A Design Methodology for Performance-Resource Optimization of a Generalized 2D Convolution Architecture with Quadrant Symmetric Kernels

Ming Z. Zhang and Vijayan K. Asari

Computational Intelligence and Machine Vision Laboratory
Department of Electrical and Computer Engineering
Old Dominion University, Norfolk, VA 23529, USA
{mzhan002,vasari}@odu.edu

Abstract. We present a design technique to meet application driven constraints for performance-resource optimization of a generalized 2D convolution with quadrant symmetric kernels. A fully pipelined multiplierless digital architecture for computing modularized 2D convolution utilizing the quadrant symmetry of the kernels is proposed in this paper. Pixels in the four quadrants of the kernel region with respect to an image pixel are considered simultaneously with distributed queues for computing the partial results of the convolution sum in time-sliced fashion. The new architecture performs computations in log-domain by utilizing low complexity high performance $\log_2$ and inverse-$\log_2$ estimation modules. An effective data handling strategy is developed to minimize routing of data path in conjunction with the logarithmic modules to eliminate the necessity of the multipliers in the architecture. The proposed architecture is capable of performing convolution operations for 45.5 (1024×1024) frames or 47.73 million outputs per second in minimum resource configuration with 8×8 kernels in a Xilinx's Virtex XC2V2000-4ff896 field programmable gate array (FPGA) at maximum clock frequency of 190.92 MHz. Analysis shows that the performance and resource utilization between the fully parallel and fully resource constrained architectures are proportional to f and 1/f, respectively where f is the application driven reusability of the main computing components. In addition to resource reduction from optimization scheme, evaluation in Xilinx's core generator also showed that the proposed design results in 60% reduction in hardware when compared to the design using pipelined multipliers.

Keywords: 2D convolution, log-domain computation, multiplier-less architecture, quadrant symmetric kernels, modularized optimization, FPGA based architecture.

1 Introduction

Two-dimensional convolution is one of the most frequently used operations in image and video processing applications. Kernel size is usually limited to a small

L. Choi, Y. Paek, and S. Cho (Eds.): ACSAC 2007, LNCS 4697, pp. 220–234, 2007.

bounded range when general processors are used. It is therefore necessary to find optimal designs to reduce hardware resources and power consumption while supporting high speed operations for real-time applications. Convolution is a mathematical operator which takes two functions, f and g, and produces a third function, z, that in a sense represents the amount of overlap between f and a reversed and translated version of g. The definition of 2D convolution $O = W * I$ in general can be expressed as

$$O(m,n) = \sum_{j_1=-a_1}^{a_1} \sum_{j_2=-a_2}^{a_2} W(j_1, j_2) \times I(m - j_1, n - j_2) \tag{1}$$

where a_i =(J_i-1)/2 for i = 1, 2 and W is the kernel function. The complexity for 2D convolution (filter) operation in image processing applications is of the order O($M \times N \times J_1 \times J_2$) where $M \times N$ corresponds to the x-y dimension of I/O images and $J_1 \times J_2$ is the size of the kernel. For instance, in a video processing application, if the frame size is 1024× 1024 and the kernel size is 10×10, more than 3 billion operations per second are required to support real-time processing rate of 30 frames per second. Many researches have studied and presented various methods to implement the hardware architectures to perform convolution operation for real time applications in the last two decades as described in [1] and [2]. One of the most difficult challenges in designing the high speed convolution architecture with a large kernel is to effectively utilize the hardware resources and limited number of processing elements (PEs) to support real time processing [3][4][5][6][7][8]. Algorithms such as [1][2][9][10][11][12] optimize the hardware resources by generating the kernel coefficients dependent architectures. Such architectures may be too specific and require the supports of reconfigurability and external system to generate reconfiguration bit streams for uploading to FPGAs in the event of changing the coefficients. Hence they are more suitable for static kernel coefficients. In general, 2D convolution operations can be partitioned to a number of 1D convolutions [13] without specializing for separable kernels [14]. Specifically, the partitioned 1D kernels are first convolved simultaneously on all columns of the input image and the partial results are accumulated in a row-ordered fashion or vise versa. We presented log-domain computation to significantly lower the complexity of the 2D convolution operation with quadrant symmetric architecture with support of programmable kernel coefficients to suit different transfer functions. In this paper, we propose a modularized 2D convolution and utilize distributed queue architecture for performance-resource optimization where excessive performance can be utilized to minimize resource requirement.

2 Concept of the Modularized 2D Convolution with Quadrant Symmetric Property

From the definition of 2D convolution in (1), it can be partitioned to reflect its symmetry. With slight modification to (1), equation (2) defines the convolution

operation where the center pixel of the kernel is overlapped with center pixel of the image under the window of consideration. The rearrangement of the equation is done by reducing the summation to half and by adding the terms to be multiplied by the same kernel coefficients. For odd kernel size in equation (3), corresponding locations in all four quadrants are added before multiplication. The last term is outside the summation since the kernel coefficients on the axes may be different from those reflected about the axes. For even kernel size, the folding method is shown in equation (4). This folding scheme can be achieved within the design with respect to particular architectural style and reduce processing power to focus on a quarter of the kernel.

