A Novel SoC Architecture on FPGA for Ultra Fast Face Detection

Chun He[1], Alexandros Papakonstantinou[2], and Deming Chen[2]
[1]*Research Institute of Electronic Science & Tech., Univ. of Electronic Science & Technology of China*
[2]*Electrical & Computer Eng. Dept., Univ. of Illinois, Urbana-Champaign, IL, USA*
[1]*cerelia@uestc.edu.cn,* [2]*{apapako2, dchen}@illinois.edu*

***Abstract*— Face detection is the cornerstone of a wide range of applications such as video surveillance, robotic vision and biometric authentication. One of the biggest challenges in face detection based applications is the speed at which faces can be accurately detected. In this paper, we present a novel SoC (System on Chip) architecture for ultra fast face detection in video or other image rich content. Our implementation is based on an efficient and robust algorithm that uses a cascade of Artificial Neural Network (ANN) classifiers on AdaBoost trained Haar features. The face detector architecture extracts the coarse grained parallelism by efficiently overlapping different computation phases while taking advantage of the fine-grained parallelism at the module level. We provide details on the parallelism extraction achieved by our architecture and show experimental results that portray the efficiency of our face detection implementation. For the implementation and evaluation of our architecture we used the Xilinx FX130T Virtex5 FPGA device on the ML510 development board. Our performance evaluations indicate that a speedup of around 100X can be achieved over a SSE-optimized software implementation running on a 2.4GHz Core-2 Quad CPU. The detection speed reaches 625 frames per sec (fps).**

I. INTRODUCTION

Face detection is a challenging and active field of research which has enabled significant progress in the computing vision field during the last decade. A wide range of emerging applications, such as robotic vision, image and video indexing, video surveillance, biometric authentication, and race, gender or expression recognition [1] employ face detection techniques. The goal of face detection is the localization of an unknown number of faces in the input image data. Many different face detection algorithms have been proposed as discussed by Yang [3] and Hjelmas [1]. Through the use of statistics, neural networks and advanced feature extraction methods face detection techniques have been continually improving in localizing faces within cluttered scenes and extracting facial features accurately. Hjelmas [1] distinguishes the detection methods in two main categories: *feature-based* and *image-based*. Both classes of algorithms require a-priori face knowledge. The feature based techniques first identify facial features and then use face-knowledge in terms of distances, angles and area measurement to identify faces. On the other hand, the image-based techniques take advantage of the latest advances in pattern recognition theory by employing mapping and training schemes to implicitly incorporate face knowledge in the detection process. Image based techniques usually perform better in images with cluttered backgrounds.

In this work we use a robust image-based face detection algorithm [4] which employs a set of critical *Haar-type features* that consist of white and black rectangles (fig. 1a) to identify face regions on an image. Each Haar feature is a weak classifier that belongs to an enormously big set of rectangle-like features. A strong classifier is formed by selecting a small set of Haar features through AdaBoost training [4]. Each Haar feature is assigned a value S_F, which is equal to the difference of the sum of pixels in white rectangles and the sum of pixels in black rectangles: $S_F = S_W - S_B$ (fig 1a). By combining multiple Haar features, robust classification of face regions is accomplished. Calculation of Haar feature values can be computed efficiently by using the *integral image* representation [4]. The value of a pixel with (x_p, y_p) coordinates in the integral image is the sum of all values of the pixels with coordinates (x, y), such that $x \leq x_p$ and $y \leq y_p$ (shaded pixels left and above (x_p, y_p) pixel, inclusive, in fig 1b). Having generated the integral image representation, calculation of Haar feature values can be done in constant time regardless of their size. For example, the rectangle S in fig. 1b can be simply computed as S = A-B-C+D with 3 addition/subtraction operations.

Apart from robustness, efficiency is also extremely important for a number of applications that use face detection on huge image content inputs. Most of the software-based implementations fail to achieve real-time detection even on high-end PCs and laptops. For applications that use face detection as the basis for further processing, performance constraints are even harder to meet. These challenges motivate research activities for hardware acceleration. Especially FPGA devices offer the flexibility of reconfiguration and the potential for considerable parallelism extraction for an efficient and low power implementation. Lai [7] presented an architecture for AdaBoost-based detection that can theoretically detect 143 640x480 frames per second (fps). However, the reported speed is based on FPGA synthesis results without place & route information. Moreover, the number of Haar-type features used in their architecture is dependent on the available FPGA resources.

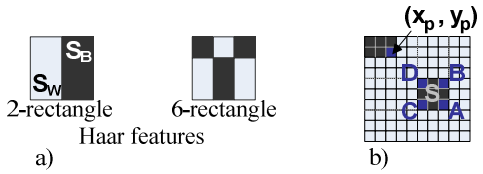

Fig. 1. a) Sample Haar features. b) Integral image.

