Квазисамосинхронный вычислитель: методологические и алгоритмические аспекты

Ю.Г. Дьяченко[1], Ю.А. Степченков[1], С.Г. Бобков[2]

[1]Институт проблем информатики РАН, YStepchenkov@ipiran.ru, ia_ste@mail.ru

[2]Научно-исследовательский институт системных исследований РАН

***Аннотация* — Представлены подходы к проектированию самосинхронной аппаратуры различных классов и рассмотрены условия внутрисистемной интеграции синхронных (С) и самосинхронных (СС) устройств на примере разработки квазисамосинхронного вычислительного устройства (в дальнейшем – ВУ), выполняющего функции деления и извлечения квадратного корня.**

I. Введение.

Синхронизация – одна из важнейших задач в цифровых системах, решающая проблему координации событий в аппаратуре и связанная, в основном, с обеспечением интерфейса между *физическим* (естественным) и *логическим* (искусственным) временем.

В середине 1950-ых годов активно исследовались две альтернативные методологии синхронизации элементов в аппаратуре: *синхронная (С) и самосинхронная (СС).* При С-методе физическое время удаляется из поведения модели, и интерфейс между физическим и логическим временем определяется системными часами. События во внешних часах отделены от модели системного поведения и не имеют завершенного причинно-следственного отношения к событиям в системе.

Простота С-подхода стала причиной преобладания так называемых FS-систем (Fully Synchronous) с одним генератором (G) и рядом устройств (D) – см. рис. 1,а.

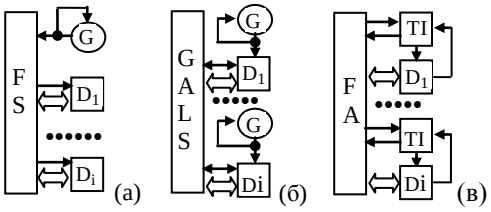

Рис. 1. Типы организации синхронизации

Для корректной работы синхронизируемой аппаратуры период синхроимпульсов выбирается из расчета на *худший случай* – максимально возможное время переключения элементов при неблагоприятных сочетаниях условий работы (напряжения питания, температуры и т.п.). Таким образом, цена корректной работы С-аппаратуры – недоиспользование ее возможностей по быстродействию по сравнению с номинально возможным быстродействием. Рис. 2 из [1] иллюстрирует временные потери, типичные для С-системы.

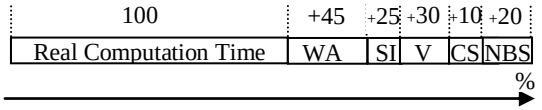

Рис. 2. Временные потери в синхронном ВУ

Вынужденная ориентация на худший случай работы WA (Worst Average), устранение перекоса сигналов CS (Clock Skew), потери из-за несбалансированности ступеней конвейера (NBC), учет разброса параметров технологии V (Variability) и необходимость сохранения правильной формы сигналов SI (Signal Integrity) вынуждает снижать быстродействие аппаратуры на 130% для обеспечения правильной работы в узком диапазоне напряжений (± 10% от номинала).

По мере уменьшения проектных норм, увеличения степени интеграции и повышения быстродействия недостатки С-подхода становятся все более очевидными: система доставки синхроимпульсов к пунктам их потребления становится менее эффективной. Отсюда актуален переход к GALS-системам (Global Asynchronous / Local Synchronous), которые переводят решение ряда проблем синхронизации устройств D в локальные области (см. рис. 1,б).

Альтернатива FS-подходу – полностью асинхронный (FA, Fully Asynchronous), в котором системное время представляется причинно-следственной связью между событиями (см. рис. 1,в). Интерфейс между логическим и физическим временами (TI-интерфейс, Timing Interface) обеспечивается механизмами индикации завершения переходных процессов в элементах системы.

Отказ от использования общих часов требует решения ряда проблем высокой сложности и, в том числе, необходимости согласования СС-аппаратуры с синхронным (в основном) окружением.

В ряде теоретических работ В.И. Варшавского предложена методология GALA-систем (Global Asynchronous / Local Arbitrary), где проектировщик может использовать широкую гамму решений – от реализации полной самосинхронности до введения параллельной задержки и использования локального генератора. Все зависит от целей проектируемого изделия.

