NOVEL FPGA BASED HAAR CLASSIFIER FACE DETECTION ALGORITHM ACCELERATION

Changjian Gao+

Wireless Connectivity
Broadcom Corp.
16340 W Bernardo Dr., San Diego, CA 92127
email: cgao@broadcom.com

Shih-Lien Lu

Microarchitecture Research
Intel Corp.
2111 NE 25th Ave, Hillsboro Oregon 97124
email: shih-lien.l.lu@intel.com

ABSTRACT

We present here a novel approach to use FPGA to accelerate the Haar-classifier based face detection algorithm. With highly pipelined microarchitecture and utilizing abundant parallel arithmetic units in the FPGA, we've achieved real-time performance of face detection having very high detection rate and low false positives. Moreover, our approach is flexible toward the resources available on the FPGA chip. This work also provides us an understanding toward using FPGA for implementing non-systolic based vision algorithm acceleration. Our implementation is realized on a HiTech Global PCIe card that contains a Xilinx XC5VLX110T FPGA chip.

1. .INTRODUCTION

Currently, the microprocessor industry is aggressively moving to chip multiprocessor (CMP) or multicore-aware architectures. However, with different applications and under different usage scenarios, a homogeneous CMP may not achieve the optimal computational throughput over cost and power. Thus, a heterogeneous CMP architecture has been proposed [1]. One possible approach to heterogeneous CMP design consists of processors with reconfigurable fabrics. These reconfigurable fabrics can customize individual cores to satisfy different computational demands of different applications.

Motivated by the need to experiment the architectural idea of adding reconfigurable fabric to form a kind of heterogeneous CMP, we start out by investigating how to speedup some statistical computing or computer vision problems with the help of FPGA (field programmable gated arrays). Our approach begins by examining the possibility to speed up functions in the Intel's Integrated Performance Primitives (IPP) library [2]. IPP library is widely used to optimize many computational intensive applications for multimedia, communication, and computer vision. It is, therefore, beneficial to examine the feasibility of mapping commonly used functions in the IPP library to reconfigurable fabric. It is hoped that by using reconfigurable fabric to implement these functions the overall performance of a large number of applications can be improved over the pure software solution.

As an experiment we chose among those computer vision applications the face detection application to verify our approach. Face detection is the computation problem of identifying and locating a human face in a photo or video regardless of the lighting, angle, and size. It is useful in many other vision based applications. We first looked at a performance profile to study the face detection program. We discover that the most time consuming function (greater than 500 clock ticks for every image pixel) in its algorithm is the Haar classifier. This Haar classifier function is implemented with Intel's IPP to detect faces [3, 4]. It occupies 93% of the total computation time.

Yang *et al.* [5] summarized different face detection algorithms. Among them, SVM (support vector machine) and NN (neural networks) achieve the best performance (in terms of detection rate and false alarm rate). Due to better generalization performance, SVM is the most popular algorithm used by the machine learning society. Our Haar classifier face detection algorithm is based on SVM [6, 7].

Most FPGA implementations of image processing utilize the systolic array structure of image data. Thus they represent similar streaming data processors [8, 9]. Irick *et al.* [10] used the streaming architecture to implement face detection algorithms based on neural network (NN) and achieved high performance. However, their pixel offset of 10 is unrealistically high. And they did not have a performance comparison against the pure microprocessor-based software implementations. Other face or object detection FPGA implementations are either slow in performance, or have a lower detection rate and higher false alarm rate [11, 12, 13, 14, 15]. Our approach has higher frontal face detection rate and lower false alarm rate when it is compared with Intel's IPP and OpenCV (Open Computer Vision Library) [16] software solution. With highly pipelined and parallel architecture, our system achieved real-time (37 frames/sec) face detection performance. Furthermore, our design used the commercially available PCIe (PCI Express) FPGA card, it is comparably easy to migrate our design to other object detection, recognition, and tracking applications with similar Haar classifier functions.

2. ALGORITHM

2.1. Haar Classifier Face Detection Algorithm

Our face detection algorithm utilizes the Haar classifier function adapted from Viola *et al.* [17, 18] and Papageorgiou *et al.* [6, 7]. Mita *et al.* [19] also proposed such Haar classifier face detection algorithm with more features (greater than five) in each classifier stage to improve the

+ Work performed while interning at Intel Corp.

