Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра электронных вычислительных средств

В. Б. Клюс, М. В. Качинский, А. Б. Давыдов

ПРОГРАММИРОВАНИЕ ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ ВЫЧИСЛИТЕЛЬНЫХ СРЕДСТВ РЕАЛЬНОГО ВРЕМЕНИ

Лабораторный практикум
для студентов специальности 1-40 02 02
«Электронные вычислительные средства»
дневной формы обучения

В 2-х частях

Часть 2

Минск БГУИР 2011

УДК 004.031.43(076.5)
ББК 32.973.26-018.2я73
К52

Р е ц е н з е н т:

доцент кафедры электронных вычислительных машин
учреждения образования
«Белорусский государственный университет информатики
и радиоэлектроники»,
кандидат технических наук, доцент А. А. Петровский

Клюс, В. Б.

К52 Программирование проблемно-ориентированных вычислительных средств реального времени : лаб. практикум для студ. спец. 1-40 02 02 «Электронные вычислительные средства» днев. формы обуч. В 2 ч. Ч. 2 / В. Б. Клюс, М. В. Качинский, А. Б. Давыдов. – Минск : БГУИР, 2011. – 36 с. : ил.

ISBN 978-985-488-527-8 (ч. 2).

Лабораторный практикум содержит описание четырех лабораторных работ, в которых рассматриваются алгоритмы прямого и обратного быстрого преобразования Фурье. Приведены примеры реализации некоторых алгоритмов быстрого преобразования Фурье и использования цифровых процессоров сигналов для формирования и анализа различных сигналов. Лабораторные работы ориентированы на использование отладочного модуля на базе процессора TMS320VC5402 и выполняются на ПЭВМ.

УДК 004.031.43(076.5)
ББК 32.973.26-018.2я73

Часть первая издана в БГУИР в 2008 г.

ISBN 978-985-488-527-8 (ч. 2)
ISBN 978-985-488-286-4

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА №1.
Программирование базовой операции алгоритмов быстрого преобразования Фурье 4
ЛАБОРАТОРНАЯ РАБОТА №2.
Программирование быстрого преобразования Фурье с прореживанием по времени 9
ЛАБОРАТОРНАЯ РАБОТА №3.
Программирование быстрого преобразования Фурье с прореживанием по частоте 15
ЛАБОРАТОРНАЯ РАБОТА №4.
Измерение амплитудно-частотной характеристики цифрового рекурсивного фильтра 21
ПРИЛОЖЕНИЕ 1 25
ПРИЛОЖЕНИЕ 2 26
ПРИЛОЖЕНИЕ 3 31
ЛИТЕРАТУРА 35

ЛАБОРАТОРНАЯ РАБОТА №1

ПРОГРАММИРОВАНИЕ БАЗОВОЙ ОПЕРАЦИИ АЛГОРИТМОВ БЫСТРОГО ПРЕОБРАЗОВАНИЯ ФУРЬЕ

ЦЕЛЬ РАБОТЫ

Знакомство с различными алгоритмами быстрого преобразования Фурье (БПФ); разработка подпрограмм для реализации базовых операций («бабочек») для алгоритмов БПФ с прореживанием по времени и частоте на ассемблере процессора TMS320VC5402 и их отладка на лабораторном макете TMS320VC5402 DSP Starter Kit (DSK).

1.1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1.1. Алгоритм быстрого преобразования Фурье с прореживанием по времени

Алгоритм БПФ с прореживанием по времени с учетом симметрии поворачивающих множителей W можно описать следующими уравнениями:

$$\begin{aligned} F_K &= G_K + W_N^K \cdot H_K, \\ F_{K+N/2} &= G_K + W_N^{K+N/2} \cdot H_K = G_K - W_N^K \cdot H_K, \end{aligned} \tag{1.1}$$

где F_K – БПФ исходной последовательности; G_K и H_K – БПФ последовательностей из четных и нечетных элементов соответственно, а

$$W_N^{nk} = e^{-j\left(\frac{2\pi}{N}\right)nk} = \cos\left(\frac{2\pi}{N}\right)nk - j \cdot \sin\left(\frac{2\pi}{N}\right)nk - \tag{1.2}$$

поворачивающие множители.

Суть алгоритма БПФ с прореживанием по времени заключается в разбиении входной последовательности на четные и нечетные элементы, выполнение над ними БПФ в два раза меньшей размерности и объединение полученных результатов снова в одну последовательность по формулам (1.1) (рис. 1.1).

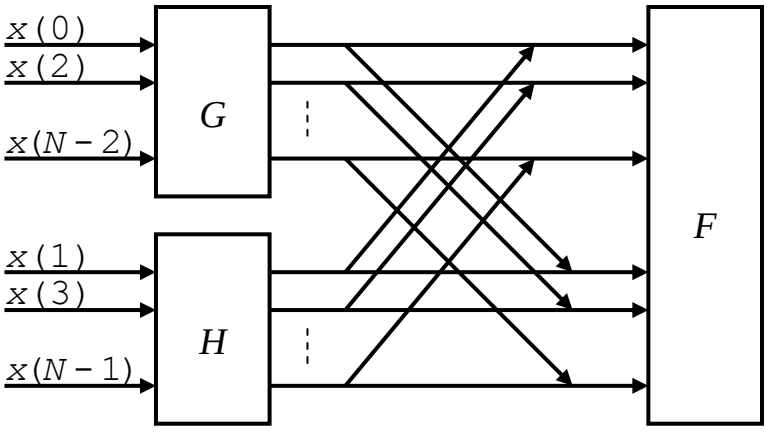

Рис. 1.1. Принцип БПФ с прореживанием по времени

Выражения (1.1) описывают базовую операцию алгоритма БПФ с прореживанием по времени, которую иногда называют «бабочкой» из-за ее графической интерпретации.

Поскольку процессор TMS320VC5402 не имеет команд для работы с комплексными числами, то действительная и мнимая части комплексного числа хранятся в паре соседних ячеек памяти, и все операции с ними выполняются отдельно. С учетом раздельного выполнения операций над действительной и мнимой частями двух элементов «бабочки» (переменные ***P*** и ***Q***) алгоритм вычисления «бабочки» для БПФ с замещением и прореживанием по времени можно записать в следующем виде:

$$P_{m+1} = P_m + Q_m \cdot W_N^k,$$
$$Q_{m+1} = P_m - Q_m \cdot W_N^k,$$

где $P_m = PR + j \cdot PI$; $Q_m = QR + j \cdot QI$; $W_N^k = \cos(x) - j \cdot \sin(x)$; $x = (2\pi/N) \cdot k$; $j = \sqrt{-1}$; откуда получим

$$\begin{aligned} P_{m+1} &= PR + j \cdot PI + (QR + j \cdot QI) \cdot [\cos(x) - j \cdot \sin(x)] = \\ &= PR + j \cdot PI + [QR \cdot \cos(x) + QI \cdot \sin(x)] + j \cdot [QI \cdot \cos(x) - QR \cdot \sin(x)] = \\ &= [\underline{PR + QR \cdot \cos(x) + QI \cdot \sin(x)}] + j \cdot [\underline{PI + QI \cdot \cos(x) - QR \cdot \sin(x)}]; \\ Q_{m+1} &= PR + j \cdot PI - (QR + j \cdot QI) \cdot [\cos(x) - j \cdot \sin(x)] = \\ &= PR + j \cdot PI - [QR \cdot \cos(x) + QI \cdot \sin(x)] - j \cdot [QI \cdot \cos(x) - QR \cdot \sin(x)] = \\ &= [\underline{PR - QR \cdot \cos(x) - QI \cdot \sin(x)}] + j \cdot [\underline{PI - QI \cdot \cos(x) + QR \cdot \sin(x)}]. \end{aligned}$$

1.1.2. Алгоритм быстрого преобразования Фурье с прореживанием по частоте

Алгоритм БПФ с прореживанием по частоте описывается следующими уравнениями:

$$\begin{aligned} G_K &= F_K + F_{K+N/2}, \\ H_K &= (F_K - F_{K+N/2}) \cdot W_N^K. \end{aligned} \qquad (1.3)$$

Суть БПФ с прореживанием по частоте заключается в разбиении входной последовательности на две по формулам (1.3) и выполнении над ними БПФ в два раза меньшей размерности, что можно проиллюстрировать рисунком (для 8-точечного БПФ) (рис. 1.2).

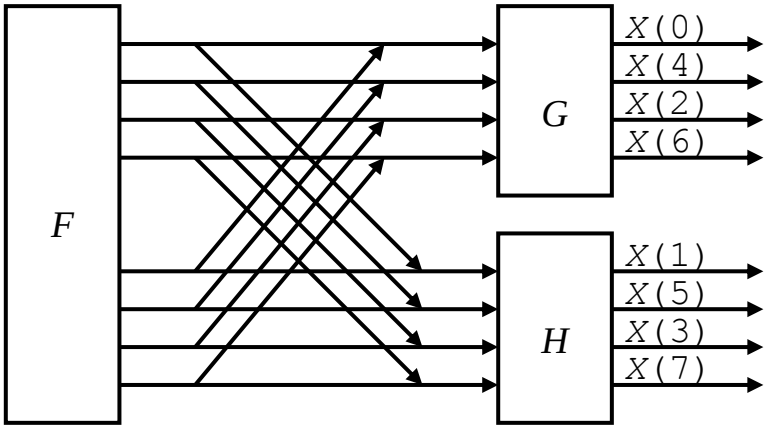

Рис. 1.2. Принцип БПФ с прореживанием по частоте

Выражения (1.3) описывают базовую операцию алгоритма БПФ с прореживанием по частоте.

С учетом раздельного выполнения операций над действительной и мнимой частями переменных ***P*** и ***Q*** алгоритм вычисления «бабочки» для БПФ с замещением и прореживанием по частоте может быть записан следующим образом:

$$P_{m+1} = P_m + Q_m,$$
$$Q_{m+1} = (P_m - Q_m)\cdot W_N^k,$$

и соответственно

$$P_{m+1} = PR + j\cdot PI + QR + j\cdot QI = [\underline{PR + QR}] + j\cdot[\underline{PI + QI}];$$
$$Q_{m+1} = (PR + j\cdot PI - QR - j\cdot QI)\cdot[\cos(x) - j\cdot\sin(x)] =$$
$$= PR\cdot cos(x) + j\cdot PI\cdot\cos(x) - QR\cdot\cos(x) - j\cdot QI\cdot\cos(x) -$$
$$- j\cdot PR\cdot\sin(x) + PI\cdot\sin(x) + j\cdot QR\cdot\sin(x) - QI\cdot\sin(x) =$$
$$= [\underline{(PR - QR)\cdot\cos(x) + (PI - QI)\cdot\sin(x)}] + j\cdot[\underline{(PI - QI)\cdot\cos(x) - (PR - QR)\cdot\sin(x)}].$$

