Dynamic management of a partial reconfigurable hardware architecture for pedestrian detection in regions of interest

Metzli Ramirez-Martinez, Francisco Sanchez-Fernandez, Philippe Brunet, Sidi Senouci, El-Bay Bourennane

Dynamic management of a partial reconfigurable hardware architecture for pedestrian detection in regions of interest

Metzli Ramirez-Martinez*, Francisco Sanchez-Fernandez*, Philippe Brunet*, Sidi M. Senouci* and El-Bay Bourennane†
*DRIVE EA 1859 Laboratory
Univ. Bourgogne Franche Comté
F58000, Nevers, France
Email: Metzli.Ramirez−Martinez@u−bourgogne.fr
†LE2I Laboratory
Univ. Bourgogne Franche Comté
F21000, Dijon, France
Email: elbey.bourennane@u−bourgogne.fr

***Abstract*—In this paper, we propose a video streaming architecture implementing an IP core design based on Histograms of Oriented Gradients (HOG) algorithm, by using a high-level synthesis workflow. Also, we present a decision algorithm for a dynamically reconfigurable architecture, applied to people detection in video over a Region Of Interest (ROI). Our method dynamically manages the resource consumption in our hardware accelerator. In the test performed with only one reconfigurable module, our method achieves a power reduction up to 7% and a speed of 51 fps, but these results could be improved according to the number of reconfigurable modules. For implementation, we used a Xilinx Zynq 7000 series platform and our experimental results are compared with other software and hardware implementations. Our hardware architecture can be used in a vast number of vision systems, including advanced driver assistance systems (ADAS).**

I. Introduction

Pedestrian detection is considered as one of the most critical elements in several domains such as automotive, industrial automation, and surveillance. However, there is a high demand to perform a low latency and accurate pedestrian detection. Current software detection running on traditional CPUs can reach high levels of accuracy with recent algorithms, but these platforms have several constraints that inhibit a real time performance [1]. Therefore, embedded platforms powered by image processing capabilities are becoming a major actor on different applications. On these platforms, energy, weight, size, and computational latency play an important role, especially for small battery powered devices, as intelligent cars, robots, and unmanned aerial vehicles. These types of systems use image feature extraction to detect and track objects, maintain camera pose, estimate motion, classify and recognize objects. Hence, feature extraction and analysis must be efficient and effective [2]. For enabling a reliable pedestrian detection, a robust visual object recognition based on a feature set for humans is required. Different feature extraction methods have been proposed and evaluated for this purpose, among which HOG is considered as the most efficient and promising for pedestrian detection.

Another aspect that could be considered for improving the performance of feature extraction architectures is the management of resources in the device, wherein the Dynamic Partial Reconfiguration (DPR) could be a useful tool. Specifically, the DPR is the ability to configure a part of a hardware architecture during the running time. It removes the need to fully reconfigure and reestablish logic, and it gives a flexibility that allows implementing a wide variety of algorithms on the embedded reconfigurable devices [3]. In addition, it could be used to efficiently manage the resources in area, energy and even in time.

Other kind of techniques to reduce the computational cost of some object detection algorithms are based on selecting one or more ROI. In this way, the image processing is only limited to specific regions. Some advantages of this kind of methods are that they decrease the amount of data to process, and consequently, the speed of the algorithm increases and the resource utilization decreases. Additionally, if the ROI is carefully computed, it helps to reduce the false positive rate, when the useless information is discarded. In [4]–[6] we can see some examples of this kind of algorithms.

In this paper, we explore a dynamic FPGA implementation of the well know HOG algorithm, it is used to extract features for pedestrian detection in ROI. A key focus has been to explore the use of a new heterogeneous solution for accomplishing feature extraction. We give details of the processes that influence the performance of HOG and how the entire algorithm has been decomposed to allow optimization in order to be deployed on FPGA hardware. The main contribution of this paper is a dynamically reconfigurable HW-SW implementation of a pedestrian detection for a SoC, which offers a dynamic and efficient resource management and reconfigu-

ration versatility but with the software programmability. We create a complete feature extraction architecture including the processing engine, drivers, hardware interfaces, and sensors. The most compute intensive parts of HOG are accelerated based on Xilinx HLS tools using the FPGA. Also, we show the performance advantages of using a reconfigurable heterogeneous platform for HOG features extraction compared to a software-only solution and static hardware implementations.

