9

Real Time Robust Embedded Face Detection Using High Level Description

Khalil Khattab[1], Philippe Brunet[1], Julien Dubois[2] and Johel Miteran[2]
[1]*DRIVE – ISAT, University of Burgundy,*
[2]*Le2i, University of Burgundy,*
France

1. Introduction

Face detection is a fundamental prerequisite step in the process of face recognition. It consists of automatically finding all the faces in an image despite the considerable variations of lighting, background, appearance of people, position/orientation of faces, and their sizes. This type of object detection has the distinction of having a very large intra-class, making it a particularly difficult problem to solve, especially when one wishes to achieve real time processing.

A human being has a great ability to analyze images. He can extract the information about it and focus only on areas of interest (the phenomenon of attention). Thereafter he can detect faces in an extremely reliable way. Indeed, a human being is able to easily locate faces in its environment despite difficult conditions such as occlusions of parts of a face and bad lightening. Many studies have been conducted to try to replicate this process, automatically using machines, because face detection is considered as a prerequisite for many computer vision application areas such as security, surveillance, and content based image retrieval.

Over the last two decades multiple robust algorithmic solutions were proposed. However, researches in the field of computer vision and pattern recognition in particular tend to focus on the algorithmic and functional parts. This generally leads to implementations with little constraints of time, computing power and memory. Most of these techniques, even if they achieve good performance in terms of detection, are not suited for real time application systems. Nonetheless, Boosting–based methods, firstly introduced by Viola and Jones in (Viola & Jones, 2001; 2002), has led the state-of-the-art in face detection systems. These methods present the first near real time robust solution and by far the best speed / detection compromise in the state-of-the-art (up to 15 frames/s and 90% detection on 320x240 images). This family of detectors relies upon a cascade of several classification stages of progressive complexity (around 20-40 stages for face detection). Depending on its complexity, each stage contains several classifiers trained by a boosting algorithm (Freund & Schapire, 1995; Lienhart, Kuranov, & Pisarevsky, 2003; Viola & Jones, 2002)

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 sub-windows that have to be evaluated by the detector which selects those containing faces. This approach is optimized for a sequential implementation but this implementation has two major drawbacks: high dependency between the different stages of the detector and irregularity in time processing.

Most of the Boosting–based face detection solutions deployed today are general purpose processors software. But 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. Even more the current technology of multi-core processor cannot be exploited to its full limits because of the dependency between the different stages. Seeking some improvement over the software, several attempts were made trying to implement face detection on multi-FPGA boards and multiprocessor platforms using programmable hardware , however in almost all the cases the resulting implementation are capable to accelerate the detection but degrade the detection accuracy.

The major difficulties in a parallel implementation of the cascade detector (boosted based methods) are the full dependency between the consecutive stages and classifier repartition which is optimized for sequential implementation. Based on this observation and our belief that a useful acceleration of the face detection should not compromise the detection performances, our main contribution is a new structure that exploits intrinsic parallelism of a boosting-based object detection algorithm without compromising its accuracy. At first we present a new stage grouping capable of equally partition the computation complexity of the algorithm. Based on this partitioning, a new parallel model is proposed. This model is capable of exploiting the parallelism and the pipelining in these algorithms, and provides regularity in time processing. It can also be customizable according to the cascade in use.

This chapter also 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. The processing blocs are modeled using SystemC. 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.

Finally, using the results of the processing blocks' implantations, we define a new architectural structure of the implementation including the interconnectivity of the memory blocks and the number and the type of the used memories. This final system proves capable of achieving 47 frames per second for 320x240 images as well as keeping the same detection accuracy as the original method. In the end, we show a detailed comparison between our system and the other state-of-the-art embedded implementation for boosting based face detection.

This chapter can be considered as a continuation of previously published work (Khattab, Dubois & Miteran 2009) in which we proposed a new architecture for an embedded real-time face detector based on a fast and robust family of methods, initiated by Viola and Jones. However only parts of the processing blocks were implemented, memories types and interconnection wasn't optimized and the system validation was made in simulation

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 as well as their size are already known. The object can be anything from a vehicle, human face (Figure 1), human hand, pedestrian (Viola, Jones, & Snow, 2003), etc. The majority of the boosting based object detectors work-to-date has 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.

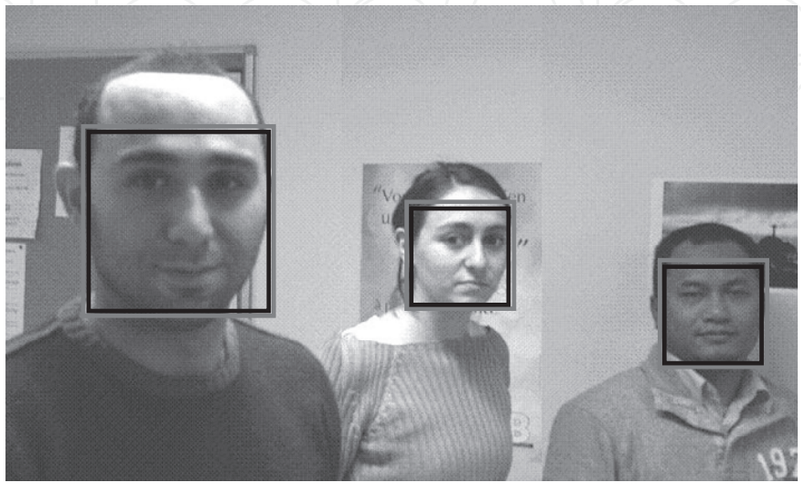


Fig. 1. Example of face detection

2.1 Theory of Boosting Based object detectors

2.1.1 Cascade detection

The structure of the cascade detector (introduced by Viola and Jones) 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 sub-windows should be evaluated by the next stage. If a sub-window is labeled at the current classifier as non-face 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.