1.1.3. Программная реализация базовых операций быстрого преобразования Фурье

Ниже в качестве примера приводятся фрагменты программ с использованием прямой адресацией для вычисления «бабочки» в алгоритмах БПФ с замещением и прореживанием по времени и частоте соответственно.

```
* DIT FFT Programm for TMS320C5402 DSP Starter Kit
* Direct Addressing            12/11/09 20:20:00
        .data
*       данные
PR      .word   0h              ; действительная часть P
PI      .word   0h              ; мнимая часть P
QR      .word   0h              ; действительная часть Q
QI      .word   0h              ; мнимая часть Q
COS     .word   0h              ; действительная часть W
SIN     .word   0h              ; мнимая часть W
        .text
*       инициализация
INIT    ld      #PR, DP         ; текущая страница данных
        ssbx    sxm             ; расширение знака
        ssbx    frct            ; умножение дробных чисел
*       «бабочка»
BTRFLY  ld      QR, T           ; T = QR
        mpy     COS, a          ; A = QR*cos
        ld      QI, T           ; T = QI
        mac     SIN, a          ; A = QR*cos+QI*sin = XR
        mpy     COS, b          ; B = QI*cos
        ld      QR, T           ; T = QR
        sth     a, QR           ; QR = a = XR
        mas     SIN, b          ; B = QI*cos-QR*sin = XI
        sth     b, QI           ; QI = b = XI
        ld      PI, 15, a       ; A = 1/2 PI
        add     QI, 15, a       ; A = 1/2 (PI+XI)
```

```
          sth     a, PI          ; PI = 1/2 new PI
          sub     QI, 16, a      ; A = 1/2 (PI-XI)
          sth     a, QI          ; QI = 1/2 new QI
          ld      QR, 15, a      ; A = 1/2 XR
          add     PR, 15, a      ; A = 1/2 (PR+XR)
          sth     a, PR          ; PR = 1/2 new PR
          sub     QR, 16, a      ; A = 1/2 (PR-XR)
          sth     a, QR          ; QR = 1/2 new QR

* DIF FFT Programm for TMS320C5402 DSP Starter Kit
* Direct Addressing             12/11/09 20:30:00
          .data
*         данные
PR        .word   0h             ; действительная часть P
PI        .word   0h             ; мнимая часть P
QR        .word   0h             ; действительная часть Q
QI        .word   0h             ; мнимая часть Q
COS       .word   0h             ; действительная часть W
SIN       .word   0h             ; мнимая часть W
          .text
*         инициализация
INIT      ld      #PR, DP        ; текущая страница данных
          ssbx    sxm            ; расширение знака
          ssbx    frct           ; умножение дробных чисел
*         «бабочка»
BTRFLY    ld      PR, 15, a      ; A = 1/2 PR
          add     QR, 15, a      ; A = 1/2 (PR+QR)
          sth     a, PR          ; PR = 1/2 new PR
          sub     QR, 16, a      ; A = XR = 1/2 (PR-QR)
          sth     a, QR          ; QR = a = XR
          ld      QI, 15, a      ; A = 1/2 QI
          add     PI, 15, a      ; A = 1/2 (PI+QI)
          sth     a, PI          ; PI = 1/2 new PI
          sub     QI, 16, a      ; A = XI = 1/2 (PI-QI)
          sth     a, QI          ; QI = a = XI
          ld      QR, T          ; T = QR = 1/2 (PR-QR)
          mpy     COS, a         ; A = 1/2 (PR-QR)*cos
          ld      QI, T          ; T = QI = 1/2 (PI-QI)
          mac     SIN, a         ; A =1/2[(PR-QR)cos+(PI-QI)sin]
          mpy     COS, b         ; B = 1/2 (PI-QI)*cos
          ld      QR, T          ; T = QR = 1/2 (PR-QR)
          sth     a, QR          ; QR = 1/2 new QR
          mas     SIN, b         ; B =1/2[(PI-QI)cos-(PR-QR)sin]
          sth     b, QI          ; QI = 1/2 new QI
```

При вычислении «бабочки» для исключения переполнения все выходные значения делятся на два.

При реализации программ БПФ прямая адресация данных должна быть заменена на косвенную адресацию с соответствующей модификацией адреса: переход от действительной части переменной (**PR** или **QR**) к мнимой части переменной (**PI** или **QI**) и обратно выполняется с использованием адресации

***ARx+** и ***ARx**– соответственно. Переход от действительной или мнимой части переменной **P** к аналогичной части переменной **Q** и обратно – с использованием адресации ***ARx+0** и ***ARx–0** соответственно. При этом подразумевается, что действительная и мнимая части каждого числа хранятся в соседних ячейках памяти (т. е. в двойном слове), а число ячеек между переменными **P** и **Q** находится в регистре **AR0**.

1.2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретические сведения по теме лабораторной работы (подразд. 1.1).
2. Получить у преподавателя задание для выполнения практической части работы.
3. Согласно заданию написать, оттранслировать и выполнить программу.
4. Продемонстрировать результат трансляции и работы программы преподавателю.
5. Оформить и защитить отчет по лабораторной работе.

1.3. СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы и исходные данные.
2. Описание алгоритма работы программы.
3. Листинг программы с комментариями.
4. Выводы по работе.

ЛАБОРАТОРНАЯ РАБОТА №2

ПРОГРАММИРОВАНИЕ БЫСТРОГО ПРЕОБРАЗОВАНИЯ ФУРЬЕ С ПРОРЕЖИВАНИЕМ ПО ВРЕМЕНИ

ЦЕЛЬ РАБОТЫ

Разработка программы БПФ для алгоритма с замещением данных и прореживанием по времени на ассемблере процессора TMS320VC5402 и ее отладка на лабораторном макете TMS320VC5402 DSP Starter Kit (DSK).

2.1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1.1. Основные модификации алгоритмов быстрого преобразования Фурье с замещением

Существует четыре основных алгоритма БПФ по модулю 2 с замещением, различающихся способом прореживания данных (по времени или по частоте) и расположением входных и выходных данных (прямое или двоично-инверсное). Особенностью алгоритмов с замещением является то, что если входные данные располагаются в обычном порядке, то выходные – в двоично-инверсном и наоборот.

В каждой паре алгоритмов один является основным и соответствует рис. 1.1 для прореживания по времени и рис. 1.2 для прореживания по частоте, а второй получается перестановкой входных и выходных данных, чтобы поменять прямой порядок данных на двоично-инверсный или наоборот.

2.1.2. Алгоритмы быстрого преобразования Фурье с замещением и прореживанием по времени

Ниже приведены граф-схема (рис. 2.1) и таблица индексов (табл. 2.1) 8-точечного БПФ, а также обобщенная схема алгоритма БПФ с прореживанием по времени и двоично-инверсным расположением входных данных (рис. 2.2). Далее приведены граф-схема (рис. 2.3) и таблица индексов (табл. 2.2) 8-точечного БПФ, а также обобщенная схема алгоритма БПФ с прореживанием по времени и прямым расположением входных данных (рис. 2.4).

Граф-схемы, таблицы индексов и схемы алгоритмов БПФ очень похожи, основное их различие заключается в начальных значениях приращений адресов и индексов, а также в направлении их изменения. При этом в дополнительном алгоритме изменяется также порядок следования поворачивающих множителей с прямого на двоично-инверсный, что незначительно усложняет алгоритм.

При программировании БПФ используются некоторые константы, определяющие положение и размерность обрабатываемых данных: **M** – число итераций БПФ, $\mathbf{N = 2^M}$ – размерность БПФ (число обрабатываемых комплексных отсчетов), **XFFT** – начальный адрес обрабатываемых данных.

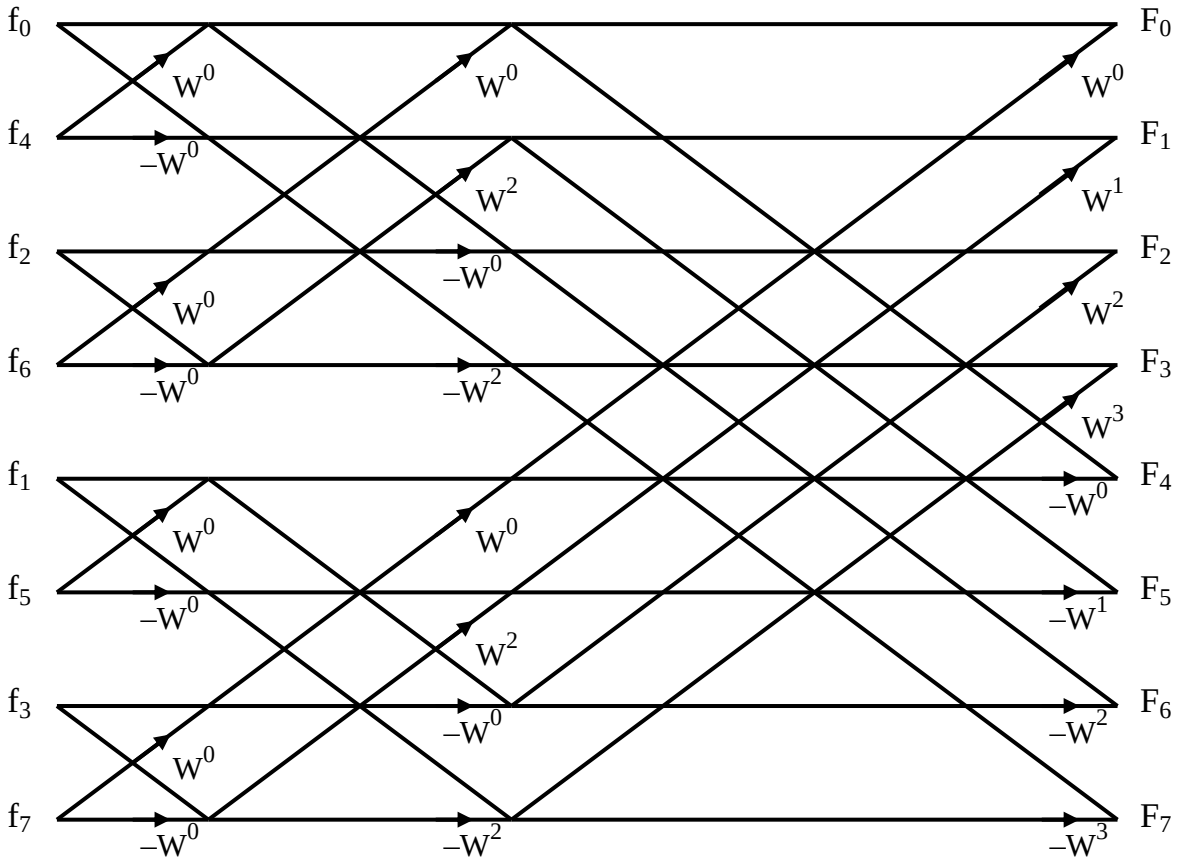

Рис. 2.1. Граф-схема БПФ с прореживанием по времени
и двоично-инверсным порядком входных данных

Таблица 2.1

Изменение индексов для БПФ с прореживанием по времени
и двоично-инверсным порядком входных данных

№ п/п	N2	IE	J	I	I + N2	K	Цикл	Переход
1	1	4	0	0	1	0	1	
2				2	3		1	
3				4	5		1	
4				6	7		1	3
5	2	2	0	0	2	0	1	
6				4	6		1	2
7			1	1	3	2	1	
8				5	7		1	3
9	4	1	0	0	4	0	2	
10			1	1	5	1	2	
11			2	2	6	2	2	
12			3	3	7	3	2	

