1. РЕАЛИЗАЦИЯ АЛГОРИТМОВ НА ПЛИС
   1. Аппаратные блоки ПЛИС

Для реализации выбранных алгоритмов необходимо использовать интегрированные в ПЛИС аппаратные блоки DSP и RAM. Если реализация будет использовать только общие ресурсы ПЛИС, то возможно значительное снижение рабочей частоты с колоссальным увеличением используемых ресурсов, так как массивы памяти будут формироваться из триггеров.

Блоки памяти RAM могут работать в различных режимах, набор которых зависит от производителя и конкретного семейства, но для реализации нам интересен “True Dual-port” режим. Конфигурация блока памяти в этом режиме приведена на рисунке 3.1.

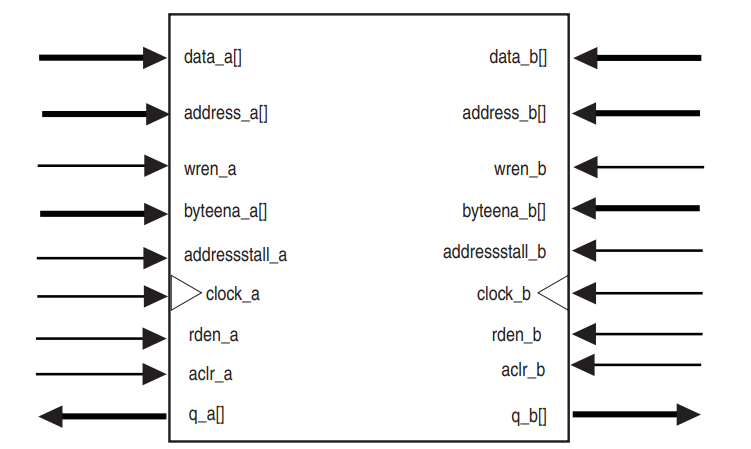


Рисунок 3.1 – Конфигурация RAM блока в режиме “True Dual-port”

В этом режиме блок памяти позволяет одновременно записывать и читать данные по независимым адресам. Причем для каждого из двух каналов рабочая частота может отличаться. Это важная особенность блока будет необходима при дальнейшей буферизации видеопотока.

Следующим важным блоком для реализации быстрых вычислений является интегрированные DSP блоки. Архитектура каждого такого блока представлена на рисунке 3.2.

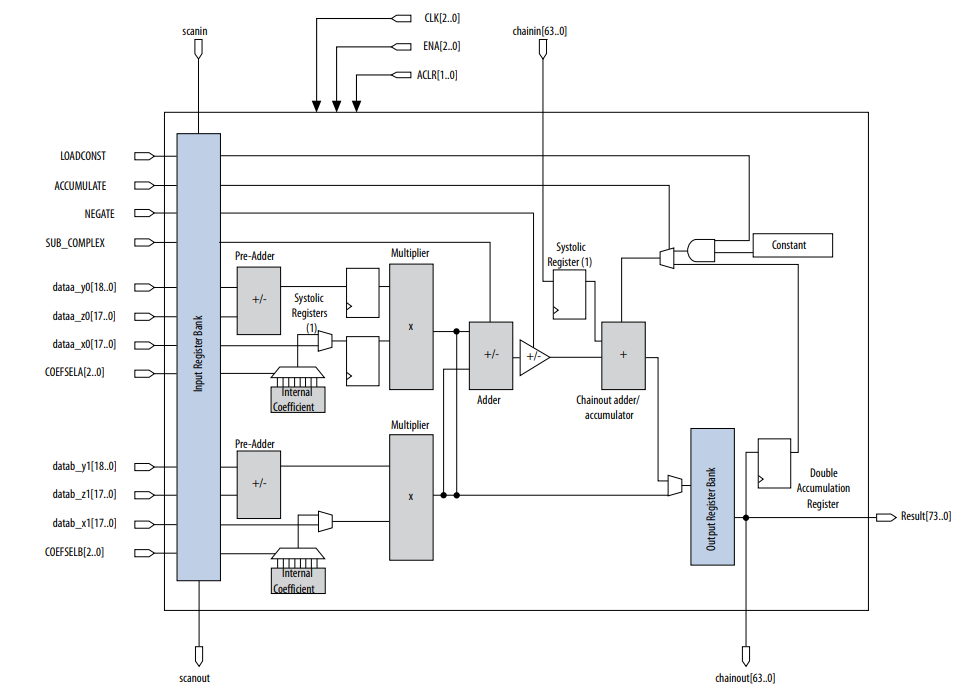


Рисунок 3.2 – Архитектура DSP блока семейства Cyclone V

Как видно из рисунка, архитектура достаточно сложная и позволяет выполнять множество различных вычислений в различных режимах.

