Akademia Górniczo - Hutnicza im. Stanisława Staszica

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Katedra Automatyki i Inżynierii Biomedycznej

Kierunek: Automatyka i robotyka

Procesory Sygnałowe i Mikrokontrolery

Generator sygnału wzorcowego na podstawie zadanej charakterystyki częstotliwościowej

---

Wykonali:
Maciej Cebula
Piotr Merynda
Maciej Podsiadło

Prowadzący:
dr inż. Tomasz Dziwiński

1 Wstęp

Celem projektu było stworzenie generatora sygnału wzorcowego na podstawie zadanej charakterystyki częstotliwościowej. Do realizacji zadania postanowiono skorzystać ze sprzętu laboratoryjnego, na który składały się:

1. Komputer klasy PC,
2. Zestaw startowy z mikrokontrolerem z rodziny STM32 (STM32F401),
3. Oscyloskop.

Korzystano z oprogramowania znajdującego się na PC, na którym zainstalowany jest system operacyjny Windows, a w szczególności z:

1. STM32CubeMX - graficznego środowiska umożliwiającego szybką konfigurację układów peryferialnych wraz generację szablonu kodu w języku C,
2. SW4STM32 - środowiska umożliwiającego szybkie tworzenie oraz debugowanie napisanego kodu,
3. MATLAB - zintegrowanego środowiska programistycznego, w którym stworzono prototyp algorytmu oraz interfejs u użytkownika.

Zdecydowano, że docelowo użytkownik zadawał będzie punkty charakterystyki z poziomu aplikacji napisanej w MATLABie. Informacje te przesłane zostaną do mikrokontrolera poprzez interfejs szeregowy zgodnie ze standardem RS-232. Mikrokontroler wygeneruje odpowiedni sygnał wzorcowy, a następnie wyśle go do przetwornika cyfrowo-analogowego. Sygnał wyjściowy możliwy będzie do obejrzenia na oscyloskopie.

2 Algorytm

2.1 Dyskretna transformacja Fouriera

Głównym narzędziem matematycznym, które wykorzystano do rozwiązania postawionego zadania była odwrotna transformacja Fouriera. Przesłana przez użytkownika charakterystyka częstotliwościowa reprezentowana będzie poprzez wektor próbek. Dodatkowo biorąc pod uwagę fizyczne ograniczenia, w tym niezerowy czas działania przetwornika postanowiono wykorzystać algorytm odwrotnej dyskretnej transformacji Fouriera (IDFT). Zakłada się, że przesyłana charakterystyka odpowiada sygnałowi rzeczywistemu oraz największa częstotliwość w niej występująca jest mniejsza od połowy częstotliwości próbkowania algorytmu.

Oznaczając prze N liczbę punktów charakterystyki częstotliwościowej zadanych przez użytkownika wprowadzono następujące oznaczenia:

$$freq = [f_1, f_2, ..., f_N]^T$$
$$amp = [A_1, A_2, ..., A_N]^T$$
$$phase = [\phi_1, \phi_2, ..., \phi_N]^T$$

gdzie $freq$ jest wektorem niezerowych częstotliwości składowych zadanego sygnału, natomiast amp oraz $phase$ odpowiadających im amplitud i przesunięć fazowych. Uwzględniając założenia:

$$f_p > \frac{max(freq)}{2} \tag{1}$$

można w sposób zwarty zapisać formułę:

$$x(k\Delta t) = \sum_{n=1}^{N} A_n \cdot \cos(2\pi f_n k\Delta t - \phi_n), \quad k = 0, 1, 2, ... \tag{2}$$

umożliwiającej wyliczenie kolejnych wartości sygnału spróbkowanego z częstotliwością $f_p = \frac{1}{\Delta t}$, którego charakterystyka częstotliwościowa została zadana.

2.2 Prototyp algorytmu

Przed przystąpieniem do implementacji algorytmu 2 na mikrokontrolerze stworzono jego prototyp w środowisku MATLAB. Z uwagi na postać 2 programowa realizacja algorytmu przebiegła sprawnie. Rezultaty przedstawione zostały w podrozdziale 3.1.

