УДК 519.687.5+53.083.92+004.414.2

Ю. П. Галюк
Санкт-Петербургский государственный университет
35, Университетский просп., Санкт-Петербург, Петергоф, 198504, Россия
E-mail: galyuck@paloma.spbu.ru

НИЗКОЧАСТОТНЫЙ ЦИФРОВОЙ РАДИОПРИЕМНИК

Использование цифровых устройств избавляет экспериментатора от многих трудностей, так как физические блоки заменяются их виртуальными компьютерными аналогами. Радиоприемное устройство в таком случае представляет собой обычный компьютер, оснащенный платой аналогово-цифрового преобразователя (АЦП) и соответствующим программным обеспечением. Такой приемник оказывается дешевле аналогового, так как компьютеры и платы АЦП из-за своей универсальности выпускаются большими партиями, и на их рынке существует огромная конкуренция. Отладка и настройка цифрового приемника, по сравнению с традиционным, упрощается и происходит быстрее; она сводится к простому редактированию текста программы и ее перекомпиляции. Однако при этом необходимо уметь программировать на языках высокого уровня и использовать современные методы цифровой обработки сигналов. В настоящей работе описаны оптимальные решения для цифровых радиоприемных устройств и в качестве примера рассмотрена организация реального СДВ/ДВ радиоприемника. В приложении приведены тексты типовых программных блоков на языке *C*/*C*++ и Ассемблера. Библиогр.: 4 назв.

Ключевые слова: цифровой радиоприемник, цифровая обработка сигналов, реальный масштаб времени, цифровая фильтрация, быстрое преобразование Фурье.

Организация радиофизического эксперимента всегда была трудной задачей в организационном плане из-за дороговизны аппаратуры. Промышленность заинтересована в выпуске универсальных приборов, большинство функций которых используется редко, а в силу единичных партий приборы оказываются дорогими. В то же время остается необходимость создавать аппаратуру сопряжения промышленных приборов с экспериментальной установкой и устройствами анализа данных, как правило, персональных компьютеров. Применение цифровых радиоприемных устройств избавляет от многих трудностей. Радиоприемник в таком случае реализуется с помощью обычного компьютера, аналогово-цифровой платы (АЦП) и набора программ. Такое решение оказывается дешевым, универсальным, простым в отладке и настройке. При этом, однако, требуется персонал, умеющий программировать на языках высокого уровня и владеющий методами цифровой обработки сигналов (ЦОС). Ниже мы рассмотрим основные этапы и приемы создания цифровых радиоприемных устройств. В основу работы положен доклад Ю. П. Галюка, М. А. Бисярина, Н. Ю. Заалова и Л. Н. Лутченко «Программно-аппаратный комплекс СПбГУ для исследования нижней ионосферы посредством радиоволн низкочастотных диапазонов», сделанный в 2012 г. на Международной конференции *EMES* в Харьковском национальном университете, посвященной памяти П. В. Блиоха [1].

Требования к компьютеру, который служит цифровым приемником, по современным меркам, достаточно скромны: *Intel*-совместимый процессор (желательно многоядерный) с оперативной памятью от 2 ГБ; *ОС Windows*\/*XP* или *Windows* 7. Среда разработки: *Microsoft Visual Studio* (2005 и выше); язык программирования *C*/*C*++.

В качестве типа проекта следует выбрать не Консольное Приложение (*Console Application*), а *Win*32. Это объясняется не только тем, что в таком проекте можно создать более удобный интерфейс с пользователем (система меню, подсказок, кнопки управления), но он более эффективно использует ресурсы процессора. Заметим, что, в отличие от *Console Application*, проект *Win*32 на самом деле представляет собой совокупность слабо связанных между собой так называемых «оконных приложений», инициализация работы которых осуществляется (через очередь сообщений) посылкой специальных запросов (*message*). Такая организация *windows*-программы позволяет разные оконные приложения выполнять на различных процессорах/ядрах. Подробнее об организации *windows*-программ, функций *Windows API* (*application programming interfaces*) и библиотеки *MFC* (*Microsoft Foundation Classes*) можно ознакомиться, например в работах [2, 3].

1. Основные этапы обработки информации в цифровом приемнике. Казалось бы, самый простой способ создания цифрового приемника – это взять блок-схему аналогового приемника и реализовать его узлы в виде набора подпрограмм. Этот путь возможен, поскольку все известные способы аналоговой обработки сигналов реализуются численно. Однако обычно цифровые приемники оказываются намного проще своих аналоговых прототипов. Это объясняется, в частности, огромным динамическим диапазоном данных, которыми оперирует современный компьютер (одинарная или двойная точность чисел с плавающей запятой). К примеру, типичный усилительный каскад имеет динамический диапазон около 40 дБ, а числа с одинарной точностью – около 150 дБ, с двойной – 340 дБ. Кроме того,

 © ИРЭ НАН Украины, 2015

некоторые алгоритмы, в частности, использующие хранение временных реализаций в линиях задержки, очень трудно реализуются в аналоговых приборах, но элементарно выполняются в цифровом виде.

Условно программа-приемник разбивается на следующие блоки:

– блок ввода данных;

– блок предварительной обработки;

– блок вычисления текущих характеристик сигнала;

– блок регистрации и визуализации результатов.

Блок ввода данных преобразует непрерывный аналоговый сигнал в ряд цифровых отсчетов, снимаемых через равные промежутки времени Δ_t. Эта процедура называется квантованием сигнала, и осуществляется с помощью АЦП. В дальнейшем мы будем использовать термины частота квантования: $F_s = 1/\Delta_t$ и частота Найквиста F_N, равная половине частоты квантования: $F_N = F_s/2$. С понятием частоты Найквиста F_N тесно связано явление подмены частот, или алиасинг [4]. При цифровой обработке могут быть однозначно интерпретированы только сигналы в частотном диапазоне $0 – F_N$. Сигналы большей частоты отображаются на эту же полосу частот по правилу «гармошки». Чтобы пояснить это явление, рассмотрим синусоидальный сигнал частоты F_a. Вычислим сначала номер гармошки M, куда попал этот сигнал. Для этого делим нацело частоту сигнала на частоту Найквиста F_N: $M = F_a / F_N$. Если число M – четное, то сигнал отображается на частоту $F_d = F_a - M F_n$, если нечетное – на частоту $F_d = (M+1)F_n - F_a$. Для однозначности в интерпретации результатов необходимо, чтобы все частотные компоненты сигнала располагались в пределах одной частотной гармошки – для этой цели служит входной аналоговый фильтр.

Блок предварительной обработки работает в крайне жестком режиме «реального времени». Его задача – сформировать из бесконечного ряда отсчетов конечные выборки по N последовательных чисел, длиной T_R, которые называются реализациями. Над ними и выполняются дальнейшие преобразования. Одновременно с этим блок осуществляет цифровую фильтрацию реализаций.

Блок расчета текущих характеристик сигнала по мере поступления новых реализаций вычисляет текущие значения параметров сигнала, отношение сигнал/шум и т. п.

Блок регистрации и визуализации результатов периодически записывает на внешние носители вычисляемые параметры сигнала и выводит на экран монитора графическую информацию о сигнале и его параметрах.

2. Основные алгоритмы ЦОС. Одной из самых затратных в ЦОС (с вычислительной точки зрения) и в то же самое время часто применяемой процедурой является, по терминологии линейной алгебры, «скалярное произведение векторов». Записывается эта процедура таким образом:

$$z = \sum_{i=0}^{N-1} x_i \, y_i , \qquad (1)$$

где x_i и y_i – отсчеты двух реализаций длиной N.

