Parallelized Architecture of Multiple Classifiers for Face Detection

Junguk Cho[†], Bridget Benson[†], Shahnam Mirzaei[‡], Ryan Kastner[†]

***Abstract*—This paper presents a parallelized architecture of multiple classifiers for face detection based on the Viola and Jones object detection method. This method makes use of the AdaBoost algorithm which identifies a sequence of Haar classifiers that indicate the presence of a face. We describe the hardware design techniques including image scaling, integral image generation, pipelined processing of classifiers, and parallel processing of multiple classifiers to accelerate the processing speed of the face detection system. Also we discuss the parallelized architecture which can be scalable for configurable device with variable resources. We implement the proposed architecture in Verilog HDL on a Xilinx Virtex-5 FPGA and show the parallelized architecture of multiple classifiers can have 3.3× performance gain over the architecture of a single classifier and an 84× performance gain over an equivalent software solution.**

I. INTRODUCTION

Face detection is the act of determining the location and sizes of faces in an image. It is an active area of research in the image processing field over the past years due to its potential applications in monitoring and surveillance [1], human computer interfaces [2], smart rooms [3], intelligent robots [4], and biomedical image analysis [5]. Numerous approaches have been proposed for face detection in images. Simple features such as color, motion, and texture are used for the face detection in early researches. However, these methods typically break down easily in real world situations due to the complexity of the image background.

The Viola and Jones method for object detection [6] is the first and one of the most popular techniques for real-time face detection. Their approach utilizes pattern classification to determine the existence of a face. More specifically they search for a sequence of Haar features that indicate the presence of a face. This algorithm requires considerable computational power due to the sheer number of Haar features that must be identified to detect a face. One face is comprised of a substantial amount of features, which typically computed over a window of 24×24 pixels. Even a small window can generate a substantial number of features, e.g. a 24×24 window contains over 180,000 Haar features. Therefore, fast and accurate detection requires careful selection of the features to search for, which is performed using the AdaBoost algorithm [7]. And while every window does not typically require the computation of all features – the detection is performed in stages and the absence of something that looks similar to a face is designed to terminate quickly – this is still a substantial amount of computation. Therefore, this constitutes a bottleneck to the application of object detection in real time.

In this paper, we present a parallelized architecture of multiple classifiers for real-time face detection. We propose hardware design techniques to accelerate the processing speed of face detection. The face detection system generates an integral image window during one clock cycle. Then it performs classification operations in parallel using multiple classifiers to detect a face in the image sequence. The main contribution of our work is design and implementation of a physically feasible hardware system to accelerate the processing speed of the operations required for real-time face detection. Therefore, this work has resulted in the development of a complete real-time face detection system employing an FPGA implemented system designed by Verilog HDL. Its performance has been measured and compared with an equivalent software implementation. The system is fully-functional; it can interface with a variety of cameras and output the results to a display.

This paper is organized as follows: In Section 2, we explain the face detection algorithm and review the related work found in literature in hardware implementations of face detection. In Section 3, we describe the hardware architecture, designed with Verilog HDL, of a face detection system using block diagrams. We also present the implementation of the real-time face detection system in an FPGA. In Section 4, we show the corresponding performance. Finally, we conclude in Section 5.

II. FACE DETECTION

A. Face Detection Algorithm

The Viola and Jones [6] face detection algorithm is used as the basis of our design. The face detection algorithm looks for specific Haar features of a human face. When one of these features is found, the algorithm allows the face candidate to pass to the next stage of detection. A face candidate is a rectangular section of the original image called a sub-window. Generally these sub-windows have a fixed size (typically 24×24 pixels). This sub-window is often scaled in order to obtain a variety of different size faces. The

algorithm scans the entire image with this window and denotes each respective section a face candidate [6].

The algorithm uses an integral image in order to process Haar features of a face candidate in constant time. It uses a cascade of stages which is used to eliminate non-face candidates quickly. Each stage consists of many different Haar features. Each feature is classified by a Haar classifier. The Haar classifiers generate an output which can then be provided to the stage comparator. The stage accumulator sums the outputs of the Haar classifiers and compares this value with a stage threshold to determine if the stage should be passed. If all stages are passed the face candidate is concluded to be a face. These terms will be discussed in more detail in the following sections.

1) Integral Image: The integral image is defined as the summation of the pixel values of the original image. The value at any location (x,y) of the integral image is the sum of the image's pixels above and to the left of location (x,y). Figure 1 illustrates the integral image generation. The shaded region represents the sum of the pixels up to position (x,y) of the image. A 3×3 image with unit values for pixels and its corresponding integral image representation are shown on the right.

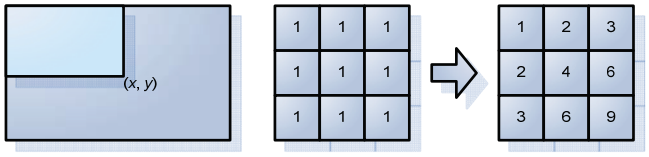

Figure 1. Integral image generation.