Помимо основных интересующих нас умножителей предусмотрены входные и выходные регистры, пред-сумматор, блок внутренних коэффициентов для реализации цифровых фильтров и т.д.

Для нашей задачи нужны будут сами умножители, сумматоры и аккумулятор, позволяющий накапливать полученный результат.

Каждый из перечисленных блоков может быть также настроен на работу в различных режимах, которые отличаются разрядностью оперируемых данных – 9, 18, 27 бит. Так как в нашей задаче пикселы имеют разрядность 12 бит, то в реализации будем предполагать режим работы с 18 битными числами.

* 1. Реализация скользящего окна

Для всех выбранных алгоритмов необходимо сформировать скользящее по изображению окно. Центр этого окна должен последовательно сдвигаться на 1 пиксел вдоль строки, а затем смещаться на 1 пиксел по строке. Для формирования скользящего окна необходимо накопить число строк равное размеру маски в общем случае. Т.е. для организации маски размером 3 на 3 пикселя, необходимо принять и буферизировать 2 строки, после чего считывать эти данные одновременно с приемом текущей третей строки, параллельно сохраняя текущую строку для обеспечения минимальной задержки кадра. Либо буферизировать все 3 строки после чего перейти к формированию окна, сохраняя параллельно текущую строку.

|  |
| --- |
| https://hsto.org/files/ea0/399/1b7/ea03991b78b9484eab7d435432410a04.png |
| Рисунок 3.3 – Скользящее окно |

Как упоминалось выше, во многих ПЛИС предусмотрены аппаратные блоки памяти, которые можно использовать с целью временной буферизации строк. Тогда необходимо организовать «конвейер», позволяющий параллельно записывать пикселы текущей строки и читать предыдущие, а вывод самой маски организовать через набор сдвиговых регистров.

Структурная схема блока, реализующего описанный алгоритм скользящего по изображению окна, представлен на рисунке 3.4.

Выходы сдвиговых регистров являются элементами скользящего окна. Текущая строка параллельно с выводом в сдвиговый регистр записывается в блок памяти, организованный по принципу FIFO, таким образом осуществляется временная буферизация строки. При таком подходе, накопив 2 строки, начнется процесс вывода пикселов маски.

Однако следует отметить, что не все пиксели в видеопотоке являются валидными, часть из них используется для передачи какой-то служебной информации, оптически черных участков, а также синхрокодов для определения начала и конца текущей строки. Поэтому необходимо предусмотреть блок определяющий интервалы валидных данных.

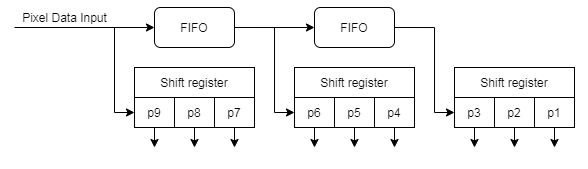


Рисунок 3.4 – Структурная схема организации скользящего окна

На самом деле можно было бы обойтись без различения, однако, тогда потребуются дополнительные накладные расходы на буферизацию ненужных данных, что повлечет за собой увеличение ресурсов для контроля за ними, а, следовательно, увеличивается и объем ресурсов, необходимый для реализации.

Для определения моментов начала кадра в системе присутствует сигнал синхронизации VD, а для определения момента начала строки сигнал HD.

Таким образом, имеются все необходимые данные для того, что определить интервалы валидных данных. Итоговая структура представлена на рисунке 3.5.

