Area-Efficient 2-D Shift-Variant Convolvers for FPGA-based Digital Image Processing

Francisco Cardells-Tormo, and Pep-Lluis Molinet
Digital ASIC Technology Group
Hewlett-Packard, R&D Large-Format Technology Lab
08147 Sant Cugat del Valles, Spain
{francisco.cardells,pep-lluis.molinet}@hp.com

Abstract□ **Two dimensional (2-D) convolutions are local by nature; hence every pixel in the output image is computed using surrounding information, i.e. a moving window of pixels. Although the operation is simple, the hardware is conditioned by the fact that due to bandwidth efficiency full raster rows must be read from the external memory, and that a row-major image scan should be performed to support shift-variant convolutions. When extending the architectures developed in prior art to support shift-variant convolutions, we realize that they require large amounts of on-chip memory. While this fact may not have a large cost increase in ASIC implementations, it makes FPGA implementations expensive or not feasible. In this paper, we propose several novel FPGA-efficient architectures for generating a moving window over a row-wise print path. Because the proposed concepts have different throughput and resource utilization, we provide a criteria to choose the optimum one for any design point.**

I. INTRODUCTION

Most image processing algorithms are local and two-dimensional (2-D) by nature. This implies that the output is computed using an NxN neighborhood of pixels of the input image around every pixel of interest, hence the result is obtained by means of a 2-D convolution. In general, the 2-D convolution is the process of multiplying every pixel within the aforementioned neighborhood with some other function, commonly referred as the convolution kernel, i.e. the 2-D filter impulse response. This is true for both algorithms that are applied to continuous-tone (contone) images, such as edge detection, image sharpening/smoothing, and those applied to halftone images (binary images used in binary displays and printers), such as depletion [1] and fortification [2]. The only differences between contone and halftone convolutions reside in the kernel size N, typically smaller for contone images, and the number of bits per pixel and per color plane (bpp) of the input images. Typical values are 8 bpp and 1 bpp for contone and halftone images respectively.

In embedded systems such as consumer printers with a high pressure in costs, image processing algorithms are usually implemented in hardware. Hardware implementations of image processing pipelines make use of a large external memory (e.g. DDR SDRAM) to store the input image and the intermediate results. Other storage alternatives such as internal SRAMs are ruled out due to the silicon cost. Image data in DRAM memories cannot be accessed at will and certain rules for data retrieval must be followed. Image color planes are stored in separate locations in memory, and each memory address stores contiguous pixels within the same row. A row is stored in consecutive addresses as shown in figure 1. Image data is accessed (read or written) in the form of raster micro-rows, each consisting of a number - given by the burst length - of concatenated memory bus words, thus spanning several columns of pixels. For example in figure 1 the burst length is given by four memory words. Each memory word represents two pixels, and therefore 8 pixels from the same row is the minimum amount to be accessed in one transaction.

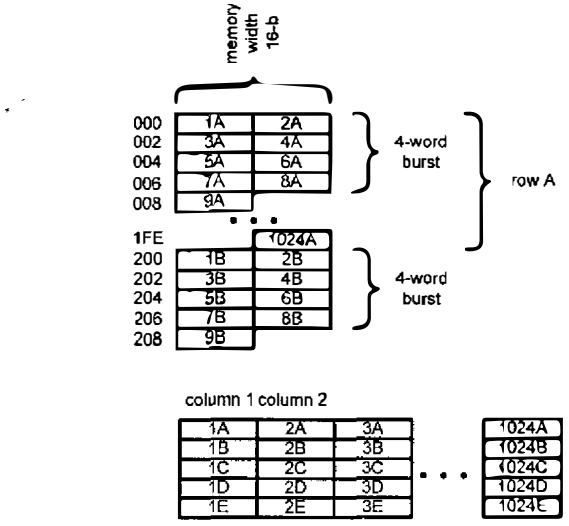

Figure 1. The figure shows the way a digital image (shown at the bottom) is mapped in memory addresses (shown at the top) for a 16-bit memory width, and 8-bit pixels. Pixels are represented with a letter from A to E to show the corresponding raster row and with a number from 1 to 8 to show the corresponding column