В зависимости от способа использования эта процедура может называться ковариацией, сверткой, фурье-преобразованием, фильтром с конечной импульсной характеристикой (КИХ), но в любом случае она сводится к сложению произведений элементов двух массивов чисел. Понятно, что в силу своей простоты эта процедура легко реализуется в виде цикла на любом языке высокого уровня, однако результат не окажется оптимальным с точки зрения скорости выполнения. Заметим, что компиляторы не умеют эффективно использовать новые возможности, предоставляемые современными процессорами, в частности, *SIMD*-расширения (*Single Instruction, Multiple Data*) набора команд. Подмножество *SIMD*-команд, эффективное именно при реализации процедур типа (1), называется *SSE*-расширением (*Streaming SIMD Extensions*), т. е. потоковым *SIMD*-расширением набора инструкций процессора.

SSE-команды производят арифметические операции над числами с плавающей запятой с одинарной или двойной точностью, располагающимися в 8 специальных 128-разрядных регистрах `xmm0÷7`. В каждый такой регистр помещается либо 2 числа с двойной точностью типа *double*, либо 4 числа с одинарной точностью типа *float*. В подавляющем большинстве случаев одинарной точности представления данных будет достаточно.

Для использования *SSE*-команд применяют язык Ассемблера. Для этого существует две возможности: использовать ассемблерные вставки (через конструкцию `__asm`) в программе на языках *C*/*C*++, либо писать и компилировать подпрограммы на языке Ассемблера, а потом вставлять полученный объектный модуль в свой проект. Мы будем ориентироваться на второй вариант.

В приложении 1 приведен текст программы `svmsse`, реализующей формулу (1). Прототип ее объявляется следующим образом:

```
extern "C" void smvsse(int N,float
*x,float *y,float *z).
```

Параметры подпрограммы соответствуют формуле (1), где N – длина реализаций, x и y – идентификаторы перемножаемых массивов, а z – адрес переменной, в которую заносится результат.

Если считать, что программа `smvsse` находится в файле `smvsse.asm`, то ее компиляция производится с помощью утилиты `ml`, поставляемой вместе с *VS*:

```
ml /c /coff smvsse.asm.
```

В результате компиляции образуется объектный файл `svmsse.obj`, который и добавляется в проект.

При использовании подпрограммы `smvsse` следует учитывать один важный момент – проблему выравнивания данных. Данные в памяти, которыми оперируют *SSE*-команды, должны находиться на границе 16 байт. К сожалению, компилятор автоматически такое выравнивание не производит – для этого в *VS* служит специальный расширенный атрибут класса памяти `_declspec`. Например, при описании массива *x*, состоящего из 1 000 элементов, следует использовать конструкцию

```
_declspec(align(16))float x[1000].
```

Параметр `align` указывает на значение выравнивания (должен быть степенью 2).

При динамическом выделении памяти под такой массив используется функция `_aligned_malloc`:

```
float *x;int N=1000;
x=(float*)_aligned_malloc(N*4,16).
```

Первым параметром функции является длина выделяемого массива в байтах (в нашем случае 1 000 4-х байтных слов), вторым – значение выравнивания (в нашем случае 16). Функция описана в заголовочном файле `malloc.h`.

Не всегда можно гарантировать, что какой-то из массивов *x* и *y* выровнен на границу 16 байт. Для этого случая ценой частичной потери производительности можно видоизменить подпрограмму `svmsse`. При отсутствии выравнивания массива *x* достаточно заменить команду `movaps` на команду `movups`, которая умеет пересылать не выровненные данные. Если не выровнены оба массива, первые две команды цикла меняются на следующий фрагмент:

```
smv: movups xmm0,oword ptr[esi+4*ecx]
     movups xmm2,oword ptr[edi+4*ecx]
     mulps xmm0,xmm2.
```

Как видно, при использовании *SSE*-команд, требуется 5–6 команд процессора для суммирования четырех членов ряда (1). Точное количество тактов процессора оценить трудно, но не более восьми, т. е. порядка двух тактов на одно звено.

Одной из областей использования формулы (1) является построение ковариационной функции S_{xy} от двух переменных *x* и *y*:

$$S_{xy}(i) = \frac{1}{N}\sum_{p=0}^{N-1} x_p\, y_{p+i}. \qquad (2)$$

Ковариация используется, в частности, при «грубом поиске» начала импульса, основывающимся на «степени сходства» двух функций.

Самой важной процедурой, которая сводится к формуле (1), является фильтр с конечной импульсной характеристикой (КИХ, или по-английски *FIR*). Цифровые фильтры можно создавать с любой заданной амплитудно-частотной характеристикой. Наиболее часто встречаются задачи построения фильтров нижних частот (ФНЧ), фильтров верхних частот (ФВЧ), а также полосовых и гребенчатых фильтров. Все эти фильтры объединяет одно общее – задается одна или несколько граничных частот и требуется обеспечить наиболее крутой спад (подъем) амплитудной характеристики в их окрестности.

Фильтры с конечной импульсной характеристикой, выражаясь математическим языком, представляют собой свертку входного сигнала с функцией отклика реализуемого фильтра. Математически это означает, что фильтрованный отсчет y_i для момента времени t_i вычисляется по формуле

$$y_i = \sum_{k=0}^{N} (x_{i-k} + x_{i+k})a_k. \qquad (3)$$

Здесь выписан вариант наиболее часто используемого симметричного КИХ-фильтра. В приведенной формуле a_k обозначают коэффициенты фильтра, x_i – входные отсчеты.

Исходя из формулы (3) выделим свойства КИХ-фильтров:

– КИХ-фильтры абсолютно устойчивы. Это объясняется тем, что сумма (3) конечна, а уже фильтрованные отсчеты никак не влияют на процедуру фильтрации.

– В вычислении сумм участвуют как более ранние – x_{i-k}, так и еще не пришедшие отсчеты x_{i+k}. В реальности это означает, что необходимо записывать входные данные по мере их поступления в буферный массив. Только после прихода $2N + 1$ отсчета мы сможем применить формулу (3). Отфильтрованный отсчет y_i оказывается сдвинутым строго на N отсчетов (на интервал времени $N\Delta_t$). Заметим, что фильтры (3) не вносят никаких фазовых искажений.

– Вследствие линейности формулы (3) можно сформулировать принцип суперпозиции: если при синтезе сложного фильтра его можно представить в виде набора *n* параллельно расположенных простых фильтров $\{a1_k\}$, $\{a2_k\}$, …, $\{an_k\}$, то коэффициенты a_k такого сложного фильтра вычисляются как сумма коэффициентов элементарных фильтров $a_k = a1_k + a2_k + \ldots + an_k$. Отсюда следует весьма нетривиальный и полезный вывод – размер фильтра (количество N его коэффициентов) не зависит от сложности его амплитудно-

частотной характеристики. Длина фильтра характеризует крутизну амплитудно-частотной характеристики (ширину переходной области Δ_f) в окрестности частоты среза фильтра. Точнее, $\Delta_f \approx 1/(2N\Delta_t)$.

