Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2009, Article ID 235032, 12 pages
doi:10.1155/2009/235032

Research Article

Cascade Boosting-Based Object Detection from High-Level Description to Hardware Implementation

K. Khattab, J. Dubois, and J. Miteran

Le2i UMR CNRS 5158, Aile des Sciences de l'Ingénieur, Université de Bourgogne, BP 47870, 21078 Dijon Cedex, France

Correspondence should be addressed to J. Dubois, jdubois@u-bourgogne.fr

Received 28 February 2009; Accepted 30 June 2009

Recommended by Bertrand Granado

Object detection forms the first step of a larger setup for a wide variety of computer vision applications. The focus of this paper is the implementation of a real-time embedded object detection system while relying on high-level description language such as SystemC. Boosting-based object detection algorithms are considered as the fastest accurate object detection algorithms today. However, the implementation of a real time solution for such algorithms is still a challenge. A new parallel implementation, which exploits the parallelism and the pipelining in these algorithms, is proposed. We show that using a SystemC description model paired with a mainstream automatic synthesis tool can lead to an efficient embedded implementation. We also display some of the tradeoffs and considerations, for this implementation to be effective. This implementation proves capable of achieving 42 fps for 320×240 images as well as bringing regularity in time consuming.

1. Introduction

Object detection is the task of locating an object in an image despite considerable variations in lighting, background, and object appearance. The ability of object detecting in a scene is critical in our everyday life activities, and lately it has gathered an increasingly amount of attention.

Motivated by a very active area of vision research, most of object detection methods focus on detecting Frontal Faces (Figure 1). Face detection is considered as an important subtask in many computer vision application areas such as security, surveillance, and content-based image retrieval. Boosting-based method has led to the state-of-the-art detection systems. It was first introduced by Viola and Jones as a successful application of Adaboost [1] for face detection. Then Li et al. extended this work for multiview faces, using improved variant boosting algorithms [2, 3]. However, these methods are used to detect a plethora of objects, such as vehicles, bikes, and pedestrians. Overall these methods proved to be time accurate and efficient.

Moreover this family of detectors relies upon several classifiers trained by a boosting algorithm [4–8]. These algorithms help achieving a linear combination of weak classifiers (often a single threshold), capable of real-time face detection with high detection rates. Such a technique can be divided into two phases: training and detection (through the cascade). While the training phase can be done offline and might take several days of processing, the final cascade detector should enable real-time processing. The goal is to run through a given image in order to find all the faces regardless of their scales and locations. Therefore, the image can be seen as a set of subwindows that have to be evaluated by the detector which selects those containing faces.

Most of the solutions deployed today are general purpose processors software. Furthermore, with the development of faster camera sensors which allows higher image resolution at higher frame-rates, these software solutions are not always working in real time. Accelerating the boosting detection can be considered as a key issue in pattern recognition, as much as motion estimation is considered for MPEG-4.

Seeking some improvement over the software, several attempts were made trying to implement object/face detection on multi-FPGA boards and multiprocessor platforms using programmable hardware [9–14], just to fell short in frame rate and/or high accuracy.

The first contribution of this paper is a new structure that exploits intrinsic parallelism of a boosting-based object detection algorithm.

As for a second contribution, this paper shows that a hardware implementation is possible using high-level SystemC description models. SystemC enables PC simulation that allows simple and fast testing and leaves our structure open to any kind of hardware or software implementation since SystemC is independent from all platforms. Mainstream Synthesis tools, such as SystemCrafter [15], are capable of generating automatic RTL VHDL out of SystemC models, though there is a list of restrictions and constraints. The simulation of the SystemC models has highlighted the critical parts of the structure. Multiple refinements were made to have a precise, compile-ready description. Therefore, multiple synthesis results are shown. Note that our fastest implementation was capable of achieving 42 frames per second for 320×240 images running at 123 MHz frequency.

The paper is structured as follows. In Section 2 the boosted-based object detectors are reviewed while focusing on accelerating the detection phase only. In Section 3 a sequential implementation of the detector is given while showing its real time estimation and drawbacks. A new parallel structure is proposed in Section 4; its benefits in masking the irregularity of the detector and in speeding the detection are also discussed. In Section 5 a SystemC modelling for the proposed architecture is shown using various abstraction levels. And finally, the firmware implementation details as well as the experimental results are presented in Section 6.

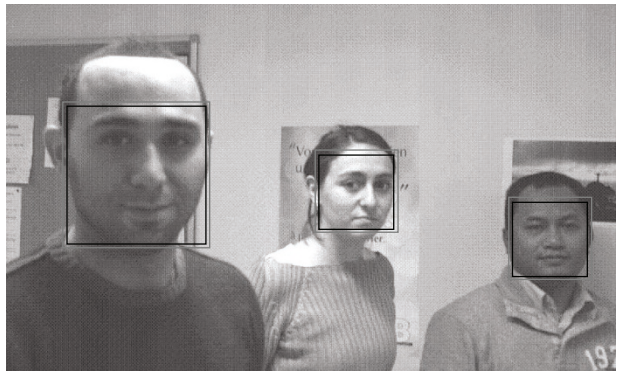

Figure 1: Example of face detection.

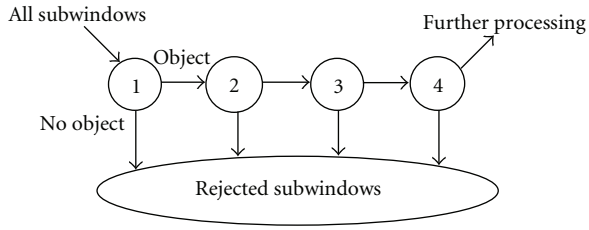

Figure 2: Cascade detector.

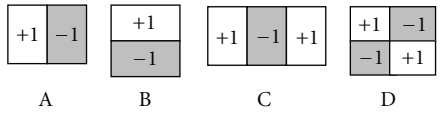

Figure 3: Rectangle features.

2. Review of Boosting-Based Object Detectors

Object detection is defined as the identification and the localization of all image regions that contain a specific object regardless of the object's position and size, in an uncontrolled background and lightning. It is more difficult than object localization where the number of objects and their size are already known. The object can be anything from a vehicle, human face (Figure 1), human hand, pedestrian, and so forth. The majority of the boosting-based object detectors work-to-date have primarily focused on developing novel face detection since it is very useful for a large array of applications. Moreover, this task is much trickier than other object detection tasks, due to the typical variations of hair style, facial hair, glasses, and other adornments. However, a lot of previous works have proved that the same family of detector can be used for different type of object, such as hand detection, pedestrian [4, 10], and vehicles. Most of these works achieved high detection accuracies; of course a learning phase was essential for each case.