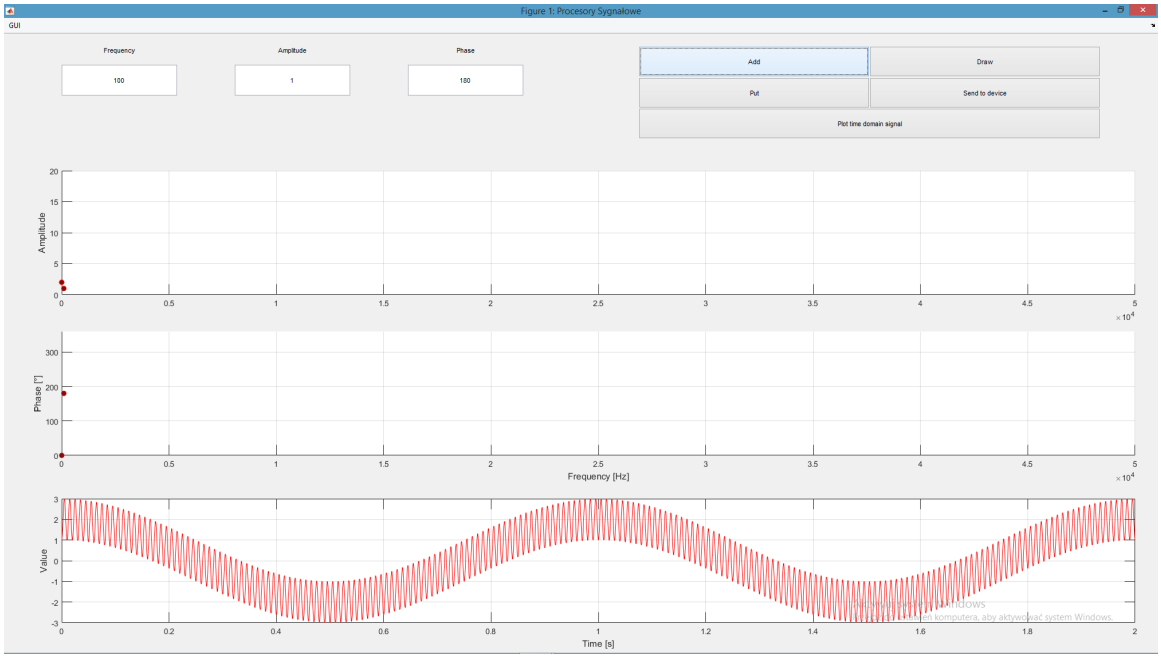
3 Implementacja

3.1 Interfejs użytkownika

Zgodnie z założeniami projektu zbudowano aplikację w środowisku MATLAB umożliwiającą użytkownikowi szybkie przesyłanie danych do mikrokontrolera. Zadbano o intuicyjność graficznego interfejsu użytkownika udostępniając następujące funkcjonalności:

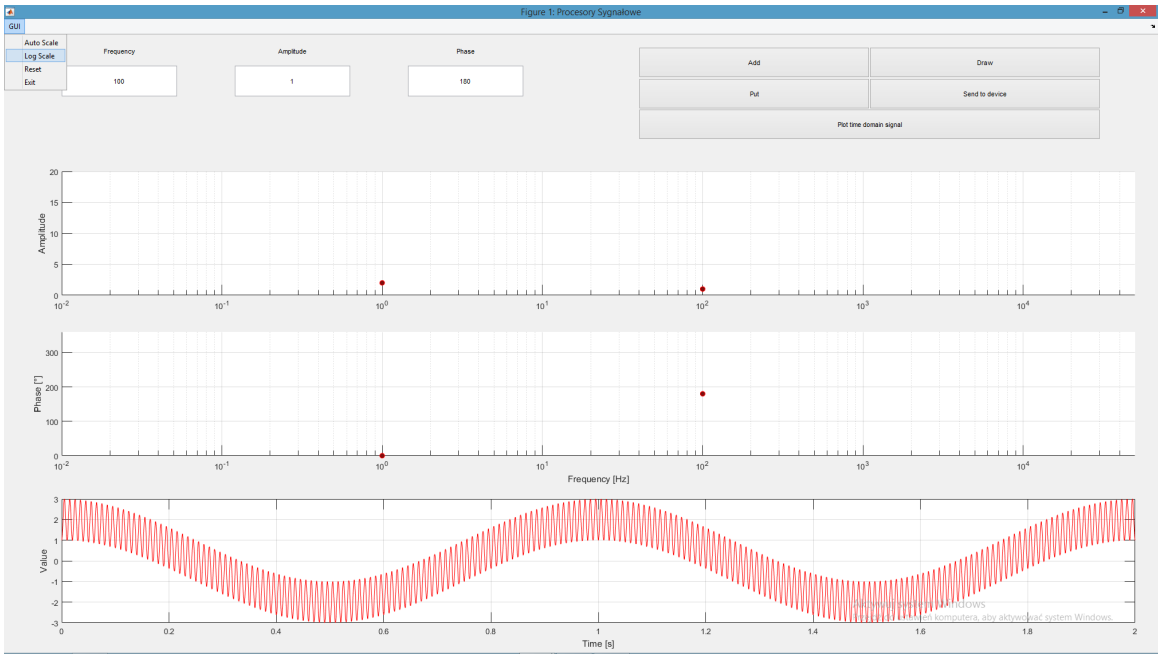
1. Ręczne wprowadzanie kolejnych punktów charakterystyki.
2. Automatyczne wprowadzanie częstotliwości po kliknięciu użytkownika na wykres.
3. Wykres amplitudy oraz przesunięcia fazowego dla wprowadzonych punktów charakterystyki częstotliwościowej.
4. Algorytm liczący sygnał w dziedzinie czasu na podstawie wprowadzonych przez użytkownika punktów (zgodnie z równaniem 2).
5. Rysowanie odpowiedzi w dziedzinie czasu *online*.
6. Automatyczne skalowanie wykresów (skala liniowa, logarytmiczna).
7. Możliwość wysłania wprowadzonej charakterystyki do mikrokontrolera (zgodnie z zaimplementowanym protokołem opisanym w podrozdziale 3.2).