Several raster micro-rows (s) below and above the pixel of interest must be read from memory before being able to process that pixel, we will refer to this area within the image as a tile. The kernel size is related to s as follows: N=2s+1. Because we will obtain many columns at the same time and we should avoid reading the same columns more than once, an intermediate tile buffer implemented by means of an on-chip memory (SRAM) is necessary. For the example shown in figure 1 and s=1, we must read rows A, B and C and three columns (1-3), yet for each access we will get 8 columns.

The 2-D convolver architectures that we find in the literature [3], make use of a linear shift register to move an NxN window over the tile. This architecture is based on the fact that the image is partitioned in micro-bands, each with a fixed width corresponding to the micro-band row length, and a height corresponding to the image height. Partitioning the page width in micro-bands is necessary to make [3] work in FPGAs. For instance for a large-format printer (28□ page width), with an image resolution of 600 dots-per-inch (dpi), and a convolution window of 5x5, the shift register would mean 656k flip-flops or bits of on-chip memory. If a low-cost FPGA is the target platform to implement the digital imaging pipeline, 656Kb of embedded memory represent 66% of the total memory of the largest Altera Cyclone-II device (EP2C70) [4] and 35% of the largest Xilinx Spartan-3 device (XC3S5000) [5], both the low-cost commercial FPGAs manufactured in the 90nm technology provided by the main vendors

Micro-band rows are read from the external memory from top to the bottom and loaded into the tile buffer. Thus, each time a new raster row is loaded, the tile moves along the micro-band in the vertical direction. The linear shift register pushes the moving window from left to right over the micro-band row. By combining this two effects, the pixel of interest moves first from left to right for each micro-band row and starting on top of the current band, until all micro-band rows have been consumed. Then, the pixel of interest moves to the top of the next band. The output image pixels are generated following the processing path we have described. The throughput is one clock per pixel.

Many image processing algorithms such as those mentioned in the first paragraph of this introduction are, with regard to the convolution between the moving window and the programmable kernel, shift-invariant, and therefore the result of the convolution does not depend on the order pixels are processed (i.e., the processing path). For those algorithms, the hardware architecture described in [3] is valid and the zig-zag scan path is suitable. Yet there are many other algorithms that are shift-variant, such as those based on dithering. For instance, dithering-like algorithms, such as error-diffusion [6], require processing a whole image row, instead of only the part corresponding to the micro-band width. In the shift-variant cases, the processing path cannot be divided in micro-bands. Therefore, the brute-force approach of designing a 2-D convolver based in [3] and extending the tile width to the whole image width would have dramatic implications in terms of memory requirements. For an ASIC implementation using the 90nm technology, and provided that we assume a memory cost model of $1/Mbit, this amount of memory represents the negligible cost of $0.64. Yet, for an FPGA implementation using the same technology we have shown above the tremendous amount of resources required.

In this paper we are concerned with the implementation of 2-D convolvers in FPGA and therefore we will investigate several alternative moving window architectures for shift-variant 2-D convolutions that use fewer resources than [3] and are therefore more suitable for an FPGA implementation. For each architecture, we will report the throughput, and hardware requirements in terms of memory and flip-flop count. We will conclude the paper by comparing the architectures in terms of cost per performance between them, so the optimum scheme can be selected for any design point. Due to the fact that this problem has not been previously addressed in prior art, the architectures described in this paper are novel and the comparison will only include the technique presented in [3].

II. Moving Window Architectures

In this section we will use some of the reference data presented in the introduction as a case study, i.e. a page width of 28□at a printing resolution of 600 dpi, a contone image with 8 bpp. In addition to this we will consider a memory bus word length of 64-bits and a burst length of 4 words (i.e., 8 pixels). In our figures we have depicted an internal memory storing 5 raster rows (5x5 moving window). Memory contents is represented with a letter from A to E to show the corresponding raster row and with a number from 1 to 32 to show the corresponding column within the burst.

