Computer Vision System-On-Chip Designs for Intelligent Vehicles

by

Yuteng Zhou

A Dissertation
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy
in
Electrical and Computer Engineering

April 2018

APPROVED:

Prof. Xinming Huang, Major Adviser, ECE Department, WPI

Prof. Jie Fu, Dissertation Committee, ECE Department, WPI

Prof. Lifeng Lai, Dissertation Committee, ECE Department, UC Davis

Abstract

Intelligent vehicle technologies are growing rapidly that can enhance road safety, improve transport efficiency, and aid driver operations through sensors and intelligence. Advanced driver assistance system (ADAS) is a common platform of intelligent vehicle technologies. Many sensors like LiDAR, radar, cameras have been deployed on intelligent vehicles. Among these sensors, optical cameras are most widely used due to their low costs and easy installation. However, most computer vision algorithms are complicated and computationally slow, making them difficult to be deployed on power constraint systems. This dissertation investigates several mainstream ADAS applications, and proposes corresponding efficient digital circuits implementations for these applications. This dissertation presents three ways of software / hardware algorithm division for three ADAS applications: lane detection, traffic sign classification, and traffic light detection. Using FPGA to offload critical parts of the algorithm, the entire computer vision system is able to run in real time while maintaining a low power consumption and a high detection rate. Catching up with the advent of deep learning in the field of computer vision, we also present two deep learning based hardware implementations on application specific integrated circuits (ASIC) to achieve even lower power consumption and higher accuracy.

The real time lane detection system is implemented on Xilinx Zynq platform, which has a dual core ARM processor and FPGA fabric. The Xilinx Zynq platform integrates the software programmability of an ARM processor with the hardware programmability of an FPGA. For the lane detection task, the FPGA handles the majority of the task: region-of-interest extraction, edge detection, image binarization, and hough transform. After then, the ARM processor takes in hough transform results

and highlights lanes using the hough peaks algorithm. The entire system is able to process 1080P video stream at a constant speed of 69.4 frames per second, realizing real time capability.

An efficient system-on-chip (SOC) design which classifies up to 48 traffic signs in real time is presented in this dissertation. The traditional histogram of oriented gradients (HoG) and support vector machine (SVM) are proven to be very effective on traffic sign classification with an average accuracy rate of 93.77%. For traffic sign classification, the biggest challenge comes from the low execution efficiency of the HoG on embedded processors. By dividing the HoG algorithm into three fully pipelined stages, as well as leveraging extra on-chip memory to store intermediate results, we successfully achieved a throughput of 115.7 frames per second at 1080P resolution. The proposed generic HoG hardware implementation could also be used as an individual IP core by other computer vision systems.

A real time traffic signal detection system is implemented to present an efficient hardware implementation of the traditional grass-fire blob detection. The traditional grass-fire blob detection method iterates the input image multiple times to calculate connected blobs. In digital circuits, five extra on-chip block memories are utilized to save intermediate results. By using additional memories, all connected blob information could be obtained through one-pass image traverse. The proposed hardware friendly blob detection can run at 72.4 frames per second with 1080P video input. Applying HoG + SVM as feature extractor and classifier, 92.11% recall rate and 99.29% precision rate are obtained on red lights, and 94.44% recall rate and 98.27% precision rate on green lights.

Nowadays, convolutional neural network (CNN) is revolutionizing computer vision due to learnable layer by layer feature extraction. However, when coming into

inference, CNNs are usually slow to train and slow to execute. In this dissertation, we studied the implementation of principal component analysis based network (PCANet), which strikes a balance between algorithm robustness and computational complexity. Compared to a regular CNN, the PCANet only needs one iteration training, and typically at most has a few tens convolutions on a single layer. Compared to hand-crafted features extraction methods, the PCANet algorithm well reflects the variance in the training dataset and can better adapt to difficult conditions. The PCANet algorithm achieves accuracy rates of 96.8% and 93.1% on road marking detection and traffic light detection, respectively. Implementing in Synopsys 32nm process technology, the proposed chip can classify 724,743 32-by-32 image candidates in one second, with only 0.5 watt power consumption.

In this dissertation, binary neural network (BNN) is adopted as a potential detector for intelligent vehicles. The BNN constrains all activations and weights to be +1 or -1. Compared to a CNN with the same network configuration, the BNN achieves 50 times better resource usage with only 1% - 2% accuracy loss. Taking car detection and pedestrian detection as examples, the BNN achieves an average accuracy rate of over 95%. Furthermore, a BNN accelerator implemented in Synopsys 32nm process technology is presented in our work. The elastic architecture of the BNN accelerator makes it able to process any number of convolutional layers with high throughput. The BNN accelerator only consumes 0.6 watt and doesn't rely on external memory for storage.

Acknowledgements

First of all, I would like to express my deepest gratitude for my adviser Professor Xinming Huang for his guidance through all stages of my Ph.D. studies and research at Worcester Polytechnic Institute. I am very grateful for his great inspiration, patience, recognition and support that have helped me overcome difficulties and finally made this dissertation possible.

I would like to thank Professor Lifeng Lai and Professor Jie Fu for their insightful comments as my dissertation committee.

Also, I deeply appreciate the support from The MathWorks fellowship to WPI. Thanks for the help from my manager Bharath Venkataraman, Don Orofino and group members.

At last, I wish to thank all my lab colleagues for their friendship and support.

This dissertation is dedicated to my parents, who always give me their love, support and encouragement during all my life.

Contents

List of Tables

List of Figures

List of Abbreviations

ADAS Advanced Driver Assistance System

AMBA Advanced Microcontroller Bus Architecture

ASIC Application Specific Integrated Circuit

BelgiumTS Belgium Traffic Sign Dataset

BING Binarized Normed Gradients for Objectness

BNN Binary Neural Network

CMOS Complementary Metal-Oxide-Semiconductor

CNN Convolutional Neural Network

DSP Digital Signal Processing

FPGA Field Programmable Gate Array

GIS Geographic Information Systems

GPS Global Positioning System

GPU Graphics Processing Unit

HOG Histogram of Oriented Gradients

ICF Integral Channel Feature

LDW Lane Departure Warning

LiDAR Light Detection and Ranging

ROI Region of Interest

ScatNet Wavelet Scattering Network

SOC System-On-Chip

SURF Speeded Up Robust Features

SVM Support Vector Machine

VLSI Very Large Scale Integrated Circuits

Chapter 1

Introduction

In this chapter, we first introduce the background and discuss the motivations of our work in Section 1.1. The major contributions of the thesis are summarized in Section 1.2. Finally, the organization of this dissertation is presented in Section 1.3.

1.1 Motivations

Nearly 1.3 million people die in road crashes each year, on average 3,287 deaths a day. An additional 20-50 million are injured or disabled. Road safety issue has been raised as a major threat to us human beings. This stimulates lots of research on advanced driver assistance systems (ADAS). The major issue we try to address in this thesis is to provide reliable, feasible, real time ADAS for a lower accident rate.

In recent years, many industrial and academic research efforts have been focused on designing intelligent vehicles. Intelligent vehicles usually need a sophisticated fusion of sensors such as LiDAR, radar, and cameras. Among these sensors, optical cameras are most widely used because of their low costs and easy installation. More-

over, cameras perceive scenes in a similar way to human eyes. Hence, vision based solutions are usually more reliable especially on recognizing man made patterns like traffic signs. Also, due to the rapid development of deep learning in computer vision, vision based algorithms have become more accurate and more robust in various driving environments [1]. Current trends in ADAS design is to adopt deep learning methods into the system.

Deep learning is gradually replacing traditional computer vision algorithms. Classical computer vision algorithms such as Viola-Jones face detector [2], scale-invariant feature transform (SIFT) [3], speeded up robust features (SURF) [4], histogram of oriented gradients (HoG) [5] have been outperformed by deep learning algorithms. Deep learning algorithms, especially deep convolutional neural networks (CNN) are very suitable for object detection and object recognition. For example, in the ImageNet large scale visual recognition challenge (ILSVRC) [6], deep CNN models such as AlexNet [7], VGGNet [8], GoogleNet [9], and ResNet [10] are achieving higher accuracy year by year.

With the popularity of deep learning, a lot of vision based solutions for intelligent vehicles have been proposed, such as traffic light recognition [11], traffic sign recognition [12], vehicle detection [13], and pedestrian detection [14]. However, most of these solutions fail to work in real time. Typically, 30fps (frames per second) speed is considered to be real time for an autonomous car [15]. A processing speed less than 30fps cannot guarantee enough response time for driver assistance.

In this thesis, we are motivated to design real time, low power, vision based solutions for intelligent vehicles. We investigated the realization of embedded vision systems for ADAS applications like traffic sign classification and traffic light detection. Building a real world ADAS system is challenging: designers need to consider multiple

factors like robustness, system response time, reliability, power consumption [16]. Executing state-of-the-art algorithms on a moving vehicle demands strong parallel computing capability. In real world ADAS applications, software and hardware co-design is often needed in that software is suitable for high level processing and control, while hardware is suitable for low level processing and parallel computing. In this thesis, we are motivated to design scalable system-on-chip (SOC) architectures of complete computer vision systems for ADAS. Furthermore, we are also motivated to design efficient architecture of deep learning models on very large scale integrated circuits (VLSI), which provides lower power consumption and higher throughput.

1.2 Summary of Contributions

We design and implement efficient digital circuits for ADAS applications. Our contributions are summarized as follows:

- Propose a real time lane detection system on SOC FPGA.

  As an early step in the ADAS system, lane detection has been studied for decades. Two common approaches to the lane detection problem are perspective transform and hough transform. In this dissertation, we propose an efficient implementation of the hough transform algorithm in FPGA fabric. By leveraging the property of sine and cosine operations, 50% of multipliers are saved in digital circuits. Implementing on the Xilinx Zynq platform, the system reaches a maximum operating frequency of 143.85MHz. Our proposed hough transform realization is generic and could be used as an individual IP core in other computer vision systems as well.

  We emphasize our contributions on the algorithm and system design in the

following aspects:

  - We propose hough transform for the lane detection problem.
  - We design a resource saving hardware architecture for hough transform, which saves 50% of multiplier resources on FPGA.

- Propose an efficient SOC architecture for traffic sign classification.

  As a classical feature extraction method, the HOG algorithm has an outstanding performance in applications like pedestrian detection [17]. Usually, the HOG algorithm is paired with SVM. SVM is a classical machine learning technique which can map data onto higher dimensional feature space and quickly find a separating boundary. Traffic sign classification is an important task in ADAS. In this dissertation, we experimented the HOG + SVM algorithms on traffic sign classification and obtained high classification rate. Furthermore, we accelerated the HOG and SVM algorithms in FPGA fabric, achieving a maximum operating frequency of 241.7MHz. The proposed system is capable of processing 115.7 frames per second at 1080P resolution. Compared to hundreds of millisecond execution latency in a general CPU, the SOC realization of traffic sign classification generates a much lower latency - 6.5us.

  We emphasize our contributions on the algorithm and system design in the following aspects:

  - Noticing that traffic sign has man made shapes, we pick the HOG algorithm as the feature extractor and achieve good performance.
  - Through experiments, we find that the linear SVM has comparable performance with nonlinear SVMs. Hence, we select the linear form of SVM as

the classifier, and design an optimized architecture of a linear multi-way SVM.

  - We propose an efficient implementation of the HOG algorithm in digital circuits, and the proposed realization is parameterized and hence could be used in other computer vision systems too.

- Propose an end-to-end SOC architecture for traffic light recognition.

  The traffic light recognition problem has long been studied, and a timely warning of red lights is life saving in many circumstances. In this dissertation, we aim to recognize green lights and red lights in real time. Through our experiments, the HOG + SVM algorithms work well for traffic light recognition. The traditional region proposal method - sliding window brings in many false positives. Hence, color information is leveraged to propose region-of-interest. In this work, image preprocessing is composed of color based filtering connecting with the BLOB detection method, which proposes potential candidates. Then, the HOG plus SVM algorithms detect and recognize a real traffic light. By accelerating the BLOB detection method in FPGA fabric, the entire system can work in real time. The highest frequency of the system is 150.1MHz, and the maximum throughput is 72.4 frames per second at 1080P video streaming input.

  We emphasize our contributions in the following aspects:

  - We propose a software and hardware co-design methodology for traffic light recognition, realizing real time performance on Xilinx Zynq platform.
  - We design an end-to-end SOC architecture for traffic light recognition, and this architecture on Xilinx Zynq platform could serve as a go-to solution

for ADAS.

- Design an elastic architecture for the PCANet algorithm in VLSI.

  Due to various driving conditions on the road, when designing a computer vision system for ADAS, designers face the challenge to trade off between algorithm robustness and real time capability. Traditional feature extraction algorithms fail to work robustly due to their fixed steps on performing feature extraction. While convolutional neural network (CNN) adapts to various tasks better through iterative training. A major issue with CNN is that CNN can hardly meet the real time requirement of a computer vision system, making it less appealing to an embedded system. In this dissertation, the PCA based network is proposed as a robust feature detector for intelligent vehicles, while keeping a simpler structure than regular CNNs. Compared to HOG, the PCANet achieves averagely 4% improvement in precision rate and recall rate on traffic light classification. The PCANet achieves comparable performance with a regular CNN. Implementing in Synopsys 32nm process technology, the proposed PCANet accelerator only consumes 0.5 watt, and is capable of classifying over 742K image candidates in one second.

  We emphasize our contributions in the following aspects:

  - We propose the PCANet algorithm as the baseline detector for vision based ADAS applications. The PCANet needs much less time on both training and inference.
  - We design an efficient architecture for the PCANet algorithm in VLSI, realizing ultra high throughput while maintaining low power consumption.
  - We prove the PCANet's performance on road marking detection and traffic

light recognition, indicating the potential of the PCANet on other object detection tasks.

- Propose an elastic architecture for binary neural network in VLSI.

  A regular CNN model typically contains hundreds of megabytes weights, which need to be stored into external memory. In 2015, research led by Matthieu Courbariaux and Yoshua Bengio shows that it is feasible to create a neural network, in which each weight is in binary format [18]. The major merit of the BNN is that it retains comparable performance with a CNN of the same configuration. In this way, memory usage and computational resources are reduced by 32 times. Our work proves that the BNN works well on tasks like pedestrian detection and car detection. Implementing the BNN in Synopsys 32nm process technology, a maximum operating frequency 350MHz is achieved with only 0.6 watt power consumption.

  We emphasize our contributions in the following aspects:

  - We propose the BNN as a potential detector for vision based ADAS applications.
  - We design an efficient and fully pipelined architecture for the BNN, achieving high power efficiency. The proposed BNN accelerator has an elastic structure and can execute any number of convolutional layers.
  - We proved BNN's performance on pedestrian detection and car detection, indicating BNN's potential on other ADAS applications.

1.3 Outline

This dissertation is organized as follows:

Chapter 2 presents the implementation a real time lane detection system. We first introduce related literature on the lane detection problem. The hough transform algorithm and necessary image processing methods are described. Then the overall architecture and individual block design are explained in details. Finally, we show the simulation performance and implementation results.

Chapter 3 presents the hardware architecture of the HOG and SVM algorithm. The HOG and SVM is evaluated on traffic sign classification and traffic light recognition. Related work and the system composition is introduced firstly. Then, the hardware design of HOG and SVM is presented, with some critical blocks explained in details. Further, the preprocessing part of the SOC system is introduced, which is used for traffic light recognition. An optimized implementation of the BLOB detection is also presented in this chapter. Finally, the simulation results and FPGA system performance are presented.

Chapter 4 presents the hardware architecture of the PCANet algorithm for ADAS applications. Related work is introduced in this chapter, showing the advantage of the PCANet over mainstream computer vision algorithms. Further, we present the optimized hardware architecture of PCANet and its implementation details in Synopsys 32nm process. Finally, the chip performance and evaluation results are presented.

Chapter 5 presents an efficient hardware implementation of the binary neural network for intelligent vehicles. The BNN algorithm was proven to be effective on dataset like Cifar-10, with near CNN performance. In this chapter, the performance of

BNN on ADAS applications like pedestrian detection and car detection are evaluated. Hardware architecture of BNN and the chip performance is presented too. Finally, evaluation results on collected dataset are presented.

Finally, chapter 6 draws the conclusions, and presents potential future work to this dissertation.

Chapter 2

SOC Architecture for Lane Detection

In general, ADAS focuses on two categories of problems: road segmentation and object detection. As an initial step of road segmentation, the lane detection problem had been studied for decades [19]. In real world, lane detection faces challenges arising from various driving conditions and limited power supply from a vehicle. In this chapter, we will present our approach on resolving these difficulties by fine tuning both algorithm design and hardware implementation. In the end, we demonstrated the our design on Xilinx Zynq FPGA platform.

In this chapter, the SOC architecture of a lane detection system on Xilinx Zynq platform is presented, and the system is proven to be resource efficient in embedded environment. We studied several mainstream lane detection techniques, and picked the most popular and robust algorithm - hough transform for this task. Through software profiling, we identified the bottleneck of the computer vision system to be angle and distance calculation, which is a necessary step in hough transform. To realize real time performance of the entire system, the hough transform is accelerated in FPGA fabric with an efficient architecture. By splitting the entire system onto

the ARM processor and FPGA fabric, we successfully archived a constant processing speed of 69.4 frames per second at 1080P resolution.

The outline of this chapter is as follows. Section 2.1 introduces the background of the lane detection problem. Section 2.2 explains algorithms involved in lane detection, and presents the entire vision system composition. The hardware architecture and individual IP core design is presented in Section 2.3. Experimental results and FPGA results are shown in Section 2.4 and Section 2.5. Finally, Section 2.6 concludes the chapter.

2.1 Introduction

The lane detection problem has long been a critical part of advanced driver assistance systems (ADAS). As an initial step for road segmentation, lane detection facilitates the scene understanding ability of an intelligent vehicle [20]. For example, lane detection is required by many automotive features, such as lane departure warning (LDW) [21] [22] [23], adaptive cruise control [24], lane centering [25], lane change assistance [26] and so on. In recent years, some good progress has been made on the lane detection problem [19].

One major issue with existing lane detection solutions is the real time processing capability. In real world, lane detection systems have to reach very low error rates in order to be useful [19]. Besides, real time is highly needed since lane detection is often used on highways [27], where short time possibly means a long braking distance for vehicles. Designing a feasible lane detection system is challenging due to issues like various driving conditions, changing lane and road appearance, image clarity issues, and poor visibility conditions. To overcome these difficulties, researchers had studied

various sensors and perception modalities for assistance. Typical techniques used for lane detection are monocular vision, light detection and ranging (LIDAR), stereo imaging, geographic information systems (GIS), global positioning system (GPS), radar. Among these sensing techniques, monocular vision is most widely used due to its low cost and easy installation [28]. Some lane detection researchers such as Miao [29] design their system based on a single camera and still achieve satisfying detection rates.

In our work, we not only aim for robustness to most of the driving conditions, but also aim for real time performance. Hence, we choose the Xilinx Zynq SOC platform for the lane detection task. The Xilinx Zynq platform contains an ARM processor and FPGA fabric on a single chip. The FPGA fabric is good at processing low level to middle level image processing algorithms in parallel, while the ARM processor handles high level decisions very well [30]. As proved by Amol Borkar's work [31], up to 90% of highway cases can be solved by a Hough transform-based approach without additional tracking methods. In our work, by dividing the entire computer vision system onto the ARM processor and FPGA fabric, we successfully deployed the whole lane detection system on Xilinx Zynq platform, archiving a good timing performance and low resource usage.

2.2 Algorithm Design

We designed our overall system as shown in Fig. 2.1. As indicated in Fig. 2.1, the first step of lane detection is to extract region of interest (ROI), then edge detection and binarization is used to binarize the pre-processed image. The hough transform works on binary image and identifies straight lines from the image. We notice that

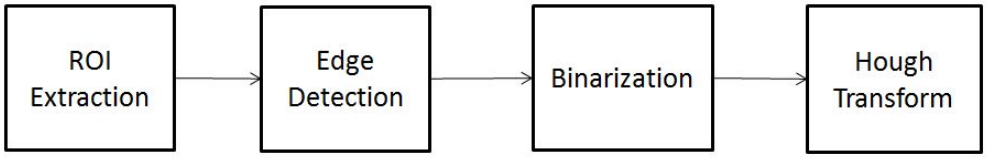

Figure 2.1: Lane detection algorithm flow

Figure 2.2: ROI region on the source image is prefixed

the upper part of the image does not contain much information of the lane, and is thus discarded during ROI extraction. Fig 2.2 shows an example image collected from highway I-90 in Worcester MA USA, and the extracted ROI region is highlighted by the red bounding box.