978-1-4244-1961-6/08/$25.00 ©2008 IEEE.

Authorized licensed use limited to: International Institute of Information Technology Bangalore. Downloaded on October 09,2024 at 03:36:33 UTC from IEEE Xplore. Restrictions apply.

detection rate and to lower the false alarm rate at the same time. Lienhart *et al.* was the first to introduce this algorithm into Intel's IPP [3, 4], which was later included into OpenCV.

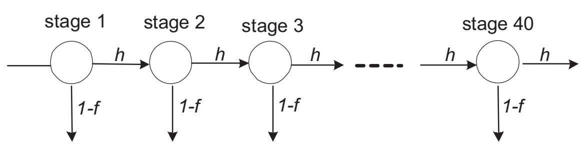

Fig. 1. Cascade of Haar classifiers, where hit rate = h^{40}; false alarm rate = f^{40}.

Viola *et al.* [18] proposed a cascaded degenerated decision tree for a fast software computing while maintaining same face detecting rate compared to other slower single stage classifier. They used AdaBoost to select a small set of features to train the classifiers. Haar wavelet features are pixel tiles similar to the rectangles shown in Fig. 2(a)-(b). Each classifier has two or three features. They have different weights for different rectangle regions, and describe the likelihood of such rectangle pairs with the features of human frontal faces [4]. Our FPGA implementation utilizes 40 classifier stages. As illustrated in Fig. 1, assuming each stage has the same detection rate *h*, and false alarm rate *f*. The final overall detection rate is h^{40}, and false alarm rate is f^{40} therefore.

In order to leverage the 16 classifier units in the FPGA (we will introduce details in Section 3.2), we re-trained the Haar classifier with multiple of 16 classifiers per stage, but kept the detection rate and alarm rate at the similar levels as in the OpenCV software version. Our final trained cascade Haar classifiers include 40 stages, 2192 Haar classifiers, and 4680 features with a 16×16 window size per pixel sub-window, compared with 22 stages, 2135 Haar classifiers, and 4630 features with a 20×20 sub-window in the OpenCV default values. The new FPGA-friendly Haar classifiers also reduced the software computational time for detecting faces from the CMU 94 frontal face images test set [20]. As listed in Table 1, our modified FPGA-friendly classifiers achieve better performance over the original default OpenCV classifiers.

Table 1. Performance comparison between our FPGA friendly 40-stage Haar classifiers and OpenCV's 22-stage Haar classifiers. The microprocessor platform to run the software is explained in Section 4.

	Detection Rate (%)	False Alarm Rate (%)	SW Execution Time (sec)
40-stage	87.6	13.5	18.1
22-stage	75.2	15.9	24.7

As illustrated in Fig. 2, after we have trained the Haar classifiers with our own training images [18], we could detect faces in three steps. The first step is *pre-processing*. It is used to calculate the integral images and lighting corrections from the scaled (scale factor is 1.2) original image. The second step is *Haar classifier function*. The main purpose during this step is to scan every image pixel with the trained-Haar classifiers to determine if it is a face pixel or not. The third step is *post-processing*. It clusters the adjacent detected faces to one or several rectangles to represent faces according to distance factor of 5.

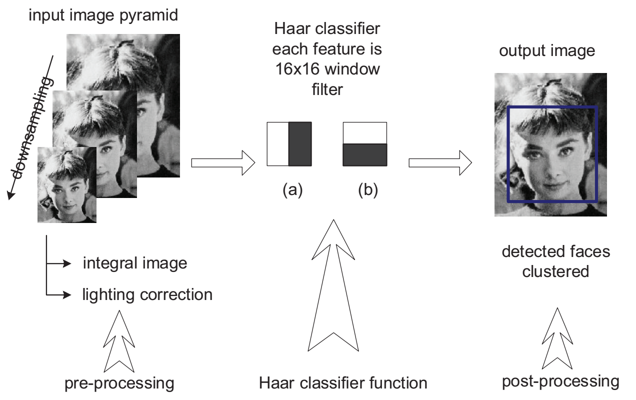

Fig. 2. Haar classifier face detection algorithm data flow.

