 float OSC_GetValue(OSC_Cfg_t *hOscCfg) {
99     float wave;
100
101     wave = hOscCfg->amplitude * sinf(hOscCfg->phase);
102     hOscCfg->phase += hOscCfg->phaseStep;
103     hOscCfg->phase = fmodf(hOscCfg->phase, DSP_PI_MUL_2);
104
105     return wave;
106 }
107
108 void OSC_SetFrequency(OSC_Cfg_t *hOscCfg, float frequency) {
109     frequency = fmodf(frequency, (DSP_CODEC_Fs / 2.0f));
110     hOscCfg->phaseStep = (frequency * DSP_2PI_DIV_FS );
111 }
112
113 void OSC_SetAmplitude(OSC_Cfg_t *hOscCfg, float amplitude) {
114     hOscCfg->amplitude = fmodf(amplitude, (DSP_CODEC_Vpp * 0.5f));
115 }

```

W sekcji kodu znajdującej się pomiędzy znacznikami `/* USER CODE BEGIN PV */` a `/* USER CODE END PV */` należy zdefiniować zmienną **Osc1**, która posłuży do przechowywania informacji o oscylatorze.

```

72 /* USER CODE BEGIN PV */
73 OSC_Cfg_t Osc1;

```

Następnie w funkcji *main* należy zmodyfikować kod pomiędzy znacznikami `/* USER CODE BEGIN WHILE */` a `/* USER CODE END WHILE */` jak to przedstawione poniżej.

```

227 /* USER CODE BEGIN WHILE */
228 float sample;
229 channel_t data;
230 OSC_Init(&Osc1, 1.0f, 100.0f);
231 while (1) {
232     if (DataNew == SET) {
233         // Wyznaczenie nowej próbki
234         sample = OSC_GetValue(&Osc1);
235         // Ustawienie poziomu na podstawie potencjometru
236         sample = DSP_SetVolumePOT(sample);
237         // Zamiana na liczbę całkowitą
238         data = DSP_GetChannelSampleFromValue(sample);
239         // Skopiowanie sygnału wejściowego z mikrofonu
240         DataOut = DataIn;
241         // Modyfikacja zawartości kanału 0
242         DataOut.channel[0] = data;
243         // Skasowanie flagi przetwarzania
244         DataNew = RESET;
245     }
246     /* USER CODE END WHILE */

```

Przygotowany program należy skompilować, uruchomić i przetestować. Działanie programu przedstawić prowadzącemu zajęcia do oceny. Po zatwierdzeniu wykonać migawkę z komentarzem „**Zadanie 2.2.1 – sygnał harmoniczny o stałej amplitudzie i częstotliwości**”.

### 2.2.2. Synteza sygnału harmonicznego o stałej amplitudzie i zmiennej częstotliwości (3 pkt.)

Jedną z podstawowych funkcjonalności cyfrowych generatorów arbitralnych jest praca w trybie przemiatania częstotliwości (ang. *sweep*). Generator taki umożliwia określenie przedziału częstotliwości oraz czasu w jakim ma nastąpić przejście między krańcowymi częstotliwościami. W związku z czym do przechowywania tych informacji należy zdefiniować strukturę ***SWEEP\_Cfg\_t*** w sekcji kodu pomiędzy znacznikami `/* USER CODE BEGIN PTD */` a `/* USER CODE END PTD */` jak to przedstawione poniżej:

```
42 typedef struct {
43     float duration;
44     float phase;
45     float phaseStep;
46     float phaseStepStart;
47     float phaseStepStop;
48     float phaseInc;
49     OSC_Cfg_t *generator;
50 } SWEEP_Cfg_t;
```

Konfiguracja pól struktury powinna zostać zrealizowana za pośrednictwem dedykowanej funkcji (***SWEEP\_Init***), która sprawdzi parametry i wyznaczy aktualny krok fazy (***phaseStep***), zmianę kroku fazy (***phaseInc***) na podstawie częstotliwości początkowej i końcowej oraz czasu przechodzenia (***duration***) podanych w parametrze. Należy dodać funkcję konfiguracji oscylatora pomiędzy znaczniki `/* USER CODE BEGIN 0 */` a `/* USER CODE END 0 */` jak to przedstawione poniżej.

