Государственный университет телекоммуникаций им. Бонч-Бруевича
кафедра «Цифровой обработки сигналов»

И. И. Гук,
доцент кафедры ЦОС
gook_igor@mail.ru

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторной работе № 5

«Оптимизация программного кода в интегрированной среде разработки CCS»

2008 год

Оглавление

***Замечание**. Рекомендуется использовать материалы статей «Реализация алгоритмов ЦОС на ассемблере TMS320C6000», «Оптимизация программного кода для ЦСП TMS320C6000» и «Тестирование программного кода для ЦСП TMS320C6000» напечатанной в журнале «Компоненты и Технологии» в 2007 году. Электронные версии данных статей и дополнительные материалы к ним можно найти на по адресу:*

http://www.scanti.ru/univ_stati.html

1. Создание проекта

Запустить интегрированную среду разработки (ИСР) *Code Composer Studio* (CCS) и создать новый проект с названием, например, «*prjKIX_CCS_asm*». После этого необходимо скопировать в папку нового проекта файлы «*main.cpp*», «*cnvKIX_my.asm*», «*runKIX.cpp*», «*initKIX.cpp*», «*constant.cpp*», «*prjKIX.h*» и «*standard.cmd*» из предыдущей лабораторной работы и добавить их к созданному проекту (кроме файла «*prjKIX.h*», который будет автоматически после компиляции проекта). Кроме этого, необходимо подключить библиотеку «*rts6400.lib*». Вид CCS после всех действий показан на рис.1.

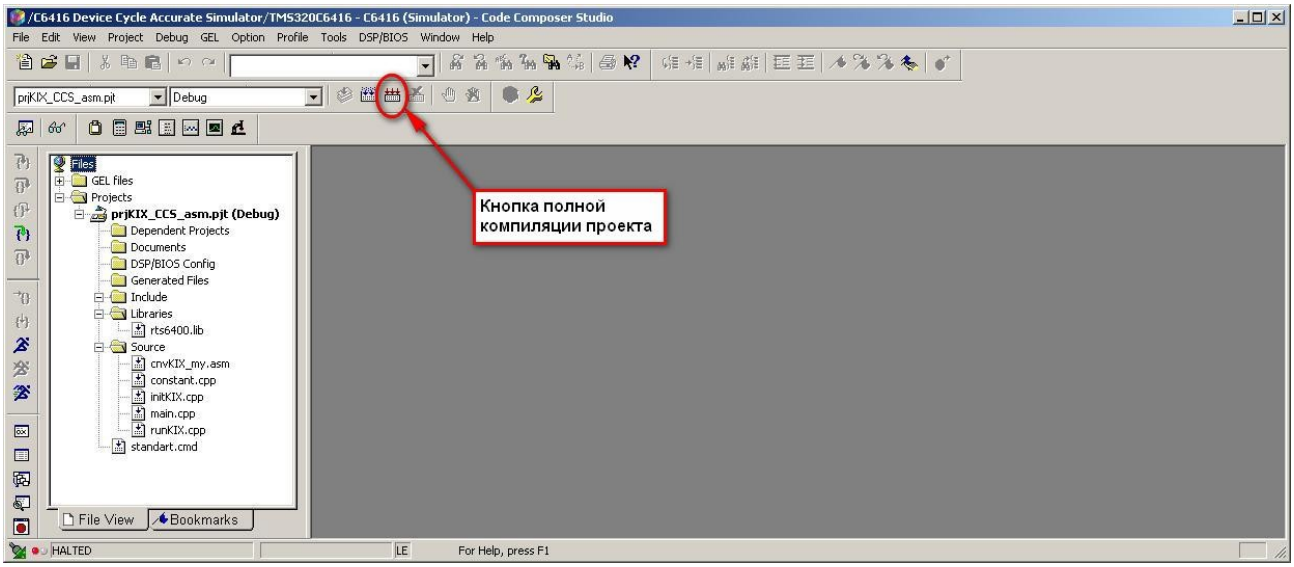


Рис.1. Вид CCS после создания проекта.

Затем уточняются настройки проекта. Для чего необходимо в главном меню CCS выбрать пункт «*Project->Build Options...*» и произвести настройку, как это описано в п.2 методических указаний к предыдущей лабораторной работе (см. описание процедуры начиная с рис.17 методических указаний к прошлой лабораторной работе). В заключение создания проекта его компилируют, нажав кнопку полной компиляции (значок  на панели кнопок быстрого запуска — см.рис.1). Вид CCS после произведенных действий показан на рис.2

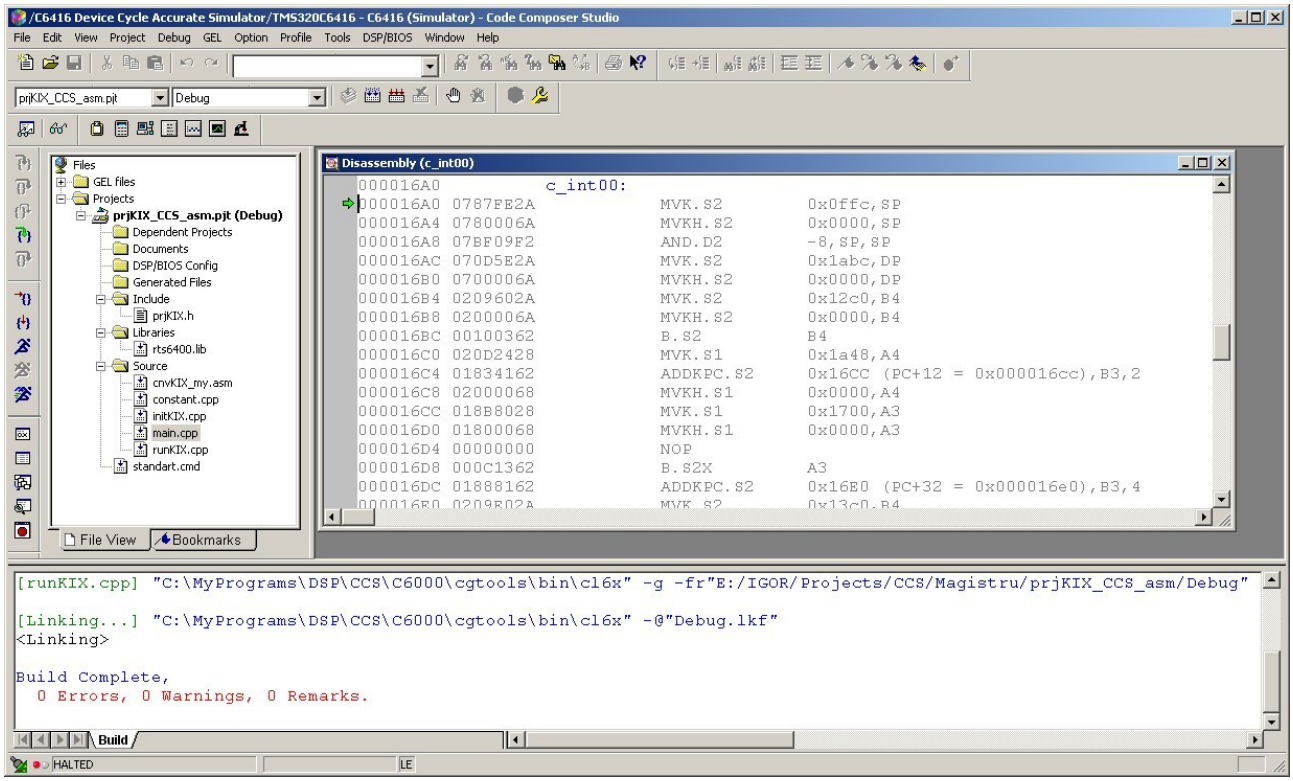


Рис.2. Вид CCS после компиляции проекта.