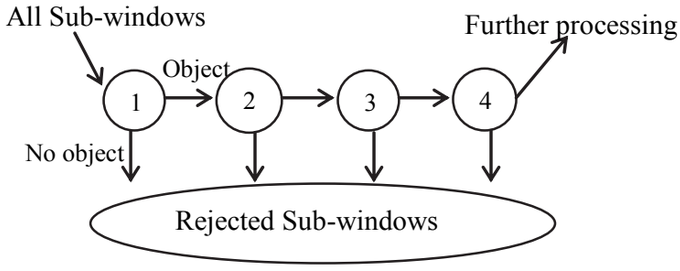


Fig. 2. Cascade detector

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 (Papageorgiou, Oren, & Poggio, 1998): 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).

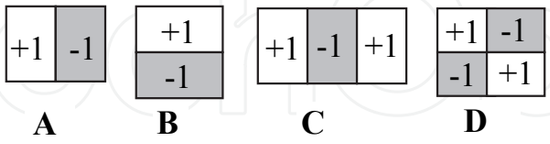


Fig. 3. Rectangle Features

Using a particular representation of the image so-called the Integral Image (II), it is possible to compute very rapidly 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 24x24 pixels sub-window size, the over-complete feature set of all possible features computed in this window is 45,396: 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.

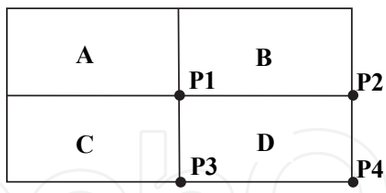


Fig. 4. The sum of pixels within Rectangle D can be calculated by using 4 array references; SD= II [P4] – (II [P3] + II [P2] – II [P1])

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:

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

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 Equation 3.

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

Where θ is the stage threshold, a_t is the weak classifier's weight and T 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 320x240 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 are considered as prerequisite step for some of the main application such as biometric, content-based image retrieval systems, surveillance, auto-navigation, etc. 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 trade-offs then 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 it use for some memory heavy applications.

For our knowledge, few attempts were made trying to implement Boosting based face detection on embedded platforms. Nevertheless, these proposed architectures were configurable hardware based implementations and most of them couldn't achieve high detection frame rate speed while keeping the detection rate close of that's of the original implementation. For instance, in order to achieve 15 frames per second for 120x120 images, Wei et al. (Wei, Bing, & Chareonsak, 2004) choose to skip the enlargement scale factor from

1.25 to 2. However such a maneuver would lower the detection rate dramatically. Theocharideset al. (Theocharides, Vijaykrishnan, & Irwin, 2006) has 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 (M Yang, Wu, Crenshaw, Augustine, and Mareachen 2006). 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 (Lai, Marculescu, Savvides, & Chen, 2008) using some of the same element as (Theocharides, Vijaykrishnan, & Irwin, 2006), this implementation achieves 40 frames per second for 320x240 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 thousands).

3. Global parallelism

In this section we provide a detailed analysis of the boosting based face detection algorithm, in order to extract as much useful information for designing an efficient parallel architecture. For this, we first present an analysis of the sequential implementation. We then analyze the different stages of the cascade, and the computational complexity of each one of them. Finally, we propose a parallel structure to accelerate this algorithm.

3.1 Sequential implementation

The strategy used in software implementation 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 i.e. going through a set of features as a programmable list of coordinate rectangles. The processing time of an image depends on two factors: the processing time of each sub-window and the number of sub-windows to process.

The processing time of a sub-window can vary dramatically depending on the complexity of its content. For example, an image of uniform color will definitely take less time to process than an image containing several faces. For this reason, the cascade-like detection algorithms are irregular and not predictable. In fact, Viola and Jones have already indicated that the speed of the cascade detector depends on the image content and accordingly the average number of weak classifiers evaluated per sub-window on an image sequence. Moreover, their tests showed that, on average, 10 weak classifiers are evaluated per sub-window. These tests were done using CMU image database (Rowley, Baluja, & Kanade, 1998).

3.1.1 OpenCV implementation

Several variants of Boosting based face detection can be found in today's literature. However the principal of cascade detection remain the same in almost all of these variants. As for the cascade /classifiers, we chose to use the database found on Open Computer Vision Library (OpenCv[1]). OpenCV provides the most used trained cascade/classifiers datasets and face-

[1]OpenCv. (2009). Open source computer vision library. http://sourceforge.net/projects/opencvlibrary/

detection software today. The particular classifiers, used on this library, are those trained with a base detection window of 24x24 pixels, using Adaboost. These classifiers are created and trained, by Lienhart et al (Lienhart & Maydt, 2002) , for the detection of upright front face detection. The detection rate of these classifiers is between 80% and 92%, depending on the images Database. This cascade includes more than 2500 features spread on 25 stages.
Using this sequential implementation, we decided to investigate each stage. For instance, the first stage classifier should be separated from the rest since it requires processing all the possible sub windows in an image, while each of the other relies on the results of previous stage and evaluates only the sub windows that passed through.

3.1.2 Classification stages

As mentioned earlier the first stage of the cascade must run all over the image and rejects the sub-windows 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.

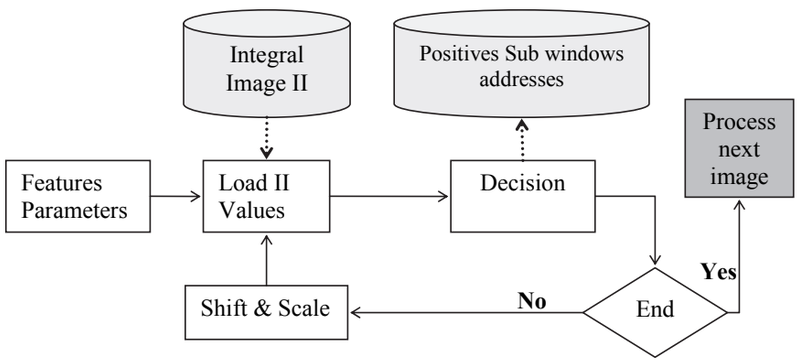


Fig. 5. First cascade stage

The other 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 stage 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 increases with the stage level.