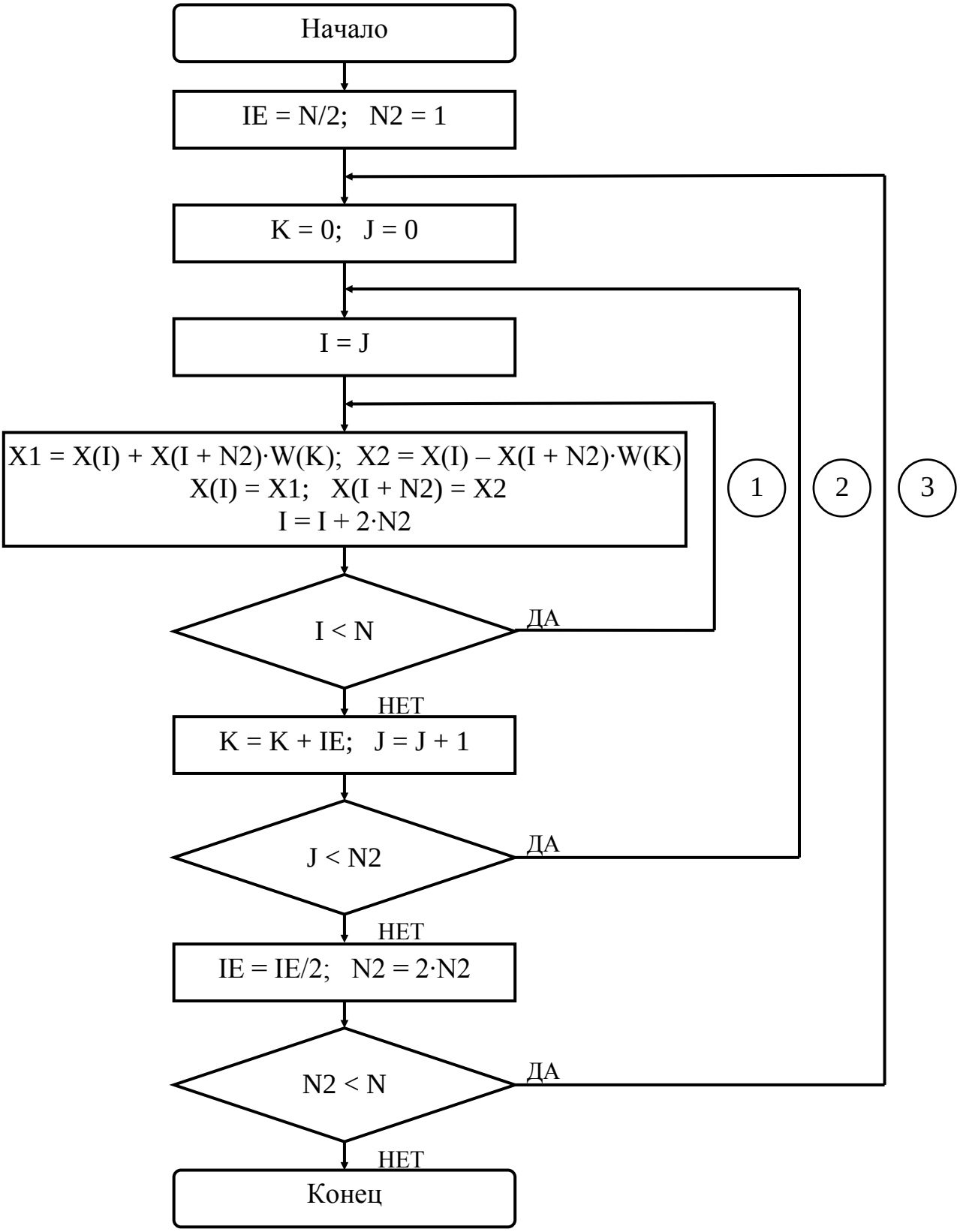

Рис. 2.2. Схема алгоритма БПФ с прореживанием по времени и двоично-инверсным порядком входных данных

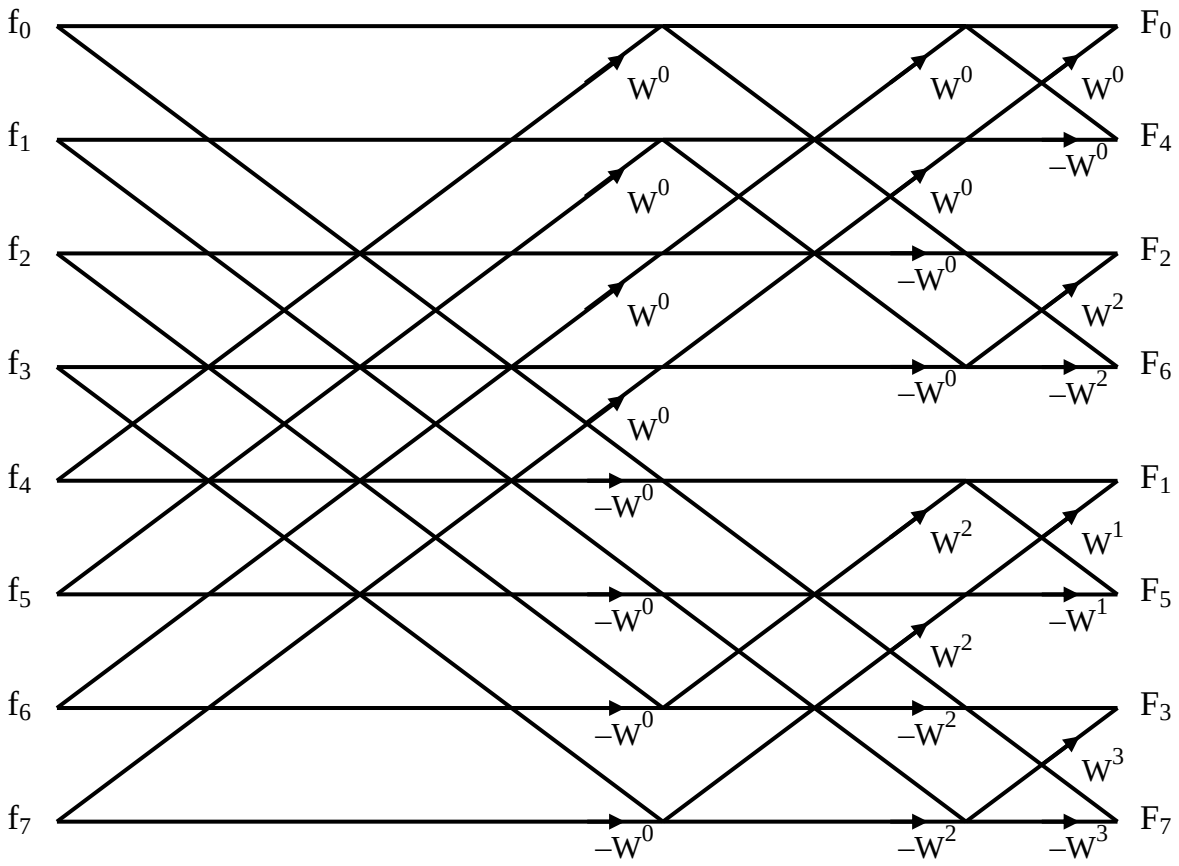

Рис. 2.3. Граф-схема БПФ с прореживанием по времени
и прямым порядком входных данных

Таблица 2.2

Изменение индексов для БПФ с прореживанием по времени
и прямым порядком входных данных

№ п/п	N2	IE	J	I	I + N2	K	Цикл	Переход
1	4	2	0	0	4	0	1	
2				1	5		1	
3				2	6		1	
4				3	7		1	3
5	2		0	0	2	0	1	
6				1	3		1	2
7			4	4	6	2	1	
8				5	7		1	3
9	1		0	0	1	0	2	
10			2	2	3	2	2	
11			4	4	5	1	2	
12			6	6	7	3	2	

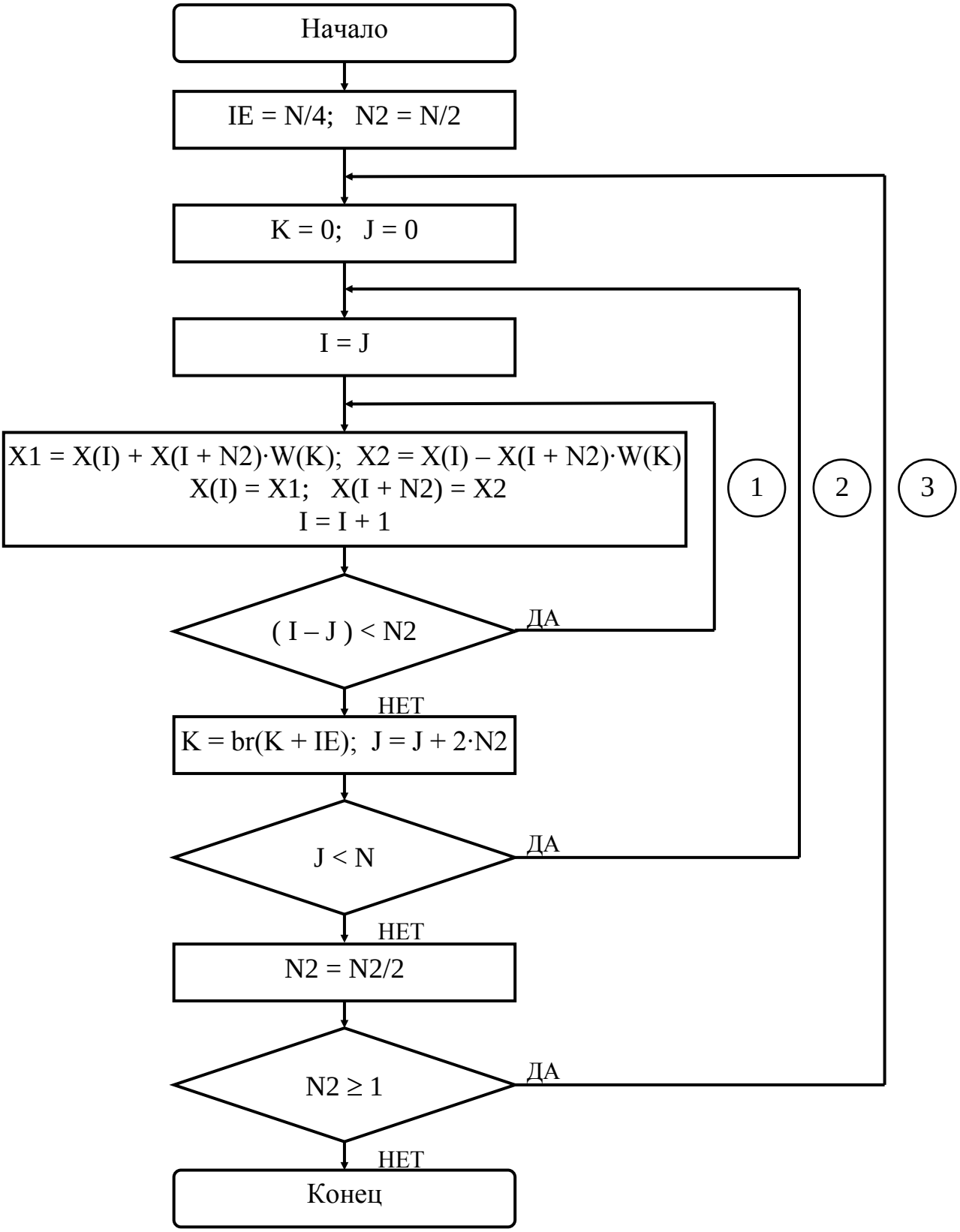

Рис. 2.4. Схема алгоритма БПФ с прореживанием по времени и прямым порядком входных данных

При написании программы БПФ используется также ряд переменных (ячеек памяти данных):

```
          .data
HOLDN     .word   0h            ; ячейка для хранения N
QUARTN    .word   0h            ; ячейка для хранения 1/4 N
N2        .word   0h            ; ячейка для переменной N2
N1        .word   0h            ; удвоенное значение N2
J         .word   0h            ; ячейка для переменной J
IADDR     .word   0h            ; адрес обрабатываемых данных
SINTBL    .word   0h            ; адрес в таблице синуса
COS       .word   0h            ; действительная часть W
SIN       .word   0h            ; мнимая часть W
I         .word   0h            ; ячейка для переменной I
IE        .word   0h            ; ячейка для переменной IE
INV       .word   0h            ; признак обратного БПФ
```

Назначение основных переменных следующее: **N2** – расстояние между двумя элементами «бабочки»; **J** – начальный индекс первого элемента группы «бабочек» с одинаковым поворачивающим множителем; **I** – индекс первого элемента текущей «бабочки»; **IE** – шаг изменения поворачивающих множителей.