В проекте появится папка «*Debug*», куда необходимо скопировать файлы «*Char_To_Text.bat*», «*Char_To_Text_Out.bat*», «*Text_To_Char.bat*», «*inp.dat*», «*out.dat*» «*Char_To_Text.exe*» и «*Text_To_Char.exe*» из предыдущего проекта.
Подготовить входной файл «*inpCCS.dat*» при помощи командного файла «*Char_To_Text.bat*».
Подготовить выходного файла «*outCCS.dat*», запустив командный файл «*Char_To_Text_Out.bat*».

Установить точки тестирования (*Breakpoint*) в коде функции *main()* на строках содержащих код «`asm(" nop ");`». Напомним, что для установки точек тестирования необходимо поместить курсор на выбранную строку, а затем нажать кнопку  на панели быстрых кнопок (см.рис.2). Затем необходимо открыть окно точек тестирования, выбрав пункт «*Debug->Breakpoints...*» в главном меню CCS (см.рис2) и через окно свойств точек тестирования (см.рис.2) подключить к проекту входной и выходной файлы («*inpCCS.dat*» и «*outCCS.dat*», соответственно). Как это сделать подробно рассматривалось в п.3 методических рекомендаций к предыдущей лабораторной работе.

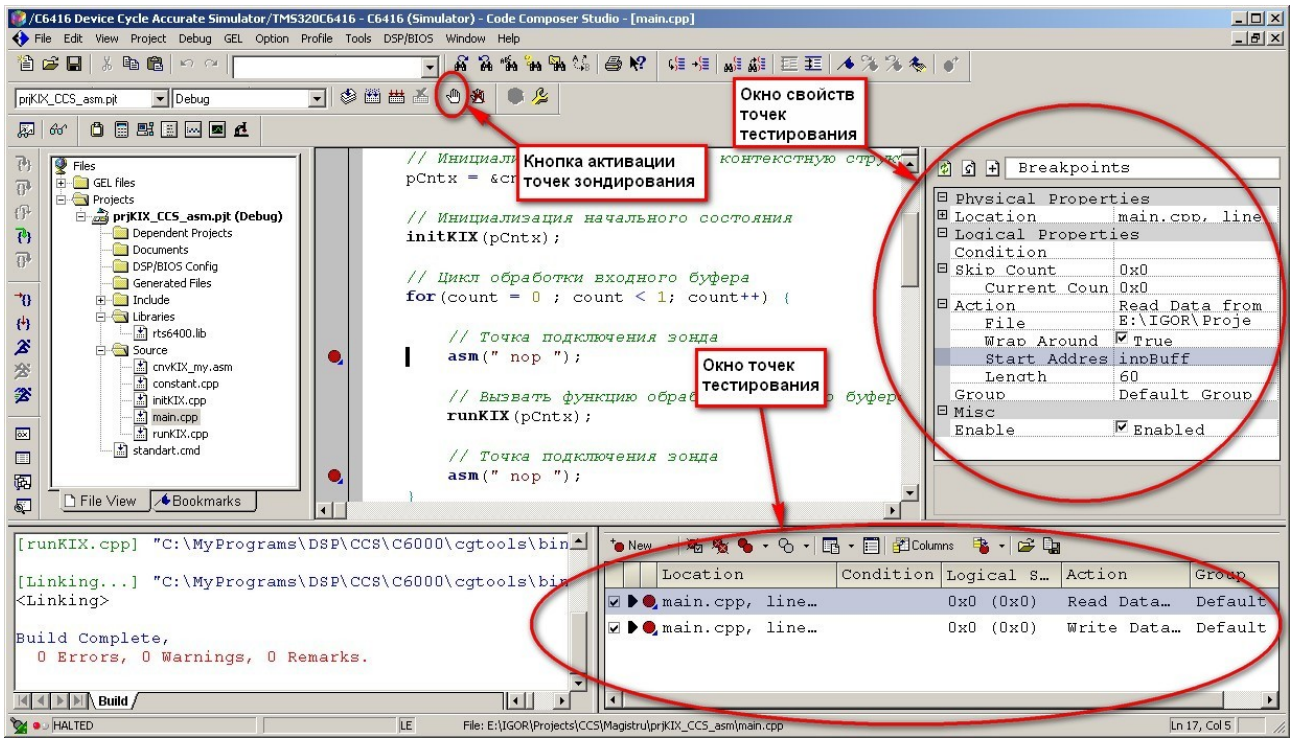


Рис.3. Подключение к проекту входного и выходного файлов.

Теперь необходимо откомпилировать проект (нажав кнопку ) и запустить его на выполнения (нажав кнопку ).
После завершения выполнения программного кода в папке «*Debug*» появиться файл «*outCCS.dat*», который необходимо преобразовать к нужному формату при помощи командного файла «*Text_To_Char.bat*». После запуска командного файла в папке «*Debug*» появится файл «*outForCmp.dat*». Его необходимо сравнить с файлом «*out.dat*» из предыдущей лабораторной работы и убедиться в корректной работе вновь собранного проекта.

2. Модификация ассемблерного кода функции *cnvKIX()*

Вначале необходимо создать новый файл и сохранить его под именем «*cnvKIX_my*01.*asm*» в папке проекта. Затем скопировать в него содержание файла «*cnvKIX_my.asm*», еще раз сохранить и подключить к проекту. После этого файл «*cnvKIX_my.asm*» необходимо исключить из процесса компилирования, откомпилировать проект и запустить на выполнение. Результат должен быть такой же, что и до создания файла и подключения к проекту «*cnvKIX_my*01.*asm*».

Затем для удобства дальнейшей оптимизации уберем из файла «*cnvKIX_my*01.*asm*» часть комментариев. Текст в файле примет вид:

```
; Назначение имен регистров
    .asg    A3,   coeff_A         ; word32 coeff;
    .asg    A4,   pCoeff_A        ; 1-ый прм. фнк. -  word16* pCoeff
    .asg    A5,   coeffB_A        ; word16 coeffB
    .asg    A6,   namberCoeff_A   ; 3-ый прм. фнк. -  word16 namberCoeff
    .asg    A7,   rezult_A        ; word32 rezult;

    .asg    B0,   count_B         ; word32 count
    .asg    B3,   adrReturn_B     ; Адрес возврата
    .asg    B4,   pShift_B        ; 2-ый прм. фнк. - word16* pShift
    .asg    B5,   coeffX_B        ; word16 coeffX;
    .asg    B6,   constA_B        ; 4-ый прм. фнк. - word32 constA
    .asg    B7,   CSROLD_B        ; Регистр для хранения значения регистра CSR
    .asg    B15,  SP              ; Указательна вершину стека

    .sect   ".text"
    .global _cnvKIX__FPsT1si

_cnvKIX__FPsT1si:

; Запрет прерываний
                    MVC     .S2     CSR, CSROLD_B
                    AND     .S2     CSROLD_B, -2, B1
                    MVC     .S2     B1, CSR

; Инициализация локальных переменных
                    ZERO    .D1     rezult_A
                    SUB     .L2X    namberCoeff_A, 1, count_B

; Цикл обработки линии задержки
loop01:
                    LDH     .D2T2   *pShift_B++, coeffX_B
                    LDH     .D1T1   *pCoeff_A++, coeffB_A
                    NOP 4
                    MPY     .M1X    coeffX_B, coeffB_A, coeff_A
                    NOP
                    ADD     .S1     rezult_A, coeff_A, rezult_A
        [count_B]   BDEC    .S2     loop01, count_B
                    NOP 5

; Корректировка значений
                    SUB     .L2     pShift_B, 2, pShift_B
                    SUB     .L2X    namberCoeff_A, 2, count_B

; Цикл сдвига линии задержки
loop02:
                    LDH     .D2T2   *pShift_B[-1], coeffX_B
                    NOP 4
                    STH     .D2T2   coeffX_B, *pShift_B--
        [count_B]   BDEC    .S2     loop02, count_B
                    NOP 5

; Переместить constA_B на сторону A.
                    MV      .S1X     constA_B, A2
; Нормирование результата
                    SHR     .S1     rezult_A, A2, A4
; Восстановление служебного регистра CSR
                    MVC     .S2     CSROLD_B, CSR
; Выход из функции
                    B       .S2     adrReturn_B
                    NOP 5
```