2) Haar Feature: Haar features are composed of either two or three rectangles. Face candidates are searched for Haar features of the current stage. The weight and size of each feature and the features themselves are generated using a machine learning algorithm from AdaBoost [6][7]. The weights are constants generated by the learning algorithm. There are a variety of forms of features as seen below in Fig. 2. Each Haar feature has a value that is calculated by taking the area of each rectangle, multiplying each by their respective weights, and then summing the results. The area of each rectangle is easily found using the integral image. By using each corner of a rectangle, the area can be computed quickly as denoted by Fig. 3. Since L1 is subtracted off twice it must be added back on to get the correct area of the rectangle. The area of the rectangle R, denoted as the rectangle integral, can be computed as follows using the locations of the integral image: L_4-L_3-L_2+L_1, as shown in Fig. 3.

3) Classifier: A Haar classifier uses the rectangle integral to calculate the value of a Haar feature. The Haar classifier multiplies the weight of each rectangle by its area and this result is compared with the threshold. The value of a Haar feature is added together. Several Haar classifiers compose a stage. A stage accumulator sums all the Haar classifier results in a stage and a stage comparator compares this summation with a stage threshold. The threshold is also a constant obtained from the AdaBoost algorithm. Each stage does not have a set number of Haar features. Depending on the parameters of the training data individual stages can have a varying number of Haar features. For example, Viola and Jones' data set used 2 features in the first stage and 10 in the second. All together they used a total of 38 stages and 6060 features [6]. Our data set is based on the OpenCV data set which used 22 stages and 2135 features in total [8][9].

Figure 2. Two examples of Haar features.

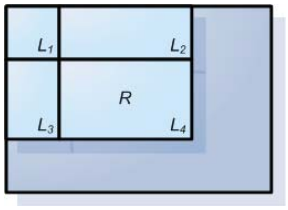

Figure 3. Calculating the area of a rectangle R.

4) Cascade: The Viola and Jones face detection algorithm eliminates face candidates quickly using a cascade of stages. The cascade eliminates candidates by making stricter requirements in each stage with later stages being much more difficult for a candidate to pass. Candidates exit the cascade if they pass all stages or fail any stage. A face is detected if a candidate passes all stages. This process is shown in Fig. 4.

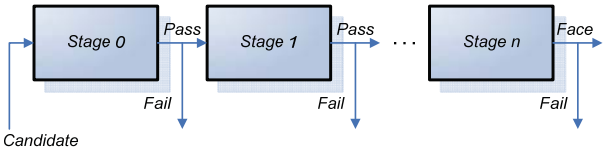

Figure 4. Cascade of stages.

B. Related Work

Viola and Jones [6] proposed a rapid and robust face detection method. This method utilized the AdaBoost algorithm [7], which identifies a sequence of Haar classifiers that indicate the presence of a face. Mita el al. [10] also proposed such Haar classifier face detection algorithm with more Haar features in each classifier stage to improve the detection rate and to lower false alarm rate at the same time. Lienhart et al. [11] were the first to introduce the face detection algorithm into Intel Integrated Performance Primitives, which was later included in OpenCV.

Almost all of the available literatures on real-time face detection are theoretical or describe a software implementation. Only a few papers have addressed a hardware design and implementation of real-time face detection. Theocharides et al. [12] presented the

implementation of a neural network based face detection in an ASIC to accelerate processing speed. However, VLSI technology requires a large amount of development time and cost. Also it is difficult to change design. McCready [13] designed and implemented face detection for the Transmogrifier-2 configurable hardware system. This implementation utilized nine FPGA boards. Sadri et al. [14] implemented neural network based face detection on the Virtex-II Pro FPGA. Skin color filtering and edge detection are used to reduce the processing time. However, some operations are implemented on hardcore PowerPC processor with embedded software. Wei et al. [15] presented FPGA implementation for face detection using scaling input images and fixed-point expressions. However, the image size is too small (120×120 pixels) to be practical and only some parts of classifier cascade are actually implemented. A low-cost detection system was implemented using a Cyclone II FPGA by Yang et al. [16]. The frame rate of this system is 13 fps with low detection rate of about 75%. Nair et al. [17] implemented an embedded system for human detection on an FPGA. It can process the images at speeds of 2.5 fps with about 300 pixels images. Gao et al. [18] presented an approach to use an FPGA to accelerate Haar feature classifier based face detection. They re-trained the Haar classifier with 16 classifiers per stage. However, only classifiers are implemented in the FPGA. The integral image generation and detected face display are processed in a host microprocessor. Also the largest Virtex-5 FPGA was used for the implementation because the design size was too large. Hiromoto et al. [19] implemented real-time object detection based on the AdaBoost algorithm. They proposed a hybrid architecture of a parallel processing module for the former stages and a sequential processing module for the subsequent stages in the cascade. Since the parallel processing module and the sequential processing module are divided after evaluating a processing time with fixed Haar feature data, it should be designed and implemented again in order to apply new Haar feature data. Also the experimental result and analysis of the implemented system are not discussed. Lai et al. [20] presented a hardware architecture design on an FPGA based on the AdaBoost algorithm for face detection. It can achieve theoretical 143 fps detection for 640×480 images. However, they implemented only 52 classifiers in single stages. Because of the small number of classifiers, its results show lower detection rate and higher false alarm rate than OpenCV's 3125 classifier implementation.

