ОПТИМИЗАЦИЯ АССЕМБЛЕРНОГО КОДА БПФ, ОРИЕНТИРОВАННОГО НА ОБРАБОТКУ OFDM СИГНАЛОВ СОТОВЫХ СЕТЕЙ СВЯЗИ

Г.О. Бокк, *директор по науке ООО «НСТТ», д.т.н.,* bokkg@yandex.ru

УДК 621.391

Аннотация. Рассматривается метод увеличения производительности ассемблерной программы БПФ/ОБПФ, использующий SSE команды параллельного вычисления. Указываются скрытые вычислительные резервы, содержащиеся на субкомпиляционном уровне. Приведен код ядра программы и экспериментальные данные, подтверждающие доступность обработки на приеме одним ПК класса i3 – CPU 3,70 GHz группового сигнала LTE с рабочей полосой до 40 МГц.

Ключевые слова: вычислительные затраты; ассемблер; SSE команды; БПФ; ОБПФ; LTE; OFDM.

OPTIMIZATION OF THE FFT ASSEMBLER CODE ORIENTED ON THE PROCESSING OF OFDM SIGNALS OF CELLULAR COMMUNICATION NETWORKS

German Bokk, *director for science of LLC «NXTT», doctor of technical sciences*

Annotation. The method of increasing the performance of the FFT/OBPF assembler program using the SSE parallel calculation commands is considered. The latent computational reserves contained at the subcompilation level are indicated. The code of the program core and the experimental data confirming the availability of processing at the reception of one PC class i3 – CPU 3.70 GHz of a group LTE signal with a working band of up to 40 MHz are given.

Keywords: computing costs; assembler; SSE commands; FFT; IFFT; LTE; OFDM.

Введение

На современном этапе развития сотовых сетей связи повсеместно утвердилось использование технологии OFDM, предполагающей, в своей основе, применение алгоритмов быстрого преобразования Фурье (БПФ) и обратного БПФ (ОБПФ) на этапах формирования и приема сигналов. OFDM модуляция обладает рядом существенных преимуществ по сравнению с применявшимися ранее традиционными решениями для формирования радиосигналов. К указанным преимуществам относится высокая спектральная эффективность [1, 2], значительно повышенная устойчивость к условиям многолучевого распространения (к задержкам информационных посылок), достижение практически полной ортогональности сигнально-кодовых конструкций, применяемых в рамках OFDM различными абонентами [3, 4], доступность гибкого перераспределения ресурса радиоканала [5, 6] и т.д. Но реализация OFDM технологии стала возможной только благодаря переходу на полностью цифровые методы обработки сигналов, которые, в свою очередь, оказались доступными для применения в сетях сотовой связи, при достижении производительности цифровых процессоров рубежа порядка 0,5 GFLOPS. Дальнейший прогресс процессорных технологий привел к стимулированию роста производительности радиоинтерфейса сотовых сетей, к переходу на использование более широких рабочих полос радиочастотного спектра, которые стали организовывать для сотовых сетей в более высоких частотных диапазонах [7-11]. Таким образом процесс оказался взаимным: развитие процессорных технологий привело к росту производительности сотовых сетей, а рост производительности до предела повысил требования к процессорным технологиям. Но современные процессорные технологии, фактически, достигли определенного предела в своем развитии. Дальнейший рост производительности требует освоения новых физических принципов, что выступает серьезным препятствием на пути развития сотовых сетей. Поэтому особую актуальность приобретают методы вычислительной оптимизации алгоритмов цифровой обработки сигналов вообще и процедур БПФ/ОБПФ, в частности. Для этого требуется более точно учитывать технологические и алгоритмические нюансы, связанные с реализацией обработки на конкретных программно-аппаратных платформах, изыскивать дополнительные резервы. В данной работе поставленная задача рассматривается на примере реализации алгоритма БПФ на персональном компьютере (ПК) с процессором Intel, с использованием ассемблерного кода с SSE (Streaming SIMD (*SingleInstruction - MultipleData*) Extensions) командами параллельных вычислений.

Технологические особенности организации вычислений на ПК Intel

ПК является мощным вычислительным средством, которое может быть с успехом использовано как для разработки/отладки алгоритмов сигнальной обработки, так и для создания специализированных комплексов связи (см., например, [12-14]) в условиях, когда не требуется миниатюризация и автономность работы. Но бурное развитие цифровых технологий не только качественно повысило показатели производительности, но и существенным образом изменило традиционное соотношение показателей различных рабочих характеристик цифровой обработки. Так для вычислительных платформ на базе ПК современных поколений наибольшие затраты наблюдаются в режимах обмена данными между процессором (сопроцессором) и ОЗУ [15], а такие операции как умножение, сложение, пересылка между регистрами для чисел в формате с плавающей точкой вычисляются практически за один такт. Это существенно изменяет подход к формированию высокопроизводительных алгоритмов. И если в основе классического подхода к организации алгоритмов БПФ/ОБПФ лежит принцип сокращения операций умножения [16], для которого построена целая теория, то для современных вычислительных средств такой принцип можно применять только условно. Требуется дополнительно исследовать возможности по сокращению числа операций обращения к ОЗУ [15]. Именно с этих позиций в представленной работе рассматривается возможность повышения производительности алгоритмов БПФ/ОБПФ, приводится конкретный пример ассемблерного кода, оценивается его быстродействие и, наконец, выполняется сопоставление производительности полученного ассемблерного кода с производительностью, достигаемой при использовании классических вариантов решения поставленной задачи [16]. Также в конце будет приведена оценка производительности полученного алгоритма БПФ для обработки на приеме OFDM сигнала сетей стандартов LTE и LTE-Advanced. Будет установлена предельная частотная полоса сети LTE, для которой возможно осуществлять цифровую обработку сигнала на приеме с помощью одного ПК Inteli3 с тактовой частотой CPU 3,70 ГГц.

