Модуль быстрого преобразования Фурье

Алексей Гребенников (Московская обл.)

Статья содержит описание модуля быстрого преобразования Фурье, написанного на языке Verilog и реализованного для ПЛИС семейства Xilinx Spartan 6.

Введение

Быстрое преобразование Фурье (БПФ) является важнейшим алгоритмом современной цифровой обработки сигналов (ЦОС) и является общим названием любого метода уменьшения вычислительной сложности дискретного преобразования Фурье (ДПФ). Первые теоретические работы по БПФ принадлежат немецкому математику Карлу Фридриху Гауссу. Широкое применение БПФ началось после опубликования в 1965 г. Д. Кули и Д. Тьюки статьи с оригинальным описанием алгоритма (Cooley–Tukey FFT Algorithm). Вычислительные машины того времени уже справлялись с задачей практической реализации БПФ, но метод Кули–Тьюки позволял ускорить вычисления в 5–6 раз. С тех пор элементная база вычислительной техники значительно изменилась, появились новые алгоритмы вычисления БПФ, однако алгоритм Кули–Тьюки остаётся популярным.

В настоящее время БПФ реализуют в основном с помощью ЦПОС и ПЛИС. Значительное увеличением ёмкости и быстродействия микросхем программируемой логики облегчает реализацию алгоритмов БПФ. Предлагаемая статья содержит описание модуля быстрого преобразования Фурье, написанного на языке Verilog и предназначенного для ПЛИС семейства Xilinx Spartan 6. Отладка проекта проводилась на тестовой плате SP605, все исходные коды проекта содержатся в архиве fft_sopc.zip (www.soel.ru).

Теоретические основы БПФ

Как упоминалось выше, БПФ – это алгоритм вычисления ДПФ, которое является методом разложения дискретного периодического сигнала в ряд тригонометрических функций. Теория метода была разработана французским физиком и математиком Жаном Батистом Фурье, который доказал, что любой дискретный периодический сигнал может быть представлен комбинацией простейших тригонометрических функций (синусов и косинусов). Математическая часть преобразования Фурье достаточно сложна и будет рассмотрена в статье только в объёме, минимально необходимом для реализации БПФ. Более подробно с теорией метода можно ознакомиться в [1, 2].

Существует два типа преобразования Фурье – действительное ДПФ и комплексное ДПФ. Для реализации БПФ необходимо использовать комплексное ДПФ. Предположим, на входе имеется дискретный сигнал $X(k)$, состоящий из N отсчётов. Его ДПФ будет выглядеть следующим образом:

$$X(k)=\sum_{n=0}^{N-1}x(n)e^{-j\left(\frac{2\pi}{N}\right)nk}$$

$$(k=0, 1, ..., N-1),$$

$$W_N=e^{-j\left(\frac{2\pi}{N}\right)},$$

Бит-реверсная адресация

Отсчёты исходной последовательности		Отсчёты после бит-реверсной сортировки	
десятичный адрес отсчёта	двоичный адрес отсчёта	десятичный адрес отсчёта	двоичный адрес отсчёта
0	0000000	0	0000000
1	0000001	64	1000000
2	0000010	32	0100000
3	0000011	96	1100000
4	0000100	16	0010000
5	0000101	80	1010000
6	0000110	48	0110000
7	0000111	112	1110000
8	0001000	8	0001000

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk}$$

$$(k=0, 1, ..., N-1).$$

Комплексный компонент уравнения W_N в англоязычной литературе часто называется twiddle factor (поворачивающий коэффициент) и также может быть выражен комбинацией синусов и косинусов:

$$W_N=\cos(2\pi/N)-j\sin(2\pi/N).$$

Количество отсчётов входного сигнала N, как правило, равно степени 2, хотя существуют методы вычисления БПФ для произвольного числа входных отсчётов. При малой величине N время вычисления ДПФ прямым методом и методом БПФ сравнимы. Однако с увеличением размерности входного сигнала преимущества БПФ по скорости вычисления могут достигать сотен раз.

Рассмотрим более подробно реализацию БПФ методом Кули–Тьюки для входного сигнала с размерностью 128 отсчётов.

Алгоритм БПФ

Суть БПФ заключается в том, что входной сигнал большой размерности N, в данном случае 128, разбивается на N сигналов единичной размерности. Затем для каждого единичного сигнала вычисляется спектр, т.е. происходит переход из временной области в частотную. На последнем этапе N единичных спектров объединяются в один общий спектр.