III. HARDWARE ARCHITECTURE

A. System Overview

We proposed a parallelized architecture of multiple classifiers for a real-time face detection system. Figure 5 shows the overview of the proposed architecture for face detection. It consists of seven modules: image interface, frame grabber, image store, image scaler, Haar classifier, display, and DVI interface. The image interface and DVI interface are implemented using ASIC custom chips with the FPGA board. The others are designed using Verilog HDL and implemented in an FPGA in order to perform face detection in real-time.

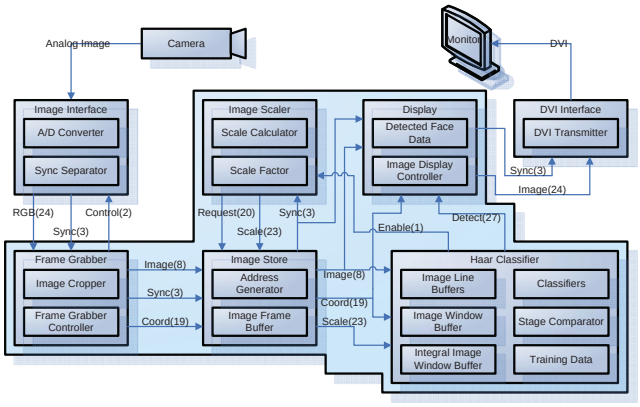

Figure 5. Block diagram of proposed face detection system.

B. FPGA Architecture

1) Frame Grabber: In the frame grabber module, the frame grabber controller generates the control signals for controlling the A/D converter which converts the analog image signals into digital image data, and the sync separator which generates the image sync signals in the image interface module. The color image data and sync signals are transferred from the image interface module to the frame grabber module. The image cropper crops the images based on the sync signals. These image data and sync signals are used in all of the modul*es of the face detection system.*

2) Image Store: The image store module stores the image data arriving from the frame grabber module frame by frame. This module transfers the image data to the Haar classifier module based on the scale information from the image scaler module. The image store module uses BRAMs of FPGA. It can store the image of a frame. The size of BRAMs can be scaled for the source image resolution.

3) Image Scaler: The images are scaled down based on a scale factor by the image scaler module. The image scaler module generates and transfers the address of the BRAMs containing a frame image in the image store module to request image data according to a scale factor. The image store module transfers a pixel data to the Haar classifier module based on the address of BRAMs required from the image scaler module.

*4) Haar Classifier: Th*e Haar classifier module performs the classification for the face detection. It is the critical module of the whole face detection system. This module consists of the image line buffers, image window buffer, integral image window buffer, line buffer controller, and window buffer controller to generate the integral image window, classifiers, training data, feature counter, stage accumulator, stage comparator, and stage training data to perform the classification as shown in Fig. 6.

The face detection is performed by Haar classification using an integral image. The integral image generation requires substantial computation. A general purpose computer of Von Neumann architecture has to access image memory at least *width*×*height* times to get the value of each

pixel when it processes an image with *width*×*height* pixels. It may take a long latency delay every frame. In order to reduce memory access and processing time, we propose a specific architecture for the integral image generation. This architecture stores the necessary pixels for processing each pixel and its neighboring pixels together. The image line buffers store some parts of the image. The image line buffers use dual port BRAMs where the number of BRAMs is the same as that of the row-1 in the image window buffer. Each dual port BRAM can store one line of an image. Thus, the *x*-coordinates of the pixels can be used as the address for the dual port BRAM. For the incoming pixel where the coordinate is (*x*,*y*), the image line buffers controller performs operations such as in (1), where *n* is the image window row size, *p*(*x*,*y*) is the incoming pixel value, and *L*(*x*,*y*) represents each pixel in the image line buffer.

$$L(x, y-k) \quad L(x, y-(k-1)) \quad \text{where } 1 \le k \le n-2 \qquad (1)$$
$$L(x, y-k) \quad p(x, y), \text{ where } k \quad 0.$$

With these operations, the pixel values in the lines of an image are stored in dual port BRAMs. Since each dual port BRAM stores one line of an image, it is possible to get one pixel value from every line simultaneously.

