Argc & argv. –Программирование / информационные технологии / стандарты. 2005. -№ 2 . – С.12-17.

Рациональные рациональные дроби

А.М.Сергиенко

С времени своего появления первые компьютеры предоставляли возможность вычислений различной сложности. Вначале в компьютеры была заложена арифметика с фиксированной запятой, которая до сих пор интенсивно используется для вычисления адресов и подсчета циклов. Почти сразу же за этим математики разработали представление данных с плавающей запятой, как обязательное для их расчетов. Арифметика с плавающей запятой первые два – три десятилетия исполнялась программно, а затем – аппаратно. Благодаря принятию в 1985 г. IEEE-стандарта представления данных с плавающей запятой, прекратился разнобой реализаций арифметики вещественных чисел, и одна и та же программа в разных компьютерах стала давать одинаковые или похожие результаты. С тех пор представление данных в виде целых чисел и с плавающей запятой – это две практически единственные альтернативы, предоставляемые программистам архитектурами компьютеров.

Существует большое множество приложений, где возможностей целочисленной арифметики маловато, а вычисления со стандартной плавающей запятой – слишком избыточны или дороги, например, в компьютерной графике, цифровой обработке сигналов. С другой стороны, возрастает число задач, в которых требуются более точные вычисления, чем те, которые дает стандартная плавающая запятая. Например, при реализации новых средств защиты данных и их проверке нужны безошибочные вычисления. Для разрешения этого противоречия можно применить арифметику рациональных дробей, на которую сейчас не обращают внимания. В данной статье рассматриваются некоторые практические аспекты применения такой арифметики.

Немного теории дробей

Рациональная дробь – это числовой объект, состоящий из целочисленных числителя и знаменателя. Её название говорит о том, что этой дробью представляется любое рациональное число. Рациональные числа - это числа, которые возникают как результаты решения линейных уравнений или систем из них. Также эти числа получаются при рациональных вычислениях цепных дробей, целочисленных полиномов и в результате деления таких полиномов друг на друга. Другими нецелыми числами являются действительные решения квадратных, кубических и других степенных уравнений, называемые иррациональными числами, а также трансцендентные числа, не относящиеся ни к рациональным, ни к иррациональным.

Рациональная дробь a/b обладает тем свойством, что она может с достаточной точностью приближаться к заданному иррациональному или трансцендентному числу x. Причем, если дробь a/b меньше x, а дробь c/d – больше x, то дробь $(a+c)/(b+d)$, называемая медиантой, стоит значительно ближе к x чем исходные дроби. Таким образом, последовательно строя медианты, можно всегда сколь угодно близко приблизиться к числу x.

Если x – это какой-либо квадратный корень, приблизительно представленный $2n$ знаками в некоторой системе счисления, то числа a и b в его представлении a/b с такой же точностью будут приблизительно не более чем n-значными. Та же закономерность приближенно касается и других иррациональных чисел, а для представления трансцендентных чисел достаточно и меньшее число знаков [1].

Еще 80 лет назад, т.е. за два десятилетия до начала эры вычислительной техники, А.Я. Хинчиным были показаны преимущества цепных дробей для представления чисел [1]. Те же доводы касаются и рациональных дробей, к которым сводятся конечные цепные дроби. Во-

первых, всякая десятичная или двоичная дробь связана с соответствующей системой счисления и поэтому отражает не свойства самого числа, а его соотношение с данной системой счисления.

Действительно, число с плавающей запятой представляет собой рациональное число, в знаменателе которого стоит некоторая степень основания счисления. Поэтому каждый программист знает, что вещественное дробное число на бумаге или на дисплее не равно такому же числу после компиляции. А всё потому, что это две аппроксимации одного числа, причем в первом случае в знаменателе стоит степень десяти, а во втором случае – степень двойки. При этом часто возникают ошибки в программах, например, когда (1/9)*9 не всегда равно 1, так как $1/9 \approx 1110001110001_2$ E-16 = 0.00111000111001_2=7281/65536. Разность числа 1/9 и его представления 16-значной двоичной дробью равна 7/589824 ≈ 0.0000187. После умножения такой дроби на 9 получим 0.9998932, а не 1. В противовес плавающей запятой, рациональные дроби ни с какой системой счисления не связаны и более точно воспроизводят свойства изображаемых ими чисел.

