Cascaded Stripe Memory Engines for Multi-Scale Object Detection in FPGA

Petr Musil, Roman Juránek, Martin Musil, and Pavel Zemčík

***Abstract*— Object detection in embedded systems is important for many contemporary applications that involve vision and scene analysis. In this paper, we propose a novel architecture for object detection implemented in FPGA, based on the *Stripe Memory Engine* (SME), and point out shortcomings of existing architectures. SME processes a stream of image data so that it stores a narrow stripe of the input image and its scaled versions and uses a detector unit which is efficiently pipelined across multiple image positions within the SME. We show how to process images with up to 4K resolution at high frame rates using cascades of SMEs. As a detector algorithm, the SMEs use boosted soft cascade with simple image features that require only pixel comparisons and look-up tables; therefore, they are well suitable for hardware implemenation. We describe the components of our architecture and compare it to several published works in several configurations. As an example, we implemented face detection and license plate detection applications that work with HD images (1280 × 720 pixels) running at over 60 frames/s on Xilinx Zynq platform. We analyzed their power consumption, evaluated the accuracy of our detectors, and compared them to *Haar Cascades* from OpenCV that are often used by other authors. We show that our detectors offer better accuracy as well as performance at lower power consumption.**

***Index Terms*— Object detections, accelerator architectures, field programmable gate arrays.**

I. Introduction

OBJECT detection in embedded systems is an important task that many applications of computer vision and scene analysis benefit from. Industrial quality control systems address various markers, traffic monitoring uses detection of cars and license plates, biometric systems detect faces and facial features, driver assistance systems detect cars and pedestrians. The detection is especially important in applications that directly rely on it, such as recognition or tracking, and in these applications, the speed, accuracy, power consumption, and/or robustness of detection matters most. In this paper, we address object detection implemented in embedded hardware. We focus on boosted detectors which analyze subwindows of an input image by a classifier composed from weak classifiers based on simple image features such as Haar [1] or Local Binary Patterns (LBP) [2]. Multi-scale detection is solved by scaling and processing of the input image in multiple resolutions – image pyramid. Embedded object detectors are often implemented directly in software using libraries such as OpenCV [3]. While this approach is easy and straightforward, it often is quite slow as detection is computationally demanding task and embedded processors tend to be simpler and slower than desktop CPUs. Another approach is to implement a custom detection algorithm exploiting various acceleration resources of the target platform – CPU [4], GPU [5] or Field Programmable Gate Array (FPGA) [2], [6]–[11] units. This is advantageous in many areas where the deployment of standard PC-based or embedded software solution is not possible, e.g. because of resource consumption, physical dimensions, industrial or military conditions, etc.

The object detection in embedded devices typically belongs to one of the three detection method categories. **1/** AdaBoost-based detectors – cascades of boosted classifiers [1] or soft cascades [12]. They typically use Haar image features [6], [9], [10], [13], or LBP [2]. **2/** Support Vector Machines (SVM) with Histograms of Oriented Gradient features (HOG) [7], [8], [14]–[16]; and **3/** Other methods implementing detection with background subtraction [17], keypoints [18], neural networks [19], or custom detection algorithms [20]. Most works, including thin one, belong to the first category, we give the detailed review of them in Section III.

In this paper, we propose a simple and easy to use building block for FPGA that solves the object detection using state of the art boosted soft cascade classifier. We focused on implementation of the detection algorithm in the FPGA that efficiently utilizes the hardware resources and provides high performance. To produce classifiers for our hardware we used an existing, previously published algorithm [21]. The solution is multi-scale so it can detect objects of wide range of sizes.

It is suitable for various industrial applications, such as license plate detection, face detection, etc. The classifiers we use are especially suitable for hardware implementation since they are based only on pixel comparisons, look-up tables and integer-only calculations. Our architecture is extensively configurable, and it offers high image throughput even with high resolution inputs. In our main applications, which are detection of faces and license plates, we use processing

Manuscript received January 5, 2018; revised August 20, 2018 and November 19, 2018; accepted November 27, 2018. Date of publication December 12, 2018; date of current version January 7, 2020. This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) Project IT4Innovations Excellence in Science–LQ1602. This paper was recommended by Associate Editor M. Paul. *(Corresponding author: Petr Musil.)*

The authors are with the Centre of Excellence IT4Innovations, Faculty of Information Technology, Brno University of Technology, 601 90 Brno, Czech Republic (e-mail: imusilpetr@fit.vutbr.cz; ijuranek@fit.vutbr.cz; imusil@fit.vutbr.cz; zemcik@fit.vutbr.cz).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2018.2886476

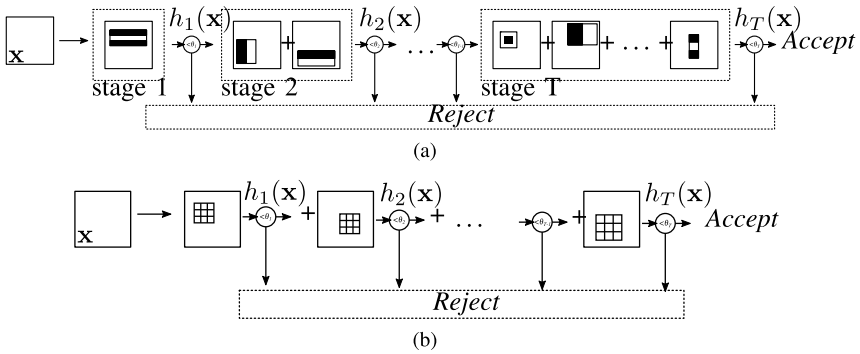

Fig. 1. Comparison of Haar Cascade detector model (a) and Soft Cascade (b) that we use in our architecture. The main difference is that Soft Cascade does not contain *stages* and accumulates the response throughout the classifier. Another difference is that in Soft Cascade case the evaluation of the response can be terminated after every weak classifier.

of HD images (1280 × 720 pixels), but we also present a configurations for processing images with resolutions up to 4K (UHD, 3840 × 2160 pixels). IP Core for face detection and other resources are available online. Our contributions are specifically:

- Advanced memory architecture for image representation in block RAM (BRAM) which allows for simple and fast data random access suitable for fast feature extraction.
- Cascading of detector blocks which allows for increasing the input image resolution and the total performance.
- Multi-scale object detection directly in FPGA enabled by cascading od SMEs, without using external components
- Re-usable detector block that can be easily incorporated into other architectures using standard interfaces.
- Efficient streaming and pipelining and advanced control that fully utilizes the engine resources.

The paper is organized as follows. We start with a brief description of sliding window-based object detection in Section II, where we introduce the framework common to many object detection methods. And we explain the difference between *cascade* and *soft cascade* classifiers. We continue with a review of existing works on object detection with boosted detectors in FPGA in Section III. Section IV contains analysis of existing solutions and describes improvements of the architecture proposed in this paper. In Section V, we describe the soft cascade classifier model that we use in our architecture. We also briefly describe the classifier training algorithm. The proposed architecture is detailed in Section VI, where we describe the components of the detector. In Section VII, we compare the accuracy of our detectors to detectors from OpenCV, that are widely used by other authors, and compare our architecture to others works. Wa also analyze power consumntion of our architecture.

Finally, in Section VIII, we present remarks on the performance of the presented architecture.

II. Detector Model

Let us first describe a framework for object detection that sliding window-based methods have in common [1], [14], [21], [22]. We assume the input image $\mathbf{I}$ to be a grayscale raster and a classifier $H(\mathbf{x})$ a function that accepts or rejects the image patch $\mathbf{x}$ and returns a confidence estimation.

A. Detection on a Single Image

The detection function $D(\mathbf{I}, H, a)$ classifies every fixed-size patch of the input image $\mathbf{I}$ by the classifier H. A patch is defined by its location (m, n). Its size (u, v) is fixed, defined during classifier training stage. We use $\mathbf{x} = \mathbf{I}(m, n, u, v)$ for patch extraction from the location (m, n). The detection function (1) returns the set of locations accepted by the classifier and scaled by factor a, and the classification confidence.

$$D(\mathbf{I}, H, a) \in \{([m, n, u, v] \cdot a, H(\mathbf{x}))\} \tag{1}$$

B. Multi-Scale Detection

The detection process is illustrated in Figure 2. From the input image $\mathbf{I}$, a pyramidal structure $\mathcal{I}$ (see Equation (2)) with k scaled versions is created, such that $\mathbf{I}_j$ is $\mathbf{I}_{j-1}$ downscaled by factor $S < 1$. In our architecture, we use $S = \frac{5}{6}$, which results approximately in a pyramidal representation with 4 scales per octave.

$$\mathcal{I} = \{\mathbf{I}_0, \mathbf{I}_1, \ldots \mathbf{I}_{k-1}\} \tag{2}$$

The scale of j-th image in $\mathcal{I}$ can be retrieved as $s_j = S^j$; therefore, $\mathbf{I}_0$ corresponds to the original image. The result of the detection on $\mathcal{I}$, see Equation (3), is simply union of the results on individual images.

$$D(\mathcal{I}, H, S) = \bigcup_j D(\mathbf{I}_j, H, S^j) \tag{3}$$

The set $D(\mathcal{I}, H, S)$ is then processed by a non-maxima suppression (NMS) algorithm to suppress nearby detections and produce the final results for the image. We use a simple, overlap-based, NMS algorithm [22] which finds clusters of overlapping detections and keeps only the strongest detection from each cluster. However, other algorithms, such as mean-shift [14], could be used as well.

