CHAPTER 18

FFT Convolution
Свертка БПФ

This chapter presents two important DSP techniques, the *overlap-add method*, and *FFT convolution*. The overlap-add method is used to break long signals into smaller segments for easier processing. FFT convolution uses the overlap-add method together with the Fast Fourier Transform, allowing signals to be convolved by multiplying their frequency spectra. For filter kernels longer than about 64 points, FFT convolution is faster than standard convolution, while producing exactly the same result.

Эта глава представляет два важных метода ЦОС, *overlap-add method* (*перекрытие - добавленный метод*), и *свертку БПФ*. *перекрытие - добавленный метод* используется, чтобы разбить длинные сигналы на меньшие сегменты, для облегчения обработки. Свертка БПФ использует перекрытие - добавленный метод вместе с Быстрым преобразованием Фурье, позволяя сигналам быть свернутым, умножая(мультиплицируя) их частотные спектры. Для ядер фильтра длиной, приблизительно, более чем 64 точки, свертка БПФ быстрее, чем стандартная свертка, при получении точно такого же результата.

The Overlap-Add Method
Перекрытие - добавленный Метод

There are many DSP applications where a long signal must be filtered in *segments*. For instance, high fidelity digital *audio* requires a data rate of about 5 Mbytes/min, while digital *video* requires about 500 Mbytes/min. With data rates this high, it is common for computers to have insufficient memory to simultaneously hold the entire signal to be processed. There are also systems that process segment-by-segment because they operate in *real time*. For example, telephone signals cannot be delayed by more than a few hundred milliseconds, limiting the amount of data that are available for processing at any one instant. In still other applications, the *processing* may require that the signal be segmented. An example is FFT convolution, the main topic of this chapter.

Имеется много приложений ЦОС, где длинный сигнал должен быть фильтрованным в сегменты. Например, высокая точность цифрового звуковоспроизведения требует скорости передачи данных приблизительно 5 Мегабайтов в минуту, в то время как цифровое видео требует приблизительно 500 Мегабайтов в минуту. С этими высокими скоростями передачи данных, это обычно для компьютеров, чтобы иметь недостаточно памяти, чтобы одновременно считать полный обрабатываемый сигнал. Имеются также системы, которые обрабатывают"сегмент за сегментом" ("segment-by-segment ") потому что они работают в *реальном времени*. Например, телефонные сигналы не могут быть отсрочены больше чем на несколько сотен миллисекунд, ограничивая количество данных, которые являются доступными для обработки в любой момент. Тем не менее, в других приложениях, обработка может требовать, чтобы сигнал был сегментирован. Пример - свертка БПФ, основная тема этой главы.

The overlap-add method is based on the fundamental technique in DSP: (1) decompose the signal into simple components, (2) process each of the components in some useful way, and (3) recombine the processed components into the final signal. Figure 18-1 shows an example of how this is done for the overlap-add method. Figure (a) is the signal to be filtered, while (b) shows the

filter kernel to be used, a windowed-sinc low-pass filter. Jumping to the bottom of the figure, (i) shows the filtered signal, a smoothed version of (a). The key to this method is how the *lengths* of these signals are affected by the convolution. When an *N* sample signal is convolved with an *M* sample filter kernel, the output signal is *N*+*M*-1 samples long. For instance, the input signal, (a), is 300 samples (running from 0 to 299), the filter kernel, (b), is 101 samples (running from 0 to 100), and the output signal, (i), is 400 samples (running from 0 to 399).

Перекрытие - добавленный метод основан на фундаментальной методике в ЦОС: (1) расчленяют сигнал на простые компоненты, (2) обрабатывают каждый из компонентов некоторым полезным способом, и (3) рекомбинируют(воссоединяют) обработанные компоненты в конечный(заключительный) сигнал. Рисунок 18-1 показывает пример того, как это сделано для перекрытия - добавленного метода. Рисунок (a) - сигнал, который будет фильтрован, в то время как (b) показывает ядро фильтра, которое нужно использовать, в windowed-sinc фильтре нижних частот. Перепрыгнем к основанию(нижней части) рисунка, (i) показывает фильтрованный сигнал, приглаженную версию (a). Ключ к этому методу - то, как на длины этих сигналов воздействует свертка. Когда сигнал выборки *N* свернут с *M* выборкой ядра фильтра, сигнал выхода – *N*+*M*-1 выборок длинной. Например, входной сигнал, (a), является 300 выборками (выполненными от 0 до 299), ядром фильтра, (b), - 101 выборки (выполненной от 0 до 100), и сигнал выхода, (i), - 400 выборок (выполненные от 0 до 399).

In other words, when an *N* sample signal is filtered, it will be *expanded* by *M*-1 points *to the right*. (This is assuming that the filter kernel runs from index 0 to *M*. If negative indexes are used in the filter kernel, the expansion will also be to the *left*). In (a), zeros have been added to the signal between sample 300 and 399 to illustrate where this expansion will occur. Don't be confused by the small values at the ends of the output signal, (i). This is simply a result of the windowed-sinc filter kernel having small values near its ends. All 400 samples in (i) are nonzero, even though some of them are too small to be seen in the graph.