– Базовым для расчетов большинства фильтров является ФНЧ. Вкратце метод его расчета сводится к умножению функции отклика идеального фильтра $\sin(x)/x$ на какую-либо конечную функцию окна. Подпрограмма `lpsd`, приведенная в приложении 2, вычисляет коэффициенты ФНЧ, используя окно Поттера *P*310. Она является результатом переработки программы *LPSPBG* из [4] с языка ФОРТРАН на язык *C*. Из полученного набора коэффициентов базового ФНЧ $\{a1_k\}$ можно получить, например, коэффициенты $\{ah_k\}$ ФВЧ. Формулы пересчета крайне простые: $ah_0 = 1 - a1_0$; $ah_k = -a1_k$, $k = 1, \ldots, N$. Интерпретировать формулу можно так: ФВЧ является результатом вычитания сигнала, прошедшего без искажения ($a_0 = 1$; $ah_k = 0$, $k = 1, \ldots, N$), и сигнала, прошедшего через ФНЧ. Поняв принцип, легко строить и другие фильтры, например, полосовой фильтр строится как разница сигналов, прошедших через два ФНЧ, настроенные на различные частоты срезов, и т. д.

– При обработке узкополосных сигналов крайне эффективно сочетание КИХ-фильтрации с одновременной децимацией (прореживанием) фильтрованных отсчетов. Именно КИХ-фильтрация позволяет для этой цели просто фильтровать каждый *n*-й отсчет (*n* – коэффициент децимации), пропуская все остальные. Децимацию проводят для того, чтобы уменьшить вычислительные затраты на дальнейшую обработку сигнала. Аналогом этой процедуры является процесс супергетеродинирования в радиотехнике, т. е. перенос спектра сигнала в низкочастотную область. Надо понимать, что частота Найквиста F_d децимированного ряда окажется в *n* раз меньше исходной: $F_d = F_N / n$. Частоты среза полосового фильтра выбираются при децимации таким образом, чтобы полоса пропускания занимала полностью одну из «складок частотной гармошки» шириной F_d.

– КИХ-фильтры идеальны для использования *SSE*-команд. К сожалению, при использовании для фильтрации формулы (1) нельзя воспользоваться симметрией коэффициентов a_k в (3), поэтому перед обращением к программе `svmsse` необходимо привести массив коэффициентов фильтра к линейному виду. Для этого можно воспользоваться, например, подпрограммой `prepsse`, приведенной в приложении 3. Входными параметрами (`mm, b`) подпрограммы являются выходные параметры подпрограммы `lpsb`. Подпрограмма возвращает в качестве результата длину массива коэффициентов, который можно использовать в программе `svmsse`; указатель на этот массив возвращается в выходной переменной *x*.

Оценим теперь предельные характеристики фильтров, которые можно получить на современных компьютерах. При частоте квантования 500 кГц на процедуру фильтрации каждого отсчета остается 2 мкс, или около 4 000 тактов 2 ГГц процессора. Использование *SSE*-команд позволяет применять фильтры длиной до 2 000 (2 такта на звено), что позволяет добиться переходной области 2–3 кГц в окрестности частот среза фильтров. Понятно, что в этом случае процессор будет занят исключительно фильтрацией, но, во-первых, современные процессоры имеют несколько ядер, а во-вторых, децимация может уменьшить нагрузку на процессор (или улучшить качество фильтра) в десятки и сотни раз.

Наряду с КИХ-фильтрами в цифровой обработке применяют и фильтры с бесконечной импульсной характеристикой (БИХ), которые относятся к классу рекурсивных фильтров, когда при обработке очередного отсчета используются как входные отсчеты, так и фильтрованные. Амплитудно-частотная характеристика БИХ-фильтра в окрестности частот среза описывается, как и в аналоговом случае, степенной функцией, поэтому они часто даже называются так же, например, «фильтр Баттерворта *n*-го порядка». Основным достоинством БИХ-фильтров является меньшее, по сравнению с КИХ, количество арифметических операций, необходимых для их реализации. Недостатки у БИХ-фильтров тоже есть: возможна неустойчивость (самовозбуждение); невозможно оценить задержку сигнала; необходимо фильтровать все отсчеты даже при узкополосной фильтрации с децимацией. К недостаткам можно также отнести невозможность эффективно использовать *SSE*-расширения команд процессоров *Intel*. БИХ-фильтры в дальнейшем рассматриваться не будут, однако программную реализацию алгоритмов БИХ-фильтрации и вычисления коэффициентов (на языке ФОРТРАН) ФНЧ и ФВЧ, а также полосовых можно найти в работе [4].

Еще одной операцией, часто применяемой при обработке сигналов и сводящейся к формуле (1), является дискретное преобразование Фурье (ДПФ). Одной из форм такой операции является цифровой синхронный детектор (ЦСД):

$$X(\omega) = \sum_{i=0}^{N-1} x_i \exp\left[j\omega\left(t_0 + i\Delta_t\right)\right], \qquad (4)$$

где ω – частота, для которой считается фурье-компонента. В радиотехнике фурье-анализ чаще всего используется для определения параметров

синусоидальных сигналов (радиоимпульсов) и построения спектров шума. Собственно, параметров у радиоимпульса с угловой частотой ω, который записывается в виде $A\sin(\omega(t-t_0)+\varphi)$, два: амплитуда A и начальная фаза φ. Эти параметры легко определяются после применения ЦСД (3): $A = 2/N \mid X(\omega) \mid$; $\varphi = \arg(X(\omega))$. Применение же фурье-анализа для анализа спектра шумов основывается на том, что фурье-компоненту (4) можно в некотором приближении считать результатом фильтрации через узкополосный фильтр со средней частотой $f = \omega/2\pi$ и полосой пропускания Δ_f обратно пропорциональной длине реализации N: $\Delta_f = (N\Delta_t)^{-1}$.

Если необходимо проделать преобразование Фурье одновременно на многих частотах, то разумно применить быстрое преобразование Фурье (БПФ). При БПФ число арифметических операций растет с увеличением длины реализации пропорционально $N\cdot\log N$, а не $N\cdot N$ как при использовании ЦСД. Особенностью БПФ является то, что вычисляются фурье-компоненты только для частот (включая нулевую), при которых укладываются целое число волн на реализации.

Можно для наглядности представить БПФ как поворот вектора X размерности N в многомерном пространстве. Здесь X – это совокупность временных отсчетов $\{x_0, \ldots, x_{N-1}\}$, входящих в реализацию. Аналогия с поворотом вектора помогает понять следующие свойства БПФ:

– Сохранение длины вектора (равенство Парсиваля), т. е. сумма квадратов временных отсчетов x_i равняется сумме квадратов модулей фурье-компонент $X(\omega_i)$.

– Количество независимых фурье-компонент (вещественных) равно при повороте количеству исходных отсчетов N. Так как фурье-компоненты комплексны, то их в 2 раза меньше. На самом деле имеется $N/2+1$ частотных компонент. Это частоты, на которых на длине реализации T_R укладывается целое число волн, т. е. удовлетворяющие соотношению $f_i = iF_s/N$, $i = 0, ..., N/2$. Две из компонент – на нулевой частоте, $i = 0$ (постоянная составляющая сигнала) и на частоте Найквиста F_N, $i = N/2$ – вещественны. Остальные $N/2-1$ фурье-компонент комплексны.

Основным ограничением при практическом использовании БПФ является невозможность задания произвольной длины реализации. Наиболее распространенными являются алгоритмы, в которых длина реализации является степенью числа 2. С появлением свободно распространяемой библиотеки *FFTW*, которая умеет проводить БПФ для реализаций с длинами, равными произведению чисел 2, 3 и 5, это ограничение стало несущественным.