Во-вторых, рациональные дроби достаточно просто позволяют находить приближенные значения числа с наперед заданной точностью, причем рациональная дробь получает наилучшее приближение [1]. Так, еще с древности известны приближения числа пи, равные 22/7 и 355/113, которые соответствуют десятичным дробям с 4 и с 7 знаками, соответственно. Причем опять-таки, такое приближение не зависит от системы счисления. Многие элементарные функции эффективно вычисляются по соответствующим формулам рационального приближения. Например, с точностью до 8 десятичных цифр тангенс вычисляется по формуле: $\mathrm{tg}\ x = (945x-105x^3+x^5)/(945-420x^2+15x^4)$. Если найден тангенс, то синус можно вычислить по формуле: $\sin 2x=2\mathrm{tg}\ x/(1+\mathrm{tg}^2 x)$.

И в-третьих, рациональные дроби предоставляют довольно простой аппарат арифметических действий. Так, умножение a/b на c/d равно $ac/(bd)$, а сложение – $(ad+bc)/(bd)$. При этом собственно деление целых чисел не выполняется. Для сравнения чисел достаточно вычислить $ad-bc$. При сопоставлении сложности операций следует учесть, что разрядности числителей и знаменателей по крайней мере вдвое меньше, чем у операций с целыми числами при той же точности представления. Поэтому сложность аппаратных сумматоров дробей приближается к сложности умножителей целых чисел, а сложность умножителей дробей оказывается вдвое меньшей, чем умножителей целых чисел.

Единственный недостаток вычислений с рациональными дробями – это быстрый рост разрядности числителя и знаменателя при выполнении серии последовательных действий. Поэтому для компенсации этого явления в дробях приходится периодически искать наибольшее общее кратное и делить на него числитель и знаменатель. А это трудоемкий процесс. Хотя без него можно обойтись, округляя результаты, как это будет показано ниже.

Рациональные дроби в компьютерах

С появлением компьютеров постоянно велась борьба за экономию памяти в них. И сразу же с внедрением плавающей запятой предлагалось представление чисел рациональными дробями. Так, для представления числа 1/9 с максимальной точностью достаточно два числа в любом формате целого, но необходимо число с плавающей запятой в формате double. Выше показан пример эффективности представления числа пи. Но разрастание числа необходимых цифр в дробях при выполнении операций с ними заставляло отказываться от такого представления [2].

Применение алгоритмов сокращения числителя и знаменателя замедляет процесс роста разрядности. Одним из таких алгоритмов является алгоритм Эвклида для поиска наибольшего общего делителя. Но программное применение этих алгоритмов может замедлить вычисления с дробями в тысячи и сотни тысяч раз.

В 70-е годы прошлого столетия основной операцией, выполняемой аппаратно, было суммирование. Поэтому поиск новых алгоритмов был направлен, в основном, на

минимизацию числа умножений и делений. Также искались алгоритмы, позволяющие отказаться от использования плавающей запятой. В это время вспомнили о рациональных дробях и стали предлагаться процессоры с аппаратно реализованной арифметикой дробей. При этом в проектах таких процессоров алгоритм Эвклида был реализован аппаратно-микропрограммно [3]. Но после появления микросхем аппаратных умножителей и сопроцессоров с плавающей запятой процессоры с арифметикой дробей были забыты.

С наступлением эры персональных компьютеров рациональные дроби стали широко применяться в математических САПРах. Так, в пакете MathCAD можно получать результаты вычислений в виде приближений рациональными дробями. При этом дроби аппроксимируют результаты с заданной разрядностью – вплоть до 15 десятичных цифр в исходном вещественном числе. Однако сами вычисления выполняются с плавающей запятой. В пакете Maple по требованию программиста все вычисления могут выполняться точно с применением арифметики дробей и с представлением данных произвольной разрядности.

Для решения многих математических проблем нужны точные арифметические расчеты. Так, решение систем уравнений, особенно с разреженной матрицей, наиболее эффективно и быстро выполняется по методу сопряженных градиентов или методу Ланцоша. Но единственное неудобство этого метода состоит в том, что он работает устойчиво, если нет округлений промежуточных результатов, т.е. тогда, когда все вычисления выполняются точно [4]. Поэтому арифметика дробей – это то, что нужно для реализации этого метода.

