Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Оренбургский государственный университет»

Кафедра промышленной электроники и
информационно-измерительной техники

А.В. Хлуденев

ОБРАБОТКА СИГНАЛОВ НА ПЛАТФОРМЕ dsPIC

Методические указания

Рекомендовано к изданию редакционно-издательским советом федерального государственного бюджетного образовательного учреждения высшего образования «Оренбургский государственный университет» для обучающихся по образовательной программе высшего образования по направлению подготовки 11.03.04 Электроника и наноэлектроника

Оренбург
2018

УДК 004.4 : 681.3.068
ББК 32.973.26
Х 60

Рецензент - кандидат технических наук, доцент С.А. Сильвашко

Хлуденев, А.В.
Х 60 Обработка сигналов на платформе dsPIC: методические указания / А.В.Хлуденев; Оренбургский гос. ун-т. – Оренбург : ОГУ, 2018.

Методические указания содержат рекомендации по выполнению практических заданий по дисциплине "Сигнальные процессоры". Рассмотрено решение типовых задач обработки сигналов на платформе микроконтроллеров dsPIC.

Методические указания предназначены для обучающихся по образовательной программе высшего образования по направлению подготовки 11.03.04 Электроника и наноэлектроника.

УДК 004.4 : 681.3.068
ББК 32.973.26

Содержание

Введение 4
1 Выполнение типовых вычислений 6
2 Вычисление взаимной корреляционной функции 11
3 Алгоритм КИХ-фильтра 21
4 Алгоритм БИХ-фильтра 28
5 Алгоритм БПФ 35
6 Цифровой синтез сигналов 40
7 Исследование КИХ-фильтра 47
Список использованных источников 53

Введение

Практикум реализован на платформе микроконтроллеров (МК) семейства dsPIC. Процессорное ядро dsPIC построено по модифицированной гарвардской архитектуре. На рынке интегральных схем их позиционируют как 16-разрядные МК с поддержкой цифровой обработки сигналов (ЦОС). Кристаллы dsPIC содержат универсальное микроконтроллерное ядро (MCU) и ядро ЦОС (DSP). Система команд dsPIC состоит из основного набора инструкций, которые выполняются универсальным ядром MCU и команд ЦОС, выполняемых ядром DSP.

Кроме этого в процессорном ядре dsPIC реализованы:

- аппаратная поддержка циклов типа DO и REPEAT, выполнение которых не требует дополнительных затрат памяти программ и времени на анализ условий окончания;

- циклическая модульная и бит-реверсная адресация памяти данных;

- возможность отображения части программной памяти в нереализованную на кристалле область памяти данных (механизм PSV).

При выполнении инструкций ядра DSP доступна параллельная выборка двух операндов из двух областей памяти данных за счет использования двух адресных генераторов.

МК семейства dsPIC имеют стандартный набор периферийных модулей (параллельные и последовательные порты, компараторы, аналого-цифровые преобразователи (АЦП), часы реального времени) и эффективно реализованные каналы прямого доступа к памяти (DMA). Различные подсемейства dsPIC также содержат специфические периферийные модули, предназначенные для решения различных задач.

Для управления двигателями и преобразователями энергии предназначены:

- широтно-импульсный модулятор (ШИМ) для управления приводом (motor control PWM);

- интерфейс квадратурного энкодера.

Для построения импульсных источников питания (SMPS) предназначены:

- специализированный модуль ШИМ с высоким разрешением (SMPS PWM);

- специализированный АЦП (SMPS ADC).

Для обработки звуковых сигналов:

- 12-разрядный АЦП;

- 16-разрядный цифро-аналоговый преобразователь (ЦАП) или (в других моделях) специализированный модуль ШИМ (output compare PWM);

- интерфейс кодирования данных DCI (I2S, AC97).

В лабораторном практикуме рассматривается реализация на платформе dsPIC типовых задач ЦОС:

- вычисление суммы произведений;

- корреляционный анализ;

- фильтрация на основе алгоритмов фильтров с конечной импульсной характеристикой (КИХ) и бесконечной импульсной характеристикой (БИХ);

- спектральный анализ на основе алгоритма быстрого преобразования Фурье (БПФ).

Все приведенные примеры разработаны и протестированы для цифрового сигнального процессора dsPIC33FJ256GP506 в интегрированной среде разработки MPLAB IDE версии 8.21 с использованием для отладки программ средств симулятора/отладчика MPLAB SIM и отладочного стенда Starter Kit for dsPIC DSC.

Основные модули программ и драйверы периферийных модулей написаны на языке C30 (диалект ANSI C), функции, реализующие алгоритмы ЦОС, на языке Ассемблера dsPIC ASM30.

Приведенные примеры предназначены для использования их в качестве прототипов для выполнения вариантов заданий на основе технологии обратного инжиниринга.

Практикум предназначен для изучения дисциплины «Сигнальные процессоры» для студентов, обучающихся по направлениям подготовки 220400 «Электроника и наноэлектроника», 11.03.03 «Конструирование и технология электронных средств».

1 Выполнение типовых вычислений

1.1 Описание задания

1.1.1 В работе рассматривается задача вычисления суммы произведений

$$s = \sum_{i=0}^{N-1} x(i) \cdot y(i),$$

где *x(i), y(i)* – элементы целочисленных массивов *x[0 .. N-1], y[0..N-1]*;

N – длина массивов.

В задачах ЦОС элементами этих массивов могут быть дискретные отсчеты сигналов и постоянные коэффициенты.

Цель лабораторной работы: получить оценки производительности микроконтроллера серии dsPIC33 на примере вычисления значения *s* для значений параметров из таблицы 1.1 при использовании только средств основного процессорного ядра и при использовании средств ядра DSP. Предполагается, что в процессе вычислений не будет происходить переполнение разрядной сетки для формата 32 разряда.

Таблица 1.1

Вариант	1	2	3	4	5	6
N	16	32	8	32	8	16
Элементы X	0 .. 15	1 .. 32	-4 .. 3	-16 .. 15	2 .. 16	-8 .. 7
Элементы Y	15 .. 0	32 .. 1	3 .. -4	15 .. -16	16 .. 2	7 .. -8

1.1.2 Программа МК имеет модульную структуру. В главном модуле программы *main.c* определяется структура данных, выполняется их инициализация. Вычислительный алгоритм реализован в модулях:

- *basic.s* – при использовании только средств основного процессорного ядра;

- *mac.s* - при использовании средств ядра DSP.

С целью получения эффективного кода вычислительных модулей, кодирование выполнено на языке Ассемблера dsPIC ASM30.

1.1.3 Исходный текст главного файла программы содержит директивы препроцессора для включения заголовочного файла и определения константы.

```
#include <p33Fxxxx.h>
#define size 16                                  // длина массивов N
```

Далее приводится описание структуры данных.

```
int i;                                           // индекс элемента массива
signed int _XBSS(64) array1[size];               // массив в области X
signed int _YBSS(64) array2[size];               // массив в области Y
signed long res;                                 // результат вычислений
```

Массивы *array1* и *array2* будут размещены в областях памяти данных X, Y, где доступно (только для ядра DSP) использование одновременной выборки. Затем приводятся прототипы внешних функций вычисления *s mac* и *basic*.

```
extern signed long mac ( signed int *, signed int *, unsigned int );
extern signed long basic ( signed int *, signed int *, unsigned int );
```

Основной модуль программы содержит операторы инициализации входных массивов *array1* и *array2*, регистра конфигурации ядра CORCON и вызова функции *mac* или *basic* (вызов одной из функций необходимо закомментировать). Указатели массивов *array1* (в области памяти данных X) и *array2* (области памяти данных Y) будут находиться в регистрах W0 и W1. Декрементированное значение длины массивов – в регистре W2. Полученный результат передается в основной модуль программы через регистровую пару (W1,W0). Далее выполняется пустой бесконечный цикл.

```
int main ( void )
{
    for (i=0; i<size; i++) {                     // инициализация array1 and array2
       array1[i] = i-8;
       array2[i] = 7-i;
    }
    CORCON = 0x00F1;                 // режим целочисленных вычислений со знаком,
                                                 // насыщение аккумулятора.
    res = basic (array1, array2, size-1);        // вызов функции
    //res = mac (array1, array2, size-1);        // вызов функции
```

```
    while (1) { }                                   // бесконечный цикл
}
```

1.1.4 Исходный модуль функции *basic* сохранен в файле *basic.s*. Модуль содержит директивы начала кода и объявления глобальной метки (имя функции). Далее следуют операторы инициализации указателей массивов W8, W10 и очистки аккумулятора в регистровой паре (W1,W0).

```
     .text
    .global _basic
_basic:

;  W0 - указатель &array1;  W1 - указатель &array2;  W2 - (size - 1)
    mov     W0,  W8             ;инициализация X pointer
    mov     W1,  W10            ;инициализация Y pointer
    clr     W0                  ;очистка аккумулятора
    clr     W1
```

Далее выполняется цикл, в котором происходит извлечение элементов массивов, выполняется их умножение. Каждое произведение из регистровой пары (W13,W12) суммируется с содержимым аккумулятора (W1,W0).

```
    DO      W2, loop
    mov     [W8++], W4
    mul.ss  W4, [W10++], W12    ; (W13,W12)=W4*[W10]
    add     W12, W0, W0
loop: addc  W13, W0, W1
    return                      ; (W1,W0) = sum(array1*array2)
.end
```

1.1.5 Исходный модуль функции *mac* сохранен в файле *mac.s*. Модуль содержит директивы начала кода и объявления глобальной метки (имя функции). Далее следуют операторы инициализации указателей массивов W8, W10 и очистки 40-разрядного аккумулятора сопроцессора DSP Acc1. Одновременно выполняется загрузка рабочих регистров W4 и W5 первой парой сомножителей и пост инкремент указателей W8 и W10.

```
    .text
```

```
        .global _mac
_mac:

;  W0 - указатель &array1;  W1 - указатель &array2;  W2 - (size - 1)
        mov         W0,  W8                         ;инициализация X pointer
        mov         W1,  W10                        ;инициализация Y pointer
        clr         A, [W8]+=2, W4, [W10]+=2, W5    ;очистка аккумулятора.
```

Затем *N* раз (определяется инкрементированным содержимым W2) выполняется базовая операция DSP ядра mac. При выполнении этой операции вычисляется произведение сомножителей из регистров W4 и W5, суммируется с содержимым Acc1. Одновременно выполняется загрузка рабочих регистров W4 и W5 следующей парой сомножителей и пост инкремент указателей W8 и W10.

```
        repeat      W2
        mac         W4*W5, A, [W8]+=2, W4, [W10]+=2, W5     ;Acc1=sum(arr1*arr2).
```

Младшие 32 разряда Acc1 копируются в регистры (W1,W0) для передачи результата в основной модуль программы.

```
        sac         A, #0, W1           ; (W1,W0) = sum(array1*array2)
        sftac       A, #-16             ; сдвиг результата влево.
        sac         A, #0, W0
        return
.end
```

1.2 Создание проекта и трансляция программы

1.2.1 В своей рабочей папке создайте папку *lr1*. Скопируйте в нее подготовленные файлы с исходным текстом программы *main.c*, *basic.s*, *mac.s*. Создайте новый проект, для этого выберите команду *Project* > *Project Wizard* и последовательно выполните действия:

- задайте тип микроконтроллера *dsPIC33FJ256GP506*;

- задайте транслятор *Microchip C30 Toolsuite*;

- задайте имя проекта lr1 и папку для нового проекта;

- включите в проект файлы с исходным текстом программы *main.c*, *basic.s*, *mac.c*.

