A System-On-Chip FPGA Design for Real-Time Traffic Signal Recognition System

Yuteng Zhou, Zhilu Chen, and Xinming Huang
Department of Electrical and Computer Engineering, Worcester Polytechnic Institute, MA 01609, USA

***Abstract*—Traffic signal detection has long been an important function in an advanced driver assistance system (ADAS). This paper presents a complete system design based on the techniques of blob detection, histogram of oriented gradients (HOG) and support vector machine (SVM). Blob detection is applied to detect potential candidates, and then HOG and SVM is for feature classification. A novel hardware/software co-design architecture is developed for traffic light recognition at real-time. With well-balanced workload on FPGA fabric and the on-chip ARM processor, the entire system-on-chip can achieve a processing rate of 60 fps for XGA 1024-by-768 video. The system can achieve an accuracy rate of over 90% on both red lights and green lights. The proposed system can be improved by replacing HOG with more advanced feature algorithm to obtain higher accuracy.**

***Index Terms*—ADAS, traffic signal, real-time, blob detection, HOG, SVM, FPGA, system-on-chip**

I. Introduction

Traffic signal recognition is an important feature in advanced driver assistance systems (ADAS) and self-driving vehicles [1]. Similar to traffic sign detection methods [2], vision-based solutions are popular for traffic light detection. The key challenge is to implement the algorithms for real-time processing, which is then meaningful to drivers, and to achieve a high detection rate at the same time [3]. A typical approach is to first detect potential blobs first, and then differentiate true traffic lights from false positives [4] [5]. Previously, FPGA-based platforms have been widely used for the implementations of real-time image processing and computer vision. Similar works such as recognizing traffic signs has already been implemented on FPGA, achieving a tremendous speedup and significant lower power consumption comparing to the software solutions on a general purpose CPU [6].

In this paper, we propose a complete system-on-chip (SOC) FPGA design with balanced workload on hardware and software. We adopt the traditional detection and classification approach [7] [2]. The detection part requires pre-filtering mainly to eliminate pixels unlikely to be part of a traffic light. The system is targeted to detect both green lights and red lights, so color filter is employed to separate them into two processing branches. Then we apply blob detection to locate the possible traffic light objects. For object recognition, we use histogram of oriented gradients (HOG) to extract shape features and then apply support vector machine (SVM) as the classifier.

In this work, detection is implemented on the FPGA fabric and classification is implemented on the on-chip ARM processor. The system implementation supports a frame rate of 60 frames per second (fps), while still attaining a high detection rate over 90% on both green and red traffic lights. The rest of this paper is organized as follows. Section II presents methodologies for traffic light detection and classification. Section III presents detailed implementation on the SOC FPGA. Section IV presents the experimental results and finally Section V concludes the paper.

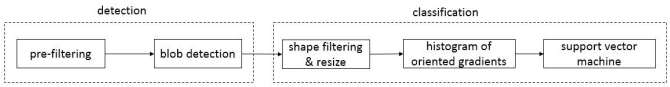

Figure 1. Overall system diagram with detection and classification

II. Methods for Traffic Light Detection and Classification

The input to the FPGA system is the real-time video stream at 60 fps captured by an RGB camera mounted on a car. We first divide the system into two parts, detection and classification as shown in Fig. 1. The detection part consists of pre-filtering which extracts green pixels and red pixels. After that, blob detection is applied to estimate the positions each blob. After detection, all the potential green blobs for green traffic lights and red blobs for red traffic lights are obtained including many false positives. Since a typical traffic light's length-width ratio is between 1/4 to 4, blobs with odd length-width ratio are eliminated. Then each potential blob is resized to 32-by-32 pixels. Furthermore, the HOG algorithm extracts 324 features from each blob that are then fed to the SVM classfier. The result of SVM decides whether current blob is a traffic light or not. By rapidly scanning through every blob on an image, we are able to detect all red and green traffic lights at the current scene.

Another feature we have to consider is to make the entire system working in real-time. Since our system is used to warn drivers while red traffic lights are on, consider the typical human response time is about 0.26 second [8], which means our system has to process images no slower than 4 fps to be meaningful. So a hardware/software co-design architecture becomes necessary, which implements computationally heavy tasks on FPGA fabric and at the same time maintains high-speed data exchange with the embedded processor. The hardware/software partition strategy and FPGA implementation are explained in details in Section III.