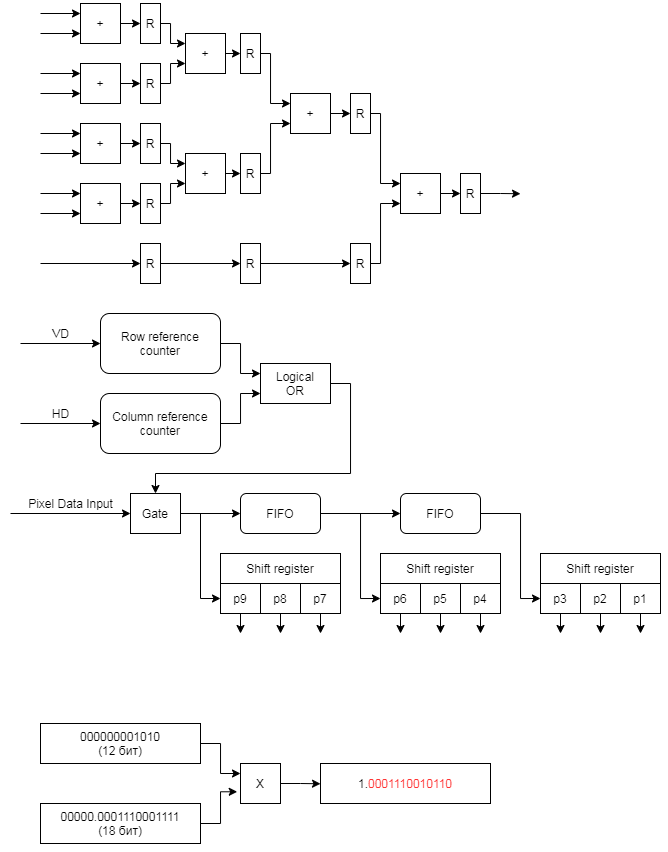


Рисунок 3.5 – Модифицированная структурная схема организации скользящего окна

* 1. Медианный фильтр

Обработка видеопотока на ПЛИС отличается от классических подходов при обработке на CPU тем, что можно выполнять ряд операций параллельно. Допустим, для сортировки пикселов в маске на CPU, требуется последовательно каждый пиксель сравнивать и перемещать, а на ПЛИС можно выполнять на каждом этапе попарные сравнения пикселов. Таким образом, осуществляя попарные сравнения можно сократить количество элементарных операций до числа пикселов в маске.

Элементарный элемент для формирования сортировочной сети можно получить простейшей комбинацией двух мультиплексоров 2-в-1 и компаратора. В зависимости от результата сравнения на выходе элемента будет отсортированная в возрастающем порядке пара пикселов (рисунок 3.6).

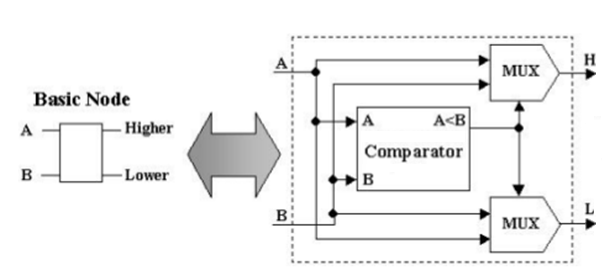


Рисунок 3.6 – Элемент сортировочной сети

Из представленных выше элементарных блоков можно составить сортировочную сеть. Структурная схема для размера маски 3 на 3 пиксела представлена на рисунке 3.7.

На выходе такой сети мы получаем отсортированный в порядке возрастания массив пикселов. Согласно идеологии медианного фильтра, мы должны выбрать центральный пиксел в отсортированном массиве и заменить им пиксел, который является центральным в маске.

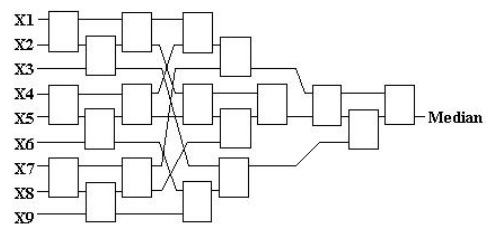


Рисунок 3.7 – Структурная схема сортировочной сети

Выполнять полноценную сортировку может быть не обязательной задачей в контексте решаемой проблемы, ведь необходимо в действительности определить только лишь центральный элемент. Однако вычисление порядкового номера пиксела, который должен находиться в центральной позиции, не позволяет отказаться от буферизации данных на время расчетов. Поэтому было решено организовать такую сортировочную сеть, чтобы обеспечить возможность дальнейших модификаций в алгоритме.