В окне менеджера проекта нажмите правой клавишей по полю с надписью *Linker Scripts*. Выберите *Add Files…*. Подключите файл *p33FJ256GP506.gld*, расположенный в каталоге *C:\MicrochipStarter Kits\dsPIC Starter Kit – 1\MPLAB C30 Suite\Support\dsPIC33F\gld*.

Нажмите правой клавишей по полю с надписью *Header Files*. Выберите *Add Files…*. Подключите файл p33FJ256GP506.inc, расположенный в каталоге *C:\MicrochipStarter Kits \dsPIC Starter Kit – 1\MPLAB C30 Suite\Support\ dsPIC33F\inc*.

Откройте окно с исходным текстом программы, для этого в окне менеджера проекта *lr1.mcw* дважды щелкните по ярлыку *main.c* в папке Source Files или выполните команду *File>Open*.

1.2.2 Выполните трансляцию программы (команда *Project>Make*). На закладке *Build* в окне *Output* будет приведена командная строка, посланная компилятору, сообщения об ошибках или успешном завершении трансляции. Если программа содержала синтаксические и другие формальные ошибки, в окне будет выведен список сообщений об ошибках (ERROR). Если дважды щелкнуть мышкой на сообщении об ошибке, то курсор автоматически перейдет к соответствующему оператору в исходном тексте. Если в программе нет подобных ошибок, компилятор выдаст сообщение BUILD SUCCEEDED.

1.3 Выполнение эксперимента

1.3.1 В главном меню MPLAB IDE выберите команду *Debugger>Simulator Settings*. На закладке *Osc / Trace* задайте значение тактовой частоты процессора 80 МГц. В окно наблюдения включите массивы *array1* и *array2*, глобальные переменные и используемые рабочие регистры.

1.3.2 Используя пошаговый режим и автоматический прогон до точки останова, отследите выполнение программы. Используя средства отладчика и *Stopwatch*, определите время выполнения функции *basic*. Оцените корректность выполнения функции.

1.3.3 В тексте основного модуля программы *main.c* переместите символ комментария с вызова функции *mac* на вызов функции *basic*. Выполните трансляцию программы (команда *Project>Make*). Убедитесь в ее успешном выполнении.

1.3.4 Используя пошаговый режим и автоматический прогон до точки останова, отследите выполнение программы. Используя средства отладчика и *Stopwatch*, определите время выполнения функции *mac*.

1.4 Содержание отчета

Отчет должен содержать:

- описание решения задачи с привлечением формул и схем алгоритмов;
- текст отлаженной программы с комментариями;
- результаты вычислений, оценки времени выполнения функций;
- выводы по работе.

1.5 Контрольные вопросы и задания

Поясните содержание задачи, выполняемой микроконтроллером.

Поясните работу функций *mac* и *basic*.

Объясните причину, по которой время выполнения этих функций различно.

Найдите в текстах модулей *mac.s* и *basic.s* команды с различными видами адресации.

Оцените эффективность команд ядра DSP.

2 Вычисление взаимной корреляционной функции

2.1 Описание задания

2.1.1 К типовым задачам ЦОС относятся задачи корреляционного анализа сигналов. В работе исследуется задача вычисления значений взаимной корреляционной функции (ВКФ)

$$r(j) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)y(n+j) = \frac{1}{N}\sum_{n=0}^{N-1} x(n-j)y(n),$$

где *x(n - j)* –дискретный отсчет задержанного сигнала *x(t - τ)*;

y(n) - дискретный отсчет сигнала *y(t)*;

N – количество принимаемых во внимание отсчетов последовательностей *x*, *y*. Для задач ЦОС реального времени необходимо вычислять последовательность значений *r(0) .. r(N)* за время поступления *N* отсчетов входных сигналов.

Цель лабораторной работы:

– определить параметры входного сигнала *x* по результатам корреляционного анализа;

– получить сравнительные оценки производительности микроконтроллера серии dsPIC33 на примере вычисления последовательности значений *r(0) .. r(N)*, для *N = 32* при использовании только средств основного процессорного ядра и при использовании средств ядра DSP.

В решаемой задаче *y* – опорный гармонический сигнал, *x* – зашумленный гармонический сигнал, сдвинутый относительно опорного на фазовый угол *φ* Значения *φ* приведены в таблице 2.1.

Таблица 2.1

Вариант	1	2	3	4	5	6
φ, град.	45	90	135	180	225	270

2.1.2 Программа микроконтроллера имеет модульную структуру. В главном модуле программы *lr2.c* определяется структура данных, выполняется их инициализация. Вычислительный алгоритм реализован в модулях:

- *basic.s* – при использовании только средств основного процессорного ядра;

- *modulo.s* - при использовании средств ядра DSP.

С целью получения эффективного кода вычислительных модулей, кодирование выполнено на языке Ассемблера dsPIC ASM30.

2.1.3 Исходный текст главного файла программы *main.c* содержит директивы препроцессора для включения заголовочных файлов.

```
#include <p33Fxxxx.h>
#include <stdio.h>
#include "sgn_noise.h"
#include "reference.h"
```

и определения константы

```
#define LEN 32                          // длина буфера
```

Далее приводится описание структуры данных.

```
signed int _XBSS(64) array1[LEN];       //массивы входных данных
signed int _YBSS(64) array2[LEN];
signed int array3[LEN];                 //массив выходных данных
unsigned int array_index;               //указатель массива.
```

Массивы *array1* и *array2* будут размещены в областях памяти данных X, Y, где доступно (только для ядра DSP) использование модульной адресации и одновременной выборки. Далее приводятся прототипы внешних функций вычисления ВКФ *modulo* и *basic*.

```
extern void modulo( signed int *, signed int *, signed int *, unsigned int );
extern void basic ( signed int *, signed int *, signed int *, unsigned int );
```

Основной модуль программы содержит операторы инициализации адресных генераторов для режима модульной адресации и основной цикл программы, в котором происходит инициализация входных массивов выборками из файлов *sgn_noise.h* и *reference.h* и вызов функции *modulo* или *basic* (вызов одной из функций необходимо закомментировать). Указатели массивов *array1* (в области памяти данных X) и *array2* (области памяти данных Y) будут находиться в регистрах W0 и W1.

```
int main ( void )
{
        CORCON |= 0x0001;           /* разрешить целочисленную арифметику */
        XMODSRT = (unsigned int) array1;
```

```
        XMODEND = (unsigned int) array1 + 2*LEN - 1;
        YMODSRT = (unsigned int) array2;
        YMODEND = (unsigned int) array2 + 2*LEN - 1;

        while (1)          /* бесконечный циклr */
        {
           for (array_index = 0; array_index < LEN; array_index++) {
                          array2[array_index] = reference[array_index];
                          array1[array_index] = sgn_noise[array_index];
            }

      //       modulo( array1, array2, array3, LEN-1);
               basic( array1, array2, array3, 2*LEN-1);
         }
}
```

2.1.4 Исходный модуль функции *basic* сохранен в файле *basic.s*. Модуль содержит директивы начала кода и объявления глобальной метки (имя функции). Далее следуют операторы сохранения контекста, инициализации указателей массивов W8, W10 и очистки аккумулятора в регистровой паре (W1,W0).

```
        .text
        .global _basic
_basic:
        push    W8              ; сохранение контекста
        push    W9
        push    W10
        push    W11
        push    W12
        push    W13
;W0 - указатель array1;  W1 - указатель array2;  W2 - указатель array3;
;W3 -  (2*LEN - 1)
        mov     W3,  W9         ; копирование 2*LEN-1
        lsr     W9,  W9         ; LEN-1
        mov     W0,  W11        ; копирование указателя array1
```

Далее выполняется внешний цикл. При каждом проходе определяется очередное значение ВКФ *r(n)* и сохраняется в элементе массива *array3[n]*. Выполняется смещение указателя *array1* для следующего цикла.

```
        DO      W9,  array_loop ; выполнить LEN раз
        mov     W1,  W10        ; инициализация указателя array2
```

```
        mov    W11, W8          ; инициализация указателя array1
        clr    W5               ; Инициализация аккумулятора (W7,W6,W5)
        clr    W6
        clr    W7
 ; Внутренний цикл 1 – вычисление одного значения ВКФ
 ; Внутренний цикл 1 – нормализация результата
        mov    W6, [W2++]               ; сохранить значение ВКФ в array3
array_loop:
        inc2   W11, W11         ; смещение указателя array1
```

После выполнения *LEN* циклов восстанавливается контекст и происходит возврат в основную программу.

```
        pop    W13              ; восстановление контекста
        pop    W12
        pop    W11
        pop    W10
        pop    W9
        pop    W8
        return
.end
```

За один проход первого внутреннего цикла происходит извлечение элементов массивов *array1* и *array2*, определяется очередное значение произведения $x(n - j) \cdot y(n)$. Выполняется контроль выхода указателя W8 за границы массива *array1*. Произведение из регистровой пары (W13,W12) дополняется знаковым расширением (W4) и суммируется с содержимым аккумулятора (W7,W6,W5).

```
; Внутренний цикл 1
        DO     W9, loop                ; выполнить LEN раз
        mov    [W8++], W4
        mul.ss W4, [W10++], W12        ; W12:W13=W4*[W10]
        add    W0,  W3, W4             ; верхняя граница array1
        cpslt  W8,  W4                 ; выход за границу array1
        mov    W0,  W8                 ; указатель на начало array1
        add    W12, W5, W5             ; накопление суммы
        addc   W13, W6, W6
        clr    W4                      ; знаковое расширение для "+"
        btsc   W13, #0xF               ; произведение отрицательное?
        com    W4,  W4                 ; знаковое расширение для "-"
loop:  addc    W7,  W4, W7
```

Во втором внутреннем цикле выполняется нормализация результата.

```
; Внутренний цикл 2
        DO      #3,  shift_loop             ; нормализация результата
        lsr     W7,  W7
shift_loop:
        rrc     W6,  W6
```

2.1.5 Исходный модуль функции *modulo* сохранен в файле *modulo.s*. Модуль содержит директивы начала кода и объявления глобальной метки (имя функции). Далее следуют операторы сохранения контекста, включения модульной адресации, инициализации указателей массивов W8, W10 и рабочего регистра W6.

```
         .text
        .global _modulo
_modulo:
        push    W8                  ; сохранение контекста
        push    W10

        mov     #0xC0A8, W8                     ; включить модульную адресацию
        mov     W8, MODCON
;  W0 - указатель array1;  W1 - указатель array2;  W2 – указатель array3;
;  W3 -  (LEN - 1)
        mov     W0,  W8                         ;инициализация X pointer
        mov     W1,  W10                        ;инициализация Y pointer
        sub     W3,#2, W6                       ; W6 = LEN-3 (для do loop )
```