The main work in the detection process is done by the classifier H which scores the individual image windows. The classifier cascade introduced by Viola and Jones [1] (or sometimes called Haar cascade), is a widely used model, see Figure 1a. The classifier cascade analyzes the input image patch $\mathbf{x}$ by a sequence of progressively more complex *stages*

TABLE I

OVERVIEW OF STATE-OF-THE-ART ARCHITECTURES. IN *Length* COLUMN WE REPORT THE NUMBER OF FEATURES AND NUMBER OF STAGES FOR CASCADES IN BRACKETS. *(SELF-ORGANIZING MAP NEURAL NETWORK)

	Year	Image size [px]	Method	Feature type	Window size	Length	Scales	Object	Description
Lai [23]	2007	640×480	Cascade	Haar	20×20	52 (3)	15	Faces	Parallel calculation of feature responses
Zemcik [24]	2007	640×480	WaldBoost	LBP/LRD	31×31	80	-	Faces	Microprogrammable engine for feature extraction
Granat [13]	2007	256×256	AdaBoost	Haar	24×24	184	-	Faces	FPGA coprocessor for DSP
Cho [6]	2008	640×480	Cascade	Haar	20×20	2 135 (22)	18	Faces	Up to 3 features per clock cycle
Hiromoto [25]	2008	640×480	Cascade	Haar	24×24	2 913 (25)	18	Faces	Parallel calculation of feature responses
Martelli [8]	2011	640×480	SVM	Covariance	128×64	–	–	Peds.	Features extracted from 5x5 blocks
Kyrkou [10]	2011	320×240	Cascade	Haar	24×24	2 913 (25)	8	Faces	Combination of detector upscale and image downscale
Huang [9]	2011	320×240	Cascade	Haar	20×20	2 135 (22)	12	Faces	Scalable performance/resources tradeoff
Brouss [26]	2012	320×240	Cascade	Haar	20×20	2 135 (22)	15	Faces	Evaluation of multiple position in parallel
Jin [27]	2012	640×480	Cascade	LBP	20×20	250 (5)	16	Faces	Synthetic classifier
Kadlcek [11]	2013	1024×1024	WaldBoost	LBP	24×24	20	–	Faces	Synthetic classifier
Zemcik [2]	2013	640×480	WaldBoost	LBP/LRD	24×24	128	4	Faces	Programmable engine evaluating multiple positions
Yazawa [28]	2015	640×480	AdaBoost	HOG	40×80	–	5	Peds.,Cars	Featurtes extracted form blocks
Ma [29]	2015	1620×1200	SVM	HOG	64×128	3 708	34	Peds.	Featurtes extracted form blocks
Said [30]	2016	640×480	SVM	HOG	64×128	2 205	–	Peds.	Featurtes extracted form blocks
Kyrkou [15]	2016	800×640	SVM Cascade	LBP	20×20	1 062	18	Faces	Neural net pre-filter
Xu [31]	2016	320×240	Cascade	Haar	20×20	2 135 (22)	8	Faces	Features in FPGA, detector in ARM
Bilal [32]	2017	640×480	SVM	HOG	64×128	–	8	Peds.	Featurtes extracted form blocks
Yang [33]	2017	256×256	AdaBoost,SOM*	LBP	16×24	600	–	Faces	Heterogeneous parallel processor
Proposed	**2017**	**up to 4K**	**WaldBoost**	**LBP/LRD**	**up to N×27**	**1024**	**as required**	**Faces, lic. plates**	Programmable engine with parallel processing of windows

Fig. 2. **(top)** The detection process on pyramidal image representation $\mathcal{I}$. The detection window of size $u \times v$ (yellow), is used for classification of every position m, n in image (example in red). The result of detection on each image is a set of locations $D(\mathbf{I}, s)$ accepted by the classifier (green). **(bottom)** The final detection result after non-maxima suppression.

composed from *weak classifiers* based on simple image *features*. After evaluating of a stage, the image patch can be either rejected (classified as background) or passed to the subsequent stage. The soft cascade, shown in Figure 1b, is not explicitly divided into stages and the rejection decision is made after evaluating each weak classifier. In this work we use the soft cascade model based on Local Binary Patterns (LBP) or Local Rank Differences (LRD) features and we describe this model in detail in Section V.

III. RELATED WORK

Current cutting edge object detection algorithms are based on deep learning and convolutional neural networks (CNN). Generally, they achieve high detection accuracy in comparison to linear classifiers (such as Adaboost or SVM) [34], [35]. On the other hand, the computation of convolutional layers is very demanding; the number of operations required for evaluation is several orders of magnitude higher compared to linear classifiers. Furthermore, the neural networks usually require large amount of intermediate results, increasing memory requirements during inference. Another issue is the number of network parameters which can easily reach many milions. Furthermore, the memory requirements of CNN-based detectors are prohibitive for FPGA implementation. Current state-of-the-art FPGA architectures is that why can process only small images [36] and they are very slow [37],or they must use clusters of very large and expensive FPGAs [38]. For these reasons, linear classifiers are still favorable for implementation in FPGAs and embedded devices in general especially when processing of large images is required.

Table I summarizes important works in the field of embedded object detection from last ten years. Here we analyze the approaches the authors used.

Lai *et al.* [23] proposed a parallel hardware architecture based on Haar cascades. They achieved a detection speed up to 143 frames per second (FPS) at VGA resolution. Due to high demands on FPGA resources they limited the cascade to only first three stages (52 features), which led to low detection accuracy. Their implementation is therefore suitable as a pre-processing unit rather than full object detector. Cho *et al.* [6] implemented a Haar cascade-based face detection algorithm. They implemented various versions with one or three parallel classifiers to accelerate the processing speed. The disadvantage is high memory demand to perform multiscale detection on a pyramid of integral images.

Huang and Vahid [9] developed a method to generate a Haar feature-based object detectors. They aimed at automatic generation of detectors with a required precision for FPGAs of various sizes. This approach allowed to reduce resource requirements of integral image memory and hardware complexity against universal implementation of detector.

Brousseau and Rose [26] improved Haar cascade-based detector in FPGA by preloading of neighboring pixels, allowing parallel evaluation of classifiers in adjacent scanning windows. They also proposed a very complex evaluation control mechanism, allowing to rearrange execution of classifiers to coalesce the memory accesses.

Zemcik *et al.* [2], proposed an approach based on WaldBoost detection algorithm with LBP or LRD features. This approach implements stripe memory with block readout and image scaling but it is limited by fixed performace and by small image resolution, if the multi-scale detection is required.

Several authors proposed detection engines based on massive parallel execution of large number of features, increasing overall performance at the expense of the resource consumption. Jin *et al.* [27] proposed a design of fully pipelined classifier for high-speed face detection with LBP cascades. The features in each stage are executed in parallel. Kadlcek and Fucik [11] proposed an automatic classifier synthesis for the FPGA. Their method generates a fast image preprocessing unit with LBP features, processing complete detection window per clock cycle. High expense of FPGA resources allows for implementation of only limited number of weak classifiers.

Most of the works implement AdaBoost Cascade of classifiers with Haar features for face detection [6], [9], [23]. But, for example Kyrkou *et al.* [7] detected traffic signs and cars. Some authors solve pedestrian detection with SVM [28]–[30], [32].

IV. Design Choices

In this section, we analyze significant works from the point of efficient hardware implementation and we summarize the outlines for the design of our architecture.

When it comes to hardware implementation, Haar features are not a good choice for several reasons. Haar feature is evaluated as a convolution of image and a mask. Each feature in the detector can cover a different and potentially large number of pixels, which means many memory accesses. Without using an integral image, this cannot be implemented to run in constant time, which is an important feature for pipelining in hardware. Using of integral image increases memory requirements as each pixel requires higher bit depth [6], [7], [9], [23]. When using integral image, each feature can be evaluated by referencing from 6 to 9 pixels, depending on the shape of the feature [6]. Reading these values from BRAM, unfortunately, means non-uniform memory access which cannot be executed in a single clock cycle; therefore, most of the works implement the sliding window as a register array with FIFO line buffers stored in BRAM. This allows for parallel access of pixels in the window and evaluation of multiple features in parallel. However, this also leads into a huge multiplexer network (20×20 search window requires 400:1 multiplexer [6], [23]), that occupies many resources in FPGA. The resource consumption increases dramatically with the size of the detection window and thus such architectures are constrained to use only small and fixed window sizes to save resources. Huang and Vahid [9] solves this drawback by limiting feature positions and simplifying the multiplexers.

Zemcik *et al.* [2] substitutes Haar features with LBP which replaces shift registers and delay lines by a set of BRAM memory blocks with organization that allowed for the multiplexing to be replaced by a simple block addressing technique. This approach has another advantage in pipelining of feature evaluation. It allows simultaneous processing of multiple image windows in the stream and thus full utilization of the pipeline, which is not possible with standard scanning window approach [9], [13]. In general, the hardware detectors based on LBP features [11], [15], [27] achieves higher performance than Haar feature based detectors which is summarized in section VII.

Multi-scale detection is, in most cases, solved by storing the input image in RAM and scaling by an algorithm or circuitry independent on the detection unit [9]. Downscaled images are then passed to the detector from RAM one after another. Brouss and Rose [26] uses resolution so small that the image fits BRAMs in the FPGA. Kyrkou *et al.* [7] combines image downscaling to half resolution and upscaling the detector window. Scaled version of the image is stored in BRAM. Granat *et al.* [13] scales the image features in the classifier and addresses the integral image at its original scale. Zemcik *et al.* [2] scales image on the fly and stores only a narrow image stripe in BRAM. Some works [8], [11], [30] do not solve multi-scale detection and detects objects of a fixed size; therefore, their architectures are more simple and exhibits apparently higher performance.

