Real-Time Image Resizing Hardware Accelerator for Object Detection Algorithms

Gaurav Mishra[#1], Yan Lin Aung[*2], Meiqing Wu[*3], Siew-Kei Lam[*4], Thambipillai Srikanthan[*5]
[#]Electronics and Communication Engineering, Indian Institute of Information Technology (Allahabad), India
[*]CHiPES Research Centre, Nanyang Technological University, Singapore
[1]mishragaurav27@gmail.com, [2, 4, 5]{layan, assklam, astsrikan}@ntu.edu.sg, [3]wume0007@e.ntu.edu.sg

***Abstract*—This paper describes motivation and hardware architecture for resizing input image frames from the camera in order to support real-time scale-invariant object recognition. Conventional implementation of object detection algorithms such as histogram of oriented gradients (HOG) based feature extraction, face detection using Haar classifiers often perform image resizing during the object recognition process. Our investigation reveals that this incurs significant performance overhead due to frequent memory accesses that are required for image resizing. This has motivated our approach to perform online resizing – simultaneously resizing of the input image when it is loaded into frame buffer memory – prior to the object recognition process. We propose a hardware architecture to accelerate image resizing and describe a hybrid processor-accelerator platform to generate different sizes of an image in real-time for object recognition.**

Keywords- Image processing; Image resize; FPGA; Haar classifier; HOG feature

I. INTRODUCTION

Innovative ideas in computer vision have made significant contribution in various application domains. Be it automotive [1], assembly line production, smart rooms [2] and other consumer electronic devices. In automotive domain, smart cars of today warn the drivers of sudden lane-departing event. The medical imaging equipment using vision-based techniques reduce human errors and offer better health care [3].

A majority of such algorithms rely on scale space theory of signals in which the input signal is convolved with Gaussian kernels of varying width [4]. A scale space provides in-depth analysis of an unknown external signal. The developed algorithm then searches and extracts useful features at different scales of the image making the overall feature set more robust for further processing. To reduce the execution time, one possibility is to use pre-downscaled version of an image, but then it will affect the accuracy of the algorithm. Many vision processing algorithms extract useful features from a given image for tracking or pattern matching by doing scale space analysis of 2D signal.

However, a subset of computer vision algorithms only require downscaled versions of the image instead of the complete scale space. Two well-known vision algorithms are Haar features based classifier for face detection [5] and feature vectors defined using HOG for detecting pedestrians [6]. Both algorithms are compute-intensive and operate iteratively on the downscaled versions of an image. Prior research work proposes implementation of complete algorithm as dedicated hardware in FPGA to achieve real-time performance. While those approaches provide the highest possible performance, flexibility and software programmability aspects are often compromised. With the advent of SoC FPGA, which features integrated Cortex-A9 dual-core processor from ARM operating from 800 MHz to 1 GHz and reconfigurable logic on the same silicon, we propose a hybrid solution with which only a carefully selected part of the algorithm is implemented in hardware while the rest remains as software; hence well-maintaining the software programmability and achieving real-time performance.

In order to emphasize on the flexibility of overall architecture level design on hybrid embedded platforms, which consists of a processor and reconfigurable logic, we propose a new approach in this paper. The rest of this paper is organized as follows: we review related work in Section II followed by Section III, which sheds light on savings of processor clock cycles offered by the bespoke design. Section IV describes the overall system-level design and Section V provides conclusions and future work.

II. RELATED WORK

This section describes evolution of the various architecture-level designs and their needs in particular, as thought of, in present scenario.

A. HOG Feature Set and Haar Classifiers

The purpose of feature vector in image processing and computer vision is to define a set of properties to represent an image. Such features include corners, edges, blobs, texture information etc. They can be defined globally or in a local neighborhood using block and cell based approaches. Image pyramid [7] accentuates the features present in an image and thus facilitates fast and accurate post-processing. However, the pyramid creation process is highly computation- and memory-intensive. Till recently, object detection algorithms (e.g. HOG and Haar classifiers) attempt to provide alternate methods using trained parameter values to generate robust feature sets thus avoiding the need to create an image pyramid.