Далее *LEN* раз выполняется цикл, каждый проход которого начинается с очистки 40-разрядного аккумулятора сопроцессора DSP Acc1. Одновременно выполняется загрузка рабочих регистров W4 и W5 первой парой сомножителей и пост инкремент указателей W8 и W10. Затем в цикле *LEN* раз выполняется базовая операция DSP ядра *mac*. При этом операция *mac* выполняется *LEN-2* раза с загрузкой рабочих регистров W4 и W5 следующей парой сомножителей и пост инкрементом указателей W8 и W10, еще один раз так же, но без пост инкремента указателя W10, последний раз без загрузки рабочих регистров и пост инкремента указателей. Полученное значение ВКФ нормализуется и сохраняется в массиве *array3* с пост инкрементом его указателя.

```
    DO    W3,   array_loop
    clr     A, [W8]+=2, W4, [W10]+=2, W5                ;очистка аккумулятора
    repeat W6
    mac   W4*W5, A, [W8]+=2, W4, [W10]+=2, W5       ;Acc1=Acc1+(W4*W5)
    mac   W4*W5, A, [W8]+=2, W4, [W10], W5
    mac   W4*W5, A
array_loop:
    sac.r   A, #4, [W2++]
```

Однократный пропуск инкремента указателя *W10* массива *array2* обеспечивает его смещение относительно указателя *W8* массива *array1* на следующем проходе цикла. После выполнения *LEN* циклов восстанавливается контекст, выключается режим модульной адресации и происходит возврат в основную программу.

```
    pop    W10
    pop    W9
    pop    W8
    clr    MODCON                          ; выключить модульную адресацию
    return
.end
```

2.2 Создание проекта и трансляция программы

2.2.1 В своей рабочей папке создайте папку *lr2*. Из меню *Пуск/ Все программы* запустите программу dsPICworks. В строке команд программы выберите закладку *Generator* и пункт *Sinusoidal*. В открывшейся панели *Generating Sinusoidal Wave* задайте значения параметров:

Signal Frequency (частота сигнала)	500 Гц
Sampling Rate (частота дискретизации)	16000 Гц
Number of Samples (количество отсчетов)	32
Angular Phase Delay (задержка фазы)	0
Peak Amplitude (from zero) (амплитуда)	1
DC offset (смещение постоянной составляющей)	0
Output File Format (формат выходного файла)	ASCII
Output Number Type (тип значений сигнала)	16 bit Fractional Fixed point

Random Noise Type (Тип шума) no noise.

Нажмите кнопку *File Name*, укажите свою рабочую папку и задайте имя файла *reference.tim*. Нажмите кнопку OK. В результате откроется окно с временной диаграммой сигнала.

Повторите аналогичные действия для сигнала частотой 500 Гц и заданным по варианту задания фазовым сдвигом. Сохраните файл сигнала *sine500Hz.tim* в рабочей папке.

В строке команд программы выберите закладку *Generator* и пункт *Noise Function*. В открывшейся панели *Generating Noise Function* задайте значения параметров:

Sampling Rate (частота дискретизации)	16000 Гц
Number of Samples (количество отсчетов)	32
Gaussian Noise - Mean (среднее)	0
Gaussian Noise – Std Deviat (стандартное отклонение)	1
Output File Format (формат выходного файла)	ASCII
Output Number Type (тип значений сигнала)	16 bit Fractional Fixed point
Random Noise Type	normal.

Нажмите кнопку *File Name*, укажите свою рабочую папку и задайте имя файла *noise.tim*. Нажмите кнопку OK. В результате откроется окно с временной диаграммой шумового сигнала.

В строке команд программы выберите *Operation* пункт *Arithmetic/Linear Combo*. На открывшейся закладке укажите имена файлов для сигналов:

- input x1(n) sine500Hz.tim;
- input x2(n) noise.tim;
- output y(n) sgn_noise.tim.

Задайте значения коэффициентов: a = 0.5; b=0.5; c=0.

Нажмите кнопку OK. В результате откроется окно с временной диаграммой зашумленного сигнала.

В строке команд программы выберите *File* пункт *Export File*. Задайте файл источника *Source File* (*reference.tim*), установите флажок на опции *Create C data*. Выберите формат экспорта файла *Export File Format* как *Fractional/Integer ASCII Hexadecimal*. Результат будет сохранен в файле *reference.h*. Повторите эти действия для сигнала *sgn_noise.tim*. Результат будет сохранен в файле *sgn_noise.h*.

2.2.2 Скопируйте в папку *lr2* подготовленные файлы с исходным текстом программы *lr2.c*, *modulo.s*, *basic.s*.

Создайте новый проект, для этого выберите команду *Project > Project Wizard* и последовательно выполните действия:

- задайте тип микроконтроллера *dsPIC33FJ256GP506*;

- задайте транслятор *Microchip C30 Toolsuite*;

- задайте имя проекта *lr2* и папку для нового проекта;

- включите в проект файлы с исходным текстом программы *lr2.c*, *modulo.s*, *basic.s*.

В окне менеджера проекта нажмите правой клавишей по полю с надписью *Linker Scripts*. Выберите *Add Files*…. Подключите файл *p33FJ256GP506.gld*, расположенный в каталоге *C:\MicrochipStarter Kits \dsPIC Starter Kit – 1\MPLAB C30 \Support\gld*.

Нажмите правой клавишей по полю с надписью *Header Files*. Выберите *Add Files*…. Подключите файл *p33FJ256GP506.inc*, расположенный в каталоге *C:\MicrochipStarter Kits \dsPIC Starter Kit – 1\MPLAB C30 \Support\inc*.

Нажмите правой клавишей по полю с надписью *Header Files*. Выберите *Add Files*…. Подключите файлы *reference.h*, *sgn_noise.h*.

Откройте окно с исходным текстом программы, для этого в окне менеджера проекта *lr2.mcw* дважды щелкните по ярлыку *lr2.c* в папке *Source Files* или выполните команду *File>Open*.

2.2.3 Выполните трансляцию программы (команда *Project>Make*). На закладке *Build* в окне *Output* будет приведена командная строка, посланная компилятору, сообщения об ошибках или успешном завершении трансляции. Если программа содержала синтаксические и другие формальные ошибки, в окне будет выведен

список сообщений об ошибках (ERROR). Если дважды щелкнуть мышкой на сообщении об ошибке, то курсор автоматически перейдет к соответствующему оператору в исходном тексте. Если в программе нет подобных ошибок, компилятор выдаст сообщение BUILD SUCCEEDED.

2.3 Выполнение эксперимента

2.3.1 В главном меню MPLAB IDE выберите команду *Debugger>Simulator Settings*. На закладке *Osc / Trace* задайте значение тактовой частоты процессора 80 МГц.

В главном меню MPLAB IDE выберите команду *Tools > DMCI – Data Monitor Control Interface*. Выберите закладку *Dynamic Data View*. Установите флажки на полях графиков *Graph 1*, *Graph 2* и *Graph 3*. Установите курсор на поле *Graph 1*. Нажмите правую кнопку и выберите команду *Configure Data Source*. На закладке *Graph 1* в поле *Data Sources* выберите массив *Array1*, задайте *Display Format – Fractional*. Нажмите *OK*. На закладке *Graph 2* в поле *Data Sources* выберите массив *Array2*, задайте *Display Format – Fractional*. Нажмите *OK*. На закладке *Graph 3* в поле *Data Sources* выберите массив *Array3*, задайте *Display Format – Fractional*. Нажмите *OK*.

2.3.2 Используя средства отладчика и *Stopwatch*, определите время выполнения процедуры *basic*. Используя *DMCI*, оцените корректность выполнения функции.

2.3.3 В тексте основного модуля программы *lr2.c* переместите символ комментария с вызова процедуры *modulo* на вызов процедуры *basic*. Выполните трансляцию программы (команда *Project>Make*). Убедитесь в ее успешном выполнении. Используя средства отладчика и *Stopwatch*, определите время выполнения процедуры *modulo*. Используя *DMCI*, оцените корректность выполнения функции.

2.3.4 По максимуму значения взаимной корреляционной функции определите фазовый сдвиг зашумленного сигнала относительно опорного.

2.4 Содержание отчета

Отчет должен содержать:

- описание решения задачи с привлечением формул и схем алгоритмов;
- текст отлаженной программы с комментариями;
- оценки времени выполнения функций;
- выводы по работе.

2.5 Контрольные вопросы и задания

Поясните содержание задачи, выполняемой микроконтроллером.

Поясните работу функций *modulo* и *basic*.

Объясните причину, по которой время выполнения этих функций различно?

Поясните принцип работы циклического буфера.

Как реализуется модульная адресация?

3 Алгоритм КИХ-фильтра

3.1 Описание задания

3.1.1 В работе исследуется реализация на платформе dsPIC алгоритма КИХ-фильтра

$$y(n) = \frac{1}{N}\sum_{i=0}^{N-1} a(i)x(n-i),$$

где *y(n)* – текущий дискретный отсчет выходного сигнала *y(t)*;

x(n - i) –дискретный отсчет задержанного на *i* тактов входного сигнала *x(t)*;

a(0) .. a(N-1) – массив весовых коэффициентов.

Исследование выполняется в среде MPLAB IDE. Для нечетных вариантов заданий весовые коэффициенты загружаются в область памяти данных X, для четных вариантов заданий – используется механизм прямого отображения памяти программ PSV. Варианты заданий приведены в таблице 3.1. Для всех вариантов частота дискретизации 8000 Гц.

Таблица 3.1

Вариант	Тип фильтра	fc1, Гц	fc2, Гц	fc3, Гц	fc4, Гц	Нз, дБ	ΔН, дБ
1	ФНЧ	800	1600			-53	0.2
2	ФНЧ	1000	2000			-65	0.1
3	ФВЧ	800	1600			-53	0.2
4	ФВЧ	1000	1500			-42	1
5	ПФ	500	1000	2500	3000	-42	1
6	ПФ	500	1500	2500	3500	-65	0.1

3.1.2 Программа микроконтроллера имеет модульную структуру. Исходный текст главного файла программы *main.c* содержит директивы препроцессора для включения заголовочных файлов, объявление внешней переменной *Filter* (для нечетных вариантов) или *psvFilter* (для четных вариантов) типа *FIRStruct* и объявление выходного массива *output*.

```
#include <p33Fxxxx.h>                    // директивы препроцессора
#include "FIR_Filter.h"
#include "composite.h"

extern FIRStruct Filter;                 // объявление внешней структуры
//extern FIRStruct psvFilter;
int output[64];                          // объявление выходного массива
```

Объявления структуры *FIRStruct* и прототипов функций инициализации структуры *FIRDelayInit* и КИХ-фильтра *FIR* приведено в заголовочном файле *FIR_Filter.h*.

3.1.3 Далее основной модуль программы содержит операторы вызова функции инициализации структуры и функции КИХ-фильтра.

```
int main(void)                                   // основной модуль программы
{
   FIRDelayInit(&<имя структуры >);   // вызов функции инициализации структуры
   Nop();                                        // для размещения точки останова
   FIR(1024, output, composite, &<имя структуры>);          // вызов функции FIR
   Nop();                                        // для размещения точки останова

   while(1);                                     // бесконечный цикл
}
```

Имя структуры следует указать в соответствии с вариантом задания (*Filter* или *psvFilter*). Функции *Nop* использованы для установки (при необходимости) точки останова.