Появление и развитие методов криптографии, особенно с открытыми ключами, заставило внедрять в программирование арифметику чисел произвольной точности. Вероятно, поэтому компиляторы таких языков, как Java и PERL, поставляются с библиотеками функций для вычислений с произвольной точностью. Рациональные числа при криптографии применяются косвенно, например, при целочисленном логарифмировании и факторизации многочленов [5].

При программной реализации арифметики большой точности программисты сталкиваются с тем, что процессоры к ней не приспособлены. Так, почти нет процессоров, которые давали бы в результате умножения 32-разрядных целых 64-разрядные результаты. Поэтому приходится сперва получать младшие разряды такого произведения, а затем "добывать" его старшие разряды или наоборот. Иногда такое вычисление получается эффективнее путем перевода чисел в формат extended double, выполнения действий в этом формате и обратного преобразования в целое.

Сейчас уже назрела необходимость добавить формат рациональных чисел к машинным форматам, таким как целое число и число с плавающей запятой. Так, в [6] предлагается ввести новый стандарт представления чисел, так называемых композитных чисел, который кроме целых чисел и чисел с плавающей запятой включает рациональные числа и логарифмические числа. При этом длина кодов чисел составляет 32, 64, 128, 256 и даже 4096 бит и в одном таком слове содержится и числитель, и знаменатель.

Рациональное вращение

Первыми специалистами, которые вплотную связали рациональные числа с вращением, были часовщики. Любая зубчатая передача – это умножение некоторого угла поворота на рациональное число, представляющее отношение числа зубьев колес. Много математических открытий было связано с решением задачи аппроксимации отношения периодов вращения светил и Земли некоторым рациональным числом, по которому изготовлялся зубчатый механизм хронометров.

Операция поворота вектора (x,y) часто используется в различных приложениях вычислительной техники. Она часто выполняется в робототехнике, вычислительной геометрии, обработке изображений, цифровой обработке сигналов. После поворота на угол θ получается вектор (x',y'), равный

$$x' = x \cdot \cos\theta - y \cdot \sin\theta; \qquad (*)$$

$$y' = x \cdot \sin\theta + y \cdot \cos\theta.$$

Если коэффициенты синусов и косинусов представлены неточно и/или операции выполняются с погрешностью, то возникают ошибки поворота, зачастую – недопустимые. Так, например, указанные формулы часто используются как процедура для генерации сигнала синусоидальной формы. При этом выходной повернутый вектор подается на вход процедуры. Если не соблюдается равенство Пифагора: $\sin^2\theta + \cos^2\theta = 1$, то погрешность длины вектора приведет к быстрому затуханию генерируемой синусоиды или к ее росту с последующим переполнением.

Синус считается трансцендентным числом. Но среди синусов часто попадаются и рациональные числа. Задача Пифагора заключается в нахождении всех прямоугольных треугольников с целочисленными сторонами, т.е. в получении решений диофантова уравнения:

$$x^2 + y^2 = z^2.$$

Из этого уравнения и равенства Пифагора следует, что для некоторых углов θ синусы и косинусы равны рациональным числам: $\sin\theta = x/z$; $\cos\theta = y/z$. Решения задачи Пифагора находятся из соотношений:

$$x = 2mn;\ y = m^2 - n^2;\ z = m^2 - n^2, \qquad (**)$$

где m и n – взаимно простые, одно из них четное, а другое - нечетное и $m > n$ [7].

Таким образом, подбирая m и n, можно найти точные значения синуса и косинуса для некоторого угла, достаточно приближенного к заданному. Так, например, $\sin 60^\circ \approx 56/65$ с погрешностью угла 0,5˚ и 780/901 – с погрешностью 0.037˚.

В работе [8] предлагаются конструктивные алгоритмы поиска рациональной аппроксимации синусов. Там же доказываются преимущества рационального представления вращающих коэффициентов в роботизации и компьютерной графике. Это, например, позволяет обходиться целочисленной арифметикой при работе графических приложений, сохранять графические объекты в более компактной форме.