978-1-4244-5028-2/09/$25.00 ©2009 IEEE

Cho [8] also proposes an AdaBoost-based hardware face detector with 3 parallel classifiers. They use a Virtex-5 Xilinx FPGA and show that they can obtain almost 19X speedup over the corresponding software version for 640x480 frames. However, the average detection rate is only 7fps, which falls short from real-time processing. A parallel architecture is described by Theocharides [9] that achieves a 52fps face detection rate based on synthesis results. However, no real implementation data and frame size information are reported. Hiromoto [10] presented a hybrid architecture that employs a parallel processing scheme in the front-end stages and a sequential processing scheme in the back-end stages of the classifier cascade. They reported a detection speed of 30fps.

In this work we present a novel FPGA-based SoC implementation of an efficient and robust face detection algorithm [4] that uses a cascaded Artificial Neural Network (ANN) classification scheme based on AdaBoost-trained Haar features [5] [6]. The ANN-based classification occurs in stages with progressively stronger classifiers in each subsequent stage. In case of negative classification during one of the stages, the classification process is interrupted and a negative result is returned (i.e. non-face classification). The implemented algorithm is also reinforced by motion detection and skin detection features to further avoid spending compute time in non-promising image areas. The FPGA-based face detector architecture extracts the coarse grained parallelism inherent in the implemented algorithm by efficiently overlapping different computation tasks while taking advantage of the available fine-grained parallelism within each task. The details of the parallelism extraction are discussed in section IV. Section V presents experimental results that portray the efficiency of our SoC implementation compared to the software implementation. For the implementation and evaluation of our architecture we used the Xilinx FX130T Virtex5 FPGA device. Our performance evaluations indicate that a speedup of up to 100X can be achieved over an SSE-optimized software-based implementation that runs on a 2.4GHz Core-2 processor. The detection speed reaches 625 640x480 fps.

II. Face Detection Algorithm

The algorithm we have implemented is based on the robust face detection framework proposed by Viola and Jones [4]. We use the integral image representation for fast computation of a small set of Haar features that have been selected through AdaBoost learning. The classification is done by neural networks organized in a cascade of nine stages. This algorithm employs motion and skin detection for more efficient face detection and is based on the software implementation proposed by Yuriy [2].

Our hardware implementation takes as input VGA-resolution (640x480) 24-bit color images which go through three processing stages: i) preprocessing stage, ii) detection stage and iii) post-processing stage. During the preprocessing stage, the image is downscaled to 80x60 frame size, and motion detection and skin detection techniques are applied.

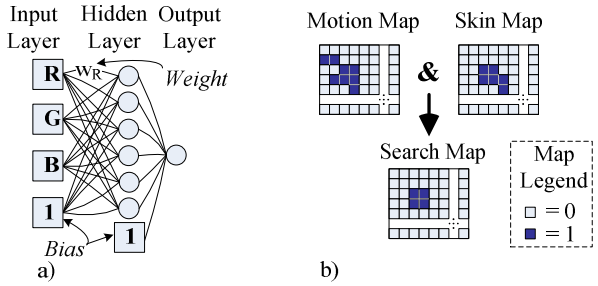

Fig. 2. a) Skin detector ANN. b) Search map generation

Subsequently, integral image generation and Haar feature classification for 3 different object sizes are performed in the main detection stage. Finally, in the post-processing stage the classification results are used to eliminate artifacts and noise introduced by window scanning overlap and downscaling, respectively.

A. Pre-Processing Stage

The preprocessing stage manipulates the input image so as to reduce the work load of the main detection stage. Initially the frame is downscaled by a factor of eight through 2D Fast Haar transform. Each 2D Haar transform consists of two 1D Haar transforms that are applied sequentially on the rows and the columns of the 2D image pixel array. The outcome of each 2D Haar transform is a compressed image of half the original dimensions. Three 2D Haar transforms are applied back-to-back in order to compress the 640x480 frames down to 80x60 frames. All subsequent processing is performed on the downsized frames.

An 8-bit single-intensity image (grayscale image with 8 bits per pixel) is generated from the 24-bit color 80x60 frame. The grayscale frame is used in motion detection while the color frame is used in skin detection. Motion detection and skin detection are employed during the preprocessing stage to further filter the potential image regions that may include faces. Motion detection identifies candidate face regions by subtracting two consecutive grayscale image frames and comparing the difference values with a threshold value. The output of motion detection is an 80x60 binary matrix, called *motion map*, where elements that correspond to difference values above the threshold are set. Motion map is further updated by setting the bits that correspond to pixels that were identified as face areas in the previous frame (to consider still faces). Skin detection is based on a 3-layer artificial neural network with an *input layer* of three neurons for the R, G and B color information, respectively (fig. 2a). Six neurons process the color data in the *hidden layer* and a single neuron *output layer* generates the skin detector result in the form of an 80x60 binary matrix, called *skin map*. *Search map* is the binary matrix produced by the intersection of motion map and skin map (fig. 2b). Image regions that may contain faces are flagged by the set ('1') elements (fig. 2b).