3.1.3 Full sequential implementation analysis

For a 320x240 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 (Viola & Jones, 2001), an average of 10 features are evaluated per sub-window. As a result, the estimated number of decision made over the cascade, for a 320x240 image, is 1.3 million as

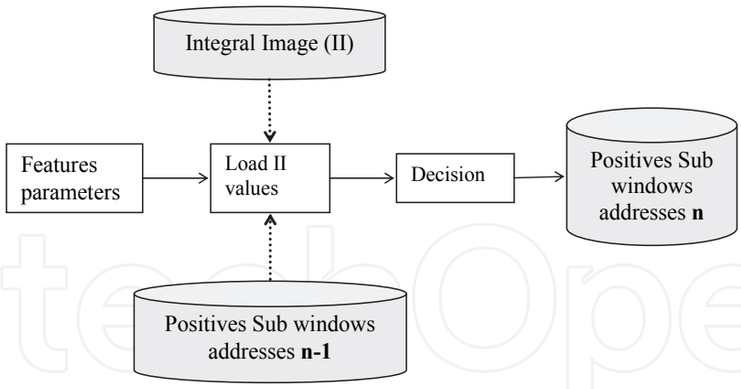


Fig. 6. n_{th} Stage classifier

on average. Thereafter around 10 million 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, are negligible comparing to the overall memory access time.
Considering the speed of the memory is 10 ns per access (100 MHz), the time needed to process a full image is around 100 ms (about 10 images per second). However, this rate can vary with the image's content.

3.2 Possible parallelism

We applied the frontal face detector, "Discrete AdaBoost" of OpenCV, on the CMU image database in order to analyze the number of sub-windows rejected per stage and subsequently the number weak classifiers (features) evaluated per sub-window. Indeed, 75 081 800 sub-windows have triggered a total of 668 659 962 evaluations of weak classifiers. Hence, only 9 weak classifiers are evaluated per sub-window on average.
Even more, these analysis revealed another major characteristic of the cascade implementation: The unbalance in processing loads between the different stages. This is caused by the fact that the boosting based face detection is optimized for sequential implementation. The training phase of the boosting methods is configured to reject as much sub-windows as early as possible.
On average, about 35% of the total memory access (and processing) load takes place in each of the first two stages while less than 32% take place in all of the remaining stages combined
In the rest of this section, we show how to take advantage of such unbalance in memory access in order to propose a feasible parallel model

3.2.1 Pipeline solution

The previous analysis of the OpenCV cascade revealed that more than a third of the memory access take place on each of the first two cascade stages while less than third in all remaining stages. This analysis leads us to suggest a new pipelined solution (shown in Figure 7) of 3 parallel blocks that work simultaneously: In the first two blocks we intend to implement respectively the first and second stage classifier, 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 positively-labeled windows addresses.

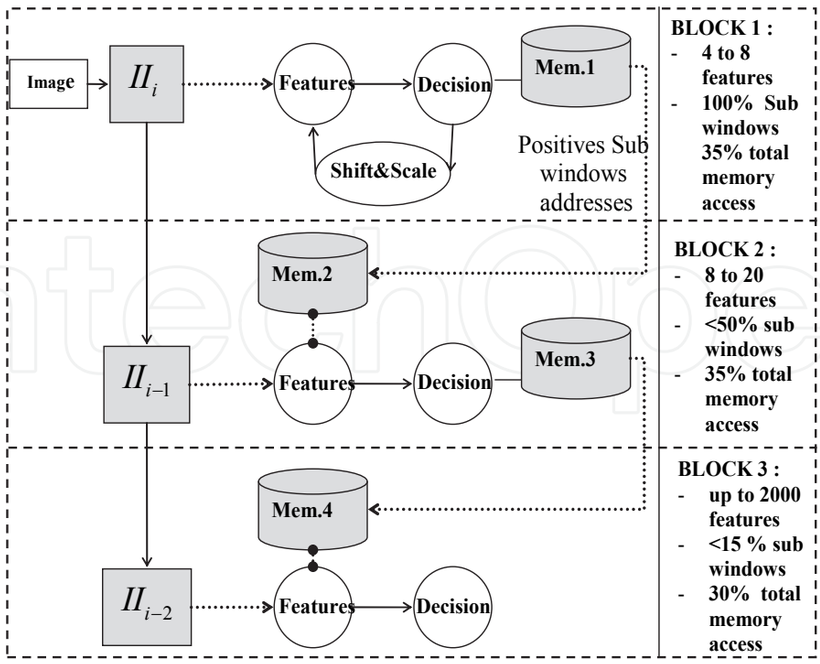


Fig. 7. Parallel structure

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 7 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 i.e. if at a given moment Block1 is working on the current image I_n, then Block2 should be working on the previous image I_{n-1} and Block3 should be working on the one before I_{n-2}.

This structure requires data dependency between the parallel blocks. The addresses of the sub-windows classified positively by Block 1 shall be transmitted to Block 2. Similarly, the addresses of sub-windows classified positively by Block 1 and 2 respectively, must be transmitted to the Block 3.

The large numbers of sub-windows addresses require the use of intermediate memories, which will manage the communication between the different blocks. At any given time, Block 1 processes on image I_n and stores the addresses of its positively labeled sub-windows in a memory (*mem.1*). At the same time Block 2 processes an image I_{n-1} but only the sub-windows positively labeled by the first and whose addresses are stored in memory *mem.2*. The addresses of sub-windows positively labeled by Block 2 are stored in a memory (*mem.3*). Respectively, Block 3 processes an image I_{n-2} , but only its sub-windows positively labeled by Block 2 and whose addresses are read from a memory mem.4. 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.