Анализ вычислительных затрат алгоритма БПФ для ПК Intel

В классической форме алгоритмы БПФ (ОБПФ) могут быть организованы по схеме с «прореживанием по частоте» и по схеме с «прореживанием по времени» [16-19]. Будем в дальнейшем ориентироваться на схему с «прореживанием по частоте», хотя изложенный подход вполне может быть использован и для схемы «прореживания по времени». Рекуррентное правило «прореживания по частоте», применяемое для организации БПФ, показано на рис. 1.

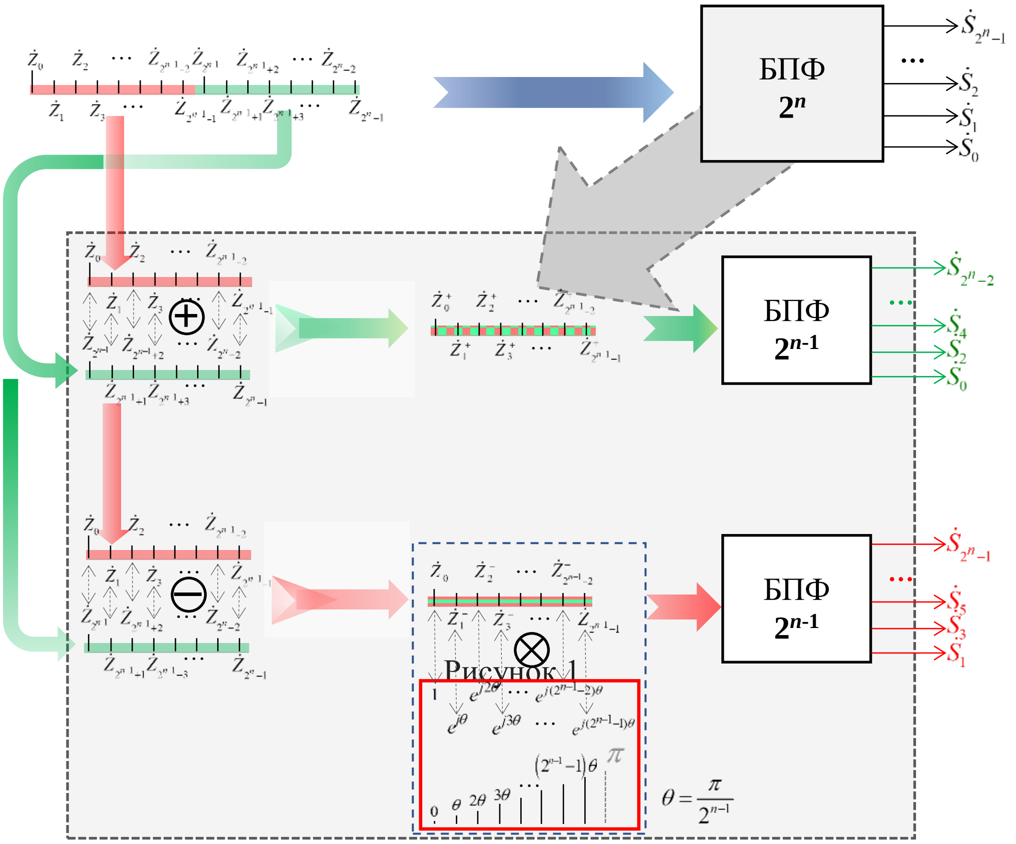

Рисунок 1

На нем показано, как путем тривиальных операций над исходной выборкой данных $\{\dot{Z}_0, \dot{Z}_1, ..., \dot{Z}_{2^n-1}\}$ БПФ размером 2^n сводится к вычислению двух БПФ размером $2^{n\text{-}1}$. Этап сведения предполагает выполнение $2^{n\text{-}1}$ операций сложения и $2^{n\text{-}1}$ операций вычитания, а также $2^{n\text{-}1}$ операцию умножения комплексных чисел. Сразу следует отметить, что часть операций умножения является тривиальными (сводится к замене знака или обмене данными между ячейками действительной и мнимой части числа). Совершенно ясно, что основные вычислительные затраты БПФ сконцентрированы в операциях подготовки сомножителей и, в меньшей степени, выполнении самих умножений. Последнее обстоятельство объясняется тем, что современные математические сопроцессоры Intel выполняют операции умножения чисел с форматом плавающей точки, примерно, за один такт. Именно поэтому умножение перестает доминировать в вычислительных затратах. А основные вычислительные затраты будут связаны с необходимостью формировать последовательность комплексных чисел $\left\{1, e^{j\theta}, e^{j2\theta}, ..., e^{j(2^{n-1}-1)\theta}\right\}$, где $\theta = \dfrac{\pi}{2^{n-1}}$. Прямое использование команды математического сопроцессора fsincos оказалось непригодным для этой цели. Замеры вычислительных затрат, выполненные согласно методике [15], продемонстрировали, что на ее однократное выполнение затрачивается недопустимо большое количество тактов CPU (превосходящее 200). Поэтому осталось только два варианта решения:

- формировать указанные множители заранее, сохраняя их в ОЗУ;

- организовать вычисление указанных множителей с помощью рекуррентных тригонометрических соотношений, предполагающих только операции сложения (вычитания) и умножения:

$$\begin{cases}\cos(m+1)\theta=\cos m\theta\cos\theta-\sin m\theta\sin\theta,\\ \sin(m+1)\theta=\sin m\theta\cos\theta+\cos m\theta\sin\theta,\end{cases} \qquad (1)$$

где: $m = 1,2, \dots , 2^{n-1}-2$.

Именно эти два варианта и нужно сравнить. Начнем со второго. Его реализацию будем рассматривать на базе ассемблерного кода с использованием SSE команд. Ориентация на SSE команды объясняется тремя полезными свойствами.

