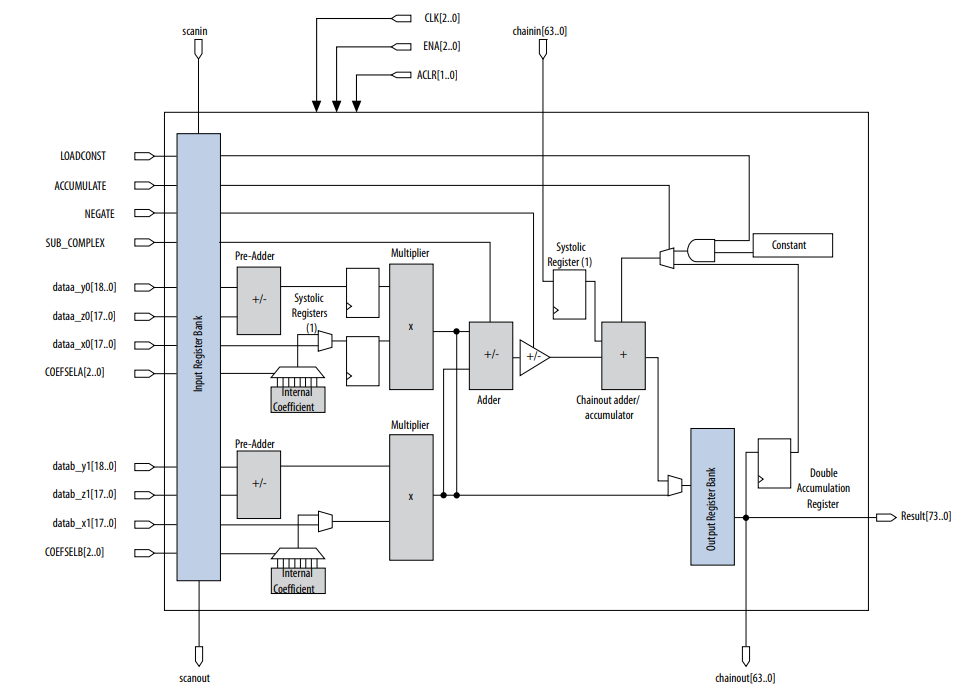
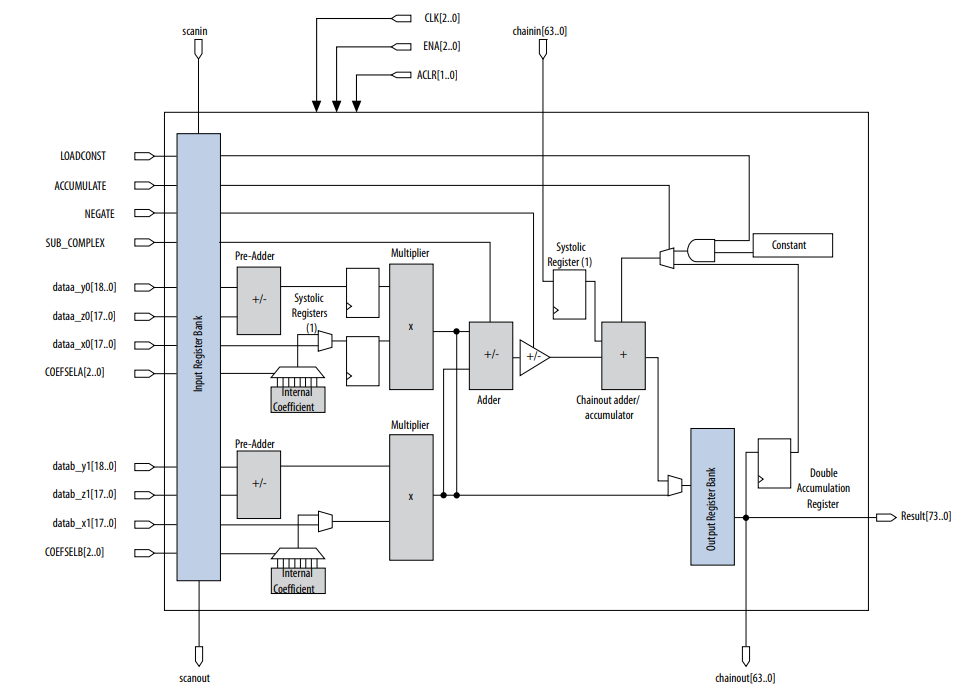
3. Реализация алгоритмов в ПЛИС

3.1. Аппаратные блоки ПЛИС, необходимые для реализации алгоритмов





3.2. Скользящее окно

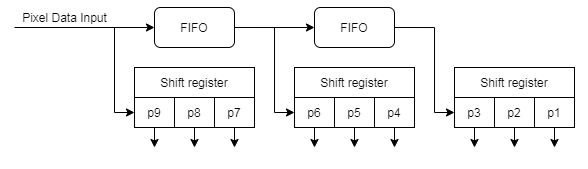
Для всех выбранных алгоритмов необходимо сформировать скользящее по изображению окно. Центр этого окна должен последовательно сдвигаться на 1 пиксел вдоль строки, а затем смещаться на 1 пиксел по строке. Для формирования скользящего окна необходимо накопить число строк равное размеру маски в общем случае. 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.