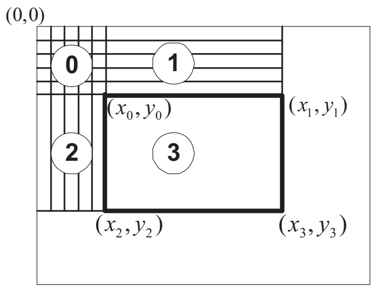

Fig. 3. Fast feature calculation. Regions $\langle 0\rangle$, $\langle 1\rangle$, $\langle 2\rangle$, and $\langle 3\rangle$ represent the rectangular integral images from origin (0, 0) to the coordinates of (x_0,y_0), (x_1,y_1), (x_2,y_2), and (x_3,y_3).

2.1.1. *Pre-processing*

Because our feature window (16×16) is fixed, we need to scale the source images down to detect faces which originally are large in size. The down-sampling uses simple linear image interpolation technique with a factor of 1.2. The most important thing during this step is to calculate the integral images and the lighting corrections. Because the rectangles of the features are different in sizes, the originally simple calculations of rectangular images have time complexity of *O(MN)*, where *M* and *N* are the horizontal and vertical sizes of the rectangle. Viola *et al.* [18] proposed to pre-calculate the integral image of the original image, which means each pixel $II(x,y)$ in the integral image in Fig. 3 represent the intensity summation of the rectangle area from the origin to the coordinate of (x,y) of the original image:

$$II(x,y)=\sum_{u=0}^{x}\sum_{v=0}^{y}S(u,v) \qquad (1)$$

where $S(u,v)$ is the source image intensity for pixel (u,v).

Thus, for example, to calculate the source image intensity summation for the bold-lined rectangle in Fig. 3, we need to calculate $\langle 3\rangle-\langle 1\rangle-\langle 2\rangle+\langle 0\rangle$. The computation complexity then drops to constant time for all different feature sizes. In terms of the integral image, we could calculate the feature rectangle intensity summation *RIS* of the bold-lined rectangle in Fig. 3 as:

$$RIS_{\text{bold-lined}}=II(x_3,y_3)-II(x_1,y_1)-II(x_2,y_2)+II(x_0,y_0) \qquad (2)$$

This constant-time *RIS* calculation is very useful as proved from the software implementations [4, 18]. We

Authorized licensed use limited to: International Institute of Information Technology Bangalore. Downloaded on October 09,2024 at 03:36:33 UTC from IEEE Xplore. Restrictions apply.

adopted this idea into our hardware implementations as well. During *pre-processing*, we also need to calculate each source image pixel's variance to compensate for the lighting differences [3].

2.1.2. *Haar classifier function*

Haar classifier function is the critical part of the whole face detection algorithm. We calculate the features according to (2), and multiply them with the trained *feature weights*. If it is greater than the *classifier threshold*, we accumulate the stage value with *value 1*, otherwise with *value 2*. When each stage is done, we compare the accumulated stage value with the *stage threshold*. If it is greater than the *stage threshold*, we go to the next stage until it passes all the 40 stages. Otherwise, we decide the source image pixel is not a face if it fails in any stage during this step. The *feature weight*, *classifier threshold*, *value 1*, *value 2*, and *stage threshold* are all trained classifier values from the Haar classifier training stage.

2.1.3. *Post-processing*

After the *Haar classifier functions* operated onto the original source image and all scaled images, we cluster the detected face pixels for any adjacent scaled images to be the final detected face rectangle.

2.2. FPGA Implementation Consideration

In order to achieve high performance face detection Haar classifier algorithm with FPGA based accelerator, we need to consider several issues including how to connect the FPGA accelerator with the host microprocessor, how to partition the computations between these two computing platforms, and how to optimize the classifiers for the FPGA.

There are several possible interface points between a general purpose processor and a configurable hardware fabric. The most tightly coupled interface point is to integrate the fabric with the processor pipeline. However, this approach, even though gives the best latency, requires modification to the processor core. One can also interface the fabric with the processor through the processor bus such as the front side bus (FSB) of an Intel processor. However, as FSB frequency continues to grow, building an FSB module requires extensive I/O signaling design. Moreover, Suh *et al.* [21] pointed out that the execution speed from standard benchmarks for Intel's multiprocessor systems with coherency traffic is even slower than the ones without coherency protocols. This means that the FSB interface does not utilize the FSB bandwidth well, unless we disable the FSB's coherency protocol enforcement for specific memory accesses. Interfacing these two building blocks through a standard I/O channel such as the PCIe bus has the highest latency. But latency is not critical in this particular application. Moreover our main goal is to evaluate the effectiveness of using configurable fabric to accelerate algorithms in particular non-systolic algorithms. Our result with PCIe-based FPGA implementation can be used to extrapolate designs with other interfaces. Based on this consideration, we chose a commercial FPGA PCIe card as our accelerating platform, which is from HiTech Global's LX110T. It uses Xilinx's Virtex 5 LX110T with embedded PCIe and DSP (digital signal processing) fabric inside the FPGA.