The work in [6] formulates an alternative feature set by constructing histograms of oriented gradients so as to get better performance compared to that of image pyramid approach. Oriented histograms as feature vectors outdo the various edge and gradient descriptors. This also reduces the computational complexity. Similarly, Haar based classifiers as proposed in [5] are used for face detection instead of the image pyramid approach without degrading the detection ratio while reducing the computational

complexity drastically. However, it is noteworthy to mention that former uses SVM, while the later uses AdaBoost to obtain trained parameters for detecting a certain object class and avoid the pyramid creation process. This approach has been adopted in many computer vision applications to detect specific class of objects like pedestrians, automobiles, airplanes, animals etc.

B. *Existing Hardware for Object Detection*

A significant amount of prior work related to the aforementioned object detection algorithms focuses on dedicated hardware design of the complete algorithm. The work in [8] suggests a deeply pipelined and parallel design implemented on FPGA. Apparently numerous implementations of using HOG for pedestrian detection and feature extraction [8, 9, 10, 11], and Haar classifiers for face detection [12, 13, 14, 15], have been proposed. However, most of them suggest dedicated architectures and fail to embrace software programmability and flexibility of conventional processor. While the dedicated architectures could achieve the highest possible performance, one major drawback is that any change in the algorithm may often necessitate redesign of the dedicated hardware.

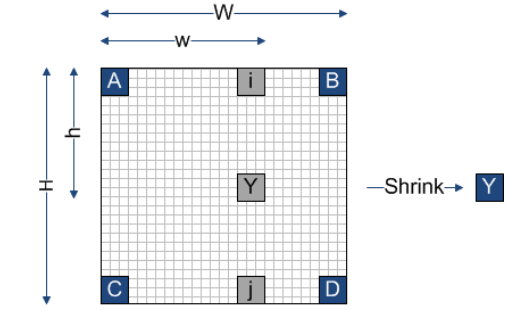
The survey undertaken in [16] compares the prevalent role played by FPGAs and SIMD processors in image processing tasks, highlighting the usefulness of a single hybrid platform comprising FPGA and processor. The leading FPGA vendors – Xilinx and Altera now provide SoC FPGA featuring integrated on-chip dual-core Cortex-A9 processor from ARM. This has motivated us to make use of the available on-chip resources thus exploiting both software and hardware programmability.

C. *Real-Time Image Resizing*

Both of the above-discussed algorithms meant for object detection or face detection utilize the rescaling of the input image frame in order to remain scale-invariant. This avoids the image convolution with several Gaussian kernels. The feature window then searches for the corresponding object in all scales of the image. The previous work in this domain includes ideas for implementing various resizing and rotation techniques [17] and other modifications like adaptive interpolation [18] and extended linear interpolation [19] to provide smooth and efficient rescaling operations.

However, the face/object detection algorithms provide sufficient hit rate by using simple interpolation methods, for example, nearest neighbor for Haar classifier and bi-linear interpolation for HOG. We focus to implement image resizing for real-time input image frames (streaming video) by creating all scaled versions of the image in parallel. We envisaged the frame buffer based implementation with which the input image from the camera is first loaded into external memory and object detection algorithms processed the image stored in the memory subsequently. We consider image resizing operation incorporating bi-linear interpolation in our proposed design.

1) Bi-Linear Interpolation: This is the default interpolation scheme used in *'cvResize()'* function of OpenCV library to resize an image for a particular scale factor. For the object detection approach using trained parameter values, this scheme meets the accuracy requirement with simplicity. The effects pointed out in [19] do not significantly affect the hit rate of the algorithm.



