Окончание. Начало в № 10`2010

Проектирование с использованием процессоров Analog Devices. Цифровой КИХ-фильтр

Александр СОТНИКОВ

Структура проекта

Рассматриваемый проект КИХ-фильтра имеет структуру, схожую с предыдущим проектом, в котором рассматривался ввод/вывод аналогового сигнала через встроенный кодек процессора ADSP-BF527. Бóльшую часть программы занимает инициализация кодека, портов и фильтра, а сама процедура фильтрации вызывается из обработчика прерывания приема последовательного порта. Настройки периферийных модулей аналогичны настройкам, использовавшимся в проекте ввода/вывода. Единственное исключение составляет частота дискретизации в АЦП и ЦАП кодека, которая выбрана равной 8 кГц.

Проектируемый фильтр будет выделять часть спектра речевого диапазона в полосе от 1000 до 2000 Гц. Обработка отсчетов левого и правого каналов осуществляется параллельно одним и тем же фильтром. Для расчета коэффициентов фильтра может быть использовано любое доступное программное обеспечение, например инструмент Filter Design and Analysis пакета MATLAB. Частотная характеристика полосового КИХ-фильтра с 41 коэффициентом, который используется в примере, показана на рис. 5.

Входные и выходные данные и коэффициенты фильтра будут представлены дробными числами с фиксированной точкой формата (1.15).

Поскольку коэффициенты фильтра изменяться в программе не будут, для их хранения может быть использован глобальный массив с модификатором ***const***, инициализируемый при объявлении.

Для хранения предыдущих входных отсчетов фильтра необходимо иметь два массива, длина которых равна количеству коэффициентов фильтра. В то же время, поскольку кодек выдает в каждом интервале дискретизации пару 16-битных отсчетов левого и правого каналов, линия задержки может быть реализована при помощи единого массива, содержащего 32-битные слова. При этом в одной половине слова будет храниться отсчет правого канала, а во второй — левого канала.

Цифровые КИХ-фильтры в программе на языке C/C++ могут быть довольно просто и эффективно реализованы несколькими способами. Первый способ заключается в использовании библиотечной функции ***fir_fr16***, которая работает с дробными знаковыми числами формата (1.15) и применяется для обработки отдельных отсчетов или буферов данных.

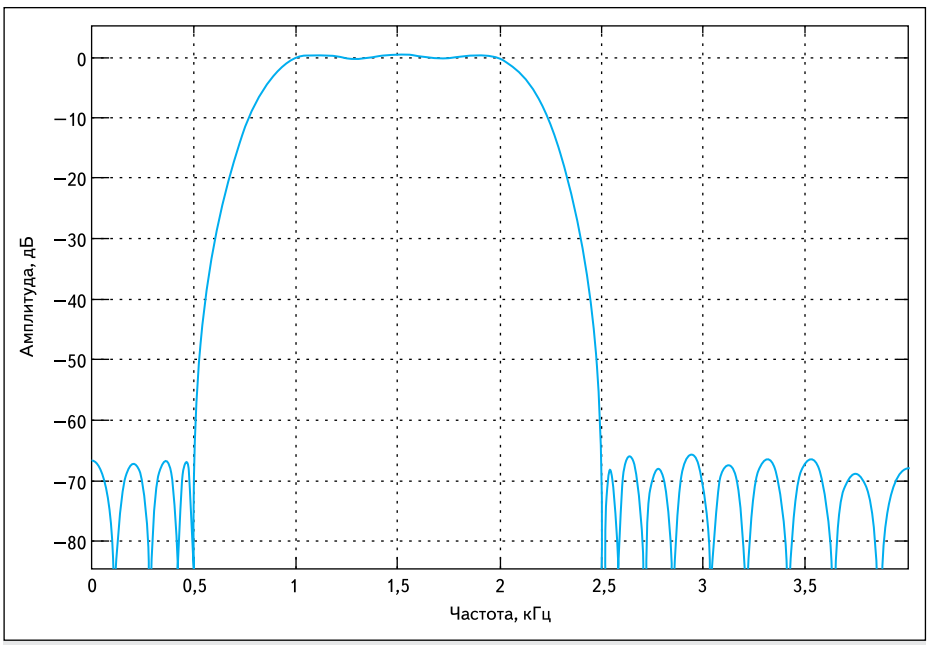

Рис. 5. Частотная характеристика КИХ-фильтра

Второй способ заключается в написании цикла для операции свертки вручную. При этом для организации циклического буфера могут быть использованы встроенные функции компилятора ***circindex*** или ***circptr***, объявленные в заголовочном файле ***ccblkfn.h***. Кроме того, если включена оптимизация, компилятор автоматически распознает оператор деления по модулю (“%”) при индексации элементов буфера как признак того, что необходима организация циклического буфера. Более подробную информацию об адресации циклических буферов и других методах оптимизации программ на C/C++ для процессоров Blackfin можно найти в руководстве по компилятору [1].

Несмотря на удобство высокоуровневого программирования, в рассматриваемом проекте функция фильтрации будет выполняться при помощи кода, написанного на языке ассемблера. Это позволит, во-первых, создать фильтр, одновременно обрабатывающий данные левого и правого каналов кодека, а во-вторых, наглядно проиллюстрировать организацию интерфейса между частями кода на языках C/C++ и ассемблера.

Интерфейс между C/C++ и ассемблером

При создании проектов, которые должны функционировать в режиме реального времени, нередко возникает необходимость написания отдельных частей алгоритма на языке ассемблера для повышения быстродействия. Это особенно актуально для подпрограмм обслуживания прерываний периферийных модулей процессора.

Использование функций на языке ассемблера внутри программы, написанной на языке C/C++, требует соблюдения определенного набора правил, касающихся объявления функций, передачи аргументов и возвращаемого значения, а также работы с аппаратными регистрами процессора.

Перед вызовом ассемблерной подпрограммы из программы на C/C++ необходимо

создать для нее прототип. Несмотря на то, что использование функций без прототипов в C/C++ допускается, делать так не рекомендуется, потому что в данном случае компилятор не может выполнить проверку типов аргументов и предполагает, что функция возвращает тип ***int***.

Первые три аргумента функции передаются в регистрах регистрового файла данных R0, R1 и R2. Для передачи последующих аргументов используется стек процессора. Если программист не хочет использовать стек, то в качестве одного из трех параметров в функцию может быть передан указатель на структуру, содержащую необходимые данные. Результат выполнения функции возвращается в регистре R0.

Аппаратные регистры процессора делятся с точки зрения компилятора на три категории: регистры, которые могут свободно использоваться в функции на языке ассемблера (scratch-регистры), и регистры, состояние которых после выхода из функции должно оставаться таким же, как до входа в нее (call-preserved регистры), а также специальные регистры, которые следует использовать в теле функции только в особых случаях (dedicated-регистры). К регистрам второго типа относятся P3–P5 и R4–R7. К регистрам первого типа относятся все остальные регистры регистровых файлов указателей и данных, регистры генераторов адресов данных (Ix, Bx, Mx), регистры циклов (LBx, LCx, LTx), а также регистр ASTAT. Если одних только этих регистров не хватает для реализации функции, то программист может использовать call-preserved регистры, сохранив содержащиеся в них данные в стек при входе в функцию и восстановив их из стека при выходе из функции. На регистры, отображенные в карте памяти процессора (отвечающие за периферийные модули процессора), эти правила, естественно, не распространяются, и они могут изменяться произвольным образом.

Специализированные регистры — это регистры указателя стека (SP), кадра стека (FP) и длины циклических буферов (L0–L3). Первые два регистра при выходе из функции должны указывать на корректный, выровненный по 32-битной границе адрес внутри стека. Что касается регистров длины циклических буферов, то компилятор автоматически обнуляет их перед входом в функцию и ожидает, что они содержат нулевое значение после выхода из функции. Поэтому если любой из этих регистров используется внутри функции, как в рассматриваемом проекте, то программист должен не забыть сбросить его в ноль при выходе из функции.

Подпрограмма цифрового КИХ-фильтра

Функция цифрового фильтра будет вызываться при поступлении каждого нового прерывания приема последовательного порта и обрабатывать одну пару отсчетов правого и левого каналов кодека. Для работы функции в нее необходимо передать сами входные отсчеты, а также значения количества отсчетов фильтра, начального адреса массива коэффициентов, начального адреса линии задержки и адрес элемента линии задержки, в который должна быть записана принятая пара отсчетов.

Входные отсчеты и длина фильтра передаются в функцию при помощи двух аргументов типа int (вспомним, что 16-битные данные в формате с фиксированной точкой, поступающие от кодека, упаковываются последовательным портом в одно 32-битное слово). Остальные параметры передаются в функцию при помощи указателя на структуру вида:

```
struct firState {
int *pDelayLine;      //Указатель на начало линии задержки
int *pos;             //Указатель на элемент линии задержки,
                      //в который будут
                      //записываться входные данные
const short *pCoeff;  //Указатель на массив коэффициентов фильтра
} fir_state;
```

Возвращаемая функцией пара выходных отсчетов фильтра так же, как и пара входных отсчетов, упаковывается в одно 32-битное слово.

Таким образом, прототип функции цифрового фильтра будет иметь вид:

```
int fir_stereo(int data_sample, int nTaps, void *pFirDesc);
```

Приведем пример кода функции фильтра на языке ассемблера:

```
.section program;
.global _fir_stereo;

_fir_stereo:

    p0 = r2;          // В P0 помещается указатель на структуру
                      // с описанием параметров фильтра
    p1 = [p0++];      // В P1 помещается указатель
                      // на начало линии задержки
    p2 = [p0++];      // В P2 помещается указатель на последний
                      // элемент линии задержки

    // Инициализация циклического буфера для линии задержки
    i0 = p2;          // Текущее значение указателя циклического
                      // буфера
    m0 = 0            // Значение модификатора инкремента
                      // (не используется)
    b0 = p1;          // Базовый (начальный) адрес циклического
                      // буфера
    r2=r1<<2;         // Размер циклического буфера указывается
                      // в байтах
    l0 = r2;

    // В регистр P2, используемый для инициализации цикла,
    // заносится значение, равное числу коэффициентов фильтра
    // минус один
    r1 += -1;
    p2 = r1;

    p1 = [p0--];      // В P1 помещается указатель на массив
                      // коэффициентов
    [i0++] = r0;      // В линию задержки записывается новый отсчет
    r1 = i0;
    [p0--]=r1;        // В соответствующий элемент структуры
                      // записывается новое
                      // значение текущего указателя линии задержки
    // Инициализация указателя массива коэффициентов,
    // регистров-аккумуляторов и первая операция загрузки
    // данных и коэффициента фильтра
    i1 = p1;
    a0 = 0;
    a1 = 0 || r0 = [i0++] || r2.l = w[i1++];

    // Основной цикл
    lsetup(fir_stereo_loop,fir_stereo_loop) LC0=p2;
    fir_stereo_loop: a0+=r0.l*r2.l,a1+=r0.h*r2.l || r0=[i0++] ||
r2.l=w[i1++];

    // Последняя операция умножения с накоплением
    // и сохранение результатов в регистр
    r0.l= (a0+= r0.l * r2.l),r0.h = (a1+= r0.h * r2.l);

    l0=0;
    rts;
._fir_stereo.end:
```

Для реализации линии задержки используются регистры I0, B0, L0 генератора адресов данных. При каждом вызове функции эти регистры инициализируются соответствующими значениями передаваемых аргументов. Для адресации массива коэффициентов используется регистр генератора адресов данных I1. Обратите внимание на то, что необходимости в циклической адресации массива коэффициентов нет. Основная часть процедуры фильтрации выполняется в цикле, содержащем единственную многофункциональную команду, которая выделена меткой ***fir_stereo_loop***:

```
a0+=r0.l*r2.l,a1+=r0.h*r2.l || r0=[i0++] || r2.l=w[i1++];
```

Эта команда осуществляет следующие операции:

- перемножение отсчетов левого и правого каналов, содержащихся в старшей и младшей половинах регистра R0, на коэффициент фильтра, содержащийся в младшей половине регистра R2;
- сложение результатов умножения с текущим содержимым регистров аккумуляторов A0 и A1;
- загрузка в регистры R0 и R2 новых значений из ячеек памяти, на которые указывают регистры I0 и I1, с постинкрементом указателей.

Первая операция загрузки из памяти и последняя операция умножения с накоплением осуществляются вне тела цикла, поэтому в счетчик цикла LC0 записывается значение, равное количеству коэффициентов фильтра минус один.

Файл с функцией ***fir_stereo***, имеющий расширение ***asm***, подключается пользователем к проекту и используется вместе с основным файлом ***main.c*** на этапе сборки.

Основной файл проекта

Основной файл проекта реализован на языке C. В нем выполняются все необходимые процедуры инициализации, после чего программа входит в бесконечный цикл и ожидает поступления прерывания, сигнализирующего о приеме нового отсчета данных по последовательному порту SPORT0. Текст основного файла проекта ***main.c***:

```
#include <cdefBF52x_base.h>
#include <ccblkfn.h>
#include "sys\exception.h"
#define NTAPS 41
```

```
volatile struct firState {
      int *pDelayLine;
      int *pos;
      const short *pCoeff;
      }
      fir_state;
void write_codec_reg(short addr, short data);
int fir_stereo(int data_sample, int nTaps, void *pFirDesc);

volatile int x,y;
section("L1_data_a") volatile int dLine[NTAPS];
section("L1_data_b") const short coeff[NTAPS] = {0xfffc,0xffae,0
xff71,0x008d,0x0210,0x00fb,0xfe3c,0xfe64,0xfffe,0xfdf2,0xfca0,0x
028c,0x0723,0x0203,0xffad,0x0675,0x0005,0xe7bb,0xe72c,0x0fbc,0
x2b51,0x0fbc,0xe72c,0xe7bb,0x0005,0x0675,0xffad,0x0203,0x0723,
0x028c,0xfca0,0xfdf2,0xfffe,0xfe64,0xfe3c,0x00fb,0x0210,0x008d,0
xff71,0xffae,0xfffc};

/* Обработчик прерывания приема последовательного порта */
EX_INTERRUPT_HANDLER(Sport0_RX_ISR)
{
      x = *pSPORT0_RX;
      y = fir_stereo(x,NTAPS,(void *)(&fir_state));
      *pSPORT0_TX=y;
}

void main()
{
      /* Инициализация линии задержки и параметров фильтра */
      int i=0;
      for(i=0;i<NTAPS;i++)
      {
                  dLine[i]=0;
      };
      fir_state.pDelayLine = (int*)dLine;
      fir_state.pos = (int *)dLine;
      fir_state.pCoeff = coeff;

      /* Настройка портов ввода/вывода общего назначения (GPIO) */
      *pPORTG_MUX=0x0152;
      *pPORTG_FER=0x87DC;
      *pPORTH_FER=0x0200;

      //Настройка порта SPI
      *pSPI_BAUD=0x07FF;
      *pSPI_FLG=0xFF20;
      *pSPI_CTL=0x5D01;

      //Инициализация кодека
      /* К АЦП кодека подключается вход микрофона, к выходу
кодека подключается ЦАП. Параметры цифрового интерфейса:
16-битный процессорный режим, кодек является источником
сигнала тактовой синхронизации, сигнал BCLK не инвертируется,
частота дискретизации 8 кГц */
      write_codec_reg(0xf,0);
      write_codec_reg(0x6,0xfd);
      write_codec_reg(0x4,0x0d);
      write_codec_reg(0x0,0x17);
      write_codec_reg(0x1,0x17);
      write_codec_reg(0x2,0x79);
      write_codec_reg(0x3,0x79);
      write_codec_reg(0x6,0x41);
      write_codec_reg(0x4,0x15);
      write_codec_reg(0x5,0);
      write_codec_reg(0x7,0x43);
      write_codec_reg(0x8,0x09);

      /* Настройка последовательного порта в режим с кадровой
синхронизацией */
      *pSPORT0_RCR1=0x6400;
      *pSPORT0_RCR2=0x001f;
      *pSPORT0_TCR1=0x6400;
      *pSPORT0_TCR2=0x001f;

      /* Инициализация прерываний */
      register_handler(ik_ivg9, Sport0_RX_ISR);
      *pSIC_IMASK0 = 0x00010000;
      write_codec_reg(0x9,0x1);

      /* Активация последовательного порта */
      *pSPORT0_TCR1 |= 1;
      *pSPORT0_RCR1 |= 1;


      while(1);
}

/* Процедура записи в регистры кодека */
void write_codec_reg(short addr, short data)
{
      *pSPI_TDBR=((addr&0xf)<<9)|(data&0x1ff);
      while(*pSPI_STAT&0x08||(!(*pSPI_STAT&0x01)));
      *pSPI_FLG=0xDF20;
      *pSPI_FLG=0xFF20;
}
```

Обратите внимание на то, как объявлены массивы ***dLine*** (линия задержки) и ***coeff*** (массив коэффициентов фильтра). Применение квалификаторов ***section*** позволит линкеру на этапе компоновки разместить их в разных банках памяти данных. В стандартном LDF-файле для процессоров ADSP-BF527, который подключается к проекту по умолчанию (находится в папке ***Blackfin\ldf*** установочного каталога Visual DSP++), уже реализована поддержка секций с именами ***L1_data_a*** и ***L1_data_b***. В то же время при подключении к проекту собственного LDF-файла программисту необходимо самостоятельно позаботиться о помещении этих входных секций объектного файла в соответствующие банки памяти.

Собрав проект в среде VisualDSP++ и получив исполняемый файл, вы можете проверить работу фильтра при помощи микрофона или генератора внешнего сигнала и колонок. Проект может быть также легко модифицирован для использования в качестве источника сигнала линейного входа платы EZ-KIT Lite.

Заключение

В этой статье на примере проекта фильтра с конечной импульсной характеристикой были показаны архитектурные особенности процессоров Blackfin, повышающие эффективность обработки. Рассмотренные принципы можно также применять во многих других задачах цифровой обработки сигналов для повышения производительности вычислений.

Литература

1. VisualDSP++ 5.0 C/C++ Compiler and Library Manual for Blackfin Processor. Revision 5.2. Analog Devices Inc. Sept. 2009.