СОДЕРЖАНИЕ.

РЕАЛИЗАЦИЯ УСТРОЙСТВА УМНОЖЕНИЯ С НАКОПЛЕНИЕМ.

ВВЕДЕНИЕ.

Микропроцессорные устройства с плавающей точкой (FPU), поддерживающие стандарт IEEE-754 для простого и двойного форматов двоичных вычислений с ПТ, могут быть разработаны с малыми затратами, малой мощностью и высокой производительностью. Некоторые микропроцессоры, предназначенные для маломощных вычислительных целей, выполняют операции умножения и сложения как одну команду:

$$\mathbf{T = (A*C) + B.}$$

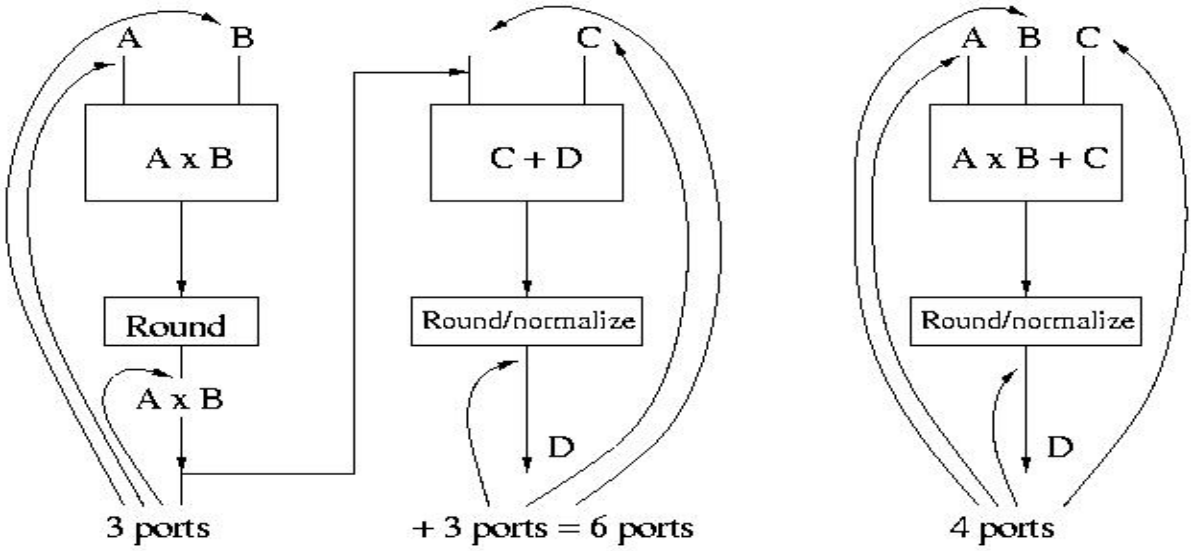
В таких случаях устройства для выполнения операций с ПТ разрабатывается целенаправленно для трех операндов для выполнения команд умножения со сложением, в то время как другие команды с ПТ требуют меньше трех операндов, они могут быть выполнены посредством той же аппаратуры установлением вместо других операндов констант. Например, чтобы выполнить команду сложения с ПТ T = A + B, нужно установить операнд C равным константе 1.0.

В основном, в FPU с использованием умножения с накоплением иногда берется массив умножения для расчета произведения AC, выравнивающий сдвигатель для выравнивания мантиссы операнда B с результатом умножения AC, основанный на разрядности операндов A, B и C, затем стоит сумматор для расчета AC + B. Промежуточный результат нормализуется и округляется для расчета окончательного результата. Умножение операндов A и C простого формата (24разряда * 24разряда) даст 48-разрядную величину. 74-разрядный сдвигатель вправо используется для выравнивания мантиссы операнда B с 48-разрядным результатом умножения AC. Далее последует сложение в 74-разрядный сумматор для расчета суммы AC плюс B.

Цена или площадь, занимаемая такой разработкой, может быть несоизмеримо высокой и может быть уменьшена сокращением размеров массива множителей.