II. Related work

HOG algorithm is a feature detection algorithm frequently used to detect people on images, Dalal and Triggs proposed the HOG algorithm [7] in 2005. HOG has been investigated by many researchers througth mapping efficiently this algorithm on accelerators, such as FPGAs, digital signal processors (DSPs) and graphics processing units (GPUs). Kadota et al. [8] proposed several modifications to simplify the computation, such as implementing the square root by means of look-up-table (LUT), and approximating the normalization factor to a power two value. Their FPGA architecture is used to achieve real-time requirements. However, these simplifications degrade the accuracy rate for pedestrian detection. They reached a performance of 30 fps VGA (640x480) by using 10 instances of their proposed hardware.

In [9], it is presented an FPGA pipelined architecture for HOG feature extraction. It uses approximation methods to replace the complex operations for adopting the parallel processing, like square root approximation for gradient computation and Newton-Raphson method to compute the inverse square root. This design can be implemented with a lower cost while performing high throughput. Also, logic cells utilization is reduced for achieving a higher operating frequency in an FPGA. Jacobsen et al. [1] present an FPGA implementation of a pedestrian detector, this architecture uses the LUV color space and HOG features to detect pedestrians. These features are extracted from a RGB frame of video and are evaluated at runtime using an attentional cascade of stages. The offline trained boosted detector evaluates each frame using HOG features in the LUV color space, pedestrian targets are identified at 27 scales. It can process VGA video at a frame rate of 30-40 fps.

Other type of platforms used for implementing pedestrian detection are DSPs. Chiang et al. [10] propose a pedestrian detection technique using the HOG algorithm for embedded driver assistance systems. The system is implemented on a DSP resulting in a processing time of 8 fps for VGA image size and reaching a detection accuracy over 92%.

III. HOG feature extraction method

The main idea behind HOG feature extractor is related to the fact that the local appearance of an object can be described by using the local intensity gradient distribution. This distribution has a great robustness because even without precise information of the intensity and its gradient for each pixel, the object shape could be characterized well enough for achieving a high detection rate [7]. This section explains how to extract HOG features, which consists in computing several histograms of orientated gradients in specific pixels structures on an image. The HOG descriptor is computed by dividing the input image in 8x8 pixels that are called cells. Also, a group of cells is called a block, the size of the block is 16x16 pixels, so it consists of four cells. Fig. 1 depicts an idea of cells and blocks used for HOG feature extraction. When computation of the current block is finished, the overlap of the next computation is one cell. In this overview, HOG feature extraction process is divided into the following steps.

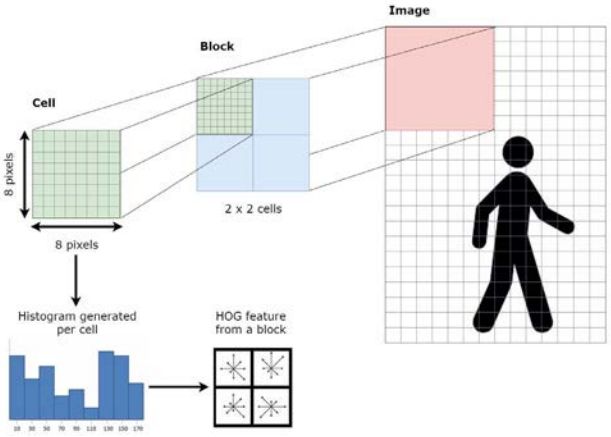

Fig. 1. HOG feature extraction process based on cell and block descriptor generation

A. Two-dimensional gradient computation

The first step consists in computing the magnitude $m(x, y)$ and direction $\theta(x, y)$ for every single pixel located at coordinate (x, y). For computing the gradient, we assume that a pixel is located at coordinate (x, y) and the value of its luminance is denoted by $f(x, y)$. Now, we need to apply 1st order differential coefficients. The gradient filters used for feature extraction are $[-1, 0, 1]$ and $[-1, 0, 1]^T$. Applying these filters to every pixel, we obtain the gradients of the x and y axes, which are denoted by $f_x(x, y)$ and $f_y(x, y)$, respectively, and they are computed by the following equations:

$$f_x(x, y) = f(x + 1, y) - f(x - 1, y) \quad (1)$$

$$f_y(x, y) = f(x, y + 1) - f(x, y - 1) \quad (2)$$

Once gradients for x and y direction are computed, we proceed to calculate the gradient magnitude as follows:

$$m(x, y) = \sqrt{f_x(x, y)^2 + f_y(x, y)^2} \quad (3)$$

The direction is given as:

$$\theta(x, y) = \arctan \frac{f_y(x, y)}{f_x(x, y)} \quad (4)$$

When an RGB image is employed as input, the gradients are computed for every pixel on each color channel separately, the gradient with the largest magnitude would be selected to be used in the next stages in the HOG algorithm.