В модулях *Pass.s* (для нечетных вариантов) и *Pass_psv.s* (для четных вариантов) задается распределение памяти и определяются значения констант:

- *NumTaps* – число весовых коэффициентов;

- *Taps* - массив весовых коэффициентов.

Функции *FIRDelayInit* и *FIR* реализованы в модуле *FIR_filter.s*. С целью получения эффективного кода вычислительных модулей, кодирование выполнено на языке Ассемблера dsPIC.

3.2 Создание проекта и трансляция программы

3.2.1 В своей рабочей папке создайте папку *lr3*. Из меню *Пуск/ Все программы* запустите программу dsPICworks для формирования массива выборок входного сигнала. Для исследования КИХ-фильтра на его вход будет подаваться двухчастотный гармонический сигнал. Частоты гармоник следует задавать таким образом, чтобы одна находилась в полосе пропускания, а вторая в полосе подавления фильтра. В строке команд программы выберите закладку *Generator* и пункт *Sinusoidal*. В открывшейся панели *Generating Sinusoidal Wave* задайте значения параметров первой гармонике по примеру:

Signal Frequency (частота сигнала)	500 Гц
Sampling Rate (частота дискретизации)	8000 Гц
Number of Samples (количество отсчетов)	64
Angular Phase Delay (задержка фазы)	0
Peak Amplitude (from zero) (амплитуда)	1
DC offset (смещение постоянной составляющей)	0
Output File Format (формат выходного файла)	ASCII
Output Number Type (тип значений сигнала)	16 bit Fractional Fixed point
Random Noise Type (Тип шума)	no noise.

Нажмите кнопку *File Name*, укажите свою рабочую папку и задайте имя файла, например, *sine500Hz.tim*. Нажмите кнопку *OK*. В результате откроется окно с временной диаграммой сигнала.

Повторите аналогичные действия для второй компоненты сигнала, например, частотой 2000 Гц. Сохраните файл сигнала, например, *sine2000Hz.tim* в рабочей папке.

В строке команд программы выберите *Operation* пункт *Arithmetic/Linear Combo*. На открывшейся закладке укажите имена файлов для сигналов:

- input x1(n) sine500Hz.tim;
- input x2(n) sine2000Hz.tim;
- output y(n) composite.tim.

Задайте значения коэффициентов: a = 0.5; b=0.5; c=0. Нажмите кнопку *OK*. В результате откроется окно с временной диаграммой двухчастотного сигнала.

В строке команд программы выберите *File* пункт *Export File*. Задайте файл источника *Source File*, установите флажок на опции *Create C data*. Выберите формат экспорта файла *Export File Format* как *Fractional/Integer ASCII Hexadecimal*. Результат будет сохранен в файле *composite.h*.

3.2.2 Для проектирования цифровых фильтров можно использовать программу DSPLinks. Рассмотрим построение КИХ-фильтра на примере фильтра нижних частот (ФНЧ).

Из меню *Пуск / Все программы* запустите программу DSPLinks. В строке пиктограмм программы выберите пиктограмму *Parts*. Установите курсор на поле рабочего окна. При нажатии на левую кнопку мышки открывается окно *Part Select*, в котором необходимо выбрать нужный тип функционального блока – *RMZLPF* (КИХ ФНЧ, синтезированный по алгоритму Ремеза). При нажатии на кнопку *OK* нужный блок размещается на рабочем поле. Наведите на него курсор и нажмите правую кнопку мышки, в открывшемся меню выберите команду *Attribute*. В окне *Part Attribute* нажмите кнопку *Change Parameters*. Откроется окно *FIR type LPF* (*Remez Algorithm*), в котором можно задать требуемые параметры фильтра:

- Sampling Frequency Fs (частота дискретизации);

- Cutoff Frequency 1 Fc1 (граничная частота полосы пропускания);

- Cutoff Frequency 2 Fc2 (граничная частота полосы подавления);

- Tap Count N (порядок фильтра);

- Attenuation at (подавление);

- Ripple Factor rp (неравномерность в полосе пропускания).

Нажатие на кнопку *Quantize Coefficients* позволяет задать требуемое число разрядов (15) и формат вычисляемых коэффициентов фильтра. Удобнее использовать десятичный формат (*Dec*). Окна закрываются нажатием кнопки *OK*.

В строке пиктограмм программы выберите пиктограмму *Monitor*, откроется окно, в котором будут представлены введенные параметры, а также частотные характеристики фильтра. В меню *Tools* выберите команду *Save Coefficients*, сохраните коэффициенты в файле, выбрав рабочую папку проекта и имя файла, например, *lpf*.

Для построения фильтров верхних частот (ФВЧ) рекомендуется использовать функциональный блок *RMZHPF*, для построения полосового фильтра (ПФ) - функциональный блок *RMZBPF*.

3.2.3 Скопируйте в папку *lr3* подготовленные файлы с исходным текстом программы *main.c*, *Pass.s* (или *Pass_psv*), *FIR_filter.s* и *FIR_filter.h*. Создайте новый проект, для этого выберите команду *Project > Project Wizard* и последовательно выполните действия:

- задайте тип микроконтроллера *dsPIC33FJ256GP506*;

- задайте транслятор *Microchip C30 Toolsuite*;

- задайте имя проекта *lr3* и папку для нового проекта;

- включите в проект файлы с исходным текстом программы *main.c*, *Pass.s* (или *Pass_psv*), *FIR_filter.s*.

В окне менеджера проекта нажмите правой клавишей по полю с надписью *Linker Scripts*. Выберите *Add Files….* Подключите файл *p33FJ256GP506.gld*, расположенный в каталоге *C:\MicrochipStarter Kits \dsPIC Starter Kit – 1\MPLAB C30 \Support\gld*.

Нажмите правой клавишей по полю с надписью *Header Files*. Выберите *Add Files*…. Подключите файл *p33FJ256GP506.inc*, расположенный в каталоге *C:\MicrochipStarter Kits \dsPIC Starter Kit – 1\MPLAB C30 \Support\inc*. Подключите файл *FIR_filter.h*.

Нажмите правой клавишей по полю с надписью *Header Files*. Выберите *Add Files*…. Подключите файл *composite.h* .

Используя программу Блокнот, откройте файл *lpf*. Откройте окно с исходным текстом подпрограммы *Pass* (или *Pass_psv*), для этого в окне менеджера проекта *lr3.mcw* дважды щелкните по ярлыку *Pass.s* (или *Pass_psv.s*) в папке *Source Files* или выполните команду *File>Open*.

Найдите в тексте подпрограммы директиву, которой задается значение параметра *NumTaps* (количество коэффициентов), измените значение этого параметра на требуемое (*Tap Count* из созданного файла *lpf)*. Найдите в тексте подпрограммы метку *Taps*. Замените значения коэффициентов на значения из файла *lpf*. Откройте окно с исходным текстом программы, для этого в окне менеджера проекта *lr3.mcw* дважды щелкните по ярлыку *main.c* в папке *Source Files* или выполните команду *File>Open*.

3.2.4 Выполните трансляцию программы (команда *Project>Make*). На закладке *Build* в окне *Output* будет приведена командная строка, посланная компилятору, сообщения об ошибках или успешном завершении трансляции. Если программа содержала синтаксические и другие формальные ошибки, в окне будет выведен список сообщений об ошибках (ERROR). Если дважды щелкнуть мышкой на сообщении об ошибке, то курсор автоматически перейдет к соответствующему оператору в исходном тексте. Если в программе нет подобных ошибок, компилятор выдаст сообщение BUILD SUCCEEDED.

3.3 Выполнение эксперимента

3.3.1 В главном меню MPLAB IDE выберите команду *Debugger>Simulator Settings*. На закладке *Osc / Trace* задайте значение тактовой частоты 80 МГц.

В главном меню MPLAB IDE выберите команду *Tools* > *DMCI – Data Monitor Control Interface*. Выберите закладку *Dynamic Data View*. Установите флажки на полях графиков *Graph 1* и *Graph 2*. Установите курсор на поле *Graph 1*. Нажмите правую кнопку и выберите команду *Configure Data Source*. На закладке *Graph 1* в поле *Data Sources* выберите массив *Composite*, задайте *Display Format – Fractional*. Нажмите *OK*. На закладке *Graph 2* в поле *Data Sources* выберите массив *Output*, задайте *Display Format – Fractional*. Нажмите *OK*.

3.3.2 Установите точки останова на функциях *Nop*. Выполните программу в режиме прогона до точки останова. Определите время выполнения функции *FIR*. Определите содержимое рабочих массивов программы.

Используя *DMCI*, оцените корректность выполнения процедуры фильтрации входного сигнала.

3.4 Содержание отчета

Отчет должен содержать:

- описание решения задачи с привлечением формул и схем алгоритмов;
- текст программы с комментариями;
- временные диаграммы входного и выходного сигналов;
- выводы по работе.

3.5 Контрольные вопросы и задания

Поясните содержание задачи, выполняемой микроконтроллером.

Поясните принцип работы КИХ-фильтра.

Поясните организацию структуры данных программы.

Каким образом были рассчитаны весовые коэффициенты?

Поясните работу процедуры *FIR*.

4 Алгоритм БИХ-фильтра

4.1 Описание задания

4.1.1 В работе исследуется реализация на платформе dsPIC алгоритма БИХ-фильтра

$$y(n) = \sum_{i=0}^{N} a_i x(n-i) - \sum_{j=1}^{M} b_j x(n-j),$$

где *y(n)* – текущий дискретный отсчет выходного сигнала *y(t)*;

x(n - i) –дискретный отсчет задержанного на *i* тактов входного сигнала *x(t)*;

a0 .. aN и *b1 .. bM* – массивы весовых коэффициентов.

Известно, что при *M > 2* реализованные подобным образом БИХ-фильтры потенциально неустойчивы. Поэтому БИХ-фильтры высокого порядка строят путем каскадного включения звеньев второго порядка вида (рисунок 4.1).

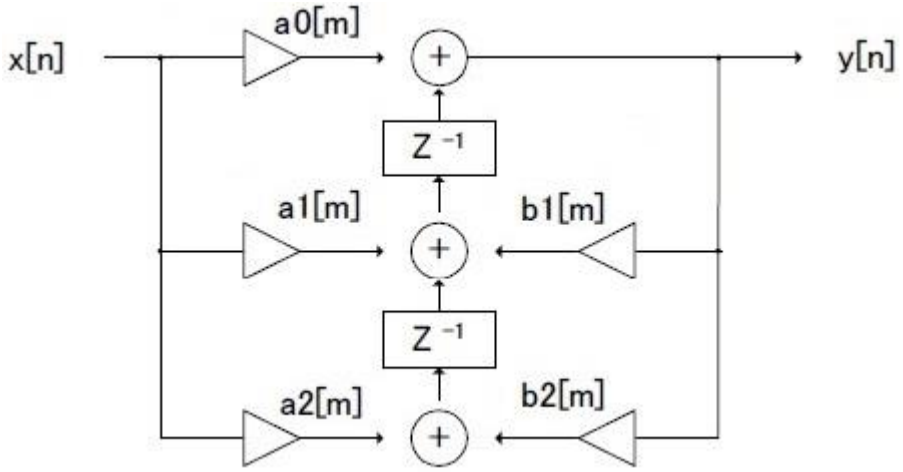

Рисунок 4.1

На рисунке 4.1 символами *a0[m] .. a2[m*], *b1[m] .. b2[m]* обозначены коэффициенты звена с номером *m*.