Во-первых, тестирование вычислительных затрат базовых команд SSE, выполненное по методике, представленной в [15], показало практически одинаковые результаты с затратами на выполнение аналогичных команд математического сопроцессора.

Во-вторых, в отличие от команд математического сопроцессора команды SSE позволяют одновременно выполнять до четырех однотипных операций (умножение, сложение, вычитание, пересылка и т.д.). Это, например, позволяет выполнить преобразование (1) не за четыре операции умножения, операцию сложения и операцию вычитания (или не за три операции умножения и пяти операций сложения, как описано в [16]), а за одну операцию параллельного умножения, одну операцию параллельного сложения и одну операцию параллельного вычитания. Последний вариант представлен в табл. 1 командами 55, 57 и 59.

В-третьих, SSE и SSE2 команды в качестве операндов используют, прежде всего, регистры xmm0-xmm7. Каждый из указанных регистров содержит 128 разрядов данных и может быть «разделен» на четыре сегмента, в каждом из которых может содержаться одно 32 разрядное число, чего вполне достаточно для хранения значения с плавающей точкой в формате одинарной точности (Single). Таким образом SSE команды открывают доступ для промежуточного хранения результатов вычислений в объеме до 32 значений формата Single. В тоже время математический сопроцессор поддерживает работу с восьмью регистрами st(0)-st(7), в которых можно сохранить только до восьми промежуточных результатов.

Поэтому ориентация на SSE команды естественна и оправданна при формировании программных кодов с повышенной производительностью. И в этом нет ничего необычного, так как именно с целью развития производительности математического сопроцессора и было разработано SSE расширение.

В процессе программно-аппаратной оптимизации (на основе команд SSE) процедуры вычислений по формуле (1) удалось выявить довольно значительный дополнительный ресурс, который можно получить при изменении последовательности расчетных точек алгоритма. Действительно, упорядочение по возрастанию фазы в неявном виде заложено в формулу (1). И алгоритм вычислений, построенный в соответствии с (1), предполагает на каждом из $2^{n-1}-2$ шагов применения однотипных вычислений, включающих: параллельное умножение, параллельное сложение, параллельное вычитание и несколько операций пересылки между регистрами xmm.

Но упорядоченное множество фазовых точек:

$$\left\{0,\theta,2\theta,\dots,\left(2^{n-1}-1\right)\theta\right\} \qquad (2)$$

можно разбить на классы так, чтобы внутри каждого из указанных классов расчетные соотношения (1) сводились к тривиальным операциям. Рис. 2 показывает возможный вариант такого разбиения.