Другими словами, когда сигнал выборки *N* фильтрован, это будет *расширено M*-1 точек *направо* (Это предполагает, что ядро фильтра выполняется от 0 до *M*. Если отрицательные индексы используются в ядре фильтра, расширение будет также *налево*). В (a), нули были добавлены к сигналу между выборкой 300 и 399, чтобы иллюстрировать, где это расширение произойдет. Не перепутайте с маленькими значениями в конце сигнала выхода, (i). Это - просто результат ядра windowed-sinc фильтра, имеющего маленькие значения около его концов. Все 400 выборок в (i) отличные от нуля, даже при том, что некоторые из них слишком маленькие, чтобы быть замеченными в диаграмме(графике).

Figures (c), (d) and (e) show the decomposition used in the overlap-add method. The signal is broken into segments, with each segment having 100 samples from the original signal. In addition, 100 zeros are added to the right of each segment. In the next step, each segment is individually filtered by convolving it with the filter kernel. This produces the output segments shown in (f), (g), and (h). Since each input segment is 100 samples long, and the filter kernel is 101 samples long, each output segment will be 200 samples long. The important point to understand is that the 100 zeros were added to each input segment to allow for the expansion during the convolution.

Рисунки (c), (d) и (e) показывают декомпозицию, используемую в перекрытии - добавленном методе. Сигнал разит на сегменты, с каждым сегментом, имеющим 100 выборок от первоначального сигнала. В дополнение, 100 нулей добавлено направо от каждого сегмента. В следующем шаге, каждый сегмент фильтрован индивидуально, свертывая это с ядром фильтра. Это производит сегменты выхода, показанные в (f), (g), и (h). Так как каж-

дый входной сегмент - 100 выборок длинной, и ядро фильтра - 101 выборка длинной, каждый сегмент выхода будет 200 выборок длинной. Важный пункт, чтобы понять - то, что 100 нулей были добавлены к каждому входному сегменту, чтобы учесть расширение в течении свертки.

Notice that the expansion results in the output segments *overlapping* each other. These overlapping output segments are added to give the output signal, (i). For instance, samples 200 to 299 in (i) are found by adding the corresponding samples in (g) and (h). The overlap-add method produces exactly the same output signal as direct convolution. The disadvantage is a much greater program complexity to keep track of the overlapping samples.

Обратите внимание, что расширение приводит к сегментам выхода, *накладывающимся* друг на друга. Эти сегменты перекрывания выхода добавлены, чтобы дать сигнал выхода, (i). Например, выборки от 200 до 299 в (i), найдены, прибавляя соответствующие выборки в (g) и (h). Перекрытие - добавленный метод производит точно тот же самый сигнал выхода как прямая свертка. Недостаток - намного большая сложность программы, чтобы следить за накладывающимися выборками.

FFT Convolution
Свертка БПФ

FFT convolution uses the principle that *multiplication* in the frequency domain corresponds to *convolution* in the time domain. The input signal is transformed into the frequency domain using the DFT, multiplied by the frequency response of the filter, and then transformed back into the time domain using the Inverse DFT. This basic technique was known since the days of Fourier; however, no one really cared. This is because the time required to calculate the DFT was *longer* than the time to directly calculate the convolution. This changed in 1965 with the development of the Fast Fourier Transform (FFT). By using the FFT algorithm to calculate the DFT, convolution via the frequency domain can be *faster* than directly convolving the time domain signals. The final result is the same; only the number of calculations has been changed by a more efficient algorithm. For this reason, FFT convolution is also called **high-speed convolution**.

Свертка БПФ использует тот принцип, что *умножение* в частотном домене соответствует свертке в домене времени. Входной сигнал преобразован в частотный домен, используя ДПФ, умноженный частотной характеристикой фильтра, и затем преобразован назад в домен времени, используя Обратный ДПФ. Эта основная методика была известна, начиная с дней Фурье; однако, никто этим действительно не озаботился. Это потому, что время, требуемое для вычисления ДПФ, было дольше, чем время, чтобы непосредственно вычислить свертку. Это изменилось в 1965 с развитием Быстрого преобразования Фурье (БПФ). Используя алгоритм БПФ, чтобы вычислить ДПФ, свертка через частотный домен может быть быстрее, чем непосредственно свертка сигналов домена времени. Конечный результат - тот же самый; только число вычислений было изменено более эффективным алгоритмом. По этой причине, свертка БПФ также называется **высокоскоростной сверткой**.