2.1. Оптимизация цикла обработки линии задержки

Дальнейшую модификацию будем проводить только для цикла обработки линии задержки:

```
; Цикл обработки линии задержки
loop01:
                    LDH     .D2T2   *pShift_B++, coeffX_B
                    LDH     .D1T1   *pCoeff_A++, coeffB_A
                    NOP 4
                    MPY     .M1X    coeffX_B, coeffB_A, coeff_A
                    NOP
                    ADD     .S1     rezult_A, coeff_A, rezult_A
          [count_B] BDEC    .S2     loop01, count_B
                    NOP 5
```

Немного видоизменим данный цикл:

```
; Цикл обработки линии задержки
loop01:
                    LDH     .D2T2   *pShift_B++, coeffX_B
                    LDH     .D1T1   *pCoeff_A++, coeffB_A
                    NOP
                    NOP
                    NOP             Вместо "NOP 4"
                    NOP
                    MPY     .M1X    coeffX_B, coeffB_A, coeff_A
                    NOP
                    ADD     .S1     rezult_A, coeff_A, rezult_A
          [count_B] BDEC    .S2     loop01, count_B
                    NOP
                    NOP
                    NOP             Вместо "NOP 5"
                    NOP
                    NOP
```

Следующий шаг — попытаться выполнить как можно больше операций паралельно. В нашем случае паралельно можно выполнить только две операции считывания данных:

```
; Цикл обработки линии задержки
loop01:
                    LDH     .D2T2   *pShift_B++, coeffX_B
    ||              LDH     .D1T1   *pCoeff_A++, coeffB_A
                    NOP
                    NOP
                    NOP
                    NOP
                    MPY     .M1X    coeffX_B, coeffB_A, coeff_A
                    NOP
                    ADD     .S1     rezult_A, coeff_A, rezult_A
          [count_B] BDEC    .S2     loop01, count_B
                    NOP
                    NOP
                    NOP
                    NOP
                    NOP
```

Затем — перенести операцию условного перехода на 5 тактов вверх:

```
; Цикл обработки линии задержки
loop01:
                    LDH     .D2T2   *pShift_B++, coeffX_B
    ||              LDH     .D1T1   *pCoeff_A++, coeffB_A
                    NOP
          [count_B] BDEC    .S2     loop01, count_B   Вместо 2-го "NOP"
                    NOP
                    NOP
5-ть тактов         MPY     .M1X    coeffX_B, coeffB_A, coeff_A
                    NOP
                    ADD     .S1     rezult_A, coeff_A, rezult_A
```

Обратите внимание, что операция условного перехода «BDEC» заменила один из операторов «NOP».

Теперь вынесем из цикла 8-мь тактов:

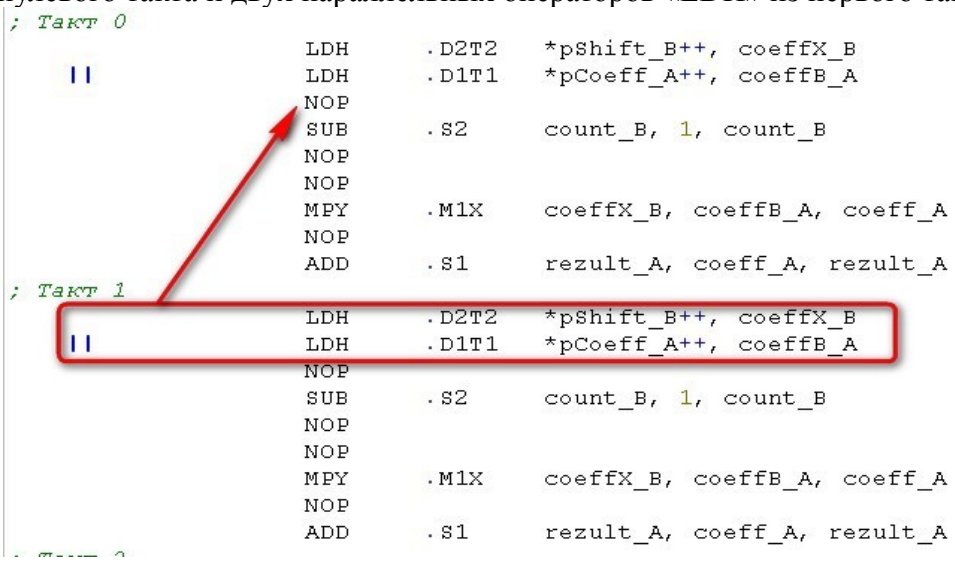
```
; Такт 0
                    LDH     .D2T2   *pShift_B++, coeffX_B
     ||             LDH     .D1T1   *pCoeff_A++, coeffB_A
                    NOP
                    SUB     .S2     count_B, 1, count_B
                    NOP
                    NOP
                    MPY     .M1X    coeffX_B, coeffB_A, coeff_A
                    NOP
                    ADD     .S1     rezult_A, coeff_A, rezult_A
; Такт 1
                    LDH     .D2T2   *pShift_B++, coeffX_B
     ||             LDH     .D1T1   *pCoeff_A++, coeffB_A
                    NOP
                    SUB     .S2     count_B, 1, count_B
                    NOP
                    NOP
                    MPY     .M1X    coeffX_B, coeffB_A, coeff_A
                    NOP
                    ADD     .S1     rezult_A, coeff_A, rezult_A
; Такт 2
                    LDH     .D2T2   *pShift_B++, coeffX_B
     ||             LDH     .D1T1   *pCoeff_A++, coeffB_A
                    NOP
                    SUB     .S2     count_B, 1, count_B
                    NOP
                    NOP
                    MPY     .M1X    coeffX_B, coeffB_A, coeff_A
                    NOP
                    ADD     .S1     rezult_A, coeff_A, rezult_A
```

```
; Такт 7
                    LDH     .D2T2   *pShift_B++, coeffX_B
     ||             LDH     .D1T1   *pCoeff_A++, coeffB_A
                    NOP
                    SUB     .S2     count_B, 1, count_B
                    NOP
                    NOP
                    MPY     .M1X    coeffX_B, coeffB_A, coeff_A
                    NOP
                    ADD     .S1     rezult_A, coeff_A, rezult_A

; Цикл обработки линии задержки
loop01:
                    LDH     .D2T2   *pShift_B++, coeffX_B
     ||             LDH     .D1T1   *pCoeff_A++, coeffB_A
                    NOP
         [count_B]  BDEC    .S2     loop01, count_B
                    NOP
                    NOP
                    MPY     .M1X    coeffX_B, coeffB_A, coeff_A
                    NOP
                    ADD     .S1     rezult_A, coeff_A, rezult_A
```

Обратите внимание, что оператор условного перехода «BDEC .S2 loop01, count_B» заменен на оператор «SUB .S2 count_B, 1, count_B».

