

## LABORATORIUM CYFROWEGO PRZETWARZANIA SYGNAŁÓW

### Ćwiczenie 7

## **Implementacja algorytmów DSP na procesorze sygnałowym**

### **1. Cel ćwiczenia**

- Zapoznanie się z architekturą karty C6713 DSK i środowiskiem IDE Texas Instruments Code Composer Studio.
- Zdobywanie umiejętności programowania procesora DSP za pomocą schematów blokowych Simulinka

### **2. Wprowadzenie**

Karta C6713 DSK firmy Spectrum Digital, oparta na procesorze sygnałowym TMS320CC6713 firmy Texas Instruments, jest jedną z najpopularniejszych platform startowych do nauki programowania i implementacji aplikacji na procesory sygnałowe DSP. Popularność karty wynika z dostępności zintegrowanego środowiska uruchomieniowego TI Code Composer Studio (CCS) zawierającego zoptymalizowany kompilator C/C++, dostępności licznych schematów i not aplikacyjnych różnych kart rozszerzeń oraz bogatej literatury z licznymi przykładami programów (głównie w języku C), np. [1-3], a także wspomaganie tworzenia aplikacji na kartę przez szeroko wykorzystywane inżynierskie środowiska programistyczne MATLAB-Simulink i LabVIEW. Dzięki temu istnieje możliwość szybkiego tworzenia aplikacji na kartę w formie schematów blokowych Simulinka lub schematów graficznego języka LabVIEW nawet bez umiejętności programowania w języku C (a tym bardziej w assemblerze), czy znajomości szczegółów architektury karty.

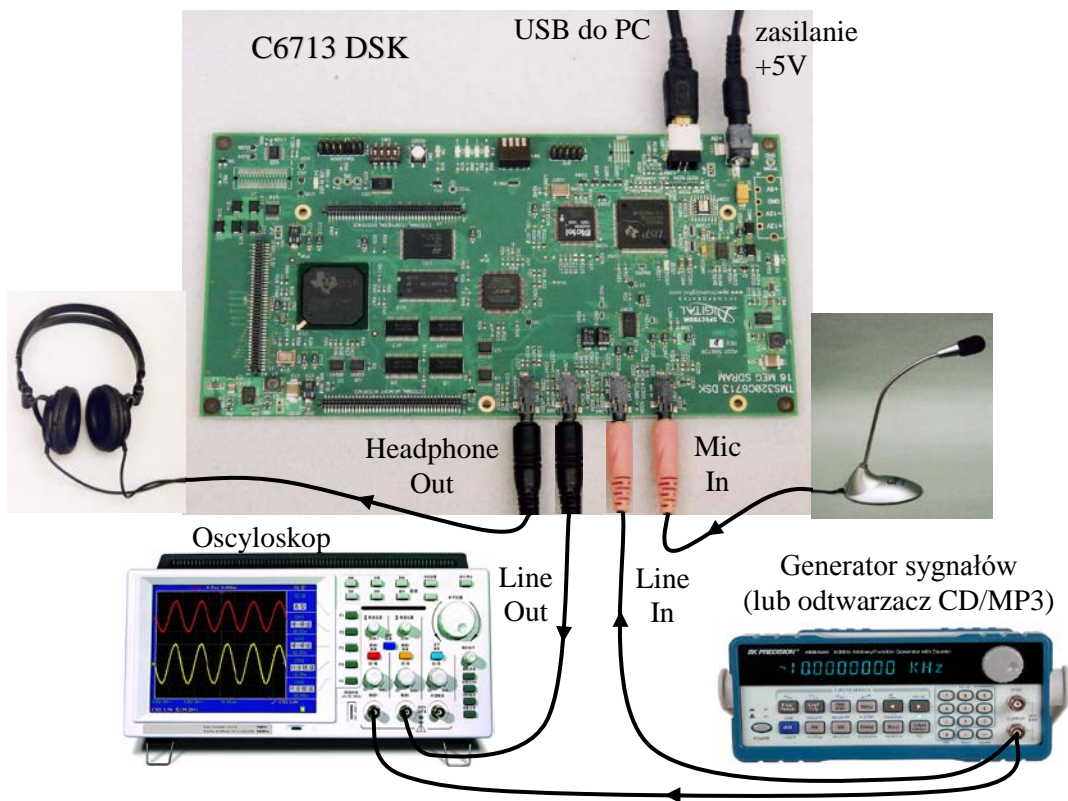
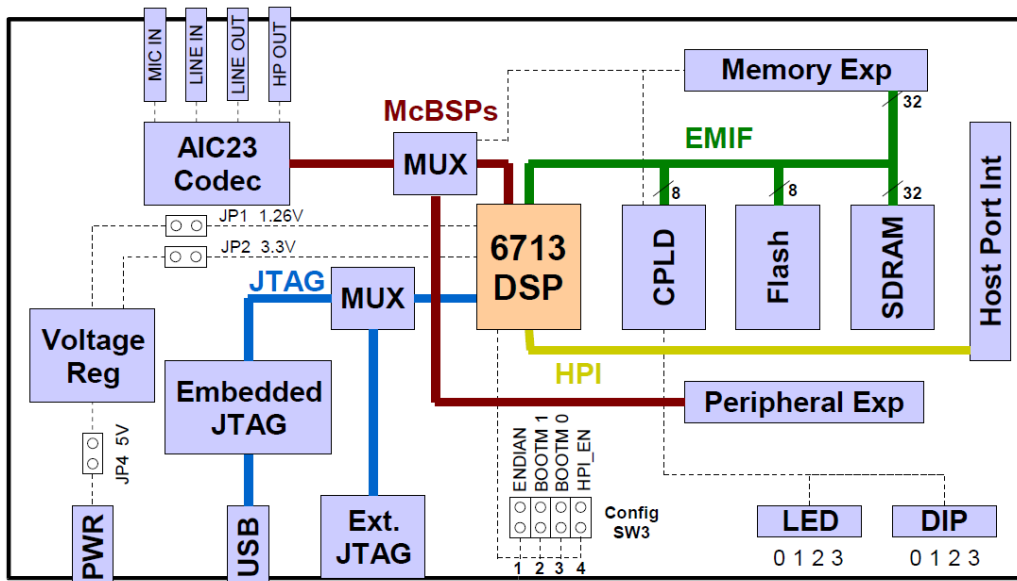
Karta C6713 DSK jest wyposażona w zestaw układów peryferyjnych umożliwiających rozbudowę jej funkcjonalności dla aplikacji o różnych wymaganiach sprzętowych przez dołączanie kart-córek. Schemat funkcjonalny karty i zdjęcie strony komponentów z typowym sposobem dołączania urządzeń zewnętrznych są przedstawione na rys. 2.1.

#### **2.1. Komponenty i cechy funkcjonalne karty C6713 DSK**

Karta C6713 DSK jest wyposażona w:

- 32-bitowy zmiennoprzecinkowy procesor sygnałowy Texas Instruments TMS320C6713 DSP (częstotliwość zegara 225 MHz, instrukcje VLIW (*Very Long Instruction Word*), 6 jednostek ALU (w tym 2 Floating-point, 2 MAC), moc obliczeniowa do 1800 MIPS / 1350 MFLOPS,
- kodek stereo AIC23 umożliwiający bezpośrednią implementację aplikacji do przetwarzania sygnałów audio,
- 16 MB synchronicznej pamięci DRAM (32-bitowa magistrała, częstotliwość 90 MHz),
- 512 KB pamięci nieulotnej Flash (8-bitowej, 256 KB do dyspozycji w konfiguracji domyślnej),
- 4 dostępne dla użytkownika diody LED i 4 mikroprzełączniki DIP,
- programowalny układ CPLD (*Complex Programmable Logic Device* firmy Altera) do konfiguracji karty za pomocą zapisów w rejestrach,
- 80-pinowe złącza rozszerzeń do dołączania kart-córek (zewnętrznej pamięci, przetworników itp.),

- emulator JTAG na z interfejsem USB hosta (do komunikacji karty ze środowiskiem CCS działającym na komputerze PC), z możliwością dołączenia emulatora zewnętrznego przez złącze JTAG,
- pojedyncze złącze napięcia zasilania +5V; stabilizatory na karcie wytwarzają napięcia +1.26V dla rdzenia procesora DSP i +3.3V dla układów I/O.

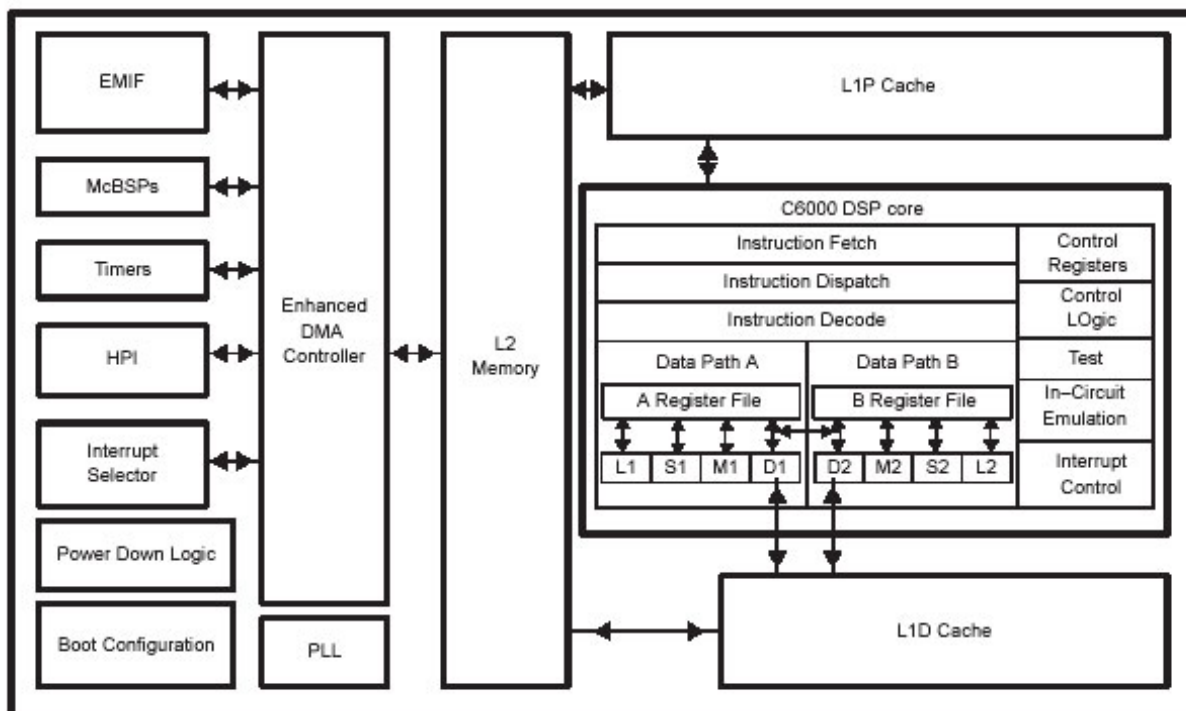


Rys. 2.1. Schemat funkcjonalny karty C6713 DSK i zdjęcie strony komponentów z typowym sposobem dołączania urządzeń zewnętrznych

Procesor C6713 jest zintegrowany w pojedynczym układzie scalonym mikrokontrolerem (rys. 2.2) o architekturze optymalizowanej pod kątem przetwarzania sygnałów audio. Oprócz podstawowego rdzenia procesorów rodziny C6000 układ zawiera 4 KB pamięci cache L1 programu i 4 KB dwukierunkowej pamięci cache L1 danych, 256KB pamięci L2, interfejs EDMA (*Enhanced Direct Memory Access*), 16-bitowy interfejs hosta HPI (*Host-Port Interface*), dwa szybkie porty szeregowo McBSP (*Multichannel*

Buffered Serial Port), dwa 32-bitowe liczniki-timery ogólnego zastosowania oraz interfejs 32-bitowej magistrali EMIF (External Memory InterFace) do komunikacji z pamięciami zewnętrznymi. Do magistrali EMIF dołączone są pamięci SDRAM i Flash karty, układ CPLD oraz złącza kart rozszerzeń.