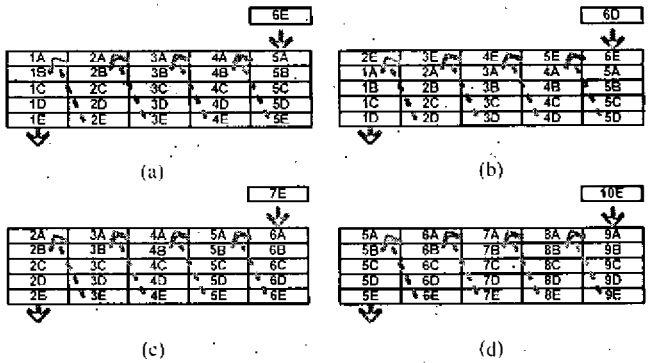

Figure 2. Shift-register contents of the column-major convolver. The figure shows the shift-register contents for the initial state (a), one cycle after the initial state (b), (2s+1) cycles after the initial state (c) and for a state where the content of two consecutive tiles is used in the computation (d)

A *Column-Major Moving Window (Architecture I)*

The first concept we propose is a modification of [3]. It consists in reading a tile from the external memory that is row-wise, storing it in a local memory and transferring the local memory contents to a shift register in a column-major

format as shown in fig. 2. The column height is coincident with the window height N. The pixels feed a shift register until a full window is loaded after *(2s+1)x(2s+1)* clock cycles as shown in figure 2.a.

Once the shift register is in its initial state, the following clock cycle, shown in figure 2.b, the following column starts to be transferred. This way the moving window is shifted to the right one position every *(2s+1)* clock cycles as depicted in fig 2.c. We remind that s is the amount of rows above and below the pixel of interest. While columns of one tile are being transferred to the shift register, another should be stored in the local memory for bandwidth efficiency, thus forming a swing buffer.

The hardware architecture for this concept has been depicted in fig. 3. The hard-ware requirements consist of a swing buffer of on-chip SRAM for two tiles (a dual-port memory of 2.5Kb for the case study), *(2s+1)x(2s+1)* pixel registers for the moving window register array (200 FFs for the case study), plus a memory bus wide register (64 FFs for the case study), plus a multiplexer that selects a column within a raster row word (a 8:1 mux for the case study). Due to the fact that *2s+1* idle clocks are needed per pixel to shift the moving window one single column, the throughput becomes *2s+1* clocks/pixel and for this case study it is translated to 5 clocks/pixel.

One negative side-effect of this architecture is that in order to process a single row of the image, *2s+1* rows must be read from the external memory, therefore there is an important overhead that leads to an elevated memory bus bandwidth requirement by this block.

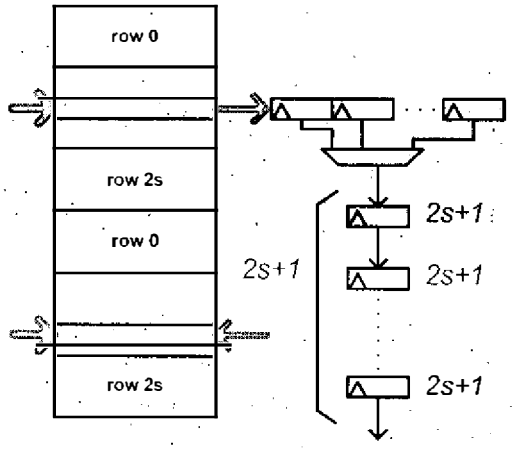

Figure 3. Architecture I. Column-major moving window architecture.

B. *Row-Major Moving Window (Architecture II)*

This concept consists in reading a tile from the external memory, storing it in an internal memory, and transferring a word, of wordlength equal to the memory bus width or w pixels, in each of the local memory rows to an array of shift registers. The resulting moving window is depicted in fig. 4. The array will be shifted one position to the left each clock cycle until the whole word has been consumed.