Несмотря на теоретически доказанные и практически подтвержденные преимущества СС-схем, до сих пор преобладают С-системы. Дело в том, что СС-схемы

также имеют свои недостатки. Изначальная ориентация СС-подхода на динамическую (поведенческую) модель схемы предопределяет б**о**льшую трудоемкость проектирования по сравнению со статической моделью аналогичной синхронной схемы.

В настоящее время ситуация кардинально меняется. В частности, переход к субмикронной технологии и необходимость ориентации перспективных САПР СБИС на поведенческий стиль проектирования приводит к выравниванию сложности проектирования всех типов СБИС и необходимости использования моделей и результатов, полученных для СС-схем, в САПР СБИС общего назначения. Основной аргумент, предопределивший в середине 50-х годов ориентацию на С-стиль проектирования, перестал действовать.

Естественно, эволюционный путь развития С-схемотехники еще не исчерпан. Инженерная смекалка и современные технологические возможности позволят некоторое время решать текущие схемотехнические проблемы в рамках С-методологии. Однако можно утверждать, что современные проблемы разработки СБИС-систем могут быть решены более эффективно при переходе на синхронно-самосинхронный стиль проектирование.

Основным преимуществом СС-схем [2], [3] является стабильная работа при любых конечных задержках элементов. Это позволяет им работать при напряжениях питания и условиях окружающей среды, близких к граничным, определяемым физическими свойствами полупроводниковых активных элементов – транзисторов, если это не критично с точки зрения быстродействия устройства. Увеличение аппаратных затрат (количества транзисторов в схеме), особенно заметное при реализации комбинационных узлов, нивелируется отсутствием схемы синхронизации и в ряде практических случаев оказывается незначительным (например, при реализации последовательностных устройств). Кроме того, достигаемое при этом расширение области работоспособности (в первую очередь – по напряжению питания) компенсирует усложнение схем в устройствах, предназначенных для работы с ограниченными источниками питания.

В данном докладе представлена разработка алгоритма функционирования СС-устройства, выполняющего функции деления и извлечения квадратного корня. Обрабатываемыми операндами служат числа в соответствии со стандартом IEEE 754 [4] одинарной и двойной точности. Вычисления реализованы в соответствии со стандартом с некоторыми упрощениями:

- на ВУ поступают только нормализованные операнды;
- результат – также нормализованное число;
- результат, не представимый в виде нормализованного числа, вызывает исключительную ситуацию.

В настоящее время известно множество алгоритмов вычисления результатов деления и извлечения квадратного корня [5]-[8]. Однако не все они могут эффективно использовать СС-схемотехнику. Особенность СС-схем – запрос-ответное взаимодействие между блоками, соседними в цепочке обработки информации – делает наиболее эффективной многостадийную реализацию любого вычислительного алгоритма, когда управление передается от предыдущей стадии к последующей асинхронно, в соответствии с реальным быстродействием каждой стадии. Деление и извлечение квадратного корня не являются в этом отношении исключением. Особенность предлагаемого решения – совмещение в одном устройстве этих двух функций.

II. Алгоритм деления и извлечения корня

В соответствии со стандартом IEEE 754 обрабатываемые операнды представляются в виде числа с плавающей точкой и состоят из бита знака, поля экспоненты и поля мантиссы. Длины полей экспоненты и мантиссы зависят от точности обрабатываемого операнда – одинарной или двойной. Алгоритм вычисления одинаков для обеих точностей: мантиссы и экспоненты обрабатываются отдельно. Вычисление экспоненты результата сводится к вычитанию экспоненты делителя из экспоненты делимого при выполнении деления или к сдвигу в сторону младших разрядов при извлечении квадратного корня. Основную же сложность представляет собой вычисление мантиссы результата. В дальнейшем под вычислением результата мы будем понимать вычисление мантиссы.

Современные быстрые алгоритмы деления используют оценку ожидаемого значения частичного результата (одного или нескольких битов мантиссы результата), начиная со старших разрядов, на основе значений нескольких старших разрядов промежуточных данных. Среди множества алгоритмов деления можно выделить два основных класса: безвозвратный и с опциональными возвратами в случае ошибочного предсказания частичного результата. Алгоритмы первого класса предусматривают возможность коррекции накапливаемого результата вычисления на последующих шагах (итерациях), если оценка на текущем шаге оказалась неточной и остаток от деления оказался отрицательным. Алгоритмы второго класса требуют получения корректной оценки на каждом шаге и в случае обнаружения ее некорректности возвращаются к ее вычислению до тех пор, пока получаемый промежуточный результат (остаток от деления) не будет неотрицательным. Наиболее быстродействующие алгоритмы основаны на использовании умножения (например, метод Newton-Raphson) или табличных методов. Однако они требуют реализации аппаратного умножителя или относительно больших объемов статической памяти или ПЗУ, которые при реализации в СС-базисе приводят к значительным аппаратным затратам и оказываются не столь эффективными. По-

этому в качестве базового был выбран безвозвратный алгоритм деления.