Следующий шаг — это «склейка» тактов. Рассмотрим этот процесс на примере первых двух вынесенных из цикла тактов. Необходимо «склеить» (или объединить) первый оператор «NOP» из нулевого такта и двух параллельных операторов «LDH» из первого такта:



```
; Такт 0
                LDH     .D2T2   *pShift_B++, coeffX_B
||              LDH     .D1T1   *pCoeff_A++, coeffB_A
                NOP
                SUB     .S2     count_B, 1, count_B
                NOP
                NOP
                MPY     .M1X    coeffX_B, coeffB_A, coeff_A
                NOP
                ADD     .S1     rezult_A, coeff_A, rezult_A
; Такт 1
                LDH     .D2T2   *pShift_B++, coeffX_B
||              LDH     .D1T1   *pCoeff_A++, coeffB_A
                NOP
                SUB     .S2     count_B, 1, count_B
                NOP
                NOP
                MPY     .M1X    coeffX_B, coeffB_A, coeff_A
                NOP
                ADD     .S1     rezult_A, coeff_A, rezult_A
```

Затем «склеивается» оператор «SUB .S2 count_B, 1, count_B» нулевого такта с оператором «NOP» первого такта. Причем, при «склеивании» любого оператора с оператором «NOP», последний пропадает. Если склеиваются два оператора не являющиеся операторами «NOP», то они объединяются при помощи знака параллельности — две вертикальные черты.
В результате два склеенных такта примут вид:

```
; Такты 0 и 1
                LDH     .D2T2   *pShift_B++, coeffX_B
||              LDH     .D1T1   *pCoeff_A++, coeffB_A
                LDH     .D2T2   *pShift_B++, coeffX_B
||              LDH     .D1T1   *pCoeff_A++, coeffB_A
                SUB     .S2     count_B, 1, count_B
                SUB     .S2     count_B, 1, count_B
                NOP
                MPY     .M1X    coeffX_B, coeffB_A, coeff_A
                MPY     .M1X    coeffX_B, coeffB_A, coeff_A
                ADD     .S1     rezult_A, coeff_A, rezult_A
                ADD     .S1     rezult_A, coeff_A, rezult_A
```

Теперь начинаем добавлять к «склеенным» 0 и 1 тактам следующий — 2 такт. Процедура аналогичная описанной выше. К первому оператору «SUB» объеденных 0 и 1 тактов добавляется параллельные операторы «LDH» из 2 такта и т.д. В результате будем иметь следующее:

```
; Такты 0, 1 и 2
                LDH     .D2T2   *pShift_B++, coeffX_B
||              LDH     .D1T1   *pCoeff_A++, coeffB_A

                LDH     .D2T2   *pShift_B++, coeffX_B
||              LDH     .D1T1   *pCoeff_A++, coeffB_A

                SUB     .S2     count_B, 1, count_B
||              LDH     .D2T2   *pShift_B++, coeffX_B
||              LDH     .D1T1   *pCoeff_A++, coeffB_A

                SUB     .S2     count_B, 1, count_B

                SUB     .S2     count_B, 1, count_B

                MPY     .M1X    coeffX_B, coeffB_A, coeff_A

                MPY     .M1X    coeffX_B, coeffB_A, coeff_A

                ADD     .S1     rezult_A, coeff_A, rezult_A
||              MPY     .M1X    coeffX_B, coeffB_A, coeff_A

                ADD     .S1     rezult_A, coeff_A, rezult_A

                ADD     .S1     rezult_A, coeff_A, rezult_A
```

Теперь добавляем 3 такт:

```
; Такты 0, 1, 2 и 3
                    LDH     .D2T2   *pShift_B++, coeffX_B
||                  LDH     .D1T1   *pCoeff_A++, coeffB_A

                    LDH     .D2T2   *pShift_B++, coeffX_B
||                  LDH     .D1T1   *pCoeff_A++, coeffB_A

                    SUB     .S2     count_B, 1, count_B
||                  LDH     .D2T2   *pShift_B++, coeffX_B
||                  LDH     .D1T1   *pCoeff_A++, coeffB_A

                    SUB     .S2     count_B, 1, count_B
||                  LDH     .D2T2   *pShift_B++, coeffX_B
||                  LDH     .D1T1   *pCoeff_A++, coeffB_A

                    SUB     .S2     count_B, 1, count_B

                    MPY     .M1X    coeffX_B, coeffB_A, coeff_A
||                  SUB     .S2     count_B, 1, count_B

                    MPY     .M1X    coeffX_B, coeffB_A, coeff_A

                    ADD     .S1     rezult_A, coeff_A, rezult_A
||                  MPY     .M1X    coeffX_B, coeffB_A, coeff_A

                    ADD     .S1     rezult_A, coeff_A, rezult_A
||                  MPY     .M1X    coeffX_B, coeffB_A, coeff_A

                    ADD     .S1     rezult_A, coeff_A, rezult_A

                    ADD     .S1     rezult_A, coeff_A, rezult_A
```

В результате склейки всех 8 тактов код примет вид:

```
; Такты 0, 1, 2, 3, 4, 5, 6 и 7
                    LDH     .D2T2   *pShift_B++, coeffX_B
||                  LDH     .D1T1   *pCoeff_A++, coeffB_A

                    LDH     .D2T2   *pShift_B++, coeffX_B
||                  LDH     .D1T1   *pCoeff_A++, coeffB_A

                    SUB     .S2     count_B, 1, count_B
||                  LDH     .D2T2   *pShift_B++, coeffX_B
||                  LDH     .D1T1   *pCoeff_A++, coeffB_A

                    SUB     .S2     count_B, 1, count_B
||                  LDH     .D2T2   *pShift_B++, coeffX_B
||                  LDH     .D1T1   *pCoeff_A++, coeffB_A

                    SUB     .S2     count_B, 1, count_B
||                  LDH     .D2T2   *pShift_B++, coeffX_B
||                  LDH     .D1T1   *pCoeff_A++, coeffB_A

                    MPY     .M1X    coeffX_B, coeffB_A, coeff_A
||                  SUB     .S2     count_B, 1, count_B
||                  LDH     .D2T2   *pShift_B++, coeffX_B
||                  LDH     .D1T1   *pCoeff_A++, coeffB_A

                    MPY     .M1X    coeffX_B, coeffB_A, coeff_A
||                  SUB     .S2     count_B, 1, count_B
||                  LDH     .D2T2   *pShift_B++, coeffX_B
||                  LDH     .D1T1   *pCoeff_A++, coeffB_A

                    ADD     .S1     rezult_A, coeff_A, rezult_A
||                  MPY     .M1X    coeffX_B, coeffB_A, coeff_A
||                  SUB     .S2     count_B, 1, count_B
||                  LDH     .D2T2   *pShift_B++, coeffX_B
||                  LDH     .D1T1   *pCoeff_A++, coeffB_A

                    ADD     .S1     rezult_A, coeff_A, rezult_A
||                  MPY     .M1X    coeffX_B, coeffB_A, coeff_A
||                  SUB     .S2     count_B, 1, count_B

                    ADD     .S1     rezult_A, coeff_A, rezult_A
||                  MPY     .M1X    coeffX_B, coeffB_A, coeff_A
||                  SUB     .S2     count_B, 1, count_B

                    ADD     .S1     rezult_A, coeff_A, rezult_A
||                  MPY     .M1X    coeffX_B, coeffB_A, coeff_A

                    ADD     .S1     rezult_A, coeff_A, rezult_A
||                  MPY     .M1X    coeffX_B, coeffB_A, coeff_A

                    ADD     .S1     rezult_A, coeff_A, rezult_A

                    ADD     .S1     rezult_A, coeff_A, rezult_A
```