Операция вращения лежит в основе многих алгоритмов цифровой обработки сигналов. Например, дискретное преобразование Фурье основано на следующем. Пусть входной комплексный сигнал представляет собой синусоиду, т.е. вектор, поворачивающийся на угол θ в соответствии с уравнениями (*). Если векторы-отсчеты вращать в противоположном направлении с углом, кратным θ' и накапливать в аккумуляторе, то результат достигнет максимума, при условии, что θ=θ', а если θ≠θ', то результат стремится к нулю. Недостаток этого метода состоит в большом количестве вычислений, если нужно знать результаты для всех θ', т.е. спектр сигнала.

Другая сложность состоит в том, что нужно хранить поворачивающие коэффициенты для всех кратных углов. В этом случае эти коэффициенты можно генерировать, используя точное значение синуса и косинуса угла θ' и вычисляя остальные значения вращающих коэффициентов по формулам (*), как это будет показано ниже.

Известный алгоритм быстрого преобразования Фурье (БПФ) основан на том, что векторы входного сигнала проходят несколько ступеней вращения и суммирования частных результатов, точнее – выполнения малоточечного преобразования Фурье. При этом в алгоритме по основанию 2 на первой ступени выполняется сложение- вычитание векторов, повернутых на угол 0 и 180º, на второй – на 0 и 90º, на третьей – на 0 и 45º и т.д.

В 1979 г. Алвин Деспейн предложил в алгоритме БПФ на первой ступени вращать на угол ±90º , на второй – на ±45º, на третьей – на ±22,5º и т.д. При этом все вращающие коэффициенты представлены в виде рациональных дробей, причем деление на общий знаменатель дробей нигде не выполняется. Так, поворот на ±45º можно выполнять вообще без умножений, а поворот на ±22,5º - путем умножения на 5 и 11. Операция умножения на коэффициент выполнялась как суммирование сдвинутых операндов, за счет чего получались минимальные аппаратные затраты и максимальное быстродействие [9].

Выходные отсчеты такого смещенного БПФ отличаются от результатов истинного БПФ тем, что они все повернуты на некоторый угол смещения и умножены на рациональный масштабный коэффициент. Такое искажение результатов не имеет значения в большинстве приложений БПФ. Зато БПФ – процессор Деспейна давал рекордно высокое быстродействие при удивительно малых аппаратурных затратах.

С появлением аппаратных умножителей о процессоре Деспейна забыли. Но о нем вспомнили сейчас в связи со следующим. Многочастотная широкополосная модуляция, применяемая, например, в системах с WLAN с модуляцией OFDM, позволяет выполнять передачу цифровых сигналов со скоростью более 10 мбит/с по радио на большие расстояния или даже по сетям высокого напряжения. Она стала возможной с появлением процессоров БПФ, работающих с большой частотой. Так вот, процессор Деспейна с его минимумом аппаратуры и соответственно, энергопотребления – это оптимальное решение для портативных средств такой связи.

Рациональные дроби в цифровой обработке сигналов

Алгоритмы и средства цифровой обработки сигналов постоянно двигаются в сторону своего усложнения и увеличения точности. Два десятилетия назад типичный DSP – процессор обрабатывал данные с 8-10 – разрядного АЦП с производительностью 5-10 млн. операций в секунду по алгоритмам фильтрации или БПФ. Теперь разрядность АЦП возросла до 16 – 24, производительность DSP – процессоров достигает сотен и тысяч млн. оп/с и алгоритмы обработки описываются сотнями тысяч строк языка Си. При этом среди алгоритмов обработки стали часто встречаться такие алгоритмы, которые требуют повышенной точности вычислений, как, например, решение систем уравнений, нахождение собственных значений, вычисление геодезических координат.

Многие задачи цифровой обработки сигналов успешно решаются на персональном компьютере, обладающем MMX-расширением команд и командами плавающей запятой. Но большую массу цифровой обработки сигналов выполняют DSP – процессоры. Хотя среди DSP – процессоров много процессоров с плавающей запятой, те используются сравнительно мало из-за своей дороговизны. "Рабочей лошадкой" цифровой обработки сигналов трудится 16- разрядный DSP-микропроцессор с фиксированной запятой.