Для использования пакета необходимо:

- Скачать последнюю прекомпилированную официальную 32/64-разрядную версию пакета для *Windows* с сайта http://www.fftw.org/.
- После распаковки пакета выполнить команду

```
lib /def:libfftw3f-3.def.
```

После отработки команды создадутся файлы `libfftw3f-3.lib` и `libfftw3f-3.exp`. Утилита `lib` входит в состав *Visual Studio*.

- Файл `libfftw3f-3.lib` включить в ваш проект, а заголовочный файл `fftw3.h` (с помощью директивы `#include`) – в текст программы на *C*/*C*++.

БПФ, как известно, оперирует комплексными массивами данных, но в языке С отсутствует соответствующий тип данных. Чтобы облегчить работу с пакетом *FFTW*, в заголовочный файл `fftw3.h` добавлен, в частности, описатель `fftwf_complex` (комплексные одинарной точности). Можно считать, что у такой переменной как бы добавляется еще один индекс, причем индекс 0 соответствует реальной части переменной, а 1 – мнимой.

Видов преобразований Фурье в пакете *FFTW* реализовано много; при обработке сигналов в подавляющих случаях применяются только прямое и обратное БПФ для вещественных массивов одинарной точности. Процесс быстрого преобразования Фурье разбивается на несколько этапов: вначале создается так называемый план, в котором задаются параметры преобразования. Полученный план используется для запуска процедуры БПФ (в том числе в цикле и в разных местах программы).

Рассмотрим эти этапы подробнее на примере прямого и обратного преобразования Фурье для реализации длиной 1 000 отсчетов:

- Описание массивов

```
float x[1000];fftwf_complex y[501].
```

- Описание планов прямого и обратного фурье-преобразований

```
fftwf_plan prf,invf.
```

- Инициализация планов прямого

```
prf=fftwf_plan_dft_r2c_1d(1000,x,y,
FFTW_FORWARD)
```

и обратного преобразований

```
invf=fftwf_plan_dft_c2r_1d(1000,y,x,
FFTW_ BACKWARD).
```

Названия функций: `fftwf` означает, что функция из пакета *FFTW* и преобразовываются данные одинарной точности; `dft` – что производится преобразование Фурье; `r2c` – что из вещественного массива получается комплексный (прямое преобразование); `c2r` – что из комп-лексного массива получается вещественный (обратное преобразование); `1d` – что массивы одномерные

(могут еще быть двух- и трехмерные). Параметры у функций следующие: длина реализации; входной массив; массив результата; флаг, указывающий на тип преобразования.

- Запуск процесса преобразования Фурье осуществляется функцией `fftwf_execute` с одним параметром (планом):

```
fftwf_execute(prf);  // прямое
fftwf_execute(invf); //обратное.
```

3. Блок ввода данных. Ввод данных в компьютер, который осуществляет АЦП, – единственная операция, для которой может потребоваться создание уникальных собственных физических устройств сопряжения. Собственно, выбрать требуется способ запуска АЦП, частоту квантования и метод ввода данных в компьютер, а также разработать входной аналоговый фильтр, согласованный по входу с антенным устройством. Решение последней задачи выходит за рамки настоящей статьи и облегчается наличием в продаже специализированных микросхем-фильтров.

Основными характеристиками АЦП являются: допустимая частота квантования, диапазон входных значений сигнала и разрядность АЦП. Чаще всего выпускаются 12-, 14- и 16-разрядные АЦП, причем чем выше разрядность АЦП, тем он дороже и тем ниже допустимая частота квантования. Обычно бывает достаточно 12 разрядов. Диапазон входных значений сигнала задается путем выбора коэффициента усиления встроенных в плату усилителей. Радиосигналы являются биполярными, поэтому один бит АЦП является знаковым, а остальные определяют точность преобразования. Для этого понятия существует специальный термин: младший значащий разряд или, по-английски, *Least significant bit* (*LSB*), равный отношению диапазона входных значений к количеству выходных дискретных значений. Для 12-разрядного АЦП с входным диапазоном +/– 10 В, $LSB = 10\ \text{В} / 2048 \approx 5$ мВ. Следует знать еще, что при выходе уровня входного сигнала за границы допустимого диапазона, АЦП выдает код, соответствующий этой границе, т. е. ограничивает (клиппирует) сигнал, срезая верхушки импульсов.

Главной проблемой при создании цифрового приемного устройства является обеспечение непрерывного ввода данных в условиях больших частот квантования. Частично эта проблема решается на аппаратном уровне, когда данные с микросхем АЦП поступают в буфер платы. Физически этот буфер может располагаться на самой плате в виде *FIFO*-буфера (*First In First Out*). Чаще всего он создается непосредственно в оперативной памяти компьютера, и данные передаются в него без участия процессора посредством прямого доступа в память (ПДП), или, по-английски, *Direct Memory Access* (*DMA*). Для плат с *ISA*-шиной размер такого буферного массива ограничивался 64 К 2-байтных слов. С появлением *PCI*-шины разработан новый режим *DMA – PCI Bus-mastering*, в котором это ограничение снято. Важной особенностью ПДП является то, что он может быть «циклическим», когда при достижении конца массива данные продолжают заноситься с его начала.

Формат данных внутри буферного массива не стандартизован, но чаще всего это целые 16-битовые числа. В любом случае, для пересылки данных из буфера в пользовательский массив используются специальные программы-драйверы, поставляемые вместе с платой АЦП. Эти программы могут преобразовать данные в единицы входного напряжения в формате с плавающей запятой при учете коэффициента усиления входных усилителей самой платы. Функции из пакета драйверов показывают, сколько данных было передано платой в буфер (положение указателя на первый свободный элемент массива при «циклическом» режиме ПДП).

Буферный массив, куда пересылаются данные уже в формате с плавающей запятой, пригодном для цифровой фильтрации, также должен быть циклическим, т. е. после его заполнения данные пишутся в него с начала. Размер массива может выбираться произвольным, но желательно, чтобы он являлся степенью 2 и был выровнен на границу 16 байт.

Режимов запуска бывает два: внутренний (от кварца из платы АЦП) и внешний. Точности встроенного кварца обычно хватает только при исследовании шумов. В противном случае приходится использовать сигнал от внешнего стандарта частоты.

Какие условия следует соблюдать при выборе частоты квантования? Критериев несколько, и все они противоречивы. Вот обязательные критерии:

- Помеха на «зеркальной» от рабочей (относительно частоты Найквиста) частоте должна эффективно подавляться входным аналоговым фильтром. Обеспечивается улучшением качества фильтра и/или увеличением частоты квантования.
- При использовании децимации входной последовательности рабочая часть спектра исследуемого сигнала должна полностью попадать в одну из «гармошек».

Желательные критерии, соблюдение которых позволит упростить алгоритмы обработки:

- Частота должна быть как можно меньшей, так как это уменьшает нагрузку на процессор и может позволить приобрести более дешевую плату АЦП (чем ниже допустимая частота квантования, тем ниже стоимость платы).
- Частота квантования должна получаться путем деления частоты задающего генератора

(внутреннего или внешнего стандарта частоты) на целое число. Это упростит реализацию внешнего формирователя частоты квантования.