After each image cycle, and the memories *mem.1* and *mem.2* are swapped, same goes for *mem.3* and *mem.4*

This can be translated into the model shown in Figure 8. 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.

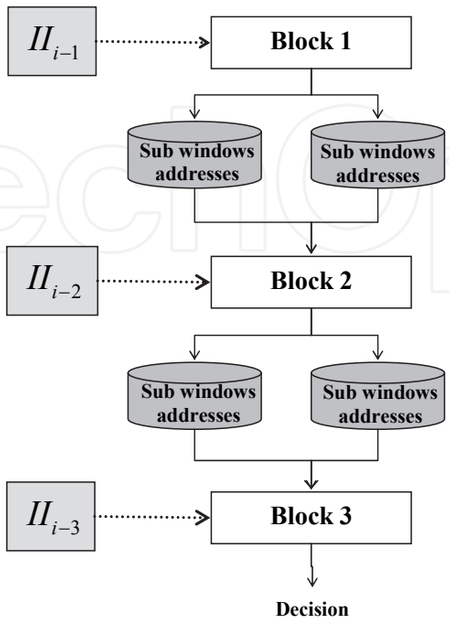


Fig. 8. Data Flow

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 three 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 3.1.4, an image needs roughly 34 ms (29 images per second) of time processing.

3.2.2 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 (especially when applied on to face detection in a non-constraint scene) is the inconsistency and the unpredictable processing time e.g. a white image will always takes a little processing time since no sub-window should be capable of passing the first stage of the cascade. As opposite, an image of thumbnails gallery will take much more time.

The proposed 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. 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; the model requires multiple memory accesses to be done simultaneously. The definition of an architectural structure of this representation depends on two factors: the nature of memory access and the desired performance of the system. For instance, the memory blocks of the integral images are used in the computation of the Haar-features rectangles. The integral images are stored in a linear fashion; however the reading access depends on the position, the size and the parameters of the weak classifier to be evaluated. It is for this reason that access to these memories are made randomly. On the other hand, the blocks of intermediate memories are used to store and read the addresses of the sub-windows. Both the write and the read of these addresses are done sequentially. To optimize performance of architecture, it is imperative to use memories appropriate to the nature of each of the different access types. Thus, as we shall see in the implantation section 4, we recommend using two different types of memory.
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.

4. Architecture definition and implementation

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, better bandwidth estimation, better functional validation, and more importantly it can help delaying the system orientation and thereafter delaying the hardware target.

4.1 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 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[2] 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.

[2]SystemC, Initiative Open. Initiative Open SystemC, (OSCI) http://www.systemc.org.

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.

4.1.1 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 3.
This modeling consists of high level SystemC modules (TLM) communicating with each other using channels, signals or even memory-blocks modules written in SystemC. 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.
Functional validation of our model SystemC is performed using a simulation phase (Figure 9). We simulate the behavior of a SystemC model by executing its processes in a pseudo concurrent way. The simulation stops when there is no eligible process and event notification. To manage the progress of simulation time, the SystemC simulator has a timed notifications schedule to be triggered. This schedule is a list of notifications of timed event that is sorted according to the time of such notification triggers.
Functional validation of the system was performed by comparing the results of the SystemC written structure with the results of OpenCV's software implementation, using 25 random images from the CMU image database.

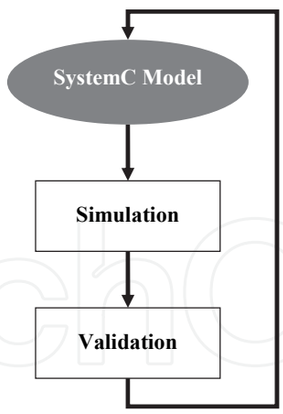


Fig. 9. SystemC functional validation flow

4.1.2 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.
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 simulation-like 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 three 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 12 modules: three for Block1, two for Block2, three for Block3, one for the Integral image transformation, 3 for the memories.
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/scheduling were taken into consideration. Note that in most cases, parallelism and pipelining were forced manually.
However, this level of description can vary depending on the needs of the high-level synthesis tool used for the hardware implementation. For example tools like Mentor Graphics CatapultC can propose and test different alternatives of parallel and pipeline implementations for high level of description C/SystemC. Other tools like SystemCrafter require manual coding of parallelism and pipeline operations.
On the other hand, not all the modules were heavily refined, for example the three memory modules were used in order simulate physical memories, which will never be synthesized no matter what the target platform is.

4.2 High level synthesis

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 depend heavily and the tool itself, and the different level of refinements done on the entry code. Figure 10 shows the two different kinds of refinements needed to achieve a successful 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 don't 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 refinements 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.

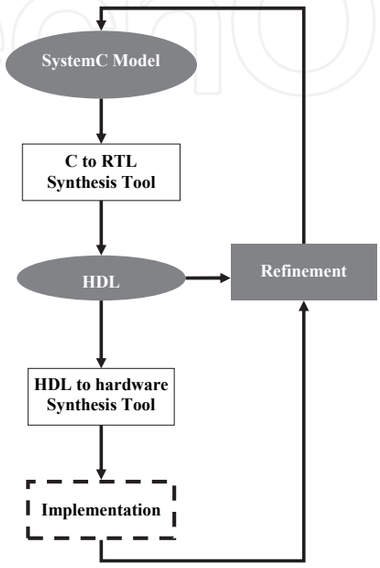


Fig. 10. SystemC to hardware implementation development flow

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 were 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 four major components: the Integral Image module, the first stage decision module, the second stage decision module and Block 3 which runs sequentially the rest of the cascade stages. 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 output of both descriptions (SystemC's and VHDL's) are identical.
The reduced number of weak classifiers in the first two stages (three and nine respectively) allows us to store their settings in internal memory (LUT type). In the case of Block3, the number of weak classifiers is about 2500. Knowing that every weak classifier has 13 parameters (addresses rectangles, weight, threshold...) and each of these parameters is defined with an integer data type of size 24 bits. The size of memory needed to store these parameters is therefore: *2500* x *13* x *24* = *780 000* = *780Kbits*. It is therefore clear that the

storage of these parameters in internal LUT type memory can occupy a large number of logical blocks (slices) within the FPGA. We chose to store these parameters in the blocks RAM (BRAM). Indeed the use of SystemCrafter facilitates this task by using the type ram_block. This structure can be used in the same way as type "ram" plus it handles the storage and the control of the FPGA's BRAMS.