Приходится идти на ухищрения, чтобы выжать из 16 – разрядного микропроцессора максимум точности. Прежде всего, экономно используется разрядная сетка. Так, в современных DSP-программах почти все промежуточные массивы данных нормализуются таким образом, чтобы в максимальном числе в массиве не осталось старших незначащих цифр. Иногда массив делят на максимальное число в нем. Во всех остальных случаях вычисляют с удвоенной разрядностью, когда старшие и младшие полуслова обрабатываются последовательно. В последнее время все больше появляется 32- разрядных DSP–процессоров. Но и такая разрядность не удовлетворяет разработчиков, например, при высококачественной обработке аудиосигналов и при решении систем уравнений.

Применение арифметики рациональных дробей позволяет не только повысить точность вычислений при цифровой обработке сигналов, но и ускорить их. Выше было показано значение рациональных дробей при БПФ. Рассмотрим еще одно применение дробей в современных DSP – процессорах. В каждом мобильном телефоне DSP – процессор по 50 раз в секунду решает систему уравнений. Аналогичную работу выполняют процессоры для адаптивной фильтрации, активного шумоподавления, интелектуальные датчики. При решении указанной задачи многократно используется операция деления. Эта операция, как правило, не реализована аппаратно и поэтому она реализуется как многотактовая подпрограмма. Также она является основным источником ошибок.

При решении системы уравнений все вычисления на первых N-1 итерациях можно выполнять с данными в виде рациональных дробей. В результате операций деления дробей собственно деление целых чисел не выполняется. Но в конце последней итерации

необходимо числители дробей поделить на их знаменатели обычным делением, чтоб получить результат в обычном представлении. Такие вычисления обеспечивают как небольшие ошибки вычислений, так и расширенный динамический диапазон представления данных в сравнении с арифметикой целых чисел, а также возможность отказаться от арифметики с плавающей запятой.

Ниже будет показано, как организовать вычисления с рациональными дробями. В работе [10] показано, что решение системы теплицевых уравнений на 16- разрядном DSP-микропроцессоре с применением арифметики дробей средняя ошибка результатов для серии случайных данных составляет около 1%. Для реальных, более обусловленных данных эта ошибка существенно меньше.

Рациональные дроби и VHDL

После всего сказанного неудивительно, что хочется проверить на практике эффективность рациональных дробей. Язык VHDL предоставляет широчайшие возможности для этого. Во-первых, в нем можно задавать векторами битов целые числа произвольной длины, алгоритм их обработки естественным образом программируется с точностью до бита. Во-вторых, описанные в нем арифметические функции перекрываются с операциями языка. Поэтому программирование выражений с рациональными числами не отличается от программирования с целыми. В-третьих, абстрактный алгоритм обработки дробей может быть транслирован во вполне конкретный спецвычислитель, реализованный, например, в ПЛИС.

