FPGA Based Traffic Sign Detection for Automotive Camera Systems

Fynn Schwiegelshohn, Lars Gierke and Michael Hübner
Chair for Embedded Systems of Information Technology
Ruhr-Universität-Bochum
Bochum, Nord-Rhein-Westfalen
Email: {Fynn.Schwiegelshohn; Lars.Gierke; Michael.Huebner}@rub.de

***Abstract*—Advanced driver assistance systems (ADAS) have become very prominent in todays automobiles. The technological advancement of already familiar assistance systems enable the car to now autonomously assess the current situation and react accordingly. In terms of data processing, there is no difference between actually acting i.e. accelerating, braking or steering and just issuing warnings to alert the driver of a dangerous situation. In this paper, we introduce a camera based image processing system for traffic sign detection for FullHD resolution. This system is able to detect speed limit traffic signs but additional traffic signs can be implemented using the same model. The hardware components consist of a Microblaze softcore from Xilinx and an extended IP core for HDMI-in and out signals. The system is implemented on a a Spartan-6-FPGA. For image acquisition, an off-the-shelf car camera is used. The developed system is able to reliably detect traffic signs on short distances on static images as well as on image streams.**

Keywords: Atlys Board; FPGA; Spartan-6; Image processing; Traffic sign detection;

I. Introduction

In the last years, image resolutions have increased considerably up to the FullHD resolution of 1920×1080 pixels. Currently, we are at the border of $4k$ resolutions becoming publicly available and development for even higher resolutions is underway. With these higher resolutions in images, object detection at larger distances becomes feasible but also higher data bandwidth is required. For a FullHD image with 60 frames per second (fps), the data stream will transfer 148.500.000 pixel per second including vertical and horizontal blanking periods. Here, the challenge is to analyze the huge amount of data without discarding or missing important information. When targeting applications in a safety critical environment, real-time constraints also have to be applied since the system should either prevent or inform the user well enough in advance of a dangerous situation. Modern high-end processors have enough computational power for these tasks but also require a lot of energy for operation. For embedded systems such as ADAS however, small power consumption and reliability is of high importance. Therefore, FPGAs provide an promising approach to solve this problem since they can adapt their hardware based to the current need of the application. Additionally, high cost pressure in the automotive domain enforces the usage of efficient hardware like FPGAs which reach high computational performance and low power consumption without requiring active cooling at design and also at run-time. A high frame rate is for traffic sign detection very important, since it ensures that objects of interest can easily be tracked over subsequent images. When a camera is recording images with 60 fps, even high velocities do not cover large distances as seen in equation (1).

$$s_{CAM} = v \cdot t = 180\frac{km}{h} \cdot \frac{1}{60} sec = 0.83m \qquad (1)$$

Generally, high image resolutions are not mandatory for traffic sign detection but they do increase the detection range of the traffic sign. With a higher image resolution, distant objects have more pixels and thus are easier to identify.

In academia and in industry, traffic sign detection is an active research field. The leading company in automotive computer vision is Mobileye Vision Technologies. They developed a System-on-a-chip (SoC) called EyeQ which supports applications such as lane, vehicle, traffic sign, and pedestrian detections [1]. The initial version is composed of two ARM processors and four specialized vision computing engines (VCE). One ARM CPU is responsible for chip control, communication to the vehicle, and general IO while the other ARM CPU executes computer vision algorithms which are not suited for dedicated hardware. The current architecture of EyeQ is heterogeneous with four CPU cores and several application specific processors for various ADAS algortihms [2]. Almost all of the leading automotive manufacturers cooperate with Mobileye when automotive computer vision is concerned. Traffic sign detection is also a very important research field in academia. Müller et al. propose a traffic sign recognition multi-core system based on LEON-3 processors [3]. The authors performed a design space exploration to determine what architecture fulfills their constraints. This resulted in an architecture with 2 LEON-3 processors and one hardware implemented support vector machine kernel. Most of the image processing was done on the LEON-3 processors while the classification was done on the dedicated hardware with a frame rate of 1.6 fps. In our approach, we are targeting 60 fps and aim to use only one processor, the Microblaze, for frame based image processing and dedicated hardware for pixel based pre-processing. In [4], Waite and Oruklu propose a traffic sign detection system based on hue detection and morphological filtering. The image is sent from the Microblaze to the dedicated hardware where the candidates for traffic signs are labeled. Then the Microblaze perfoms template matching on all labeled objects. The authors state that the most time consuming process is the template matching and thus limits this system to being only able to detect traffic signs within 50 feet of distance at a travel velocity of 44 miles per hour or