Po wprowadzeniu dwóch próbek aplikacja prezentuje się następująco: Zakres wyświe-

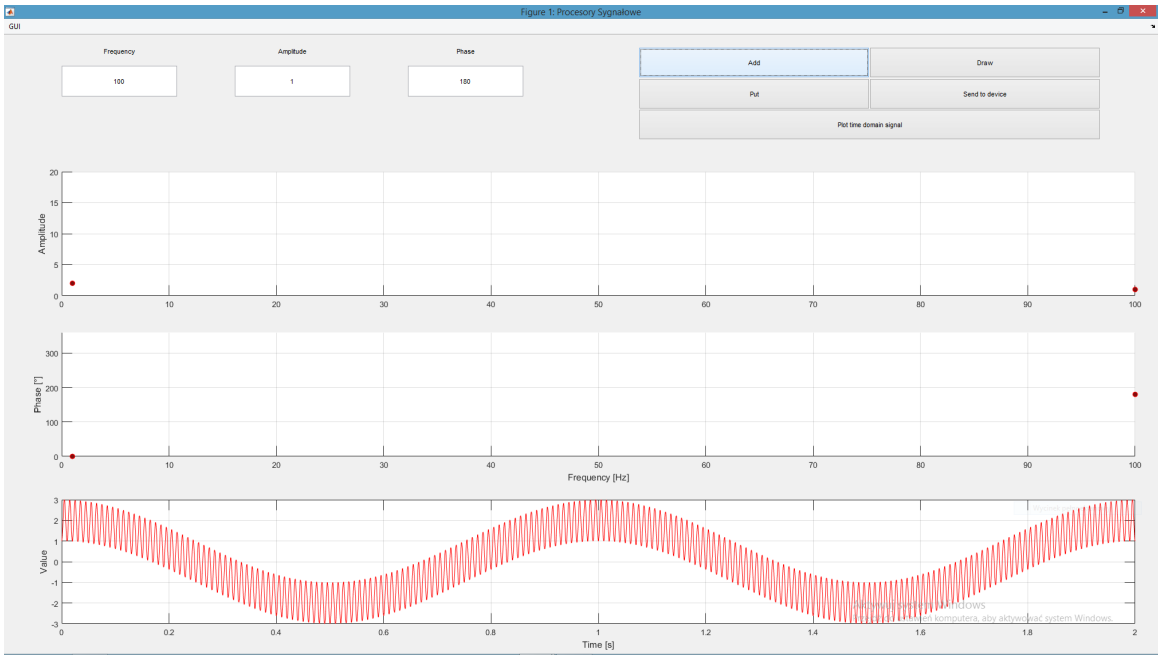


Rys. 1: Widok aplikacji po wprowadzeniu dwóch punktów na charakterystyce częstotliwościowej.

tlanych częstotliwości zgodny jest z założeniem opisanym równaniem 1. Po kliknięciu w zakładkę *GUI* w menu użytkownik może wybrać jedną z opcji skalowania wykresów,



