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

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

ПРОЕКТИРОВАНИЕ ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ ВЫЧИСЛИТЕЛЬНЫХ СРЕДСТВ ЛАБОРАТОРНЫЙ ПРАКТИКУМ

*Рекомендовано УМО вузов Республики Беларусь
по образованию в области информатики и радиоэлектроники
в качестве учебно-методического пособия для студентов учреждений,
обеспечивающих получение высшего образования по специальности
«Электронные вычислительные средства»*

Минск БГУИР 2012

УДК 004.3´12(076.5)
ББК 32.973.26-018.2я73
К52

Р е ц е н з е н т ы:
кафедра информационных систем и технологий учреждения образования
«Белорусский государственный технологический университет»
протокол №9 от 07.04.2011;

заведующий кафедрой информатики учреждения образования
«Минский государственный высший радиотехнический колледж»,
кандидат технических наук, доцент
Ю. А. Скудняков

Клюс, В. Б.

К52 Проектирование проблемно-ориентированных вычислительных средств : лабораторный практикум : учеб.0метод. пособие / В. Б. Клюс, М. В. Качинский, А. Б. Давыдов. – Минск : БГУИР, 2012. – 68 с. : ил.
ISBN 978-985-488-766-1.

Практикум содержит описание восьми лабораторных работ, в которых рассматриваются различные режимы работы процессора, методы формирования сигналов различной формы и алгоритмы работы модема с частотной модуляцией. Приведены примеры реализации некоторых алгоритмов формирования сигналов и работы модулятора и демодулятора цифрового модема. Лабораторные работы ориентированы на использование отладочного модуля на базе процессора TMS320VC5402 и выполняются на ПЭВМ.

УДК 004. 3´12(076.5)
ББК 32.973.26-018.2я73

ISBN 978-985-488-766-1

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА №1

ВЫПОЛНЕНИЕ ОСНОВНЫХ КОМАНД ПРОЦЕССОРА TMS320VC5402 ПРИ РАЗНЫХ РЕЖИМАХ РАБОТЫ

ЦЕЛЬ РАБОТЫ:

Знакомство с системой программирования и отладки программ Code Composer Studio (прил. 1), режимами расширения знака и коррекции переполнения в арифметическо-логическом устройстве и их влиянием на результат выполнения различных команд процессора TMS320VC5402. Разработка алгоритма и программы для проверки влияния битов состояния процессора на результат выполнения некоторых команд на ассемблере процессора TMS320VC5402 и отладка программы на лабораторном макете TMS320VC5402 DSP Starter Kit.

1.1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1.1. Режим расширения знака SXM

Арифметическо-логическое устройство (АЛУ) и аккумуляторы в процессоре TMS320VC5402 имеют по 40 разрядов, а ячейки памяти данных – по 16 разрядов. Поэтому возникает вопрос о заполнении недостающих 24 разрядов при загрузке аккумуляторов и передаче данных на вход АЛУ из памяти данных. При этом следует учитывать, что данные при передаче могут быть сдвинуты влево и вправо. И если недостающие младшие разряды логично заполнить нулями, то заполнение старших разрядов не столь однозначно.

Для управления заполнением старших разрядов в регистре состояния процессора **ST1** (прил. 2) имеется специальный бит режима расширения знака **SXM** (sign extension mode). Именно этот бит и определяет правило заполнения недостающих старших разрядов. Если бит **SXM** = 0, то старшие разряды заполняются нулями, а если бит **SXM** = 1, то старшие разряды заполняются знаковым разрядом загружаемого числа. Таким образом, происходит расширение знакового (старшего) разряда числа, считанного из ячейки памяти данных, во все старшие разряды АЛУ или аккумулятора. Следовательно, если старший разряд ячейки памяти равен нулю, то недостающие старшие разряды АЛУ или аккумулятора всегда заполняются нулями (независимо от состояния бита **SXM**). Если же старший разряд ячейки памяти равен единице, то недостающие старшие разряды АЛУ или аккумулятора заполняются или нулями (если бит **SXM** = 0), или единицами (если бит **SXM** = 1).

Другими словами, бит **SXM** определяет, с какой арифметикой мы имеем дело – знаковой (если бит **SXM** = 1), или беззнаковой (если бит **SXM** = 0). Бит **SXM** влияет на большинство арифметических команд и не влияет на логические и некоторые специальные (беззнаковые) команды. Например, если в ячейке памяти **Smem** находится число 1234_{16}, то после выполнения команды **LD Smem, 8, A** (загрузка из памяти в аккумулятор со сдвигом на восемь двоич-

ных разрядов влево) в аккумуляторе **A** будет значение 00 0012 3400_{16} независимо от состояния бита **SXM**. Если же в ячейке памяти **Smem** находится число 8765_{16}, то после выполнения команды **LD Smem, 8, A** в аккумуляторе **A** будет значение 00 0087 6500_{16} , если бит **SXM** = 0, и значение FF FF87 6500_{16} – если бит **SXM** = 1.

1.1.2. Режим коррекции переполнения OVM

При выполнении арифметических операций в АЛУ возможно возникновение переполнения. АЛУ в процессоре TMS320VC5402 имеет 40 разрядов, из которых 32 разряда считаются основными, а 8 – защитными. При выполнении операций переполнение фиксируется в том случае, если 9 старших разрядов (8 защитных и знак) не совпадают. Большинство процессоров может только фиксировать переполнение. Наличие бита **OVM** в процессоре TMS320VC5402 позволят корректировать неправильный результат в случае переполнения до ближайшего правильного. Например, если при сложении двух положительных чисел получился отрицательный результат, то при установленном бите **OVM** вместе с фиксацией переполнения результат будет скорректирован до максимального положительного числа. Для знаковых чисел максимальным положительным числом будет значение 00 7FFF $FFFF_{16}$ (все девять старших разрядов – нули), а максимальным отрицательным числом будет значение FF 8000 0000_{16} (все девять старших разрядов – единицы). Для беззнаковых чисел максимальным числом будет значение 00 FFFF $FFFF_{16}$ (восемь старших разрядов – нули).

Коррекция переполнения имеет смысл только в том случае, если скорректированный результат отличается от истинного значения всего на несколько единиц младшего разряда. Максимальное положительное дробное число в формате с фиксированной запятой равно $1–2^{–31}$ (для нашего АЛУ). Поэтому, если при сложении чисел 0,49 и 0,51 вместо точной 1 получится значение $1–2^{–31}$, то это приемлемый результат. Но если при сложении чисел 0,99 и 0,99 вместо значения 1,98 получится значение $1–2^{–31}$(т. е. в два раза меньше), то такой результат ошибочен.

При возникновении переполнения всегда устанавливается соответствующий флаг переполнения – **OVA**, если результат операции заносится в аккумулятор **A**, или **OVB**, если результат операции заносится в аккумулятор **B**. Данные биты остаются установленными до тех пор, пока они не будут сброшены командой **RSBX** или не будут проверены командой условного перехода **BC** с соответствующим условием.

В регистре состояния процессора имеется также специальный флаг переноса **C**, который при выполнении различных арифметических операций устанавливается или сбрасывается в зависимости от наличия переноса из 31-го разряда АЛУ в 32-й. В отличие от флагов переполнения данный флаг при выполнении большинства операций может как устанавливаться, так и сбрасываться.

Рассмотрим для иллюстрации несколько примеров.

Пример 1. В аккумуляторе **A** находится число 00 7FFF 8002_{16}, а в ячейке памяти **Smem** – число 8001_{16}. При выполнении команды **ADD Smem, A** (сло-

жение содержимого ячейки памяти с аккумулятором) при выключенном бите **SXM** на выходе АЛУ получится значение 00 8000 0003_{16}, при этом будет установлен флаг переполнения **OVA** и сброшен флаг переноса **C**. Такой результат получился потому, что выполнялось сложение двух положительных чисел (00 7FFF 8002_{16} и 00 0000 8001_{16}), а результат получился отрицательным (31-й бит стал равен единице) и не было переноса из 31-го бита (0 + 0 + 1 = 1). Если при этом бит коррекции переполнения **OVM** был выключен, то в аккумулятор **A** будет занесен полученный результат (т. е. совершенно неправильное значение 00 8000 0003_{16}). Если же бит коррекции переполнения **OVM** был включен, то в аккумулятор **A** будет занесено максимальное положительное число 00 7FFF $FFFF_{16}$ (т. е. всего на четыре единицы меньше правильного значения).

Пример 2. В аккумуляторе **A** и в ячейке памяти **Smem** находятся те же числа. При выполнении команды **ADD Smem, A** при включенном бите **SXM** на выходе АЛУ получится значение 00 7FFF 0003_{16}, при этом не будет установлен флаг переполнения **OVA**, но будет установлен флаг переноса **C**. Такой результат получился потому, что выполнялось сложение положительного и отрицательного чисел (00 7FFF 8002_{16} и FF FFFF 8001_{16}), и результат получился положительным (31-й бит остался равен единице), но был перенос из 31-го бита (0 + 1 + 1 = 0 и 1 перенос в 32-й бит).

1.1.3. Программная проверка режимов работы АЛУ

Проверка режимов работы процессора выполняется на трех простых примерах. В каждом примере необходимо выполнить настройку режимов работы процессора, загрузить 32-разрядный операнд в один из аккумуляторов, выполнить заданную операцию и сохранить результат в память. По каждому примеру в отчет нужно включить исходные данные, заданное состояние битов режимов работы (**SXM** и **OVM**), результат и значения битов состояния процессора (**C** и **OV**). Ниже в качестве примера приводится примерный вид программы для одного из заданий.

```
        .def    _c_int00        ; определение начала программы
        .text                   ; начало области программы
_c_int00:                       ; начальная метка программы
        LD      #X, DP          ; текущая страница данных
        SSBX    SXM             ; режим расширения знака
        RSBX    OVM             ; нет коррекции переполнения
        RSBX    OVA             ; сброс бита переполнения
        dld     X, A            ; загрузка двойного слова X
;       RSBX    SXM             ; нет расширения знака
;       SSBX    OVM             ; коррекция переполнения
        NOP                     ; задержка для изменения SXM,OVM
        ADD     Y, A            ; прибавление слова Y к X
        DST     A, Z            ; сохранение двойного слова Z
        NOP                     ; завершение операций в конвейере
        NOP                     ; завершение операций в конвейере
        NOP                     ; точка останова
        .align                  ; выравнивание на страницу
```

```
        .data                   ; начало области данных
X       .long   0x12345678      ; двойное слово X [адрес 0x0100]
Y       .word   0x4321          ; слово Y         [адрес 0x0102]
Z       .long   0               ; двойное слово Z [адрес 0x0104]
```

При написании программы используются специальные директивы ассемблера для управления трансляцией программы и задания переменных и констант (прил. 3).

Перед выполнением загрузки первого операнда обязательно должен быть установлен режим расширения знака (число знаковое) и сброшен флаг переполнения (при выполнении операций он может только устанавливаться). Если в примере требуется выполнить операцию в режиме без расширения знака, то после загрузки первого операнда необходимо сбросить бит **SXM** (в примере соответствующая команда закомментирована). Изменение бита **SXM** вступит в силу только через один такт работы процессора, поэтому в программу вставлена пустая команда **NOP**. Аналогично поступаем и с битом коррекции переполнения **OVM**.

Последней командой программы должна быть пустая команда **NOP** для остановки выполнения программы. Поскольку все команды процессора выполняются в конвейере, то для завершения последней команды программы может понадобиться один или два дополнительных такта (в зависимости от типа команды). Для этого перед точкой останова добавляются одна или две пустые команды **NOP**.

Пример 1. Сложение 32-разрядного числа X и 16-разрядного числа Y при разных значениях режима расширения знака **SXM**.

В данном примере заданы две пары чисел и каждую пару нужно сложить при включенном и выключенном режиме расширения знака **SXM**. Таким образом, в первом примере необходимо получить четыре результата, которые нужно представить в следующем виде:

X = 12345678; Y = 4321; SXM = 0; Z = 12349999; C = 0; OVA = 0.

Пример 2. Сложение 32-разрядного числа X и 16-разрядного числа Y при разных значениях режимов расширения знака **SXM** и коррекции переполнения **OVM**.

В данном примере задана одна пара чисел, и сложение нужно выполнить при всех возможных комбинациях битов **SXM** и **OVM**. Таким образом, во втором примере также необходимо получить четыре результата, которые нужно представить в виде, аналогичном первому примеру.

Пример 3. Выполнение операций получения модуля (**ABS**) и изменения знака (**NEG**) 32-разрядного числа X при разных значениях режима коррекции переполнения **OVM**.

В данном примере заданы два разных значения числа и с каждым из них нужно выполнить две разные операции при включенном и выключенном режиме коррекции переполнения **OVM**. Таким образом, в третьем примере необхо-

димо получить восемь результатов, которые нужно представить в виде, аналогичном первому примеру.

1.1.4. Замечания по программированию примеров для проверки режимов работы АЛУ

При программировании примеров для проверки работы АЛУ при различных режимах расширения знака и коррекции переполнения необходимо учесть следующие замечания:

– все три примера рекомендуется оформить в виде одной общей программы со всеми необходимыми командами и данными;

– неиспользуемые в текущем примере команды нужно не удалять из программы, а закомментировать (т. е. поставить перед командой точку с запятой, как в примере);

– при переходе к новым данным старые данные также нужно не удалять из программы, а закомментировать и ниже ввести новые данные;

– для ускорения работы с программой можно сразу получать два результата во втором аккумуляторе для разных состояний одного из битов режима процессора и записывать их в две разные ячейки памяти (например Z1 и Z2);

– при работе с двумя результатами необходимо только один раз загрузить первый операнд и изменить нужный бит режима между командой первой операции и записью первого операнда (не потребуется пустая команда **NOP**);

– при работе сразу с двумя результатами в программе нужно задать две точки останова для того, чтобы записать состояние флагов **C** и **OVA** (или **OVB**, если результат записывается в аккумулятор **B**).

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 рассчитать значения всех переменных, необходимых для работы программы.
4. Разработать алгоритм работы программы.
5. Написать, оттранслировать, отладить и выполнить программу.
6. Продемонстрировать результат трансляции и работы программы преподавателю.
7. Оформить и защитить отчет по лабораторной работе.

1.3. СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы и исходные данные.
2. Используемые аналитические формулы и выполненные расчеты.
3. Алгоритм работы программы.

4. Листинг программы с комментариями.
5. Результат работы программы.
6. Выводы по работе.

1.4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Разрядность АЛУ и аккумуляторов процессора TMS320VC5402.
2. Разрядность ячеек памяти процессора TMS320VC5402.
3. Прямая адресация данных.
4. Регистр номера текущей страницы данных.
5. Режим расширения знака.
6. Режим коррекции переполнения.
7. Биты переноса и переполнения.
8. Команды управления битами регистров состояния.
9. Команды загрузки данных в аккумулятор.
10. Команды сложения с аккумулятором.
11. Команда изменения знака числа.
12. Команда нахождения абсолютного значения числа.
13. Команды сохранения данных.
14. Директивы ассемблера для определения секций команд и данных.
15. Директивы ассемблера для описания данных.

ЛАБОРАТОРНАЯ РАБОТА №2
ГЕНЕРАТОР ПИЛООБРАЗНЫХ И ТРЕУГОЛЬНЫХ СИГНАЛОВ

ЦЕЛЬ РАБОТЫ:

Разработка алгоритма и программы генератора пилообразных и треугольных сигналов с использованием режимов расширения знака и коррекции переполнения на ассемблере процессора TMS320VC5402 и отладка программы на лабораторном макете TMS320VC5402 DSP Starter Kit.

2.1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1.1. Типовые блоки радиоэлектронной аппаратуры для формирования и преобразования сигналов

Интегратор. Интегрирование в системах цифровой обработки сигналов (ЦОС) сводится к вычислению суммы, а диапазон интегрирования определяется формулой $\tau = N \cdot T$, где T – период дискретизации; N – число отсчетов за время интегрирования.

Вычисление этой суммы удобно вести с помощью рекурсивного звена с передаточной функцией вида $T_{\text{Ц}} = 1/(1-Z^{-1})$ и разностным уравнением $y_T(k) = x_T(k) + y_T(k-1)$, где $y_T(k)$, $y_T(k-1)$ – текущий и предыдущий отсчеты на выходе интегратора; $x_T(k)$ – текущий отсчет на входе интегратора. Интегратор (предыдущий отсчет на выходе интегратора $y_T(k-1)$) перед началом суммирования должен программно сбрасываться в нуль.

Структурная схема интегратора представлена на рис. 2.1.

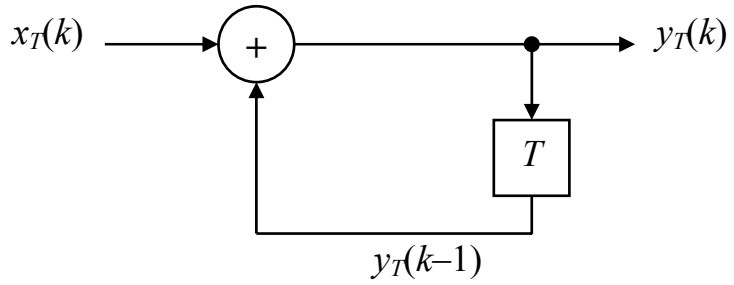

Рис. 2.1. Схема интегратора

Дифференциатор. Простой дифференциатор описывается разностным уравнением $y_T(k) = x_T(k) - x_T(k-1)$ и отображается структурной схемой, представленной на рис. 2.2.

Более сложные и точные дифференциаторы реализуются в виде нерекурсивного фильтра. Их программирование по сути ничем не отличается от программирования простого интегратора. Дополнительной операцией при реализации фильтра является операция умножения входного сигнала на коэффициенты передаточной характеристики фильтра.