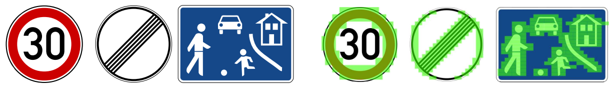

Fig. 1. Traffic Signs and their color feature detection

less. Since we aim to enable a frame rate of 60 fps with an image resolution of 1080p, we are able to detect traffic signs at a larger distance and at higher velocities. Han and Oruklu introduce a real-time traffic sign recognition system based on the Zynq FPGA [5]. They implemented a similar system to [4] on the Zynq SoC with hue detection, morphological filtering, candidate labeling, and finally template matching. They managed to increase the performance compared to [4] by the factor 8 to 96.5 ms. This still only results in a possible frame rate of 10 fps which we aim to increase by the factor 6.

Our paper consists of the following parts. Sections II and III describe the hardware and software feature extraction preprocessing steps which are needed for the traffic sign detection introduced in section IV. We conclude this paper with section V.

II. Hardware Feature Extraction

As already mentioned in section I, the feature extraction has to fulfill certain requirements. One is that this system should be able to process

$$1920 \cdot 1080 \cdot 60 \frac{1}{sec} + Blank_{horizontal} + Blank_{vertical}$$
$$= 2200 \cdot 1125 \cdot 60 \frac{1}{sec} = 148.500.000$$

pixel per second. Therefore, all hardware components should have an operation frequency of at least $f_{1080p,60fps} \geq 148,5$MHz. Another requirement is the robustness towards varying lighting conditions. When using the RGB color space, difference in lighting conditions changes every color component, thus making color classification with simple thresholding not possible. In order to eliminate the lighting dependency, the RGB color space is converted in to the YC_bC_r color space [13]. Since the Y parameter represents the luma component in the YC_bC_r color space, object detection based on color information from the C_b and C_r component is possible. An own implementation of the color space conversion is not necessary since Xilinx provides an IP core with the required functionality.

In order to detect different traffic signs, unique features of each traffic sign have to be identified. Fig. 1(a) shows several different traffic signs which need to be classified. We define a unique feature as a coherent area with one dominant color. Based on these results, thresholds for each color are determined. Since timing constraints have to be considered, the system is not able to analyze every pixel by itself. Therefore, subsampling with eight horizontal pixel is performed which also results in a low-pass filter for the respective pixel block. As seen in Fig. 1(a), the red color feature can be applied to speed limit traffic signs. The feature for the ”limits no longer apply” traffic sign is the numerous transitions between black and white pixel. These transitions also results in gray pixel directly at the edges of these transitions. Therefore, the features

TABLE I. YC_bC_r THRESHOLD VALUES FOR THE COLORS RED, BLUE, YELLOW, BLACK, GRAY, AND WHITE

Color	Y_{lo}	Y_{hi}	$C_{b,lo}$	$C_{b,hi}$	$C_{r,lo}$	$C_{r,hi}$
red	30	180	85	130	140	255
blue	30	155	85	130	140	255
yellow	90	255	30	84	145	180