As a basic building block in our architecture, we use an improved architecture by Zemcik *et al.* [2]. Specifically we improved the performance of pipelining, image scaling algorithm, the bit depth of the image and we extended it with cascading capabilites, described below. Our architecture differs from the others in several aspects. We use *soft cascade* instead of cascade of classifiers(see in Section II). Soft cascade is usually more efficient in terms of the number of extracted features [22]. We use features that do not need integral image and that can be evaluated directly from the input image – LBP and LRD [39].

In our approach, the sliding window is not stored in FPGA registers. Instead, *Stripe Memory Engine* (SME) is used to store a narrow stripe of the input image in BRAM, see in VI-A. The stripe must be higher than the of classifier window (we use classifiers with height up to 24 pixels and stripe height is 32 pixels). In the classifier window, we limit geometric size of the features to 6×6 pixels which allows uniform reading of a fixed size pixel blocks from SME in one clock cycle. Juránek *et al.* [40] shows that limitation of feature block size does not have adverse effects on detector accuracy. Image is represented on 8 bits per pixel which saves resources compared to integral image where even 20 and more bits per pixel need to be used [7], [23]. The detector size is limited only by the height of the detection window but not by the width, which can be of virtually any size. We also do not use RAM to store the input image; instead, the image is scanned as it comes from the source and its scaled versions are generated on the fly (see in VI-D). Many image scales are stored in the same SME. The stripe memory, due to its organization, allows the evaluation of multiple scanning windows simultaneously

and enables efficient pipelining and scheduling of the detector evaluation process. Moreover, dual-port BRAM allows us to implement two pipelines and therefore up to two features can be extracted in a clock cycle.

Another contribution of this work is that our detection engine can be cascaded in order to increase the performance and the image resolution using *Stripe Memory Cascades* (see in VI-F). It is basically a chain of SMEs where one SME sends the image data to the subsequent one. The number of instances is only limited by available resources. In practical setup, one instance can hold few high-resolution image scales and the other the rest low-resolution scales; therefore, the maximum image resolution is bigger compared to one instance solution. Moreover, both instances run in parallel and thersefore the performance is also increased. The number of instances in the cascade is limited only by available resources.

All of these differences – detector based on simple image features, image representation in SME, cascading and efficient pipelining – contribute to low resource requirements and overall performance of the proposed architecture.

V. Classifier Model

The main part of the detection is the evaluation of the classifier $H(\mathbf{x})$ on image patches. It consists of the feature extraction and the classifier response accumulation, which we describe in the following text.

A. Feature Extraction

Given an image patch $\mathbf{x}$, a feature extraction is a function $y = f(\mathbf{x}, \pi), y \in \mathbb{N}$ which extracts a value from $\mathbf{x}$ based on the parameters π. As a feature extraction function, we use Local Binary Patterns (LBP) with

$$\pi = (x, y, w, h)$$

or Local Rank Differences (LRD) [39] with

$$\pi = (x, y, w, h, a, b)$$

where x,y,w,h define the feature position and the size in the patch $\mathbf{x}$ and a,b are indices of two distinct cells in the LRD case.

The feature response $f(\mathbf{x}, \pi)$ is evaluated from values of 3×3 cells whose positions and sizes are defined by the parameters π. The cell values $\mathbf{c} = C(\mathbf{x}, x, y, w, h)$ are obtained as a sum of pixel values in the respective cell. The two feature types we use, LBP and LRD, differ in how the values $\mathbf{c}$ are processed.

B. Local Binary Patterns (LBP)

In general, LBP is based on comparison of pixels from a circular neighborhood to the central pixel and generating binary code [41], forming the feature output. Extended versions attempt to reduce the number of possible output values by rotating the resulting bit pattern or by restriction of the number of 0-1 and 1-0 transitions in the code [42].

In this work, we use a simplistic variant of LBP which takes 3×3 cell values and generate 8 bit code form comparison of the central cell to the border cells. Mathematically, the calculation can be written as Equation (4) where $>$ operator compares all values of a vector to a scalar value, resulting in binary vector. Weights $\mathbf{w}$ correspond to powers of two

$$w = [1, 2, 4, 8, 0, 16, 32, 64, 128],$$

so the dot product effectively sets the bits in the result. The zero weight, $\mathbf{w}_5$, corresponds to the central cell $\mathbf{c}_5$ which is used as a basis for the comparison. The range of the resulting values of $\mathrm{lbp}(\mathbf{c})$ is $[0; 255]$.

$$\mathrm{lbp}(\mathbf{c}) = (\mathbf{c} > \mathbf{c}_5)\mathbf{w}^\top \tag{4}$$

Equation (5) shows how the feature value is calculated, given an image patch $\mathbf{x}$ and parameters π.

$$f(\mathbf{x}, \pi) = \mathrm{lbp}(C(\mathbf{x}, x, y, w, h)) \tag{5}$$

C. Local Rank Differences (LRD)

Features based on local ranks proved to be successful in object detection tasks [39]. LRD uses scheme similar to LBP – processing of 9 values in 3×3 cells. It calculates the ranks of two distinct cells and outputs their difference. Mathematically, the function can be described as Equation (6), where a and b are indices of two distinct cells. The resulting value of the $\mathrm{lrd}(\mathbf{c}, a, b)$ values is in $[-8; +8]$ range.

$$\mathrm{lrd}(\mathbf{c}, a, b) = \sum \mathbf{c} > \mathbf{c}_a - \sum \mathbf{c} > \mathbf{c}_b \tag{6}$$

Equation (7) shows how the feature value is calculated, given an image patch $\mathbf{x}$ and parameters π.

$$f(\mathbf{x}, \pi) = \mathrm{lrd}(C(\mathbf{x}, x, y, w, h), a, b) \tag{7}$$

D. The Classifier

A classifier H is represented as a sequence of T weak classification functions

$$h_i = (\pi_i, \theta_i, \mathbf{a}_i), \quad i \in 1, 2, \ldots T \tag{8}$$

where π are parameters for feature extraction, θ rejection threshold, and $\mathbf{a}$ look-up tables with response values. Given an image patch $\mathbf{x}$, the response of the classifier of length t, Equation (9), is a sum of predictions produced by the individual weak classifiers.

$$H_t(\mathbf{x}) = \sum_{i=1}^{t} \mathbf{a}_i(f_i(\mathbf{x}, \pi_i)) \tag{9}$$

The sample $\mathbf{x}$ can be rejected (classified as background) after evaluating $k < T$ weak classifiers when $H_k(\mathbf{x}) < \theta_k$. And it is classified as detected object only if all T weak classifiers were evaluated. $H_T(\mathbf{x})$ is then used as classification confidence. The evaluation is summarized in Algorithm 1.

The number of features evaluated on a sample is, therefore, not fixed as each image patch can be rejected by different number of weak classifiers. The number of weak classifiers varies depending on the image patch content. We can statistically evaluate the *average number* of weak classifiers required for classification of a patch – $\bar{t}$. The value can be viewed as

Algorithm 1 Evaluation of Classifier H on the Sample $\mathbf{x}$

```
1: procedure H(x)
2:     h = 0
3:     for t ← 1, T do
4:         (x, y, w, h, a, b) = π_t          ▷ Decode parameters
5:         c = C(x, x, y, w, h)              ▷ Extract cells
6:         g = lrd(c, a, b)                  ▷ Or g = lbp(c)
7:         H = H + a_t(g)                    ▷ Accum. the confidence
8:         if H < θ_t then
9:             return ('reject', 0)          ▷ Reject x
10:    return ('accept', H)                  ▷ Accept x
```

computational effort required for classifier evaluation. It can be calculated on a dataset using (10) by counting the number of evaluated weak classifiers W and classified image patches P.

$$\bar{t} = \frac{W}{P} \tag{10}$$

The value largely depends on the task and training data. Usual values values are $2 < \bar{t} < 5$ [40]. Lower values means faster detectors. For illustration purposes, later in this paper, we use $\bar{t} = 2.5$ which is a realistic value e.g. for face detection [40]. The value is especially important since it directly influences the performance of the proposed architecture, see Section VI.

In practise, the classifier of length T with LBP features is represented by three matrices $\mathbf{F}$, $\mathbf{A}$ and $\mathbf{T}$, where F is $4 \times T$ matrix with feature extraction parameters $\pi_t = (x, y, w, h)$, $\mathbf{A}$ is $256 \times T$ matrix with lookup tables $\mathbf{a}_t$, and $\mathbf{T}$ is $1 \times T$ matrix with rejection thresholds θ_t for each weak classifier. The t-th column of the matrices correspond to parameters h_t. Note that in the case of LRD features, the size of $\mathbf{F}$ is $6 \times T$ and the size of $\mathbf{A}$ is $17 \times T$, since LRD has six parameters $\pi_t = (x, y, w, h, a, b)$ and 17 output values for indexing. In Section VI we use matrix $\mathbf{F}$ as a part micro program of the detection engine $\mathbf{A}$ and $\mathbf{T}$ are stored as lookup tables.

E. Classifier Training

Detectors in this work are trained by WaldBoost algorithm [21]. But other algorithm producing a sequence of feature parameters and associates them with the corresponding response values can be used as well, e.g. [12]. The detailed description of the training algorithm is out of the scope of this paper since we focus mainly on the hardware implementation of the detection process. We kindly refer reader to the original paper [21]. Here we only provide informal description of the algorithm for reader to understand how it works.