Procesor może wykonywać do 8 instrukcji na jeden cykl zegarowy, obsługuje natywnie instrukcje zgodne z IEEE754, arytmetykę z nasyceniem i kontrolą przepelnienia (co jest bardzo potrzebne do implementacji algorytmów DSP), 32/64-bitowe słowa danych, liczby o pojedynczej i podwójnej precyzji.



Rys. 2.2. Architektura procesora C6713

Karta C6713 DSK jest wyposażona w 16 MB zewnętrznej pamięci SDRAM adresowanej bajtowo. Konfigurowany programowo kontroler pamięci jest zintegrowany w EMIF. Pamięć pracuje z częstotliwością 90 MHz (1/5 częstotliwości PLL równej 450 MHz). Pamięć Flash jest skonfigurowana jako 256K słów 16-bitowych w celu umożliwienia opcji 16-bitowego bootowania karty DSK. Jednak oprogramowanie dostarczane z DSK ma domyślny tryb bootowania 8-bitowego i w takiej konfiguracji starsze 8 bitów pamięci Flash jest ignorowane. Przestrzeń adresowa pamięci karty DSK jest przedstawiona na rys. 2.3.

Address	C67x Family Memory Type	6713 DSK
0x00000000	Internal Memory	Internal Memory
0x00030000	Reserved Space or Peripheral Regs	Reserved or Peripheral
0x80000000	EMIF CE0	SDRAM
0x90000000	EMIF CE1	Flash
0xA0000000	EMIF CE2	CPLD
0xB0000000	EMIF CE3	Daughter Card

0x90080000

Rys. 2.3. Przestrzeń adresowa pamięci procesora na karcie C6713 DSK

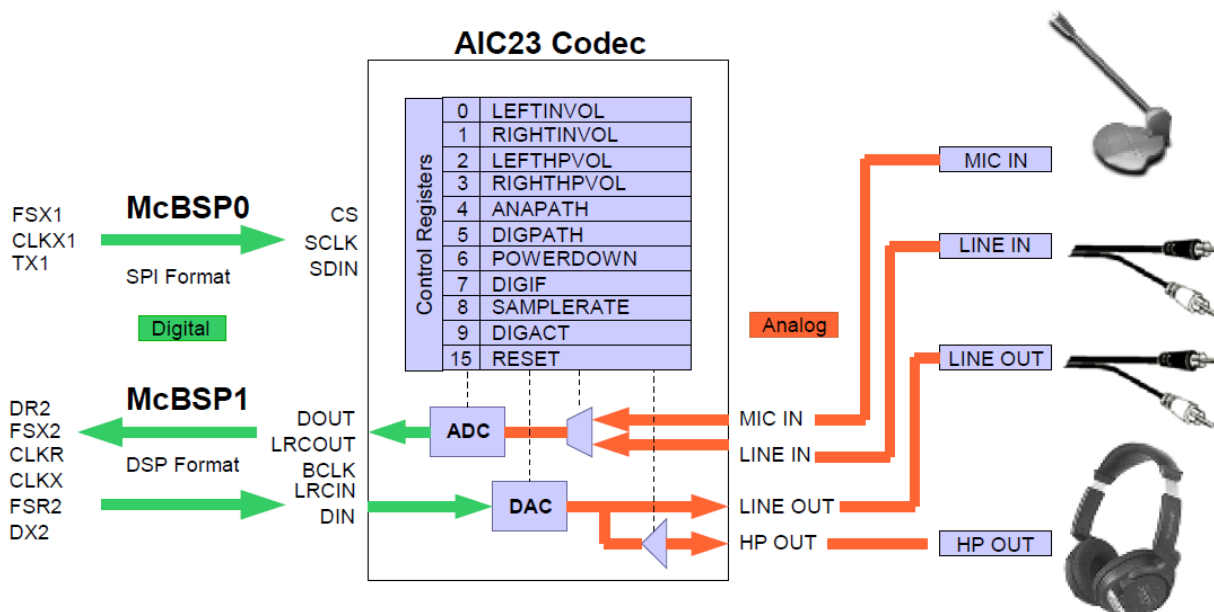
Analogowe sygnały audio są przetwarzane przez kodek AIC23 wyposażony w 4 standardowe złącza stereo jack 3.5mm (rys. 2.4). Jako aktywne źródło sygnału wejściowego można wybrać wejście liniowe lub wejście mikrofonowe. Wyjściowe sygnały analogowe są przekazywane zarówno na wyjście liniowe jak i na wyjście głośnikowe o programowanym wzmacnieniu.

Przetwarzanie ADC (analogowo-cyfrowe) i DAC (cyfrowo-analogowe) realizują (wielobitowe) przetworniki  $\Sigma$ - $\Delta$  (Sigma-Delta) (SNR=90 dB dla ADC i 100 dB dla DAC), które mogą działać z częstotliwością próbkowania 8/32/44.1/48/96 kHz i operować na słowach o długości 16/20/24/32 bitów. Przetwornik ADC ma układ kształtowania szumu kwantowania 3-go rzędu, pracuje z oversamplingiem i decymacją do zadanej częstotliwości próbkowania, co efektywnie ogranicza częstotliwość sygnału do pasma Nyquista i działa jak filtracja antyaliasingowa. Przetwornik DAC realizuje z kolei interpolację do wysokiej częstotliwości, co zapewnia prawie idealną rekonstrukcję sygnału na wyjściu.

Komunikacja kodeka z procesorem DSP odbywa się poprzez dwa porty szeregowy McBSP (rys. 2.5) połączone z portami McBSP procesora i działające w trybie synchronicznym. Port McBSP0 służy do jednokierunkowego wysyłania poleceń do rejestrów sterujących kodeka, natomiast cyfrowe dane audio są przesyłane w obu kierunkach przez port McBSP1. Porty McBSP mogą być programowo przełączone do złącz rozszerzeń karty DSK (np. w celu dołączenia zewnętrznego przetwornika ADC/DAC).

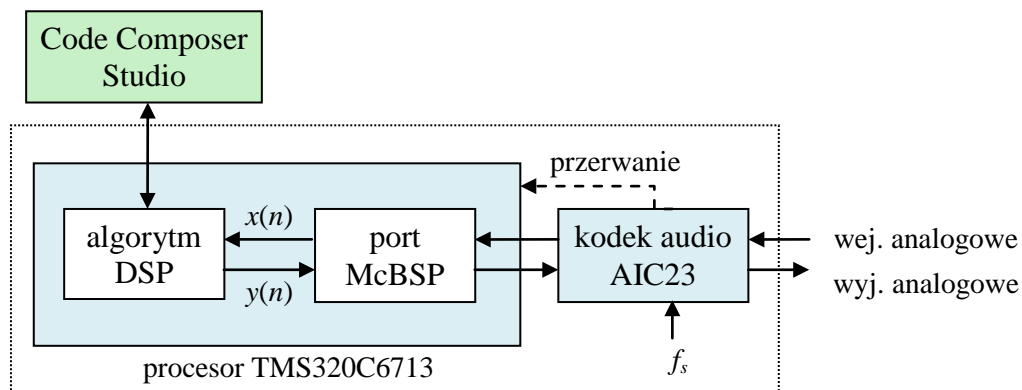
Przetwarzanie sygnału przez kodek jest sterowane przerwaniem. W każdej chwili próbkowania kodek łączy dwie próbki z przetwornika ADC w formacie int16 (lewy/prawy kanał) w słowo uint32, przekazuje je do rejestru McBSP1 i generuje przerwanie do procesora. W procedurze obsługi przerwania ISR (*Interrupt Service Routine*) procesor odbiera dane z McBSP, wykonuje operacje przetwarzania (np. filtrację) i odsyła słowo wyjściowe przez McBSP do kodeka, który rozdziela słowo na części dla lewego i prawego kanału i realizuje przetwarzania DAC.

Przy częstotliwości przetwarzania  $f_s=48$  kHz czas przetwarzania nie może przekroczyć  $T_s=1/f_s=20.8$  usec. Przy jednym taktie zegarowym trwającym  $T_c=4.44$  nsec ( $1/225$  MHz) w okresie próbkowania mieści się ok. 4685 taktów zegarowych. Procesor może wykonywać do 8 instrukcji w jednym taktie zegara, czyli do najwyżej 37480 instrukcji w jednym okresie próbkowania (z tego połowę na jeden kanał), jeżeli nie jest obciążony żadnymi innymi operacjami.



Rys. 2.4. Kodek Audio AIC23 karty DSK

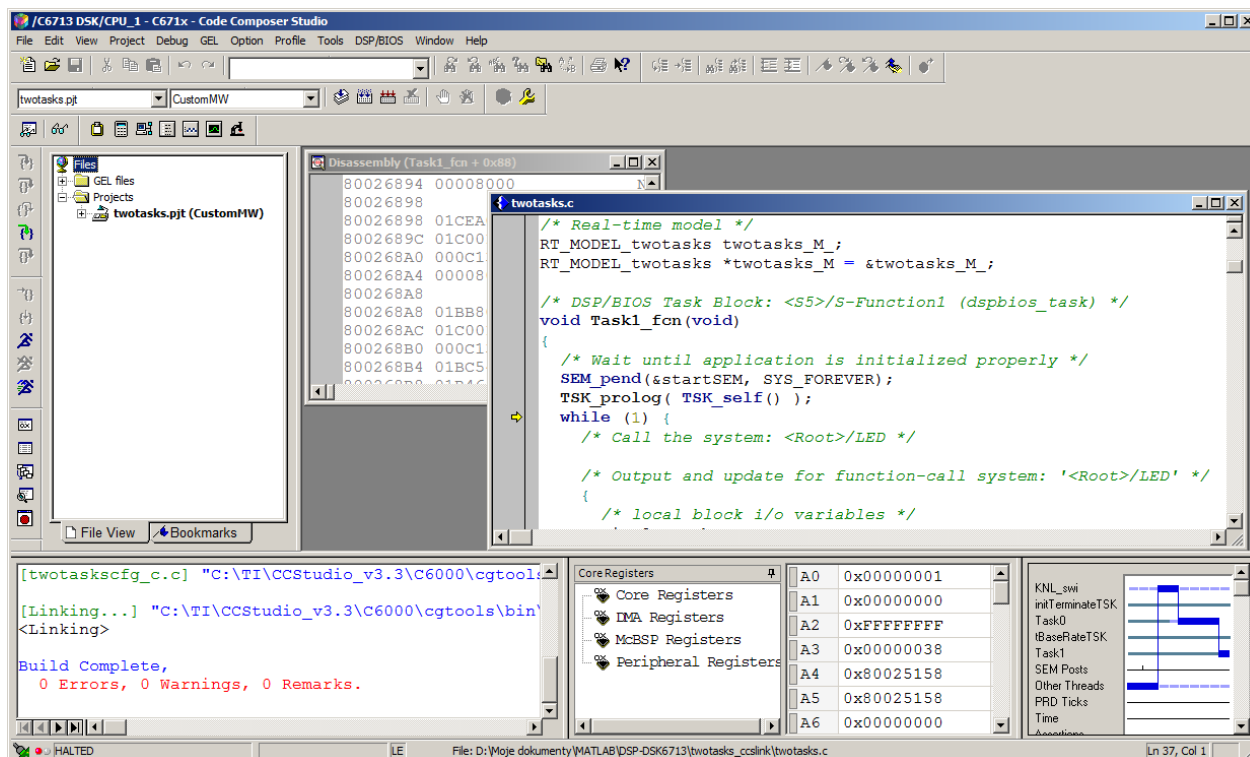
Programowalny układ CPLD (oparty na 100-pinowym układzie QFP typu EEPROM) służy do zaprogramowania połączeń logicznych integrujących komponenty karty i komponenty dołączanych do złączy rozszerzeń kart-córek. Konfiguracji dokonuje się przez odpowiednie zapisy do rejestrów sterujących lub odczyt z rejestrów statusu układu CPLD (4 rejestry mapowane do obszaru adresowego pamięci).