2.1. Theory of Boosting-Based Object Detectors

2.1.1. Cascade Detection. The structure of the cascade detector (introduced in face detection by Viola and Jones [1]) is that of a degenerated decision tree. It is constituted of successively more complex stages of classifiers (Figure 2). The objective is to increase the speed of the detector by focusing on the promising zones of the image. The first stage of the cascade will look over for these promising zones and indicates which subwindows should be evaluated by the next stage. If a subwindow is labeled at the current classifier as nonface, then it will be rejected and the decision upon it is terminated. Otherwise it has to be evaluated by the next classifier. When a sub-window survives all the stages of the cascade, it will be labeled as a face. Therefore the complexity increases dramatically with each stage, but the number of sub-windows to be evaluated will decrease more tremendously. Over the cascade the overall detection rate should remain high while the false positive rate should decrease aggressively.

2.1.2. Features. To achieve a fast and robust implementation, Boosting based faces detection algorithms use some rectangle Haar-like features (shown in Figure 3) introduced by [16]: two-rectangle features (A and B), three-rectangle features (C), and four-rectangle features (D). They operate on grayscale images and their decisions depend on the threshold difference between the sum of the luminance of the white region(s) and the sum of the luminance of the gray region(s).

Using a particular representation of the image so-called the Integral Image (II), it is possible to compute very rapidly

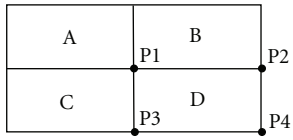

Figure 4: The sum of pixels within Rectangle D can be calculated by using 4 array references; S_D= II [P4] – (II [P3] + II [P2] – II [P1]).

the features. The II is constructed of the initial image by simply taking the sum of luminance value above and to the left of each pixel in the image:

$$ii(x, y) = \sum_{x'<x, y'<y} i(x', y') \tag{1}$$

where $ii(x, y)$ is the integral image, and $i(x, y)$ is the original image pixel's value. Using the Integral Image, any sum of luminance within a rectangle can be calculated from II using four array references (Figure 4). After the II computation, the evaluation of each feature requires 6, 8, or 9 array references depending on its type.

However, assuming a 24×24 pixels sub-window size, the over-complete feature set of all possible features computed in this window is 45 396 [1]: it is clear that a feature selection is necessary in order to keep real-time computation time compatibility. This is one of the roles of the Boosting training step.

2.1.3. Weak Classifiers and Boosting Training. A weak classifier $h_j(x)$ consists of a feature f_j, a threshold θ_j, and a parity p_j indicating the direction of the inequality sign:

$$h_j(x) = \begin{cases} 1, & \text{if } p_j f_j(x) < p_j \theta_j, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Boosting algorithms (Adaboost and variants) are able to construct a strong classifier as a linear combination of weak classifiers (here a single threshold) chosen from a given, finite or infinite, set, as shown in (3):

$$h(x) = \begin{cases} 1, & \sum_{t=1}^{T} \alpha_t h_t(x) > \theta, \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

where θ is the stage threshold, α_t is the weak classifier's weight, and T is the total number of weak classifiers (features).

This linear combination is trained in cascade in order to have better results.

There, a variant of Adaboost is used for learning object detection; it performs two important tasks: feature selection from the features defined above and constructing classifiers using selected features.

The result of the training step is a set of parameters (array references for features, constant coefficients of the linear combination of classifiers, and thresholds values selected by Adaboost). This set of features parameters can be stored easily in a small local memory.

2.2. Previous Implementations. The state-of-the-art initial prototype of this method, also known as Viola-Jones algorithm, was a software implementation based on trained classifiers using Adaboost. The first implementation shows some good potential by achieving good results in terms of speed and accuracy; the prototype can achieve 15 frames per second on a desktop computer for 320×240 images. Such an implementation on general purpose processors offers a great deal of flexibility, and it can be optimized with little time and cost, thanks for the wide variety of the well-established design tools for software development. However, such implementation can occupy all CPU computational power for this task alone; nevertheless, face/object detection is considered as prerequisite step for some of the main application such as biometric, content-base image retrieval systems, surveillance, and autonavigation. Therefore, there is more and more interest in exploring an implementation of accurate and efficient object detection on low-cost embedded technologies. The most common target technologies are embedded microprocessors such as DSPs, pure hardware systems such as ASIC, and configurable hardware such as FPGAs.

Lot of tradeoffs can be mentioned when trying to compare these technologies. For instance, the use of embedded processor can increase the level of parallelism of the application, but it costs high power consumption, all while limiting the solution to run under a dedicated processor.

Using ASIC can result better frequency performance coupled with high level of parallelism and low power consumption. Yet, in addition to the loss of flexibility, using this technology requires a large amount of development, optimization, and implementation time, which elevates the cost and risk of the implementation.

FPGAs can have a slightly better performance/cost tradeoffs than previous two, since it permits high level of parallelism coupled with some design flexibility. However some restriction in design space, costly rams connections as well as lower frequency comparing to ASIC, can rule-out its use for some memory heavy applications.

For our knowledge, few attempts were made trying to implement Boosting-based face detection on embedded platforms; even so, fewer attempted such an implementation for other object type, for example, whole body detection [10].

Nevertheless, these proposed architectures were configurable hardware-based implementations, and most of them could not achieve high detection frame rate speed while keeping the detection rate close of that of the original implementation. For instance, in order to achieve 15 frames per second for 120×120 images, Wei et al. [11] choose to skip the enlargement scale factor from 1.25 to 2. However such a maneuver would lower the detection rate dramatically.

Theocharides et al. [12] have proposed a parallel architecture taking advantage of a grid array processor. This array processor is used as memory to store the computation data

and as data transfer unit, to aid in accessing the integral image in parallel. This implementation can achieve 52 frames per second at a 500 MHz frequency. However, details about the image resolution were not mentioned.

Another complex control scheme to meet hard real-time deadlines is proposed in [13]. It introduces a new hardware pipeline design for Haar-like feature calculation and a system design exploiting several levels of parallelism. But it sacrifices the detection rate and it is better fitted for portrait pictures.

And more recently, an implementation with NoC (Network-on-Chip) architecture is proposed in [14] using some of the same element as [12]; this implementation achieves 40 frames per second for 320 × 240 images. However detection rate of 70% was well below the software implementation (82% to 92%), due to the use of only 44 features (instead of about 4600 in [1]).