The search map is further processed by applying *dilation* followed by *erosion* using a 5x5 square window (structuring element). This step aims to reduce noise and scratch-like artifacts in the search map. Dilation connects areas that are

separated by spaces smaller than the structuring element, while erosion removes object smaller than the structuring element.

B. Main Detection Stage

The first task of the main detection stage is to generate the integral image in order to facilitate fast Haar feature computation. The 80x60 grayscale image is leveraged to create the integral image. Then, three object sizes are used for window scanning of the integral image: 11x11, 19x19 and 27x27 (which correspond to 88x88, 152x152 and 216x216 object sizes in the original 640x480 frame). When the scanning window central pixel corresponds to a set bit in the search map, the classification process is triggered to determine if a face is present in the image region covered by the window.

Classification is done by a 9-stage cascade of artificial neural networks. Each ANN stage takes as input a set of Haar features that have been computed using the integral image and is only activated if the previous ANN stage has produced a positive output (except from the first stage which is activated by the search map flag of the scanning window central bit). Fig. 3 demonstrates the structure of the 9-stage cascade. A total of 115 Haar features (each one with up to 20 rectangles) are used as inputs for the ANNs in the 9-stage classifier. The earlier stages of the cascade use simpler ANN classifiers which take fewer Haar features as input. Moreover, the cascade is designed so as to ensure that the ANNs in the early stages have high negative predictivity (true neg. / [true neg. + false neg.]) in order to accurately reject the vast majority of non-face objects, while positive predictivity (true positive / [true positive + false positive]) increases in the latter stages for accurate positive predictions of actual faces. This way, the classifier will spend less time processing Haar features in image regions without face objects. On the other hand, for a face to be detected all ANN stages should be activated through a ripple of positive classifications throughout the cascade. A positive classification out of the 9th stage of the cascade is stored in a 2D matrix, called *object map*. This matrix is used in the post-processing stage for determining all the face locations in the image.

C. Post-Processing Stage

The post-processing stage leverages the three *object maps* generated during the detection stage by the three object size classifiers to remove the detection noise and the overlapping artifacts (i.e. multiple overlapping detections of the same face). This is achieved by scanning the object maps with a 5x5 window and identifying the face locations in descending order of the sum of the elements within the 5x5 square regions. When a face region is identified in an object map, a check for overlap with previously found face regions is done and if no overlap is found the corresponding object window location is added to the face regions list and the object map bits covered by the object window are cleared. The identification of new face locations continues until there are no 5x5 square regions in any of the object maps that have an

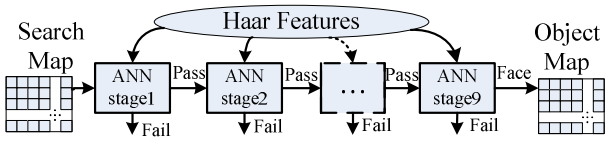

Fig. 3. 9-stage cascade ANN classifier.

element sum greater than a predefined threshold value. Face detection on the current frame completes when this occurs.

III. System Details & Parallelism Extraction

For our implementation we have used a top-tier Xilinx Virtex-5 FPGA (XC5VFX130T) which integrates a wide range of resources that render it a suitable SoC platform. The FPGA device is part of the ML510 development board which includes 1GB of DDR2, VGA and several other interfaces and modules. The DDR2 memory stores the input video or still image data that is sourced to the SoC face detector. Data may be live video sourced through one of the available board interfaces (e.g. USB) or archived content transferred from the host PC. In our experimental results we evaluate the performance of the face detection system assuming data is always available in the DDR2 memory of the board. The block diagram of our system implementation is depicted in fig. 4, while the architecture implementation and the parallelism extraction techniques are discussed in the following subsections.