The second step in lane detection is edge detection and image binarization. The Sobel edge detector and the Canny edge detector are two most widely used edge detectors. Through experiment on our collected dataset in Worcester MA USA, the Sobel detector and Canny detector achieves similar results. Hence, the Sobel edge

detector is used because it has a simpler architecture and executes faster on hardware too. The Sobel operator is named after Irwin Sobel and Gary Feldman, and is mainly used to create image emphasizing edges. The expressions of Sobel operation are:

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A \tag{2.1}$$

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \tag{2.2}$$

In the expressions above, A is the source image, G_x and G_y are horizontal and vertical derivative approximations respectively, and $*$ is the convolution operation. The Sobel operator is essentially a discrete differentiation operator, which computes an approximation of the gradient of the image intensity function. After convolving with two 3-by-3 kernels, edges in the source image are identified.

The purpose of image binarization is to binarize image before further processing. After then, all pixels in the image will become either 255 or 0 (typically a pixel is represented in 8 bits, $2^8 - 1 = 255$ is the largest value in 8 bit data format).

2.2.1 Hough Transform

After image binarization, the hough transform is used to detect two lines of the host lane. The hough transform algorithm is a popular feature extraction technique which is widely used in image analysis, image processing, and computer vision. The hough transform can be used to detect both analytic curves and non-analytic curves [32].

According to a survey of hough transform [33], the hough transform algorithm has long been recognized as a technique of almost unique promise for shape and motion analysis in images containing noisy, missing and extraneous data. In other words, the hough transform is an ideal algorithm for line detection. In our work, the hough transform also achieves high detection rate.

To detect lines of a source image, we first need to know how lines are represented uniquely by two parameters. In rectangular coordinates, a line is represented in the following equation:

$$y = a \cdot x + b \tag{2.3}$$

In polar coordinates, a line can be rewritten as:

$$r = x \cdot cos\theta + y \cdot sin\theta \tag{2.4}$$

In equation 2.4, parameter θ is the angle of the line, and parameter r is the distance from line to the coordinate origin. Equation 2.4 can be rewritten as equation 2.5 to look like equation 2.3.

$$y = -\frac{cos\theta}{sin\theta} \cdot x + \frac{r}{sin\theta} \tag{2.5}$$

All lines can be represented by equation 2.5, where $\theta \in [0, 180)$ and $r \in \Re$. The hough transform uses form in equation 2.4 or in equation 2.5, which are representations in polar coordinates. The hough space for lines has two dimensions: θ and r. Hence, a line can be represented by a single point of (θ, r) in hough space.

In this way, a line in rectangular coordinate system can be mapped to a single

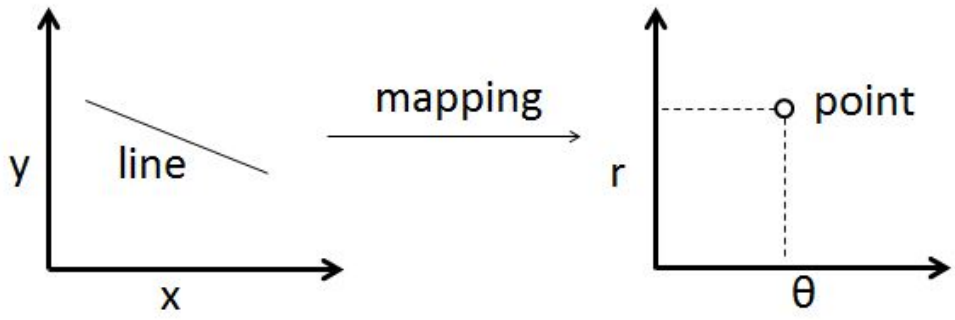

Figure 2.3: A line in rectangular coordinate system is mapped to a point in Hough space

point in hough space, as indicated in Fig. 2.3.

In the other way around, a point in rectangular coordinate system represents all lines passing through that point. In this way, one point maps to all possible lines passing through this point in hough space. As shown in Fig. 2.4, a point maps to a sine-like line in the Hough space.

To detect straight lines of an image, each point in the source image votes to its corresponding possible lines, which are pairs of (θ, r) in Hough space. After processing every pixel in the input image, a histogram table is generated. The line with the largest count denotes the line with the most pixels on it. The process of picking the line with the most pixels is called HoughPeak. Usually, HoughPeak picks a few lines instead of a single line.

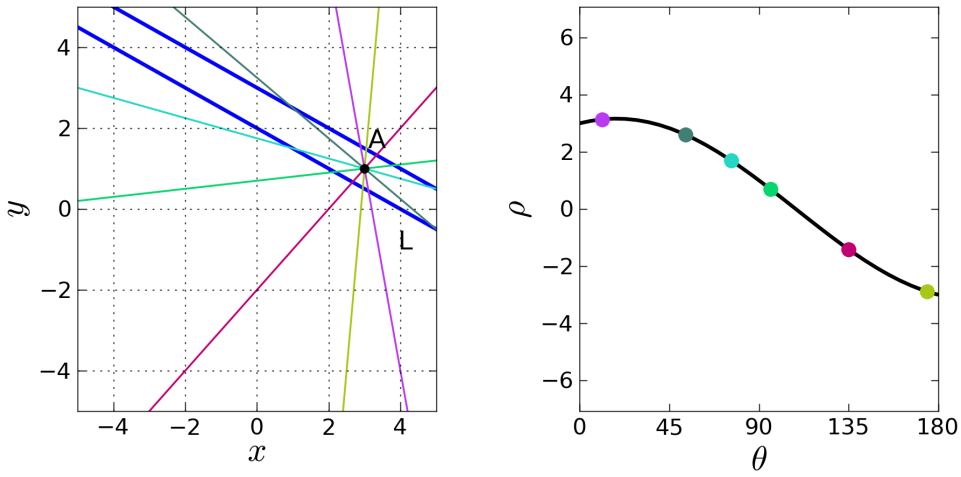

Figure 2.4: A point in Hough space maps to a line in rectangular coordinate system

2.3 Hardware Architecture

2.3.1 Image Pre-processing

The FPGA fabric is an ideal platform for low level image processing in terms of throughput, resource usage and power consumption [34]. In our work, we implement image pre-processing on FPGA. A general structure to perform kernel-based computation is shown in Fig. 2.5. The sobel filter is basically implemented using a matrix of multipliers and an array of accumulators. Noticing that Sobel filter parameters merely have three forms: +1, -1, 0. In our work, instead of using floating point multipliers, we use signed adders to realize multiplication, thus reducing resource utilization on hardware.

Image binarization on hardware is very straightforward, a selector and an unsigned comparator is used in our work as shown in Fig. 2.6.

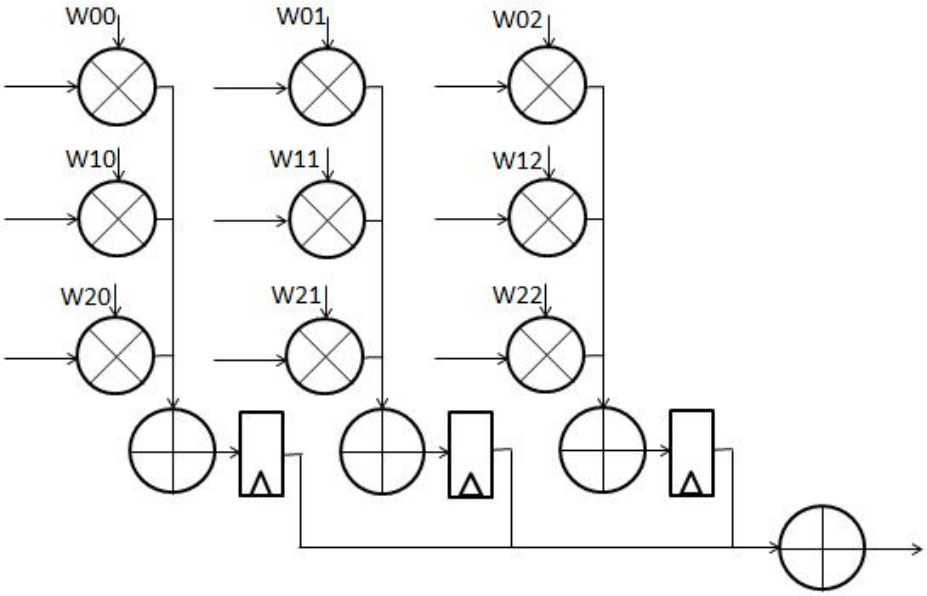

Figure 2.5: For Sobel filter, multipliers are replaced with signed adders

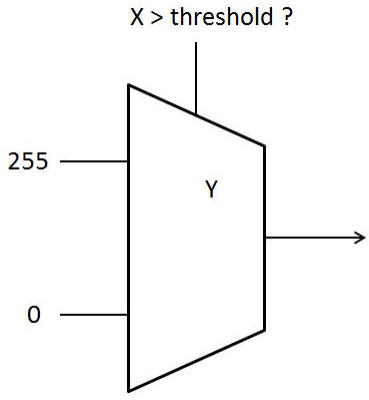

Figure 2.6: An unsigned comparator and a selector is used for image binarization

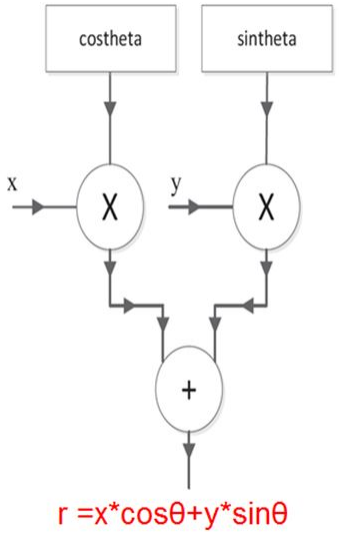

Figure 2.7: Two multipliers and one adder is used to compute distance

2.3.2 Hough Transform

In software, executing the hough transform algorithm on a 720P image takes around 1 second as profiled on an i5 2.4GHz CPU, which can hardly meet the real time requirement of a modern advanced driver assistance system (ADAS) [35]. In this chapter, we accelerated the hough transform algorithm using FPGA fabric. For each pixel in the input image, lines from all possible directions are counted. The line angle range is $[0, 180)$, and for each angle, its distance to the origin is calculated using the structure in Fig. 2.7. Such a block shown in Fig. 2.7 can be regarded as the basic processing element for hough transform. We make the degree resolution to be 1 degree between angles, hence 180 such processing elements are needed on FPGA for 180 degrees range. The result of each processing element is accumulated and stored into memory, as shown in Fig. 2.8.

As we can see from Fig. 2.7 and Fig. 2.8, each processing element needs two

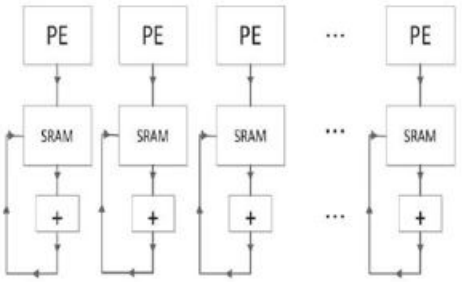

Figure 2.8: 180 processing elements are needed for hough transform

multipliers, and a total of 180 such processing elements are used, resulting in 360 multipliers on FPGA. In our work, leveraging properties of sine and cosine functions, we further optimize multiplier usage. Such properties are expressed in following equations:

$$sin(\theta) = sin(180^o - \theta) \tag{2.6}$$

$$cos(\theta) = -cos(180^o - \theta) \tag{2.7}$$

Where $\theta_1 + \theta_2 = 180^o$, $sin(\theta_1) = sin(\theta_2)$, and $cos(\theta_1) = -cos(\theta_2)$. In other words, once $sin(\theta_1)$ and $cos(\theta_1)$ is calculated, $sin(\theta_2)$ and $cos(\theta_2)$ can be obtained through simple add/minus operations. In this way, only 180 multipliers are used instead of 360, saving 50% of the total number of multipliers. Fig. 2.9 shows the proposed structure. Signed subtractors are used for cosine functions, as highlighted in red box in Fig. 2.9.

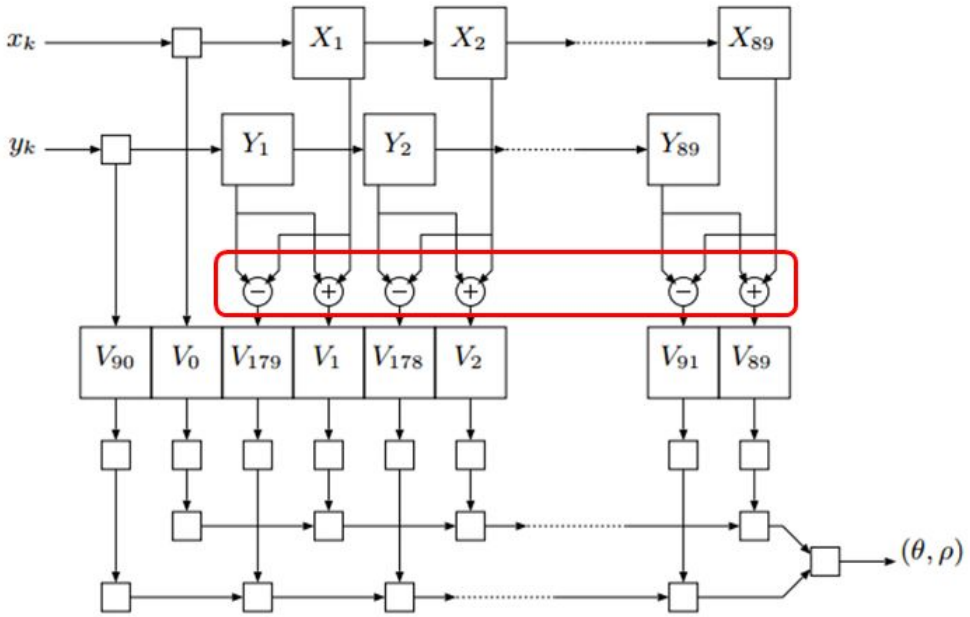

Figure 2.9: 50% multipliers are saved through reusing sine and cosine blocks

2.3.3 SOC Architecture

After hough transform, the HoughPeak algorithm is performed to pick straight lines. The HoughPeak algorithm is a grid search algorithm which picks the largest value in a two dimensional matrix. Through software profiling, this algorithm is not time consuming, so we implement it on the ARM processor. The overall SOC architecture is shown in Fig. 2.10. The coming video streaming is split into two channels: one channel goes through the hough transform and produces a two dimensional matrix, and the other channel retains the original video information for further processing. The ARM processor scans the 2D array and decides which two lines make up the host lane. Finally, depending on the line positions, the ARM processor highlights the host lane in the input image and sends out the overlayed image.

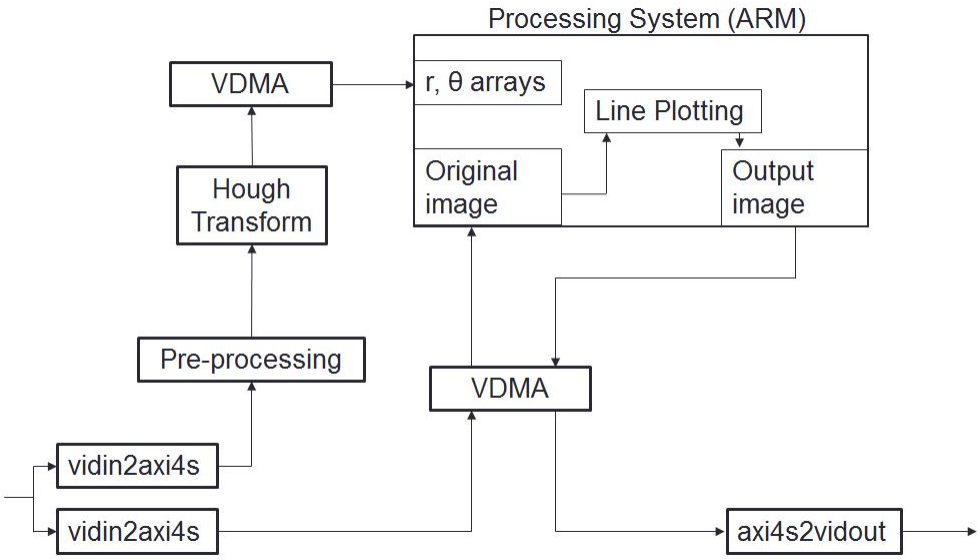

Figure 2.10: SOC Architecture of the lane detection system

2.4 Experimental Results

The lane detection system is evaluated on KITTY road dataset, where 289 training images and 290 test images are used. The lane detection system achieves an accuracy of 70% on KITTY dataset. Noticing that KITTY dataset mostly contains non highway images, and lane detection is mostly used in a highway scenario. Hence, we collected additional test data on highway I-290 in Massachusetts. In a one-minute video clip, the host lane can be detected on over 99% of video frames. Lines mismatch only occurs at cornering, when no straight lines could be found in the image. In most of the driving cases, our hough transform based implementation can achieve a stable performance.

Resources	Used	Available	Utilization
Slice LUTs	35,426	218,600	16.66%
Slice Registers	32,300	437,200	7.38%
Block RAM	206	545	37.79%
DSP	168	900	18.66%

Table 2.1: FPGA Resource Usage on Xilinx Zynq706 platform

	Processing Platform	Throughput	Performance
our work	Xilinx Zynq ZC706	69.4 fps	99% on Worcester highway
Portable [36]	ARM7+FPGA	25 fps	99.57% day / 98.88 night
Fast Marking [37]	NVIDIA 7600 GPU	10 fps	one of the finalists in DARPA

Table 2.2: Comparison with other lane detection systems

2.5 FPGA Results

In our work, we demonstrated the proposed SOC architecture on Xilinx Zynq 706 SOC platform. The resource usage of the system is shown in Table 2.1. Due to the efficient implementation of the hough transform algorithm, only 168 DSP blocks are used which means our design can possibly fit in low end FPGAs. The highest frequency of the system is 143.85MHz. The maximum processing speed of the design is $\frac{10^9}{1920\times1080\times\frac{1000}{143.85}} = 69.4$ frames per second at 1080P resolution.

There are other real time lane detection systems as well. The mainstream processing platforms are either GPU or FPGA. The comparison of our implementation with other lane detection systems is shown in Table 2.2.

From the table, we can see that our implementation outperforms typical lane detection systems on system throughput. One point worth mentioning in Table 2.2 is that Lipski's work [37] is designed for urban driving where a moving speed of 15mph

is restricted, while our implementation can catch up with a vehicle speed of 65mph on highway.

2.6 Conclusion

Lane detection is a critical part of an ADAS, and has long been studied. In this chapter, we present a real time, low power implementation of the lane detection system. Through hardware and software co-design, we had successfully delivered a go-to solution for the lane detection problem on Xilinx Zynq platform. With over 99% accuracy on local highway, our implementation can process video at a speed of 69.4 frames per second at 1080P resolution. Furthermore, our design can be deployed onto lower end FPGAs according to the resource usage reported in Table 2.1. The SOC architecture proposed in this chapter is general, and could be extended to other computer vision tasks.

Chapter 3

SOC Architecture for Histogram of Oriented Gradients

The histogram of oriented gradients (HOG) [5] is very suitable for feature extraction and the support vector machine (SVM) [38] usually works as a classifier. In this chapter, we present the effectiveness of the HOG and SVM algorithm for ADAS applications. Furthermore, we design efficient SOC architecture for HOG to realize real time performance. According to the research done by N. Dalal [17], the HOG algorithm has an outstanding performance on pedestrian detection. In our work, we found that the HOG algorithm works very well for ADAS applications too. Compared to state-of-the-art convolutional neural networks, the HOG algorithm is less complex, and hence requires less resource on hardware. The SVM algorithm is a classical machine learning method, which is very suitable for most classification tasks [39]. In this chapter, we investigate two popular ADAS applications: traffic sign classification and traffic light recognition. Evaluation results on standard dataset and our collected dataset show that the HOG + SVM algorithm achieves superb performance on these

two applications. Furthermore, we design fully pipelined, resource efficient architecture for HOG and SVM algorithm, realizing real time performance on Xilinx Zynq platform.

For traffic sign classification, we evaluate the algorithm performance on the BelgiumTS dataset [40]. We target classifying 48 classes of traffic signs in real time. By accelerating HOG and SVM modules on FPGA fabric, we successfully achieved an operating frequency of $241.7MHz$, which is higher than the frequency needed by a 1080P video data streaming on FPGA. Besides, our implementation only generates 6.5us response time, which is neglectable for a modern ADAS system.

The traffic light recognition task has long been an important task in ADAS applications. In this chapter, we investigate efficient ways to realize a traffic light recognition system based on the HOG and SVM algorithm. We aim to propose an end-to-end computer vision system for traffic light recognition. An end-to-end computer vision system often requires region proposal methods at first. In our work, BLOB analysis based on color information is used to propose potential candidates of traffic lights. For traffic light detection, the HOG + SVM algorithm is applied as the detector and classifier because HOG + SVM is proven to be effective on traffic light detection through our experiments on collected dataset in local Worcester, MA USA. In this chapter, we recognize green lights and red lights at the same time. Yellow lights recognition are not included due to lack of data. Since we are only recognizing green lights and red lights, color information is leveraged to extract region of interest and generate potential candidates for classication. Compared to the exhaustive search method - sliding window, the BLOB analysis largely reduces the number of false positives and thus improves the classication accuracy. In this chapter, a complete computer vision system including pre-processing, region proposal, detection and

classication are presented. For traffic light recognition, we present a novel SOC architecture which processes different parts of the system on either FPGA or the ARM processor. By balancing the workload on FPGA fabric and ARM processor, the entire system can achieve a processing speed of 72.4 fps at 1080P resolution, with an operating frequency of 150.1MHz.

