Jacek Misiurewicz
Krzysztof Kulpa
Piotr Samczyński
Mateusz Malanowski
Piotr Krysik
Łukasz Maślikowski
Damian Gromek
Artur Gromek
Marcin K. Bączyk

Zakład Teorii Obwodów i Sygnałów
Instytut Systemów Elektronicznych
Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Laboratorium Cyfrowego Przetwarzania Sygnałów

Wersja do wydruku - bez części teoretycznej

Warszawa 2019

Ćwiczenie 8

Przetwarzanie sygnałów w czasie rzeczywistym – układy FPGA

Część teoretyczną w tej wersji opuszczono.

8.2. Zadania do pracy własnej studenta

Podobne zadania mogą znaleźć się na wejściówce. Nie dotyczy to zadań oznaczonych tu jako „trudne".

1) Naszkicuj schemat filtru NOI (albo SOI) rzędu 2, używając elementów: opóźnienie, układ mnożący, sumator.

2) Dla narysowanego w poprzednim punkcie schematu zapisz w każdym węźle, ilu bitów potrzeba, aby reprezentować dane bez zaokrągleń (uwaga – to się da zrobić tylko dla SOI; zastanów się dlaczego).

3) Dla schematu NOI zapisz liczby bitów takie, aby szum zaokrąglenia był porównywalny z szumem kwantowania sygnału wejściowego (uwaga – to zależy od współczynników: przyjmij sobie jakieś wartości do obliczeń).

4) Oblicz stosunek mocy (średniej za okres) sygnału błędu do sygnału użytecznego przy arytmetyce z nasyceniem i przy standardowej arytmetyce w kodzie U2; aby było łatwiej, przyjmij do obliczeń, że sygnał przekracza zakres arytmetyki $\sqrt{(2)}$-krotnie.

5) Oblicz, jaką amplitudę może mieć maksymalnie sygnał na wyjściu filtru, gdy na wejściu jest sygnał o amplitudzie nie większej niż 1:

– dla filtru SOI (FIR) rzędu N o współczynnikach nie większych niż 1 (co do modułu),
– dla filtru NOI (IIR) rzędu 1 – o jednym biegunie p_0, którego moduł jest równy $|p_0| = 0{,}9$,
– *trudne* dla filtru NOI (IIR) rzędu N, którego bieguny mają moduł nie większy niż r.

Wskazówka: Rozpatrz sygnał na wyjściu jako splot sygnału wejściowego z $h(n)$ i użyj inteligentnie nierówności dla wartości bezwzględnych.

8.3. Dostępny sprzęt i oprogramowanie

8.3.1. Układ laboratoryjny

W ćwiczeniu będziemy implementować algorytmy przetwarzania sygnałów, wykorzystując układ programowalny „Zynq" firmy Xilinx[1], składający się z mikroprocesora ARM i matrycy bramek FPGA w jednym układzie scalonym.

Układ ten, wraz z układami pomocniczymi (dla nas najważniejsze wśród nich są przetworniki – A/C i C/A) zamknięty jest w urządzeniu National Instruments „MyRio" i będziemy go programowali za pośrednictwem środowiska LabView.

Sygnały będziemy przetwarzali za pomocą części FPGA, natomiast ARM będzie potrzebny tylko do komunikacji z PC. Komputer PC będzie używany do:
- przygotowania konfiguracji FPGA,
- ładowania (*downloading*) konfiguracji do MyRio,
- przekazywania ustawianych przez operatora parametrów (np. współczynników filtru – jeśli zaprojektujemy je jako zmienne),
- wyświetlania sygnałów przechwyconych wewnątrz FPGA (sygnały wejściowe i wyjściowe wolimy oglądać naprawdę – tj. na oscyloskopie).

W środowisku LabView procesor ARM określany jest skrótem „RT" (*realtime*), ponieważ pracuje pod kontrolą systemu czasu rzeczywistego; nie zmienia to faktu, że naprawdę wysokie szybkości przetwarzania można uzyskać dopiero za pomocą FPGA.