Rys. 2.5. Komunikacja procesora DSP C6713 z kodekiem audio AIC23

## 2.2. Code Composer Studio i MATLAB-Simulink

Code Composer Studio jest kompletnym środowiskiem IDE dla procesorów Texas Instruments. Zawiera zoptymalizowany kompilator C/C++, assembler, linker, debugger i loader programów. CCS komunikuje się z kartą DSK poprzez złącze USB 2.0. Oprócz cech ułatwiających programowanie, CCS umożliwia sterowanie i śledzenie programu wykonywanego przez procesor (odczyt i zapis rejestrów procesora i pamięci, pracę krokową, ustawianie punktów wstrzymania programu itd.) za pośrednictwem wbudowanego emulatora JTAG karty DSK.



Rys. 2.5. Główne okno Code Composer Studio v.3.3

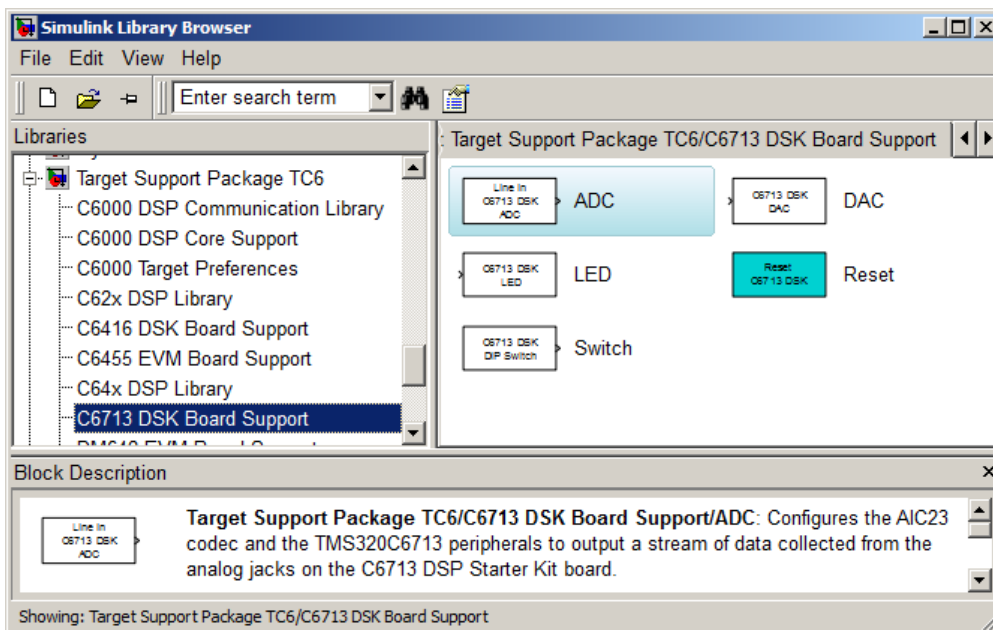
Jeżeli procesor na karcie DSK wykonuje program pod kontrolą systemu DSP/BIOS, to CCS (lub inna aplikacja działająca na komputerze hosta) może za pośrednictwem mechanizmu RTDX (*Real Time Data eXchange*) wymieniać z programem informacje (np. odczytywać wyniki lub modyfikować parametry) w czasie rzeczywistym bez jego zatrzymywania. Przepustowość kanału RTDX przez USB jest ograniczona do ok. 30-40 kB/sec, a jego obsługa obciąża procesor.

Do wymiany informacji pomiędzy środowiskiem MATLAB-Simulink a CCS służy biblioteka Matlaba **Embedded IDE Link CC** tworząca specjalne obiekty `ticcs` obsługujące komunikację z CCS. Główne składniki IDE Link CC to:

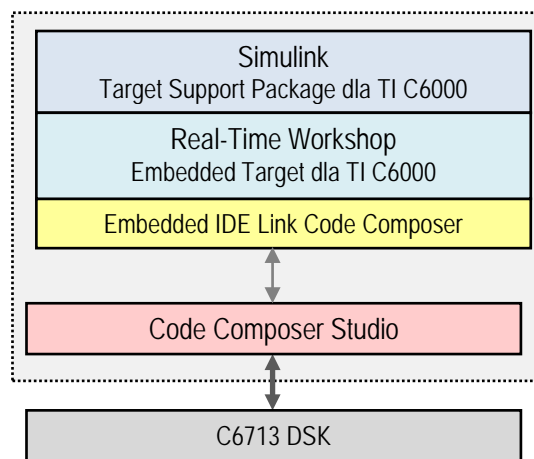
- Automation Interface – zestaw metod umożliwiających dostęp do funkcji i danych API CCS za pomocą funkcji Matlab, a za pośrednictwem CCS transfer danych do i z procesora na karcie DSK (w szczególności debugowanie programu i obsługę RTDX), oraz automatyzację tworzenia i budowania projektów CCS (definicji preprocesora, dołączania bibliotek, definicji GEL architektury itd.),
- Project Generator – zestaw metod wykorzystujących API CCS do budowania projektów CCS na określoną platformę (target) na podstawie wygenerowanego w Matlabie (za pomocą narzędzia Real-Time Workshop) kodu C reprezentującego np. model (schemat blokowy) Simulinka.

Oprócz tego IDE Link CC umożliwia weryfikację i profilowanie wygenerowanego kodu.

**Target Support Package dla TI C6000** dodaje do Simulinka bibliotekę bloków obsługujących funkcje procesorów rodziny C6000 i komponentów popularnych kart startowych z procesorami tej rodziny (rys. 2.6). Zarówno te bloki jak i prawie wszystkie inne bloki Simulinka mają reprezentację w C. Dzięki temu „translator” Real-Time Workshop (w nowszych wersjach Matlab: Simulink Coder) może przetworzyć program w formie schematu blokowego Simulinka na kod ANSI C przeznaczony do implementacji w czasie rzeczywistym. Embedded Target (Embedded Coder) dla TI C6000 dostarcza RTW API potrzebne do wygenerowania kodu przeznaczonego dla konkretnej platformy. Na podstawie tego kodu IDE Link CC może następnie automatycznie zbudować odpowiedni projekt dla CCS na C6713 DSK i załadować otrzymany po kompilacji kod wynikowy do pamięci procesora DSP (rys. 2.7).



Rys. 2.6. Biblioteka Simulinka Target Support Package TC6 z blokami obsługi komponentów karty C6713 DSK



Rys. 2.7. Schemat przetwarzania modelu Simulinka na aplikację procesora DSP C6713

### 2.3. System operacyjny czasu rzeczywistego DSP/BIOS

DSP/BIOS jest systemem umożliwiającym w czasie rzeczywistym szeregowanie z wywłaszczaniem (*pre-emptive scheduling*) i analizę wątków aplikacji uruchamianych na procesorach DSP Texas Instruments. Umożliwia to m.in. działanie mechanizmu RTDX, czyli wymianę w czasie rzeczywistym danych pomiędzy procesorem DSP a programem działającym na komputerze PC (host), np. w Matlabie.

Wątki (*threads*) są obiektami DSP/BIOS zawierającymi kod (funkcje) programu. Algorytm szeregowania decyduje, który wątek wykonywać ze względu na jego typ i priorytet. W aplikacjach działających pod DSP/BIOS można używać kilku typów wątków (wymienione według malejącego priorytetu):

- Przerwania sprzętowe HWI (*HardWare Interrupts*) – wątki o najwyższym priorytecie, których wykonanie jest wyzwalane przez przerwania procesora lub układów zewnętrznych. Działają one zawsze do ukończenia i nie mogą być wywłaszczane, dlatego powinny być używane do operacji krytycznych czasowo, a procedury ich obsługi (ISR) powinny być jak najkrótsze,
- Przerwania programowe SWI (*SoftWare Interrupts*) – wątki wyzwalane wewnątrz programu i o ustalonym poziomie priorytetu. Wątki SWI mogą być wywłaszczane przez SWI o wyższym priorytecie i przez HWI. Typowo SWI są tworzone przez HWI, dzięki czemu bardziej czasochłonne przetwarzanie może być wykonywane jako SWI i nie blokuje innych HWI.
- Funkcje periodyczne PRD – szczególny typ SWI, wyzwalane przez dedykowane timery sprzętowe.
- Zadania TSK (*Task*) – używane do operacji mniej krytycznych czasowo, mają ustalany poziom priorytetu. Wykonywanie zadań zaczyna się z chwilą uruchomienia aplikacji. Zadania mogą być też tworzone dynamicznie przez aplikacje DSP/BIOS i wtedy wykonywanie zaczyna się w chwili utworzenia. Zadania nie mogą być tworzone przez procedury obsługi wątków HWI czy SWI.
- Funkcje IDL (*Idle functions*) działające w czasie bezczynności (w tle) – są powtarzane w pętli jako części wątku aplikacji DSP/BIOS o najniższym priorytecie. Pętla IDL zawiera domyślnie funkcje, które przesyłają w czasie rzeczywistym dane analizy z DSP do hosta (umożliwiające np. podgląd programu w CCS).

Właściwości obiektów DSP/BIOS tworzących aplikację są konfigurowane w projekcie CCS.

Biblioteka Simulinka **Target Support Package TC6 | DSP/BIOS Library** zawiera bloki HWI, TSK i Triggered TSK, które w modelu Simulinka wyzwalają realizację funkcji (subsystemów) stanowiących ich procedury obsługi.


### 3. Zadania do wykonania na stanowisku laboratoryjnym z C6713 DSK

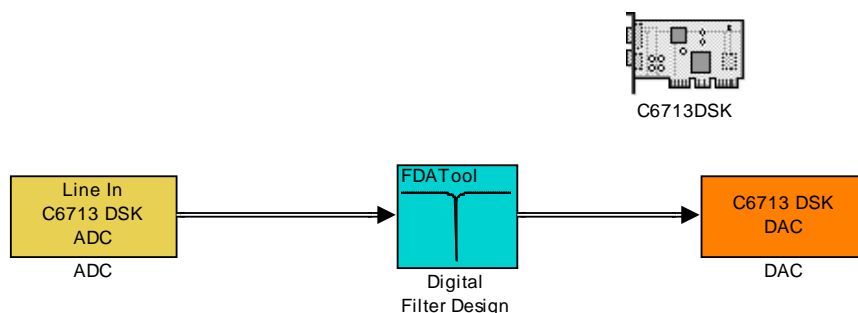
- W ćwiczeniu wykorzystywane są biblioteki symulacyjne Simulinka **Target Support Package TC6** oraz **Signal Processing Blockset** oraz środowisko uruchomieniowe **Code Composer Studio (CCS)** firmy Texas Instruments.
- Uruchomić Matlaba i w oknie **Current Folder** przejść do folderu **Moje dokumenty/MATLAB/DSP/C6713DSK**. Jest to folder roboczy dla ćwiczeń *Cyfrowego przetwarzania sygnałów* z kartą procesora sygnałowego.
- Sprawdzić zgodność połączeń karty C6713 DSK ze schematem na rys.2.1. Generator ma być ustawiony na generowanie sygnału sinusoidalnego o częstotliwości akustycznej. Wykorzystanie dwóch kanałów oscyloskopu umożliwia porównanie sygnału przed i po przetworzeniu
- Uruchomić program C6713DSK *Diagnostics* ze skrótu na pulpicie i przeprowadzić test diagnostyki układów karty DSK, w szczególności połączenia USB.
- Uruchomić generator i oscyloskop i sprawdzić, czy napięcie na wyjściu generatora podłączonym do wejścia liniowego karty DSK (i do oscyloskopu) nie przekracza typowego zakresu wejścia liniowego. **Dopuszczalny zakres napięcia peak-to-peak wynosi  $V_{pp}=\pm 1.0$  V. Nie wolno przekraczać takiego zakresu napięcia wejścia liniowego karty DSK** (pokrętko amplitudy AMP na panelu generatora funkcyjnego uniwersalnego przyrządu laboratoryjnego, rys.3.1).