A. Coarse-Grained Parallelism Extraction

Face detection includes several compute intensive tasks. Thus, these processing tasks are implemented on the reconfigurable fabric of the FPGA device in order to extract

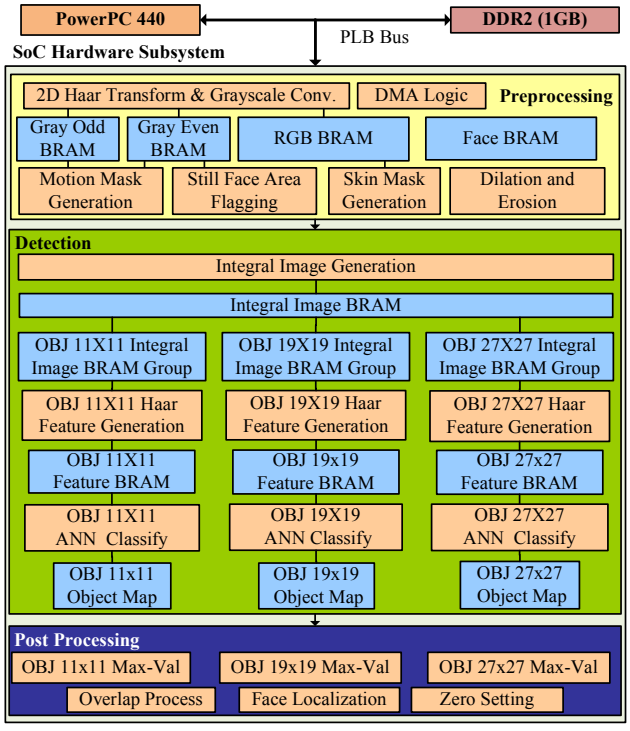

Fig. 4. FPGA-based implementation block diagram.

parallelism both at the task level and the operation level. PowerPC is used for control and configuration tasks, such as the configuration of the DMA bursts from DDR2 and various face detection parameters (e.g. ANN classifier weights and normalization parameters). Moreover, applications that require face detection information (e.g image indexing) can also run on the embedded PowerPC CPU.

The face detection process is triggered by PowerPC through the PLB bus. DMA logic fetches the input data from DDR2 and streams them through the "2D Haar transform & grayscale conversion" module for compression and RGB-to-grayscale conversion. Two 80x60 frames (an RGB and a grayscale) are stored into two BRAMs of size 4800x24bit and 4800x8bit, respectively (RGB and Gray BRAMs in fig. 4). 2D Haar transform is a highly compute intensive task, thus it is parallelized with the rest of the frame processing in a pipelined fashion. 2D Haar transform operates on frame *N+1* while the rest of the hardware subsystem operates on frame *N* (fig. 5).

With regard to the Gray BRAM memory, a double-buffering scheme (Gray Even and Gray Odd BRAMs) is used to enable extra parallelism extraction. The 2D Haar transform module stores the grayscale frames in the two gray BRAMS in alternating fashion. Thus execution of next frame's motion detection task can be parallelized with the current frame's ANN classification and post-processing (fig. 5).

A major latency-reduction optimization in our FPGA implementation is the execution of the classification tasks for the three object sizes in parallel, rather than in sequence, as is done in the software implementation. This has a direct impact in the FPGA resource utilization, as three times the resources are needed. Nonetheless, through careful design and CAD tool optimizations (see section IV.C) resource utilization issues were resolved.

A significant advantage of the FPGA implementation is that parallelism can be extracted in a highly flexible fashion. In particular, as depicted in fig. 5, the classification process is partially parallelized with the generation of the integral image and the search map, on which it is data dependent. This is possible through careful synchronization of the different tasks to allow for hazard-free concurrency. For example, in the case of the 11x11 object detection, Haar feature computation begins as soon as the first 6 rows of the search map and the first 11 rows of the integral image have been fully computed. In the case of bigger object sizes, launch of feature computation is slightly further delayed in order to allow for the extra required rows of the search map and the integral image to be computed.

In a similar way, hazard-free synchronization enables partial parallelization of ANN classification with Haar feature computation (fig. 5). The original software implementation cascades the Haar feature computation along with the ANN classification. The FPGA implementation, on the other hand, computes the Haar features in a non-cascaded fashion until either all features are computed or the classification process outputs a negative resolution. Thus, with the exception of the first stage, the following ANN classification stages are not delayed by the Haar feature computation.

The Max Value unit in the post-processing stage (fig. 4) performs the compute intensive task of sorting the sums of the 5x5 squares in the generated object map through iterative scanning. Hazard-free synchronization in each object dataflow enables concurrent sorting in descending order of the promising face regions and identification of the most promising face region in parallel with the main detection stage (fig. 5).

B. Fine-Grained Parallelism Extraction

Apart from the coarse-grained parallelism at the task level, the hardware part of the face detection SoC has been carefully designed to extract parallelism at the operation level while considering the impact in resource allocation. In this section we discuss extra design details on the Haar feature generator and the ANN classifier.