B. Histogram generation and voting

The next step in the algorithm is the generation of a histogram for groups of 8x8 pixels. We need to generate a histogram of orientations for each cell in the image based on the values $m(x, y)$ and $\theta(x, y)$. The orientation interval is evenly divided over 0°-180°according to the number of orientations for producing the orientation bins needed for computing the histogram. In the original HOG algorithm, it was deduced that using 9 orientations bins, it is possible to reach a very high detection rate in the case of pedestrian detection.

Basically, histogram generation is created using the next steps:

1) Determine the membership of $\theta(x, y)$ by comparing with the direction of all orientation bins.
2) Increment the value of the class related to the direction found in the previous step.
3) Repeat preceding operations for all gradients belonging to a cell.

In order to mitigate aliasing effect, we should update the two nearest bins to each gradient direction. This updating process is accomplished by defining two different weighting rules based on the distance of $\theta(x, y)$ to the edge angle of each bin. The main weighting rule α can be computed as:

$$\alpha = (n + 0.5) - \frac{b * \theta(x, y)}{\pi} \tag{5}$$

Where b is the number of bins and n is the bin to which $\theta(x, y)$ belongs. Now, both values of the two neighboring bins are incremented. m_n indicate a class number where $\theta(x, y)$ belongs and $m_{nearest}$ is the nearest bin to $\theta(x, y)$. The incremented values m_n and $m_{nearest}$ are given by:

$$m_n = (1 - \alpha) * m(x, y) \tag{6}$$

and

$$m_{nearest} = \alpha * m(x, y) \tag{7}$$

Basically, every gradient at pixel (x, y) would contribute to the histogram by a vote which is a function of the gradient magnitude.

C. Block normalization

Finally, a normalization process across blocks is required in HOG feature extraction. Once the histogram of each cell is computed, the histograms of adjacent cells are grouped together to form a spatial block. Every block has a defined number of overlapped adjacent cells. In the original work, every block is comprised of $2x2$ neighboring cells. A histogram normalization is generated by combining all histograms belonging to one block, i.e. a set of four cells. After the large combined histogram is obtained, it is normalized as follows:

$$v_i = \frac{V_i}{\sqrt{\|V\|_2^2 + \epsilon}} \tag{8}$$

Where V is a vector corresponding to a combined histogram of the cells in a block, v is a normalized vector, which is a final HOG feature, and ϵ is a small constant to avoid a zero enumerator.

IV. HOG HARDWARE IMPLEMENTATION

Due to the intensive computations required for extracting HOG feature descriptors and the inadequacy of a software implementation to achieve the real-time requirements, it is imperative to implement a hardware module to fulfill these needs. HOG feature extraction implementation is based on a pipelined architecture, where as in previous architectures, floating point operations and trigonometric computations are replaced by a fixed point representation and Look-Up Tables (LUT) implementations. Also, a pipelined architecture of the implementation contributes to achieve better timing results and throughput, while the resource utilization is considerably low. Implementing an object detection algorithm like HOG implies two main challenging tasks that need to be overcome for reaching a reliable and real-time implementation. The first is the parallelism exploration of the algorithm, allowing calculations being completed simultaneously. Usually, it consists in splitting the algorithm into smaller functional blocks and defining all data dependencies between these blocks for improving the data exchange among them, in order to accelerate feature extraction process.

The second challenge that inhibits a high throughput is related to a high demand of memory access in feature extraction algorithms. Every algorithm contains different stages in the processing, and each stage produces different values that should be stored and accessed in future stages. In different situations, the computed values in a stage could not be sent directly to the next, because the following stage requires performing several computations and obtaining other values before to be able to use data from a previous stage. We observe that in feature extraction algorithms is inevitable to use storage elements, but it is necessary to consider that updating storage elements at the end of one stage and reading back the data in the following step will create a delay in the data flow of the processing blocks, and consequently, throughput will be reduced.