Этот алгоритм аналогичен делению "в столбик" и основан на использовании рекуррентной формулы:

$$P_{i+1} = r \cdot P_i - D \cdot q_i, \quad i=0,\ldots n-1, \qquad (1)$$

где P_{i+1}, P_i – промежуточные остатки от деления на i-том и (i+1)-ом шагах алгоритма; r – основание алгоритма (radix); D – делитель; q_i – частичный результат, полученный на i-ом шаге; n – число шагов алгоритма, зависящее от размерности обрабатываемых операндов N и основания алгоритма: $n=]N/(2^{r-1})[$. В качестве P_0 используется исходное делимое. Окончательный результат формируется из частичных результатов путем их конкатенации:

$$Q = \{q_0 q_1 q_2 \ldots q_{n-1}\}.$$

Основание алгоритма r может принимать значения из двоичного ряда {1, 2, 4, 8,..}. Чем больше основание алгоритма, тем за меньшее число шагов формируется результат. При этом частичный результат содержит один или несколько битов окончательного результата.

Простейшая реализация безвозвратного алгоритма (реализация "в лоб") предполагает, что на каждом шагу (на каждой итерации) частичный результат представляет собой неотрицательное число, максимальное значение которого определяется как $(2^{r-1}-1)$. При этом наибольшую сложность представляет корректная оценка частичного результата на каждом шаге алгоритма. Прямолинейное решение выливается в реализацию вычислений по формуле (1) для всех возможных значений q_i, что для высоких оснований алгоритма становится весьма аппаратно затратным. Избежать этого позволяет SRT-алгоритм, названный так по начальным буквам фамилий разработчиков (Sweeney, Robertson, Tocher), предложивших его независимо друг от друга [9]. Особенности алгоритма:

1) Частичный результат q_i может быть как положительным, так и отрицательным. Положительное значение означает, что накопленный к данному шагу алгоритма промежуточный результат (старшие разряды частного от деления) – правильный, а отрицательное значение вносит коррекцию в накопленный результат так, чтобы он соответствовал правильному окончательному результату. Поскольку в SRT-алгоритме оценка очередного частичного результата проводится на основе реализации вычислений по формуле (1) с участием только нескольких старших разрядов операндов (их количество зависит от основания алгоритма r), ошибка в выборе правильного значения частичного результата не столь редка. Но она оказывается некритичной из-за возможности последующей коррекции отрицательными значениями частичного результата на дальнейших итерациях.

2) Промежуточный остаток, вычисляемый по формуле (1) хранится и используется в избыточном представлении. Каждый разряд остатка P_{i+1} состоит из двух битов: бита суммы S и бита переноса C. Это ускоряет вычисление остатка на каждом шаге алгоритма, поскольку для его формирования не требуется реализовывать сквозной перенос в каком бы то ни было виде. Вместо полного многоразрядного сумматора достаточно использовать сумматор с сохранением переноса, в котором сигнал переноса в каждом разряде – внешний. Однако такая реализация увеличивает количество сигналов, используемых в вычислениях, и требует применения многоразрядного сумматора с распространением переноса в конце цикла вычисления результата для формирования истинного остатка от деления. При этом составляющие суммы и переноса из избыточного представления промежуточного остатка являются двумя слагаемыми этого сумматора.

3) Для ускорения вычислений на каждом шаге алгоритма накопление результата осуществляется также в избыточном представлении: одновременно формируются, хранятся и используются на последующих шагах два результата – истинный на данный момент результат Q_i и он же за вычетом единицы младшего разряда R_i. Последний вариант подставляется вместо истинного результата, если на очередном шаге получается частичный результат $q_i = -1$, и требуется скорректировать накопленный результат. В противном случае в такой ситуации пришлось бы вычитать единицу младшего разряда из накопленного результата Q_i с помощью многоразрядного вычитателя.