Rys. 3.1. Panel generatora funkcyjnego uniwersalnego przyrządu laboratoryjnego

#### 3.1. Program filtracji w czasie rzeczywistym

- A. Otworzyć bibliotekę bloków Simulinka **Library Browser** (przycisk  na belce Matlaba), utworzyć nowy model Simulinka i zbudować schemat blokowy przedstawiony na rys. 3.2 przeciągając do okna schematu potrzebne bloki z odpowiedniej bibliotek elementów i łącząc odpowiednie wyjścia z wejściami liniami przepływu sygnałów za pomocą myszy (szczegółowy opis poniżej). Dwukrotne kliknięcie bloku otwiera okno parametrów bloku.

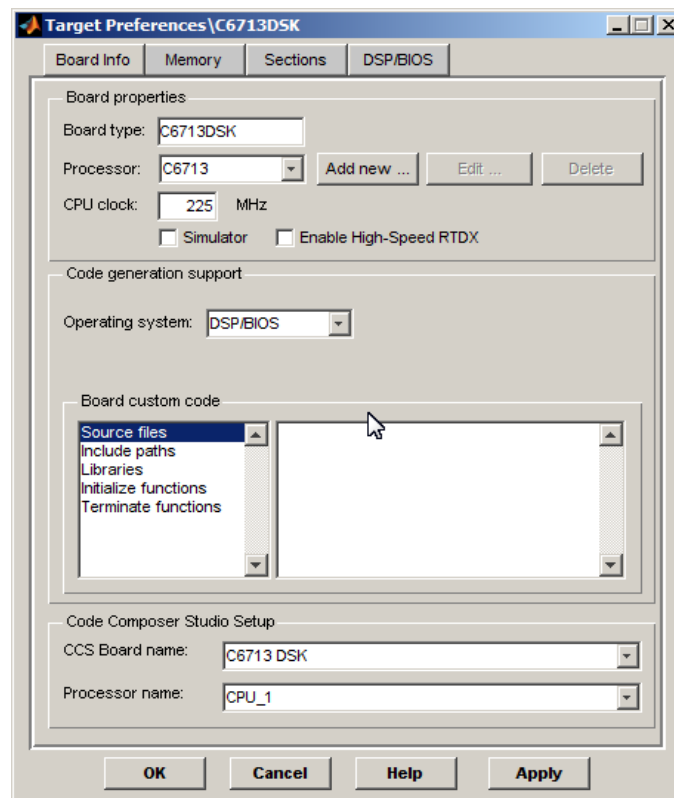


Rys. 3.2. Podstawowy schemat Simulinka filtracji cyfrowej na karcie C6713 DSK

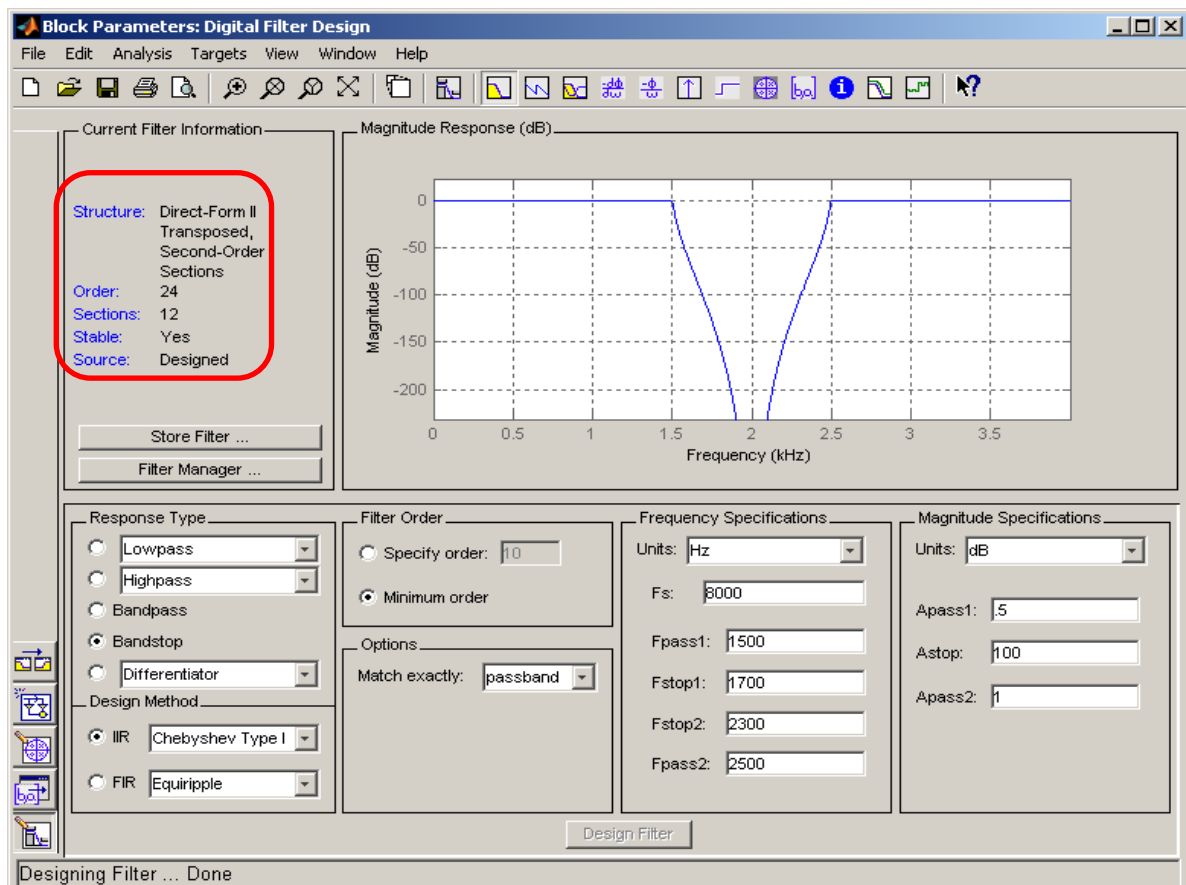
Bloki schematu i okna ich parametrów:



- C6713 DSK (biblioteka **Target Support Package TC6 | C6000 Target Platforms**) – blok określający platformę, na której ma działać model jako jedyne zadanie procesora pod kontrolą DSP/BIOS.



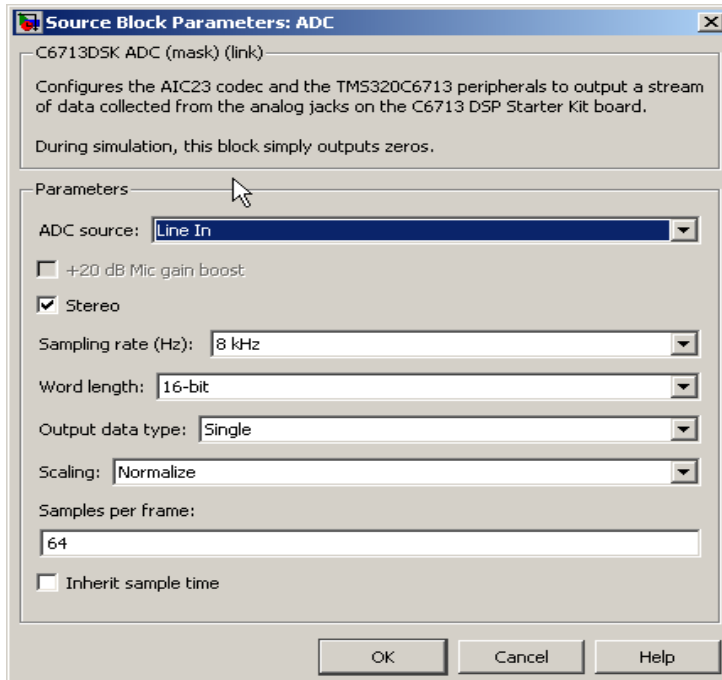
- Digital Filter Design (biblioteka **Signal Processing Blockset | Filtering | Filter Design**) – blok FDA Tool (Filter Design & Analysis) projektowania filtra cyfrowego do przetwarzania sygnału audio.



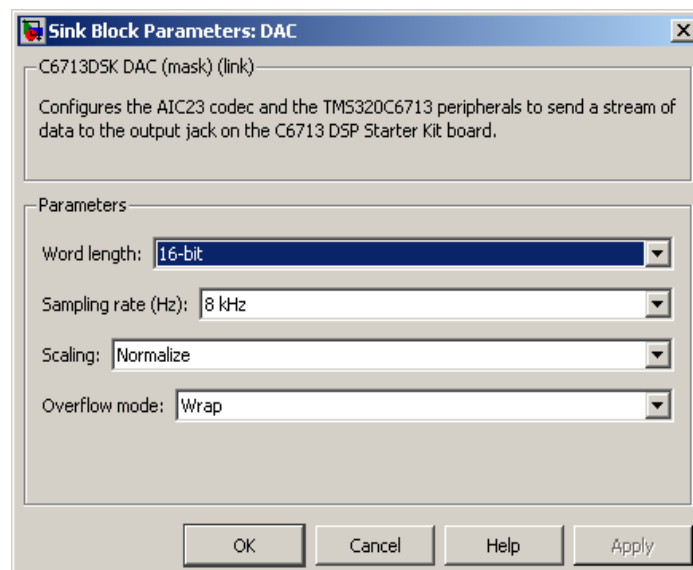
Zaprojektować filtr pasmowozaporowy (BS) Czebyszewa typu I o częstotliwości próbkowania  $F_s=8000$  Hz, końcu pierwszego pasma przepustowego  $F_{pass1}=1500$  Hz, początku drugiego pasma przepustowego  $F_{pass2}=2500$  Hz i tłumieniu w paśmie zaporowym  $A_{stop}=100$  dB. Rząd filtra i forma jego realizacji zostaną określone automatycznie (zgodnie z parametrami domyślnymi, patrz zaznaczone pole **Current Filter Information** w oknie bloku **FDA Tool**).

- **Line In C6713 DSK ADC** (biblioteka **Target Support Package TC6 | C6713 DSK Board Support**) – przetwarzanie analogowo-cyfrowe sygnału akustycznego (z wejścia liniowego **Line In** karty) przez kodek audio.

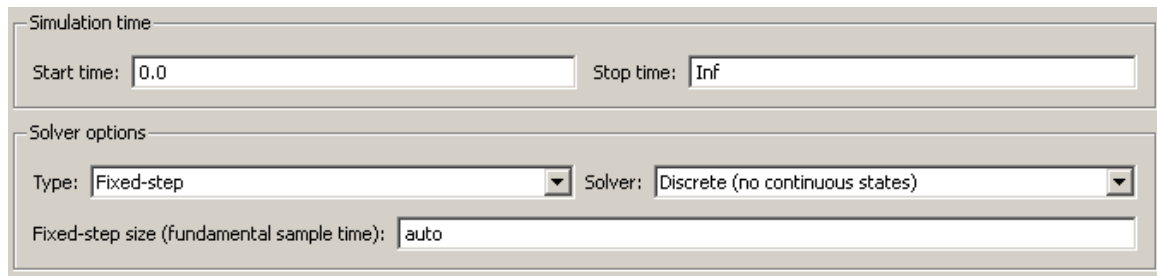
Uwaga: Jeżeli częstotliwość próbkowania  $F_s$  zaprojektowanego filtra jest określona w Hz (kHz, MHz), to częstotliwości próbkowania (**Sampling rate**) bloków przetwarzania ADC i DAC muszą być równe tej częstotliwości!