4.3 Performances

The Xilinx Virtex-4 XC4VL25 was selected as a target FPGA. The VHDL model was back annotated using the Xilinx ISE.

4.3.1 Non-optimized implementation

The synthesis results of the design implementation for each of the components are given on Table1.

	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%
	Maximum frequency	**129 MHz**		
BLOCK1	Number of occupied Slices:	1281	10752	12%
	Number of Slice Flip Flops:	626	21504	3%
	Number of 4 input LUTs:	2360	21504	11%
	Maximum frequency	**47 MHz**		
BLOCK2	Number of occupied Slices:	3624	10752	34%
	Number of Slice Flip Flops:	801	21504	4%
	Number of 4 input LUTs:	7042	21504	33%
	Maximum frequency	**42 MHz**		
BLOCK3	Number of occupied Slices:	3178	10752	29%
	Number of Slice Flip Flops:	722	21504	3%
	Number of 4 input LUTs:	3014	21504	14%
	Maximum frequency	**43 MHz**		

Table 1. The synthesis results of the components implementations

The clock rate of the design did not exceed the rate of its slowest component. Therefore it is necessary to simulate and estimate the average speed of each block. Another big advantage of SystemC is the possibility of using C++/SystemC testbenches with VHDL models. And using simulation tools such as ModelSim, we can determine exactly the number of cycles needed to process a sub-window and thereafter the speed of each block. For instance Block1 can operate with a maximum frequency of 47 MHz and can process a sub-window in 42 clock cycles. Table 2 shows the average speed of each hardware Block using the CMU image database.

This actual implementation is capable of achieving only up to 11 frames per second on 320x240 images. Accelerating Block 1, Block 2 and Block3 is essential in order to achieve higher detection speed.

	Maximum Frequency (MHz)	Number of Cycle per Sub-windows	Average speed (frame per second)
Block1	47	42	11
Block2	42	112	11,3
Block3	43	192 to 2400	16

Table 2. speed of processing blocks before optimization

4.3.2 Optimized implementation

Analyzing the automatically generated VHDL code shows that despite all the refinement already done, the SystemCrafter 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 that occurs regularly in our design e.g. updating classifiers parameters. Simulation tests proved that these last manipulations can widely slowdown the design frequency. Therefore more refinements have 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 the new improve and more refined decision modules are shown in Table 3. The refinements made allow faster, lighter, and more efficient implementation for all 3 modules. Even more, the ModelSim simulation of our design shows that the refinements also allow achieving less cycles per decision (sub-windows processing) in all 3 blocks. Table 4 shows the new average speed of each VHDL Block using the CMU image database.

	Logic Utilization	Used	Available	Utilization
BLOCK1	Number of occupied Slices:	713	10752	7%
	Number of Slice Flip Flops:	293	21504	1%
	Number of 4 input LUTs:	1091	21504	5%
	Maximum frequency	**127 MHz**		
BLOCK2	Number of occupied Slices:	2582	10752	24%
	Number of Slice Flip Flops:	411	21504	2%
	Number of 4 input LUTs:	5082	21504	24%
	Maximum frequency	**127 MHz**		
BLOCK3	Number of occupied Slices:	1703	10752	16%
	Number of Slice Flip Flops:	405	21504	2%
	Number of 4 input LUTs:	2616	21504	12%
	Maximum frequency	**127 MHz**		

Table 3. The synthesis results for the new improved decision modules

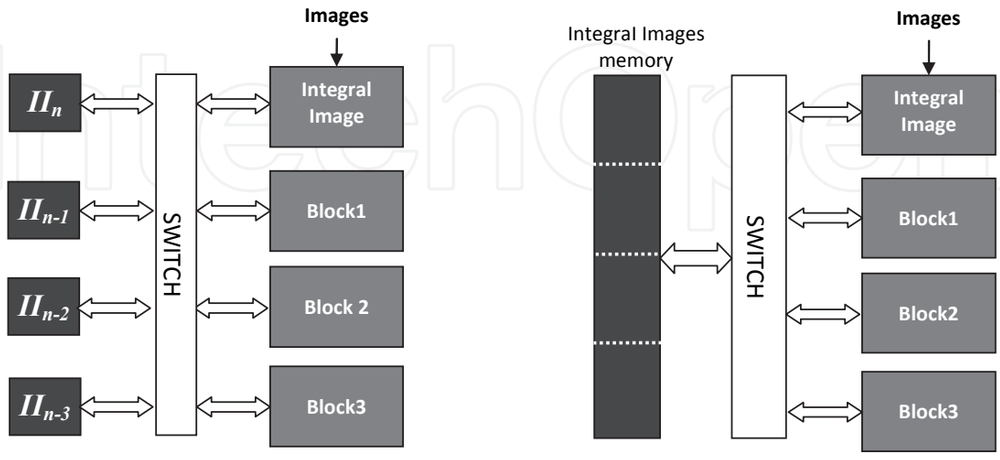
The FPGA can operate with a frequency of 127 MHz. Using the same logic as before, a system is as fast as its slowest blocks, therefore the new design can achieve up to 47 frames per second on 320x240 images.
The design can run on even faster pace, if more refinements and hardware considerations are taken. 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 auto-generated VHDL codes from the crafter with manually optimized ones.