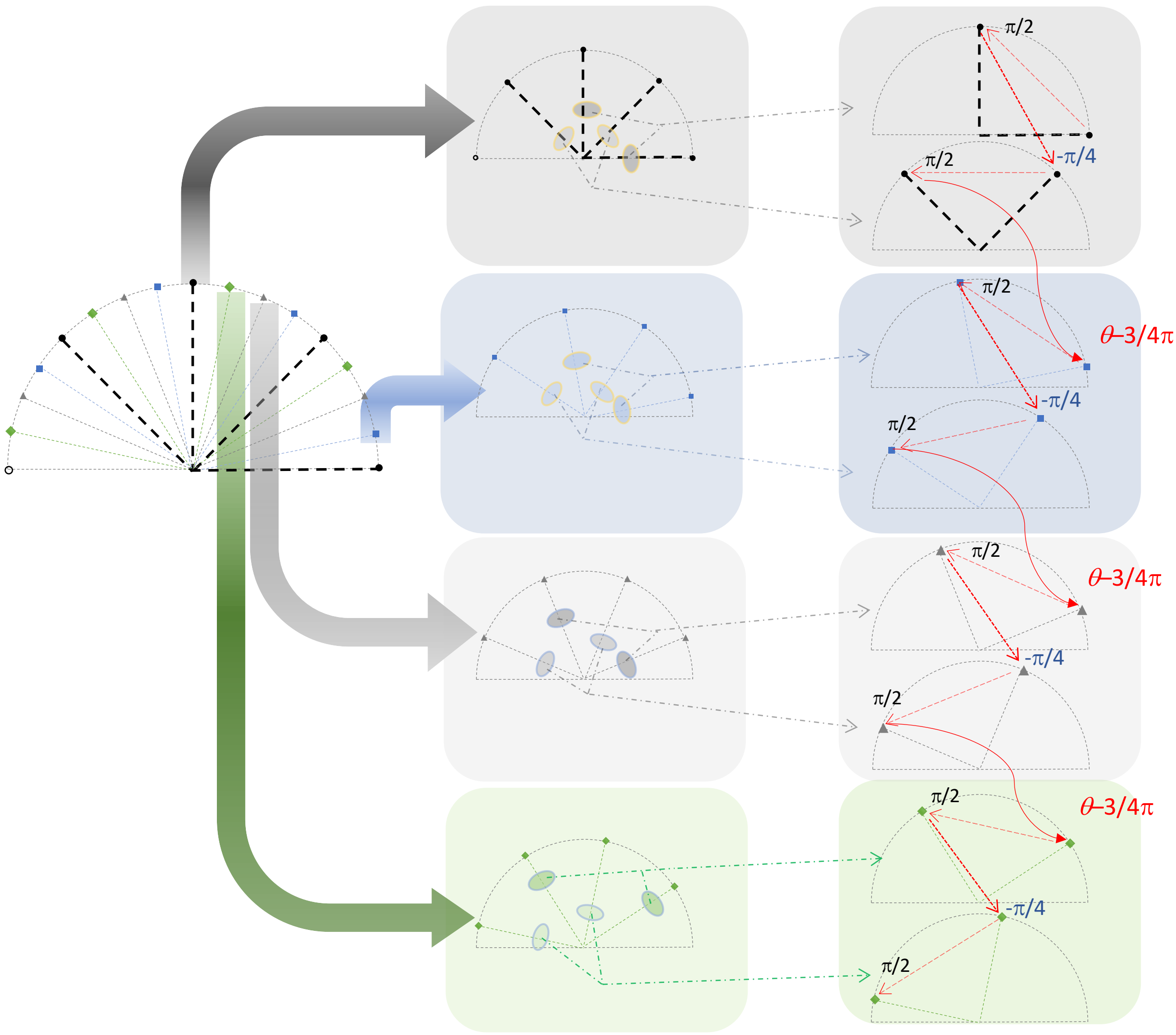

Рисунок 2

На нем слева показано множество комплексных значений $\left\{1, e^{j\theta}, e^{j2\theta}, \ldots, e^{j\left(2^{n-1}-1\right)\theta}\right\}$, соответствующее (2) и подлежащих расчету. В центре рис. 2 показано разбиение на множество классов, в каждом из которых содержится четыре «точки» со значениями фаз, отстоящими друг от друга на $\frac{\pi}{4}$. Поэтому формула (1) внутри выделенных классов будет сводиться к тривиальному виду:

$$\begin{cases} \cos\theta_{i+1}^{k} = \dfrac{1}{\sqrt{2}}\left(\cos\theta_{i}^{k} - \sin\theta_{i}^{k}\right), \\ \sin\theta_{i+1}^{k} = \dfrac{1}{\sqrt{2}}\left(\cos\theta_{i}^{k} + \sin\theta_{i}^{k}\right), \end{cases} \qquad (3)$$

где:

$k = 0,1, \ldots, 2^{n-3}$-1 – номер класса, $i = 0,1,2$ – номер точки внутри класса. Как можно видеть, формула (3) предполагает два умножения, одно сложение и одно вычитание. И только при переходе от 3-й («последней») точки одного класса к 0-й («начальной») точке последующего потребуется вычисление по формуле подобной (1):

$$\begin{cases} \cos\theta_0^{k+1} = \cos\theta_3^k \cos\psi - \sin\theta_3^k \sin\psi, \\ \sin\theta_0^{k+1} = \cos\theta_3^k \sin\psi + \sin\theta_3^k \cos\psi, \end{cases} \tag{4}$$

где:

$\psi = \theta - \frac{3}{4}\pi$ – величина, которая может быть вычисленная заранее.

Но формула (3) может быть дополнительно упрощена по вычислительным затратам. Для этого каждый выделенный класс следует разбить на два двухточечных подкласса так, как показано в правой части рис. 2. В двухточечных подклассах фазы позиций отстоят на $\frac{\pi}{2}$ и поэтому формула (1) реализуется через тривиальную операцию «пересылки»:

$$\begin{cases} \cos\theta_{m,1}^k = -\sin\theta_{m,0}^k, \\ \sin\theta_{m,1}^k = \cos\theta_{m,0}^k, \end{cases} \tag{5}$$

где:

m = 0,1 – номер подкласса, k = 0,1, … , 2^{n-3}-1 – номер класса.

При этом переход между 1-й («последней») точкой первого подкласса и 0-й («начальной») точкой второго подкласса осуществляется по формуле подобной (3):

$$\begin{cases} \cos\theta_{1,0}^k = \frac{1}{\sqrt{2}}\left(\cos\theta_{0,1}^k + \sin\theta_{0,1}^k\right), \\ \sin\theta_{1,0}^k = \frac{1}{\sqrt{2}}\left(\sin\theta_{0,1}^k - \cos\theta_{0,1}^k\right). \end{cases} \tag{6}$$

А переход между классами осуществляется с помощью указанной выше формулы (4), которая с учетом введенных индексов для подклассов принимает вид:

$$\begin{cases} \cos\theta_{0,0}^{k+1} = \cos\theta_{1,1}^k \cos\psi - \sin\theta_{1,1}^k \sin\psi, \\ \sin\theta_{0,0}^{k+1} = \cos\theta_{1,1}^k \sin\psi + \sin\theta_{1,1}^k \cos\psi, \end{cases} \tag{7}$$

где:

$\psi = \theta - \frac{3}{4}\pi$, как и в (4).

В результате, если провести переупорядочивание вычислений на базе предложенной схемы разделения на классы и подклассы, то на каждые четыре точки потребуется две тривиальные операции пересылки вида (5), одна упрощенная расчетная система операций вида (6) и одна полноценная расчетная система операций вида (7). Вычислительная сложность при этом уменьшается примерно в два раза по сравнению с алгоритмом, построенным на прямом воспроизведении расчетов по формуле (1).

В табл. 1 приведено ядро ассемблерного программного кода на базе SSE команд, реализующего БПФ в соответствии с алгоритмом отвечающем формулам (5-7