In the previous section, HOG feature extraction is decomposed in three main stages. In the following sections, we explain how they are adapted for an FPGA architecture without decreasing robustness.

A. Gradient extraction

This is the first step for computing a HOG feature, we could observe from Equation 3 that two squared and one square root operations are needed to compute the magnitude of a pixel. However, the cost of both operations is really high on a hardware architecture. Our proposed implementation uses the square root approximation method proposed in [11], it is used for avoiding complex operations that are highly costly in hardware and it could be computed as follows:

$$m(x, y) \approx max((0.875a + 0.5b), a) \tag{9}$$

Where:

$$a = max(f_x(x,y), f_y(x,y)) \tag{10}$$

$$b = min(f_x(x,y), f_y(x,y)) \tag{11}$$

Using an approximation for substituting the magnitude computation allows utilizing shift units instead of multiplier operations that consume more hardware resources. The gradient of the pixel (x, y) is calculated by using its surrounding values. The pixel values are forwarded line by line of the image as a stream. Therefore, row buffers are needed in order to buffer two image rows and three pixels of a third row.

The second step in gradient computation is $\theta(x, y)$ estimation, which was calculated previously and the resulting rounded values were saved in a LUT table. This provides a fast and reliable method for obtaining the bin where $\theta(x, y)$ belongs, additionally, the rounding and the repetitive nature of the tangent functions allows to reduce the amount of stored data.

B. Histogram voting

For generating a histogram of HOG features, we need the values $m(x, y)$ and $\theta(x, y)$, these parameters are used to determine α which is fundamental for a proportional voting of $m(x, y)$ in order to decrease aliasing. However, one disadvantage of using a rounding approximation for computing $\theta(x, y)$ is that α could not be accurately computed. As a consequence, we consider α as a constant with a value of 0.5. Therefore, Equations 6 and 7 becomes a shift operation in hardware as follows:

$$m_n \approx m(x,y) >> 1 \tag{12}$$

$$m_{nearest} \approx m(x,y) >> 1 \tag{13}$$

C. Feature normalization

In this step of the hardware implementation, the resulting feature vector from the previous step is normalized with respect to neighboring cells that form a block. We observe from Equation 8 that it consists in the division of each element of the block by the summation of all histogram elements of the block, to result in a normalized histogram. This equation consumes several resources in hardware because division and square root are high-complexity operations. Subsequently, it will always result in decimals and floating point numbers which are complicated to implement and manage in the FPGA.

In order to improve resource utilization and speed, we simplify the inverse square root by using the Newton-Raphson method as follows:

$$y = \frac{1}{\sqrt{x}} \Rightarrow f(y) = \frac{1}{y^2} - x = 0 \tag{14}$$

$$f'(y) = -\frac{2}{y^3} \tag{15}$$

It is possible to obtain an approximate value $y_n \approx y$ by executing several iterations of the Newton-Raphson equation:

$$y_{n+1} = y_n - \frac{f(y_n)}{f'(y_n)} \Rightarrow y_{n+1} = \frac{y_n(3 - xy_n^2)}{2} \tag{16}$$

However, using a right value could impact directly in the time execution, by reducing the number of iterations performed to find a convergence. Hence, we implement the initial value $I_v = 0x5f3759df$, used in [9] for finding the y_n in only one iteration. The first step for computing the approximation of the inverse square root is:

$$y_{nIEEE754} = (x_{IEEE754} >> 1) - I_v \tag{17}$$

Where $x_{IEEE754}$ and $y_{nIEEE754}$ are the IEEE754 single precision binary floating-point format of values *x* and y_n, respectively. The next step is to transform $y_{nIEEE754}$ into a decimal representation, expressed as y_n. Finally, the approximate value of *y* is computed with Equation 16. The idea is to reduce complex operations and iterations for saving resources and execution time, respectively.

V. Dynamic Partial Reconfiguration

As previously mentioned, our implementation is designed to work together with a method to detect a ROI. A fundamental point for the reconfigurable architecture is selecting a ROI in each frame of the video, in this way, the image processing is limited only to this specific region. The ROI could be selected through different methods as in [4], [5] or [12], particularly, we test using saliency map algorithms [6], [13], [14], which select the outstanding regions of an image at speed of up to 91 fps. This application could be useful in specific environments as film sets, where the background and the outstanding objects are clearly determined.