In order to achieve high performance of computation with FPGA, Herbordt *et al.* [22] summarized four important things the designers need to accommodate into FPGA implementations. What we did with our FPGA implementations of face detection algorithms echoes with their ideas. First, we reconstructed the software-based face detection application. Because the Haar classifier function costs more than 95% of the total time, we populated only the *Haar classifier function* step onto the FPGA board, and left the *pre-processing* and *post-processing* on the host PC (personal computer). We also changed the data flow (loop cycles) from looping for every classifier per pixel, to looping for every stage per pixel.

Second, in order to reduce resources requirement and use FPGA's intrinsic parallel granularity, we replace all FP (floating point) operations with integer computations. With such transformation, we could utilize Xilinx Virtex 5's embedded DSP48E building blocks to accelerate the multiplications and additions in the *Haar classifier function*. A single cycle of FPGA operation represents 100× to 1000× of software clock ticks to achieve the same functionality. This lower data precision does not affect the final detection accuracy.

Third, we employ extensive pipelining to increase algorithm level parallelism and to maintain frequency. Our FPGA implements more than 16 pipeline stages for the Haar classifier functions. We then tried to match the 17×17 pixel sub-window with 16*N* classifiers for each stage, where *N* is an integer. For example, the first stage has 16 classifiers, compared with three classifiers from software version. Although each pixel needs more computations to pass the first stage in the FPGA implementation, however, the first stage in FPGA dropped more than 90% of the total non-face pixels, compared with 50% drop rate for the software version.

Fourth, we design our FPGA implementation for the Haar classifier function with reuse in mind. It consists of intellectual property (IP) building blocks which could be used for other applications with similar time-consuming Haar functions. The data path units, such as the multiplier and adders, can be easily populated to other implementations too. And the design files could be easily scaled for different parameters.

3. IMPLEMENTATION

3.1. SYSTEM SETUP

Fig. 4 illustrates the system architecture, in which the host microprocessor executes the *pre-processing* and sends the integral image and image variances to the FPGA accelerator through PCIe bus. The FPGA stores a 32×32 image buffer, proceeds with the *Haar classifier function*, and then sends the detected faces' coordinates back to host microprocessor through PCIe. The host microprocessor finishes the face detection algorithm with the *post-processing*.

Authorized licensed use limited to: International Institute of Information Technology Bangalore. Downloaded on October 09,2024 at 03:36:33 UTC from IEEE Xplore. Restrictions apply.

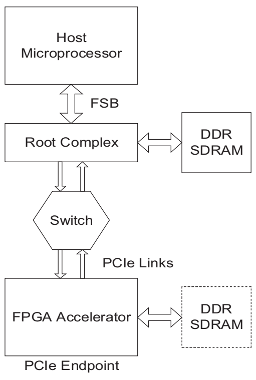

Fig. 4. System architecture of the PCIe based FPGA accelerator.

3.2. FPGA implementation details

As shown in Fig. 5, we store the integral image into the BRAM (Block RAM) as a 32×32 array. During *pixel scan mode*, in each cycle, FPGA reads out one line of the integral image to the 17×17 buffer sub-window, and latches the buffer sub-window data to the 17×17 image window every 17 cycles as the source image's integral sub-window to compute with the classifiers. We have a 289:12 MUX (multiplexer) to select 12 data from the 17×17 image window according to the feature coordinates stored in the BRAM. The selected features and the image variance (or norms) are fed into the classifier engine to compute the class value to compare with the *stage threshold* every 17 clock cycles to decide if the pixel passes the stage or not. The image variance is stored in the BRAM too. As illustrated in Fig. 8, we expect the pixel could not pass the first stage, and this *pixel scan mode* continues from one pixel to the next pixel (offset is one). After 28 clock delays for the first pixel in this mode, the pipeline goes to a loop with 16 clock cycles for a continuous Haar classifier function computations.