MAF FPU.

Впервые такая структура была предложена еще в 1990 году в разработке IBM RISC System/6000. Ключевая особенность структуры такого FPU заключается в едином выполнении операции умножения и сложения. Это простое функциональное устройство уменьшает длительность для конвейерных операций. Также уменьшает число сумматоров/нормализаторов комбинированием суммирования, требуемого для быстрого умножения с накоплением. MAF практичен благодаря быстрому сдвигателю, который упрощает перекрывание умножения и сложения и блок предварительного обнаружения начального нуля/единицы, который упрощает перекрывание нормализации и сложения.



Работу MAF FPU можно разделить на пять независимых конвейеризованных тактов:

- Стадия умножения: включает в себя массив множителей для умножения 24-разрядной мантиссы A на 24-разрядную мантиссу C; и одновременно с этим выравнивающий сдвиг мантиссы B к промежуточному результату AC. Занимает 2 такта.
- Стадия сложения: производит сложение результата AC и выровненной

мантиссы операнда B; и находит первую значимую единицу в результате для его последующей нормализации.

- Стадия нормализации результата.
- Стадия округления для получения окончательного результата.

Как и во многих современных микропроцессорах, стадия умножения содержит алгоритм Бута и дерево Уоллеса с сумматорами с сохранением переноса для суммирования частичных произведений, полученных для умножения двух 24-разрядных мантисс.

Рисунок 1. Выполнение операции умножения с накоплением для вещественных чисел по тактам

ПРЕИМУЩЕСТВА MAF FPU.

В СБИС очень важен фактор связности как для цены, так и для производительности. Так как общее использование умножения с ПТ - для расчета скалярных произведений и простая операция сложения с накоплением будет иметь 6 выводов (3+3), в то время как MAF - только четыре. Это сокращение числа выводов согласуется с философией RISK-технологий производства максимально оптимизированных функций для решения наиболее часто требуемых функций. В случае MAF'a требуется лишь один сумматор и нормализатор, уничтожая таким образом, выход с умножителя и один вход у сумматора.

Вдобавок к уменьшению длительности и сокращению выводов, устранение нормализатора увеличивает точность, так как простое устройство принуждает архитектуру производит более эффективный код, улучшая команду умножения со сложением. Ценой за это уменьшение устройств и выводов будет добавочная задержка на сумматоре, ставшая причиной 50типроцентного увеличения площади сумматора. Этот факт можно считать более чем компенсацией в уплотнении этого простого конвейера.

РЕАЛИЗАЦИЯ ПОТОКА ДАННЫХ.

Рассмотрим здесь логический дизайн для отдельных частей потока данных: умножителя, сдвигателей, сумматора и блока предварительного определения старшей единицы или нуля.

Математическая реализация умножения вещественных чисел и операции умножения со сложением.

Существенное сокращение массива множителей при большой разрядности множителей получается при использовании алгоритма Бута. При этом число частичных произведений сокращается в два раза. Формируются частичные произведения множимого A на двухразрядные группы множителя C = C[23:0] в соответствии с выражением

$PP[i] = A \cdot (-C[i+1]*2 + C[i] + C[i-1]) \cdot 2^i$, где i = 0, 2, 4, ... , 22.

C[i-1]	C[i]	C[i+1]	PP[i]
0	0	0	+0*A
0	0	1	+1*A
0	1	0	+1*A
0	1	1	+2*A
1	0	0	-2*A
1	0	1	-1*A
1	1	0	-1*A
1	1	1	-0*A

Таблица 1. Значения частичных произведений алгоритма Бута.