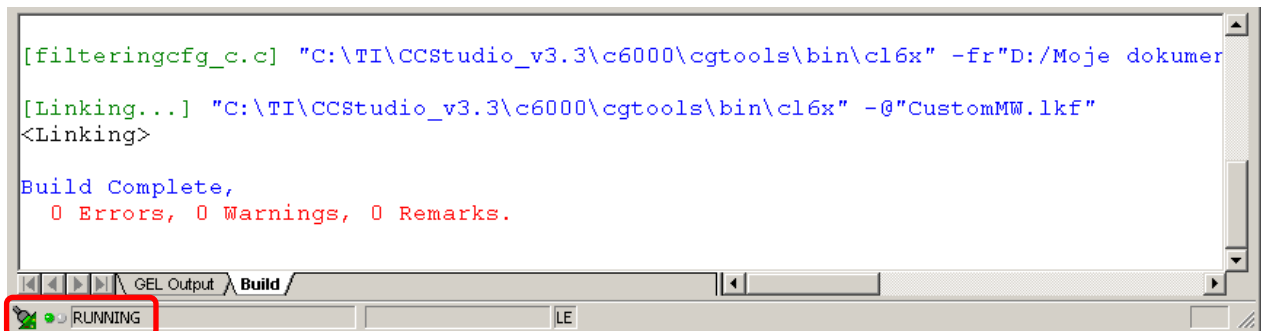
- Zwrócić uwagę, że dane z wejścia liniowego przetwornika A/C są przekształcane do formatu **single** oraz buforowane w formie ramek po 64 próbki (**Samples per frame**).
- **C6713 DSK DAC** (biblioteka **Target Support Package TC6 | C6713 DSK Board Support**) – przetwarzanie cyfrowo-analogowe sygnału dyskretnego z wyjścia filtra i przesłanie go (z normalizacją) na wyjście liniowe **Line Out** i głośnikowe **Headphone** kodeka audio.




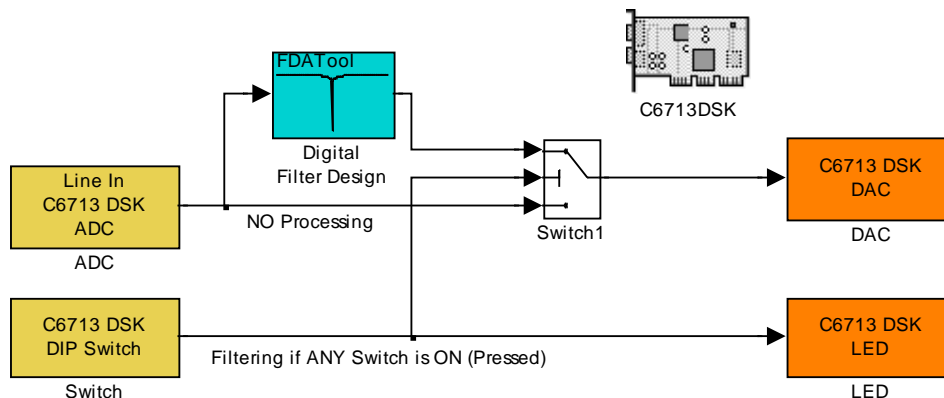
- Ustawić parametry symulacji wybierając z poleceń menu **Simulation | Configuration Parameters** (czas działania modelu – bez ograniczeń `Stop time = Inf`, solver czasu dyskretnego o stałym okresie próbkowania), `Periodic sample time constraint = Unconstrained`.



- Zapisać zbudowany model w pliku `.mdl` w folderze roboczym i uruchomić budowanie kodu modelu za pomocą polecenia menu **Tools | Real Time Workshop | Build Model** (kombinacja **Ctrl+B**) lub przycisku **Incremental Build**  z belki okna modelu (nie przebudowuje bibliotek). Polecenie to przetwarza schemat blokowy Simulinka na program w języku C i uruchamia środowisko CCS, gdzie następuje linkowanie bibliotek i kompilacja do kodu maszynowego, a następnie przesłanie programu do pamięci na karcie C6713 DSK i uruchomienie jego wykonywania przez procesor DSP. Komunikaty procesu budowania i uruchamiania programu można obserwować w oknie komend Matlaba, a następnie w zakładce **Build** głównego okna środowiska CCS. Stan połączenia z kartą DSK i działania programu sygnalizowane są na dolnej belce okna CCS.

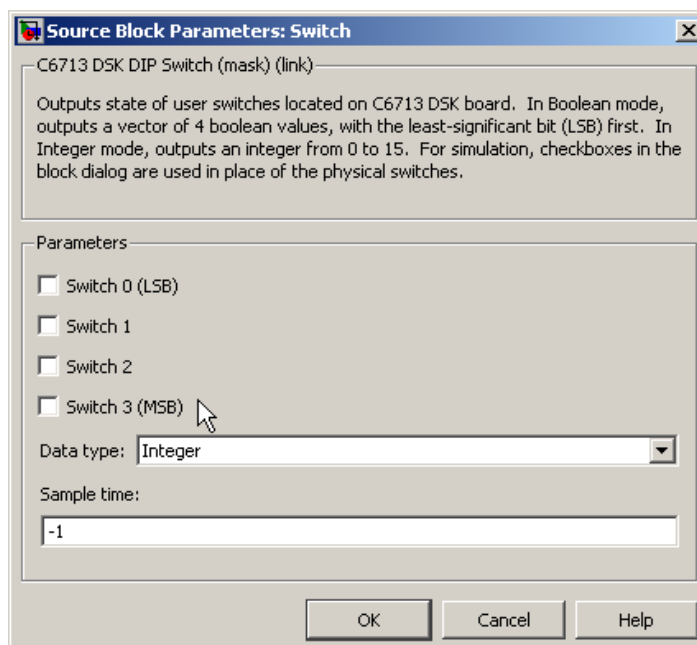


- Sprawdzić prawidłowość działania filtra zmieniając częstotliwość wejściowego sygnału harmonicznego od ok. 50 Hz do wartości powyżej częstotliwości Nyquista filtra (czyli połowy częstotliwości próbkowania zaprojektowanego filtra) i słuchając sygnał wyjściowy filtra na głośnikach (słuchawkach) oraz obserwując go na oscyloskopie.
    - Czy zakres tłumionych częstotliwości zgadza się z zakresem pasma zaporowego filtra?
    - Od jakiej (niskiej) częstotliwości dźwięk staje się słyszalny (dolny zakres pasma przenoszenia używanych głośników/słuchawek)?
    - Co dzieje się po przekroczeniu częstotliwości Nyquista i dlaczego?
  - Program realizowany na karcie DSK można zatrzymać za pomocą przycisku **Halt**  na belce po lewej stronie okna CCS.
- B.** Rozbudować model z rys. 3.3 do schematu przedstawionego na rys. 3.3. Celem jest przesyłanie na wyjścia DAC sygnału wyjściowego z filtra po spełnieniu warunku wciśnięcia któregośkolwiek z przełączników DIP Switch na karcie. Jeżeli żaden przełącznik nie jest wciśnięty na wyjścia DAC przekazywany jest bez zmian sygnał z wejścia Line In karty.



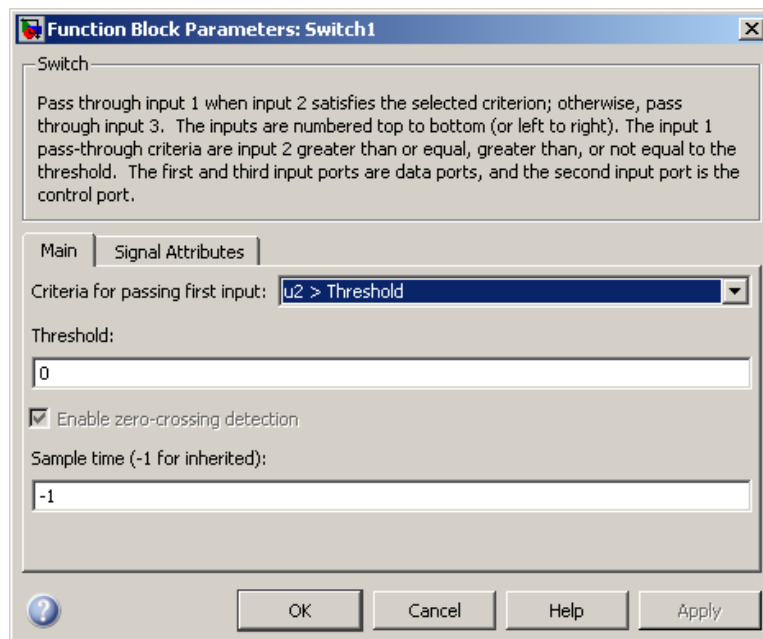
Rys. 3.3. Schemat Simulinka filtracji cyfrowej na karcie C6713 DSK po przełączeniu DIP switcha

- **Switch C6713 DSK DIP Switch** (biblioteka **Target Support Package TC6 | C6713 DSK Board Support**) – blok obsługi stanów czterech przełączników DIP Switch na karcie C6713 DSK.

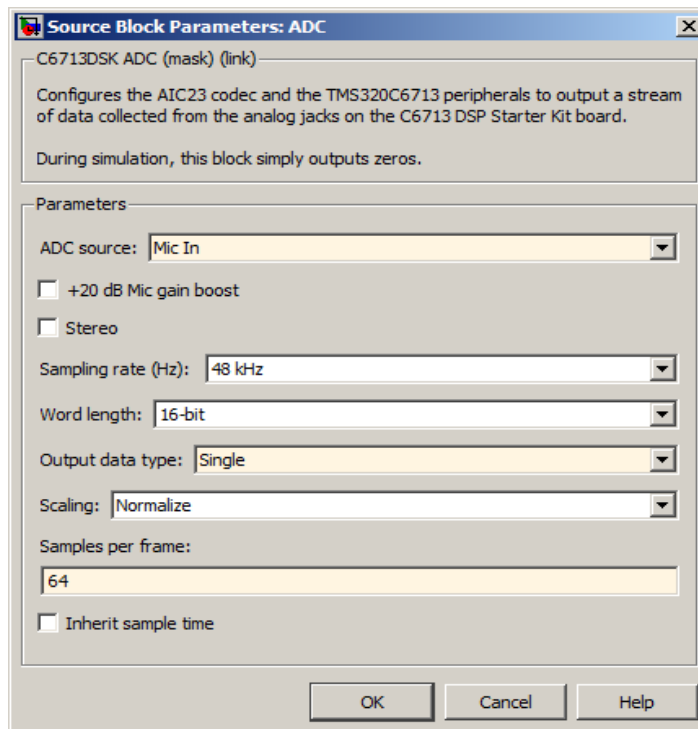


- **LED C6713 DSK LED** (biblioteka **Target Support Package TC6 | C6713 DSK Board Support**) – blok obsługi czterech diod LED na karcie. W tym przykładzie diody wskazują jedynie stany przełączników DIP Switch.
- **Switch1** (biblioteka **Simulink | Signal Routing** lub **Commonly Used Blocks**) – blok przełączania na wyjście sygnału z wejścia 1 (górnego) lub 3 (dolnego) w zależności od relacji pomiędzy wartością sygnału z wejścia 2 (środkowego) a wewnętrzną wartością progu (parametr `Threshold`).

Na wyjścia DAC karty będzie przesyłany sygnał wyjściowy z filtra, jeżeli stan któregośkolwiek przełącznika DIP Switch jest równy ON (wciśnięty, wyjściowe słowo kodowe bloku `Switch` będzie wtedy różne od zera i spełniony będzie warunek  $u_2 > \text{Threshold} = 0$ ). W przeciwnym razie na wyjścia DAC przesyłany jest bez przetwarzania sygnał z wejścia `Line In`.