Для работы с дробями был разработан следующий пакет.

```
library IEEE;
use IEEE.Numeric_bit.all;
package FRACT_lib is
 constant nn:positive:=16;                          --разрядность дробей
 type FRACTV is record N : signed(nn-1 downto 0);-- тип дроби
      D : signed(nn-1 downto 0);
 end record;
 constant PLUS1:signed(nn-1 downto 0):=(nn-1=>'0',others=>'1');
 constant NIL:FRACTV:=((others=>'0'),PLUS1);    -- число 0
 constant ONE: FRACTV:=(PLUS1,PLUS1);           -- число +1
 function "-"(x:FRACTV) return FRACTV ;         --  -  минус
 function "+"(x,y:FRACTV) return FRACTV;        --  +  сложение
 function "-"(x,y:FRACTV) return FRACTV;        --  -  вычитание
 function "*"(x,y:FRACTV) return FRACTV;        --  *  умножение
 function "/"(x,y:FRACTV) return FRACTV;        --  /    деление
 function FRACT_REAL(x:FRACTV) return real;     -- перевод в real
end package;
package body FRACT_lib is
 function"-"(x:FRACTV) return FRACTV is
      variable n,d:bit_vector(nn-1 downto 0);
 begin
      (n,d):=x;
      return (bit_vector(-signed(n)),d);
 end;
 function"-"(x:FRACTV) return FRACTV is
      variable n,d:signed(nn-1 downto 0);
 begin
      (n,d):=x;
      return (-n,d);
 end;
 function"+"(x,y:FRACTV) return FRACTV is
      variable n1,d1,n2,d2:signed(nn-1 downto 0);
      variable n3,d3:signed(2*nn-1 downto 0);
      variable n3s,d3s:signed(2*nn-1 downto nn-1);
 begin
```

```
        (n1,d1):=x;
        (n2,d2):=y;
        n3:=n1*d2+n2*d1;                -- вычисление числителя
        d3:=d1*d2;                      -- вычисление знаменателя
        if (n3(2*nn-1)=n3(2*nn-2))then -- некоторая нормализация
             n3:=n3(2*nn-2 downto 0)&'0'; --числителя
             d3:=d3(2*nn-2 downto 0)&'0';
        end if;
        if  (n3(2*nn-1)=n3(2*nn-2))and (d3(2*nn-1)=d3(2*nn-2))then
             n3:=n3(2*nn-2 downto 0)&'0'; -- нормализация числителя
             d3:=d3(2*nn-2 downto 0)&'0'; -- и знаменателя
        end if;
        n3s:=n3(2*nn-1 downto nn-1);
        n3s:=n3s+1;                             --округление числителя
        d3s:=d3(2*nn-1 downto nn-1);
        d3s:=d3s+1;                             --округление знаменателя
        return (n3s(2*nn-1 downto nn),d3s(2*nn-1 downto nn));
 end;
function FRACT_REAL(x:FRACTV) return real is
        variable d:signed(nn-1 downto 0);
        variable t:bit;
 begin
        t:='0';
        d:= x.D ;
        for i in 0 to nn-1 loop
             t:=t or d(i);
        end loop;
        if t='0' then d(0):='1'; end if ;
   return real(to_integer(signed(x.N)))/real(to_integer(signed(d)));
 end;
end package body;
```

В пакете дробь представлена типом FRACTV, который представляет собой запись, состоящую из числителя N (numerator) и знаменателя D (denominator). Числитель и знаменатель представлены nn-разрядными целыми числами со знаком в дополнительном коде, обозначенными подтипом signed типа bit_vector. Это упрощает их реализацию в аппаратуре или подпрограмме. Дроби в нормальном состоянии должны быть предпочтительно нормализованными, т.е. или числитель, или знаменатель не должны иметь старших незначащих цифр. Такое представление отличается от представления классических рациональных дробей и далеко не всегда обеспечивает точные вычисления. Зато оно освобождает от слежения за переполнением разрядной сетки и по своим качествам приближает эти дроби к представлению с плавающей запятой.

Константы 0 и 1 представлены кодами NIL =0000/0111 и ONE =0111/0111, соответственно (для nn=4). Числитель и знаменатель типа FRACTV представляют собой число с запятой, фиксированной после знакового разряда. Поэтому действия их умножения и сложения выполняются по правилам, соответствующих действиям с двоичными дробями с фиксированной запятой. Это значит, что умножение двух таких нормализованных чисел даст нормализованный 2nn-1 – разрядный результат, или результат, требующий одного сдвига для нормализации. Результирующие числитель и знаменатель получаются путем округления и отбрасывания n младших разрядов произведений и их сумм.

Операции сложения, вычитания, умножения и деления выполняются по формулам, представленным в первом разделе статьи. Отличие в том, что после вычислений числитель и знаменатель нормализуются сдвигом влево на 1 или 2 разряда. Потребность в нормализации на большее число разрядов возникает гораздо реже. Но ненормализованные дроби обрабатываются аналогично нормализованным и по ходу алгоритма чаще всего они становятся нормализованными. Для округления числителя и знаменателя используется прибавление 1 в старший отбрасываемый разряд.

В тексте тела пакета опущены описания функций вычитания, деления, умножения, т.к. они похожи на приведенное описание функции сложения. Функция FRACT_REAL предназначена для перевода дроби в действительное число.