Умножение на +2 это сдвиг влево на один разряд мантиссы A, Умножение на -1 и на -2 потребуют сдвига влево на один разряд (для -2), инверсии мантиссы операнда A и добавления единицы (для двоичного вычитания), а именно:

$$-1 \times A = \overline{A} + 1$$
$$-2 \times A = \overline{2 \times A} + 1$$

Добавление единицы требует дополнительного сумматора. Это увеличит время задержки. Эта проблема была решена конкатенацией двоичного '01' к младшим разрядам следующего за отрицательным частичного произведения (так как следующее частичное произведение будет расположено на 2 разряда левее). Последнее частичное произведение всегда положительно, что не требует последующего добавления единицы. Такая конфигурация достигается применением мультиплексоров 5:1 (для получения элементов 0, +1*A, +2*A, -1*A или -2*A), с помощью которых получим 0, A, смещенное на 1 разряд A, инверсию A или смещенное влево и

инвертированное A для формирования частичных произведений.

Из-за того, что генерируемые частичные произведения могут быть как положительными, так и отрицательными, суммирование частичных произведений даст знаковые расширения окончательному результату (48 разрядов для PP[0]).

Из-за того, что генерируемые частичные произведения могут быть как положительными, так и отрицательными, суммирование частичных произведений даст знаковые расширения окончательному результату (48 разрядов для PP[0]), Однако, высокопроизводительные множители. Уменьшенный формат расширения знака даст 28-разрядное частичное произведение 0 (pp0) и 27-разрядные от pp1 до pp11.

Частичные произведения могут быть сложены посредством дерева Уоллеса из CSA. Получим результат в двухрядном коде в виде переноса и суммы. Их сумма даст 48-разрядное произведение, как показано в рис. 3, где n0 - n10 представляют единицы Бута для pp0 - pp11, соответственно. Единица Бута не требуется для n11, потому что '0' привязан к старшему биту множителя, вынуждающего pp11 быть положительным.

Массив множителей может также включать добавление выровненной мантиссы B к сумме и переносу частичных произведений. Суммирование будет с помощью 3:2 CSA. Мантиссы B и AC могут быть выровнены (степени откорректированы согласно расчету сдвига) сдвигом мантиссы операнда B относительно мантиссы AC (потому что мантисса операнда B обычно доступна в начале цикла, в то время как результат AC получен только после нескольких вычислений). Выравнивание B осуществляется вправо, устанавливая его сначала на 27 разрядов мантиссы операнда B левее от запятой результата AC.

Сдвиг мантиссы операнда B тогда рассчитывается так:

$$\text{сдвиг} = \exp(AC) - \exp(B) + 27,$$

где exp (AC) - это степень операнда A, добавленная к степени операнда

C, и

exp (B) - степень операнда B.

24-разрядная мантисса операнда B должна быть установлена по меньшей мере на 24 разряда левее результата AC (для сдвига только вправо). Результат AC (24-разряда * 24-разряда) приводит к 48-разрядному промежуточному результату с двумя битами слева от запятой. Два дополнительных бита, установленные между мантиссой операнда B и результатом AC, как показано в рис. 4, позволяют битам защиты и округления (guard и round разряды) быть нулевыми и для AC рассчитать sticky-бит, когда B операнд не надо сдвигать (для нулевых или отрицательных сдвигов). Таким образом, реализуется округление по стандарту IEEE. Полная реализация умножителя требует также сдвигателя вправо для мантиссы операнда B, которая может сдвигать 74 разряда, и, соответственно 74-разрядного сумматора.

Сокращение массива множителей.

При использовании дерева Уоллеса для расчета суммы частичных произведений существенно сокращается количество сумматоров. Для большей эффективности число стадий должно быть существенно уменьшено из-за использования СБИС-технологии, КМОП имеет существенную задержку из-за больших соединений, требуемых в дереве Уоллеса. В этой работе используется сумматор с сохранением переноса 4:2. Это означает, что после сложения четырех n-разрядных чисел получится двухрядный код, задающий сумму и перенос результата. Код задан такими логическими формулами:

$$C_o = \overline{(i_o \cdot i_1 + i_2 \cdot i_3)},$$

$$C = (i_2 \oplus i_3 + \overline{c}_{in}) \cdot i_0 \oplus i_1 + i_0 \cdot i_1 \cdot i_2 \cdot i_3 + i_2 \oplus i_3 \cdot \overline{c}_{in},$$

$$S = i_0 \oplus i_1 \oplus i_2 \oplus i_3 \oplus \overline{C_{in}},$$

i_0, i_1, i_2, i_3 - соответствующие разряды n-разрядных чисел.