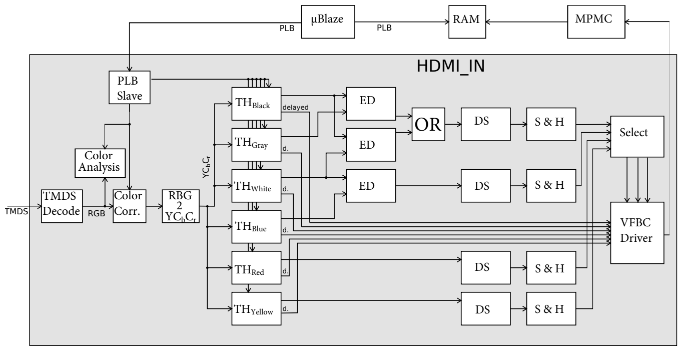

Fig. 2. Modified HDMI-IN-IP core with integrated color thresholding and feature detection

which result in the detection of this traffic sign are black-white, white-black, black-gray, and gray-black color transitions. For the last presented traffic sign in Fig. 1(a), the feature is similar to the ”limits no longer apply” traffic sign. Here, the blue-white transitions within the image have to be considered since only blue pixel might also resemble the sky. The detection results for each color feature applied to the respective signs is presented in Fig. 1(b). It can be clearly seen that all feature are able to produce coherent areas which make a unique classification possible. The blue sign is the only one which does not result in a complete coherent feature, since some blue-white transitions are embedded in a large blue field. In order to decrease the fragmentation of the feature, 16 horizontal pixel block can be used. Based on the analysis of several traffic sign images and the resulting coherent feature areas, the following thresholds are defined as seen in Table I. The table shows that the conversion from the RGB to the YC_bC_r color space has reduced the luma dependency for colors. The colors red, blue, and yellow are valid through almost all values of the Y component, but have no common threshold values when considering the C_b and C_r components combined. In the YC_bC_r color space, values are considered to be colorless when the values of C_b and C_r are in the region of 128. Therefore, the colors black, gray, and white do not have an overlap in the Y component but have almost the same value range for C_b and C_r. Still, in order to robustly detect the correct color, all color space components have to be analyzed. The goal for determining these thresholds was to cover all color values in a certain range. Since the YC_bC_r color space does not eliminate all the influence the luminance has on the colors, different colors might appear when considering the threshold borders.

All the above mentioned steps such as down sampling, color conversion, and thresholding have to be implemented into the HDMI-IN-IP core. The components of the modified HDMI-IN-IP core are shown in Fig. 2. The raw TMDS data is converted into RGB values by the ”TMDS Decode” block. The RGB values are then analyzed and based on that analysis,

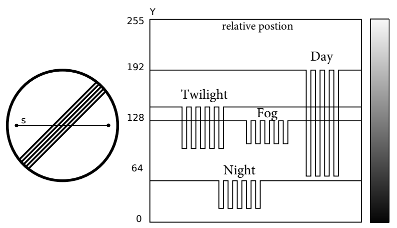

Fig. 3. Y values for the traffic sign "limits no longer apply".

color correction is performed in order to equalize the color values to the available value range. After color correction, the RBG data is converted into YC_bC_r values which are then sent to the Thresholding units, depicted as TH$_{color}$ in Fig. 2. After thresholding is performed, the color detection for each pixel is directly sent to the VFBC driver. Additionally, the colors yellow and red are sent to the down sampling unit (DS) since these color correspond directly to traffic sign features. This cannot be done for the others colors since the features are transitions between the respective colors. Therefore, an edge detection (ED) is executed before sending the respective pixel to the DS unit. As the DS unit requires 8 pixel for the feature calculation, a sample and hold unit (S & H) is needed to synchronously forward the correct value of every DS unit at the same time. The output generated from the DS units are the input values for the select unit. The select unit verifies if one feature exists in the current image and sends this information to the VFBC driver as well. The VFBC driver will then transfer the pixel information to the RAM. The Microblaze is able to access this information in the RAM and perform the traffic sign classification. The following subsections will describe each hardware components in more detail.

A. Color Analysis and Color Correction

The colors white, black and gray are identified by having their C_b and C_r values around 128. Thus, $Y > 200$ is white, $91 \leq Y \leq 200$ is gray, and $Y < 91$ is black. As already expressed in section I, our detection system should be robust when confronted with varying lighting conditions. The first step is to split the colors and the luminance into different parameters. This is done through the transformation from the RGB to the YC_bC_r color space. However, it is also difficult to distinguish black, white, and gray under different lighting conditions, as seen in Fig. 3. Here, the different Y values are shown when traversing the "no limits apply" traffic sign. During the day, Y has the largest range between white and black. However, the Y value of white during the night is even smaller than the Y value of black during the day. This makes, traffic sign detection and classification almost impossible. Therefore, a color analysis and color correction unit have been implemented before converting from the RGB to the YC_bC_r color space. The goal for both units is to perform an automatic white balance of each frame. This is usually done with histogram analysis. It is assumed that 99.9% of all pixel of a frame occupy the complete RGB color space. These 0.1% pixel have to be set to the minimum or maximum value of the color space. For this, two threshold values have to be defined for each color component for the color correction. The threshold is calculated through a count-margin unit, which is depicted in Fig. 4. Since RGB values range from 0 to 255,

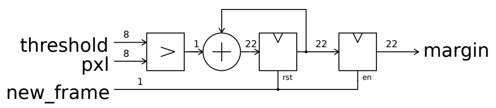

Fig. 4. Architecture of the count-margin unit

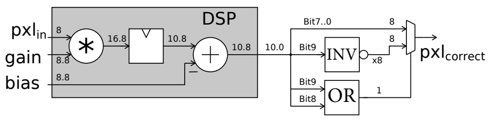

Fig. 5. Architecture of the color correction unit

8 bits are needed to compare the pixel value with a threshold value. Then, all pixel are counted which fall above or below the respective threshold. Only 0.05% of the pixel are allowed to be in either the lower or upper threshold register. One FullHD image contains $1920 \times 1080 = 2.073.600$ pixel. Therefore in the worst case, a memory with a 22 bit width is required. When the a new image frame arrives, the amount of pixel will be reseted and the number of pixel above and below the respective thresholds will be sent to the Microblaze via the PLB. The Microblaze will then recalculate the thresholds based on the current results from each count-margin unit. Additionally, the Microblaze also calculates the gain and bias which is required for the color correction unit with the equations in (2) and (3).

$$gain = \frac{256}{bound_{up} - bound_{low}} \tag{2}$$

$$bias = bound_{low} \cdot gain \tag{3}$$

The color correction function implements equation (4) and is shown in Fig.5.

$$pxl_{correct} = pxl_{in} \cdot gain - bias \tag{4}$$

The signal pxl_{in} is a 8 bit value and is multiplied with $gain$ calculated by the Microblaze. The Microblaze calculates the $gain$ and the $bias$ with 8 pre-decimal point positions and 8 decimal places, in the following represented as 8.8. The multiplication result requires 24 bit in the format 16.8. The 7 most significant bits are discarded since the value of the $gain$ ranges between 1 and 2 and therefore the multiplication cannot reach higher values than 511 resulting in a 9.8 format. The adder calculates in the two's complement format. Therefore, an additional sign bit '0' has to be appended to the output of the multiplication resulting in an 10.8 format as input for the adder unit. The bias has to be sign extended in order to fit the input of the adder and converted to its negative two's complement representation for subtraction. The 10.8 result is rounded down to 10.0 by ignoring the decimal places of the subtractions result. The described operations can be efficiently performed by one of the DSP48-slices of the SPARTAN-6. Lastly, clipping to the range of values 0 to 255 has to be performed on the subtraction's result. The three possible cases are $input > 255$, $input < 0$, and $0 < input < 255$. Due to the subtraction, negative values can occur as a result. Therefore, the sign bit (Bit 9) and the overflow bit (Bit 8) are analyzed. If one of these bits is set as '1', the negated value of the sign bit will be set as an 8 bit vector output. Otherwise the bits 7 to 0 are sent to the output. The whole color correction unit synthesizes at 324,675 MHz and thus fulfills the speed requirement for FullHD image processing.