	Maximum Frequency (MHz)	Number of Cycle per Sub-windows	Average speed (Image per second)
Block1	127	28	47
Block2	127	76	47,7
Block3	127	132 to 22890	50

Table 4. Speed of processing blocks after optimization

4.4 Architectural structure and memory blocks connectivity

The synthesis results from the previous paragraph helps showing the limitation in processing speed. However the performance of design also depends on the architectural structure of the implementation including the interconnectivity of the memory blocks and the number of physical memory used.
A first possible solution is to use a system with separated memory blocks (Figure 11.a.). At the end of each image cycle, a "switch" module is in charge of swapping memory blocks at a physical level. This solution can maximize processing performances, but it is too costly in terms of physical memory and resultant physical interconnections.
Reducing the number of physical memory can be explored in other solutions that integrate timesharing systems. Figure 11.b. shows a solution with a single physical memory for



a. Separated memory blocks b. Timesharing memory system

Fig. 11. Different architecture solution for memory's interconnection

Integral Images (same can be done for the intermediate memory blocks). The physical memory is logically divided into four memory banks that work in queues. The swapping of the memory access is also done by "Switch". But unlike the previous solution, where the swapping is made at a physical level, they are calculated at a logic level. This solution is optimal when the number of memory accesses is small. However in cases where memory accesses are the limiting factor of the system, this solution is less efficient than the using separated memories.

4.4.1 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 inter-blocks 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 320x240 image and a base sub-window size of 24x24, a word of 32 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 worst case scenario occurs when half of the possible sub-windows manage to pass through first block. That leads to around 2 x 53,000 (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 168 000. Eventually, a combined memory capacity of less than 1 Mbytes is needed. 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 76 cycles (case of mem.3 and mem.4). The access on these memories is regular since the writing and the reading are always done sequentially. Due to these facts, we propose a timesharing system (shown in Figure 12) using four memory banks, working as a FIFO block, with only one physical memory. In order to determine the exact characteristics of the needed memory, several testbenchs were created to compute the maximal bandwidth needed as well as the optimal FIFO queue size in worst

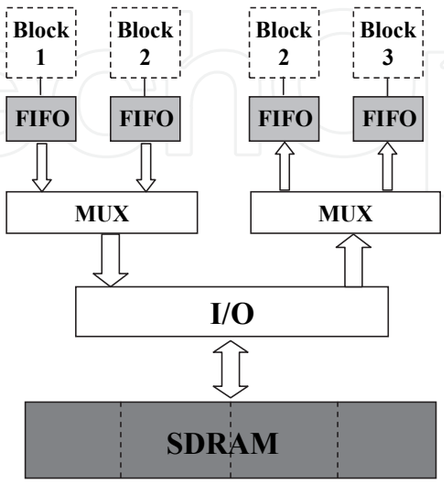


Fig. 12. Intermediate Memories structure

case scenario. The results of these simulation shows that a frequency of 17 MHz and a buffer size (each FIFO block) of 10 are enough for our configuration.
Typical hardware implementation of a 1 Mbytes SDRAM memory, running on a frequency of anything higher than 17 MHz, is enough to replace the four logical memories. Moreover, the required buffer size is very small. They FIFO are easily implemented with a limited number of block RAM (BRAM). We can use two BRAMS in dual-port mode or four BRAMS single port mode.

4.4.2 Integral image memory

The set of processing blocks (Integral Image, Block 1, Block2 and Block3) need to access 4 different Integral Images simultaneously. To achieve a detection of 47fps on 320x240 images, each of block must operate at their maximum frequency. In the worst case scenario the total bandwidth needed to access all Integral images is about 10,7 Gigabits/s.
However unlike the intermediate memory, the access to the integral images is never sequential or regular. The memory usage of SDRAM is not suited. In fact, the non-consecutive data transfer will drop dramatically the SDRAM bandwidth. Typically, for a latency of two cycles, the available bandwidth is divided by 3.The use of SRAMs appears to be more appropriate. For these reasons, we propose a solution with four SRAM memory units (Figure 13). Though, a solution with less memory units can be considered, the use of four minimizes the complexity of the switching module "SWITCH_II". At the end of each image cycle, a circular swapping logic is performed between memory units.

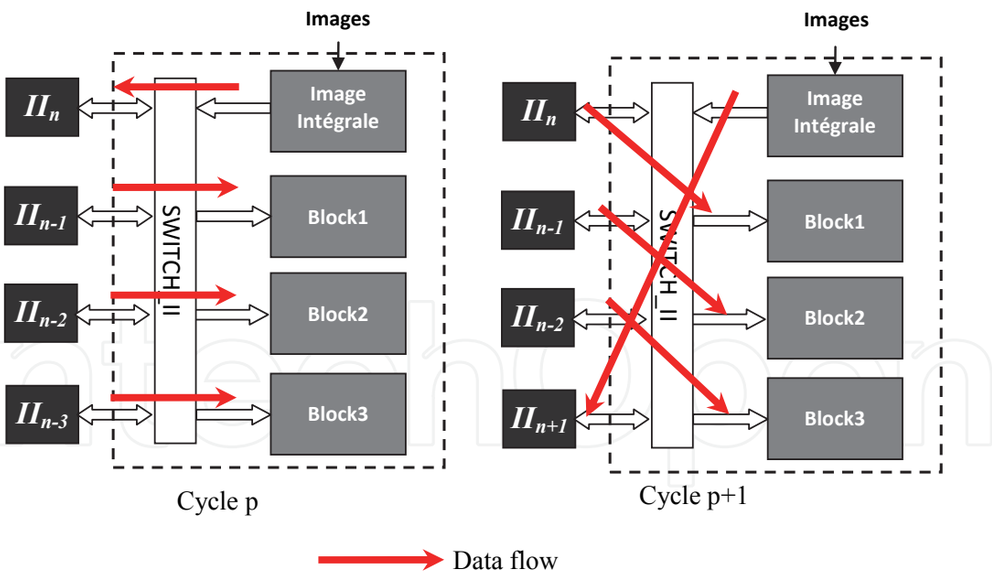


Fig. 13. Integral image interconnection

4.4.3 Final architecture structure

After establishing the interconnectivity of our architectural structure, we synthesize the whole system which includes Block 1 to 3, the Integral image module, and the switching modules. The results and the performances are shown in Table 5. The FPGA can operate at

a clock speed of 127 MHz. Figure 14 Shows the final proposed architecture which is capable of 47 images per second.