The input of the algorithm is a pool of feature parameters, target *false negative rate* α, and a large set of training instances. E.g., when training a face detector, the training instances are image patches representing faces. The parameter α represents tradeoff between the final detector speed and its accuracy. Higher values of α (e.g. $\alpha = 0.2$) produces fast detectors with low value of $\bar{t}$, since they can reject background samples more rapidly. Low values (e.g. $\alpha = 0.01$) produces slower detectors with higher $\bar{t}$. We analyze this tradeoff in Section VII on the task of face detection.

The training algorithm works in rounds, training weak classifiers one by one in s greedy manner. On the beginning of a round t, the algorithm loads background samples from a large set of images (not containing the target patterns) using the already trained classifier (i.e. weak classifiers from h_1 to h_{t-1}). For each feature in the pool, weak learner trains confidence values in lookup tables using AdaBoost [43]. In this step, the values are quantized to the resolution required by the FPGA. This is better than ex-post quantization (after the classifier is trained) since it allows training algorithm to adapt on errors caused by the computation with reduced precision [2], [24]. Then, the weak classifier minimizing exponential loss function [1], [43] is selected as h_t. Based on the distribution of H_t, for target and background samples, θ_t is selected so that as much as possible background samples may be rejected while discarding as few as possible target samples for the next round and satisfying target *false negative rate* α.

For the detector training in this work, we use our custom training software which produces detectors suitable for hardware, taking into account all possible quantization effects of the input image and values in lookup tables.

VI. The Architecture

We propose a hardware architecture that implements the key steps of sliding window object detection – image scaling, feature extraction, and classification of image patches. In the following text, we describe the design of the detector and its interface, and compare it to the equivalent software implementation in order to validate it. Figure 5 shows the overall schematic description of the detector.

A. Stripe Memory

The key part of our architecture is a *Stripe Memory Engine* (SME) which stores the active part of the input image and its scaled variants in multiple BRAMs, see Figure 5 for an illustration. When a new line is read from the image source, the data in SME are updated and scaled on the fly. The number of scales stored in SME is limited by the total width of SME raster, which is 4 096 pixels in this paper.

The architecture of SME is optimized for reading a block of pixels in a single clock cycle, so all data required for feature extraction are available in *constant* time. The data access is done in two stages. First, a fixed-size block aligned to certain position is retrieved from BRAMs to registers. Then, from this intermediate block, a sub-block with any size and alignment is retrieved by simple addressing. We store the image stripe in multiple BRAMs organized in a way that each BRAM is referenced only once when reading an aligned, fixed size block of pixels. BRAMs create a pattern of size U×V, and each BRAM stores B pixel block. This is illustrated in Figure 3 for $U = 2$, $V = 3$ and $B = 6$. This requires $U \cdot V$ BRAMs to store the image stripe. Such an organization allows for reading $B \cdot U \times V$ pixel blocks (aligned to B pixels horizontally) in a single clock cycle by referencing all BRAMs.

Although SME can be configured almost arbitrarily, it is limited by the size of BRAM in the target platform.

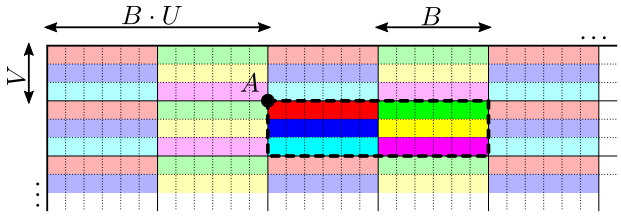

Fig. 3. An example of SME in 2×3 organization and 6px blocks ($U = 2$, $V = 3$ and $B = 6$) stored in 6 BRAMS (color coded). Aligned block A of size 12×3 pixels can be retrieved from the memory in one clock cycle.

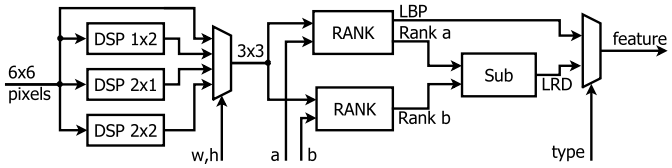

Fig. 4. Circuit for feature extraction. The input is 6×6 pixel block from which, depending on feature parameters w, h, 3×3 cells are extracted. The resulting cells are used to calculate LRD or LBP features.

For practical applications, we use 4096×32 pixel raster, $U = 4$, $V = 8$, $B = 4$ and pixels represented on up to 9 bits. On our target FPGA, the SME takes 32 BRAMs with 36 kbit capacity. This organization allows us to retrieve 16×8 pixel blocks aligned to 4 pixel position. The, for feature extraction, we take 6×6 pixel sub-block or 8×8 sub-block for image scaling.

B. Feature Extraction Unit

The detection engine implements LBP and LRD image features. The size of the feature cells is limited to $w \leq 2$, $h \leq 2$, and thus the feature area is limited to a maximum 6×6 pixels. The position x, y is not limited in any way.

The block diagram of feature extractor is shown in Figure 4. The input is the 6×6 pixel block read from SME according to the absolute feature position in image (taking into account position of analyzed window). DSP blocks extracts all possible variants of $\mathbf{c}$ from the SME. One of the variants is selected for evaluation according to the feature parameters w, h from π. The ranks of elements a and b are calculated as the number of positive comparisons of an element $\mathbf{c}_a$ (resp. $\mathbf{c}_b$) to all other elements in $\mathbf{c}$. The ranks are subtracted to obtain the LRD feature value. Evaluation of an LBP feature is similar – parallel comparison of the central element $\mathbf{c_{1,1}}$ to the elements at the boundary. The response of a weak classifier is obtained from the look-up table $\mathbf{a}$ associated with the extracted feature.

It should be noted that the circuit is designed to extract both LRD and LBP features simultaneously; however, in case that only one feature type is used, the circuitry for the other type is optimized-out during synthesis.

C. Detector Control

The detector implements Algorithm 1. For every position (m, n) in SME image a sequence of instructions is executed. Each instruction reads the 6×6 pixel block from SME, extracts 3×3 cells $\mathbf{c}$, evaluates the feature function and accumulates the response value read from table $\mathbf{A}$. Then, the window is rejected or passed to the next stage based on the threshold value from $\mathbf{T}$. Everything is driven by the parameters from the instruction code. In case of rejection, new position is scheduled for evaluation. When all the instructions are finished, the window coordinates and the confidence value are sent to output.

The detector itself is controlled by a programmable automaton driven by a 32-bit instruction set. An instruction encode parameters for feature extraction – particularly the feature parameters from matrix $\mathbf{F}$ and sequence number identifier t for addressing matrices $\mathbf{A}$ and $\mathbf{T}$. We use 8 bits to encode each coordinate of feature position (x, y), 2 bits for (w, h), and 4 bits for each from (a, b). Note, that values a and b are present in the instruction code even for LBP-based detectors where they are unused. In the matrix $\mathbf{A}$, we store the response values on 9 bits and the thresholds in $\mathbf{T}$ table on 18 bits.

The detector microcode contains a sequence of up to 1 024 instructions; it means the length of the classifier is $T \leq 1024$. The number of instructions can be decreased or even increased, having linear impact on the memory requirements. Current implementation requires 1 BRAM for storing instructions, 1 BRAM for thresholds and 5 BRAMs for $\mathbf{A}$ and $\mathbf{T}$ in LRD case (64 BRAMs in LBP case) 2 BRAMs are occupied by instructions for static execution scheduler (see VI-D).

The evaluation of the classifier is pipelined. The pipeline is 14 clock cycles long and thus up to 14 positions are evaluated simultaneously. Thanks to the memory architecture described above, the pipeline can be utilized to 100 % which is impossible to achieve by previous scanning window approaches [9], [13]. We use two-port BRAM in SME, so we use two pipelines to double the performance. However, a small portion of memory accesses from the second pipeline needs to be reserved for image scaling and for storing the incoming image lines and the down-scaled data back to the SME – we leave one out of every 4 clock cycles for the scaling unit to generate the scaled images, and therefore the overall performance is $\bar{p} = 1.75$ features extracted per clock cycle.

D. Image Scaling

Besides the original image, SME stores scaled variants of the image. The scaling is done on-the-fly over few last image lines. We use a block-based approach for scaling with fixed factor $S = \frac{5}{6}$, where 6×6 pixels blocks from a base scale are transformed to 5×5 pixels blocks in the subsequent scale. We implemented the separable, integer version of Lanczos [44] scaling algorithm for 8 bit images.