C_{in} - разряд переноса с предыдущего компрессора.

C_o - разряд переноса на следующий компрессор.

C, S - соответствующие разряды переноса и суммы, получаемые в результате.

Полное умножение двух 24-разрядных чисел требует 12 устройств кодирования по алгоритму Бута по основанию 4, которое даст 12 частичных произведений, которые будут сложены посредством CSA 4:2. 12 частичных произведений сложены с помощью пяти групп CSA (три в первом уровне для сложения 12 частичных произведений; одна во втором уровне, чтобы сложить две суммы и два переноса от первого уровня; одна в третьем уровне, чтобы сложить оставшиеся сумму и перенос с первого уровня и сумму и перенос получившиеся на втором уровне) чтобы уменьшить суммирование до двух частичных произведений (выводы суммы и переноса формируют нижний 4:2 CSA уровень). Выровненная мантисса операнда B затем добавляется к частичным произведениям суммы и переноса дерева Уоллеса с помощью 3:2 CSA.

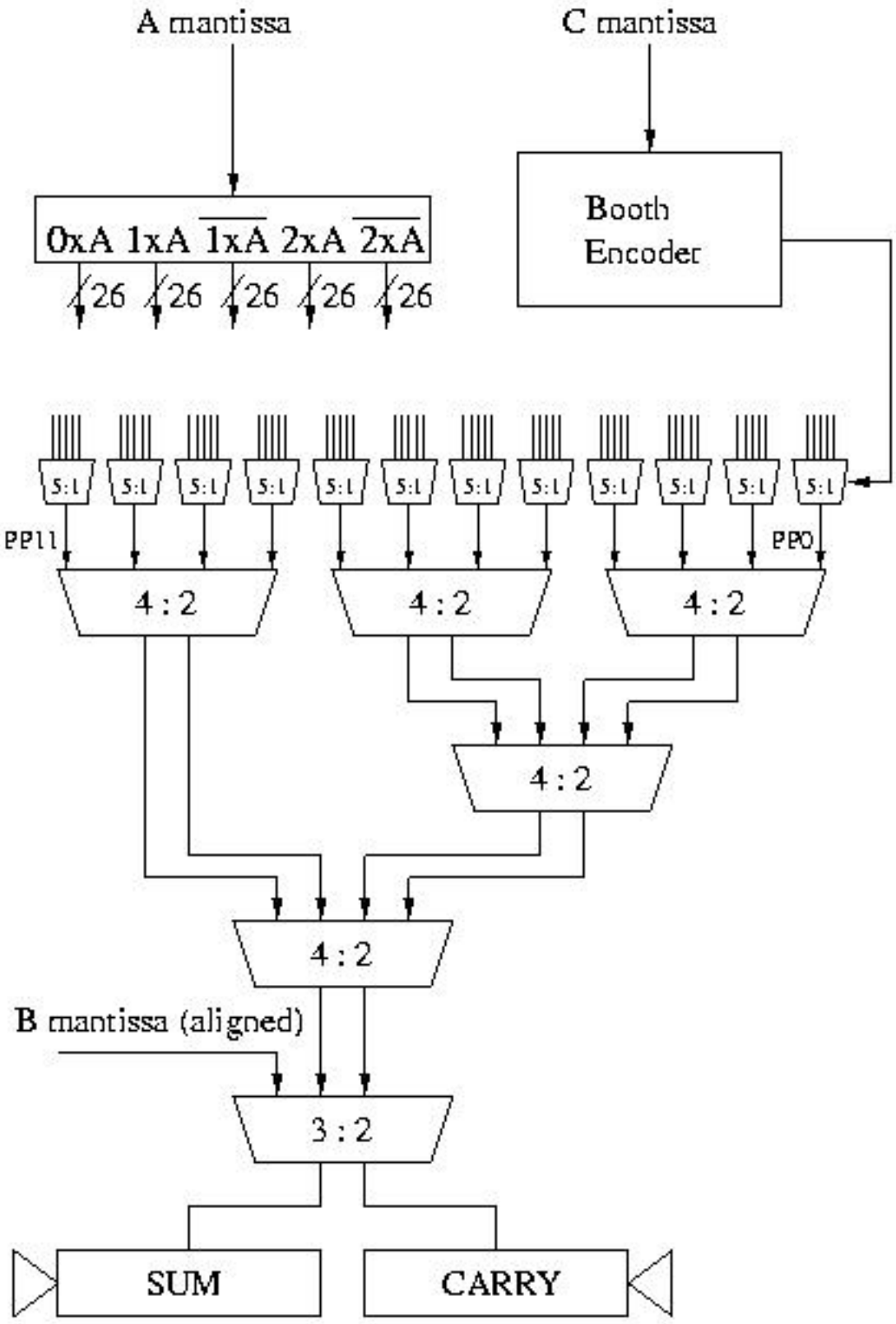


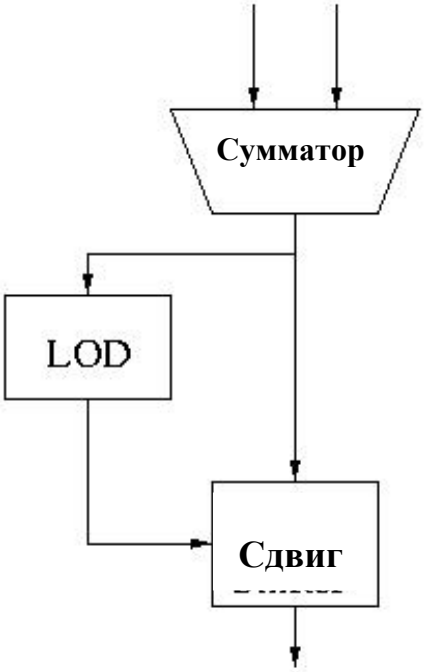
Рисунок 2. Дерево Уоллеса для 24-разрядной мантиссы.

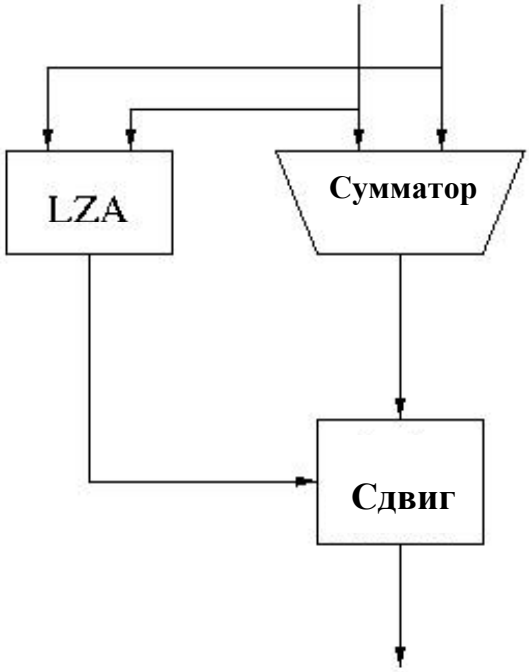
Предварительное нахождение старшего разряда.

Для понижения задержки MAF'a до минимума в 3 цикла нормализация выполняется параллельно со сложением. Это делает LZA, который получает сигналы генерации и распространения от сумматора и на выходе дает значение сдвига. Так как известно, что распространение переноса займет время log(N), где N - разрядность сумматора, было бы неплохо, чтобы нахождение старшего 0/1 было рассчитано за этот же период времени. По тому же принципу, что и распространение переноса сумматора, первый ненулевой элемент может быть определен при помощи распространения переноса от младшего разряда слова к старшему.