Исследование выполняется в среде MPLAB IDE. Для нечетных вариантов заданий весовые коэффициенты загружаются в область памяти данных X, для четных вариантов заданий – используется механизм прямого отображения памяти программ PSV. Варианты заданий приведены в таблице 4.1. Для всех вариантов частота дискретизации 8000 Гц.

Таблица 4.1

Вариант	Тип фильтра	fc1, Гц	fc2, Гц	Порядок фильтра	ΔH, дБ
1	ФНЧ	1000	-	6	0.5
2	ФНЧ	1500	-	6	1
3	ФВЧ	800	-	6	1
4	ФВЧ	1000	-	6	0.5
5	ПФ	1000	2000	6	0.5
6	ПФ	500	1500	6	1

4.1.2 Программа микроконтроллера имеет модульную структуру. Исходный текст главного файла программы *main.c* содержит директивы препроцессора для включения заголовочных файлов, объявление внешней переменной *Filter* (для четных вариантов) или *psvFilte*r (для нечетных вариантов) типа *FIRStruct* и объявление выходного массива *output*.

```
#include <p33Fxxxx.h>                    // директивы препроцессора
#include "IIR_Filter.h"
#include "composite.h"

extern IIRTransposedStruct Filter;       // объявление внешней структуры
//extern IIRTransposedStruct psvFilter;

int output[64];                          // объявление выходного массива
```

Объявления структуры *IIRTransposedStruct* и прототипов функций инициализации структуры *IIRTransposedInit* и БИХ-фильтра *IIRTransposed* приведено в заголовочном файле *IIR_Filter.h*.

4.1.3 Далее основной модуль программы содержит операторы вызова функции инициализации структуры и функции БИХ-фильтра.

```
int main(void)                                  // основной модуль программы
{
   IIRTransposedInit(&<имя структуры >);        //вызов функции инициализации
                                                // структуры
   Nop();                                       // для размещения точки останова
   IIRTransposed(64, output, composite, &<имя структуры>); // вызов функции
   Nop();                                       // для размещения точки останова
```

```
  while(1);                                    // бесконечный цикл
}
```

Имя структуры следует указать в соответствии с вариантом задания (*Filter* или *psvFilter*). Функции *Nop* использованы для установки (при необходимости) точки останова.

В модулях *Pass.s* (для нечетных вариантов) и *Pass_psv.s* (для четных вариантов) задается распределение памяти и определяются значения констант:

- *NumSections* – количество звеньев второго порядка;

- *Coefs* - массив весовых коэффициентов.

Функции *IIRTransposedInit* и *IIRTransposed* реализованы в модуле *IIRT_filter.s*. С целью получения эффективного кода вычислительных модулей, кодирование выполнено на языке Ассемблера dsPIC.

4.2 Создание проекта и трансляция программы

4.2.1 В своей рабочей папке создайте папку *lr4*. Подготовьте и скопируйте в эту папку файл двухчастотного сигнала *composite.h*.

4.2.2 Для проектирования цифровых фильтров можно использовать программу DSPLinks. Рассмотрим построение БИХ-фильтра на примере ФВЧ.

Из меню *Пуск / Все программы* запустите программу DSPLinks. В строке пиктограмм программы выберите пиктограмму *Parts*. Установите курсор на поле рабочего окна. При нажатии на левую кнопку мышки открывается окно *Part Select*, в котором необходимо выбрать нужный тип функционального блока – *CHBHPF* (БИХ ФВЧ, синтезированный по аппроксимации Чебышева). При нажатии на кнопку *OK* нужный блок размещается на рабочем поле. Наведите на него курсор и нажмите правую кнопку мышки, в открывшемся меню выберите команду *Attribute*. В окне *Part Attribute* нажмите кнопку *Change Parameters*. Откроется окно *Chebychev HPF Parameters*, в котором можно задать требуемые параметры фильтра:

- Sampling Frequency Fs (частота дискретизации);

- Cutoff Frequency Fc (граничная частота полосы пропускания);

- Pass Band Ripple Ac (неравномерность в полосе пропускания)

- Order (порядок – число звеньев).

Нажатие на кнопку *Quantize Coefficients* позволяет задать требуемое число разрядов (15) и формат вычисляемых коэффициентов фильтра. Удобнее использовать десятичный формат (*Dec*). Окна закрываются нажатием кнопки *OK*.

В строке пиктограмм программы выберите пиктограмму *Monitor*, откроется окно, в котором будут представлены введенные параметры, а также частотные характеристики фильтра. В меню *Tools* выберите команду *Save Coefficients*, сохраните коэффициенты в файле *hpf*, выбрав рабочую папку проекта и имя файла.

Для построения ФНЧ рекомендуется использовать функциональный блок *CHBLPF*, для построения полосового фильтра - функциональный блок *CHBBPF*.

4.1.3 Скопируйте в папку *lr4* подготовленные файлы с исходным текстом программы *main.c*, *Pass.s* (или *Pass_psv*), *IIRT_filter.s* и *IIR_filter.h*. Создайте новый проект, для этого выберите команду *Project > Project Wizard* и последовательно выполните действия:

- задайте тип микроконтроллера *dsPIC33FJ256GP506*;

- задайте транслятор *Microchip C30 Toolsuite*;

- задайте имя проекта *lr4* и папку для нового проекта;

- включите в проект файлы с исходным текстом программы *main.c*, *Pass.s* (или *Pass_psv*), *IIRT_filter.s.*

В окне менеджера проекта нажмите правой клавишей по полю с надписью *Linker Scripts*. Выберите *Add Files*…. Подключите файл *p33FJ256GP506.gld*, расположенный в каталоге *C:\MicrochipStarter Kits \dsPIC Starter Kit – 1\MPLAB C30 \Support\gld*.

Нажмите правой клавишей по полю с надписью *Header Files*. Выберите *Add Files*…. Подключите файл *p33FJ256GP506.inc*, расположенный в каталоге *C:\MicrochipStarter Kits \dsPIC Starter Kit – 1\MPLAB C30 \Support\inc*. Нажмите правой клавишей по полю с надписью *Header Files*. Выберите *Add Files*…. Подключите файлы *IIR_filter.h* и *composite.h.*

Используя программу Блокнот, откройте файл с результатом синтеза фильтра *hpf*. Пример результата синтеза ФВЧ шестого порядка, состоящего из трех звеньев.

```
U_0
Chebychev HPF
Sampling Frequency =    8000.0
cutoff1 =    1000.0
Pass Band Ripple =0.100000
Order = 6th
Quantized by 15 [bits]
**** 1st biquad ****
a0 =        13204
a1 =       -26408
a2 =        13204
b1 =       -22482
b2 =        14153
**** 2nd biquad ****
a0 =        10080
a1 =       -20160
a2 =        10080
b1 =       -15251
b2 =         8839
**** 3rd biquad ****
a0 =         5444
a1 =       -10888
a2 =         5444
b1 =        -3809
b2 =         1667
```

Откройте окно с исходным текстом подпрограммы *Pass*, для этого в окне менеджера проекта *lr4.mcw* дважды щелкните по ярлыку *Pass.s* в папке *Source Files* или выполните команду *File>Open.*

Найдите в тексте подпрограммы директиву, которой задается значение параметра *NumSections* (количество звеньев), измените значение этого параметра на требуемое (значение *Order* из файла *hpf* поделите на 2). Найдите в тексте подпрограммы метку *Coefs*. Замените значения коэффициентов на требуемые из файла *hpf.* При выполнении этих действий следует учесть, что необходимо изменить на противоположные знаки коэффициентов *b1* и *b2*. Пример директив для задания весовых коэффициентов ФВЧ.

```
Coefs:
;**** 1st biquad ****
.word  13204  ; a0
.word -26408  ; a1
.word  22482  ; b1
.word  13204  ; a2
.word -14153  ; b2
;**** 2nd biquad ****
.word  10080  ; a0
.word -20160  ; a1
.word  15251  ; b1
.word  10080  ; a2
.word   -8839  ; b2
;**** 3rd biquad ****
.word    5444  ; a0
.word -10888  ; a1
.word    3809  ; b1
.word    5444  ; a2
.word   -1667  ; b2
```

4.2.4 Откройте окно с исходным текстом программы, для этого в окне менеджера проекта *lr4.mcw* дважды щелкните по ярлыку *main.c* в папке *Source Files* или выполните команду *File>Open*.

Выполните трансляцию программы (команда *Project>Make*). На закладке *Build* в окне *Output* будет приведена командная строка, посланная компилятору, сообщения об ошибках или успешном завершении трансляции. Если программа содержала синтаксические и другие формальные ошибки, в окне будет выведен список сообщений об ошибках (ERROR). Если дважды щелкнуть мышкой на сообщении об ошибке, то курсор автоматически перейдет к соответствующему оператору в исходном тексте. Если в программе нет подобных ошибок, компилятор выдаст сообщение BUILD SUCCEEDED.

4.3 Выполнение эксперимента

В главном меню MPLAB IDE выберите команду *Debugger>Simulator Settings*. На закладке *Osc / Trace* задайте значение тактовой частоты процессора 80 МГц.

В главном меню MPLAB IDE выберите команду *Tools* > *DMCI – Data Monitor Control Interface*. Выберите закладку *Dynamic Data View*. Установите флажки на полях графиков *Graph 1* и *Graph 2*. Установите курсор на поле *Graph 1*. Нажмите правую кнопку и выберите команду *Configure Data Source*. На закладке *Graph 1* в поле *Data Sources* выберите массив *Composite*, задайте *Display Format – Fractional*. Нажмите *OK*. На закладке *Graph 2* в поле *Data Sources* выберите массив *Output*, задайте *Display Format – Fractional*. Нажмите *OK*.

3.3.2 Установите точки останова на функциях Nop. Выполните программу в режиме прогона до точки останова. Определите время выполнения функции *IIRTransposed*. Определите содержимое рабочих массивов программы.

Используя DMCI, оцените корректность выполнения процедуры фильтрации входного сигнала.

4.4 Содержание отчета

Отчет должен содержать:

- описание решения задачи с привлечением формул и схем алгоритмов;
- текст отлаженной программы с комментариями;
- временные диаграммы входного и выходного сигналов;
- выводы по работе.

4.5 Контрольные вопросы и задания

Поясните содержание задачи, выполняемой микроконтроллером.

Поясните принцип работы БИХ-фильтра.

Поясните организацию структуры данных программы.

Поясните работу функций из файла *IIRT_filter.s*.