Первый этап разделения входного сигнала называется декомпозицией. На нём применяется т.н. бит-реверсная адресация. Допустим, отсчёты входного сигнала нумеруются в прямой последовательности 0, 1, 2, 3 и т.д. Для формирования бит-реверсной последовательности необходимо единицы адреса отсчёта в двоичной форме переставить в обратном порядке, как показано в таблице. Для адресации 128 отсчётов требуется семь адресных битов. В таблице приведены первые девять отсчётов, для всех остальных отсчётов бит-реверсный адрес вычисляется аналогично. После бит-реверсной сортировки входные отсчёты бу-

дут следовать в порядке 0, 64, 32, 96, 16, 80 и т.д.

Затем для каждого единичного сигнала вычисляется спектр. На этом этапе не требуется никаких программных действий, поскольку спектр единичного сигнала равен соответствующему базисному сигналу ДПФ. Последний шаг БПФ заключается в объединении единичных спектров. Этот шаг является самым сложным и выполняется в несколько этапов. Основная операция объединения спектров, изображённая на рисунке 1, называется «бабочкой» (butterfly). Алгоритм, при котором операция «бабочка» одновременно выполняется для двух входных сигналов, называется БПФ с основанием 2. Именно такой алгоритм рассматривается в данной статье. Другим распространённым основанием БПФ является 4.

Операция «бабочка», в свою очередь, состоит из нескольких этапов. Сначала второй входной сигнал IN2 умножается на поворачивающий коэффициент W_N (twiddle factor). Затем первый выходной сигнал OUT1 получается путём суммирования результата умножения и первого входного сигнала IN1. Вторым выходным сигналом является разность между первым входным сигналом IN1 и результатом умножения. Для полного объединения спектра размерностью N отсчётов требуется выполнение $\log_2 N$ циклов операций «бабочка». При размерности 128 требуется 7 циклов, причём каждый цикл состоит из 64 операций, так как при основании 2 в одной базовой операции БПФ задействовано два входных отсчёта.

Полная схема БПФ для 128 отсчётов достаточно громоздкая, поэтому на рисунке 2 в качестве примера изображена схема БПФ для 8 отсчётов, которая состоит из трёх циклов ($\log_2 8$). В зависимости от порядка использования поворачивающих коэффициентов и операции «бабочка» различают БПФ с прореживанием по времени и БПФ с прореживанием по частоте. На рисунке 2 изображён алгоритм с прореживанием по времени, этот же алгоритм реализован в модуле для ПЛИС; видно, что входные отсчёты отсортированы согласно бит-реверсной адресации.

Во время первого цикла для всех «бабочек» используется один и тот же поворачивающий коэффициент W_0. Пары отсчётов для второго цикла формируются согласно рисунку 2, при этом поворачивающие коэффициенты начинают чередоваться: для первой «бабочки» используется коэффициент W_0, для второй – W_2, для третьей и четвёртой – снова коэффициенты W_0 и W_2 соответственно. Во время третьего цикла используются все поворачивающие коэффициенты $W_0 – W_3$.

Заметим, что общее число поворачивающих коэффициентов равно половине размерности входного сигнала, т.е. для входного сигнала размерностью 128 отсчётов потребуется 64 коэффициента. Пары сигналов в этом случае будут формироваться аналогично БПФ для 8 отсчётов. Во время первого цикла все операции умножения будут выполняться с поворачивающим коэффициентом W_0, во время второго цикла будут чередоваться коэффициенты W_0 и W_{32}, во время третьего цикла будут чередоваться коэффициенты W_0-W_{16}-W_{32}-W_{48}, во время четвёртого цикла – W_0-W_8-W_{16}-W_{24}-W_{32}-W_{40}-W_{48}-W_{56}. Таким образом, число задействованных коэффициентов будет увеличиваться, и во время последнего, седьмого цикла, будут задействованы все 64 поворачивающих коэффициента.