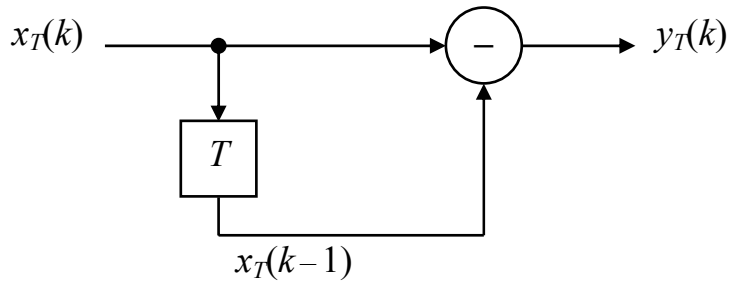

Рис. 2.2. Схема дифференциатора

2.1.2. Реализация генераторов периодических сигналов

Генератор пилообразного напряжения (ГПН). Данный генератор моделируется программной реализацией разностного уравнения первого порядка

$$y_T(k) = A_0 + y_T(k-1), \qquad (2.1)$$

где A_0 – константа, определяющая период ГПН исходя из емкости сумматора. При этом суммирование должно выполняться командой **ADD** с выключенным режимом коррекции переполнения (**OVM**).

Период T_k генератора пилообразного напряжения (рис. 2.3) определяется периодом дискретизации T и значением постоянной A_0 (на рисунке $A_0 = A_1 / 6$):

$$T_K = \frac{K}{A_0} T ,$$

где K – диапазон чисел в аккумуляторе ($K = 2A_1 = 2^n$, где n – число разрядов сумматора); A_0 – шаг приращения пилообразного напряжения. При числе разрядов сумматора, равном 16, значение $A_1 = 32768$. Из-за несимметричности числового диапазона реальное положительное значение амплитуды будет на единицу меньше, и сигнал будет изменяться в диапазоне от –32768 до +32767.

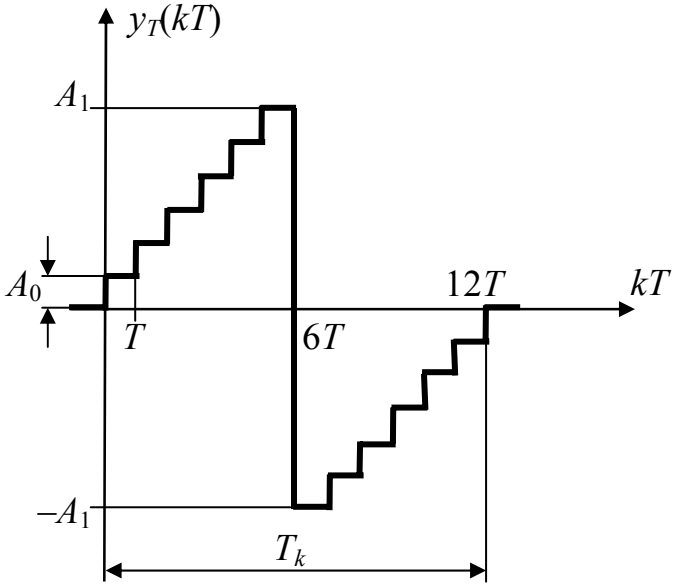

Рис. 2.3. Пилообразное напряжение

Если режим коррекции переполнения выключен, то при превышении максимального положительного значения функция автоматически будет переходить в область отрицательных значений. Если режим коррекции переполнения включен, то функция достигнет максимального положительного значения A_1 и перейдет в режим насыщения. В этом режиме поучить пилообразный сигнал невозможно.

Генератор треугольных сигналов (ГТС). Генератор треугольных сигналов реализуется на основе ГПН с использованием двухпериодного выпрямления командой **ABS** (абсолютное значение аккумулятора). Вид полученного сигнала после центрирования относительно нуля приведен на рис. 2.4.

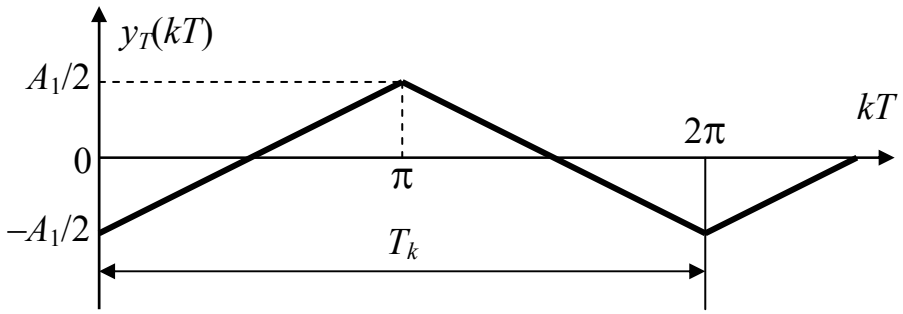

Рис. 2.4. Треугольный сигнал

Если при взятии абсолютного значения (модуля) оставить выключенным режим коррекции переполнения, как это было в ГПН, то правильный треугольный сигнал не получится. Это обусловлено тем, что для максимального отрицательного числа (0x8000 для 16-разрядных чисел в шестнадцатеричной системе счисления) не существует соответствующего положительного значения. Следовательно, при выключенном режиме коррекции переполнения число 0x8000 не изменится, т. е. останется отрицательным. Вид полученного при этом сигнала показан на рис. 2.5.

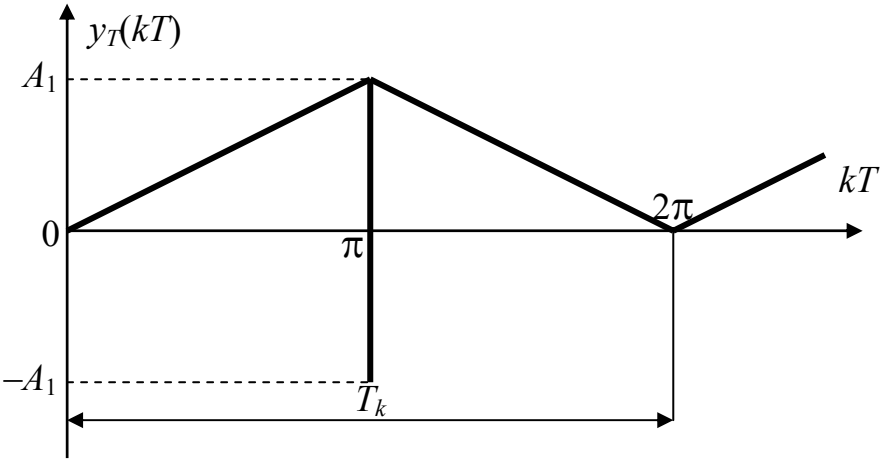

Рис. 2.5. Модуль сигнала ГПН при выключенном бите OVM

Таким образом, перед выполнением команды **ABS** бит **OVM** нужно устанавливать, а перед выполнением команды **ADD** (в ГПН) – сбрасывать.

2.1.3. Замечания по программированию генераторов

При программировании генератора треугольных сигналов необходимо учесть следующие замечания:

– при программировании ГТС сначала нужно построить генератор пилообразного напряжения по формуле (2.1) и проверить его работу;

– для проверки работы генератора необходимо сохранить выходные значения $y_T(k)$ в памяти данных и отобразить полученный массив в графическом виде;

– для отображения графика в заданном масштабе в свойствах графика необходимо указать формат данных (число разрядов после запятой), установив для параметра Q-Value значение 15;

– добавить в основной цикл программы команды, необходимые для получения пилообразного напряжения, в соответствии с рис. 2.4;

– добавить к программе команды, необходимые для изменения амплитуды сигнала и смещения графика относительно нуля, в соответствии с полученным заданием;

– при переключении значения бита **OVM** учесть задержку в конвейере, т. е. обеспечить дополнительный такт между командами изменения **OVM** и командами **ADD** или **ABS** путем изменения порядка следования команд или вставки пустой команды **NOP**.

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 рассчитать значения всех переменных, необходимых для работы программы.
4. Разработать алгоритм работы программы.
5. Написать, оттранслировать, отладить и выполнить программу.
6. Продемонстрировать результат трансляции и работы программы преподавателю.
7. Оформить и защитить отчет по лабораторной работе.

2.3. СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы и исходные данные.
2. Используемые аналитические формулы и выполненные расчеты.
3. Алгоритм работы программы.
4. Листинг программы с комментариями.
5. Результат работы программы.
6. Выводы по работе.

2.4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Прямая адресация данных.
2. Косвенная адресация данных.
3. Форматы целых и дробных данных.
4. Режим расширения знака.
5. Режим коррекции переполнения.
6. Счетчик повторения блока команд.
7. Команды управления битами регистров состояния.
8. Команды занесения данных в регистры.
9. Команды загрузки данных в аккумулятор.
10. Команды сложения с аккумулятором.
11. Команды вычитания из аккумулятора.
12. Команда нахождения абсолютного значения числа.
13. Команды сдвигов данных.
14. Команды сохранения данных.
15. Команды для организации циклов.

ЛАБОРАТОРНАЯ РАБОТА №3
ГЕНЕРАТОР ТРАПЕЦЕИДАЛЬНЫХ СИГНАЛОВ

ЦЕЛЬ РАБОТЫ:

Разработка алгоритма и программы генератора трапецеидальных сигналов на основе генератора треугольных сигналов с использованием режимов расширения знака и коррекции переполнения на ассемблере процессора TMS320VC5402 и отладка программы на лабораторном макете TMS320VC5402 DSP Starter Kit.

3.1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

3.1.1. Алгоритм построения генератора трапецеидальных сигналов

Генератор трапецеидальных сигналов можно легко получить из генератора треугольных сигналов добавлением к программе команды **ADD** со сдвигом операнда на 16 разрядов (сложение со старшим словом аккумулятора). В результате при включенном режиме **OVM** после нескольких сложений в аккумуляторе возникает «насыщение» и в течение нескольких тактов формируется плоская вершина (рис. 3.1).

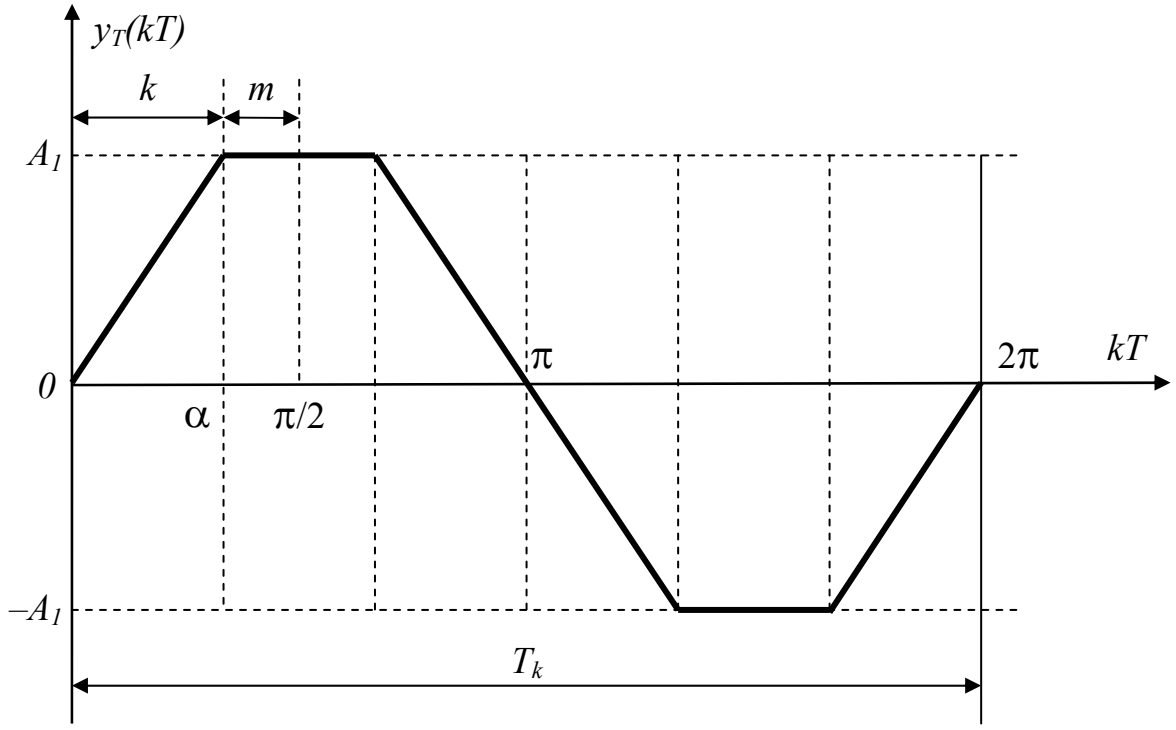

Рис. 3.1. Трапецеидальный сигнал

При этом чем больше команд сложения выполнить, тем вершина будет шире. Например, если взять треугольный сигнал, представленный на рис. 2.4, и прибавить его к старшей части аккумулятора два раза (предварительно обнулив

его), то получится тот же пилообразный сигнал, но с амплитудой A_1. Если же повторить команду сложения три раза, то получится трапецеидальный сигнал с соотношением k/m, равным 2 (т. е. 2/1). При повторении команды сложения четыре раза соотношение k/m будет равным 1 (т. е. 2/2 или 1/1), пять раз – 2/3 и т.д.

В частотной области составляющие трапецеидального сигнала описываются выражением

$$y = \frac{4A}{\pi\alpha}\left(\sin\alpha\sin x + \frac{1}{9}\sin 3\alpha\sin 3x + \frac{1}{25}\sin 5\alpha\sin 5x + \ldots\right). \qquad (3.1)$$

Знание спектрального состава сигнала важно при использовании данного вида сигнала в качестве синусоидального после пропускания его через фильтр нижних частот. В этом случае для получения минимального количества гармоник необходимо, чтобы $\alpha = \pi/3$ (см. рис. 3.1), тогда формула (3.1) примет вид

$$y = \frac{6A\sqrt{3}}{\pi^2}\left(\sin x - \frac{1}{25}\sin 5x + \frac{1}{49}\sin 7x - \ldots\right).$$

Таким образом, на выходе фильтра нижних частот получим высококачественный синусоидальный сигнал с основной составляющей $\frac{6A\sqrt{3}}{\pi^2}$ и подавленными фильтром гармониками, начиная с пятой.

Периоды колебаний T_k рассматриваемых сигналов соответствуют периоду колебаний ГПН, на основе которого они реализованы.

3.1.2. Замечания по программированию генератора

При программировании генератора трапецеидальных сигналов необходимо учесть следующие замечания:

– программа генератора трапецеидальных сигналов разрабатывается на основе программы генератора треугольных сигналов, которая формирует сигнал с амплитудой ½, представленный на рис. 2.4;

– в зависимости от заданного значения k определить необходимую амплитуду исходного треугольного сигнала, разделив 1 на k;

– в зависимости от заданного значения m определить необходимое число суммирований исходного треугольного сигнала, прибавив значение m к значению k;

– если амплитуда получается равной единице (для $k = 1$), то можно оставить амплитуду, равную ½ (как для $k = 2$), но увеличить определенное ранее число суммирований в два раза;

– добавить в основной цикл программы генератора треугольных сигналов команды, необходимые для получения трапеции: сдвиг (**SFTA**), повторение (**RPTZ**) и сложение (**ADD**);

– добавить к программе команды, необходимые для изменения амплитуды сигнала и смещения графика относительно нуля в соответствии с полученным заданием;

– все добавляемые команды должны выполняться при включенном бите **OVM**.

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 рассчитать значения всех переменных, необходимых для работы программы.
4. Разработать алгоритм работы программы.
5. Написать, оттранслировать, отладить и выполнить программу.
6. Продемонстрировать результат трансляции и работы программы преподавателю.
7. Оформить и защитить отчет по лабораторной работе.

3.3. СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы и исходные данные.
2. Используемые аналитические формулы и выполненные расчеты.
3. Алгоритм работы программы.
4. Листинг программы с комментариями.
5. Результат работы программы.
6. Выводы по работе.

3.4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Косвенная адресация данных.
2. Режим расширения знака.
3. Режим коррекции переполнения.
4. Команды управления битами регистров состояния.
5. Команды занесения данных в регистры.
6. Команды загрузки данных в аккумулятор.
7. Команды сложения с аккумулятором.
8. Команды вычитания из аккумулятора.
9. Команда нахождения абсолютного значения числа.
10. Команды сдвигов данных.
11. Команды повторения одиночной команды.
12. Команды сохранения данных.
13. Команды организации циклов.

ЛАБОРАТОРНАЯ РАБОТА №4

ГЕНЕРАТОР СИНУСОИДАЛЬНЫХ СИГНАЛОВ НА ОСНОВЕ НЕУСТОЙЧИВОГО ЗВЕНА ВТОРОГО ПОРЯДКА

ЦЕЛЬ РАБОТЫ:

Разработка алгоритма и программы генератора синусоидальных сигналов заданной частоты на основе неустойчивого звена 2-го порядка на ассемблере процессора TMS320VC5402 и отладка программы на лабораторном макете TMS320VC5402 DSP Starter Kit.