$$Y = A(1-w)(1-h) + B(w)(1-h) + C(h)(1-w) + D(wh)$$

Figure 1. Bi-linear Interpolation

To calculate the value of an interpolated pixel, input pixel values in consecutive rows from the image are required. The two consecutive rows, so used, must be separated by certain number of rows, which is equivalent to the factor by which the dimensions of destination image differs from the source. If 'src' is the source image array, 'dst' is the destination array, 'W' corresponds to the width of the source image, 'H' is the height of the source image and the image data is scanned progressively using the 'index' pointer. Fig. 2 represents the pseudo code for the interpolation process as shown in Fig. 1.

```
1   ImageResize(src, dst, W, H)
2
3   w = dstImg_width
4   h = dstImg_height
5
6   factor = W/w
7   pixel  = 0
8
9   for i = 1 to h
10  for j = 1 to w
11    do
12      x      = FLOOR(factor * j)
13      y      = FLOOR(factor * i)
14      x_diff = (factor * j) - x
15      y_diff = (factor * i) - y
16      index  = y * W + x
17
18      A = src[index]
19      B = src[index + 1]
20      C = src[index + W]
21      D = src[index + W + 1]
22
23      val = A(1-x_diff)(1-y_diff)+
24            B(  x_diff)(1-y_diff)+
25            C(  y_diff)(1-x_diff)+
26            D(  x_diff)(  y_diff)
27      dst[pixel++] = FLOOR(val)
28  end
29  end
```

Figure 2. Pseudo Code for Bi-Linear Interpolation

III. Analysis of Processor-based Software Implementation

We implemented OpenCV-equivalent people detect application in ANSI-C and used the open-source face detection application based on Haar classifiers from [14]. Both applications are not limited to the input image size. For testing purpose, we used 320×240 resolution images. The feature window used is of size 24×24 for Haar and 64×128 for HOG. Thus, an initial scale value of 1.02 for Haar and 1.05 for HOG create 12 resized images for the given resolution. The HOG operation uses the default bi-linear interpolation method whereas the Haar operation uses the nearest neighbor scheme to rescale the input image. We then executed the C source code for both object detection algorithms on two different platforms: Intel Core-i7 with 8GB RAM and ARM Cortex-A9 with 1GB RAM. We used 'rdtsc' assembly instruction, which accesses time-stamp counter, for the Intel platform and counter register in the performance monitoring unit of ARM platform to measure the number of clock cycles required by the image resize operation and object detection operation in the same resized image. The 'Resize' and the 'Detect' in Fig. 3 and 4 represent the cycle counts respectively.

A. Hardware Platforms

1) Intel Core-i7 with 8GB RAM

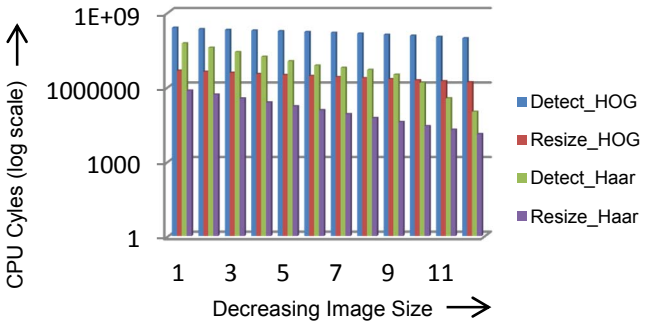


Figure 3. Processor Cycle Counts for Intel Platform

2) ARM Cortex-A9 with 1GB RAM

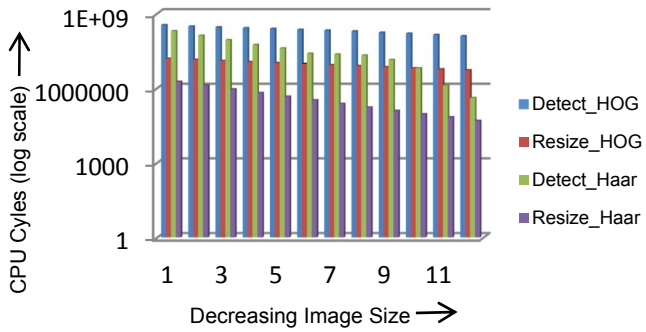


Figure 4. Processor Cycle Counts for ARM Cortex-A9 Platform

From the plots shown in the Fig. 3 and 4, it is inferred that CPU cycle counts required for the resize operation varies linearly with that required by the post-detection process. In fact the cycle count for the resize operation is one order higher in case of ARM platform than that in Intel platform. A total of 4470581 CPU cycles can be saved in Intel and 15323326 cycles in ARM processor if the resizing operation is performed prior to detection by using a hardware accelerator.

IV. System Architecture for Hybrid Platform

Fig. 5 instills the proposed system-level architecture. It illustrates the overall design flow using a hard processing system (HPS), which includes the ARM-A9 processor along with the memory controller module, and a programmable hardware accelerator.