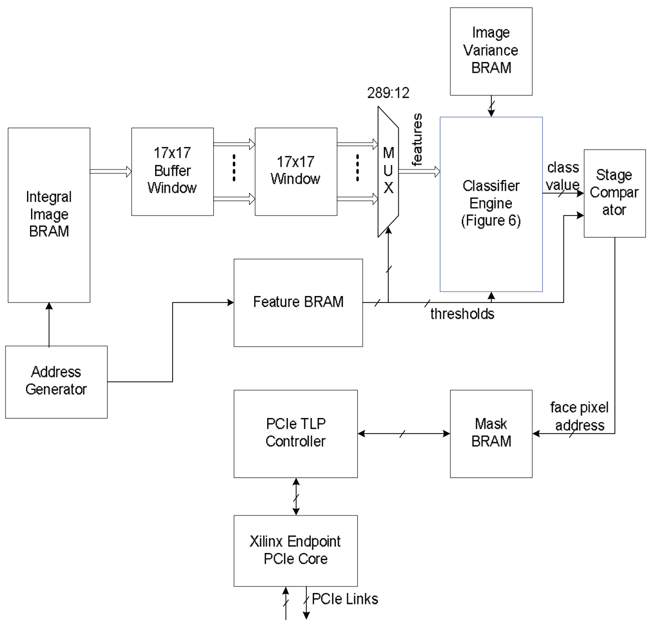

Fig. 5. FPGA block diagram

Fig. 6. If any pixel passes the first stage, the *pixel scan mode* will become *stage scan mode*, in which FPGA retrieves back the pass-first-stage pixel's index and goes to the second stage and so on to decide if this pixel could pass all of 40 stages. During this time, FPGA dedicates its resources to this specific pixel, buffers the following classifier data into the classifier engine, and stops buffering the next pixel integral sub-window. Because more than 90% of pixels could not pass the first stage, the processing time for each pixel in the *pixel scan mode* is approximately 17 clock cycles.

Fig. 6 illustrates the internal implementations of the classifier engine of Fig. 5. During each cycle, 12 integral data from the 17×17 image window and classifier parameters feed into the classifier engine and finish calculating the class value output. The first pixel latency for the classifier engine is six clock cycles.

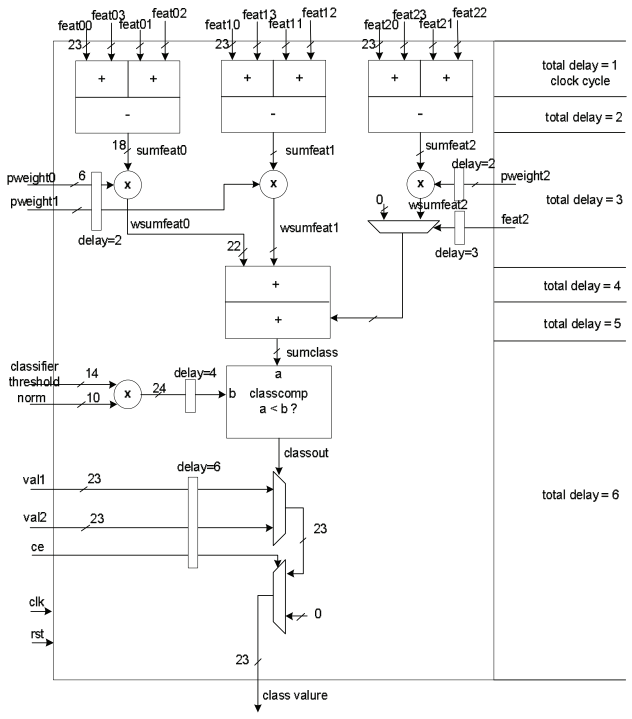

Fig. 7. Classifier block diagram. The value of total delay means the clock cycle delay from the input signal to the current stage of computation.