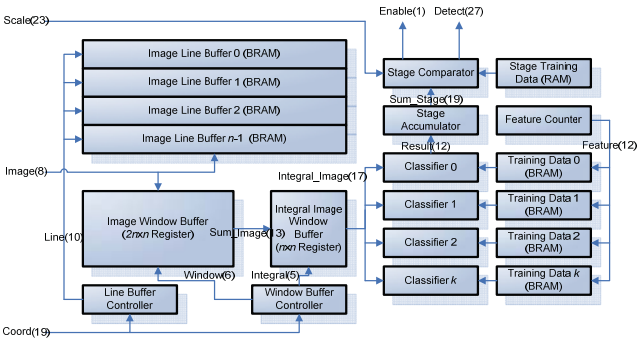

Figure 6. Block diagram of proposed face detection system.

The image window buffer stores pixel values moving from the image line buffer. It has two *n*×*n* windows, actually *n*×2*n* window. The first *n*×*n* window calculates the accumulated values of each column of the image window buffer. Each column except the right-most column has only one adder as shown in Fig. 7. The adder of the most second right column calculates the summation of first row and second row pixel values in the right-most column. The adder of the third right column calculates the summation of the first, second, and third row pixel values in the second right column. Finally, the adder of *n-th* column calculates the summation of all pixel values in the (*n*-1)-*th* column. The pipeline scheme is applied in this part, so the latency of first summation of all pixel values in the column is n clock cycles. The second *n*×*n* window latches and moves the accumulated values of the column to the adjacent column. The accumulated values are used to generate the integral image window. Since pixels of an image window buffer are stored in registers, it is possible to access all pixels in the image window buffer simultaneously to generate the integral image window. For the incoming pixel with coordinate (*x*,*y*), the image window buffer controller performs operation as in (2) where *n* and 2*n* are the row and column size of the image window buffer, respectively. *p*(*i*,*j*) is the incoming pixel value in the image window buffer; *p*(*x*,*y*) is the incoming pixel value; *I*(*i*,*j*) represents each of the pixels in the image window buffer; and *L*(*x*,*y*) represents each of the pixels in the image line buffer.

$$I(i-k, j) \quad I(i-(k-1), j) \quad \text{where } 1 \le k \le 2n-1, 1 \le l \le n-1 =$$
$$I(i, j-l) \quad L(x, y-(l-1)) \quad \text{where } k = 0, 1 \le l \le n-1 \quad = \qquad (2)$$
$$I(i-k, j-l) \quad p(i,k) \quad p(x, y), \text{ where } k \quad l \quad 0.$$

The accumulation of pixels in one column is calculated in the image window buffer as in (3). when $k + 1 \quad n-1$,

$$I(i-k, j \quad 1) \quad I(i-(k-1), j \quad 1) + I(i-(k-1), j \quad (l+1)), \qquad (3)$$
where $1 \le k \le n-1, 0 \le l \le n-2$.

The integral image window buffer calculates the integral image value of *n*×*n* window using the accumulated value of the column in the image window buffer. Every cell of the integral image window has one adder and one subtractor as shown in Fig. 7. Each cell of the integral image window adds the previous integral values to the accumulated values from the image window buffer, and subtracts the accumulated values from the left-most column of the image window buffer. Using this mechanism and architecture, we can generate the integral image of current *n*×*n* window during one clock cycle. Since pixels of an integral image window buffer are stored in registers, it is possible to access all integral pixels in the integral image window buffer simultaneously to perform the Haar classification. For incoming pixel with coordinate (*i*,*j*), the integral image window buffer controller performs operation as in (4) where n is the row and column size of the integral image window buffer. *II*(*s*,*t*) represents each of the integral pixels in the integral image window buffer; and I(*i*,*j*) represents each of the pixels in the image window buffer.

$$II(s-u, t-v) \quad II(s-u, t-v) + I(i-k, j-l) - I(i-(2n-1), j \quad l), \qquad (4)$$
where $0 \le u \le n-1$, $0 \le v \le n-1$, $0 \le l \le n-1 =$
$n-1 \le k \le 2n-2$.

Figure 7 shows all of the actions in the proposed architecture to generate the integral image. The contents of the image line buffers, image window buffer, and integral image window buffer are updated according to any stage fail signal or the all stages pass signal from the stage comparator. So while the Haar classification is processing, they maintain their value corresponding the current window. For every image from the frame grabber module, the integral image of the current window is calculated to perform the Haar classification using the integral image.

A Haar classifier has a Haar feature which consists of two or three rectangles and their weight, threshold, and left and right values. Each rectangle presents four points, *x*, *y*, *x+width*, *y+height*, using the coordinates (*x*,*y*) of most left and up point, width, *width*, and height, *height*, as shown in Fig. 8. The integral image value of each rectangle can be calculated using these points from the integral image window buffer as shown in Fig. 9. Since integral pixel values in an integral image window buffer are stored in registers, it is possible to access all integral pixel values in the integral image window buffer simultaneously to calculate the integral image value of the rectangles of the Haar classifier. It enables us to save the memory access time.