Таким образом, SRT-алгоритм ускоряет вычисление частного при делении, вынося часть операций за пределы основного цикла, но вносит одновременно некоторую избыточность в массив формируемых и обрабатываемых промежуточных операндов.

В настоящее время существуют реализации SRT-алгоритма для различных оснований, вплоть до $r = 512$. Однако для реализации блока вычисления частного и квадратного корня наиболее целесообразным является SRT-алгоритм с основанием $r = 2$, позволяющий объединить функции деления и извлечения квадратного корня в одном блоке за счет минимальных дополнительных затрат [10], [11]. Это немаловажно при реализации цифровых устройств в СС-базисе.

Действительно, рекуррентная формула для вычисления квадратного корня в безвозвратном алгоритме с основанием $r = 2$ очень похожа на формулу выполнения деления и выглядит так:

$$P_{i+1} = 2 \cdot P_i - (2 \cdot Q_{i-1} + q_i \cdot 2^{-i}) \cdot q_i, \qquad (2)$$

где Q_{i-1} – результат извлечения корня, накопленный к i-тому шагу алгоритма. В качестве P_0 используется исходное подкоренное выражение, а $Q_{-1}=0$. Формулы (1) и (2) отличаются только тем, что в случае деления из удвоенного остатка вычитается взвешенный частичным результатом q_i делитель, а при вычислении квадратного корня – вспомогательный операнд, определяемый "на лету" из накопленного к текущему шагу алгоритма результата Q_{i-1}. При этом результат накап-

ливается в обоих случаях одинаковым образом.

Свойство самосинхронности алгоритму придают парафазная дисциплина кодирования всех информационных сигналов, индицирование моментов окончания всех переходных процессов и организация запрос-ответного взаимодействия между соседними блоками в ВУ. Аналогичные средства уже использовались в работах [10], [11]. Но они не удовлетворяют требованиям строгой самосинхронности; заложенные в них схемы запрос-ответного взаимодействия не рассчитаны на возможность появления большого разброса в задержках однотипных элементов из-за деградации характеристик активных структур или локальных отклонений технологических параметров.

Ниже раскрываются функциональные особенности реализации описанного алгоритма. Для простоты изложения формулы и функции приводятся в нотации для инфазного представления информационных сигналов. Парафазная реализация получается путем применения известных правил кодирования [2].

III. Функциональная реализация

Структурная схема устройства, реализующего один шаг алгоритма совмещенного вычисления частного и квадратного корня (стадия), представлена на рис. 3.

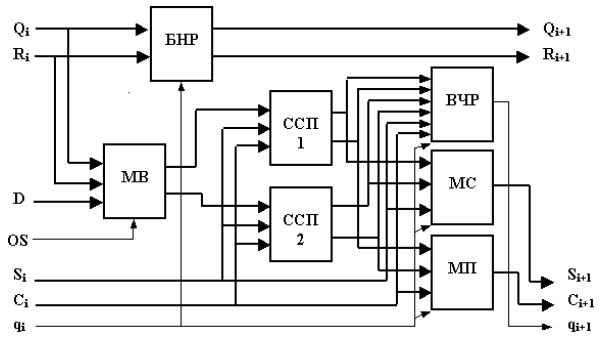

Рис. 3. Структурная схема стадии алгоритма

Стадия содержит следующие основные блоки:

- БНР – блок накопления результата;
- МВ – мультиплексор вычитаемого (второго члена формул (1) и (2));
- ССП1, ССП2 – сумматоры с сохранением переноса, первый из которых реализует формулу (1) для случая $q_i = -1$, а второй – для случая $q_i = +1$;
- ВЧР – выбор частичного результата q_i;
- МС – мультиплексор суммы;
- МП – мультиплексор переноса.

Блок БНР формирует "на лету" одновременно как сам результат вычисления Q, так и его дополнение R, равное результату минус единица текущего разряда. Это позволяет при значении частичного результата $q = -1$ просто подменять текущий результат Q на R, не выполняя вычитания, в соответствии с формулами:

$$Q_i = (1-|q|)\cdot Q_i + \frac{(|q|-q)}{2}\cdot(R_i + P_i) + \frac{(|q|+q)}{2}\cdot(Q_i + P_i),$$

$$R_i = (1-|q|)\cdot(R_i + P_i) + \frac{(|q|-q)}{2}\cdot R_i + \frac{(|q|+q)}{2}\cdot Q_i,$$

где Q_i, R_i – i-тый бит результата и его дополнения; q – частичный результат, полученный на предыдущем шаге и принимающий одно из значений {–1,0,+1}; P_i – флаг активности разряда; i=0,..,55 – номер бита результата.