Even though the integral image plays an important role in reducing the computations required for calculating Haar rectangles, the generation of Haar features is one of the most compute intensive tasks of the detection process. In order to speed up this computation, our Haar feature generator design (fig. 6) employs three related features: i) Integral Image BRAM Group (IIBG), ii) 8-way rectangle calculator and iii) tree-based feature composition. The IIBG is used to split the integral image in multiple BRAMs so as to enable a multi-banked memory organization for multiple references per cycle. This way the dual port limitation of BRAMs is overcome in the design of the Haar feature generator. Each bank in the IIBG feeds a rectangle accumulator which is responsible for calculating a specific rectangle of the

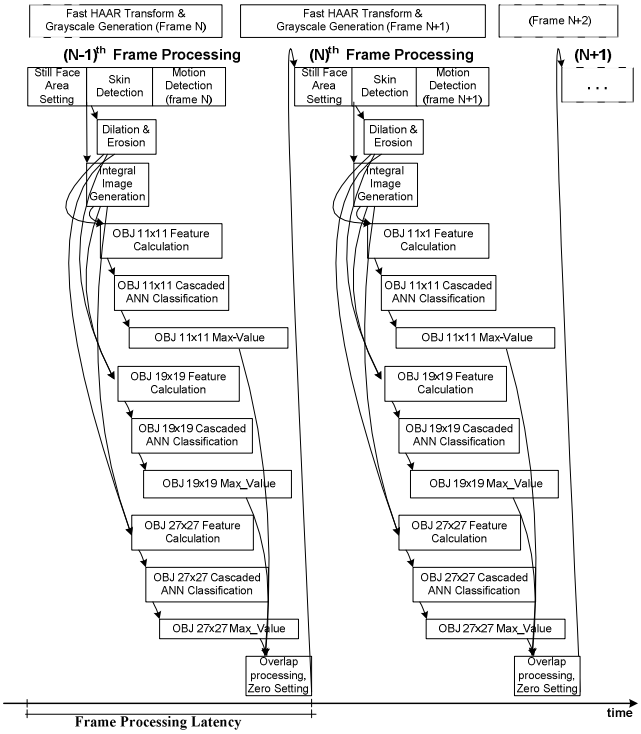

Fig. 5. Task-level parallelism and inter-task data dependencies.

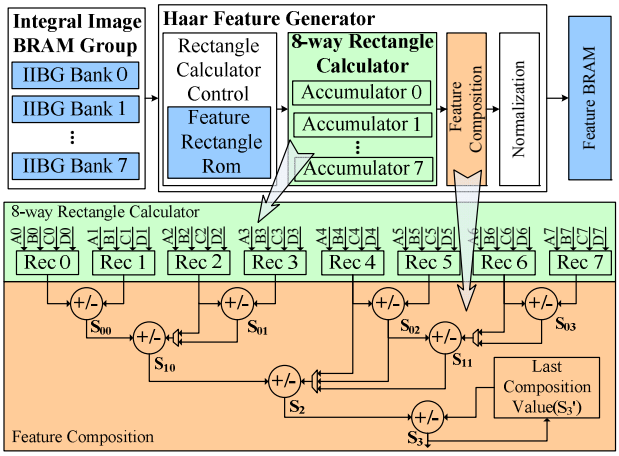

Fig. 6. Haar feature generator

computed feature. Eight rectangles can be concurrently computed for each feature in the 8-way rectangle calculator. The rectangles are fed to the tree-based feature composer which uses a tree-like organization (fig. 6) to compose rectangles into Haar features. The feature composition module consists of eight add/subtract functional units which are connected in a 4-level tree hierarchy. Depending on the number of rectangles, *N*, included in the Haar feature, the composition may be terminated at level log_2N of the tree structure, if *N* is not greater than 8. For features with more than eight rectangles, the tree output is stored and used at the root level (level 4) of the following invocation of the feature composition tree. Multiple invocations may be required until all rectangles have been processed for features with many rectangles.

After feature composition, feature normalization is performed before storing the features in the corresponding BRAM for use by the ANN classifier (fig. 6). The Haar feature generator extracts parallelism with a pipelined architecture, in which *rectangle calculation*, *feature composition* and *feature normalization* comprise the pipeline stages. Each pipeline stage operates on a set of rectangles and passes the results to the next stage for further processing, while it starts working on the next set of rectangles.