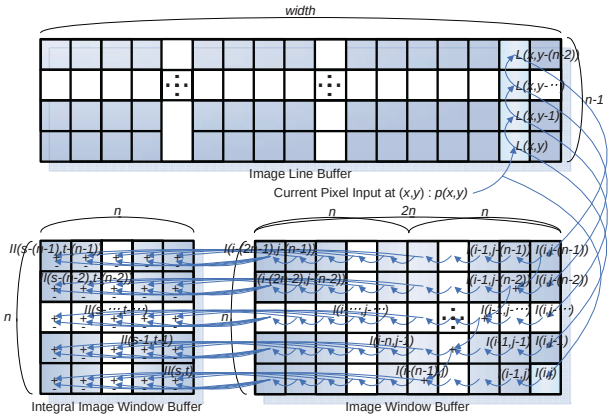

Figure 7. Architecture for generating integral image window.

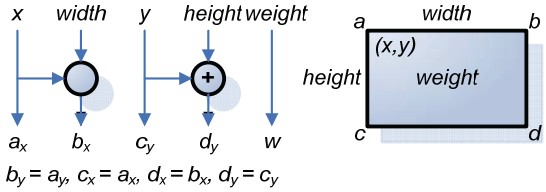

Figure 8. Haar feature calculation of Haar classifier.

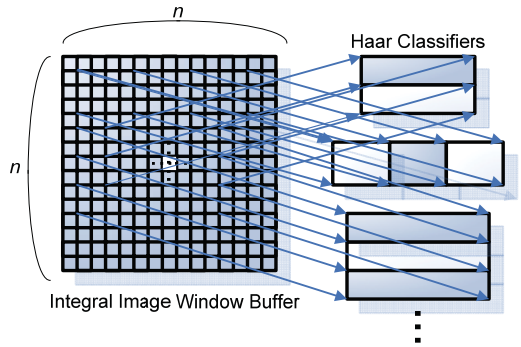

Figure 9. Simultaneous access to integral image window in order to calculate integral image of Haar classifiers.

Figure 10 shows the architecture of a Haar classifier for face detection. All training data are stored in the BRAMs. Four points, *x*, *y*, *x+width*, *y+height*, of one rectangle of the Haar feature are calculated by the method as shown in Fig. 8. The integral image values of Haar classifier are obtained from the integral image window buffer as shown in Fig. 9. Integral image value of each rectangle of Haar feature multiplies with its weight, *weight*. The summation of all integral image values multiplied by their weight is the result of one Haar classifier. This result is compared with the threshold, *threshold*. If the result is smaller than the threshold, the final resultant value of this Haar classifier is the left value, *left*. Otherwise, the final resultant value is the right value, *right*. This final resultant value is accumulated during the same stage. The accumulated value of the stage is compared with the stage threshold, *stage threshold*, when each stage is done. If the accumulated value is larger than the stage threshold, it goes to the next stage and so on to decide if this image window could pass all stages. Otherwise, this image window is not a face if it fails in any stage. The proposed architecture of the Haar classifier is implemented based on a pipeline scheme as shown in Fig. 10. During each clock cycle, the integral image values of Haar classifier from the integral image window buffer and the training data of Haar classifier from the training data BRAMs are fed to calculate the result of classification continuously. The latency for the first Haar classifier is five clock cycles.

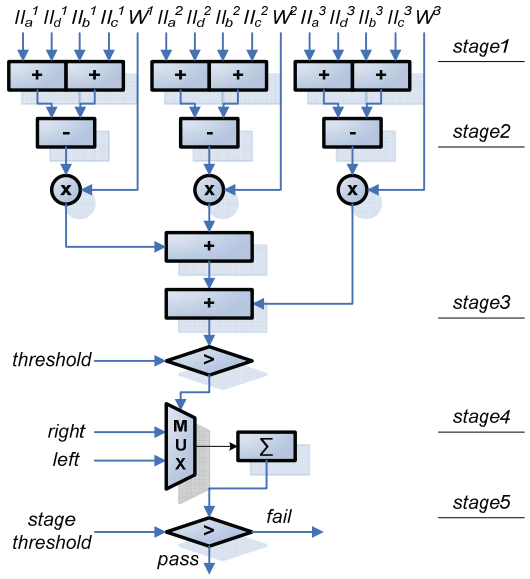

Figure 10. Architecture for performing Haar classification.

5) Display: The display module stores the information of the detected faces which are transferred from the classifier module. It displays the white squares on the faces in the image sequence by converting pixel value to white color according to the information of the detected faces. In the display module, the Digital Visual Interface (DVI) specification is applied to display the processed image sequence to the LCD monitor through a DVI transmitter in the DVI interface module. This module generates the sync signals and image data for the DVI transmitter using the image data and sync signals from the frame grabber module.