The outline of this chapter is as follows. Section 3.1 introduces the background of the traffic sign classification problem and the traffic light recognition problem. Section 3.2 presents the HOG and SVM algorithm. Necessary preprocessing algorithms used for the traffic light recognition system is presented in Section 3.3 The hardware architecture and individual IP core design of HOG and SVM is presented in Section 3.4. Section 3.5 presents the hardware architecture of an efficient BLOB detection algorithm. Hardware and software evaluation results are shown in Section 3.6. Finally, Section 3.7 concludes this chapter.

3.1 Introduction

A complete computer vision system typically has two parts: detection and classification [41]. The classification part is usually the throughput bottleneck of a computer vision system. The HOG algorithm is widely used in the computer vision field, and typical applications are motion detection, face recognition [42], as well as traffic signs classification [43]. However, the HOG algorithm is relatively complex and slow to execute. It takes seconds to execute the HOG + SVM algorithms on a general purpose processor [44]. Some researchers had already accelerated the HOG algorithm on FPGA such as [45] and [46].

The SVM algorithm is a supervised machine learning method. It constructs a

set of high dimensional space which separates the training data points into multiple classes. For traffic sign classification, an SVM with nonlinear kernel results in a better performance while the linear SVM has lower complexity. Usually for FPGA implementation, the linear SVM is used more [47, 48]. Most of the previous SVM implementations implement two class SVM, which only gives out yes or no decision. In this task, a total of 48 traffic signs need to be classified as shown in Fig. 3.15. Hence, we also design a parallel architecture for a 48 one-vs-all SVM classifier on FPGA.

Traffic sign classification is an important task for driver assistance systems and self driving cars [12]. Many accidents occurred because drivers failed to notice stop signs at intersections. A lot of challenges are posed for the traffic sign classification problem. Firstly, traffic signs vary a lot by countries and by states. Secondly, various backgrounds, illumination conditions, and occlusions cause difficulties for the task. Lastly, real time performance is highly demanded but hard to achieve.

In this chapter, an FPGA accelerator for traffic sign classification is presented. We implemented the HOG algorithm using minimum resources compared to existing work [45]. We also realized a one-vs-all SVM efficiently on FPGA fabric. Our proposed architecture of the multi-class SVM proves to be fast and resource efficient, and can be further extended to others classification problems.

Similar to traffic sign classification methods [49], vision based solutions are the mainstream for traffic light recognition. The key impacting factor of traffic light recognition is the execution time - 30 frames per second is considered to be feasible for an ADAS system [50]. Algorithm robustness to various driving conditions is taken into consideration as well [51, 52].

In this chapter, we also designed an end-to-end vision processing system on Xilinx

Zynq platform. Color information is used to help extract region-of-interest. Then, the blob detection method is used to identify positions of connected regions. The HOG algorithm is used to extract shape related features, and the SVM algorithm works as the final decision maker. Since FPGA is suitable for low level processing, the preprocessing part is executed in FPGA fabric. Then we restrict the number of candidates selected through BLOB analysis. In this way, the detection and classification part is implemented on ARM with a processing speed of 100 frames per second, catching up with the FPGA processing speed. The approach demonstrates our approach to balance the throughput of the ARM processor and the FPGA fabric. The entire system is able to run at a speed of 72.4 frames per second, while attaining over 90% detection rate.

3.2 Algorithms Design

In this chapter, input to the HOG algorithm is resized to be 32-by-32 image patch for feature extraction. The HOG and SVM algorithm will be explained in detail in this section.

3.2.1 HoG Algorithm

The HoG algorithm is a feature extraction method which extracts shape related features from images. By repeatedly generating histograms of each pixel's gradient, the HoG outputs a vector which is called the HoG feature descriptor. Fig. 3.1 shows the general steps used in HoG calculation. Detailed computational steps are to be presented as follows.

For traffic sign classification, all image candidates are resized to be 32-by-32. Each

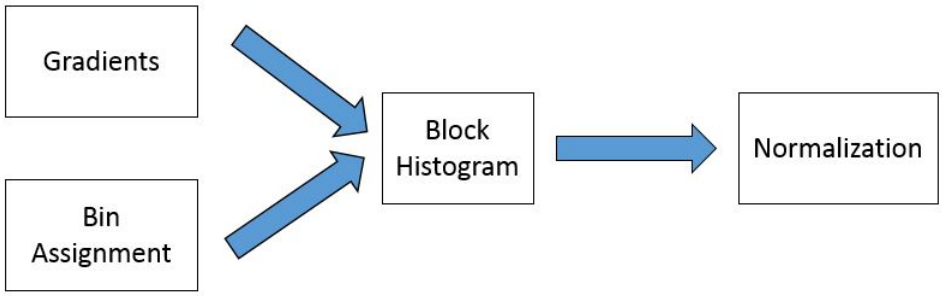

Figure 3.1: HOG calculation flow contains three parts: gradient and bin calculation, histogram generation, and normalization

input image candidate is divided into blocks and cells, with block size to be 16-by-16, and cell size to be 8-by-8. The block first slides horizontally in the window with a step size of 8 pixels, and then the block slides vertically with the same step size. One horizontal line in the window contains $(32 - 16 + 8)/8 = 3$ blocks, and one vertical line in the window contains $(32-16+8)/8 = 3$ blocks, resulting in a total of $3 \times 3 = 9$ blocks of one image candidate.

A single 16-by-16 block generates 36 HoG descriptors and then the current block moves to its next position and repeats computation. As shown in Fig. 3.1, the HoG calculation has three steps: weighted magnitude and bin class calculation, block histogram generation, and normalization. Each part will be explained in details in following subsections.

3.2.1.1 Weighted Magnitude and Bin Class Calculation

For each pixel from the source image candidate, gradients on two directions are to be determined using following equations:

$$G_x(x, y) = |M_x(x + 1, y) - M_x(x - 1, y)| \tag{3.1}$$

$$G_y(x,y) = |M_y(x,y+1) - M_y(x,y-1)| \tag{3.2}$$

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

$$G(x,y) = \sqrt{G_x(x,y) + G_y(x,y)} \tag{3.3}$$

$$\theta = \arctan \frac{G_y(x,y)}{G_x(x,y)} \tag{3.4}$$

Pixels on boundaries are padded with zeros, and zeros are padded as boundary pixels' neighbors.

3.2.1.2 Block Histogram Generation

According to the value of the gradient angle, the current gradient is assigned to 9 different bins. That is, from degree 0 to 179, each 20 degrees denote one bin. For each 8-by-8 cell, a corresponding histogram is generated: based on current pixel's bin value, the weighted magnitude is summed up for the corresponding bin. A total of 36 bin values from four cells' histograms are generated from each block.

3.2.1.3 Normalization

Normalization is a necessary step to provide robustness to illumination and contrast changes. Normalization usually has the forms of L1 and L2. For easy implementation on FPGA, the simplified equation derived from L1-Sqrt-Norm is given:

$$b_{norm} = \sqrt{\frac{b}{sum(b)}} \tag{3.5}$$

Through experiments, we found that L1 form's classification accuracy is very close to L2 form's.

For an entire window, all histograms from 9 blocks are concatenated together, generating a HOG descriptors with 9 * 36 = 324 elements.

3.2.2 SVM Algorithm

The SVM algorithm can quickly separate hyperplanes between different classes, and effectively map inputs onto high dimensional feature space. In our work, linear SVM is used because it requires fewer computational resources comparing to nonlinear SVMs. In one sliding window, 324 HOG descriptors are multiplied with 48 pre-trained 324-elements vector, then added up with 48 offsets respectively. Equation is given as below:

$$Y = \alpha y^T + \gamma \tag{3.6}$$

In Equation 3.6,α is the support vector, y is the HOG feature, γ is the bias. Since SVM in this work has 48 ways, α is a 324*48 matrix, γ is a 48-elements vector. Result Y is a 48-elements vector, and the traffic sign belongs to the class with the largest value.

3.3 Preprocessing Algorithm Design

A complete end-to-end computer vision system contains three parts in general: ROI extraction, detection and classification as shown in Fig. 3.2. The first part of the computer vision system is pre-filtering, and the function of pre-filtering is to suppress

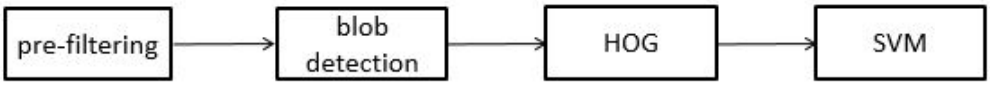

Figure 3.2: Overall algorithm design of the traffic light recognition system

unwanted distortions to improve image quality. After pre-filtering, the blob detection is used to calculate the position of each blob. The pre-filtering and blob detection makes up the image preprocessing part. After preprocessing, all the potential green blobs for green traffic lights and red blobs for red traffic lights are obtained including false positives and false negatives. Classification is then needed to distinguish traffic lights from false positives and false negatives.

In the traffic light recognition task, since a typical traffic light's aspect ratio is between 1/4 to 4, blobs with odd aspect ratios are discarded. After then, each potential blob is resized to 32-by-32. Furthermore, the HOG algorithm extracts 324 features from each blob, and HOG features are then multiplied with the SVM vector. The result of SVM tells whether current blob is a traffic light or not. In this way, by scanning through every blob from the input image, we are able to detect all red and green traffic lights.

For traffic light recognition, another feature we have to realize is real time. A proper software/hardware co-design approach is needed to implement computationally heavy tasks on FPGA fabric, while maintaining high speed hardware/software data exchange. Following are detailed explanations of each module as shown in Fig. 3.2. Since the HOG and SVM algorithms are already explained in Chapter 3, only pre-filtering and blob detection algorithms are to be explained in this chapter.

3.3.1 Pre-filtering

The purpose of pre-filtering is to enhance image quality and filter out unwanted information. In the traffic light recognition system, input pixel format to our system is RGB, and each pixel is represented by 3 bytes. Due to changing luminance and varied weather conditions on road, a pixel appearing green to human eyes doesn't always indicate a large absolute value in the green channel. The major issue of RGB color space is that it doesn't reflect the correlation between color channels. Hence, the first step of pre-filtering is to convert image from RGB color space to human perspective color domain like HSV color space. HSV is a cylindrical-coordinate representations of colorful pixels, representing relationships between each color channel. HSV stands for hue, saturation, and value. In HSV color domain, green and red colors can be easily picked out by setting absolute thresholds on each channel.

Equations below show the formula translating RGB to HSV [53]:

$$H = \begin{cases} 60 \times \frac{G-B}{MAX-MIN} + 0 \ (if\ MAX = R) \\ 60 \times \frac{B-R}{MAX-MIN} + 120 \ (if\ MAX = G) \\ 60 \times \frac{R-G}{MAX-MIN} + 240 \ (if\ MAX = B) \end{cases} \tag{3.7}$$

$$S = MAX - MIN \tag{3.8}$$

$$V = MAX \tag{3.9}$$

The color conversion is a nonlinear transformation. After color conversion, each pixel is binarized for further processing: output 1 if current pixel is considered green,

otherwise 0. The same operation on red pixels is performed in parallel on hardware.

3.3.2 One-Pass Blob Detection

Blob detection collects connected pixels after pre-filtering. The primary idea is to label different groups of pixels on the input image. In the chapter, the 4-connectivity methodology is used to determine whether neighboring pixels are connected or not. The 4-connectivity means that, only the 4 pixels with one pixel distance from the current pixel are considered to be current pixel's neighbors. In this work, to achieve real time, one-pass labeling is utilized, which is to output all the potential blobs by scanning through the entire image once.

3.4 Hardware Architecture of HOG + SVM

The HoG algorithm can hardly achieve real time performance when been implemented on embedded processors or general purpose processors. A 48-way one-vs-all SVM requires a lot of computations by doing comparisons between classes, and cannot run in real time either. In this work, we accelerated both HoG and SVM algorithms on FPGA. Fig. 3.3 shows each step in the HoG computation and SVM calculation. The structure of each module is to be presented in details in following subsections.

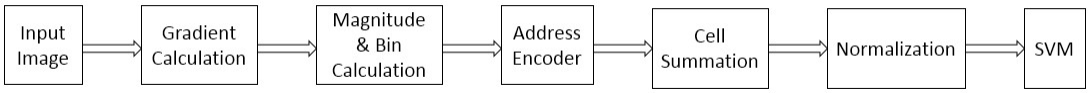

Figure 3.3: Overall algorithm flow of the traffic sign classification system

3.4.1 Gradient Calculation

As shown in Fig 3.4, at each clock cycle, one pixel is taken in. Then, gradient of current pixel is calculated using one shift register and two subtractors. The operating frequency of the circuits is the same with the pixel rate.

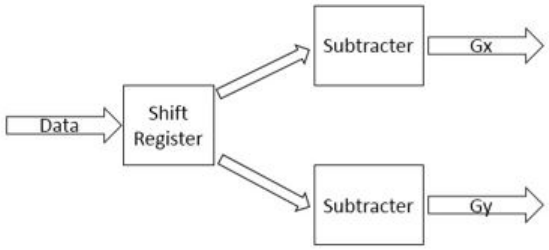

Figure 3.4: Gradient calculation structure on FPGA

3.4.2 Magnitude and Bin Calculation

To calculate the gradient angle, fixed coefficient multipliers are used in place of dividers to save FPGA computing resources. The structure is shown in Fig. 3.5: the magnitude computation is implemented by two multipliers, one adder and one square root unit. The bin class calculation uses four fixed coefficient multipliers, which are more resource efficient than regular multipliers on FPGA. The classifier in Fig. 3.5 is essentially a selector which decides the bin class of the current pixel.

3.4.3 Address Encoder

In the HoG algorithm, blocks have overlaps with each other. To reduce computation effort, results of each cell are to be pre-stored in memories. In this way, when block slides, some cells do not need to be calculated again. The address encoder block

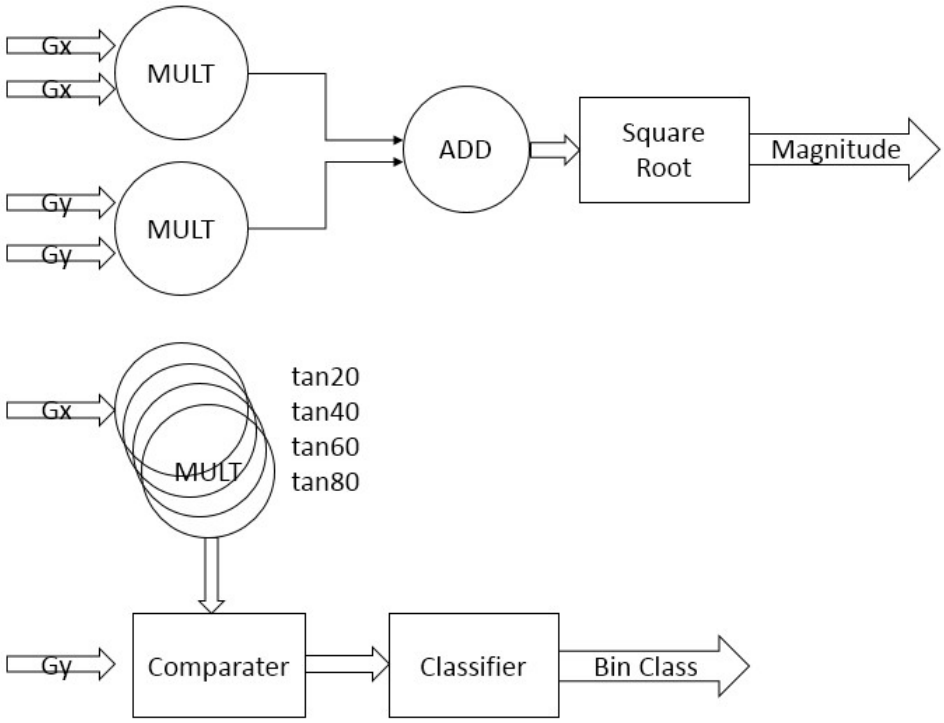

Figure 3.5: Magnitude and bin calculation structure on FPGA

simply converts data streaming into a memory, and results of each cell are stored into specific locations of the block RAM. Fig. 3.6 shows the address encoder architecture. This architecture is a good example demonstrating FPGA's computing power: FPGA can utilize additional memories or computing resources to gain higher throughput of a certain algorithm.

3.4.4 Cell Summation

The cell summation module computes the histogram of each group of 64 pixels in one cell. This module reads out 64 pixels' magnitudes and bin class values, sends these values to an adder tree, then stores computed histograms to another block RAM with corresponding address. In this way, the nine bin values of each cell are stored in a

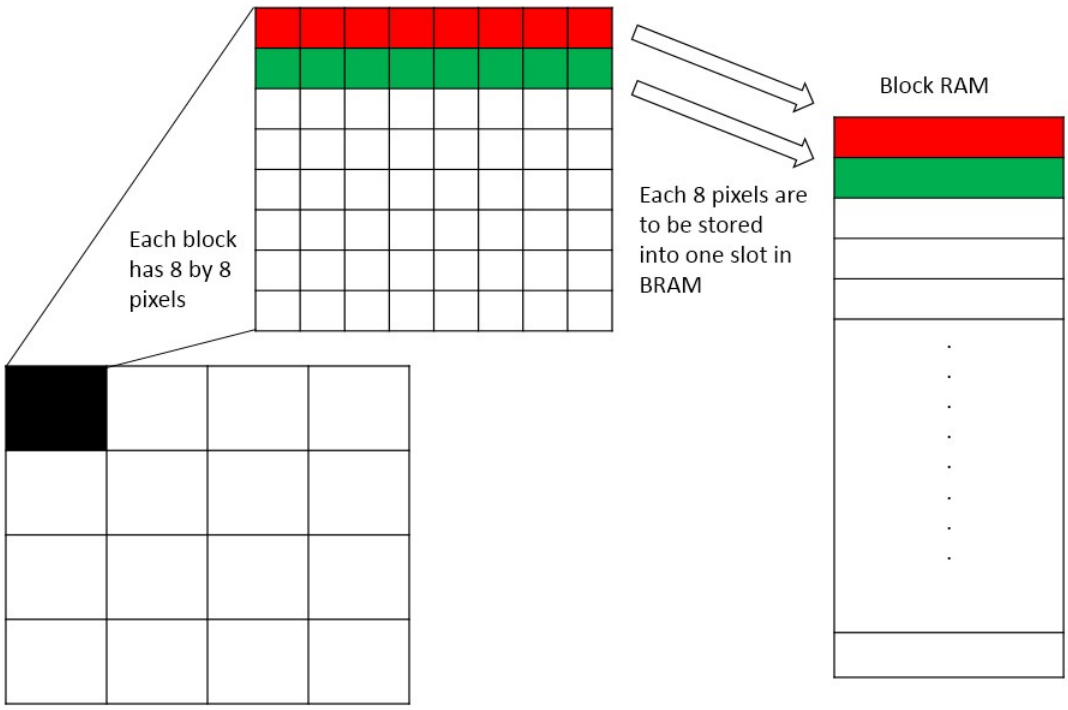

Figure 3.6: Each 8-by-8 cell is pre-stored in the block RAM on FPGA

single memory slot of the target block RAM. Fig. 3.7 shows the architecture.

3.4.5 Normalization

The normalization module takes in histograms from a single block, and sums up all 36 bin values. After then, one divider is used to compute the inverse of the summation, connected with 36 multipliers to get the normalized results before square rooting. Fig. 3.8 presents the architecture: 36 Bin values are read out at one clock cycle, and be processed to get the normalized results. Then, every 36 normalized values are stored to corresponding addresses of the target memory. These normalized results make up the HOG features. Since there are nine memory slots in the block RAM, with each slot storing 36 HOG features. Hence, 9*36 = 324 HOG features are extracted from a

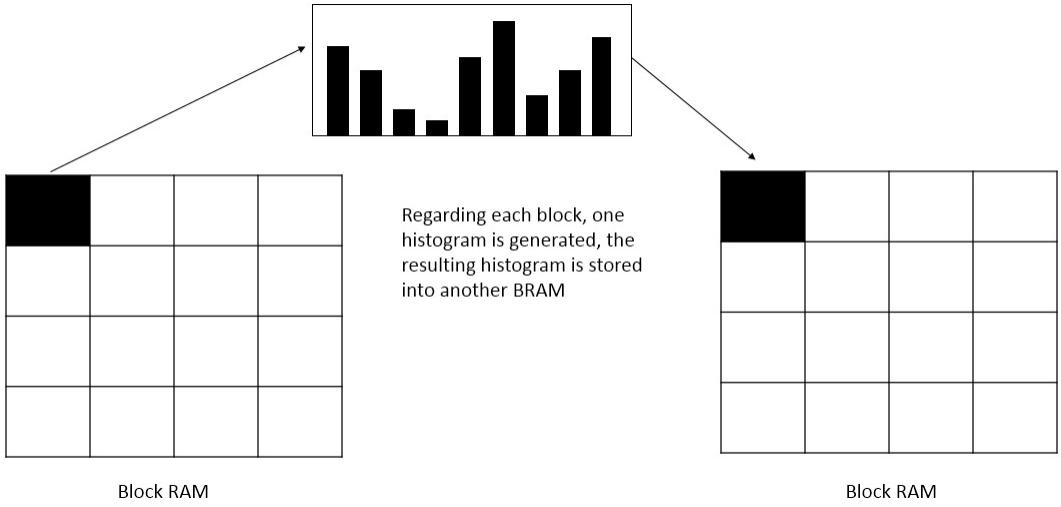

Figure 3.7: Each 8-by-8 cell's histogram is stored into one memory slot in the other block RAM

32-by-32 image patch.