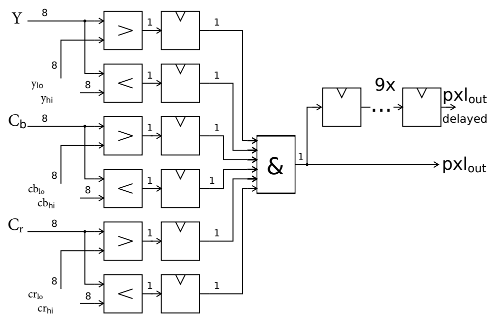

Fig. 6. Architecture of the threshold unit

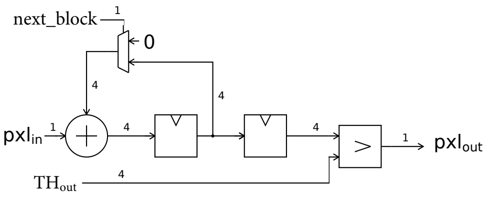

Fig. 7. Architecture of the down sampling and low-pass filter unit

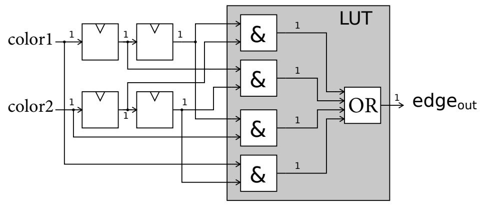

Fig. 8. Architecture of the edge detection unit

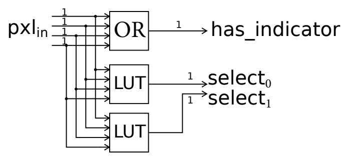

Fig. 9. Architecture of the select unit

B. Threshold Unit

After the color corrected RGB values have been converted into the YC_bC_r color space, each 24 bit YC_bC_r value has to be assigned to one of the respective colors. These upper and lower thresholds can vary for different colors. The hardware for one color threshold unit is depicted in Fig. 6. Each of the pixels color space components will be compared to the respective two threshold for the Y, C_b, and C_r parameter. Then, the 6 results of the comparisons are inserted into a LUT which functions as a 6-Input-AND. The results is then sent to the next hardware unit directly to the VFBC driver. In order to synchronously receive the results from the down sampling and low-pass filter, the pxl_{out} signal to the VFBC driver block has to be delayed by 9 clock cycles. This unit synthesizes at 502,513MHz and therefore meets the timing requirements for FullHD video processing.

C. Down Sampling Unit

The down sampling unit also incorporates the low-pass filter since they cannot be divided. As seen in Fig. 7, it sums up 8 horizontal pixel of one color and determines if the respective feature threshold has been reached. If the incoming signal pxl_{in} is '1', the counter will be incremented. When 8 pixel have been processed, the counter will get reseted by the $next\_block$ signal. A register stores the current counter value before it is compared to the down sampling threshold TH_{out}. This threshold is important for the robustness of the color feature detection since a false threshold value might result in multiple feature detections. In the case of 8 pixel, two or more color features can be valid if the threshold value is chosen poorly. Therefore, the threshold has to fulfill the following requirement.

$$TH_{out} \geq \frac{Block\ size}{2} + 1, \tag{5}$$

with $Block\ size$ being the number of horizontal pixels which are used to determine the sampling threshold. This requirement is valid for single color features, but does not hold for color transition features such as black-white or blue-white color features. Here, for a block size of 8 pixel at least 5 transitions have to be detected on the traffic sign. These 5 transitions would blur into a gray color in case of a black-white feature. Therefore, these thresholds are fixed at a threshold value of 1 or 2. Therefore, the select unit in subsection II-E employs a priority based mutual exclusion when determining the valid pixel feature. The down sampling threshold can be changed via software if needed. This component synthesizes at 330,142MHz.

D. Edge Detection Unit

Not all color features which are sent to the DS units consist of one color but consist of a transition between two colors. In an image, this is represented as an edge. Therefore, edge detection has to be performed for the color features black-white, black-gray, and blue-white. Fig. 8 shows the edge detection component. The input bits from each color feature are delayed for 2 clock cycles in order to analyze three pixel at once. This is done to reliably detect a color transition and represents a horizontal edge detection. All possible pixel combinations are compared to each other and the output signal $edge_{out}$ defines if an edge is detected or not. Since all pixel combinations require 6 inputs and one output, this logic can be synthesized as a LUT.