978-1-4799-5341-7/16/$31.00 ©2016 IEEE

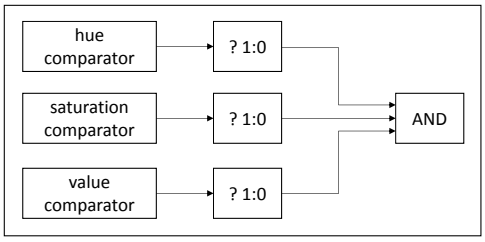

Figure 2. Apply HSV threshold to obtain binary images

A. Pre-filtering

The input to our system is in RGB format. Each pixel is represented by 3 bytes, with each byte representing one color channel. Due to changing luminance and varied weather conditions on road, a pixel appearing green to human eyes does not necessarily indicate a large absolute value in green channel, because it also relies on values of other two color channels. The disadvantage of RGB color space is that it cannot reflect the relations among red, green and blue. As the first step in pre-filtering, we convert RGB to HSV color space. HSV is a cylindrical-coordinate representations of colorful pixels, representing relationships between each color channel [9]. HSV stands for hue, saturation, and value. In HSV color domain, green and red colors can be easily picked out by setting proper thresholds.

Equations below show the pixel format conversions from RGB to HSV [10]:

$$H = \begin{cases} 60 \times \frac{G-B}{MAX-MIN} + 0 \ (if\ MAX = R) \\ 60 \times \frac{B-R}{MAX-MIN} + 120 \ (if\ MAX = G) \\ 60 \times \frac{R-G}{MAX-MIN} + 240 \ (if\ MAX = B) \end{cases} \quad (1)$$

$$S = MAX - MIN \quad (2)$$

$$V = MAX \quad (3)$$

As the last step in pre-filtering, a single pixel is binarized. For instance, value is 1 if pixel is considered green, otherwise 0, as indicated in Fig. 2. The same process is repeated for red pixels in parallel.

B. One-Pass Blob Detection

Blob detection collects connected pixels from pre-filtering step. The principal idea is to label different clusters of pixels to different values on the entire image. Here we use 4-connectivity to determine whether pixels are connected or not. 4-connectivity means, for center pixel, only 4 pixels (N,E,W,S) are considered to be its neighbors. For the purpose of high efficiency, one-pass labeling is utilized. We are able to output all the potential blobs by scanning through the entire image only once. More details on one-pass implementation is explained in Section 3.2.

Figure 3. Examples of some typical traffic lights

Figure 4. Diagram of HOG computation procedure

C. HOG Algorithm

Standard traffic lights have fixed length-width ratio from 1/4 to 4 as indicated in Fig. 3. Blobs with too large or too small length-width ratio can be eliminated. Prior to computing HOG, each blob is resized to 32-by-32 pixels. Then each input image is firstly divided into blocks. A block size is 16 * 16 pixels, containing 4 cells with each cell size 8 * 8 pixels. Next the block starts sliding horizontally and then vertically, with a step size of 8 pixels. This results in a total of 9 blocks on a 32-by-32 image.

The HOG computation typically consists of three steps. They are weighted magnitude and bin class calculation, block histogram generation, normalization as illustrated by Fig. 3.

As the first step, gradients of each pixel in both x and y directions are computed:

$$G_x(x,y) = |M_x(x+1,y) - M_x(x-1,y)| \quad (4)$$

$$G_y(x,y) = |M_y(x,y+1) - M_y(x,y-1)| \quad (5)$$

Then, the gradient magnitude and the gradient angle can be calculated:

$$G(x,y) = \sqrt{G_x(x,y) + G_y(x,y)} \quad (6)$$

$$\theta = \arctan \frac{G_y(x,y)}{G_x(x,y)} \quad (7)$$

The gradient magnitude is further divided into 9 different bin classes. According to angle value with range 0-180 degrees, every 20 degrees represent one bin class. For each cell, a block histogram is generated by summing up the weighted magnitudes for the corresponding bin class, resulting in 9 feature descriptors in one cell. For the whole image, there are 324 feature descriptors in total.

The last step of normalization makes algorithm more robust to varied illuminations.

$$b_{norm} = \sqrt{\frac{b}{sum(b)}} \quad (8)$$

D. Linear SVM

Linear SVM maps input non-linear descriptors to higher dimension feature space, then a linear decision surface can be constructed [11]. The linear SVM is expressed in (9).

$$Y = \alpha y^T + \gamma \tag{9}$$

where α is the support vector, y is HOG feature descriptors vector, and γ is the SVM offset. In our work, support vector α and SVM offset γ is pre-trained using labeled traffic light samples. The result of (9) indicates whether a target blob contains a traffic light or not.