3. Sequential Implementation

In software implementation the strategy used consists of processing each sub-window at a time. The processing on the next sub-window will not trigger until a final decision is taken upon the previous one, that is, going through a set of features as a programmable list of coordinate rectangles.

In attending to implement such a cascade algorithm, each stage is investigated alone. For instance, the first stage classifier should be separated from the rest since it requires processing all the possible subwindows in an image, while each of the other relies on the result of previous stage and evaluates only the subwindows that passed through.

3.1. First Classification Stage. As mentioned earlier this classifier must run all over the image and reject the subwindows that do not fit the criteria (no face in the window). The detector is scanned across locations and scales, and subsequent locations are obtained by shifting the window some number of pixels k. Only positive results trigger in the next classifier.

The addresses of the positive sub-windows are stored in a memory, so that next classifier could evaluate them and only them in the next stage. Figure 5 shows the structure of such classifier. The processing time of this first stage is stable and independent from the image content; the algorithm here is regular. The classifier complexity on this stage is usually very low (only one or two features are considered; the decision is made of one to comparisons, two multiplications, and one addition).

3.2. Remaining Classification Stages. Next classification stages, shown in Figure 6, do not need to evaluate the whole image. Each classifier should examine only the positive results, given by the previous stage, by reading their addresses in the memory, and then takes a decision upon each one (reject or pass to the next classifier stage).

Each remaining classifier is expected to reject the majority of sub-windows and keep the rest to be evaluated later in the cascade. As a result, the processing time depends largely on the number of positive sub-windows resulted from the previous stage. Moreover the classifier complexity (number of comparisons, multiplications, and additions) increasse with the stage level.

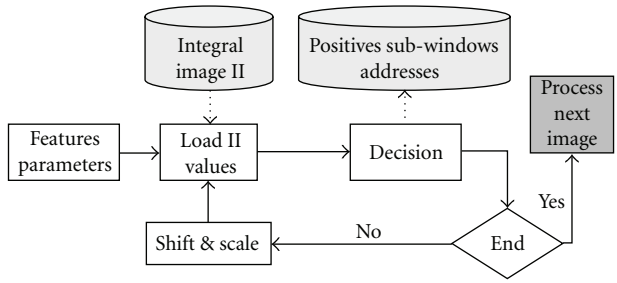

Figure 5: First cascade stage.

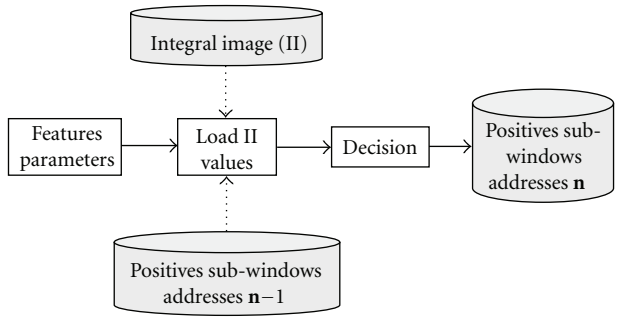

Figure 6: nth Stage classifier.

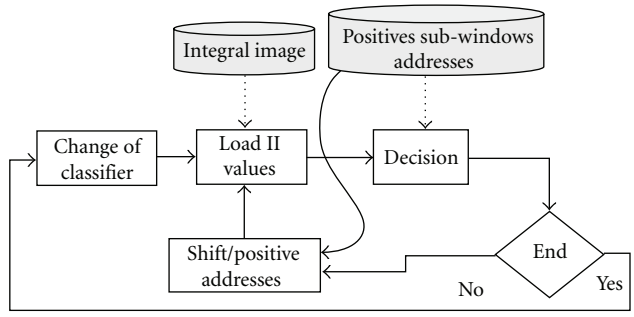

Figure 7: Sequential implementation.

3.3. Full Sequential Implementation. The full sequential implementation of this cascade is proposed in Figure 7.

For a 320 × 240 image, scanned on 11 scales with a scaling factor of 1.25 and a step of 1.5, the number of total sub-windows to be investigated is 105 963. Based on tests done in [1], an average of 10 features is evaluated per sub-window. As a result, the estimated number of decision made over the cascade, for a 320 × 240 image, is 1.3 million as an average. Thereafter around 10 millions memory access (since each decision needs 6, 8, or 9 array references to calculate the feature in play). Note that the computation time of the decision (linear combination of constants) as well as the time needed to build the integral image is negligible comparing to the overall memory access time.

Considering the speed of the memory to be 10 nanosecond per access (100 MHz), the time needed to process a full image is around 100 millisecond (about 10 images per second). However, this rate can vary with the image's content. Nevertheless, this work has been performed several times [1–3] using standard PC, and the obtained processing rate is around 10 images/s; the implementation is still not well suited for embedded applications and does not use parallelism.

4. Possible Parallelism

As shown in Section 3, Boosting-based face detector got a few drawbacks: first, the implementation still needs to be accelerated in order to achieve real time detection, and second the processing time for an image depends on its content.

4.1. Algorithm Analysis and Parallel Model. A degraded cascade of 10 stages is presented in [1]. It contains less than 400 features and achieves a detection rate between 75% and 85%. Another highly used and more complex cascade can be found in OpenCV [17] (discussed later in Section 5.2). This cascade includes more than 2000 features spread on 22 stages and achieves higher detection rates than the degraded version (between 80% and 92%) with less false positive detections. Analyzing those 2 cascade, one could notice that about 35% of the memory access takes place on each of the first two classifier while 30% on all remaining stages, which leads us to suggest a new structure (shown in Figure 8) of 3 parallel blocks that work simultaneously: in the first two blocks we intend to implement, respectively, the first and second stage classifiers, and then a final block assigned to run over all remaining stages sequentially.

Unlike the state-of-the-art software implementation, the proposed structure tends to run each stage as a standalone block. Nevertheless, some intermediate memories between the stages must be added in order to stock the positive-label windows addresses.

The new structure proposed above can upsurge the speed of the detector in one condition: since that the computation complexity is relatively small and the time processing depends heavily on the memory access, an integral image memory should be available for each block in order to gain benefit of three simultaneous memory accesses. Figure 8 shows the proposed parallel structure. At the end of every full image processing cycle, the positive results from Block1 trigger the evaluation of Block2. The positive results from Block2 trigger the evaluation of Block3. And the positive results from Block3 are labeled as faces. It should be noted that blocks cannot process simultaneously on the same image, that is, if at a given moment Block2 is working on the current image (I_1), then Block1 should be working on the next image (I_2) and Block3 should be working on the previous image (I_0). As mentioned in Section 3, the first classifier stage is slightly different from the others since it should evaluate the whole image. Hence, a "shift-and-scaling" model is needed. The positive results are stored in a memory (mem.1) and copied in another memory (mem.2) in order to be used on the second stage. The positive results are stored in a memory (mem.3, duplicated in mem.4) in order to be used in the final block.