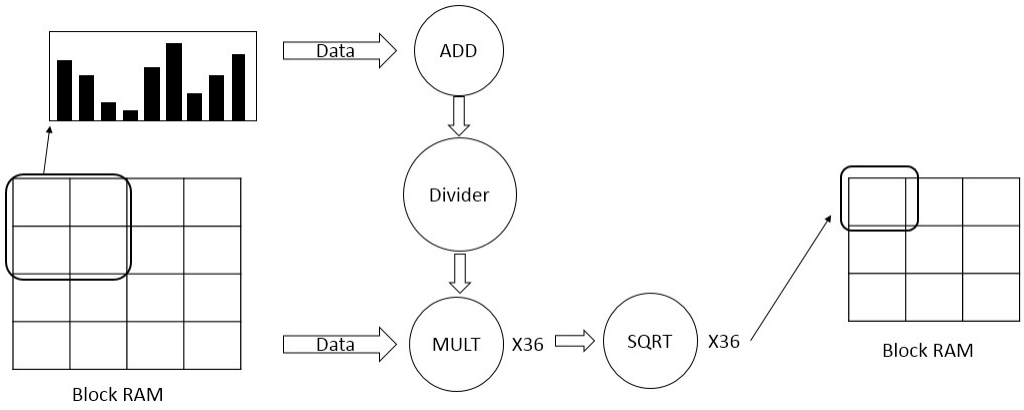

Figure 3.8: For each 16-by-16 block, normalized result is stored into one memory slot of the block RAM

3.4.6 SVM Calculation

In our work, we use a 48-way SVM to classify 48 classes of traffic signs, which means a total of $48 * 324 = 15,552$ support vectors are computed. Regarding such a long

support vector, and considering many multiplication operations needed, we reuse multipliers and process support vectors in batches on FPGA. As shown in Fig. 3.9, a ROM is created to store all these pre-trained support vectors. The SVM module continuously reads out HOG features and multiplies with corresponding support vectors. In this design, we decide to use 36 multipliers which is proven to be a good tradeoff between execution time and resource usage. After all 324 HOG features have been read out, multiplied with corresponding support vectors, an accumulator is used to give out the final result to the adder to add up with the bias. The result is then fed into a comparator which always picks the bigger value. After 48 rounds of SVM computations, the comparator is able to output the biggest value among all 48 classes.

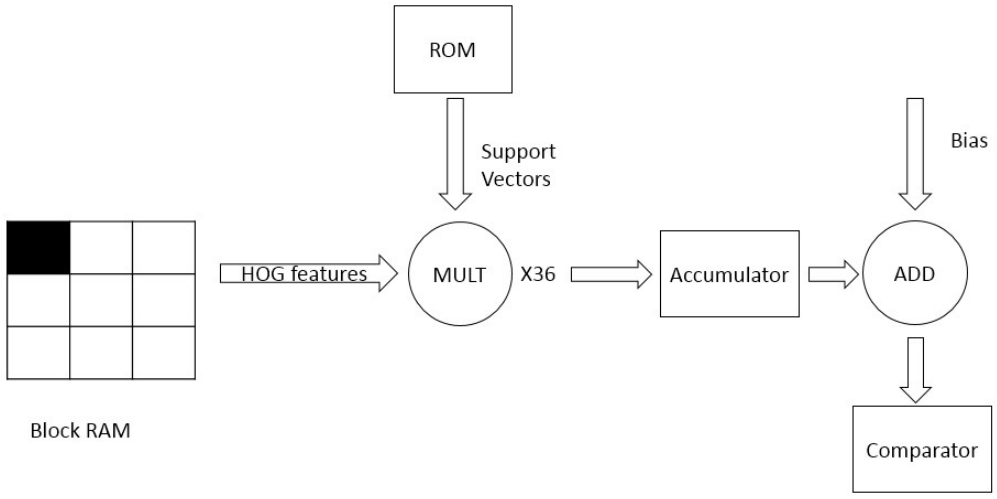

Figure 3.9: An accumulator is used to reuse limited number of multipliers

3.5 Hardware Architecture for Preprocessing

Through software profiling, the HoG + SVM algorithms could be implemented on ARM processor at a processing speed of 100fps as long as the number of potential

candidates is limited to no more than 256. Through experiments, 256 is an optimal number which well balances between high detection accuracy and short execution time. In this way, the pre-processing part becomes the throughput bottleneck of the entire system. This section mainly explains the methodology used to accelerate the blob detection on FPGA fabric, as well as the overall SOC architecture.

3.5.1 SOC Architecture

To implement the whole system on SOC FPGA, delicate partition of software and hardware is needed. In this work, blob detection is computationally heavy in software, and hence needs hardware acceleration. As shown in Fig. 3.10, the input image data streaming is divided into two channels: one channel goes through blob detection, and the other transmits the original image. The HOG and SVM algorithms are implemented on the ARM processor, taking less than 10ms to process all selected blobs of a single image.

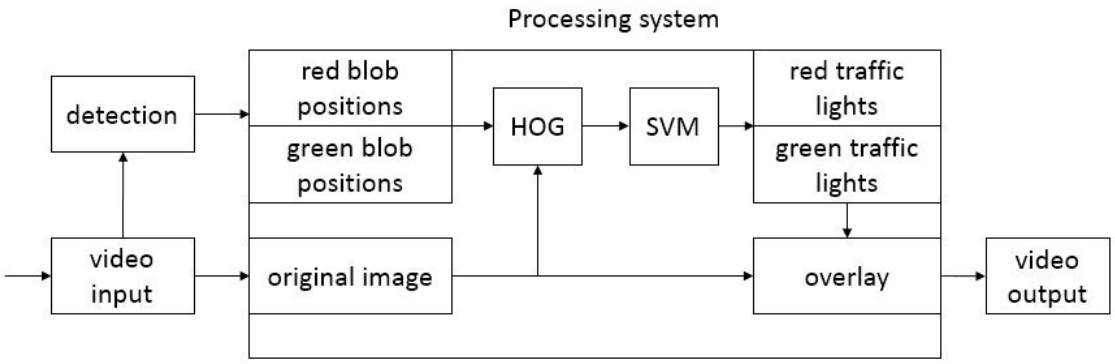

Figure 3.10: Software/Hardware division on SOC

3.5.2 Pipeline Structure for Preprocessing

As described in Section 3.3, image preprocessing consists of color conversion, binarization and blob detection. Fig. 3.11 shows the corresponding hardware architecture. In our work, since two types of traffic lights are to be recognized, two blob detection blocks are used in parallel. Also, a hardware friendly one-pass blob detection is designed in order to achieve high processing speed.

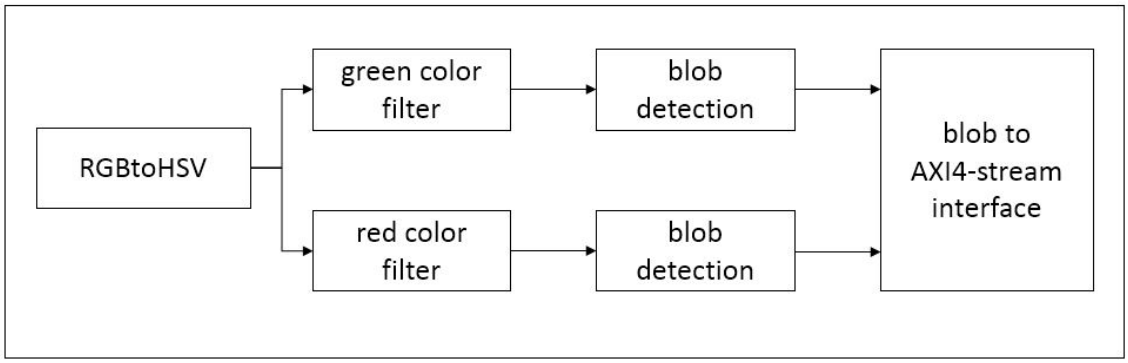

Figure 3.11: Hardware architecture of preprocessing, red blobs are green blobs are processed in parallel

To implement blob detection algorithm on FPGA, one must-have architecture is the blob position table which records the position of each detected blob. As shown in Fig. 3.12, a label counter is created to keep track of current label number: when a new blob is detected, label counter accumulates its value by 1. The blob position table is made up of four memory blocks, recording four vertices of every blob. For a specific blob with label number n, its position information is stored at the nth slot in each of these four memory blocks.

The main difference between one-pass blob detection and multi-pass detection is that one-pass needs an additional connection table on hardware to avoid additional traverse. The connection table checks whether two blobs with different labels con-

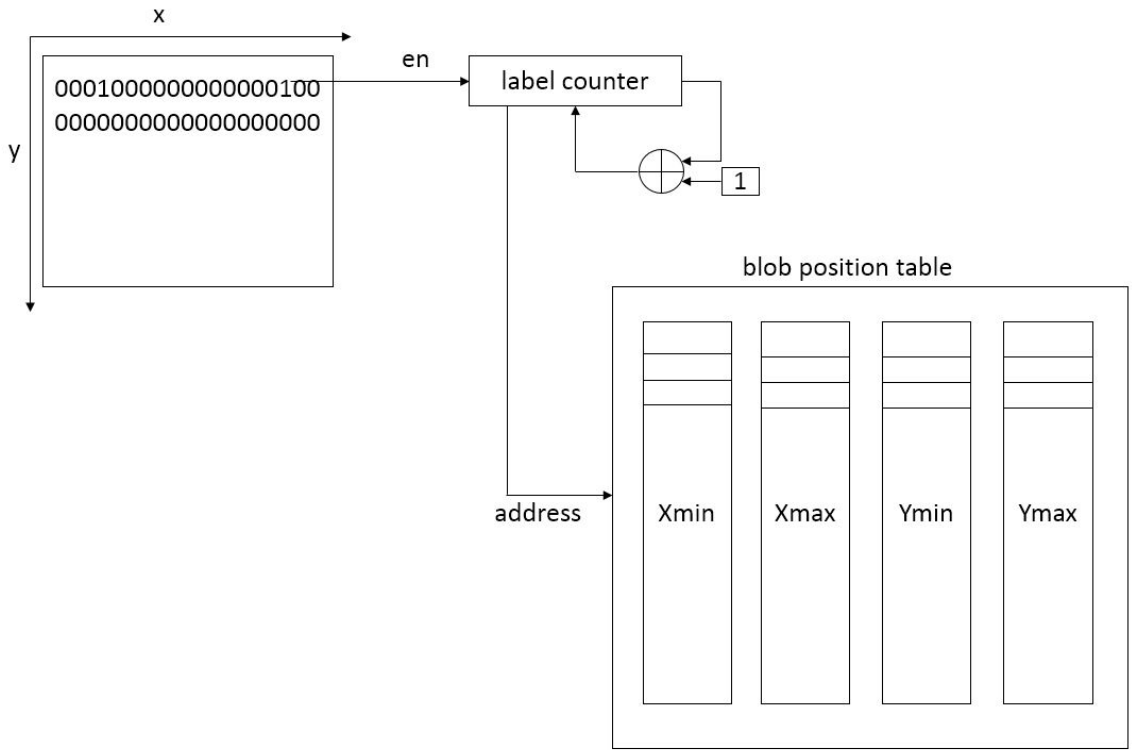

Figure 3.12: A label counter is used as the address encoder to help store positions of each label into the blob position table

nect to each other or not. After scanning through the whole image once, all label connection information are stored into the connection table. An example in Fig. 3.13 illustrates the procedure: when the center pixel is labeled to 5, connection label logic decides that blob 5 and blob 7 are neighboring blobs, hence in the connection table, value 5 is written to the7th memory slot to record the information.

After scanning through the whole image, all information is stored into connection table and blob position table. Position information of same blobs is used to merge memory slots in the blob position table. As indicated in Fig. 3.14, by checking the connection table, we know which labels should be merged, and then update position information in the blob position table accordingly.

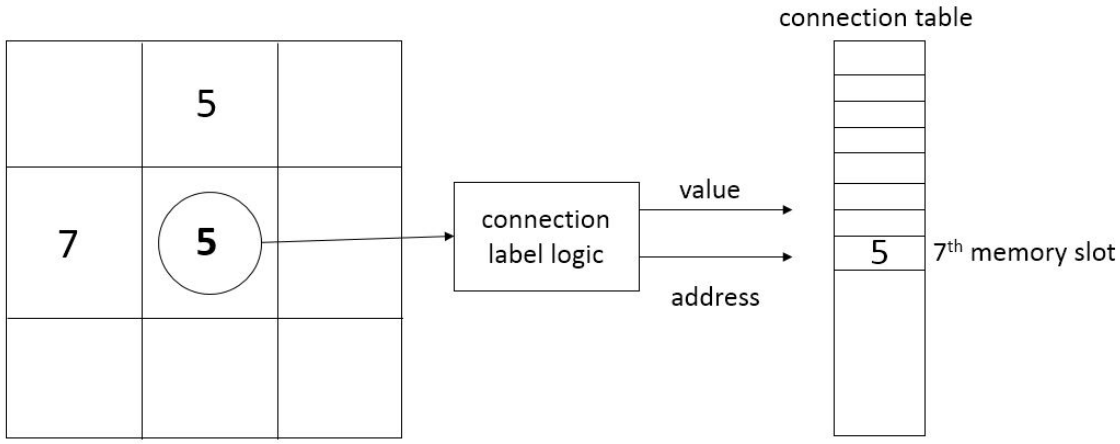

Figure 3.13: The connection table keeps record of connecting labels

3.6 Experimental Results

In this chapter, we evaluated the HOG + SVM algorithm on two ADAS applications and achieve satisfying performance.

3.6.1 Traffic Sign classification

For traffic sign classification, there are three major dataset: German Traffic Sign Recognition Benchmar (GTSRB) [54], Belgium Traffic Sign Dataset [40], and LISA Traffic Sign Dataset [55]. The LISA dataset has less data samples than the other two. Compared to GTSRB, the Belgium signs look closer to US traffic signs. Hence, we used the BelgiumTS dataset to train our computer vision system. Fig. 3.15 shows 48 classes of traffic signs we will classify in our system. For traffic sign classification, we select the most popular HOG + SVM approach. Through experiments, we showed that HoG and SVM can obtain an overall good accuracy on this task, while retaining reasonable algorithm complexity. In this chapter, the HOG algorithm and the SVM algorithm are explained in details.

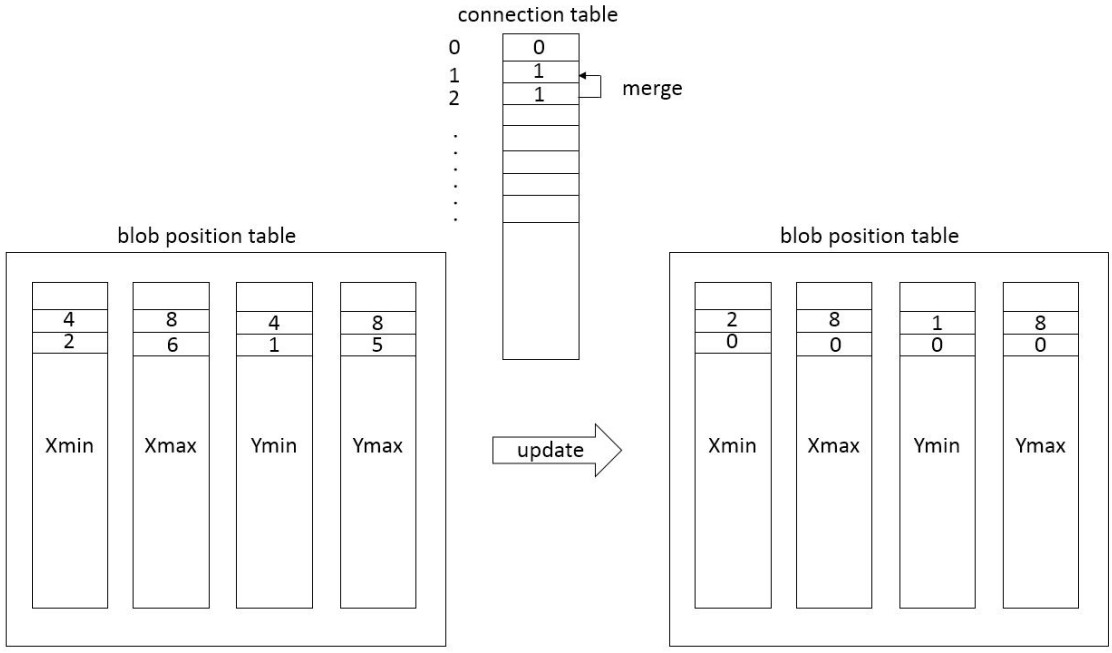

Figure 3.14: Merge memory slots in the blob position table according to the connection table

The traffic sign classification system is evaluated on BelgiumTS dataset. The system is evaluated on 2,181 test images from BelgiumTS dataset. Among all the test images, 1,991 traffic signs can be correctly classified, resulting in an average accuracy of $\frac{1991}{2181} = 91.3\%$ over these 48 classes traffic signs.

Our hardware design is very efficient in terms of resource usage and throughput. The HoG computation unit and SVM computation unit can be used individually as IP cores in other computer vision systems. Table 3.1 shows the entire FPGA resource usage. These two tables show that our implementation only uses a small portion of the entire FPGA, thus could be extended to lower end FPGA platforms. The maximum frequency of our implementation is $241.7MHz$, exceeding $148.5MHz$, which is the input clock frequency used in a 1080P video transmission system.

Analyzing the HoG block and SVM block separately, the HoG block generates 693

Figure 3.15: A collection of 48 traffic signs that can be classified using our system

clock cycles delay, and the SVM block generates 864 clock cycles delay. Operating at $240MHz$ frequency, our implementation has a latency of $(693+864)*4.17ns = 6.5us$, which is neglectable to a real time ADAS system. Assuming the entire system runs at $240MHz$ frequency, with input image size 1080P, the maximum frame rate of our system is:

$$\frac{240 * 10^6}{1920 * 1080} = 115.7fps$$

Along with our work, there are also other efficient implementations for the traffic sign classification problem. In Sheldon's work [56], the system is implemented on the Xilinx Virtex-5 FPGA, with only 777 ms on processing a 40-by-40 image patch. A SURF-based traffic sign detection system [57] has also been implemented on Xilinx KC705 FPGA, obtaining a processing speed of 60 fps at 800-by-600 resolution. The

Resources	Used	Available	Utilization
Slice LUTs	8,168	53,200	15.35%
Slice Registers	11,789	106,400	11.08%
Block RAM	17	140	12.14%
DSP	37	220	16.82%

Table 3.1: SVM Resource Usage

	Processing Platform	Throughput	Accuracy
our work	Xilinx Zynq ZC702	115.7 fps	91.3%
Color Shape [58]	NVIDIA GeForce 560	25 fps	90%
Multi-Scale [59]	NVIDIA K20	27.9 fps	91.7%
FPGA-based [56]	Xilinx Virtex-5	8.8 fps	93.3%

Table 3.2: Comparison with typical traffic sign detection systems

traffic sign classification system is implemented on GPU as well. Table 3.2 presents the comparison of our implementation with other mainstream implementations.

Table 3.2 shows that our performance is comparable to other mainstream implementations. Nowadays, one state-of-the-art computer vision algorithm for traffic sign detection is the integral channel feature (ICF) [60], which archieves over 95% classification rate on the BelgiumTS dataset. There is no existing work which implements the ICF algorithm on FPGA. Comparing to HOG, the ICF is less hardware compatible in that the number of windows extracted from a single feature is not deterministic. A potential furture work could be to implement the ICF algorithm on a larger FPGA chip in order to obtain the state-of-the-art results.

3.6.2 Traffic Light Recognition

The traffic light recognition system is demonstrated on Xilinx Zynq ZC702 board too. Fig. 3.16 shows a sample image from our collected traffic light dataset, and detected green lights are highlighted in white bounding boxes. The highest operating frequency of FPGA implementation can reach is 150.1MHz, indicating that larger resolutions can possibly be supported. Overall FPGA utilization is shown in Table 3.3. Through system profiling on ARM processor, time to process all traffic lights on a single frame varies from 1.96ms to 9.66ms, which is always less than 10ms, meaning that 100 fps performance can be achieved. The maximum processing speed of our system is $\frac{10^9}{1920 \times 1080 \times \frac{1000}{150.1}} = 72.4$ frames per second at 1080P resolution.