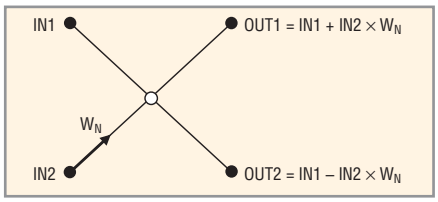

Рис. 1. Базовая операция БПФ

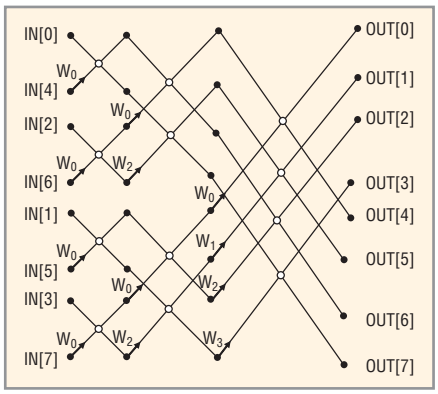

Рис. 2. БПФ для 8-точечного входного сигнала

Аппаратная реализация БПФ

Рассмотрим более подробно реализацию описанного выше алгоритма БПФ для ПЛИС. Блок-схема системы на кристалле, реализующей алгоритм, изображена на рисунке 3. Через порт UART происходит тестирование ядра

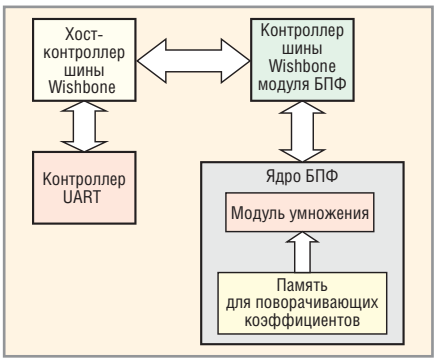

Рис. 3. Блок-схема системы на кристалле

БПФ, обмен данными осуществляется через шину Wishbone. В ядре БПФ отдельными модулями выделены блоки умножения и память для поворачивающих коэффициентов. В данной версии реализации ядро БПФ принимает данные от контроллера UART, которые, в свою очередь, генерируются тестовой программой на компьютере. Полученные данные обрабатываются: для входного вектора данных вычисляется БПФ, результирующие данные пересылаются обратно в UART и затем в графическом виде отображаются на компьютере.

Все математические операции в ядре БПФ выполняются в формате с фиксированной точкой. Входные данные от UART содержат 14 бит в формате Q5.9, т.е. для представления дробной части служат девять младших разрядов, затем четыре разряда представляют целую часть и старший разряд определяет знак. Отрицательные числа представлены в формате дополнения до двух. Далее входные данные расширяются до 18 бит; в последующем все данные представлены в формате Q9.9, т.е. на целую часть отведено 8 разрядов. При таком представлении данных возможно хранение чисел в диапазоне от –256 до 255,998046875 с шагом 0,001953125. Шаг определяется как минимальное значение формата Q9.9. При 9 разрядах, отведённых под дробную часть, возможно хранение 512 комбинаций чисел 1/512 = 0,001953125.

Работой ядра управляет конечный автомат, блок-схема которого показана на рисунке 4. Сам конечный автомат содержится в файле fft_top.v. При отсутствии входных данных автомат находится в состоянии FFT_IDLE. Сигнал начала преобразования *conv_start_i* генерируется модулем шины Wishbone (файл wb_fft.v) после получения пакета данных от контроллера UART. При этом конечный автомат ядра БПФ переходит в состояние FFT_STAGE1. Бит-реверсная сортировка выполняется модулем контроллера шины Wishbone путём реверса адресных битов: assign buf_ptr_inv = {buf_ptr[0], buf_ptr[1], buf_ptr[2], buf_ptr[3], buf_ptr[4], buf_ptr[5], buf_ptr[6]}, где buf_ptr_inv – адрес реверсированных отсчётов.

После выполнения умножения и сложения на стадии FFT_STAGE1, автомат переходит в состояние FFT_STAGE2. Таким образом, последовательно выполняются все семь ступеней преобразования, и после окончания этого процесса конечный автомат переходит в состояние FFT_DONE. При этом устанавливается в единичное значение сигнал *conv_rdy_o*, преобразованные данные передаются модулю контроллера шины Wishbone и далее персональному компьютеру через UART.

Ядро содержит два буфера памяти. На первой стадии входные данные берутся из одного буфера и результаты вычислений записываются в другой буфер. На следующей стадии всё происходит наоборот. Все математические операции выполняются модулем умножения (см. рис. 3), исходный код которого находится в файле mult.v.