The process of downscaling and detector execution on individual SME lines is statically scheduled and driven by the microcode stored in BRAM. The classifier operations are performed to every line but every scale has a different number of lines to process. Moreover, the scaling is a block operation which is performed every 6-th line. This can cause occasional bursts of detector executions. The static scheduling allows us to distribute the execution of detector and scaling to avoid

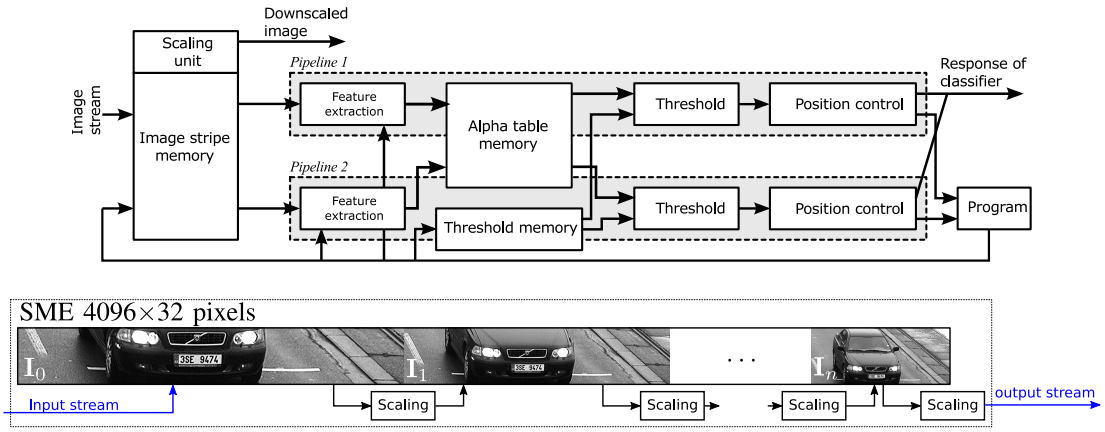

Fig. 5. **(Top)** Block diagram of detector. The SME unit, which stores the image and produces downscaled images and two detection pipelines, driven by microcode program. **(Bottom)** Illustration of the stripe memory that we use for image representation and source of data for detector evaluation is shown, too. Incoming line (blue) is stored as a last line in the buffer. When possible, 6×6 blocks on the bottom of the buffer are scaled and stored as 5×5 blocks in subsequent scales. See the text and Figure 3 for details on how the data are stored in FPGA BRAMs.

this execution bursts and ensure regular processing of image stream.

The maximum height of scanning window is given by height of SME minus size of block produced by scaling unit, which is 27px (32-5px in our case). This height is sufficient for many of detection tasks and also standard detectors uses similar dimensions – 21×21 or 24×24 pixels [1], [21].

E. Detector Interface

From the outside view, the detector is a computational block with one input, one configuration interface and two outputs. The input reads a stream of incoming image data of the given resolution. The configuration interface itself is composed from the detector definition (instructions and associated look-up tables), input image size, and sizes of scaled versions. The first output is a stream of the image data from the smallest scale in the SME. This output is used as an input for another detector instance. The second output contains detection results – coordinates and scale of detected objects. For both image input and output, we use AXI Stream Video interface for configuration the AXI-Lite interface and AXI Stream for detection results. This interfaces simplify integration of the detector to applications.

F. Stripe Memory Cascades

A single detector block is limited by the width of SME (4 096 pixels in our case) and by the performance of feature extraction, which is $\bar{p} = 1.75$ features/Hz (i.e. 350 M features/s with 200 MHz clock). From the performance point of view, it is not efficient to build the detector with a wider window (buffer) to hold more image scales, because the limitation of feature extraction speed would still remain. Our design allows for a more efficient solution – cascaded connection of detector blocks which we call *Stripe Memory Cascade*, illustrated in Figure 6. In the cascade, one detector instance generates scaled version of image and passes it to the subsequent instance. No limitation exists on the number of instances, except for the resources available on the target platform.

TABLE II

EXAMPLES OF CASCADE CONFIGURATIONS, THEIR PREDICTED PERFORMANCES, AND RESOURCE REQUIREMENTS. VALID FOR DETECTOR OF SIZE 21×21 PX, $\bar{t} = 2.5$, AND $f = 200$ MHz

Version	Feature	Res. [pixels]	Scales	Insts.	BRAM	REG	LUT	FPS
VGA/1	LRD	640×480	18	1	41	7640	9933	160
HD/2	LRD	1280×720	20	2	82	15292	19919	64
HD/3	LRD	1280×720	20	3	123	22944	29905	94
HD/4	LRD	1280×720	20	4	164	30596	39891	159
FHD/4	LRD	1920×1080	22	4	164	30596	39891	60
UHD/7	LRD	3840×2160	26	7	288	53552	69849	17
VGA/1	LBP	640×480	18	1	100	7650	9978	160
HD/2	LBP	1280×720	20	2	200	15312	20009	64
HD/3	LBP	1280×720	20	3	300	22974	30040	94
HD/4	LBP	1280×720	20	4	400	30636	40071	159
FHD/4	LBP	1920×1080	22	4	400	30636	40071	60
UHD/7	LBP	3840×2160	26	7	700	53622	70164	17

All instances operates in simulataneously, effectively increasing the speed of the feature extraction. Output streams from all the instances are simply merged to form the output of the cascade.

Table II shows several configurations of cascaded instances, their performance, and resources they require. The naming convention we use for the configurations encodes the resolution processed and the number of detector block instances, e.g. `VGA/1` is configuration for processing of VGA image with one detector block instance and it is similar to what was proposed by Zemcik *et al.* [2]. Versions `HD/2`, `HD/3` and `HD/4` are configurations for HD image with different performance and resource requirements due to different assignment of image scales to detector instances. Versions `FHD/4` and `UHD/7` are for Full HD and 4K images. The versions for LBP and LRD differs mainly in memory requirements because LBP requires more BRAMs for classifier definition as described earlier in this paper.

G. Speed Analysis

The theoretical maximum throughput (in frames per second) for one instance of the detector unit can be estimated using

TABLE III

COMPARISON OF THREE DIFFERENT CASCADE DESIGNS, ALL FOR HD RESOLUTION. THE VALUES ARE SHOWN FOR DETECTOR OF SIZE 21×21 PX WITH AVERAGE $\bar{t} = 2.5$ FEATURES PER POSITION, AND $f = 200$ MHz CLOCK. IT CAN BE OBSERVED THAT TRADE-OFF BETWEEN THE NUMBER OF INSTANCES AND DESIRED DETECTION PERFORMANCE EXISTS

Scale	Resolution	$P \cdot \bar{t}$	HD/2			HD/3			HD/4		
			$\sum P \cdot \bar{t}$	$\sum W$	FPS	$\sum P \cdot \bar{t}$	$\sum W$	FPS	$\sum P \cdot \bar{t}$	$\sum W$	FPS
1	1280×720	2200103	5468605	3979	64	3714188	2347	94	2200103	1280	159
2	1067×600	1514085							1514085	1067	231
3	890×500	1040628				3058318	2856	114	1754418	1632	199
4	742×417	713790									
5	619×348	488865									
6	516×290	332887									
7	430×242	225972									
8	359×202	152945									
9	300×169	103230	1497155	3536	233				1497155	3536	233
10	250×141	68700				193255	1312	1811			
11	209×118	45590									
...	...	...									
20	42×25	210									
FPS					**64**			**94**			**159**

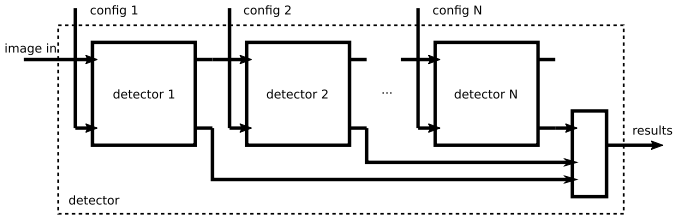

Fig. 6. Cascading of detector instances. Each detector takes scaled input image from the previous one, the output coordinates and classifier response of detected objects are merged into one stream. Each detector is configured separately.

Equation (11) where f is the operating frequency, $\bar{p}$ the number of features extracted in one clock cycle, $\bar{t}$ is the average number of weak classifiers evaluated per window, and P represents the number of positions to evaluate in the image and its scaled versions assigned to the detector. The numerator of Equation (11) represents the total number of features extracted by the detector, the denominator is the average number of features that must be extracted on an image. As explained in Section VI-C, in our architecture has $\bar{p} = 1.75$. The value of $\bar{t}$ is the property the particular classifier, the average number of features that needs to be extracted from the image in order to decide the class of one analyzed window (see Section V). It reflects the average case and it can change locally with irregularities in data that are hard to predict. We use $\bar{t} = 2.5$ for illustration purposes which is a realistic value for face detection. See Section VII with the analysis of detectors we use.

$$F = \frac{f \cdot \bar{p}}{P \cdot \bar{t}} \tag{11}$$

The throughput of the whole cascade of detectors is limited by the slowest unit in the chain and it depends largely on sizes of images processed by the individual instances. In the Table III, we show breakdown of `HD/*` variants from Table II). Each version processes 20 image scales, the difference is in the manner how the scales are assigned to the detectors in the chain, and in the length of the chain.

Let us focus, for example, on the `HD/2` version, with two instances of detector. The first instance contains four image levels (resolutions from 1 280 pixels to 742 pixels of width) and we estimate around 5.4 M features need to be calculated on those four levels on average. Therefore, the speed of the first instance is around 64 FPS, calculated using Equation (11). The second instance contains the rest of the image scales (resolutions from 619 pixels to 42 pixels of width) and its speed is estimated to 233 FPS. The total speed of the `HD/2` is therefore 64 FPS as it is the minimal framerate from all detectors in the chain.

H. Validation

During the design phase, we developed a software implementation of the detection algorithm which uses the same input data as the hardware implementation (look-up tables, instructions, thresholds, etc), and is based on the same image scaling algorithm. The architecture was validated by comparison of the results produced by the software implementation to the results produced by our architecture on a large set of images. The results were identical; therefore, we assume that the subtle differences in implementation in software and hardware are acceptable.

VII. RESULTS AND EVALUATION

In our applications we use Xilinx Zynq SoC with ARM CPU and FPGA. This combination allows for simple configuration of the detector and post processing of the results. However, if required, everything can be fixed and implemented in FPGA only. The design was written completely in VHDL with only few platform-dependent blocks (such as 36 kbit BRAM capacity); thus, it could be relatively easily adapted to various FPGAs, even from different vendor.