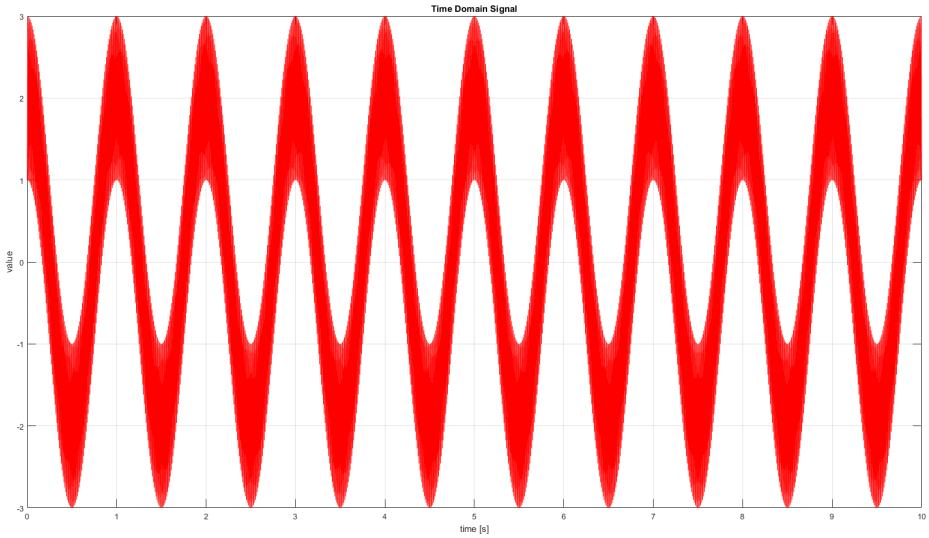
Rys. 2: Wyświetlenie wprowadzonej charakterystyki w skali logarytmicznej.



Rys. 3: Wyświetlenie charakterystyki w skali liniowej - automatycznie dopasowanej do wprowadzonych punktów.

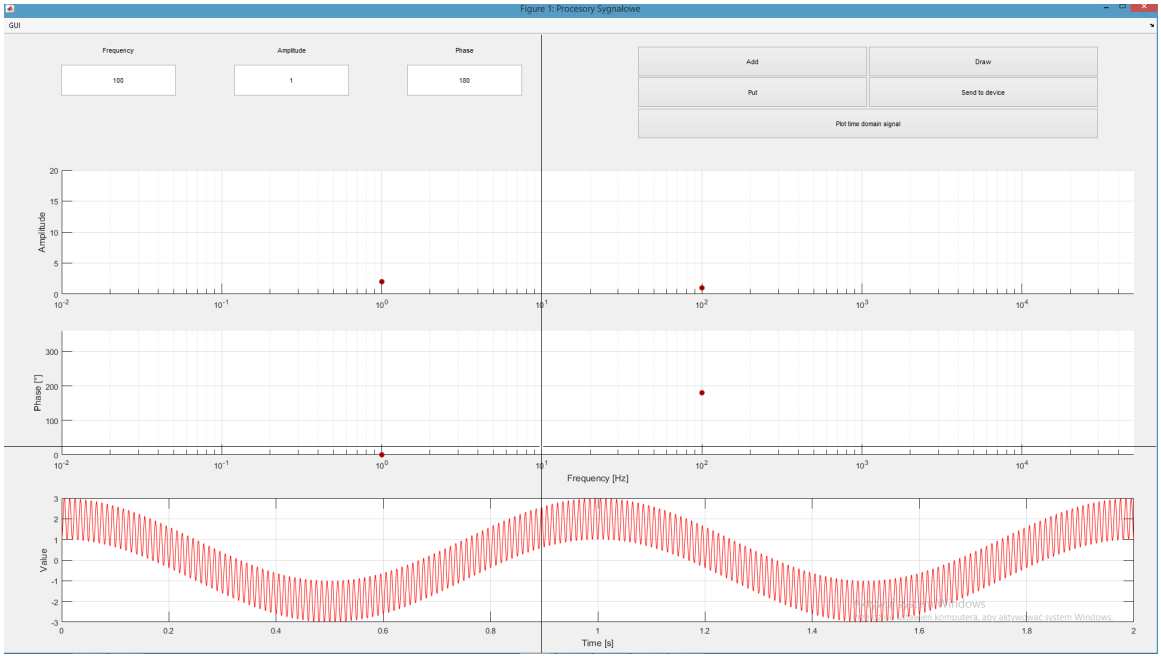
a także zresetować aktualnie wprowadzone punkty lub zakończyć działanie aplikacji w sposób zapewniający zwolnienie wszystkich jej zasobów. W każdej chwili możliwe jest wygenerowanie wykresu w nowym oknie w sposób umożliwiający łatwy zapis, a także porównanie z sygnałem wyliczonym prze mikrokontroler. Po naciśnięciu przycisku *Plot*

time domain signal pojawi się wykres 4. Użytkownik po wciśnięciu przycisku *Put* może