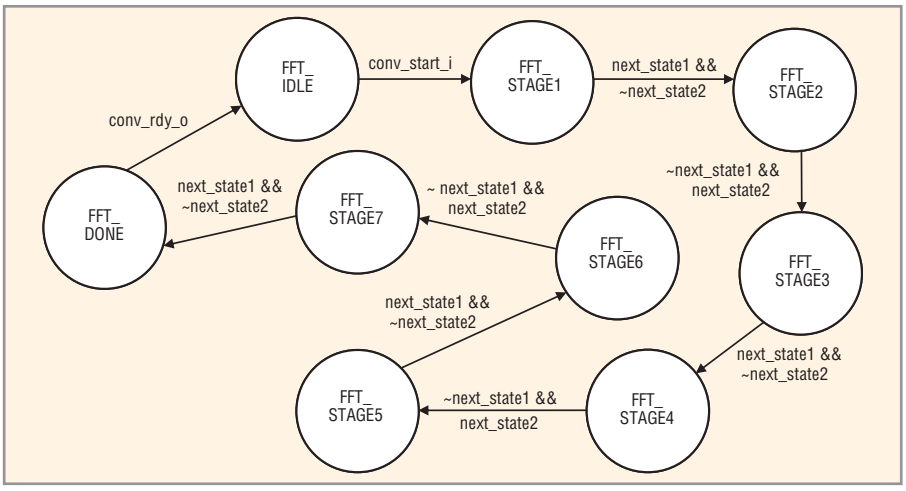

Рис. 4. Блок-схема конечного автомата ядра БПФ

Рассмотрим более подробно работу этого модуля.

Модуль умножения

Основой модуля умножения является примитив DSP48A1, предназначенный для ПЛИС семейства Spartan 6. Подробное описание блока DSP48A1 содержится в [3]. Примитив DSP48A1 позволяет аппаратно выполнять умножение, а также пре- и пост-суммирование или вычитание данных. Эти функции часто востребованы в приложениях ЦОС.

Упрощённая схема блока DSP48A1 изображена на рисунке 5. Блок содержит 8-битный конфигурационный регистр, с помощью которого можно включать или выключать различные функции примитива. Прямоугольниками обозначены регистры, с помощью которых можно буферизировать входные и промежуточные сигналы. Буквенные названия входов и выходов соответствуют названиям в шаблоне блока DSP48A1 на языке Verilog в среде PlanAhead. Поскольку для быстрого вычисления ДПФ необходимо применение комплексного БПФ, модуль умножения работает с комплексными числами.

Пусть на входе имеются два комплексных числа $x + yi$ и $u + vi$. Формула умножения комплексных чисел выглядит следующим образом:

$$(x + yi)(u + vi) = (xu - yv) + (xv + yu)i.$$

Соответственно, для умножения двух комплексных чисел необходимо использование четырёх умножителей – примитивов DSP48A1. Блок-схема умножителя комплексных чисел показана на рисунке 6. На четыре умножителя подаются различные комбинации действительной и мнимой частей комплексных чисел. Входные данные имеют разрядность 18 бит. После умножения получается 36-битное число, которое в модуле DSP48A1 расширяется до 48 бит. Результаты умножения суммируются и вычитаются в соответствии с вышеприведённой формулой, и затем получается итоговое комплексное число с действительной частью *re_o* и мнимой частью *im_o*.

Примитив DSP48A1 выдаёт 48-битный результат, который необходимо округлить до 18 бит, чтобы использовать в качестве входа во время следующего цикла вычислений. Для округления числа с фиксированной точкой в

формате Q9.9 с 48 бит до 18 бит отбрасываются младшие девять бит результата и фиксируются последующие 18 бит, т.е. в выходном регистре P[47:0] используются биты P[26:9]. Однако если результат умножения используется в дальнейших операциях сложения/вычитания в блоке DSP48A1, то используются все 48 разрядов.

Поскольку в операции «бабочка» умножение требуется для второго входного отсчёта и коэффициента поворота, как это видно из рисунка 2 (IN2*W_N), именно эти составляющие подаются на входы умножителей. Таблица коэффициентов поворота хранится в файле trom.v. Эти коэффициенты также являются комплексными числами. Действительная часть равна значениям косинуса, а мнимая – значениям синуса. Первый коэффициент twd[0] равен cos(0) + *j*sin(0), т.е. вычисляется для нулевого угла. Затем для каждого последующего коэффициента угол уменьшается на π/64, и новый аргумент равняется arg – π/64.