We also designed, simulated, and synthesized another more resource aggressive Haar stage (classifier) engine for Virtex 5 LX330T. Because it occupies 290% of Virtex 5 LX110T's resources (registers), we did not populate our design onto our currently used HiTech Global Virtex 5 LX110T PCIe card. Fig. 7 shows the aggressive stage engine for the Haar classifier function, where FPGA computes 16 classifiers at one clock cycle. As we could image, we need 192 integral image data to feed into the stage engine. We also need to latch 17×17 pixel sub-window in every clock cycle. This requires a higher memory bandwidth and more MUX's to retrieve those features from the integral image sub-window. We could successfully synthesize the design onto the Virtex 5 LX330T device. We plan to continue our research for this aggressive stage engine design in the future when the hardware platform is obtained. The synthesis result

Authorized licensed use limited to: International Institute of Information Technology Bangalore. Downloaded on October 09,2024 at 03:36:33 UTC from IEEE Xplore. Restrictions apply.

gives us some understanding of the hardware resource requirement for tradeoff comparison. It also proves that our design is scalable with hardware resources.

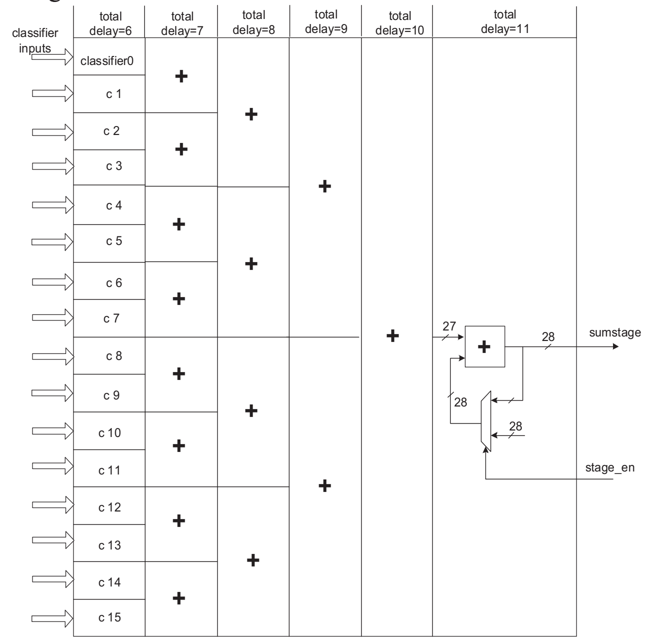

Fig. 8. Stage engine block diagram for the aggressive version of face detection, which includes 16 parallel classifiers in the front.

4. RESULTS

Table 2. Performance comparison for the SW (software) version, 1-classifier FPGA version, and 16-classifier FPGA version and the time on Haar function and overall application.

	Time (sec)	Speedup (x)
SW Haar	18	1
SW overall	18.9	1
1-classifer FPGA Haar	1.8	10
1-classifier FPGA overall	2.5	8
16-classifier FPGA Haar	0.25	72
16-classifier FPGA overall	0.95	20

Table 3. FPGA resource costs for the 1-classifier and 16-classifier versions.

	Freq (MHz)	LUT (%)	BRAM (%)	DSP48E (%)
1-classifer[1]	125	20	20	6
16-classifier[2]	125	95	30	33

Notes: [1] based on Virtex 5 LX110T; [2] based on Virtex 5 LX330T.

Table 2 and Table 3 list the performance comparisons of software and FPGA implementations for the Haar classifier face detection application, and the resource costs for the two FPGA implementations. The performance comparison baseline is the OpenCV (v1.0) version of the Haar-classifier based face detection software, running on a workstation system with Intel 2.66 GHz Core 2 Duo CPU and 8 GB memory. The OS for this system is Red Hat Enterprise Linux 5. The 1-classifier FPGA implementation is synthesized and populated onto Virtex 5 LX110T residing on the Hi-Tech Global PCIe card. The 16-classifier FPGA implementation is only simulated and synthesized onto Virtex 5 LX330T device. We resized all the CMU test images to 256×192.

From Table 2, we could see that for the software version, the Haar classifier face detection application could only achieve performance of 5 frames/sec, while for 1-classifier FPGA implementation 37 frames/sec, and 16-classifier FPGA implementation 98 frames/sec. With much fewer resources consumed for the 1-clssifier FPGA implementation than that for the 16-classifier FPGA implementation, we could already achieve real time operations for the face detection applications.

The PCIe implementation (with 8-lane DMA) is at least 500 MB/sec performance for downloading. Thus, the PCIe bus spends 0.055 seconds to download the 94 CMU test images (27.7 MB data) and 0.001 seconds to upload the result face pixels (0.56 MB data). The PCIe transfer time occupies less than 3% of the FPGA Haar classifier operation time, and hence is not the bottleneck in the FPGA accelerator system for the face detection application.