The final block is similar to the second, but it is designed to implement all the remaining stages. Once the processing on mem.4 is finished, block 3 works the same way as in the sequential implementation: the block run back and forth through all remaining stages, to finally give the addresses of the detected faces.

This can be translated into the model shown in Figure 9. A copy of the integral image is available to each block as well as three pairs of logical memory are working in ping pong to accelerate the processing.

The given parallel model ought to run at the same speed rate as its slower block. As mentioned earlier, the first stage of the cascade requires more access memory and therefore more time processing than the second stage alone or all the remaining stages together. In the first classifier stage, all 105 963 sub-windows should be inspected using four features with eight array references each. Therefore, it requires about 3.4 million of memory access per image. Using the same type of memory as in Section 3.3, an image needs roughly 34 millisecond (29 images per second) of time processing.

4.2. Parallel Model Discussion. Normally the proposed structure should stay the same, even if the cascade structure changes, since most of the boosting cascade structures have the same properties as long as the first two cascade stages.

One of the major issues surrounding boosting-based detection algorithms (specially when applied on to object detection in a non constraint scene) is the inconsistency and the unpredictable processing time; for example, a white image will always takes a little processing time since no sub-window should be cable of passing the first stage of the cascade. As opposite, an image of thumbnails gallery will take much more time.

Though this structure not only gives a gain in speed, this first stage happens to be the only regular one in the cascade, with fixed time processing per image. This means that we can mask the irregular part of the algorithm by fixing the detector overall time processing.

As a result, the whole system will not work at 3 times the speed of the average sequential implementation, but a little bit less. However, theoretically both models should be running at the same speed if encountering a homogenous image (e.g., white or black image). Further work in Section 5 will show that the embedded implementation can benefit from some system teaks (pipelining and parallelism) within the computation that will make the architecture even faster.

Due to the masking phenomena in the parallel implementation, decreasing the number of weak classifiers can accelerate the implementation, but only if the first stage of the cascade is accelerated.

For this structure to be implemented effectively, its constraints must be taken into consideration. The memory, for instance, can be the most greedy and critical part;

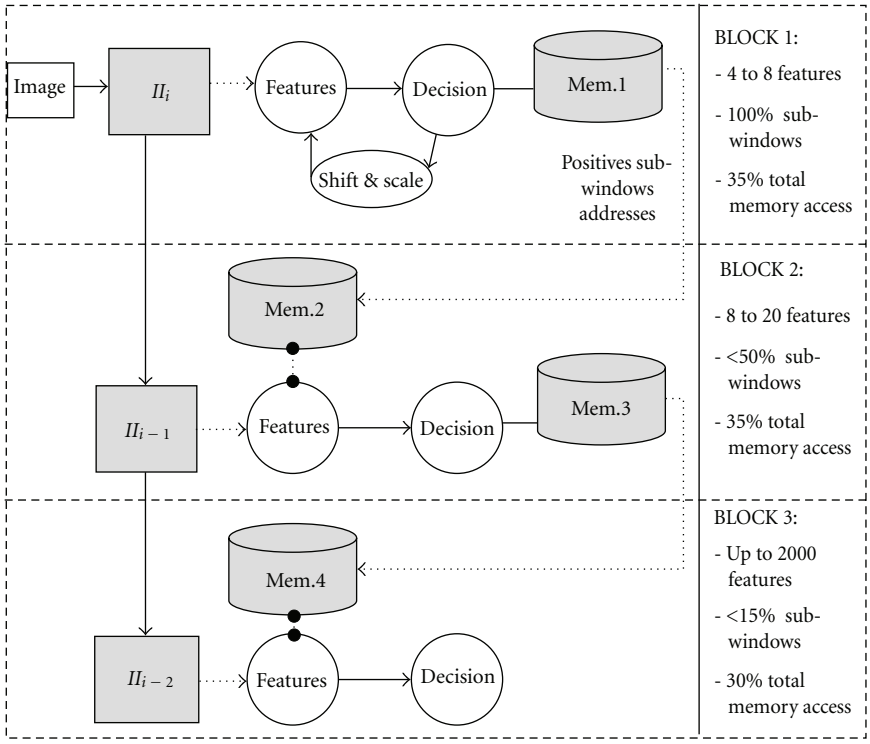

Figure 8: Parallel structure.

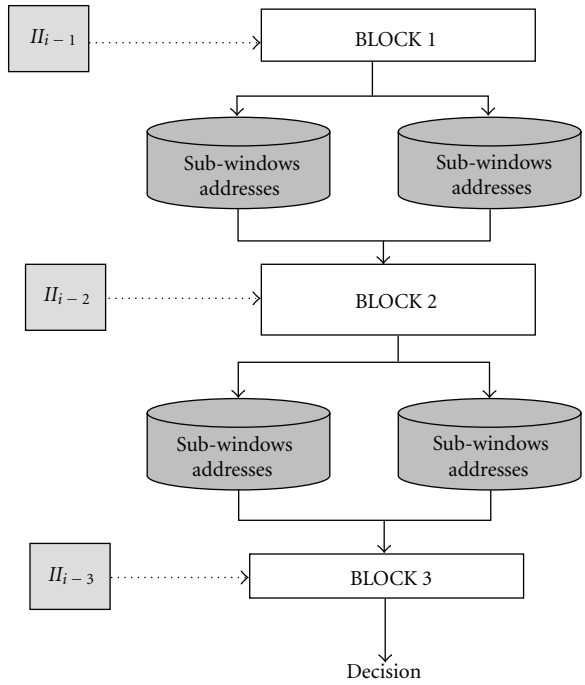

Figure 9: Data Flow.

the model requires multiple memory accesses to be done simultaneously.

It is obvious that a generic architecture (a processor, a global memory and cache) will not be enough to manage up to seven simultaneous memory accesses on top of the processing, without crashing it performances.

5. Architecture Definition: Modelling Using SystemC

Flexibility and target architecture are two major criteria for any implementation. First, a decision has been taken upon building our implementation using a high level description model/language. Modelling at a high-level of description would lead to quicker simulation, and better bandwidth estimation, better functional validation, and for most it can help delaying the system orientation and thereafter delaying the hardware target.