Рассмотрим применение рациональных дробей для генерации синусоиды и косинусоиды. Вычисление синуса и косинуса будет вестись по формулам (*). Считаем, что период синусоиды должен быть равным NN отсчетов, NN>16, т.е угол $\theta = 2\pi/NN$. Вращающие коэффициенты $\sin\theta$ и $\cos\theta$ будем вычислять, решая задачу Пифагора с применением решения (**). Ниже приведена VHDL-программа, выполняющая поиск дробей, аппроксимирующих эти коэффициенты.

```
library IEEE;
use IEEE.MATH_REAL.all;
entity sin_cos_find is
end sin_cos_find;
architecture sin_cos_find of sin_cos_find is
 constant Nc:real:=256.0;          --  ищем sin для угла 2Пи/Nc
 constant eps:real:=0.001;         -- ошибка угла
 signal clk:bit;
 signal m,n,num:integer:=1;
 signal x,y,z:integer;             -- результаты  sin=x/z; cos=y/z.
 signal sin,fi:real;
begin
 clk<=not clk after 10 ns;         -- генератор синхросерии
 process(clk)
      variable tx,ty,tz:integer;
      variable fii:real;           -- текущий угол
 begin
      if clk'event then
           n<=n+1;                  -- изменение 1-го параметра
           if n=30 then
                n<=1;
                m<=m+1;             --изменение 2-го параметра
           end if;
           if ((m mod 2 =1) and (n mod 2 = 0)) or
                ((m mod 2 =0) and (n mod 2 = 1)) then
                tx:=2*m*n;
                ty:=m*m-n*n;
                tz:=m*m+n*n;
                fii:= Nc*arctan(real(tx)/real(ty))/MATH_2_PI;
                if ABS(fii-1.0)<eps then
                     x<=tx;             --  нашли результат
                     y<=ty;             --  sin=x/z; cos=y/z
                     z<=tz;
                     fi<=fii;          -- для угла fi*2Пи/Nc
                end if;
           end if;
      end if;
 end process;
end sin_cos_find;
```

Поиск оптимальных числителей x,y и знаменателя z выполняется путем перебора параметров m и n. Так как угол θ – небольшой, то параметр n пробегает значения от 1 до 30, а параметр m пробегает значения от 1 до нескольких тысяч. По каждому фронту сигнала clk проверяется новая комбинация параметров m и n, так что за 30000 тактов моделирования будут найдены все варианты аппроксимирующих дробей, представляемых 20-разрядными двоичными числами. Для угла $\theta = 2\pi/256$ были найдены $\sin\theta$ =652/26573 и $\cos\theta$ = 26565/26573.

Экспериментальные синусоида и косинусоида генерируются следующей программой.

```
library IEEE;
use IEEE.NUMERIC_BIT.all;
```

```
use FRACT_lib.all;
entity singen is
 port(SINE,COSIN:out real);
end singen;
architecture FRACT of singen is
-- вращающие коэффициенты, nn -разрядность числителя и знаменателя
-- Коэффициенты для периода 256
 constant sn:FRACTV:=(TO_SIGNED(652,nn),TO_SIGNED(26573,nn));
 constant cn:FRACTV:=(TO_SIGNED(26565,nn),TO_SIGNED(26573,nn));
 signal CLK:bit;
 signal n:integer:=0;
 signal si:FRACTV:=NIL;
 signal co:FRACTV:=ONE;
begin
 CLK<=not CLK after 10 ns;
 process(clk) begin
      if clk'event then
           si<=si*cn+co*sn; -- вращение вектора (co,si)
           co<=co*cn-si*sn;
           n<=n+1;          -- счетчик тактов для контроля
      end if;
 end process;
 SINE<=FRACT_REAL(si);
 COSIN<=FRACT_REAL(co);
end FRACT;
```

При запуске программы вектор (co,si) принимает значение (ONE, NIL), т.е. (1.0, 0.0). Затем в каждом такте длительностью 10 нс выполняется вращение этого вектора путем вычисления формул (*) с найденными коэффициентами cn, sn. Функция FRACT_REAL переводит полученные значения вектора (co,si) в действительные значения косинуса и синуса, которые выдаются на выходы COSIN и SINE, соответственно.