Поскольку каждый элемент «бабочки» является комплексной переменной и занимает в памяти две ячейки, то истинное расстояние в памяти между элементами «бабочки» задается переменной **N1 = 2·N2**. Вместо переменной **K** в программе используется переменная **SINTBL**, которая хранит не индекс, а значение адреса в таблице синуса.

2.2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретические сведения по теме лабораторной работы (подразд. 2.1).
2. Получить у преподавателя задание для выполнения практической части работы.
3. Согласно заданию написать, оттранслировать и выполнить программу.
4. Продемонстрировать результат трансляции и работы программы преподавателю.
5. Оформить и защитить отчет по лабораторной работе.

2.3. СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы и исходные данные.
2. Описание алгоритма работы программы.
3. Текст программы с комментариями.
4. Выводы по работе.

ЛАБОРАТОРНАЯ РАБОТА №3

ПРОГРАММИРОВАНИЕ БЫСТРОГО ПРЕОБРАЗОВАНИЯ ФУРЬЕ С ПРОРЕЖИВАНИЕМ ПО ЧАСТОТЕ

ЦЕЛЬ РАБОТЫ

Разработка программы БПФ для алгоритма с замещением данных и прореживанием по частоте на ассемблере процессора TMS320VC5402 и ее отладка на лабораторном макете TMS320VC5402 DSP Starter Kit (DSK).

3.1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

3.1.1. Алгоритмы быстрого преобразования Фурье с замещением и прореживанием по частоте

Как и в паре алгоритмов с прореживанием по времени, так и в паре алгоритмов с прореживанием по частоте один является основным и соответствует рис. 1.2, а второй получается перестановкой входных и выходных данных, чтобы поменять прямой порядок данных на двоично-инверсный и наоборот.

Ниже приведены граф-схема (рис. 3.1) и таблица индексов (табл. 3.1) 8-точечного БПФ, а также обобщенная схема алгоритма БПФ с прореживанием по частоте и прямым расположением входных данных (рис. 3.2). Далее приведены граф-схема (рис. 3.3) и таблица индексов (табл. 3.2) 8-точечного БПФ, а также обобщенная схема алгоритма БПФ с прореживанием по частоте и двоично-инверсным расположением входных данных (рис. 3.4).

Граф-схемы, таблицы индексов и схемы двух алгоритмов БПФ с прореживанием по частоте очень похожи друг на друга и алгоритмы БПФ с прореживанием по времени. Основное их различие заключается в начальных значениях приращений адресов и индексов, а также в направлении их изменения (увеличение или уменьшение). При этом в дополнительном алгоритме изменяется также порядок следования поворачивающих множителей с прямого на двоично-инверсный, что незначительно усложняет алгоритм.

При частоте используются такие же константы и переменные, как и при программировании алгоритма БПФ с прореживанием по времени.

Для получения поворачивающих множителей используется таблица одного периода синуса длиной **N** отсчетов. Реально необходимо только три четверти таблицы, так как необходимы только поворачивающие множители на половине периода. Поэтому первая и вторая четверти таблицы используются для получения синуса, а вторая и третья четверти – косинуса. Для перехода от синуса к косинусу и необходимо содержимое ячейки – **QUARTN** (четверть **N**).

Поскольку входные и выходные данные БПФ для удобства обработки должны располагаться в обычном порядке, то в зависимости от алгоритма перед вычислением БПФ или после вычисления необходимо выполнить двоично-инверсную перестановку данных.

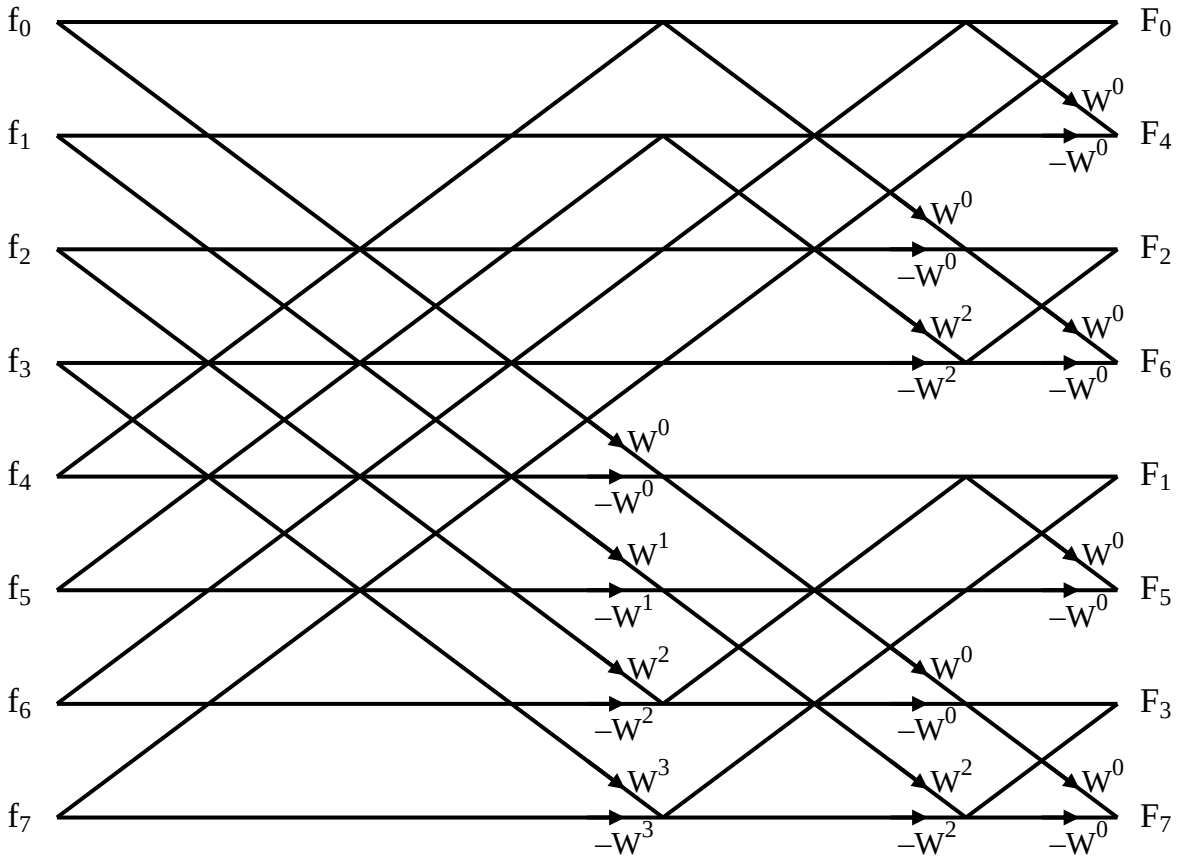

Рис. 3.1. Граф-схема БПФ с прореживанием по частоте
и прямым порядком входных данных

Таблица 3.1

Изменение индексов для БПФ с прореживанием по частоте
и прямым порядком входных данных

№ п/п	N2	IE	J	I	I + N2	K	Цикл	Переход
1	4	1	0	0	4	0	2	
2			1	1	5	1	2	
3			2	2	6	2	2	
4			3	3	7	3	2	3
5	2	2	0	0	2	0	1	
6				4	6		1	2
7			1	1	3	2	1	
8				5	7		1	3
9	1	4	0	0	1	0	1	
10				2	3		1	
11				4	5		1	
12				6	7		1	

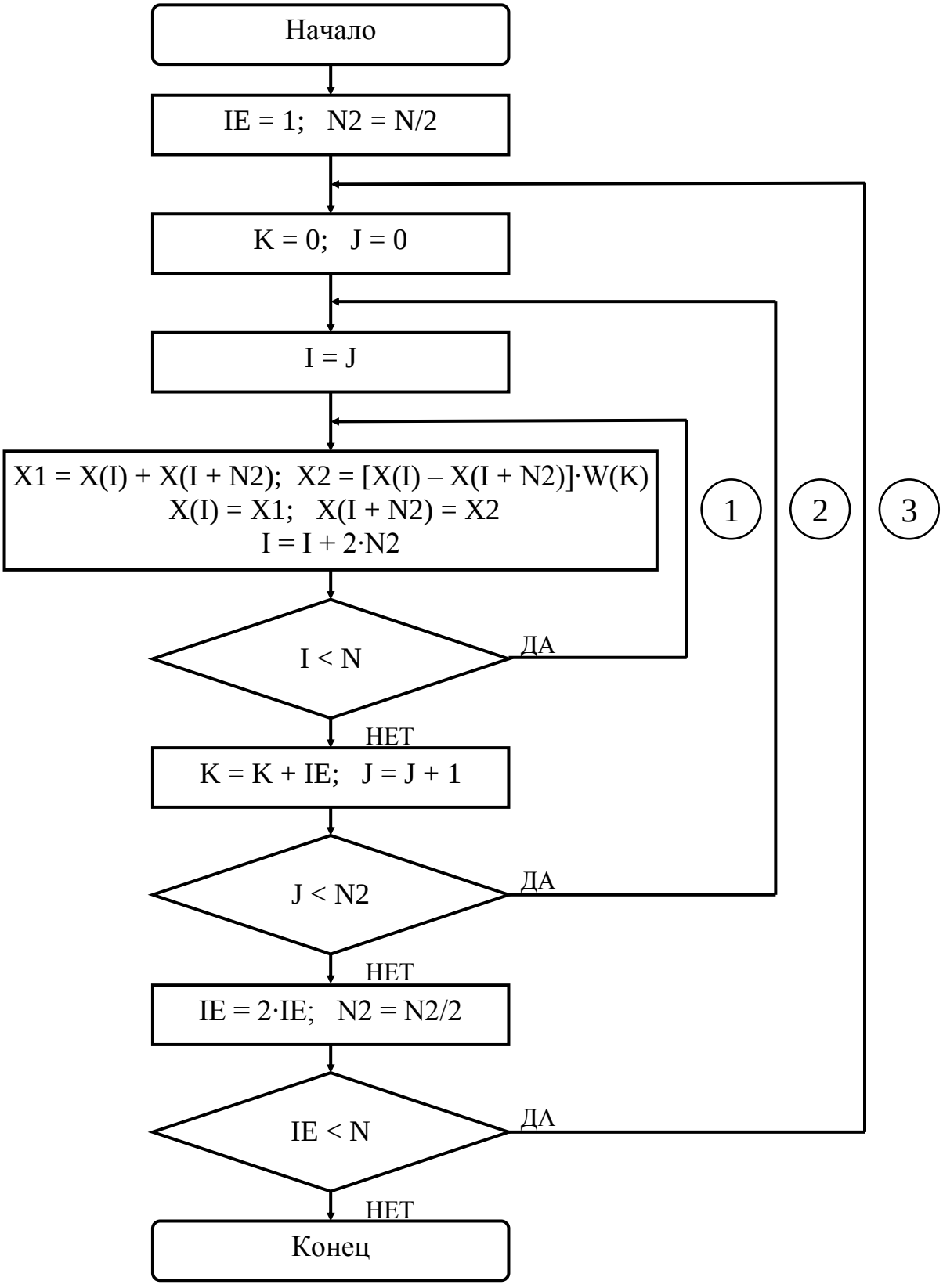

Рис. 3.2. Схема алгоритма БПФ с прореживанием по частоте
и прямым порядком входных данных