E. Select Unit

As seen in Fig. 9, the select component checks if one of the input signals coming from the four different features is valid i.e. has the value '1'. If more than one feature is valid, then the select unit will choose the feature with the highest priority. Since four features are available, a 2 bit priority encoder is implemented with 2 LUTs.

F. Resource Usage

Table II shows the resource usage of the implemented components on the Spartan-6 FPGA. Only 47% of all available slices are used for this design. With a usage of only 14% of the slice registers, the design is not very resource hungry in this regard. When regarding the number of occupied slice LUTs, 30% is relatively high in comparison. This is because most of the components can be easily designed without fully utilized

TABLE II. RESOURCE UTILIZATION OF THE TRAFFIC SIGN DETECTION ON THE SPARTAN-6 FPGA

Resources	Used	Available	Utilization
No. of occupied Slices	3.260	6.822	47%
Slice Registers	7962	54.576	14%
Slice LUTs	8.244	27.288	30%
DSP48A	10	58	17%
BRAM	19	116	16%

LUTs. With further optimizations, this number can be reduced. The resource consumption of 17% and 16% for DSP48 and BRAM blocks respectively is acceptable, especially since the Microblaze is also included in this design.

III. SOFTWARE FEATURE EXTRACTION

The task of the software feature extraction is to determine coherent feature regions based on the extracted pixel information. This is done with the flood algorithm [14]. Since the positions of the detected objects are also of interest for further image processing, the horizontal and vertical minimum and maximum value of the respective feature region is also saved. These values represent a bounding box and will be saved in memory, as soon as the stack is empty. As already mentioned in section **??**, the first byte which is transferred to the DDR memory is reserved for the flood algorithm. The algorithm reads the first byte and determines, if the respective pixel has a feature. This is represented by the byte value ’1’. If no feature is detected for the pixel, the value is ’0’. The flood algorithm now tries to detect objects out of regions of features. If such a region qualifies as an object, the first byte of every pixel in this region is assigned the respective object number starting at ’2’. This makes the tracking of up to 253 objects possible. In order to not always reach the maximum number of detectable objects in an image, smaller objects and objects with the wrong proportions will be discarded. This step is executed with traffic sign classification step which is introduced in section IV. Since the target frame rate is 60fps, the flood algorithm for the complete image should be processed within a fraction of the time slot of $t_{slot} = \frac{1}{60} sec \approx 16.66 msec$. Since accessing every pixel of a FullHD image requires too much time, the image is divided into 8×8 pixel blocks. This results to $1920/8 \cdot 1080/8 = 32400$ pixel which are analyzedby the flood algorithm. Without optimizations, the flood algorithm requires $1,5 \cdot t_{slot}$ in an empty image. In order to reduce execution time, horizontally as well as vertically only every even pixel block is analyzed, resulting in only 25% of memory accesses in an empty image and thus reducing the execution time of the algorithm to $0,4 \cdot t_{slot}$ in an empty image. Activating compiler additional optimizations like saving variables in registers also results in execution time reduction. With all optimizations, an execution time of $0,3 \cdot t_{slot}$ is required for an empty image.

IV. TRAFFIC SIGN CLASSIFICATION

The previous steps determined objects of interest in an image. Now, these object have assigned to the respective traffic signs. Fortunately, the features of each traffic sign are unique so that a preselection can take place.

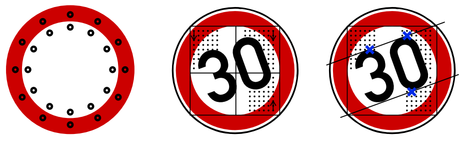
(a) Checking if the detected object resembles a circle (b) Determining the rotation angle of a speed limit sign. The three starting positions are the upper left, upper right, and lower right corner of the bounding box.