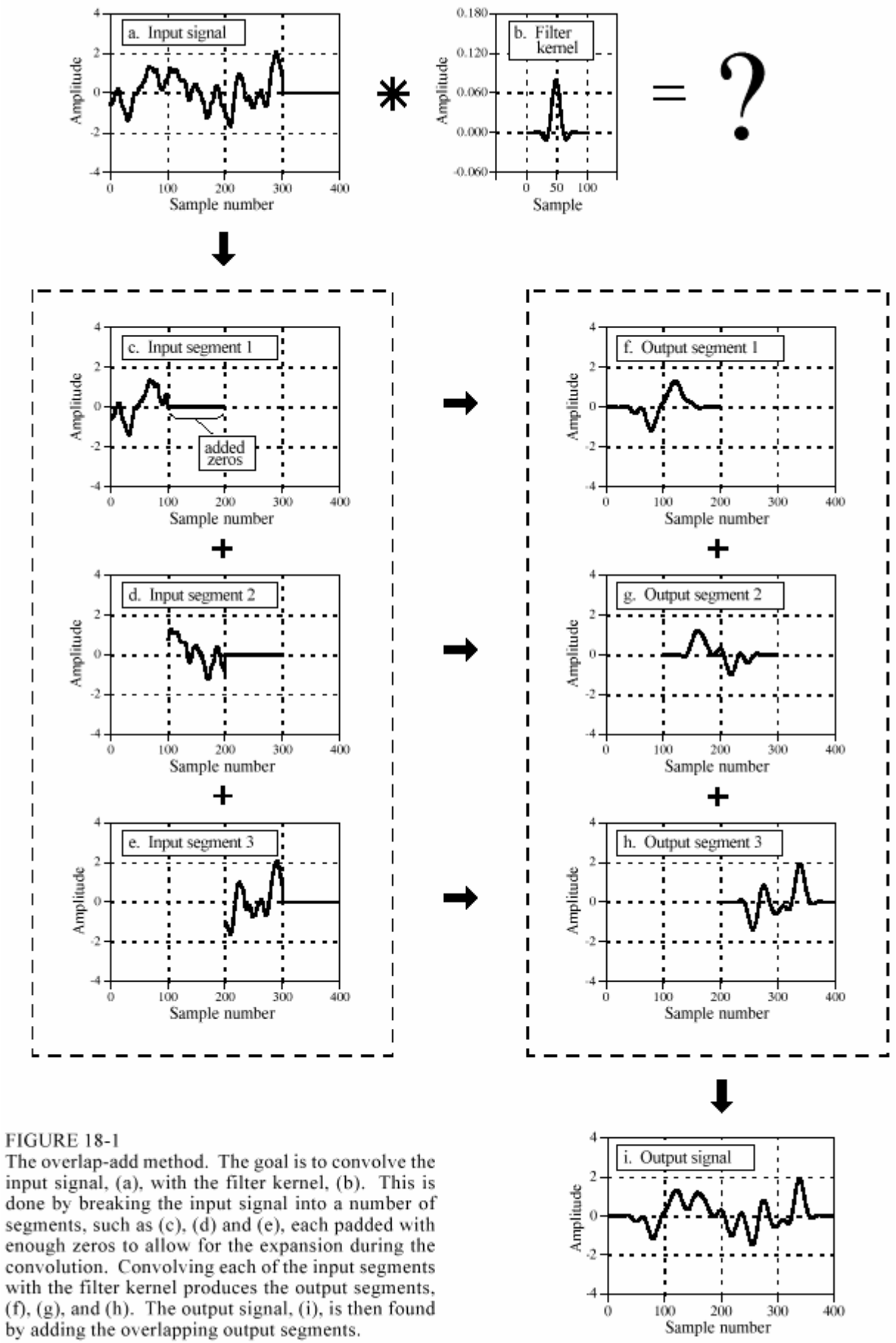


FIGURE 18-1
The overlap-add method. The goal is to convolve the input signal, (a), with the filter kernel, (b). This is done by breaking the input signal into a number of segments, such as (c), (d) and (e), each padded with enough zeros to allow for the expansion during the convolution. Convolving each of the input segments with the filter kernel produces the output segments, (f), (g), and (h). The output signal, (i), is then found by adding the overlapping output segments.

FIGURE 18-1. The overlap-add method.

The goal is to convolve the input signal, (a), with the filter kernel, (b). This is done by breaking the input signal into a number of segments, such as (c), (d) and (e), each padded with enough zeros to allow for the expansion during the convolution. Convolving each of the input segments with the filter kernel produces the output segments, (f), (g), and (h). The output signal, (i), is then found by adding the overlapping output segments.

РИСУНОК 18-1. Перекрытие - добавленный метод.
Цель состоит в том, чтобы свернуть входной сигнал, (a), с ядром фильтра, (b). Это сделано, разбивая входной сигнал на ряд сегментов, типа (c), (d) и (e), каждый дополняемый достаточным количеством нулей, чтобы учесть расширение в течение свертки. Свертка каждого из входных сегментов с ядром фильтра производит сегменты выхода, (f), (g), и (h). Сигнал выхода, (i), тогда найден, складывая накладывающиеся сегменты выхода.