Флаг активности разряда P_i равен 1, если его индекс i соответствует номеру текущего формируемого бита результата. В остальных случаях он равен нулю. Результат формируется побитно, начиная со старшего разряда. Начальные значения операндов Q и R – нулевые. Как видно из схемы на рис. 3, выходы блока БНР используются только на следующем шаге алгоритма. Следовательно, блок БНР не входит в критический путь данной стадии, если его собственная задержка не превышает задержки критического пути.

Мультиплексор вычитаемого формирует второй член формул (1) и (2) на основе накопленного промежуточного результата Q и R (для операции извлечения квадратного корня) или входного операнда – делителя D. Поскольку вычитание выполняется в дополнительном коде как суммирование с инверсным представлением операнда с добавлением единицы в младший разряд, делитель представлен одновременно своим исходным и инверсным значениями.

Вычитаемое для случая $q = -1$ определяется по формуле:

$$A_i = (R_i + P_i + P_{i-1})\cdot\overline{OS} + D_i\cdot OS,$$

где P_i, P_{i-1} – флаги активности текущего и предыдущего разрядов; OS – признак выполняемой операции – деление (OS=1) или извлечение квадратного корня (OS=0); D_i – i-тый бит делителя. Вычитаемое для случая $q = +1$ формируется в соответствии с формулой:

$$S_i = (\overline{Q_i}\cdot\overline{P_i})\cdot\overline{OS} + \overline{D_i}\cdot OS.$$

Сумматоры с сохранением переноса работают параллельно, хотя на каждом шаге алгоритма фактически требуется не более одного из них. Их особенность заключается в том, что сигнал переноса, формируемый в каждом разряде, передается в соседний по старшинству разряд сумматора не текущей стадии, а следующей. Информация при этом не теряется за счет хранения результата суммирования в избыточном представлении: и суммы, и переноса для каждого разряда. Для получения безызбыточного представления такой суммы требуется использовать сумматор с распространением переноса. В данном алгоритме это делается в конце циклических вычислений для формирования итогового остатка и округления результата.

Использование двух сумматоров с сохранением переноса дополнительно повышает быстродействие алгоритма, уменьшая количество каскадов на критическом пути обработки данных; в противном случае

пришлось бы добавить перед сумматором мультиплексор для выбора нужного вычитаемого в соответствии со значением частичного результата на предыдущем шаге алгоритма.

Анализ показывает, что наибольшей задержкой среди блоков, формирующих выходы стадии на рис. 3, обладает блок ВЧР (с особенностью его работы можно познакомиться в [12]). Можно с уверенностью утверждать, что частичный результат q_{i+1} формируется последним из всех выходных сигналов, показанных на рис. 3, и к моменту его формирования остальные входные сигналы для блоков БНР, МС, МП и ВЧР следующей стадии будут уже готовы. Так обеспечивается минимальное время выполнения одного шага алгоритма при многостадийной организации вычислений.

Реализация схемы индикации окончания переходных процессов зависит от требований, предъявляемых к разрабатываемому проекту. Если в С-конвейерах отсутствует фиксация окончания переходных процессов и в памяти (FF, Flip-Flop), и в CL (Combination Logic) – см. рис. 4,а, то в квазисамосинхронных конвейерах окончание переходных процессов в FF фиксируется, а окончание переходного процесса в CL моделируется встроенной задержкой DL (DeLay) – рис. 4,б.

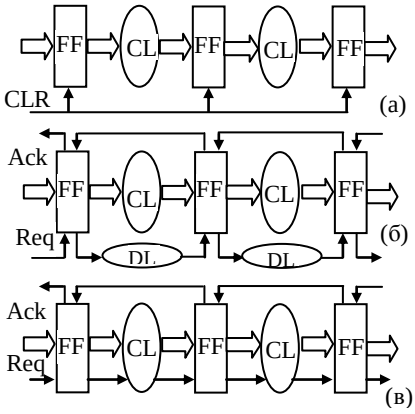

Рис. 4. Варианты реализации конвейеров

В строго самосинхронных схемах требуется индицировать окончание переключения каждого элемента, чтобы вовремя отловить появление неисправности в схеме и остановить вычисления (см. рис. 4,в). Реализация этого принципа обеспечивает также и независимость работоспособности схемы от соотношения задержек составляющих ее элементов. Однако практическая реализация полномасштабной индикации связана с большими накладными расходами – до 50 % от сложности самой схемы. Поэтому иногда руководствуются принципом "ожидаемого соотношения задержек элементов, реализованных на одном кристалле". В соответствии с ним до момента появления критических дефектов на кристалле можно гарантировать следующее:

- однотипные элементы в любом месте кристалла будут иметь близкие задержки переключения;
- чем больше входов у элемента, тем меньше его среднее быстродействие;
- более нагруженные цепи имеют и бо́льшие задержки.