Все полученные значения синусов и косинусов затем представляются в формате с фиксированной точкой Q9.9 и запоминаются в файле trom.v. Для автоматизированного вычисления коэффициентов поворота и многих других отладочных функций использовалась вспомогательная программа XilinxCOM, исходные коды и исполняемый файл которой находятся в архиве XilinxCOM.zip.

Работой модуля умножения управляет конечный автомат, блок-схема которого изображена на рисунке 7. В отсутствие входных данных автомат находится в состоянии MULT_IDLE. После установки сигнала *start_i* в единичное значение конечный автомат переходит в состояние MULT_M1 и начинается операция умножения комплексных чисел – второго входного сигнала *re2_i*, *im2_i* и коэффициента поворота *twr_i*, *twi_i*. Согласно вышеприведённой формуле, умножение комплексных чисел включает в себя операции как собственно умножения, так и сложения и вычитания. На стадии MULT_M1 вычисляются произведения *re2i* * *twr_i*, *im2_i* * *twi_i*, *re2_i* * *twi_i*, *twr_i* * *im2_i*. Во все последующие состояния конечный автомат переходит на каждом периоде тактового сигнала. В состоянии MULT_P1 не выполняется никаких математических действий.

Согласно рисунку 6, для получения конечного результата умножения комплексных чисел необходимо, чтобы продукты умножения компонентов *mult2* и *mult4* прошли стадию пост-суммирования. Для этого требуется дополнительный период тактового сигнала. Пост-суммирование результата умножения для *mult2* и *mult4* выполняется в состоянии MULT_P1 (стадия *propagate* в программе) в виде сложения с единицей, и результаты умножения остаются неизменными. Затем автомат переходит в стадию MULT_A1, где выполняются операции сложения и вычитания для получения конечного результата умножения комплексных чисел. После завершения стадии MULT_A1 на выходах умножителей 1 и 3 появляется результат умножения второго входного отсчёта и коэффициента поворота.

Следующим шагом является получение окончательных результатов операции «бабочка» или суммы IN1 + + IN2*W_N и разности IN1 – IN2*W_N. Эта операция выполняется в состоянии конечного автомата MULT_S1, при этом блоки DSP48A1 не задействованы. Затем конечный автомат переходит в состояние MULT_DONE, и на выходе модуля умножения появляются два 18-битных комплексных числа, т.е. результат выполнения основной операции БПФ – «бабочки».

Отладка и тестирование модуля БПФ

Отладку проекта можно разделить на три части – проверки правильности передачи данных внутри блоков, соответствия временным диаграммам и правильности выполнения математических операций. Предварительная отладка временных параметров производилась в программе ModelSim, затем в различные части системы на кристалле встраивались блоки ChipScope для проверки данных в реальном масштабе времени. Для использования «встроенного осциллографа» ChipScope необходимо в тестируемый модуль вставить два IP-модуля из библиотеки PlanAhead. В модуле контроллера *chipscope_icon* не требуется задавать никаких параметров – ширина шины обмена данными с контролируемым блоком принимается по умолчанию равной 36 бит. Во втором IP-модуле *chipscope_ila* задаётся необходимое число контролируемых сигналов, включая триггеры, от которых начинается отсчёт времени.