5. CONCLUSION

We have demonstrated that reconfigurable hardware available in an FPGA can adapt to implement the Haar classifier function for face detection applications and achieve real-time performance. Several innovations such as algorithm adaptation for hardware, microarchitecture design using pipelining and high utilization of parallel arithmetic units, all contribute to the speeding up of this non-systolic algorithm. We have also discussed how our approach can be made scalable for configurable hardware with variable resources. This design paves the way for utilizing configurable hardware to accelerate other non-systolic applications.

6. ACKNOWLEDGMENT

The authors are very grateful to our colleagues at Intel: Yangzhou Du, Yimin Zhang, Tao Wang for assisting us with the face-detection software; Nrupal Jani for providing the Linux driver to interface with the HiTech Global card; and Vladimir Dudnik and Alexander Kibkalo for helping us with IPP.

7. REFERENCES

[1] R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan, "Heterogeneous chip multiprocessors," Computer, vol. 38, pp. 32-38, 2005.

[2] Intel, "Intel Integrated Performance Primitives 5.3," 2008, http://www.intel.com/cd/software/products/asmo-na/eng/302910.htm.

[3] R. Lienhart, A. Kuranov, and V. Pisarevsky, "Empirical analysis of detection cascades of boosted classifiers for rapid object detection," Intel MRL Technical Report, vol. MRL-TR-May02, 2002.

[4] R. Lienhart and J. Maydt, "An extended set of Haar-like features for rapid object detection," presented at 2002 International Conference on Image Processing, pp. I-900-I-903 vol.1, 2002.

Authorized licensed use limited to: International Institute of Information Technology Bangalore. Downloaded on October 09,2024 at 03:36:33 UTC from IEEE Xplore. Restrictions apply.

[5] M. H. Yang, D. J. Kriegman, and N. Ahuja, "Detecting faces in images: a survey," IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 24, pp. 34-58, 2002.

[6] C. P. Papageorgiou, M. Oren, and T. Poggio, "A general framework for object detection," presented at Sixth International Conference on Computer Vision, pp. 555-562, 1998.

[7] M. Oren, C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio, "Pedestrian detection using wavelet templates," presented at 1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 193-199, 1997.

[8] D. B. K. Trieu and T. Maruyama, "Implementation of a parallel and pipelined watershed algorithm on FPGA," presented at International Conference on Field Programmable Logic and Applications FPL 2006, Madrid, Spain, pp. 1-6, 2006.

[9] G. Saldana and M. Arias-Estrada, "FPGA-based customizable systolic architecture for image processing applications," presented at ReConfig 2005 (International Conference on Reconfigurable Computing and FPGAs), pp. 1-8, 2005.

[10] K. Irick, M. DeBole, V. Narayanan, R. Sharma, H. Moon, and S. Mummareddy, "A unified streaming architecture for real time face detection and gender classification," presented at International Conference on Field Programmable Logic and Applications FPL 2007, Amsterdam, Netherlands, pp. 267-272, 2007.

[11] P. McCurry, F. Morgan, and L. Kilmartin, "Xilinx FPGA implementation of an image classifier for object detection applications," presented at 2001 International Conference on Image Processing, pp. 346-349, 2001.

[12] C. A. Waring and X. Liu, "Face detection using spectral histograms and SVMs," IEEE Trans. on Systems, Man and Cybernetics, Part B, vol. 35, pp. 467-476, 2005.

[13] G. Wall, F. Iqbal, J. Isaacs, X. Liu, and S. Foo, "Real time texture classification using field programmable gate arrays," presented at 33rd Applied Imagery Pattern Recognition Workshop (AIPR'04), pp. 1-6, 2004.

[14] Y. Wei, X. Bing, and C. Chareonsak, "FPGA implementation of AdaBoost algorithm for detection of face biometrics," presented at 2004 IEEE International Workshop on Biomedical Circuits & Systems, pp. S1.6-17-20, 2004.

[15] M. Hiromoto, K. Nakahara, and H. Sugano, "A specialized processor suitable for AdaBoost-based detection with Haar-like features," presented at IEEE Conference on Computer Vision and Pattern Recognition CVPR'07, pp. 1-8, 2007.

[16] Sourceforge, "Open Computer Vision Library," 2008, http://sourceforge.net/projects/opencvlibrary/.