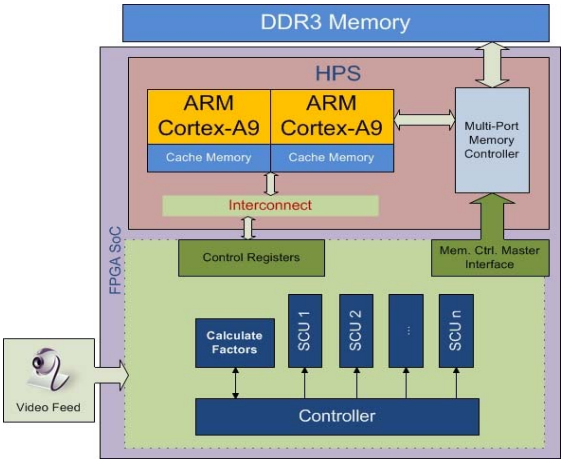


Figure 5. System-Level Architecture

The user operations of FPGA module present on the programmable logic (PL) side allows the configuration of registers in addition with the control of the data flow from the FPGA module to the main memory. The overall dataflow can be summarized sequentially in terms of the involved modules as: video feed, controller, scale computation unit (SCU), main memory. Using a C code, the user maps the addresses in the main memory according to the size of different scales of the image and configures the control registers such as image size, the window size, total number of possible scales and the preferred initial scale to begin with. The memory control master interface (MCMI) will interact with the multi-port memory controller (MPMC). This will enable the flow of interpolated pixel values simultaneously from different SCUs. The hardware fills the memory mapped addresses with interpolated pixel values for different scales. The following sub-sections describes the proposed image resizing hardware accelerator on the PL.

A. Controller

The controller is responsible for configuring the register in each SCU with the corresponding factor values and also handles the flow of input pixel data according to the requirements of each SCU. The complete block on the PL side utilizes the values present in following registers:

1) Scale: to store the initial scale value

2) No_of_Scales: to store the total number of possible scales for given resolution of the video feed

3) Image_Width: to store the width of a frame

4) Image_Height: to store the height of a frame

5) Win_Width: to store the width of the feature window, which is to be used

6) Win_Height: to store the height of the feature window, which is to be used.

We anticipate an initial latency of one row for the very first video frame from the input feed. Henceforth with a pipelined design, one can generate the required interpolated pixel of the scaled image simultaneously for every pixel value read from the input side.

B. Calculate Factors

The controller provides the value present in the configuration registers stated above in Section A. Using the initial scale, this block iteratively calculates the different scale values required until the final resized image becomes comparable to the dimensions of the feature window under consideration. For example, with image resolution 320×240, initial scale 1.05 and 64×128 feature window the last resized image to store will be of 178×134 resolution. It should be noted that the maximum time taken by this block to generate all the required scaling factors would be always less than or equal to the total number of columns of the input video frame. The fact helps in calculating the scale factors while the reading of the first row of the initial frame is in progress. Once the computation is complete the controller configures all the 'Factor_val' register of the SCUs.

C. Scale Computation Unit

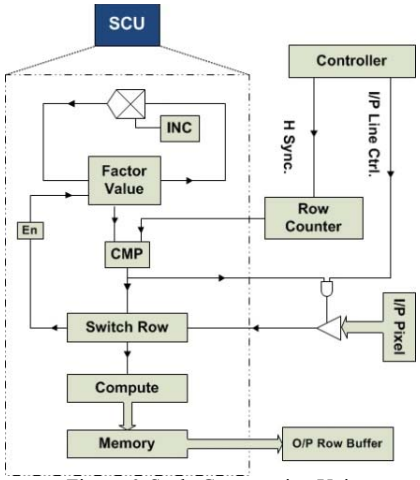


Figure 6. Scale Computation Unit

A single SCU provides the interpolated pixels for a particular scale of the given input image. Depending on the total number of scale factors required by the algorithm several of these SCUs are used in parallel. The number of SCUs generated can be parameterized, according to the number of scales before the synthesis process. Also two input (I/P) row buffers and one output (O/P) buffer are used by a single SCU for computation process. The row buffers are modulo-n counters with 'n' as the frame width. Fig. 6 shows the control and the data lines coming to a single SCU. The overall functioning of the module can be divided into the following three sections.