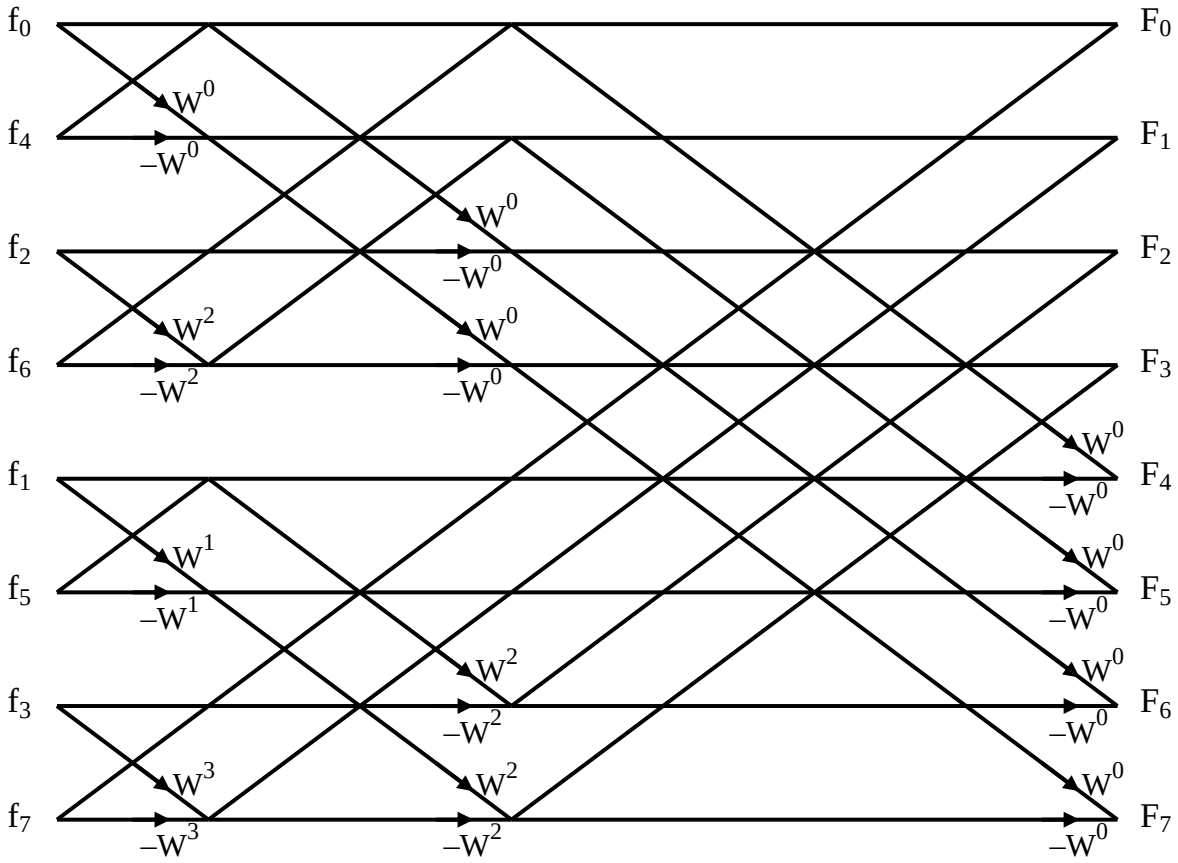

Рис. 3.3. Граф-схема БПФ с прореживанием по частоте
и двоично-инверсным порядком входных данных

Таблица 3.2

Изменение индексов для БПФ с прореживанием по частоте
и двоично-инверсным порядком входных данных

№ п/п	N2	IE	J	I	I + N2	K	Цикл	Переход
1	1	2	0	0	1	0	2	
2			2	2	3	2	2	
3			4	4	5	1	2	
4			6	6	7	3	2	3
5	2		0	0	2	0	1	
6				1	3		1	2
7			4	4	6	2	1	
8				5	7		1	3
9	4		0	0	4	0	1	
10				1	5		1	
11				2	6		1	
12				3	7		1	

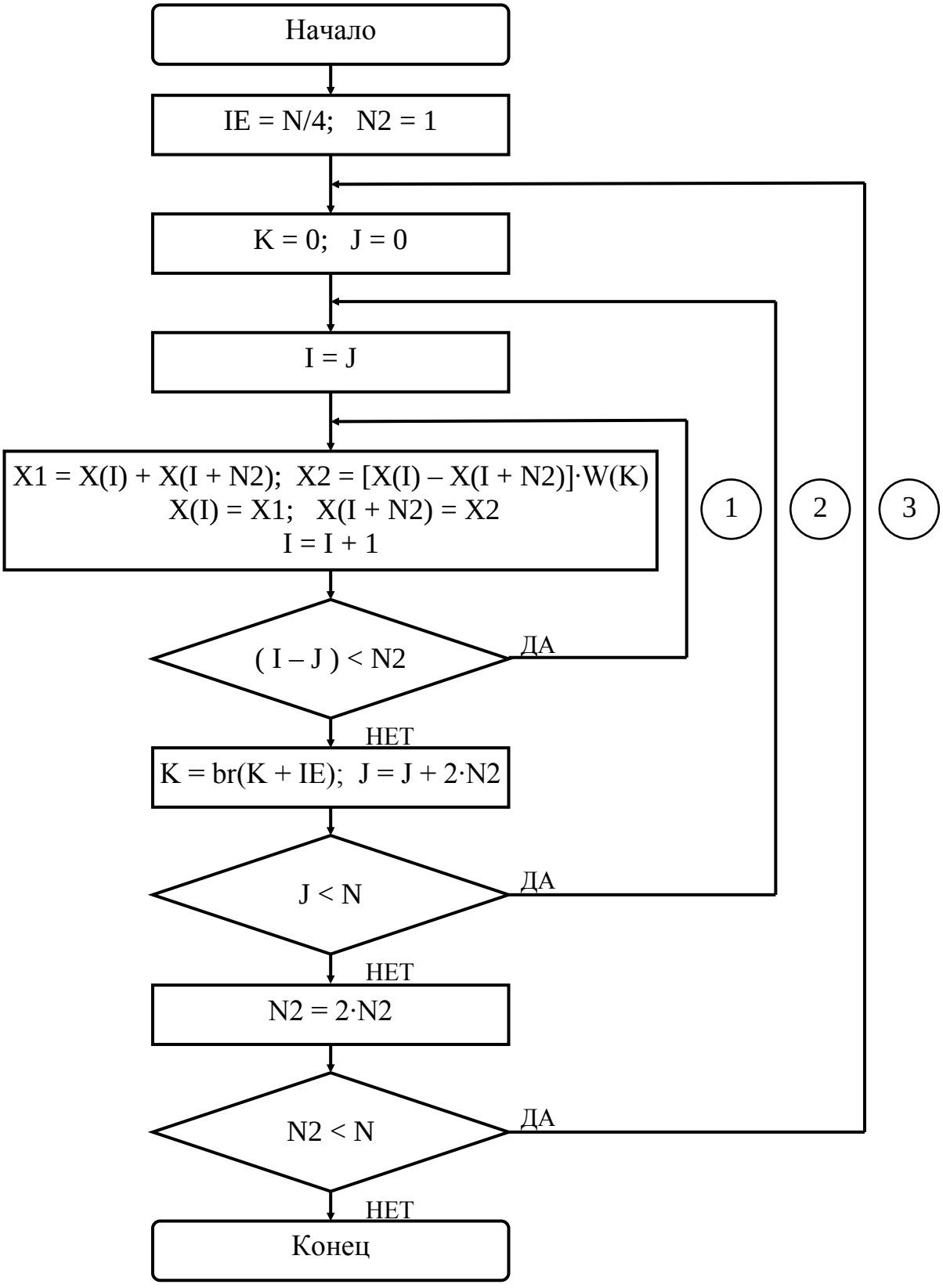

Рис. 3.4. Схема алгоритма БПФ с прореживанием по частоте и двоично-инверсным порядком входных данных

3.1.2. Проверка программ быстрого преобразования Фурье на простых сигналах

Для проверки правильности работы программ БПФ можно использовать различные сигналы – импульс, ступенька, синусоида. Поскольку такие сигналы легко формировать, а их изображение Фурье также простое и известное, то с их помощью легко проверяется правильность работы программы БПФ.

Так, изображением импульсного во временной области сигнала в частотной области будет прямая линия, так как это широкополосный сигнал, состоящий из суммы всех частот от 0 до частоты Найквиста.

Для постоянного сигнала (прямая линия во временной области) в частотной области изображение будет в виде импульса, т. е. постоянная составляющая и никаких частот.

И, наконец, для синусоидального сигнала частотой

$$F_{\text{с}} = F_{\text{д}} \cdot k/N, \qquad (3.1)$$

где $F_{\text{с}}$ – частота сигнала; $F_{\text{д}}$ – частота дискретизации; k – номер гармоники; N – размер выборки для БПФ, изображение будет содержать значения на соответствующей частоте и нули на остальных частотах. При этом соотношение между действительной и мнимой частями изображения будет определяться фазой анализируемого сигнала. Например, для синуса будет только мнимая часть, а для косинуса – только действительная.

Если частота сигнала не равна (3.1), то энергия такого сигнала размывается между ближайшими частотными отсчетами, и вместо одного отсчета получается несколько отсчетов с постепенно убывающей амплитудой.

Для получения обратного БПФ достаточно только поменять знак у функции синуса.

3.2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретические сведения по теме лабораторной работы (подразд. 3.1).
2. Получить у преподавателя задание для выполнения практической части работы.
3. Согласно заданию написать, оттранслировать и выполнить программу.
4. Продемонстрировать результат трансляции и работы программы преподавателю.
5. Оформить и защитить отчет по лабораторной работе.

3.3. СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы и исходные данные.
2. Описание алгоритма работы программы.
3. Текст программы с комментариями.
4. Выводы по работе.

ЛАБОРАТОРНАЯ РАБОТА №4

ИЗМЕРЕНИЕ АМПЛИТУДНО-ЧАСТОТНОЙ ХАРАКТЕРИСТИКИ ЦИФРОВОГО РЕКУРСИВНОГО ФИЛЬТРА

ЦЕЛЬ РАБОТЫ

Разработка алгоритма и написание программы для измерения амплитудно-частотной характеристики (АЧХ) цифрового рекурсивного фильтра с использованием БПФ на ассемблере процессора TMS320VC5402 и ее отладка на лабораторном макете TMS320VC5402 DSP Starter Kit (DSK).

4.1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

4.1.1. Алгоритм измерения амплитудно-частотной характеристики фильтра

Для измерения АЧХ фильтра может быть использован один из генераторов синусоидальных сигналов, разработанный в [1]. На вход фильтра подается сигнал, представляющий собой сумму синусоид с возрастающей частотой (от минимальной до частоты Найквиста). При этом для исключения резкого скачка сигнала необходимо суммировать синусоиды с различной фазой. Наиболее простой способ – сдвигать очередной синус на 1–2 отсчета.

По истечении времени, необходимого на установку сигнала на выходе фильтра (не менее двух периодов первой гармоники), следует измерить мощность сигнала на каждой частоте, используя программу БПФ из лабораторной работы №2 или №3. Мощность сигнала на соответствующей частоте определяется следующим выражением:

$$E(k) = \mathrm{Re}^2(k) + \mathrm{Im}^2(k), \qquad (4.1)$$

где $E(k)$ – мощность сигнала k-й гармоники; $\mathrm{Re}(k)$ – действительная часть изображения Фурье (после БПФ) k-й гармоники; $\mathrm{Im}(k)$ – мнимая часть изображения Фурье (после БПФ) k-й гармоники; k – номер гармоники от 0 до частоты Найквиста.