5.1. SystemC Description. C++ implements Object-Orientation on the C language. Many Hardware Engineers may consider that the principles of Object-Orientation are fairly remote from the creation of Hardware components. Nevertheless, Object-Orientation was created from design techniques used in Hardware designs. Data abstraction is the central aspect of Object-Orientation which can be found in everyday hardware designs with the use of publicly visible "ports" and private "internal signals". Moreover, component instantiation found in hardware designs is almost identical to the principle of "composition" used in C++ for creating hierarchical design. Hardware components can be modelled in C++, and to some extent, the mechanisms used are similar to those used in HDLs. Additionally C++ provides

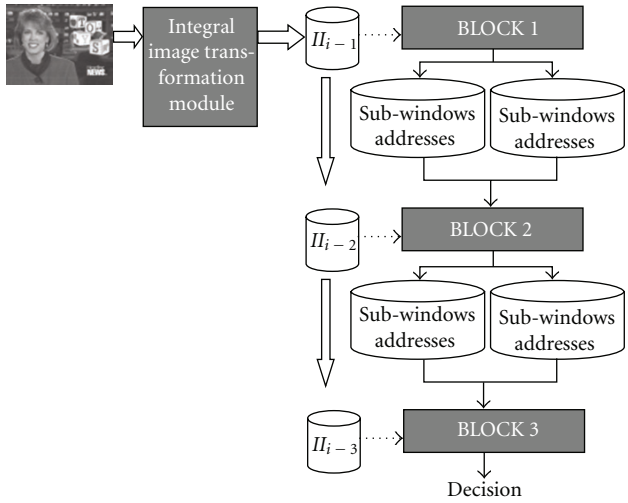

Figure 10: SystemC architecture implementation.

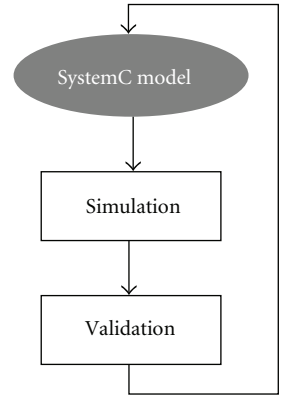

Figure 11: SystemC functional validation flow.

inheritance as a way to complement the composition mechanism and promotes design reuse.

Nonetheless, C++ does not support concurrency which is an essential aspect of systems modelling. Furthermore, timing and propagation delays cannot easily expressed in C++.

SystemC [18] is a relatively new modeling language based on C++ for system level design. It has been developed as standardized modeling language for system containing both hardware and software components.

SystemC class library provides necessary constructs to model system architecture from reactive behaviour, scheduling policy, and hardware-like timing. All of which are not available using C/C++ standalone languages.

There is multiple advantages of using SystemC, over a classic hardware description languages, such as VHDL and Verilog: flexibility, simplicity, simulation time velocity, and for most the portability, to name a few.

5.2. SystemC Implementation for Functional Validation and Verification. The SystemC approach consists of a progressive refinement of specifications. Therefore, a first initial implementation was done using an abstract high-level timed functional representation.

In this implementation, we used the proposed parallel structure discussed in Section 4.

This modeling consists of high-level SystemC modules (TLM) communicating with each other using channels, signals, or even memory-blocks modules written in SystemC (Figure 10). Scheduling and timing were used but have not been explored for hardware-like purposes. Data types, used in this modelling, are strictly C++ data types.

As for the cascade/classifiers, we chose to use the database found on Open Computer Vision Library [17] (OpenCV). OpenCV provides the most used trained cascade/classifiers datasets and face-detection software (Haar-Detector) today, for the standard prototype of Viola-Jones algorithm. The particular classifiers, used on this library, are those trained for a base detection window of 24×24 pixels, using Adaboost. These classifiers are created and trained, by Lienhart et al. [19], for the detection of upright front face detection. The detection rate of these classifiers is between 80% and 92%, depending on the images Database.

The output of our implementation is the addresses of the sub-windows which contain, according to the detector, an object of particular type (a face in our case). Functional validation is done by simulation (Figure 11). Then, multiple tests were done, including visual comparisons on a dataset of images, visual simulation signals, and other tests that consist of comparing the response of each classifier with its correspondent implemented on OpenCV's Haar-Detector software. All of these tests indicate that we were able to achieve the same result in detection rate as using the software provided by OpenCV. The images, used in these tests, were taken from the CMU+MIT face databases [20].

The choice of working with faces, instead of other object types, can help the comparison with other recent works. However, using this structure for other object-type detection is very feasible, on the condition of having a trained dataset of classifiers for the specific object. This can be considered a simple task, since OpenCV also provides the training software for the cascade detector. Even more, classifiers from other variant of boosting can be implemented easily, since the structure is written in a high-level language. As a result, changing the boosting variant is considered a minor modification since the architecture of the cascade detector should stay intact.

5.3. Modelling for Embedded Implementation. While the previous SystemC modelling is very useful for functional validation, more optimization should be carried out in order to achieve a hardware implementation. Indeed, SystemC standard is a system-level modelling environment which allows the design of various abstraction levels of systems. The design cycle starts with an abstract high-level untimed or timed functional representation that is refined to a bus-cycle accurate and then an RTL (Register Transfer Level) hardware model. SystemC provides several data types, in addition to those of C++. However these data types are mostly adapted for hardware specification.

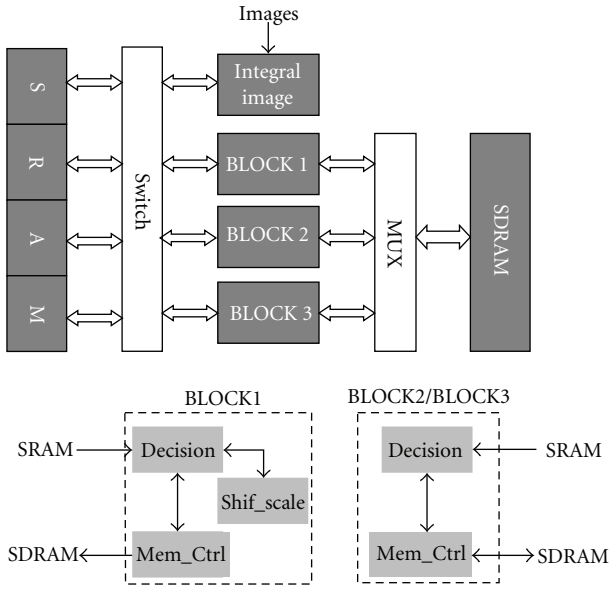

Figure 12: The global architecture in SystemC modules.