Например, необходимо посмотреть, какие входные и выходные сигналы

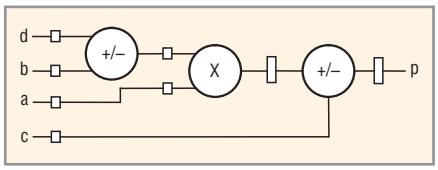

Рис. 5. Упрощённая схема блока DSP48A1

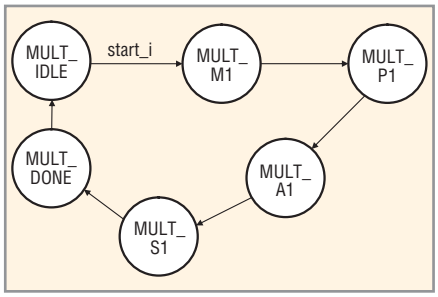

Рис. 6. Блок-схема умножителя комплексных чисел

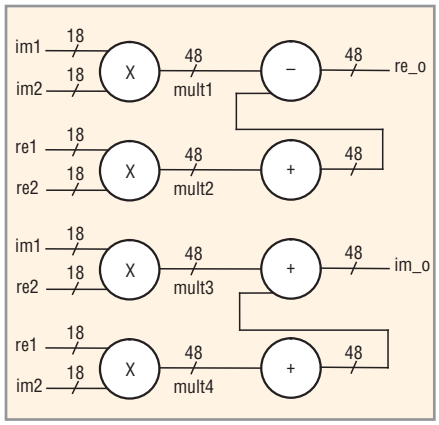

Рис. 7. Блок-схема конечного автомата модуля умножения

присутствуют в блоке умножения на стадии основного автомата БПФ FFT_STAGE1. Для этого генерируется блок *chipscope_ila*, который содержит все интересующие нас сигналы. Затем запускается программа ChipScope (входящая в состав среды PlanAhead), и триггером принимается состояние основного конечного автомата (*fft_fsm_state* в файле *fft_top.v*). Значение триггера устанавливается равным 4'b0001, соответствующим состоянию конечного автомата FFT_STAGE1. После перехода конечного автомата в состояние FFT_STAGE1 IP-модуль *chipscope_ila* начинает записывать все интересующие сигналы во внутреннюю память ПЛИС на каждом фронте синхросигнала. Затем сохранённые данные передаются управляющей программе на компьютер и отображаются в графическом или текстовом виде. Направление фронта задаётся при генерации IP-модуля; синхросигнал определяется на этапе компиляции проекта, в данном случае он совпадает с тактовой частотой модуля БПФ. Заметим, что для рабо-