Для нахождения АЧХ фильтра необходимо найти отношение амплитуды синусоидального сигнала на выходе фильтра к амплитуде на входе фильтра на каждой частоте. Зная мощность сигнала, можно найти амплитуду выходного сигнала из следующего соотношения:

$$E = \frac{A^2}{2}, \qquad (4.2)$$

где E – мощность сигнала, A – амплитуда синусоидального сигнала.

Взяв для исключения переполнения амплитуды синусоидальных составляющих входного сигнала, равные $\mathbf{2}^{-\boldsymbol{M}}$, можно получить значение АЧХ фильтра, сдвинув амплитуды выходного сигнала на $\boldsymbol{M}$ разрядов влево.

4.1.2. Программа измерения амплитудно-частотной характеристики рекурсивного фильтра

Для измерения АЧХ фильтра используются программы генераторов синусоидальных сигналов из лабораторной работы №1 или №2 и программа цифрового рекурсивного фильтра из лабораторной работы №3, описанные в [1].

После обнуления буфера входного сигнала необходимо сформировать $\boldsymbol{N + 2}$ отсчетов синуса первой гармоники и добавить их в буфер, затем сформировать и прибавить отсчеты второй и всех последующих гармоник. Полученный таким образом полигармонический сигнал необходимо подать на вход фильтра в течение не менее $\boldsymbol{2N}$ тактов и сохранить сигнал с выхода фильтра в выходном буфере.

После этого необходимо выполнить *N*-точечное БПФ над второй половиной выходного буфера и, используя выражения (4.1) и (4.2), определить амплитуды всех гармоник на выходе фильтра.

Записав все полученные амплитуды в буфер АЧХ и отобразив этот буфер в графическом виде, увидим на экране монитора АЧХ рассматриваемого фильтра.

Для вычисления квадратного корня $\boldsymbol{X}$ из некоторого числа $\boldsymbol{Y}$ можно воспользоваться следующим алгоритмом:

1. Взять в качестве первого приближения квадратного корня $\boldsymbol{X_1 = 2^{-1}}$.
2. Возвести текущее приближение квадратного корня $\boldsymbol{X_i}$ в квадрат.
3. Сравнить полученное значение $\boldsymbol{X_i^2}$ со значением $\boldsymbol{Y}$.
4. Если значение $\boldsymbol{X_i^2 > Y}$, то вычесть из $\boldsymbol{X_i}$ значение $\boldsymbol{2^{-i}}$.
5. Вычислить очередную степень двойки $\boldsymbol{2^{-(i+1)}}$.
6. Получить очередное приближение квадратного корня $\boldsymbol{X_{i+1}}$, прибавив к $\boldsymbol{X_i}$ значение $\boldsymbol{2^{-(i+1)}}$.
7. Повторить процедуру начиная с п. 2 еще четырнадцать раз для получения всех пятнадцати разрядов значения квадратного корня $\boldsymbol{X}$.

Данный алгоритм может быть легко реализован с использованием арифметического устройства вспомогательных регистров и соответствующих способов модификации косвенного адреса.

Для этого необходимо во вспомогательный регистр, например **AR1**, поместить первое приближение квадратного корня $\boldsymbol{2^{-1}}$. Это же значение помещается и во вспомогательный регистр **AR0** в качестве текущей степени двойки.

Для реализации операции вычитания из $\boldsymbol{X_i}$ значения $\boldsymbol{2^{-i}}$ (п. 4) воспользуемся способом модификации косвенного адреса ***AR1-0**. Получение очередной степени двойки $\boldsymbol{2^{-(i+1)}}$ (п. 5) выполняется способом модификации косвенного адреса ***AR0+0B** (двоично-инверсная адресация или сложение с обратным переносом). Реализация операции сложения $\boldsymbol{X_i}$ со значением $\boldsymbol{2^{-(i+1)}}$ (п. 6) выполняется способом модификации косвенного адреса ***AR1+0**.

Общая схема алгоритма программы измерения АЧХ фильтра представлена на рис. 4.1.

Рис. 4.1. Схема алгоритма измерения АЧХ фильтра

При обработке реальных физических сигналов они заносятся в действительные части комплексных чисел временного процесса, а мнимые части при этом обнуляются. Но с помощью одной программы БПФ можно обработать сразу два действительных сигнала. Для этого необходимо использовать эти два действительных сигнала для задания действительной и мнимой частей временного процесса. После вычисления БПФ необходимо произвести разделение полученного изображения комплексного процесса на два изображения действительных процессов в соответствии с формулами

$$\begin{cases} \mathrm{Re}[X_1(k)] = \dfrac{\mathrm{Re}[U(k)] + \mathrm{Re}\,U(N-k)}{2}, \\ \mathrm{Im}[X_1(k)] = \dfrac{\mathrm{Im}[U(k)] - \mathrm{Im}\,U(N-k)}{2}, \\ \mathrm{Re}[X_2(k)] = \dfrac{\mathrm{Im}[U(k)] + \mathrm{Im}\,U(N-k)}{2}, \\ \mathrm{Im}[X_2(k)] = \dfrac{-\mathrm{Re}[U(k)] + \mathrm{Re}\,U(N-k)}{2}, \; k = \overline{0,\, N/2-1}. \end{cases} \tag{4.3}$$

где $\boldsymbol{X_1}$, $\boldsymbol{X_2}$ – изображения первого и второго действительных процессов; $\boldsymbol{U}$ – изображение комплексного процесса, полученное с помощью БПФ.

Преобразовав выражения (4.3) для расчета $\boldsymbol{U(k)}$ по изображениям двух действительных процессов $\boldsymbol{X_1}$, $\boldsymbol{X_2}$, можно вычислить обратное БПФ и получить временные реализации двух независимых действительных сигналов.

4.2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретические сведения по теме лабораторной работы (подразд. 4.1).
2. Получить у преподавателя задание для выполнения практической части работы.
3. Согласно заданию написать, оттранслировать и выполнить программу.
4. Продемонстрировать результат трансляции и работы программы преподавателю.
5. Оформить и защитить отчет по лабораторной работе.

4.3. СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы и исходные данные.
2. Описание алгоритма работы программы.
3. Листинг программы с комментариями.
4. Выводы по работе.

ПРИЛОЖЕНИЕ 1

Установка и настройка системы программирования Code Composer Studio

После установки с диска системы программирования и отладки Code Composer Studio v.2.1 (CCS) необходимо запустить программу настройки, используя ярлык «Setup CCS 2 ('C5000)» на рабочем столе, и выполнить следующие действия:

1. Удалить (Remove) установленную по умолчанию конфигурацию «C55x Simulator (Texas Instruments)».

2. Добавить (Add to System…) необходимую конфигурацию «C54x Simulator (Texas Instruments)» или «C54x Parallel Port (Texas Instruments)» при наличии отладочного модуля.

3. Последовательно, переходя от вкладки к вкладке, в окне свойств выбрать конфигурационный файл «sim5402.cfg», включить предупреждение о конфликтах в конвейере «Display Pipeline Warning – Enable», выбрать файл инициализации ЦПУ «c5402.gel».

4. Закончить настройку, сохранить новую конфигурацию и запустить программу CCS.

Настройка системы CCS выполняется однократно и в дальнейшем ее запуск выполняется с использованием ярлыка «CCS 2 ('C5000)» на рабочем столе.

При первом запуске рекомендуется выполнить некоторые дополнительные настройки системы.

1. В меню выбрать команду **Option | Customize…** .

2. В открывшемся окне «Customize» на вкладке «Editor Properties» установить шаг табуляции (Tab stops) – 8 (по умолчанию он равен 4), что позволяет использовать метки и символические имена длиной до семи символов без нарушения общего вида программы из четырех колонок: метки и имена, команды и директивы, операнды и параметры, комментарии.

3. На вкладке «Program Load Options» установить флажок автоматической загрузки программы после сборки (Load Program After Build), что избавит от необходимости ручной загрузки программы после каждой трансляции и сборки.

4. При необходимости можно произвести и другие изменения настроек, например, поменять цвета, задать каталоги для поиска файлов и т. д.

ПРИЛОЖЕНИЕ 2

Создание проекта и отладка программ в системе Code Composer Studio

После завершения настройки системы программирования и отладки Code Composer Studio можно переходить к созданию проекта, написанию и отладке программы. Система CCS является обычной многооконной системой с интуитивно понятными меню и по своим структуре и возможностям напоминает системы программирования на языках высокого уровня С, Паскаль и т. п. При этом следует отметить, что данная система позволяет программировать не только на языке ассемблера для цифровых процессоров обработки сигналов фирмы Texas Instruments платформы С5000 (обычном и алгебраическом), но и на языке С. Разработка и отладка программы начинается с создания проекта. Список всех открытых проектов отображается в окне «Project», которое автоматически открывается при запуске CCS.

Для создания проекта необходимо выполнить следующие действия:

1. В меню выбрать команду **Project | New…** .
2. В открывшемся окне «Project Creation» выбрать папку для сохранения проекта (Location) и задать имя проекта (Project Name), при этом в выбранной папке автоматически создается папка с именем проекта и файлом описания проекта с расширением pjt (не рекомендуется для проекта, файлов и папок использовать длинные имена и имена с русскими буквами).
3. В окне списка проектов появляется созданный проект с набором компонентов: DSP/BIOS Config, Generated Files, Include, Libraries, Source (исходные файлы программ).
4. Для хранения исходных текстов программ средствами ОС в папке проекта создать отдельную папку с именем Source, Src или аналогичным.
5. При работе с несколькими проектами можно создать отдельную папку для хранения различных общих таблиц и подпрограмм с именем Include, Incl или аналогичным.
6. В меню выбрать команды **File | New ▸ | Source File** для создания файла с исходным текстом программы, а если такой файл уже был набран ранее в каком-либо редакторе, то скопировать его в папку Source.
7. В открывшемся пустом окне с именем Untitled1 ввести исходный текст программы.
8. Сохранить файл с исходным текстом программы в папке Source, заменив имя файла Untitled1 на свое и выбрав тип файла Assembly Source Files (*.asm).
9. Подключить файл с исходным текстом программы к проекту, выбрав в меню команду **Project | Add Files to Project…**, и в открывшемся окне выбрать тип файла Asm Source Files (*.a*;*.s*).
10. После подключения файла его имя появится в компоненте проекта Source.
11. Для того чтобы использовать подключаемые файлы из папки include, необходимо указать в настройках проекта путь к этим файлам относительно

папки с исходным текстом, выбрав команду **Project | Build Options…**, и в открывшемся окне на вкладке «Compiler» в категории (Category:) Preprocessor в поле Include Search Path (-i): указать путь (например, если исходный текст программы находится в папке myprojects\ppovsrv\source, а подключаемые файлы в папке myprojects\include, то необходимо указать путь ..\..\include).

12. В этом же окне настроек проекта (Build Options for имя_проекта.pjt) в категории Basic в поле Processor Version (-v): можно задать версию процессора (5402), в категории Assembly установить флажок Generate Assembly Listing Files (-al) для вывода листинга программы при трансляции и в категории Diagnostics установить флажок Warn on Pipeline Conflict для выдачи предупреждений о конфликтах в конвейере команд.

13. Если проект содержит несколько исходных файлов, то путь к папке с подключаемыми файлами можно задать для каждого из исходных файлов отдельно, используя команду **Project | File Specific Options…**, при этом заданные в открывшемся окне настроек файла (Build Options for имя_файла.asm) параметры имеют более высокий приоритет.

14. Для удобства работы с программой CCS рекомендуется создать так называемое рабочее пространство (Workspace), для чего необходимо открыть ряд окон в дополнение к окну с исходным текстом программы.

15. С помощью команды **View | CPU Registers ▸ | CPU Registers** открыть окно с регистрами процессора, такими, как аккумуляторы, регистры состояния, вспомогательные регистры и т. д.