Besides, SystemC hardware model can be synthesizable for various target technologies. Numerous behavioural synthesis tools are available on the market for SystemC (e.g., Synopsys Cocentric compiler, Mentor Catapult, SystemCrafter, and AutoESL). It should be noted that for, all those available tools, it is necessary to refine the initial simulatable SystemC description in order to synthesize into hardware. The reason behind is the fact that SystemC language is a superset of the C++ designed for simulation.

Therefore, a new improved and foremost a more refined "cycle accurate RTL model" version of the design implementation was created.

Our design is split into compilation units, each of which can be compiled separately. Alternatively, it is possible to use several tools for different parts of your design, or even using the partition in order to explore most of the possible parallelism and pipelining for more efficient hardware implementation. Eventually, the main block modules of the design were split into a group of small modules that work in parallel and/or in pipelining. For instance, the module BLOCK1 contains tree compilation units (modules): a "Decision" Module which contains the first stage's classifiers. This module is used for computation and decision on each sub-window. The second module is "Shift-and-Scale" used for shifting and scaling the window in order to obtain all subsequent locations. Finally, a "Memory-Ctrl" module manages the intermediate memory access.

As result, a SystemC model composed of 11 modules (Figure 12): tree for BLOCK1, two for BLOCK2, two for BLOCK3, one for the Integral image transformation, two for the SRAM simulation, and one for the SDRAM intermediate memory (discussed later in this chapter).

Other major refinements were done: divisions were simplified in order to be power of two divisions, dataflow model was further refined to a SystemC/C++ of combined finite state-machines and data paths, loops were exploited, and timing and scheduling were taken into consideration. Note that, in most cases, parallelism and pipelining were forced manually. On the other hand, not all the modules were heavily refined; for example, the two module of SRAM were used in order to simulate a physical memory, which will never be synthesized no matter what the target platform is.

5.4. Intermediate Memory. One of the drawbacks of the proposed parallel structure (given in Section 4) is the use of additional intermediate memories (unnecessary in the software implementation). Logically, an interblocks memory unit is formed out of two memories working in ping-pong.

A stored address should hold the position of a particular sub-window and its scale; there is no need for two-dimensional positioning, since the Integral Image is created as a monodimensional table for a better RAM storage.

For a 320×240 image and an initial mask's size of 24×24, a word of 20 bits would be enough to store the concatenation of the position and the scale of each sub-window.

As for the capacity of the memories, a worse case scenario occurs when half of the possible sub-windows manage to pass through first block. That leads to around 50 000 (50% of the sub-windows) addresses to store. Using the same logic on the next block, the total number of addresses to store should not exceed the 75 000. Eventually, a combined memory capacity of less than 192 Kbytes is needed.

Even more, the simulation of our SystemC model shows that even when facing a case of consecutive positive decisions for a series of sub-windows, access onto those memories will not occur more than once every each 28 cycles (case of mem.1 and mem.2), or once each 64 cycles (case of mem.3 and mem.4).

Due to these facts, we propose a timesharing system (shown in Figure 13) using four memory banks, working as a FIFO block, with only one physical memory. Typical hardware implementation of a 192 Kbytes SDRAM or DDRAM memory, running on a frequency of at least 4 times the frequency of the FIFO banks, is necessary to replace the four logical memories.

SystemC simulation shows that 4 Kbits is enough for each memory bank. The FIFOs are easily added using SystemC own predefined sc_fifo module.

6. Hardware Implementation and Experimental Performances

6.1. Hardware Implementation. SystemC hardware model can be synthesizable for various target technologies. However, no synthesizer is capable of producing efficient hardware from a SystemC program written for simulation. Automatic synthesis tool can produce fast and efficient hardware only if the entry code accommodates certain difficult requirements such as using hardware-like development methods. Therefore, the results of the synthesis design implementation and the tool itself and the different levels of refinements done depend heavily on the entry code. Figure 14 shows the two different kinds of refinements needed to achieve a successful

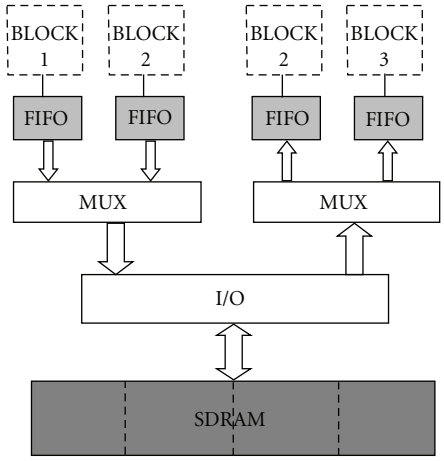

Figure 13: Intermediate Memories structure.

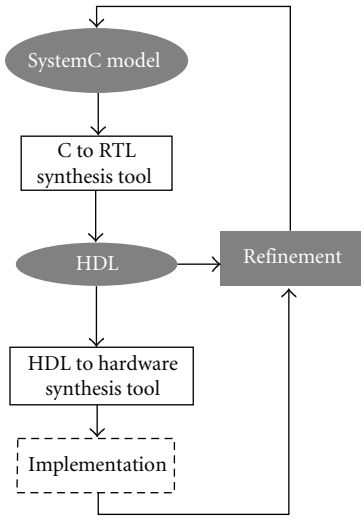

Figure 14: SystemC to hardware implementation development flow.

fast implementation, using a high-level description language. The first type of refinements is the one set by the tool itself. Without it, the tool is not capable of compiling the SystemC code to RTL level. Even so, those refinements do not lead directly to a good proven implementation. Another type of refinements should take place in order to optimize the size, the speed and sometimes (depending on the used tool) power consumption.

For our design, several refinement versions have been done on different modules depending on their initial speed and usability.

The SystemC scheduler uses the same behavior for software simulation as for hardware simulation. This works to our advantage since it gives the possibility of choosing which of the modules to be synthesized, while the rest works as SystemC test bench for the design.

Our synthesis phase was performed using an automatic tool, named SystemCrafter, which is a SystemC synthesis tool that targets Xilinx FPGAs.

Table 1: The synthesis results of the components implementations.

