Wprowadzenie

Celem drugiego ćwiczenia jest modyfikacja projektu 'C Sampled-Based TT' w taki sposób, aby korzystając z doświadczenia w wywoływaniu z programu w języku C programu napisanego w asemblerze SHARC'a, dodać filtrację FIR.
Wymaga to następujących czynności:

- Zaprojektowania filtru FIR, na przykład w Matlab-ie korzystając z aplikacji `fdatool`.
- Dołączenia do źródeł w projekcie definicji zmiennych stałoprzecinkowych (`fract`) zawartych w bibliotece `stdfix.h` aby móc definiować takie zmienne czy struktury danych:
  `#include <stdfix.h>.`
- Przedefiniowania typów zmiennych używanych w oryginalnym programie z `float` na `fract`, ale tylko dla próbek sygnałów.
- Wstawienia w miejsce przekazywania próbek wejściowych do buforów wyjść wywołania programu w asemblerze realizującego filtrację FIR.
- Napisania procedury filtrującej w asemblerze SHARC, deklarując dla niej wektor współczynników filtru typu `fract`, który należy wypełnić danymi z projektu w Matlabie(eksport danych z np. przestrzeni roboczej Matlaba). Rezultat z rejestr MRF pełniącego rolę akumulatora układu mnożącego należy przesłać do rejestru wyniku R0 za pomocą instrukcji zaokrąglenia:
  `R0= RND MRF;`
  Przykładowo, taka procedura może wyglądać następująco:

```
_filterFIR:
.global _filterFIR;

BIT SET Mode1 0x1000000;   // set CBUFF mode
nop;
i4=r4;   // DM circular sample buffer
i12=r8;  // PM FIR coefficients
m12=1;
b4=r4; l4=TAPS-1;    // circular buffer settings
i4=dm(cpoint);       // Load cbuff pointer
dm(i4,1)=r12;        // put new sample into buffer
m4=-1;
dm(cpoint)=i4;       // save updated cbuff pointer
modify(i4,m4);
mrf = 0;
r0=dm(i4,m4); r4=pm(i12,m12);
lcntr=TAPS-1, do endsplot until LCE;
endsplot: mrf=mrf+r0*r4(ssf), r0=dm(i4,m4), r4=pm(i12,m12);
r0=rnd mrf;
BIT CLR Mode1 0x1000000;   // clear CBUFF mode
nop;

   leaf_exit;
_filterFIR.end:
```

- Deklaracji bufora cyklicznego dla próbek filtrowanego sygnału z któregoś z wejść ADC, oraz wskaźnika tego bufora przechowującego aktualne położenie najnowszej próbki.

Demonstracyjny projekt 'C Sampled-Based TT' znajduje się na nowej stronie zespołu Systemów Wbudowanych:
http://www.embedded.agh.edu.pl/www/dsp/dydaktyka/DSP/Laboratorium/Cwiczenie2/

Program ćwiczenia:

1. Skopiuj do swojego katalogu na dysku D: projekt ‘C Sampled-Based TT’ spakowany w pliku *.zip z adresu WWW podanego powyżej. Oczywiście, rozpakuj go.
2. Otwórz ten projekt w środowisku CrossCore Embeded Studio.
3. Skompiluj go i uruchom na rzeczywistym procesorze ADSP-2148(7)9 wykorzystując moduł EzKit.
4. Sprawdź czy sygnały z wejść przekazywane są na wyjścia zgodnie z opisem zawartym w komentarzach projektu. W tym celu uruchom program Spectralab (lub SpectraPlus) i podłącz wejścia i wyjścia modułu SHARC do wejść i wyjść audio komputera pamiętając, że wejścia łączymy z wejściami, a wyjścia z wejściami audio komputera.
5. Zaprojektuj w programie Matlab filtr FIR wybierając jego parametry tak, aby otrzymać rozsądną, niezbyt dużą liczbę współczynników (maksymalnie 100).
6. Wykonaj modyfikacje projektu ‘C Sampled-Based TT’ wymienione we wstępie.
7. Modyfikacje wykonuj iteratywnie, kompilując zmiany, aż do skutku, którym jest poprawne działanie filtru obserwowane w programie Spectralab.
8. Dokonaj pomiaru czasu wykonania filtracji posługując się sygnalizacją bitami rejestru FLAG wyprowadzonymi na nóżki procesora podłączonych do diod LED. Informację o numerach nóżek i flag znajdziesz w dokumentacji modułu EzKit.
   Konfigurację nóżek procesora podłączonych do diod LED można przeprowadzić w pliku main.c lub w oddzielnym pliku zawierającym procedurę inicjującą:

```
SRU(FLAG6_O,DPI_PB06_I);  /* connect Flag6 output to DPI_PB06 input (LED1) */
SRU(FLAG7_O,DPI_PB13_I);  /* connect Flag7 output to DPI_PB13 input (LED2) */
SRU(FLAG4_O,DPI_PB14_I);  /* connect Flag4 output to DPI_PB14 input (LED3) */

SRU(HIGH,DPI_PBEN06_I);          /* LED1 */   // Enable outputs
SRU(HIGH,DPI_PBEN13_I);          /* LED2 */
SRU(HIGH,DPI_PBEN14_I);          /* LED3 */

/* setting flag pins as outputs */
sysreg_bit_set( sysreg_FLAGS, (FLG4O|FLG6O|FLG7O) );

/* clearing flag pins */
sysreg_bit_clr( sysreg_FLAGS, (FLG4|FLG6|FLG7) );
```