The simulation tests, used in section 4.1 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. These tests prove that we were able to achieve the same detection results as in using the software provided by OpenCV.

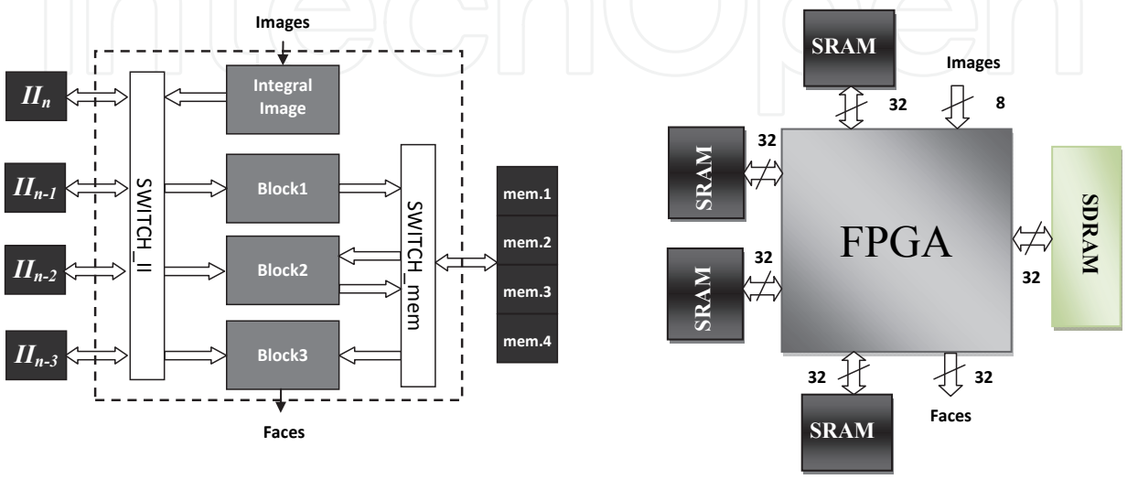


Fig. 14. Proposed architecture

Logic Utilization	Used	Available	Utilization
Number of occupied Slices:	6075	10752	57%
Number of Slice Flip Flops:	1729	21504	8%
Number of 4 input LUTs:	10711	21504	50%
Number of DSPs:	9	48	19%
Maximum frequency	**127 MHz**		

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

4.5 State-of-the-art comparision

We compare in this section, the performance of our embedded implementation with other known embedded implementations for the boosting based face detection in the literature. Comparisons are made in terms of speed (frame per second) and detection rates.The results of these comparisions are shown in Table 6.

One of the major challenges when trying to implement a real time embedded solution for this type of algorithms is the large number of features to be implemented in a cascade. In fact, the number of used features is usually betwen 2000 and 6000, depending on the training phase. Furthermore, the cascade implementation and the irregular nature of the stages of the cascade, make it extremely difficult to proposed a parallel structure capable of accelerating the detection without degrading the performances.

Implementation	Image size	smallest sub-window size	Frames per second	Test database	Features used	Detection rate
Wei & al. (Wei, Bing, & Chareonsak, 2004)	120x120	24x24	15	CMU	NA	50%
Yang et al.	320x240	24x24	13	P I DB[3]	140	75%
Theocharides (Theocharides, Vijaykrishnan, & Irwin, 2006)	NA	NA	52	P I DB	Less than 150	NA
Lai & al. (Lai, Marculescu, Savvides, & Chen, 2008)	320x240	20x20	40	P I DB	42	75%
Author's implementation	**320x240**	**24x24**	**47**	**CMU**	**2500**	**88%**

Table 6. Comparison of embedded implementation of Boosting based face detection

In the literature we can find a lot of attempts to accelerate the boosting based face detection. However the authors in these works have sought to reduce the overall computation burden, without taking into consideration the local burden of computation at each stage of classification. By consequence, they have abandoned the cascaded architecture in favor of a single complex stage of classification(e.g.only 42 features in the implementation of Lai & al.). Indeed, it is easier to exploit the parallelism of a single stage with several features than the parallelism of a complex cascade with very high data dependencies. However, this approach has two major drawbacks:

- Each sub-window must be evaluated by a large number of features (42 to 150) , when the average number of evaluated features per sub-windows in a cascade is generally less than 10.
- The evaluation of these features must be done in parallel to speed up the detection time. The amount of necessary resources for these computations is thereby increased, which explains the limited number of features used in the hardware implementations. And therefore the detection rates of these implementations are well under the ones set by the software implementations.

However, unlike the listed embedded implementations, our architecture is capable of supporting a large number of features. Indeed, we were able to implement the same full cascade (more than 2500 features) as the "default" one found in OpenCV. Hence the detection rates are the same as the software implementation. Our implementation can achieve up to 47 fps on the CMU image database, while processing about 106 000 sub-windows. The implementation proposed by Theocharides can achieve slightly higher of number of frames per second, but the authors did not provide sufficient details on important factors, such as images size and the smallest sub-window size. These configurations are essential in determining the speed of the detection (and the detection rates), for exemple taking changing the smallest sub-window size from 24x24 to 32x32 can divide the computation burden by 2 and therefore accelerate the detection by a factor of 2.