The general idea is to propose a dynamically reconfigurable system that, taking in consideration the amount of data to process, is capable of selecting the necessary number of HOG modules executed in parallel to achieve a specific speed, without spending extra resources. Due to the area limitations of the current FPGAs, this architecture was tested and implemented with a maximum of two HOG modules in parallel, but in the future, it could be expanded to several modules.

Our implemented dynamic architecture is capable of reconfiguring itself for using one or two HOG modules in parallel depending on the ROI size, which is equivalent to the amount of data to process, and therefore, it is directly related to the system speed. In our implementation, we propose a target speed of 50 fps. Therefore, taking into consideration this target speed and the speed of our HOG module, we selected a threshold of the 50% of the ROI size to change between the possible configurations. It basically means that if the ROI is smaller than 50% of the full image size, only one module HOG will be used and the other module will be set as a black-box, i.e. the second module will be empty, allowing to reduce the area and energy consumption. This configuration is denominated C_1. By the contrary, in the configuration C_2, if the ROI is bigger or equal to 50% of the full image size, the second module will be reconfigured to work as a second HOG module in parallel, this allows to achieve the requirements of speed, but also, the resource consumption increases.

To choose a configuration, we define different measures:

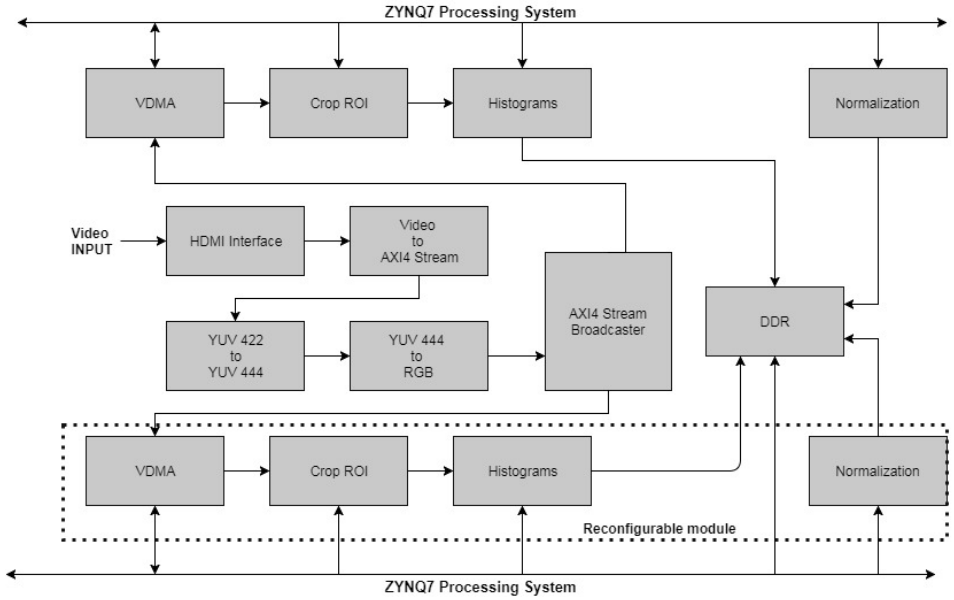

Fig. 2. FPGA architecture with reconfigurable modules

- t_o is the time to process every frame according to the desired speed, e.g. if we want a minimum speed of 50 fps, then $t_o = 0.02s$.
- t_r is the time spent in each reconfiguration (in our implementation $t_r = 0.006s$)
- t_{mi} is the time required by the configuration *i* to process the current ROI. It is computed as:

$$t_{mi} = \frac{size\_roi}{v_i * 100} \tag{18}$$

Where $size\_roi$ is the percentage size of the current ROI and v_i is the speed of the configuration *i*. In our case, we only have two configurations C_1 with v_1= 25 fps and C_2 with v_2=50 fps.

- t_{ac} is the accumulated time. It refers to the gained or lost time in previous frames. As we can see, the processing time depends on the ROI size, for example, if we have a ROI of 70%, and we choose the second configuration, we will be able to process this frame in 0.014 s, faster than the required 0.02 seconds. Therefore, we obtain a theoretical gain of time $t_{ac} = +0.006s$. The t_{ac} could be positive or negative in the case of not achieving the required time.