Такой подход позволяет существенно сократить затраты на индикацию схемы и ускорить ее работу. В большинстве практических случаев, не рассчитанных на работу в экстремальных условиях, этого оказывается достаточно для обеспечения бессбойной работы схемы в широком диапазоне условий эксплуатации. И в том, и в другом подходе неизменным остается одно – организация запрос-ответного взаимодействия между соседними блоками.

Индикаторный выход стадии является сборкой всех индикаторных сигналов внутри стадии. Традиционно для этой цели используются гистерезисные триггеры [2] или С-элементы [3]. Переключение индикатора в новое состояние свидетельствует об окончании переходных процессов во всех индицируемых элементах стадии. Вход управления разрешает переключение стадии в соответствующую фазу работы. Это до некоторой степени аналог синхросигнала в традиционной синхронной схемотехнике.

Одна из проблем реализации многостадийных вычислений – хранение промежуточного результата. В синхронных реализациях она традиционно решается с помощью регистров, разделяющих соседние стадии. В СС-схемах все стадии работают автономно, переключаясь по своим собственным задержкам, зависящим от типа обрабатываемых данных; поэтому соседние стадии могут иметь разные задержки переключения. Но моменты окончания этих переключений индицируются, сигнализируя о готовности результата на выходе каждой стадии. Этого достаточно для организации четкого обмена данными между стадиями. Проблема хранения промежуточного результата может быть решена за счет использования элементов с памятью. В последнем случае для реализации ВУ эффективным оказывается применение элементов с динамической логикой [10]. Парафазная дисциплина информационных сигналов упрощает перевод таких элементов в спейсер и предохраняет от появления на выходе ложных кратковременных рабочих состояний при переключении в рабочую фазу.

Количество стадий в конвейерной реализации алгоритма может быть произвольным, но практически оно ограничено требованиями к габаритам топологической реализации и к мощности потребления. Теоретический анализ показывает [10], что оптимальное (с точки зрения быстродействия) число стадий в таком алгоритме равно 4 или 5. Именно в этом случае конвейер оказывается сбалансированным наилучшим образом.

На рис. 5 показана структурная схема устройства, использующая конвейер в четыре стадии:

- ST1…ST4 – стадии конвейера;
- ВРО – входной регистр операндов и признаков операции;

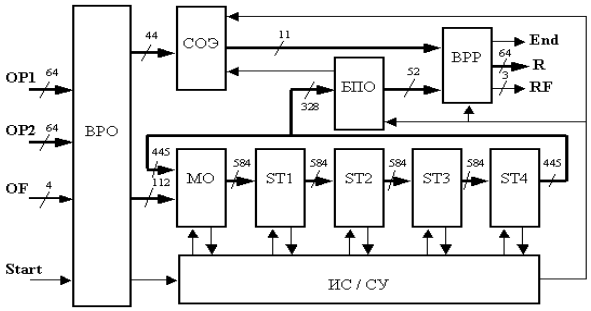

Рис. 5. Структурная схема Вычислителя

- МО – входной мультиплексор операндов;
- индикаторная схема (ИС) со схемой управления (СУ);
- СОЭ – схема обработки экспонент;
- БПО – блок постобработки мантиссы;
- ВРР – выходной регистр результата.

Блоки ВРО и ВРР реализуют асинхронный интерфейс с синхронным окружением. Входной сигнал *Start* служит для синхронизации записи в регистр ВРО входных данных: операндов *OP1*, *OP2* и признаков операции *OF*, после чего схема самостоятельно начинает выполнять операцию. Результат операции вместе с флагами исключительных ситуаций записывается в регистр ВРР. По окончании формирования и записи в выходной регистр результата поднимается флаг готовности *End*. По этому признаку синхронное окружение устройства определяет момент, когда результат и флаги исключительных ситуаций могут быть считаны из регистра ВРР.