Feature classification is based on a versatile module with a novel architecture that combines 20 MAC units in an elaborate feedback-based 2-stage pipeline. This module is used for the implementation of all the classifiers in the 9-stage cascade. The pipeline involves a MAC computation stage and a sigmoid normalization stage (fig. 7). The MAC units correspond to the ANN neurons which work in parallel to implement one layer of a neural network, each cycle. Through feedback paths the MAC units can be fed with data that correspond to the outputs of the previous ANN layer. Each MAC takes as input one weight value which is looked up in a weight ROM and a second value which depends on the implemented stage and layer of the cascaded ANN classifier. This value might be a computed feature referenced from the Feature BRAM, a feedback value computed by the previous layer of the ANN or the constant value '1' when

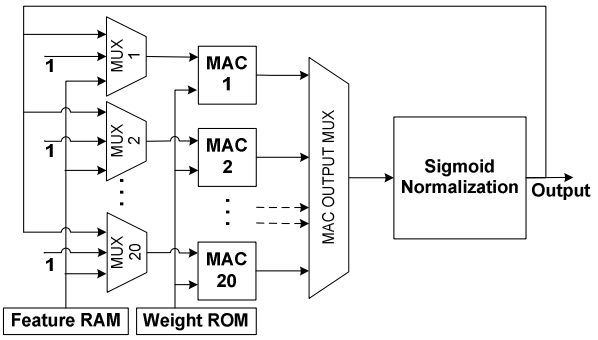

Fig. 7. ANN implementation in the classifier module.

considering the classifier threshold (aka *bias*). The control logic determines how many MAC units will be used in each layer and how many MAC outputs will be sent through the feedback path as inputs to the next layer. For example, for a 3-20-1 ANN that has a single hidden layer with 20 neurons and an output layer with 1 neuron, all the MAC units are utilized during the hidden layer computation, while only one of the MAC units is utilized during the output layer computation.

Sigmoid normalization is used to scale the MAC outputs into a suitable range of values for use in the next ANN layer. It is implemented through table look-up for higher efficiency compared to a CORDIC IP implementation [11]. As illustrated in fig. 7, normalization of the MAC outputs is done in a sequential fashion by a single table reference per cycle. Nevertheless, parallelism is not hurt as the normalization step is overlapped with the next layer's MAC step. This is portrayed in fig. 8, which depicts the dataflow characteristics of a 3-20-1 ANN classifier. Initially features *F1*, *F2* and *F3* are sequentially provided to all the MAC units. Values *HR1* to *HR20* are concurrently produced by the hidden layer MACs, and the normalized *HN1* to *HN20* values are sequentially generated by the sigmoid module. While normalization values are produced, MAC1 starts computing the output layer result. *OR1* is the output layer MAC result and *ON1* is the normalized output layer neuron result.

C. FPGA Implementation Details

The face detection SoC was implemented with the help of Xilinx ISE and EDK development environments. In particular, ISE was used for the development of the hardware subsystem in Verilog, whereas EDK was used for the development of the software modules and the integration of the two subsystems in a hardware-software co-designed SoC. The communication of the software and hardware subsystems was facilitated with the PLB bus. A PLB interface for the face detection module was designed in VHDL and PLB peripheral driver functions were used in the PowerPC software to facilitate communication with the face detection module.

As described in the previous sections, the implemented algorithm involves a long sequence of different compute intensive tasks. For its implementation we made wide use of two embedded resource elements: BRAM memories and DSP

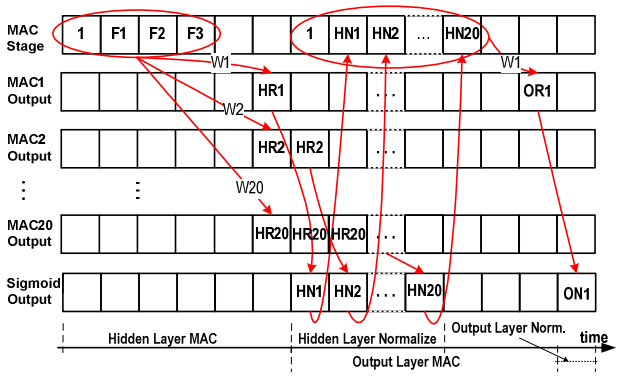

Fig. 8. Dataflow for a 3-20-1 ANN in the implemented classifier module.

slices. BRAM memories are very convenient for storing vector and 2D array data which are widely used in face detection. Moreover, the dual port characteristic of BRAMs makes them ideal for building buffers in streaming-like applications, where the producer uses one port to store its outputs and the consumer uses the second port to load its inputs. This type of interface between different modules is used extensively in our implementation (fig. 4).