16. С помощью команды **View | Disassembly** открыть окно с машинными кодами отлаживаемой программы для контроля хода выполнения программы.

17. С помощью команды **View | Memory…** открыть окно с содержимым ячеек памяти данных для контроля изменения переменных, при этом в открывшемся окне настроек (Memory Window Options) в поле Addres: необходимо задать адрес первой из отображаемых ячеек памяти, а в полях Title:, Q-Value:, Format: и Page: можно дополнительно задать имя окна, число разрядов дробной части, формат отображения данных (десятичные, шестнадцатеричные, двоичные, знаковые, беззнаковые 16-разрядные, 32-разрядные) и отображаемое пространство памяти (данных, программ, ввода-вывода).

18. С помощью команды **View | Graph ▸ | Time/Frequency…** открыть окно, отображающее массив с содержимым ячеек памяти данных, отображаемых в виде графика, при этом в открывшемся окне настроек (Graph Property Dialog) в поле Start Address: необходимо задать адрес первой из отображаемых ячеек памяти, в полях Acquisition Buffer Size: и Display Data Size: – длину отображаемого буфера данных, в поле DSP Data Type: – формат отображаемых данных (обычно 16-bit signed integer), а в полях Display Type, Graph Title, Index Increment, Q-Value, Autoscale и Magnitude Display Scale можно дополнительно задать тип графика (например двух переменных), имя окна, шаг выборки данных (например два), число разрядов дробной части, автоматический выбор масштаба и линейный или логарифмический масштаб отображения данных.

19. При необходимости открыть дополнительные окна для отображения данных в числовом и графическом виде.

20. Для увеличения свободного места в окне программы можно перенести панель инструментов отладки (Debug Toolbar – слева) и дополнительные панели (Plug-in Toolbars – третья строка) в первую и вторую строки панелей инструментов, закрыть окно проекта (Project). Получим вид окна программы CCS, показанный на рис. П.2.1.

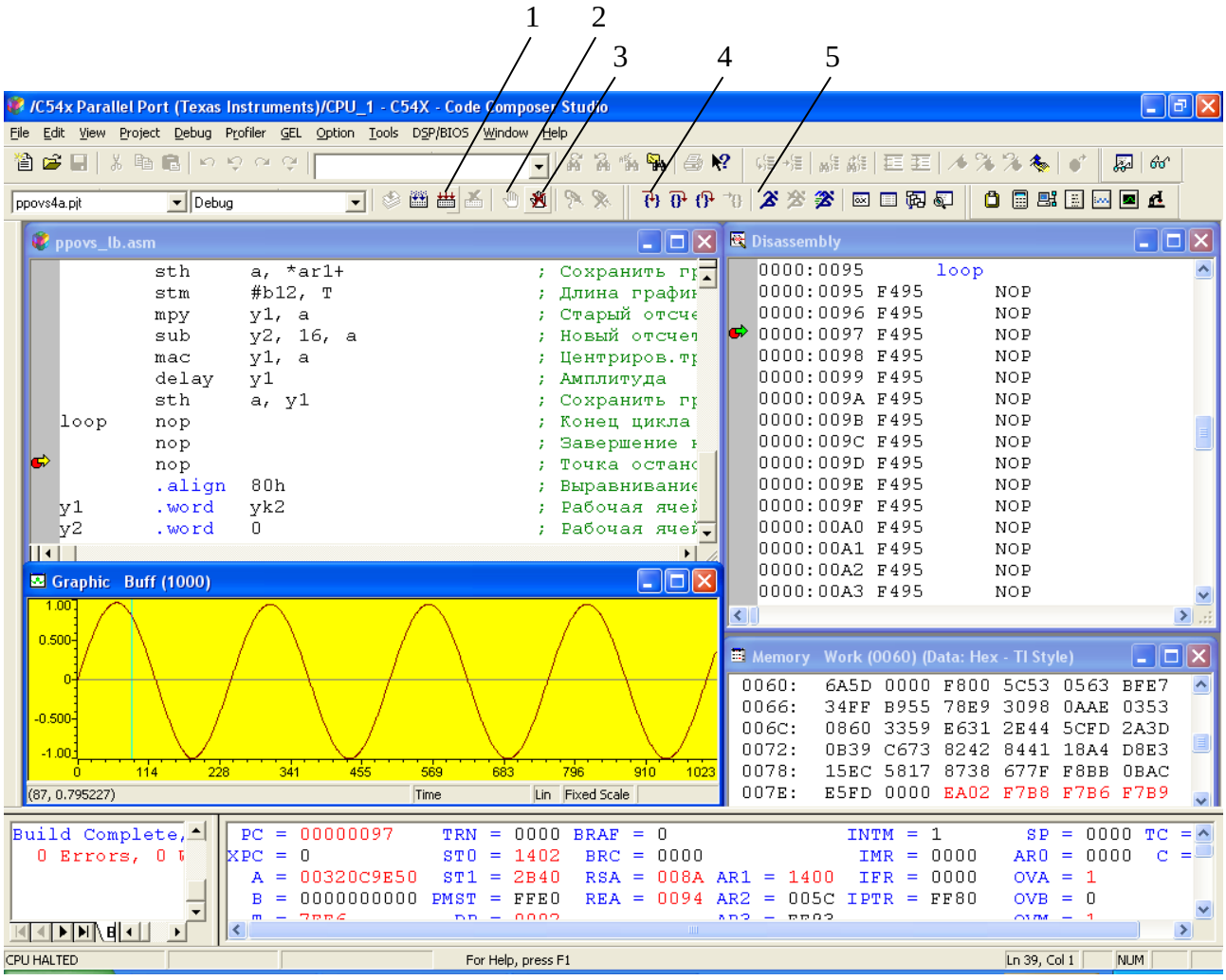

Рис. П.2.1. Примерный вид окна программы CCS

21. Созданное таким образом рабочее пространство необходимо сохранить на диске в папке проекта, используя для этого команду **File | Workspace ▸ | Save Workspace As…** .

22. При следующем запуске программы не нужно будет повторять все ранее перечисленные действия, а достаточно только выполнить команду **File | Recent Workspaces ▸**, и проект вместе с созданными ранее окнами появится на экране.

23. Если нужного рабочего пространства не окажется в списке, то необходимо выполнить команду **File | Workspace ▸ | Load Workspace…** и загрузить рабочее пространство из папки проекта.

24. Создав проект и рабочее пространство, можно переходить к вводу исходного текста программы.

25. Введенную программу необходимо откомпилировать и собрать, для чего можно воспользоваться кнопкой «Rebuild All» (см. рис. П.2.1, позиция 1).

26. Результаты трансляции отображаются в окне «Output» и не должны содержать сообщений об ошибках (допускается только одно предупреждение при отсутствии командного файла).

27. При наличии ошибок и предупреждений необходимо внести исправления в исходный текст программы и повторить процесс компиляции.

28. При отсутствии ошибок откомпилированная и собранная программа должна загрузиться, а ее код появиться в окне дизассемблера.

29. Перед запуском программы необходимо установить в конце программы точку останова кнопкой «Toggle breakpoint» на панели инструментов (см. рис. П.2.1, позиция 2), при этом в конце программы обычно добавляются две или три пустые команды (NOP) для завершения операций в конвейере.

30. Для быстрого сброса всех точек останова используется кнопка «Remove all breakpoints» на панели инструментов (см. рис. П.2.1, позиция 3).

31. Точки останова обозначаются красными точками на сером фоне слева от текста программы и могут быть быстро установлены и сброшены двойным щелчком левой кнопки мыши на сером фоне слева от соответствующей строки программы.

32. Для пошагового выполнения программы используется кнопка «Single Step» (см. рис. П.2.1, позиция 4), а следующие за ней кнопки «Step Over», «Step Out» и «Run to Cursor» позволяют выполнить команду вызова подпрограммы, не входя внутрь подпрограммы, выполнить ряд команд до возврата из подпрограммы или до строки программы, в которой в настоящее время находится курсор.

33. Для выполнения программы в автоматическом режиме используется кнопка «Run» (см. рис. П.2.1, позиция 5), а для останова программы – следующая за ней кнопка «Halt».

34. При выполнении программы все изменяющиеся данные в окне регистров процессора и окне данных отображаются красным цветом.

35. В процессе выполнения программы можно легко изменять содержимое ячеек памяти и регистров, для чего достаточно дважды щелкнуть левой кнопкой мыши на соответствующей ячейке, регистре или бите состояния и ввести новое значение (шестнадцатеричные числа отображаются в формате, принятом в языке C, – 0x0000).

36. Для заполнения последовательности ячеек памяти одинаковыми значениями можно использовать команду **Edit | Memory ▸ | Fill…** и в открывшемся окне настроек (Setup Filling Memory) в поле Address: задать адрес первой из заполняемых ячеек памяти, в поле Length: – количество заполняемых ячеек, в поле Fill Pattern: – записываемые данные, а в поле Memory Type: – выбрать тип заполняемой памяти (Data, Program или I/O).

37. Система CCS позволяет эмулировать выполнение команд ввода-вывода, используя файлы данных специального формата.

38. Для эмуляции ВЫВОДА необходимо установить контрольную точку (см. рис. П.2.1, кнопка «Toggle Probe Point» справа от позиции 3) после команды вывода, которая обозначается голубым ромбом на сером фоне слева от текста программы.

39. Задать выходной файл командой **File | File I/O…** и на вкладке «File Output» добавить файл, нажав кнопку «Add File», выбрав папку, тип файла и введя его имя.

40. На этой же вкладке выбрать страницу в поле Page: (Data, I/O), задать адрес порта Address: и длину данных Length: (число слов, выводимых за один раз – обычно 1).

41. Связать выбранный файл с контрольной точкой, нажав кнопку «Add Probe Point».

42. В открывшемся окне «Break/Probe Points» на вкладке «Probe Points» выбрать контрольную точку в поле Probe Point:, выбрать файл в поле Connect To: и соединить контрольную точку с файлом, нажав кнопку «Replace».

43. Не связанная контрольная точка (No Connection) будет связана с выбранным файлом.

44. Для эмуляции ВВОДА необходимо установить контрольную точку перед командой ввода (для последовательного порта за 2 команды, т. е. +1 такт).

45. Выбрать входной файл командой **File | File I/O…** и на вкладке «File Input» выбрать файл, нажав кнопку «Add File».

46. На этой же вкладке выбрать страницу в поле Page: (Data, I/O), задать адрес порта Address: и длину данных Length: (число слов, выводимых за один раз – обычно 1) и установить флажок Wrap Around – повторять с начала при перезапуске и достижении конца файла.

47. Связать выбранный файл с контрольной точкой.

ПРИЛОЖЕНИЕ 3

Директивы ассемблера для процессора TMS320VC5420

При написании программ на ассемблере кроме команд процессора, приведенных в прил. 3 [1], используются специальные директивы ассемблера. Эти директивы позволяют определять переменные и константы, содержимое ячеек памяти, различные части памяти программ и данных, режимы и условия трансляции программы и т. п. Ниже приведены основные директивы ассемблера, сгруппированные по назначению.

1. Директивы определения переменных и констант:

а) директива **.set** – присвоить переменной числовое значение,
формат: ***имя переменной .set значение*** ;

б) директива **.asg** – присвоить символьное значение переменной,
формат: ***.asg ["]символьная строка["], имя переменной*** ;

в) директива **.eval** – вычислить и присвоить значение переменной,
формат: ***.eval выражение, имя переменной*** ;

г) директива **.mmregs** – определить адреса регистров процессора и некоторых периферийных устройств, отображаемых на память.

Поскольку директива **.asg** присваивает символьное значение, то конечный результат зависит от порядка переменных в выражении. Для получения однозначного результата можно использовать скобки. При этом директива **.asg** является единственной директивой, которая может ссылаться вперед.