	Logic utilization	Used	Available	Utilization
Integral Image	Number of occupied Slices	913	10752	8%
	Number of Slice Flip Flops	300	21504	1%
	Number of 4 input LUTs	1761	21504	8%
	Number of DSPs	2	48	4%
	Maximum frequency		**129** MHz	
BLOCK 1	Number of occupied Slices	1281	10752	12%
	Number of Slice Flip Flops	626	21504	3%
	Number of 4 input LUTs	2360	21504	11%
	Number of DSPs	1	48	2%
	Maximum frequency		**47** MHz	
BLOCK 2	Number of occupied Slices	3624	10752	34%
	Number of Slice Flip Flops	801	21504	4%
	Number of 4 input LUTs	7042	21504	33%
	Number of DSPs	3	48	6%
	Maximum frequency		**42** MHz	

It should be noted that the used SystemC entry code can be described as VHDL-like synchronous and pipelined C-code (bit accurate): most parallelism and pipelining within the design were made manually using different processes, threads, and state-machines. SystemC data types were used in order to minimize the implementation size. Loops were exploited, and timing as well as variables lengths was always a big factor.

Using the SystemCrafter, multiple VHDL components are generated and can be easily added or merged into/with other VHDL components (notably the FIFO's modules). As for the testbench set, the description was kept in high-level abstraction SystemC for faster prototyping and simulation.

Basically, our implementation brings together three major components: the integral image module, the first stage decision module, and the second stage decision module (block 3 of the structure is yet to be implemented). Other components such as memory controllers and FIFO's modules are also implemented but are trifling when compared to the other big three.

Each of these components was implemented separately in order to analyze their performances. In each case, multiple graphic simulations were carried out to verify that the outputs of both descriptions (SystemC's and VHDL's) are identical.

6.2. Performances. The Xilinx Virtex-4 XC4VL25 was selected as a target FPGA. The VHDL model was back annotated using the Xilinx ISE. The synthesis results of the design implementation for each of the components are given in Table 1.

The synthesis results of the design implementation for the whole design (BLOCK1, BLOCK 2 and integral image combined) are given in Table 2.

Table 2: The synthesis results of the entire design implementation.

Logic utilization	Used	Available	Utilization
Number of occupied Slices	5941	10752	55%
Number of Slice Flip Flops	1738	21504	8%
Number of 4 input LUTs	11418	21504	53%
Number of DSPs	6	48	13%
Maximum frequency		**42** MHz	

Table 3: The synthesis results of the decision modules implementation.

	Logic utilization	Used	Available	Utilization
BLOCK 1 Decision	Number of occupied Slices	1281	10752	12%
	Number of Slice Flip Flops	626	21504	3%
	Number of 4 input LUTs	2360	21504	11%
	Number of DSPs	1	48	2%
	Maximum frequency		**47** MHz	
BLOCK 2 Decision	Number of occupied Slices	3624	10752	34%
	Number of Slice Flip Flops	801	21504	4%
	Number of 4 input LUTs	7042	21504	33%
	Number of DSPs	3	48	6%
	Maximum frequency		**42** MHz	

Table 4: The synthesis results for the new improved decision modules.

	Logic utilization	Used	Available	Utilization
BLOCK 1 Decision	Number of occupied Slices	713	10752	7%
	Number of Slice Flip Flops	293	21504	1%
	Number of 4 input LUTs	1091	21504	5%
	Number of DSPs	1	48	2%
	Maximum frequency		**127** MHz	
BLOCK 2 Decision	Number of occupied Slices	2582	10752	24%
	Number of Slice Flip Flops	411	21504	2%
	Number of 4 input LUTs	5082	21504	24%
	Number of DSPs	3	48	6%
	Maximum frequency		**123** MHz	

Table 5: The synthesis results of the refined implementation for the entire design.

Logic utilization	Used	Available	Utilization
Number of occupied Slices	4611	10752	43%
Number of Slice Flip Flops	1069	21504	5%
Number of 4 input LUTs	8527	21504	40%
Number of DSPs	6	48	13%
Maximum frequency		**123** MHz	

The clock rate of the design did not exceed the rate of its slowest component (BLOCK2). The design is capable of running with a frequency of 42 MHz. In the first block, a decision is taken on a sub-window each 28 clock cycles. Hence, this system is capable of achieving only up to 15 frames per second or 320×240 images.

Accelerating BLOCK1 and BLOCK2 is essential in order to achieve higher detection speed. BLOCK1 includes three important modules: "Decision" module, "Shift-and-Scale" module, and "Memory_ctrl" module. As for BLOCK2 it includes only "Decision" module and "Memory_ctrl" module. The decision modules however use some division and multiplication operators, which are costly in clock cycle frequency. Therefore, each "Decision" module of these two components is synthesized alone, and their synthesis results are shown in Table 3.

As expected the "Decision" Modules in both BLOCK1 and BLOCK2 are holding the implementation onto a low frequency.

Analyzing the automatic generated VHDL code shows that despite all the refinement already done, the System-Crafter synthesis tool still produces a much complex RTL code than essentially needed. Particularly, when using arrays in loops, the tool creates a register for each value, and then wired it into all possible outputs. Things get worse when trying to update all the array elements within one clock cycle. A scenario which occurs regularly in our design, for example, updating classifiers parameters after a Shifting or a Scaling. Simulation tests proved that these last manipulations can widely slowdown the design frequency. Therefore more refinement has been made for the "Decision" SystemC modules. For instance, the arrays updating were split between the clock cycles, in a way that no additional clock cycles are lost while updating a single array element per cycle.

The synthesis results for new improve and more refined decision modules are shown in Table 4. The refinements made allow faster, lighter, and more efficient implementation for the two modules. A new full system implementation is made by inserting the new "Decision" modules, its results and performances are shown in Table 5. The FPGA can operate at a clock speed of 123 MHz. Using the same logic as before, a decision is taken on a sub-window each 28 clock cycles; therefore the new design can achieve up to 42 frames per second on 320×240 images.

The simulation tests, used in Section 5.2 for the functional validation of the SystemC code, were carried out on the VHDL code mixed with a high-level test bench (the same SystemC test bench used for the SystemC validation model). The outputs of the VHDL code were compared to the outputs of the OpenCV's implementation after the first two classification stages. These tests prove that we were able to achieve the same detection results as in using the software provided by OpenCV. The design can run on even faster pace, if more refinements and hardware considerations are taken. However, it should be noted that using different SystemC synthesis tools can yield different results. After all, the amount and effectiveness of the refinements depend largely on the tool itself.

Other optimizations can be done by replacing some of the autogenerated VHDL codes from the crafter by manually optimized ones.

7. Conclusion

In this paper, we proposed a new architecture for an embedded real-time object and face detector based on a fast and robust family of methods, initiated by Viola and Jones [1].