4.1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

4.1.1. Генератор синусоидальных колебаний на основе неустойчивого звена 2-го порядка

За основу при построении генератора синусоидальных колебаний взято простейшее рекурсивное звено 2-го порядка – цифровой резонатор (колебательное звено), передаточная функция которого имеет вид

$$T_{\text{Ц}}(z) = \frac{1}{1 - b_1 z^{-1} - b_2 z^{-2}} .$$

Условием устойчивости такого звена является выполнение соотношений $|b_1| < 2$; $b_2 < 0$; $|b_2| < 1$. Резонансные свойства цифрового резонатора характеризуются полосой пропускания (по уровню 0,707 от максимального)

$$2\Delta\omega = 2 \times 2\pi\Delta f = \frac{1}{T_{\text{Д}}} 2 \arcsin \frac{1 + b_2}{2\sqrt{-b_2}}$$

и добротностью $Q = \omega_0 / 2\Delta\omega$, где ω_0 – резонансная частота:

$$\omega_0 = 2\pi f_0 = \frac{1}{T_{\text{Д}}} \arccos \frac{b_1}{2\sqrt{|b_2|}} .$$

Видно, что при $b_2 = -1$ звено становится неустойчивым:

$$2\Delta\omega = \frac{1}{T_{\text{Д}}} 2 \arcsin \frac{1-1}{2\sqrt{1}} = 0, \quad Q = \frac{\omega_0}{0} \to \infty$$

и начинают генерироваться колебания при получении начального возбуждения.

Разностное уравнение $y(kT) = y_T(k) = x_T(k) + b_1 y_T(k-1) + b_2 y_T(k-2)$ при $x_T(k) = 0$, $b_2 = -1$ принимает вид

$$y_T(k) = b_1 y_T(k-1) - y_T(k-2) . \qquad (4.1)$$

Структурная схема полученного генератора синусоидальных сигналов показана на рис. 4.1.

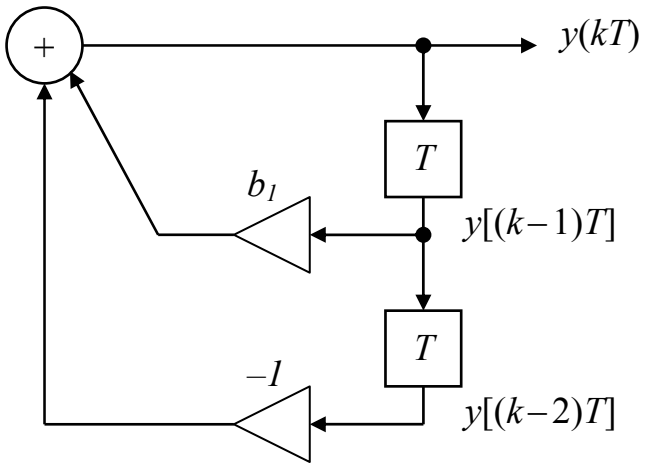

Рис. 4.1. Структурная схема генератора синусоидальных колебаний

Для работы генератора необходимо сформировать начальное возбуждение, т. е. задать значения $y_T(k-2)$ и $y_T(k-1)$ (рис. 4.2).

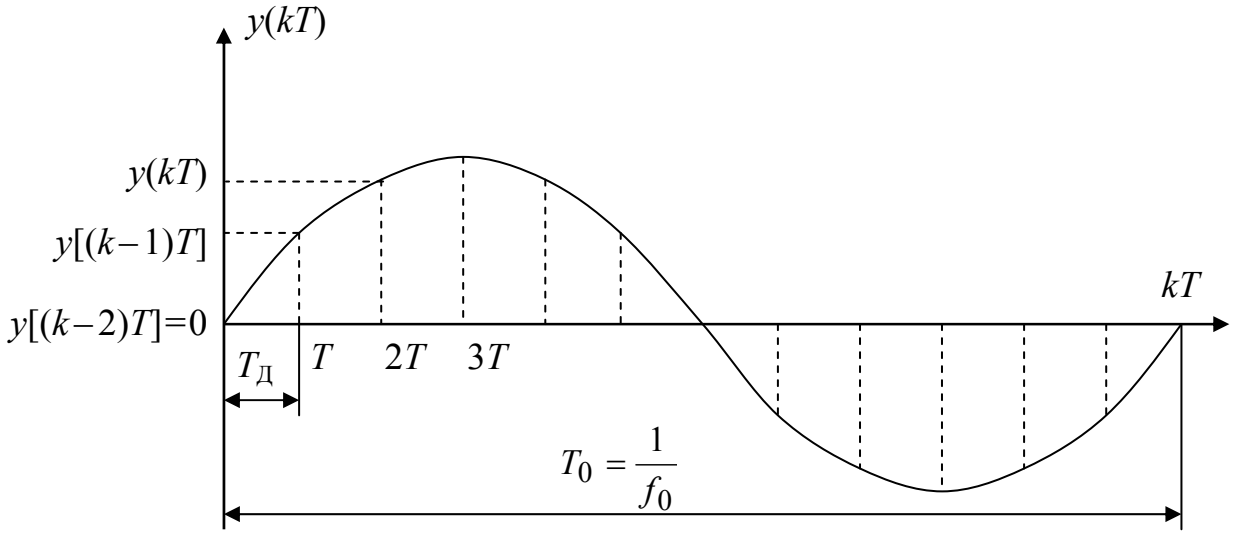

Рис. 4.2. Формирование начальных условий

Из рисунка видно, что можно ввести любые два соседних отсчета. Для удобства примем $y_T(k-2) = 0$, тогда

$$y_T(k-1) = \sin 2\pi \frac{T_Д}{T_0} = \sin 2\pi \frac{f_0}{f_Д} .$$

Пример 1. Простой генератор синусоидальных колебаний с частотой f_0 = 1700 Гц; $f_Д$ = 27,3 кГц. Произведем следующие действия:

а) расчет коэффициента b_1:

$$\omega_0 = 2\pi f_0 = \frac{1}{T_Д} \arccos \frac{b_1}{2} = f_Д \arccos \frac{b_1}{2};$$

$$\frac{b_1}{2} = \cos \frac{2\pi f_0}{f_Д} = \cos \frac{2\pi \times 1{,}7}{27{,}3} = 0{,}924429;$$

б) вычисление отсчетов начального возбуждения:

$$y_T(k-2)=0;$$

$$y_T(k-1)=\sin 2\pi\frac{f_0}{f_Д}=\sin 2\pi\frac{1{,}7}{27{,}3}=0{,}381354;$$

в) представление дробных чисел в формате целых чисел:

$$\frac{b_1}{2}\to\frac{b_1}{2}\times 2^{15}=0{,}924429\times 2^{15}=30292;$$

$$y_T(k-1)\to y_T(k-1)\times 2^{15}=0{,}381354\times 2^{15}=12496.$$

Пример 2. Частотно-манипулированный генератор на основе неустойчивого звена 2-го порядка. Генератор формирует сигналы с частотой 1300 и 2100 Гц в зависимости от задающего сигнала. Задающий сигнал может быть представлен последовательностью как однополярных, так и биполярных импульсов. Произведем следующие действия:

а) расчет коэффициентов b_1:

$$\frac{b_1}{2}=\cos\frac{2\pi f_0}{f_Д};$$

частота генератора 1300 Гц: $\frac{b_1}{2}=\cos\frac{2\pi\times 1{,}3}{27{,}3}=0{,}955573;$

частота генератора 2100 Гц: $\frac{b_1}{2}=\cos\frac{2\pi\times 2{,}1}{27{,}3}=0{,}885456;$

б) вычисление отсчетов начального возбуждения:

$$y_T(k-2)=0;\ y_T(k-1)=\sin\frac{2\pi f_0}{f_Д};$$

частота генератора 1300 Гц: $y_T(k-1)=\sin\frac{2\pi\times 1{,}3}{27{,}3}=0{,}294755;$

частота генератора 2100 Гц: $y_T(k-1)=\sin\frac{2\pi\times 2{,}1}{27{,}3}=0{,}464723;$

в) представление дробных чисел в формате целых чисел:

$$\frac{b_1}{2}(f_0=1300\,\text{Гц})\to\frac{b_1}{2}\times 2^{15}=0{,}995573\times 2^{15}=31312;$$

$$\frac{b_1}{2}(f_0=2100\,\text{Гц})\to\frac{b_1}{2}\times 2^{15}=0{,}885456\times 2^{15}=29015;$$

$$y_T(k-1)(f_0=1300\,\text{Гц})\to y_T(k-1)\times 2^{15}=0{,}294755\times 2^{15}=9659;$$

$$y_T(k-1)(f_0=2100\,\text{Гц})\to y_T(k-1)\times 2^{15}=0{,}464723\times 2^{15}=15228.$$

4.1.2. Замечания по программированию генератора

При программировании генератора синусоидальных сигналов необходимо учесть следующие замечания:

– поскольку в уравнении (4.1) коэффициент $b_1 > 1$, то прямое программирование данного уравнения невозможно;

– для физической реализации уравнения (4.1) его необходимо преобразовать к следующему виду: $y_T(k) = (b_1/2)\cdot y_T(k-1) - y_T(k-2) + (b_1/2)\cdot y_T(k-1)$;

– после вычисления очередного значения $y_T(k)$ необходимо сдвинуть предысторию фильтра $y_T(k-1) \rightarrow y_T(k-2)$, $y_T(k) \rightarrow y_T(k-1)$;

– для выполнения первой операции сдвига можно использовать команду задержки **DELAY**, а второй сдвиг выполнить простой записью нового значения $y_T(k)$ в ячейку для $y_T(k-1)$;

– все вычисления должны выполняться при включенном бите **OVM**;

– если амплитуда полученного сигнала затухает, необходимо увеличить целочисленное представление коэффициента $b_1/2$ на одну-две единицы.

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 рассчитать значения всех переменных, необходимых для работы программы.
4. Разработать алгоритм работы программы.
5. Написать, оттранслировать, отладить и выполнить программу.
6. Продемонстрировать результат трансляции и работы программы преподавателю.
7. Оформить и защитить отчет по лабораторной работе.

4.3. СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы и исходные данные.
2. Используемые аналитические формулы и выполненные расчеты.
3. Алгоритм работы программы.
4. Листинг программы с комментариями.
5. Результат работы программы.
6. Выводы по работе.

4.4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Прямая адресация данных.
2. Косвенная адресация данных.

3. Режим расширения знака.
4. Режим коррекции переполнения.
5. Команды занесения данных в регистры.
6. Команды загрузки данных в аккумулятор.
7. Команды сложения с аккумулятором.
8. Команды вычитания из аккумулятора.
9. Команды умножения.
10. Команды умножения со сложением и вычитанием.
11. Правила умножения целых и дробных чисел.
12. Бит управления умножением дробных чисел.
13. Режим насыщения аккумулятора.
14. Команда задержки данных.
15. Команды сдвигов данных.
16. Команды сохранения данных.
17. Команды организации циклов.

ЛАБОРАТОРНАЯ РАБОТА №5

ПРОГРАММНО-УПРАВЛЯЕМЫЙ ГЕНЕРАТОР ДЛЯ МОДУЛЯТОРА FSK-МОДЕМА

ЦЕЛЬ РАБОТЫ:

Разработка алгоритма и программы табличного генератора синусоидальных сигналов с управляемой частотой для модулятора FSK-модема на ассемблере процессора TMS320VC5402 и отладка программы на лабораторном макете TMS320VC5402 DSP Starter Kit.

5.1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

5.1.1. Принцип работы модема с частотной модуляцией

Передача цифровых данных по линиям связи осуществляется с использованием модемов (модулятор-демодулятор). При этом цифровой сигнал преобразуется в аналоговый сигнал определенным образом. Одним из способов модуляции является частотная модуляция (FSK – frequency shift keyed), при которой передаваемым единицам и нулям соответствуют сигналы разной частоты. Существует множество стандартов для модемов с частотной модуляцией (FSK-модемов). Некоторые из них представлены в табл. 5.1.

Таблица 5.1

Основные параметры FSK-модемов различных стандартов

Стандарт		Скорость, бод	Несущая, Гц	Частота 1, Гц	Частота 0, Гц
V.21	Исходный	300	1080	980	1180
	Ответный	300	1750	1650	1850
BELL 103	Исходный	300	1170	1270	1070
	Ответный	300	2125	2225	2025
V.23		600,1200	1700	1300	2100
BELL 202		1200	1700	1200	2200

В качестве примера для дальнейших расчетов выберем стандарт V.23 со скоростью передачи 1200 бод.

Каждый модем состоит из двух частей: модулятора и демодулятора. В основе работы модулятора лежит следующее соотношение:

$$S(t) = \cos[(\omega_c \pm \delta_\omega)t + \varphi] , \qquad (5.1)$$

где $S(t)$ – передаваемый сигнал; ω_c – несущая частота; δ_ω – девиация частоты, t – время, φ – сдвиг фазы.

На каждом из интервалов T сигнал $S(t)$ с частотой $f_0 = (f_c + \delta f)$ или $f_1 = (f_c - \delta f)$ соответствует передаче бита «0» или «1». Из выражений для определения частот ω_0 и ω_1:

$$\omega_0 = (\omega_c + \delta_\omega) ,$$

$$\omega_1 = (\omega_c - \delta_\omega) ,$$

можно получить выражения для ω_c и δ_ω:

$$\omega_c = (\omega_0 + \omega_1)/2 ,$$

$$\delta_\omega = (\omega_0 - \omega_1)/2 .$$

Ниже на рис. 5.1 приведен примерный вид сигнала на выходе модема при передаче кода 010.

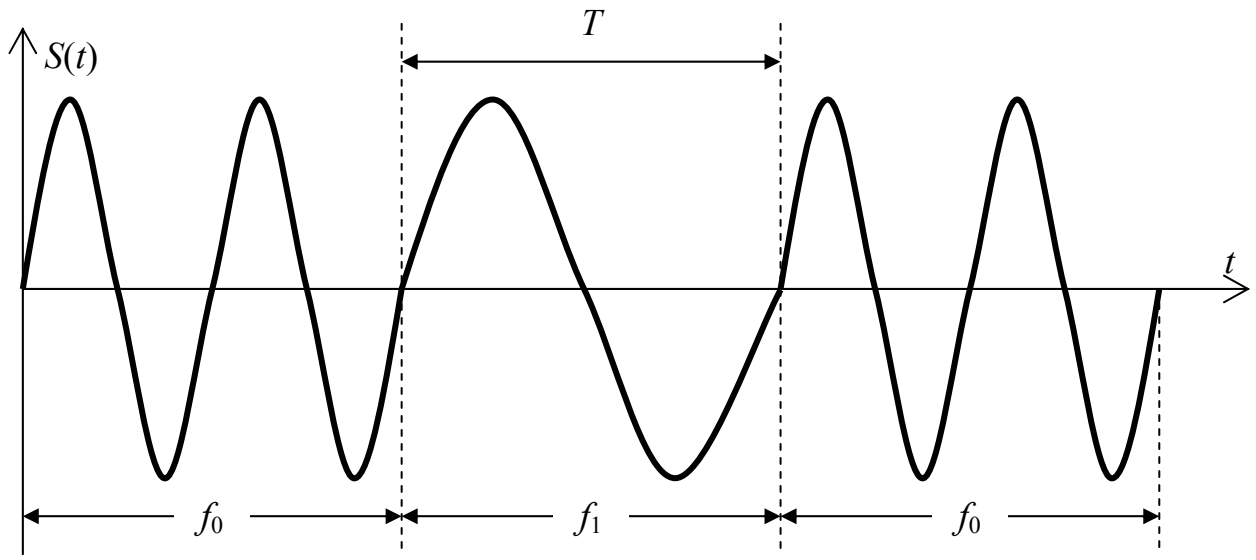

Рис. 5.1. Сигнал на выходе передатчика для кода 010

5.1.2. Программно-управляемый генератор синусоидальных сигналов

Для выбранного стандарта заданы следующие частоты: $f_c = 1700$ Гц, $f_1 = 1300$ Гц, $f_0 = 2100$ Гц, $\delta f = 400$ Гц.

Простейшим способом формирования сигнала в модуляторе при использовании цифровой техники является табличный метод формирования синусоиды. Для этого в памяти модулятора хранится таблица синуса из некоторого числа точек в диапазоне от 0 до 2π.

Число точек в таблице определяется частотой дискретизации, скоростью передачи и частотами единицы и нуля. Частота дискретизации выбирается по теореме Котельникова исходя из максимальной передаваемой частоты и скорости передачи. Согласно теореме Котельникова соотношение частоты дискретизации $f_Д$ и максимальной частоты (в нашем примере f_0) должно быть не менее двух. На практике обычно выбирается соотношение четыре или больше:

$$f_Д = 2100 \times 4 = 8400 \text{ Гц} . \quad (5.2)$$

За время передачи одного бита информации должно передаваться целое число отсчетов синуса. Частота дискретизации выбирается кратной скорости передачи из ряда 600, 1200, 2400, 4800, 9600 Гц. Таким образом, ближайшее значение, превышающее (5.2), равно 9600 Гц.

При выбранной частоте дискретизации за секунду передается 9600 отсчетов синуса и на один бит передаваемой информации приходится

$$n = f_Д / V = 9600 / 1200 = 8 \text{ (отсчетов)} ,$$

где $V = 1200$ бод – выбранная скорость передачи данных.

Если выбрать размер таблицы синуса в 9600 отсчетов и выдавать с частотой дискретизации каждый отсчет из таблицы, то за 1 с будут переданы все отсчеты из таблицы (один период), что соответствует частоте 1 Гц. Если выбирать из таблицы отсчеты с шагом 2 – получим сигнал с частотой 2 Гц, а если с шагом 1300 – с частотой 1300 Гц соответственно.

Таким образом, таблица из 9600 отсчетов позволяет формировать сигналы с частотами, кратными 1 Гц. Но поскольку необходимо формировать сигнал с частотами 1300 Гц и 2100 Гц, то совсем не обязательно иметь такую большую таблицу. В нашем случае достаточна дискретность по частоте в 100 Гц.