Układ FPGA, który dla nas jest zasadniczym obiektem zainteresowania, będzie się komunikował z przetwornikami i wykonywał operacje opóźnienia, mnożenia i sumowania. Faktycznie więc układ ten, po skonfigurowaniu, będzie wykonywał całość przetwarzania samodzielnie.

Do wykonania ćwiczenia będą potrzebne: urządzenie MyRio, kabel USB typ A-B, zasilacz do MyRio, 2 sztuki kabli 2BNC⟶miniJack. MyRio musi nosić numer stanowiska – inaczej bedą problemy z licencją, która wiąże MyRio z komputerem PC.

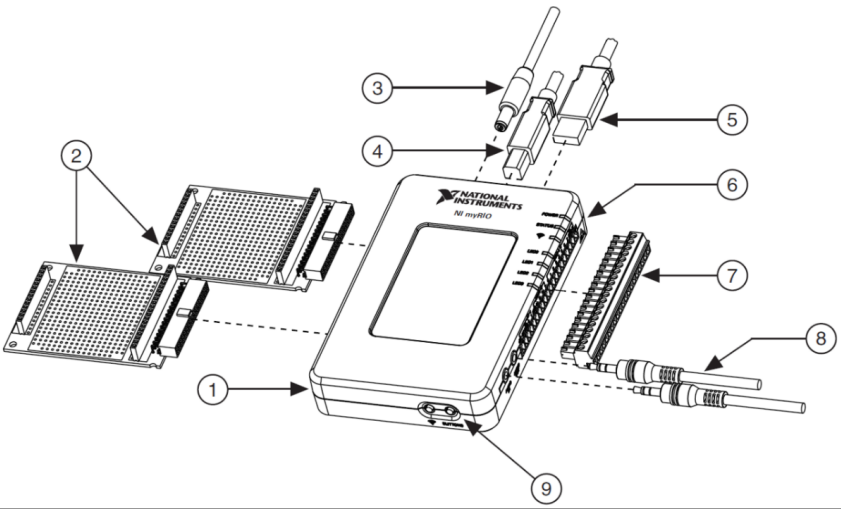
8.3.2. Zestaw uruchomieniowy NI myRIO

Używany w laboratorium wielofunkcyjny zestaw uruchomieniowy *NI myRIO--1900* jest to zintegrowana platforma czasu rzeczywistego oparta na układzie *Zynq-7010* zawierającym układ FPGA i procesor ARM. Oprogramowanie producenta (tzw. *firmware*) zostało zapisane na urządzeniu w pamięci trwałej typu *flash*. W skład tego oprogramowania wchodzi system operacyjny czasu rzeczywistego (*embedded linux*), środowisko uruchomieniowe (*LabVIEW RT*) oraz sterowniki. Dzięki temu zachowana jest komunikacja pomiędzy komputerem PC a platformą, i możliwe jest załadowanie (z poziomu środowiska LabView) kodu programu użytkownika i jego wykonanie.

[1] Czytaj „zaj-links".

System ten został uzupełniony o zasilacz sieciowy, kabel komunikacyjny USB, karty rozszerzeń oraz komputerowy zestaw słuchawkowy (wraz z mikrofonem).

Na rysunku 8.1 przedstawiono wszystkie elementy laboratoryjnego zestawu uruchomieniowego.



