Note that the weights for an edge detection filter should sum to zero, so that the response is zero for regions of uniform pixel value. For edge detection, the weights can be multiplied by an arbitrary constant, since the edges correspond either to the positions of the maximum derivatives, or the locations of the zero crossings of the Laplacian.

8.2.3 Edge Enhancement

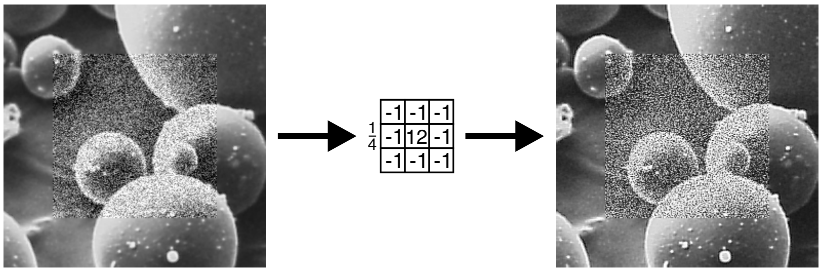
An edge enhancement filter sharpens edges by increasing the gradient at edges. In this sense it is the opposite of edge blurring, where the gradient is decreased by attenuating the high frequency content. Edge enhancement works by boosting the high frequency content of the image. One such filter is shown in Figure 8.13. A major limitation of the high frequency gain is that any noise within the image will also be amplified. This cannot be easily avoided, so linear edge enhancement filters should only be applied to relatively noise-free images. Over-enhancement of edges can result in ringing, which will become more severe as the enhancement is increased.

The weights of an edge enhancement filter must sum to one to avoid changing the global contrast of the image.

8.2.4 Linear Filter Techniques

Although most of the examples shown in this section are 3×3 filters, the techniques readily extend to larger filter sizes. The obvious implementation of a linear filter is illustrated in Figure 8.14, especially if the FPGA has plentiful multiplication or DSP blocks. Each pixel within the window is multiplied in parallel by the corresponding filter weight and then added. Note that the bottom right window position is the oldest pixel and corresponds to the top left pixel within the image in the image. The filter weights are shown here as constants, but could also be programmable and stored in a set of registers.

The propagation delay through the multiplication and adders may exceed the system clock cycle and require pipelining. Since each input pixel contributes to several pixels, rather than delaying the inputs and accumulating the output, the transpose filter structure performs all the multiplication with the input and delays the product terms (Mitra, 1998). The transpose filter structure does this by feeding data through the filter backwards (swapping the input and the output) and swapping summing junctions and pick-off points. This is applied to each row in Figure 8.15, and to the whole window (Benkrid *et al.*, 2003a) in Figure 8.16. The advantage of the transposed structures is that the output is automatically pipelined. When transposing the whole window, it is necessary to cache the partial sums using row buffers until the remainder of the window appears. Depending on the filter coefficients, these partial sums may require a



$\frac{1}{4}\begin{array}{|c|c|c|}\hline -1 & -1 & -1 \\ \hline -1 & 12 & -1 \\ \hline -1 & -1 & -1 \\ \hline\end{array}$

Figure 8.13 Linear edge enhancement filter.

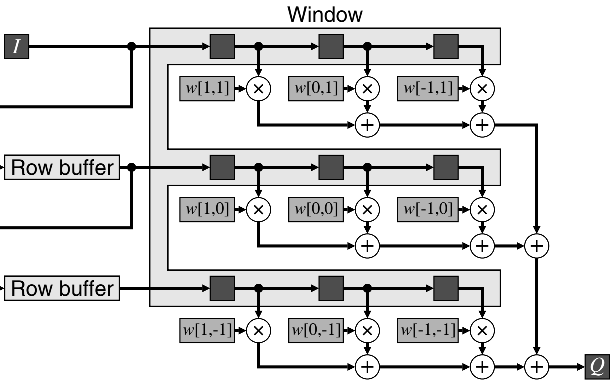


Figure 8.14 Direct implementation of linear filtering.