$$O(m,n) = \sum_{j_1=0}^{J_1} \sum_{j_2=0}^{J_2} W(j_1,j_2) \cdot I\left(m + j_1 - \frac{J_1}{2} + 1, n + j_2 - \frac{J_2}{2} + 1\right) \tag{2}$$

$$\begin{aligned} O(m,n) = & \sum_{j_1=0}^{\frac{J_1}{2}-1} \sum_{j_2=0}^{\frac{J_2}{2}-1} W(j_1,j_2) \cdot I\left(m \pm j_1 + \tfrac{J_1}{2}, n \pm j_2 + \tfrac{J_2}{2}\right) \\ & + W\left(\tfrac{J_1}{2}, \tfrac{J_2}{2}\right) \cdot I(m,n) \end{aligned} \tag{3}$$

$$\begin{aligned} O(m,n) = & \sum_{j_1=0}^{\frac{J_1-1}{2}} \sum_{j_2=0}^{\frac{J_2-1}{2}} W(j_1,j_2) \times \\ & \left[\begin{array}{l} I\left(m + j_1 - \frac{J_1}{2} + 1, n + j_2 - \frac{J_2}{2} + 1\right) \\ + I\left(m - j_1 + \frac{J_1}{2}, n + j_2 - \frac{J_2}{2} + 1\right) \\ + I\left(m + j_1 - \frac{J_1}{2} + 1, n - j_2 + \frac{J_2}{2}\right) \\ + I\left(m - j_1 + \frac{J_1}{2}, n - j_2 + \frac{J_2}{2}\right) \end{array}\right] \end{aligned} \tag{4}$$

The $\log_2$ and inverse-$\log_2$ modules presented in [15] can further be employed to replace the essential multiplications in (3) and (4) with additions. The logical logarithmic operators in (5) require very low hardware complexity. We generalized the operators to handle input with fractions and derived very compact implementation without compromising hardware resource and performance compared to conventional (unrolled pipelining) architectures which operate on integers only [16], [17]. Eq. (5) states that the $\log_2$ scale of V can be calculated by concatenating the index I_V of leading 1's in V with the fractions (remaining bits after I_V^{th} bit). The reversed process holds true as well, except the leading 1's and fractions, L_f, are shifted to the left by L_i (integer of L) bits as shown in (5).

$$\log_2(V) \cong \{I_V\} + \{(V - I_V) \gg I_V\} \Leftrightarrow \log_2^{-1}(L) \cong \{1 \ll L_i\} + \{L_f \ll L_i\} \tag{5}$$

For even kernels, (4) can be modularized to balance the performance and hardware resource based on the constraint imposed by the application. For example of a high-end FPGA design (usually has greater throughput rate than needed by the application) deployed to low-end system, the excessive bandwidth can be distributed for reuse to minimize the resource. Vise versa, a low-end design can

sustain very high throughput by increasing internal parallelism with more resource. The partitioning scheme can be defined by (6), where $I_4(.)$ is the folded pixel value, and i is the i^{th} partition for i in $0..\lceil J_2/(2f)\rceil - 1$, and f is the desired reusability driven by application constraint. The design of modularized architecture is discussed in section 3.

$$\begin{aligned} O(m,n) &= \sum_{j_1=0}^{\frac{J_1-1}{2}} \sum_{j_2=0}^{\frac{J_2-1}{2}} W(j_1,j_2)\cdot I_4(.) \\ &= \sum_{i=0}^{\lceil J_2/(2f)\rceil-1} \left\{ \sum_{j_2=i\times f}^{(i+1)\times f-1} \sum_{j_1=0}^{\frac{J_1-1}{2}} W(j_1,j_2)\cdot I_4(.) \right\},\ j_2 \subseteq 0...\tfrac{J_2-1}{2} \end{aligned} \tag{6}$$

3 Architecture of Modularized 2D Convolution

3.1 Dataflow of Modularized Architecture

In this section, even kernel size is assumed for the discussion. The design also applies to odd kernel size with minor difference. It is also assumed the data comes in stream form (i.e. pixel by pixel fetched in real-time). Equation (4) is translated into dataflow of the convolution architecture as illustrated in Fig. 1. Incoming pixels pass through a series of line buffers (LBs) and fold at vertical position. This folding corresponds to the equation (4) without modification. The results from vertical folding are sent to a set of delay registers. The horizontal folding takes account of the inherent delays of systolic architecture rather than direct translation of (4). This is compensated by rerouting the folding points accordingly as shown in Fig. 1. The processing bandwidth is shared by f horizontally folded lines as reflected by the partition for each i in (6). The partial results from horizontal folding are successively accumulated to the right side and merged to form complete output.

3.2 Overview of Modularized 2D Convolution Architecture

Fig. 2 shows the block diagram of the quadrant symmetric convolution architecture with distributed storage queues. The LBs consist of J_2-1 line delays and create massive internal parallelism for concurrent processing. The folding is determined at nodes $LB(i)$+$LB(J_2 - i)$ where i ranges from 0 to (J_2-1)/2. The resulting nodes are registered and fed into multiplexers (MUXes) for time-sliced processing where each selected input occupies computational modules once every f cycles. The horizontal folding, as illustrated in Fig. 1, is done accordingly at the nodes $MQ(0)$+$DQ(2k+1)$ (where MQ is the registered output of MUX and DQ is the distributed storage queue for vertically folded data) rather than straight from the equations due to the pipelining involved in horizontal folding. The registered results from horizontal folding are sent to shared PE arrays (PEAs) for multiplication with the kernel coefficients and successive accumulation. The multipliers are reduced by three quarters by simply performing the folding procedure and by another f factor for sharing of processing bandwidth.