FFT convolution uses the overlap-add method shown in Fig. 18-1; only the way that the input segments are converted into the output segments is changed. Figure 18-2 shows an example of how an input segment is converted into an output segment by FFT convolution. To start, the frequency response of the filter is found by taking the DFT of the filter kernel, using the FFT. For instance, (a) shows an example filter kernel, a windowed-sinc band-pass filter. The FFT converts this into the real and imaginary parts of the frequency response, shown in (b) & (c). These frequency domain signals may not *look* like a band-pass filter because they are in rectangular form. Remember, polar form is usually best for humans to understand the frequency domain, while rectangular form is normally best for mathematical calculations. These real and imaginary parts are stored in the computer for use when each segment is being calculated.

Свертка БПФ использует перекрытие - добавленный метод, показанный в рис. 18-1; только путь, которым входные сегменты преобразованы в сегменты выхода, изменен. Рисунок 18-2 показывает пример того, как входной сегмент преобразован в сегмент выхода сверткой БПФ. Чтобы начинаться, частотная характеристика фильтра найдена, беря ДПФ ядра фильтра, используя БПФ. Для примера, (a) показывает пример ядра фильтра, windowed-sinc полосового фильтра. БПФ преобразовывает это в вещественные и мнимые части частотной характеристики, показанной в (b) и (c). Эти сигналы частотного домена не могут *напоминать* полосовой фильтр, потому что они находятся в прямоугольной форме. Помните, полярная форма - обычно лучше всего для людей, чтобы понять частотный домен, в то время как прямоугольная форма - обычно лучше всего для математических вычислений. Эти вещественные и мнимые части сохранены в компьютере для использования, когда каждый сегмент вычисляется.

Figure (d) shows the input segment to being processed. The FFT is used to find its frequency spectrum, shown in (e) & (f). The frequency spectrum of the output segment, (h) & (i) is then found by multiplying the filter's frequency response, (b) & (c), by the spectrum of the input segment, (e) & (f). Since these spectra consist of real and imaginary parts, they are multiplied according to Eq. 9-1 in Chapter 9. The Inverse FFT is then used to find the output segment, (g), from its frequency spectrum, (h) & (i). It is important to recognize that this output segment is exactly the same as would be obtained by the direct convolution of the input segment, (d), and the filter kernel, (a).

Рисунок (d) показывает входной сегмент к обработке. БПФ используется, чтобы найти его спектр частот, показанный в (e) и (f). Спектр частот сегмента выхода, (h) и (i) тогда найден, умножая частотную характеристику фильтра, (b) и (c), спектром входного сегмента, (e) и (f). Так как эти спектры состоят из вещественных и мнимых частей, они умножены согласно уравнению 9-1 в главе 9. Обратное БПФ тогда используется, чтобы найти сегмент выхода, (g), от его спектра частот, (h) и (i). Важно признать, что этот сегмент выхода

- точно, тот же самый как был бы получен прямой сверткой входного сегмента, (d), и ядра фильтра, (a).

The FFTs must be long enough that *circular convolution* does not take place (also described in Chapter 9). This means that the FFT should be the same length as the output segment, (g). For instance, in the example of Fig. 18-2, the filter kernel contains 129 points and each segment contains 128 points, making output segment 256 points long. This calls for 256 point FFTs to be used. This means that the filter kernel, (a), must be padded with 127 zeros to bring it to a total length of 256 points. Likewise, each of the input segments, (d), must be padded with 128 zeros. As another example, imagine you need to convolve a very long signal with a filter kernel having 600 samples. One alternative would be to use segments of 425 points, and 1024 point FFTs. Another alternative would be to use segments of 1449 points, and 2048 point FFTs.

БПФ должны быть достаточно длинны, что круговая(циклическая) свертка не имела место (также описано в главе 9). Это означает, что БПФ должно быть той же самой длины как сегмент выхода, (g). Например, в примере рис. 18-2, ядро фильтра содержит 129 точек, и каждый сегмент содержит 128 точек, делая сегмент выхода 256 точек длинной. Это запрашивает 256 точек БПФ, которые нужно использовать. Это означает, что ядро фильтра, (a), должно быть дополнено 127 нулями, чтобы принести это к полной длине 256 точек. Аналогично, каждый из входных сегментов, (d), должен дополниться 128 нулями. Как другой пример, вообразите, что Вы должны свернуть очень длинный сигнал с наличием в ядре фильтра 600 выборок. Одна альтернатива была бы должна использовать сегменты по 425 точек, и 1024точки БПФ. Другая альтернатива была бы должна использовать сегменты по 1449 точек, и 2048 точек БПФ.

Table 18-1 shows an example program to carry out FFT convolution. This program filters a 10 million point signal by convolving it with a 400 point filter kernel. This is done by breaking the input signal into 16000 segments, with each segment having 625 points. When each of these segments is convolved with the filter kernel, an output segment of 625 + 400 - 1 = 1024 points is produced. Thus, 1024 point FFTs are used. After defining and initializing all the arrays (lines 130 to 230), the first step is to calculate and store the frequency response of the filter (lines 250 to 310). Line 260 calls a mythical subroutine that loads the filter kernel into XX[0] through XX[399], and sets XX[400] through XX[1023] to a value of zero. The subroutine in line 270 is the FFT, transforming the 1024 samples held in XX[ ] into the 513 samples held in REX[ ] & IMX[ ], the real and imaginary parts of the frequency response. These values are transferred into the arrays REFR[ ] & IMFR[ ] (for: REal and IMaginary Frequency Response), to be used later in the program.

