Реализация целочисленного БПФ на процессорах с архитектурой ARM

Среди микроконтроллеров и микропроцессоров для встроенных систем одними из самых производительных и перспективных являются устройства с 32-разрядным ядром ARM, в частности ARM7TDMI. Их выпускают многие известные фирмы, такие как Atmel, Cirrus Logic, Samsung и др. При средней цене $10–40 они имеют производительность десятки MIPS, низкое потребление и, как правило, много различной периферии на кристалле, что делает их идеальным средством для обработки сигналов в недорогих устройствах.

Алгоритм БПФ Кули-Тюки

N-точечное дискретное преобразование Фурье – это N взвешенных сумм, каждая из которых состоит из N слагаемых. Каждое конкретное слагаемое получается комплексным умножением одной из входных точек на один из известных коэффициентов. Таким образом, прямое вычисление ДПФ требует около N^2 операций комплексного умножения и N^2 операций комплексного сложения.

Быстрое преобразование Фурье – это способ производить вычисления таким образом, чтобы получить тот же результат за меньшее число операций. Если N=A·B – составное число, то результат можно получить, сделав A B-точечных и B A-точечных преобразований. Доказательство этого можно найти во многих книгах по цифровой обработке сигналов, например в книге Л. Рабинера и Б. Гоулда "Теория и применение цифровой обработки сигналов". Рассуждая аналогично, можно сделать вывод, что если N есть степень двойки, то нужно произвести порядка $N \cdot \log_2 N$ 2-точечных преобразований. В этом и состоит алгоритм БПФ Кули-Тюки. В упомянутой книге можно найти его исчерпывающее описание.

Таким образом, время вычисления БПФ определяется временем вычисления 2-точечного преобразования, которое принято называть "бабочкой". Эта базовая операция выглядит так:

$$Y_1 = X_1 + X_2 \cdot W,\ Y_2 = X_1 - X_2 \cdot W,$$

где X_1, X_2 – исходные точки, Y_1, Y_2 – результат, W – комплексный коэффициент.

Реализация

Перед автором стояла задача написать преобразование Фурье на 2048 точек при разрядности исходных данных 16 бит. Из-за отсутствия арифметического сопроцессора пришлось делать целочисленное преобразование, что создало некоторые трудности. При разрядности исходных данных 16 бит разрядность коэффициентов должна быть не менее 16, чтобы не происходило потери точности. Их произведение содержит 32 разряда. 2048 точек дают еще 11 дополнительных разрядов, а это значит, что в 32-разрядное процессорное слово промежуточные данные не помещаются. Вычисление каждой "бабочки" ведется с точностью 64 разряда, а результат округляется до 32 разрядов.