Найдем наибольший общий делитель для чисел 1300, 2100 и 9600 (f_1, f_0 и $f_Д$ соответственно), который равен 100. Разделив данные числа на 100, получим следующие значения: $S1 = 13$, $S0 = 21$ и $N = 96$, где $S1$ и $S0$ – шаг по таблице для «1» и «0» соответственно, а N – размер таблицы синуса.

Выбирая данные из таблицы с шагом 1, за 1 с (9600 тактов) переберем таблицу 100 раз (100 периодов синуса), что дает сигнал с частотой 100 Гц, с шагом 13 – 1300 Гц, а с шагом 21 – 2100 Гц, что и требуется при заданных исходных данных.

5.1.3. Замечания по программированию управляемого генератора

При программировании генератора синусоидальных сигналов необходимо учесть следующие замечания:

– все используемые в программе константы ($S0$, $S1$, N и др.) должны быть определены в начале программы с помощью директивы **.set**;

– при выборе данных из таблицы синуса необходимо применять косвенную циклическую адресацию ***ARx+0%** (прил. 4), занеся предварительно в регистр размера кольцевого буфера **BK** длину таблицы синуса, а во вспомогательный регистр **AR0** – нужный шаг (в зависимости от передаваемого бита);

– для правильной работы циклической адресации таблицу синуса нужно расположить начиная с адреса, кратного степени двойки, большей, чем длина таблицы синуса;

– таблицы синуса для всех вариантов подготовлены заранее и хранятся в виде отдельных файлов;

– данные из таблиц синуса можно не копировать в свою программу, а просто подключить файл, используя директиву **.include "sin96.tab"** (нужный файл предварительно необходимо скопировать в папку с программой);

– формируемый генератором сигнал необходимо записать в память данных в виде массива и отобразить на экране в графическом виде.

5.2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретические сведения по теме лабораторной работы (подразд. 5.1).
2. Получить у преподавателя задание для выполнения практической части работы.
3. Согласно заданию рассчитать значения всех переменных, необходимых для работы программы.
4. Разработать алгоритм работы программы.
5. Написать, оттранслировать, отладить и выполнить программу.
6. Продемонстрировать результат трансляции и работы программы преподавателю.
7. Оформить и защитить отчет по лабораторной работе.

5.3. СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы и исходные данные.
2. Используемые аналитические формулы и выполненные расчеты.
3. Алгоритм работы программы.
4. Листинг программы с комментариями.
5. Результат работы программы.
6. Выводы по работе.

5.4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Назначение модема.
2. Принцип частотной модуляции.
3. Принцип работы модулятора FSK-модема.
4. Табличный способ формирования синусоиды.
5. Косвенная циклическая адресация данных.
6. Определение размера и начального адреса кольцевого буфера.
7. Определение требуемого размера таблицы синуса.
8. Расчет шага в таблице для формирования нужных частот.
9. Определение количества отсчетов синуса на один бит информации.
10. Команды занесения данных в регистры.
11. Команды загрузки данных в аккумулятор.
12. Команды повторения блока команд.
13. Команды сохранения данных.

ЛАБОРАТОРНАЯ РАБОТА №6
МОДУЛЯТОР FSK-МОДЕМА

ЦЕЛЬ РАБОТЫ:

Разработка алгоритма и программы модулятора FSK-модема на ассемблере процессора TMS320VC5402 и отладка программы на лабораторном макете TMS320VC5402 DSP Starter Kit.

6.1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

6.1.1. Принцип работы модулятора FSK-модема

Принцип работы модулятора заключается в анализе очередного бита передаваемых данных и выборке из таблицы размерностью N отсчетов с шагом $S1$ или $S0$ (в зависимости от передаваемого бита) в течение n тактов и выдачи их в ЦАП с частотой $f_Д$. Затем производится анализ следующего передаваемого бита и выборка продолжается с тем же или другим шагом. Так как при переходе от одного бита к другому изменяется только шаг в таблице (приращение аргумента синуса), то формируемый сигнал не имеет разрывов фазы. Точность формирования сигнала зависит только от стабильности задающего генератора. При выходе адреса за пределы таблицы его необходимо скорректировать, уменьшив на длину таблицы.

Поскольку при передаче данных используется асинхронный метод, то необходимо предусмотреть возможность синхронизации приемника по принимаемым данным. Для этого в поток передаваемых данных включаются специальные биты синхронизации. Это так называемые «стартовый» и «стоповый» биты.

В качестве «стартового» бита обычно используется бит нуля, а в качестве стопового – бит единицы. Обычно передача данных выполняется байтами и первому биту данных всегда предшествует «стартовый» бит. Для контроля правильности передачи после последнего бита данных передается контрольный бит, который формируется таким образом, чтобы число единичных битов данных вместе с контрольным было четным или нечетным. Получают контрольный бит путем суммирования всех битов данных по модулю два. В качестве контрольного бита используется или полученная сумма (контроль на четность), или ее инверсия (контроль на нечетность).

Например, если необходимо передать байт 01001110, то, просуммировав все его биты по модулю два, получим 0. Таким образом, при контроле на четность к передаваемому байту добавляется бит 0, а при контроле на нечетность – 1. Если передача выполняется со старших битов, то необходимо передать последовательность из девяти битов, равную 010011100, при контроле на четность или равную 010011101 – при контроле на нечетность. Если посчитать количест-

во единиц в первой последовательности, то оно получится равным четырем (четное), а во второй – пяти (нечетное).

На приемной стороне контроль правильности принятых данных проверяется суммированием всех девяти битов последовательности по модулю два и сравнением полученной суммы с нулем (контроль на четность) или с единицей (контроль на нечетность).

Данный метод позволяет обнаруживать только одиночные ошибки. Для обнаружения двойных ошибок и коррекции принятых данных используются более сложные методы контроля и коррекции данных.

Для синхронизации приема следующего байта каждый передаваемый байт обязательно сопровождается одним или двумя «стоповыми» битами. Когда передачи данных нет, в линию обычно передается последовательность «стоповых» битов, т. е. постоянный синусоидальный сигнал с частотой, соответствующей передаче единицы.

Таким образом, для проверки работы модема модулятор должен сформировать последовательность из двух «стоповых» битов (эмуляция отсутствия передачи данных), одного «стартового» бита, 16 битов данных (процессор обычно оперирует 16-разрядными словами) и двух «стоповых» битов (передача без контроля четности).

Если в качестве данных задать слово, состоящее из восьми единиц и восьми нулей, то на выходе модулятора при заданных исходных данных должен получиться сигнал, представленный на рис. 6.1.

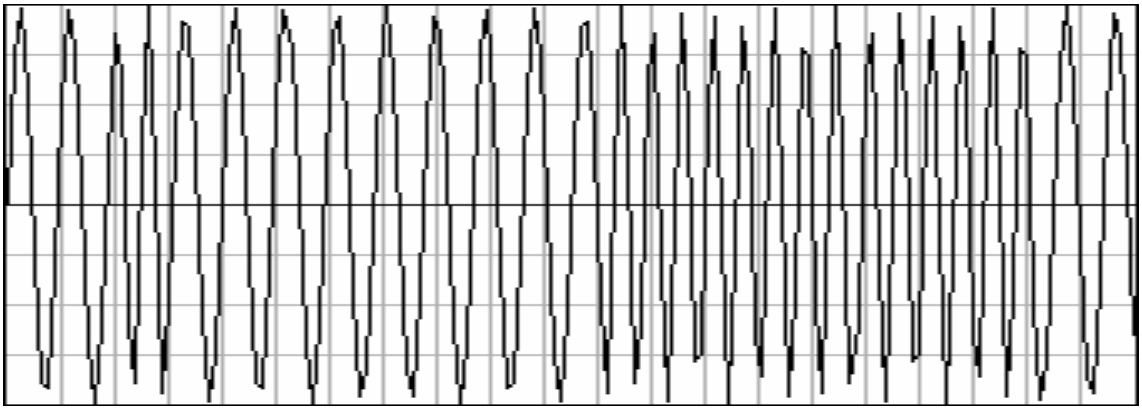

Рис. 6.1. Сигнал на выходе передатчика модема

Для проверки передаваемых битов данных можно использовать различные методы и команды процессора. В лабораторной работе предлагается использовать один из следующих способов: *тестирование битов или сдвиг*.

Первый способ заключается в использовании специальной команды проверки (тестирования) битов **BITT**. Данная команда имеет один явно адресуемый операнд – ячейку памяти данных, адресуемую с использованием прямой или косвенной адресации типа **Smem**. В данной ячейке должно находиться передаваемое слово данных. Вторым неявным операндом данной команды является временный регистр **T**, который должен содержать в четырех младших разрядах номер тестируемого бита. Следует иметь в виду, что в данной команде

принята обратная нумерация битов, т. е. старший бит имеет номер 0, а младший бит – номер 15. Таким образом, если передачу данных нужно начинать со старшего бита, то в регистр **T** вначале нужно занести 0. Для проверки очередного бита содержимое регистра **T** нужно увеличивать на единицу. Результатом работы команды является копирование проверяемого бита в специальный флаг **TC**, который может быть проверен командой условного перехода.

Для формирования «стоповых» и «стартового» битов при использовании данного метода нужно просто перед вызовом подпрограммы программно-управляемого генератора задавать нужный шаг.

Второй способ заключается в использовании команд сдвига. При использовании данного метода передаваемое слово данных должно быть предварительно загружено в соответствующие разряды аккумулятора. Для сдвига данных можно использовать практически любую команду сдвига (циклический, логический или арифметический сдвиги). Для проверки очередного бита содержимое аккумулятора необходимо сдвинуть на один разряд. При этом выдвигаемый бит заносится во флаг переноса **C**, который может быть проверен командой условного перехода.

Поскольку передаваемое слово данных состоит из 16 двоичных разрядов, а емкость основной части аккумулятора 32 бита, то в аккумулятор можно загрузить слово данных вместе с битами синхронизации (всего 21 бит) и использовать единый алгоритм для передачи всего пакета.

6.1.2. Замечания по программированию модулятора

При программировании генератора синусоидальных сигналов необходимо учесть следующие замечания:

– для упрощения проверки правильности работы программы передаваемое слово данных необходимо определить в начале программы с помощью директивы **.set**;

– первоначальную проверку правильности работы программы произвести с использованием слова данных, состоящего из восьми единиц и восьми нулей, и убедиться, что график сигнала имеет вид, аналогичный рис. 6.1;

– в программе модулятора для передачи одного бита информации необходимо использовать подпрограмму управляемого генератора синусоидальных сигналов, предварительно загружая в регистр **AR0** нужный шаг;

– наиболее эффективный способ загрузки нужного шага в регистр **AR0** заключается в использовании команды условного выполнения **XC** (прил. 5), которая в зависимости от проверяемого условия (прил. 6) выполняет следующую за ней команду или пропускает ее;

– при использовании команды **XC** последовательность команд загрузки регистра **AR0** может быть следующей: занести в регистр шаг для нуля, проверить командой **XC** условие передачи единицы, занести в регистр шаг для единицы (данная команда выполнится, только если будет передаваться единица);

– формируемый модулятором сигнал необходимо записать в память данных в виде массива и отобразить на экране в графическом виде.

6.2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретические сведения по теме лабораторной работы (подразд. 6.1).
2. Получить у преподавателя задание для выполнения практической части работы.
3. Согласно заданию рассчитать значения всех переменных, необходимых для работы программы.
4. Разработать алгоритм работы программы.
5. Написать, оттранслировать, отладить и выполнить программу.
6. Продемонстрировать результат трансляции и работы программы преподавателю.
7. Оформить и защитить отчет по лабораторной работе.

6.3. СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы и исходные данные.
2. Используемые аналитические формулы и выполненные расчеты.
3. Алгоритм работы программы.
4. Листинг программы с комментариями.
5. Результат работы программы.
6. Выводы по работе.

6.4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Принцип частотной модуляции.
2. Принцип работы модулятора FSK-модема.
3. Табличный способ формирования синусоиды.
4. Косвенная циклическая адресация данных.
5. Способы определения передаваемого бита информации.
6. Команды сдвигов данных.
7. Команды тестирования битов.
8. Бит переноса **C** и бит проверки и управления **TC**.
9. Команды загрузки данных в аккумулятор.
10. Команды загрузки регистров.
11. Команды пересылки данных.
12. Команды условного перехода.
13. Команда условного выполнения.
14. Команды повторения одиночной команды.
15. Команды сохранения данных.
16. Команды организации циклов.

ЛАБОРАТОРНАЯ РАБОТА №7

МОДУЛЬ ФАЗОСДВИГАТЕЛЯ ДЛЯ ДЕМОДУЛЯТОРА FSK-МОДЕМА

ЦЕЛЬ РАБОТЫ:

Разработка алгоритма и программы фазосдвигателя и умножителя для демодулятора FSK-модема на ассемблере процессора TMS320VC5402 и отладка программы на лабораторном макете TMS320VC5402 DSP Starter Kit.

7.1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

7.1.1. Принцип работы демодулятора FSK-модема

Задачей демодулятора является определение частоты принимаемого сигнала и формирование соответствующего цифрового кода. Для этого можно применить, например, аналоговую или цифровую фильтрацию. Однако, учитывая небольшую разницу частот «1» и «0», а также малую выборку сигнала с одной частотой, данный метод является неэффективным.

Другой метод основан на перемножении двух сигналов, принятого и задержанного на некоторое время (формула произведения двух углов).

Согласно выражению (5.1) принятый из канала сигнал в момент времени $t = n \cdot \Delta t$ равен

$$x(n) = \cos((\omega_c \pm \delta_\omega)n + \varphi) . \qquad (7.1)$$

Этот сигнал умножается на такой же, но задержанный на время $\tau = m \cdot \Delta t$ сигнал:

$$y(n) = x(n - m) = \cos((\omega_c \pm \delta_\omega)(n - m) + \varphi) .$$

В результате после умножения получаем выражение для произведения косинусов, которое может быть преобразовано в сумму косинусов:

$$2 \cdot x(t) \cdot x(t - \tau) = 2 \cdot \cos[(\omega_c \pm \delta_\omega)t + \varphi] \cdot \cos[(\omega_c \pm \delta_\omega)(t - \tau) + \varphi] =$$

$$= \cos[2(\omega_c \pm \delta_\omega)t - (\omega_c \pm \delta_\omega)\tau + 2\varphi] + \cos[(\omega_c \pm \delta_\omega)\tau] , \qquad (7.2)$$

где τ – время задержки, Δt – период дискретизации.

Если выбрать τ таким, что $\omega_c \cdot \tau = \pi/2$, то после низкочастотной фильтрации (ФНЧ) удвоенной несущей частоты ω_c получается

$$\cos[(\omega_c \pm \delta_\omega)\tau] = \cos(\pi/2 \pm \delta_\omega\tau) = -\sin(\pm\delta_\omega\tau) = -[\pm\sin(\delta_\omega\tau)] .$$

Отсюда следует, что по знаку сигнала после низкочастотной фильтрации (в блоке детектора) можно определить значение передаваемой информации. Отрицательный результат соответствует передаче «1», а положительный – «0».

Таким образом, структурная схема демодулятора имеет следующий вид (рис. 7.1):

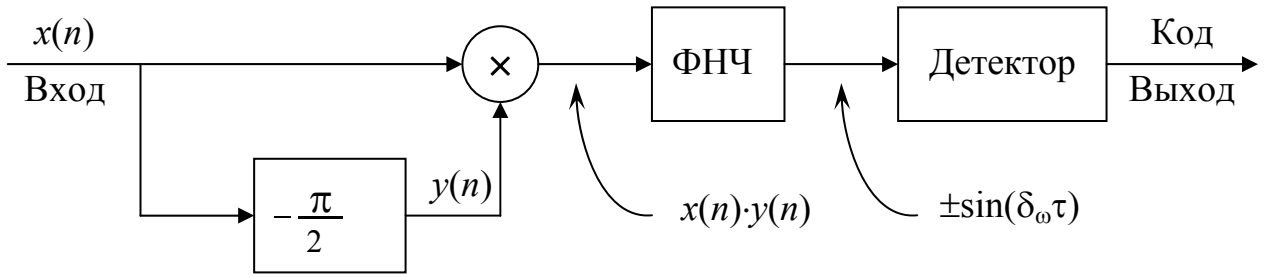

Рис. 7.1. Структурная схема демодулятора FSK-модема

Для минимизации ошибки детектирования сигнала необходимо как можно точнее задать задержку $\pi/2$. Для реализации задержки можно использовать цифровой нерекурсивный фильтр с одним нулем:

$$y(n) = x(n) + \beta \cdot x(n-1)\ . \tag{7.3}$$

Передаточная функция такого фильтра равна

$$H(z) = 1 + \beta \cdot z^{-1}\ ,$$

а групповая задержка фильтра определяется как

$$\tau = -\frac{d\varphi(\omega)}{d\omega} = \frac{-d}{d\omega}\left(\operatorname{arctg}\left(\frac{-\beta\sin(\omega)}{1+\beta\cos(\omega)}\right)\right) = \frac{\beta(\beta+\cos(\omega))}{1+2\beta\cos(\omega)+\beta^2}.$$

Переходя к относительным частотам и задержкам, получаем

$$D = \frac{\beta(\beta+\cos(\omega))}{1+2\beta\cos(\omega)+\beta^2},$$

откуда

$$\beta = \frac{-(1-2D)\cos(\omega) \pm \sqrt{(1-2D)^2\cos^2(\omega) + 4D(1-D)}}{2(1-D)}. \tag{7.4}$$

В данном выражении в качестве ω используется относительная круговая частота

$$\omega = 2\pi\,(f_C / f_Д) = 2\pi\,(\,1700 / 9600\,) = 1{,}1126474 \text{ рад} \tag{7.5}$$

и относительная задержка D, равная

$$D = (f_Д / f_C) / 4 = (\,9600 / 1700\,) / 4 = 1{,}4117647 \text{ такта}\ . \tag{7.6}$$

Так как задержку на один такт можно легко получить, взяв вместо $x(n)$ значение $x(n-1)$, то в качестве задержки для цифрового фильтра берем дробную часть (7.6), которая равна 0,4117647.