При сложении вещественных чисел в результате вычитания может потребоваться сдвиг влево на шаге нормализации. Самым простым решением этой проблемы будет дождаться результата сложения, затем определить старший бит единицы и посчитать количество предшествующих ему единиц. Это число затем будет использоваться для нормализации и формирования окончательного результата. Такой подход слишком долог оттого, что определение старшей единицы не может начаться, пока не завершено сложение.

Предсказание старшей единицы(нуля) - это методика, при которой число стоящих в старших разрядах результата нулей(единиц) может быть предсказано сразу, исходя из значений входных операндов с точностью до одного разряда, в параллели с шагом сложения/вычитания. Ошибка может получиться при возможном переносе. Результат предсказания может быть затем скорректирован, когда станет ясно, есть ли перенос. Ниже на рисунке показана разница между этими двумя методиками.





Нахождение старшей единицы **Предсказание старшей единицы**

Предсказание старшей единицы производится по следующей методике.

X^i - строка X длиной i.

X^*- строка X любой длины (включая пустую строку)

Например, запись X*YZ* означает, что строка начинается с любого числа X'ов, затем следует один разряд Y, а заканчивается любым числом Z'ов.

a[i] и b[i] - i-тые биты входных операндов A и B, соответственно.

$$T[i] = a[i] \oplus b[i]$$

$$Z[i] = \overline{a[i] + b[i]}$$

$$G[i] = a[i] \cdot b[i]$$

$$P[i] = a[i] + b[i]$$

Самый старший бит имеет номер 0.

Примеры: При A=11000 и B=10001 строка будет $GTZZT$

При A=1111110000 и B=0000010000 строка будет T^5GZ^4

Прежде всего, надо заметить, что предсказание старшего 0/1 требуется

не во всех случаях. Например, сдвиг старших нулей требуется, если мы получим такие битовые диаграммы: Z* или T*GZ*. Сдвиг старших единиц будет в случае G* или T*ZG*.

Некоторые FPU, например, у процессора Intel i860, используют компаратор, чтобы убедиться, что результат всегда положителен. Для таких устройств потребуется только случай T*GZ*.

Рассмотрим все 4 случая. LZA работает так. Для каждого разряда i создается сигнал SH[i]=1, если будет найдена одна из 4-х битовых диаграмм. Логическое выражение SH[i] будет выглядеть так:

$$SH[i] = Z^i + T^j GZ^k + G^i + T^j ZG^k$$

где j и k целые и находятся в интервале [0,i-1] и j+k=i-1.

Для N-разрядных операндов выходом LZA будет N-разрядный вектор - массив SH[i] - состоящий из строки единиц, за котрой следует строка нулей. Переход из единицы в ноль показывает местоположение старшей единицы или нуля. Иногда массив SH[i] может содержать одни нули, когда не было найдено соответствующих диаграмм.

В RS/6000, например, SH[0] всегда равен 1, следовательно массив содержит по меньшей мере единицу. С другой стороны, выход LZA может быть представлен в коде одним из N с простым обозначением местоположения старшей единицы или нуля. Эти два представления легко взаимозаменяемы. В обоих случаях возможный сдвиг, содержащийся в SH[i], может быть сокращен одним разрядом из-за возможного переноса. В зависимости от исполнения LZA, общий сдвиг будет равен:

$$SH_{total} = SH_{coarse} + C_{fine}, или$$

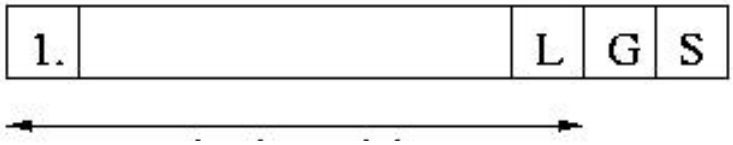
$$SH_{total} = SH_{coarse} - C_{fine},$$

Где SH_{coarse} - число единиц в массиве SH[i]

и $C_{fine} = 1$, если требуется корректировка.