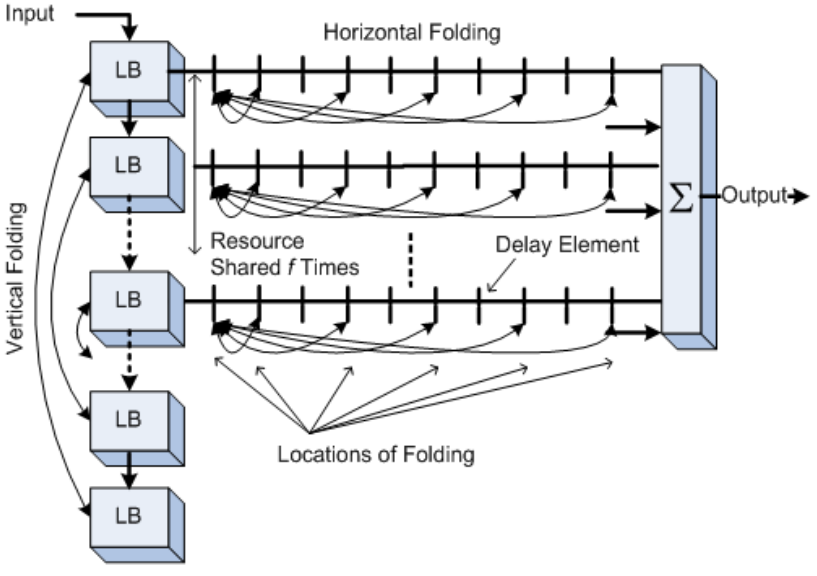

Fig. 1. Dataflow version of folding illustrates the compensated delays induced in the systolic architecture. The nodes involved with vertical folding at symmetric locations are added together. The horizontal folding takes place in every other nodes.

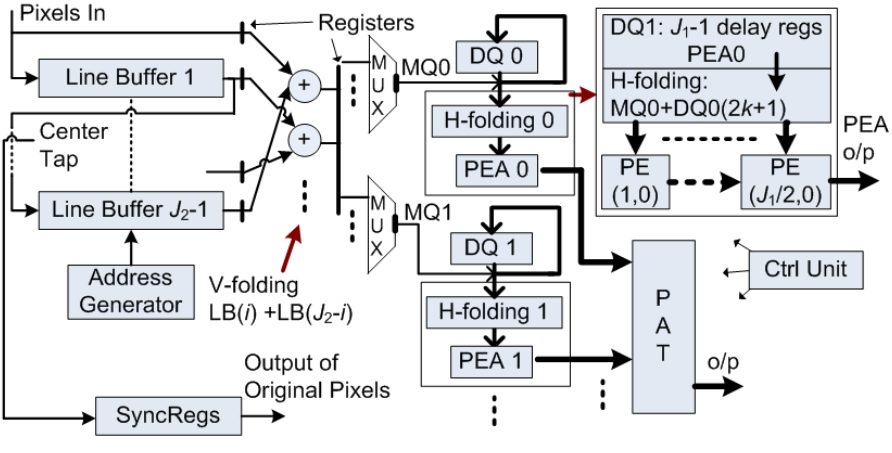

Fig. 2. Block diagram for overall architecture of the modularized 2D convolution with quadrant symmetric property in the kernel

The overall output is computed by merging results of the shared PEAs through pipelined adder tree (PAT). The original pixel values can be obtained, along with convolved data, through synchronized registers (SyncRegs).

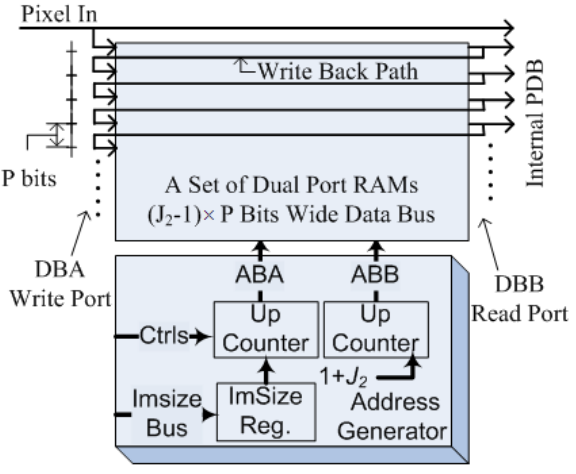

Fig. 3. Design of Data Buffer Unit with DPRAMs

3.3 Data Buffer Unit

The data buffer unit (DBU), which generates the internal parallelism is implemented with the dual port RAMs (DPRAMs) as shown in Fig. 3. One set of DPRAMs is utilized to form LBs and store just enough lines of image to create massive internal parallelism for concurrent processing. The pixels are fetched to the data buffer in raster-scan fashion which requires unity bandwidth suitable for real time streaming applications in video processing. The DPRAM based implementation has advantage of significantly simplifying the address generator compared to commonly known first-in-first-out (FIFO) based approach. Tracking of items during transient stage is eliminated as opposed to LBs implemented by FIFOs. The address generator with the DPRAMs based implementation makes scalability of DBU consistent and simple. It consists of two counters to automatically keep track of the memory locations to insert and read the data to internal parallel data bus (PDB). Data bus A (DBA) of $(J_2\text{-}1) \times P$ bits wide, which is formed with just enough number of DPRAMs in parallel, is used to insert pixel values through write-back paths to the memory location designated by address bus A (ABA). P is 8 bits for 8-bit pixel resolution. The data bus B (DBB) is used to read the pixel values onto PDB and write to the write-back paths. Only one address generator is necessary in buffering scheme. The DBU is only active once every f cycles. The vertical folding and modularizing scheme is discussed next.