C. FPGA Implementation

The proposed parallelized architecture of multiple classifiers for face detection has been designed using Verilog HDL and implemented in an FPGA. We use the Haar training data from OpenCV to detect the frontal human faces based on the Viola and Jones algorithm [8][9]. This cascade of Haar training data were trained by frontal faces of size 20×20. This cascade includes a total of 22 stages, 2135 Haar classifiers, and 4630 Haar features. Table 1 shows the number of Haar classifiers in each stage.

TABLE I. NUMBER OF HAAR CLASSIFIERS IN EACH STAGE

Stage	Number of Classifier	Stage	Number of Classifier	Stage	Number of Classifier
0	3	8	56	16	140
1	16	9	71	17	160
2	21	10	80	18	177
3	39	11	103	19	182
4	33	12	111	20	211
5	44	13	102	21	213
6	50	14	135	**Total**	**2135**

The proposed face detection is performed in two major parts. The first part is grabbing and scaling. This part consists of the frame grabber, image store, and image scaler modules. These modules are for grabbing images and generating scaled images. Sub-windows for the Haar classifier are expanded to detect large objects in Viola and Jones object detection algorithm. Since the Haar feature classifier consists of simple rectangles, scaling a sub-window is not hard. Therefore, this method is widely used for software object detection implementation. However, the larger cache memory of the integral image is required according to the larger size of a sub-window to achieve fast memory access, which is difficult to implement in hardware. A scaling image technique is used in hardware instead of the scaling sub-window because it does not need a huge cache memory for fast memory access and it is easy to implement in hardware. Since our architecture has a 20×20 fixed integral image window, it needs to scale input images down to detect large faces. To make scaled images, we use a nearest neighbor interpolation algorithm with a factor of 1.2. A pixel value in the scaled images is set to the value of the nearest pixel in the original images. This is the simplest interpolation algorithm that requires a lower computation cost. The number of the scaled images depends on the input image resolution. Our scaler module performs the down-scaling of input images until the height of the scaled image is similar with the size of the integral image window. The scaler module for 320×240 pixels images has 14 scale factors (1.2^0~1.2^{13}), the scaler module for 640×480 pixels images has 18 scale factors (1.2^0~1.2^{17}). The more scale factors need more processing time because of the increase of the candidate windows.

The second part of the face detection system is performing Haar classification using the integral image. This part consists of a Haar classifier module which has the image line buffers, image window buffer, integral image window buffer, classifiers, stage comparator, and training data blocks. Since generating the integral image of the whole scaled image requires substantial computation power and time, we generate the integral image of only the current image window. The image line buffer (19 lines), image window buffer (20×40 cells), and integral image window buffer (20×20 cells) are implemented to generate the integral image of the current window during one clock cycle. The pixel data are stored and moved in the image line buffer according to the mechanism of the architecture explained in the previous section.

We design and implement scalable multiple classifiers (1, 2, 4, 6, 8 classifiers). These classifiers have multiple classifiers which process in parallel. The integral image window buffer can be accessed simultaneously by each classifier because the integral image window stores the integral pixel values in registers. The training data are stored in the BRAMs of an FPGA. The BRAMs for the training data consist of 7 BRAMs: 3 BRAMs for 3 rectangles of Haar feature (*x*, *y*, *width*, *height*, *weight*), 3 BRAMs for the threshold, left and right, respectively, and 1 BRAM for the stage threshold. Although Haar classifiers are composed of either two or three rectangles, all Haar classifiers are uniformed as having only 3 rectangles for hardware implementation. If the Haar classifier has 2 rectangles, the third rectangle has 0 values. These values are called according to the stage and feature numbers. The classifiers module calculates the stage and feature numbers, and then generates the address of the training data BRAMs to read the Haar feature values. In order to implement parallel processing of multiple classifiers, training data should be accessed simultaneously. Since BRAM allows only access to one address, the contents of training data BRAMs are divided and stored in several BRAMs to allow multiple accesses of the training data. We divided the contents of each BRAM into 1, 2, 4, 6, 8 sets of BRAMs for the 1, 2, 4, 6, 8 classifiers, respectively. For example, for 4 classifiers, the first content of training data BRAM is for the first classifier, the second content is for the second classifier, the third content is for the third classifier, and the forth content is for the fourth classifier. Again, the fifth content is for the first classifier, the sixth content is for the second classifier, the seventh content is for the third classifier, and the eighth content is for the fourth classifier. This routine continues until the end of BRAM contents. Therefore, 7 BRAMs are used for each single classifier and total 28 BRAMs are used for the 4 classifiers. Since the quantity of the training data is fixed, the allocated resource for training data BRAMs of the multiple classifiers is the same regardless of the number of the multiple classifiers.

Table 2 shows a comparison of the device utilization characteristics for the parallelized architecture of multiple classifiers for face detection. There are 10 implementations: 1, 2, 4, 6, 8 classifiers for both 320×240 (QVGA) resolution images and 640×480 (VGA) resolution images. The face detection systems of the multiple classifiers are designed using Verilog HDL, synthesized using Synplify Pro, and implemented in Virtex-5 LX330 FPGA using ISE design suite [16].