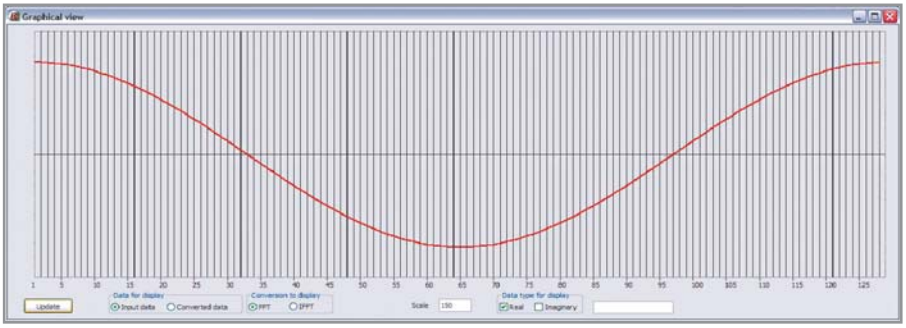

Рис. 8. Входной сигнал cos*X*

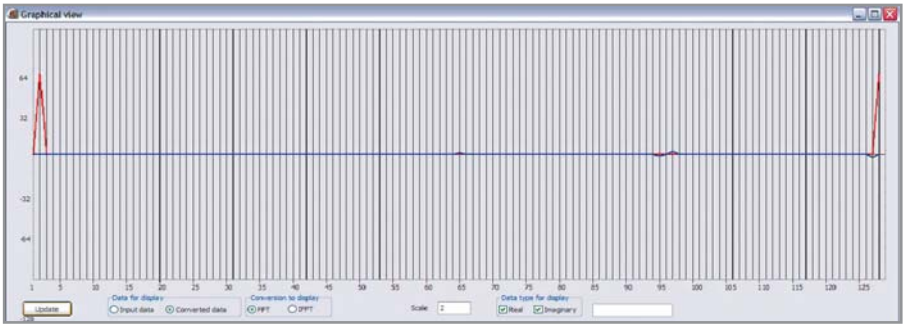

Рис. 9. Результат преобразования сигнала cos*X*

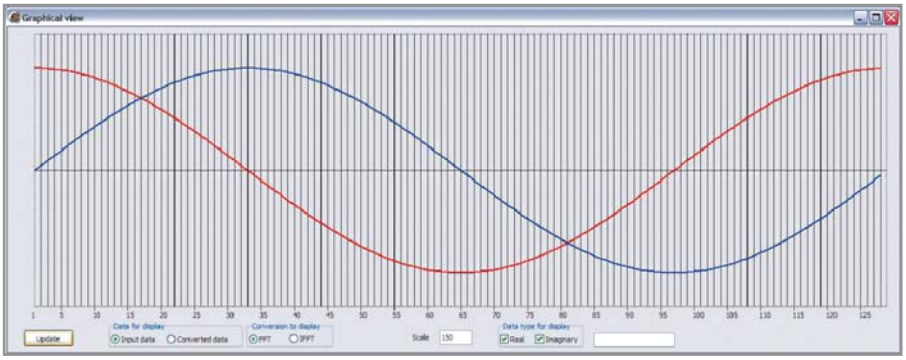

Рис. 10. Входной сигнал cos*X* + sin*X*

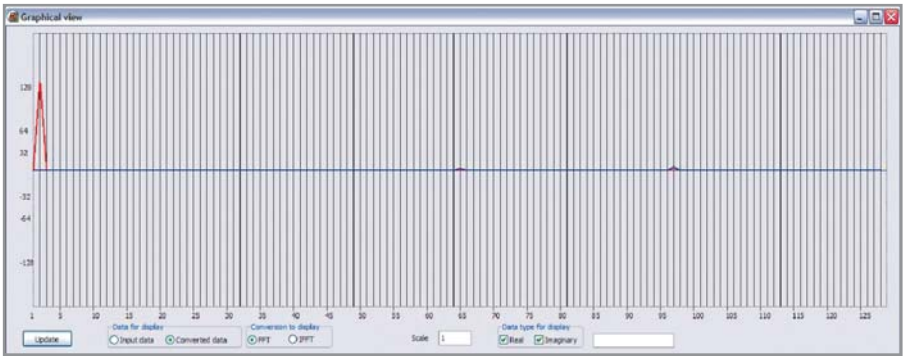

Рис. 11. Результат преобразования сигнала cos*X* + sin*X*

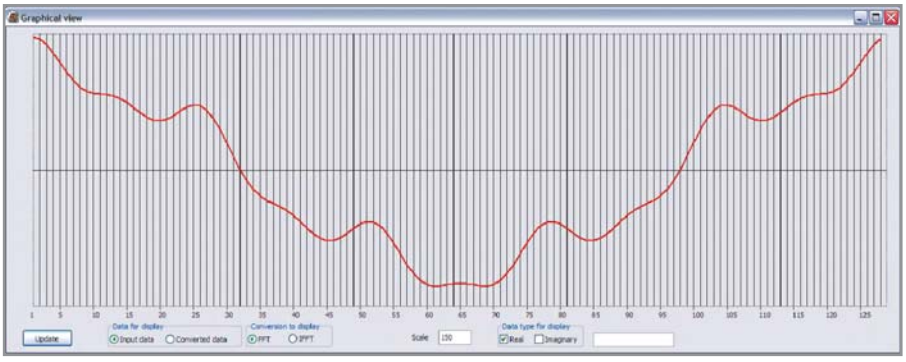

Рис. 12. Входной сигнал cos*X* + cos5*X* + cos10*X*

ты модуля ChipScope требуются дополнительные ресурсы ПЛИС, в частности ОЗУ. После отладки проекта логика chipscope может быть удалена из проекта для высвобождения ресурсов.

Для отладки выполнения математических операций необходимо на вход модуля БПФ подавать определённые тестовые векторы данных и затем анализировать результаты математических операций. В идеальном случае сразу проверяются выходные данные. Но, как правило, необходимо знать результаты вычислений на всех стадиях алгоритма БПФ, включая промежуточные. Для этого была написана программа на языке высокого уровня (Pascal), реализующая точно такой же алгоритм, что и ядро БПФ в ПЛИС. Реализация БПФ на языке высокого уровня значительно проще, так как можно сконцентрироваться на математической части реализации алгоритма. Эта программа (часть программы XilinxCOM) вычисляет БПФ для 128 точек и выдаёт промежуточные результаты вычислений для каждой стадии. Затем с помощью ChipScope в реальном времени считываются промежуточные результаты с ПЛИС и сравниваются с теоретическими значениями, полученными из программы XilinxCOM.

Тестовые векторы также генерируются программой XilinxCOM; 128-точечное БПФ может детектировать частоты от постоянной составляющей до 63 гармоники синуса и косинуса. Соответственно, входной сигнал должен содержать все возможные комбинации гармоник. Рассмотрим несколько примеров тестовых векторов и результатов преобразования БПФ.