3.4 Modularized Processing

The vertical folding is accomplished by pre-adding the nodes, $LB(i)+LB(J_2-i)$, on the PDB where i ranges from 0 to $(J_2\text{-}1)/2$ for even size kernels. The aligned results from vertical folding form a group of V-folded bus (VB) with the width

of $(P+1)\times f$ bits. Each VB is connected to an array of $P+1$ MUXes of f-to-1 type in such a way that the binary select lines of the MUXes incrementally activate appropriate data from PDB for time-sliced processing. Hence, each active ordered bus of $P+1$ bits wide will periodically occupy the PEA on one of those f time slots. The routing scheme shown in Fig. 2 and Fig. 4 (left) from VB to MQ can be generalized as

$$VB_i\{\upsilon\} \Rightarrow MQ_i \left\{ \begin{array}{ll} \upsilon, & \upsilon = (P+1)f-1 \\ \{\upsilon \times (P+1)\} \,\%\, \{(P+1)f-1\}, & otherwise \end{array} \right\} \quad (7)$$

for all i in (6), υ= 0..$(P+1)\times f-1$, where υ is the index to the VB which is mapped to the inputs of MUX array, and % denotes the modulo operation. For all select lines of MUXes, only one common counter suffices the time slicing necessary according to the partitioned convolution equation in (6).

3.5 Queuing of Active VB Data

The time-sliced data on MQ_i output is combined with the circular queue DQ_i, implemented with f registers of $(J_1-1)\times(P+1)$bits wide as illustrated in Fig. 4 (right). The merged signals $MDQ_i[0..J_1-1] = [MQ_i, DQ_i[f]]$ are connected to horizontal folding scheme described by

$$\mathrm{HB}_i(k) = \left\{ \begin{array}{ll} \mathrm{MDQ}_i[0] + \mathrm{MDQ}_i[2k], & \text{for even } J_1,\ \forall k \\ \mathrm{MDQ}_i[0] + \mathrm{MDQ}_i[2k+1], & \text{for odd } J_1, \mathrm{k} \neq 0 \\ \mathrm{MDQ}_i[0], & \text{for odd } J_1, \mathrm{k}{=}0 \end{array} \right. \quad (8)$$

to form horizontally folded bus (HB). The k ranges from 0 to $\lceil J_1/2 \rceil - 1$ for both even and odd kernels (i.e. J_1 is even number). The last $P+1$ bits of MDQ_i are dropped out with the remaining bits clocked back into the $DQ_i[0]$ register, mimicking the shifting operation of $P+1$ bits shift registers. Hence, the functional equivalence of fully parallel approach in [18] can be achieved in time-sliced fashion without introducing additional hardware components. The architecture of processing elements is discussed in the next section.

3.6 Processing Elements in PEAs

The design of each PE utilizes the log-domain computation to eliminate the need of hardware multipliers [15], [18]. The data from H-fold register is converted to $\log_2$ scale as shown in Fig. 5 (left) and added with the output of $\log_2$ scaled kernel coefficients queue (LKC) implemented with register set which holds the f desired spatial samples of the transfer functions according to the partitioned in (6). The result from last stage is converted back to linear scale with range check (RC). If the overflow or underflow occurs, the holding register of this pipeline stage is set or clear, respectively. Setting and clearing contribute the max and min values representable to $P+2$ bits (resolution of the $HB_i(k)$ data is 10 bits

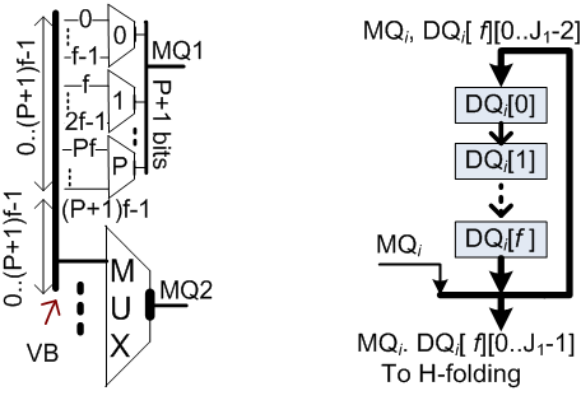

Fig. 4. Left: routing scheme for vertically folded data, Right: feedback path of DQ_i