Учитывая (7.5) и (7.6), из выражения (7.4) получаем

$$\beta = \frac{-0{,}17647\cos(1{,}11265) \pm \sqrt{(0{,}17647\cos(1{,}11265))^2 + 0{,}96886}}{1{,}17647} =$$

$$= -0{,}06634 \pm 0{,}83929 .$$

Таким образом, имеем два значения коэффициента β, равные +0,77295 и –0,90663. Из двух коэффициентов необходимо выбрать тот, который дает лучшую форму сигнала. Обычно это положительный коэффициент. Использование отрицательного коэффициента чаще всего дает на выходе функцию, смещенную относительно нуля и перевернутую. В данном примере нужно выбрать коэффициент, равный +0,77295.

Структура фильтра для выбранного примера представлена на рис. 7.2.

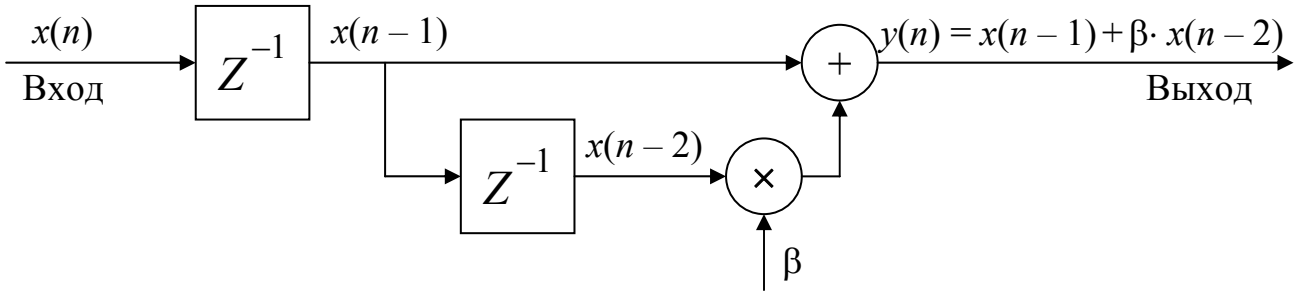

Рис. 7.2. Структура фильтра для сдвига фазы входного сигнала

Если на вход демодулятора подать сигнал, представленный на рис. 6.1, то форма сигнала на выходе умножителя (7.2) будет такой, какая показана на рис. 7.3.

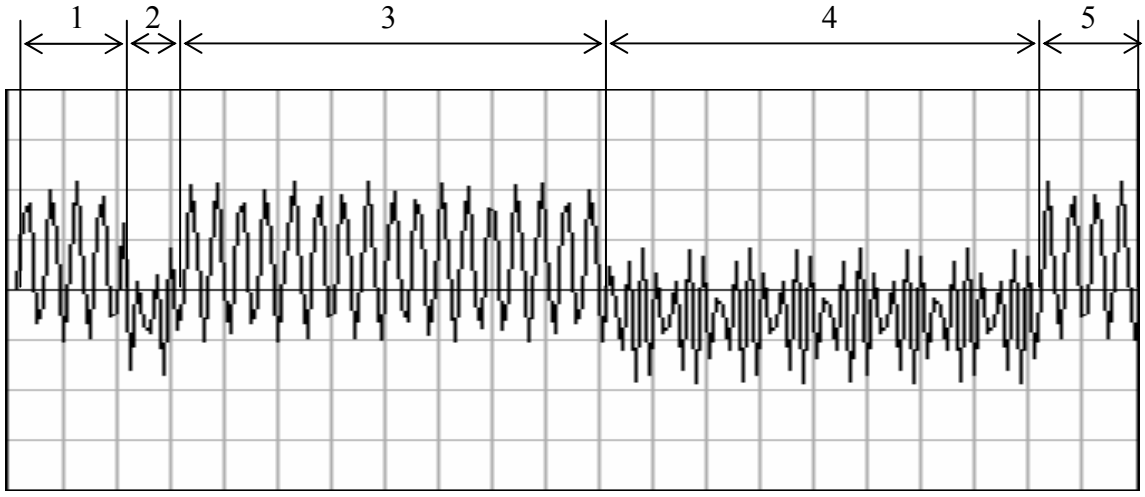

Рис. 7.3. Вид сигнала на выходе умножителя демодулятора

На рис. 7.3 легко можно различить пять отдельных участков сигнала. Участки 1, 3 и 5 соответствуют передаче кода «1» (два «стоповых» бита, восемь битов единичных данных и снова два «стоповых» бита). Эти участки представляют собой сдвинутый вверх и модулированный частотой $2(\omega_c - \delta_\omega)$ сигнал. Участки 2 и 4 соответствуют передаче кода «0» («стартовый» бит и восемь би-

тов нулевых данных). Эти участки представляют собой сдвинутый вниз и модулированный частотой $2(\omega_c+\delta_\omega)$ сигнал.

7.1.2. Замечания по программированию фазосдвигателя и умножителя

При программировании фазосдвигателя и умножителя демодулятора FSK-модема необходимо учесть следующие замечания:

– в качестве входного сигнала демодулятора используется массив данных из памяти, сформированный программой модулятора из предыдущей лабораторной работы;

– поскольку коэффициент β в уравнении (7.3) получается близким к единице или большим единицы (в зависимости от варианта задания), то прямое программирование данного уравнения невозможно;

– для физической реализации уравнения (7.3) на процессоре с фиксированной запятой его необходимо промасштабировать, разделив левую и правую части на 2^m;

– показатель степени m выбирается таким, чтобы значение $\beta/2^m$ получилось меньше единицы;

– если коэффициент β получился близким к единице, но меньше единицы, то уравнение (7.3) также нужно промасштабировать на два, так как при соседних значениях сигнала, близких к единице, в результате вычисления получится значение, превышающее единицу (т. е. переполнение);

– если после масштабирования уравнения (7.3) все равно возникает переполнение, то показатель степени m необходимо увеличить на единицу;

– таким образом, после масштабирования и учитывая, что задержка D находится в диапазоне от 1 до 2, уравнение (7.3) преобразуется к следующему виду: $y(n)/2^m = x(n-1)/2^m + \beta/2^m \cdot x(n-2)$;

– все операнды в преобразованном уравнении по модулю меньше единицы, и результат вычислений также не выходит за рамки диапазона дробных чисел с фиксированной запятой;

– если получаемые на выходе фазосдвигателя значения не выходят за пределы диапазона ±0,5, то перед умножением на входной сигнал их можно увеличить в два раза, сдвинув влево на один разряд;

– если сигнал на выходе умножителя (рис. 7.3) не выходит за пределы диапазона ±0,5, то для повышения помехоустойчивости его также можно увеличить в два раза;

– сигнал после умножителя необходимо записать в память данных в виде массива и отобразить на экране в графическом виде;

– при необходимости для отладки программы формируемый фазосдвигателем сигнал также можно записать в память данных в виде массива и отобразить на экране в графическом виде.

7.2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретические сведения по теме лабораторной работы (подразд. 7.1).
2. Получить у преподавателя задание для выполнения практической части работы.
3. Согласно заданию рассчитать значения всех переменных, необходимых для работы программы.
4. Разработать алгоритм работы программы.
5. Написать, оттранслировать, отладить и выполнить программу.
6. Продемонстрировать результат трансляции и работы программы преподавателю.
7. Оформить и защитить отчет по лабораторной работе.

7.3. СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы и исходные данные.
2. Используемые аналитические формулы и выполненные расчеты.
3. Алгоритм работы программы.
4. Листинг программы с комментариями.
5. Результат работы программы.
6. Выводы по работе.

7.4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Назначение демодулятора модема.
2. Принцип работы демодулятора FSK-модема.
3. Назначение фазосдвигателя демодулятора.
4. Расчет коэффициента фильтра фазосдвигателя.
5. Бит режима умножения дробных чисел.
6. Счетчик повторения блока команд.
7. Команды управления битами состояния.
8. Команды загрузки данных в аккумулятор.
9. Команды загрузки регистров.
10. Команды умножения данных.
11. Команды умножения данных со сложением.
12. Команды для организации линии задержки данных.
13. Команды сохранения данных.
14. Команды пересылки данных.
15. Команды организации циклов.

ЛАБОРАТОРНАЯ РАБОТА №8
ДЕМОДУЛЯТОР FSK-МОДЕМА

ЦЕЛЬ РАБОТЫ:

Разработка алгоритма и программы демодулятора FSK-модема на ассемблере процессора TMS320VC5402 и отладка программы на лабораторном макете TMS320VC5402 DSP Starter Kit.

8.1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

8.1.1. Фильтрация высокочастотных составляющих в демодуляторе FSK-модема

Сигнал с выхода фильтра (фазосдвигателя) $y(n)$ после умножения на входной сигнал демодулятора $x(n)$ поступает на вход фильтра нижних частот (ФНЧ). В качестве ФНЧ можно использовать цифровой эллиптический рекурсивный фильтр четвертого порядка. Расчет коэффициентов фильтра может быть выполнен в системе MATLAB или с использованием специальной программы skif.exe, которая требует задания следующих исходных данных: тип фильтра и аппроксимация амплитудно-частотной характеристики (АЧХ), граничные частоты подавления (*Fa*) и пропускания (*Fp*), максимальное затухание в полосе пропускания (*Ap*) и минимальное затухание в полосе подавления (*Aa*).

Первые два параметра (фильтр нижних частот и эллиптическая аппроксимация) выбираются из списков, а вторые два задаются в долях частоты Найквиста ($f_Н = f_Д / 2 = 4800$ Гц).

Частота пропускания фильтра определяется скоростью передачи информации V:

$$Fp = (V / f_Н) = (1200 / 4800) = 0{,}25 ,$$

а частота подавления – удвоенной минимальной частотой принимаемого сигнала $2 f_1$:

$$Fa = (2 f_1) / f_Н = (2 \cdot 1300) / 4800 = 0{,}54 .$$

Оставшиеся два параметра выбираются по умолчанию ($Ap = 0{,}1$ дБ и $Aa = 45$ дБ).

Программа skif.exe всегда рассчитывает коэффициенты фильтра для каскадного соединения фильтров второго порядка с максимальным коэффициентом передачи на любой частоте не более единицы. Таким образом, для фильтра четвертого порядка должно получиться два каскада (звена). Если при расчете фильтра получается больше двух каскадов, необходимо повторить расчет, немного уменьшив частоту пропускания фильтра или значение затухания в полосе подавления.

При заданных ранее исходных значениях после расчета получаются следующие коэффициенты фильтров, приведенные в табл. 8.1.

Таблица 8.1

Коэффициенты передаточной функции фильтра

Звено	$a0$	$a1$	$a2$	$b0$	$b1$	$b2$
0	0,0785821	0,1232584	0,0785821	1,0000000	–1,0656795	0,3461022
1	0,2611068	0,1030999	0,2611068	1,0000000	–1,0887550	0,7213092

Передаточная функция одного каскада рекурсивного фильтра второго порядка описывается уравнением

$$y(n) = a0 \cdot x(n) + a1 \cdot x(n-1) + a2 \cdot x(n-2) - b1 \cdot y(n-1) - b2 \cdot y(n-2) . \qquad (8.1)$$

Для реализации двухкаскадного фильтра формулу (8.1) необходимо запрограммировать дважды. В качестве входного сигнала первого каскада используется сигнал с выхода умножителя, а в качестве входного сигнала второго каскада – сигнал с выхода первого каскада. Поскольку при поступлении на вход фильтра первого отсчета $x(n)$ значения предыдущих отсчетов входного сигнала $x(n-1), x(n-2)$ и выходного сигнала $y(n-1), y(n-2)$ не определены, то они предварительно должны обнуляться. После каждого такта работы фильтра его предыстория обновляется, т. е. $x(n-1)$ записывается на место $x(n-2)$, $x(n)$ – на место $x(n-1)$, $y(n-1)$ – на место $y(n-2)$ и $y(n)$ – на место $y(n-1)$. Обычно при работе каждого каскада в памяти сдвигаются только входные отсчеты, так как выходные отсчеты будут сдвинуты следующим каскадом, поскольку они для него являются входными. И только после последнего каскада необходимо произвести сдвиг его выходных отсчетов. Если расположить в памяти данные предыстории в следующем порядке: $x(n)$, $x(n-1)$, $x(n-2)$, то их сдвиг можно выполнить с помощью специальной команды **DELAY**, применив ее сначала к ячейке памяти с переменной $x(n-1)$, а затем – с переменной $x(n)$.

Если на вход ФНЧ поступает сигнал с выхода умножителя, представленный на рис. 7.3, то сигнал после ФНЧ будет иметь вид, приведенный на рис. 8.1.

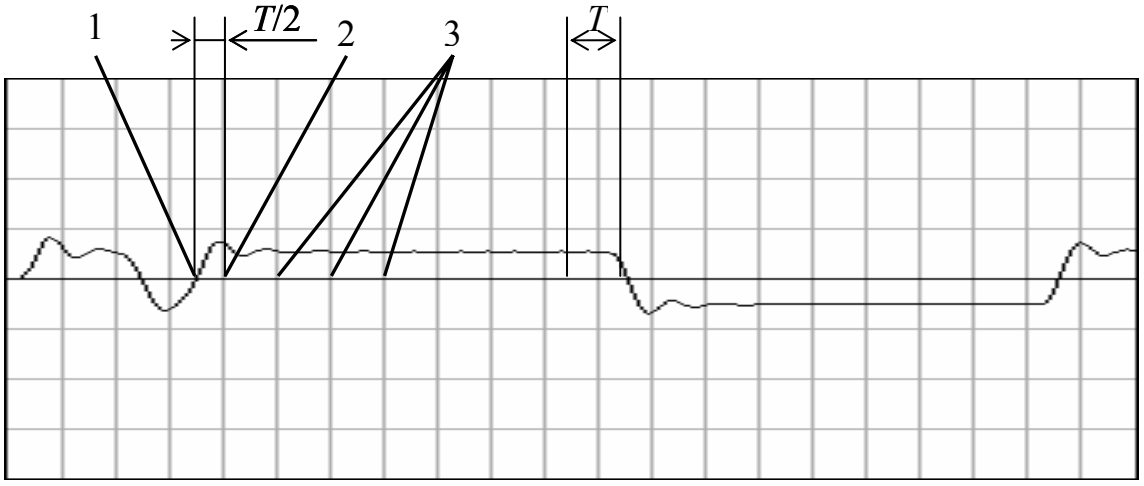

Рис. 8.1. Вид сигнала на выходе ФНЧ демодулятора

На рис. 8.1 вначале видна область установления выходного значения фильтра до уровня приема единичных («стоповых») битов, которая занимает примерно половину такта, соответствующего передаче одного бита информации. Затем в течение двух тактов уровень сигнала выше нуля (два «стоповых» бита). В течение следующего такта уровень сигнала ниже нуля (один «стартовый» бит), затем в течение 16 тактов идет прием данных (сначала восемь единиц, затем восемь нулей) и на последних полутора тактах уровень сигнала снова выше нуля («стоповые» биты).

Обычно программа фильтра пишется для одного каскада с использованием косвенной адресации переменных и коэффициентов. При этом переменные в памяти располагаются таким образом, чтобы все указатели на эти переменные после отработки очередного каскада автоматически указывали на переменные следующего каскада. Тогда остается только зациклить данную программу нужное число раз в зависимости от числа каскадов и в конце выполнить сдвиг выходных данных последнего каскада.

8.1.2. Построение детектора для демодулятора FSK-модема

Последним элементом демодулятора является детектор, который путем подсчета числа положительных и отрицательных отсчетов сигнала после ФНЧ за время передачи одного бита информации определяет, какой бит – «1» или «0» был принят.

Для синхронизации приемника и передатчика каждое слово данных сопровождается битами синхронизации. Передача начинается всегда со «стартового» бита, который в нашем случае равен «0».

При отсутствии передачи данных в линию обычно передаются посылки, соответствующие передаче «1» и на входе детектора будет положительный сигнал.

Первая задача детектора – синхронизировать прием данных. Для этого он должен четко определить момент приема «стартового» бита, которому соответствует последовательность из n отрицательных отсчетов (в нашем примере значение $n = 8$). Определение момента приема «стартового» бита можно реализовать с помощью простейшего счетчика, который в случае положительного входного сигнала обнуляется, а в случае отрицательного – увеличивается на единицу. Момент достижения счетчиком значения n и будет определять завершение приема «стартового» бита (см. рис. 8.1, позиция 1).

После приема «стартового» бита детектор должен принять заданное число битов данных (в нашем случае 16). Детектирование и прием битов данных можно выполнять разными способами. Рассмотрим для примера *два простых способа детектирования данных*.

Первый способ заключается в анализе амплитуды и знака сигнала в середине интервала приема очередного бита данных. Для этого после приема «стартового» бита необходимо пропустить $n/2$ отсчетов (рис. 8.1, позиция 2), чтобы попасть точно на середину интервала приема первого бита данных и

проанализировать знак сигнала на выходе ФНЧ. Если сигнал положительный, то первый принимаемый бит – «1», если отрицательный – «0». Для обеспечения лучшей помехозащищенности можно проанализировать также амплитуду сигнала, которая должна быть выше определенного порога. Для детектирования каждого следующего бита данных необходимо пропускать n отсчетов, чтобы попадать на середину интервала приема очередного бита данных (рис. 8.1, позиция 3), и снова анализировать знак сигнала на выходе ФНЧ.