Fig. 10. Steps in traffic sign classification

A. *Red $\rightarrow$ Speed limit*

As already mentioned in section III, objects with the wrong proportions have to be discarded so that red cars or red lights will be detected as potential traffic sign candidates. Therefore, all objects consisting of more than 200 and less than 10 pixel blocks are ignored.Furthermore, objects where the width/height is larger than the double height/width are also ignored since these object are likely too distorted to be analyzed and classified correctly. In section III, a bounding box was saved in memory for each object. In order to extract detailed information about the content of the traffic sign, the white inner part has to separated from the red outer part. This is done with two ellipses which are drawn inside the respective bounding box. An ellipse inside a rectangle with the dimensions $2a \cdot 2b$ can be described with the two equations

$$\begin{aligned} x &= a \cdot cos(\alpha) \\ y &= b \cdot sin(\alpha). \end{aligned} \tag{6}$$

The generation of two ellipses inside one bounding box is executed by multiplying the x and y coordinates with either the scale factor 1,08 or 0,95. 12 points of each ellipse will be calculated and analyzed further. Figure 10(a) shows the resulting two ellipses inside the speed limit traffic sign. The corresponding pixel to the 12 points which belong to the outer ellipse should all be red. The pixel belonging to the inner ellipse should have a non-red color. However, small deviations are acceptable so that the threshold for non-red color currently has been set to 83%. The scale factors have been chosen based on several measurements with traffic signs with different orientations. The critical information of the speed limit sign is still inside the white region. Therefore, a more accurate bounding box is calculated for the white region of the traffic sign. This is done by jumping 25% of the edge length into the white region and traversing to the outer region pixel by pixel until a red pixel is reached. Starting points for these four jumps are the middle of the respective bounding box edges. Now, the various digits have to be detected in the more accurate bounding box. One of the challenges is the different appearances of speed limit traffic signs such as compressed, shifted, or rotated digits These various traffic signs can be detected by first determining the rotation, number, and type of digits. The angle of the digit rotation can be determined by searching for the first black pixel inside the bounding box with three different starting points. The process of determining the angle is shown in Fig. 10(b). The three starting points are depicted as $\downarrow$ for the upper corners and $\uparrow$ for the remaining lower right corner. For each starting point, the bounding box is analyzed column wise until the first black pixel of the

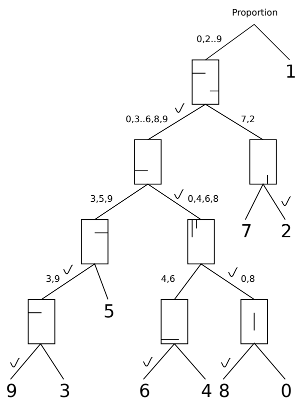

Fig. 11. The binary decision tree for digit detection. The depicted pixel positions will be examined. If a black pixel is detected at the respective position, the path with $\surd$ is followed.

respective column is found. The position of the black pixel is defined as the distance to the upper part of the bounding box. This procedure is done until the distance increases again. The last pixel position is then considered to be the highest pixel of the digit. For the other two remaining starting positions, this is done analogous. Both upper points describe a line which intersects with one border of the bounding box. This angle resembles the rotation of the digits. With this angle, a rectangle which is aligned to the digits can be defined and is used for the following digit detection. The maximum number of digits which can occur on a traffic sign is 3. Between each digit, there must be at least one column with non-black pixel followed by a columns with black pixel. This 'white' column then resembles the start of the next digit if the maximum number of digits is not exceeded. If that is the case, this information is ignored. In these boundaries, digit detection is performed. The digit detection follows a binary decision tree, depicted in Fig. 11 The first step is to check the proportion of the digit. If it is much taller than it is wide, the digit is detected as '1'. Otherwise, the pixels will be examined at the positions defined in the respective nodes of the tree. If a pixel is detected as black at the specified position, then path with $\surd$ is followed. When a non-black pixel is present at the position, the other path is followed. For the first and last digit, not all paths have to be checked, since the available digits are very limited. For instance, the first digit can only be a '1', while the last digit usually either is a '0' or a '5'. In these cases, the number of comparisons can be reduced significantly. As a whole, the complete traffic sign classification requires not more than $6\% \cdot t_{slot}$ which fully meets the requirements of the system.