- Zapisać zmodyfikowany model w folderze roboczym pod własną nazwą. Przeprowadzić procedurę uruchamiania programu modelu (**Ctrl+B**).
  - Sprawdzić prawidłowość działania programu.
  - C. Przeprowadzić eksperyment z układem jak na rys.3.3z filtrem BS pracującym z częstotliwością próbkowania 48 kHz i o następujących specyfikacjach częstotliwościowych:  $F_s=48000$  Hz,  $F_{pass1}=3000$  Hz,  $F_{stop1}=3500$  Hz,  $F_{stop2}=5500$  Hz,  $F_{pass2}=6000$  Hz, tłumienie w paśmie zaporowym  $A_{stop}=100$  dB, inne parametry jak poprzednio.
  - Pamiętać o konieczności zmiany częstotliwości próbkowania również w blokach ADC i DAC modelu.
  - Zapisać zmodyfikowany model pod własną nazwą i przeprowadzić procedurę uruchamiania.
  - Zwiększając częstotliwość (w zakresie drugiego pasma przepustowego) określić górny zakres pasma przenoszenia używanych głośników/słuchawek (lub czułość własnego słuchu).
  - Zaobserwować na oscyloskopie do jakiej częstotliwości (powyżej pasma słyszalnego) na wyjściu liniowym karty DSK utrzymuje się nietłumiony sygnał harmoniczny.
  - D. Zmodyfikować model do realizacji filtracji górnoprzepustowej sygnału akustycznego z wejścia mikrofonowego karty.
  - Zmienić źródło sygnału wejściowego w bloku ADC na wejście mikrofonowe `Mic In` i odznaczyć przetwarzanie `Stereo`.
- Uwaga: 1) Można wypróbować efekt zwiększenia wzmocnienia zaznaczając opcję `+20dB Mic gain boost`, ale może to spowodować sprzężenie akustyczne. 2) W przypadku braku mikrofonu jako źródła sygnału można użyć wyjścia słuchawkowego odtwarzacza CD/MP3 dołączonego do wejścia liniowego karty i w takim przypadku należy pozostawić wybór `ADC source = Line In`.



- W bloku FDA Tool zaprojektować filtr Highpass IIR o paśmie zaporowym np. do 500 Hz ( $F_s=48000$  Hz,  $F_{stop}=500$  Hz,  $F_{pass}=700$  Hz,  $A_{stop}=80$  dB).
- Sprawdzić, jakiego rzędu jest zaprojektowany filtr IIR i jakiego rzędu będzie zaprojektowany w FDA Tool filtr FIR (SOI) o takich samych specyfikacjach.
- Zapisać zmodyfikowany model pod własną nazwą, przeprowadzić procedurę uruchamiania i przetestować efekty działania programu na sygnale mowy z mikrofonu (lub sygnale z innego podłączonego do karty źródła).

### 3.2. Modelowanie efektu pogłosu (reverberation)

A. Zadanie polega na zaprogramowaniu dodawania do sygnału wejściowego z wejścia mikrofonowego (lub z innego źródła podłączonego do wejścia liniowego karty) prostego efektu pogłosu modelowanego za pomocą filtra NOI.

- W modelu z pkt. 3.1.D skasować blok filtra FDA Tool i wstawić w jego miejsce schemat filtra do realizacji pogłosu jak pokazano na rys. 3.4.

- Blok Delay (biblioteka **Signal Processing Blockset** | **Signal Operations**) jest blokiem opóźnienia dyskretnego liczonego w próbkach (*samples*) lub ramkach próbek (*frames*). Kliknąć blok dwa razy i ustawić  $Delay (samples) = 6000$ .

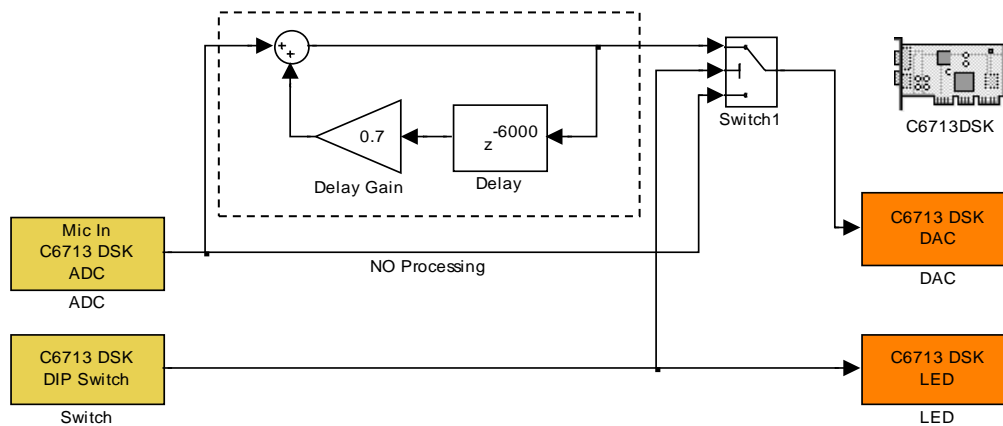
Blok Gain jest wzmacniaczem (biblioteka **Simulink** | **Math Operations**). Ustawić współczynnik wzmocnienia  $Gain=0.7$ .

Sumator Sum (biblioteka **Simulink** | **Math Operations**) sumuje sygnały wejściowe ze znakami określonymi w oknie parametrów bloku ( $List\ of\ signs = |++$ ).

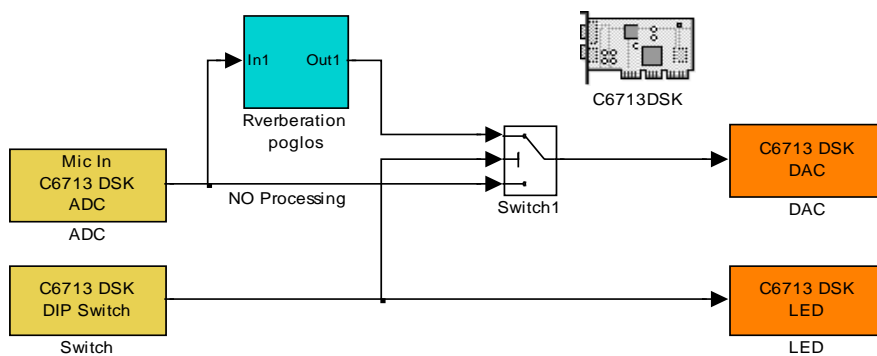
Uwaga: Edycję kierunku przepływu sygnału na schemacie (w torze sprzężenia zwrotnego) umożliwia polecenie **Format** | **Flip block (Rotate block)**, które dokonuje odbicia symbolu bloku w pionie (obrotu symbolu bloku).

- Zaznaczyć nowy fragment schematu jak na rys. 3.4 i utworzyć z niego subsystem za pomocą polecenia **Edit** | **Create Subsystem**, aby otrzymać model pokazany na rys. 3.5.

Uwaga: Po utworzeniu subsystemu może być potrzebne jego odbicie w pionie poleceniem **Flip block**, aby schemat był czytelny.



Rys. 3.4. Schemat modelu z filtrem do realizacji efektu pogłosu

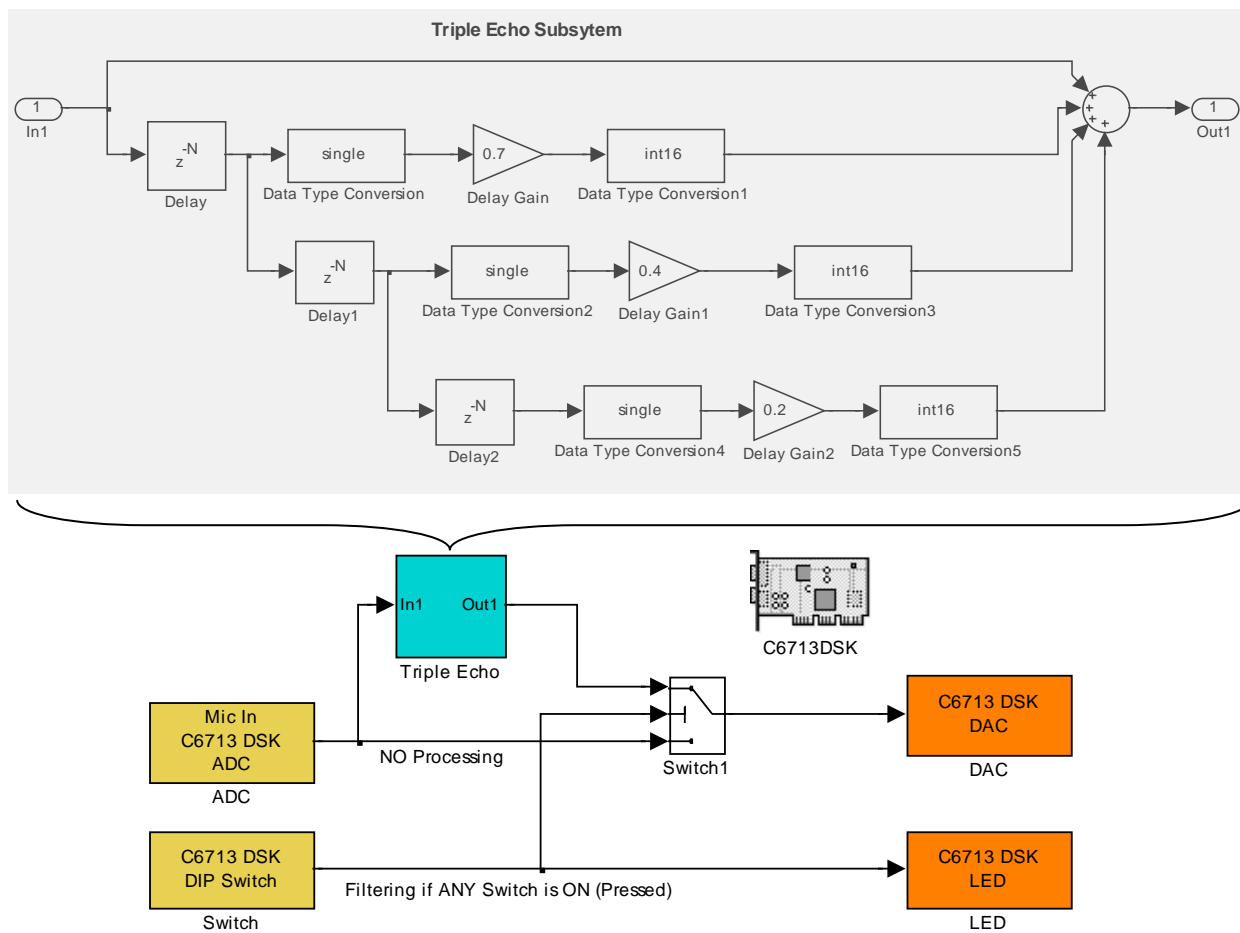


Rys. 3.5. Model z subsystemem Reverberation-pogłos filtra do realizacji efektu pogłosu

- Zapisać zmodyfikowany model pod własną nazwą i przeprowadzić procedurę uruchamiania.
  - Mówiąc do mikrofonu przetestować uzyskany efekt pogłosu.
  - Jakie opóźnienie (w milisekundach) jest realizowane w torze sprzężenia zwrotnego filtra biorąc pod uwagę częstotliwość próbkowania układu?
  - Napisać równanie różnicowe wiążące wejście  $x(n)$  z wyjściem  $y(n)$  oraz transmitancję  $H(z)$  filtra pogłosu. Czy filtr może być niestabilny?
- B.** Powtórzyć kilka razy eksperyment kompilując i uruchamiając program ze zmienionymi wartościami parametrów Delay i/lub Gain filtra pogłosu.
- Skomentować, w jaki sposób wartości parametrów filtra wpływają na uzyskany efekt pogłosu.