We built a prototype of a smart camera with HD CMOS image sensor and Zynq SoC Z-7020 chip. The camera captures

image at 60 FPS and passes it through the HD/2 detector. The detection results are processed on ARM core (non-maxima suppression, filtering) and the image along with the coordinates of detected objects are streamed through the network. We demonstrate the architecture on the detection of frontal faces and detection of license plates. As an example of our technology, we provide an IP Core of version VGA/1 and HD/2 detector with built-in face detector.[1] This IP Core takes approximately 15 % of Zynq Z-7020 resources.

A. Detector Evaluation

Properties of WaldBoost detectors were experimentally evaluated many times on various problems [21], [40], [45], [46]. We tested our architecture in two example scenarios – face detection and license plate detection. These two applications are important in surveillance tasks. However, the detector can be used for detection of other rigid objects as well - cars [47], pedestrians [28] etc. We compare our detectors to the pre-trained detectors from OpenCV which implement Haar and LBP Cascades used by other state-of-the-art architectures. We report Receiver operating curve (ROC) – the tradeoff between *false positive rate* (the number of false detections generated per one image) and *miss rate* (the ratio of missed objects). Figure 9 shows a few images from each of the tasks.

Detection of Frontal Faces

We trained frontal face detectors on a large dataset of faces and compared them to OpenCV cascade detectors widely used by other authors as a baseline [6], [9], [26], [31]. The detector window size (u, v) was set to 24×24 pixels and the detector length to $T = 1024$. We trained four detectors with different target *false negative rate* $\alpha \in \{0.01, 0.05, 0.1, 0.2\}$, see Section V-E for details. From OpenCV, we used `haarcascade_frontalface_alt`, as it gives the best results from the built-in detectors. We tested the detectors on our set of 102 high resolution images with 1 857 annotated frontal faces (which is bigger and more challenging than MIT-CMU usually used for testing of frontal face detectors). The results in Figure 7 show that our detector (with $\alpha = 0.1$) gives almost $10\times$ less false positives compared the detectors from OpenCV at the same recall level. The *recall* of OpenCV detectors is 94 % as reported by others [2], [6]–[11]. Table IV summarizes the speed and recall tradeoffs of the detectors trained with different value of α and their predicted performance in FPS when executed in version HD/2 architecture. 60 FPS margin is satisfied by classifiers with $\alpha \leq 0.1$.

Detection of License Plates

In law enforcement applications, such as speed measurement, detection of licence plates is a crucial step where accuracy and speed matters very much. We trained a license plate detector on a proprietary database of images taken by speed enforcement cameras. The dataset contains 30 000 automatically obtained samples of axis aligned license plates. The test set contains 1 000 images with manually corrected annotations. The dataset covers a wide range of conditions – day, night, sun, rain, snow and fog. For our experiments, the detector window size (u, v) was set to 84×12 pixels and the detector length was $T = 1024$. Accuracy evaluation in Figure 7 shows that the detection rate of WaldBoost detector is over 99 % when a false alarm occurs on one out of one hundred images. Detector speed measured on the test set is $\bar{t} = 2.7$, corrensponding to 62 FPS in HD/2. This is more than sufficient for this kind of application. For comparison, we trained Haar and LBP cascade from OpenCV on the same data using tools installed with the library. As Figure 7 suggests, our detector outperforms OpenCV detectors by a large margin.

[1] All resources can be downloaded from https://github.com/PetrMusilCZ/WaldBoost_FPGA after review.

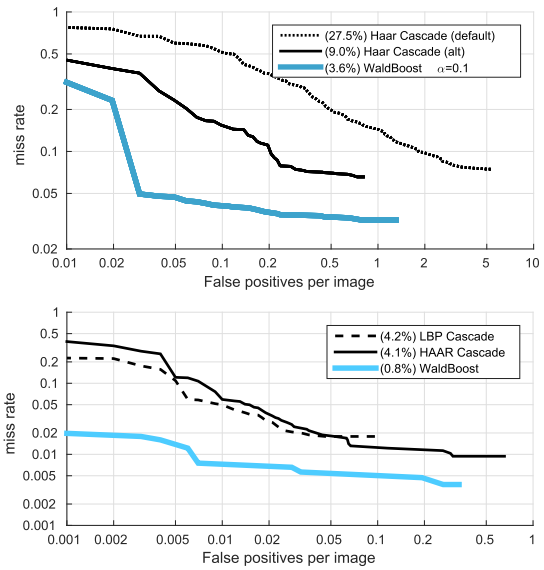

Fig. 7. Accuracy evaluation of our detectors (WaldBoost) and comparison to OpenCV detectors (Haar and LBP cascade) for frontal face detection (top) and license plate detection (bottom). WaldBoost gives lower false positives at the same accuracy level.

TABLE IV

SPEED ANALYSIS OF OUR FACE DETECTORS ON HD/2. FAST DETECTORS HAVE SLIGHTLY LOWER ACCURACY. THE IMPORTANT VALUE IS $\bar{t}$, WHICH DIRECTLY INFLUENCE THE PERFORMANCE OF OUR ARCHITECTURE

	$\alpha = 0.01$	$\alpha = 0.05$	$\alpha = 0.1$	$\alpha = 0.2$
Recall	0.970	0.969	0.964	0.952
$\bar{t}$	7.54	4.43	2.50	1.96
F [FPS]	21	36	64	82

B. Power Consumption Comparison

Table V shows estimation of power consumption of different platforms executing the face detection algorithm with Waldboost classifier ($\bar{t} = 2.5$, $\alpha = 0.1$) on 1280×720 images. On CPU, GPU, and Tegra, we used an OpenCL implementation of the detection algorithm from Herout *et al.* [5]. The critical steps were implemented in OpenCL and compiled for the target platform, efficiently exploiting SSE/AVX instructions and multi-thread execution on CPU and computing cores as well as texturing units on GPU. For the Intel CPU,

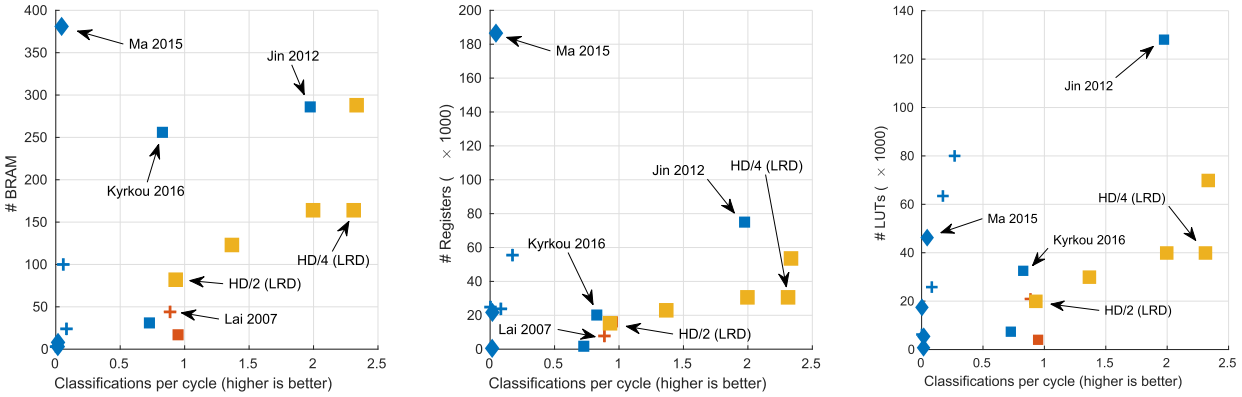

Fig. 8. Comparison of classifications per cycle (*wpc*) and resource requirements of the architectures from Table I. Yellow color encodes *proposed* architectures from this paper, red color *preprocessing units*, blue color the rest of the architectures. Marker shape encodes feature type used by the architecture (◆ stands for HOG, ✚ for Haar features, and ■ for LBP/LRD features).

TABLE V
POWER CONSUMPTION COMPARISON. FPGA-BASED DETECTOR IS MORE POWER EFFICIENT COMPARED TO PC AND GPU SOLUTIONS

	platform	FPS	Power (W)	mJ / Frame
PC	Intel i7 3770K	22	77	3500
GPU	GeForce 1080Ti	915	120	131
SOC	Tegra K1	38	4	105
HD/2	Artix7 xc7a75	64	1.52	24
HD/4	Artix7 xc7a200	159	2.95	19

we report maximum thermal design power (TDP). In case of GPU and SoC, the accurate chip power consumption is available. Power consumption of the FPGA was estimated using Xilinx Power Estimator, assuming the worst case with a 100% toggle rate (i.e. when signals flips every clock cycle). For the measurement purposes, we synthesized the HD/2 in the different FPGA of the same family without the ARM core, so the results are not influenced by the power consumption of the ARM which is, in fact, not required during the detection. For all platforms, we report the metric which expresses energy consumed by the platform per one frame (Joules/Frame). The Table V shows that the FPGA design requires approximately five-times less energy than SoC Nvidia Tegra.

C. Comparison to Other Architectures

Table I shows the comparison to other architectures in terms of the maximum image resolution, detection algorithm and features, scanning window size, and type of detected object.

Due to our unique stream memory cascades, the detector can process images at very high resolution (up to 4K) while it is still capable of detection of very small objects. This property may be important e.g. when surveillance camera covers a large area. The most state-of-the-art architectures are capable of processing up to 1Mpix images.