Округление.

В данной работе используется округление к ближайшему четному по стандарту IEEE-754. Это означает, что число округляется к ближайшему, а в случае, если находится посередине, то к четному.



Разрядность мантиссы

На рисунке показана мантисса до округления. L - это младший значимый бит, G - guard-бит, S - sticky бит. G - Это следующий после L бит, который требуется форматом. A S - это знак того, что разряды младше G ненулевые. S - это логическое ИЛИ всех следующих после G разрядов.

L	G	S	Действия	A
X	0	0	Результат точный. Округления не требуется.	0
X	0	1	Результат неточный. Округления не требуется.	0
0	1	0	Мантисса четная. Округления не требуется.	0
1	1	0	Мантисса нечетная. Округление к ближайшему четному.	1
X	1	1	Округление к ближайшему	1

Округленный результат получается после добавления A к разряду L.

ЭКОНОМИЯ ТАКТА ПРИ ОКРУГЛЕНИИ.

Фактически, округление при таком подходе - это добавление единицы или нуля к младшему биту мантиссы. А это значит, что мы должны ставить на последнем такте сумматор или инкрементор, что станет причиной повышения задержки. Поэтому предлагается передавать на вход следующего аналогичного элемента знак, степень и мантиссу результата. В коде операции

должно быть указано, требуется брать новый элемент из регистра либо использовать данные, поступающие с выхода предыдущего элемента. Для этого на входе стоят мультиплексоры.

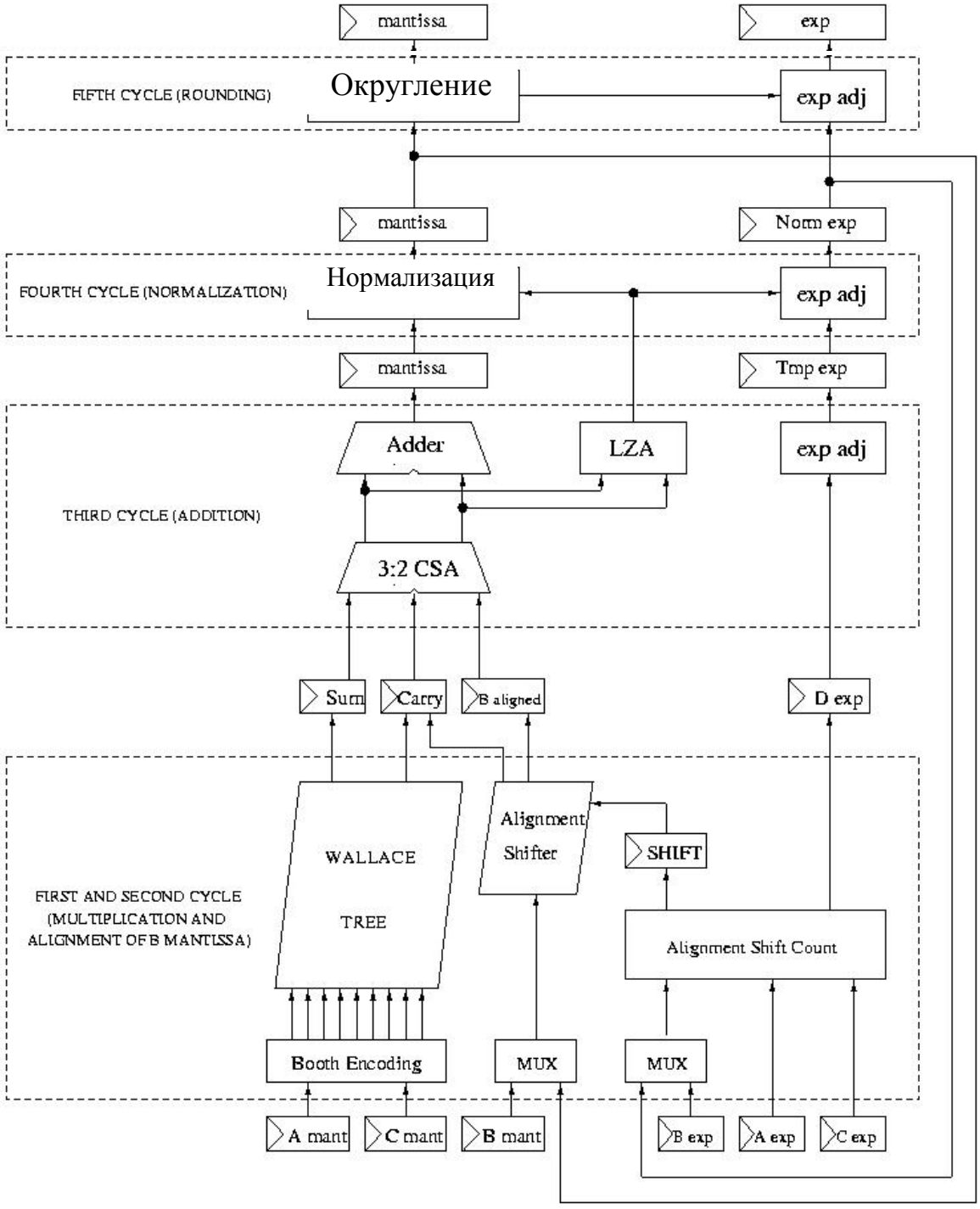
Синхронизация в 5 тактов. Имеется последовательность команд. Четвертая команда должна выполняться только после получения результата первой.