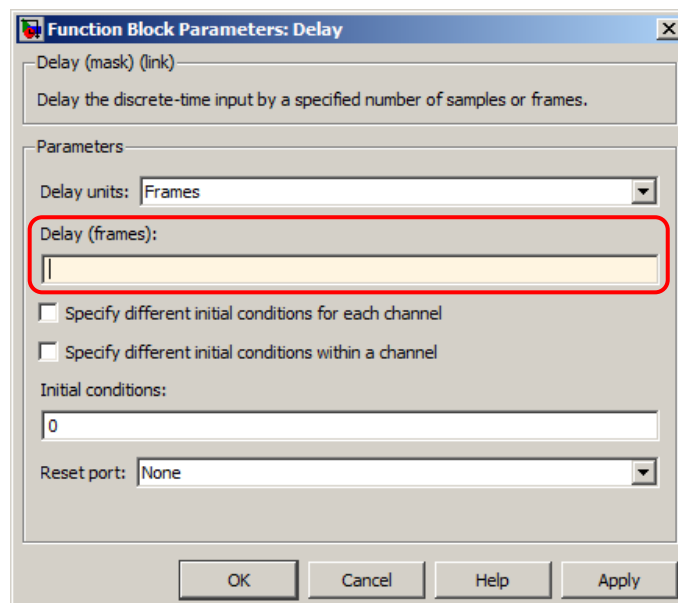
### 3.3. Modelowanie efektu echa

- A.** Zadanie polega na zaprogramowaniu dodawania do sygnału wejściowego z podłączonego do karty mikrofonu prostego efektu echa modelowanego za pomocą filtra SOI.
- Dokonać modyfikacji subsystemu w modelu z pkt. 3.2.A (rys. 3.5):
- 1) W bloku C6713 DSK ADC zmienić typ danych wyjściowych z bloku na Output data type = Integer (w celu zmniejszenia potrzebnego obszaru pamięci) oraz rozmiar ramki Samples per frame = 256 (pozostałe parametry jak w pkt. 3.2).
  - 2) Zmienić schemat filtra w bloku subsystemu w sposób pokazany na rys. 3.6.



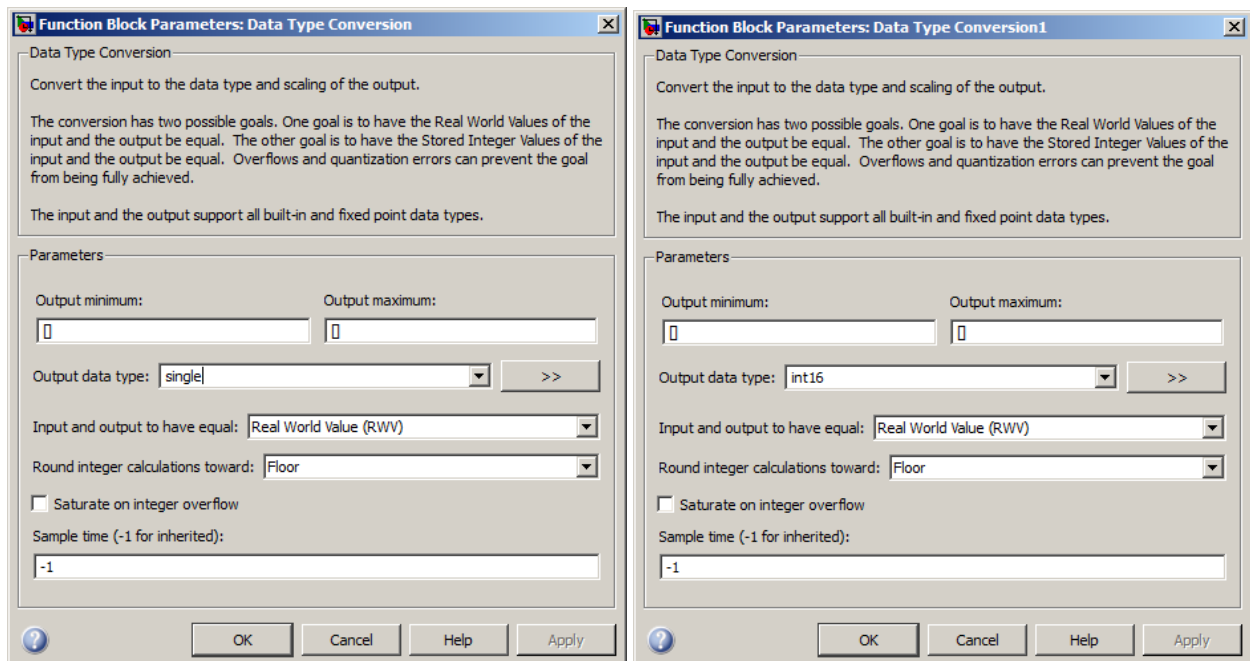
Rys. 3.6. Schemat modelu do realizacji efektu potrójnego echa

- W blokach opóźnień  $z^{-N}$  ustawić zadawanie opóźnienia jako liczby ramek  
Delay units = Frames.
- Wyznaczyć i zadać wartości Delay (jako całkowitą liczbę ramek  $N$ ) tak, aby każda gałąź filtra realizowała opóźnienie sygnału (okres powtarzania echa) o ok. 500 milisekund (biorąc pod uwagę częstotliwość próbkowania i liczbę próbek na jedną ramkę).





- Bloki Data Type Conversion (biblioteka **Simulink** | **Signal Attributes**) dokonują konwersji format danych na `single`, a po pomnożeniu przez współczynnik wzmocnienia z powrotem na `int16`.



- Lista znaków sygnałów wejściowych sumatora z czterema wejściami dająca efekt pokazany na schemacie powinna mieć postać: `List of signs = |++++`.
- Obliczyć wielkość pamięci potrzebnej do przechowywania opóźnionych próbek sygnału (w formacie `int16`) potrzebnych do działania filtra.
- Zwrócić uwagę na coraz mniejsze wzmocnianie (**Delay gain**) bardziej opóźnionych sygnałów w kolejnych gałęziach filtra.
- Zapisać zmodyfikowany model pod własną nazwą i przeprowadzić procedurę uruchamiania.
- Mówiąc do mikrofonu przetestować uzyskany efekt echa.
- B.** Powtórzyć eksperyment kilka razy kompilując i uruchamiając program ze zmienionymi wartościami parametrów `Delay` i/lub `Gain` filtra echa.
- Skomentować, w jaki sposób wartości parametrów filtra wpływają na uzyskany efekt echa.

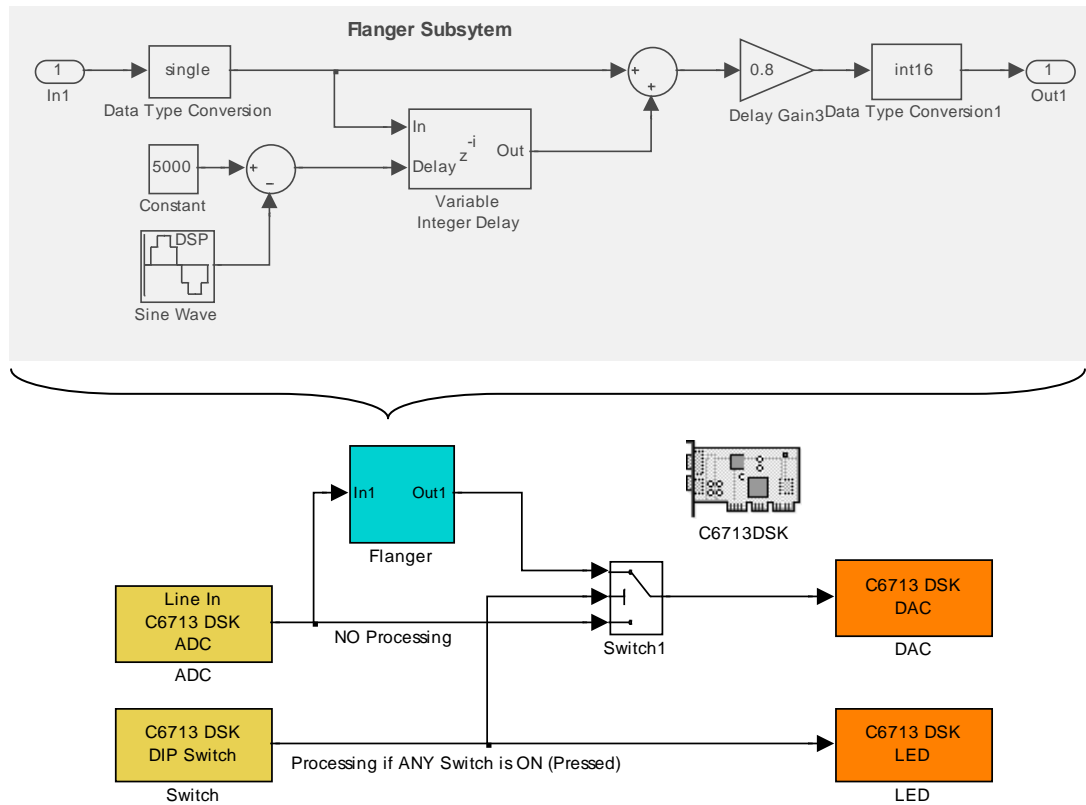
### 3.4. Modelowanie efektu fazowego (flanger)

Zadanie polega na zaprogramowaniu efektu flangeru polegającego na dodawaniu do sygnału oryginalnego jego opóźnionej (zwykle również zmodulowanej) wersji, przy czym opóźnienie jest (okresowo) zmienne w czasie. Efekt jest opisany równaniem różnicowym

$$y(n) = x(n) + ax(n - d(n)), \quad \text{gdzie: } d(n) = D + \frac{D}{2} \left( 1 - \sin \frac{2\pi f_d n}{f_s} \right)$$

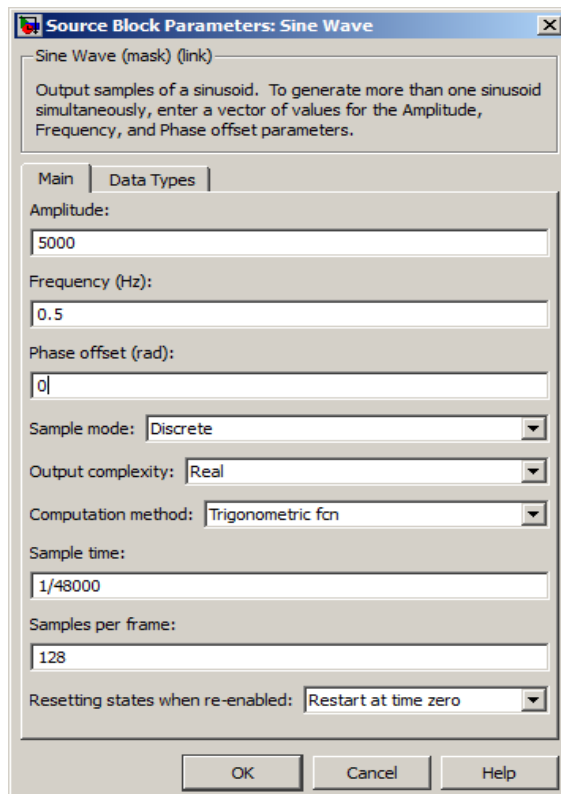
gdzie:  $D$  – stałe opóźnienie,  $f_d$  – częstotliwość flangeru.

- Utworzyć model Simulinka do realizacji efektu flangeru jak na rys. 3.7 (np. poprzez odpowiednią modyfikację modelu do realizacji echa z pkt. 3.3).
- W bloku **C6713 DSK ADC** ustawić: `ADC Source=Line In`, `Sampling Rate=48 kHz`, `Word length=16-bit`, `Output Data Type=Integer`, `Samples per frame=128`.
- Blok **Variable Integer Delay** (biblioteka **Signal Processing Blockset** | **Signal Operations**) realizuje zmienne opóźnienie sygnału z wejścia `In` równe wartości sygnału na wejściu `Delay`. W bloku tym zadać `Maximum delay=10000 samples` (czyli ok. 0.2 sekundy przy  $f_s=48$  kHz).