The advantage of the proposed architecture, comparing to others, is the optional size of the detection window, which is limited only by the height of SME, while the width remains unlimited and freely adjustable. In other architectures [6], [9], [13], [26] the changing of window size means re-synthesizing of the whole design and, what is worse, larger windows takes more resources for multiplexer networks required for reading out the pixels. This is completely avoided in the proposed solution.

We compare our architecture to other similar ones. However, they are realized by different FPGA technology, have different input sizes, classifier strides in image and other parameters; these parameters are discussed in Section VIII. To make comparison possible, we characterize all the architectures by number of processed scanning windows per clock cycle (*wpc*) which gives raw performance measure independent on the used technology, frame rate and other parameters. Table VI summarizes performance and resource requirements of our architecture and compares it to the other published works. Plots in Figure 8 show the dependency of resource consumption on the *wpc* classification for some architectures. Various proposed configurations of cascade instances (from Table II) are plotted in each graph. It can be observed that overall performance increases with number of cascaded SME instances which proves its benefit.

The resource consumption and performance of configuration HD/2 is comparable to Kadlcek and Fucik [11] and Lai *et al.* [23], anyway, their design achieves only low detection accuracy caused by very short detector and limits their possible usage for preprocessing purposes only. Our architecture, on the other hand, works as fully featured object detector while providing sufficient performance even at high image resolutions. In summary, the graphs on Figure 8 and Table VI shows the performance superiority of LBP/LRD feature based detectors to Haar and HOG based detectors.

The solution by Jin *et al.* [27] and Kyrkou *et al.* [15] achieves very high *wpc*, comparable to our FHD/4 and HD/2 configurations, but they require multiple-times more resources against our solution, even for only low image resolution. Comparing to work of Zemcik *et al.* [2], the architecture

TABLE VI

COMPARISON OF CRITICAL PARAMETERS OF THE PUBLISHED ARCHITECTURES REPORTED BY THE AUTHORS TO THE PROPOSED ARCHITECTURE. SOME WORKS DO NOT REPORT ALL PARAMETERS OR REPORT *slices* (SL) OR *logical elements* (LE). *SUITABLE ONLY FOR PREPROCESSING PURPOSES

	Feature	Platform	Res. [pixels]	FPS	Stride	*wpc*	f [MHz]	BRAM	LUT	REG	DSP
Lai* [23]	Haar	Virtex2 VP30	640×480	143	1	0.886	126	44	20 900	7 800	–
Granat [13]	Haar	Virtex2 LX250	256×256	< 5	1	0.058	24	100	–	–	–
Cho [6]	Haar	Virtex5 LX110T	640×480	7	1	–	–	41	66 900	21 900	–
Hiromoto [25]	Haar	Virtex5 LX330	640×480	30	1	0.173	160	–	63 440	55 515	–
Martelli [8]	Covariance	Virtex6 LX240T	640×480	132	8	0.002	154	3	1 553 (SL)	–	22
Kyrkou [10]	Haar	Virtex2 VP30	320×240	64	1	0.083	100	24	25 800	23 800	–
Huang [9]	Haar	Virtex5 LX155T	320×240	100	1	0.268	65	–	80 000	–	–
Brouss [26]	Haar	Stratix4 GX530	320×240	50	1	0.083	125	–	–	–	–
Jin [27]	LBP	Virtex5 LX330	640×480	300	1	1.974	125	286	128 041	74 997	–
Kadlcek* [11]	LBP	Virtex2 LX250	1024×1024	130	1	0.948	137	17	1 007 (SL)	–	–
Zemcik [2]	LBP/LRD	Spartan6 LX45T	640×480	160	1	0.726	152	31	7 373	1 732	–
Yazawa [28]	HOG	Cyclone III	640×480	13	5	0.002	70	–	17 419 (LE)	11 306	–
Ma [29]	HOG	Virtex-6 LX760	1620×1200	10	4	0.045	150	381	46 238	186 531	190
Said [30]	HOG	Virtex6 LX240T	640×480	292	4	0.017	222	8	1 357 (SL)	–	46
Kyrkou [15]	LBP	Spartan6 LX150T	800×600	40	1	0.827	70	256	32 532	20 153	59
Bilal [32]	HOG	Cyclone IV	640×480	25	4	0.015	50	3	751	496	0
Proposed LRD (HD/2)	**LRD**	**Zynq Z-7020**	**1280×720**	**64**	**1**	**0.930**	**200**	**82**	**19 919**	**15 292**	**0**
Proposed LBP (HD/2)	**LBP**	**Zynq Z-7045**	**1280×720**	**64**	**1**	**0.930**	**200**	**200**	**20 009**	**15 312**	**0**
Proposed LRD (UHD/7)	**LRD**	**Zynq Z-7045**	**3840×2160**	**17**	**1**	**2.334**	**200**	**288**	**69 849**	**53 552**	**0**

Fig. 9. Examples of detected objects on selected images from testing datasets for face detection (top) and license plate detection (bottom).

we improved, we achieved higher performance, *wpc*, and maximum image resolution due to a cascading nature of our design.

SVM based classifiers using HOG features presented by Said and Atri [30] and Martelli*et al.* [8] achieves high framerate mainly due to detection stride, where they process only every x-th image row and column, effectively making the image $x\times$ smaller, which reflects into low throughput. Moreover, they can only detect objects with fixed size 128×64 pixels as they do not solve multi-scale detection. This is why their architecture performs so well with so limited resources. However, to detect smaller objects, they need to upscale the image, and for multi scale detection, pyramidal representation need to be created, increasing the search space and slowing down the detection.

The proposed architecture achieves the highest performance (represented by *wpc*) compared to the others and also has a relatively low resource consumption as is evident from the Table VI and graphs on the Figure 8.

VIII. DISCUSSION

The unique feature of the proposed architecture is the cascading nature of detector blocks, where one block passes re-scaled image data to the subsequent block in the chain. The total speed is then limited by the slowest element in the chain. It is therefore possible to conveniently exchange resources for throughput.

In the architecture description, we were addressing the situation when all image windows and scales are processed – a full search. This is quite computationally demanding and, in many applications, unnecessary. The performance of the detection can be improved by several application-dependent methods. Scanning every n-th position in the image can increase the throughput proportionally to n. This can be used especially when detector window is large – e.g. in the license plate detection scenario above, we can scan every second or even third position often without serious impact on detection accuracy, while increasing throughput two or three times. In scenarios where the size of detected objects is known, or where we do not need to detect very small objects, image pre-scaling can be applied before it is passed to the detector. Moreover, the detector can be controlled to scan only certain scales. It can be controlled what lines are passed to the detection unit and overall performance can be improved by using only lines from certain image areas. This can be important e.g. in license plate detection where only license plates in the

central part of the image are important for the application. During the development we always considered clock frequency $f = 200\,\text{MHz}$, however, the frequency can be increased, especially when the design is simple and takes little resources, to improve the throughput. Another way how to increase clock frequency is to prolong the detector pipeline.

IX. Conclusion

In this paper, we proposed and described in detail a novel architecture for object detection in FPGA. Our architecture uses detectors trained by WaldBoost algorithm with LBP or LRD image features, as opposed to AdaBoost Cascades used in other architectures. The key features of the architecture are simple representation of the detector by microcode, memory representation of the input image in *Stripe memory Engine*, cascading of detector units to scale performance, and parallel processing of image patches. We presented several configurations of detector cascades and analyzed their properties in detail. For research purposes, we provide an IP core for face detection in VGA and HD resolution. In the future, we will improve our architecture in order to include decision tree based detectors that promises better detection speed and accuracy.

References

[1] P. Viola and M. J. Jones, "Robust real-time face detection," *Int. J. Comput. Vis.*, vol. 57, no. 2, pp. 137–154, 2004.

[2] P. Zemcik, R. Juranek, P. Musil, M. Musil, and M. Hradis, "High performance architecture for object detection in streamed videos," in *Proc. Int. Conf. Field Program. Log. Appl.*, Sep. 2013, pp. 1–4.

[3] *OpenCV Library*. [Online]. Available: http://opencv.org

[4] R. Juránek, A. Herout, and P. Zemcík, "Implementing the local binary patterns with SIMD instructions of CPU," in *Proc. Int. Conf. Central Eur. Comput. Graph.*, 2010, pp. 39–42.

[5] A. Herout, R. Jošth, and R. Juránek, J. Havel, M. Hradiš, and P. Zemcík, "Real-time object detection on CUDA," *J. Real-Time Image Process.*, vol. 6, no. 3, pp. 159–170, 2011.

[6] J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, "Fpga-based face detection system using Haar classifiers," in *Proc. Int. Symp. Field Program. Gate Arrays*, 2009, pp. 103–112.

[7] C. Kyrkou, C. Ttofis, and T. Theocharides, "FPGA-accelerated object detection using edge information," in *Proc. Int. Conf. Field Program. Log. Appl.*, Sep. 2011, pp. 167–170.

[8] S. Martelli, D. Tosato, M. Cristani, and V. Murino, "Fast FPGA-based architecture for pedestrian detection based on covariance matrices," in *Proc. 18th IEEE Int. Conf. Image Process.*, Sep. 2011, pp. 389–392.

[9] C. Huang and F. Vahid, "Scalable object detection accelerators on FPGAs using custom design space exploration," in *Proc. IEEE 9th Symp. Appl. Specific Process.*, Jun. 2011, pp. 115–121.