TABLE II. DEVICE UTILIZATION CHARACTERISTICS FOR THE FACE DETECTION SYSTEMS

320×240 Resolution Images				
Number of Classifiers	***Registers***	***LUTs***	***BRAMs***	***DSP48s***
1	17906	32438	40	7
2	18453	37423	44	10
4	19397	50765	47	16
6	20371	62144	50	22
8	21270	73741	53	28
640×480 Resolution Images				
Number of Classifiers	***Registers***	***LUTs***	***BRAMs***	***DSP48s***
1	18544	33790	96	7
2	19034	38843	100	10
4	20013	51050	103	16
6	20944	63643	106	22
8	21819	74734	109	28

IV. EXPERIMENT/RESULT

We measure the performance of the proposed parallelized architecture of multiple classifiers for face detection. Since the system performance of face detection depends on the number of faces in the images, the implemented face detection systems are tested on 5 images, which contain 1, 3, 6, 9, 12 faces, respectively. Table 3 shows the average performance of the face detection systems which have 1, 2, 4, 6, 8 classifiers, respectively, when they are applied to images consisting of both 320×240 and 640×480 pixels. When applying to the 320×240 resolution images, The 1 classifier face detection system is capable of processing the images at speeds of an average of 18.26 fps. The 2 classifiers face detection system is capable of processing the image at speed of an average of 25.64 fps. The 2 classifiers face detection system has the performance improvement of 1.4 times over the 1 classifier one. The 8 classifiers face detection system is capable of processing the image at speed of an average of 61.02 fps. The 8 classifiers face detection system has the performance improvement of 3.34 times over the 1 classifier one. When applying to 640×480 resolution images, The 1 classifier face detection system is capable of processing the images at speeds of an average of 5.24 fps. The 2 classifiers face detection system is capable of processing the image at speed of an average of 6.84 fps. The 2 classifiers face detection system has the performance improvement of 1.3 times over the 1 classifier one. The 8 classifiers face detection system is capable of processing the image at speed of an average of 16.08 fps. The 8 classifiers face detection system has the performance improvement of 3.06 times over the 1 classifier one. This is due to the concurrent operations of multiple classifiers by the parallelized architecture for face detection. Although the usage of the system resource increases, the system performance increases dramatically.

The performance of the equivalent software implementation is determined by measuring the computation time required for performing face detection on the PC; in this case using a Intel Core 2 Quad CPU (2.4 GHz), 8 GB DDR2 SDRAM (800 MHz), Microsoft Windows Vista Business (64-bit), and Microsoft Visual Studio. All of the software programs are developed using Microsoft Visual C++. The algorithm and parameters used in software face detection are exactly the same as the one of hardware face detection. When the face detection system, using the software program, is applied to the same conditions as the hardware face detection, it is capable of processing the images at speeds of an average of 0.72 fps when applied to the 320×240 resolution images and 0.43 fps when applied to the 640×480 resolution images. In order to make a fair comparison, any techniques such as detecting skin color or motion, down-sampling images, and decreasing scale factors, are not applied to the software implementation. The hardware face detection systems has the performance improvement up to 84.75 times the software face detection system with the 320×240 resolution images and up to 37.39 times the software face detection system with the 640×480 resolution images.

Figure 11 shows the experimental result of the proposed face detection system. The white squares present the detected face on the image. It shows that the face can be detected successfully.

TABLE III. PERFORMANCE OF PROPOSED FACE DETECTION SYSTEMS

Number of Classifiers	**320×240 Pixels Images**	**Improvement**	**640×480 Pixels Images**	**Improvement**
S/W 1	1,373ms (0.72 fps)	1.00	2,319 ms (0.43 fps)	1.00
H/W 1	54.735 ms (18.26 fps)	25.36	190.541 ms (5.24 fps)	12.18
H/W 2	38.997 ms (25.64 fps)	35.61	146.033 ms (6.84 fps)	15.90
H/W 4	24.405 ms (40.97 fps)	56.90	81.499 ms (12.27 fps)	25.20
H/W 6	21.053 ms (47.49 fps)	65.95	62.154 ms (16.08 fps)	28.53
H/W 8	16.387 ms (61.02 fps)	84.75	62.154 ms (16.08 fps)	37.39

Figure 11. Experimental result of face detection systems.

V. CONCLUSION

We present a parallelized architecture of multiple classifiers for face detection based on the Viola and Jones object detection method. This method makes use of the AdaBoost algorithm, which identifies a sequence of Haar classifiers that indicate the presence of a face. In our architecture, the scaling image technique is used instead of the scaling sub-window, and the integral image window is generated per window instead of per image during one clock cycle. The Haar classifier is designed using a pipelined scheme, and the multiple classifiers which have 1, 2, 4, 6, 8 classifiers processed in parallel is adopted to accelerate the processing speed of the face detection system. Also we discuss the parallelized architecture which can be scalable for configurable devices with variable resources. We implement the proposed architecture in Verilog HDL on a Xilinx Virtex-5 FPGA and show the parallelized architecture of multiple classifiers can have 3.3× performance gain over the architecture of a single classifier and an 84× performance gain over an equivalent software solution. This enables real-time operation (>60 frames/sec on QVGA video, >15 frames/sec on VGA video).