Обратите внимание на выделенный блок — он содержит все операторы цикла обработки линии задержки, который мы оптимизируем. Данный блок будет ядром нашего нового цикла обработки линии задержки. Все что находиться до него — это пролог, после — эпилог цикла:

```
; Инициализация локальных переменных
                      ZERO    .D1     rezult_A
                      SUB     .L2X    namberCoeff_A, 8, count_B

; Цикл обработки линии задержки
; Пролог цикла
                      LDH     .D2T2   *pShift_B++, coeffX_B
    ||                LDH     .D1T1   *pCoeff_A++, coeffB_A

                      LDH     .D2T2   *pShift_B++, coeffX_B
    ||                LDH     .D1T1   *pCoeff_A++, coeffB_A

        [count_B]     BDEC    .S2     loop01, count_B
    ||                LDH     .D2T2   *pShift_B++, coeffX_B
    ||                LDH     .D1T1   *pCoeff_A++, coeffB_A

        [count_B]     BDEC    .S2     loop01, count_B
    ||                LDH     .D2T2   *pShift_B++, coeffX_B
    ||                LDH     .D1T1   *pCoeff_A++, coeffB_A

        [count_B]     BDEC    .S2     loop01, count_B
    ||                LDH     .D2T2   *pShift_B++, coeffX_B
    ||                LDH     .D1T1   *pCoeff_A++, coeffB_A

        [count_B]     BDEC    .S2     loop01, count_B
    ||                MPY     .M1X    coeffX_B, coeffB_A, coeff_A
    ||                LDH     .D2T2   *pShift_B++, coeffX_B
    ||                LDH     .D1T1   *pCoeff_A++, coeffB_A

        [count_B]     BDEC    .S2     loop01, count_B
    ||                MPY     .M1X    coeffX_B, coeffB_A, coeff_A
    ||                LDH     .D2T2   *pShift_B++, coeffX_B
    ||                LDH     .D1T1   *pCoeff_A++, coeffB_A
; Ядро цикла
loop01:
        [count_B]     BDEC    .S2     loop01, count_B
    ||                ADD     .S1     rezult_A, coeff_A, rezult_A
    ||                MPY     .M1X    coeffX_B, coeffB_A, coeff_A
    ||                LDH     .D2T2   *pShift_B++, coeffX_B
    ||                LDH     .D1T1   *pCoeff_A++, coeffB_A

; Эпилог цикла
                      ADD     .S1     rezult_A, coeff_A, rezult_A
    ||                MPY     .M1X    coeffX_B, coeffB_A, coeff_A

                      ADD     .S1     rezult_A, coeff_A, rezult_A
    ||                MPY     .M1X    coeffX_B, coeffB_A, coeff_A

                      ADD     .S1     rezult_A, coeff_A, rezult_A
    ||                MPY     .M1X    coeffX_B, coeffB_A, coeff_A

                      ADD     .S1     rezult_A, coeff_A, rezult_A
    ||                MPY     .M1X    coeffX_B, coeffB_A, coeff_A

                      ADD     .S1     rezult_A, coeff_A, rezult_A
    ||                MPY     .M1X    coeffX_B, coeffB_A, coeff_A

                      ADD     .S1     rezult_A, coeff_A, rezult_A

                      ADD     .S1     rezult_A, coeff_A, rezult_A

; Цикл обработки линии задержки
;loop01:
;                     LDH     .D2T2   *pShift_B++, coeffX_B
;   ||                LDH     .D1T1   *pCoeff_A++, coeffB_A
;                     NOP
;       [count_B]     BDEC    .S2     loop01, count_B
;                     NOP
;                     NOP
;                     MPY     .M1X    coeffX_B, coeffB_A, coeff_A
;                     NOP
;                     ADD     .S1     rezult_A, coeff_A, rezult_A
```

Обратите внимание на следующие моменты:

1. При инициализации количества повторений цикла (переменная «count_B») оно уменьшено по сравнению с исходным значением на 7 (в итоге получается уменьшение на 8, так как в исходном коде уже было уменьшение на единицу).
2. Все операторы «SUB .S2 count_B, 1, count_B» в прологе и ядре цикла заменены на «[count_B] BDEC .S2 loop01, count_B», причем все они должны быть первым оператором в блоке параллельных операций.
3. Перед ядром цикла ставиться метка «loop01:».
4. Все операторы «SUB .S2 count_B, 1, count_B» в эпилоге цикла удаляются.
5. Старый цикл обработки линии задержки комментируется, или полностью удаляется.

Оптимизация цикла обработки линии задержки завершена.

2.2. Оптимизация цикла сдвига линии задержки

Теперь повторим описанную выше процедуру для второго цикла — цикла сдвига линии задержки:

```
; Цикл сдвига линии задержки
loop02:
                    LDH     .D2T2   *pShift_B[-1], coeffX_B
                    NOP 4
                    STH     .D2T2   coeffX_B, *pShift_B--
          [count_B] BDEC    .S2     loop02, count_B
                    NOP 5
```

Первый шаг в данном случае — проанализировать код и решить, сколько блоков параллельных операций может быть в будущем ядре оптимизированного цикла. Оценка происходит исходя из общего числа ненулевых (то есть не «NOP») операций в цикле и возможных конфликтов модулей. В приведенном выше цикле всего три операции. Так как возможно выполнение до 8-ми операций в блоке параллельных команд, то с этой точки зрения возможно все операции выполнить в одном блоке. Но! Две операции используют один и тот же модуль (модуль «.D2»), что вынуждает либо организовать два блока параллельных операций в ядре цикла, либо модифицировать код таким образом, что бы избавиться от конфликта модулей. Так как операций всего три, целесообразно попытаться избавиться от конфликта модулей.