Таблица 1

№	Ассемблерный код операции	Состояние регистра xmm после операции	Описание операции	Затраты в тактах процессора	Такты замены считывания
1	@1: moveax,esi	Не изменяются	Начало цикла. Сохраняем в регистре eax текущее значение указателя в массиве данных, подвергающихся БПФ	0,5	0,5
2	movsd xmm1,qword ptr [esi+edi]	xmm1: r0 $Re(Z_i)$; r1 $Im(Z_i)$; r2 X; r3 X	Запись в 64 младших разряда xmm1 действительной и мнимой компонент отсчета $i(i<2^{n-1})$	2,2	0
3	addesi,ebx	Не изменяются	Смещаем указатель в массиве данных на 2^{n-1}позиций	0,5	0
4	movsd xmm2,qword ptr [esi+edi]	xmm2: r0 $Re(Z_{i+2^{n-1}})$; r1 $Im(Z_{i+2^{n-1}})$; r2 X; r3 X	Запись в 64 младших разряда xmm2 действительной и мнимой компонент отсчета $i+2^{n-1}$	2,2	0
5	movdqa xmm3,xmm1	xmm3: r0 $Re(Z_i)$; r1 $Im(Z_i)$; r2 X; r3 X	Дублируем данные из xmm1 в регистр xmm3	0,47	0
6	addps xmm1,xmm2	xmm1: r0 $Re(Z_i)+Re(Z_{i+2^{n-1}})$; r1 $Im(Z_i)+Im(Z_{i+2^{n-1}})$; r2 X; r3 X	Складываем отсчеты данных, отстоящие на 2^{n-1} позиций	0,85	0
7	subps xmm3,xmm2	xmm3: r0 $Re(Z_i)-Re(Z_{i+2^{n-1}})$; r1 $Im(Z_i)-Im(Z_{i+2^{n-1}})$; r2 X; r3 X	Находим разность отсчетов данных, отстоящих на 2^{n-1} позиций.	1,3	0
8	pshufd xmm3,xmm3,00010100b	xmm3: r0 $Re(Z_i^-)$; r1 $Im(Z_i^-)$; r2 $Im(Z_i^-)$; r3 $Re(Z_i^-)$	Дублируем в старших 64 разрядах регистра xmm3 младшие разряды, подготавливая состояние для последующей операции комплексного умножения	0,75	0
9	pshufd xmm4,xmm0,11010000b	xmm4: r0 $\cos(i\cdot\theta)$; r1 $\cos(i\cdot\theta)$; r2 $\sin(i\cdot\theta)$; r3 $-\sin(i\cdot\theta)$	Размещаем в регистре xmm4 отсчеты гармонических квадратур в порядке, нужном для комплексного умножения	0,75	0
10	mulps xmm3,xmm4	xmm3: r0 $Re(Z_i^-)\cos(i\cdot\theta)$; r1 $Im(Z_i^-)\cos(i\cdot\theta)$; r2 $Im(Z_i^-)\sin(i\cdot\theta)$; r3 $-Re(Z_i^-)\sin(i\cdot\theta)$	Умножение. Формируем компоненты комплексного произведения	1,8	0
11	movHLps xmm2,xmm3	xmm2: r0 $Im(Z_i^-)\sin(i\cdot\theta)$; r1 $-Re(Z_i^-)\sin(i\cdot\theta)$; r2 X; r3 X	Записываем старшие 64 разряда регистра xmm3 в младшие разряды регистра xmm2	0,75	0
12	addps xmm2,xmm3	xmm2: r0 $Re(Z_i^- e^{-j(i\cdot\theta)})$; r1 $Im(Z_i^- e^{-j(i\cdot\theta)})$; r2 X; r3 X	В младших 64 разрядах регистра xmm2 компоненты комплексного произведения $Z_i^-\cdot e^{-j(i\cdot\theta)}$	1,1	0
13	movsd qword ptr [esi],xmm2	Не изменяются	Сохраняем в рабочем массиве ОЗУ результат произведения $Z_i^-\cdot e^{-j(i\cdot\theta)}$	1,1	0
14	subesi,ebx	Не изменяются	Убираем смещение на 2^{n-1}позиций указателя массива данных, внесенное операцией №3	1	0
15	movsd qword ptr [esi],xmm1	Не изменяются	Сохраняем в рабочем массиве ОЗУ значение Z_i^+ , используемое на следующем шаге рекуррентной процедуры БПФ	1,1	0
16	add esi,edx; add esi,edx	Не изменяются	Смещаем указатель в массиве данных на числопозиций 2^{n-2}, что соответствует сдвигу фазы гармонического множителя на $\pi/2$	1	0,5
17	pshufd xmm0,xmm0,10010011b	xmm0: r0 $\cos(i\cdot\theta+\pi/2)$; r1 $\sin(i\cdot\theta+\pi/2)$; r2 $-\cos(i\cdot\theta+\pi/2)$; r3 $-\sin(i\cdot\theta+\pi/2)$	Меняя порядок значений в xmm0 ($-\sin(i\theta)$; $\cos(i\theta)$; $\sin(i\theta)$;$-\cos(i\theta)$), добиваемся эффекта смещения фазы на $\pi/2$ для гармонического множителя	0,75	0,75
18 ⋮ 31	Повторение операций №№ 2 - 15	~	Для повторяющихся операций не используется организация в виде вызываемой процедуры для экономии производительности	15,87	0
32	subesi,edx	Не изменяются	Смещаем указатель данных на -2^{n-3} позиций. Это соответствует сдвигу фазы гармонического множителя на $-\pi/4$ (т.е. esi указывает на точку $i+2^{n-3}$)	1	0,5
33	pshufd xmm3,xmm0,00111001b	xmm3: r0 $\cos(i\cdot\theta)$; r1 $\sin(i\cdot\theta)$; r2 $-\cos(i\cdot\theta)$; r3 $-\sin(i\cdot\theta)$	Записываем в xmm3 значения из регистра xmm0 в измененном порядке, подготавливая расчет гармонического множителя со сдвигом фазы на $\pi/4$	0,75	0,75