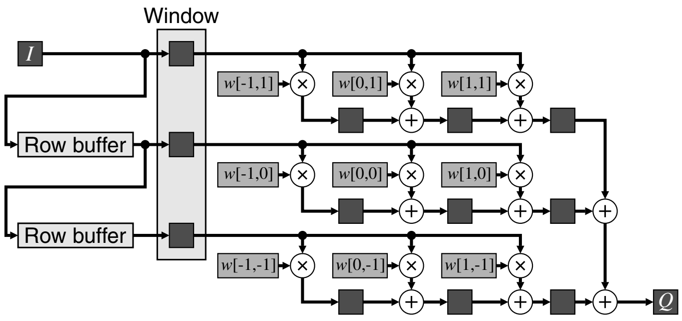


Figure 8.15 Pipelined linear filter. Note the different order of coefficients to Figure 8.14 because the transpose filter structure is used for each row.

few guard bits to prevent accumulation of rounding errors. The row buffers, therefore, may need to be a few bits wider than the input pixel stream. The transposed filter structures may also require a little more effort to manage the image boundaries, although that can also be incorporated into the filter structure (Benkrid *et al.*, 2003a).

There are several techniques that may be used to simplify and reduce the logic required by linear filters. Many filters are symmetric, therefore many of their filter coefficients share the same value. If using the direct implementation of Figure 8.14, the corresponding input pixel values can be added prior to the multiplication. Alternatively, since each input pixel is multiplied by a number of different coefficients, the input pixel value can be multiplied by each unique coefficient once, with the results cached until needed (Porikli, 2008). This fits well with the transposed implementation of Figure 8.16. In both cases, the number of multipliers is reduced to the number of unique coefficients.

If using a relatively slow clock rate, and hardware is at a premium, then significant hardware savings can often be made by doubling or tripling the clock rate but keeping the same data throughput. The result is a multiphase design which enables expensive hardware (multipliers in the case of linear filters) to be reused in different phases of the clock. This is shown for a three-phase system in Figure 8.17. With the accumulator, a 0 is fed back in phase zero to begin the accumulation for a new pixel.

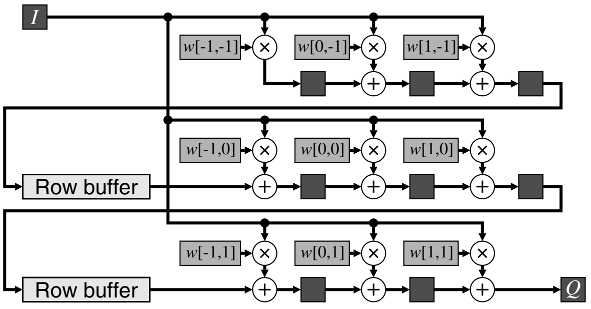


Figure 8.16 Transposed implementation of linear filtering. Note the changed coefficient order.

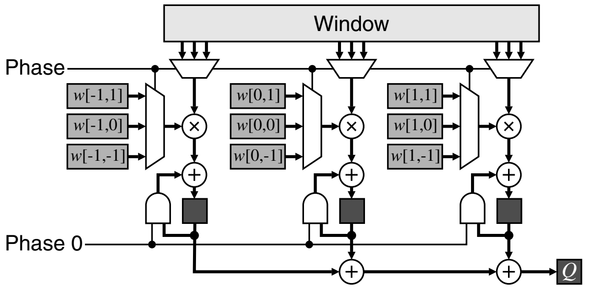


Figure 8.17 Reducing the number of multipliers by using a higher clock speed.

Many useful filters are separable:

$$w[i,j] = w_x[i]w_y[j] \tag{8.11}$$

This means that a two-dimensional filter may be decomposed into a cascade of two one-dimensional filters: a $1 \times W$ filter operating on the columns and a $W \times 1$ filter operating on the rows. This will reduce the number of both multiplications and associated additions from W^2 to $2W$. Note that the column filter does not require a separate pass through the image. It, too, can be streamed by replacing the pixel delays within the window by row buffers, as shown in Figure 8.18. This is effectively implementing the filters

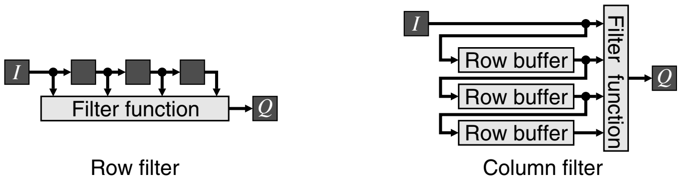


Figure 8.18 Converting a row filter to a column filter by replacing pixel delays with row buffers.

for each column in parallel, but sequentially stepping the filter function from one column to the next as the data is streamed in. Consequently, the row and column filters may be directly pipelined.

While not all filters are directly separable, even arbitrary two-dimensional filters may be decomposed as a sum of separable filters (Lu *et al.*, 1990; Bouganis *et al.*, 2005)

$$w[i,j] = \sum_k w_{kx}[i] w_{ky}[j] \tag{8.12}$$

through singular value decomposition of the filter kernel. The vectors associated with the k largest (significant) singular values will account for most of the information within the filter. With such a decomposition, the column filters should be implemented before the row filters to enable a single set of row buffers to be used for all the parallel filters. For a similar reason, the transpose filter structure cannot be used for the column filters, although it may be used for the row filters. Such a decomposition can give savings if $k \leq W/2$, which will be the case if the original filter is symmetrical either horizontally or vertically.

Also worth considering are series decompositions of filters. The kernel of a composite filter is the convolution of the kernels of the constituent filters:

$$\begin{aligned} w[i,j] &= w_1[i,j] \otimes w_2[i,j] \\ &= \sum_{x,y} w_1[x,y] w_2[i-x, j-y] \end{aligned} \tag{8.13}$$

A common example of this is approximating a Gaussian filter by repeated application of a rectangular box filter. For k repetitions of a window of width W, the standard deviation of the resultant approximate Gaussian is:

$$\sigma = \sqrt{\frac{1}{12} k (W^2 - 1)} \tag{8.14}$$

For many applications, three or four repetitions will provide a sufficiently close approximation to a Gaussian filter.

Of course, when implementing constant coefficient filters, if any of the coefficients is a power of two, the corresponding multiplication is a trivial shift of the corresponding pixel value. Such shifts can be implemented without logic. If not using multiplier blocks, the multiplications may be implemented with shift and add algorithms. For any given coefficient, the smallest number of shifts and adds (or subtracts) is obtained by using the canonical signed digit representation for the coefficients. Further optimisations may be obtained by identifying common subexpressions and factorising them out (Hartley, 1991). For example, $165 = 101\ 001\ 01_2$ requires three adders for a canonical signed digit multiplication, but only two by factorising as 5×33 ($= 101_2 \times 1\ 000\ 01_2$). A range of techniques has been developed to find the optimal (in some sense) factorisation and arrangement of terms (Dempster and Macleod, 1994; Potkonjak *et al.*, 1996; Martinez-Peiro *et al.*, 2002; Al-Hasani *et al.*, 2011).

A simple example will illustrate the power of these techniques. Consider a 21×21 Gaussian smoothing filter with $\sigma = 3$. A direct implementation would require 441 multipliers and 440 adders. Representing the coefficients with fixed-point numbers with resolution of 2^{-16} has the coefficients in the range from 0 to 1160, requiring 11 bits for the central few coefficients. Of the coefficients, 40 of them are zero, so do not need to be multiplied or added in. This reduces the number of adders to 400. Of the

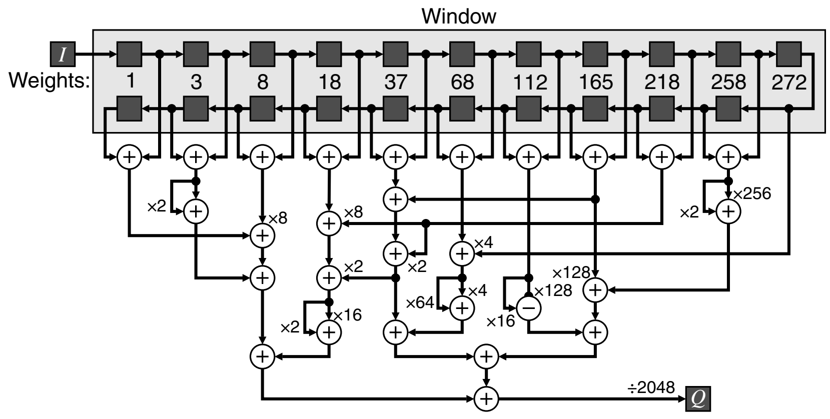


Figure 8.19 Adder only implementation of a 21 × 1 Gaussian filter with $\sigma = 3$.

coefficients, 96 are direct powers of two (1, 2, 4 and 8); these also will not require multipliers but will still require adders. Since the filter is symmetric, most of the coefficients will be used four or eight times. This can be exploited by adding the corresponding pixel values first and then multiplying by the coefficient. Since there are 40 unique coefficients (not counting the zero or powers of two), this reduces the number of multipliers to 40, although 400 adders are still required. A further 10 of the coefficients can be represented as the sum or difference of two powers of two. This allows those multiplications to be performed by a single addition rather than a multiplication. The number of operations is then 30 multiplications and 410 additions.

A further observation is that the Gaussian filter is separable. This allows the 21 × 21 filter to be implemented as cascade one-dimensional Gaussian filters (1 × 21 and 21 × 1). The fixed-point coefficients with a resolution of 2^{-11} are shown along the top of Figure 8.19. Each one-dimensional filter would require nine multiplications (because of symmetry and two coefficients are powers of two) and 20 additions. A further six coefficients can be represented as a sum or difference of two powers of two, reducing the requirements to three multiplications and 25 additions (one addition can be removed through common subexpression elimination: $272 = 4 \times 68$). The remaining three multiplications can also be eliminated by reusing common subexpressions, in the form of a shifted sum of two existing coefficients ($37 = 2 \times 18 + 1$; $165 = 37 + 128$; $218 = 8 \times 18 + 2 \times 37$) allowing even those multipliers to be replaced with single additions. Therefore, the complete 21 × 1 filter can be implemented with only 28 additions, as demonstrated in Figure 8.19. A similar filter can be used for the vertical filter, replacing each the register in the chain across the top with a row buffer.

With the increasing prevalence of high speed pipelined multipliers within FPGAs, the need for such decompositions has diminished somewhat in the last few years. The optimised hardware multipliers are hard to out-perform with the relatively slow adder logic of the FPGA fabric (Zoss *et al.*, 2011).

Other decompositions and approximations are also possible, especially for large windows. For example, Kawada and Maruyama (2007) approximate large circularly symmetric filters by octagons and rely on the fact that the pixels with a common coefficient lie on a set of lines. As the window scans, the total for each octagon edge is updated to reflect the changes from one window position to the next.

A simpler version of this can be applied to averaging using a rectangular window with equal weights (McDonnell, 1981). Firstly, a large rectangular window is separable, allowing the filter to be implemented as a cascade of two one-dimensional filters. Secondly, the equal weights make the filter

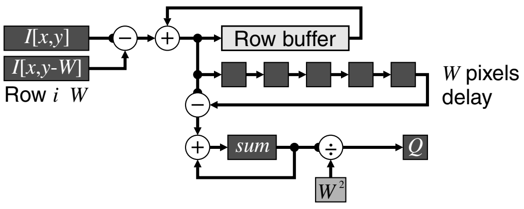


Figure 8.20 Efficient implementation of a $W \times W$ box average filter.

amenable to a recursive implementation:

$$\begin{aligned} S[x+1] &= \sum_{i=-\frac{W-1}{2}}^{\frac{W-1}{2}} I[x+1+i] \\ &= \sum_{i=-\frac{W-1}{2}}^{\frac{W-1}{2}} I[x+i] + I\left[x+1+\tfrac{W-1}{2}\right] - I\left[x-\tfrac{W-1}{2}\right] \\ &= S[x] + I\left[x+\tfrac{W+1}{2}\right] - I\left[x+\tfrac{W+1}{2}-W\right] \end{aligned} \tag{8.15}$$

which reduces the one-dimensional filter to a single addition and subtraction regardless of the size of the window. The same recursive mechanism may also be used for the column filter. An input pixel is then only required twice, once when it first enters the window and again when it leaves the window. For large windows, a large number of row buffers are required to cache the input for its second access. Therefore, if the input stream is from a frame buffer which has sufficient bandwidth to stream from two rows simultaneously, the row buffering requirements can be reduced significantly, with only the current column sum buffered. Such an implementation for a $W \times W$ box average is illustrated in Figure 8.20.

8.3 Nonlinear Filters

While the theory behind linear filters is well established and the filters are relatively simple to implement, restricting f in Equation 8.1 to a linear combination of the input values has some significant limitations (Bailey *et al.*, 1984). In particular, the filters have difficulty distinguishing between legitimate changes in pixel value (for example at an edge) from undesirable changes resulting from noise. Consequently, linear filters will blur edges while attempting to reduce noise, or be sensitive to noise while detecting edges or lines within the image.

Linear filters may be modified to improve their characteristics, for example by making the filter weights adaptive, or dependent on the image content. The simplest of these are *trimmed filters*, which omit some pixels within the window from the calculation. While the mean possesses good noise reduction properties, it is sensitive to outliers. A trimmed mean will discard the outliers by discarding the extreme pixels and calculate the mean of the remainder (Bednar and Watt, 1984). With a heavy tailed noise distribution, this can give an improvement in signal-to-noise ratio. A related application is smoothing noise without significantly blurring edges; it is desirable to average the pixel values only from one side of the edge. By selecting only the pixel values that are similar to the central pixel value, the probability of using pixel values on the opposite side of the edge is reduced, with a corresponding reduction in blur.