Для этого перепишем код в виде:

```
; Инициализация дополнительного указателя
                    SUB     .S1X    pShift_B, 2, pCoeff_A
; Цикл сдвига линии задержки
loop02:
                    LDH     .D1T1   *pCoeff_A--, coeff_A
                    NOP 4
                    MV      .L2X    coeff_A, coeffX_B
                    STH     .D2T2   coeffX_B, *pShift_B--
          [count_B] BDEC    .S2     loop02, count_B
                    NOP 5
```

К коду добавлен дополнительный указатель на линию задержки (он будет храниться в регистрах на стороне «A» в регистре «pCoeff_A») и проведена его инициализация. Значение элемента линии задержки вначале считывается в переменную «coeff_A», которая так же находиться на стороне «A», затем переноситься в переменную «coeffX_B» на стороне «B» и записывается в буфер уже при помощи указателя «pShift_B», находящегося так же на стороне «B». Таким образом мы избавились от конфликта модулей за счет добавление дополнительных операций. Однако, только одна из добавленных операций находиться в цикле, следовательно, число активных операций (не «NOP») меньше 8-ми (а именно — четыре), что позволяет организовать только один блок параллельных операций в ядре цикла.

Небольшое замечание, после каждого шага желательно проводить повторную компиляцию проекта (можно пользоваться кнопкой  на панели быстрых кнопок CCS), формирование тестового выходного файла «outForCmp.dat» (при помощи командного файла «Text_To_Char.bat») и сравнение результата с тестовым выходным файлом «out.dat». Это позволит выявить ошибки на ранних этапах оптимизации и упростить процесс отладки.

Второй шаг — замена операторов «NOP»:

```
        ; Инициализация дополнительного указателя
                            SUB     .S1X    pShift_B, 2, pCoeff_A
        ; Цикл сдвига линии задержки
        loop02:
                            LDH     .D1T1   *pCoeff_A--, coeff_A
                            NOP
                            NOP
                            NOP
                            NOP
                            MV      .L2X    coeff_A, coeffX_B
                            STH     .D2T2   coeffX_B, *pShift_B--
                  [count_B] BDEC    .S2     loop02, count_B
                            NOP
                            NOP
                            NOP
                            NOP
                            NOP
```

Третий шаг — перенос оператора «BDEC» вверх на 5-ть тактов:

```
        ; Цикл сдвига линии задержки
        loop02:
                            LDH     .D1T1   *pCoeff_A--, coeff_A
                  [count_B] BDEC    .S2     loop02, count_B
                            NOP
                            NOP
                            NOP
                            MV      .L2X    coeff_A, coeffX_B
                            STH     .D2T2   coeffX_B, *pShift_B--
```

Четвертый шаг — выбор паралельно выполняемых команд. В данном случае такими команд нет.

Пятый шаг — вынос нескольких тактов из цикла (в данном случае 8-ми):

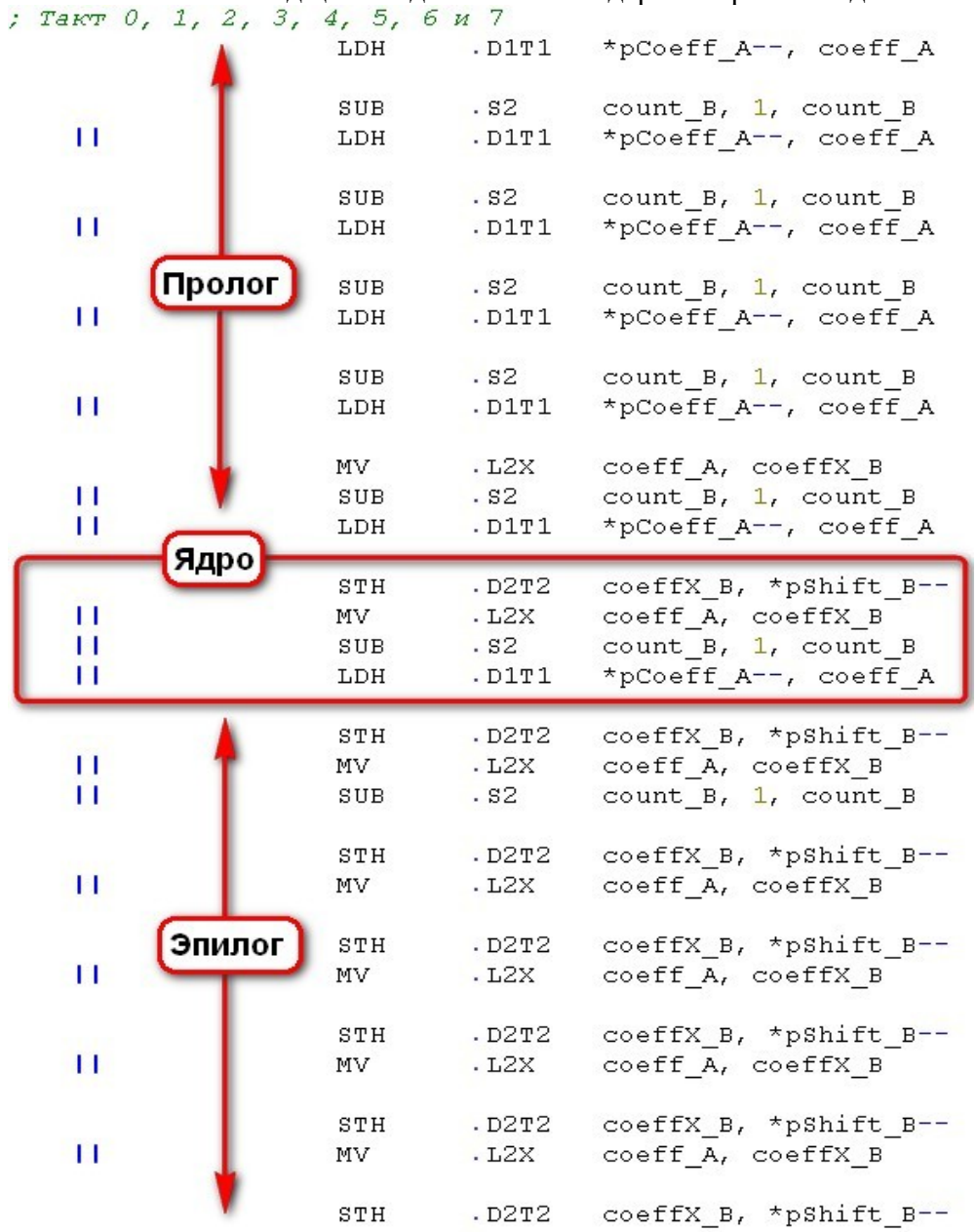
```
; Такт 0
                    LDH     .D1T1   *pCoeff_A--, coeff_A
                    SUB     .S2     count_B, 1, count_B
                    NOP
                    NOP
                    NOP
                    MV      .L2X    coeff_A, coeffX_B
                    STH     .D2T2   coeffX_B, *pShift_B--
; Такт 1
                    LDH     .D1T1   *pCoeff_A--, coeff_A
                    SUB     .S2     count_B, 1, count_B
                    NOP
                    NOP
                    NOP
                    MV      .L2X    coeff_A, coeffX_B
                    STH     .D2T2   coeffX_B, *pShift_B--

                    o   o   o

; Такт 7
                    LDH     .D1T1   *pCoeff_A--, coeff_A
                    SUB     .S2     count_B, 1, count_B
                    NOP
                    NOP
                    NOP
                    MV      .L2X    coeff_A, coeffX_B
                    STH     .D2T2   coeffX_B, *pShift_B--

; Цикл сдвига линии задержки
loop02:
                    LDH     .D1T1   *pCoeff_A--, coeff_A
            [count_B] BDEC  .S2     loop02, count_B
                    NOP
                    NOP
                    NOP
                    MV      .L2X    coeff_A, coeffX_B
                    STH     .D2T2   coeffX_B, *pShift_B--
```