Второй способ заключается в подсчете положительных и отрицательных значений сигнала на интервале приема очередного бита данных T. Для этого после приема «стартового» бита необходимо в течение n тактов считать число положительных и отрицательных отсчетов сигнала. Для этого в свою очередь можно использовать один счетчик, который при положительном сигнале увеличивается на единицу, а при отрицательном – уменьшается. Если после n тактов в счетчике будет положительное значение, то принимаемый бит – «1», если отрицательное – «0». Для обеспечения лучшей помехозащищенности можно проанализировать и абсолютное значение счетчика, которое должно быть равно n (все отсчеты одного знака), $n-2$ (один отсчет другого знака) или $n-4$ (два отсчета другого знака).

При любом способе детектирования необходимо также сформировать слово из принимаемых битов. Для этого можно использовать различные команды сдвига, устанавливая предварительно нужное состояние бита переноса по результатам работы детектора.

8.1.3. Замечания по программированию фильтра и детектора

При программировании фильтра и детектора демодулятора FSK-модема необходимо учесть следующие замечания:

– поскольку обычно коэффициент $b1$ в уравнении (8.1) по модулю больше единицы (см. табл. 8.1), то прямое программирование данного уравнения невозможно;

– для физической реализации уравнения (8.1) произведение $b1 \cdot y(n-1)$ необходимо разбить на две части: $b1/2 \cdot y(n-1)$ и $b1/2 \cdot y(n-1)$;

– поскольку при суммировании шести произведений в преобразованном уравнении (8.1) может произойти переполнение промежуточных результатов, то программа фильтра должна выполняться при выключенном бите **OVM**;

– так как из-за погрешностей задания коэффициентов и вычислений может произойти незначительное переполнение окончательного результата на выходе фильтра, необходимо установить бит **SST** в регистре режима работы процессора **PMST** (коррекция переполнения при записи результата в память);

– при любом способе детектирования данных выделение «стартового» бита производится путем обнаружения непрерывной последовательности из отрицательных значений сигнала на выходе ФНЧ, длина которой равна (или несколько меньше) числу отсчетов синуса на один бит передаваемых данных.

8.2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретические сведения по теме лабораторной работы (подразд. 8.1).
2. Получить у преподавателя задание для выполнения практической части работы.
3. Согласно заданию рассчитать значения всех переменных, необходимых для работы программы.
4. Разработать алгоритм работы программы.
5. Написать, оттранслировать, отладить и выполнить программу.
6. Продемонстрировать результат трансляции и работы программы преподавателю.
7. Оформить и защитить отчет по лабораторной работе.

8.3. СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы и исходные данные.
2. Используемые аналитические формулы и выполненные расчеты.
3. Алгоритм работы программы.
4. Листинг программы с комментариями.
5. Результат работы программы.
6. Выводы по работе.

8.4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Назначение демодулятора модема.
2. Принцип работы демодулятора FSK-модема.
3. Назначение фильтра нижних частот демодулятора.
4. Определение полосы пропускания и подавления ФНЧ.
5. Расчет коэффициентов фильтра нижних частот.
6. Способы детектирования принятых данных.
7. Бит режима умножения дробных чисел.
8. Команды загрузки данных в аккумулятор.
9. Команды умножения данных.
10. Команды для организации линии задержки данных.
11. Команды сохранения данных.
12. Команды модификации регистров.
13. Команды управления битами состояния.
14. Команды условных переходов.
15. Команды сдвига данных.
16. Команды организации циклов.

ПРИЛОЖЕНИЕ 1

Работа с системой 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.

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 масштабы 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.1.1.

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

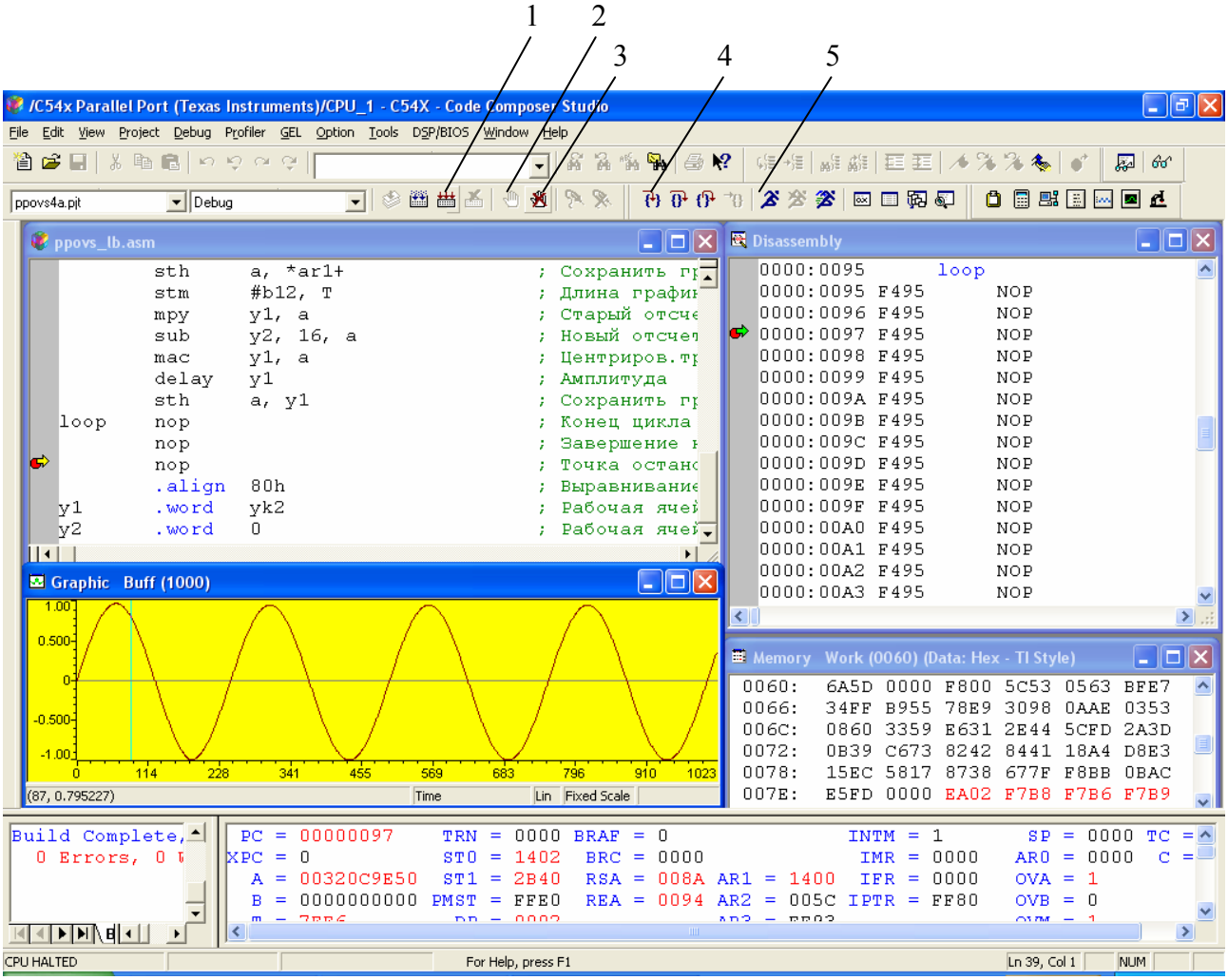

Рис. П.1.1. Примерный вид окна программы CCS

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.1.1, 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.1.1, 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.1.1, 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.1.1, 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.1.1, 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 С, – 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.1.1, 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. Несвязанная контрольная точка (No Connection) будет связана с выбранным файлом.

44. Для эмуляции ВВОДА необходимо установить контрольную точку перед командой ввода (для последовательного порта за две 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, вводимых 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 2

Обозначения и сокращения в описаниях команд

Символ	Значение
A	Аккумулятор A
ALU	Арифметическо-логическое устройство
AR	Вспомогательный регистр
ARx	Определяет указанный вспомогательный регистр (0…7)
ARP	Указатель текущего вспомогательного регистра (3-битное поле в ST0)
ASM	5-битное поле режима сдвига аккумулятора в ST1 (–16…15)
B	Аккумулятор B
BK	Регистр размера кольцевого буфера
BRAF	Флажок активности повторения блока в ST1
BRC	Счетчик повторения блока
BITC	4-битное значение, № бита, проверяемого командой BITC (0…15)
C16	Бит режима двухсловной/двойной точности арифметики в ST1
C	Бит переноса в ST0
CC	2-битный код условия (0…3)
CMPT	Бит режима совместимости в ST1
CPL	Бит режима трансляции в ST1
cond	Условие, проверяемое командами условного выполнения
[D]	Режим задержки
DAB	Шина адреса D
DAR	Регистр адреса D
dmad	Непосредственный 16-битный адрес памяти данных (0…65535)
Dmem	Операнд в памяти данных
DP	9-битное поле указателя страницы памяти данных в ST0 (0…511)
dst	Аккумулятор-приемник (A или B)
dst _	Противоположный аккумулятор: если dst = A, то dst_ = B и наоборот
EAB	Шина адреса E
EAR	Регистр адреса E
extpmad	Непосредственный 19-битный адрес памяти программ
FRCT	Бит режима дробных чисел в ST1
hi (A)	Старшая часть аккумулятора A (биты 31–16)
HM	Бит режима захвата (HOLD-режима) в ST1
IFR	Регистр флагов прерываний
IMR	Регистр масок прерываний
INTM	Бит запрещения (маскирования) прерываний в ST1
K	Короткое непосредственное значение (константа) меньше, чем 9 битов
K3	3-битное непосредственное значение (0…7)
K5	5-битное непосредственное значение (–16…15)
K9	9-битное непосредственное значение (0…511)
lk	16-битное длинное непосредственное значение
Lmem	32-битный операнд в памяти данных с адресацией двойного слова
mmr, MMR	Любой регистр, отображаемый на память
MMRx, MMRy	Регистр, отображаемый на память: AR0 – AR7 или SP
n	Число слов после команды XC: n = 1 или 2
N	Регистр состояния (0 или 1), изменяемый в командах RSBX, SSBX

Символ	Значение
OVA	Флаг переполнения аккумулятора A в ST0
OVB	Флаг переполнения аккумулятора B в ST0
OVdst	Флаг переполнения аккумулятора приемника (A или B)
OVdst_	Флаг переполнения аккумулятора, противоположного приемнику
OVsrc	Флаг переполнения аккумулятора источника (A или B)
OVM	Бит режима переполнения в ST1
PA	Непосредственный 16-битный адрес порта (0…65 535)
PAR	Регистр адреса программы
PC	Программный счетчик
pmad	Непосредственный 16-битный адрес памяти программ (0…65 535)
Pmem	Операнд в памяти программ
PMST	Регистр состояния режима процессора
prog	Операнд в памяти программ
[R]	Режим округления
RC	Счетчик повторения
REA	Регистр конечного адреса повторяемого блока
rnd	Округление
RSA	Регистр начального адреса повторяемого блока
RTN	Регистр быстрого возврата, используемый в команде RETF[D]
SBIT	Разряд ST0, ST1, изменяемый в командах RSBX, SSBX (0…15)
SHFT	4-битное значение сдвига (0…15)
SHIFT	5-битное значение сдвига (–16…15)
Sind	Операнд, использующий косвенную адресацию
Smem	16-битный операнд в памяти данных одиночного доступа
SP	Указатель стека
src	Аккумулятор-источник (A или B)
ST0	Регистр состояния 0
ST1	Регистр состояния 1
SXM	Бит режима расширения знака в ST1
T	Временный регистр
TC	Флаг проверки/управления в ST0
TOS	Вершина стека
TRN	Переходный регистр
TS	Значение сдвига, указанное битами 5–0 регистра T (–16…31)
uns	Без знака
XF	Бит внешнего флага в регистре состояния ST1
XPC	Регистр расширения программного счетчика
Xmem	16-битный операнд в памяти данных двойного доступа, используемый в двухоперандных и некоторых однооперандных командах
Ymem	16-битный операнд в памяти данных двойного доступа, используемый в двухоперандных командах
– – SP	Значение указателя стека уменьшается на 1
++ SP	Значение указателя стека увеличивается на 1
++ PC	Значение программного счетчика увеличивается на 1

ПРИЛОЖЕНИЕ 3

Директивы ассемблера для процессора TMS320VC5420

При написании программ на ассемблере кроме команд процессора 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 внутри цикла при повторении команд программы или директив ассемблера с изменяющимися параметрами. Для организации цикла применяются специальные директивы **.loop** – начало цикла и **.endloop** – конец цикла. Директива **.loop** имеет один параметр – количество повторений цикла. В отличие от команд процессора, в которых число повторений всегда на единицу больше, чем указано, в данной директиве число повторений совпадает с заданным.

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 8 раз
        .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:
[имя] .word значение1[, значение2, ..., значениеN] ;

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

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

г) директива **.space** – зарезервировать заданное число битов, формат:
[имя] .space число битов .

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 восьми значащих битов, и если значение выходит за пределы восьми 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.

Директива **.space** (**.bes**) резервирует заданное число битов в памяти. При этом резервируется минимальное число слов, содержащих указанное число битов.

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
                                 ; данные в области программ
    .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            ; область данных
                .align 200h      ; выравнивание на 512
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).

ПРИЛОЖЕНИЕ 4

Методы адресации и модификации косвенного адреса

Поле MOD	Синтаксис операнда	Функции	Описание
Одинарный операнд (Smem)			
0000 (0)	*ARx	адрес = ARx	ARx содержит адрес памяти данных
0001 (1)	*ARx-	адрес = ARx ARx = ARx - 1	После доступа к памяти адрес в ARx уменьшается на 1
0010 (2)	*ARx+	адрес = ARx ARx = ARx + 1	После доступа к памяти адрес в ARx увеличивается на 1
0011 (3)	*+ARx	ARx = ARx + 1 адрес = ARx	Перед доступом адрес в ARx увеличивается на 1, и новый адрес используется для адресации операнда (только для записи)
0100 (4)	*ARx-0B	адрес = ARx ARx = B(ARx-AR0)	После доступа AR0 вычитается из ARx с обратным переносом (rc)
0101 (5)	*ARx-0	адрес = ARx ARx = ARx - AR0	После доступа AR0 вычитается из ARx
0110 (6)	*ARx+0	адрес = ARx ARx = ARx + AR0	После доступа AR0 прибавляется к ARx
0111 (7)	*ARx+0B	адрес = ARx ARx = B(ARx+AR0)	После доступа AR0 прибавляется к ARx с обратным переносом (rc)
1000 (8)	*ARx-%	адрес = ARx ARx = circ (ARx-1)	После доступа адрес в ARx уменьшается с использованием циклической адресации
1001 (9)	*ARx-0%	адрес = ARx ARx=circ(ARx-AR0)	После доступа AR0 вычитается из ARx с использованием циклической адресации
1010 (10)	*ARx +%	адрес = ARx ARx = circ (ARx+1)	После доступа адрес в ARx увеличивается с использованием циклической адресации
1011 (11)	*ARx+0%	адрес = ARx ARx=circ(ARx+AR0)	После доступа AR0 прибавляется к ARx с использованием циклической адресации
1100 (12)	*ARx(lk)	адрес = ARx + lk ARx = ARx	Сумма ARx и смещения длиной 16 битов (lk) используется как адрес памяти данных. ARx не модифицируется
1101 (13)	*+ARx(lk)	ARx = ARx + lk адрес = ARx	Перед доступом адрес в ARx увеличивается на lk, и новый адрес используется для адресации операнда памяти данных
1110 (14)	*+ARx(lk)%	ARx =circ (ARx+lk) адрес = ARx	Перед доступом адрес в ARx увеличивается на lk с использованием циклической адресации, и новый адрес используется для адресации операнда памяти данных
1111 (15)	* (lk)	адрес = lk	Смещение длиной 16 битов без знака (lk) используется как абсолютный адрес памяти данных (абсолютная адресация)
Двойной операнд (Xmem или Ymem) – используются только регистры AR2, AR3, AR4, AR5			
00 (0)	*ARx	адрес = ARx	ARx содержит адрес памяти данных
01 (1)	*ARx-	адрес = ARx ARx = ARx - 1	После доступа к памяти адрес в ARx уменьшается на 1
10 (2)	*ARx+	адрес = ARx ARx = ARx + 1	После доступа к памяти адрес в ARx увеличивается на 1
11 (3)	*ARx+0%	адрес = ARx ARx=circ(ARx+AR0)	После доступа AR0 прибавляется к ARx с использованием циклической адресации