Приведенная ниже функция, производящая вычисление "бабочки", написана на ассемблере процессоров архитектуры ARM7TDMI в режиме ARM.

```
;прототип функции:
; void Butterfly(b1, b1mm, ar, pow),
;где:
; ar - указатель на массив данных (действительная часть), мнимая часть находится по адресу ar+8192
; b1, b1mm - индексы исходных данных для "бабочки"
; pow - индекс поворотного множителя W, адрес массива wr импортируется,
;параметры передаются в r0-r3 слева направо:
; b1-r0, b1mm-r1, ar-r2, pow-r3

Butterfly
        STMDB   sp!,{r4-r12,lr}
        cmp     r3,#512
        rsbge   r3,r3,#1024 ;     r3=pow, if (r3>512) { r3=(1024-r3); sign=-1 } else { sign=1 }
        mov     r3,r3,lsl #1
;     тип long long - 64-битное целое со знаком.
;    комплексное умножение a*w:
;     long long tbr = sign*(long long)ar[b1mm]*long(wr[pow]) - (long long)ai[b1mm]*long(wr[512-pow]);
;     long long tbi = sign*(long long)ai[b1mm]*long(wr[pow]) + (long long)ar[b1mm]*long(wr[512-pow]);
;временные переменные хранятся в (старш., младш.):
;      tbr (64 бита) - r5, r4
;      tbi (64 бита) - r7, r6
;     wr[pow], wr[512-pow] (по очереди) - r8
;     ar[b1mm], ai[b1mm] - r9, r12;
        ldr     r10,=wr
        add     r11,r2,#8192;         r11 - ai
        ldrsh   r8,[r10,r3];          r8 = wr[pow]
        rsbge   r8,r8,#0;             r8 = sign*wr[pow]
        ldr     r9,[r2,r1,lsl #2];    r9 = ar[b1mm]
        smull   r4, r5, r8, r9;       r5, r4 - первое слагаемое tbr
        ldr     r12,[r11,r1,lsl #2];  r12 - ai[b1mm]
        smull   r6, r7, r8, r12;      r7, r6 - первое слагаемое tbi
        rsb     r3,r3,#1024
        ldrsh   r8,[r10,r3];          r8 = wr[512-pow]
        smlal   r6, r7, r8, r9;       r7, r6 - tbi
        rsb     r8,r8,#0
        smlal   r4, r5, r8, r12;      r5, r4 - tbi
;     long long tar = (long long)(ar[b1])<<14;
;     long long tai = (long long)(ai[b1])<<14;
;округление и сдвиг фиксированной точки
;     ar[b1] = (tar+tbr+(1<<13))>>14;
;     ai[b1] = (tai+tbi+(1<<13))>>14;
;     ar[b1mm] = (tar-tbr+(1<<13))>>14;
;     ai[b1mm] = (tai-tbi+(1<<13))>>14;
;      tar - r9, r8
;      tai - r11, r10
;
        mov     r12,r11;              r12 - ai
        mov     r3,#0
        ldr     r8,[r2,r0,lsl #2];    r8 - ar[b1];
        ldr     r10,[r12,r0,lsl #2];  r10 - ai[b1]
        mov     r9,r8,asr #18
        mov     r11,r10,asr #18
        mov     r8,r8,lsl #14
        mov     r10,r10,lsl #14
        adds    r8,r8,#8192;          r9, r8 = ((long long)(ar[b1])<<14)+(1<<13)
        adc     r9,r9,r3
        adds    r10,r10,#8192;        r11, r10 = ((long long)(ai[b1])<<14)+(1<<13)
        adc     r11,r11,r3
        STMDB   sp!,{r8-r9}
        adds    r8,r8,r4
        adc     r9,r9,r5
        mov     r8,r8,lsr #14
        orr     r8,r8,r9,lsl #18
```

```
str    r8,[r2,r0,lsl #2];     store ar[b1]
adds   r8,r10,r6
adc    r9,r11,r7
mov    r8,r8,lsr #14
orr    r8,r8,r9,lsl #18
str    r8,[r12,r0,lsl #2];    store ai[b1]
LDMIA  sp!,{r8-r9}
subs   r8,r8,r4
sbc    r9,r9,r5
mov    r8,r8,asr #14
orr    r8,r8,r9,lsl #18
str    r8,[r2,r1,lsl #2];     store ar[b1mm]
subs   r10,r10,r6
sbc    r11,r11,r7
mov    r10,r10,asr #14
orr    r10,r10,r11,lsl #18
str    r10,[r12,r1,lsl #2];   store ai[b1mm]
LDMIA  sp!,{r4-r12,lr}
bx     lr
```

В процедуре использованы очень удобные команды 64-разрядного умножения со знаком SMULL и 64-разрядного умножения/сложения со знаком SMLAL. Время выполнения этих команд составляет всего 2–5 тактов в зависимости от размера множителя (при условии отсутствия тактов ожидания при обращении к памяти программ и данных). Как видно из текста программы, основное время занимает извлечение данных из памяти, сдвиг результата (вычисления идут с фиксированной точкой, и при умножении она сдвигается влево) и сохранение его в памяти. Несмотря на большое количество режимов адресации памяти и удобную систему команд, не удалось сделать время вычисления бабочки менее 150 тактов.

БПФ вычислялось на процессоре AT91M40400 фирмы Atmel с тактовой частотой 33 МГц. Функция БПФ работала во внутренней 32-разрядной памяти, коэффициенты хранились там же. Исходные данные были во внешней 8-битной памяти с одним тактом ожидания. Одно вычисление БПФ занимало время порядка 1/12 секунды. Сама функция БПФ написана на С, поэтому если написать все на ассемблере, результат можно несколько улучшить. Исходный текст полной функции БПФ можно скачать по адресу www.platan.ru/shem/.

Павел Филимонов,
paulfilimonov@mail.ru