Figure 3.16: The sample image shows that green traffic lights detected

For traffic light recognition, we tested our system on 10 video clips recorded in different road and weather conditions. The data is collected in local Worcester MA

Resources	Used	Available	Utilization
Slice LUTs	49,183	53,200	92.45%
Slice Registers	47,656	106,400	44.79%
Block RAM	24	140	17.14%
DSP	25	220	11.36%

Table 3.3: FPGA Resource Usage

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

Table 3.4: Recall rate and precision rate on two traffic lights

USA during both summer and winter. A sample image from the collected dataset is shown in Fig. 3.16.

Table 3.4 shows that we achieved a high recall and precision rate. Equations are given:

$$recall = \frac{true\,positives}{true\,positives + false\,negatives} \tag{3.10}$$

$$precision = \frac{true\,positives}{true\,positives + false\,positives} \tag{3.11}$$

There are other real time traffic lights systems as well. Typical implementations are based on GPU or CPU. Table 3.5 lists several typical implementations of traffic light detection systems.

Comparing to GPU or CPU, our FPGA based implementation achieves a higher throughput and still retains comparable accuracy. Besides, FPGA consumes much less power than a GPU does, and FPGA doesn't need to work with a CPU. Hence,

	Processing Platform	Throughput	Accuracy
our work	Xilinx Zynq ZC702	72.4 fps	95.3%
Agent-based [61]	NVIDIA GTX 470	26 fps	90%
Attention Model [62]	i3 M330 CPU	7 fps	96%

Table 3.5: Comparison with typical traffic sign recognition systems

our proposed system serves as a go-to solution for traffic light detection.

3.7 Conclusion

In this chapter, we present efficient acceleration for the HOG and SVM algorithm. We show that the HOG + SVM algorithm is suitable for ADAS applications such as traffic sign classification and traffic light recognition.

For traffic sign classification, we demonstrated the entire system on Xilinx Zynq 702 platform. By dividing the HoG block into submodules and fine tuning each submodule, we achieve a processing speed of 115.7 frames per second at 1080P resolution. The way we design a 48-way SVM could be extended to any way SVMs, and the proposed SVM architecture is elastic. Furthermore, our FPGA design only generates a $6.5us$ latency, much lower than other traffic sign classification systems to the best of our knowledge. With such a low latency and low resource usage, our HoG unit and SVM unit can be used individually in other computer vision systems.

For traffic light recognition, an FPGA based design for real-time end-to-end system is presented. Evaluating on collected data, we successfully achieved high precision rate and recall rate on traffic light recognition. Also, we successfully demonstrate the entire system on Xilinx Zynq board in real time, with proper resource usage. The maximum operating frequency of the system is 150.1 MHz, resulting in a processing

speed of 72.4 frames per second at 1080P resolution. The hardware/software co-design approach proposed in this chapter provides insights to design similar computer vision systems

Chapter 4

PCANet on VLSI

In recent years, intelligent vehicles have attracted lots of attentions and research interest. Multiple sensors like LiDAR, radar, cameras are deployed on intelligent vehicles. Among these sensors, cameras are cheap and robust. Hence, research on vision based techniques for intelligent vehicles has become the mainstream. Due to various driving environments, traditional hand-crafted computer vision techniques fail to adapt to changing driving environments. On the other hand, deep learning especially multilayer convolutional neural networks (CNN) are proven effective in difficult situations. The major drawback of CNN is that it is not power efficient. To achieve robust performance and high throughput at the same time, we investigated the principal component analysis based network (PCANet) and its applications for intelligent vehicles. A single chip PCANet detector is proposed to resolve the robustness and efficiency dilemma. In this chapter, performance evaluations on road marking detection and traffic light recognition demonstrate that the PCANet is a robust algorithm for object detection. The proposed PCANet chip design using Synopsys 32nm process and achieves a processing speed of 274 frames per second at 1080P video input. To

minimize power consumption, an efficient chip design is proposed which consumes only 0.5 watt. The proposed single-chip PCANet detector strikes a good balance between robustness, efficiency and low power consumption.

The outline of this chapter is as follows. Section 4.1 introduces background of the PCANet. Section 4.2 explains how PCANet is trained and why it is much faster than CNN on training. The hardware architecture of the PCANet is presented in Section 4.3, and VLSI implementation results are presented in Section 4.4. Section 4.5 discusses possible ADAS applications of the PCANet. Finally, Section 4.6 concludes this chapter.

4.1 Introduction

Recently, many industrial and academic research efforts have been focused on intelligent vehicles. Intelligent vehicles usually require a sophisticated fusion of sensors including LiDAR, radar, and cameras. Among these sensors, optical cameras are most widely used because of their low costs and easy installation. Also, due to the rapid development of deep learning in the field of computer vision, vision based algorithms have become more accurate and more robust in various driving environments [1]. One typical deep learning application is advanced driver assistance systems (ADAS), which is becoming very popular among the latest generation of vehicles.

In real world, there are many challenges to vision based ADAS solutions besides the real time requirement. Traditional computer vision algorithms are usually not robust enough to handle the intra-class variability arose from varied lighting conditions, misalignment, occlusion and corruptions, and non-rigid deformations [63]. Although researchers tried to manually design low-level features to counter intra-class variabil-

ity, those hand-crafted low-level features cannot be general enough to be effective in unseen conditions [64] [65], while deep neural networks are able to discover multiple layers of representations of a target object, countering intra-class variability. Because higher level features are resistant to intra-class variability on an image [63], deep neural networks are particularly suitable for tasks like object detection. The major problem of deep neural networks is that efficiency is hard to achieve due to their model complexity [66].

When researchers design computer vision systems for intelligent vehicles, they have to balance robustness, efficiency and power consumption. On one hand, some hand-designed features can achieve good efficiency [67], but fail to counter intra-class variability effectively. On the other hand, deep CNNs are robust enough [66], but fail to achieve high efficiency and low power consumption at the same time. In our work, considering both robustness and efficiency, we propose the PCANet as a robust detector for vision based ADAS solutions.

The idea of PCANet arises from wavelet scattering networks (ScatNet) [68] [69], in which the convolutional filters are prefixed, needing no training at all. Since convolutional filters in a ScatNet are prefixed, the ScatNet does not generalize very well for intra-class variability, illumination change and corruption [63], let alone vision based ADAS tasks. Adopting the simple architecture of ScatNet and the robust performance of multi-layer CNNs, PCANet is fast to train, and still invariant to intra-class variability.

In this chapter, we design efficient hardware architecture for PCANet in Synopsys 32nm process technology, speeding up the PCANet algorithm to 274fps at 1080P while consuming only 0.5 watt power. In this way, we provide a PCANet based single-chip solution for vision based ADAS applications.

4.2 Algorithms Design

In this section, the PCANet's structure is described in detail. The general processing steps of PCANet are shown in Fig. 4.1. Sequentially, an input image is processed through several stages including patch-mean removal, PCA filter with 2D convolution, binary quantization, block-wise histogram, and SVM classification. Note that this work is focused on the implementation of the PCANet detector and the training procedure is not implemented in hardware. Coefficients of the PCA filter and SVM classifier are pre-trained offline using datasets from the target applications.

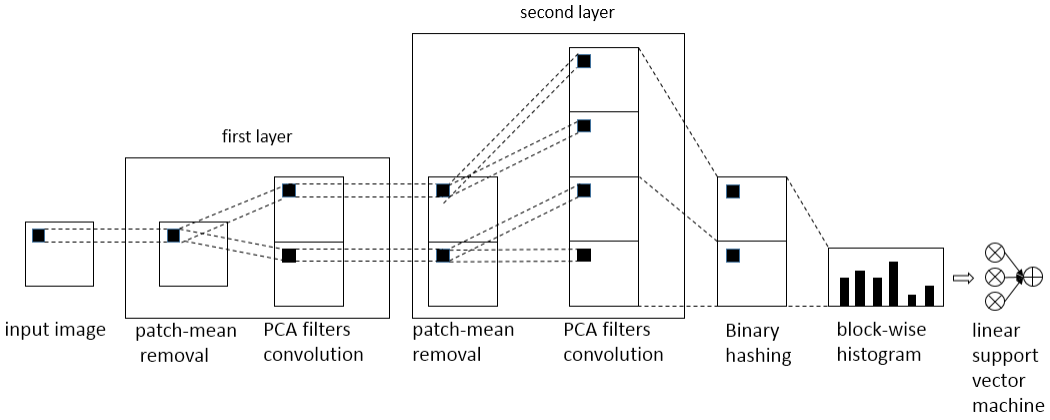

Figure 4.1: General structure of the PCANet algorithm

4.2.1 Patch-Mean Removal

Similar to traditional image filtering, the first step is to use a sliding window to extract image blocks, also called image patches, e.g. 7-by-7 pixels. Prior to applying the PCA filters, all pixels in the patch need to have zero means. Hence, the process is called "patch-mean removal": calculating the mean of all pixels in the patch and then subtract it from the pixel values. For each pixel in the image, a patch is generated

with the pixel at the center position. For the pixels around the edges, the image is padded with zeros. After removing its mean, each patch is stored separately and goes through the PCA filter [63].

4.2.2 2D Convolution as PCA Filter

As the core part of the PCANet, PCA filters extract features from an input image patch like convolutional filters. A total of two layers of PCA filters are used in our design. At the first layer, kernel-based filters are convolved with the input image, and the size of the filters is the same as the patch size, typically 3-by-3, 5-by-5 and 7-by-7. Multiplying with such filters are needed to extract feature maps from a single image. Coefficients of these filters are pre-trained in a feedforward manner by applying the principle component analysis on training samples. For each kernel filter, pixels in a patch are multiplied with the corresponding filter coefficients to perform 2D convolution. Subsequently, products from the same patch are summed up to obtain the convolutional result. In order to extract more hidden features of an image, the PCANet cascades a second layer of PCA filtering process. In fact, this idea is similar to the principle of convolution neural networks, in which multiple convolutional layers are stacked together. Using eight PCA filters as an example, the first layer produces eight filtered feature maps. After another round of patch-mean removal, each of these images goes through the second layer of eight filters, thus resulting in a total of 64 feature maps.

Prior to 2D convolution, coefficients in PCA filters need to be determined. Suppose the input image size is $m \times n$, all PCA filter's size is $a \times b$ at both stages. Number of filters at each stage is N, and there are M training images in total. So there are a total of $m \times n$ patches, for the ith patch ($i = 1...mn$), pixels are denoted as:

$$X_i \quad = \quad [x_1, x_2, ..., x_{ab}] \tag{4.1}$$

In image processing, porch is the additional paddings around the edges of a valid image, and porch is mainly used to provide better boundary effects. The patch-mean removal step is performed right after principal component analysis. Combining the matrices from all the input images, we get the following equation:

$$X \quad = \quad [X_1, X_2, X_3, ..., X_M] \tag{4.2}$$

After patch-mean removal, PCA filters are trained. PCA filters are expected to minimize the reconstruction error of a set of orthonormal filters. The equation for reconstruction error is expressed as:

$$E = \| \ X - VV^T X \ \|^2 \tag{4.3}$$

In equation 4.3, $V^T V$ is the identical matrix of size $N \times N$, and can also be expressed as the first L principal eigenvectors of matrix XX^T. Thus, the equation for a single PCA filter is given as:

$$P_i = mat_{a,b}(p_i(XX^T)) \ i = 1, 2, ..., N \tag{4.4}$$

In equation 4.4, $mat_k(x)$ is the function which maps a vector v to a matrix, and $p_i(x)$ extracts the ith principal eigenvector of matrix x. Consequently, the first principal eigenvector represents the major variation in the patches, the second principal

eigenvector represents the secondary major variation in the patches, so on and so forth.

The second layer in PCANet can be trained in almost the same manner expect for differed data bit width. Equation 4.5 shows the second PCANet layer:

$$Q_i = mat_{a,b}(p_i(YY^T))\ i = 1, 2, ..., N \tag{4.5}$$

As defined in our work:

$$\begin{cases} Y = [Y_1, Y_2, Y_3, ..., Y_N] \\ Y_i = [Y_i^1, Y_i^2, ..., Y_i^M]\ (i = 1, 2, 3...N) \\ Y_i^k = [y_{i,k,1}, y_{i,k,2}, ..., y_{i,k,mn}] \end{cases} \tag{4.6}$$

where $y_{i,k,l}\ (l = 1, 2, ...mn)$ is the lth mean_removed patch in kth filter output of the first stage of the ith input image.

In this way, we can finally acquire the output in equation 4.7.

$$O_i^k \quad = \quad \{I_i^k * Q_k\}_{k=1}^N \tag{4.7}$$

where I_i^k is the kth filter's output from the first stage of the ith input image.

4.2.3 Binary Hashing

After two layers of patch-mean removal and PCA filter convolutions, binary quantization and concatenation are then performed to obtain generalized features. Binary

quantization is essentially a step function, which outputs 1 if the value is positive, and 0 otherwise. Binary hashing is employed for concatenation. Assume that each layer has L feature maps, the data of different feature maps are multiplied with weights from 2^0 to 2^{L-1}. Binary hashing is applied by summing up these L weighted feature maps, which effectively merges them into one feature map.

4.2.4 Block-Wise Histogram

After binary hashing, the range of feature values becomes $[0, 2^L - 1]$, resulting in a total of 2^L bins. To calculate block-wise histograms, the output feature map is divided into blocks, whose size is exactly the same as PCA filter size. Blocks are obtained by shifting through the large feature map. For the data in each individual block, a histogram with range $[0, 2^L - 1]$ is computed. All these block-wise histograms are regarded as the PCANet features.

4.2.5 Linear Support Vector Machine

The aim of SVM is to quickly separate hyperplanes between different categories and to map those extracted features onto high-dimensional feature spaces. The advantage of SVM is that it is quick to train. In this work, linear SVM is applied, since it has better timing performance on hardware and is more efficient on resource usage comparing to kernel-based SVMs. Previous chapters show that linear SVM can achieve a good recognition rate for image classification,. For traffic light recognition, the linear SVM achieves a high classification rate as will be presented in Section 4.5.

4.3 Hardware Architecture

For the implementation of PCANet on hardware, we chose typical values for input image size, the number of stages of PCANet, and the PCA filter size to optimize hardware implementation. All the input image patches are resized to be 28-by-28. We set the convolutional filter size to 7-by-7 and the number of filters at each stage to 8. Software simulation shows that these settings produce the most accurate classification results [63]. The hardware architecture is slightly different from the structure illustrated in Fig. 4.1 due to modifications made to achieve better timing performance on hardware.

The main difference between software execution and hardware implementation comes from adding porch to the input image before patch-mean removal. The input image is first padded with porch around the edges, thus patches in the original input image can be smoothly extracted without slowing down the incoming data rate. In addition, a patch generation module is introduced on the hardware design, so it can output 49 pixel values during a single clock cycle. In following subsections, detailed designs for each module will be presented. The overall computational steps on hardware are shown in Fig. 4.2.

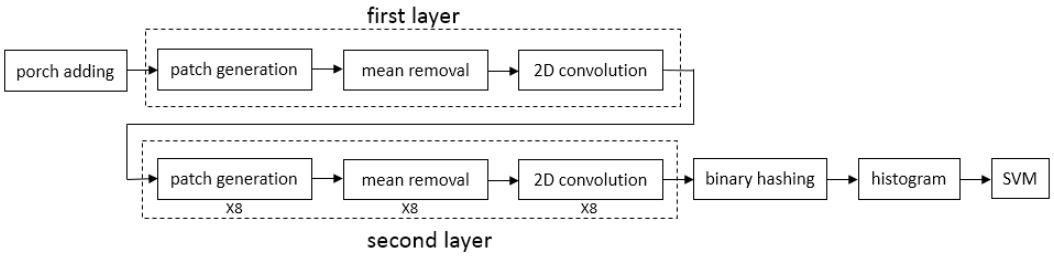

Figure 4.2: Overall hardware architecture of PCANet, it consists of porch adding, two layers of convolutions, binary hashing, histogram and SVM.

To add porch to input image on hardware, the input image is first padded with porch as indicated in Fig. 4.2. In this way, patches around each pixel in the original input image can be smoothly extracted in the same clock domain. In addition, patch generation module on hardware is needed to output $7 \times 7 = 49$ pixel values at each clock cycle. In following subsections, detailed explanations for each module are to be explained.

4.3.1 Patch Generation

Since the mean value of each patch needs to be calculated during patch-mean removal, the patch generation module is expected to output one 7-by-7 patch at each clock cycle. A total of six line buffers plus several registers are used in the hardware design, which can output $7 \times 7 = 49$ pixel values of a patch within a single clock cycle. The patch generation design is shown in Fig. 4.3. In Fig. 4.3, W is the length of one line, which is 34 in our case. With data coming in continuously, after $6 * (28 + 6) + 7 = 211$ clock cycles delay, the circuit outputs the corresponding 7-by-7 patch. Equation 4.8 gives the general formula to calculate clock cycles delay.

$$delay = porch \times (line\,width + porch) + filter\,width \tag{4.8}$$

4.3.2 Patch-Mean Removal

The patch-mean removal module takes in 49 pixel values, followed by an adder tree to compute the summation of these 49 data in 6 clock cycles. Then the sum needs to be divided by 49 to obtain the mean value of current patch. However, division is very resource and time consuming in digital circuits. A usual trick in digital circuit design

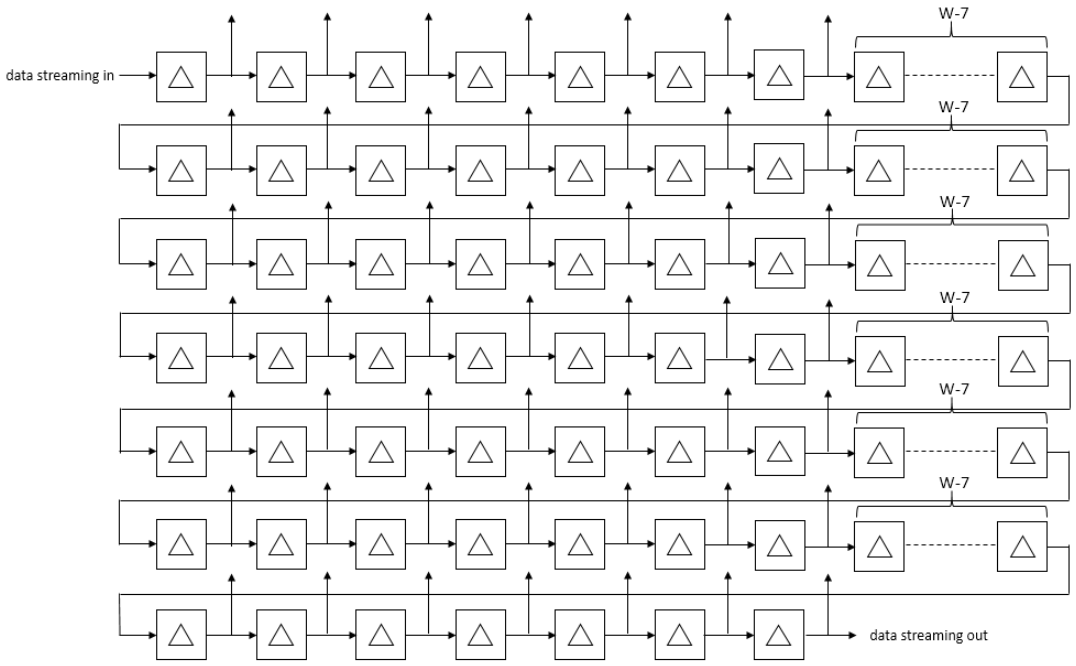

Figure 4.3: Use a line buffer structure to generate one 7-by-7 patch in a single clock cycle

is to replace division with multiplication. In our design, instead of using a divider, the sum is first multiplied by the approximated value of $2^{10}/49 \approx 21$, followed by shifting the results to the right by 10 bits, which is equivalent to division by 2^{10}. Subsequently, each pixel data in the same patch is subtracted by the mean of the current patch at the same clock cycle.

4.3.3 2D Convolution

The operation of 2D convolution is to multiply data in a patch piece-wisely with coefficients from each of the 8 different PCA filters and then to obtain the sum of the 49 products. An adder tree structure is designed for the operation. After 6 clock cycles, the convolutional result is obtained. Since there are 8 feature maps extracted at the first layer, we instantiate 8 such 2D convolution modules for the first layer of

the PCANet on hardware.

4.3.4 Second Stage Computation

The first stage computation consists of one patch generation module, one patch-mean removal module and eight 2D convolution modules. At the second stage, each feature map from the first stage is again expanded by convolutions with 8 PCA filters. In this way, there are a total of 8 patch generation modules, 8 patch-mean removal modules, and 64 2D convolution modules at the second stage. The computational steps are almost identical to the first stage computations except for the enlarged bit width of data being processed. We anticipate that a large number of multipliers will be used due to many 2D convolution modules. At first stage, there are 8 such modules. At second stage, there are 64 such modules. Each 2D convolution module contains 49 multipliers, so the first two layers of PCANet consumes up $(8+64)\times 49 = 3,528$ multipliers on chip. Implementing thousands of multipliers on hardware is very resource consuming. Hence, we further optimize the second stage computation, which will be explained in subsection 4.3.8.