First we have built a sequential structure model which reveals to be irregular in time processing. As estimation, the sequential implementation of a degraded cascade detector can achieve on an average of 10 frames per second.

Then a new parallel structure model is introduced. This structure proves to be at least 2.9 times faster than the sequential and provides regularity in time processing.

The design was validated using SystemC. Simulation and hardware synthesis were done, showing that such an algorithm can be fitted easily into an FPGA chip, while having the ability to achieve the state-of-the-art performances in both frame rate and accuracy.

The hardware target, used for the validation, is an FPGA based board, connected to the PC using an USB 2.0 Port. The use of SystemC description enables the design to be easily retargeted for different technologies. The implementation of our SystemC model onto a Xilinx Virtex-4 can achieve theoretical 42 frames per second detection rate for 320×240 images.

We proved that SystemC description is not only interesting to explore and validate a complex architecture. It can also be very useful to detect bottlenecks in the dataflow and to accelerate the architecture by exploiting parallelism and pipelining. Then eventually, it can lead to an embedded implementation that achieves state-of-the-art performances, thanks to some synthesis tools. More importantly, it helps developing a flexible design that can be migrated to a wide variety of technologies.

However, experiments have shown that refinements made to the entry SystemC code add up to substantial reductions in size and total execution time. Even though, the extent and effectiveness of these optimizations is largely attributed to the SystemC synthesis tool itself and designer's hardware knowledge and experience. Therefore, one very intriguing perspective is the exploration of this design using other tools for comparison purposes.

Accelerating the first stage can lead directly to a whole system acceleration. In the future, our description could be used as a part of a more complex process integrated in a SoC. We are currently exploring the possibility of a hardware/software solution, by prototyping a platform based on a Wildcard [21]. Recently, we had successful experiences, implementing a similar type of solutions in order to accelerate a "Fourier Descriptors for Object Recognition using SVM" [22] and motion estimation for MPEG-4 coding [23]. For example, the Integral Image block as well as the first and second stages can be executed in hardware on the wildcard, while the rest can be implemented in software on a Dual core processor.

References

[1] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '01)*, vol. 1, pp. 511–518, 2001.

[2] S. Li, L. Zhu, Z. Q. Zhang, A. Blake, H. J. Zhang, and H. Shum, "Statistical learning of multi-view face detection," in *Proceedings of the 7th European Conference on Computer Vision*, Copenhagen, Denmark, May 2002.

[3] J. Sochman and J. Matas, "AdaBoost with totally corrective updates for fast face detection," in *Proceedings of the 6th IEEE International Conference on Automatic Face and Gesture Recognition*, pp. 445–450, 2004.

[4] P. Viola, M. J. Jones, and D. Snow, "Detecting pedestrians using patterns of motion and appearance," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV '03)*, vol. 2, pp. 734–741, October 2003.

[5] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of online learning and an application to boosting," in *Proceedings of the ECCLT*, pp. 23–37, 1995.

[6] J. Kivinen and M. K. Warmuth, "Boosting as entropy projection," in *Proceedings of the 12th Annual Conference on Computational Learning Theory (COLT '99)*, pp. 134–144, ACM, Santa Cruz, Calif, USA, July 1999.

[7] P. Pudil, J. Novovicova, and J. Kittler, "Floating search methods in feature selection," *Pattern Recognition Letters*, vol. 15, no. 11, pp. 1119–1125, 1994.

[8] J. Sochman and J. Matas, "WaldBoost: learning for time constrained sequential detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '05)*, vol. 2, San Diego, Calif, USA, June 2005.

[9] M. Reuvers, *Face detection on the INCA+ system*, M.S. thesis, University of Amsterdam, 2004.

[10] V. Nair, P. O. Laprise, and J. J. Clark, "An FPGA-based people detection system," *EURASIP Journal on Applied Signal Processing*, no. 7, pp. 1047–1061, 2007.

[11] Y. Wei, X. Bing, and C. Chareonsak, "FPGA implementation of AdaBoost algorithm for detection of face biometrics," in *Proceedings of the IEEE International Workshop on Biomedical Circuits and Systems*, 2004.

[12] T. Theocharides, N. Vijaykrishnan, and M. J. Irwin, "A parallel architecture for hardware face detection," in *Proceedings of the IEEE Computer Society Annual Symposium on Emerging Technologies and Architectures (VLSI '06)*, 2006.

[13] M. Yang, Y. Wu, J. Crenshaw, B. Augustine, and R. Mareachen, "Face detection for automatic exposure control in handheld camera," in *Proceedings of the 4th IEEE International Conference on Computer Vision Systems (ICVS '06)*, 2006.

[14] H.-C. Lai, R. Marculescu, M. Savvides, and T. Chen, "Communication-aware face detection using noc architecture," in *Proceedings of the 6th International Conference on Computer Vision (ICVS '08)*, vol. 5008 of *Lecture Notes in Computer Science*, pp. 181–189, 2008.

[15] http://www.systemcrafter.com/.

[16] C. Papageorgiou, M. Oren, and T. Poggio, "A general framework for object detection," in *Proceedings of the International Conference on Computer Vision*, 1998.

[17] Open Source Computer Vision Library, February 2009, http://sourceforge.net/projects/opencvlibrary/.

[18] S. Swan, *An Introduction to System Level Modeling in SystemC 2.0*, Cadence Design Systems, Inc., 2001.

[19] R. Lienhart, A. Kuranov, and V. Pisarevsky, "Empirical analysis of detection cascades of boosted classifiers for rapid object detection," in *Proceedings of the 25th Pattern Recognition Symposium (DAGM '03)*, pp. 297–304, 2003.

[20] H. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 1, pp. 22–38, 1998.

[21] Annapolis Microsystems Inc, Annapolis WILDCARD System Reference Manual, Revision 2.6, 2003, http://www.annapmicro.com/.

[22] F. Smach, J. Miteran, M. Atri, J. Dubois, M. Abid, and J.-P. Gauthier, "An FPGA-based accelerator for Fourier Descriptors computing for color object recognition using SVM," *Journal of Real-Time Image Processing*, vol. 2, no. 4, pp. 249–258, 2007.

[23] J. Dubois, M. Mattavelli, L. Pierrefeu, and J. Mitéran, "Configurable motion-estimation hardware accelerator module for the MPEG-4 reference hardware description platform," in *Proceedings of the International Conference on Image Processing (ICIP '05)*, vol. 3, Genova, Italy, 2005.