Таблица 18-1 показывает пример программы, чтобы выполнить свертку БПФ. Эта программа фильтрует 10 миллионов точек сигнала, свертывая, это с 400 точками ядра фильтра. Это сделано, разбивая входной сигнал на 16000 сегментов, с каждым сегментом, имеющим по 625 точек. Когда каждый из этих сегментов свернут с ядром фильтра, сегмент выхода 625 + 400 - 1 = 1024 точки - произведенный. Таким образом, используется 1024 точки БПФ. После определения и инициализации всех массивов (строки от 130 до 230), первый шаг должен вычислять и сохранить частотную характеристику фильтра (строки от 250 до 310). Строка 260 вызывает мифическую подпрограмму, которая загружает ядро фильтра в XX[0] через XX[399], и устанавливает XX[400] через XX[1023] к значению нуля. Подпрограмма в строке 270 - БПФ, преобразовывает 1024 выборки, проведенные(поддержанные) в XX[ ] в 513 выборках, проведенных(поддержанных) в REX[ ] и IMX[ ], вещественные и мнимые части частотной характеристики. Эти значения переданы(перемещены) в массивы REFR[ ] и IMFR[ ] (для: REal and IMaginary Frequency Response), чтобы использоваться позже в программе.

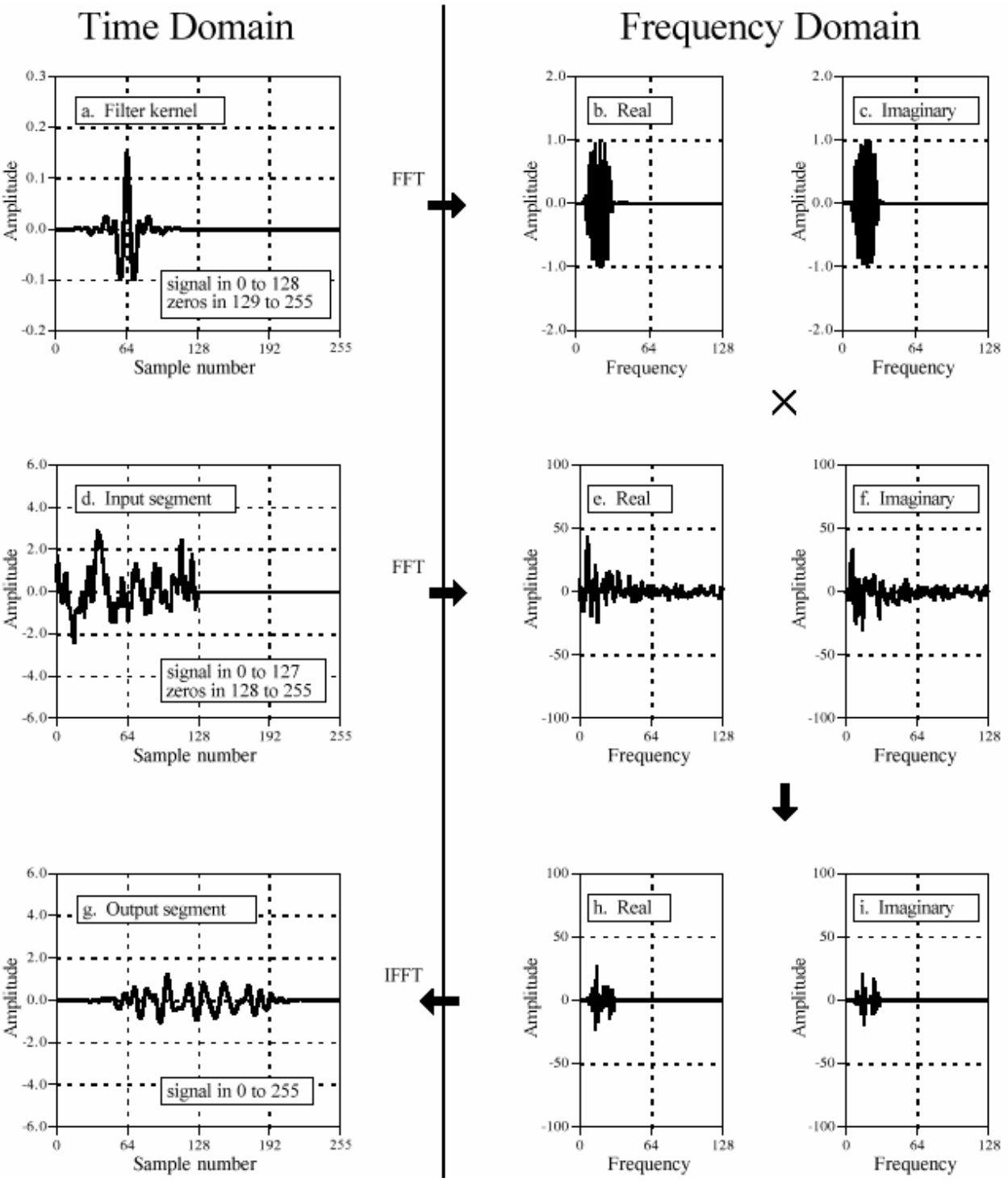


FIGURE 18-2
FFT convolution. The filter kernel, (a), and the signal segment, (d), are converted into their respective spectra, (b) & (c) and (d) & (e), via the FFT. These spectra are multiplied, resulting in the spectrum of the output segment, (h) & (i). The Inverse FFT then finds the output segment, (g).