Обратите внимание на замену в вынесенных тактах оператора «BDEC» на оператор «SUB».
Шестой шаг — «склеивание» вынесенных из цикла тактов. Обратите внимание, что склеивание производиться со сдвигом на один такт, то есть ко второму оператору 0-го такта добавляется первый оператор 1-го такта. Запомните, что это верно только в том случае, если предполагается наличие только одного блока параллельных команд в ядре будущего цикла. Если же предполагается, что в ядре будет два блока параллельных команд, то сдвигать при «склеивании» необходимо на два такта. Если в ядре — три блока параллельных команд, то сдвиг — на три такта. И так далее.
После выполнения 6-го шага код цикла сдвига линии задержки примет вид:




```
; Такт 0, 1, 2, 3, 4, 5, 6 и 7
                    LDH     .D1T1   *pCoeff_A--, coeff_A

                    SUB     .S2     count_B, 1, count_B
||                  LDH     .D1T1   *pCoeff_A--, coeff_A

                    SUB     .S2     count_B, 1, count_B
||                  LDH     .D1T1   *pCoeff_A--, coeff_A

                    SUB     .S2     count_B, 1, count_B
||                  LDH     .D1T1   *pCoeff_A--, coeff_A

                    SUB     .S2     count_B, 1, count_B
||                  LDH     .D1T1   *pCoeff_A--, coeff_A

                    MV      .L2X    coeff_A, coeffX_B
||                  SUB     .S2     count_B, 1, count_B
||                  LDH     .D1T1   *pCoeff_A--, coeff_A

                    STH     .D2T2   coeffX_B, *pShift_B--
||                  MV      .L2X    coeff_A, coeffX_B
||                  SUB     .S2     count_B, 1, count_B
||                  LDH     .D1T1   *pCoeff_A--, coeff_A

                    STH     .D2T2   coeffX_B, *pShift_B--
||                  MV      .L2X    coeff_A, coeffX_B
||                  SUB     .S2     count_B, 1, count_B

                    STH     .D2T2   coeffX_B, *pShift_B--
||                  MV      .L2X    coeff_A, coeffX_B

                    STH     .D2T2   coeffX_B, *pShift_B--
||                  MV      .L2X    coeff_A, coeffX_B

                    STH     .D2T2   coeffX_B, *pShift_B--
||                  MV      .L2X    coeff_A, coeffX_B

                    STH     .D2T2   coeffX_B, *pShift_B--
||                  MV      .L2X    coeff_A, coeffX_B

                    STH     .D2T2   coeffX_B, *pShift_B--

; Цикл сдвига линии задержки
loop02:
                    LDH     .D1T1   *pCoeff_A--, coeff_A
        [count_B]   BDEC    .S2     loop02, count_B
                    NOP
                    NOP
                    NOP
                    MV      .L2X    coeff_A, coeffX_B
                    STH     .D2T2   coeffX_B, *pShift_B--
```

Цветом выделено будущее ядро (блок параллельных операндов, включающих все операции оптимизируемого цикла), пролог и эпилог.

Седьмой шаг — окончательная оптимизация циклических вычислений:

- уменьшить количества повторений на 6-ть (переменная «count_B»), в итоге получается уменьшение на 8, так как в исходном коде уже было уменьшение на два;
- производиться замена операторы «SUB .S2 count_B, 1, count_B» в прологе и ядре цикла заменены на «[count_B] BDEC .S2 loop02, count_B», причем все они должны быть первым оператором в блоке параллельных операций;
- перед ядром цикла ставится метка «loop02:»;
- операторы «SUB .S2 count_B, 1, count_B» в эпилоге цикла удаляются.
- старый цикл обработки линии задержки комментируется, или полностью удаляется.

После этого код примет вид:

```
; Корректировка значений
                    SUB     .L2     pShift_B, 2, pShift_B
                    SUB     .L2X    namberCoeff_A, 8, count_B
; Инициализация дополнительного указателя
                    SUB     .S1X    pShift_B, 2, pCoeff_A
; Цикл сдвига линии задержки
; Пролог
                    LDH     .D1T1   *pCoeff_A--, coeff_A

        [count_B]   BDEC    .S2     loop02, count_B
    ||              LDH     .D1T1   *pCoeff_A--, coeff_A

        [count_B]   BDEC    .S2     loop02, count_B
    ||              LDH     .D1T1   *pCoeff_A--, coeff_A

        [count_B]   BDEC    .S2     loop02, count_B
    ||              LDH     .D1T1   *pCoeff_A--, coeff_A

        [count_B]   BDEC    .S2     loop02, count_B
    ||              LDH     .D1T1   *pCoeff_A--, coeff_A

        [count_B]   BDEC    .S2     loop02, count_B
    ||              MV      .L2X    coeff_A, coeffX_B
    ||              LDH     .D1T1   *pCoeff_A--, coeff_A

; Ядро
loop02:
        [count_B]   BDEC    .S2     loop02, count_B
    ||              STH     .D2T2   coeffX_B, *pShift_B--
    ||              MV      .L2X    coeff_A, coeffX_B
    ||              LDH     .D1T1   *pCoeff_A--, coeff_A

; Эпилог
                    STH     .D2T2   coeffX_B, *pShift_B--
    ||              MV      .L2X    coeff_A, coeffX_B

                    STH     .D2T2   coeffX_B, *pShift_B--
    ||              MV      .L2X    coeff_A, coeffX_B

                    STH     .D2T2   coeffX_B, *pShift_B--
    ||              MV      .L2X    coeff_A, coeffX_B

                    STH     .D2T2   coeffX_B, *pShift_B--
    ||              MV      .L2X    coeff_A, coeffX_B

                    STH     .D2T2   coeffX_B, *pShift_B--
    ||              MV      .L2X    coeff_A, coeffX_B

                    STH     .D2T2   coeffX_B, *pShift_B--

; Цикл сдвига линии задержки
;loop02:
;                   LDH     .D1T1   *pCoeff_A--, coeff_A
;       [count_B]   BDEC    .S2     loop02, count_B
;                   NOP
;                   NOP
;                   NOP
;                   MV      .L2X    coeff_A, coeffX_B
;                   STH     .D2T2   coeffX_B, *pShift_B--
```


Оптимизация цикла сдвига линии задержки завершена.

3. Дополнительное задание

Необходимо провести оптимизацию одноциклового варианта функции *cnvKIX()* (приложение 2), ассемблерный код которой Вы должны были написать в качестве дополнительного задания в прошлой лабораторной работе.

4. Приложения

Приложение 1.