№	Ассемблерный код операции	Состояние регистра xmm после операции	Описание операции	Затраты в тактах процессора	Такты замены считывания
34	addps xmm0,xmm3	xmm0: r0 = $\cos(i\cdot\theta)-\sin(i\cdot\theta)$; r1 = $\sin(i\cdot\theta)+\cos(i\cdot\theta)$; r2 = $-\cos(i\cdot\theta)+\sin(i\cdot\theta)$; r3 = $-\sin(i\cdot\theta)-\cos(i\cdot\theta)$	Складываем значения в xmm0 со значениями xmm3, подготавливая расчет гармонического множителя со сдвигом фазы на $\pi/4$	0,85	0,85
35	mulps xmm0,xmm7	xmm0: r0 = $\cos(i\cdot\theta+\pi/4)$; r1 = $\sin(i\cdot\theta+\pi/4)$; r2 = $-\cos(i\cdot\theta+\pi/4)$; r3 = $-\sin(i\cdot\theta+\pi/4)$	Умножаем значения ячеек xmm0 на $1/\sqrt{2}$, содержащиеся в ячейках xmm7. Получаем компоненты гармонического множителя со сдвигом фазы на $\pi/4$	1,8	1,8
16 ⋮ 49	Повторение операций №№ 2 - 15	~	Для повторяющихся операций не используется организация в виде вызываемой процедуры для экономии производительности	15,87	0
50 51	Повторение операций №16, №17	~	Для повторяющихся операций не используется организация в виде вызываемой процедуры для экономии производительности	1,75	1,25
52	dececx	*Не изменяются*	Уменьшаем счетчик повторений цикла	1	0
53	jz @2	*Не изменяются*	Если цикл закончен, то переходим на метку @2	2 (в случае без перехода)	0
54	pshufd xmm0,xmm0,00010100b	xmm0: r0 = $\cos(i\theta+3/4\pi)$; r1 = $\sin(i\theta+3/4\pi)$; r2 = $\sin(i\theta+3/4\pi)$; r3 = $\cos(i\theta+3/4\pi)$	Меняем значения в ячейках регистра xmm0, подготавливая к расчету гармонического множителя с фазой $(i+1)\cdot\theta$	0,75	0,75
55	mulps xmm0,xmm5	xmm0: r0 = $\cos(\theta-3/4\pi)\cdot\cos(i\cdot\theta+3/4\pi)$; r1 = $\cos(\theta-3/4\pi)\cdot\sin(i\cdot\theta+3/4\pi)$; r2 = $-\sin(\theta-3/4\pi)\cdot\sin(i\cdot\theta+3/4\pi)$; r3 = $\sin(\theta-3/4\pi)\cdot\cos(i\cdot\theta+3/4\pi)$	Одним параллельным умножением готовим все компоненты для расчета гармонического множителя с фазой $(i+1)\cdot\theta$ (xmm5 содержит квадратуры с фазой $\theta-3/4\ \pi$)	1,8	1,8
56	movHLps xmm3,xmm0	xmm3: r0 = $-\sin(\theta-3/4\pi)\cdot\sin(i\cdot\theta+3/4\pi)$; r1 = $\sin(\theta-3/4\pi)\cdot\cos(i\cdot\theta+3/4\pi)$; r2 = X; r3 = X	Записываем старшие 64 разряда xmm0 в младшие разряды xmm3. Это формирует слагаемые для расчета гармонического множителя с фазой $(i+1)\cdot\theta$	0,75	0,75
57	addps xmm0,xmm3	xmm0: r0 = $\cos((i+1)\cdot\theta)$; r1 = $\sin((i+1)\cdot\theta)$; r2 = X; r3 = X	В ячейках 64 младших разрядов регистра xmm0 теперь содержаться компоненты комплексного гармонического множителя с фазой $(i+1)\cdot\theta$	1,1	1,1
58	xorps xmm3,xmm3	xmm3: r0 = 0; r1 = 0; r2 = 0; r3 = 0	Обнуляем регистр xmm3	0,95	0,95
59	subps xmm3,xmm0	xmm3: r0 = $-\cos((i+1)\cdot\theta)$; r1 = $-\sin((i+1)\cdot\theta)$; r2 = X; r3 = X	Вычитаем из обнуленного xmm3 значения регистра xmm0. В младших 64 разрядах получаем отрицательные значения компонент комплексного гармонического множителя с фазой $(i+1)\cdot\theta$	1,3	1,3
60	movLHps xmm0,xmm3	xmm0: r0 = $\cos((i+1)\cdot\theta)$; r1 = $\sin((i+1)\cdot\theta)$; r2 = $-\cos((i+1)\cdot\theta)$; r3 = $-\sin((i+1)\cdot\theta)$	Записываем в старшие ячейки регистра xmm0 значения 2-х младших ячеек регистра xmm3. Теперь состояние регистра xmm0 соответствует началу очередного прохода цикла (от точки с номером $(i+1)$)	0,75	0,75
61	movesi,eax	*Не изменяются*	Восстанавливаем из eax значение указателя в массиве данных, подвергающихся БПФ, отвечающее завершающемуся проходу (от точки с номером i)	0,5	0,5
62	add esi,2*TypeSingle	*Не изменяются*	Смещаем значение указателя в массиве данных, подвергающихся БПФ, на позицию одной точки. Получаем состояние, отвечающее началу очередного прохода цикла (от точки с номером $(i+1)$)	0,5	0
63	jmp @1	*Не изменяются*	Переход на начало цикла	2	0
64	@2:	*Не изменяются*	Метка выхода из цикла	0	0

В первом столбце указаны порядковые номера ассемблерных команд, во втором – сами команды, в третьем – состояние xmm регистра-приемника после завершения текущей команды, в четвертом – комментарии и пояснения к выполняемой команде ассемблера, в пятом – вычислительные затраты на выполнение команды (данные получены согласно методике, изложенной в [15]), в шестом – повторяются вычислительные затраты пятого столбца, если текущая команда по своему назначению предназначена для формирования отсчета «гармонического» множителя (т.е. используется для того, чтобы избежать операции считывания данных из ОЗУ), в противном случае в позицию шестого столбца помещается нулевое значение.

Суммируя вычислительные затраты пятого столбца, получаем, что на четыре расчетные позиции требуется 69,41 тактов CPU. При этом из них 14,8 такта затрачивается непосредственно на операции формирования «гармонических множителей», которые заменяют операции считывания из ОЗУ, составляющие основу альтернативного варианта алгоритма БПФ, рассмотренного ниже. В итоге получаем, что на одну расчетную позицию алгоритма БПФ в предложенном ассемблерном коде будет в среднем затрачиваться 17,35 тактов CPU, из которых 3,7 такта будет расходоваться на формирование «гармонического множителя».