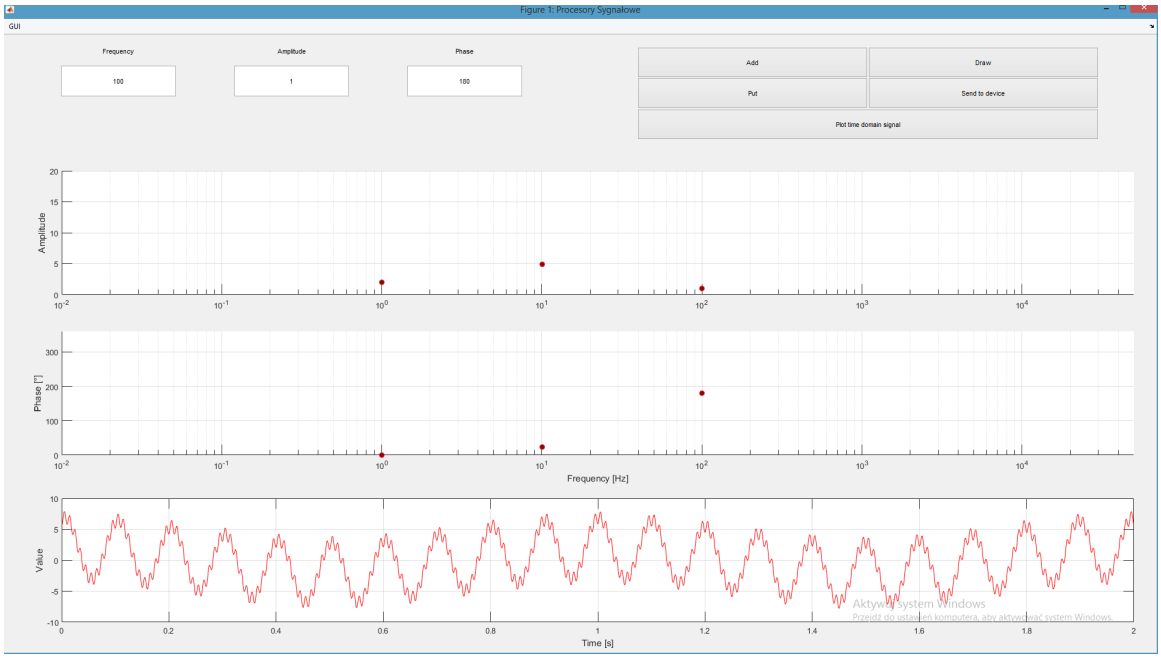
Rys. 4: Wykres w dziedzinie czasu.

ręcznie wprowadzić kolejny punkt charakterystyki poprzez najechanie kursorem w odpowiednie miejsce na wykresach - amplitudy oraz przesunięcia fazowego. Po wciśnięciu



Rys. 5: Wprowadzanie punktów do charakterystyki częstotliwościowej za pomocą kursora.

lewego przycisku myszy w sytuacji zobrazowanej na rysunku 5 główny ekran aplikacji prezentuje się następująco - rys. 6.



Rys. 6: Ekran aplikacji po wprowadzeniu punktu za pomocą kursora.

3.2 Protokół komunikacyjny

Dane z komputera do mikrokontrolera przesyłane są za pomocą protokołu *RS232*, którego parametry podano w tabeli 1. Specyfika implementowanego algorytmu wymagała zaproponowania jednolitego sposobu przesyłania danych opisujących zadaną charakterystykę.
Należało uwzględnić następujące czynniki:

1. możliwość przesyłania dowolnie wielu zestawów danych opisujących pojedynczy punk charakterystyki, tzn. częstotliwość, amplitudę i przesunięcie fazowe,
2. możliwość przesyłania nowego zestawu danych bez konieczności resetowania układu.
3. liczy opisujące poszczególne współczynniki mogą mieć różną liczbę cyfr.