Due to the fact that s additional pixels are needed at both sides of the word, the system is a little bit more complex and two adjacent words are simultaneously transferred to the array of shift registers. The hardware architecture is depicted in figure 5. In order to allow the transfer of two contiguous words in the same clock cycle we have partitioned the tile buffer into two physical memories. One stores the words with an even position within the burst and the other memory stores the words with an odd position within the burst.

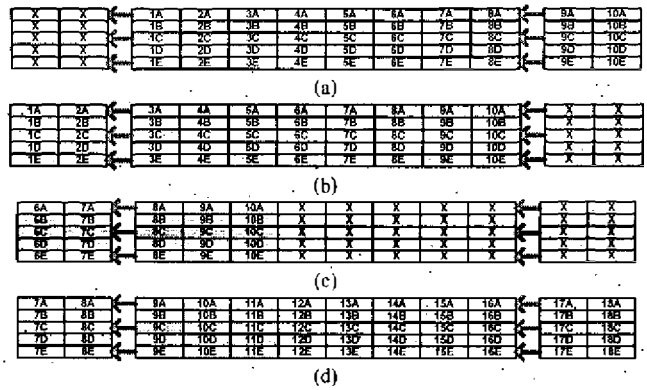

Figure 4. Shift-register contents of the row-major convolver. The figure shows the shift-register contents for the initial state (a), one cycle after the initial state (b), w cycles after the initial state (c) and for a state where the shift-register contents is re-loaded again (d).

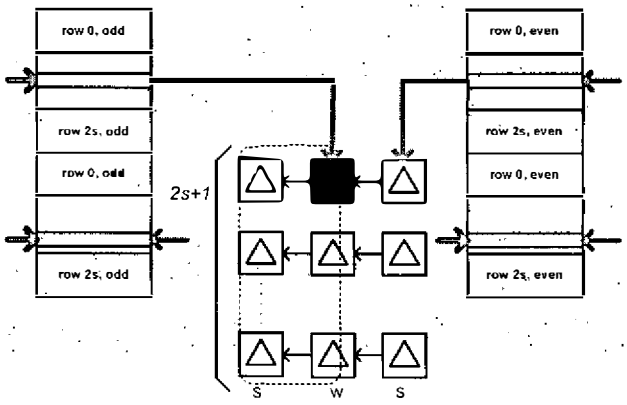

Figure 5. Architecture II. Row-major moving window architecture.

The hardware requirements for this architecture consist of a swing buffer for tiles partitioned into two physical memories (2x1.25Kb for our current example), plus *(2s+w)x(2s+1)* pixel registers for the moving register array (480 FFs for the case study). Concerning the throughput, w pixels are produced every *2s+1+w* clock cycles: *2s+1* clock cycles to load the shift registers plus w cycles to shift the word. For our study case this means 1.625 clocks/pixel, this means this concept provides more throughput than the previous one.

In this concept we find the same disadvantage as in I: in order to process a single row of the image, 2s+1 rows must be read from the external memory, therefore there is an im-

portant overhead that leads to an elevated memory bus bandwidth requirement by this block when compared to [3].

C. *Moving Window with Rotation Stage (Architecture III)*

This concept consists in reading a tile from the external memory, storing it in an internal memory. Then, memory words are transferred to a rotator following the process depicted in figure 7. In this transfer only a fraction of the memory word, and corresponding to *2s+1* pixels, is copied to the rotator. For instance, in fig. 7.a it is shown that the first *2s+1* columns are copied to the rotator. Once the rotator is full, column data is ultimately transferred to a shift register. In figure 7 from a) to c) we can see how data is transferred to the shift register. The register is shifted one position to the right each clock cycle until the whole row has been consumed. We can also notice that while data is being transferred from the rotator to the shift register, column data for the next cycles is being generated. Therefore the throughput becomes one pixel per clock (considering some latency cycles at the beginning of each row negligible).