Additionally, we define the variable DT_i, if the configuration *i* at the time *k* is different from configuration selected at the time *k-1*, then DT_i is determined by:

$$DT_i = t_o - t_r + t_{ac} - t_{mi} \tag{19}$$

Otherwise:

$$DTi = t_o + t_{ac} - t_{mi} \tag{20}$$

Subsequently, we update t_{ac} and choose the configuration according to:

$$Configuration_k = arg\ min(DT_i > 0) \tag{21}$$

$$t_{ac} = min(DTi > 0) \tag{22}$$

In this way, we take into consideration the time available to process the current frame without fail the desired speed. It is important to highlight that this decision algorithm is useful to be applied in a bigger architecture with multiple HOG modules in parallel.

VI. Hardware Architecture

The architecture design for pedestrian detection is implemented in the Zynq Xilinx platform. The main feature of this device is that it combines a dual core ARM Cortex-A9 processor with an FPGA logic fabric. Zynq is connected internally by industry standard Advanced eXtensible Interfaces (AXI), which provide high bandwidth and low latency connections between the processors and the FPGA, resulting in an outstanding communication performance by removing or reducing the overhead in the interaction between the two physically separate devices.

Our design was implemented on a Xilinx Zynq 7020 device. The Video frames are sent from the CPU, they are buffered in off-chip DRAM, and then they are streamed through a processing pipeline. Frame management is performed by a Video Direct Memory Access (VDMA) module. The pipeline performs color space conversion, ROI extraction, HOG feature extraction and normalization by blocks. The HOG feature extraction is parallelized and optimized for using FPGA logic. The entire pipeline is parallelized to extract HOG features in real-time from the video stream. That part of the architecture designed in the FPGA is shown in Fig. 2.

The control of the reconfiguration is performed in the dual-core ARM Cortex-A9 processor of the ZYNQ device,

it reconfigures the programmable logic in the FPGA through the Processor Configuration Access Port (PCAP), which is a 32-bit interface that configures the FPGA at a maximum rate of 100 MHz. In our architecture, we propose to use several modules HOG in parallel, in this way we can increase the speed by processing several frames at the same time.

Our implemented reconfigurable HOG module is composed of four submodules: the VDMA module, the module to crop the ROI, histograms generation by cells, and normalization by blocks. The overall module achieves a speed up to 25 fps, with a full frame of 640x480 pixels, however the speed increments inversely proportional to the ROI size. Finally, a personal computer is utilized for implementing high-level tasks such as ROI computation and the descriptor classification. A brief graphical representation is shown in Fig. 3.

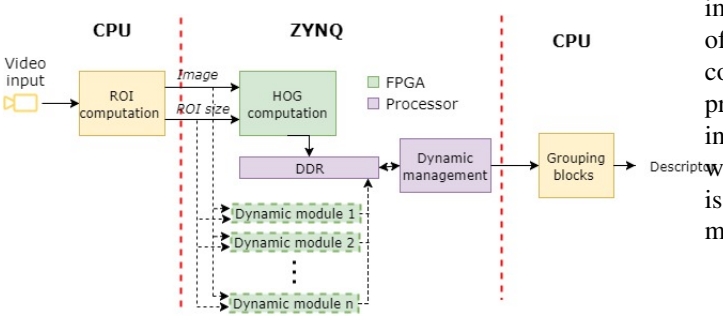

Fig. 3. Dynamic reconfigurable architecture design for pedestrian detection showing the different devices for its implementation

In our architecture, a ROI is computed by frame in a personal computer. If there is more than one ROI they are joined in a global ROI. Subsequently, the input and output are configured by the ARM processor, and the images and ROI data are stored for extracting HOG features. Depending on the desired processing speed and the number of Reconfigurable Modules (RM) implemented, we choose a configuration by using the algorithm explained in V. In our case, we implemented two RM and we propose an objective speed of 50 fps. After that, a series of reconfigurable HOG modules are used to crop the current ROI from the source frame, and to extract features only in the ROI from the incoming video frame. Finally, HOG block features are stored in DDR memory, to later, transmitting to a CPU to concatenate them into a final descriptor for the classification.