РИСУНОК 18-2. Свертка БПФ.
Ядро фильтра, (a), и сегмент сигнала, (d), преобразованы в их соответствующие спектры, (b) и (c) и (d) и (e), через БПФ. Эти спектры умножены, приводя к спектру сегмента выхода, (h) и (i). Обратное БПФ тогда находит сегмент выхода, (g).

The FOR-NEXT loop between lines 340 and 580 controls how the 16000 segments are processed. In line 360, a subroutine loads the next segment to be processed into XX[0] through XX[624], and sets XX[625] through XX[1023] to a value of zero. In line 370, the FFT subroutine is used to find this segment's frequency spectrum, with the real part being placed in the 513 points of REX[ ], and the imaginary part being placed in the 513 points of IMX[ ]. Lines 390 to 430 show the multiplication of the segment's frequency spectrum, held in REX[ ] & IMX[ ], by the filter's frequency response, held in REFR[ ] and IMFR[ ]. The result of the multiplication is stored in REX[ ] & IMX[ ], overwriting the data previously there. Since this is now the frequency spectrum of the output segment, the IFFT can be used to find the output segment. This is done by the mythical IFFT subroutine in line 450, which transforms the 513 points held in REX[ ] & IMX[ ] into the 1024 points held in XX[ ], the output segment.

Для FOR-NEXT цикла между строками 340 и 580 управление, как 16000 сегментов обработаны. В строке 360, подпрограмма загружает следующий сегмент, который будет обработан в XX[0] через XX[624], и устанавливает XX[625] через XX[1023] к значению нуля. В строке 370, используется подпрограмма БПФ, чтобы найти спектр частот этого сегмента, с вещественной частью, помещаемой в 513 точек REX[ ], и мнимой части, помещаемой в 513 точек IMX []. Строки от 390 до 430, показывают умножение спектра частот сегмента, проведенного(поддержанного) в REFR[ ] и IMFR[ ], частотной характеристикой фильтра, проведенной(поддержанной) в REFR[ ] and IMFR[ ]. Результат умножения сохранен в REX[ ] & IMX[ ], записывая поверх данных которые предварительно находятся там. Так как это - теперь спектр частот сегмента выхода, IFFT может использоваться, чтобы найти сегмент выхода. Это сделано мифической IFFT подпрограммой в строке 450, которая преобразовывает 513 точек, проведенные(поддержанные) в REX[ ] и IMX[ ] в 1024 точках, проведенных(поддержанных) в XX[ ], сегмента выхода.

Lines 470 to 550 handle the overlapping of the segments. Each output segment is divided into two sections. The first 625 points (0 to 624) need to be combined with the overlap from the *previous* output segment, and then written to the output signal. The last 399 points (625 to 1023) need to be saved so that they can overlap with the *next* output segment.

Строки от 470 до 550, обрабатывают перекрывание сегментов. Каждый сегмент выхода разделен на два раздела. Первые 625 точек (от 0 до 624) должны быть объединены с перекрытием от предыдущего сегмента выхода, и затем записаны в сигнал выхода. Последние 399 точек (от 625 до 1023) должны быть сохранены так, чтобы они могли накладываться со следующим сегментом выхода.

To understand this, look back at Fig 18-1. Samples 100 to 199 in (g) need to be combined with the overlap from the *previous* output segment, (f), and can then be moved to the output signal (i). In comparison, samples 200 to 299 in (g) need to be saved so that they can be combined with the *next* output segment, (h).

Чтобы понимать это, оглянитесь назад в рис. 18-1. Выборки от 100 до 199 в (g), должны быть объединены с перекрытием от *предыдущего* сегмента выхода, (f), и могут тогда быть перемещен в выходной сигнал (i). Для сравнения, выборки от 200 до 299 в (g), должны быть сохранены так, чтобы они могли быть объединены со следующим сегментом выхода, (h).

Now back to the program. The array OLAP[ ] is used to hold the 399 samples that overlap from one segment to the next. In lines 470 to 490 the 399 values in this array (from the previous output segment) are added to the output segment currently being worked on, held in XX[ ]. The mythical subroutine in line 550 then outputs the 625 samples in XX[0] to XX[624] to the file holding the output signal. The 399 samples of the current output segment that need to be held over to the next output segment are then stored in OLAP[ ] in lines 510 to 530.