Листинг ассемблерного оптимизированного варианта функции *cnvKIX()*

```
; Назначение имен регистров
    .asg    A3,  coeff_A          ; word32 coeff;
    .asg    A4,  pCoeff_A         ; 1-ый прм. фнк. -  word16* pCoeff
    .asg    A5,  coeffB_A         ; word16 coeffB
    .asg    A6,  namberCoeff_A    ; 3-ый прм. фнк. -  word16 namberCoeff
    .asg    A7,  rezult_A         ; word32 rezult;

    .asg    B0,  count_B          ; word32 count
    .asg    B3,  adrReturn_B      ; Адрес возврата
    .asg    B4,  pShift_B         ; 2-ый прм. фнк. - word16* pShift
    .asg    B5,  coeffX_B         ; word16 coeffX;
    .asg    B6,  constA_B         ; 4-ый прм. фнк. - word32 constA
    .asg    B7,  CSROLD_B         ; Регистр для хранения значения регистра CSR
    .asg    B15, SP               ; Указательна вершину стека

    .sect   ".text"
    .global _cnvKIX__FPsT1si

_cnvKIX__FPsT1si:

; Запрет прерываний
                    MVC     .S2     CSR, CSROLD_B
                    AND     .S2     CSROLD_B, -2, B1
                    MVC     .S2     B1, CSR

; Инициализация локальных переменных
                    ZERO    .D1     rezult_A
                    SUB     .L2X    namberCoeff_A, 8, count_B

; Цикл обработки линии задержки
; Пролог цикла
                    LDH     .D2T2   *pShift_B++, coeffX_B
    ||              LDH     .D1T1   *pCoeff_A++, coeffB_A

                    LDH     .D2T2   *pShift_B++, coeffX_B
    ||              LDH     .D1T1   *pCoeff_A++, coeffB_A

        [count_B]   BDEC    .S2     loop01, count_B
    ||              LDH     .D2T2   *pShift_B++, coeffX_B
    ||              LDH     .D1T1   *pCoeff_A++, coeffB_A

        [count_B]   BDEC    .S2     loop01, count_B
    ||              LDH     .D2T2   *pShift_B++, coeffX_B
    ||              LDH     .D1T1   *pCoeff_A++, coeffB_A

        [count_B]   BDEC    .S2     loop01, count_B
    ||              LDH     .D2T2   *pShift_B++, coeffX_B
    ||              LDH     .D1T1   *pCoeff_A++, coeffB_A

        [count_B]   BDEC    .S2     loop01, count_B
    ||              MPY     .M1X    coeffX_B, coeffB_A, coeff_A
    ||              LDH     .D2T2   *pShift_B++, coeffX_B
    ||              LDH     .D1T1   *pCoeff_A++, coeffB_A

        [count_B]   BDEC    .S2     loop01, count_B
    ||              MPY     .M1X    coeffX_B, coeffB_A, coeff_A
    ||              LDH     .D2T2   *pShift_B++, coeffX_B
    ||              LDH     .D1T1   *pCoeff_A++, coeffB_A
; Ядро цикла
loop01:
        [count_B]   BDEC    .S2     loop01,       count_B
    ||              ADD     .S1     rezult_A,     coeff_A,   rezult_A
    ||              MPY     .M1X    coeffX_B,     coeffB_A,  coeff_A
    ||              LDH     .D2T2   *pShift_B++,  coeffX_B
    ||              LDH     .D1T1   *pCoeff_A++,  coeffB_A

; Эпилог цикла
                    ADD     .S1     rezult_A, coeff_A, rezult_A
    ||              MPY     .M1X    coeffX_B, coeffB_A, coeff_A

                    ADD     .S1     rezult_A, coeff_A, rezult_A
    ||              MPY     .M1X    coeffX_B, coeffB_A, coeff_A

                    ADD     .S1     rezult_A, coeff_A, rezult_A
    ||              MPY     .M1X    coeffX_B, coeffB_A, coeff_A

                    ADD     .S1     rezult_A, coeff_A, rezult_A
    ||              MPY     .M1X    coeffX_B, coeffB_A, coeff_A

                    ADD     .S1     rezult_A, coeff_A, rezult_A
    ||              MPY     .M1X    coeffX_B, coeffB_A, coeff_A

                    ADD     .S1     rezult_A, coeff_A, rezult_A

                    ADD     .S1     rezult_A, coeff_A, rezult_A
```

```
; Корректировка значений
                    SUB     .L2     pShift_B, 2, pShift_B
                    SUB     .L2X    namberCoeff_A, 8, count_B

; Инициализация дополнительного указателя
                    SUB     .S1X    pShift_B, 2, pCoeff_A
; Цикл сдвига линии задержки
; Пролог
                    LDH     .D1T1   *pCoeff_A--, coeff_A

        [count_B]   BDEC    .S2     loop02, count_B
    ||              LDH     .D1T1   *pCoeff_A--, coeff_A

        [count_B]   BDEC    .S2     loop02, count_B
    ||              LDH     .D1T1   *pCoeff_A--, coeff_A

        [count_B]   BDEC    .S2     loop02, count_B
    ||              LDH     .D1T1   *pCoeff_A--, coeff_A

        [count_B]   BDEC    .S2     loop02, count_B
    ||              LDH     .D1T1   *pCoeff_A--, coeff_A

        [count_B]   BDEC    .S2     loop02, count_B
    ||              MV      .L2X    coeff_A, coeffX_B
    ||              LDH     .D1T1   *pCoeff_A--, coeff_A

; Ядро
loop02:
        [count_B]   BDEC    .S2     loop02, count_B
    ||              STH     .D2T2   coeffX_B, *pShift_B--
    ||              MV      .L2X    coeff_A, coeffX_B
    ||              LDH     .D1T1   *pCoeff_A--, coeff_A

; Эпилог
                    STH     .D2T2   coeffX_B, *pShift_B--
    ||              MV      .L2X    coeff_A, coeffX_B

                    STH     .D2T2   coeffX_B, *pShift_B--
    ||              MV      .L2X    coeff_A, coeffX_B

                    STH     .D2T2   coeffX_B, *pShift_B--
    ||              MV      .L2X    coeff_A, coeffX_B

                    STH     .D2T2   coeffX_B, *pShift_B--
    ||              MV      .L2X    coeff_A, coeffX_B

                    STH     .D2T2   coeffX_B, *pShift_B--
    ||              MV      .L2X    coeff_A, coeffX_B

                    STH     .D2T2   coeffX_B, *pShift_B--

; Переместить constA_B на сторону A.
                    MV      .S1X     constA_B, A2
; Нормирование результата
                    SHR     .S1     rezult_A, A2, A4
; Восстановление служебного регистра CSR
                    MVC     .S2     CSROLD_B, CSR
; Выход из функции
                    B       .S2     adrReturn_B
                    NOP 5
```

Листинг одноциклового варианта С-кода функции *cnvKIX()*

```
// Функция вычисление свертки
#include "prjKIX.h"

word16 cnvKIX(word16* pCoeff, word16* pShift, word16 namberCoeff, word32 constA){
    // Объявление локальных переменных
    word16 coeffX;
    word16 coeffB;
    word32 count;
    word32 coeff;
    word32 rezult;

    // Корректировка значения
    namberCoeff--;

    // Установка указателей на конец буферов отсчетов и коэффициентов
    pShift += namberCoeff;
    pCoeff += namberCoeff;

    // Чтение текущего отсчета
    coeffX = *pShift--;

    // Чтение соответствующего коэффициента фильтра
    coeffB = *pCoeff--;

    //  Инициализация начального значения результата
    rezult = coeffX * coeffB;

    // Цикл обработки линии задержки
    for(count = 0; count < namberCoeff; count++){

        // Чтение текущего отсчета
        coeffX = *pShift--;

        // Запись текущего отсчета в следующую ячейку
        pShift[2] = coeffX;

        // Чтение соответствующего коэффициента фильтра
        coeffB = *pCoeff--;

        // Умножение отсчета на коэффициент
        coeff = coeffX * coeffB;

        // Накопление результата
        rezult += coeff;
    }

    // Нормирование результата суммирования
    rezult >>= constA;

    // Выход из функции
    return rezult;
}
```