VII. Results

We provide a comparison of our approach in different configurations in a database, we tested 6 different video sequences of the Marx Plank Institute computing department [15], [16]. The first analysis is related to the HOG hardware accelerator. As explained above, HOG feature extraction is computationally complex and unsuitable for hardware implementation, therefore, all floating point operations were replaced by fixed-point and some approximate techniques were adopted to reduce implementation complexity and to improve feature extraction speed. However, performed approximations could decrease HOG accuracy. In Table I, we show the number of True Positives (TP), True Negatives (TN), and the accuracy of our hardware accelerator in the FPGA in comparison to a HOG software-based implementation. We observe that the approximations slightly decrease the performance of hardware accelerator, but it is still acceptable for many pedestrian detection applications.

TABLE I
Accuracy comparison between SW and HW implementation

Method	*TP*	*TN*	*Accuracy*
Standard HOG algorithm	94.8%	87.5%	91.1%
Approximation HOG	95.4%	83.3%	89.4%
Accuracy difference:			1.7%

In table II, we observe hardware resources consumed for implementing our architecture. We present resource utilization of our HOG feature extraction module and both reconfigurable configurations, which includes all the necessary blocks to process the video streaming. In [9] and [17] we can find some interesting comparison tables of other FPGA architectures, which let us see that our implementation module utilization is competitive and in many cases it outperforms other implementations.

TABLE II
Total architecture utilization (Includes reconfigurable modules and VDMA)

Resources	*C1 (1 module)*	*C2 (2 modules)*	*HOG module (individually)*
LUT	21964	34665	13250
LUTRAM	918	1139	328
FLIP-FLOP	20237	29587	9783
BRAM	26	51	25
DSP	47	90	43

Our proposed decision algorithm is designed to work with multiple modules HOG in parallel, but due to limited resources of the FPGA, we only tested the implementation with two HOG modules. Both configurations compute the histograms and normalization, but the first configuration uses only one module, and the second configuration uses two modules in parallel for computing the same calculations. It means that in the second configuration, two frames are processed in parallel, therefore, the speed and resources increase.

TABLE III
Usage statistics of reconfigurable reconfiguration for feature extraction.

Method	*Power*	*Frames not processed in time*	*Average speed*
Only configuration C_1	2.176 W	86%	37 fps
Only configuration C_2	2.485 W	0%	75 fps
Our method	2.296 W	0.5%	51 fps

In our implementation with two HOG modules in parallel (C_2), we achieve a minimum speed of 50 fps in the worst case (when the ROI is the 100% of the image size) and our goal is to keep that speed using the least amount of resources. However,

the ROI sizes vary between 97% and 28% of image size, and on average, only 8% of the images have a ROI smaller than 50%, but thanks to the consideration of the accumulated time in the decision algorithm, we achieve to select the smallest configuration (C_1) in the 52% of the frames, reducing the energy consumption without reducing the minimum processing speed of 50 fps.

Table III shows different approaches and their results in power, the frames percentage that surpasses the objective time assigned to every frame (t_o), and the average speed. The different methods compared are:

- Only configuration C_1: All the frames are processed using the configuration with only one HOG module.
- Only configuration C_2: All the frames are processed using the configuration with two HOG module in parallel.
- Our method: We implement the proposed decision algorithm explained in section V, to switch between configurations.

According to the table III, C_1 does not achieve the desired speed, and C_2, the fastest configuration, process the video at 75 fps, which far surpass the target, but the power consumption and resource utilization are the highest. On the contrary, our method slightly surpasses the objective speed with 51 fps on average, and almost 100% of the time, we can wait for an output in the period t_o. Additionally, the power consumption is close to the smallest and the area utilization is substantially reduced, leaving space for other operations. As we can see, the difference between C_1 and C_2 is 0.309 W, which corresponds to a single HOG module, but in multi-module implementation, this difference in energy increases considerably. Finally, we highlight the fact that despite the power difference between the C_2 implementation and our method is only 7%, the amount of saved energy is meaningful when we consider that these kinds of system are designed to work hours or even whole days without interruption.

VIII. Conclusions

In this article, we presented a dynamic approach to efficiently manage speed and resource utilization for people detection task. We proposed a multi-module system capable of being reconfigured according to the size of a ROI previously computed. In this way, we can activate or deactivate a different number of modules depending on the amount of data to process. The algorithm for the reconfiguration control could be applied to a different number of modules, depending on the available resources. Also, it is flexible enough to be adapted to different detection methods based on tiles, and different ways of selecting the ROI. In our case, we perform the test with two different configurations. We set a target speed of 50 fps, which was met without problem. The system used the cheapest configuration 52% of the time, even if the 91.5% of the frames tested has a ROI bigger than 50% of the full frame size. This could be traduced as an efficient resource management, and it is the result of considering the accumulated time in previous frames in the decision algorithm of the dynamic reconfiguration. Additionally, our implementation of the HOG algorithm in the hardware architecture is quite accurate, it only has a reduction accuracy up to 1.7%.