5 Алгоритм БПФ

5.1 Описание задания

5.1.1 В работе исследуется реализация на платформе dsPIC алгоритма БПФ

$$\dot{X}(k) = \frac{1}{N}\sum_{i=0}^{N} \dot{w}(kn)\dot{x}(n),$$

где $\dot{X}(k)$ – комплексная k-я гармоника частотного спектра;

$\dot{x}(n)$ –дискретный отсчет комплексного сигнала $\dot{x}(t)$;

$\dot{W}(kn) = exp[-(j2\pi k \cdot n)/N]$ – комплексные коэффициенты;

$n = 0 \ldots N-1$ – номер отсчета сигнала;

$k = 0 \ldots N-1$ – номер гармоники частотного спектра;

$N = 2^m$ – число отсчетов.

В лабораторной работе используются функции DSP Library:

- *TwidFactorInit* – формирует первую половину комплексных коэффициентов $\dot{W}(kn)$;

- *FFTComplexIP* – вычисляет комплексные гармоники $\dot{X}(k)$ и сохраняет их в массиве;

- *BitReverseComplex* –выполняет перестановку комплексных гармоник $\dot{X}(k)$ в бит – реверсном порядке;

- *SquareMagnitudeCplx* – вычисляет значения квадратов модулей гармоник

$$M(k)=Re^2(\dot{X}(k))+Im^2(\dot{X}(k)).$$

В программе dsPIC выполняется БПФ входного двухчастотного сигнала и БПФ сигнала, обработанного цифровым фильтром (из лабораторной работы 4).

5.1.2 Программа микроконтроллера имеет модульную структуру. Исходный текст главного файла программы *main.c* содержит директивы препроцессора для включения заголовочных файлов, объявление внешней переменной *Filter* (или *psvFilter*) типа *FIRStruct* и объявления переменных, констант и массивов.

```
#include <p33Fxxxx.h>                    // директивы препроцессора
```

```
#include "IIR_Filter.h"
#include "composite.h"
#include <dsp.h>

extern IIRTransposedStruct Filter;              // объявление внешней структуры
//extern IIRTransposedStruct psvFilter;

int output[64];                                 // объявление выходного массива фильтра
int i;                                          // объявление переменной цикла
                                                // определение параметров БПФ
#define FFT_BLOCK_LENGTH    64                  // длина блока
#define LOG2_BLOCK_LENGTH   6                   // log2 64

// рабочие массивы алгоритма БПФ
fractcomplex fftInputOutput[FFT_BLOCK_LENGTH]
                _YBSS(FFT_BLOCK_LENGTH * 2 * 2);   //входные-выходные данные
fractcomplex twiddleFactors[FFT_BLOCK_LENGTH/2]
                _XBSS(FFT_BLOCK_LENGTH * 2);       //поворотные коэффициенты
fractional preFilterFFTMag[FFT_BLOCK_LENGTH];  //амплитуды до фильтра
fractional postFilterFFTMag[FFT_BLOCK_LENGTH]; // амплитуды после фильтра
```

Объявления структуры *IIRTransposedStruct* и прототипов функций инициализации структуры *IIRTransposedInit* и БИХ-фильтра *IIRTransposed* приведено в заголовочном файле *IIR_Filter.h*.

5.1.3 Далее основной модуль программы содержит операторы вызова функции инициализации структуры и функции БИХ-фильтра:

```
int main(void)                                  // основной модуль программы
{
   IIRTransposedInit(&<имя структуры >);        //вызов функции инициализации
                                                // структуры
   // Инициализация массива коэффициентов
   TwidFactorInit (LOG2_BLOCK_LENGTH, &twiddleFactors[0], 0);
    /*************************************************************
    * БПФ входного сигнала фильтра
    *************************************************************/
   for(i = 0; i < FFT_BLOCK_LENGTH; i ++)           //инициализация входного
   {                                                //массива
      fftInputOutput[i].real = ((fractional)composite[i]) >> 1;
      fftInputOutput[i].imag = 0x0;
   }
```

```
    FFTComplexIP(LOG2_BLOCK_LENGTH,  fftInputOutput, twiddleFactors,
                        COEFFS_IN_DATA);

    BitReverseComplex(LOG2_BLOCK_LENGTH, fftInputOutput);

    SquareMagnitudeCplx(FFT_BLOCK_LENGTH, fftInputOutput, preFilterFFTMag);

    Nop();                                          // для размещения точки останова

    // вызов функции фильтра
    IIRTransposed(512, output, composite, &HighpassFilter);
     /*************************************************************
     * БПФ выходного сигнала фильтра
     *************************************************************/
    for(i = 0; i < FFT_BLOCK_LENGTH; i ++)          //инициализация входного
    {                                               //массива
       fftInputOutput[i].real = ((fractional)output[i]) >> 1;
       fftInputOutput[i].imag = 0x0;
    }

    FFTComplexIP(LOG2_BLOCK_LENGTH,  fftInputOutput, twiddleFactors,
                        COEFFS_IN_DATA);
    BitReverseComplex(LOG2_BLOCK_LENGTH, fftInputOutput);
    SquareMagnitudeCplx(FFT_BLOCK_LENGTH, fftInputOutput, postFilterFFTMag);

    Nop();                                          // для размещения точки останова

   while(1);                                        // бесконечный цикл
}
```

Функции *TwidFactorInit*, *FFTComplexIP*, *BitReverseComplex* и *SquareMagnitudeCplx* реализованы в предварительно компилированной библиотеке *libdsp_coff.a*. Прототипы этих функций находятся в заголовочном файле библиотеки *dsp.h*.

5.2 Создание проекта и трансляция программы

5.2.1 В своей рабочей папке создайте папку *lr5*. Скопируйте в эту папку файлы *composite.h*, *pass.s*, *IIRT_filter.s* и *IIR_filter.h* из папки *lr4*. Исходный текст главного модуля программы хранится в файле *main.c*.

5.2.2 Создайте новый проект, для этого выберите команду *Project > Project Wizard* и последовательно выполните действия:

- задайте тип микроконтроллера *dsPIC33FJ256GP506*;

- задайте транслятор *Microchip C30 Toolsuite*;

- задайте имя проекта *lr5* и папку для нового проекта;

- включите в проект файлы с исходным текстом программы *main.c, pass.s, IIRT_filter.s.*

В окне менеджера проекта нажмите правой клавишей по полю с надписью *Linker Scripts*. Выберите *Add Files*…. Подключите файл *p33FJ256GP506.gld*, расположенный в каталоге *C:\MicrochipStarter Kits \dsPIC Starter Kit – 1\MPLAB C30 \Support\gld*.

Нажмите правой клавишей по полю с надписью *Header Files*. Выберите *Add Files*…. Подключите файл *p33FJ256GP506.inc*, расположенный в каталоге *C:\MicrochipStarter Kits \dsPIC Starter Kit – 1\MPLAB C30 \Support\inc*. Нажмите правой клавишей по полю с надписью *Header Files*. Выберите *Add Files*…. Подключите файлы *IIR_filter.h* и *composite.h.*

Нажмите правой клавишей по полю с надписью *Library Files*. Выберите *Add Files*…. Подключите файл *libdsp_coff.a*, расположенный в каталоге *C:\MicrochipStarter Kits \dsPIC Starter Kit – 1\MPLAB C30 \lib*.

Откройте окно с исходным текстом программы, для этого в окне менеджера проекта *lr5.mcw* дважды щелкните по ярлыку *main.c* в папке *Source Files* или выполните команду *File>Open*.

5.2.3 Выполните трансляцию программы (команда *Project>Make*). На закладке *Build* в окне *Output* будет приведена командная строка, посланная компилятору, сообщения об ошибках или успешном завершении трансляции. Если программа содержала синтаксические и другие формальные ошибки, в окне будет выведен список сообщений об ошибках (ERROR). Если дважды щелкнуть мышкой на сообщении об ошибке, то курсор автоматически перейдет к соответствующему оператору в исходном тексте. Если в программе нет подобных ошибок, компилятор выдаст сообщение BUILD SUCCEEDED.

5.3 Выполнение эксперимента

5.3.1 В главном меню MPLAB IDE выберите команду *Debugger>Simulator Settings*. На закладке *Osc / Trace* задайте значение тактовой частоты процессора 80 МГц.

В главном меню MPLAB IDE выберите команду *Tools > DMCI – Data Monitor Control Interface*. Выберите закладку *Dynamic Data View*. Установите флажки на полях графиков *Graph 1*, *Graph 2*, *Graph 3* и *Graph 4*. Установите курсор на поле *Graph 1*. Нажмите правую кнопку и выберите команду *Configure Data Source*. На закладке *Graph 1* в поле *Data Sources* выберите массив *composite*, задайте *Display Format – Decimal*. Нажмите *OK*. На закладке *Graph 2* в поле *Data Sources* выберите массив *preFilterFFTMag*, задайте *Display Format – Decimal*. Нажмите *OK*. Аналогично на закладках *Graph 3* и *Graph 4* выберите массивы *output* и *postFilterFFTMag*.

Используя DMCI, оцените корректность выполнения процедуры вычисления спектров входного и выходного сигнала фильтра. Определите значения частот гармоник входного и выходного сигналов. Разрешающая способность по частоте получается делением частоты дискретизации f_s на число точек БПФ *FFT_BLOCK_LENGTH*.

5.4 Содержание отчета

Отчет должен содержать:

- описание решения задачи с привлечением формул и схемы алгоритма основного модуля программы;
- текст отлаженной программы с комментариями;
- временные и спектральные диаграммы сигналов;
- выводы по работе.

5.5 Контрольные вопросы и задания

Поясните содержание задачи, выполняемой микроконтроллером.

Поясните принцип работы алгоритма БПФ.

Поясните назначение и работу использованных функций из библиотеки *libdsp_coff.a*.

Поясните характер частотного спектра входного и выходного сигнала цифрового фильтра.

6 Цифровой синтез сигналов

6.1 Описание задания

6.1.1 В лабораторной работе исследуется работа синтезатора аналоговых сигналов. Для выполнения работы используется отладочный стенд *Starter Kit for dsPIC DSC*. На платформе dsPIC реализован метод прямого цифрового синтеза сигналов (DDS), основанный на циклическом опросе массива *composite*, хранящего 64 оцифрованных отсчетов сигнала заданной формы. Для восстановления аналогового сигнала используется встроенный модуль *OC1* в режиме ШИМ, интервальный таймер *TMR2* и внешний восстанавливающий ФНЧ. Интервальный таймер *TMR2* и модуль *OC1* настроены на работу с несущей частотой ШИМ 32 кГц. В отладочном стенде *Starter Kit for dsPIC DSC* к выходу модуля *OC1* подключен аналоговый ФНЧ четвертого порядка с граничной частотой полосы пропускания 3300 Гц. Подавление несущей частоты 32 кГц составляет более 70 дБ. Подавление постоянной составляющей выходного сигнала выполняет усилитель звуковых частот.

В программе dsPIC предусмотрено управление уровнем воспроизводимого сигнала. Масштабный коэффициент *gain* устанавливается кнопками «+» и «–» в диапазоне от 1/16 до 1 с шагом 1/16. Вывод нормированных дискретных отсчетов в модуль ШИМ происходит по каналу прямого доступа к памяти *DMA1*, который работает в режиме «пинг-понг». В этом режиме поочередно используются буферы

buffer1 и *buffer2*. Признак *bufferIndicator* указывает на свободный для заполнения буфер. Запрос для передачи по каналу *DMA1* тоже происходит при переполнении таймера *TMR2*. Таким образом, поток отсчетов в модуль ШИМ предается с частотой 32 кГц. Значение длительности импульса для текущего отсчета определяется содержимым регистра *OC1R*, следующего – содержимым регистра *OC1RS*.