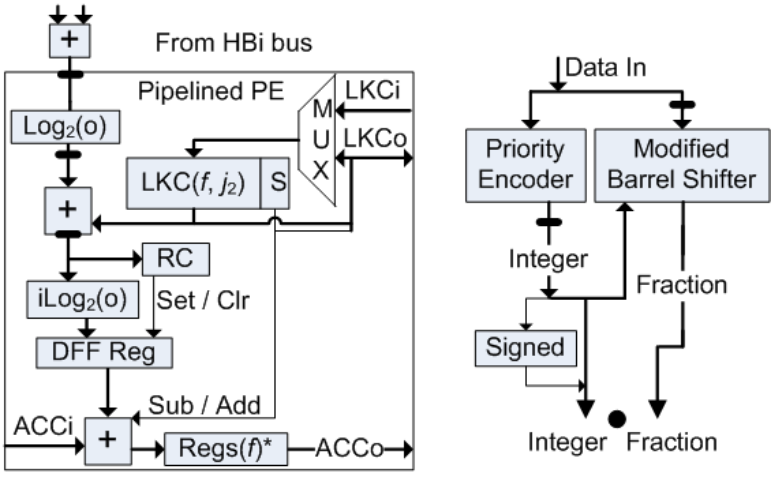

Fig. 5. Left: design of PE, Right: pipelined $\log_2$ module

for 8-bit image) register. The output of this stage is successively accumulated along the accumulation line (ACCi) and queued by f registers at the output (ACCo) to synchronize the computed partial results correctly to the assigned time-slice. 'ACCi' is set to 0's for the leftmost PE in each PEA. For the rightmost PE, the output buffer at 'ACCo' is replaced with cyclic adder for cyclic accumulation of the results produced by the PEAs. Cyclic adder is similar to conventional accumulator used in multiply-accumulate (MAC) unit with periodic reset signal. Cyclic adder sums up the results produced by each f-cycle interval. The output of LKC is looped back to its input through 2-to-1 MUX. Thus, the coefficients stored and rotated in LKC can be synchronized to appropriate

time-slice. The other input on the MUX is connected to the coefficient input bus, 'LKCi' from its neighbor PE. With the coefficient output, 'LKCo', an inter-chained bus can be organized for streamed initialization of kernel coefficients for different convolution masks. The PE can easily be modified to support negative inputs from horizontal folding. This is done by separating out the sign bit and exclusively OR-ed the registered sign bit with the sign from output of *LKC* queue. Hence if either one is negative, a subtraction is carried out alone the accumulation line. Otherwise, addition is performed. The $\log_2$ architecture shown in Fig. 5 (right) is very similar to [15], except full precision is used and registers are introduced to approximately double the performance. The maximum logic delay is reduced to single component and makes no sense to pipeline further.

4 Simulation and Error Analysis

4.1 Simulation

Both low-pass and high-pass transfer functions are applied to convolve with 8-bit grayscale images (336×336) in the simulation. The $\log_2$ and inverse-$\log_2$ modules are optimized based on the resolution of the test images shown in Fig. 6a and 6b. Fig. 6a is derived from Fig. 6b by adding Gaussian noise with$\mu = 0$and$\sigma^2 = 0.02$. Each test image is fetched into the architecture pixel by pixel in raster-scan fashion. After transient state, the outputs become available and are collected for error analysis. The resulting images produced by the Matlab software are shown in Fig. 6c and 6d for noise filtering and edge detection kernels, respectively. The corresponding output images from hardware simulation are shown in Fig. 6e and 6f. Both images are visually identical to Fig. 6c and 6d. Error Analysis is given next to determine quantitatively the degree of difference.

4.2 Error Analysis

Typical histograms of the error between software algorithm and hardware simulation are shown in Fig. 7a and 7b for the test images. The error produced in noise filtering illustrated in Fig. 7a has average error of 2.19 pixel intensity with peak of 22.23. The average error for applying the edge detection kernel in Fig. 7b is 4.17. Simulation with large set of images shows majority of the errors in this design is less than 5 pixel intensities. The peak errors are generally larger for high-pass transfer functions since the kernel coefficients typically have broader range of sample values. Images with evenly distributed histograms also have tendency to increase the peak error while contribute less to average error. The error measure includes the fact that the hardware simulation is bounded to approximation error and specific number of bits representable in the architecture where the software algorithm is free from these constraints. The performance-resource relationship is characterized in section 5.