   Sygnalizację przeprowadzamy ustawiając flagę na ‘1’ przed obliczeniami, a zerując po:

```
sysreg_bit_set( sysreg_FLAGS, (FLG4|FLG6|FLG7) );
{ obliczenia …}
sysreg_bit_clr( sysreg_FLAGS, (FLG4|FLG6|FLG7) );
```

9. Drugim krokiem jest modyfikacja procedury filtrującej mającej na celu wykonanie obliczeń w trybie SIMD, co powinno przyśpieszyć obliczenia dwukrotnie. Rezultat należy sprawdzić dokonując pomiaru czasu trwania obliczeń na oscyloskopie korzystając z sygnalizacji bitami flag.
10. Policz na podstawie pomiarów oscyloskopowych ile średnio czasu trwa wykonanie jednego kroku splotu (operacji mnożenia i akumulacji) w trybie SISD i SIMD, i sprawdź, czy to zgadza się z częstotliwością taktowania procesora SHARC.
11. Powyższe czynności powtórz dla wersji blokowej filtracji FIR korzystając z projektu `‘AD1939_Block_Based_Talkthru_192kHz’` dla procesora ADSP-21479 dostępnego w tym miejscu:
    http://download.analog.com/tools/EZBoards/21479/Releases/Release_1.0.0/ADI_ADSP-21479_EZKIT-Rel1.0.0.exe
    (UWAGA: Projekt jest dostępny z menu: Help->Browse examples po zainstalowaniu pakietu przykładów z tego odnośnika)
12. Aby uzyskać blokową filtrację FIR w wersji stałoprzecinkowej należy wykonać następujące adaptacje projektu `‘AD1939_Block_Based_Talkthru_192kHz’`:

a. W pliku inicjacji kodeka audio `init1939viaSPI.c` zmienić częstotliwość próbkowania ADC i DAC na 48kHz.
b. W pliku `blockProcess_audio.c`:
   - założyć bufor cykliczny dla próbek sygnału wejściowego (typu `fract`), o długości co najmniej będącej sumą długości bloku próbek (`NUM_SAMPLES`) i ilości współczynników zaprojektowanego filtra FIR, oraz wskaźnik do ostatniej lub pierwszej próbki nowego bloku próbek.
   - Zakomentować linię 119:
     `memcopy(fBlockA.Rx_L1, fBlockA.Tx_L1, NUM_SAMPLES);`
   - Zakomentować linię 158 (aby nie zostały nadpisane przefiltrowane próbki wyjściowe) :
     `fixData(txA_block_pointer[blockIndex]+0, fBlockA.Tx_L1, NUM_TX_SLOTS, NUM_SAMPLES);`
   - Dopisać funkcję w C, która kopiuje blok próbek wejściowych kanału L1 (długość `NUM_SAMPLES`) w formacie `int` z bufora DMA `rxA_block_pointer[blockIndex]` do utworzonego bufora cyklicznego (co `NUM_RX_SLOTS` próbka !).
   - W linii 119, w miejsce zakomentowanej funkcji kopiowania bloków próbek typu `float` wstawić wywołanie blokowej funkcji filtrującej, której parametrami powinny być:
     - adres utworzonego bufora cyklicznego,
     - adres bufora DMA wyjściowych próbek typu `int` dla kanału L1 `txA_block_pointer[blockIndex]`, w którym należy umieścić przefiltrowane próbki z krokiem `NUM_TX_SLOTS`,
     - adres (wskaźnik) do pierwszej (lub ostatniej – zależy od gustu) nowej próbki w buforze cyklicznym próbek.
c. Utworzyć plik typu .asm, a w nim napisaną w asemblerze blokową funkcję filtrującą (powiększona o zewnętrzną pętlę o długości bloku próbek funkcję z wersji pojedynczo-próbkowej), w której musi znaleźć się wektor współczynników zaprojektowanego filtru. Próbki wyliczone należy wstawiać do wyjściowego bufora DMA (podanego jako parametr funkcji) z krokiem `NUM_TX_SLOTS`.