6.1.2 Программа микроконтроллера имеет модульную структуру. Исходный текст главного файла программы *main.c* содержит директивы препроцессора для включения заголовочных файлов, директивы для настройки слова конфигурации, директивы определения констант.

```
#include <p33Fxxxx.h>                    // директивы препроцессора
#include "OCPWMDrv.h"
#include "composite.h"
#include "sask.h"

_FGS(GWRP_OFF & GCP_OFF);          //слово конфигурации
_FOSCSEL(FNOSC_FRC);
_FOSC(FCKSM_CSECMD & OSCIOFNC_ON & POSCMD_NONE);
_FWDT(FWDTEN_OFF);

#define FRAME_SIZE          64          // длина кадра
```

Далее следует объявление буфера *ocPWMBuffer*, используемого каналом *DMA1* и переменной *gain*:

```
int    ocPWMBuffer    [OCPWM_DMA_BUFSIZE]   __attribute__((space(dma)));
unsigned char   gain;                              // масштабный коэффициент
```

Затем следуют объявление переменной *ocPWMHandle* типа *OCPWMHandle* (структура драйвера) и ее указателя *pOCPWMHandle*.

```
OCPWMHandle   ocPWMHandle;                  // объявление структуры драйвера
OCPWMHandle   *pOCPWMHandle = &ocPWMHandle;      //указатель на структуру
```

Объявления констант, экземпляра структуры драйвера ШИМ *OCPWMHandle* и прототипов функций драйвера.

```
void OCPWMInit  (OCPWMHandle * pHandle,int * pBufferInDMA);
```

```
void OCPWMStart (OCPWMHandle * pHandle);
void OCPWMWrite(OCPWMHandle * pHandle,int *buffer,int size, unsigned char gain);
int OCPWMIsBusy      (OCPWMHandle * pHandle);
void OCPWMStop       (OCPWMHandle * pHandle);
```

приведено в заголовочном файле *OCPWMDrv.h*. Исходные тексты этих функций приведены в файле *OCPWMDrv.c*.

6.1.3 Основной модуль программы содержит операторы инициализации тактового генератора МК, вызова функции инициализации структуры драйвера, функции запуска ШИМ, функции инициализации портов и оператор инициализации переменной *gain*.

```
int main(void)                                  // основной модуль программы
{
/* Настройка тактового генератора на частоту 40MHz.
        * Fosc= Fin*M/(N1*N2), Fcy=Fosc/2
        * Fosc= 7.37M*40/(2*2)=80Mhz for 7.37M input clock */

        PLLFBD=41;                      /* M=41+2  */
        CLKDIVbits.PLLPOST=0;           /* N1=2    */
        CLKDIVbits.PLLPRE=0;            /* N2=2    */
        OSCTUN=0;

        __builtin_write_OSCCONH(0x01);          /*Внутренний FRC с PLL*/
        __builtin_write_OSCCONL(0x01);
        while (OSCCONbits.COSC != 0b01);        /*Ожидание запуска и захвата*/
        while(!OSCCONbits.LOCK);

        OCPWMInit (pOCPWMHandle,ocPWMBuffer);  // инициализация OCPWM
        OCPWMStart (pOCPWMHandle);              // запуск OCPWM
        SASKInit();                             //инициализация портов
        gain = 16;                              //инициализация усиления
```

6.1.4 В основном цикле программы опрашивается флаг *isWriteBusy* до тех пор, пока он не будет сброшен по прерыванию от канала *DMA1*. После этого вызывается функция *OCPWMWrite*, которая заполняет свободный буфер канала *DMA1* значениями нормированных выборок выходного сигнала. Проверяется состояние линий порта ввода, к которым подключены кнопки. При активном состоянии выполняется коррекция нормирующего коэффициента *gain* в диапазоне от 1 до 16.

```
    while(1)                                    //основной цикл для каждого кадра
    {
        /* Ожидание доступности ОС для нового кадра*/
        while(OCPWMIsBusy(pOCPWMHandle));

        /* Запись кадра на вывод*/
        OCPWMWrite (pOCPWMHandle, composite, FRAME_SIZE, gain);

        if((CheckSwitchS1()) == 1)      //коррекция gain
        {
            if (gain < 17)
                gain = ++gain;          /* инкремент gain.*/
        }

        if((CheckSwitchS2()) == 1)
        {
            if (gain > 0)
                gain = --gain;          /* декремент gain.*/
        }
    }

}
```

6.2 Создание проекта и трансляция программы

6.2.1 В своей рабочей папке создайте папку *lr6*. Из меню *Пуск/ Все программы* запустите программу dsPICworks. В строке команд программы выберите закладку *Generator* и пункт *Sinusoidal*. В открывшейся панели *Generating Sinusoidal Wave* задайте значения параметров:

Signal Frequency (частота сигнала)	500 Гц
Sampling Rate (частота дискретизации)	32000 Гц
Number of Samples (количество отсчетов)	64
Angular Phase Delay (задержка фазы)	0
Peak Amplitude (from zero) (амплитуда)	1
DC offset (смещение постоянной составляющей)	0
Output File Format (формат выходного файла)	ASCII
Output Number Type (тип значений сигнала)	16 bit Fractional Fixed point

Random Noise Type (Тип шума) no noise.

Нажмите кнопку *File Name*, укажите свою рабочую папку и задайте имя файла *sine500Hz.tim*. Нажмите кнопку *OK*. В результате откроется окно с временной диаграммой сигнала.

Повторите аналогичные действия для сигнала частотой 2000 Гц. Сохраните файл сигнала *sine2000Hz.tim* в рабочей папке.

В строке команд программы выберите *Operation* пункт *Arithmetic/Linear Combo*. На открывшейся закладке укажите имена файлов для сигналов:

- input x1(n) sine500Hz.tim;
- input x2(n) sine2000Hz.tim;
- output y(n) composite.tim.

Задайте значения коэффициентов: a = 0.5; b=0.5; c=0. Нажмите кнопку *OK*. В результате откроется окно с временной диаграммой двухчастотного сигнала.

В строке команд программы выберите *File* пункт *Export File*. Задайте файл источника *Source File*, установите флажок на опции *Create C data*. Выберите формат экспорта файла *Export File Format* как *Fractional/Integer ASCII Hexadecimal*. Результат будет сохранен в файле *composite.h*.

6.2.2 Скопируйте в папку lr6 подготовленные файлы с исходным текстом программы *main.c*, *OCPWMDrv.c*, *OCPWMDrv.h*, *sask.c*, *sask.h*. Создайте новый проект, для этого выберите команду *Project > Project Wizard* и последовательно выполните действия:

- задайте тип микроконтроллера *dsPIC33FJ256GP506*;
- задайте транслятор *Microchip C30 Toolsuite*;
- задайте имя проекта *lr6* и папку для нового проекта;
- включите в проект файлы с исходным текстом программы *main.c*, *OCPWMDrv.c*, *sask.c*.

В окне менеджера проекта нажмите правой клавишей по полю с надписью *Linker Script>*. Выберите *Add Files*…. Подключите файл *p33FJ256GP506.gld*,

расположенный в каталоге *C:\MicrochipStarter Kits \dsPIC Starter Kit – 1\MPLAB C30 \Support\gld*.

Нажмите правой клавишей по полю с надписью *Header Files*. Выберите *Add Files*…. Подключите файл *p33FJ256GP506.inc*, расположенный в каталоге *C:\MicrochipStarter Kits \dsPIC Starter Kit – 1\MPLAB C30 \Support\inc*. Подключите файлы *OCPWMDrv.h, sask.h, composite.h.*

6.2.3 Выполните трансляцию программы (команда *Project>Make*). На закладке *Build* в окне *Output* будет приведена командная строка, посланная компилятору, сообщения об ошибках или успешном завершении трансляции. Если программа содержала синтаксические и другие формальные ошибки, в окне будет выведен список сообщений об ошибках (ERROR). Если дважды щелкнуть мышкой на сообщении об ошибке, то курсор автоматически перейдет к соответствующему оператору в исходном тексте. Если в программе нет подобных ошибок, компилятор выдаст сообщение BUILD SUCCEEDED.

6.3 Выполнение эксперимента

6.3.1 В главном меню MPLAB IDE командой *Debugger>Select Tool*. выберите MPLAB SIM. В главном меню MPLAB IDE выберите команду *Debugger>Simulator Settings*. На закладке *Osc / Trace* задайте значение тактовой частоты процессора 80 МГц. Установите флаг *Trace All* и задайте размер буфера трассы отладчика 40 *K lines*. В главном меню MPLAB IDE выберите команду *View>Simulator Logic Analyzer*. В окне *Logic Analyzer* нажатием кнопки *Channels* выберите сигнал *OC1* для отображения в окне логического анализатора. Подтвердите выбор кнопкой *OK*.

6.3.2 Откройте окно наблюдения командой *View/Watch*, выберите для отображения в окне наблюдения указатель структуры *pOCPWMHandle*. Определите значения указателей буферов *buffer1* и *buffer2*, используемых каналом *DMA1*. Командой *View/File Registers* откройте окно регистрового файла, в котором подготовьте для наблюдения 64 ячейки, начиная с адреса по указателю *buffer1*.

6.3.3 Откройте окно исходного текста основного модуля программы *main.c*, установите точку останова на операторе вызова подпрограммы заполнения буферов

OCPWMWrite (pOCPWMHandle,composite,FRAME_SIZE, gain);

Выполните несколько прогонов программы до точки останова, наблюдайте заполнение данными буферов канала *DMA1*. Одновременно наблюдайте формирование ШИМ сигнала в окне логического анализатора. Скопируйте содержимое буферов и окна логического анализатора в отчет.

6.3.4 Подключите аудио выход *Speaker Out* стенда *Starter Kit for dsPIC DSC* к линейному входу звуковой карты персонального компьютера. Перемычку J6 установите в положение *OCPWM*. Перемычку J7 установите в положение *Line In*. Подключите стенд *Starter Kit for dsPIC DSC* к USB порту персонального компьютера. Подключите отладочный модуль стенда к отладчику системы MPLAB командой *Debugger>Select Tool>Starter Kits.*

Убедитесь, что на закладке *Starter Kit Debugger* в окне *Output* появилось сообщение:

Starter Kit board connected
Firmware version: 00.00.19.

Выполните программирование dsPIC командой *Debugger>Program*. Запустите процесс исполнения программы командой *Debugger>Run*.

6.3.5 Из папки dsPIC\soft запустите на исполнение программу виртуального прибора – осциллографа (Osc). Используя виртуальный осциллограф, оцените корректность формирования аналогового сигнала.

6.3.6 Остановите процесс исполнения программы командой *Debugger>Halt*. Отключите отладочный модуль стенда от отладчика MPLAB IDE командой *Debugger>Select Tool>None.* Отключите стенд от USB порта персонального компьютера. Отключите аудио выход *Speaker Out* стенда *Starter Kit for dsPIC DSC* от линейного входа звуковой карты персонального компьютера.