4.3.5 Binary Hashing Module

After two layers of 2D convolutions, each of the 8 feature maps created by the second stage need to be merged together. We take the sign of the input, then multiply those signs with different weights from $[128, 64, 32, 16, 8, 4, 2, 1]$. This process is called binary hashing. In digital circuits, we create an 8-bit register, and then put the sign of each data into different slots of this 8-bit register. The resulting 8-bit register is the sum of these 8 weighted values. Such a structure eliminates the need for multiplication

operations, hence reducing power consumption.

4.3.6 Block-Wise Histogram

As the last step of the PCANet feature extraction, histogram is needed to generalize the features. After binary hashing, there are 8 feature maps, each containing a 28-by-28 array of 8-bit data. Each single feature map is then divided into 6-by-6 blocks, with each block containing 7-by-7 pixels. The first block is taken from the up left corner of the feature map, then the block slides to the right by 4 pixels to generate the second block. The block keeps sliding to the right until it slides out of the 28-by-28 area, then it slides down by 4 pixels and starts from the leftmost pixel to obtain the next block. A total of $6 \times 6 = 36$ blocks are generated in this way. For each block, one histogram ranging from 0 to 255 is then generated. In this block, 256 comparators and 256 6-bit counters are used, after $7 \times 7 = 49$ clock cycles, data from one block is sent to such a circuit, all 256 features are generated at the 50th clock cycle.

4.3.7 Large-vector SVM

Linear SVM makes the classification decision based on the PCANet features. In our work, the input to SVM consists of 8 feature maps, each feature map contains 36 blocks, and each block has 256 histogram values. Therefore, each image patch contains a total of $8 \times 36 \times 256 = 73,728$ PCANet features. As indicated in Fig. 4.4, a total of $8 \times 256 = 2048$ multiplications are needed. In digital circuits, it is impractical to create over 2,000 multipliers on chip. Hence, we divided these features into small pieces and multiplied with coefficients piece-wisely. To balance throughput and power consumption, we decided to use a total of 128 multipliers in the hardware

design, which means up to 128 multiplication operations can be performed within each clock cycle. A state machine based module is deployed to control the flow of the multiplications. Once all multiplications are completed, it proceeds to adding with the bias and then providing the final classification results of the PCANet.

As indicated in Fig. 4.4, this data is firstly stored into buffers, then $\frac{1}{16}$ of these data which is $8 \times 256/16 = 128$ are to be taken for multiplication, after multiplication, an adder tree is attached to compute the summation of these 128 data. SVM coefficients are taken from 32 separate ROMs, with each ROM size to be 64-by-1024 bits.

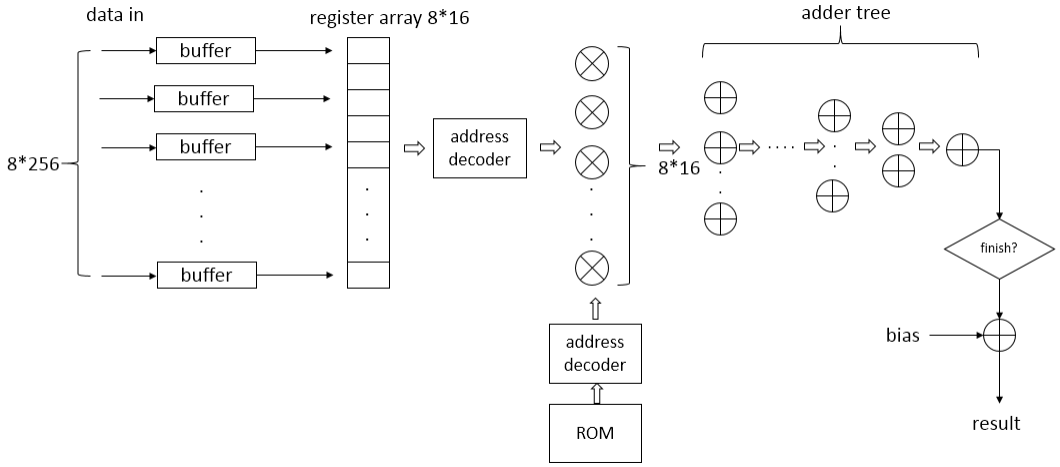

Figure 4.4: Architecture of the linear support vector machine, 32 separate ROMs are created.

4.3.8 Optimized Architecture

As shown in the previous section, second stage implementation uses up to 8 patch generation modules, 8 patch-mean removal modules, and 64 2D convolution modules in total. A separate analysis on second stage circuit shows it uses $5.8mm^2$ of chip space and more than 1 *watt* power consumption. So an optimal design approach is to reuse these modules at the second stage, thus reducing on-chip resource usage and

power consumption. To reuse computational units, additional buffers are inserted between the first stage and the second stage, buffering 8 feature maps from the first layer. In this way, only 1 patch generation module, 1 patch-mean removal module and 8 2D convolution modules are needed. Correspondingly, features generated from second stage need to be buffered too. Since there are too many features from second PCA filter convolution, buffers are inserted after the binary hashing module, consuming much less memory. Although the new architecture sacrifices the overall throughput to some extent, it comes with a much smaller chip size and much lower power consumption. Fig. 4.5 shows the detailed chip architecture of our low power design.

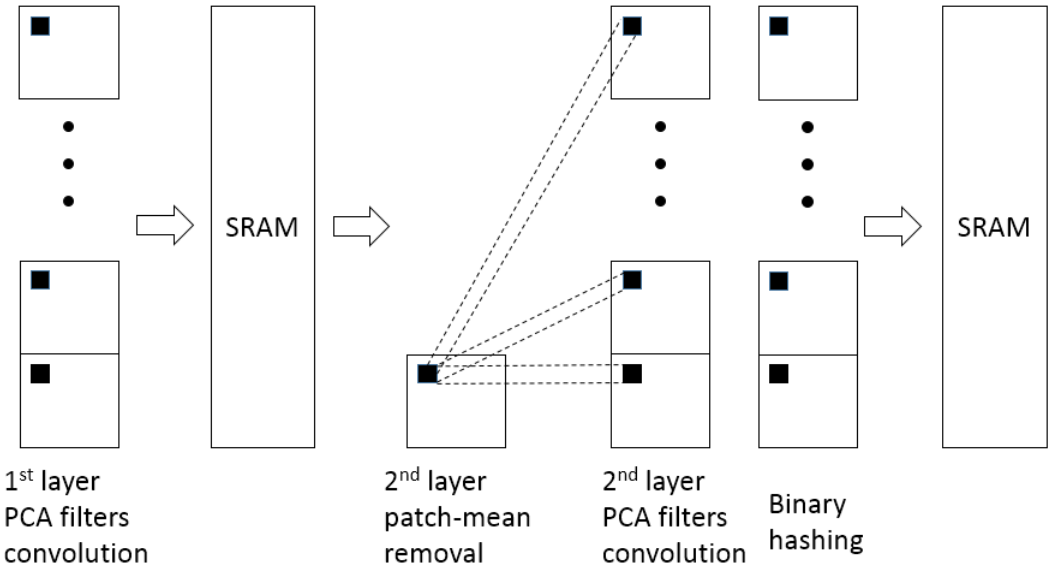

Figure 4.5: Overall architecture of the optimal chip design with resource reuse

4.4 VLSI Results

The circuit is implemented in Synopsys 32nm process technology, the chip power consumption is $0.49\,watt$, and chip size is $3.25mm^2$. The highest frequency is 454.5MHz. Our chip can process up to 724,743 28-by-28 image patches in one second, which is equivalent to 274fps at 1080P. Table 4.1 shows the comparison of PCANet chip with mainstream CNN chip implementations.

Table 4.1: Comparison of PCANet and CNN chip implementations

	PCANet	NeuFlow [70]	Origami [71]	ConvEngine [72]
Chip Area/mm^2	3.25	12.5	3.09	2.4
Power/W	0.49	0.6	1.24	0.76
Max Freq./MHz	454.5	400	350	204
VLSI Process/mm	32	45	65	45
Throughput/fps	274	45	88.5	30
Image Size	1080P	500×375	1080P	1080P
Precision	fixed24	fixed16	fixed12	fixed10

Note in table 4.1, the PCANet chip acts like a classifier, and input to the chip is actually 28-by-28 image patch. The throughput reporting at 1080P resolution is to measure how many pixels can be processed through our network. Throughput comparison, the proposed PCANet accelerator beats mainstream CNN chips on power consumption and throughput.

4.5 ADAS Applications

The PCANet has proved to be effective on many image processing problems [73]. In our work, targeting ADAS applications, we evaluated the PCANet detector on road marking detection and traffic light recognition.

4.5.1 Road Marking Detection

Road marking detection is a very important function for advanced driver assistance systems. In our work, we evaluated the PCANet detector on road marking dataset provided by Wu and Ranganathan [74]. The BING algorithm [75] is used for detection. The BING algorithm can run at a speed of 300fps on a single laptop CPU, while achieving 96.2% object detection rate with 1,000 proposals [75]. The road marking dataset [74] contains 1,443 street view images, each with a size of 800-by-600. There are a total of 11 road markings in the dataset, only 9 classes are selected because the other 2 classes do not provide sufficient samples for training. Some road marking examples from the dataset are shown in Fig. 4.6. We randomly divided all images into 60/40 with no overlapping. That is, 60% images are used for training, and 40% images are used for testing.

On the road marking dataset, we achieved an overall accuracy of 96.8% on 9 classes. The PCANet classification accuracy is more consistent and much better than the original road marking detection work done by Wu and Ranganathan [74], especially on "FORWARD" sign detection, where PCANet achieves an accuracy of 96.8%, far exceeding their achieved value of 23.13%.

4.5.2 Traffic Light Recognition

Traffic light recognition, especially red light recognition is very critical for drivers in that ignoring a red light can be life-threatening. One of our conference papers [76] presented a real time traffic light detection system. In that work, we used color-based pre-filtering and blob detection for candidate proposal, and we used HoG plus SVM for detection. By implementing pre-filtering and blob detection algorithms onto

Figure 4.6: Some samples from the road marking dataset

	Recall Rate	Precision Rate	Total True Positives
HoG	80.3%	89.1%	3,586
PCANet	93.1%	93.2%	3,752

Table 4.2: Traffic light detector accuracy comparisons between PCANet and HoG

digital circuits, we had successfully realized 60fps processing speed across the entire system.

In our current work, we built up our own traffic light dataset around the city of Worcester, Massachusetts, USA. The traffic light dataset contains video data collected during summer and winter. On the dataset, we still choose color-based pre-filtering and blob detection as the candidate proposal method. We compared the performance of HoG with PCANet as the detector. The result of HoG and PCANet performance is shown in Table 4.2.

Table 4.2 shows that the PCANet outperforms traditional HoG algorithm on traffic light detection. Noticeably, there is an over 10% improvement on precision rate.

4.6 Conclusion

In this chapter, we investigate the PCANet algorithm and its hardware architecture as a single-chip object detector. The PCANet detector achieves satisfactory performance on road marking detection and traffic light detection, and is applicable to many other ADAS applications. In Synopsys 32nm process, the ASIC implementation is able to process 724,743 image patches in one second, with only 0.5watt power consumption. Compared to other mainstream CNN based chip implementations, the PCANet implementation achieves better power efficiency and higher throughput. Moreover, the PCANet only needs 6.9K Bytes of weights, which is much lesser than that of a CNN. Consequently, all weights can be stored on chip for fast data access. The proposed PCANet detector is a high-throughput and power-efficient solution for real-time vision applications for intelligent vehicles.

Chapter 5

Binarized Neural Network on VLSI

Advanced driver assistance systems (ADAS) have long been studied in order to aid drivers with vehicle operation. For ADAS, power consumption has always been the bottleneck of the system design. Hence, low power VLSI technique [77] is becoming the mainstream ADAS solution. In this chapter, we propose a novel homogeneous binary neural network based chip architecture to perform pedestrian detection and car detection at the same time, with comparable performance to the state-of-the-art solutions.

Our work uses binarized neural networks, which is a modification of convolutional neural network. Binarized neural network constrains all activations and weights to be +1 or -1 while still retaining a high accuracy rate [78], making it very attractive to digital implementations especially for ADAS. In our work, we trained a binarized neural network with samples from INRIA, and Cifar-10 datasets. After training, our binary neural network only uses 22KB weights, which is much lesser than most of the efficient neural networks such as SqueezeNet [79], which still needs 0.5MB weights. Our design can still achieve an overall accuracy of over 95% on recognizing cars and

pedestrians.

The rest of this chapter is organized as follows. Section 5.1 presents background of the Binary Neural Network. Section 5.2 shows the method of training our Binary Neural Network, explaining the trick of binarizing activations and weights. Section 5.3 presents hardware architecture and implementation details of our ADAS chip. Section 5.4 shows the evaluation results of our design. Section 5.5 presents the VLSI design of our proposed architecture in Synopsys 32nm process technology. Finally, Section 5.6 concludes the paper.

5.1 Introduction

Advanced driver assistance system (ADAS) has become a key technology in modern vehicles that can improve driver safety and reduce road accidents. It is also an important step towards fully autonomous vehicles in the future. Vision-based driver assistance systems have been studied previously with typical applications such as forward collision warning [80] [81], pedestrian detection [14] [82], traffic signal recognition [83] [84], and traffic sign recognition [40] [85].

In recent years, the commercial success of MobileEye ADAS chips [86] stimulated more and more research designing multi-core SoC chips for advanced driver assistance system (ADAS). One typical VLSI implementation is Texas Instrument's TDA series ADAS chips [77] [87]. Other SoC architectures have also been studied [88] [89] [90] for ADAS. All these ADAS chips have heterogeneous architectures with different applications running on different vision acceleration engines, achieving an overall high accuracy, low response time, and low power consumption. However, due to design considerations and chip resource constraint, these ADAS chips still use traditional

computer vision and machine learning methods, failing to achieve state-of-the-art accuracy.

In recent years, deep CNN models such as AlexNet [7], VGGNet [8], GoogleNet [9], ResNet [10] are achieving higher accuracy on the ImageNet dataset [6], outperforming traditional computer vision and machine learning techniques. However, when it comes to embedded applications, those state-of-the-art models are often too complex and power-hungry for an embedded system.

In order to close the gap between deep learning models and embedded platform execution, two main approaches have been studied. One approach is deep compression technology [91], it has been shown that AlexNet can be shrunk by 35 times, and VGG-16 can be shrunk by 49 times without losing any accuracy. With deep compression, the deep learning model needs typically 50MB of weights, which still exceeds on-chip memory available on state-of-the-art FPGA like virtex-7. The other approach is to leverage this repetitive property of CNN. So by creating parallel processing elements from digital circuits [92], complex CNNs can be implemented on GPU [93], FPGA fabric [94] and VLSI [95]. A good example is Eyeriss project [96], in which the AlexNet has been successfully implemented on a single chip. The side effect of the second approach is that data exchange with external memory still consumes lots of power and also increases latency.

In reality, we require CNN with outstanding performance, implemented on a single chip, which can be then easily interfaced with ADAS controllers. In this chapter, we aim to provide a one-chip solution with moderate resource usage and minimum power consumption for mainstream ADAS applications. We are targeting the most needed functions like pedestrian detection, and car detection on highway [97]. One problem of recognizing these two classes of objects in one neural network comes from differed

aspect ratios. Typically, cars' aspect ratio is 1:1, while people's aspect ratio is 2:1. To tackle this problem, we trained the network using only top half of pedestrians. Through experiments with Histogram of Oriented Gradients (HOG) [5], we found that we could still obtain near state-of-the-art performance and reduce computation by 50% while using the top half of the image. Coming to algorithm selection, to make ADAS systems implementable on mobile applications, we required small to moderate size neural network, whose memory requirement is the least and accuracy is expected to be comparable to state-of-the-art algorithms. In our work, we picked the Binary Neural Network (BNN) [98] as our algorithm framework, then modified the neural network and optimized some parts of it to be hardware-compatible. We have successfully implemented our algorithm on integrated circuits in Synopsys 32nm process technology. With an overall high accuracy of 95% on classes of cars and pedestrians, our design achieved consumes only 0.6watt due to the merit of binary operations.

5.2 Algorithm Design

In this section, the binarized neural network algorithm is to be explained in detail. The overall architecture of binarized neural network is shown in Fig. 5.1. Our implementation takes in a 32-by-32 image patch, and gives out the decision which is the class of the image patch. This section mainly explains the method of training a binarized neural network [78].

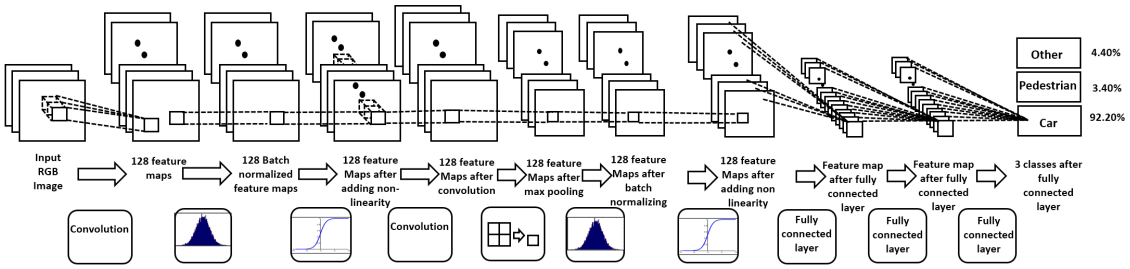

Figure 5.1: Overall Architecture of Binarized Neural Network

5.2.1 Convolutional Neural Network

Neural networks are made up of layers of neurons. Each neuron receives some input, performs a dot product between input and its weight, adds the bias and applies the non-linearity. When input is in image form, convolutional kernel could be applied to extract feature maps in a sliding window fashion. Unlike regular neural networks, convolutional neural networks don't need a connection for each pixel in input image, thus greatly reducing connections compared to fully connected networks.

The convolutional layer is the core building block of CNN, and also consumes most of the computational resources [78]. The typical expression of three dimensional convolution is shown in equation 5.1:

$$Y[n,i,j] = \sum_{d=0}^{D-1} \sum_{y=0}^{K-1} \sum_{x=0}^{K-1} W[n,d,2-x,2-y] * X[d,i+x,j+y] \tag{5.1}$$

In the above expression, input feature map is of size $D \times W \times H$ and output feature map is of size $N \times W \times H$, where N is number of feature maps. $K \times K$ is the convolutional kernel size. Above expression gives(i,j) value of n^{th} feature map.

5.2.2 Backpropagation

Backpropagation is a common method of training neural network with an optimization function such as gradient descent. Backpropagation was first introduced by Yann LeCun [99]. In his paper, backpropagation was applied to train a neural network for hand written digit recognition. Backpropagation typically has three phases: forward propagation, backward propagation, and weight update. Forward propagation generates the network's output values given a training input. The backward propagation is used to generate the deltas (the difference between the targeted and actual output values) of all output and hidden neurons. As the final step, weights at each layer of the network are updated by the amount of the deltas. By repeating the above steps over the entire training dataset a number of times, the neural network can be trained.

As previously illustrated, each neuron has a non-linearity function. A typical choice of such non-linearity function is sigmoid function, which has forms like equation 5.2 or equation 5.3:

$$S_t = \frac{1}{1+e^{-t}} \tag{5.2}$$

$$S_t = \frac{e^t - e^{-t}}{e^t + e^{-t}} \tag{5.3}$$

5.2.3 Deterministic vs Stochastic Binarization

In a binarized neural network, weights and activations are constrained to be either +1 or -1 [78]. In order to convert floating point values to these two values, there are

two ways to binarize. One way is deterministic binarization as shown in equation 5.4:

$$x^b = sign(x) = \begin{cases} +1 & x \geq 0 \\ -1 & otherwise \end{cases} \tag{5.4}$$

where $x^b$is the binarized approximation of real valued variable x. The other way is stochastic binarization as shown in Equation 5.5:

$$x^b = \begin{cases} +1 & with\ probability\ p = \sigma(x) \\ -1 & with\ probability\ 1 - p \end{cases} \tag{5.5}$$

where σ is the "hard sigmoid" function as shown in Fig. 5.2.

$$\sigma(x) = clip(\frac{x+1}{2}, 0, 1) = max(0, min(1, \frac{x+1}{2})) \tag{5.6}$$

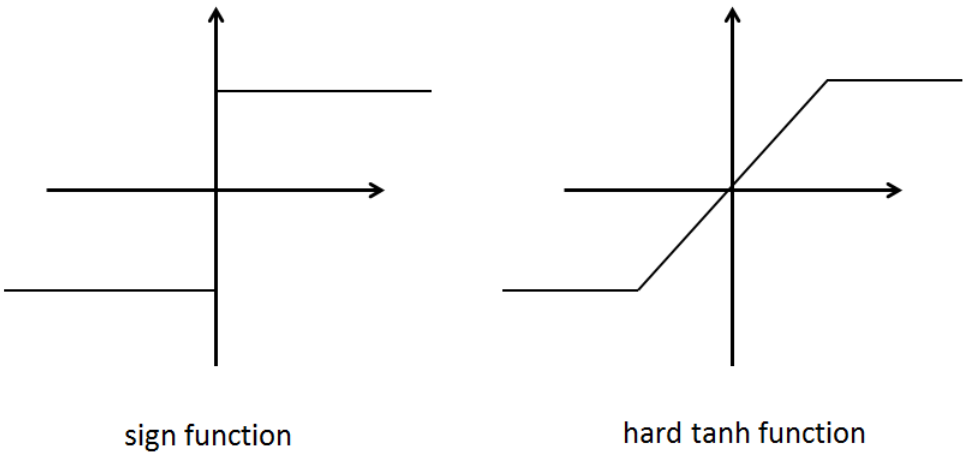

Figure 5.2: Backpropagation through hard tanh function

The stochastic binarization is more appealing than the deterministic binarization due to its better performance. However, deterministic binarization requires less com-

putation and hence is simpler to implement on hardware. In our work, we choose the deterministic binarization to reduce resource usage. Furthermore, together with deterministic binarization, we can optimize hardware implementation of the batch normalization layer, which will be explained in subsection 5.3.2.