Rys. 3.7. Schemat modelu do realizacji efektu flangeru

- Blok Sine Wave DSP (biblioteka **Signal Processing Blockset** | **Signal Processing Sources**) generuje sygnał harmoniczny. Zadać przykładowe wartości parametrów jak poniżej.

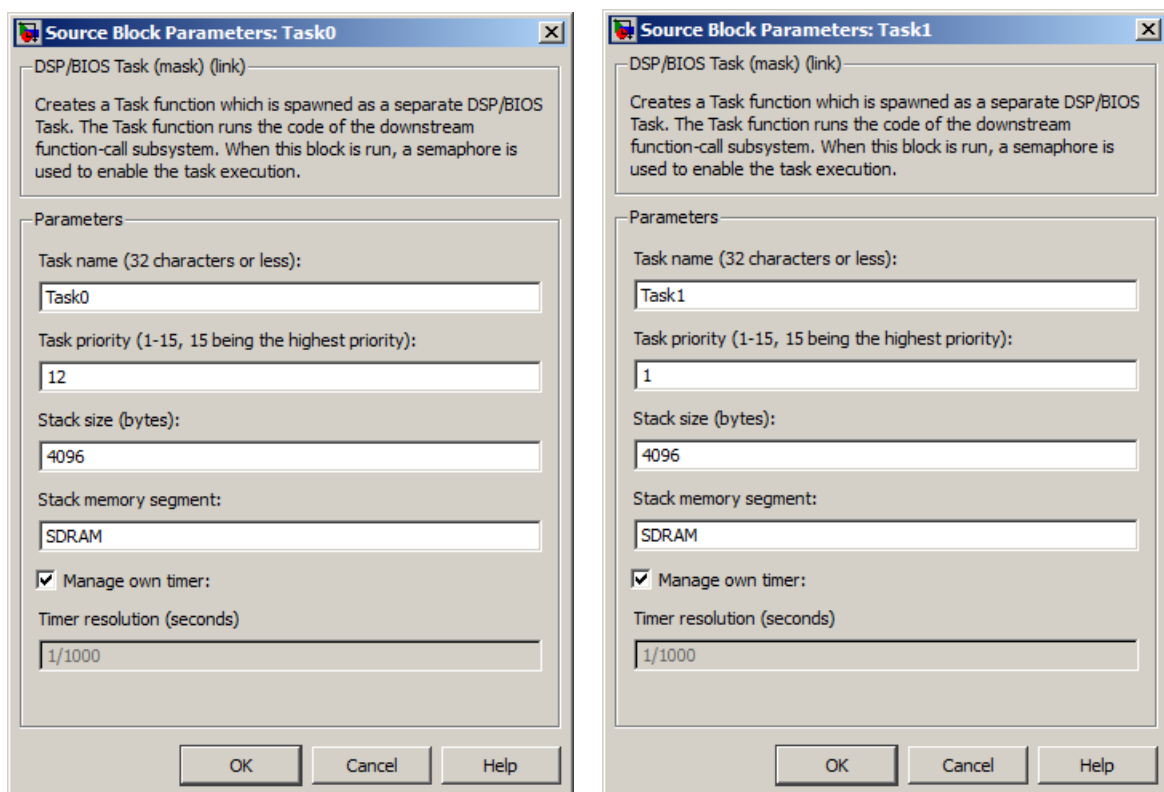


- Blok Constant (biblioteka **Simulink** | **Sources**) jest źródłem sygnału o stałej wartości.
- Wejście liniowe karty C6713 DSK podłączyć do wyjścia generatora sygnału sinusoidalnego o częstotliwości ok.  $1 \div 3$  kHz.
- Zapisać model pod własną nazwą i przetestować efekt dla kilku różnych kombinacji wartości parametrów  $D$  (Constant i Sine Wave Amplitude) i  $f_d$  (Sine Wave Frequency). Skomentować otrzymane wyniki.
- Zmodyfikować model w taki sposób, żeby przetwarzany sygnał pochodził z wejścia mikrofonowego Mic In karty. Przetestować efekt i skomentować wyniki.

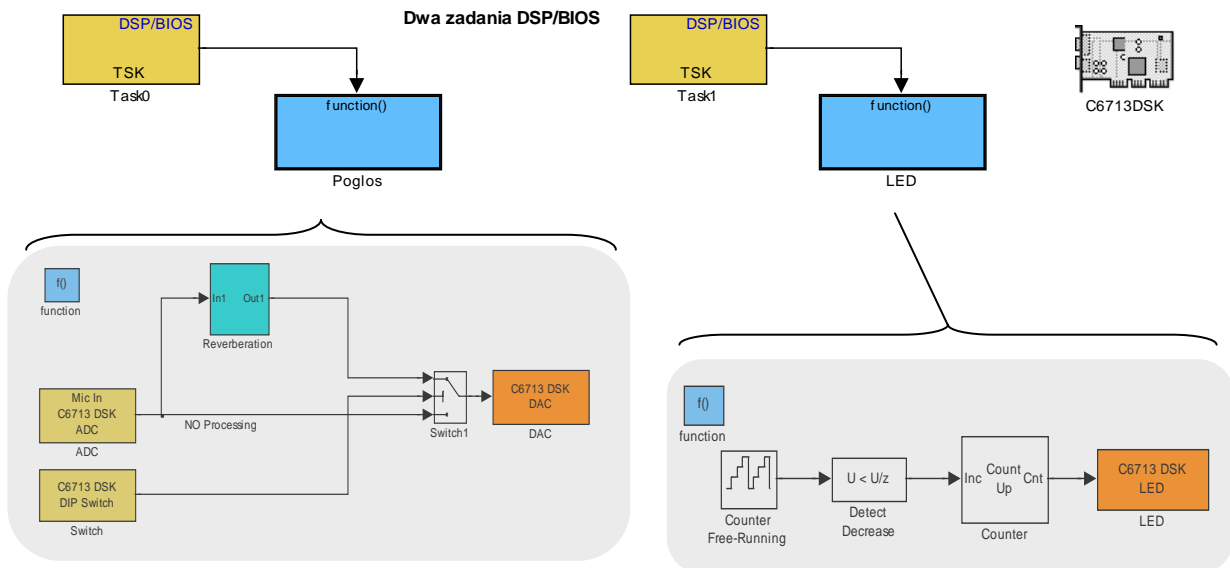
### 3.5. Dwuzadaniowa aplikacja DSP/BIOS

Zadanie polega na stworzeniu modelu realizującego na procesorze DSP program złożony z dwóch oddzielnych zadań systemu DSP/BIOS.

- Otworzyć nowy model Simulinka i wstawić do niego dwa bloki Task DSP/BIOS (biblioteka **Target Support Package TC6** | **DSP/BIOS Library**), dwa bloki wywołania funkcji Function-Call Subsystem (biblioteka **Simulink** | **Ports & Subsystems**) oraz blok platformy docelowej C6713 DSK.
- Ustawić parametry bloków zadań DSP/BIOS. Task0 będzie zadaniem głównym o wysokim priorytecie, Task1 – zadaniem dodatkowym o niskim priorytecie. Zaznaczenie Manage own timer powoduje, że zadanie samo zarządza czasem odczytując timer (o rozdzielczości podanej przez Timer resolution (1 ms) - parametr tylko do odczytu) z procesora DSP.

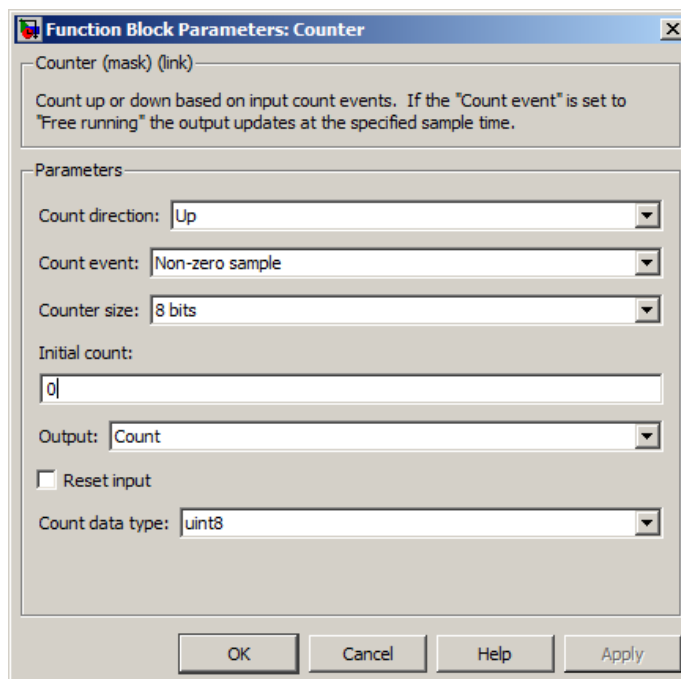


- Otworzyć zapisany w pkt. 3.2 model realizujący efekt pogłosu (rys. 3.5) i skopiować go w całości (oprócz bloku platformy C6713) do subsystemu function() Poglos (rys. 3.8, usunąć zawartość domyślną bloku). Będzie to funkcja wywoływana przez zadanie Task0.
- Przeprowadzić edycję subsystemu funkcji function() LED wywoływanej przez zadanie Task1 jak na rys. 3.8. Funkcja zlicza pełne cykle pierwszego licznika w liczniku drugim, którego 4 najmniej znaczące bity sterują diodami LED na karcie.



Rys. 3.8. Model programu złożonego z dwóch zadań DSP/BIOS i subsystemy funkcji wywoływanych przez zadania

- Counter Free-Running (biblioteka **Simulink | Sources**) – blok swobodnie zliczającego licznika-źródła, którego wartość jest zwiększana przy każdorazowym uaktywnieniu zadania Task1. Po osiągnięciu wartości maksymalnej licznik ponownie zaczyna zliczanie od zera. Ustawić Number of Bits=18.
- Detect Decrease (biblioteka **Simulink | Logic and Bit Operations**) – blok ustawiający wyjście (na wartość True=1) po wykryciu zmniejszenia się wartości wejściowej w porównaniu z poprzednim krokiem, czyli po każdym wyzerowaniu się pierwszego licznika.
- Counter Count Up (biblioteka **Signal Processing Blockset | Signal Management | Switches and Counters**) – blok zliczający impulsy (zdarzenia) wejściowe (w tym przypadku w górę).



- Zapisać model pod własną nazwą, przeprowadzić procedurę uruchamiania i sprawdzić, czy program działa prawidłowo.

## 4. Opracowanie sprawozdania

W sprawozdaniu należy zamieścić schematy blokowe uruchamianych modeli Simulinka oraz opisać i skomentować efekty przeprowadzonych eksperymentów, ze szczególnym uwzględnieniem problemów wskazanych symbolem ➤.

### Literatura

1. Chassaing R., Reay D.: *Digital Signal Processing and Applications with TMS320C6713 and TMS320C6416 DSK*, 2 ed., John Wiley & Sons, 2008.
2. Kuo S., Li B., Tian W.: *Real-Time Digital Signal Processing. Implementations and Applications*, 2 ed., John Wiley & Sons, 2006.
3. Kehtarnavaz N.: *Real-Time Digital Signal Processing Based on TMS320C6000*, Newnes, 2005.
4. *TMS320C6713 DSK. Technical Reference*, Spectrum Digital Inc., 2004.
5. *Target Support Package 4. User's Guide, For Use with Texas Instruments C6000*, The Mathworks Inc., 2010.
6. Praca zbiorowa pod red. A. Dąbrowskiego: *Przetwarzanie sygnałów przy użyciu procesorów sygnałowych*, Wyd. Politechniki Poznańskiej, 1998.
7. Kowalski H.: *Procesory DSP dla praktyków*, Wyd. BTC, 2011.
8. Smith S.W.: *Cyfrowe przetwarzanie sygnałów. Praktyczny poradnik dla inżynierów i naukowców*, Wyd. BTC, 2007.

Opracował: Dr inż. Janusz Baran

Częstochowa, 1999-2013