Теперь назад к программе. Массив OLAP[ ] используется, чтобы провести(держать) 399 выборок, которые накладываются от одного сегмента до следующего. В строках от 470 до 490, 399 значений в этом массиве (от предыдущего сегмента выхода) добавлены к текущему сегменту выхода в настоящее время(будучи) разрабатываемым, проведеным (подержанным) в XX [ ]. Мифическая подпрограмма в строке 550 тогда выводит 625 выборок в XX[0] к XX[624] к файлу, держащему(проводящему) сигнал выхода. 399 выборок текущего сегмента выхода, которые должны быть отложены(сохранены) к следующему сегменту выхода, тогда сохранены в OLAP [ ] в строках от 510 до 530.

After all 0 to 15999 segments have been processed, the array, OLAP[ ], will contain the 399 samples from segment 15999 that should overlap segment 16000. Since segment 16000 doesn't exist (or can be viewed as containing all zeros), the 399 samples are written to the output signal in line 600. This makes the length of the output signal 16000 x 625 + 399 = 10000399 points. This matches the length of input signal, plus the length of the filter kernel, minus 1.

После того как все от 0 до 15999 сегментов были обработаны, массив, OLAP [ ], будет содержать 399 выборок от сегмента 15999, который должен наложиться на сегмент 16000. С тех пор сегмент 16000, не существует (или может быть просмотрен как содержащий все нули), 399 выборок написаны к сигналу выхода в строке 600. Это делает длину сигнала выхода 16000 x 625 + 399 = 10000399 точек. Это соответствует длине входного сигнала, плюс длина ядра фильтра, минус 1.

Speed Improvements
Уточнения Быстродействия

When is FFT convolution faster than standard convolution? The answer depends on the length of the filter kernel, as shown in Fig. 18-3. The time for standard convolution is directly proportional to the number of points in the filter kernel. In comparison, the time required for FFT convolution increases very slowly, only as the *logarithm* of the number of points in the filter kernel. The crossover occurs when the filter kernel has about 40 to 80 samples (depending on the particular hardware used).

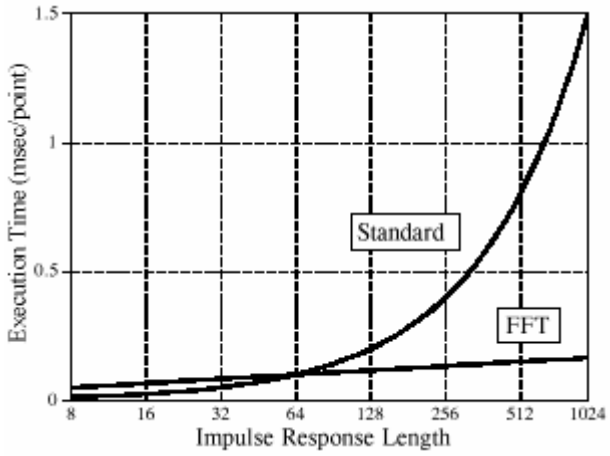
Когда - свертка БПФ быстрее чем стандартная свертка? Ответ зависит от длины ядра фильтра, как показано в рис. 18-3. Время для стандартной свертки - непосредственно пропорционально числу точек в ядре фильтра. Для сравнения, время, требуемое для свертки БПФ, увеличивается очень медленно, только как *логарифм* числа точек в ядре фильтра. Пересечение происходит, когда ядро фильтра имеет приблизительно от 40 до 80 выборок (в зависимости от специфических используемых аппаратных средств).

```
100             'FFT CONVOLUTION
110             'This program convolves a 10 million point signal with a 400 point filter kernel. The input
120             'signal is broken into 16000 segments, each with 625 points. 1024 point FFTs are used.
130             '
130 '           'INITIALIZE THE ARRAYS
140 DIM XX[1023] '              the time domain signal (for the FFT)
150 DIM REX[512]                'real part of the frequency domain (for the FFT)
160 DIM IMX[512]                'imaginary part of the frequency domain (for the FFT)
170 DIM REFR[512]               'real part of the filter's frequency response
180 DIM IMFR[512]               'imaginary part of the filter's frequency response
190 DIM OLAP[398]               'holds the overlapping samples from segment to segment
200 '
210 FOR I% = 0 TO 398           'zero the array holding the overlapping samples
220 OLAP[I%] = 0
230 NEXT I%
240 '
250 '                           'FIND & STORE THE FILTER'S FREQUENCY RESPONSE
260 GOSUB XXXX                   'Mythical subroutine to load the filter kernel into XX[ ]
270 GOSUB XXXX                  'Mythical FFT subroutine: XX[ ] --> REX[ ] & IMX[ ]
280 FOR F% = 0 TO 512 'Save the frequency response in REFR[ ] & IMFR[ ]
290 REFR[F%] = REX[F%]
300 IMFR[F%] = IMX[F%]
310 NEXT F%
320 '
330 '                           'PROCESS EACH OF THE 16000 SEGMENTS
340 FOR SEGMENT% = 0 TO 15999
350 '
360 GOSUB XXXX                  'Mythical subroutine to load the next input segment into XX[ ]
370 GOSUB XXXX                  'Mythical FFT subroutine: XX[ ] --> REX[ ] & IMX[ ]
380 '
390 FOR F% = 0 TO 512 'Multiply the frequency spectrum by the frequency response
400 TEMP = REX[F%]*REFR[F%] - IMX[F%]*IMFR[F%]
410 IMX[F%] = REX[F%]*IMFR[F%] + IMX[F%]*REFR[F%]
420 REX[F%] = TEMP
430 NEXT F%
440 '
450 GOSUB XXXX                  'Mythical IFFT subroutine: REX[ ] & IMX[ ] --> XX[ ]
460 '
470 FOR I% = 0 TO 398           'Add the last segment's overlap to this segment
480 XX[I%] = XX[I%] + OLAP[I%]
490 NEXT I%
500 '
510 FOR I% = 625 TO 1023        'Save the samples that will overlap the next segment
520 OLAP[I%-625] = XX[I%]
530 NEXT I%
540 '
550 GOSUB XXXX                  'Mythical subroutine to output the 625 samples stored
560 ' 'in XX[0] to XX[624]
570 '
580 NEXT SEGMENT%
590 '
600 GOSUB XXXX                  'Mythical subroutine to output all 399 samples in OLAP[ ]
610 END
```