5.2.4 Backpropagating Through Binarized Neuron

One problem with the above sign function is that the derivative of the sign function is zero almost everywhere. This problem remains even if stochastic binarization is used [78]. Such a property of binarization makes the gradient to be zero, nullifying backpropagation.

To tackle this problem, the straight-through estimator [100] is introduced. For instance, consider a deterministic binarization function as equation 5.7:

$$q = sign(r) \tag{5.7}$$

$$g_q = \frac{\partial C}{\partial q} \tag{5.8}$$

In equation 5.8, g_q is the estimator of gradient $\frac{\partial C}{\partial q}$. Hence, the straight-through estimator is:

$$g_r = \frac{\partial C}{\partial r} = \frac{\partial C}{\partial q} \times \frac{\partial q}{\partial r} = g_q 1_{\|r\| \leq 1} \tag{5.9}$$

In equation 5.9, the derivative $1_{\|r\| \leq 1}$ can be regarded as propagating the gradient through the hard tanh function. The hard tanh function is shown in equation 5.10:

$$htanh(x) = clip(x, -1, 1) = max(-1, min(1, x)) \tag{5.10}$$

When r becomes too large, the performance of the network suffers, therefore the gradient should be canceled [78]. As illustrated in Fig. 5.2, the binarized neural network uses sign function as the non-linear function. When backpropagating through the neural network, since sign function's gradient is zero almost everywhere, clipped hard tanh function is used to approximate the actual gradient as shown in Fig. 5.2.

5.2.5 Batch Normalization Layer

Batch normalization is meant to accelerate deep neural network training by reducing internal covariate shift [101]. One big problem with training deep neural network is that the distribution of each layer's inputs change during training due to the change of weights. This effect slows down the training, and makes the model hard to train using saturating non-linearities. Such a phenomenon is called internal covariate shift, and one way to counter it is to normalize each layer's inputs. Batch normalization allows a higher training rate and takes care of bad initialization. Besides, through experiments on our binarized neural network, without batch normalization layer, training becomes very slow to converge, and the final result is not as good as the model with batch normalization. In our work, batch normalization is inserted after each convolutional layer and fully connected layer, before applying non-linearities. We used the following expression for batch normalization:

$$y = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}\gamma + \beta \tag{5.11}$$

In equation 5.11, x is the input data , y is output data. During training, μ and

σ^2 are mean and variance of current input mini batch x, and during testing they are replaced by average statistic over the entire training data. Batch normalization layer forces activations through the network to take a unit Gaussian distribution, accelerates the training and also reduces the overall impact of weights' scale.

5.3 Hardware Architecture of Binarized Neural Network

This section presents the hardware architecture of the binarized neural network. We have a total of two convolutional layers and two fully connected layers. In our work, we propose an all-in-one chip solution in which all weights are stored in on-chip memory. The primary reason under the hood is that most of the power consumption comes from the data exchange between the chip and the external memory [91]. Since low power is more desired than low area for embedded system [102]. In our work, we primarily focus on reducing chip power consumption.

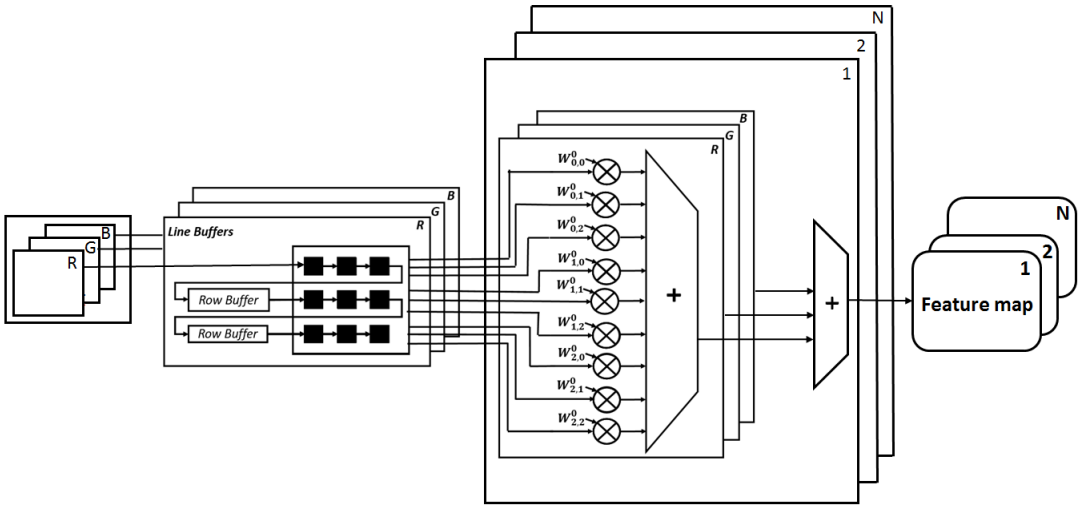

Figure 5.3: First Convolutional Layer Structure

5.3.1 First Convolutional Layer

In our binarized neural network, input image size is constantly 32-by-32, with RGB image format. After convolution, a total of N feature maps are generated as illustrated in Fig. 5.3. In the first convolutional layer, input image pixel value is not binarized, which still remains eight bit per pixel. Since one input feature channel generates N feature maps after convolution, we need one set of line buffers for each input channel. Extra buffers are inserted to avoid a large fanout in ASIC. Since weights in the first convolutional layer are all binarized, we further optimize this operation by splitting the operation into eight multiplexers, with binarized weight being the selection bit. After then, we shift each bit and add them up to get the result.

5.3.2 Optimized Batch Normalization Layer

Batch normalization has a non-linear equation as shown in equation 5.11. It is resource consuming to implement non-linear and floating point operations on VLSI. To implement equation 5.11, we need at least one high precision multiplier. One feature we noticed in the network is that right after each batch normalization layer, there is a non-linear layer. In our binarized neural network, the non-linear layer is the sign function, which quantizes output of batch normalization layer to +1 or -1. Since only the sign of the value is retained after passing through the non-linear layer, we found that the actual value of the batch normalization result does not need to be computed. Hence, for each batch normalization output y, we can perform the scaling using equation 5.12:

$$y\frac{\sqrt{\sigma^2+\epsilon}}{\gamma} = x - \mu + \beta\frac{\sqrt{\sigma^2+\epsilon}}{\gamma} \tag{5.12}$$

Since γ and $\sqrt{\sigma^2 + \varepsilon}$ are always positive, multiplying with $\frac{\sqrt{\sigma^2+\varepsilon}}{\gamma}$ doesn't change the sign of the output. In this chapter, we treat $\frac{\sqrt{\sigma^2+\varepsilon}}{\gamma} - \mu$ as a single coefficient, thus only one low precision adder is needed, largely reducing resource usage and computation efforts.

5.3.3 Binarized Convolutional Operation

Except for the first convolutional layer, which takes in image patches, all other convolutional layers take in binarized activations from previous layers. In the previous section, we binarize all the activations and weights to be +1 or -1. In hardware implementation, we constrain those values to be either 1 or 0, making them representable in one bit. With binary inputs, the XNOR gate is used to perform convolution in place of a multiplier. In our design, we use a 3-by-3 filter which is inspired by VGGNet [8]. Such a convolutional kernel is very efficient on digital implementation. In our implementation, we have a total number of 128-by-128 convolutional operations on a single layer. To make our design fully pipelined, we need to implement $128 \times 128 \times 3 \times 3 = 147,456$ XNOR operations in one clock cycle. In this way, we push our chip to its highest throughput to keep up with the exhaustive searching method.

However, not every computer vision system uses exhaustive search. Implementing $128 \times 128 \times 3 \times 3$ XNOR operations all at the same time seems unnecessary and wasteful considering resource usage and power consumption at times. Therefore, we propose an optimized implementation, which executes $128 \times 3 \times 3$ XNOR operations within one clock cycle, and utilizes dual port SRAM to store intermediate results. The proposed architecture is shown in Fig. 5.4.

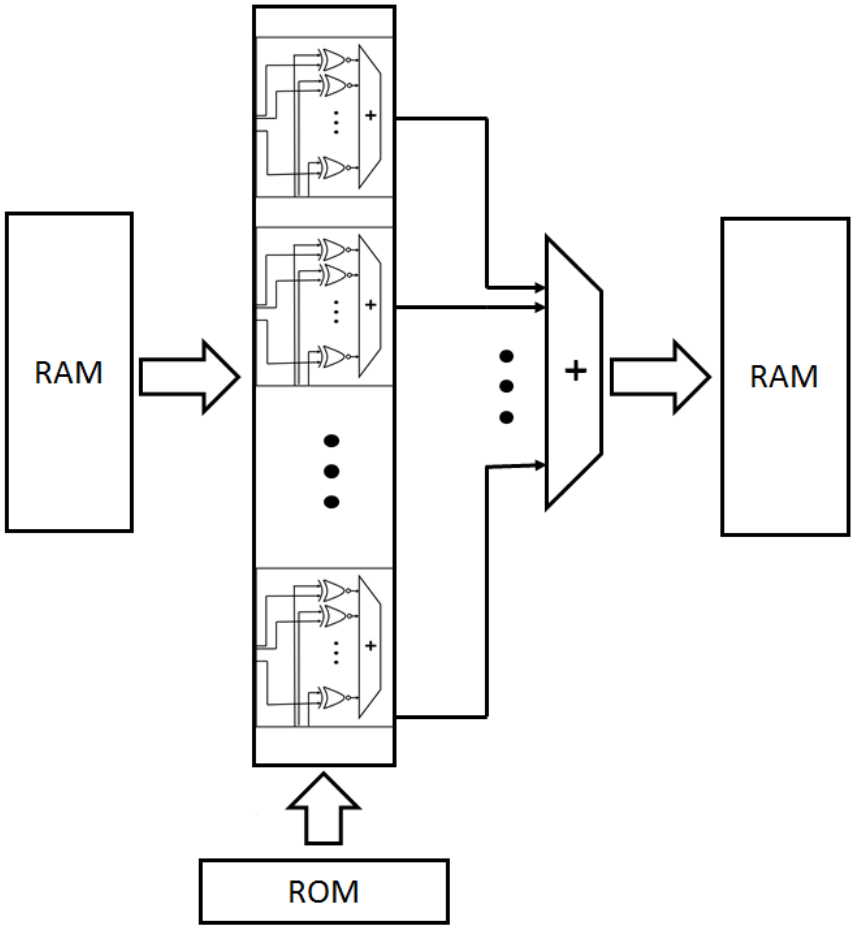

Figure 5.4: Low power architecture reuses XNOR-and-add units

5.3.4 ROM-Based Dense Layer

Unlike convolutional layers, in which a convolutional kernel is needed to process the entire input feature map, dense layers need one parameter for each input from previous feature maps. That means each parameter in a dense layer is only used once. In this way, we can save all parameters in ROM, and at each clock cycle one parameter is read out and multiplied with its corresponding feature value. The structure of dense layer is shown in Fig. 5.5.

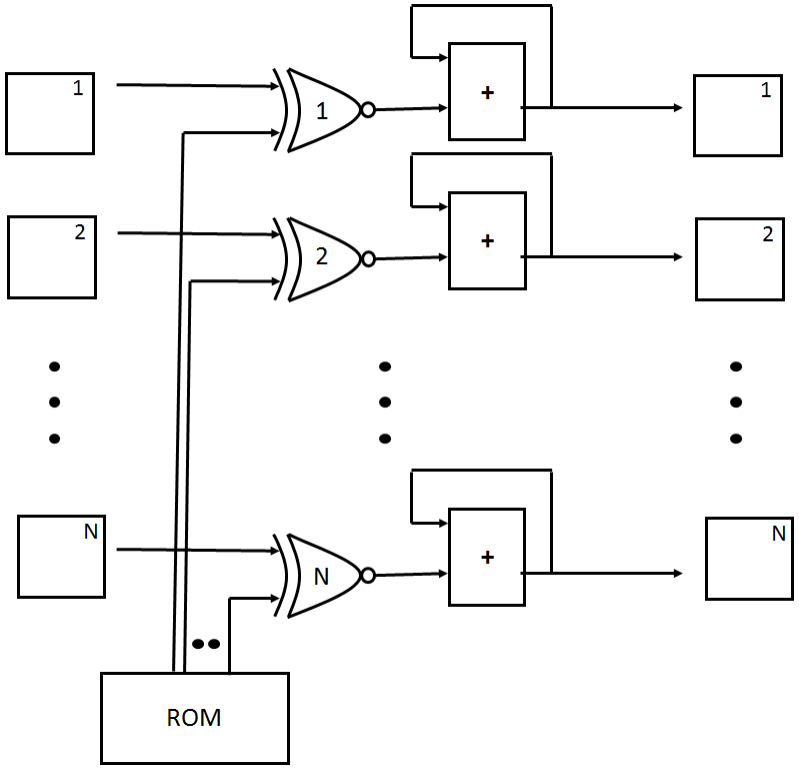

Figure 5.5: Fully connected layer stores weights in on-chip ROM

5.3.5 AMBA Bus Support

ARM microprocessors are dominant in the embedded computing world because of their low power. Our chip is meant for mobile applications, hence we designed ARM AMBA (Advanced Microcontroller Bus Architecture) bus interface for the chip. Our chip supports both AMBA AXI4 Lite and AXI4 Stream bus interfaces, supporting register access mode and burst mode respectively.

5.4 Performance on ADAS Applications

In our work, we target two most common ADAS applications: car detection and pedestrian detection. Fig. 5.6 shows an example of our training data. Among all

these applications, car detection is the most challenging task due to occlusions, variate views of angles, varying lighting conditions, and intra-class variations [13]. To tackle this, we need to collect more car samples to cover as many variations as possible. In this work, we collected our car samples from Cifar-10 dataset.

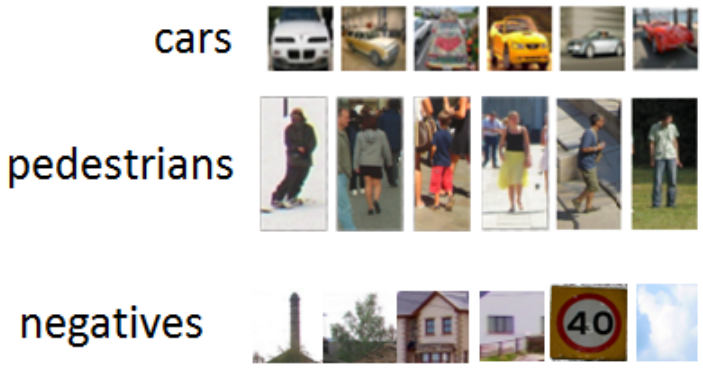

Figure 5.6: Our training samples from Cifar-10 and INRIA dataset

For pedestrian detection, since histogram of oriented gradients has first been used for people detection [17], more and more efficient detection framework had been proposed [103]. Hardware implementation of pedestrian detection has been proposed as well [104]. Overall, pedestrian detection task has been well solved [105].

Cars have 1:1 aspect ratio, while pedestrians have 2:1 aspect ratio normally. In order to put these two applications into one framework, we need to make them to be the same aspect ratio. The trick here is to only use the top half of a pedestrian sample.

In this work, we proposed a novel framework design of an ADAS chip. Firstly, we evaluated the performance of traditional computer vision algorithms. We used the classical Histogram of Gradient Orientation (HOG) plus Support Vector Machine (SVM) for car detection and pedestrian detection. The overall performance of HOG and SVM is shown in Table 5.1. Then, we trained various binarized neural net-

works for these two classes of objects. The BNN performance is shown in Table 5.2. Compared to the HOG algorithm, our binarized neural network generally has 6% improvement on precision and recall rates. Further, we also trained a regular convolutional neural network with the same configuration, the result is shown in Table 5.3. Comparing Table 5.2 against Table 5.3, we can see that our binarized neural network's performance is very close to a regular convolutional neural network's performance, while gaining 32 times memory saving. The Equations 5.13 and 5.14 are used to calculate precision and recall rates.

$$precision = \frac{true\,positives}{true\,positives + false\,positives} \tag{5.13}$$

$$recall = \frac{true\,positives}{true\,positives + false\,negatives} \tag{5.14}$$

Table 5.1: HOG+SVM Classification Accuracy

	precision rate	recall rate	positive samples
Car	85.37%	84.00%	500
Pedestrian	92.11%	88.38%	190
Negative Samples	95.25%	94.82%	2000

Table 5.2: BNN Classification Accuracy

	precision rate	recall rate	positive samples
Car	94.47%	92.20%	500
Pedestrian	92.35%	95.26%	190
Negative Samples	97.61%	97.90%	2000

Table 5.3: CNN Classification Accuracy

	precision rate	recall rate	positive samples
Car	94.68%	92.60%	500
Pedestrian	92.82%	95.26%	190
Negative Samples	97.71%	98.00%	2000

5.5 VLSI Results

The BNN accelerator is implemented using Synopsys 32nm process technology. The maximum frequency of the chip is 350MHz, and the chip area is $5.05mm^2$. The BNN chip consumes 0.6W in total, which is suitable for most of the mobile applications. The throughput calculation is given as:

$$throughput = \frac{1}{C \times T} \tag{5.15}$$

where C is total cycles needed to process a 32-by-32 image candidate, T is the period of one clock cycle. The processing unit generates additional 10 clock cycles delay from its adder tree. Hence, the 2-layer BNN's throughput given as:

$$\frac{350 \times 10^6}{128 + 128 + 10 + 128 + 10 + 3 + 10} = 839,328 \tag{5.16}$$

Distinct number of convolutional layers are also tested on the chip. Table 5.4 lists the throughput executing different number of convolutional layers.

According to the throughput table, our chip can be used with most of the mainstream candidate proposal methods, such as selective search [106] and edgeboxes [107]. The reported 0.6W power consumption makes our implementation deployable for most

Table 5.4: Different layers' throughput comparison

Conv. Layer Num.	throughput / candidates per second
2	839,328
3	431,565
4	324,976
5	188,679
6	155,624

of embedded platforms.

Besides binarization, other techniques are also applied to compress existing CNN models. Low-rank factorization is an effective method to estimate informative parameters by using matrix/tensor decomposition [108]. Moreover, special structural convolutional filters could be trained to save parameters. The efficient inference engine (EIE) is one efficient implementation of compressed deep neural network [109], and it is capable of processing 1.88×10^4 frames/sec FC layers of AlexNet. Another state-of-the-art implementation is the YodaNN, which is an ultra-low power CNN accelerator based on binary weights [110]. Since the bottleneck of processing a CNN is usually the memory bandwidth, a massively parallel in-memory processing architecture is proposed and claims to archieve high power efficiency [111]. Table 5.5 presents the comparison of our BNN accelerator with other compressed CNN accelerators.

Table 5.5: Comparison of our BNN accelerator with other compressed CNN accelerators

	Our work	EIE [109]	YodaNN [110]	BRein [111]
Chip Area/mm^2	5.05	40.8	2.16	3.9
Power/W	0.6	0.59	0.15	0.6
Max Freq./MHz	350	800	400	400
VLSI Process/mm	32	45	65	65
Throughput	5.7TOPS	102GOPS	1510GOPS	2.3TOPS
Power Efficiency	9.5TOPS/W	172GOPS/W	10.07TOPS/W	3.83TOPS/W

Throught comparison, we can see that our BNN accelerator achieves close to YodaNN power efficiency due to the merit of binarization. One major limitation

of BNN is that the performance is significantly lowered when dealing with large CNNs [108]. Hence, 4-bit or 8-bit fixed quantization are more commonly seen in larger convolutional neural networks.