ПРИЛОЖЕНИЕ 5

Система команд процессора TMS320VC5402

Синтаксис	Выражение (операция)	С†	Ц†	Зависит от	Влияет на	Класс
Арифметические операции						
Команды сложения						
ADD Smem, src	src = src + Smem {src += Smem}	1	1	SXM, OVM	C, OV↑	3A, 3B
ADD Smem, TS, src	src = src + Smem << TS {src += …}	1	1	SXM, OVM	C, OV↑	3A, 3B
ADD Smem, 16, src [, dst]	dst = src + Smem << 16 {dst += …}	1	1	SXM, OVM	C↑, OV↑	3A, 3B
ADD Smem [,SHIFT], src [,dst]	dst = src + Smem << SHIFT {dst += …}	2	2	SXM, OVM	C, OV↑	4A, 4B
ADD Xmem, SHFT, src	src = src + Xmem << SHFT {src += …}	1	1	SXM, OVM	C, OV↑	3A
ADD Xmem, Ymem, dst	dst = Xmem << 16 + Ymem << 16	1	1	SXM, OVM	C, OV↑	7
ADD #lk [,SHFT], src [,dst]	dst = src + #lk << SHFT {dst += …}	2	2	SXM, OVM	C, OV↑	2
ADD #lk, 16, src [, dst]	dst = src + #lk << 16 {dst += …}	2	2	SXM, OVM	C, OV↑	2
ADD src [, SHIFT] [, dst]	dst = dst + src << SHIFT {dst += …}	1	1	SXM, OVM	C, OV↑	1
ADD src, ASM [, dst]	dst = dst + src << ASM {dst += …}	1	1	SXM, OVM	C, OV↑	1
ADDC Smem, src	src = src + Smem + C {+CARRY} {src += …}	1	1	C, OVM	C, OV↑	3A, 3B
ADDM #lk, Smem	Smem = Smem + #lk {Smem += #lk}	2	2	SXM, OVM	C, OV↑	18A,18B
ADDS Smem, src	src = src + uns(Smem) {src += …}	1	1	OVM	C, OV↑	3A, 3B
Команды вычитания						
SUB Smem, src	src = src – Smem {src -= Smem}	1	1	SXM, OVM	C, OV↑	3A, 3B
SUB Smem, TS, src	src = src – Smem << TS {src -= …}	1	1	SXM, OVM	C, OV↑	3A, 3B
SUB Smem, 16, src [, dst]	dst = src – Smem << 16 {dst -= …}	1	1	SXM, OVM	C↓, OV↑	3A, 3B
SUB Smem [,SHIFT], src [,dst]	dst = src – Smem << SHIFT {dst -= …}	2	2	SXM, OVM	C, OV↑	4A, 4B
SUB Xmem, SHFT, src	src = src – Xmem << SHFT {src -= …}	1	1	SXM, OVM	C, OV↑	3A
SUB Xmem, Ymem, dst	dst = Xmem << 16 – Ymem << 16	1	1	SXM, OVM	C, OV↑	7
SUB #lk [, SHFT], src [, dst]	dst = src – #lk << SHFT {dst -= …}	2	2	SXM, OVM	C, OV↑	2
SUB #lk, 16, src [, dst]	dst = src – #lk <<16 {dst -= …}	2	2	SXM, OVM	C, OV↑	2
SUB src[, SHIFT] [, dst]	dst = dst – src << SHIFT {dst -= …}	1	1	SXM, OVM	C, OV↑	1
SUB src, ASM [, dst]	dst = dst – src << ASM {dst -= …}	1	1	SXM, OVM	C, OV↑	1

Синтаксис	Выражение (операция)	С†	Ц†	Зависит от	Влияет на	Класс
SUBB Smem, src	src = src – Smem – ~C (-BORROW) {src -= …}	1	1	C, OVM	C, OV↑	3A, 3B
SUBC Smem, src {SUBC (Smem, src)}	Если (src – Smem << 15) ≥ 0 src = (src – Smem<<15) << 1 + 1 Иначе src = src << 1	1	1	SXM	C, OV↑	3A, 3B
SUBS Smem, src	src = src – uns(Smem) {src -= …}	1	1	OVM	C, OV↑	3A, 3B
Команды умножения						
MPY Smem, dst	dst = T * Smem	1	1	FRCT, OVM	OV↑	3A, 3B
MPYR Smem, dst	dst = rnd(T * Smem)	1	1	FRCT, OVM	OV↑	3A, 3B
MPY Xmem, Ymem, dst	dst = Xmem * Ymem, T = Xmem	1	1	FRCT, OVM	OV↑	7
MPY Smem, #lk, dst	dst = Smem * #lk , T = Smem	2	2	FRCT, OVM	OV↑	6A, 6B
MPY #lk, dst	dst = T * #lk	2	2	FRCT, OVM	OV↑	2
MPYA dst	dst = T * A(32-16) {dst = T * hi(A)}	1	1	FRCT, OVM	OV↑	1
MPYA Smem	B = Smem * A(32-16), T = Smem {B = Smem * hi(A)[, T = Smem]}	1	1	FRCT, OVM	OVB↑	3A, 3B
MPYU Smem, dst	dst = uns(T) * uns(Smem) {dst = T * uns(Smem)}	1	1	FRCT, OVM	OV↑	3A, 3B
SQUR Smem, dst	dst = Smem * Smem, T = Smem {dst = square()}	1	1	FRCT, OVM	OV↑	3A, 3B
SQUR A, dst	dst = A(32-16) * A(32-16) {dst = hi(A)*hi(a)} {dst = square()}	1	1	FRCT, OVM	OV↑	1
Команды умножения со сложением и вычитанием						
MAC Smem, src	src = src + T * Smem {src += T*Smem}	1	1	FRCT, OVM	OV↑	3A, 3B
MAC Xmem, Ymem, src [, dst]	dst = src + Xmem * Ymem, {dst += …} T = Xmem	1	1	FRCT, OVM	OV↑	7
MAC # lk, src [, dst]	dst = src + T * #lk {dst += …}	2	2	FRCT, OVM	OV↑	2

MAC Smem, # lk, src [, dst]	dst = src + Smem * #lk, {dst += …}
	T = Smem
		2	2	FRCT, OVM	OV↑	6A, 6B

MACR Smem, src	src = rnd(src + T * Smem)	1	1	FRCT, OVM	OV↑	3A, 3B
MACR Xmem, Ymem, src [, dst]	dst = rnd(src + Xmem * Ymem), T = Xmem	1	1	FRCT, OVM	OV↑	7
MACA Smem [, B]	B = B + Smem * A(32-16), T = Smem {B = B + Smem * hi(A)} {B += …}	1	1	FRCT, OVM	OVB↑	3A, 3B

Синтаксис	Выражение (операция)	С†	Ц†	Зависит от	Влияет на	Класс
MACA T, src [, dst]	dst = src + T * A(32-16) {dst = src + T * hi(A)} {dst += …}	1	1	FRCT, OVM	OV↑	1
MACAR Smem [, B]	B = rnd(B + Smem * A(32-16)), T = Smem {B = rnd(B + Smem * hi(A))}[, T = Smem]}	1	1	FRCT, OVM	OVB↑	3A, 3B
MACAR T, src [, dst]	dst = rnd(src + T * A(32-16)) {dst = rnd(src + T * hi(A))}	1	1	FRCT, OVM	OV↑	1
MACD Smem, pmad, src {MACD (Smem, pmad, src)}	PAR = pmad, src = src + Smem * Pmem(PAR), T = Smem, PAR = PAR + 1, (Smem + 1) = Smem	2	3	FRCT, OVM	OV↑	23A,23B
MACP Smem, pmad, src {MACP (Smem, pmad, src)}	PAR = pmad, src = src + Smem * Pmem(PAR), T = Smem, PAR = PAR + 1	2	3	FRCT, OVM	OV↑	22A,22B
MACSU Xmem, Ymem, src	src = src + uns(Xmem) * Ymem, {src += …} T = Xmem	1	1	FRCT, OVM	OV↑	7
MAS Smem, src	src = src – T * Smem {src -= …}	1	1	FRCT, OVM	OV↑	3A, 3B
MASR Smem, src	src = rnd(src – T * Smem)	1	1	FRCT, OVM	OV↑	3A, 3B
MAS Xmem, Ymem, src [, dst]	dst = src – Xmem * Ymem, {dst -= …} T = Xmem	1	1	FRCT, OVM	OV↑	7
MASR Xmem, Ymem, src [, dst]	dst = rnd(src – Xmem * Ymem), T = Xmem	1	1	FRCT, OVM	OV↑	7
MASA Smem [, B]	B = B – Smem * A(32-16), T = Smem {B = B - Smem * hi(A)} {B -= …}	1	1	FRCT, OVM	OVB↑	3A, 3B
MASA T, src [, dst]	dst = src – T * A(32-16) {dst = src – T * hi(A)} {dst -= …}	1	1	FRCT, OVM	OV↑	1
MASAR T, src [, dst]	dst = rnd(src – T * A(32-16)) {dst = rnd(src – T * hi(A)}	1	1	FRCT, OVM	OV↑	1
SQURA Smem, src	src = src + Smem * Smem, T = Smem {dst = src + square(Smem)} {dst += square()}	1	1	FRCT, OVM	OV↑	3A, 3B
SQURS Smem, src	src = src – Smem * Smem, T = Smem {dst = src - square(Smem)} {dst -= square()}	1	1	FRCT, OVM	OV↑	3A, 3B
Команды с двойным операндом						
DADD Lmem, src [, dst] {dst = src + dbl(Lmem)} {dst += …} {dst = src + dual(Lmem)} {dst += …}	Если C16 = 0 dst = Lmem + src, иначе dst(39-16) = Lmem(31-16) + src(39-16) dst(15-0) = Lmem(15-0) + src(15-0)	1	1	SXM, OVM (если C16=0)	C, OV↑	9A, 9B
DADST Lmem, dst {dst = dadst(Lmem, T)}	Если C16 = 0 dst = Lmem + (T<<16 + T), иначе dst(39-16) = Lmem(31-16)+ T dst(15-0) = Lmem(15-0) – T	1	1	SXM, OVM (если C16=0)	C, OV↑	9A, 9B

Синтаксис	Выражение (операция)	С†	Ц†	Зависит от	Влияет на	Класс
DRSUB Lmem, src {src = dbl(Lmem) – src} {src = dual(Lmem) – src}	Если C16 = 0 src = Lmem – src, иначе src(39-16) = Lmem(31-16) – src(39-16) src(15-0) = Lmem(15-0) – src(15-0)	1	1	SXM, OVM (если C16=0)	C, OV↑	9A, 9B
DSADT Lmem, dst {dst = dsadt (Lmem, T)}	Если C16 = 0 dst = Lmem – (T<<16 + T), иначе dst(39-16) = Lmem(31-16) – T dst(15-0) = Lmem(15-0) + T	1	1	SXM, OVM (если C16=0)	C, OV↑	9A, 9B
DSUB Lmem, src {src = src - dbl(Lmem)} {src -= …} {src = src - dual(Lmem)} {src -= …}	Если C16 = 0 src = src – Lmem, иначе src(39-16) = src(39-16) – Lmem(31-16) src(15-0) =src(15-0) –Lmem(15-0)	1	1	SXM, OVM (если C16=0)	C, OV↑	9A, 9B
DSUBT Lmem, dst {dst = dbl(Lmem) – T} {dst = dual(Lmem) – T}	Если C16 = 0 dst = Lmem – (T<<16 + T), иначе dst(39-16) = Lmem(31-16) – T dst(15-0) = Lmem(15-0) – T	1	1	SXM, OVM (если C16=0)	C, OV↑	9A, 9B
Проблемно-ориентированные команды						
ABDST Xmem, Ymem {ABDST (Xmem, Ymem)}	B = B + |A(32-16)| A = (Xmem – Ymem) << 16	1	1	SXM, FRCT, OVM	C, OVA↑,OVB↑	7
ABS src [, dst]	dst = |src|	1	1	OVM	C, OV↑	1
CMPL src [, dst]	dst = ~src	1	1			1
DELAY Smem {delay(Smem)}	(Smem + 1) = Smem	1	1			24A,24B
EXP src {T = exp(src)}	T = число незначащих битов в (src) – 8	1	1			1
FIRS Xmem, Ymem, pmad {firs(Xmem, Ymem, pmad)}	PAR = pmad, B = B + A(32+16) * Pmem(PAR), A = (Xmem + Ymem) << 16, PAR = PAR + 1	2	3	SXM, FRCT, OVM	C, OVA↑,OVB↑	8
LMS Xmem, Ymem {lms(Xmem, Ymem)}	B = B + Xmem * Ymem A = A + Xmem << 16 + 2^{15}	1	1	SXM, FRCT, OVM	C, OVA↑,OVB↑	7
MAX dst	dst = max(A, B)	1	1		C	1
MIN dst	dst = min(A, B)	1	1		C	1
NEG src [, dst]	dst = –src	1	1	SXM, OVM	C, OV↑	1
NORM src [, dst]	dst = src << TS {dst = norm(src, TS)}	1	1	SXM, OVM	C, OV↑	1
POLY Smem {poly(Smem)}	B = Smem << 16 A = rnd(A(32-16) * T + B)	1	1	SXM, FRCT, OVM	C, OVA↑	3A, 3B
RND src [, dst]	dst = src + 2^{15} {dst = rnd(src)}	1	1	OVM	C, OV↑	1
SAT src {saturate(src)}	Насыщение src до 32 бит [все защитные разряды равны знаковому]	1	1	no SXM, no OVM	OV↑	1

Синтаксис	Выражение (операция)	С†	Ц†	Зависит от	Влияет на	Класс
SQDST Xmem, Ymem {sqdst(Xmem, Ymem)}	B = B + A(32-16) * A(32-16) A = (Xmem – Ymem) << 16	1	1	SXM, FRCT, OVM	C, OVA↑,OVB↑	7
Логические операции						
Команды И						
AND Smem, src	src = src & Smem {src &= Smem}	1	1			3A, 3B
AND # lk [, SHFT], src [, dst]	dst = src & #lk << SHFT {src &= …}	2	2			2
AND # lk, 16, src [, dst]	dst = src & #lk << 16 {src &= …}	2	2			2
AND src [, SHIFT] [, dst]	dst = dst & src << SHIFT {src &= …}	1	1			1
ANDM # lk, Smem	Smem = Smem & #lk {src &= …}	2	2			18A,18B
Команды ИЛИ						
OR Smem, src	src = src | Smem {src |= Smem}	1	1			3A, 3B
OR # lk [, SHFT], src [, dst]	dst = src | #lk << SHFT {src |= …}	2	2			2
OR # lk, 16, src [, dst]	dst = src | #lk << 16 {src |= …}	2	2			2
OR src [, SHIFT] [, dst]	dst = dst | src << SHIFT {src |= …}	1	1			1
ORM # lk, Smem	Smem = Smem | #lk {src |= …}	2	2			18A,18B
Команды Исключающее ИЛИ						
XOR Smem, src	src = src ^ Smem {src ^= Smem}	1	1			3A, 3B
XOR # lk [, SHFT], src [,dst]	dst = src ^ #lk << SHFT {src ^= …}	2	2			2
XOR # lk, 16, src [, dst]	dst = src ^ #lk << 16 {src ^= …}	2	2			2
XOR src [, SHIFT] [, dst]	dst = dst ^ src << SHIFT {src ^= …}	1	1			1
XORM # lk, Smem	Smem = Smem ^ #lk {src ^= …}	2	2			18A,18B
Команды проверки и сравнения						
BIT Xmem, BITC {TC = bit(Xmem, bitc)}	TC = Xmem(15 – BITC)	1	1		TC	3A
BITF Smem, #lk {TC = bitf(Smem, #lk)}	TC = (Smem & #lk)	2	2		TC	6A, 6B
BITT Smem {TC = bitt(Smem)}	TC = Smem(15 – T(3-0))	1	1		TC	3A, 3B
CMPM Smem, #lk	TC = (Smem == #lk)	2	2		TC	6A, 6B
CMPR CC, ARx CC= EQ,LT,GT,NEQ {TC=(AR0 ? ARx)} ? = ==,> ,< ,!=	Сравнение ARx с AR0 TC отражает выполнение условия сравнения	1	1		TC	1

Синтаксис	Выражение (операция)	С†	Ц†	Зависит от	Влияет на	Класс
	Команды сдвига					
ROL src {src = src\\CARRY}	Циклический сдвиг src(31-0) влево через C	1	1	C	C	1
ROLTC src {roltc (src)}	Сдвиг влево src(31-0) из TC	1	1	TC	C	1
ROR src {src = src//CARRY}	Циклический сдвиг src(31-0) вправо через C	1	1	C	C	1
SFTA src, SHIFT [, dst] {src = src <<C SHIFT}	dst = src(39-0) << SHIFT {арифметический}	1	1	SXM, OVM	C, OV↑	1
SFTC src {shiftc (src)}	Если src(31)=src(30), то src=src(39-0)<<1, TC=0 иначе (или если src=0) TC=1	1	1		TC	1
SFTL src, SHIFT [, dst] {src = src <<< SHIFT}	dst = src(31-0) << SHIFT {логический}	1	1		C	1
	Команды управления ходом выполнения программы					
	Команды перехода					
B[D] pmad {[d]goto pmad}	Безусловный переход PC = pmad(15-0)	2	4/ 2¶			29A
BACC[D] src {[d]goto src}	Безусловный переход по аккумулятору PC = src(15-0)	1	6/ 4¶			30A
BANZ[D] pmad, Sind {if(Sind!=0) [d]goto pmad}	Условный переход по счетчику в ARx Если ARx≠0, то PC = pmad(15-0)	2	4‡/ 2§¶			29A
BC[D] pmad,cond [,cond[,cond]] {if (cond(s)) [d]goto pmad}	Условный переход Если cond(s)=ИСТИНА, то PC = pmad(15-0)	2	5‡/ 3§¶	cond	OV↓ (cond=OV)	31A
FB[D] extpmad {far [d]goto extpmad}	Дальний безусловный переход PC = extpmad(15-0), XPC = extpmad(19-16)	2	4/ 2¶			29A
FBACC[D] src {far [d]goto src}	Дальний переход по аккумулятору PC = src(15-0), XPC = src(19-16)	1	6/ 4¶			30A
	Команды вызова подпрограмм					
CALA[D] src {[d]call src}	Вызов подпрограммы по аккумулятору --SP, TOS = PC + 1[3¶], PC = src(15-0)	1	6/ 4¶			30B
CALL[D] pmad {[d]call pmad}	Вызов подпрограммы --SP, TOS = PC + 2[4¶], PC = pmad(15-0)	2	4/ 2§			29B
CC[D] pmad, cond [,cond[,cond]] {if (cond(s)) [d]call pmad}	Условный вызов подпрограммы Если cond(s)=ИСТИНА, то --SP, TOS = PC + 2[4¶], PC = pmad(15-0)	2	5‡/ 3§¶	cond	OV↓ (cond=OV)	31B