Two commonly used operations across many tasks in the face detection algorithm are summation and accumulation of multiplied values. Xilinx FPGAs contain a large number of DSP slices which include a 25x18bit multiplier, an adder and an accumulator [12]. Implementing the integral image and Haar feature generators as well as the ANN classifier with DSP slices was crucial to performance. Both DSP slices and BRAM memories are explicitly instantiated in our hardware subsystem in order to control the resource usage. Careful allocation of resources was important for successful mapping of the design on the FPGA. Moreover, significant time was spent in the exploration of the optimal mapping and routing options in order to achieve a fully routed implementation with a reasonable clock period. The resource utilization summary for the FX130T device is presented in table 1. Note that the BRAM utilization is relatively high, as we allow part of logic to be mapped onto spare BRAMs in order to facilitate better routing. The frequency achieved after synthesis was 127MHz, whereas after place and route the SoC design was able to run at 73MHz.

IV. Experimental Results

The implemented face detection algorithm is based on Viola and Jone's robust and efficient face detection framework [4] and is extended with motion and skin detection that help boost software detection rate to 49 fps on a 2.2 GHz single core processor [2]. However, the detection rate can be heavily impacted by input that contains images with multiple faces or video with motion-intensive scenes. In this section we compare the performance of the SoC implementation with the software version at the granularity of i) task, ii) frame and iii) frame sequence. Both implementations execute the same algorithm in terms of functionality, even though they differ in terms of task schedule organization. The software version is written in MS Visual C++ with SSE-optimized kernels. Its performance is measured on Intel Core2 Quad CPU running at 2.4GHz. The SoC version is implemented on the FX130T Virtex5 FPGA with the software subsystem running on the embedded PowerPC at 400MHz. Both chips are fabricated with 65nm technology.

Table 1. XC5VFX130T Resource Utilization Summary

Resource (Avail)	Used Resources	Utilization
Registers (81920)	37828	46%
LUTs (81920)	67704	83%
BlockRAM (298)	276	93%
DSP48Es (320)	161	50%

Table 2 compares the execution latency of different tasks between the software and the SoC implementations. The execution time of the classification task depends on the number of stages that generate a positive result. Thus we have provided comparison times for all possible classification scenarios. As shown in the 4th column of table 2, significant task level speedups are achieved that range up to 103X. Note that these speedups do not include the performance benefits from the extracted coarse-grained parallelism across tasks.

Table 3 compares the execution latency for a single frame with different number of portrayed faces. For both implementations the execution latency displays a rising trend as the number of faces increases with the exception of the software latencies for five and six-face images. This exception is probably due to the variation in the number of ANN stages outputting false positive results during classification of non-face regions. Column 4 lists the speedups achieved by the SoC with an average of 88X. These speedups include the coarse-grained parallelism extracted across tasks, but do not consider the parallelism extracted across frames. Figure 9a shows the output of the face detection for a six-face color image.

To evaluate the performance of the SoC implementation for video input, we measured the detection latency for a sequence of 60 frames. The results are listed in table 4. As shown in column 3, the speedup is significantly increased compared to single frames. This is mainly due to the image downscaling and the motion detection handling. In our SoC implementation image downscaling and RGB-to-grayscale conversion is parallelized with the rest of the face detection processing in a pipelined fashion. Moreover, motion detection does not impact latency as it is performed during processing of the previous frame by utilizing the double Gray

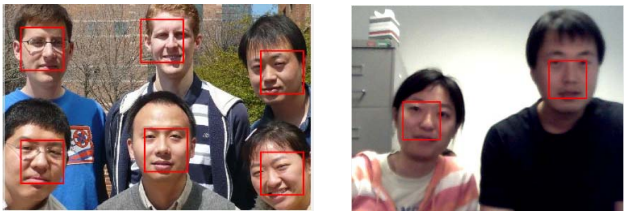

Fig. 9. a) Sample Processed Image b) Sample Processed Video Frame.

BRAM buffering (fig. 4). Fig. 9b displays a video frame after face detection. The FPGA SoC performance was also evaluated against a Pentium4 processor running the software version on a 3GHz clock, resulting to a 145X speedup.

We also evaluated the power dissipated by our SoC implementation, by importing all the physical implementation data in the Xilinx Power Estimator [13]. The total power dissipated by both the reconfigurable fabric and the PowerPC processor was estimated at 5.95W. Comparing this value with the reported power dissipations of the Intel Core2 Quad (95W) and the Pentium4 (84W) processors [14] we can calculate the power advantage of the FPGA implementation at 16X and 14.1X, respectively.

In summary, our FPGA implementation offers an impressive speedup over the software implementation at a much smaller power footprint. The main source of speedup comes from task level parallelism (fig. 5) which is accomplished by the data streaming architecture in tandem with the hazard-free task synchronization (i.e. each task can start executing as soon as enough data is generated from the previous task). This is complemented by operation-level parallelism within each task (such as the 8-way rectangle calculation and the 20-MAC ANN implementation) and the lack of data handling overhead that is inherent in general-purpose processors (e.g. data shifting, splitting and merging).