- После децимации в *n* раз на длине реализации должно укладываться целое число отсчетов ($n\,\Delta_t$).
- Коэффициент децимации n должен делиться на 4. Тогда при КИХ-фильтрации можно будет применять более эффективный вариант *SSE*-команд, использующих данные, выровненные на границу 16 байт (4 плавающих числа одинарной точности).

Усиление входного аналогового сигнала складывается из усиления, обеспечиваемого входным фильтром, и усилителями, встроенными в плату АЦП. Интуитивно ясно, что желательно иметь максимально большой суммарный коэффициент усиления, при котором еще не происходит ограничение сигнала, приводящее к нелинейным искажениям. Впрочем, при наличии сильной негауссовой импульсной помехи такое ограничение может оказаться даже полезным. Физическое объяснение этого критерия связано с тем, что значение квантованного сигнала складывается из трех частей: полезного сигнала, атмосферного шума и шума квантования. Первые две составляющие увеличиваются с увеличением коэффициента усиления. Шум квантования имеет закон распределения, близкий к нормальному со среднеквадратичным отклонением равным *LSB*/2. Отсюда видно, что увеличивая коэффициент усиления, мы уменьшаем вклад шума квантования в общий сигнал. В типичном случае широкополосного входного фильтра полезный сигнал оказывается много меньше атмосферного шума. Из этого следует, что не стоит добиваться большой разрядности АЦП (т. е. малого *LSB*) и большого усиления.

Может показаться, что большая разрядность АЦП нужна для того, чтобы полезный сигнал, лежащий ниже уровня шумов, был достоверно оцифрован с точностью, выше значения *LSB*, но это совершенно необязательно. Шум квантования подчиняется тем же законам, что и атмосферный. Это значит, что при корреляционной обработке отношение сигнал/(шум атмосферный + + шум квантования) будет расти пропорционально корню квадратному от времени накопления (количеству отсчетов, вошедших в обработку).

Осталась еще одна проблема, которую надо решить при реализации блока ввода данных: как обеспечить периодический запуск этой подпрограммы, не связанный с работой других блоков? Проще всего для этого создать отдельную нить (*thread*). Это общий метод, реализующий концепцию параллельного выполнения блоков программы.

Сначала нужно описать прототип функции (у нас она называется `TimerProc`), которая будет выполняться в этом отдельном потоке:
`UINT TimerProc(LPVOID lParam).`

Затем описываем само тело функции `TimerProc`. Эта функция будет работать в едином для программы адресном пространстве, поэтому внутри нее могут использоваться любые глобальные переменные и массивы. Единственным параметром `lParam`, передаваемым в эту функцию, скорее всего, воспользоваться не придется.

Цикличность вызова блока ввода данных обеспечивается применением бесконечного цикла `while`:

```
UINT TimerProc(LPVOID lParam)
{// Описания локальных переменных
  while (1) { Sleep(Millisec);
// Блок ввода данных
            } //конец блока ввода
} // Конец функции TimerProc.
```

Системная функция `Sleep` с параметром типа `int` приостанавливает работу нити на заданное число миллисекунд, обеспечивая периодичность ее работы. Важно, что функция `Sleep` не использует ресурсы процессора. Задержка выбирается такой, чтобы по ее окончании заведомо не произошло переполнение входного циклического буфера.

Сама нить создается с помощью системной функции `AfxBeginThread` в начале работы программы и, благодаря наличию бесконечного цикла, будет существовать до ее окончания:

```
AfxBeginThread(*TimerProc,NULL,
THREAD_PRIORITY_TIME_CRITICAL).
```

Первым параметром является указатель на созданную нами функцию `TimerProc`; вторым параметром выбран `NULL`, так как мы не собираемся передавать никаких параметров; третий параметр – приоритет. Для случая ввода данных, когда нельзя приостанвливать процедуру обработки даже на системные операции ввода/вывода, выбирается максимально возможный приоритет
`THREAD_PRIORITY_TIME_CRITICAL`.

Возможны еще следующие приоритеты:
`THREAD_PRIORITY_LOWEST` – для фоновых задач и `THREAD_PRIORITY_NORMAL` (по умолчанию).

Функция `AfxBeginThread` имеет еще три параметра, которые можно не задавать.

4. Блок предварительной обработки. Этот блок работает именно с принятыми данными, поэтому естественнее всего помещать его в таймерную функцию, рассмотренную в блоке ввода данных, непосредственно за ним.

Основная задача, которую выполняет блок, – это цифровая фильтрация. Ее особенность состоит в том, что массив отсчетов, который подвергается фильтрации, является циклическим, поэтому использовать подпрограмму `svmsse` нельзя. Вариант подпрограммы `fircsse`, учитывающий этот факт, приведен в приложении 4. Ее прототип объявляется следующим образом:

```
extern "C" void fircsse(int M, int N,
int P, float *x,float *y,float *z).
```

Как видно, в подпрограмме добавилось 2 параметра:

- M – определяет размер массива x в 2^M отсчетов;
- P – индекс ячейки массива x, с которой начинается процедура (1). Смысл остальных параметров тот же, что у подпрограммы `svmsse`.

Команда `and eax,edx` обеспечивает очистку старших разрядов текущего индекса массива x (в регистре `eax`) после его изменения (прибавления числа 4), что обеспечивает его цикличность.

Чтобы адрес ячейки массива x, на которую ссылается указатель P, был выровнен на границу 16, необходимо, чтобы он делился на 4. Это возможно, только если коэффициент децимации кратен 4. В противном случае надо заменить команду `movaps` (в цикле суммирования) на более медленную команду `movups`. Компилируется и подключается к проекту подпрограмма так же, как и `svmsse`**.**

Следующей по важности задачей блока является формирование массивов фильтрованных отсчетов заданной длины, используемых блоком расчета текущих характеристик сигнала, который является отдельным «оконным приложением». После формирования очередного массива этому приложению посылается, с помощью функции *PostMessage*, запрос, одним из параметров которого может служить, например, указатель на этот массив. Чтобы обеспечить непрерывность в обработке, при формировании этих массивов используется двойная буферизация, когда после заполнения первого массива начинается заполнение второго; после заполнения второго массива снова начинается заполнение первого и т. д. Проще всего двойную буферизацию можно организовать, объявив двумерный массив с первой размерностью 2. Теперь достаточно сначала присвоить индексу `i` значение 0, а потом, при смене буферов, применять операцию `i=1-i`. Эта схема работает как при формировании массива по известной временной диаграмме, так и в случае работы в ждущем режиме, когда сигнал к началу выборки дается по некоторому условию, например, при превышении входного сигнала заданного порога срабатывания.

Блок может формировать и массивы нефильтрованных отсчетов, которые применяются при исследовании статистических свойств входного шума.

5. Блок расчета текущих характеристик сигнала. Этот блок является «оконным приложением» и инициируется запросом (*message*), посылаемым из блока предварительной обработки. Одним из его параметров является адрес очередного массива, передаваемого на обработку. Дополнительно приложению доступны и все глобальные переменные и массивы, используемые в других блоках программного комплекса.

Этот блок производит предварительную обработку данных и вычисляет текущие значения параметров сигнала. Окончательные значения будут определены только после завершения цикла накопления. Блок вызывается относительно редко, поэтому в нем могут применяться сложные алгоритмы обработки, связанные со спецификой сигнала. Однако некоторые общие приемы мы можем обрисовать.