1	NI MyRIO-1900	6	LEDy
2	Karty portów rozszerzeń (MXP)	7	System mini portów złączy śróbowych
3	Kabel zasilający	8	Wejścia/wyjścia liniowe (stereo audio)
4	Port główny USB (typu B) urządzenia	9	Przycisk ogólnego przeznaczenia
5	Port USB (typu A) ogólnego przeznaczenia		

Rysunek 8.1. Zestaw uruchomieniowy NI myRIO-1900 wraz z osprzętem

Platforma uruchomieniowa *NI myRIO-1900* (rys. 8.2) daje studentowi możliwość korzystania z szerokich zasobów sprzętowych – w tym dwurdzeniowego mikroprocesora ARM z wbudowanym systemem czasu rzeczywistego i układu logiki programowalnej FPGA. Połączenie w jednym układzie procesora i matrycy FPGA pozwala na efektywne wykorzystanie zasobów procesora i przerzucenie żmudnych i czasochłonnych obliczeń na algorytmy zrealizowane sprzętowo w strukturze FPGA, pozwalając dowolnie „budować" elementy cyfrowego systemu przetwarzania sygnałów. Oprócz tego udostępniony jest szereg analogowych oraz cyfrowych układów wejściowo-wyjściowych, jak również niezbędne interfejsy komunikacyjne .

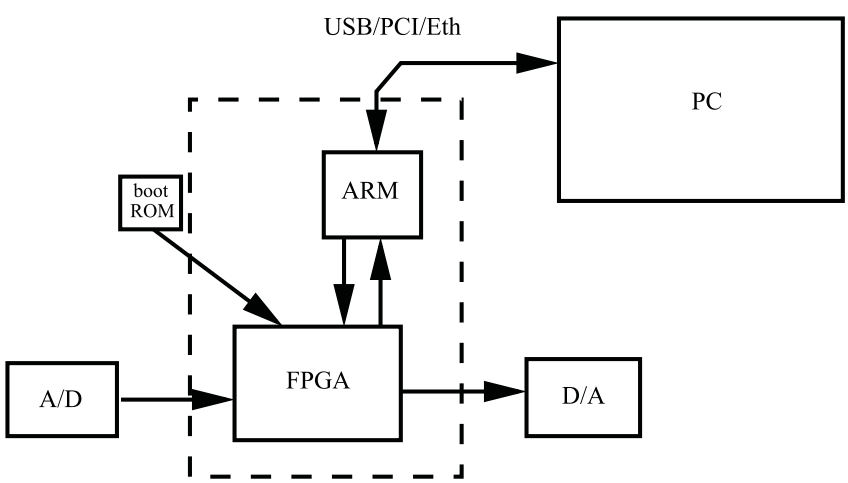
Spośród dostępnych z poziomu FPGA układów we/wy, w laboratorium będziemy wykorzystywali wyłącznie tor audio – przetworniki A/C i C/A. Przy założeniu, że użytkownik nie używa innych wejść analogowych, maksymalna możliwa do uzyskania częstotliwość próbkowania toru wynosi 250 kS/s.

Sercem platformy myRio jest układ SoC (ang. *System on a Chip*) firmy *Xilinx* serii *Zynq-7010*. Na jego pokładzie (jak już wspomniano wcześniej) znajduje się dwurdzeniowy mikroprocesor ARM®*Cortex*™*-A9* oraz układ FPGA z rodziny

Rysunek 8.2. Zdjęcie laboratoryjnego zestawu uruchomieniowego NI myRIO-1900

Artix®-7 zawierający `33 tys.` programowalnych komórek logicznych i `90` programowalnych bloków `DSP48E1`, co plasuje go na pozycji układu średniej klasy złożoności. W typowym zastosowaniu platforma połączona jest interfejsem USB z komputerem PC (rys. 8.3).

Przy pracy samodzielnej (bez PC) konfiguracja FPGA może być przechowywana w miniaturowym układzie ROM na tej samej płytce drukowanej.



Rysunek 8.3. Współpraca elementów zestawu MyRio i PC

8.4. Eksperymenty do wykonania w laboratorium

8.4.1. Proste implementacje filtrów

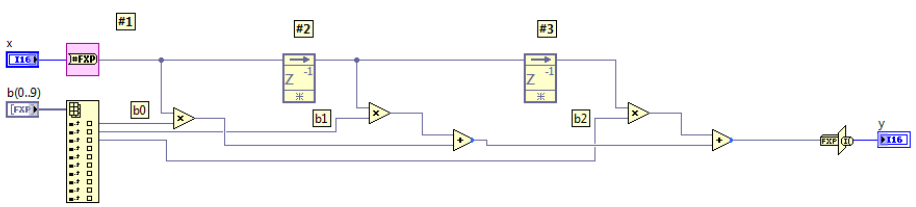
Podłącz MyRio kablem USB do komputera i dołącz zasilacz. Na ekranie pojawi się „MyRio USB Monitor” – zamknij go.