REFERENCES

[1] Z. Guo, H. Liu, Q. Wang, and J. Yang, "A Fast Algorithm of Face Detection for Driver Monitoring," *In Proceedings of the International Conference on Intelligent Systems Design and Applications*, vol.2, pp.267 - 271, 2006.

[2] M. Yang, N. Ahuja, "Face Detection and Gesture Recognition for Human-Computer Interaction," *The International Series in Video Computing*, vol.1, Springer, 2001.

[3] Z. Zhang, G. Potamianos, M. Liu, T. Huang, "Robust Multi-View Multi-Camera Face Detection inside Smart Rooms Using Spatio-Temporal Dynamic Programming," *In Proceedings of the International Conference on Automatic Face and Gesture Recognition*, pp.407-412, 2006.

[4] W. Yun; D. Kim; H. Yoon, "Fast Group Verification System for Intelligent Robot Service," *IEEE Transactions on Consumer Electronics*, vol.53, no.4, pp.1731-1735, 2007.

[5] V. Ayala-Ramirez, R. E. Sanchez-Yanez and F. J. Montecillo-Puente "On the Application of Robotic Vision Methods to Biomedical Image Analysis," *In IFMBE Proceedings of Latin American Congress on Biomedical Engineering*, pp.1160-1162, 2007.

[6] P. Viola and M. Jones, "Robust Real-Time Face Detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137-154, 2004.

[7] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generaliztion of On-Line Learning and an Application to Boosting," *Journal of Computer and System Sciences*, no. 55, pp. 119-139, 1997.

[8] Open Computer Vision Library, , March. 2009. DOI:http://sourceforge.net/projects/opencvlibrary/.

[9] G. Bradski and A. Kaehler, "Learning OpenCV: Computer Vision with the OpenCV Library," O'Reilly Media, Inc., 2008.

[10] T. Mita, T. Kaneko, O. Hori, "Joint Haar-like Features for Face Detection," *In Proceedings of the IEEE Conference on Computer Vision*, pp. 1619-1626, vol. 2, 2005.

[11] R. Lienhart and J. Maydt, "An Extended Set of Haar-like Features for Rapid Object Detection," *In Proceedings of the International Conference on Image Processing*, pp. I-900-I-903, vol.1, 2002.

[12] T. Theocharides, N. Vijaykrishnam, and M. J. Irwin, "A Parallel Architecture for Hardware Face Detection," *In Proceedings of the IEEE Computer Society Annual Symposium Emerging VLSI Technologies and Architectures*, pp. 452-453, 2006.

[13] R. McCready "Real-Time Face Detection on a Configurable Hardware System," *In Proceedings of the Roadmap to Reconfigurable Computing, International Workshop on Field-Programmable Logic and Applications*, pp.157-162, 2000.

[14] M. S. Sadri, N. Shams, M. Rahmaty, I. Hosseini, R. Changiz, S. Mortazavian, S. Kheradmand, and R. Jafari, "An FPGA Based Fast Face Detector," *In Global Signal Processing Expo and Conference*, 2004.

[15] Y. Wei, X. Bing, and C. Chareonsak, "FPGA Implementation of AdaBoost Algorithm for Detection of Face Biometrics," *In Proceedings of the IEEE Workshop Biomedical Circuits and Systems*, page S1, 2004.

[16] M. Yang, Y. Wu, J. Crenshaw, B. Augustine, and R. Mareachen, "Face Detection for Automatic Exposure Control in Handheld Camera," *In Proceedings of the IEEE Conference on Computer Vision System*, pp.17, 206.

[17] V. Nair, P. Laprise, and J. Clark, "An FPGA-Based People Detection System," *EURASIP Journal of Applied Signal Processing*, 2005(7), pp. 1047-1061, 2005.

[18] C. Gao and S. Lu, "Novel FPGA Based Haar Classifier Face Detection Algorithm Acceleration," *In Proceedings of the International Conference on Field Programmable Logic and Applications*, 2008.

[19] M. Hiromoto, K. Nakahara, H. Sugano, "A Specialized Processor Suitable for AdaBoost-Based Detection with Haar-like Features," *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp.1-8, 2007.

[20] H. Lai, M. Savvides, T. Chen, "Proposed FPGA Hardware Architecture for High Frame Rate (>100 fps) Face Detection Using Feature Cascade Classifiers," *In Proceedings of the IEEE Conference Biometrics: Theory, Applications, and System*, pp. 1-6, 2007.

[21] Xilinx Inc., "Virtex-5 Family Overview," February 2009. DOI:http://www.xilinx.com/.