TABLE 18-1

FIGURE 18-3
Execution times for FFT convolution. FFT convolution is faster than the standard method when the filter kernel is longer than about 60 points. These execution times are for a 100 MHz Pentium, using single precision floating point.

РИСУНОК 18-3. Времена выполнения для свертки БПФ. Свертка БПФ быстрее чем стандартный метод, когда ядро фильтра длиннее чем приблизительно 60 точек. Эти времена выполнения - для 100 MHz Pentium, используя одинарную прецизионность с плавающей запятой.



The important idea to remember: filter kernels shorter than about 60 points can be implemented faster with standard convolution, and the execution time is proportional to the kernel length. Longer filter kernels can be implemented faster with FFT convolution. With FFT convolution, the filter kernel can be made as long as you like, with very little penalty in execution time. For instance, a 16,000 point filter kernel only requires about *twice* as long to execute as one with only 64 points.

Важная идея к запоминанию: ядра фильтра короче чем, приблизительно, 60 точек могут быть осуществлены быстрее со стандартной сверткой, и время выполнения пропорционально к длине ядра. Длинные ядра фильтра могут быть осуществлены быстрее со сверткой БПФ. Со сверткой БПФ, ядро фильтра может быть сделано таким длинным, как Вам нравится, с очень небольшой потерей времени выполнения. Например, с 16000 точками ядра фильтра, требуется время выполнения только *вдвое* больше, чем при выполнении ядра с 64 точками.

The *speed* of the convolution also dictates the *precision* of the calculation (just as described for the FFT in Chapter 12). This is because the round-off error in the output signal depends on the total number of calculations, which is directly proportional to the computation time. If the output signal is calculated *faster*, it will also be calculated more *precisely*. For instance, imagine convolving a signal with a 1000 point filter kernel, with single precision floating point. Using standard convolution, the typical round-off noise can be expected to be about 1 part in 20,000 (from the guidelines in Chapter 4). In comparison, FFT convolution can be expected to be an order of magnitude *faster*, and an order of magnitude more *precise* (i.e., 1 part in 200,000).

Быстродействие свертки также диктует прецизионность вычисления (также, как описано для БПФ в главе 12). Это - то, потому что ошибка округления в сигнале выхода зависит от общего количества вычислений, который является непосредственно пропорциональны к времени вычисления. Если сигнал выхода рассчитан быстрее, он будет также рассчитан более точно. Например, вообразите свертывание сигнала с 1000 точек ядра фильтра, с одинарной прецизионностью с плавающей запятой. Используя стандартную свертку, типичный шум округления может ожидаться, чтобы быть приблизительно 1/20000 (от руководящих(основных) принципов в главе 4). Для сравнения, свертка БПФ может ожидаться, чтобы быть порядком величины быстрее, и порядком величины более точная (то есть, 1 часть в 200000).

Keep FFT convolution tucked away for when you have a large amount of data to process and need an extremely long filter kernel. Think in terms of a *million* sample signal and a *thousand* point filter kernel. Anything less won't justify the extra programming effort. Don't want to write your own FFT convolution routine? Look in software libraries and packages for prewritten code. Start with this book's web site (see the copyright page).

Сохраните свертку БПФ, припрятанную для того, когда Вы имеете большое количество данных, чтобы обрабатывать и нуждаться в чрезвычайно длинном ядре фильтра. Думайте скажем о *миллионе* выборок сигнала, и *тысячи* точек ядра фильтра. Что - нибудь меньше не будет обосновывать дополнительное усилие затраченное на программирование. Не хотите записать вашу собственную подпрограмму свертки БПФ? Смотрите в программных библиотеках и пакетах для кода в текстовой форме. Начните с web сайта этой книги (см. страницу авторского права).