Для ПК Inteli3 – CPU 3,70 GHz полученные результаты приводят к следующим расчетным значениям временных затрат:

- БПФ 256-3,68 мкс/активное процессорное ядро;
- БПФ 1024-15,48 мкс/активное процессорное ядро;
- БПФ 4096-103,56мкс/активное процессорное ядро.

По умолчанию ОС Windows на платформе современных многоядерных ПК распределяет доминирующей вычислительной задаче два процессорных ядра, как показано на рис. 3.

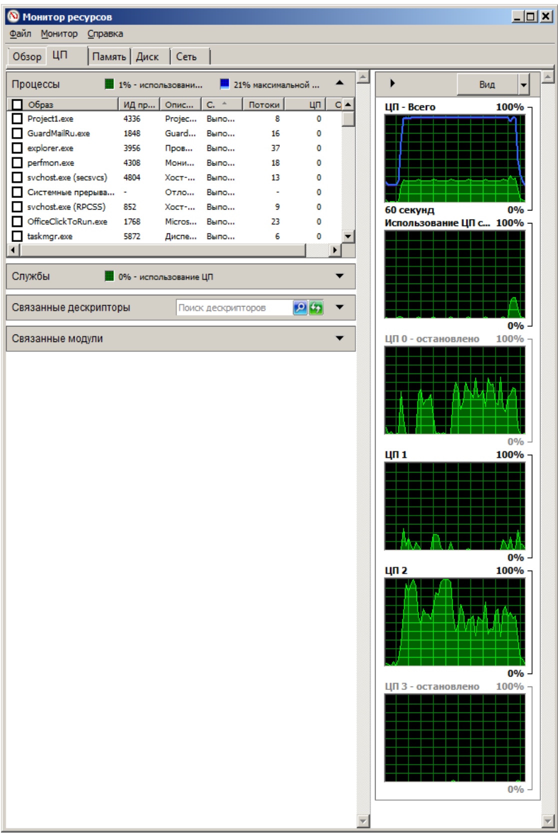

Рисунок 3

Это приводит к следующим временным затратам на вычисления:

- БПФ 256-1,84 мкс;

- БПФ 1024-8,74 мкс;
- БПФ 4096-51,78 мкс.

В первом варианте реализации алгоритма БПФ, в котором «гармонические сомножители» рассчитываются заранее, а в процессе работы считываются из ОЗУ, требуется выполнение следующих операций:

- сохранить в ОЗУ значение из регистра (пусть это будет регистр eax), содержащее, например, указатель смещения рабочей позиции в массиве данных (на это затрачивается порядка 10 тактов CPU);
- считать указатель смещения в массиве «гармонических множителей» (в eax) (на это затрачивается порядка 10 тактов CPU);
- увеличить указатель массива «гармонических множителей» на одну позицию (на это затрачивается порядка 0,5 такта CPU);
- считать в регистр xmm значения квадратур нужного «гармонического множителя» (на это затрачивается порядка 2,2 тактов CPU, если алгоритм построен для чисел формата Single (одинарной точности));
- сохранить из регистра (eax) в ОЗУ указатель смещения в «массиве гармонических множителей» (на это затрачивается порядка 10 тактов CPU);
- восстановить из ОЗУ значение регистра eax, в котором изначально (см. шаг 1) предполагалось нахождение указателя смещения рабочей позиции в массиве данных (на это затрачивается порядка 10 тактов CPU).

Такие большие затраты на обмен данными с ОЗУ вызваны тем, что с одной стороны тактовая частота шины данных в ПК в 3-4 раза ниже тактовой частоты CPU, а с другой – тем, что центральный процессор обладает примерно в два раза более низкой производительностью в режиме обмена с ОЗУ по сравнению с математическим сопроцессором.

В итоге для первого варианта реализации БПФ на формирование квадратур «гармонического сомножителя» будет затрачиваться в среднем 42,7 такта CPU. Это значительно превосходит значение 3,7 тактов CPU, полученное для случая с расчетом «гармонических сомножителей». Если рассматривать алгоритм БПФ в целом, то для варианта со считыванием из ОЗУ получим, что на одну расчетную позицию будет затрачиваться 56,35 тактов CPU, что приведет к увеличению вычислительных затрат в 56,35/17,35 = 3,25 раз по сравнению с алгоритмом, использующим предложенную технику расчета «гармонических сомножителей».

Реальные замеры временных затрат, выполненные на ПК Inteli3 – CPU3,70GHz по методике, изложенной в [15], в режиме двух активных процессорных ядер показали, что на выполнение БПФ 1024 затрачивается:

- 8,74 мкс – в случае использования предложенного алгоритма с расчетом «гармонических сомножителей»;
- 30,5 мкс – в случае использования алгоритма со считыванием из ОЗУ, предварительно сформированных «гармонических сомножителей».

То есть, выигрыш от применения предложенного алгоритма с расчетом составил 30,5/8,74 = 3,48 раз, что оказалось даже несколько выше расчетного показателя (3,25 раз).

В заключение представляется полезным привести численные оценки, устанавливающие границы возможного применения разработанного алгоритма БПФ на платформе ПК, для обработки OFDM-сигналов сотовых сетей стандартов LTE и LTE-Advanced. Стандартом LTE (и LTE-Advanced) устанавливается базовый режим работы линии Down с длительностью OFDM символа 71,3 мкс [17]. При этом разнос между поднесущими составляет 15 кГц, а рабочая полоса может составлять 3 МГц (200 поднесущих), 5 МГц (333 поднесущих), 10 МГц

(667 поднесущих), 15 МГц (1000 поднесущих), 20 МГц (1333 поднесущих), 30 МГц (только для LTE-Advanced, 2000 поднесущих) и 40 МГц (только для LTE-Advanced, 2667 поднесущих). В результате, если ориентироваться на работу в полосе до 30 МГц, то вполне достаточно алгоритма БПФ 2048, который по оценкам временных затрат на ПК i3 – CPU 3,70 GHz, требует для выполнения порядка 21 мкс, что занимает менее 1/3 от длительности символа OFDM, составляющего 71,3 мкс. Остальные вычислительные резервы могут быть использованы для выполнения других операций по обработке. А вот для обработки полосы в 40 МГц, если не применять алгоритм Винограда [16], потребуется БПФ 4096, который по оценкам временных затрат требует на выполнение порядка 51,78 мкс. Это значение меньше длительности символа OFDM, но резерва на оставшиеся операции по обработке становится недостаточно (таких резервов должно быть примерно в два раза больше, чем затраты на БПФ). Поэтому для этого случая требуется задействовать ресурсы более двух ядер ПК (на современных ПК их обычно четыре и более).