На выходе сортировочной сети необходимо всегда выбирать центральный элемент и подавать его на выход медианного фильтра, что и было реализовано на практике.

* 1. Фильтр Гаусса

Фильтр Гаусса представляет собой линейный неадаптивный фильтр, выходное значение которого есть линейная комбинация уровней яркости пикселов в окрестности центрального.

В отличие от медианного фильтра, где все операции представляли собой по сути логические операции и не требовался математический расчет, для фильтра Гаусса необходимо реализовать аппаратные умножение и деление.

Для упрощения алгоритма вычисления можно прибегнуть к замене дробных коэффициентов целыми числами, сумма которых будет кратна степени числа 2 для упрощения реализации операции деления. Часто в таком случае пользуются биномиальными коэффициентами. Например, для маски размером 3 на 3 пиксела биномиальные коэффициенты представляют последовательность {1, 2, 1}. Если из этой последовательности сформировать вектор, транспонировать его и перемножить с самим собой, то получается маска, где сумма всех коэффициентов равна 16. Число 16 является степенью числа 2, поэтому операцию деления можно заменить на операцию логического сдвига вправо на 4 разряда.

Таким образом расчет сводится к одиночным операциям умножения на целые числа, суммированию и сдвигу результата (рисунок 3.8).

Как было описано выше, идея биномиального фильтра достаточно проста. Его выход представляет собой линейную комбинацию значений входных пикселов. Другими словами, есть сумма исходных пикселов умноженных на биномиальные коэффициенты с последующей нормировкой (делением на сумму коэффициентов).

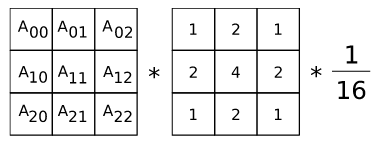


Рисунок 3.8 – Использование биномиальных коэффициентов в фильтре Гаусса

Также необходимо реализовать операцию суммирования. Сама по себе она является достаточно простой и эффективно реализуется с помощью внутренней логики ПЛИС, однако при увеличении разрядности пиксела и числа слагаемых, рабочая частота схемы может быть заметно снижена и увеличиться количество используемых ресурсов.

Для того, чтобы избежать проблем при дальнейшем масштабировании схемы и обеспечить детерминированную задержку на выполнение операции, реализуем ее в виде конвейер, где на каждом этапе будем параллельно попарно суммировать данные. Например, для сложения 9 пикселов потребуется 4 стадии. На первом этапе будут просуммированы пикселы под номерами 1-2, 3-4, 5-6, 7-8. Последний 9 пиксел сохраняется в неизменном виде для последующих операций. Тогда после первого этапа останется 5 операндов, после второго 3 операнда, четвертого 2 операнда и, наконец, результат самого суммирования. Так как организация конвейерная, то и новые данные для суммирования могут поступать на вход каждый такт. С задержкой 4 периода тактовой частоты на выходе схемы будет появляться результат суммы (рисунок 3.9).

Соответственно, с увеличением числа слагаемых необходимо увеличивать число элементарных сумматоров и регистров временного хранения. Такая реализация приведет к увеличению задержки на обработку, однако, рабочая частота схемы практически не будет зависеть от каких-либо иных параметров.

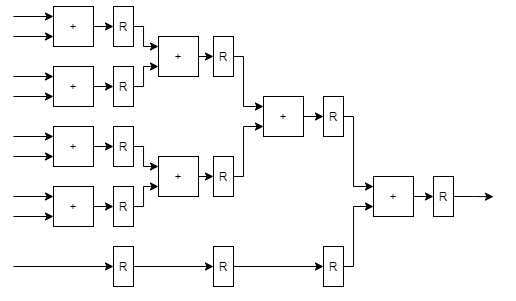


Рисунок 3.9 – Структура конвейерного сумматора

С учетом всех вышеперечисленных особенностей можно составить итоговую структурную схемы фильтра Гаусса, которая может быть реализована непосредственно в ПЛИС (рисунок 3.10).