Схема индикации охватывает своими сигналами все блоки, координируя вместе с СУ их работу.

IV. Заключение

На основе описанного алгоритма была разработана программа деления и извлечения квадратного корня на языке C++. Ее тестирование на статистически значимом наборе входных данных для различных типов операций, точностей представляемых операндов, режимов округления продемонстрировало безошибочность работы алгоритма в различных ситуациях.

Таким образом, разработка СС-алгоритма деления и извлечения квадратного корня показала:

- эффективное решение ВУ в базисе СС-схем возможно только на основе многостадийной – конвейерной – организации вычислительного тракта; в данном случае – с помощью четырех однотипных стадий, каждая из которых вычисляет один бит результата;
- предложенный алгоритм обеспечивает одинаковое быстродействие устройства при выполнении обеих операций – деления и извлечения квадратного корня;
- СС-реализация вычислительного тракта позволяет отказаться от использования регистров для хранения промежуточных результатов и за счет этого снизить энергопотребление схемы в целом;
- большое количество информационных сигналов из-за их парафазного кодирования предъявляет особые требования к технологии (число слоев разводки, паразитное влияние соседних трасс и т.д.), по которой предполагается изготавливать устройство, реализующее данный алгоритм.

Устройство деления и извлечения квадратного корня, разработанное в соответствии с данным алгоритмом, реализовано в составе тестовой схемы по КМОП-технологии с проектными нормами 0.18 мкм [12]. Ее изготовление и тестирование позволит оценить эффективность предложенного решения по сравнению с синхронными аналогами и работоспособность схемы в диапазоне изменяющихся напряжения питания и температуры окружающей среды.

Литература

[1] P. Beerel, J. Cortadella, and A. Kondratyev Bridging the gap between asynchronous design and designers (Tutorial) // VLSI Design Conference, (Mumbai). - 2004.

[2] Автоматное управление асинхронными процессами в ЭВМ и дискретных системах. Под ред. В.И. Варшавского. - М.: Наука, 1986. - 400 с.

[3] Varshavsky V., Kishinevsky M., Marakhovsky V. et al. Self-timed Control of Concurrent Processes, Ed. by V.Varshavsky - Kluver Academic Publishers. - 1990. – 245 p.

[4] IEEE Standard for Binary Floating-Point Arithmetic / IEEE Std. 754. - New York ANSI. - 1985, Aug.

[5] S.E. McQuillan and J. V. McCanny Fast VLSI algorithms for division and square root // J. VLSI Signal Processing. - Oct. 1994. - V. 8. - P. 151-168.

[6] P. Montuschi, L. Ciminiera, and A. Guistina Division unit with Newton-Raphson approximation and digit-by-digit refinement of the quotient // IEEE Proceedings: Computers and Digital Techniques. – 1994. - V. 141. - P. 317-324.

[7] Lang T. and Montuschi P. Very-high radix combined division and square root with prescaling and selection by rounding // In Proceedings of the 12th IEEE Symposium on Computer Arithmetic. - 1995, July. - P. 124–131.

[8] J. Arjun Prabhu and Gregory B. Zyner. 167 MHz Radix-8 Divide and Square Root Using Overlapped Radix-2 Stages // In Proceedings of the 12th Symposium on Computer Arithmetic. – 1995. - P. 155-162.

[9] Ajay N., Atul D., Warren J., Debjit D. S. 1-GHz HAL SPARC64 Dual Floating Point Unit with RAS Features // In Proceedings of the 15th IEEE Symposium on Computer Arithmetic. - 2001.

[10] T.E.Williams. M.A.Horowitz. A Zero-Overhead Self-Timed 160-ns 54-b CMOS Divider // IEEE Journal of Solid-State Circuits. - 1991. - V. 26. - №11. - P. 1651-1661.

[11] G. Matsubara, N. Ide, H. Tago, S. Suzuki and N. Goto. 30-m 55-b Shared Radix 2 Division and Square Root Using a Self-Timed Circuit // In Proceedings of the 12th Symposium on Computer Arithmetic. - 1995. - P. 98-105.

[12] Ю.Г. Дьяченко, Ю.В. Рождественский, В.Н. Морозов, Д.Ю. Степченков. Квазисамосинхронный вычислитель: практическая реализация. / В настоящем сборнике трудов.