Заключение

Предложен высокопроизводительный алгоритм БПФ, ориентированный на реализацию на платформе ПК с использованием ассемблерных команд SSE-расширения математического сопроцессора. Отмечается, что современные технологии цифровой обработки кардинально изменили соотношения вычислительных затрат базовых ассемблерных команд. Поэтому нужно стремиться сокращать не умножения, а операции обмена между регистрами процессора и ОЗУ. На примере БПФ предложена структура алгоритма, построенного на базе ассемблерного программного кода с SSE командами, в котором минимизировано число операций обращения к ОЗУ. Экспериментально и теоретически показано, что предложенный алгоритм обладает в 2,5 раза более высоким быстродействием по сравнению с вариантом, в котором «гармонические сомножители» БПФ формируются заранее и сохраняются в ОЗУ. Установлено, что для реализации обработки на прием OFDM сигналов линии Down сотовой сети стандарта LTE-Advanced с рабочей полосой до 40 МГц достаточно одно ПК класса i3 – CPU 3,70 GHz.

Литература

1. Аверьянов Р.С., Шорин А.О. Оптимизация ансамбля OFDM-сигналов в сетях мобильной связи // Электросвязь, 2017. – № 2. – С. 41-46.
2. Аверьянов Р.С., Шорин А.О. Оценка оптимальных параметров OFDM-сигналов с учетом мобильности абонентов // Электросвязь, 2015. – № 12. – С. 60-65.
3. Шорин А.О. Оценка влияния скорости перемещения абонентов на удельную интенсивность потока потерь соединений в сетях с OFDM-сигналами // Электросвязь, 2017. – № 1. – С. 35-39.
4. Лохвицкий М.С. Алгоритмы оптимального приема сигнала и обучающие последовательности в сотовой связи // Сборник материалов (тезисов) 35-й международной конференции РАЕН «Мобильный бизнес: перспективы развития и реализации систем радиосвязи в России и за рубежом». Китай, 2014. – С. 16-25.
5. Volodina E., Plossky A. Features of the Digital Dividend Implementation in Conditions of Great Population Density Discontinuity and Limitation of the Frequency Resource // В сборнике: Proceedings of EMC Europe 2011 York - 10th International Symposium on Electromagnetic Compatibility. – 2011. – С. 664-669.
6. Володина Е.Е., Плосский А.Ю. Критерии кластерного подхода к перераспределению радиочастотного спектра при внедрении цифрового телевидения // T-Comm: Телекоммуникации и транспорт, 2012. – Т. 6. – № 12. – С. 4-6.
7. Шорин О.А., Токарь Р.С. Алгоритмы синтеза сотовых систем связи 2G // Спецтехника и связь, 2008. – № 1. – С. 58-63.
8. Lokhvitskij M.S. MOSITALTEL Training Centre // Электросвязь, 1995. – № 11. – С. 15.
9. Володина Е.Е., Девяткин Е.Е., Бессилин А.В. Прогноз развития рынка услуг перспективных радиотехнологий в России // Век качества, 2011. – № 1. – С. 52-55.

10. Бутенко В.В., Володина Е.Е., Девяткин Е.Е., Бессилии А.В., Суходольская Т.А. Концепция WAPECS как современное направление использования РЧС // Электросвязь, 2008. – № 9. – С. 1-7.
11. Володина Е.Е. Экономические вопросы использования радиочастотного спектра как производственного ресурса и объекта государственного регулирования // Электросвязь, 2015. – № 4. – С. 50-54.
12. Шорин О.А., Щучкин В.М. Использование интеллектуальных антенн в системах мобильной связи для снижения перегрузок // Труды МАИ, 2012. – № 53. – С. 19.
13. Патент № 2069055 РФ. Система охранной сигнализации «РОСА» / Аристархов Г.М., Ерохин Г.А., Николаев В.Т., Пантикян Р.Т., Шорин О.А. Заявлено 19.07.1994.
14. Акимов В.Н., Бабин А.И., Шорин А.О. Радиомодемы диапазонов VHF/UHF в задачах охраны и мониторинга объектов // Спецтехника и связь, 2009. – № 1. – С. 50-58.
15. Шорин А.О. Технология повышения производительности программ ассемблера, использующих SSE команды, для цифровой обработки радиосигналов на ПК // Сборник материалов (тезисов) 40-й международной конференции РАЕН «Мобильный бизнес: перспективы развития и реализации систем радиосвязи в России и за рубежом». Маврикий, 2017. – С. 7-9.
16. Блейхут Р. Быстрые алгоритмы цифровой обработки сигналов. – М.: Мир, 1989. – 448 с.
17. Sesia S., Toufik I., Baker M. LTE – The UMTS Long Term Evolution: From Theory to Practice/ A John Wiley & Sons, Ltd., 2011. – 752 p.
18. Lokhvitskij M.S. Mositaltel training centre // Электросвязь, 1995. – № 11. – С. 15.
19. Шорин А.О. Оценка влияния скорости перемещения абонентов на удельную интенсивность потока потерь соединений в сетях с OFDM-сигналами // Электросвязь, 2017. –№ 1. – С. 35-39.