```
126 void SWEEP_Init(SWEEP_Cfg_t *hCfg, OSC_Cfg_t *hGen, float freqStart,
127     float freqStop, float duration) {
128     hCfg->duration = duration;
129     hCfg->phaseStepStart = (freqStart * DSP_2PI_DIV_FS );
130     hCfg->phaseStepStop = (freqStop * DSP_2PI_DIV_FS );
131     hCfg->phaseStep = hCfg->phaseStepStart;
132     hCfg->phaseInc = ((hCfg->phaseStepStop - hCfg->phaseStepStart)
133         / (duration * DSP_CODEEC_Fs ));
134     hCfg->generator = hGen;
135     hCfg->generator->phaseStep = (freqStart * DSP_2PI_DIV_FS );
136 }
```

Pomiędzy znaczniki `/* USER CODE BEGIN 0 */` a `/* USER CODE END 0 */` należy dodać kod funkcji ***SWEEP\_GetValue*** odpowiedzialnej za wyznaczanie nowej próbki, jak to przedstawione poniżej.

```
138 float SWEEP_GetValue(SWEEP_Cfg_t *hCfg) {
139     float wave = OSC_GetValue(hCfg->generator);
140     hCfg->phaseStep += hCfg->phaseInc;
141     if (hCfg->phaseStep > hCfg->phaseStepStop)
142         hCfg->phaseStep -= (hCfg->phaseStepStop - hCfg->phaseStepStart);
143     hCfg->generator->phaseStep = hCfg->phaseStep;
144
145     return wave;
146 }
```

W sekcji kodu znajdującej się pomiędzy znacznikami `/* USER CODE BEGIN PV */` a `/* USER CODE END PV */` należy zdefiniować zmienną ***Sweep1***, która posłuży do przechowywania informacji o oscylatorze.

```
72 /* USER CODE BEGIN PV */
73 SWEEP_Cfg_t Sweep1;
```

Następnie w funkcji *main* należy zmodyfikować kod pomiędzy znacznikami `/* USER CODE BEGIN WHILE */` a `/* USER CODE END WHILE */` jak to przedstawione poniżej.

```

226  /* USER CODE BEGIN WHILE */
227  float sample;
228  channel_t data;
229  OSC_Init(&Osc1, 1.0f, 100.0f);
230  SWEEP_Init(&Sweep1, &Osc1, 1.0f, DSP_CODEC_Fs / 4, 1.0f);
231  while (1) {
232      if (DataNew == SET) {
233          // Wyznaczenie nowej próbki
234          sample = SWEEP_GetValue(&Sweep1);
235          // Ustawienie poziomu na podstawie potencjometru
236          sample = DSP_SetVolumePOT(sample);
237          // Zamiana na liczbę całkowitą
238          data = DSP_GetChannelSampleFromValue(sample);
239          // Skopiowanie sygnału wejściowego z mikrofonu
240          DataOut = DataIn;
241          // Modyfikacja zawartości kanału 0
242          DataOut.channel[0] = data;
243          // Skasowanie flagi przetwarzania
244          DataNew = RESET;
245      }
246  /* USER CODE END WHILE */

```

Przygotowany program należy skompilować, uruchomić i przetestować. Działanie programu przedstawić prowadzącemu zajęcia do oceny. Po zatwierdzeniu wykonać migawkę z komentarzem „**Zadanie 2.2.2 – sygnał harmoniczny o stałej amplitudzie i zmiennej częstotliwości**”.

## 2.3. Cyfrowa filtracja sygnału harmonicznego (6 pkt.)

Cyfrowa filtracja sygnału przedstawiona zostanie na przykładzie implementacji filtra FIR. Do przechowania konfiguracji filtra wymagane jest zapamiętanie jego rzędu ( $n$ ) bufora próbek ( $x$ ) oraz tablicy współczynników ( $h$ ). W związku z czym do przechowywania tych informacji należy zdefiniować strukturę **FIR\_Cfg\_t** w sekcji kodu pomiędzy znacznikami `/* USER CODE BEGIN PTD */` a `/* USER CODE END PTD */` jak to przedstawione poniżej:

```

52 typedef struct {
53     uint32_t n;
54     float *x;
55     float *h;
56 } FIR_Cfg_t;

```

Pomiędzy znaczniki `/* USER CODE BEGIN 0 */` a `/* USER CODE END 0 */` należy dodać kod funkcji **FIR\_GetValue** odpowiedzialnej za wyznaczanie nowej próbki wyjściowej filtra, jak to przedstawione poniżej.

```

160 float FIR_GetValue(FIR_Cfg_t *hCfg, float value) {
161     int32_t i;
162     float y = 0.0f;
163
164     hCfg->x[0] = value; // Dodanie próbki do bufora
165     for (i = 0; i < hCfg->n; i++) { // Suma iloczynów
166         y += (hCfg->h[i]) * (hCfg->x[i]);
167     }
168     for (i = (hCfg->n - 2); i >= 0; i--) { // Przesunięcie bufora
169         hCfg->x[i+1] = hCfg->x[i];
170     }
171
172     return y;
173 }

```

### 2.3.1. Implementacja filtra FIR o średniej kroczącej (3 pkt.)

Filtr o średniej kroczącej jest to rodzaj filtru dolnoprzepustowego wyznaczający średnią arytmetyczną  $N$  ostatnich próbek sygnału. Wszystkie współczynniki tego filtru są równe  $1/N$ , a jego rząd jest równy  $N$ . Konfiguracja pól struktury filtru powinna zostać zrealizowana za pośrednictwem dedykowanej funkcji (**FIR\_InitAvr**), która zarezerwuje pamięć dla buforów próbek i współczynników oraz wyznaczy współczynniki filtru ze średnią kroczącej. Należy dodać tę funkcję pomiędzy znaczniki `/* USER CODE BEGIN 0 */` a `/* USER CODE END 0 */` jak to przedstawione poniżej.

```
149 void FIR_InitAvr(FIR_Cfg_t *hCfg, uint32_t n) {
150     hCfg->n = n;
151     hCfg->x = (float*) malloc(n * sizeof(float));
152     hCfg->h = (float*) malloc(n * sizeof(float));
153
154     for (uint32_t i = 0; i < n; ++i) {
155         hCfg->x[i] = 0;
156         hCfg->h[i] = 1.0f / n;
157     }
158 }
```

W sekcji kodu znajdującej się pomiędzy znacznikami `/* USER CODE BEGIN PV */` a `/* USER CODE END PV */` należy zdefiniować zmienną **FirAvr**, która posłuży do przechowywania informacji o oscylatorze.

```
72 /* USER CODE BEGIN PV */
73 FIR_Cfg_t FirAvr;
```

Następnie w funkcji *main* należy zmodyfikować kod pomiędzy znacznikami `/* USER CODE BEGIN WHILE */` a `/* USER CODE END WHILE */` jak to przedstawione poniżej.

```
219 /* USER CODE BEGIN WHILE */
220 float sample;
221 channel_t data;
222 OSC_Init(&Osc1, 1.0f, 100.0f);
223 SWEEP_Init(&Sweep1, &Osc1, 1.0f, DSP_CODEC_Fs / 2, 0.5f);
224 FIR_InitAvr(&FirAvr, 11);
225 while (1) {
226     if (DataNew == SET) {
227         // Wyznaczenie nowej próbki
228         sample = SWEEP_GetValue(&Sweep1);
229         // Wyznaczenie próbki po przejściu przez filtr
230         sample = FIR_GetValue(&FirAvr, sample);
231         // Ustawienie poziomu na podstawie potencjometru
232         // Zamiana na liczbę całkowitą
233         data = DSP_GetChannelSampleFromValue(sample);
234         // Skopiowanie sygnału wejściowego z mikrofonu
235         DataOut = DataIn;
236         // Modyfikacja zawartości kanału 0
237         DataOut.channel[0] = data;
238         // Skasowanie flagi przetwarzania
239         DataNew = RESET;
240     }
241 /* USER CODE END WHILE */
```

Przygotowany program należy skompilować, uruchomić i przetestować. Działanie programu przedstawić prowadzącemu zajęcia do oceny. Po zatwierdzeniu wykonać migawkę z komentarzem „**Zadanie 2.3.1 – filtr o średniej kroczącej**”.

### 2.3.2. Implementacja filtru FIR o kształtowanej charakterystyce (3 pkt.)

Cyfrowa filtracja sygnału z użyciem filtru FIR o charakterystyce wyznaczonej na podstawie podanych parametrów wymaga dodania zmodyfikowanej postaci funkcji inicjującej strukturę filtru. Należy dodać tę

funkcję **FIR\_Init** pomiędzy znaczniki `/* USER CODE BEGIN 0 */` a `/* USER CODE END 0 */` jak to przedstawione poniżej.

```
159 void FIR_Init(FIR_Cfg_t *hCfg, uint32_t n, float * h) {
160     hCfg->n = n;
161     hCfg->x = (float*) malloc(n * sizeof(float));
162     hCfg->h = h;
163
164     for (uint32_t i = 0; i < n; ++i) {
165         hCfg->x[i] = 0;
166     }
167 }
```

Następnie w funkcji *main* należy zmodyfikować kod pomiędzy znacznikami `/* USER CODE BEGIN WHILE */` a `/* USER CODE END WHILE */` jak to przedstawione poniżej.

```
231 /* USER CODE BEGIN WHILE */
232 float sample;
233 channel_t data;
234 OSC_Init(&Osc1, 1.0f, 100.0f);
235 SWEEP_Init(&Sweep1, &Osc1, 1.0f, DSP_CODEC_Fs / 2, 0.25f);
236 FIR_Init(&FirAvr, (uint32_t)N, (float *)h);
237 while (1) {
238     if (DataNew == SET) {
239         // Wyznaczenie nowej próbki
240         sample = SWEEP_GetValue(&Sweep1);
241         // Wyznaczenie próbki po przejściu przez filtr
242         sample = FIR_GetValue(&FirAvr, sample);
243         // Ustawienie poziomu na podstawie potencjometru
244         // Zamiana na liczbę całkowitą
245         data = DSP_GetChannelSampleFromValue(sample);
246         // Skopiowanie sygnału wejściowego z mikrofonu
247         DataOut = DataIn;
248         // Modyfikacja zawartości kanału 0
249         DataOut.channel[0] = data;
250         // Skasowanie flagi przetwarzania
251         DataNew = RESET;
252     }
253 /* USER CODE END WHILE */
```

Przygotowany program należy skompilować, uruchomić i przetestować. Działanie programu przedstawić prowadzącemu zajęcia do oceny. Po zatwierdzeniu wykonać migawkę z komentarzem „**Zadanie 2.3.1 – filtr FIR o kształtowanej charakterystyce**”.

### 3. Zadania rozszerzające do realizacji (14 pkt.)

W rozdziale tym przedstawione zostały zadania dodatkowe, za realizację których można podnieść ocenę końcową za wykonane ćwiczenie. Ich realizację należy wykonać w dotychczasowym projekcie.

#### 3.1.1. Implementacja modulacji amplitudy sygnałem harmonicznym (3 pkt.)

Zadanie to polega na zademonstrowaniu modulacji amplitudy sygnału nośnego o częstotliwości  $f_0 = 500$  Hz i amplitudzie  $A_0 = 1,0$  V sygnałem modulującym  $f_m = 50$  Hz, amplitudzie  $A_m = 0,25$  V i składowej stałej  $A = 0.5$  V.

Przygotowany program należy skompilować, uruchomić i przetestować. Działanie programu przedstawić prowadzącemu zajęcia. Po zatwierdzeniu i ocenieniu wykonać migawkę z komentarzem „**Zadanie 3.1.1 – modulacji amplitudy sygnałem harmonicznym**”.

#### 3.1.2. Modulacja częstotliwości sygnałem harmonicznym (4 pkt.)

Zadanie to polega na zademonstrowaniu modulacji częstotliwości sygnału nośnego o częstotliwości  $f_0 = 500$  Hz i amplitudzie  $A_0 = 1,0$  V sygnałem modulującym  $f_m = 5$  Hz, amplitudzie  $A_m = 0,25$  V. Należy przyjąć dewiację częstotliwości  $k = \pm 100$  Hz.

Przygotowany program należy skompilować, uruchomić i przetestować. Działanie programu przedstawić prowadzącemu zajęcia. Po zatwierdzeniu i ocenieniu wykonać migawkę z komentarzem „**Zadanie 3.1.2 – modulacja częstotliwości sygnałem harmonicznym**”.

#### 3.1.3. Projekt filtru pasmowo zaporowego na bazie filtru dolnoprzepustowego (7 pkt.)

Korzystając z własności funkcji transmitancji filtru cyfrowego i kombinacji połączenia filtrów równoległe i szeregowo przygotować filtr pasmowozaporowy.

Przygotowany program należy skompilować, uruchomić i przetestować. Działanie programu przedstawić prowadzącemu zajęcia. Po zatwierdzeniu i ocenieniu wykonać migawkę z komentarzem „**Zadanie 3.1.3 – modulacja częstotliwości sygnałem harmonicznym**”.