Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Projekt ‘filtr NLMS’

Z przedmiotu:

SDUP

wykonali:
Michał Stankiewicz
Jakub Szymański

Kraków 19.06.2023r.

1. Założenia projektowe

Celem projektu było stworzenie bloku IP, który mógłby realizować zadanie filtracji cyfrowej FIR, LMS i NLMS. Blok ten mógłby być używany w wielu aplikacjach wymagających przetwarzania sygnałów, jak np. audio. Założenia wysokopoziomowe które przyświecały projektowi to:

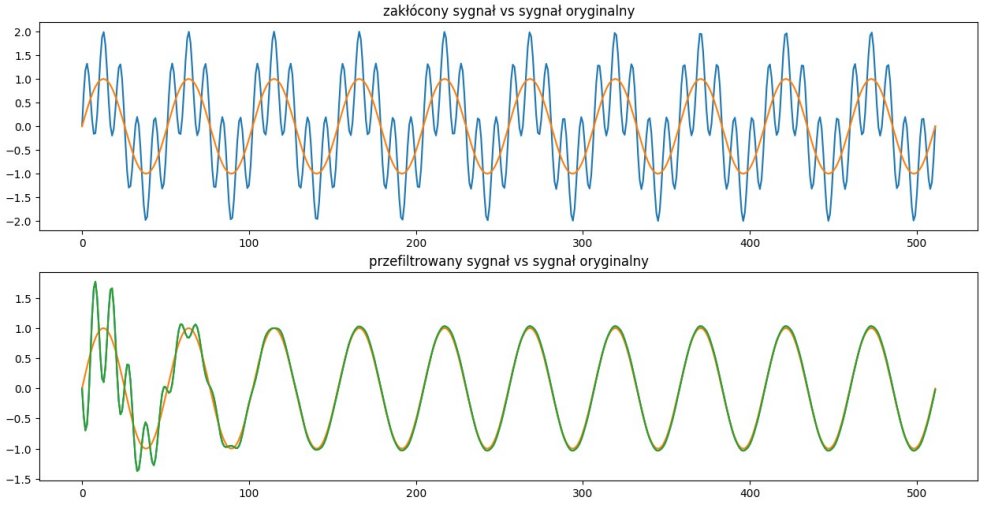
- uniwersalność – projekt jest agnostyczny względem zewnętrznego systemu i jego architektury
- przenośność – projekt można przenosić między różnymi układami, a także między środowiskami różnych producentów
- skalowalność – możliwe jest przeskalowanie projektu do swoich potrzeb, tak by osiągnąć pożądany balans między osiągami a wykorzystywanymi zasobami
- programowalność – możliwa jest zmiana konfiguracji działania z poziomu systemu w czasie działania układu
- prostota rozbudowy

W celach testowych wybrano następującą platformę:

- środowisko FPGA Vivado 2022.2
- środowisko softwareowe Vitis
- płytka rozwojowa Cora-Z7-07 z układem xc7z007sclg400-1

2. Opis teoretyczny

Algorytmy LMS (Least Mean Squares) to klasa adaptacyjnych filtrów używanych do imitowania pożądanego filtru poprzez znalezienie współczynników filtru h, które prowadzą do minimalizacji średniego kwadratu sygnału błędu (różnicy między pożądanym a rzeczywistym sygnałem). Przykład działania takiego algorytmu widać poniżej:

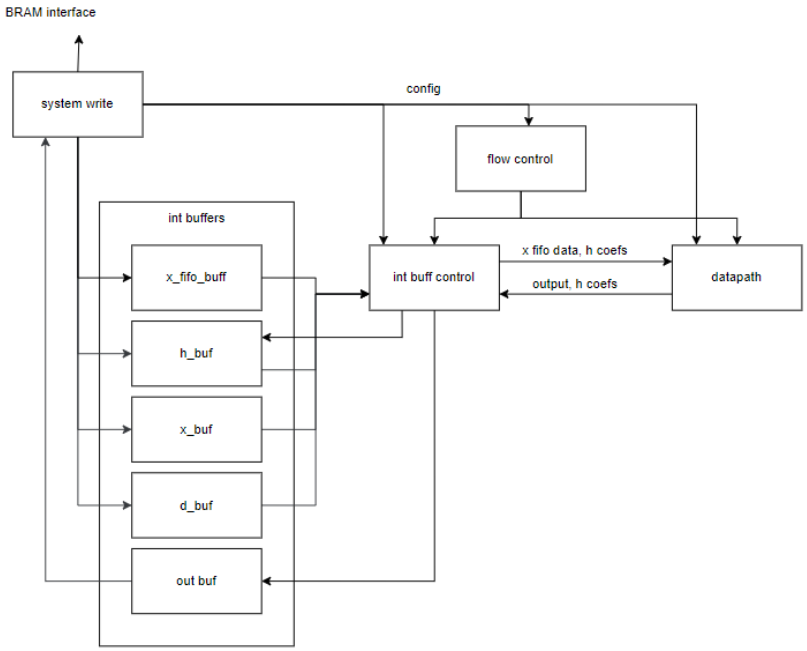


Jak widać, filtr potrzebuje chwili żeby „nauczyć się” sygnału, co objawia się zakłóconym sygnałem na początku przebiegu. Jednak potem filtr bardzo sprawnie odzyskuje sygnał i odrzuca zakłócenie.

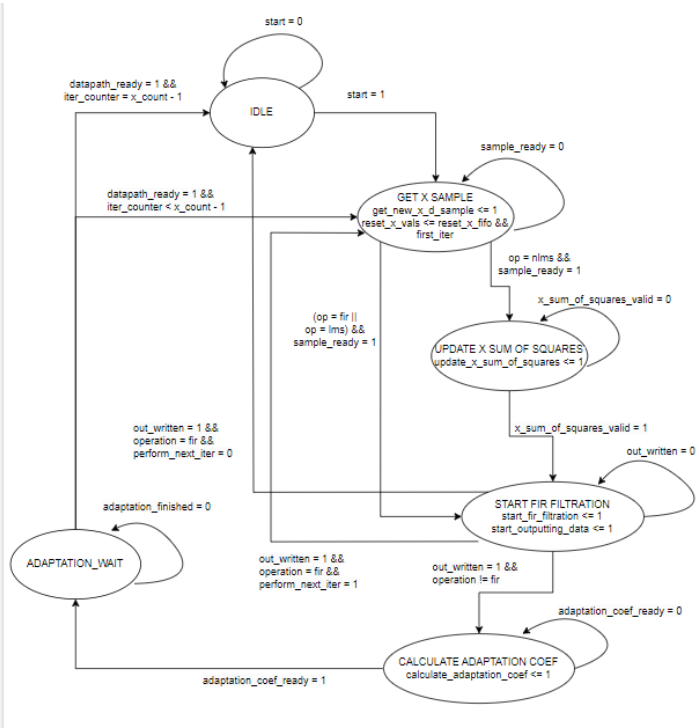
Wadą tego algorytmu jest wrażliwość na skalowanie wejścia x(n), co powoduje, że jest prawie niemożliwe odpowiednio dobrać współczynnik μ. Filtr NLMS (Normalised Least Mean Squares) rozwiązuje ten problem dzięki normalizację względem mocy sygnału wejściowego.

3. Schematy blokowe i działanie

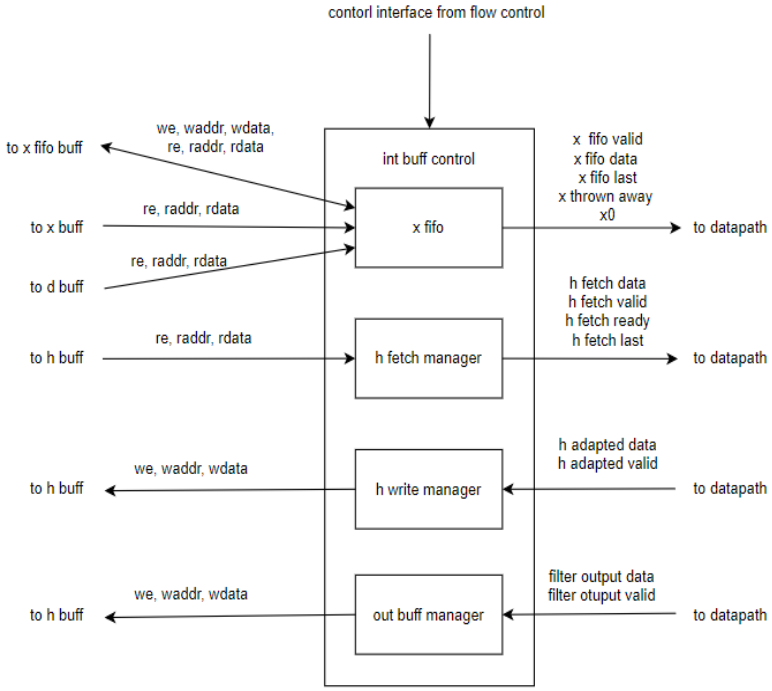
Zaimplementowany projekt można wysokopoziomowo przedstawić następującym schematem blokowym:



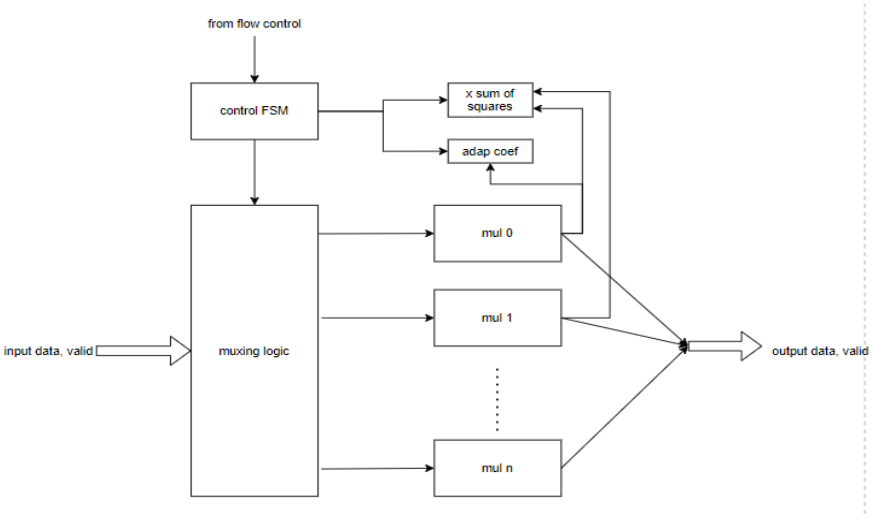
Moduł „system write” odpowiada za kontakt z systemem. Wystawia on interfejsy BRAM (zarówno od strony systemu jak i układu) i sygnały konfiguracyjne, pełniąc tym samym rolę zbioru rejestrów kontrolno sterujących i demultipleksera dostępu do pamięci.
Moduł „int buffers” przechowuje w sobie wszystkie moduły BRAM wykorzystywane w układzie. Zgrupowanie ich w jednym miejscu ułatwi rozbudowanie układu, ponieważ zastępując ten moduł przez np. DMA można łatwo przenieść wewnętrzne bufory do pamięci systemu (np. DDR) jeśli zajdzie taka potrzeba.
Moduł „flow control” implementuje główny FSM układu, kontrolujący przebieg przetwarzania. Jego schemat znajduje się poniżej:



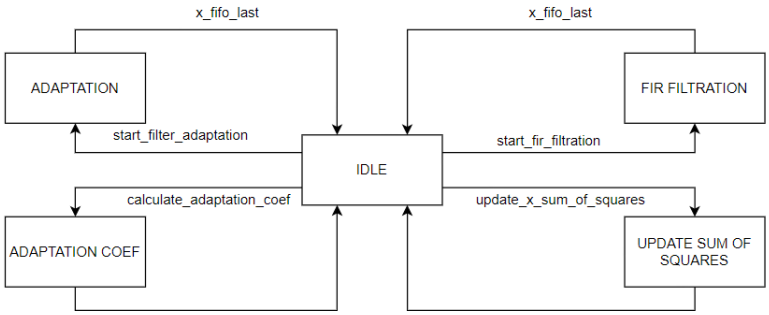
Jak widać, wspiera on zarówno przetwarzanie FIR, LMS i NLMS, o programowalnej ilości próbek do przefiltrowania i programowalnym rzędzie filtru.
Moduł „int buff control” odpowiada za zaopatrywanie modułu „datapath” w dane pochodzące z wewnętrznych buforów. Jego schemat blokowy z elementami funkcjonalnymi znajduje się poniżej:



Moduł „x fifo” implemetuje funkcjonalność bufora FIFO przy wykorzystaniu zewnętrznego bufora pamięciowego. Gdy dostanie polecenie z „flow control” aby pobrać dane, zaciąga nową próbkę z bufora „x buff” i zapisuje w buforze przeznaczonym dla FIFO na odpowiednim miejscu. Jednocześnie pobiera z bufora d kolejną próbkę sygnału referencyjnego. Uaktualniane są również sygnały x_0 i x_thrown_away, które odpowiednio zawierają pierwszą próbkę w buforze i tą która została z niego usunięta. Są one potrzebne przy filtracji NLMS do obliczenia znormalizowanego współczynnika μ. Gdy z „flow control” przyjdzie polecenie by dostarczyć dane, z bufora odczytywane są kolejne próbki (tyle ile wynosi liczba mnożarek) i wysyłane są do modułu „datapath”. Pojawienie się nowych próbek sygnalizowane jest stanem wysokim na sygnale x_fifo_valid, natomiast pojawienie się ostatniej próbki sygnalizowane jest sygnałem x_fifo_last.
Moduł „h fetch manager” zaopatruje datapath w współczynniki filtra. Gdy dostanie polecenie aby dostarczyć dane, pobiera z bufora h współczynniki (tyle ile wynosi liczba mnożarek) i wystawia na interfejs połączony z datapath. Kolejna próbka zostaje pobrana dopiero wtedy, gdy datapath wystawi sygnał h_fetch_ready. Wynika to z tego, iż podczas adaptacji próbki x muszą przepropagować się przez potok, więc występuje kilka taktów zwłoki między poleceniem dostarczenia danych a tym kiedy datapath może je przyjąć. Takie rozwiązanie zapewnia elastyczność przy zmianie długości potoku.
Moduły „h write manager” i „out buff manager” są bardzo podobne, mianowicie przyjmują strumień danych z datapathu i zapisują w odpowiednich buforach wewnętrznych.
Moduł „datapath” wykonuje właściwe przetwarzanie danych. Ma on formę potokową i może przyjmować dane cykl po cyklu. Obecnie składa się on z dwóch modułów, „multipliers” oraz „product processor”.
Moduł „multipliers” zawiera w sobie potokowe układy mnożące, oraz logikę kontrolującą ich wejścia i wyjścia. Jego wysokopoziomowy schemat znajduje się poniżej:



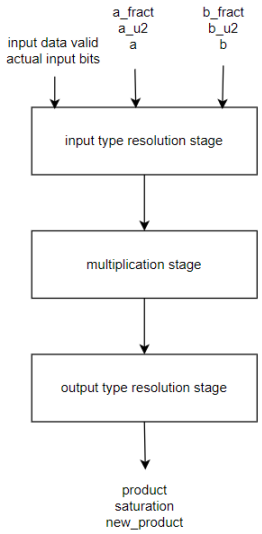
Wejściowe multipleksery są kontrolowane przez prosty FSM:



W zależności od stanu, na wejścia mnożarek podawane są inne wartości:

- przy filtracji FIR na wejścia wszystkich podawane są próbki x z bufora FIFO i współczynniki filtra
- przy aktualizacji sumy kwadratów wyrazów x z bufora FIFO na wejścia mnożarek 0 i 1 podawane są odpowiednio wartości pierwszej próbki i próbki odrzuconej, obie w celu podniesienia do kwadratu
- przy obliczaniu współczynnika adaptacji na wejście mnożarki 0 podawane są wartości błędu i wartość współczynnika μ
- przy adaptacji na wejścia wszystkich mnożarek podawane są próbki x z bufora FIFO i współczynnik adaptacji

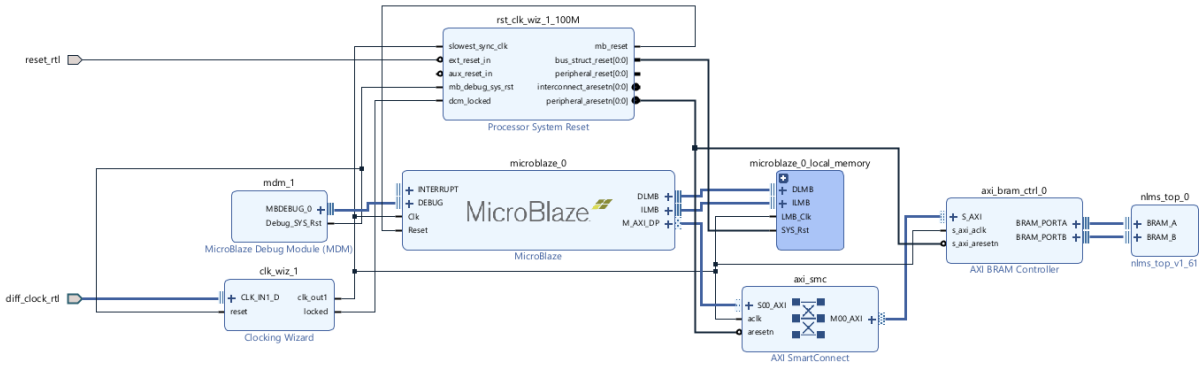
Same mnożarki prezentują się następująco:



Każdy stopień potoku może przetwarzać inny typ danych, co zwiększa elastyczność.
Ostatnim stopniem modułu „datapath” jest „product processor”. Dodaje on do siebie wyniki mnożenia otrzymane od „multipliers” przy filtracji FIR i przeprowadza adaptację. Oblicza również sumę kwadratów próbek x w buforze FIFO i podawałby te dane do modułu „mi calculator”, który nie został zaimplementowany.