1) CMP: As evident from the overview of bi-linear interpolation approach, an interpolated pixel is calculated using the pixel values from two rows separated by corresponding 'factor' number of rows. Thus a SCU is active only when the two input rows are separated with certain distance, in the I/P frame. This task is accomplished using the 'Row Counter' (row_ctr) block, which is using the 'hsync' signal, from the I/P device, as trigger. 'CMP' is a comparator block to check if the row counter has the same value as the factor value. This check helps to ensure the correct flow of data into the I/P row buffers. A positive signal from this block enables computation for O/P data and calculation of the next factor multiple, to be used to detect following I/P row. In Fig. 7, rows in red color match the criteria to get accepted by a SCU.

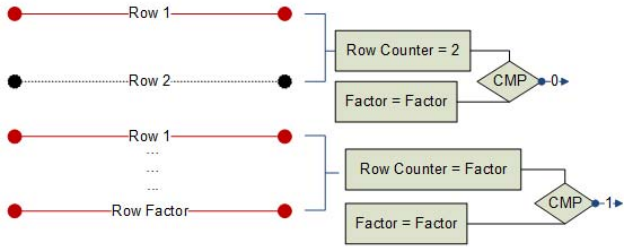


Figure 7. I/P Row Selection

2) Switch Row: This block decides the primary row, out of the two I/P row buffers. This hierarchy of the rows, in a given iteration, is important as the I/P pixel values are scanned in a certain fashion as defined by the interpolation algorithm. In order to reduce the memory usage we annotate implicitly one of the row buffers as primary while the other as secondary. Whenever a high signal is received from the CMP block the row considered primary, in the previous iteration, is now to be considered as secondary and vice-versa for the secondary row. The I/P pixel data is always fed into the primary row buffer and computation of the corresponding interpolated O/P row buffer begins utilising the, already filled, secondary I/P row buffer. For example, in Fig. 7 'Row 1' is secondary and 'Row 1 * Factor' is primary; then for the next iteration 'Row 1 * Factor' becomes secondary and 'Row 2 * Factor' becomes primary.

3) Compute: This module consists of a a fixed-point unit block with a proposed precision of 8 decimal digits to satisfy the required accuracy. The 'Round' block is hardware description of the rounding operation used for calculating integers from decimal numbers. Both of these blocks utilize the pixel values from the two I/P row buffers.

We utilize two counters in each of row buffers in order to read the pixel values serially. A register 'column counter' (col_ctr) is meant to store the number of pixel values read from the buffers.

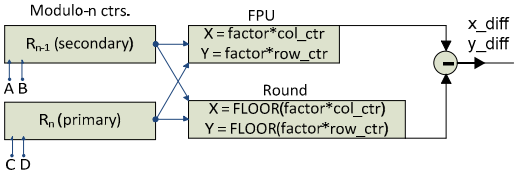


Figure 8. Single Pixel Calculation

The values 'x_diff' and 'y_diff' are computed as shown in Fig. 8 and then used to calculate the interpolated pixel value as in Line 23 of Fig. 2.

D. Store the Output

With all the SCUs working in parallel a lot of output data has to be handled in the design. This is managed using the MPMC. This controller provides multiple parallel ports for data communication with the main memory. Each data port is categorized according to the priority level. In proposed system design the MCMI on PL side handles the data from different SCUs and passes it to MPMC. With multiple output ports to the main memory the controller can write simultaneously to the different memory mapped regions. In this way different resized version of the input frame will be created at once.

V. Conclusions and Future Work

We have proposed the complete system-level design required for the current emerging hybrid platforms. The disadvantages of creating the dedicated hardware architectures for complex algorithms are now apparent. Also the complete reliance on software-only implementation of object detection algorithms does not lead to a satisfactory solution. This is in accordance with the processor clock cycle counts, which we have obtained in our experiment on two processor-based platforms. With hardware accelerators to resize the image in real-time, along with software application to manage the control and dataflow, we are able to improve the execution time of object detection algorithms. Our future work includes implementation of the proposed architecture using a HDL language and later using HLS tools. Our eventual aim is to evaluate performance-area trade-offs while meeting the real-time constraints.