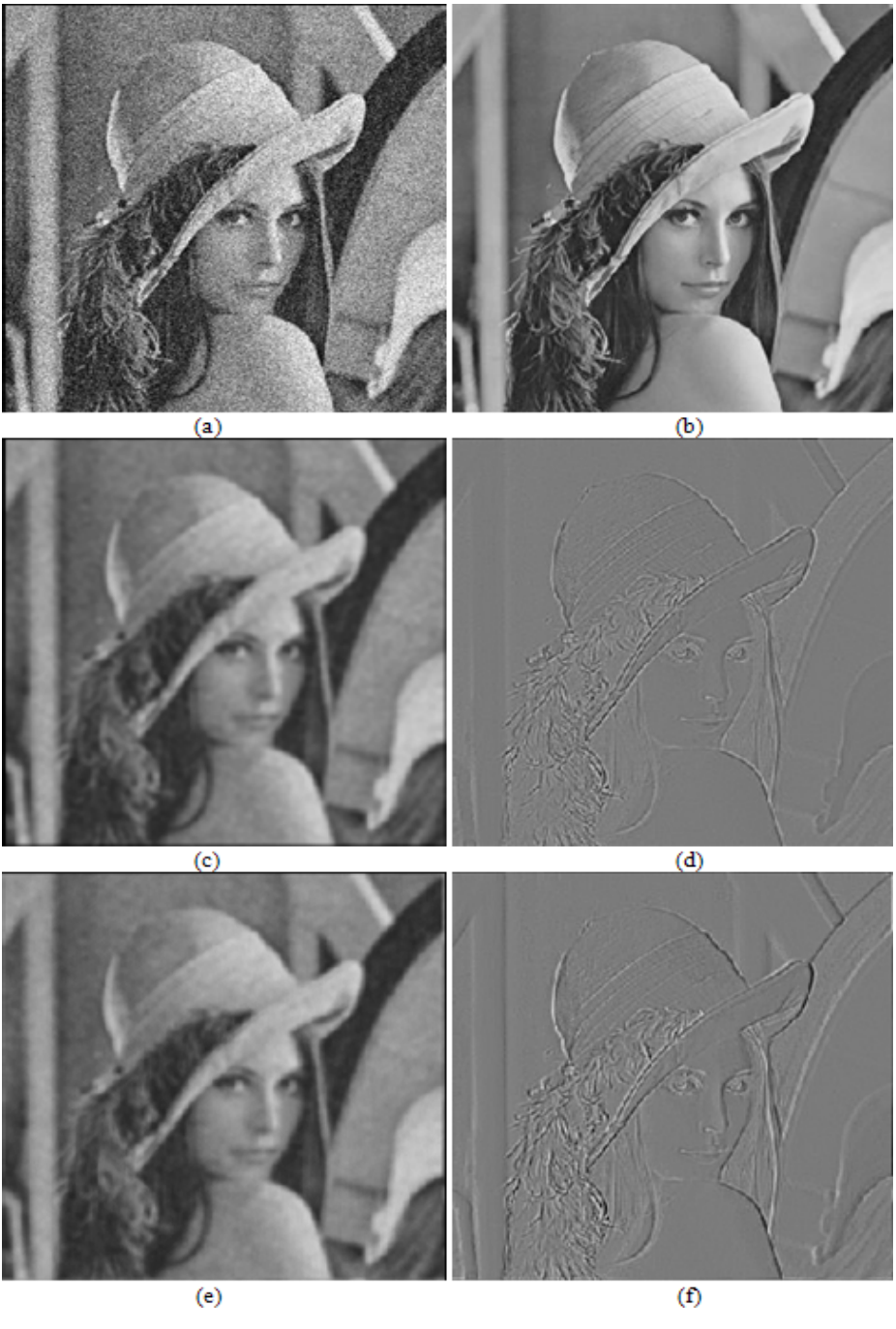

Fig. 6. (a) and (b) are test images for convolution operation with low-pass and high-pass transfer functions, respectively. (c) and (d) are the output images produced by Matlab software. (e) and (f) are the results from hardware simulation.

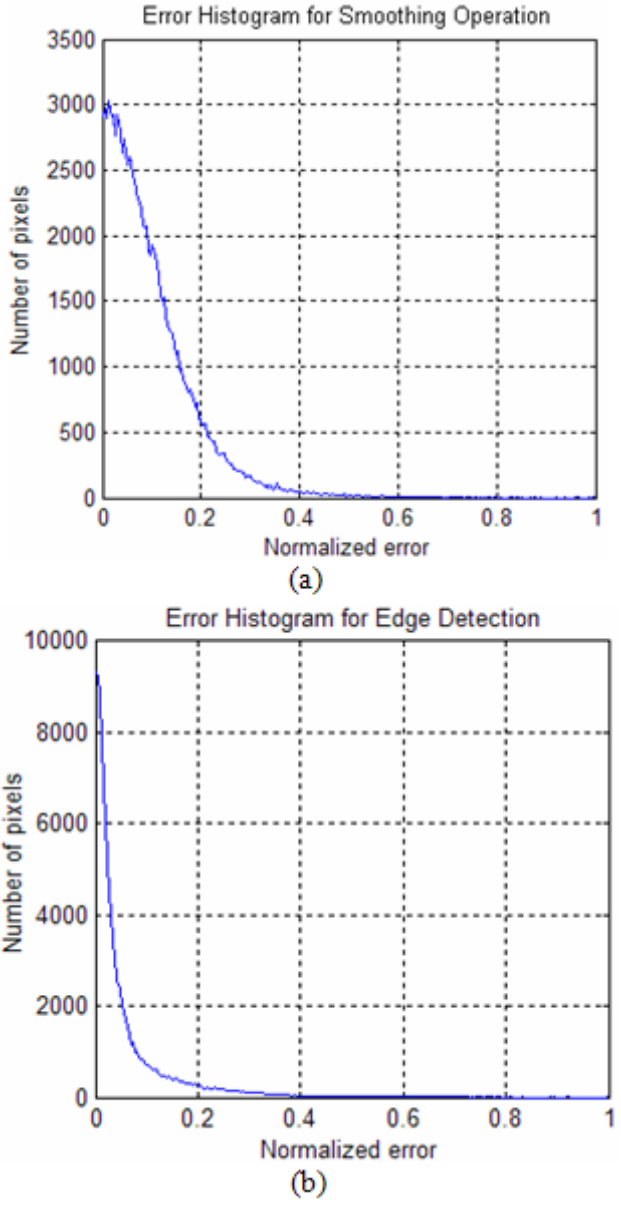

Fig. 7. Histograms of error for (a) noise filtering and (b) edge detection kernels. The corresponding average errors are 2.19 and 4.17 pixel intensities with the peak error at 22.23 and 70.97. The reason such large peak errors exist is that the range of kernel coefficients is broader with well distributed histogram on the test images.