[10] C. Kyrkou and T. Theocharides, "A flexible parallel hardware architecture for AdaBoost-based real-time object detection," *IEEE Trans. Very Large Scale Integr.*, vol. 19, no. 6, pp. 1034–1047, Jun. 2011.

[11] F. Kadlček and O. Fučík, "Automatic synthesis of small AdaBoost classifier in FPGA," in *Proc. IEEE 16th Int. Symp. Des. Diagnostics Electron. Circuits Syst.*, Apr. 2013, pp. 92–97.

[12] P. Dollár, R. Appel, S. Belongie, and P. Perona, "Fast feature pyramids for object detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 8, pp. 1532–1545, Aug. 2014.

[13] J. Granát, A. Herout, M. Hradiš, and P. Zemcík, "Hardware acceleration of AdaBoost classifier," in *Proc. Workshop Multimodal Interact. Rel. Mach. Learn. Algorithms*, 2007, pp. 1–12.

[14] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1, Jun. 2005, pp. 886–893.

[15] C. Kyrkou, C.-S. Bouganis, T. Theocharides, and M. M. Polycarpou, "Embedded hardware-efficient real-time classification with cascade support vector machines," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 1, pp. 99–112, Jan. 2016.

[16] T. Kryjak, M. Komorkiewicz, and M. Gorgon, "FPGA implementation of real-time head-shoulder detection using local binary patterns, SVM and foreground object detection," in *Proc. Conf. Des. Archit. Signal Image Process.*, Oct. 2012, pp. 1–8.

[17] C.-H. Yeh, C.-Y. Lin, K. Muchtar, H.-E. Lai, and M.-T. Sun, "Three-pronged compensation and hysteresis thresholding for moving object detection in real-time video surveillance," *IEEE Trans. Ind. Electron.*, vol. 64, no. 6, pp. 4945–4955, Jun. 2017.

[18] D. Bouris, A. Nikitakis, and I. Papaefstathiou, "Fast and efficient FPGA-based feature detection employing the SURF algorithm," in *Proc. 18th IEEE Annu. Int. Symp. Field-Program. Custom Comput. Mach.*, May 2010, pp. 3–10.

[19] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, Jun. 2010. pp. 257–260.

[20] Y. D. Sato and Y. Kuriya, "Multi–scale elastic graph matching for face detection," *EURASIP J. Adv. Signal Process.*, vol. 2013, p. 175, Nov. 2013.

[21] J. Sochman and J. Matas, "WaldBoost—Learning for time constrained sequential detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2, Jun. 2005, pp. 150–156.

[22] P. Dollár, Z. Tu, P. Perona, and S. Belongie, "Integral channel features," in *Proc. Brit. Mach. Vis. Conf.*, 2009, pp. 91.1–91.11.

[23] H.-C. Lai, M. Savvides, and T. Chen, "Proposed FPGA hardware architecture for high frame rate (>100 fps) face detection using feature cascade classifiers," in *Proc. 1st IEEE Int. Conf. Biometrics, Theory, Appl., Syst.*, Sep. 2007, pp. 1–6.

[24] P. Zemcik and M. Zadnik, "Adaboost engine," in *Proc. Int. Conf. Field Program. Log. Appl.*, Aug. 2007, pp. 656–660.

[25] M. Hiromoto, H. Sugano, and R. Miyamoto, "Partially parallel architecture for AdaBoost-based detection with Haar-like features," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 1, pp. 41–52, Jan. 2009.

[26] B. Brousseau and J. Rose, "An energy-efficient, fast FPGA hardware architecture for OpenCV-compatible object detection," in *Proc. Int. Conf. Field-Program. Technol.*, Dec. 2012, pp. 166–173.

[27] S. Jin, D. Kim, T. T. Nguyen, D. Kim, M. Kim, and J. W. Jeon, "Design and implementation of a pipelined datapath for high-speed face detection using FPGA," *IEEE Trans. Ind. Informat.*, vol. 8, no. 1, pp. 158–167, Feb. 2012.

[28] Y. Yazawa *et al.*, "FPGA hardware with target-reconfigurable object detector," *IEICE Trans. Inf. Syst.*, vol. E98-D, no. 9, pp. 1637–1645, 2015.

[29] X. Ma, W. A. Najjar, and A. K. Roy-Chowdhury, "Evaluation and acceleration of high-throughput fixed-point object detection on FPGAs," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 6, pp. 1051–1062, Jun. 2015.

[30] Y. Said and M. Atri, "Efficient and high-performance pedestrian detector implementation for intelligent vehicles," *IET Intell. Transp. Syst.*, vol. 10, no. 6, pp. 438–444, Aug. 2016.

[31] Z. Xu, R. Shi, Z. Sun, Y. Li, Y. Zhao, and C. Wu, "A heterogeneous system for real-time detection with AdaBoost," in *Proc. IEEE 18th Int. Conf. High Perform. Comput. Commun.*, Dec. 2016, pp. 839–843.

[32] M. Bilal, A. Khan, M. U. K. Khan, and C.-M. Kyung, "A low-complexity pedestrian detection framework for smart video surveillance systems," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 10, pp. 2260–2273, Oct. 2017.

[33] J. Yang, Y. Yang, Z. Chen, L. Liu, J. Liu, and N. Wu, "A heterogeneous parallel processor for high-speed vision," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 3, pp. 746–758, Mar. 2018.

[34] S. Ren, K. He, R. Girshick, and J. Sun. (2015). "Faster R-CNN: Towards real-time object detection with region proposal networks." [Online]. Available: https://arxiv.org/abs/1506.01497

[35] J. Redmon and A. Farhadi. (2016). "YOLO9000: Better, faster, stronger." [Online]. Available: https://arxiv.org/abs/1612.08242

[36] S. Bhattarai, A. Madanayake, R. J. Cintra, S. Duffner, and C. Garcia, "Digital architecture for real-time CNN-based face detection for video processing," in *Proc. Cogn. Commun. Aerosp. Appl. Workshop*, Jun. 2017, pp. 1–6.

[37] J. Qiu *et al.*, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. Int. Symp. Field-Program. Gate Arrays*, New York, NY, USA: ACM, 2016, pp. 26–35.

[38] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, "Energy-efficient CNN implementation on a deeply pipelined FPGA cluster," in *Proc. Int. Symp. Low Power Electron. Design*. New York, NY, USA: ACM, 2016, pp. 326–331.

[39] M. Hradiš, A. Herout, and P. Zemcík, "Local rank patterns—Novel features for rapid object detection," in *Proc. Int. Conf. Comput. Vis. Graph.*, 2008, pp. 1–12.
[40] R. Juranek, M. Hradis, and P. Zemcík, *Real-Time Algorithms of Object Detection Using Classifiers*. Rijeka, Croatia: InTech, 2012, pp. 1–22.
[41] L. Zhang, R. Chu, S. Xiang, S. Liao, and S. Z. Li, "Face detection based on multi–block LBP representation," in *Proc. Int. Conf. Biometrics*, 2007, pp. 11–18.
[42] T. Ojala, M. Pietikäinen, and T. Mäenpää, "Gray scale and rotation invariant texture classification with local binary patterns," in *Proc. Eur. Conf. Comput. Vis.*, 2000, pp. 404–420.
[43] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Mach. Learn.*, vol. 37, no. 3, pp. 297–336, 1999.
[44] K. Turkowski, "Properties of surface-normal transformations," in *Graphics Gems*. Cambridge, MA, USA: Academic, 1990, pp. 539–547.
[45] J. Šochmanand and J. Matas, "Learning fast emulators of binary decision processes," *Int. J. Comput. Vis.*, vol. 83, no. 2, pp. 149–163, Jun. 2009.
[46] C. Caraffi, T. Vojír, J. Trefný, J. Šochman, and J. Matas, "A system for real-time detection and tracking of vehicles from a single car-mounted camera," in *Proc. 15th Int. IEEE Conf. Intell. Transp. Syst.*, Sep. 2012, pp. 975–982.
[47] A. Broggi, E. Cardarelli, S. Cattani, P. Medici, and M. Sabbatelli, "Vehicle detection for autonomous parking using a soft-cascade AdaBoost classifier," in *Proc. IEEE Intell. Vehicles Symp. Proc.*, Jun. 2014, pp. 912–917.

Petr Musil received the M.Sc. degree from the Brno University of Technology, where he is currently pursuing the Ph.D. degree. He is currently a member of the Graph@FIT Group, Department of Computer Graphics and Multimedia, Faculty of Information Technology, Brno University of Technology. His interests include image processing and computer vision algorithms and their acceleration on programmable hardware and embedded devices.

Roman Juránek received the Ph.D. degree from the Faculty of Information Technology, Brno University of Technology, Czech Republic, where he is currently a Research Scientist. His research interests include machine learning, image processing, computer vision, and fast object detection. He also focuses on computer vision in traffic surveillance.

Martin Musil received the M.Sc. degree from the Brno University of Technology, where he is currently pursuing the Ph.D. degree. He is currently a member of the Graph@FIT Group, Department of Computer Graphics and Multimedia, Faculty of Information Technology, Brno University of Technology. He also focuses on embedded systems, programmable hardware, and acceleration of signal and image processing algorithms.

Pavel Zemčík received the Ph.D. degree from the Faculty of Electrical Engineering and Computer Science, Brno University of Technology, Czech Republic. He is currently a Full Professor, the Dean, and a member of the Graph@FIT Group, Department of Computer Graphics and Multimedia, Faculty of Information Technology, Brno University of Technology. His interests include computer vision and graphics algorithms, acceleration of algorithms, programmable hardware, and also applications.