References

[1] G. Zhibo, L. Huajun, W. Qiong, and Y. Jingyu, "A Fast Algorithm of Face Detection for Driver Monitoring," in Intelligent Systems Design and Applications, 2006. ISDA '06. Sixth International Conference on, 2006, vol. 2, pp. 267-271.

[2] Z. Zhang, G. Potamianos, M. Liu, and T. Huang, "Robust Multi-View Multi-Camera Face Detection inside Smart Rooms Using Spatio-Temporal Dynamic Programming," in Automatic Face and Gesture Recognition, 2006. FGR 2006. 7th International Conference on, 2006, pp. 407-412.

[3] V. Ayala-Ramirez, R. E. Sanchez-Yanez, and F. J. Montecillo-Puente, "On the Application of Robotic Vision Methods to Biomedical Image Analysis," in IV Latin American Congress on Biomedical Engineering 2007, Bioengineering Solutions for Latin America Health, vol. 18, C. Müller-Karger, S. Wong, and A. Cruz, Eds.: Springer Berlin Heidelberg, 2008, pp. 1160-1162.

[4] A. P. Witkin, "Scale-space filtering: A new approach to multi-scale description," in Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '84., 1984, vol. 9, pp. 150-153.

[5] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, 2001, vol. 1, pp. I-511-I-518 vol.511.

[6] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, 2005, vol. 1, pp. 886-893 vol. 881.

[7] C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden. (1984). Pyramid Methods in Image Processing. [Online].

[8] K. Negi, K. Dohi, Y. Shibata, and K. Oguri, "Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm," in Field-Programmable Technology (FPT), 2011 International Conference on, 2011, pp. 1-8.

[9] C. Tam Phuong, D. Guang, and D. Mulligan, "Implementation of real-time pedestrian detection on FPGA," in Image and Vision Computing New Zealand, 2008. IVCNZ 2008. 23rd International Conference, 2008, pp. 1-6.

[10] R. Kadota, H. Sugano, M. Hiromoto, H. Ochi, R. Miyamoto, and Y. Nakamura, "Hardware Architecture for HOG Feature Extraction," in Intelligent Information Hiding and Multimedia Signal Processing, 2009. IIH-MSP '09. Fifth International Conference on, 2009, pp. 1330-1333.

[11] K. Mizuno, Y. Terachi, K. Takagi, S. Izumi, H. Kawaguchi, and M. Yoshimoto, "Architectural Study of HOG Feature Extraction Processor for Real-Time Object Detection," in Signal Processing Systems (SiPS), 2012 IEEE Workshop on, 2012, pp. 197-202.

[12] T. Theocharides, N. Vijaykrishnan, and M. J. Irwin, "A parallel architecture for hardware face detection," in Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on, 2006, vol. 00, p. 2 pp.

[13] L. Hung-Chih, M. Savvides, and C. Tsuhan, "Proposed FPGA Hardware Architecture for High Frame Rate (≫100 fps) Face Detection Using Feature Cascade Classifiers," in Biometrics: Theory, Applications, and Systems, 2007. BTAS 2007. First IEEE International Conference on, 2007, pp. 1-6.

[14] M. Hiromoto, H. Sugano, and R. Miyamoto, "Partially Parallel Architecture for AdaBoost-Based Detection With Haar-Like Features," Circuits and Systems for Video Technology, IEEE Transactions on, vol. 19, pp. 41-52, 2009.

[15] J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, "Fpga-based face detection system using Haar classifiers," in Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays, Monterey, California, USA, 2009, pp. 103-112.

[16] C. Kyrkou. Image Processing on FPGAs - A Survey. [Online].

[17] R. D. Turney and C. H. Dick. Real Time Image Rotation and Resizing, Algorithms and Implementations. [Online].

[18] X. Jianping, Z. Xuecheng, L. Zhenglin, and G. Xu, "Adaptive Interpolation Algorithm for Real-time Image Resizing," in Innovative Computing, Information and Control, 2006. ICICIC '06. First International Conference on, 2006, vol. 2, pp. 221-224.

[19] L. Chung-chi, S. Ming-hwa, C. Huann-Keng, T. Wen-kai, and W. Zeng-chuan, "Real-time FPGA architecture of extended linear convolution for digital image scaling," in ICECE Technology, 2008. FPT 2008. International Conference on, 2008, pp. 381-384.