5 Performance-Resource Optimization

5.1 Hardware Utilization of Minimum PEA Configuration

The hardware resource utilization is characterized based on the Xilinx's Virtex II XC2V2000-4ff896 FPGA on the multimedia platform and the Integrated Software Environment (ISE). The particular FPGA chip we target has 10,752 logic slices, 21,504 flip-flops (FFs), 21,504 lookup tables (4-input LUTs), 56 block RAMs (BRAMs), and 56 embedded 18-bit signed multipliers in hardware; however, we do not utilize the built-in multipliers. The resource allocation for various sizes of the kernels with minimum PEA is shown in Table 1. For 16×16 kernels, the computational power is 8 log-domain additions instead of 256 multiplications with fully parallel approach. The maximum windows can be utilized on target FPGA is mainly constraint by the storage capacity rather than the LUTs. In addition to trading the performance for resource, the design utilizes approximately 60% less hardware resource when compared to the quadrant symmetric architecture which implements fully pipelined multipliers.

5.2 Performance Evaluation of Minimum PEA Configuration

The critical timing analysis of Xilinx's ISE shows that the 190.922MOPS of the PEA is the most optimal throughput achievable with the maximum clock frequency of 190.922 MHz. The overall output of convolution architecture for various sizes is shown in the last column of Table 1. It should be clear that the performance decreases with large kernels. The propagation delay of log-based architecture is comparable to Xilinx's enhanced multiplier-based architecture which is 0.21ns faster. Further evaluation of pipelining the critical path suggests that increasing the level of pipelining does not gain significant throughput rate. Given 1024×1024 image frame, it can process over 45.5 frames per second without frame buffering with 8×8 kernel. Characterization of optimization scheme is discussed next.

5.3 Characterization of Optimization Scheme

In this section, we characterize the relationship between the performance and the resource allocation of the architecture in various configurations. We first analyze

Table 1. Hardware resource utilization for various sizes of the kernels with its corresponding performance indicates the overall effectiveness of the architecture

Kernel Size	Logic Slices	Slice FFs	LUTs	BRAMs	Perf (MOPS)
4×4	3%	2%	2%	2	95.46
8×8	6%	5%	3%	4	47.73
10×10	9%	7%	4%	4	38.18
12×12	12%	10%	4%	5	31.82
13×13	15%	13%	5%	6	27.28
15×15	19%	16%	6%	7	23.87
16×16	19%	17%	6%	7	23.87

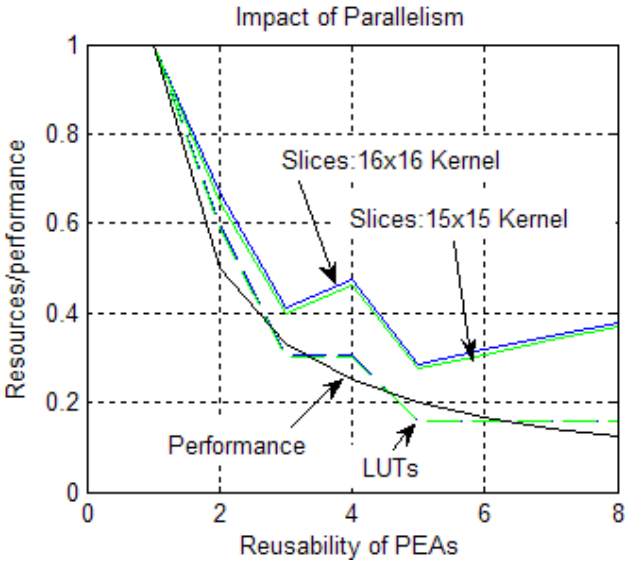

Fig. 8. The impact of reuse of PEAs over the performance and resource utilization of the architecture

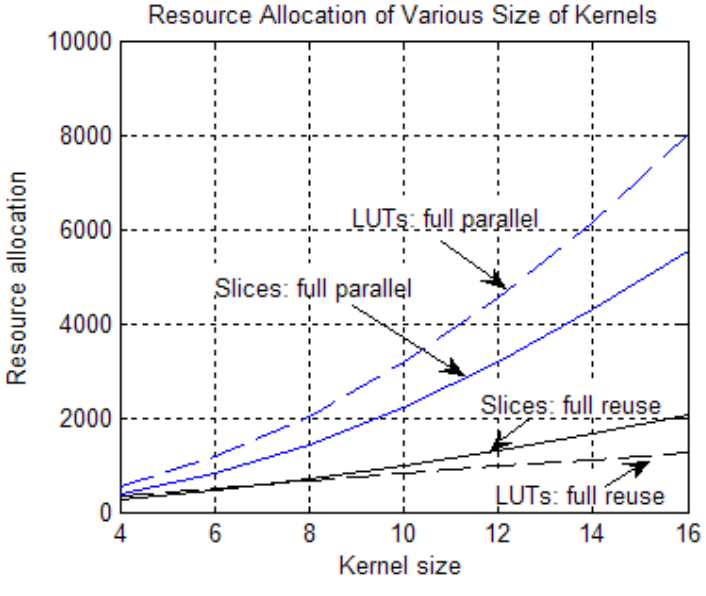

Fig. 9. The proportionality between the performance and resource allocation for various sizes of kernels

the impact of the parallelism on performance and resource requirement with fixed J_1 and J_2 attributes, as graphed in Fig. 8. The x-axis indicates the frequency of reuse of PEAs (i.e. the f attribute in(6)) while the y-axis is the resource and

performance quantities normalized by the design parameters configured to the fullest parallelism. Fig. 8 suggests that the resource utilization for even and odd sizes of kernels has minor difference. A more interesting point should be noted is that the performance is directly proportional to the utilization of the LUTs with varying f attribute. The Slices also tend to increase with larger f since storage capacity must be increased for the DQ queues, LKC queues, and output queues of the PEs. While the performance-resource has exponential relationship to the kernel size, its trading is nearly linear with f as depicted in Fig. 9. For example, the ratio for the LUTs of fully parallel and reused configurations is about 8 (f =8) while performance of fully reused architecture is reduced by $1/f$.