III. SOC FPGA Implementation

A. Software/Hardware Co-Design

To implement the whole system on SOC FPGA, delicate division of software and hardware is required. In this application, scanning through the whole image is quite computationally heavy, so the detection part must be implemented on FPGA fabric. Images after blob detection are no longer original images, and only blob images are needed for HOG computation. As shown in Fig. 5, we divide the input data streaming into two paths: one goes through blob detection and the other retains the original image. As to HOG and SVM, experiments in Section IV shows it takes less than 10ms to process a single image, proving the feasibility of our partition between hardware and software.

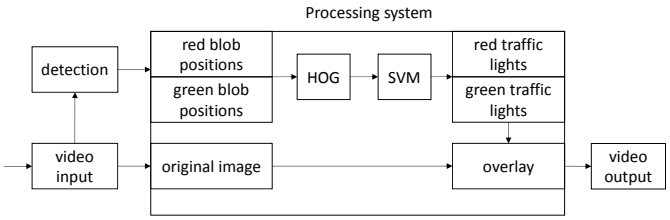

Figure 5. Hardware/sofwtare partition on SOC

B. Pipeline Structure for Detection

As described in Section 2.1, detection part consists of color conversion and blob detection. Fig. 6 shows the hardware architecture of detection part on the FPGA fabric. Since two types of traffic lights are to be recognized simultaneously, two blob detection blocks are used. Also, we implement one-pass blob detection in order to achieve a frame rate of 60 fps.

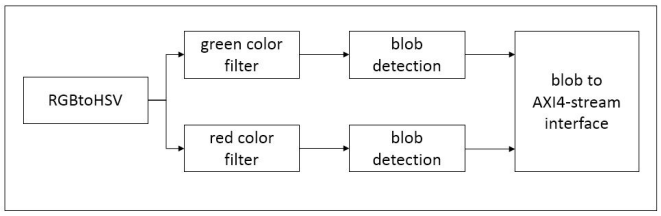

Figure 6. Hardware architecture of detection part using FPGA fabric

For the implementation of blob detection algorithm on FPGA, blob position table is required which records positions of each blob detected. Shown in Fig. 7, there is a label counter keeping track of current label number - each time a new blob is detected, label counter adds its value by 1. The blob position table is made up of 4 memory blocks, recording 4 vertices of every blob. For a specific blob with label number n, its position information is stored at nth slot in each of these 4 memory blocks.

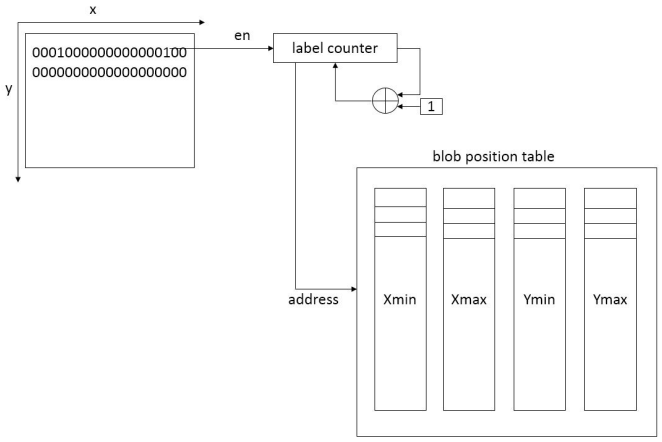

Figure 7. Store positions for different labels in blob position table

The main difference between one-pass blob detection and multi-pass detection is that one-pass has an extra connection table which checks if two different labels actually indicates a common blob. In this way, all the connected labels information are stored into the connection table as shown in Fig. 8. We do not need to scan through the entire image any more. As an example in Fig. 8, when labeling the center pixel to be 5, connection label logic knows label 7 and label 5 are indeed in the same blob, so value 5 is written to the7th memory slot in the connection table.

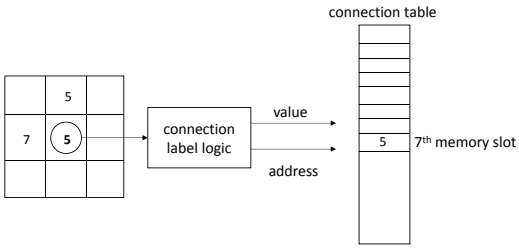

Figure 8. Connection table keeps record of connecting labels

After scanning through the whole image, all the information are stored into the connection table and blob position table. We further merge position information of same blobs as shown in Fig. 9. The connection table indicates which labels are to be merged, and we can update position information in blob position table accordingly.