Biorąc pod uwagę wszystkie powyższe wymagania oraz specyfikę protokołu *RS232* zdecydowano się przesyłać dane w postaci pojedynczych znaków. Procedura *składania* liczby z przesłanych znaków została zaimplementowana w samym urządzeniu. Aby algorytm mógł rozróżnić czy przesłany znak jest częścią obecnie składanej liczby czy dotyczy już następnego parametru, wprowadzono znak *tabulacji*, który oddziela cyfry kolejnych liczb.
Przyjęto, że pierwsza liczba przesłana do mikrokontrolera określa liczbę punktów charakterystyki. Następnie przesyłane są kolejne zestawy danych składające się z trzech liczb: częstotliwości, amplitudy i przesunięcia fazowego. Każda z tych trzech liczb zapisywana jest w odpowiedniej tablicy.
Do przechowywania poszczególnych wartości wykorzystywane są dwa zestawy dynamicznie alokowanych tablic. Pierwszy zestaw służy do przechowywania danych obecnie generowanego sygnału (wskaźniki freqTab, ampTab, phaseTab), natomiast drugi wykorzystywany jest do zapisywania aktualnie przesyłanego zestawu (wskaźniki freqTabRT,

ampTabRT, phaseTabRT).
Po przesłaniu całego zestawy danych wskaźniki pierwszego zestawu są podmieniane ze wskaźnikami zestawu drugiego (linie 47-66 w listingu 1). Dzięki zastosowaniu tego zabiegu uniknięto konieczności dynamicznego alokowania pamięci w przerwaniu oraz zaoszczędzono czas potrzebny na przepisywanie danych z jednego zestawu do drugiego. Kod od przerwania odebrania nowego znaku zamieszczony jest na listingu 1.

Listing 1: Implementacja protokołu komunikacji

```
1  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
2
3      static int i=0;
4      volatile static int sample_iter = -1;
5      static int liczbaProbek = -1;
6      volatile static int idTab = 0;
7      uint8_t data[50];
8      uint16_t size = 0;
9
10     if(Received == 9 || i==ReceivedTabSize) // zatwierdzenie liczby TABem
11     {
12         volatile int recNumber = 0;
13         for (int k = 0;k < i;k++)
14         {
15             recNumber += power(i-k-1) * ReceivedTab[k];
16         }
17         i = 0;
18
19         if(sample_iter == -1)
20         {
21             size = sprintf(data, "lp:\t %u \n\r", recNumber);
22             HAL_UART_Transmit_IT(&huart2, data, size);
23             liczbaProbek = recNumber;
24             MAX = liczbaProbek;
25             sample_iter = 0;
26         }
27         else
28         {
29             if(sample_iter%3 == 0) // freq
30             {
31                 freqTabRT[idTab] =  recNumber;
32                 freq = recNumber;
33             }
34             else if(sample_iter%3 == 1) // amp
35             {
36                 ampTabRT[idTab] = recNumber;
37             }
38             else if(sample_iter%3 == 2) // phase
39             {
40                 phaseTabRT[idTab] = recNumber;
41                 idTab++;
42             }
43             sample_iter += 1;
44             size = sprintf(data, "sampleIter:\t %d \n\r", sample_iter);
45             HAL_UART_Transmit_IT(&huart2, data, size);
46         }
47         if(idTab == liczbaProbek)
48         {
49             freqTabTemp = freqTab;
50             ampTabTemp = ampTab;
```

```
51              phaseTabTemp = phaseTab;
52
53              freqTab = freqTabRT;
54              ampTab = ampTabRT;
55              phaseTab = phaseTabRT;
56
57              freqTabRT = freqTabTemp;
58              ampTabRT = ampTabTemp;
59              phaseTabRT = phaseTabTemp;
60
61
62              size = sprintf(data, "koniec \n\r");
63              HAL_UART_Transmit_IT(&huart2, data, size);
64              sample_iter = -1;
65              idTab = 0;
66          }
67
68      }
69      else
70      {
71          ReceivedTab[i] = atoi(&Received);
72          vv = atoi(&Received);
73          i++;
74      }
75
76
77      HAL_UART_Receive_IT(&huart2, &Received, r_size);
78  }
```

3.3 Generacja sygnału wyjściowego

Sygnał wyjściowy generowany jest na podstawie algorytmu opisanego w rozdziale 2, cyklicznie w przerwaniu od timera nr 11. Kod wyliczający odwrotną transformatę przedstawiony jest listingu 2.

