Przetwarzanie sygnałów
STM32 Cortex-M3

Spis treści

1 Wstęp

Jeszcze kilka lat temu, w dobie w której królowały mikrokontrolery 8-bitowe, niemalże wszystkie urządzenia w których była potrzeba nawet prostego przetwarzania sygnałów wymagały użycia procesora DSP. Niestety procesory sygnałowe były z reguły słabo wyposażone w peryferia typowe dla mikrokontrolerów. Wobec tego wymagane było stosowanie zarówno procesora sygnałowego jak i mikrokontrolera, co znacznie komplikowało układ. Obecnie rynek, w dużej części, należy do mikrokontrolerów z popularnym 32-bitowym rdzeniem ARM Cortex-M3. Rdzeń ten co prawda nie jest aż tak wydajny, aby całkowicie zastąpić DSP, nie mniej jednak znakomicie wypełnia lukę między mikrokontrolerami 8-bitowymi, a procesorami DSP. Jednocześnie świetnie sprawdza się w prostych aplikacjach pomiarowych i sterujących.

Kolejne dwa ćwiczenia mają na celu przedstawienie jednego z popularnych obecnie mikrokontrolerów wyposażonych w rdzeń Cortex-M3. Jako platformę sprzętową wybrano tani devkit STM32VLDISCOVERY z mikrokontrolerem firmy ST Microelectronics – STM32F100RB. Wybór podyktowany był przede wszystkim łatwą dostępnością mikrokontrolerów STM32 na polskim rynku, ich niską ceną, oraz niską ceną dev-kit'ów z tymi mikrokontrolerami. Ponadto do mikrokontrolerów STM32 firma Atollic udostępnia darmową (nieco okrojoną) wersję środowiska programistycznego True Studio Lite.

Głównym celem ćwiczeń jest pokazanie możliwości rdzenia Cortex-M3 w zadaniach przetwarzania sygnałów w czasie rzeczywistym.

2 Stanowisko laboratoryjne

2.1 Sprzęt

Stanowisko zostało wyposażone w oscyloskop oraz devkit STM32LDISCOVERY. Dokładny opis devkita, oraz dokumentację można znaleźć na stronie http://www.st.com/internet/evalboard/product/250863.jsp.

Główne cechy STM32LDISCOVERY:

- mikrokontroler STM32F100RB, 128 KB Flash, 8 KB RAM in 64-pin LQFP,
- wbudowany programator ST-Link który może służyć zarówno do programowania mikrokontroler na płytce, jak i jako zewnętrzny programator,
- układ może być zasilany zarówno z portu USB jak i z zewnętrznego źródła 5V lub 3.3V,
- układ może zasilać urządzenie docelowe,
- dwie diody LED (niebieska i zielona),
- jeden przycisk użytkownika,
- złącze goldpin z wyprowadzonymi wszystkimi liniami I/O pozwalające łatwo wpiąć moduł w płytkę prototypową

Poniżej przedstawiono zdjęcie :

2.2 Oprogramowanie

Ćwiczenia zostały przygotowane z wykorzystaniem środowiska Atollic True Studio w wersji Lite. Środowisko to istnieje zarówno w wersji generic dla różnych rdzeni z rodzin ARM7, ARM9 czy Cortex-M,

jak i wersjach dedykowanych dla konkretnych rodzin mikrokontrolerów – między innymi dla STM32. Środowisko True Studio oparte jest o popularne IDE – Eclipse. Poniżej przedstawiono możliwości wersji Lite w porównaniu z wersją komercyjną.

Atollic TrueSTUDIO®	**Lite**	**Pro**
Price	Free	Commercial
Languages supported	Asm, C	Asm, C/C++
Powerful IDE based on ECLIPSE™	√	√
Many additional IDE features	-	√
Basic cmdline tools for ARM®	√	√
Additional cmdline tools for ARM®	-	√
Basic cmdline tools for PC	-	√
Additional cmdline tools for PC	-	√
Basic debugger features	√	√
Professional debugger features	-	√
Advanced tracing in debugger	-	√
Graphical UML modeling	-	√
Version control system integration	-	√
Bug database integration	-	√
Code review & review meetings	-	√
I/O redirection of runtime libraries	-	√
Optional tiny printf/sprintf/fprintf	-	√
Number of supported JTAG probes	Limited	Extensive
Static source code inspection	Demo	1 week fully working
Code coverage analysis	Demo	1 week fully working
Test automation	Demo	1 week fully working
Unlimited code-size	√	√
Unlimited usage-time	√	√
Advertisement free	-	√
Customers must advertise Atollic TrueSTUDIO in end-user manuals of developed products	√	-
Technical support	-	Available

Jako, że środowisko Eclipse jest bardzo popularne opis korzystania z niego nie będzie przedstawiany. Wszelkie informacje na ten temat można znaleźć w dokumentacji na stronie http://atollic.com. Podstawowe informacje zawarto w dokumencie: *AtollicTrueSTUDIO® for STMicroelectronics® STM32™ Quickstart Guide*.

2.3 Przygotowanie do ćwiczeń.

Ćwiczenia zostały przygotowane jako dwa osobne projekty w jednym wspólnym worspace Eclipse'a. Przed przystąpieniem do wykonywania ćwiczeń każdy z zespołów powinien stworzyć swoją własną kopię wzorcowego katalogu workspace - katalog *STM32_workspace* w *D:\PSiM.*

Następnie, podczas uruchamiania środowiska True Studio, gdy program zapyta o ścieżkę do katalogu workspace należy podać ścieżkę do własnej kopii. Jeżeli podczas uruchamiania program nie zapyta o workspace tylko uruchomi się z domyślnym można go zmienić klikając kolejno:

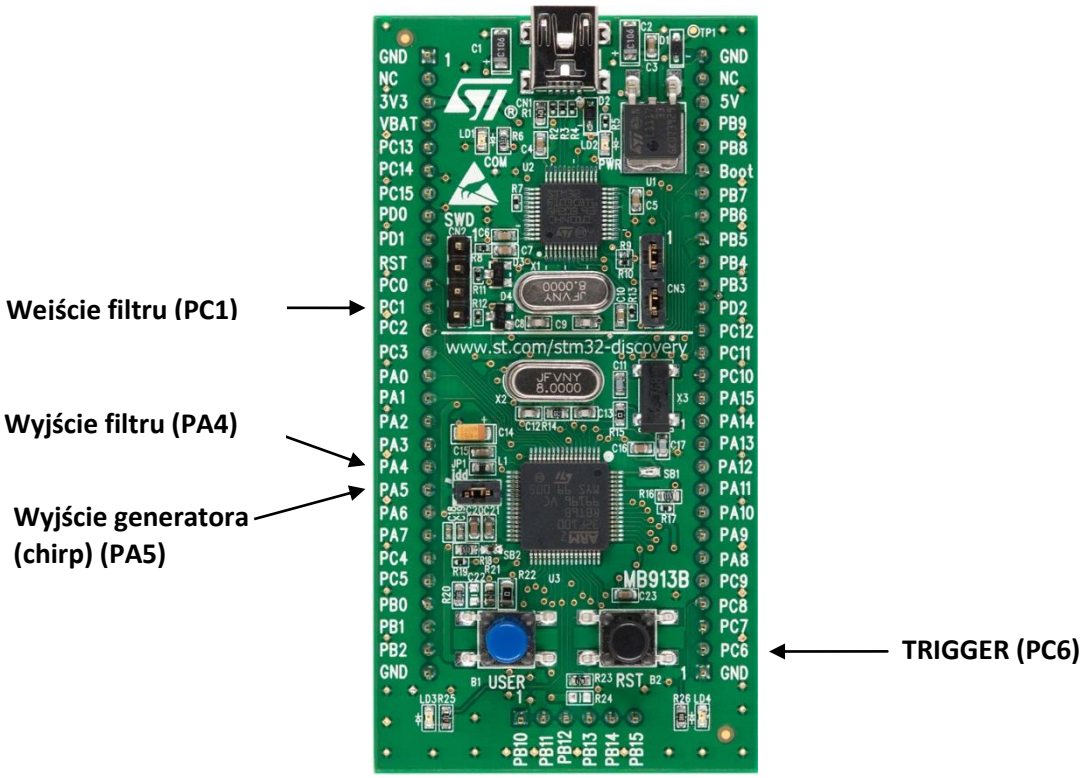
File->Switch Workspace->Other…

Podczas wykonywania ćwiczeń najlepiej aby jednocześnie mieć otwarty tylko jeden projekt – odpowiedni dla danego ćwiczenia.

3 Filtry IIR.

3.1 Przygotowanie stanowiska.

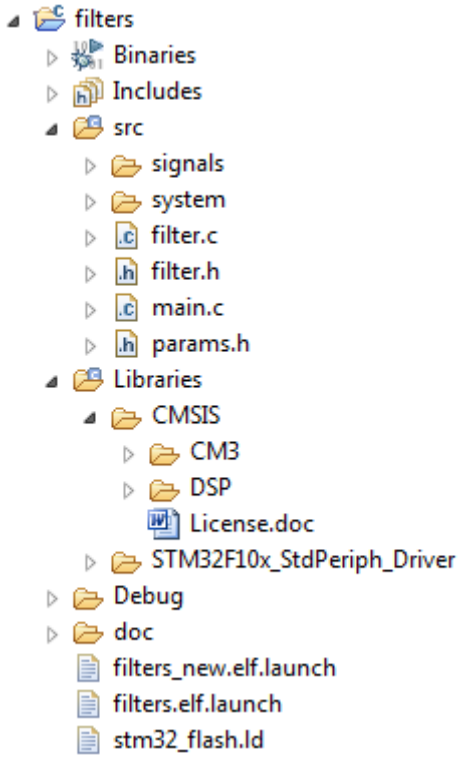
3.1.1 Podłączenia układu.



Przygotowanie układu polega tylko na połączeniu wyjścia generatora z wejściem filtru.

3.1.2 Katalog projektu.

Poniżej przedstawiono drzewo projektu wraz z opisem wybranych plików i katalogów.



Nazwa	Opis
src/signals	Tablice sygnałów dla generatora DAC
src/system	BSP i pliki systemowe
src/filter.h src/filter.c	Implementacja filtru
src/params.h	Parametry taktowania ADC i DAC
Libraries/CMSIS/CM3	Biblioteki CMSIS do rdzenia Cortex-3M
Libraries/CMSIS/DSP	Biblioteki DSP
Libraries/ STM32F10x_StdPeriph_Driver	Biblioteka peryferiów ST
Debug	Degub bin
Release	Release bin

3.2 Zadanie.

3.2.1 Opis

Zadanie polega na zaprojektowaniu co najmniej dwóch filtrów typu IIR o różnych częstotliwościach granicznych w zakresie 100Hz – 2kHz. Procedurę projektowania filtrów i wyznaczania współczynników przedstawiono poniżej.

Wyznaczone współczynniki należy wpisać do programu. Następnie należy skonfigurować Timer taktujący ADC (za pomocą define'ów w plików *params.h*) tak by uzyskać zadaną częstotliwość próbkowania równą 5kHz. Zostanie to dokładniej opisane w kolejnych podrozdziałach.

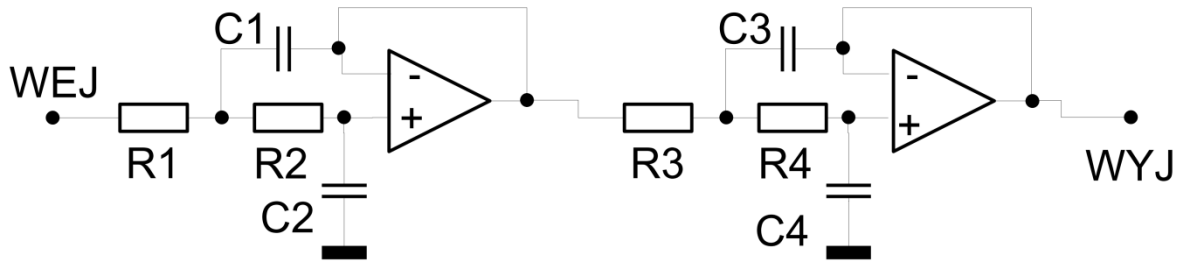
Na filtry będzie podawany sygnał chirp zmieniający się logarytmicznie lub liniowo od 10 Hz do 1kHz w czasie 100ms. Należy obserwować tłumienie filtru dla różnych częstotliwości.

3.2.2 Projektowanie filtru

3.2.2.1 Filtr analogowy

Jedną z metod projektowania filtrów cyfrowych, jest projektowanie ich na bazie filtrów analogowych o takiej samej charakterystyce. Wobec tego pierwszym etapem projektowania filtru cyfrowego jest projekt filtru analogowego.

Celem tego ćwiczenia jest zaprojektowanie analogowego dolnoprzepustowego filtru czwartego rzędu według poniższego schematu.



Transmitancja takiego filtru dana jest wzorem:

$$G(s) = \frac{1}{(s^2 R_1 R_2 C_1 C_2 + sC_2(R_1 + R_2) + 1)(s^2 R_3 R_4 C_3 C_4 + sC_4(R_3 + R_4) + 1)}$$

Na potrzeby projektowania filtrów dolnoprzepustowych wyprowadzono znormalizowane transmitancje dla filtrów czwartego rzędu. Przedstawione transmitancję odpowiadają pulsacji granicznej $\omega_{gr} = 1\left[\frac{rad}{s}\right]$.

Filtr Czebyszewa I

Dla falistości równej 3dB.

$$G(s) = \frac{1}{(5.5339s^2 + 2.1853s + 1)(1.2009s^2 + 0.1964s + 1)}$$

Filtr Butterworth'a

$$G(s) = \frac{1}{(s^2 + 1.8478 + 1)(s^2 + 0.7654s + 1)}$$

Filtr Bessela

$$G(s) = \frac{1}{(0.4889s^2 + 1.3397s + 1)(0.3890s^2 + 0.7743s + 1)}$$

Przykład – projekt filtru Butterworth'a

Na potrzeby projektowania analogowego filtru który ma być później podstawą do projektowania filtru cyfrowego możemy założyć wartości wszystkich rezystancji równe 1Ω. Wobec tego:

$$C_1 C_2 = 1$$

$$C_2(R_1 + R_2) = 1.8478$$

$$C_4(R_3 + R_4) = 0.7654$$

Z tego wynika:

$$C_1 = 1.0824, C_2 = 0.9239, C_3 = 2.6130, C_4 = 0.3827$$

Obliczone wartości dotyczą filtru o częstości granicznej 1 [rad/s]. W celu przeskalowania filtru należy każdą z pojemności podzielić przez $2\pi f_{gr}$.

Po obliczeniu pojemności odpowiednich dla wybranej częstotliwości granicznej należy wyznaczyć współczynniki w transmitancji odpowiadające zaprojektowanemu filtrowi.

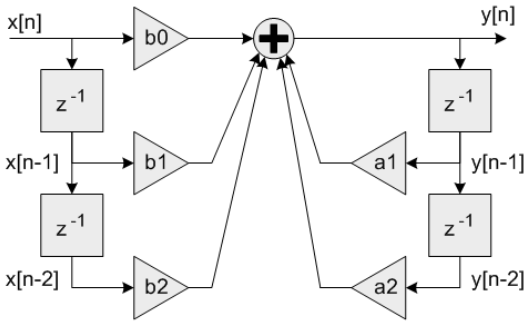
3.2.2.2 Filtr cyfrowy (na podstawie filtru analogowego)

Jednym z najczęściej stosowanych sposobów przejścia z transmitancji ciągłej (w dziedzinie ra s) na dyskretną (w dziedzinie z) jest zastosowanie transformacji Tustina. Polega ona na podstawieniu:

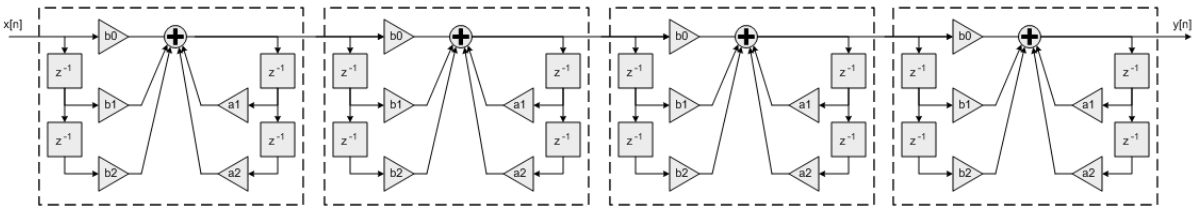
$$s = \frac{2}{T_0}\frac{z-1}{z+1}$$

gdzie T_0 to okres próbkowania.

Ze względu na fakt, że filtry IIR wysokich rządów są niestabilne, gdy wymagany jest rząd filtru wyższy niż 2 realizuję się go jako kaskadowe połączenie wielu filtrów drugiego rzędu. Poniżej przedstawiono schemat pojedynczego członu drugiego rzędu, oraz kaskadowe połączenie kilku takich członów.



Rys. 1 Jeden stopień (2-go rzędu) filtru kaskadowego. [7]



Rys. 2 Filtr IIR wyższego rzędu - kaskadowe połącznie członów 2-go rzędu. [7]

Wobec tego projektowany przez nas filtr 4-go rzędu musi zostać zrealizowany jako kaskadowe połączenie dwóch takich filtrów. Dlatego też transmitancja w dziedzinie operatora z musi zostać sprowadzona do postaci:

$$G(z) = G_1(z)G_2(z) = \frac{b_{10} + b_{11}z^{-1} + b_{12}z^{-2}}{1 + a_{11}z^{-1} + a_{12}z^{-2}} \cdot \frac{b_{20} + b_{21}z^{-1} + b_{22}z^{-2}}{1 + a_{21}z^{-1} + a_{22}z^{-2}}$$

Współczynniki filtrów można także łatwo wyznaczać korzystając z MATLABa. Proszę o zapoznanie się z funkcjami MATLABa: $butter, cheby1, cheby2, zp2sos$ (na podstawie helpa).

3.2.3 Modyfikowanie plików projektu.

3.2.3.1 Współczynniki filtru.

Obliczone współczynniki należy wpisać do tablicy o nazwie *coeffs* w pliku *src/filter.c*. Kolejność współczynników powinna być następująca:

$$coeffs[] = \{b_{10}, b_{11}, b_{12}, -a_{11}, -a_{12}, b_{20}, b_{21}, b_{22}, -a_{21}, -a_{22}\}$$

UWAGA!!!

Postać filtru używana przez biblioteki nieco różni się od postaci wynikającej wprost z transmitancji. Należy pamiętać, że obliczone przez nas współczynniki a_{ij} należy wpisać do tablicy z przeciwnym znakiem. Dotyczy to także współczynników obliczonych z użyciem funkcji *zp2sos* w Matlabie!

Należy także zwrócić uwagę na dokładność z jaką podajemy współczynniki do tablicy. Zbyt duże zaokrąglenie może powodować **utratę stabilności** filtru. Współczynniki powinny być wpisane z dokładnością co najmniej **do 4 miejsc znaczących**.

Proszę zapoznać się z rozdziałem *BiquadCascade IIR Filters Using Direct Form I Structure* w dokumentacji do biblioteki *CMSIS DSP* [7].

3.2.3.2 Sygnał chirp.

Istnieje możliwość wyboru sposobu generowania sygnału chirp. Sygnał ten może się zmieniać liniowo lub logarytmicznie. W przypadku liniowym, zaniedbując że startujemy od 10Hz można przyjąć, że w przybliżeniu 10ms na osi czasu (na oscyloskopie) odpowiada 100Hz. W przypadku logarytmicznym jedna dekada to 50ms.

Tryb generowania sygnału chirp jest konfigurowany w pliku src/*params.h* za pomocą odpowiedniego define'a. Odpowiednio:

```
#define GENERATOR_SIGNAL_NAME    chirp_lin
#define GENERATOR_SIGNAL_NAME    chirp_log
```

3.2.3.3 Częstotliwość próbkowania.

Za próbkowanie z zadaną częstotliwością odpowiedzialny jest jeden z timer'ów mikrokontrolera. Jego rola polega na cyklicznym wyzwalaniu konwersji przetwornika analogowo-cyfrowego. Częstotliwość wyzwalania dana jest wzorem:

$$f_s = \frac{24MHz}{(AI\_PRESCALER + 1)(AI\_PERIOD + 1)}$$

Należy tak dobrać wartości parametrów *AI_PRESCALER* i *AI_PERIOD*, aby częstotliwość próbkowania była równa 5kHz.

Obliczone wartości należy wpisać w pliku *src/params.h* w odpowiednich define'ach.

3.2.4 Obserwacja działania filtru na oscyloskopie.

Aby ułatwić obserwację działania filtru na oscyloskopie mikrokontroler **generuje na porcie PC6 sygnał prostokątny**, który należy wykorzystać **do synchronizacji**. Jest to sygnał o wypełnieniu 50% i okresie 200ms. Rosnące zbocze jest zsynchronizowane z momentem rozpoczęcia generowania sygna-

łu chirp, gdy ma on częstotliwość równą 10Hz. Gdy sygnał osiągnie częstotliwość 1kHz na porcie PC6 pojawia się zbocze opadające. Następnie, w czasie trwania stanu niskiego, sygnał chirp zmniejsza swoją częstotliwość do 10Hz i cykl się powtarza.

Aby móc oglądać sygnał należy sondę drugiego kanału oscyloskopu podpiąć do pinu PC6. Następnie należy ustawić trigger na zboczu narastającym i tak dobrać podstawę czasu aby cały przedział, gdzie sygnał synchronizujący jest w stanie wysokim, był widoczny.

Można także sprawdzić jak długo trwa obliczanie filtru. W tym celu należy sondę oscyloskopu podpiąć do portu PC9 (port ten odpowiada także zielonej diodzie – podczas obliczeń dioda się zapala). Czas trwania stanu wysokiego na tym porcie jest równy czasowi obliczeń.

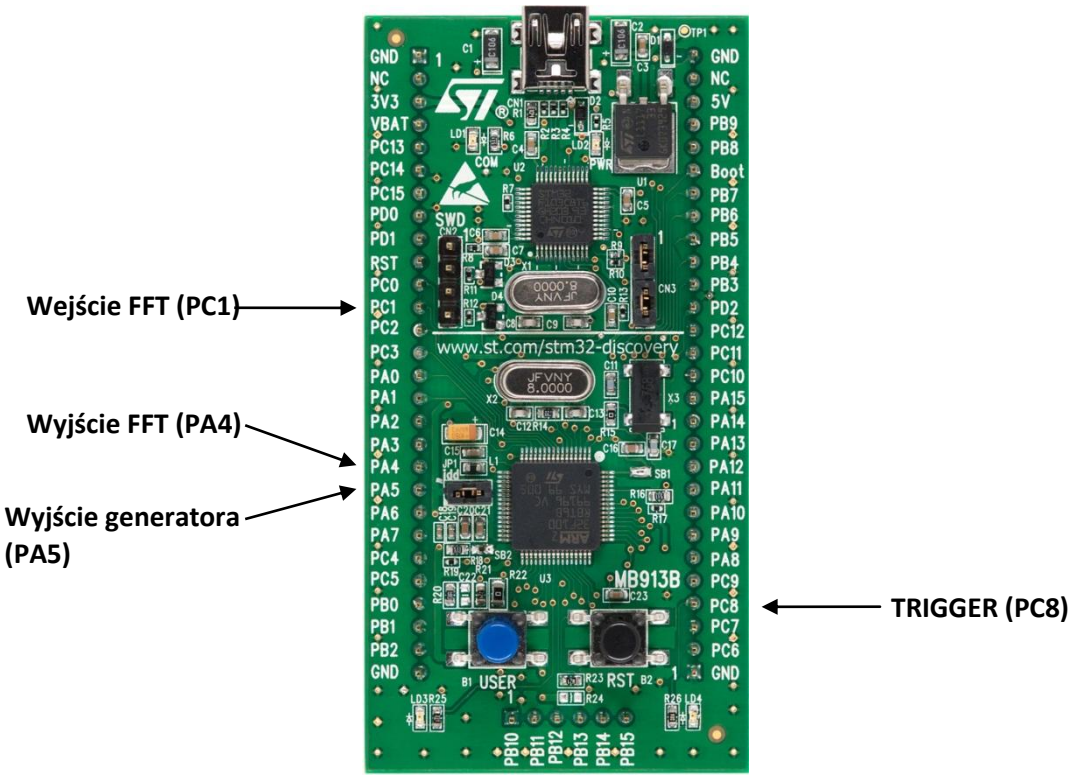
3.3 Zadanie dodatkowe

Niestety rdzeń Cortex-M3 nie posiada jednostki FPU, więc wszelkie operacje na zmiennych typu *float* trwają bardzo długo, gdyż są realizowane programowo. W związku z tym często w mikrokontrolerach stosuje się filtry operujące na zmiennych stałoprzecinkowych. Zadanie polega na zapoznaniu się z dokumentacją biblioteki *CMSIS DSP* i zaprojektowaniu analogicznego filtru jak poprzednio (dla takiej samej częstotliwości granicznej) tylko, że działającego na zmiennych stałoprzecinkowych. Następnie należy porównać czas obliczania takiego filtru w stosunku do odpowiednika na *float'ach*.

4 FFT

4.1 Przygotowanie stanowiska.

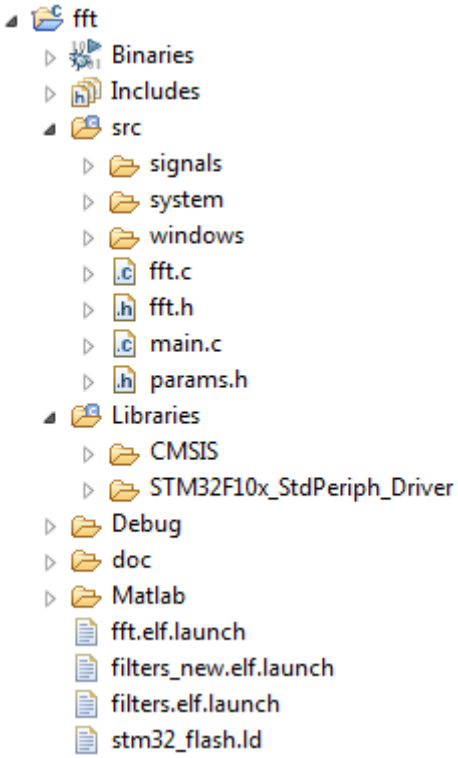
4.1.1 Podłączenia układu.



Przygotowanie układu polega tylko na połączeniu wyjścia generatora z wejściem FFT.

4.1.2 Katalog projektu.

Poniżej przedstawiono drzewo projektu wraz z opisem wybranych plików i katalogów.



Nazwa	Opis
src/signals	Tablice sygnałów dla generatora DAC
src/windows	Tablice z oknami
src/system	BSP i pliki systemowe
src/fft.h src/fft.c	Implementacja FFT
src/params.h	Parametry taktowania ADC i DAC, oraz parametry FFT
Libraries/CMSIS/CM3	Biblioteki CMSIS do rdzenia Cortex-3M
Libraries/CMSIS/DSP	Biblioteki DSP
Libraries/ STM32F10x_StdPeriph_Driver	Biblioteka peryferiów ST
Debug	Degub bin
Release	Release bin
Matlab	Dodatkowe narzędzia Matlabowe

4.2 Zadanie.

4.2.1 Opis

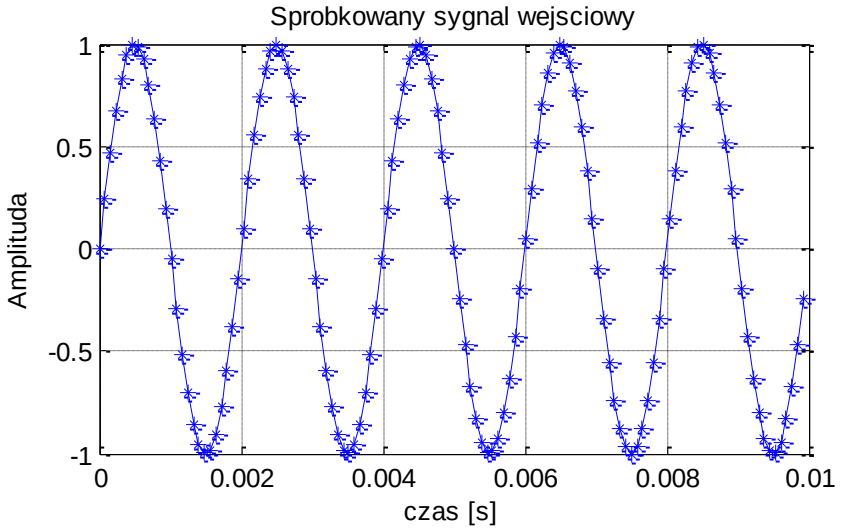
Celem ćwiczenia jest zapoznanie się z Szybką Transformatą Fouriera (FFT) w implementacji mikroprocesorowej, ze szczególnym uwzględnieniem wpływu okien oraz długości transformaty na uzyskiwane rezultaty. Ćwiczenie będzie polegało na obserwacji transformaty sygnału który jest sumą dwóch sinusów:

- $f = 800Hz, V_{pp} = 2V$
- $f = 2kHz, V_{pp} = 0.02V$

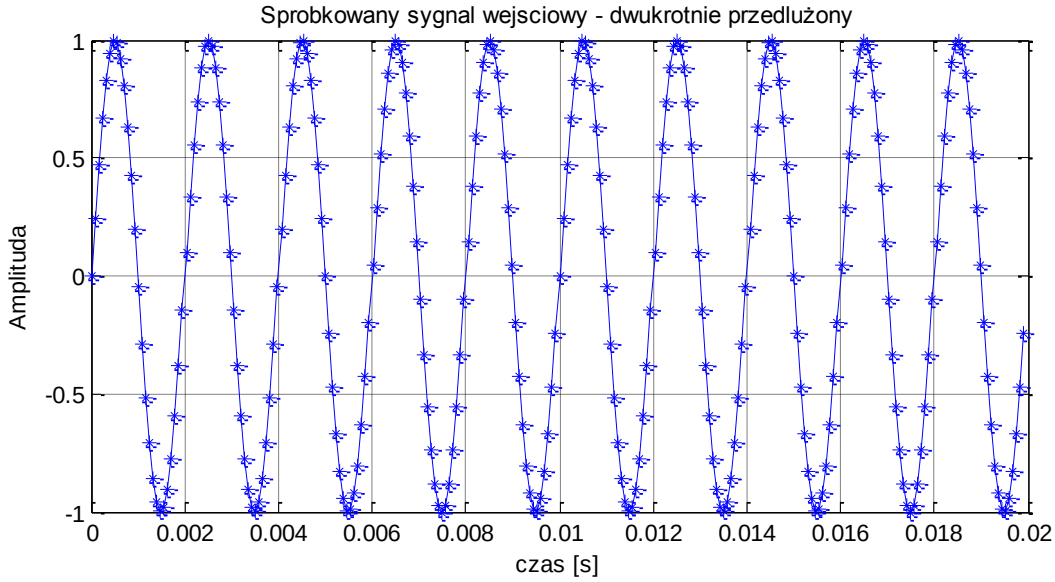
4.2.2 Wstęp teoretyczny.

4.2.2.1 Wpływ zmiany fazy początkowej sygnału sinusoidalnego, wyciek widma

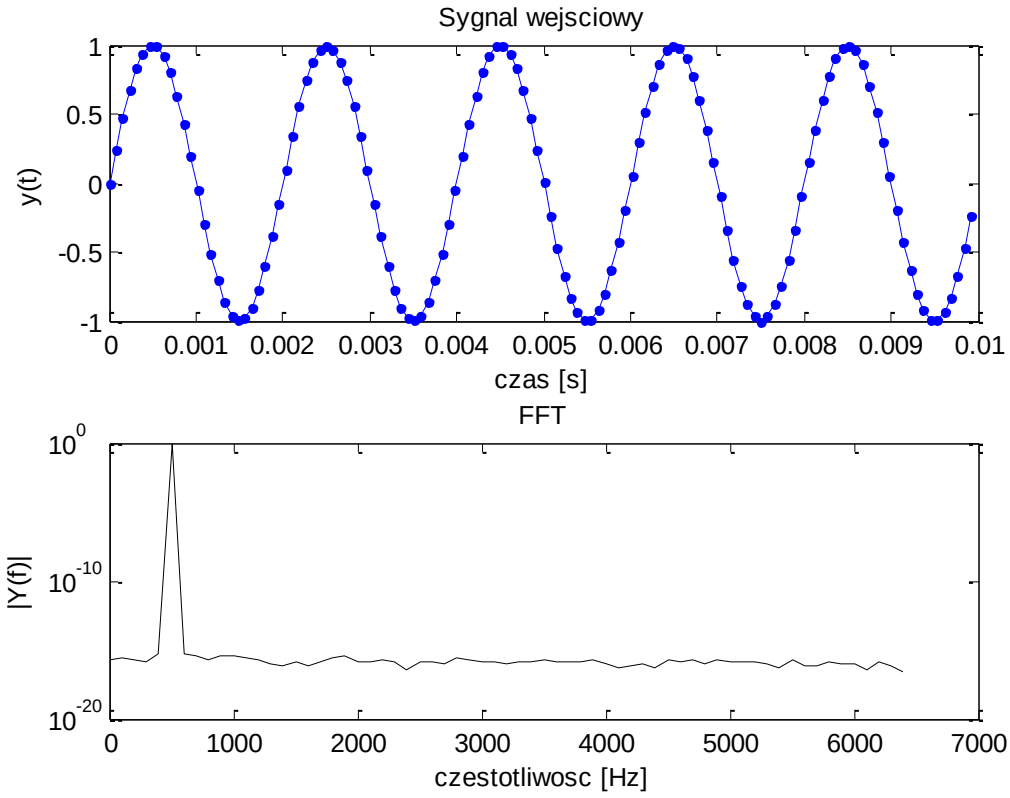
Podczas wyznaczania Dyskretnej Transformaty Fourier'a zakłada się iż sygnał jest okresowy i nieskończony, a bufor wejściowy zawiera całkowitą wielokrotność jego okresów. Rys. 3 przedstawia sytuację, w której założenie to zostało spełnione. Sygnał wejściowy o częstotliwości 500Hz został 'idealnie' spróbkowany w taki sposób, że bufor wejściowy zawiera 5 pełnych okresów. Gdyby przedłużyć taki sygnał do nieskończoności nie pojawią się żadne zniekształcenia. Rys. 4 przedstawia dwukrotnie przedłużony sygnał wejściowy. Dalsze przedłużanie również nie prowadziłoby do żadnych nieprawidłowości. Po wyznaczeniu Szybkiej Transformaty Fouriera wykreślono następujący wykres - Rys. 5. Widać wyraźny pik oznaczający wystąpienie w sygnale częstotliwości 500Hz.



Rys. 3 Spróbkowany sygnał wejściowy o częstotliwości 500Hz – odpowiednio dobrana długość okna

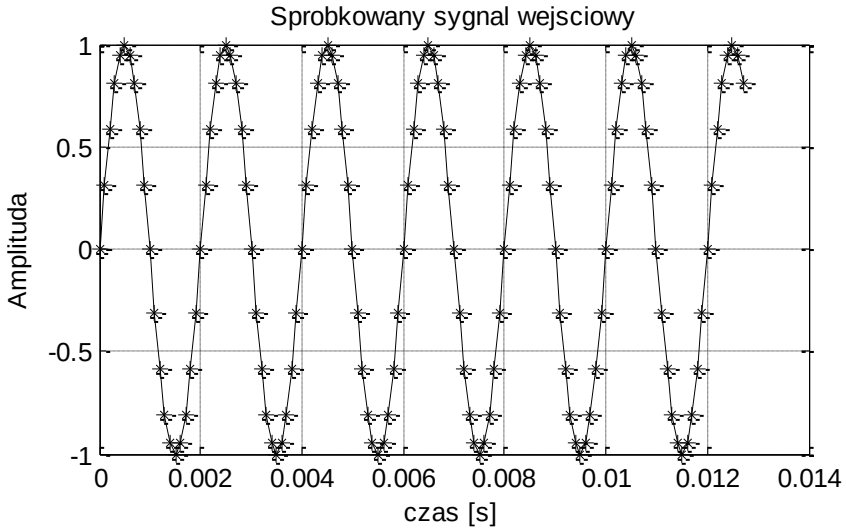


Rys. 4 Spróbkowany sygnał wejściowy o częstotliwości 500Hz – dwukrotnie wydłużony, odpowiednio dobrana długość okna



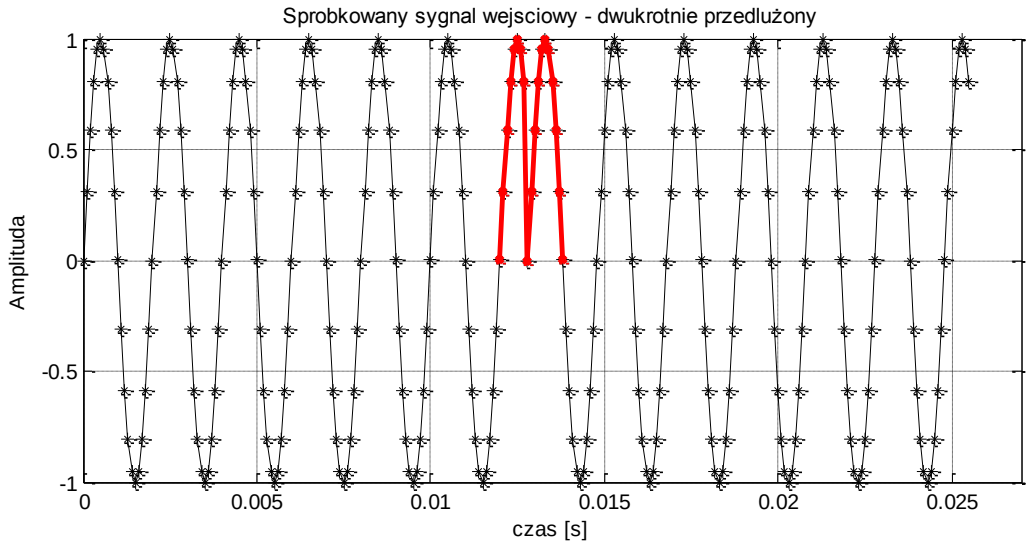
Rys. 5 Wynik transformaty Fouriera – odpowiednio dobrana długość okna

Niestety takie spróbowanie sygnału wejściowego jest w praktyce właściwie niespotykane. Jeżeli w buforze wejściowym nie znajdzie się pełen okres mierzonego sygnału to zaobserwować można zjawisko ‘przecieku’ widma (ang. leakage). Rys. 6 przedstawia sytuację, gdy bufor wejściowy nie zawiera całkowitej wielokrotności okresu sygnału sinusoidalnego o częstotliwości 500Hz.



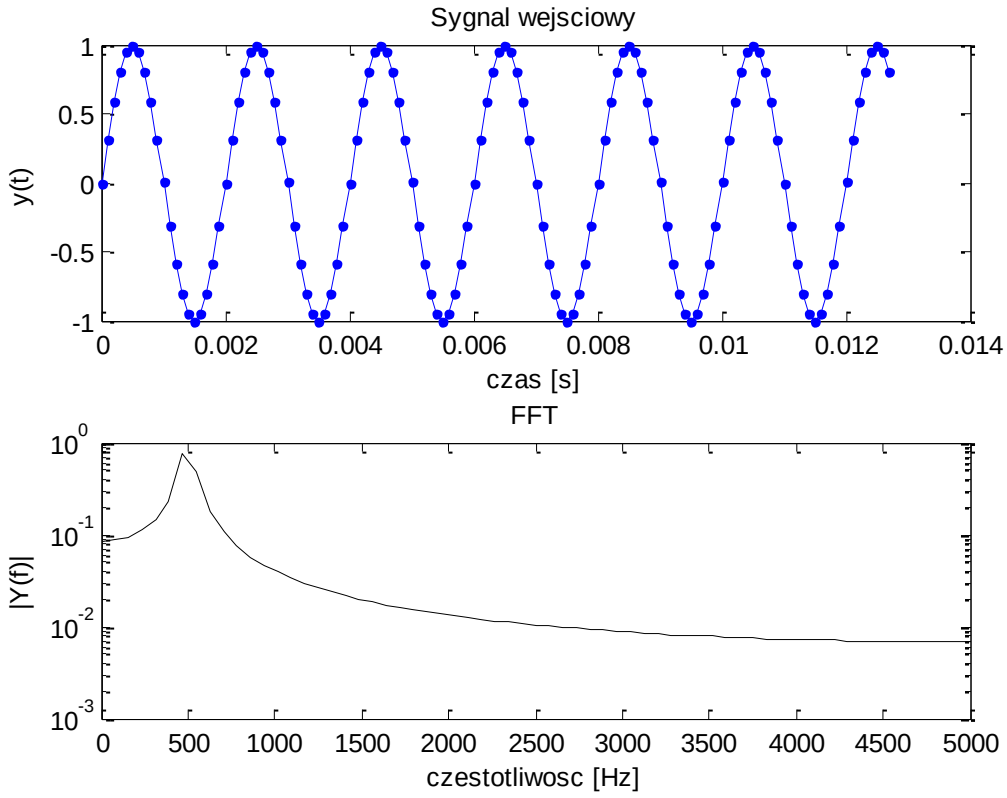
Rys. 6 Spróbkowany sygnał wejściowy o częstotliwości 500Hz

Zawartość bufora po dwukrotnym przedłużeniu przedstawiona została na Rys. 7



Rys. 7 Spróbkowany sygnał wejściowy o częstotliwości 500Hz – dwukrotnie wydłużony – widoczny brak okresowości

Na czerwono oznaczony został fragment przebiegu, w którym w wyniku przedłużenia wyraźnie zakłócony zostaje sinusoidalny przebieg. Zbadajmy zatem jaki będzie wynik transformaty Fouriera wykonanej na tak spróbowanym sygnale – Rys. 8.



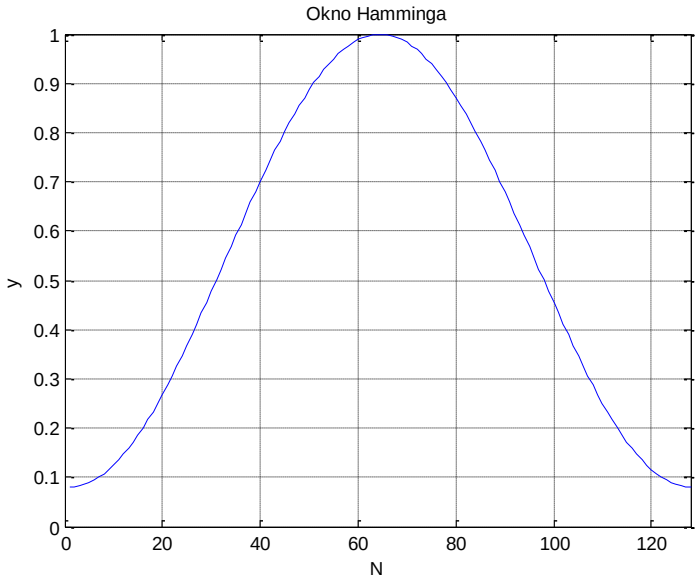
Rys. 8 Wynik transformaty Fouriera

Jak widać, pik oznaczający wystąpienie sygnału o częstotliwości 50Hz jest znacznie mniej wyraźny. Stosunek maksymalnej wartości piku do najniższej wartości uzyskanej w wyniku FFT nie wynosi tak jak na Rys. 5 około 10^{15} lecz jedynie około 10^{2}. Sytuacja taka byłaby szczególnie niekorzystna gdyby

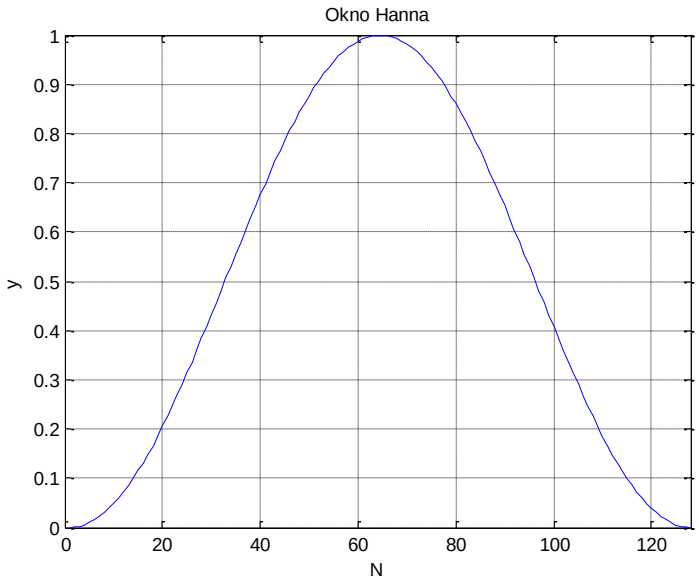
w analizowanym sygnale znajdowała się dodatkowa składowa o amplitudzie np. 100 razy mniejszej niż sygnał 500Hz. Wyciek widma z sygnału 500Hz znacznie utrudniłby lub wręcz uniemożliwiłby wykrycie piku odpowiadającego dodatkowej składowej zmiennej. Dlatego też podczas analizy widma sygnału stosuję się tzw. ‘oknowanie’ które często redukuje niekorzystny wpływ wycieku.

4.2.2.2 Redukcja efektu wycieku widma poprzez oknowanie sygnału

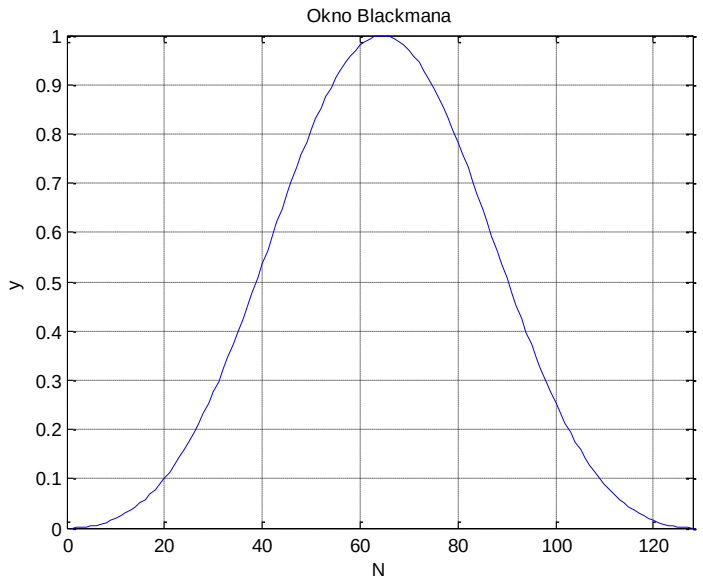
Oknowanie sygnału to po prostu przemnożenie bufora wejściowego, z którego wyznaczamy transformatę Fouriera przez funkcję zwaną oknem. Okna zazwyczaj przyjmują wartości od [0,1]. Posiadają maksimum oraz są symetryczne względem prostej przechodzącej przez to maksimum. Na krańcach przedziału określoności przyjmują wartości bliskie lub równe zeru. Długość okna zwykle równa jest długości bufora.



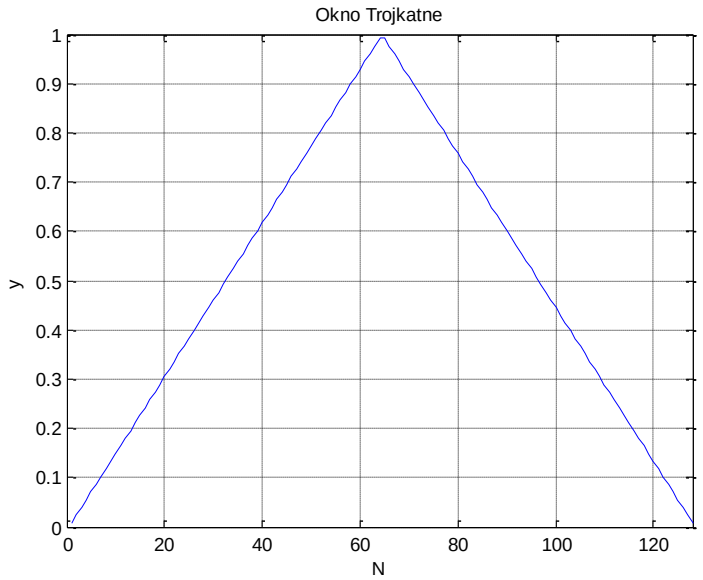
Rys. 9 Okno Hamminga – długość N=128



Rys. 10 Okno Hanna – długość N=128

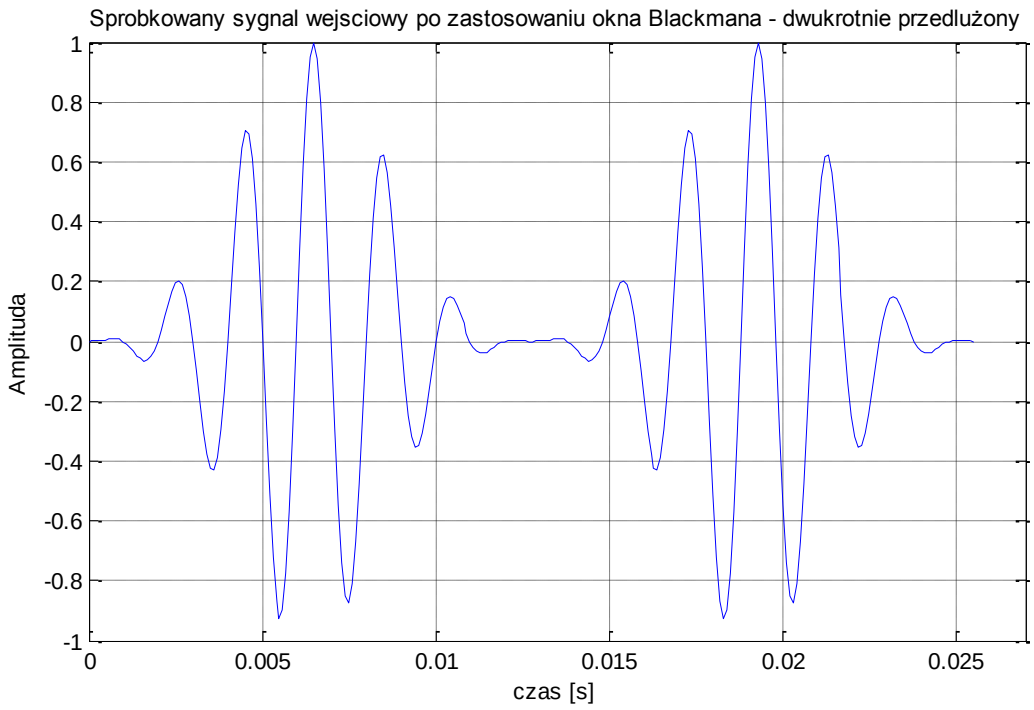


Rys. 11 Okno Blackmana– długość N=128



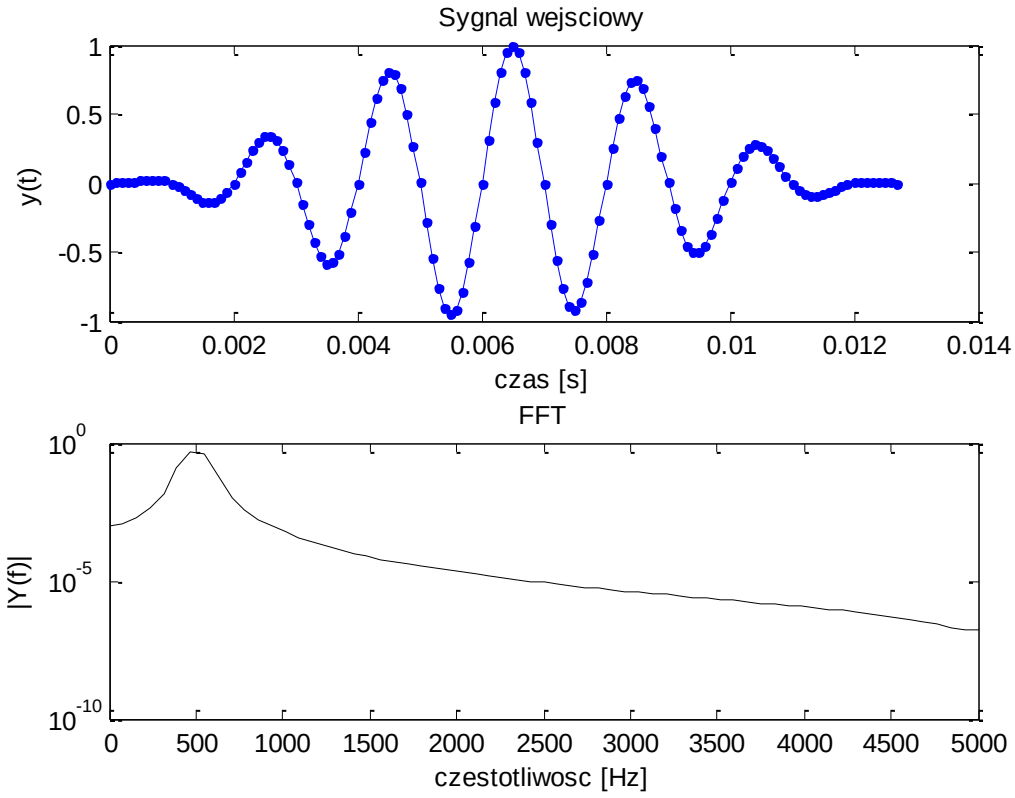
Rys. 12 Okno Trójkątne– długość N=128

W praktyce celem stosowania okien jest takie przekształcenie sygnału wejściowego w dziedzinie czasu, aby na krańcach przedziału amplituda wynosiła 0 lub była mu bliska, co redukuje niekorzystne efekty podczas przedłużania sygnału do nieskończoności. Rys. 13 przedstawia dwukrotnie przedłużony sygnał z Rys. 6 po zastosowaniu okna Blackmana.



Rys. 13 Spróbkowany sygnał wejściowy o częstotliwości 500Hz – dwukrotnie wydłużony – uprzednio poddany knowaniu oknem Blackmana

Jak widać w punkcie przedłużenia nie występuje tym razem żadna widoczna nieciągłość. Sygnał stał się okresowy. Modulacji uległa amplituda sygnału, lecz nie jego częstotliwość. Rys. 14 przedstawia efekt transformaty Fouriera dokonanej na sygnale wejściowym poddanym uprzednio oknowaniu.

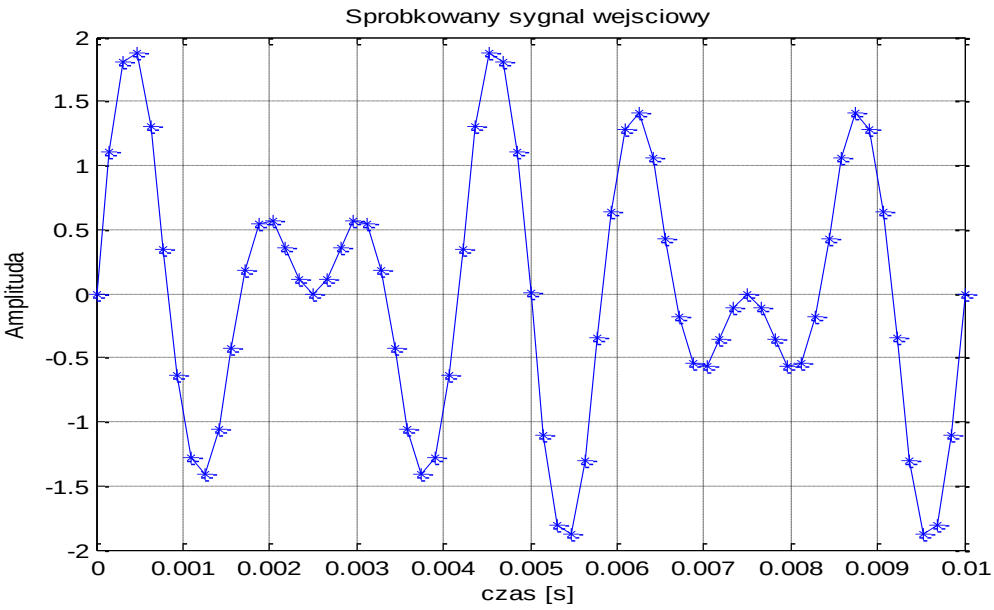


Rys. 14 Wynik transformaty Fouriera – uprzednio poddany knowaniu oknem Blackmana

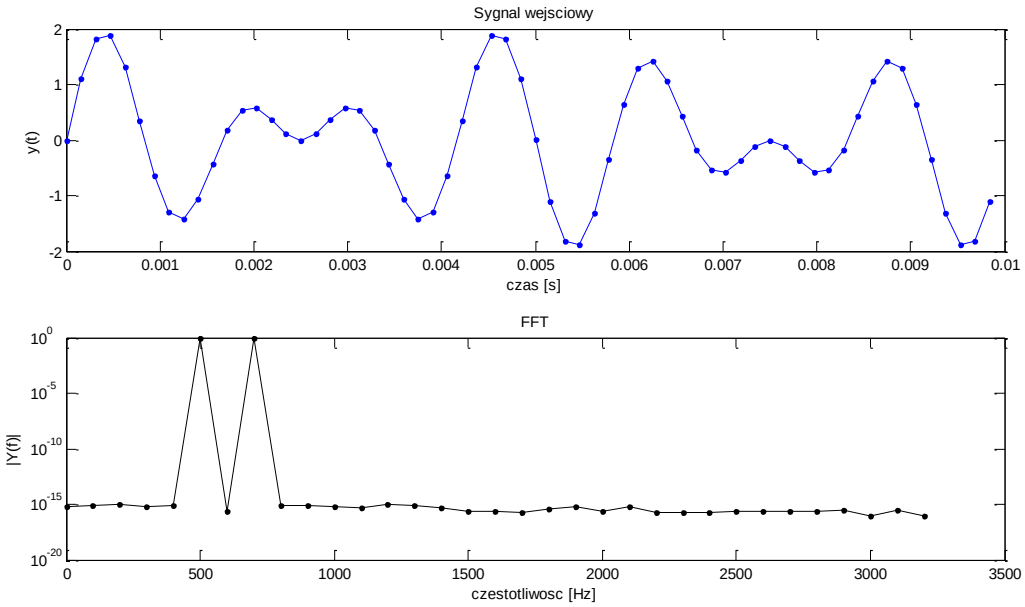
Stosunek maksymalnej wartości piku odpowiadającego za składową 500Hz do wartości 'tła' wynosi teraz około 10^5- 10^6 zatem uzyskano poprawę o ponad 3 dekady.

4.2.2.3 Wpływ długości bufora na jakość FFT

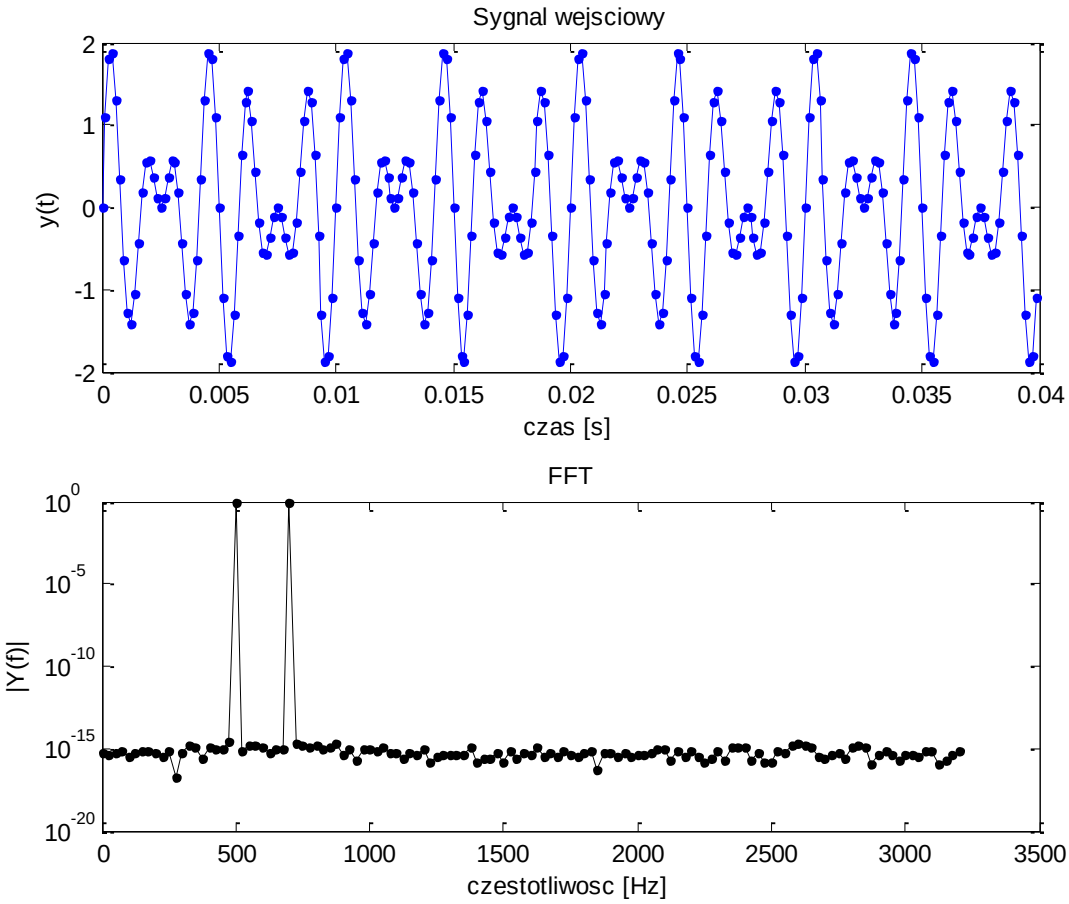
Rozważmy teraz wpływ długości bufora wejściowego transformaty na jakość uzyskiwanych rezultatów. Analizie poddamy sygnał y(t)=sin(500Hz)+sin(700Hz). Dla uproszczenia sygnał był tak próbkowany aby bufor zawierał pełen jego jeden okres lub wielokrotność. Zmieniano jedynie długość bufora. Testy przeprowadzono dla dwóch długości $N_1 = 64$ oraz $N_2 = 256$. Rys. 15 Przedstawia analizowany sygnał w dziedzinie czasu. Rys. 16 i Rys. 17 przedstawiają widmo sygnału dla N_1 i N_2.



Rys. 15 Złożenie dwóch sygnałów sinusoidalnych tworzących sygnał wejściowy – dziedzina czasu.



Rys. 16 Wynik analizy Fourierowskiej dla N = 64



Rys. 17 Wynik analizy Fourierowskiej dla N = 256

Zarówno transformata wykonana na 64 jak i 256 próbkach umożliwiła wykrycie dwóch częstotliwości z których składał się sygnał wejściowy. Jednakże stosując jedynie 64 próbki, uzyskane 'piki' odpowiedzialne za wykrycie obu częstotliwości są znacznie szersze. Piki te prawie nakładają się siebie. Gdyby częstotliwości były bliższe sobie zastosowanie tak małej długości bufora mogłoby okazać się niewystarczające do ich wykrycia. Po zastosowaniu 4 krotnie dłuższego bufora 'piki' stały się znacznie węższe i oddalone od siebie.

Wraz ze wzrostem długości bufora FFT rośnie rozdzielczość transformaty. Odległość między kolejnymi punktami na osi częstotliwości wyznaczyć można z prostego wzoru $\Delta f = f_s/N$, gdzie f_s to częstotliwość próbkowania, a N to długość bufora wejściowego. Należy pamiętać iż maksymalna częstotliwość jaką można wykryć przy użyciu FFT nie zależy od długości bufora i wynosi zawsze $f_s/2$.

Z powyższych eksperymentów wywnioskować można, że im dłuższe FFT tym lepsza rozdzielczość, a co za tym idzie lepsza jakość analizy widmowej. Należy jednak pamiętać, iż zwiększanie długości transformaty powoduje wzrost czasu obliczeń. Należy zatem zachować kompromis pomiędzy dostępną mocą obliczeniową, a rozdzielczością. Jest to szczególnie istotne w przypadku małych systemów mikroprocesorowych, w szczególności tych pracujących w czasie rzeczywistym.

4.2.3 Realizacja zadania

4.2.3.1 Obserwacja FFT na oscyloskopie.

Obliczanie FFT jest wykonywane dopiero gdy w buforze zbierze się odpowiednia ilość próbek, zależna od długości FFT. W celu umożliwienia obserwacji FFT na ekranie oscyloskopu mikrokontroler generuje **na porcie PC8 sygnał prostokątny do synchronizacji.** Okres tego sygnału zależy od długości FFT i okresu próbkowania sygnału wejściowego (który w tym ćwiczeniu jest stały i równy 5kHz). Należy podpiąć sondę drugiego kanału oscyloskopu do pinu PC8 i ustawić wyzwalanie zboczem rosnącym. Następnie należy tak dobrać podstawę czasu, aby na ekranie był wyświetlany cały przedział czasu w którym sygnał synchronizujący jest w stanie wysokim.

Wynik FFT w zakresie częstotliwości 0 – 2.5kHz wyświetlany jest gdy sygnał synchronizujący znajduje się w stanie wysokim. W stanie niskim obserwujemy transformatę dla częstotliwości ujemnych, która jest symetryczna względem 0Hz, gdyż sygnał wejściowy jest czysto rzeczywisty. Wyniki transformaty są zawsze normalizowane do 1. Oś wzmocnienia (pionowa) jest wyskalowana logarytmicznie tak, że **1 dekadzie odpowiada 1V**. Ponadto **napięcie** wyjściowe **0V odpowiada wartości 10^{-3}, a napięcie 3V wartości 1**.

Dodatkowo można sprawdzić **jak długo trwa obliczanie FFT**. W tym celu należy zmierzyć **czas** trwania stanu **wysokiego na porcie PC9 przy użyciu oscyloskopu.** Zmiana długości FFT.

Pierwsze zadanie polega na zmianie długości obliczanego FFT i obserwacji jak długość FFT wpływa na uzyskiwane wyniki. Długość transformaty można zmienić w pliku *src/params.h* za pomocą odpowiedniego define'a (*FFT_LENGTH*).

Dostępne są tylko dwie możliwe długości do wyboru: **64** i **256**.

4.2.3.2 Synchronizacja pomiaru.

Standardowo mikrokontroler zbiera dane aż do zapełnienia bufora i oblicza FFT. Jak tylko FFT zostanie obliczone mikrokontroler ponownie zaczyna zbieranie danych. Takie podejście sprawia, że za każdym razem w buforze może być inny fragment sygnału. W efekcie uzyskiwane transformaty różnią się od siebie, co objawią się „skakaniem" obrazu na oscyloskopie.

Aby zaradzić tej sytuacji zaimplementowano mechanizm synchronizacji, który sprawia, że zbieranie danych nie rozpoczyna się od razu po zakończeniu obliczania FFT. Mikrokontroler czeka, aż sygnał będzie bliski zeru i dopiero wtedy zaczyna zapisywać pomiary. Dzięki temu dane w buforze w kolejnych iteracjach są bardzo podobne, a widmo które otrzymujemy na ekranie oscyloskopu jest powtarzalne.

Synchronizację można włączyć w pliku *src/params.h* za pomocą odpowiedniego define'a. Dodatkowo można ustawić próg wykrywania przejścia przez zero. Przejście przez zero jest wykrywane gdy:

$$x^2 < FFT\_SYNCHRONIZATION\_TRESHOLD$$

gdzie x to wartość pomiaru.

4.2.3.3 Wykorzystanie okien.

Mikrokontroler domyślnie oblicza FFT używając okna prostokątnego. Istnieje jednak możliwość wykorzystania innych okien. Dodatkowo używane okno można wybrać w pliku *src/params.h* za pomocą

odpowiedniego define'a. **Należy jednak pamiętać, że długość okna musi być równa długości transformaty**. Wybrane okno może być aktywowane w czasie działania programu poprzez przytrzymanie niebieskiego PushButton'a opisanego: *USER*. Puszczenie przycisku sprawia, że mikrokontroler z powrotem korzysta z okna prostokątnego.

Podczas wykonywania ćwiczenia należy zwrócić szczególną uwagę na wyniki jakie otrzymujemy korzystając z różnych okien. Jakie są główne różnice i jakie cechy posiadają poszczególne okna.

4.3 Zadanie dodatkowe

Zadanie dodatkowe polega na znalezieniu innych typów okien i dodanie ich do programu. Po czym ich przetestowanie i porównanie z tymi dostępnymi domyślnie. W katalogu *Matlab* znajduje się funkcja która pozawala generować plik **.c* z tablicą float'ów z tablicy w Matlab'ie. Znajduje się tam także skrypt z przykładami użycia tej funkcji. Należy pamiętać, że **okna muszą mieć długość dopasowaną do długości FFT**.

5 Bibliografia

[1] http://st.com

[2] http://atollic.com

[3] http://arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php

[4] *True Studio Feature list and feature comparison guide for v2.2 products*, Atollic

[5] *UM0919: STM32VLDISCOVERY STM32 value line Discovery*, ST Microelectronics

[6] *UM0987: Developing your STM32VLDISCOVERY application using the AtollicTrueSTUDIO® software*, ST Microelectronics

[7] *CMSIS DSP Library Documentation*, ARM