При определении параметров периодически повторяющихся одинаковых импульсов, что характерно для навигационных систем, самым эффективным способом накопления оказывается простое сложение временных реализаций, относящихся к разным импульсам. Этот прием возможен, только если на временной диаграмме импульсы разделяет целое количество квантов времени. В противном случае каждую реализацию придется обрабатывать по отдельности, усредняя вычисленные параметры.

При обработке нефильтрованных отсчетов целесообразно организовать сбор информации о статистике входного шума (максимальный по модулю уровень, процент клиппированных отсчетов, гистограмма распределения уровня сигнала). Эта информация может помочь выбору правильного коэффициента усиления во входных цепях приемника.

Полезным может оказаться и получение усредненного спектра мощности входного шума, для чего достаточно провести прямое преобразование Фурье массивов входных отсчетов и усреднить квадраты модулей спектральных компонент.

Об окончании процесса усреднения, определяемого либо по количеству обработанных реализаций, либо по достижении конца заданного промежутка времени, извещается блок регистрации и визуализации результатов посредством посылки специального запроса через функцию *PostMessage*.

6. Блок регистрации и визуализации результатов. В этом блоке результаты обработки записываются на магнитном носителе в формате, удобном для дальнейшей обработки. Для этого

каждая запись сопровождается ярлыком, в котором указывается время и дата наблюдения, а при использовании нескольких однотипных приемников, работающих по единой программе, и идентификатором самого приемника. Если дата наблюдения входит в название файла, то эти данные следует размещать в следующем порядке: год, месяц, день. В этом случае в оглавлении названия файлов будут идти в порядке заполнения. При большом объеме *log*-файлов, имеет смысл сохранять результаты в какой-либо базе данных, используя *SQL*-запросы.

Эту же информацию можно выводить и оперативно, на экран дисплея. Однако особенность восприятия информации человеком диктует свои требования к ее представлению:

– при возможности информацию следует выводить в виде графиков, гистограмм, а не таблиц;

– фон графиков необходимо делать белым;

– однотипные графики надо выводить в одно окно разными цветами;

– окна вывода не должны иметь фиксированный размер и положение – есть кнопка минимизации;

– содержимое окон должно меняться достаточно часто – от нескольких секунд до минут.

Что имеет смысл выводить оперативно на экран монитора? Прежде всего, это временные записи наблюдаемых импульсов. В принципе, можно график импульса вывести по точкам, используя функцию *LineTo*. Однако получившийся график вряд ли будет «красивым», потому что для графического отображения импульса желательно иметь 10÷20 точек на протяжении одного периода. При линейной интерполяции между точками график окажется «изломанным». Это положение исправляют, применив интерполяцию с помощью БПФ. Сначала делают прямое БПФ. При обратном БПФ получается временной ряд в исходных точках, а также в промежуточных. Для этого дополняют массив частотных фурье-компонент нулями, увеличив его размер в k раз, и делают обратное БПФ с увеличенным в k раз количеством точек. Для обеспечения правильной нормировки делят на k полученный временной массив.

Очень информативным для человека может оказаться график спектра мощности импульса и график усредненного спектра мощности входного шума.

Статистические свойства входного шума (распределение амплитуд импульсов, интервалов между ними и т. п.) оформляются в виде гистограмм.

Интересной может оказаться и амплитудно-частотная характеристика используемых цифровых фильтров. Для ее получения достаточно подвергнуть прямому фурье-преобразованию массив (выровненных подпрограммой `prepsse`) коэффициентов фильтра.

Если, как в случае приема сигналов радионавигационной системой (РНС), целью приема является оценка параметров передаваемого сигнала известной формы, цифровая обработка позволяет дополнительно оценить точность определяемых параметров, т. е. отношение сигнал/шум, или, по-английски, *Signal-to-Noise Ratio* (*SNR*). Будем исходить из его определения: $SNR = P_{signal} / P_{noise}$, как отношение мощности сигнала к мощности шума в реализации. Обычно измеряется отношение сигнал/шум в децибелах

$$SNR = 10 \log_{10}(P_{signal} / P_{noise}). \quad (5)$$

Метод расчета мощности сигнала (числитель дроби) зависит от вида этого сигнала. В наиболее часто встречающемся случае радиоимпульса известной длины и частоты мы сначала с помощью процедуры синхронного детектирования по формуле (4) вычисляем фурье-компоненту на этой частоте $X(\omega)$. Энергия радиоимпульса оказывается равной квадрату ее модуля $E_{\text{имп}} = \left| X(\omega) \right|^2$, а мощность, понимаемая как энергия, отнесенная к одному отсчету, $P_{\text{имп}} = \left| X(\omega) \right|^2 / N$.

Мощность шума (знаменатель дроби) вычислить легко: это просто сумма квадратов временных отсчетов: $P_{sum} = \sum x(i)\, x(i) / N$. Так как реализация состоит из смеси сигнала и шума, необходимо вычесть из результата мощность сигнала, полученную ранее, $P_{noise} = P_{sum} - P_{signal}$. Теперь можно воспользоваться формулой (5) для расчета отношения сигнал/шум. Эта характеристика прямо влияет на априорную погрешность определяемых параметров.

Выведению формул для оценки априорных погрешностей амплитуды и фазы радиоимпульса нам помогут следующие физические соображения: при узкополосной фильтрации шум можно представить себе как радиоимпульс с частотой, равной средней частоте фильтра, но со случайной фазой, а весь входной сигнал представляется в виде суммы:

$$x(t) = A \sin(\omega t + \varphi) + A_{ns}(t) \sin(\omega t) + \\ + A_{nc}(t) \cos(\omega t), \quad (6)$$

где A – амплитуда радиоимпульса; A_{ns} и A_{nc} – случайные независимые величины, распределенные по нормальному закону. Из определения следует, что мощность каждой из ортогональных компонент шума равна половине дисперсии амплитуд A_{ns} и A_{nc}, а из независимостей распределения – их равенство. Среднеквадратичное отклонение (СКО) компонент шума вычисляется по формуле

$$\sigma = \sqrt{P_{noise}} = \sqrt{(P_{sum} - P_{signal})}. \quad (7)$$

Это СКО характеризует погрешность измерения амплитуды: $\Delta A = \sigma$. Просто из геометрических соображений для случая большого отношения сигнал/шум можно получить погрешность определения фазы (в радианах): $\Delta\varphi = \Delta A/A = \sigma/A$. Везде под погрешностью понимается интервал, в который с вероятностью 68 % попадает случайная величина.

7. Пример: комплексный приемник сигналов РНС «*Loran-C*» и «Альфа». Иллюстрировать изложенные в настоящей статье рецепты мы будем на примере цифрового регистратора сигналов РНС «Альфа» и «*Loran-C*» [1].

Основной идеей, стоящей перед разработчиками при реализации проекта, была возможность создания комплекса, способного анализировать сигналы двух принципиально различных систем (СДВ и ДВ) радионавигации, используя их для определения состояния нижней ионосферы.

При реализации проекта за основу был взят уже существующий комплекс – приемоиндикатор РНС «Альфа». Трудности возникли уже при реализации, казалось бы, технической задачи – переходе на более современную программную среду (с *Visual Studio* 6 и *Compaq Fortran* 6.5, на *VS*2005 и *Intel Fortran* 11). Как выяснилось, был изменен формат нескольких системных функций; многие строковые функции объявлены как устаревшие, и их пришлось заменить на более безопасные аналоги. Использование компилятора *Intel Fortran* 11, который совместим с *VS*2005, привело не только к более компактному и быстрому коду, но и помогло обнаружить несколько ошибок в программе, которые не выявлялись предыдущей версией компилятора.