Listing 2: Kod funkcji obliczającej odwrotną transformatę Fouriera w przerwaniu od timera 11.

```
1  void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
2  {
3      static long long int t = 0;
4      int index = 0;
5
6      float v;
7
8      if(htim->Instance == TIM11){
9
10         HAL_DAC_SetValue(&hdac,DAC_CHANNEL_1,DAC_ALIGN_12B_R,output);
11         if(freqTab==NULL || ampTab==NULL ||phaseTab==NULL)
12         {
13             return;
14         }
15         out = 0;
16         t=t+1;
17         for (int ii=0;ii<MAX;ii++)
18         {
19             v = 360 * tp * freqTab[ii]*t + 180 - phaseTab[ii];
20             cos = cosApproximation(v);
```

```
21                 out = out + ampTab[ii]*cos;
22             }
23
24             // skalowanie amplitudy -50 do 50 na od 0 do 4095
25             output = (int)(out*40.96 + 2048);
26         }
27     }
```

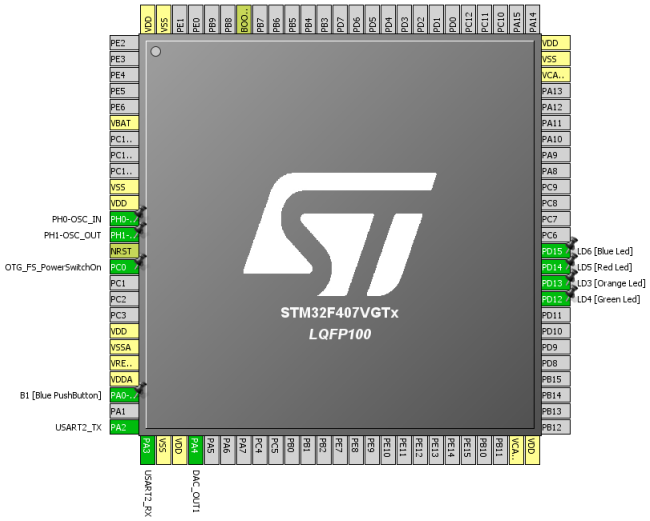
Aby zaoszczędzić czas potrzebny na wyliczanie wartości cosinusa kąta w każdej iteracji pętli algorytmu zdecydowano się na zapisanie wartości tej funkcji w pamięci mikokontrolera z rozdzielczością 1^0. Rozdzielczość ta jest w wielu wypadkach zadawalająca, jednak w przypadkach wymagających większej precyzji można skorzystać z funkcji *cosApproximation*, której kod podano w listingu 3, a która to aproksymuje wartość cosinusa kąta funkcją liniową pomiędzy dwiema stablicowanymi wartościami.

Listing 3: Funkcja aproksymująca wartość cosinusa.

```
1  inline float cosApproximation(float v)
2  {
3      float value = 0;
4
5      uint16_t a = ((int) v) % 360;
6      uint16_t b = (a + 1) % 360;
7
8      float cos_a = cosTable[a];
9      float cos_b = cosTable[b];
10     wsp_a = cos_b - cos_a;
11     wsp_b = cos_a - wsp_a * a;
12
13     value = wsp_a * (v - (int)(v) + a) + wsp_b;
14
15     return value;
16 }
```

4 Konfiguracja

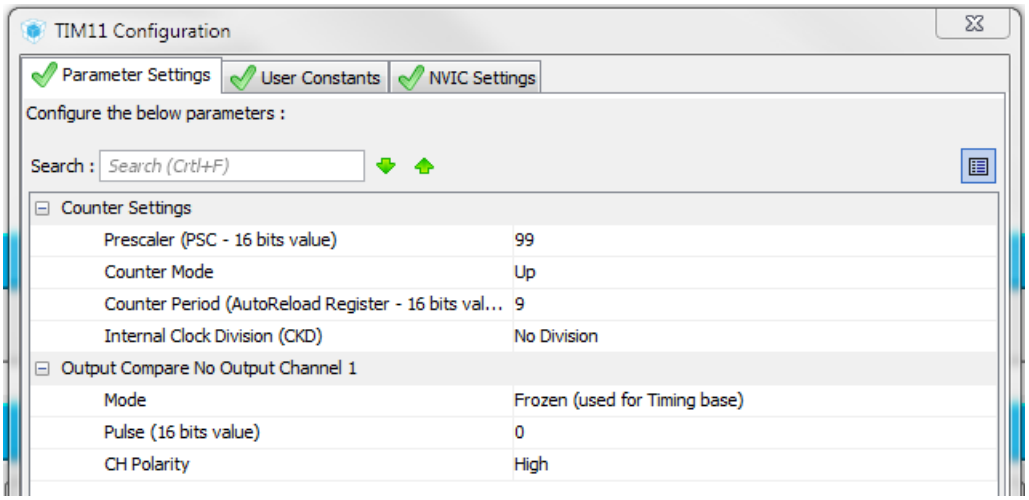
Konfiguracja wszystkich peryferiów wymaganych do uruchomienia układu została przeprowadzona w programie *CubeMX*. Następnie na jej podstawie wygenerowano projekt do dalszego rozwijania w środowisku *Eclipse*. Na rysunku 7 przedstawiono konfiguracje poszczególnych portów mikrokontrolera.