8.4.1.1. Trywialny filtr SOI

Uruchomimy prosty filtr SOI na układzie FPGA. Będzie to filtr drugiego rzędu, na początek z wektorem współczynników [1, 0, 0] – faktycznie będzie on więc przepuszczał sygnał wprost (potem go trochę skomplikujemy). Dla oszczędności czasu użyjemy gotowej (prekompilowanej) konfiguracji FPGA.

W przygotowanej konfiguracji używamy 12 bitów do reprezentowania danych (tyle ma przetwornik); miejscami jednak specyfika LabView powoduje, że musimy użyć 16 bitów.

– Skopiuj folder `D:/LMyRio/FIR-FPGA` na pulpit. Pozwoli Ci to na edycję Twojej kopii projektu. Jeśli folder docelowy już istnieje, nadpisz go.
– Otwórz folder FIR-FPGA[2] i kliknij `CYPS_MyRio_FIRFPGA.lvproj`
Rozwiń `MyRio-1900` i otwórz `RT_Main.vi` – to jest kod dla procesora ARM, który będzie ładował konfigurację do FPGA, przekazywał współczynniki i wyświetlał sygnały.
Rozwiń `Chassis MyRio` ⟶`FPGA Target`, otwórz `FPGA Main.vi` (główne „opakowanie” kodu dla FPGA) oraz `FPGA FIR.vi` (właściwy kod przetwarzania sygnału). Widzisz w tej chwili „front panel” czyli deklaracje wejść i wyjść.
W każdym z okien dotyczących FPGA naciśnij Ctrl-E aby otworzyć widok implementacji. Spróbuj zorientować się w tym kodzie, szczególnie przyjrzyj się implementacji filtru. W graficznym opisie implementacji filtru dwa elementy



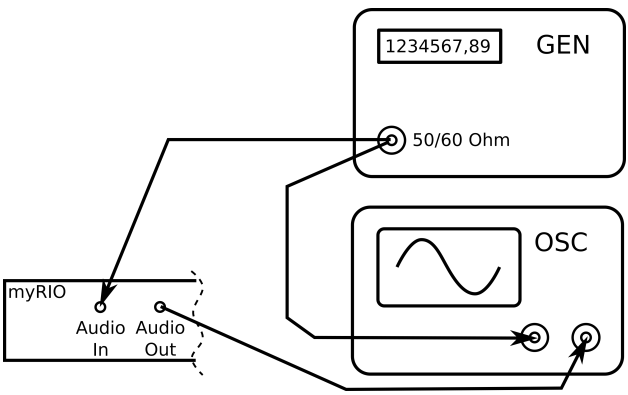
Rysunek 8.4. Implementacja filtru SOI w FPGA

„FXP” są tylko znacznikami dla kompilatora, że obliczenia będą wykonywane na liczbach stałoprzecinkowych (i definiują liczbę bitów oraz pozycję przecinka). Prostokąt z konektorem „b(0..9)” definiuje interfejs, przez który można dostarczyć współczynniki z procesora ARM. Reszta jest – mamy nadzieję – oczywista (ale spróbuj zrozumieć jak to działa).

[2] „FIR filter” to angielska nazwa filtru SOI (Finite Impulse Response).

Zwróć uwagę, że w schemacie użyto generycznych elementów cyfrowych – rejestrów, układów mnożących i sumatorów. Umyślnie nie użyliśmy tu bloków „DSP48" (specyficznych dla Xilinx'a) – po pierwsze, chcieliśmy pokazać schemat w sposób uniwersalny, niezależny od firmowych szczegółów; po drugie, kompilator na pewno zaimplementuje to, wykorzystując DSP48 lepiej, niż my byśmy to zrobili[3].

- Dołącz generator i oscyloskop do gniazdek mini-jack w MyRio (patrz rys. 8.5). Użyj prawego kanału – **czarnej** wtyczki BNC. Lewa (czerwona) przechodzi w domyślnej konfiguracji bez filtracji.

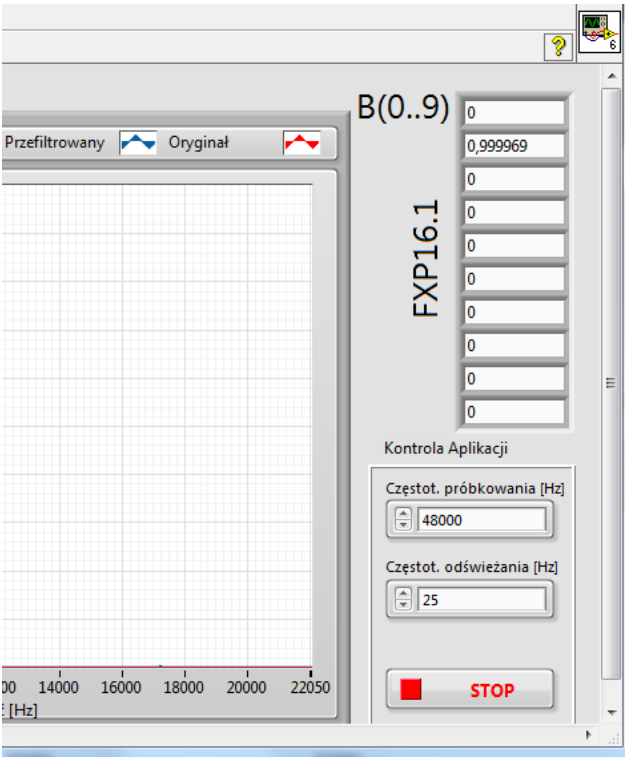


Rysunek 8.5. Połączenia do testowania filtru FPGA

- Sprawdź współczynniki i częstotliwość próbkowania w oknie RT_Main. Zauważ, że nie da się wpisać „+1" – w stałoprzecinkowej reprezentacji ułamkowej największą możliwą wartością jest $1 - 2^{-15}$ (i takiej użyj).
- Uruchom cały kod LabView wciskając „Run" **w oknie RT_Main**. Ta akcja uruchomi program na PC, prześle oprogramowanie do procesora ARM i uruchomi go (jeśli zapyta, potwierdź „Save"), a na koniec załaduje kod konfiguracyjny do FPGA.
- Upewnij się, że sygnał, który badasz, *rzeczywiście* przechodzi przez MyRio (zmień częstotliwość, rozłącz kable itp.).
- Naszkicuj / Odpowiedz ? ***Narysuj*** sygnał wyjściowy z oscyloskopu dla sinusoidy o niewielkiej częstotliwości (mniej niż 1/10 f_s). Przyjrzyj się efektowi „schodków". ***Odpowiedz na pytanie*:** Z czego ten efekt może wynikać?
  Wskazówka: Użyj przycisku RUN/STOP na oscyloskopie do zatrzymania obrazu.
- Przełącz generator na sygnał o kształcie prostokątnym i spróbuj zmierzyć opóźnienie między sygnałem wejściowym i wyjściowym (wybierz jako punkt odniesienia np. połowę zbocza „skoku"). ***Zanotuj*** zmierzoną wartość i wyraź ją

Zanotuj

[3] Oczywiście, doświadczony inżynier może osiągnąć lepszy wynik niż kompilator – w przyszłości na pewno będziecie doświadczonymi inżynierami…



Rysunek 8.6. Okno RT_main z panelem zadawania współczynników

następnie jako procent okresu próbkowania. **Wskazówka:** Upewnij się, które zbocze na wyjściu odpowiada danemu zboczu w sygnale wejściowym – np. zmieniając częstotliwość sygnału.

- Zmień współczynniki na [0, 1, 0] i zmierz (oraz ***zanotuj***) opóźnienie ponownie. **Wskazówka:** Aby wczytać nowe współczynniki musisz zatrzymać przetwarzanie (przycisk STOP) i uruchomić je ponownie. Zanotuj
- Przełącz się znów na sinusoidę i sprawdź, co się stanie gdy przekroczysz częstotliwość Nyquista. Użyj różnych kształtów sygnału. ***Opisz efekty dla sygnału trójkątnego***. Zanotuj

8.4.1.2. Prosty filtr SOI

Za pomocą Matlaba zaprojektuj współczynniki filtru 2. rzędu o charakterystyce wycinającej zadaną częstotliwość (ang. *notch filter*).

- W zależności od numeru stanowiska, niech Twój filtr tłumi sygnały o częstotliwości unormowanej równej *nrStanowiska*/30. **Wskazówka:** Przy tak niskim rzędzie narzędzia typu `fdatool()` na pewno zawiodą; użyj `poly()` i `freqz`, ustaw parę zer sprzężonych $re^{\pm j\theta}$ na okręgu jednostkowym dla żądanej częstotliwości.

- Gdy już uzyskasz współczynniki w wektorze *wierszowym* `B`, sprawdź (poleceniem `zplane(B,1)`) gdzie faktycznie umieściłeś zera. Możesz także użyć `zplane(z,0)`, gdzie w wektorze *kolumnowym* `z` umieścisz swoje zera[4].
- Zanotuj  ***Zapisz*** współczynniki, i zanotuj kod Matlabowy użyty do zaprojektowania filtru. ***Naszkicuj*** teoretyczną charakterystykę amplitudy w funkcji częstotliwości.
- Naszkicuj  Wpisz współczynniki do implementowanego filtru. Zauważ, że reprezentacja stałoprzecinkowa zmusi Cię zapewne do przeskalowania współczynników. **Wskazówka:** Nie skaluj *położenia zera*, tylko *współczynniki filtru!*
- Używając generatora i oscyloskopu sprawdź charakterystykę filtru w 2–4 wybranych punktach (używaj sygnałów wejściowych o amplitudzie poniżej 1 V); ***zaznacz wyniki na rysunku charakterystyki idealnej***.

Naszkicuj 

8.4.1.3. Własny filtr

- Otwórz `FPGA FIR.vi` i zmodyfikuj go, aby uzyskać wyższy rząd filtru (na przykład 6).
  - Użyj Ctrl-c Ctrl-v aby rozmnożyć elementy, dorysuj połączenia.
  - Rozciągnij myszką obiekt, który dostarcza współczynników.

  Zmieniłeś połączenia bloków w FPGA (i uaktywniłeś nowe bloki) – musisz *skompilować nową konfigurację FPGA*. Znajdź okno całości projektu (*.lvproj). Rozwiń `Project...` ⟶`MyRio....` ⟶`Chassis` ⟶`FPGA target` ⟶`Build specification`, kliknij prawym przyciskiem myszy na `FPGA FIR Build` i wybierz `Rebuild`; potwierdź „OK”.

  Kompilacja potrwa **około 5 minut** – zrób sobie przerwę albo przełącz się na Matlaba i zaprojektuj współczynniki filtru.

  Uwaga: uruchomienie kompilacji inaczej, niż opisano powyżej, może dać w efekcie (po ww. pięciu minutach) wspaniały kod programujący FPGA, ale nie „podłączony” do całego środowiska z LabView na PC i procesorem ARM – ostatecznie więc nasz filtr nadal zadziała że „starym” kodem.
- Zaprojektuj filtr według własnego pomysłu – wybierz np. spośród:
  - LP o częstotliwości odcięcia 4 kHz (dźwięk będzie brzmiał jak z telefonu),
  - BP 1–4 kHz,
  - ...

  Użyj np. `fdatool()`.

[4] To, że ta funkcja działa inaczej przy argumentach wierszowych, a inaczej przy kolumnowych, jest pięknym przykładem, co potrafi zrobić informatyk, aby ~~uprościć sobie~~ uprzykrzyć innym życie.

> Narzędzie `fdatool()` daje współczynniki wyskalowane tak, aby nigdy nie było przepełnienia – za to tracimy na dynamice i nie zobaczymy efektów przepełnienia. Zaleca się poprawić dynamikę (przeskalować współczynniki w górę np. $2\times$, albo do „prawie jedynki").

- (Sprawdź, czy filtr już się skompilował!) Skopiuj współczynniki do okna sterowania FPGA na PC i uruchom program („Run" **w oknie RT_Main**) – to załaduje nową konfigurację do FPGA.
  Wskazówka: Przepisując współczynniki, zastanów się ile cyfr znaczących kopiować, skoro Twój filtr używa 12 bitów.
- Na koniec przetestuj charakterystykę generatorem i oscyloskopem. W sprawozdaniu ***narysuj*** żądaną charakterystykę i zaznacz wyniki testowania. Naszkicuj
  Wskazówka: w `fdatool` użyj „Export ⟶coefficients ⟶to workspace", a w Matlabie zobaczysz współczynniki w zmiennej, którą da się wydrukować `disp()`.

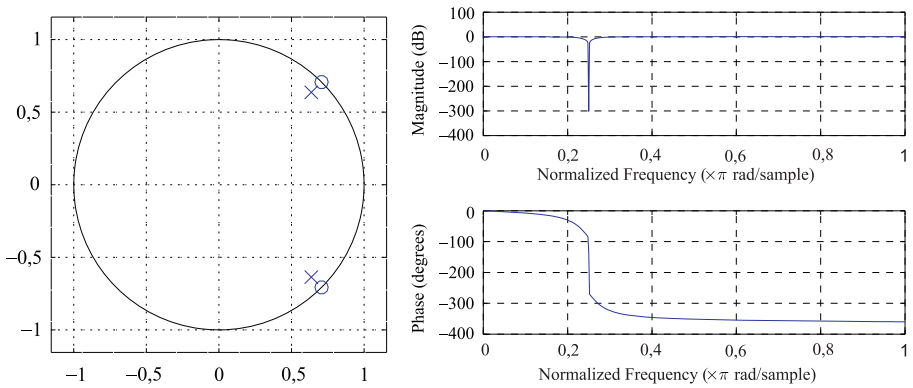
8.4.1.4. Efekty wielkosygnałowe

Sprawdź zachowanie filtru przy większych amplitudach sygnału wejściowego.