Синтаксис	Выражение (операция)	С†	Ц†	Зависит от	Влияет на	Класс
FCALL[D] extpmad {far [d]call extpmad}	Дальний вызов подпрограммы --SP, TOS = PC + 2[4¶], --SP, TOS = XPC, PC = extpmad(15-0), XPC = extpmad(19-16)	2	4/ 2¶			29B
FCALA[D] src {far [d]call src}	Дальний вызов по аккумулятору --SP, TOS = PC + 1[3¶], --SP, TOS = XPC, PC = src(15-0), XPC = src(19-16)	1	6/ 4¶			30B
Команды прерывания						
INTR K {int(k)}	--SP, TOS = ++PC, PC = IPTR(15-7) + K<<2, INTM = 1	1	3		INTM↑	35
TRAP K {trap(k)}	--SP, TOS = ++PC, PC = IPTR(15-7) + K<<2	1	3			35
Команды возврата						
FRET[D] {far [d]return}	Дальний возврат из подпрограммы XPC = TOS, ++SP, PC = TOS, ++SP	1	6/ 4¶			34
FRETE[D] {far [d]return_enable}	Дальний возврат с разрешением прерывания XPC = TOS, ++SP, PC = TOS, ++SP, INTM = 0	1	6/ 4¶		INTM↓	34
RC[D] cond [, cond [, cond]] {if (cond(s)) [d]return}	Условный возврат из подпрограммы Если cond(s)=ИСТИНА, то PC = TOS, ++SP	1	5‡/ 3§¶	cond		32
RET[D] {[d]return}	Возврат из подпрограммы PC = TOS, ++SP	1	5/ 3¶			32
RETE[D] {[d]return_enable}	Возврат из прерывания PC = TOS, ++SP, INTM = 0	1	5/ 3¶		INTM↓	32
RETF[D] {[d]return_fast}	Быстрый возврат из прерывания PC = RTN, ++SP, INTM = 0	1	3/ 1¶		INTM↓	33
Команды повторения						
RPT Smem {repeat(Smem)}	Повторение одной команды: RC = Smem	1	3			5A, 5B
RPT #K {repeat(#k)}	Повторение одной команды: RC = #K	1	1			1
RPT #lk {repeat(#lk)}	Повторение одной команды: RC = #lk	2	2			2
RPTB[D] pmad {[d]blockrepeat(pmad)}	Повторение блока команд RSA = PC + 2[4¶], REA = pmad, BRAF = 1	2	4/ 2¶		BRAF↑	29A
RPTZ dst, #lk {re- peat(#lk),dst=0}	Повторение одной команды: RC = #lk, dst = 0	2	2			2

Синтаксис	Выражение (операция)	С†	Ц†	Зависит от	Влияет на	Класс
	Команды управления стеком					
FRAME K	SP = SP + K {SP += K}	1	1			1
POPD Smem {Smem = pop()}	Smem = TOS, ++SP	1	1			17A,17B
POPM MMR {MMR = pop()} {mmr(MMR)=pop()}	MMR = TOS, ++SP	1	1			17A
PSHD Smem {push(Smem)}	--SP, TOS = Smem	1	1			16A,16B
PSHM MMR {push(MMR)} {push(mmr(MMR))}	--SP, TOS = MMR	1	1			16A
	Другие команды управления программой					
IDLE K	idle(K) Приостановка программы (K=1,2,3)	1	4	INTM		36
MAR Smem {mar(Smem)}	Если CMPT = 0, то модифицируется ARx ; Если CMPT = 1 и ARx ≠ AR0, то модифицируется ARx, ARP = x ; Если CMPT = 1 и ARx = AR0, то модифицируется AR(ARP)	1	1	CMPT		1, 2
NOP	Нет операции	1	1			1
RESET	Программный сброс	1	3		Все	35
RSBX N, SBIT {SBIT = 0}	STN (SBIT) = 0 {ST(N,SBIT) = 0}	1	1		Заданный↓	1
SSBX N, SBIT {SBIT = 1}	STN (SBIT) = 1 {ST(N,SBIT) = 1}	1	1		Заданный↑	1
XC n, cond [, cond[, cond]] {if (cond(s)) execute(n)}	Если cond(s)=ИСТИНА, выполняются команды из следующих n ячеек памяти; n = 1,2	1	1	cond	OV↓ (cond=OV)	1
	Команды загрузки, сохранения и пересылки данных					
	Команды загрузки					
DLD Lmem, dst	dst = Lmem {dst = **dbl**|**dual**(Lmem)}	1	1	SXM		9A, 9B
LD Smem, dst	dst = Smem	1	1	SXM		3A, 3B
LD Smem, TS, dst	dst = Smem << TS	1	1	SXM, OVM	OV↑	3A, 3B
LD Smem, 16 , dst	dst = Smem << 16	1	1	SXM, OVM	OV↑	3A, 3B
LD Smem [, SHIFT], dst	dst = Smem << SHIFT	2	2	SXM, OVM	OV↑	4A, 4B
LD Xmem, SHFT, dst	dst = Xmem << SHFT	1	1	SXM, OVM	OV↑	3A
LD #K, dst	dst = #K	1	1	SXM		1
LD #lk [, SHFT], dst	dst = #lk << SHFT	2	2	SXM, OVM	C, OV↑	2

Синтаксис	Выражение (операция)	С†	Ц†	Зависит от	Влияет на	Класс
LD #lk, 16, dst	dst = #lk << 16	2	2	SXM, OVM	C, OV↑	2
LD src, ASM [, dst]	dst = src << ASM	1	1	SXM, OVM	C, OV↑	1
LD src [, SHIFT][, dst]	dst = src << SHIFT	1	1	SXM, OVM	C, OV↑	1
LD Smem, T	T = Smem	1	1			3A, 3B
LD Smem, DP	DP = Smem(8-0) {DP = Smem}	1	3			5A, 5B
LD #k9, DP	DP = #k9	1	1			1
LD #k5, ASM	ASM = #k5	1	1			1
LD #k3, ARP	ARP = #k3	1	1			1
LD Smem, ASM	ASM = Smem(4-0) {ARP = Smem}	1	1			3A, 3B
LDM MMR, dst	dst = MMR {dst = mmr(MMR)}	1	1	no SXM		3A
LDR Smem, dst	dst = rnd(Smem) [Smem<<16 + 1<<15]	1	1	SXM		3A, 3B
LDU Smem, dst	dst = uns(Smem)	1	1	no SXM		3A, 3B
LTD Smem {ltd(Smem)}	T = Smem, (Smem + 1) = Smem	1	1			24A,24B
Команды сохранения						
DST src, Lmem	Lmem = src **{dbl|dual**(Lmem) = dst}	1	2			13A,13B
ST T, Smem	Smem = T	1	1			10A,10B
ST TRN, Smem	Smem = TRN	1	1			10A,10B
ST #lk, Smem	Smem = #lk	2	2			12A,12B
STH src, Smem	Smem = hi(src)	1	1			10A,10B
STH src, ASM, Smem	Smem = hi(src) << ASM	1	1	SXM		10A,10B
STH src, SHFT, Xmem	Xmem = hi(src) << SHFT	1	1	SXM		10A
STH src [, SHIFT], Smem	Smem = hi(src) << SHIFT	2	2	SXM		11A,11B
STL src, Smem	Smem = src	1	1			10A,10B
STL src, ASM, Smem	Smem = src << ASM	1	1			10A,10B
STL src, SHFT, Xmem	Xmem = src << SHFT	1	1			10A,10B
STL src [, SHIFT], Smem	Smem = src << SHIFT	2	2			11A,11B
STLM src, MMR	MMR = src {mmr(MMR) = src}	1	1			10A
STM #lk, MMR	MMR = #lk {mmr(MMR) = #lk}	2	2			12A

Синтаксис	Выражение (операция)	С†	Ц†	Зависит от	Влияет на	Класс
Команды условного сохранения						
CMPS src, Smem {cmps(src, Smem)}	Если src(31-16) > src(15-0), то Smem = src(31-16), TRN<<1, TC=0; иначе Smem = src(15-0), TRN<<1+1, TC=1	1	1		TC	10A,10B
SACCD src, Xmem, cond	Если cond=ИСТИНА, то Xmem = hi(src) << ASM {if(cond) Xmem = hi(src) << ASM}	1	1	SXM, cond		15
SRCCD Xmem, cond	Если cond=ИСТИНА, то Xmem = BRC {if(cond) Xmem = BRC}	1	1	cond		15
STRCD Xmem, cond	Если cond=ИСТИНА, то Xmem = T {if(cond) Xmem = T}	1	1	cond		15
Команды параллельной загрузки и сохранения						
ST src, Ymem || LD Xmem, dst	Ymem = hi(src) << ASM || dst = Xmem << 16	1	1	SXM		14
ST src, Ymem || LD Xmem, T	Ymem = hi(src) << ASM || T = Xmem	1	1	SXM		14
Команды параллельной загрузки и умножения						
LD Xmem, dst || MAC Ymem, dst_	dst = Xmem << 16 || dst_ = dst_ + T * Ymem {dst_ += …}	1	1	SXM, FRCT, OVM	C, OVdst_↑	7
LD Xmem, dst || MACR Ymem, dst_	dst = Xmem << 16 || dst_ = rnd(dst_ + T * Ymem)	1	1	SXM, FRCT, OVM	C, OVdst_↑	7
LD Xmem, dst || MAS Ymem, dst_	dst = Xmem << 16 || dst_ = dst_ - T * Ymem {dst_ -= …}	1	1	SXM, FRCT, OVM	C, OVdst_↑	7
LD Xmem, dst || MASR Ymem, dst_	dst = Xmem << 16 || dst_ = rnd(dst_ - T * Ymem)	1	1	SXM, FRCT, OVM	C, OVdst_↑	7
Команды параллельного сохранения и сложения/вычитания						
ST src, Ymem || ADD Xmem, dst	Ymem = hi(src) << ASM || dst = (Xmem << 16) + dst_	1	1	SXM, OVM	C, OV↑	14
ST src, Ymem || SUB Xmem, dst	Ymem = hi(src) << ASM || dst = (Xmem << 16) - dst_	1	1	SXM, OVM	C, OV↑	14
Команды параллельного сохранения и умножения						
ST src, Ymem || MAC Xmem, dst	Ymem = hi(src) << ASM || dst = dst + T * Xmem {dst += …}	1	1	SXM, FRCT, OVM	C, OV↑	14

Синтаксис	Выражение (операция)	С†	Ц†	Зависит от	Влияет на	Класс
ST src, Ymem || MACR Xmem, dst	Ymem = hi(src) << ASM || dst = rnd(dst + T * Xmem)	1	1	SXM, FRCT, OVM	C, OV↑	14
ST src, Ymem || MAS Xmem, dst	Ymem = hi(src) << ASM || dst = dst – T * Xmem {dst -= …}	1	1	SXM, FRCT, OVM	C, OV↑	14
ST src, Ymem || MASR Xmem, dst	Ymem = hi(src) << ASM || dst = rnd(dst – T * Xmem)	1	1	SXM, FRCT, OVM	C, OV↑	14
ST src, Ymem || MPY Xmem, dst	Ymem = hi(src) << ASM || dst = T * Xmem	1	1	SXM, FRCT, OVM	C, OV↑	14
Команды пересылки данных						
MVDD Xmem, Ymem	Ymem = Xmem	1	1			14
MVDK Smem, dmad	EAR = dmad, data(EAR) = Smem EAR = EAR + 1 {data(dmad) = Smem}	2	2			19A,19B
MVDM dmad, MMR	DAR = dmad, MMR = data(DAR) DAR = DAR + 1 {mmr(MMR) = data(dmad)}	2	2			19A
MVDP Smem, pmad	PAR = pmad, prog(PAR) = Smem PAR = PAR + 1 {prog(pmad) = Smem}	2	4			20A,20B
MVKD dmad, Smem	DAR = dmad, Smem = data(DAR) DAR = DAR + 1 {Smem = data(dmad)}	2	2			19A,19B
MVMD MMR, dmad	EAR = dmad, data(EAR) = MMR EAR = EAR + 1 {data(dmad) = mmr(MMRx)}	2	2			19A
MVMM MMRx, MMRy	MMRy = MMRx {mmr(MMRy) = mmr(MMRx)}	1	1			1
MVPD pmad, Smem	PAR = pmad, Smem = prog(PAR) PAR = PAR + 1 {Smem = prog(pmad)}	2	3			21A,21B
PORTR PA, Smem	Smem = port(PA)	2	2			27A,27B
PORTW Smem, PA	port(PA) = Smem	2	2			28A,28B
READA Smem	PAR = A(19-0), Smem = prog(PAR), PAR = PAR + 1 {Smem = prog(A)}	1	5			25A,25B
WRITA Smem	PAR = A(19-0), prog(PAR) = Smem, PAR = PAR + 1 {prog(A) = Smem}	1	5			26A,26B

Примечания:
† Количество слов (С) и циклов (Ц) соответствуют использованию DARAM для данных.
‡ Условие истинно.
§ Условие ложно.
¶ Задержанная команда.

ПРИЛОЖЕНИЕ 6

Коды и условия переходов

Условие	Группа	Описания	Код условия	Условие	Группа	Описания	Код условия
BIO ±	2C	/BIO низкий	0000 0011	NBIO ±	2C	/BIO высокий	0000 0010
C ±	2B	C = 1	0000 1100	NC ±	2B	C = 0	0000 1000
TC ±	2A	TC = 1	0011 0100	NTC ±	2A	TC = 0	0010 0000
AEQ	1A	(A) = 0	0100 0101	BEQ	1A	(B) = 0	0100 1101
ANEQ	1A	(A) ≠ 0	0100 0100	BNEQ	1A	(B) ≠ 0	0100 1100
AGT	1A	(A) > 0	0100 0110	BGT	1A	(B) > 0	0100 1110
AGEQ	1A	(A) ≥ 0	0100 0010	BGEQ	1A	(B) ≥ 0	0100 1010
ALT	1A	(A) < 0	0100 0011	BLT	1A	(B) < 0	0100 1011
ALEQ	1A	(A) ≤ 0	0100 0011	BLEQ	1A	(B) ≤ 0	0100 1111
AOV ±	1B	Переполнение A	0111 0000	BOV ±	1B	Переполнение B	0111 1000
ANOV ±	1B	Нет переполнения A	0110 0000	BNOV ±	1B	Нет переполнения B	0110 1000
UNC ±		Безусловный	0000 0000				

Примечание.
± Данные условия нельзя использовать в командах условного сохранения.

ЛИТЕРАТУРА

1. Солонина, А. И. Алгоритмы и процессоры цифровой обработки сигналов / А. И. Солонина, Д. А. Улахович, Л. А. Яковлев. – СПб. : БХВ-Петербург, 2002.

2. Матюшкин, Б. Д. Цифровая обработка сигналов : процессоры, алгоритмы, средства проектирования / Б. Д. Матюшкин, М. С. Куприянов. – 2-е изд., перераб. и доп. – СПб. : Политехника, 2002.

3. Петровский, А. А. Методы и микропроцессорные средства обработки широкополосных и быстропротекающих процессов в реальном времени / А. А. Петровский. – Минск : Наука и техника, 1988.

4. 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.

5. Потемкин, А. И. Справочник по системе MATLAB / А. И. Потемкин. – М. : МИФИ, 1998.

6. TMS320C54x DSP. Reference Set. Vol. 1 : CPU and Peripherals / Texas Instruments. – 2001.

7. TMS320C54x DSP. Reference Set. Vol. 2 : Mnemonic Instruction Set / Texas Instruments. – 2001.

8. TMS320C54x DSP. Reference Set. Vol. 4 : Applications Guide / Texas Instruments. – 1996.

9. TMS320C54x DSP. Reference Set. Vol. 5 : Enhanced Peripherals / Texas Instruments. – 2001.

10. TMS320C54x DSP. Enhanced Peripherals : Reference Guide / Texas Instruments, 2007.

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

12. Digital Signal Processing Applications with the TMS320 Family. Theory, Algorithms, and Implementations. Vol. 1 / Texas Instruments. – 1989.

13. Digital Signal Processing Applications with the TMS320 Family. Theory, Algorithms, and Implementations. Vol. 2 / Texas Instruments. – 2001.

14. TMS320C54x Assembly Language Tools User’s Guide / Texas Instruments. – 2002.

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

16. Справочная система среды отладки программ Code Composer Studio.

Учебное издание

Клюс Владимир Борисович
Качинский Михаил Вячеславович
Давыдов Александр Борисович

ПРОЕКТИРОВАНИЕ ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ ВЫЧИСЛИТЕЛЬНЫХ СРЕДСТВ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор *Т.П. Андрейченко*
Корректор *Е.Н. Батурчик*
Компьютерная верстка *А.В. Бас*

Подписано в печать	Формат 60x84 1/16.	Бумага офсетная.
Гарнитура «Таймс».	Отпечатано на ризографе.	Усл. печ. л.
Уч.-изд. л. 4,0.	Тираж 130 экз.	Заказ 214.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03. 2009. ЛП №02330/0494175 от 03.04. 2009.
220013, Минск, П. Бровки, 6.