Директива **.set(.equ**) в отличие от директив **.asg** и **.eval** не может переприсвоить значение переменной, определенной ранее.

Директива **.eval** обычно используется при повторении команд и директив с изменяющимися параметрами при использовании директив **.loop** и **.endloop**.

Пример фрагмента программы, использующий данные директивы, приведен ниже.

```
        .def    _c_int00        ; метка начала программы
x1      .set    10h             ; x1 = 10h
x2      .set    x1+8h           ; x2 = 10h+8h  = 18h
        .asg    x1+8h, x3       ; x3 = x1+8h   = 18h
        .asg    (x1+8h), x4     ; x4 = (x1+8h) = 18h
;y1     .set    Smem            ; недопустимая ссылка вперед
;       .eval   Smem, y2        ; недопустимая ссылка вперед
        .asg    Smem, y3        ; допустимая ссылка вперед
        .text                   ; секция программы
_c_int00:                       ; начало программы
        ld      #Smem, DP       ; текущая страница данных
        st      #2*x2, Smem     ; сохранить 2*18h     = 30h
        st      #x2*2, Smem+1   ; сохранить 18h*2     = 30h
        st      #2*x3, Smem+2   ; сохранить 2*x1+8h   = 28h
        st      #x3*2, Smem+3   ; сохранить x1+8h*2   = 20h
        st      #2*x4, Smem+4   ; сохранить 2*(x1+8h) = 30h
```

```
        st      #x4*2, Smem+5   ; сохранить (x1+8h)*2 = 30h
        stm     #y3, ar3        ; сохранить в регистр  Smem
        nop                     ; точка останова
        .data                   ; секция данных
        .align  80h             ; выровнять на начало страницы
        .eval   0, x            ; присвоить начальное значение
Smem                            ; массив данных
        .loop   8               ; повторить директивы
        .word   x               ; зарезервировать слово
        .eval   x+1, x          ; увеличить значение
        .endloop                ; конец повторяемого блока
        .end                    ; конец программы
```

2. Директивы инициализации ячеек памяти:

а) директива **.word** – преобразовать значение к 16 битам,
формат: ***.word значение1[, значение2, ..., значениеN]*** ;

б) директива **.byte** – преобразовать значение к 8 битам,
формат: ***.byte значение1[, значение2, ..., значениеN]*** ;

в) директива **.long** – преобразовать значение к 32 битам,
формат: ***.long значение1[, значение2, ..., значениеN]*** .

Директива **.word** (**.int**, **.half**, **.short**, **.uword**, **.uint**, **.uhalf**, **.ushort**) преобразует заданные значения в формат 16-битного слова, и если значение выходит за его пределы, то выдается предупреждение, а значение усекается.

Директива **.byte** (**.char**, **.ubyte**, **.uchar**) преобразует заданные значения в формат 16-битного слова до 8 значащих битов и если значение выходит за пределы 8 битов, то выдается предупреждение, а значение усекается.

Директива **.long** (**.ulong**) преобразует заданные значения в формат 32-битного (двойного) слова. Буква **u** в директивах обозначает преобразование беззнакового (т. е. положительного) значения.

В качестве значения может использоваться строковая переменная, заключенная в одинарные (') или двойные (") кавычки. При этом строковая переменная, заключенная в двойные кавычки, преобразуется в последовательность слов (по одному символу в слове), а заключенная в одинарные кавычки усекается до слова или байта.

Числовые значения могут задаваться в различных системах счисления: десятичной (по умолчанию), шестнадцатеричной (символ **h** в конце), восьмеричной (**q**) и двоичной (**b**).

При задании переменных допускается использовать символы арифметических, логических операций и сдвига: **+**, **-**, *****, **/** – арифметические операции сложения, вычитания, умножения и деления соответственно; **&**, **|**, **~** – логические операции И, ИЛИ, НЕ; **<<**, **>>** – сдвиг влево и вправо (число после символов указывает количество разрядов сдвига, а знак минус перед ним меняет направление сдвига). Все вышеперечисленные операции имеют приоритет: наивысший – умножение и деление, затем сложение и вычитание, потом сдвиги и самый низкий приоритет – логические операции. Для получения однозначного результата можно использовать круглые скобки.

Фрагмент листинга трансляции отдельных директив инициализации ячеек памяти с различными форматами значений и операциями приведен ниже.

```
000000          _c_int00:
000000 EA01"    ld      #X, DP   ; Текущая страница
000001 F495     nop              ; Точка останова
                .align 80h       ; Выравнивание на страницу

    .word  10, -1, "ABCD", 40000, 'abcd', 65535, X, _c_int00,
           0+'ABCD', 0+"abcd" , 01010101b, 555q, 555h
000080 000A, FFFF, 0041 0042 0043 0044, 9C40, 6162, FFFF,
000089 0001", 0000', 4142, 6364, 0055, 016D, 0555

    .int   10, -1, "ABCD", 40000, 'abcd', 65535, X, _c_int00,
           0+'ABCD', 0+"ab" , 01010101b, 555q, 555h
000090 000A, FFFF, 0041 0042 0043 0044, 9C40, 6162, FFFF,
000099 0001", 0000', 4142, 6162, 0055, 016D, 0555

    .byte  10, -1, "ABCD", 200, 'abcd', 255, X, _c_int00,
           0+'ABCD', 0+"abcd" , 01010101b, 55q, 55h
0000A0 000A, 00FF, 0041 0042 0043 0044, 00C8, 0062, 00FF,
0000A9 0001, 0000, 0042, 0064, 0055, 002D, 0055

    .char  10, -1, "ABCD", 200, 'a', 255, X, _c_int00, 0+'A',
           0+"a" , 01010101b, 55q, 55h
0000B0 000A, 00FF, 0041 0042 0043 0044, 00C8, 0061, 00FF,
0000B9 0001, 0000, 0041, 0061, 0055, 002D, 0055

000000          .data
                .align 200h
000000 0000     .word  0
000001 0000 X   .word  0
```

3. Директивы подключения внешних файлов к программе и управления выводом листинга:

а) директива **.include** – включить содержимое файла в программу, формат: ***.include "имя файла"*** ;

б) директива **.copy** – копировать содержимое файла в программу, формат: ***.copy "имя файла"*** ;

в) директива **.list** – разрешить выдачу листинга программы;

г) директива **.nolist** – запретить выдачу листинга программы.

Директивы **.include** и **.copy** вставляют содержимое указанного файла в основной файл программы перед трансляцией программы. Таким образом, из нескольких файлов собирается один файл (при этом не требуется определения глобальных переменных), который и транслируется для получения объектного кода. Единственное различие этих директив в том, что включенный файл (директива **.include**) не распечатывается в листинге. Чаще всего директива **.include**

используется для подключения различных таблиц, а директива **.copy** – для подпрограмм.

Запретить вывод команд и директив в листинг можно с помощью директивы **.nolist**, а снова разрешить – директивой **.list**. Данные директивы игнорируются во включенном файле (директива **.include**).

4. Директивы определения секций и объединения объектных модулей:

а) директива **.sect –** задать имя инициализированной секции программы,
формат: ***.sect "секция[:подсекция]"*** ;

б) директива **.usect –** задать имя неинициализированной секции,
формат: ***метка .usect "секция[:подсек.]", слов[, [блок][, слово]]*** ;

в) директива **.def** – указать имена переменных, заданных в программе и используемых в других программах,
формат: ***.def переменная[, переменная]...*** **;**

г) директива **.ref** – указать имена переменных, используемых в программе и заданных в другой программе;
формат: ***.ref переменная[, переменная]...*** ;

д) директива **.global** – указать имена общих переменных (заменяет директивы **.def** и **.ref**),
формат: ***.global переменная[, переменная]...*** .

Имеется три стандартных имени секций, не требующих использования директивы **.sect**:

.text – секция программы (исполняемого кода);

.data – секция данных (таблицы, инициализированные переменные);

.bss – секция данных (неинициализированные переменные). Формат директивы ***.bss имя, слов [, [блок] [, слово]]***, где ***имя*** – имя первого слова секции, ***слов*** – число слов в секции, ***блок*** – выравнивание на границу блока, ***слово*** – выравнивание на границу двойного слова.

Если программа состоит из нескольких частей (файлов), то дополнительные файлы могут транслироваться отдельно (в этом случае необходимы директивы .def, .ref или .global). При этом в проект командой **Project | Add Files to Project…** включаются файлы с объектным кодом (расширение obj – оттранслированы заранее).

Файлы с объектным кодом могут быть подключены к проекту также с помощью командного файла, в котором задаются и физические адреса начала каждой секции. Порядок объединения файлов задается командой **Project | Build Options… | Link Order** или их порядком в командном файле.

Примечания:

1. Файлы, подключенные к проекту, объединяются первыми.
2. В командном файле различаются в имени файла большие и малые буквы и не допускаются имена файлов со знаком минус *"-"* (часть имени от знака минус игнорируется, так как этот знак является признаком режима).

ЛИТЕРАТУРА

1. Клюс, В.Б. Программирование проблемно-ориентированных вычислительных средств реального времени: лаб. практикум для студ. спец. 1-40 02 02 «Электронные вычислительные средства» днев. формы обуч. В 2 ч. Ч. 1 / В. Б. Клюс, М. В. Качинский, А. Б. Давыдов. – Минск : БГУИР : 2008. – 34 с. : ил.

2. Проектирование проблемно-ориентированных вычислительных средств: лаб. практикум для студ. спец. «Электронные вычислительные средства». В 2 ч. / А. А. Петровский [и др.]. – Минск : БГУИР. – Ч. 1, 1999. – 51 с. ; Ч. 2, 2001. – 46 с.

3. Петровский, А. А. Программирование цифровых фильтров на процессорах для работы в реальном времени : учеб.-метод. пособие по курсам «Проектирование проблемно-ориентированных вычислительных систем» и «Программирование проблемно-ориентированных ЭВС реального времени» для студ. спец. Т 08 02 00 «Проектирование и технология электронных вычислительных средств» / А. А. Петровский, В. Б. Клюс, В. В. Серков. – Минск : БГУИР, 1997. – 40 с.

4. TMS320C54x DSP. Reference Set. Vol. 2 / Texas Instruments. – 1990.

5. TMS320C54x DSKplus. User's Guide / Texas Instruments. – 1996.

6. Digital Signal Processing Applications with the TMS320 Family. Theory, Algorithms, and Implementations. Vol. 2 : Mnemonic Instruction Set / Texas Instruments. – 2001.

7. Code Composer Studio. Getting Started Guide / Texas Instruments. – 2001.

Св. план 2010, поз. 14

Учебное издание

Клюс Владимир Борисович
Качинский Михаил Вячеславович
Давыдов Александр Борисович

ПРОГРАММИРОВАНИЕ ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ ВЫЧИСЛИТЕЛЬНЫХ СРЕДСТВ РЕАЛЬНОГО ВРЕМЕНИ

Лабораторный практикум
для студентов специальности 1-40 02 02
«Электронные вычислительные средства»
дневной формы обучения

В 2-х частях

Часть 2

Редактор Е. Н. Батурчик
Корректор Л. А. Шичко
Компьютерная верстка А. В. Тюхай

Подписано в печать 05.01.2011	Формат 60x84 1/16.	Бумага офсетная.
Гарнитура «Таймс».	Печать ризографическая.	Усл. печ. л. 2,21
Уч.-изд. л. 2,1.	Тираж 100 экз.	Заказ 5.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ № 02330/00494371 от 16.03. 2009. ЛП № 02330/0494175 от 03.04. 2009.
220013, Минск, П. Бровки, 6.