[17] P. Viola and M. Jones, "Robust real-time face detection," presented at Second International Workshop on Statistical and Computational Theories of Vision - Modeling, Learning, Computing, and Sampling, Vancouver, Canada, pp. 1-25, 2001.

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

[19] T. Mita, T. Kaneko, and O. Hori, "Joint Haar-like features for face detection," presented at Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on, pp. 1619-1626 Vol. 2, 2005.

[20] CMU, "Frontal Face Images," 2008, http://vasc.ri.cmu.edu/idb/html/face/frontal_images/.

[21] T. Suh, S.-L. L. Lu, and H.-H. S. Lee, "An FPGA approach to quantifying coherence traffic efficiency on multiprocessor systems," presented at International Conference on Field Programmable Logic and Applications FPL 2007, Amsterdam, Netherlands, pp. 47-53, 2007.

[22] M. C. Herbordt, T. VanCourt, Y. Gu, B. Sukhwani, A. Conti, J. Model, and D. DiSabello, "Achieving high performance with FPGA-based computing," Computer, vol. 40, pp. 50-57, 2007.

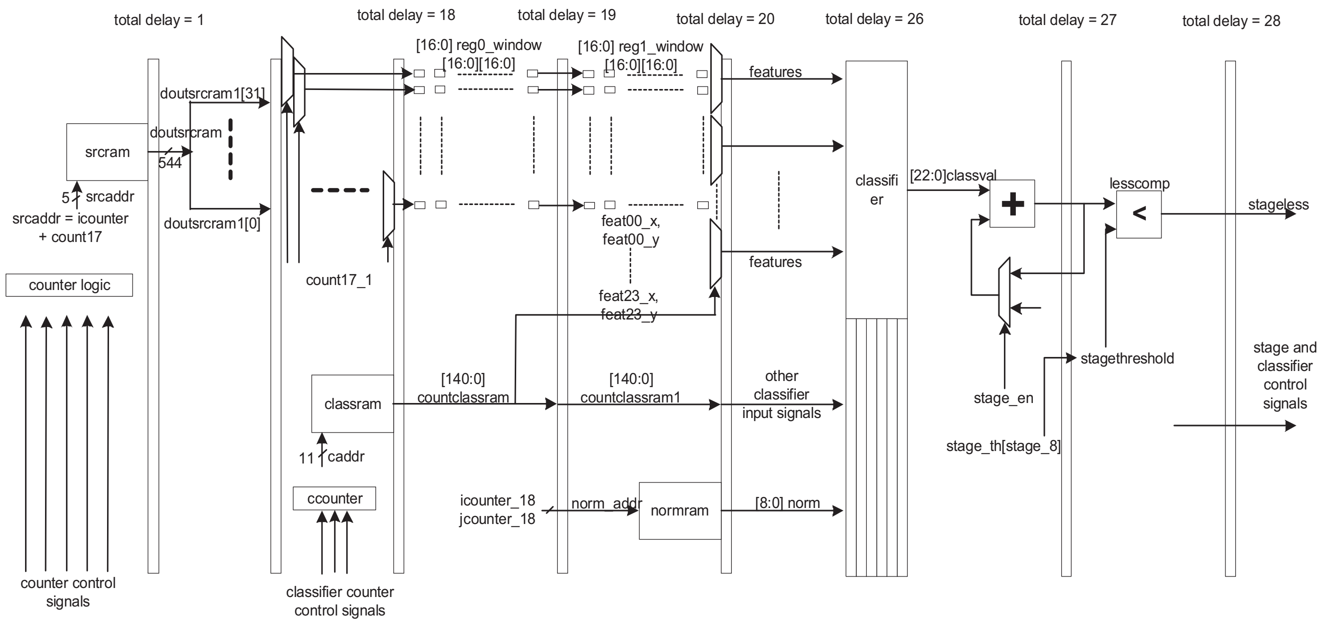

Fig. 9. Pipelined design of FPGA face detection accelerator block diagram, where total delay is the latency for the first pixel; srcram is the source integral image BRAM; classram is the classifier features BRAM; normram is the image variance BRAM; classifier is the classifier computational engine; lesscomp is the stage threshold comparing engine.

Authorized licensed use limited to: International Institute of Information Technology Bangalore. Downloaded on October 09,2024 at 03:36:33 UTC from IEEE Xplore. Restrictions apply.