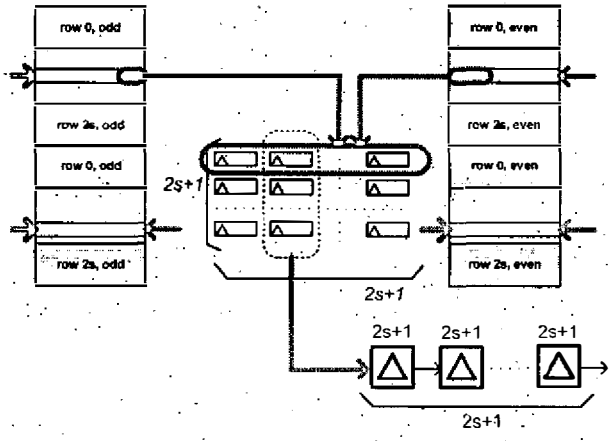

Figure 6. Architecture III. Moving window with rotation stage architecture.

In figure 6 we depict the hardware architecture for concept III. The hardware requirements for this architecture consist of two swing buffer for tiles implemented in two dual-port SRAM memories (2x1.25Kb for the case study), plus *(2s+1)x(2s+1)* pixel registers for the rotator (for the case study this means 200 FFs for an 5x5 rotator register array), plus *(2s+1)x(2s+1)* pixel registers for the shift register array (200 FFs for the case study).

In this concept we find the same disadvantage as in I and II: in order to process a single row of the image, *2s+1* rows must be read from the external memory, therefore there is an important overhead that leads to elevated memory bus bandwidth requirement by this block when compared to [3].

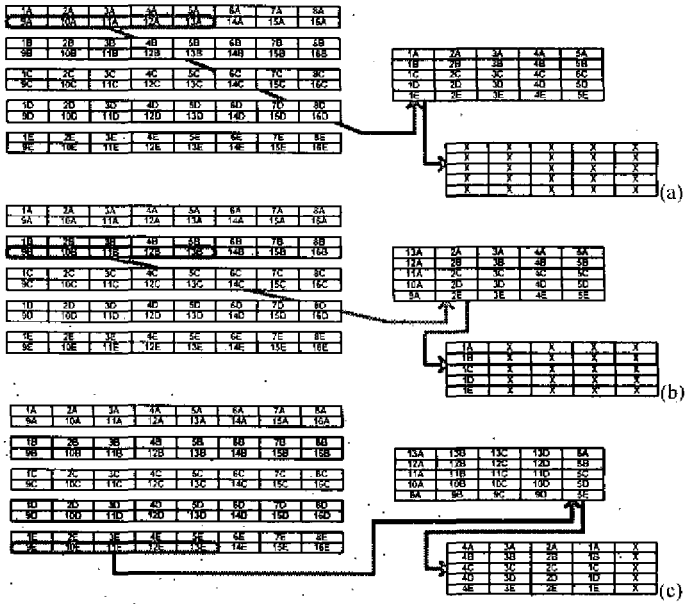

Figure 7. The figure shows the pipeline contents for the initial state (a), one cycle after the initial state (b), and 2s cycles after the initial state (c).

III. Architecture Selection

In table 1 we have summarized the main features of the proposed architectures: throughput, given in terms of clocks/pixel; area-utilization measured in terms of pixel registers (flip-flops are obtained by multiplying this figure by the bpp) and memory bits; latency measured in terms of clock cycles; and external memory bandwidth, taking [3] as the unit of reference. Memory bits are related to the burst-length (BL) that in our case study was 4, the pixels per memory word (W) that in our case study was 8, and the number of pixels per page width (PW) that in our case study is obtained from the page width (28L) and the printing resolution (600dpi).