Все входные тестовые векторы содержат действительную и мнимую части. Действительная часть представлена гармониками косинуса, а мнимая часть – гармониками синуса. Спектр сигнала также состоит из действительной и мнимой частей. Первый элемент массива характеризует постоянную составляющую, следующие 63 элемента – гармоники с 1 по 63. Вторая половина спектра – элементы массива с 65 по 128 – содержат элементы с отрицательными частотами.

На рисунке 8 изображён входной сигнал, состоящий из одной частоты cos*X*. Мнимая часть равна нулю, поэтому на рисунке не изображена. БПФ этого сигнала изображено на рисунке 9. Красным цветом выделена действительная часть преобразования, синим

цветом – мнимая. На втором канале действительной части наблюдается пик амплитудой 64 – это БПФ сигнала cos*X*. На 128 канале наблюдается зеркальное отражение того же пика. В мнимой части спектра наблюдаются некоторые шумы, связанные с округлением во время преобразования и формирования тестового вектора.

На рисунке 10 изображён входной сигнал, включающий в себя сигналы sin*X* и cos*X*. БПФ этого сигнала показано на рисунке 11. Амплитуда пика на втором канале в этом случае в два раза больше (128), а зеркальное отображение спектра отсутствует.

На рисунке 12 показан тестовый сигнал, состоящий из гармоник cos*X*, cos5*X* и cos10*X*. Амплитуды гармоник обратно пропорциональны частоте. Если амплитуду первой гармоники принять за единицу, амплитуда пятой гармоники будет в пять раз меньше, а амплитуда десятой гармоники – в 10 раз меньше. На рисунке 13 изображено БПФ этого сигнала. Здесь также наблюдается зеркальное отражение спектра, поскольку входной сигнал состоит только из гармоник косинуса. Соотношение амплитуд пиков первой, пятой и десятой гармоник соответствует соотношению амплитуд входного сигнала.

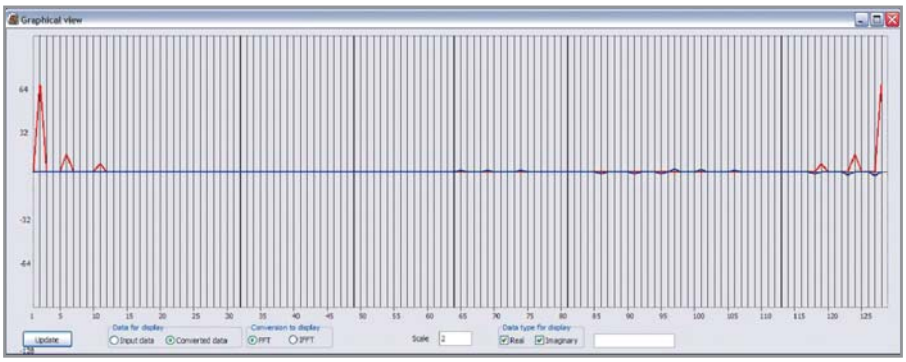

Рис. 13. Результат преобразования сигнала cos*X* + cos5*X* + cos10*X*

Заключение

В настоящее время существует большое разнообразие алгоритмов БПФ, отличающихся быстродействием, размерностью, используемыми вычислительными ресурсами и другими параметрами. Важной характеристикой реализации алгоритма БПФ является арифметика, используемая для вычислений (плавающая или фиксированная точка). Реализация алгоритма с плавающей точкой обладает высокой точностью, но является сложной и затратной по вычислительным ресурсам. В приложениях, не требующих высокой точности, более простая реализация алгоритма БПФ с фиксированной точкой вполне оправданна, так как она требует меньше ресурсов, что актуально для ПЛИС. Модуль БПФ, описанный в статье, показал удовлетворительные результаты при тестировании и пригоден для приложений, не требующих высокой точности.

Литература

1. *Рабинер Л., Гоулд Б.* Теория и применение цифровой обработки сигналов. Мир, 1978.
2. *Блейхут Р.* Быстрые алгоритмы цифровой обработки сигналов. Мир, 1989.
3. Spartan-6 FPGA DSP48A1 Slice User Guide. UG389(v1.1). August 13, 2009.