Измерения выходной синусоиды показали, что синусоида генерируется устойчиво с периодом 256,054 такта. Синусоида имеет начальную амплитуду 0,998 и смещение относительно нуля –0,0017. Затухание синусоиды через 1000 периодов, т.е. через 256 тыс. итераций вращения составило 1,029 раза, т.е. ошибки округления вычислений недостаточно, чтобы накопиться до существенной величины за столько итераций. Для сравнения, такой же генератор, работающий с целыми 32-разрядными числами и с оптимизированными вращающими коэффициентами выдавал синусоиду с периодом 256,1 тактов, которая затухла до нуля через 47 тыс. итераций.

Арифметика дробей была использована в экспериментальной параллельной ВС для решения задач с теплицевыми матрицами. За основу процессорного элемента ВС было взято ядро 16-разрядного RISC-микропроцессора, описанного в [11]. Этот процессорный элемент состоит из базового ядра и аппаратного ускорителя. Аппаратный ускоритель реализует расширение команд с арифметикой дробей и других операций, которые часто используются при цифровой обработке сигналов. Поэтому операции с дробями программируются в процессорном элементе на ассемблере, как в обычном RISC-процессоре программируются вычисления с целыми. При этом выполнение умножения и деления дробей длится один такт, а их сложение – два такта. Перед записью в память числитель и знаменатель результата нормализуются путем сдвига на 0, 1 или 2 разряда влево.

В одну микросхему ПЛИС Xilinx Virtex2 объемом 1 млн. вентилей помещаются 10 таких ПЭ, работающих с тактовой частотой 80 МГц. При решении задач с теплицевыми матрицами такая ВС имеет среднюю производительность около 170 млн. операций в сек., которые эквивалентны операциям с плавающей запятой [10]. Если бы такие задачи решались с фиксированной или плавающей запятой, то и аппаратные затраты, и время вычислений увеличилось бы в несколько раз.

Заключение

Вычислительная техника предназначена прежде всего для расчетов. Каждому программисту предоставляется в распоряжение только арифметика целых чисел или арифметика с плавающей запятой. Но арифметика целых чисел способна эффективно обрабатывать данные с разрядностью не более чем 16. Вычисления с плавающей запятой годятся для данных с разрядностью более 24. Но далеко не все микропроцессоры обладают аппаратной реализацией таких вычислений.

Рационально применять арифметику рациональных дробей, когда в микропроцессор не встроена система команд с плавающей запятой и требуются вычисления с повышенной точностью. Например, арифметика дробей дает возможность решать многие практические задачи линейной алгебры с высокой производительностью в дешевых микропроцессорах с фиксированной запятой. Её применение в ПЛИС обеспечивает высокое отношение производительность - аппаратные затраты, благодаря высокой степени распараллеливания и эффективному использованию аппаратных блоков умножения. Также в ПЛИС можно обеспечить необходимую точность вычислений путем выбора оптимальной разрядности рациональных дробей.

Литература

1. Хинчин А.Я. Цепные дроби. –М.:Наука.-1978.-112с.
2. Кнут Д.Е. Искусство программирования. Т.1, Т2.
3. Irvin M.J., Smith D.R. A rational arithmetic processor // Proc.5th Symp. Comput. Arithmetic. –1981. –p.241-244.
4. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы. –М.: Наука. – 1987. –598с.
5. Василенко О.Н. Теоретико-числовые алгоритмы в криптографии. –М.:Изд-во МЦНМО. –2003.–325с.
6. Holmes W.N. Composite Arithmetic: Proposal for a new Standard // Computer, -1997. -№3. – p.65-72.
7. Оре О. Приглашение в теорию чисел. –М.: Наука. –Биб-ка Квант, вып.3. – 1980. – 126с.
8. Canny J., Donald B., Resslert E.K. A Rational Rotation Method for Robust Geometric Algorithms.// 8th Annual Computational Geometry, Berlin.-1992.-p.251-260.
9. Despain A.M. Very Fast Fourier Transform Algorithms Hardware for Implementation. //IEEE Trans. Computers. -V28. –1979. -№5.-p.333-341.
10. Клименко А.Н., Сергиенко А.М., Шевченко Ю.В., Овраменко С.Г. Конфигурируемая вычислительная система для решения задач линейной алгебры // Электрон. Моделирование. –2005. – Т27. -№1.-с.109-114.
11. Сергиенко А.М. VHDL для проектирования вычислительных устройств. –Киев: DiaSoft. –2003. – 208 с.