4. Symulacje

Symulacje przeprowadzano na testowym systemie, wyposażonym w procesor Microblaze. Obserwowano jego magistralę AXI, wejściowy interfejs BRAM układu, a także wewnętrzne sygnały układu.



Wykonano również syntezę układu:

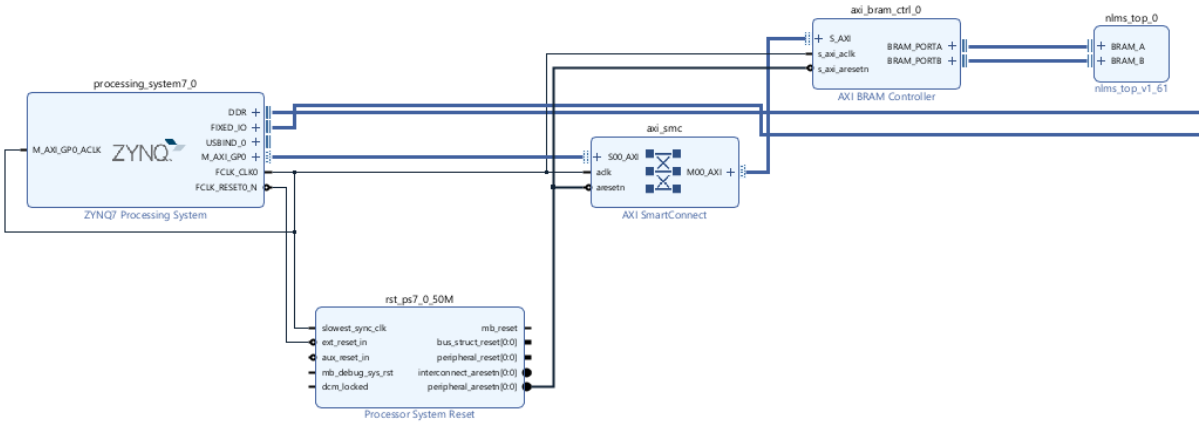
Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	1073	0	0	14400	7.45
LUT as Logic	1073	0	0	14400	7.45
LUT as Memory	0	0	0	6000	0.00
Slice Registers	1005	0	0	28800	3.49
Register as Flip Flop	1005	0	0	28800	3.49
Register as Latch	0	0	0	28800	0.00
F7 Muxes	32	0	0	8800	0.36
F8 Muxes	0	0	0	4400	0.00

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	4.5	0	0	50	9.00
RAMB36/FIFO*	1	0	0	50	2.00
RAMB36E1 only	1				
RAMB18	7	0	0	100	7.00
RAMB18E1 only	7				

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	4	0	0	66	6.06
DSP48E1 only	4				

5. Wyniki i wnioski

Poniżej znajduje się testowy system, który został zaimplementowany na układzie FPGA:



Jako interfejs między magistralą AXI a układem zastosowano kontroler BRAM dostarczony przez firmę Xilinx, który zamienia transakcje AXI na standardowe operacje dostępu do pamięci BRAM. Jako procesor ogólnego przeznaczenia wykorzystano sprzętowy Zynq Processing System, zawierający pojedynczy rdzeń ARM. Poniżej znajduje się raport z implementacji systemu:

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs	2487	0	0	14400	17.27
LUT as Logic	2257	0	0	14400	15.67
LUT as Memory	230	0	0	6000	3.83
LUT as Distributed RAM	96	0			
LUT as Shift Register	134	0			
Slice Registers	2231	0	0	28800	7.75
Register as Flip Flop	2231	0	0	28800	7.75
Register as Latch	0	0	0	28800	0.00
F7 Muxes	32	0	0	8800	0.36
F8 Muxes	0	0	0	4400	0.00

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	4.5	0	0	50	9.00
RAMB36/FIFO*	1	0	0	50	2.00
RAMB36E1 only	1				
RAMB18	7	0	0	100	7.00
RAMB18E1 only	7				

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	4	0	0	66	6.06
DSP48E1 only	4				

Następnie napisano testową aplikację w języku C, która otrzymywała próbki przez interfejs UART, wysyłała je do przetworzenia przez układ, odbierała i odsyłała z powrotem poprzez UART. Stworzono również aplikację na PC w języku Python, która tworzy sygnał testowy, wysyła poprzez port szeregowy, odbiera przetworzone dane i prezentuje na wykresie.