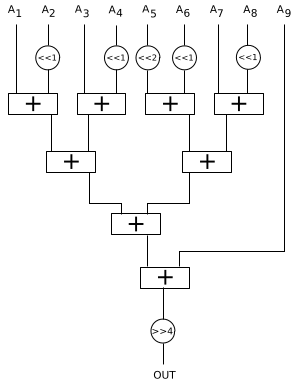


Рисунок 3.10 – Структурная схема фильтра Гаусса для размера маски 3х3

* 1. Усредняющий фильтр

В отличие от фильтра Гаусса нам не потребуется никаких операций умножения, так как требуется определить только среднеарифметическое значение. Для этого используем разработанный ранее конвейерный сумматор и специальный блок деления.

В общем случае операция деления достаточно ресурсоемкая и требует, зачастую, большую задержку на обработку, так как алгоритмы в основном итерационные, что не позволяет использовать его в конвейере обработки изображения.

Однако допустив, что размер маски фиксирован, можно существенно упростить операцию деления, заменив ее операцией умножения на обратную дробь, представленную в виде двоичного числа с фиксированной запятой. Например, для размера маски 3х3 необходимо разделить результат суммирования на 9 для нахождения среднего значения. Найдем обратную числу 9 дробь и переведем ее в двоичный формат с точностью 12 знаков после запятой. Так как яркость пиксела не может быть дробной, то результат умножения необходимо округлить до целого числа. Простейшим видом округления является отбрасывание дробной части, для этого отбросим первые 12 разрядов и полученный результат подадим на выход модуля (рисунок 3.11).

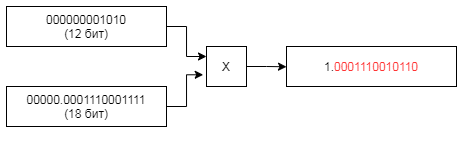


Рисунок 3.11 – Реализация операции деления

Как видно, исходное число имеет разрядность 12 бит, что соответствует заданной разрядности пиксела по ТЗ. Делитель был представлен в двоичном виде разрядностью 18 бит. Размерность операндов имеет значение для корректного распознавания средствами САПР и автоматической имплементации операций умножения с использованием DSP блоков. Дробную часть (младших 12 бит) мы отбрасываем, так как нас интересует только целая часть, и на выход модуля подаем старшие 12 бит, на примере выше – 1.

* 1. Оценка SNR по-черному

Для оценки уровня шума и качества работы алгоритмов шумоподавления зачастую используются такие показатели как PSNR, MSE. Однако для их расчета необходимо иметь опорный и не зашумленный кадр, которого в условиях анализа видеоряда просто нет. Тем не менее можно оценить уровень случайных шумов в каждом кадре, так как помимо основной информационной части кадра сенсоры передают различную служебную информацию. Одним из видов передаваемой основной информации является сигнал оптически черный, который соответствует закрытой части фотоматрицы и никакого накопления оптического сигнала в этой области не происходит, весь накопленный уровень является следствием воздействия шумов различной природы, в основном теплового шума.

Таким образом, используя эту информацию, можно оценить уровень теплового шума фотоматрицы, а также шумов преобразования и т.д. Также уровень черного можно вычесть из основного кадра, так он добавляется к каждому пикселу в изображении, следовательно, неизбежно искажает истинный уровень освещенности, соответствующий пикселу. Операция вычитания уровня черного называется «фиксации уровня черного» и зачастую выполняется видеопроцессором.

Мы же хотим спроектировать модуль, оценивающий не только уровень черного, но и SNR по этому участку после фильтрации для оценки качества шумоподавления реализуемых фильтров.

SNR есть отношение квадрата среднего значения интенсивности к среднеквадратическому отклонению от него (2).

, (2)

где Imean – среднее значение уровня яркости в области оптически черного сигнала; Ix,y – значение уровня яркости пиксела с координатами x и y.

Как видно из формулы, необходимо найти среднее значение интенсивности оптически черного участка изображения. Также необходимо найти среднеквадратическое отклонение от среднего значения. Структурная схема разрабатываемого модуля представлена на рисунке 3.12.

Для поиска среднего значения пикселов используем аппаратный блок DSP интегрированный в ПЛИС, а именно 64-битный аккумулятор, который является его составной частью.