5.6 Conclusion

In this chapter, the BNN is proposed as a potential feature extractor and classifier for intelligent vehicles. Two popular object detection tasks are selected to demonstrate the BNN's performance. For benchmarks of car and pedestrian classification, the BNN achieves an accuracy of over 95%, outperforming the traditional HOG algorithm. Compared to a CNN with the same configuration, BNN has less than 1% accuracy loss while gaining a few ten times resource advantages. Furthermore, an elastic architecture is designed for BNN using Synopsys 32nm process technology, realizing high throughput and low power consumption at the same time. Our work shows that the BNN has enormous potential to be the baseline classifier for intelligent vehicle object detection applications.

Chapter 6

Conclusions

6.1 Summery of Results

This dissertation is devoted to accelerating computer vision and machine learning algorithms for vision based intelligent vehicles solutions. We design optimized hardware architecture on either FPGA or ASIC for vision based ADAS applications.

- Firstly, we propose a real time lane detection system on FPGA. We adopt the hough transform algorithm for lane detection, and propose an efficient implementation of the hough transform algorithm on FPGA fabric. Compared to the standard hough transform implementation, we successfully reduce 50% multiplier usage in digital circuits. Implementing on the Xilinx Zynq platform, the proposed system reaches a maximum operating frequency of 143.85MHz and a processing speed of 69.4 frames per second at 1080P video input.

- Secondly, we propose an efficient SOC architecture for traffic sign classification. Since traffic signs are made to have human made shapes and colors, the traditional HOG algorithm is very suitable for feature extraction. Consider-

ing resource usage on hardware, we use linear SVM as the classifier for decision making. We proved that the HoG + SVM algorithm achieves good performance on BelgiumTS dataset. Furthermore, we accelerated the HOG and SVM algorithms in FPGA fabric, achieving a maximum operating frequency of 241.7MHz. The proposed system is capable of processing 116 frames per second at 1080P resolution. Compared to typically hundreds of millisecond latency on a general CPU, the SOC realization of traffic sign classification generates only 6.5us latency, which is neglectable for embedded systems.

- Thirdly, we propose an end-to-end SOC architecture for traffic light recognition. Regarding the task, we collect our own dataset in local Worcester, MA. For traffic light recognition, we still adopt the HOG + SVM algorithms. In this task, we leverage color information to extract region-of-interest. After binarizing input images, we apply BLOB analysis to propose potential candidates of traffic lights. Then, we use the SVM algorithms to decide whether current candidate is a traffic light no not. By accelerating BLOB detection method in FPGA fabric, the entire system can run in real time. The highest frequency of the system is 150.1MHz, and the maximum throughput is 72.4 frames per second with 1080P video streaming input.

- Fourthly, we design an efficient digital circuit architecture for the PCANet. ADAS designers often face the dilemma to balance between algorithm robustness and real time processing capability. The PCANet algorithm has trainable filters, which certainly outperforms traditional hand-crafted computer vision algorithms in various conditions. Also, the PCANet achieves similar accuracy with state-of-the-art CNNs while retaining a much simpler structure. Hence,

the PCA based network is proposed to be a baseline feature detector for ADAS. Furthermore, we accelerate the PCANet using Synopsys 32nm process technology. The proposed architecture only consumes 0.5 watt, and is still capable of classifying over 742K image candidates in one second.

- Finally, we propose a highly efficient architecture of binary neural network for ADAS. Compared to a regular CNN, the BNN binarizes internal activations and weights, reducing memory usage and computation at the same time. Moreover, the BNN retains comparable accuracy with the CNN. In this dissertation, we show that the BNN works well on tasks like pedestrian detection and car detection. By implementing the BNN in Synopsys 32nm process technology, a maximum operating frequency 350MHz is achieved with only 0.6 watt power consumption. The proposed BNN architecture is general and could be extended to other computer vision applications as well.

6.2 Recommendations for Future Work

In this dissertation, we mainly investigated the digital circuit implementations of vision based solutions for intelligent vehicles. In recent years, we notice that LiDAR is becoming cheaper and lighter for intelligent vehicles. LiDAR measures distance to a target by illuminating that target with a pulsed laser light and measuring the reflected pulses with a sensor. Hence, LiDAR is a useful complementary sensor for CMOS sensors. One potential work could be studied further is LiDAR data processing on digital circuits, thus creating a dense point cloud of the road, and better understanding the driving environments. Moreover, to build a complete sensing kit for a modern intelligent vehicle, sensors like stereo cameras, Radar, ultrasonic sensor

are used as well. In this way, sensor fusion is needed to fuse sensing information together and provide a robust prediction results to drivers.

In this dissertation, we mainly studied the object detection methods and their hardware acceleration. In ADAS applications, besides object detection, road segmentation is also an important and necessary task. The road segmentation serves as the initial step of a completed ADAS by providing drivable areas for further processing like path planning. Unlike object detection, road segmentation requires each pixel in the input image to be labeled, and hence needs lots of computation as well. The hardware acceleration of road segmentation methods is worth investigating and together with hardware accelerated object detection method, can serve as a complete solution for ADAS.

Bibliography

[1] M. B. Jensen, M. P. Philipsen, A. Møgelmose, T. B. Moeslund, and M. M. Trivedi, "Vision for looking at traffic lights: Issues, survey, and perspectives," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 7, pp. 1800–1815, 2016.

[2] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1. IEEE, 2001, pp. I–I.

[3] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[4] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," *Computer vision–ECCV 2006*, pp. 404–417, 2006.

[5] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[6] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[11] 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.

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

[13] S. Sivaraman and M. M. Trivedi, "Looking at vehicles on the road: A survey of vision-based vehicle detection, tracking, and behavior analysis," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 4, pp. 1773–1795, 2013.

[14] M. Oren, C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio, "Pedestrian detection using wavelet templates," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun 1997, pp. 193–199.

[15] M. Mody, P. Swami, K. Chitnis, S. Jagannathan, K. Desappan, A. Jain, D. Poddar, Z. Nikolic, P. Viswanath, M. Mathew *et al.*, "High performance front camera adas applications on ti's tda3x platform," in *High Performance Computing (HiPC), 2015 IEEE 22nd International Conference on.* IEEE, 2015, pp. 456–463.

[16] P. Morignot, J. P. Rastelli, and F. Nashashibi, "Arbitration for balancing control between the driver and adas systems in an automated vehicle: Survey and approach," in *Intelligent Vehicles Symposium Proceedings, 2014 IEEE.* IEEE, 2014, pp. 575–580.

[17] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[18] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems*, 2015, pp. 3123–3131.

[19] A. B. Hillel, R. Lerner, D. Levi, and G. Raz, "Recent progress in road and lane detection: a survey," *Machine vision and applications*, vol. 25, no. 3, pp. 727–745, 2014.

[20] S. Beucher and M. Bilodeau, "Road segmentation and obstacle detection by a fast watershed transformation," in *Intelligent Vehicles' 94 Symposium, Proceedings of the*. IEEE, 1994, pp. 296–301.

[21] H.-Y. Cheng, B.-S. Jeng, P.-T. Tseng, and K.-C. Fan, "Lane detection with moving vehicles in the traffic scenes," *IEEE Transactions on intelligent transportation systems*, vol. 7, no. 4, pp. 571–582, 2006.

[22] R. Danescu and S. Nedevschi, "Probabilistic lane tracking in difficult road scenarios using stereovision," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 2, pp. 272–282, 2009.

[23] R. Jiang, R. Klette, T. Vaudrey, and S. Wang, "New lane model and distance transform for lane detection and tracking," in *Computer Analysis of Images and Patterns*. Springer, 2009, pp. 1044–1052.

[24] U. Hofmann, A. Rieder, and E. D. Dickmanns, "Radar and vision data fusion for hybrid adaptive cruise control on highways," *Machine Vision and Applications*, vol. 14, no. 1, pp. 42–49, 2003.

[25] S.-J. Wu, H.-H. Chiang, J.-W. Perng, C.-J. Chen, B.-F. Wu, and T.-T. Lee, "The heterogeneous systems integration design and implementation for lane keeping on a vehicle," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 2, pp. 246–263, 2008.

[26] Y. Jiang, F. Gao, and G. Xu, "Computer vision-based multiple-lane detection on straight road and in a curve," in *Image Analysis and Signal Processing (IASP), 2010 International Conference on.* IEEE, 2010, pp. 114–117.

[27] A. A. Assidiq, O. O. Khalifa, M. R. Islam, and S. Khan, "Real time lane detection for autonomous vehicles," in *Computer and Communication Engineering, 2008. ICCCE 2008. International Conference on.* IEEE, 2008, pp. 82–88.

[28] E. Royer, M. Lhuillier, M. Dhome, and J.-M. Lavest, "Monocular vision for mobile robot localization and autonomous navigation," *International Journal of Computer Vision*, vol. 74, no. 3, pp. 237–260, 2007.

[29] X. Miao, S. Li, and H. Shen, "On-board lane detection system for intelligent vehicle based on monocular vision." *International Journal on Smart Sensing & Intelligent Systems*, vol. 5, no. 4, 2012.

[30] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc.* Strathclyde Academic Media, 2014.

[31] A. Borkar, M. Hayes, M. T. Smith, and S. Pankanti, "A layered approach to robust lane detection at night," in *Computational Intelligence in Vehicles and Vehicular Systems, 2009. CIVVS'09. IEEE Workshop on.* IEEE, 2009, pp. 51–57.

[32] D. H. Ballard, "Generalizing the hough transform to detect arbitrary shapes," *Pattern recognition*, vol. 13, no. 2, pp. 111–122, 1981.

[33] J. Illingworth and J. Kittler, "A survey of the hough transform," *Computer vision, graphics, and image processing*, vol. 44, no. 1, pp. 87–116, 1988.

[34] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of fpga, gpu and cpu in image processing," in *Field programmable logic and applications, 2009. fpl 2009. international conference on.* IEEE, 2009, pp. 126–131.

[35] 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.

[36] P. Y. Hsiao, C. W. Yeh, S. S. Huang, and L. C. Fu, "A portable vision-based real-time lane departure warning system: Day and night," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 4, pp. 2089–2094, May 2009.

[37] C. Lipski, B. Scholz, K. Berger, C. Linz, T. Stich, and M. Magnor, "A fast and robust approach to lane marking detection and lane tracking," in *2008 IEEE Southwest Symposium on Image Analysis and Interpretation*, March 2008, pp. 57–60.

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

[39] J. A. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[40] M. Mathias, R. Timofte, R. Benenson, and L. V. Gool, "Traffic sign recognition: How far are we from the solution?" in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, Aug 2013, pp. 1–8.

[41] D. Forsyth and J. Ponce, *Computer vision: a modern approach*. Upper Saddle River, NJ; London: Prentice Hall, 2011.

[42] O. Déniz, G. Bueno, J. Salido, and F. De la Torre, "Face recognition using histograms of oriented gradients," *Pattern Recognition Letters*, vol. 32, no. 12, pp. 1598–1603, 2011.

[43] I. M. Creusen, R. G. Wijnhoven, E. Herbschleb, and P. de With, "Color exploitation in hog-based traffic sign detection," in *Image Processing (ICIP), 2010 17th IEEE International Conference on*. IEEE, 2010, pp. 2669–2672.

[44] X. Qingsong, S. Juan, and L. Tiantian, "A detection and recognition method for prohibition traffic signs," in *Image Analysis and Signal Processing (IASP), 2010 International Conference on*. IEEE, 2010, pp. 583–586.

[45] T. Groleat, M. Arzel, and S. Vaton, "Hardware acceleration of svm-based traffic classification on fpga," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2012 8th International*. IEEE, 2012, pp. 443–449.

[46] S. Bauer, S. Köhler, K. Doll, and U. Brunsmann, "Fpga-gpu architecture for kernel svm pedestrian detection," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*. IEEE, 2010, pp. 61–68.

[47] O. Pina-Ramirez, R. Valdes-Cristerna, and O. Yanez-Suarez, "An fpga implementation of linear kernel support vector machines," in *Reconfigurable Computing and FPGA's, 2006. ReConFig 2006. IEEE International Conference on*. IEEE, 2006, pp. 1–6.

[48] M. Papadonikolakis and C.-S. Bouganis, "A novel fpga-based svm classifier," in *Field-Programmable Technology (FPT), 2010 International Conference on*. IEEE, 2010, pp. 283–286.

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

[50] Y.-C. Chung, J.-M. Wang, and S.-W. Chen, "A vision-based traffic light detection system at intersections," *åÿńåďğåŋÿåăś: œŢÿçŘĘèĹĞçğŚœŁĂéąđ*, vol. 47, no. 1, pp. 67–86, 2002.

[51] 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.

[52] 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.

[53] 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.

[54] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark," in *International Joint Conference on Neural Networks*, no. 1288, 2013.

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

[56] S. Waite and E. Oruklu, "Fpga-based traffic sign recognition for advanced driver assistance systems," *Journal of Transportation technologies*, vol. 3, no. 01, p. 1, 2013.

[57] J. Zhao, S. Zhu, and X. Huang, "Real-time traffic sign detection using surf features on fpga," in *2013 IEEE High Performance Extreme Computing Conference (HPEC)*, Sept 2013, pp. 1–6.

[58] A. Nikonorov, P. Yakimov, and P. Maksimov, "Traffic sign detection on gpu using color shape regular expressions," *VISIGRAPP IMTA-4*, vol. 2013, 2013.

[59] Z. Chen, X. Huang, Z. Ni, and H. He, "A gpu-based real-time traffic sign detection and recognition system," in *Computational Intelligence in Vehicles and Transportation Systems (CIVTS), 2014 IEEE Symposium on.* IEEE, 2014, pp. 1–5.

[60] M. Mathias, R. Timofte, R. Benenson, and L. Van Gool, "Traffic sign recognitionâĂŤhow far are we from the solution?" in *Neural Networks (IJCNN), The 2013 International Joint Conference on.* IEEE, 2013, pp. 1–8.

[61] Z. Shen, K. Wang, and F. Zhu, "Agent-based traffic simulation and traffic signal timing optimization with gpu," in *Intelligent transportation systems (itsc), 2011 14th international ieee conference on*. IEEE, 2011, pp. 145–150.

[62] Y. Ji, M. Yang, Z. Lu, and C. Wang, "Integrating visual selective attention model with hog features for traffic light detection and recognition," in *Intelligent Vehicles Symposium (IV), 2015 IEEE*. IEEE, 2015, pp. 280–285.

[63] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, "Pcanet: A simple deep learning baseline for image classification?" *arXiv preprint arXiv:1404.3606*, 2014.

[64] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[65] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[66] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[67] P. Dollár, S. J. Belongie, and P. Perona, "The fastest pedestrian detector in the west." in *BMVC*, vol. 2, no. 3. Citeseer, 2010, p. 7.

[68] J. Bruna and S. Mallat, "Invariant scattering convolution networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1872–1886, 2013.

[69] L. Sifre and S. Mallat, "Rotation, scaling and deformation invariant scattering for texture discrimination," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 1233–1240.

[70] P.-H. Pham, D. Jelaca, C. Farabet, B. Martini, Y. LeCun, and E. Culurciello, "Neuflow: Dataflow vision processing system-on-a-chip," in *Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on*. IEEE, 2012, pp. 1044–1047.

[71] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A convolutional network accelerator," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*. ACM, 2015, pp. 199–204.

[72] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. A. Horowitz, "Convolution engine: balancing efficiency & flexibility in specialized computing," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3. ACM, 2013, pp. 24–35.

[73] D. K. Hu, L. Zhang, W. D. Zhao, and T. Yan, "Object classification via pcanet and color constancy model," in *Applied Mechanics and Materials*, vol. 635. Trans Tech Publ, 2014, pp. 997–1000.

[74] T. Wu and A. Ranganathan, "A practical system for road marking detection and recognition," in *Intelligent Vehicles Symposium (IV), 2012 IEEE*. IEEE, 2012, pp. 25–30.

[75] M.-M. Cheng, Z. Zhang, W.-Y. Lin, and P. Torr, "Bing: Binarized normed gradients for objectness estimation at 300fps," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 3286–3293.

[76] Y. Zhou, Z. Chen, and X. Huang, "A system-on-chip fpga design for real-time traffic signal recognition system," in *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 1778–1781.

[77] M. Mody, P. Swami, K. Chitnis, S. Jagannathan, K. Desappan, A. Jain, D. Poddar, Z. Nikolic, P. Viswanath, M. Mathew *et al.*, "High performance front camera adas applications on ti's tda3x platform," in *High Performance Computing (HiPC), 2015 IEEE 22nd International Conference on*. IEEE, 2015, pp. 456–463.

[78] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[79] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[80] E. Dagan, O. Mano, G. P. Stein, and A. Shashua, "Forward collision warning with a single camera," in *Intelligent Vehicles Symposium, 2004 IEEE*. IEEE, 2004, pp. 37–42.

[81] N. Srinivasa, "Vision-based vehicle detection and tracking method for forward collision warning in automobiles," in *Intelligent Vehicle Symposium, 2002. IEEE*, vol. 2, June 2002, pp. 626–631 vol.2.

[82] P. Dollar, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: An evaluation of the state of the art," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 4, pp. 743–761, April 2012.

[83] Y. Zhou, Z. Chen, and X. Huang, "A system-on-chip fpga design for real-time traffic signal recognition system," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2016, pp. 1778–1781.

[84] V. John, K. Yoneda, B. Qi, Z. Liu, and S. Mita, "Traffic light recognition in varying illumination using deep learning and saliency map," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, Oct 2014, pp. 2286–2291.

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

[86] G. P. 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.* IEEE, 2005, pp. 130–130.

[87] J. Sankaran and N. Zoran, "Tda2x, a soc optimized for advanced driver assistance systems," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on.* IEEE, 2014, pp. 2204–2208.

[88] J. Tanabe, S. Toru, Y. Yamada, T. Watanabe, M. Okumura, M. Nishiyama, T. Nomura, K. Oma, N. Sato, M. Banno *et al.*, "18.2 a 1.9 tops and 564gops/w heterogeneous multicore soc with color-based object classification accelerator for image-recognition applications," in *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International.* IEEE, 2015, pp. 1–3.

[89] K. J. Lee, K. Bong, C. Kim, J. Jang, K.-R. Lee, J. Lee, G. Kim, and H.-J. Yoo, "A 502-gops and 0.984-mw dual-mode intelligent adas soc with real-time semiglobal matching and intention prediction for smart automotive black box system," *IEEE Journal of Solid-State Circuits*, 2016.

[90] K. J. Lee, K. Bong, C. Kim, J. Jang, H. Kim, J. Lee, K.-R. Lee, G. Kim, and H.-J. Yoo, "14.2 a 502gops and 0.984 mw dual-mode adas soc with rnn-fis engine for intention prediction in automotive black-box system," in *Solid-State Circuits Conference (ISSCC), 2016 IEEE International*. IEEE, 2016, pp. 256–257.

[91] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[92] G. Lacey, G. W. Taylor, and S. Areibi, "Deep learning on fpgas: Past, present, and future," *arXiv preprint arXiv:1602.04283*, 2016.

[93] G. V. STOICA, R. DOGARU, and C. Stoica, "High performance cuda based cnn image processor," 2015.

[94] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2015, pp. 161–170.

[95] M. Kang, S. K. Gonugondla, M.-S. Keel, and N. R. Shanbhag, "An energy-efficient memory-based high-throughput vlsi architecture for convolutional networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1037–1041.

[96] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, 2016.

[97] 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.

[98] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[99] B.-P. Network, "Handwritten digit recognition with," 1989.

[100] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[101] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[102] S. Fatima and V. Vishwanatha, "Low power vlsi–survey on advanced power management technology," *Indian Journal of Scientific Research*, pp. 137–141, 2015.

[103] P. Dollár, S. J. Belongie, and P. Perona, "The fastest pedestrian detector in the west." in *BMVC*, vol. 2, no. 3. Citeseer, 2010, p. 7.

[104] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann, and K. Doll, "Fpga-based real-time pedestrian detection on high-resolution images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2013, pp. 629–635.

[105] R. Benenson, M. Omran, J. Hosang, and B. Schiele, "Ten years of pedestrian detection, what have we learned?" in *European Conference on Computer Vision*. Springer, 2014, pp. 613–627.

[106] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International journal of computer vision*, vol. 104, no. 2, pp. 154–171, 2013.

[107] C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges," in *European Conference on Computer Vision*. Springer, 2014, pp. 391–405.

[108] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," *arXiv preprint arXiv:1710.09282*, 2017.

[109] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 243–254.

[110] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "Yodann: An ultra-low power convolutional neural network accelerator based on binary weights," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2016, pp. 236–241.

[111] K. Ando, K. Ueyoshi, K. Orimo, H. Yonekawa, S. Sato, H. Nakahara, M. Ikebe, T. Asai, S. Takamaeda-Yamazaki, T. Kuroda, and M. Motomura, "Brein memory: A 13-layer 4.2 k neuron/0.8 m synapse binary/ternary reconfigurable in-memory deep neural network accelerator in 65 nm cmos," in *2017 Symposium on VLSI Circuits*, June 2017, pp. C24–C25.