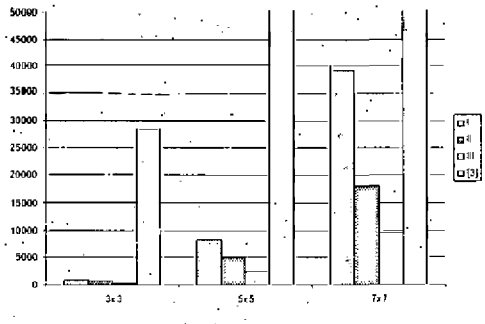

Figure 8. The figure shows a bar diagram comparing the area efficiency metric for techniques I, II, III and [3] and for window sizes 3x3, 5x5 and 7x7 using the parameters of the case study. The lower the bar, the more efficient.

In order to choose an architecture for a particular design, we propose maximizing the throughput with respect to the amount of resources used. For concept I, II and III, because the amount of memory bits is equal between them, mini-

TABLE I FEATURES OF MOVING WINDOW ARCHITECTURES

Architecture	Clocks/Pixel	Pixel Registers	Latency Cycles	Memory Pixels	Bandwidth	Clocks/Pixel*FF
I	2s+1	(2s+1)x(2s+1)+w	(2s+1)x(2s)	(2s+1)xBLxW	2s+1	1320
II	1+(2s+1)/w	(2s+w)x(2s+1)	0	(2s+1)xBLxW	2s+1	780
III	1	2x(2s+1)x(2s+1)	(2s+1)	(2s+1)xBLxW	2s+1	400
[3]	1	(2s+1)x(2s+1)	(2s+1)	(2s+1)xPW	1	-

mizing the product clocks-per-cycle times flip-flop number, is a suitable metric to maximize the value (performance) of the hardware resources used. We report this product for our case study in the last column of table 1. The higher this figure is, the more expensive the architecture is for this particular design point. In table I, we can see that for our case study III is more suitable than I and II.

The architecture found in [3] has not been included in the comparison because it requires a much larger internal memory and is less demanding in terms of bandwidth. In order to complete the selection process we define as optimum the architecture that minimizes the product resources times clocks-per-pixel times bandwidth required. In figure 8 we show the aforementioned metric for all three concepts and reference [3] for window sizes between 3x3 and 7x7. The remaining variables are the same described for the case study. In this design exploration we notice that III is superior to the rest of the architectures. The same principles can be used to determine the most area efficient architecture for any other point of the design space.

IV. CONCLUSION

In this paper we have presented three architectures for shifting a moving window over an image for 2-D shift-variant convolution. These techniques require much less memory than a direct implementation of prior art and they are therefore suitable for a low-cost FPGA implementation. We have proposed a criteria for selecting which architecture is the most suitable for each design point. This criteria is based on providing the maximum throughput per area unit.

ACKNOWLEDGMENT

The authors would like to acknowledge the useful comments of Tom Pritchard and Lluis Abello. The authors would also like to acknowledge the support of the Inkjet Commercial Division for this research. The opinions expressed by the authors are theirs alone and they do not represent the opinions of Hewlett-Packard.

REFERENCES

[1] D. M. Wetchler, and J. H. Bauman, "Multi-level pixel density reduction for printers," U.S. Patent 6 389 167, May 14, 2002.

[2] [2] J. Underwood, "Apparatus and method for fortification of black pigment based ink using black dye based ink," U.S. Patent 6 779 864, August 24, 2004.

[3] B. Bosi, G. Bois, and Y. Savaria, "Reconfigurable pipelined 2-D convolvers for fast digital signal processing," IEEE Trans. Very Large Scale Integration (VLSI) Systems, vol. 7, no. 3, pp. 229–308, Sept. 1993.

[4] Altera Corp. Cyclone-II FPGA Family Overview [Online]. Available: http://www.altera.com/products/devices/cyclone2/overview/cy2-overview.html

[5] Xilinx Inc. Spartan-3 FPGA Product Table [Online]. Available: http://www.xilinx.com/products/tables/fpga.htm#s3

[6] M. Mese, and P. P. Vaidyanathan, "Recent Advances in Digital Halftoning and Inverse Halftoning Methods," IEEE Trans. on Circuits and Systems - I, vol. 49, no. 6, pp. 790–805, June 2002.