Необходимо оценить, будет ли достаточно такой разрядности аккумулятора для выполнения задачи или необходимо использоваться несколько DSP блоков для достижения цели. В качестве ориентира для расчетов примем сенсор фирмы Sony IMX265. У него оптически черное поле представляет собой непрерывную область высотой 10 строк и шириной строки с учетом поля для постобработки 1936 пикселов. Соответственно для того, чтобы избежать переполнения аккумулятора, его разрядность должна быть выше разрядности суммы всех пикселов изображения. Реальные же значения в процессе работы будут скорее ближе к нижней границе яркости, однако, мы обязаны предусмотреть самый худший вариант, при котором яркость всех пикселов будет максимальной.

Рассчитаем необходимую разрядность аккумулятора. Она равна разрядности пиксела, умноженной на общее число пикселов, и должна быть округлена в большую сторону. Таким образом разрядность аккумулятора должна быть не меньше 27, что с большим запасом удовлетворяет имеющемуся в DSP блоке 64-битному аккумулятору.

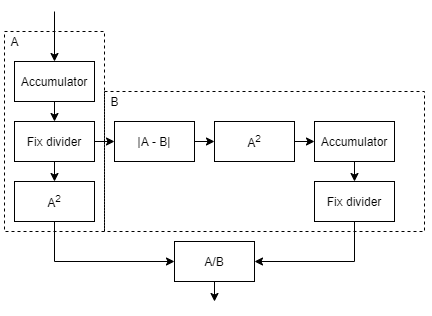


Рисунок 3.12 – Структурная схема измерителя SNR

Строго говоря, значение средней яркости и СКО должно быть рассчитано для каждого кадра. Однако мат. ожидание практически не меняется от кадра к кадру, что позволяет существенно упростить обработку и вычислить СКО на основе мат. ожидания для предыдущего кадра. Таким образом нет необходимости в буферизации всего участка изображения во время вычисления м.о. и разделить структурную схему на 2 сегмента, которые семантически разделяют обработку текущего (Б) и предыдущего кадра (А).

Для нахождения среднеарифметического значения значение аккумулятора необходимо разделить на число слагаемых, однако, их число будет фиксировано для конкретного сенсора, что позволяет использовать уже разработанный модуль деления на фиксированное число (см. рисунок 3.11).

Для нахождения СКО необходимо найти разницу между средним значением и значением текущего пиксела для чего используем режим сумматора у DSP в режиме «вычитания», а также сразу после этого необходимо возвести в квадрат полученный результат и сохранить его в аккумуляторе.

Так как имеет место операция умножения, то и разрядность результата должна быть также учтена. При умножении 12-битных чисел результат будет 24-битный. Однако умножители, как мы указывали, будут работать в 18-битном режиме, следовательно, их произведение будет на самом деле 36 битным. Тем не менее, чуть меньше половины старших разрядов всегда будут пустыми, а само значение разности, возводимое в квадрат, зачастую будет иметь малое число значащих бит, так как из среднего вычитается близкие к нему значения. В случае больших выбросов числа будут больше, но все же их величина ограничена и, следовательно, тех же 64 бит в аккумуляторе на практике хватит с большим запасом.

Единственным модулем, который необходимо разработать специально для реализации этой части является аппаратное деление произвольных чисел.

Скорость работы этого блока не является критичной для задачи, ввиду того, что вычисление частного необходимо только 1 раз на каждый кадр. Поэтому модуль может осуществлять операцию деления параллельно с вычислениями и накоплением данных текущего кадра.

В регистр «dividend» на этапе загрузки защелкивается значение делимого, а в регистр «divisor» значение делителя. На каждом этапе из делимого вычитается значение делителя, а счетчик считает количество операций вычитания. Значение разности записывается обратно в регистр «dividend» и весь цикл повторяется заново. Когда значение делимого становится меньше значения делителя, текущее значение счетчика является целой частью от деления числа A на число B. Структурная схема описанного блока представлена на рисунке 3.13.

Число тактов, необходимое на обработку каждого конкретного запроса зависит от величины загруженных значений, поэтому необходимо также предусмотреть сигнал-индикатор окончания расчета. По окончанию вычислений результат записывается в выходной регистр и устанавливается сигнал «valid» до подтверждения приема данных следующим блоком.

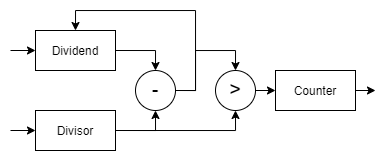


Рисунок 3.13 – Структурная схема модуля деления