Важной проблемой оказался выбор частоты квантования АЦП. В индикаторе приема РНС «Альфа» эта частота была равна 50 кГц, что было связано с необходимостью фильтрации частот 11,9; 12,6 и 14,9 кГц с полосой порядка 60 Гц. Легко видеть, что при коэффициенте децимации 100 ширина «гармошки» оказывается равной 250 Гц, а расстояние от рабочих частот до краев своей «гармошки» оказывается не менее 100 Гц. С другой стороны, длина получившегося после децимации кванта времени составляет 20 мкс·100 = 2 мс. На длине реализаций (3,6 с) и сегментов (0,2 и 0,4 с) укладывается целое число таких квантов, что обеспечивает стабильность положения диаграммы излучения. Размер фильтра 1 000 обеспечивает ширину переходной зоны ≈ 25 Гц относительно краев среза полосовых фильтров шириной 120 Гц.

Рабочей зоной системы РНС «*Loran-C*» является окрестность частоты 100 кГц с полосой 20÷40 кГц. Длины реализаций отличаются для разных цепочек станций, но всегда пропорциональны 10 мкс. Самым простым решением, удовлетворяющим требованиям обеих РНС, явилось увеличение частоты квантования в 10 раз, т. е. 500 кГц. Такую частоту легко формировать, используя в качестве опорного генератора сигнал от цезиевого стандарта 5 МГц. Паразитная «зеркальная» частота относительно рабочей частоты РНС «*Loran-C*» в 100 кГц составляет при этом 400 кГц, и для ее подавления достаточно применить очень простой аналоговый фильтр. Длина кванта времени составляет при этом 2 мкс, удовлетворяя условию, что для всех цепочек на длине реализаций укладывается их целое количество. Размер полосового фильтра для рабочей области в окрестности 100 кГц был выбран порядка 200, что обеспечивает полосу расфильтровки в окрестности частот среза фильтра равной 2,5 кГц. Децимация сигнала не производится, и фильтруются все входные отсчеты. Для РНС «Альфа» все параметры фильтрации были увеличены в 10 раз: размер фильтра стал равным 10 000, а коэффициент децимации – 1 000.

Размер кольцевых буферов, используемых для ввода данных от АЦП и цифровой фильтрации, был выбран в 512 K слов. При частоте квантования 500 кГц массивы переполнятся через 1 с, поэтому блок ввода данных опрашивается в 4 раза чаще, каждые 250 мс.

Массивы фильтруемых данных для двух РНС различны:

- Для РНС «Альфа» – это 3 массива, соответствующих трем рабочим частотам системы длиной 900 чисел (3,6 с – цикл работы системы).
- Для РНС «*Loran-C*» фильтруется только одна рабочая частота в окрестности 100 кГц. Для каждой из цепочек станций создаются массивы длиной, равной периоду повторения каждой цепочки. Эти массивы суммируются циклически, т. е. по достижении конца массива новые элементы не замещают ранее введенные значения, а складываются с ними. Таких суммирований делается около 100.

Кроме того, поток входных нефильтрованных отсчетов нарезается на массивы длиной 1 000, после чего они подвергаются операция прямого БПФ, и квадраты спектральных компонент (шаг 0,5 кГц) суммируются. Графики таких усредненных спектров входного шума выводятся в отдельном окне.

Все указанные действия (ввод данных, фильтрация с децимацией трех частот для РНС «Альфа», фильтрация частоты 100 кГц для РНС «*Loran-C*», формирование массивов реализаций для дальнейшей обработки, суммирование спектров мощности входного шума) производятся в одной нити с критическим приоритетом. Не понадобилось даже распределить эти, не связанные между собой, задачи между разными нитями.

При запуске приемного устройства инициализируется, кроме входного блока, еще одна нить – программа расчета поправок к распространению для РНС «Альфа». Поправки меняются крайне медленно (от изменения точки приема и времени суток), поэтому задачу их вычисления можно было сделать фоновой, выполняя в низкоприоритетном режиме. В силу исторических причин программа расчета поправок была написана на языке ФОРТРАН, что не помешало включить ее в один проект с основной программой на языке *C++*. Средством синхронизации, блокирующим попытки одновременного обращаться к массиву поправок из разных нитей, была выбрана так называемая «критическая секция» (*Critical Section*), входящая в набор инструментов *Windows API*.

Отдельной проблемой, которую необходимо решать каждый раз при включении приемного устройства, является установление диаграммы излучения станций, так как приемник включается в произвольный момент времени. Эту проблему можно решить только путем учета специфики работы конкретной системы. В РНС «Альфа» методика поиска начала цикла системы в 3,6 с основывается на том, что у ведущей станции фаза первого сегмента излучения в 0,4 с на частоте 14,9 кГц меняется на π в каждой следующей посылке. Начало цикла определяется как отсчет, в котором модуль разницы ЦСД, вычисленного по формуле (4) для реализации длиной 0,4 с (200 отсчетов) и ЦСД, отстоящего от него на 3,6 с будет максимальным.

Начальная привязка для РНС «*Loran-C*» производится по похожему принципу. Фазы 16 импульсов, относящихся к ведущей и ведомым станциям по-разному, но по известному закону, меняют фазу внутри одной посылки. При поиске положения ведущей и ведомых станций ищутся максимумы сумм 15 коэффициентов корреляции отрезка реализации, равного длине одного импульса, с отрезками такой же длины, отстоящими от исходного, согласно их диаграмме излучения.

Количество окон, в которые выводятся результаты обработки, довольно большое. При разработке дизайна окон применялось правило: каждое окно отвечает за одну конкретную функцию; содержимое окон не зависит друг от друга; их положение и размер настраивается индивидуально с помощью мыши. Окна создаются при запуске приемника, и их нельзя удалить, можно только свернуть. Только главное окно имеет систему меню, через которое можно изменить параметры приемника, а также снять задачу. Обновляется содержимое окон при поступлении новой информации. Для окон, связанных с РНС «Альфа», этот период составляет 3,6 с – цикл системы. Интервалы обновления окон РНС «*Loran-C*» связаны с номером каждой цепочки и составляют 13÷20 с. В каждое такое окно выводится усредненный график временной реализации, вид усредненных импульсов ведущей и ведомых станций и для сравнения эталонный импульс. График усредненного спектра мощности входного шума выводится в отдельное окно каждые 10 с.