V. Conclusion

This paper introduces an efficient traffic sign detection as driver assistance system. The complete system was designed and implemented on a Spartan-6-FPGA with the requirement to be able to process FullHD video streams. A great advantage of a FPGA compared to a general purpose processor is the focus and optimization on one task. This enables the reduction of memory accesses when converting an image from one color space to another since these conversion can be performed on the data stream. Additionally, FPGAs, especially the Spartan-6 since it is approved for the automotive domain, provide a low cost solution for designing and implementing future ADAS which is very important in the automotive domain. The methods which were developed in this paper aim to enable the autonomous driving but can also be adapted to other autonomous robotic applications such as health care, agriculture, and space exploration. In these cases, the features just have to defined differently and the classification needs to be adapted. While the performance of the designed system is satisfactory, several extensions have to be implemented for an actual field test in automobiles. The system design also allows the detection of other objects for future projects and applications.

Acknowledgment

This research has been supported by the Horizon 2020 grant for RADIO, Project Number 643892.

References

[1] G. Stein, E. Rushinek, G. Hayun, and A. Shashua, "A computer vision system on a chip: a case study from the automotive domain," in *Computer Vision and Pattern Recognition - Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, June 2005, pp. 130–130.

[2] M. Newsletter, "Moving closer to automated driving, mobileye unveils eyeq4 system-on-chip with its first design win for 2018," online, Mobileye Vision Technologies Ltd., Mar. 2015. [Online]. Available: www.mobileye.com/blog/press-room/moving-closer-automated-driving-mobileye-unveils-eyeq4-system-chip-first-design-win-2018/

[3] M. Müller, A. Braun, J. Gerlach, W. Rosenstiel, D. Nienhuser, J. Zollner, and O. Bringmann, "Design of an automotive traffic sign recognition system targeting a multi-core soc implementation," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, March 2010, pp. 532–537.

[4] S. Waite and E. Oruklu, "Fpga-based traffic sign recognition for advanced driver assistance systems," *Journal of Transportation Technologies*, vol. 3, no. 1, pp. 1–12, Jan. 2013.

[5] Y. Han and E. Oruklu, "Real-time traffic sign recognition based on zynq fpga and arm socs," in *Electro/Information Technology (EIT), 2014 IEEE International Conference on*, June 2014, pp. 373–376.

[6] *Atlys Board Reference Manual*, Digilent Inc., 1300 Henley Court, Pullman, WA 99163, Aug. 2013. [Online]. Available: www.digilentinc.com/data/products/atlys/atlys_rm_v2.pdf

[7] *XA Spartan-6 Automotive FPGA Family Overview*, Xilinx Inc., Dec. 2012. [Online]. Available: www.xilinx.com /support/documentation/-data_sheets/ds170.pdf

[8] *LogiCORE IP Processor Local Bus (PLB) v4.6 (v1.05a)*, Xilinx Inc., Sep. 2010. [Online]. Available: www.xilinx.com /support/documentation/ip_documentation/plb_v46.pdf

[9] *LogiCORE IP Local Memory Bus (LMB) v3.0*, Xilinx Inc., Mar. 2013. [Online]. Available: www.xilinx.com /support/documentation/ip_documentation/lmb_v10/v3_0/pg113-lmb-v10.pdf

[10] B. Feng, "Implementing a tmds video interface in the spartan-6 fpga," Xilinx Inc., Tech. Rep., Dec. 2010. [Online]. Available: www.xilinx.com /support/documentation/application_notes/xapp495_S6TMDS_Video_Interface.pdf

[11] D. Phanthavong and J. Ou, "Implementing a video frame buffer controller (vfbc) in system generator," Xilinx Inc., Tech. Rep., Jun. 2009. [Online]. Available: www.xilinx.com /support/documentation/application_notes/xapp1136.pdf

[12] *LogiCORE IP Multi-Port Memory Controller (MPMC) (v.6.03.a)*, Xilinx Inc., Mar. 2011. [Online]. Available: www.xilinx.com /support/documentation/ip_documentation/mpmc.pdf

[13] W. K. Pratt, *Digital Image Processing*, 3rd ed. John Wiley& Sons, Inc., 2001, ch. 3, pp. 45–88.

[14] R. Khudeev, "A new flood-fill algorithm for closed contour," in *Control and Communications, 2005. SIBCON '05. IEEE International Siberian Conference on*, Oct 2005, pp. 172–176.