6.4 Содержание отчета

Отчет должен содержать:

- описание решения задачи с привлечением формул и схем алгоритмов;
- текст отлаженной программы с комментариями;
- содержимое буферов канала *DMA1*, временные диаграммы ШИМ сигнала и восстановленного аналогового сигнала.

6.5 Контрольные вопросы и задания

Поясните содержание задачи, выполняемой микроконтроллером.

Поясните назначение слова конфигурации.

Поясните, каким образом выполнена настройка тактирования МК.

Поясните принцип работы канала прямого доступа к памяти в режиме «пинг-понг».

Поясните работу модуля ОС в режиме ШИМ.

Поясните работу основной программы.

Поясните работу функций драйвера модуля ШИМ.

7 Исследование КИХ-фильтра

7.1 Описание задания

7.1.1 В лабораторной работе исследуется работа КИХ-фильтра, реализованного на платформе dsPIC. Для выполнения работы используется отладочный стенд *Starter Kit for dsPIC DSC.* Для оцифровки входного сигнала используется встроенный в dsPIC модуль АЦП, для восстановления аналогового сигнала используется встроенный модуль ШИМ и ФНЧ.

7.1.2 Программа микроконтроллера имеет модульную структуру. Исходный текст главного файла программы *main.c* содержит директивы препроцессора для включения заголовочных файлов, директивы для настройки слова конфигурации, директивы определения констант.

```
#include <p33Fxxxx.h>                    // директивы препроцессора
#include "OCPWMDrv.h"
#include "ADCChannelDrv.h"
#include "FIR_Filter.h"

_FGS(GWRP_OFF & GCP_OFF);                //слово конфигурации
_FOSCSEL(FNOSC_FRC);
_FOSC(FCKSM_CSECMD & OSCIOFNC_ON & POSCMD_NONE);
_FWDT(FWDTEN_OFF);

#define FRAME_SIZE          128          // длина кадра
#define BLOCK_LENGTH    128              //длина блока.
```

Далее следует объявление буферов *adcBuffer*, *ocPWMBuffe*, используемых каналом DMA, массива выборок сигнала *samples*.

```
int    adcBuffer    [ADC_CHANNEL_DMA_BUFSIZE] __attribute__((space(dma)));
int    ocPWMBuffer      [OCPWM_DMA_BUFSIZE]   __attribute__((space(dma)));
int    samples          [FRAME_SIZE];         // объявление массива выборок
```

Затем следуют объявление переменных *adcChannelHandle* типа *ADCChannelHandle* (структура драйвера АЦП) *ocPWMHandle* типа *OCPWMHandle* (структура драйвера ШИМ) и их указателей *pADCChannelHandle* и *pOCPWMHandle*, а также переменной *Filter* типа *FIRStruct* и массива *FilterOut* для хранения выборок отфильтрованного сигнала.

```
ADCChannelHandle adcChannelHandle;           // объявление структуры драйверов
OCPWMHandle    ocPWMHandle;
//указатели на структуры драйверов
ADCChannelHandle *pADCChannelHandle  = &adcChannelHandle;
OCPWMHandle    *pOCPWMHandle = &ocPWMHandle;

extern FIRStruct Filter;                     // объявление внешней структуры фильтра
int FilterOut[BLOCK_LENGTH];                 // объявление массива выборок на выходе
```

Объявления констант, экземпляра структуры драйвера АЦП *ADCChannelHandle* и прототипов функций драйвера приведено в заголовочном файле *ADCChannelDrv.h*. Исходные тексты этих функций приведены в файле *ADCChannelDrv.c*.

Объявления констант, экземпляра структуры драйвера ШИМ *OCPWMHandle* и прототипов функций драйвера приведено в заголовочном файле *OCPWMDrv.h*. Исходные тексты этих функций приведены в файле *OCPWMDrv.c*.

Объявления структуры *FIRStruct* и прототипов функций инициализации структуры *FIRDelayInit* и КИХ-фильтра *FIR* приведено в заголовочном файле *FIR_Filter.h*.

7.1.3 Далее основной модуль программы содержит операторы инициализации тактового генератора МК, вызова функции инициализации структур драйверов АЦП и ШИМ, функции запуска АЦА и ШИМ, функции инициализации КИХ-фильтра.

```
int main(void)
{
    /* Настройка тактового генератора на частоту 40MHz.
     * Fosc= Fin*M/(N1*N2), Fcy=Fosc/2
     * Fosc= 7.37M*43/(2*2)=80Mhz for 7.37M input clock */

    PLLFBD=41;                      /* M=41+2  */
    CLKDIVbits.PLLPOST=0;           /* N1=2    */
    CLKDIVbits.PLLPRE=0;            /* N2=2    */
    OSCTUN=0;                       //центральная частота 7.37МГц

    __builtin_write_OSCCONH(0x01);          /*Внутренний FRC с PLL*/
    __builtin_write_OSCCONL(0x01);
    while (OSCCONbits.COSC != 0b01);    /*Ожидание переключения и захвата*/
    while(!OSCCONbits.LOCK);

    ADCChannelInit  (pADCChannelHandle,adcBuffer);   //инициализация АЦП
    OCPWMInit (pOCPWMHandle,ocPWMBuffer);            //инициализация OCPWM

    ADCChannelStart (pADCChannelHandle);             // запуск АЦП
    OCPWMStart      (pOCPWMHandle);                  // запуск OCPWM
    FIRDelayInit(&Filter);                           //инициализация фильтра
```

7.1.4 В основном цикле программы опрашивается флаг *ADCChannelIsBusy* до тех пор, пока он не будет сброшен по прерыванию от канала *DMA0*. После этого вызывается функция *ADCChannelRead*, которая копирует заполненный буфер канала *DMA0* значениями выборок входного сигнала в массив *samples*. Затем функция КИХ-фильтра *FIR* выполняет фильтрацию и заполняет массив *FilterOut*. После

сброса флага *OCPWMIsBusy* функция *OCPWMWrite* выполняет их нормирование и вывод через свободный буфер канала *DMA1*.

```
    while (1)
    {
        /* Ожидание доступности канала ввода для нового кадра    */
        while (ADCChannelIsBusy(pADCChannelHandle));
        /* Заполнение массива samples */
        ADCChannelRead(pADCChannelHandle,samples,FRAME_SIZE);
        /* Функция КИХ-фильтра  */
        FIR(BLOCK_LENGTH,&FilterOut[0],&samples[0],&Filter);
        /* Ожидание доступности ОС для нового кадра*/
        while (OCPWMIsBusy(pOCPWMHandle));
        /* Запись кадра на вывод */
        OCPWMWrite(pOCPWMHandle,FilterOut,FRAME_SIZE);
    }

}
```

7.2 Создание проекта и трансляция программы

7.2.1 В своей рабочей папке создайте папку *lr7*. Скопируйте в папку *lr7* подготовленные файлы с исходным текстом программы *main.c*, *Pass.s*, *FIR_filter.s*, *FIR_filter.h*, *ADCChannelDrv.c*, *ADCChannelDrv.h*, *OCPWMDrv.c*, *OCPWMDrv.h*. Создайте новый проект, для этого выберите команду *Project > Project Wizard* и последовательно выполните действия:

- задайте тип микроконтроллера *dsPIC33FJ256GP506*;

- задайте транслятор *Microchip C30 Toolsuite*;

- задайте имя проекта *lr7* и папку для нового проекта;

- включите в проект файлы с исходным текстом программы *main.c*, *Pass.s*, *FIR_filter.s*, *ADCChannelDrv.c*, *OCPWMDrv.c*.

В окне менеджера проекта нажмите правой клавишей по полю с надписью <*Linker Scripts*>. Выберите <*Add Files…*>. Подключите файл <*p33FJ256GP506.gld*>, расположенный в каталоге <*C:\MicrochipStarter Kits \dsPIC Starter Kit – 1\MPLAB C30 \Support\gld*>.

Нажмите правой клавишей по полю с надписью <*Header Files*>. Выберите <*Add Files*…>. Подключите файл <*p33FJ256GP506.inc*>, расположенный в каталоге <*C:\MicrochipStarter Kits \dsPIC Starter Kit – 1\MPLAB C30 \Support\inc*>. Подключите файлы *FIR_filter.h*, *ADCChannelDrv.h* , *OCPWMDrv.h*.

7.2.2 Выполните трансляцию программы (команда *Project>Make*). На закладке *Build* в окне *Output* будет приведена командная строка, посланная компилятору, сообщения об ошибках или успешном завершении трансляции. Если программа содержала синтаксические и другие формальные ошибки, в окне будет выведен список сообщений об ошибках (ERROR). Если дважды щелкнуть мышкой на сообщении об ошибке, то курсор автоматически перейдет к соответствующему оператору в исходном тексте. Если в программе нет подобных ошибок, компилятор выдаст сообщение BUILD SUCCEEDED.

7.3 Выполнение эксперимента

7.3.1 Подключите аудио выход *Speaker Out* стенда *Starter Kit for dsPIC DSC* к линейному входу звуковой карты персонального компьютера. Подключите аудио вход *Line In* стенда Starter Kit for dsPIC DSC к линейному выходу звуковой карты персонального компьютера. Перемычку J6 установите в положение *OCPWM*. Перемычку J7 установите в положение *Line In*. Подключите стенд *Starter Kit for dsPIC DSC* к USB порту персонального компьютера. Подключите отладочный модуль стенда к отладчику системы MPLAB IDE командой *Debugger>Select Tool>Starter Kits.* Убедитесь, что на закладке *Starter Kit Debugger* в окне *Output* появилось сообщение:

Starter Kit board connected
Firmware version: 00.00.19.

Выполните программирование dsPIC командой *Debugger>Program*. Запустите процесс исполнения программы командой *Debugger>Run*.

7.3.2 Из папки dsPIC\soft запустите на исполнение программы виртуальных приборов – генератора звуковых сигналов (Звуковой генератор 4.0) и осциллографа (Osc).

Используя виртуальные приборы, снимите АЧХ реализованного фильтра, оцените корректность выполнения процедуры фильтрации входного сигнала.

7.3.3 Остановите процесс исполнения программы командой *Debugger>Halt*. Отключите отладочный модуль стенда от отладчика MPLAB IDE командой *Debugger>Select Tool>None.* Отключите стенд от USB порта персонального компьютера.

7.4 Содержание отчета

Отчет должен содержать:

- описание решения задачи с привлечением формул и схем алгоритмов;
- текст отлаженной программы с комментариями;
- график АЧХ реализованного фильтра.
- выводы по работе.

7.5 Контрольные вопросы и задания

Поясните содержание задачи, выполняемой микроконтроллером.

Поясните принцип работы каналов прямого доступа к памяти.

Поясните особенности работы каналов обмена с АЦП и ШИМ.

Поясните работу функций драйвера АЦП.

Поясните работу функций драйвера ШИМ.

Список использованных источников

1 dsPIC33FJXXXGPX06/X08/X10 Data Sheet: High-Performance, 16-Bit Digital Signal Controllers, DS70286C, Microchip Technology Inc., 2009. [Электронный ресурс]. – Режим доступа: ww1.microchip.com/downloads/en/DeviceDoc/70286C.pdf. – 26.04.2018.