Furthermore, we describe all the architecture design methodology, including all the necessary elements to implement the full system, like the peripherals, memories and the VDMA module, which is a fundamental piece for many embedded video applications.

References

[1] M. Jacobsen, Z. Cai, and N. Vasconcelos, "FPGA implementation of HOG based pedestrian detector," in *2015 International SoC Design Conference (ISOCC)*. IEEE, nov 2015, pp. 191–192.

[2] G. van der Wal, D. Zhang, I. Kandaswamy, J. Marakowitz, K. Kaighn, J. Zhang, and S. Chai, "FPGA acceleration for feature based processing applications," in *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, jun 2015, pp. 42–47.

[3] S. G. S. B, Y.-h. E. Yang, and A. Panangadan, *Applied Reconfigurable Computing*, ser. Lecture Notes in Computer Science, V. Bonato, C. Bouganis, and M. Gorgon, Eds. Cham: Springer International Publishing, 2016, vol. 9625.

[4] S. Lee, K. Kim, J.-Y. Kim, M. Kim, and H.-J. Yoo, "Familiarity based unified visual attention model for fast and robust object recognition," *Pattern Recognition*, vol. 43, no. 3, pp. 1116–1128, mar 2010.

[5] A. Oliva, A. Torralba, M. Castelhano, and J. Henderson, "Top-down control of visual attention in object detection," in *Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429)*, vol. 1, no. x. IEEE, 2003, pp. I–253–6.

[6] M.-M. Cheng, N. J. Mitra, X. Huang, P. H. S. Torr, and S.-M. Hu, "Global Contrast Based Salient Region Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 3, pp. 569–582, mar 2015.

[7] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1. IEEE, 2005, pp. 886–893.

[8] R. Kadota, H. Sugano, M. Hiromoto, H. Ochi, R. Miyamoto, and Y. Nakamura, "Hardware architecture for HOG feature extraction," in *IIH-MSP 2009 - 2009 5th International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, no. 3. IEEE, sep 2009, pp. 1330–1333.

[9] Pei-Yin Chen, Chien-Chuan Huang, Chih-Yuan Lien, and Yu-Hsien Tsai, "An Efficient Hardware Implementation of HOG Feature Extraction for Human Detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 2, pp. 656–662, apr 2014.

[10] Chuan-Yen Chiang, Yen-Lin Chen, Kun-Cing Ke, and Shyan-Ming Yuan, "Real-time pedestrian detection technique for embedded driver assistance systems," in *2015 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, jan 2015, pp. 206–207.

[11] D. D. Gajski, in *Principles of Digital Design*, Upper Saddle River, NJ. USA: Prentice Hall, 1997.

[12] S. Goferman, L. Zelnik-Manor, and A. Tal, "Context-Aware Saliency Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 10, pp. 1915–1926, oct 2012.

[13] L. Itti, C. Koch, and E. Niebur, "A Model of Saliency-Based Visual Attention for Rapid Scene Analysis," *IEEE Transactions on Automatic Control*, vol. 20, no. 11, pp. 1254–1259, 1998.

[14] F. Perazzi, P. Krahenbuhl, Y. Pritch, and A. Hornung, "Saliency filters: Contrast based filtering for salient region detection," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, jun 2012, pp. 733–740.

[15] M. Andriluka, S. Roth, and B. Schiele, "Pictorial structures revisited: People detection and articulated pose estimation," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE. IEEE, jun 2009, pp. 1014–1021.

[16] C. Wojek, S. Walk, and B. Schiele, "Multi-cue onboard pedestrian detection," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE. IEEE, jun 2009, pp. 794–801.

[17] K. Mizuno, Y. Terachi, K. Takagi, S. Izumi, H. Kawaguchi, and M. Yoshimoto, "Architectural study of HOG feature extraction processor for real-time object detection," in *IEEE Workshop on Signal Processing Systems, SiPS: Design and Implementation*, 2012.