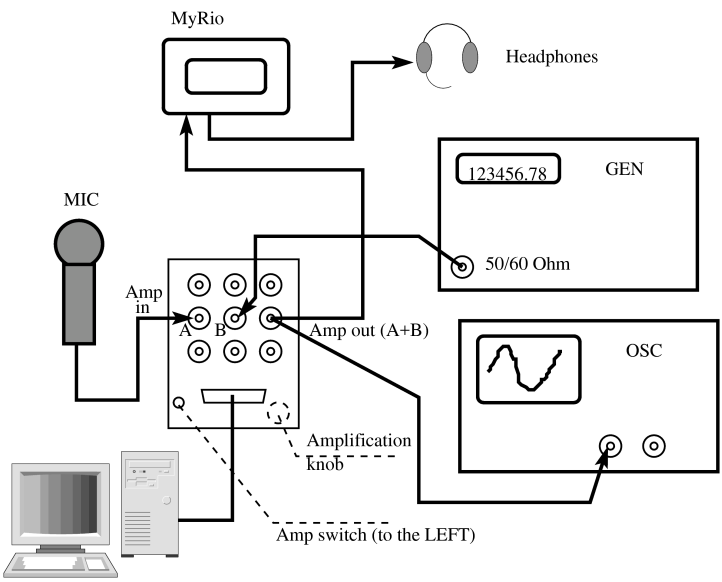
- ***Zanotuj*** maksymalną amplitudę, przy której sygnał wyjściowy jest jeszcze poprawny dla wszystkich użytecznych częstotliwości. ***Naszkicuj*** efekt wynikający z „zawijania" liczb po przekroczeniu maksymalnej amplitudy. Zanotuj Naszkicuj
- Przeskaluj współczynniki tak, aby filtr poprawnie pracował przy sygnale wejściowym o maksymalnej amplitudzie (wg specyfikacji jest to ± 2.5 V). ***Zanotuj przeskalowane współczynniki***. Zauważ i ***zanotuj*** co stało się z amplitudą na wyjściu. Zanotuj Zanotuj
- Sprawdź właściwości bloków arytmetycznych (prawy klik na bloku ⟶ „properties" ⟶ "output configuration"). ***Naszkicuj*** schemat filtru i zaznacz, jakie amplitudy mogą wystąpić na poszczególnych węzłach (przy zaprojektowanych przez Ciebie wartościach współczynników – oryginalnych, nie przeskalowanych w dół). Spróbuj odpowiednio zwiększyć liczbę bitów w blokach arytmetycznych, przełącz też sumatory w tryb nasycenia („saturation"). Naszkicuj
- Przekompiluj filtr (tak, kolejne 5 minut czekania) i sprawdź ponownie. ***Zanotuj*** o ile udało się zwiększyć zakres dynamiczny filtru. Zanotuj
  Wskazówka: Zakres dynamiczny to stosunek największego do najmniejszego poprawnie przetwarzanego sygnału. Ograniczeniem może tu być szum analogowy lub cyfrowy. **Bardzo przybliżoną** miarę zakresu dynamicznego można uzyskać, zakładając, ze szum cyfrowy (zaokrągleń) będzie miał amplitudę najmniej znaczącego bitu na wyjściu – więc im większy dozwolony sygnał na wyjściu, tym lepszy zakres dynamiczny.

8.4.1.5. Zadanie extra Bikwadratowy (NOI) filtr wycinający

- Zmień układ na NOI – skopiuj na pulpit `D:/LMyRio/IIR-FPGA`, a jako główny program otwórz `CYPS_MyRio_IIRFPGA`.
- Zaprojektuj filtr wycinający NOI 2. rzędu – taki filtr ma parę zer na okręgu jednostkowym i parę biegunów na tej samej częstotliwości, lecz o promieniu nieco mniejszym od jedności (rys. 8.7).



Rysunek 8.7. Przykładowy filtr wycinający (notch) i jego charakterystyka uzyskana funkcją `freqz()`



Rysunek 8.8. Połączenia z zakłóceniem wąskopasmowym

- Użyj swojego filtru do wycinania zakłócenia wąskopasmowego, którym będzie dosumowany sygnał sinusoidalny z generatora (rys. 8.8).
- Zastanów się i zbadaj, jak blisko okręgu jednostkowego można umieścić biegun, aby błędy zaokrągleń nie spowodowały jakiejś katastrofy (pamiętaj, że filtr zbudowany jest z arytmetyką 12-bitową).

Uwaga na uszy! Nigdy nie przełączaj układu ani jego parametrów ze słuchawkami na uszach – zawsze je nieco odsuń. Zawsze zaczynaj od włączonego tłumika „Att -20 dB" na generatorze.

8.4.1.6. Zadanie extra Kaskada NOI

Zaprojektuj filtr NOI wyższego rzędu i zrealizuj go w strukturze kaskady sekcji bikwadratowych.

- Jak powielić opis sprzętu – już wiesz z poprzedniego zadania.
- W Matlabie jest gotowa funkcja do przekształcenia współczynników klasycznego filtru NOI na współczynniki kaskady: `LCPS_casc_biquad()`.