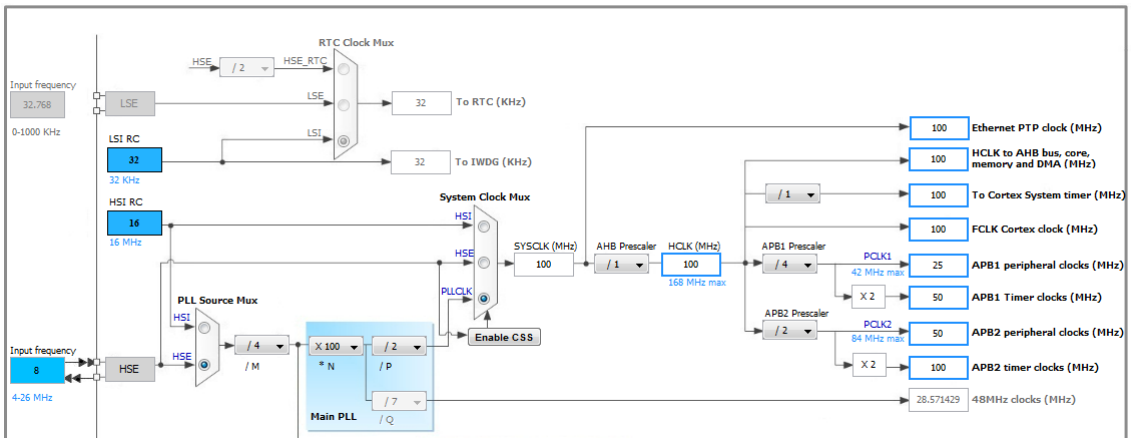
Rys. 7: Konfiguracja portów mikrokontrolera Stm32 w programie *CubeMX*.

4.1 Konfiguracja timerów

Do cyklicznego generowania kolejnych próbek sygnału wyjściowego wykorzystano przerwanie od Timera nr 11. Licznik skonfigurowano tak aby generował przerwanie z częstotliwością 100 kHz. Na rysunku 8 pokazany jest panel konfiguracyjny timer-a z programu *CubeMX*, natomiast na rysunku 9 konfiguracja sygnałów zegarowych w całym układzie.



Rys. 8: Przykład konfiguracji timera 11.



Rys. 9: Konfiguracja sygnałów zegarowych w układzie.

4.2 Konfiguracja przetwornika DAC

Przetwornik DAC został skonfigurowany w trybie 12 bitowym, a do generacji sygnału wyjściowego wykorzystywany był pierwszy kanał przetwornika na pinie PA4. Z racji na ograniczoną rozdzielczość przetwornika zdecydowano się na ograniczenie wartości amplitudy generowanego sygnału do ±50. Ponieważ wartości poszczególnych punktów postaci czasowej sygnału mogą przyjmować wartości zarówno dodatnie jak i ujemne, to przyjęto, że wartość zero odpowiada liczbie 2048 wpisanej do rejestru przetwornika.

4.3 Konfiguracja portu szeregowego

Do przesyłania danych z komputera do mikrokontrolera wykorzystany został protokół szeregowy *RS232*. Parametry charakteryzujące transmisję podane są w tabeli 1.

Tabela 1: Parametry portu szeregowego.

Parametr	**Wartość**
Prędkość transmisji	115200 Bitów/s
Długość słowa	8 bitów
Bit parzystości	-
Bit stopu	1

5 Wnioski

Spis treści