В первом случае результат выполнения четвертой команды получим только на десятом такте. Во втором случае - уже на восьмом. При увеличении количества последовательно следующих команд умножения с накоплением производительность значительно увеличится.

Такт	1	2	3	4	5	6	7	8	9	10
Умножение 1	I1	I2	I3			I4(I1)				
Умножение 2		I1	I2	I3			I4(I1)			
Сложение			I1	I2	I3			I4(I1)		
Нормализация				I1	I2	I3			I4(I1)	
Округление					I1	I2	I3			I4(I1)

Такт	1	2	3	4	5	6	7	8	9	10
Умножение 1	I1	I2	I3	I4(I1)	I5(I2)	I6(I3)				
Умножение 2		I1	I2	I3	I4(I1)	I5(I2)	I6(I3)			
Сложение			I1	I2	I3	I4(I1)	I5(I2)	I6(I3)		
Нормализация				I1	I2	I3	I4(I1)	I5(I2)	I6(I3)	
Округление					I1	I2	I3	I4(I1)	I5(I2)	I6(I3)

Рисунок 3. Общая схема алгоритма умножения с накоплением с предсказанием старшего бита и экономией такта при округлении



Эта структура была смоделирована в Verilog HDL. Правильность структуры была проверена с помощью специализированного теста.

ЗАКЛЮЧЕНИЕ.

Реферат содержит в себе основные тезисы дипломной работы, которая была защищена в 2000 году на кафедре микроэлектроники Московского Государственного Инженерно-Физического Института. В ней предложен новый подход к выполнению операции умножения с накоплением для вещественных чисел (MAF FPU), который позволяет существенно сократить время на реализацию данной операции.

В первой части реферата объясняется преимущество операции MAF FPU над простым умножением со сложением. Выигрыш существенен не только в количестве внешних выводов и времени выполнения операции, но и в точности. Особенно при работе с большими массивами чисел.

Вторая часть реферата содержит сам алгоритм MAF FPU, при котором одновременно с умножением по алгоритму Бута с использованием дерева Уоллеса происходит выравнивание мантиссы слагаемого для последующей их нормализации и округления.

В третьей части работы объясняется алгоритм предварительного предсказания сдвига для нормализации результата, а также округления без потери такта работы процессора.

Структура, работа которой рассмотрена теоретически в реферате, была реализована автором средствами Verilog HDL и показала правильность своей работы.

Дипломная работа была выполнена в Московском Центре СПАРК-Технологий под научным руководством д.т.н. Горштейна В.Я.