*Рис.1. Демонстрация скользящего окна*

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 такого блока будет выглядеть следующим образом (рис.1.).



*Рис.1. Структурная схема скользящего окна*

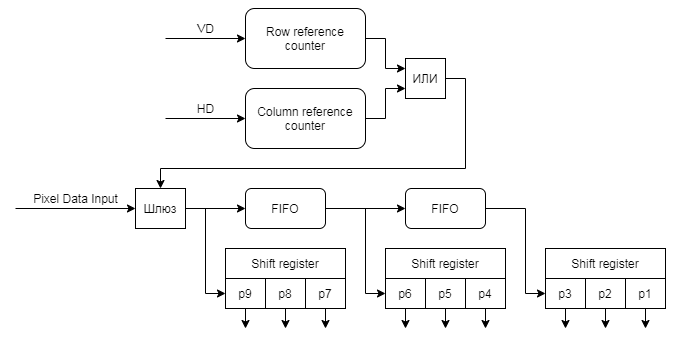
Выходы сдвиговых регистров являются элементами скользящего окна. Текущая строка параллельно с выводом в сдвиговый регистр записывается в блок памяти, организованный по принципу FIFO, таким образом осуществляется временная буферизация строки. При таком подходе, накопив 2 строки, начнется процесс вывода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

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 2.



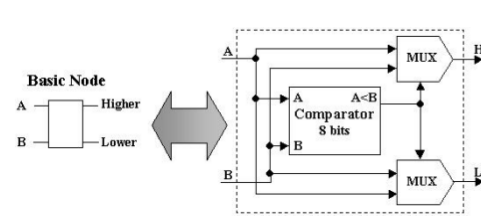
*Рис.2. Структурная схема скользящего окна со шлюзом разрешения*

Таким образом можно принять видеопоток и сформировать из него необходимую маску.

3.3. Медианный фильтр

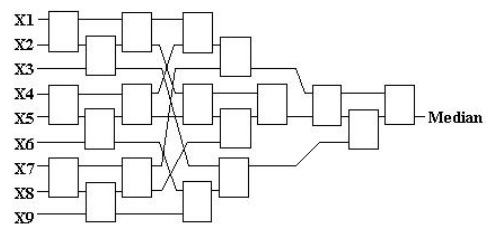
Обработка видеопотока на ПЛИС отличается от классических подходов при обработке на CPU тем, что можно выполнять ряд операций параллельно. Допустим, для сортировки пикселов в маске на CPU, требуется последовательно каждый пиксель сравнивать и перемещать, а на ПЛИС можно выполнять на каждом этапе попарные сравнения пикселов. Таким образом осуществляя попарные сравнения можно сократить до количество операций до пикселов в маске. Допустим для маски размером 3 на 3 число пикселов 9, и в худшем случае, если каждый пикселы будут распределены ровно в обратном порядке, тогда за 9 попарных сравнений и перемещений все окажутся отсортированными.

Элементарным попарным сравнивателем с перемещением представлен на рисунке 3.



*Рис.3. Элементарный преобразователь и сравниватель*

А сама сортировочная сеть выглядит так на рисунке 4.

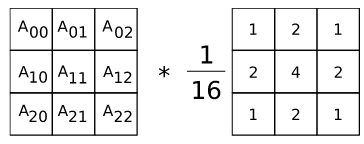


*Рис.4. Сортировочная сеть*

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

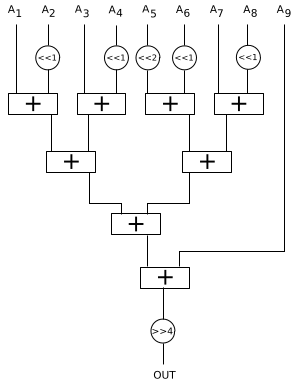
3.4. Биномиальный фильтр

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).



*Рис.5. Умножение матрицы коэффициентов на маску пикселов*

Часто для операций умножения и деления на степень 2 используются логические сдвиги. Потому округление коэффициентов до степени 2 позволит существенно выиграть в сложности реализации, сведя все к простым действиям.



*Рис.6. Реализация фильтрации для маски 3х3*

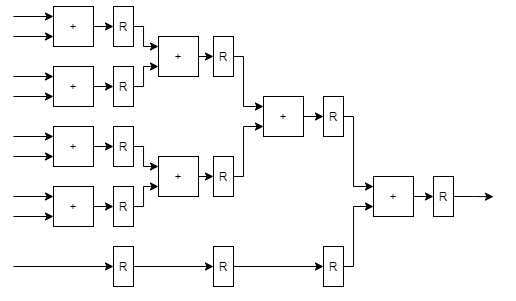
Однако далеко не всегда возможно округлять коэффициенты, для реализации более точных вычислений, возможно, следует выполнять их даже в виде вычислений с плавающей запятой. Тогда округление и замена операций умножения и деления должны быть сохранены. Для таких целей в ПЛИС есть аппаратные интегрированные блоки DSP.

Так как разрабатываемый фильтр должен иметь возможность масштабирования, то умножение на коэффициенты будут выполнять на DSP. А деление ввиду кратности степени 2 (для нормировки) путем сдвига.

Для дальнейшей реализации алгоритмов не раз потребуется складываться большое число пикселов. Сама по себе операция не является сложной, однако при параллельном сложении нескольких чисел большой разрядности может существенно снижаться рабочая частота схемы. Для этого реализуем конвейерное суммирование слагаемых. На каждом этапе параллельно будем попарно суммировать полученные данные. Таким образом, имея на входе при маске 3х3 - 9 пикселов, на первом этапе будут просуммированы 1-2, 3-4, 5-6, 7-8 пикселы. А последний 9 пиксел останется сохраненным в неизменном виде для последующих операций. Тогда после первого этапа останется 5 операндов, после второго 3 операнда, четвертого 2 и наконец результат самого суммирования. Так как организация конвейерная, то и новые данные для суммирования могут поступать на вход каждый такт. С задержкой равной округленной до ближайшего целого логарифма по основанию 2 числа входных данных, на выходе схемы будет результат суммы.

Соответственно при увеличении числа слагаемых необходимо увеличить число элементарных сумматоров и регистров временного хранения.

Структурная схема такого сумматора представлена на рисунке 3.



*Рис.3. Конвейерный сумматор для 9 слагаемых*

3.5. Усредняющий фильтр.

В отличие от биномиального фильтра нам не потребуется никаких умножений на коэффициенты, так как нам нужно лишь среднеарифметическое значение. Для этого нам понадобится также описанный конвейерный сумматор и блок деления на размер маски.

В общем случае операция деления достаточно ресурсозатратная 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 с фиксированной запятой.

Любое число можно представить в виде целой и дробной части. Для привычной нам десятеричной системы счисления дробная часть показывает сколько в числе десятых, сотых, тысячных долей. Для двоичной системы ситуация очень похожа, однако отличается трактовка разрядов.

Допустим, мы хотим представить число 4,375 в видео двоичного числа с фиксированной запятой, тогда целая часть будет представлять последовательность «100», где каждый разряд отвечает за степень 2 в итоговом числе:

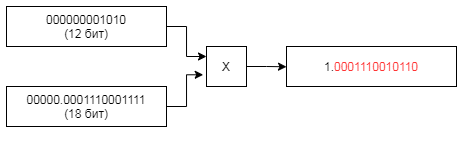
* 1й разряд – 2^0;
* 2й разряд – 2^1;
* 3й разряд – 2^2;
* …

Трактовка разрядов после запятой есть отрицательные степени 2:

* 1й разряд – 2^(-1);
* 2й разряд – 2^(-2);
* 3й разряд – 2^(-3);
* …

Тогда для представления дробной части 0,375 необходимо представить ее в виде последовательности «011».

Таким образом, с принятой точностью деление можно заменить на умножение на обратное число. Допустим, для размера маски 3х3 нам понадобится деление результата на 9. Но мы умножим его на 1/9, переведя его в двоичный формат с точностью до 12 знаков после запятой. Тогда и результат умножения будем брать, начиная с 13 разряда, который соответствует целому числу, ведь уровень пикселя не может быть дробным. Просто будем отбрасывать дробную часть. Например, разделим таким способом десятичное число 10 на 9 (иллюстрация на рисунке 9).



*Рис.9. Деление замененное на операцию умножения*

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 корректной имплементации алгоритма в ПЛИС с последующим распознаванием поведения схемы для автоматического использования DSP. Дробную часть мы отбрасываем, так как она нас не интересует, берем целую часть – 1.

Реализация данного фильтра представляет собой последовательное соединение конвейерного сумматора, разработанного ранее, и разработанного константного делителя.

3.6. Оценка SNR по черному

Так как для оценки качества шумоподавления зачастую используются такие показатели как PSNR, MSE. Однако для этого необходимо иметь опорный кадр, которого в наших условиях нет. Однако фотоприемники помимо основной информации кадра еще передают сигнал оптически черный, который соответствует закрытой части фотоприемника и никакого накопления оптического сигнала в этой области не происходит.

Эти данные используются для оценки уровня тепловых шумов самого фотоприемника, а также шумов преобразования и т.д. Эту информацию можно использовать и для вычитания из основного кадра. Если присутствует ненулевой уровень черного сигнала, то он неизбежно также добавляется и к каждому пикселу в изображении, следовательно искажается истинный уровень освещенности соответствующий пикселу. Его вычитание называется операцией «фиксации черного» и зачастую выполняется видеопроцессором.

Мы же будем 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.