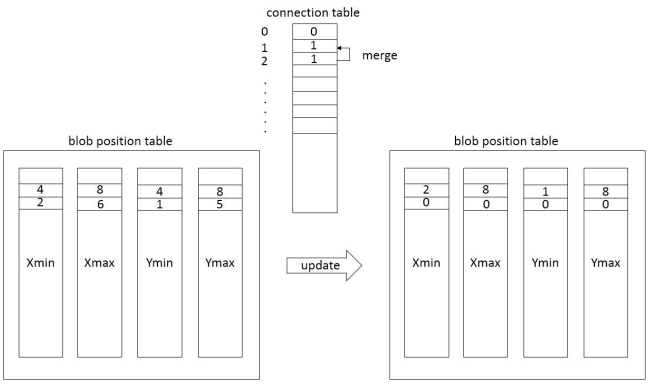

Figure 9. Merging information in blob position table

C. Blob to AXI4-Stream Interface

The interface is employed to transfer the blob position information onto AXI4-stream bus, along with a video DMA, realizing high-speed from FPGA to frame buffers in DDR memory. Subsequently, the on-chip embdded ARM processor can access image frames from DDR with a very high pixel rate.

IV. Onboard Implementation Result

We implement the entire system on Xilinx Zynq ZC-702 board. The input video resolution is 1024×768 XGA. Highest frequency of FPGA implementation reaches 147.34MHz, so higher resolutions can also be supported. Overall FPGA utilization is shown in Table 1. Since input data streaming rate is at 60 fps, our one-pass blob detection is adequate to follow such a rate for object detection. We also measure the time taken by the processor to classify all the traffic light candidates on a single frame. The time varies from 1.96ms to 9.66ms, orless than 10ms all the time. For the embedded ARM processor, we are able to achieve over 100 fps performance.

Table I
FPGA resource utilization

	Used	Available	utilization
Slice Registers	47656	106400	44.78%
Slice LUTs	49183	53200	92.44%
DSP48E1s	4	220	1.81%
Block RAM	25	140	17.85%

Also, we test our system through 10 video clips recorded in different road and weather conditions. A sample image is shown in Fig. 10. Table 2 shows we have achieved a high recall and precision rate. Equations are given:

$$recall = \frac{true\,positives}{true\,positives + false\,negatives} \tag{10}$$

$$precision = \frac{true\,positives}{true\,positives + false\,positives} \tag{11}$$

Figure 10. Green traffic lights is detected by the proposed system at real-time

Table II
Detection Accuracy

	Recall	Precision
Red traffic lights	92.11%	99.29%
Green traffic lights	94.44%	98.27%

V. Conclusion

In this paper, we present an FPGA based SOC design for real-time traffic light recognition. We successfully implement the entire system on Xilinx Zynq board, achieving real-time processing rate of 60 fps and beyond. With the advent of deep learning network, it is likely to obtain a higher detection rate by replacing HOG algorithm with stronger feature extractors.

References

[1] R. Okuda, Y. Kajiwara, and K. Terashima, "A survey of technical trend of adas and autonomous driving," in *VLSI Technology, Systems and Application (VLSI-TSA), Proceedings of Technical Program-2014 International Symposium on*. IEEE, 2014, pp. 1–4.

[2] A. Møgelmose, M. M. Trivedi, and T. B. Moeslund, "Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 13, no. 4, pp. 1484–1497, 2012.

[3] Y.-C. Chung, J.-M. Wang, and S.-W. Chen, "A vision-based traffic light detection system at intersections," *Journal of Taiwan Normal University: Mathematics, Science and Technology*, vol. 47, no. 1, pp. 67–86, 2002.

[4] M. Omachi and S. Omachi, "Traffic light detection with color and edge information," in *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*. IEEE, 2009, pp. 284–287.

[5] Y. Shen, U. Ozguner, K. Redmill, and J. Liu, "A robust video based traffic light detection algorithm for intelligent vehicles," in *Intelligent Vehicles Symposium, 2009 IEEE*. IEEE, 2009, pp. 521–526.

[6] Y. Zhou, Z. Chen, and X. Huang, "A pipeline architecture for traffic sign classification on an fpga," in *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*, May 2015, pp. 950–953.

[7] J. V. Gomes, P. R. Inácio, M. Pereira, M. M. Freire, and P. P. Monteiro, "Detection and classification of peer-to-peer traffic: A survey," *ACM Computing Surveys (CSUR)*, vol. 45, no. 3, p. 30, 2013.

[8] "Reaction time statistics." [Online]. Available: http://www.humanbenchmark.com/tests/reactiontime/statistics

[9] "Hsl and hsv." [Online]. Available: https://en.wikipedia.org/wiki/HSL_and_HSV

[10] T. Hamachi, H. Tanabe, and A. Yamawaki, "Development of a generic rgb to hsv hardware," in *The 1st International Conference on Industrial Application Engineering 2013 (ICIAE2013)*, 2013.

[11] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.