6 Conclusion

A technique for optimization of 2D convolution architecture with quadrant symmetric kernels is being presented in this paper to balance the performance and hardware resource based on the constraints imposed by the application. The design regularizes the reusability of the core components determined by application specification. We also demonstrated the generalized procedure for realization of the low complexity architecture with effective data path routing scheme and the assistance of logarithmic estimation modules. For the implementation highly limited to the resource, we showed that the architecture is able to sustain 45.5 1024×1024 frames per second (fps), which is more than real-time criteria of 30 fps, with minimum resource of 6% Slices and 3% LUTs for 8×8 kernels in a Xilinx's Virtex XC2V2000-4ff896 FPGA at maximum clock frequency of 190.92 MHz. We also determined that the optimization of the performance and resource has a relationship of $1/f$, where the f is evaluated by the application. With larger f, both normalized performance and resource are reduced to $1/f$.

References

1. Breitzman, A.F.: Automatic Derivation and Implementation of Fast Convolution Algorithms. PhD Dissertation, Drexel University (2003)
2. Jamro, E.: Parameterised Automated Generation of Convolvers Implemented in FPGAs. PhD Dissertation, University of Mining and Mentallurgy (2001)
3. Chang, H.M., Sunwoo, M.H.: An Efficient Programmable 2-D Convolver Chip. In: Proc. of the 1998 IEEE Intl. Symp. on Circuits and Systems, ISCAS, vol. 2, pp. 429–432. IEEE Computer Society Press, Los Alamitos (1988)
4. Hecht, V., Ronner, K., Pirsch, P.: An Advanced Programmable 2-D Convolution for Real Time Image Processing. In: Proc. of IEEE Intl. Symp. on Circuits and Systems, ISCAS, pp. 1897–1900. IEEE Computer Society Press, Los Alamitos (1991)
5. Kim, J.H., Alexander, W.E.: A Multiprocessor Architecture for 2-D Digital Filters. IEEE Trans. On Computer C-36, 876–884 (1987)
6. Nelson, A.E.: Implementation of Image Processing Algorithms on FPGA Hardware. MS Thesis, Vanderbilt University (2000)
7. Lee, S.Y., Aggarwal, J.K.: Parallel 2-D Convolution on a Mesh Connected Array Processor. IEEE Trans. On Pattern Analysis and Machine Intelligence PAMI-9, 590–594 (1987)

8. Bosi, B., Bois, G.: Reconfigurable Pipelined 2-D Convolvers for Fast Digital Signal Processing. IEEE Trans. On Very Large Scale Integration (VLSI) Systems 7(3), 299–308 (1999)
9. Wiatr, K., Jamro, E.: Constant Coefficient Multiplication in FPGA Structures. In: IEEE Proc. of the 26th Euriomicro Conference, Maastricht, The Netherlands, vol. 1, pp. 252–259 (2000)
10. Samueli, H.: An Improved Search Algorithm for the Design of Multiplier-less FIR Filters with Powers-of-Two Coefficients. IEE Trans. Circuits systems, 1044–1047 (1989)
11. Ye, Z., Chang, C.H.: Local Search Method for FIR Filter Coefficients Synthesis. In: Proc. 2nd IEEE Int. Workshop on Electronic Design, Test and Applications (DELTA-2004), Perth, Australia, pp. 255–260. IEEE Computer Society Press, Los Alamitos (2004)
12. Yli-Kaakinen, J., Saramäki, T.: A Systematic Algorithm for the Design of Multiplierless FIR Filters. In: Proc. IEEE Intl. Symp. Circuits Syst., Sydney, Australia, pp. 185–188. IEEE Computer Society Press, Los Alamitos (2001)
13. Kung, H.T., Ruane, L.M., Yen, D.W.L.: A Two-Level Pipelined Systolic Array for Multidimensional Convolution. Image and Vision Computing 1(1), 30–36 (1983)
14. Lakshmanan, V.: A Separable Filter for Directional Smoothing. IEEE Transaction on Geoscience and Remote Sensing Letters 1(3), 192–195 (2004)
15. Zhang, M.Z., Ngo, H.T, Asari, V.K.: Design of an Efficient Multiplier-Less Architecture for Multi-dimensional Convolution. In: Srikanthan, T., Xue, J., Chang, C.-H. (eds.) ACSAC 2005. LNCS, vol. 3740, pp. 65–78. Springer, Heidelberg (2005)
16. Mitchell, J.N.: Computer Multiplication and Division Using Binary Logarithms. IRE Transactions on Electronic Computers, 512–517 (1962)
17. SanGregory, S.L., Brothers, C., Gallagher, D., Siferd, R.: A Fast Low-Power Logarithm Approximation with CMOS VLSI Implementation. In: Proc. of the IEEE Midwest Symposium on Circuits and Systems, vol. 1, pp. 388–391. IEEE Computer Society Press, Los Alamitos (1999)
18. Zhang, M.Z., Asari, K.V.: An Efficient Multiplier-less Architecture for 2-D Convolution with Quadrant Symmetric Kernels. Integration, the VLSI Journal (in print)