Приложения

1. Подпрограмма *svmsse* скалярного произведения массивов с использованием *SSE*-команд (Ассемблер)

```
.686
.XMM
.MODEL FLAT,C
.CODE
;Объявление параметров функции smvsse ;и
способ их передачи (C)
smvsse PROC C USES esi edi ecx,\
N: DWORD,x: DWORD, y: DWORD,z: DWORD
      mov   esi,x
      mov   edi,y
      xor   ecx,ecx
      xorps xmm1,xmm1
;Цикл суммирования, результат в xmm1
smv: movaps xmm0,oword ptr[esi+4*ecx]
     mulps  xmm0,oword ptr[edi+4*ecx]
     addps  xmm1,xmm0
     add    ecx,4
     cmp    ecx,N
     jb     smv
;Суммирование 4-х сумматоров xmm1
;Результат в младшей части xmm0
      movhlps xmm0,xmm1
      addps   xmm0,xmm1
      movaps  xmm1,xmm0
      shufps  xmm0,xmm0,1
      addss   xmm0,xmm1
;Сохранение суммы в 4-м параметре (z)
      mov     esi,z
      movss   dword ptr[esi],xmm0
      ret
smvsse ENDP
       END
```

2. Подпрограмма *lpsd* вычисления коэффициентов ФНЧ (язык *C*)

```
void lpsd(int mm, double t,
double bw, double *b)
// Программа вычисляет веса ФНЧ
// для окна Поттера P310
// Всего имеется 2*mm+1 весов
// Входные параметры подпрограммы:
// mm - размах фильтра
// t  - интервал выборки в секундах
// bw - полоса пропускания в Гц.
// b  - массив результата
{double pi=3.14159265358979;
 double d[]=
 {0.35577019, 0.2436983, 0.07211497,
  0.00630165};
```

```
 int i,k;
 double fact,fi,sum,sumg;
 fact=2.0*bw*t;
 b[0]=fact;fact*=pi;
 for(i=1;i<=mm;i++)
 {fi=i;b[i]=sin(fact*fi)/(pi*fi);};
 b[mm]*=0.5;sumg=b[0];
 for(i=1;i<=mm;i++)
 {sum=d[0];fact=(pi*i)/mm;
  for(k=1;k<4;k++)
  sum+=2.0*d[k]*cos(fact*k);
  b[i]*=sum;sumg+=2.0*b[i];};
 for(i=0;i<=mm;i++) b[i]/=sumg;
 return;
}
```

3. Подпрограмма prepsse подготовки коэффициентов КИХ-фильтра, использующего *SSE*-инструкции (язык *C*)

```
int prepsse(int mm,float *b,float *x)
// Входные параметры:
// mm – размер фильтра
// b – массив коэффициентов фильтра
// Выходные параметры (для svmsse):
// x – массив коэффициентов,
// память выделяется подпрограммой
// Возвращаемое значение (prepsse):
// длина массива коэффициентов
{int m,i,j=0;
// Вычисление длины массива
 m=(2*mm+1)/4+1;m*=4;
// Выделение памяти под массив
 x=(float*)_aligned_malloc(m*4,16);
// Заполнение левой половины
 for(i=0;i<mm;i++) x[j++]=b[mm-i];
// Заполнение правой половины
 for(i=0;i<=mm;i++) x[j++]=b[i];
// Дополнение недостающими нулями
 while(j<m) x[j++]=0.0;
 return m;
}
```

4. Подпрограмма *fircsse* фильтрации входных отсчетов, находящихся в кольцевом буфере с использованием *SSE*-команд (Ассемблер)

```
.686
.XMM
.MODEL FLAT,C
.CODE
;Параметры функции fircsse
;и способ их передачи (C)
fircsse PROC C USES esi edi ecx,\
M: DWORD, N: DWORD, P: DWORD,\
x: DWORD, y: DWORD,z: DWORD
      mov   ecx,32
      sub   ecx,M
      mov   edx,FFFFFFFFh
      shr   edx,cl ; edx - маска
      mov   eax,P
      mov   esi,x
      mov   edi,y
      xor   ecx,ecx
      xorps xmm1,xmm1
;Цикл суммирования, результат в xmm1
fir: movaps xmm0,oword ptr[esi+4*eax]
     mulps  xmm0,oword ptr[edi+4*ecx]
     addps  xmm1,xmm0
     add    eax,4
     and    eax,edx
     add    ecx,4
     cmp    ecx,N
     jb     fir
;Суммирование 4-х сумматоров xmm1
;Результат в младшей части xmm0
      movhlps xmm0,xmm1
      addps   xmm0,xmm1
      movaps  xmm1,xmm0
      shufps  xmm0,xmm0,1
      addss   xmm0,xmm1
;Сохранение суммы в 6-м параметре (z)
      mov     esi,z
      movss   dword ptr[esi],xmm0
      ret
firsse ENDP
       END
```

Библиографический список

1. *Программно-аппаратный* комплекс СПбГУ для исследования нижней ионосферы посредством радиоволн низкочастотных диапазонов / Ю. П. Галюк, М. А. Бисярин, Н. Ю. Заалов, Л. Н. Лутченко // Междунар. конф. «Электромагнитные методы исследования окружающего пространства» (EMES'2012) [Электронный ресурс]: тез. докл. – С. 144–146. – Режим доступа: http://ri.kharkov.ua/emes/EMES2012_Thesis.pdf. – Загл. с экрана.
2. *Шилдт Г.* MFC: Основы программирования / Г. Шилдт; пер. с англ. под ред. В. Р. Гинзбурга. – К.: BHV, 1997. – 556 с.
3. *Мешков А.* Visual C++ и MFC: в 2 т. / А. Мешков, Ю. Тихомиров. – 2-е изд. – СПб: BHV, 2001. – 468 с. (Т. 1); 482 с (Т. 2).
4. *Отнес Р.* Прикладной анализ временных рядов. Основные методы / Р. Отнес, Л. Эноксон; пер. с англ. под ред. И. Г. Журбенко. – М.: Мир, 1982. – 432 с.

Рукопись поступила 28.08.2014.

Y. P. Galyuk

LOW FREQUENCY DIGITAL RECEIVER

Application of digital devices removes many experimental obstacles, as the physical elements are replaced by their virtual computer analogs. A receiver is realized in an ordinary computer supplied by the ADC (analog-digital converter) board and by the appropriate soft. Such a device appears to be much cheaper than an analog one, since the computers and ADC boards are universal pieces of equipment manufactured in great series, and there is a substantial competition in the market. Tuning and adjustments of a digital receiver is simpler than those of an ordinary one and is made much faster: it is reduced to simple editing of the program listing and its re-compilation. However, one must be able to write the programs in the languages of high level and to know the update techniques of signal processing. In the present study, we provide optimal solutions for digital receivers. As an example, we present the organization of a real VLF/LF receiver.

Appendix contains the listings of typical program blocks in the C/C++ and Assembler languages.

Key words: digital receiver, numerical signal processing, real time processing, digital filtration, fast Fourier transform.

Ю. П. Галюк

НИЗЬКОЧАСТОТНИЙ ЦИФРОВИЙ РАДІОПРИЙМАЧ

Використання цифрових пристроїв позбавляє експериментатора від багатьох труднощів, оскільки фізичні блоки замінюються їх віртуальними комп'ютерними аналогами. Радіоприймальний пристрій у такому випадку перетворюється у звичайний комп'ютер, що оснащений платою аналогово-цифрового перетворювача (АЦП) і відповідним програмним забезпеченням. Такий приймач виявляється дешевший за аналоговий, бо комп'ютери та плати АЦП завдяки їх універсальності випускають великими партіями, а на ринку – величезна конкуренція. Налагодження і настройка цифрового приймача, у порівнянні з традиційним, значно спрощується і відбувається швидше; вони зводяться до простого редагування тексту програми та її перекомпіляції. Однак при цьому потрібно вміти програмувати на мовах високого рівня й використовувати сучасні методи цифрової обробки сигналів. У даній роботі описано оптимальні рішення для цифрових радіоприймальних пристроїв і, як приклад, розглядається організація реального СДВ/ДВ радіоприймача. У додатку наведено тексти типових програмних блоків на мові *C*/*C*++ та Ассемблер.

Ключові слова: цифровий радіоприймач, цифрова обробка сигналів, реальний масштаб часу, цифрова фільтрація, швидке перетворення Фур'є.