Table 2. Task execution times and speedups

TASK	SOFTWARE	SOC	SPEEDUP
Integral image gen.	1398 us	20.5 us	68.2
Skin detection (1 pixel)	30 us	0.29 us	103.4
1-stage classify	24 us	0.52 us	46.2
2-stage classify	56 us	1.14 us	49.1
3-stage classify	93 us	1.70 us	54.7
4-stage classify	137 us	2.34 us	58.5
5-stage classify	203 us	3.22 us	63.0
6-stage classify	325 us	4.38 us	74.2
7-stage classify	442 us	5.71 us	77.1
8-stage classify	567 us	8.63 us	65.7
9-stage classify	838 us	11.48 us	73.0

Table 3. Single Frame Detection Latency Comparison

FACE #	SW EXEC. TIME	SOC EXEC TIME	SPEEDUP
0	95 ms	1.397 ms	68.0
1	182 ms	2.058ms	88.4
2	226 ms	2.704 ms	83.6
3	322 ms	3.896 ms	82.6
4	398 ms	5.377 ms	74.0
5	487 ms	6.899 ms	70.6
6	415 ms	6.993 ms	59.3

Table 4. 60-Frame Detection Latency Comparison

SW LATENCY (FPS)	SOC LATENCY (FPS)	SPEEDUP
9755ms (6.2)	95.9ms (625.7)	101.7

V. CONCLUSIONS

We have presented a novel SoC architecture on FPGA for face detection which can detect faces at speeds of roughly two orders of magnitude (100X) higher than the corresponding software implementation running on a 2.4GHz CPU. This performance efficiency is achieved through coarse-grained parallelism extraction across tasks, as well as fine grained parallelism across operations. Each image frame is processed in a streaming-like fashion using BRAM buffers for interfacing sequential processing stages, while inter-frame parallelism is exploited with regards to image downscaling and motion detection. Note that the algorithm we used in this work is very robust and highly efficient (based on the AdaBoost framework [4]) but may display poor detection accuracy for faces of smaller sizes than the object window sizes we used. Ideally more object window sizes could be used. This would potentially result in further speedup of the SoC implementation compared to the software one, since detection of different object sizes are performed in parallel in the SoC architecture, while the software version evaluates different object sizes in sequence.

ACKNOWLEDGEMENT

This work is partially supported by the Chinese Scholarship Council and NSF grant CCF 07-46608.

REFERENCES

[1] E. Hjelmas, and B. K. Low, “Face detection: a survey,” *Computer Vision and Image Understanding*, Vol. 83, No 3, 2001.
[2] C. Yuriy, “Ultra rapid object detection in computer vision applications with haar wavelet features,” Oct 2008.
[3] M.H. Yang, D. J. Kriegman, and N. Ahuja, “Detecting faces in images: a survey,” *PAMI*, Vol. 24, Issue 1, pp. 34-58, 2002.
[4] P. Viola and M. Jones, “Robust real-time face detection,” *International Journal of Computer Vision*, 57(2), 137-154, 2004.
[5] H. A. Rowley, S. Baluja, and T. Kanade, “Neural-network based face detection,” *Pattern Analysis and Machine Intelligence*, Vol 20, 1998.
[6] C. C. Han et al., “Fast face detection via morphology-based pre-processing,” *Pattern Recognition*, Vol 33, 2000.
[7] Hung-Chih Lai, Marios Savvides, Tsuhan Chen, "Proposed FPGA hardware architecture for high frame rate (>100 fps) face detection using feature cascade classifiers," *Biometrics: Theory, Applications, and Systems*, pp.1-6, 2007.
[8] Junguk Cho, Ryan Kastner, Jason Oberg, and Ryan Kastner, “FPGA-based face fetection system using Haar classifiers,” *International symposium on Field programmable gate arrays*, 2009.
[9] T. Theocharides, N. Vijaykrishnam, and M. J. Irwin, “A parallel architecture for hardware face detection,” *Symp. on Emerging VLSI Technologies and Architectures*, pp. 452-453, 2006.
[10] M. Hiromoto, K. Nakahara, H. Sugano, “A specialized processor suitable for AdaBoost-based detection with Haar-like features,” *Computer Vision and Pattern Recognition*, pp.1-8, 2007.
[11] Y. Lee, and S. B. Ko, "FPGA implementation of a face detector using neural networks," *Canadian conf. electrical and computer eng.*, 2006.
[12] Xilinx, “Virtex-5 FPGA overview,” http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf, 2009.
[13] Xilinx, “Power Solutions,” http://www.xilinx.com/products/design_resources/power_central/index.htm, 2009.
[14] Intel, “Processor Spec Finder,” http://processorfinder.intel.com