[3]proprietary image databases

5. Conclusion

This chapter can be considered as a continuation of previously published work (Khattab, Dubois & Miteran 2009) in which we proposed a new architecture for an embedded real-time face detector based on a fast and robust family of methods, initiated by Viola and Jones. The most notable differences between the 2 articles are: The implementation of the third processing block (Block3), the new architecture structure including memories types and interconnection, and finally a full system validation and tests.

First we analyse the sequential structure model which reveals to be irregular in time processing and in load partitioning. Then a new parallel structure model is introduced. This structure proves to be at least 3.4 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 a 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 a 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 a theoretical 47 frames per second detection rate for 320x240 images. And Unlike the state-of-the-art embedded implementation, we were able to implement the whole cascade detector (with all the features) as the one use in the software implementation. This has led to achieve practically the same result in detection rates as in the software implementation.

On the other hand, 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. Recently, we had successful experiences, implementing a similar type of solutions in order to accelerate a "Fourier Descriptors for Object Recognition using SVM"(Smach, Miteran, Atri, Dubois, & Gauthier, 2007) and motion estimation for MPEG-4 coding(Dubois, Mattavelli, Pierrefeu, & Mitéran, 2005) . 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.

6. References

Dubois, J., Mattavelli, M., Pierrefeu, L., & Mitéran, J. (2005). Configurable Motion-Estimation Hardware Accelerator Module for the MPEG-4 Reference Hardware Description Platform. *Proceeding of IEEE International Conference on Image processing*.

Freund, Y., & Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. *In Computational Learning Theory: Eurocolt. Springer-Verlag*, 23–37.

Khattab, K., Dubois, J., & Miteran, J. (2009). Cascade Boosting Based Object Detection from High Level Description to Hardware Implementation. *EURASIP Journal of Embedded Systems, 2009 (Design and Architectures for Signal Image Processing)*

Lai, H. C., Marculescu, R., Savvides, M., & Chen, T. (2008). Communication-Aware Face Detection Using Noc Architecture. *in IEEE International Conference on Computer Vision Systems (ICVS)*.

Lienhart, R., Kuranov, A., & Pisarevsky, V. (2003). Empirical analysis of detection cascades of boosted classiers for rapid object detection. *In Proceedings of the 25th DAGM-Symposium*, 297-304.

Lienhart, R., & Maydt, J. (2002). An extended set of haar-like features for rapid object detection. *In Proceedings of the IEEE International Conference on Image Processing*, 900-903.

Moghaddam, B., & Pentland, A. (1997). Probabilistic visual learning for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 696-710.

Papageorgiou, C., Oren, M., & Poggio, T. (1998). A general framework for object detection. *In Proceedings of the IEEE Conference on Computer Vision*, 555-562.

Rowley, H. A., Baluja, S., & Kanade, T. (1998). Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*, 23-38.

Schneiderman, H., & Kanade, T. (2000). A statistical model for 3d object detection applied to faces and cars. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, *1*, 746-751.

Smach, F., Miteran, J., Atri, M., Dubois, J., & Gauthier, J. P. (2007). An FPGA-based accelerator for Fourier Descriptors computing for color object recognition using SVM. *Journal of Real-Time Image Processing (JRTIP), Springer*, *2*, 249-258.

Sung, K.-K., & Poggio, T. (1998). Example-based learning for view-based human face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39-51.

Theocharides, T., Vijaykrishnan, N., & Irwin, M. J. (2006). A parallel architecture for hardware face detection in ISVLSI '06. Washington, DC, USA : IEEE Computer Society, 2006, p. 452. *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures* (p. 452). Washington, DC, USA.

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, *1*, 511-518.

Viola, P., & Jones, M. (2002). Fast and robust classication using asymmetric adaboost and a detector cascade. *In Advances in Neural Information Processing Systems (NIPS), MIT Press*, 1311-1318.

Viola, P., Jones, M., & Snow, D. (2003). Detecting Pedestrians Using Patterns of Motion and Appearance. *IEEE International Conference on Computer Vision (ICCV)* (pp. 734-741).

Wei, Y., Bing, X., & Chareonsak, C. (2004). Fpga implementation of adaboost algorithm for detection of face biometrics. *IEEE InternationalWorkshop on Biomedical Circuits and Systems* (pp. S1/6 - 17-20).

Yang, M.-H. (2004). Recent advances in face detection. Technical report. *IEEE International Conference on Pattern Recognition Tutorial*.

Yang, M.-H., Kriegman, D. J., & Ahuja, N. (2002). Detecting faces in images: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*, 34-58.

Yang, M., Wu, Y., Crenshaw, J., Augustine, B., & Mareachen, R. (2006a). Face detection for automatic exposure control in handheld camera. *IEEE International Conference on Computer Vision Systems, ICVS*, *04*(07), 17.

Yang, M., Wu, Y., Crenshaw, J., Augustine, B., & Mareachen, R. (2006b). Face detection for automatic exposure control in handheld camera. *in IEEE International Conference on Computer Vision Systems (ICVS)*. Toronto, Canada.

New Approaches to Characterization and Recognition of Faces
Edited by Dr. Peter Corcoran

ISBN 978-953-307-515-0
Hard cover, 252 pages
Publisher InTech
Published online 01, August, 2011
Published in print edition August, 2011

As a baby, one of our earliest stimuli is that of human faces. We rapidly learn to identify, characterize and eventually distinguish those who are near and dear to us. We accept face recognition later as an everyday ability. We realize the complexity of the underlying problem only when we attempt to duplicate this skill in a computer vision system. This book is arranged around a number of clustered themes covering different aspects of face recognition. The first section presents an architecture for face recognition based on Hidden Markov Models; it is followed by an article on coding methods. The next section is devoted to 3D methods of face recognition and is followed by a section covering various aspects and techniques in video. Next short section is devoted to the characterization and detection of features in faces. Finally, you can find an article on the human perception of faces and how different neurological or psychological disorders can affect this.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen