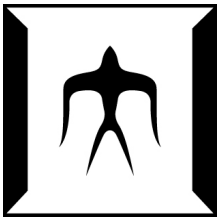
論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Application-Specific Instruction-Set Processor Architectures for Embedded Vision
著者(和文)	XIAOShanlin
Author(English)	Shanlin Xiao
出典(和文)	学位:博士(学術), 学位授与機関:東京工業大学, 報告番号:甲第10709号, 授与年月日:2017年12月31日, 学位の種別:課程博士, 審査員:一色 剛,上野 修一,髙橋 篤司,原 祐子,中原 啓貴,伊藤 和人
Citation(English)	Degree:Doctor (Academic), Conferring organization: Tokyo Institute of Technology, Report number:甲第10709号, Conferred date:2017/12/31, Degree Type:Course doctor, Examiner:,,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Application-Specific Instruction-Set Processor Architectures for Embedded Vision

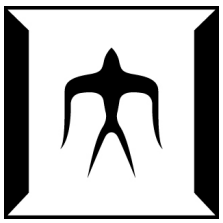


Department of Communications and Computer Engineering
Tokyo Institute of Technology

Shanlin Xiao

September 2017

Application-Specific Instruction-Set Processor Architectures for Embedded Vision



Shanlin Xiao

Advisor: Professor Tsuyoshi Isshiki

Department of Communications and Computer Engineering

Tokyo Institute of Technology

This dissertation is submitted for the degree of

Doctor of Philosophy

September 2017

Acknowledgements

First and foremost, I would like to express my gratitude and thanks to my advisor, Prof. Tsuyoshi Isshiki, for giving me the opportunity to do a Ph.D. study on such an interesting topic, and for his support during my Ph.D. career. He has been a great mentor. Without his guidance, encouragement and intuitive ideas, this thesis would not have been completed.

I would like to extend my gratitude to Prof. Hiroaki Kunieda and Assistant Prof. Dongju Li for sharing their knowledge with me and for providing suggestions on my research. I also grateful to Prof. Shuichi Ueno, Prof. Atsushi Takahashi, Associate Prof. Yuko Hara, Associate Prof. Hiroki Nakahara and Prof. Kazuhito Ito for serving on my defense committee.

It's impossible to mention everyone that has helped me over the years, but I would like to extend special thanks to Ms. Miwa Tashiro and Ms. Yumiko Kondo for their help and support throughout my stay in the laboratory.

Also, I would like to thank all the members of Isshiki laboratory for their support and assistance, including Dr. Hao Xiao, Dr. Zhiqiang Hu, Ms. Lily Tiong Yu Wen, Dr. Hsuan-Chun Liao, Mr. Yang Li, Mr. Pipat Methavanitpong, Ms. Nabilah Shabrina, Mr. Ikumi Endo, Mr. Koshiro Date, Mr. Keitarou Kojima, Mr. Kazuki Zenba, and Mr. Mustafa Abdul-Halim Yassin.

Finally, my greatest thanks to my parents, wife, and daughter, for their love, encouragement, and support. No matter where I am, the family is always my motivation moving forward.

Publication

Journal papers

1. **Shanlin Xiao**, Tsuyoshi Isshiki, Dongju Li, Hiroaki Kunieda, "Design of an Application Specific Instruction Set Processor for Real-Time Object Detection Using AdaBoost Algorithm," IEICE Transactions on Fundamentals, Vol. E100-A, No. 07, pp. 1384-1395, July. 2017.

2. **Shanlin Xiao**, Tsuyoshi Isshiki, Dongju Li, Hiroaki Kunieda, "HOG-Based Object Detection Processor Design Using ASIP Methodology," IEICE Transactions on Fundamentals, Vol. E100-A, No.12, pp. - , Dec. 2017.

Conference papers

1. **Shanlin Xiao**, Tsuyoshi Isshiki, Dongju Li, Hiroaki Kunieda, "An Efficient Embedded Processor for Object Detection Using ASIP Methodology," 2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP), pp. 225-226, London, England, July 2016.

2. **Shanlin Xiao**, Dongju Li, Hiroaki Kunieda, Tsuyoshi Isshiki, "Design of an Efficient ASIP-Based Processor for Object Detection Using AdaBoost Algorithm," 2016 7th International Conference of Information and Communication Technology for Embedded Systems (IC-ICTES), pp. 96-99, Bangkok, Thailand, Mar. 2016.

Abstract

Computer vision has been utilized to enable many applications and will affect many aspects of human life in the era of big data. Object detection is an essential and expensive process in almost all the computer vision systems. It involves the acquisition, processing, analysis, and understanding of real-world visual information. Standard off-the-shelf embedded processors are hard to meet the trade-offs among performance, power consumption, and flexibility required by object detection applications.

The Application-Specific Instruction-set Processor (ASIP) emerges as a promising solution to balance the performance/power/flexibility trade-offs. This approach combines a software programmable processor and application-specific hardware components. The functional units of ASIPs can execute in parallel and utilize special registers for internal communication. ASIPs can offer better performance and energy characteristic than GPPs/DSPs. The control of functional units in an ASIP is not fixed but taken over by an instruction decoder and a program. ASIPs can provide higher flexibility than ASICs. However, these benefits, in turn, involve a high-cost development flow. The traditional development of ASIP is a very demanding and complex task involving instruction set, micro-architecture, RTL, compiler, and debugger design.

The motivation of this dissertation is to study an efficient ASIP design methodology and to provide a design framework to support the ASIP-based embedded vision processor design. The design framework is based on an Architecture Description Language (ADL) and provides the processor architect with a seamless design flow that covers the software interface with an open source vision library, hardware/software partition, ASIP programming model,

hardware/software co-simulation and RTL implementation. Furthermore, on the basis of analyzing the vision algorithms, we provide some design considerations/guidelines to help the processor architect in designing an efficient embedded vision processor. Our thinking includes algorithms consideration, applications understanding, architecture consideration, and design reuse consideration.

To fully demonstrate the efficiency of the proposed framework and our thinking on vision processor design, we develop two ASIPs for embedded vision using the proposed framework as reference implementations. The algorithms are Haar-like features with AdaBoost classifier and Histogram of Oriented Gradients (HOG) features with Support Vector Machine (SVM) classifier. In the two ASIPs, we state the hardware/software partition principle, exploit the hardware-friendly object detection algorithms, perform the hardware/software co-design, and apply the design reusability to improve the design efficiency. The Single Instruction Multiple Data (SIMD) architecture is adopted for fully exploiting data-level parallelism inherent to the target algorithm. Multiple data reuse strategies are introduced to minimize the data movement and improve the power efficiency. Finally, the throughput, area, and power consumption estimation are carried out based on the synthesis results. The experimental results show that the ASIPs achieve 13x to 63x speed-up compared to its baseline processor. When compared with commercial GPP/DSP, the ASIPs have 7x to 113x better throughput, show 10x to 750x better area efficiency and 6x to 184 better power efficiency. Furthermore, the ASIPs show comparable performance with hard-wired designs.

Keywords: ASIP, design methodology, computer vision, object detection, machine learning.

Contents

List of Figures

List of Tables

Terminology

ADL Architecture Description Language

ALU Arithmetic Logic Unit

ASIC Application Specific Integrated Circuit

ASIP Application Specific Instruction-set Processor

CISC Complex Instruction Set Computer

CNN Convolutional Neural Network

CPU Central Processing Unit

DLP Data Level Parallelism

DMA Direct Memory Access

DSP Digital Signal Processor

FPGA Field Programmable Gate Array

GPP General Purpose Processor

GPR General Purpose Register

GPU Graphics Processing Unit

HDL Hardware Description Language

HLS High-Level Synthesis

HOG Histogram of Oriented Gradients

ISA Instruction Set Architecture

MAC Multiply Accumulate

MPSoC Multi-Processor System-on-Chip

ORB Oriented FAST and Rotated BRIEF

PE Processing Element

PR Precision-Recall

RISC Reduced Instruction Set Computer

ROC Receiver Operating Characteristic

SIFT Scale Invariant Feature Transform

SIMD Single Inctruction Multiple Data

SoC System-on-Chip

SRAM Static Random Access Memory

SURF Speeded Up Robust Features

SVM Support Vector Machine

TCT Tight Coupled Thread

VLIW Very Long Instruction Word

Chapter 1

Introduction

1.1 Background

The microprocessor has become a ubiquitous part of the world today. More advanced features are created and integrated with the modern microprocessors. Intelligent vision processing becomes one of the required features in the era of big data. The background and motivation of this dissertation are demonstrated as follows.

- **The ever-growing visual data.**

This is the era of big data. Hundreds of Zettabytes (10^{21} bytes) of data is created every year; Petabytes (10^{15} bytes) of data is generated per second [9]. The data volumes are exploding, more data has been created in the past two years than before in the history of human race [10].

The video is perhaps the biggest of the big data. It takes over 70% of today's Internet traffic [2]. For example, over 800 million hours of video is collected every day for video surveillance in 2015 [11]. With the exponential growth in the use of sensors (as shown in Figure 1.1, 10 billion in 2013, 1 trillion are expected by 2020 [1]) and connected devices (6.4 billion in 2016, 20.8 billion are expected by 2020 [12]), the visual data is in the trend of ever-growing.

- **The increasing computer vision applications.**

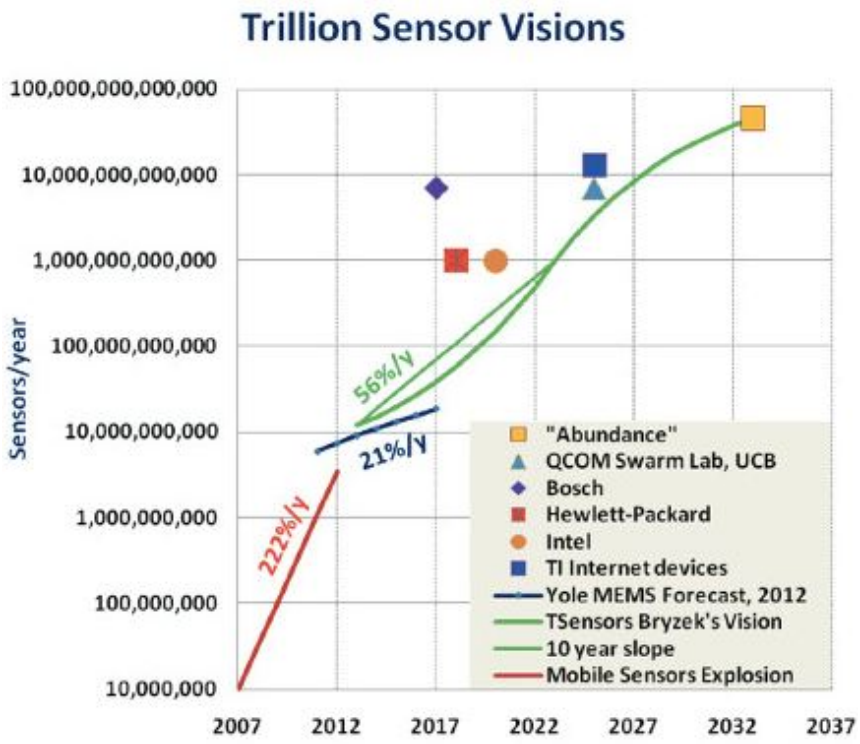


Figure 1.1 Trillion sensors visions [1].

In many visual applications, it would be desirable to extract the meaningful information from visual input using computer vision algorithms. Computer vision brings together image processing, machine learning, and pattern recognition to enable the machine to analyze, understand, and respond to the scenes.

Computer vision has been utilized to enable many different applications and will affect many aspects of human life. In home applications, vision technology plays an important role in the smart home and smart building. It can be used for security, presence detection, detecting unusual events, and remote monitoring etc. In industry, vision technology is applied to streamlining waste haulage. In agriculture, vision technology is being used for selectively harvest crops. In medicine, vision technology is used to prevent heart failure. Computer vision will be everywhere.

- **The shifting application focus.**

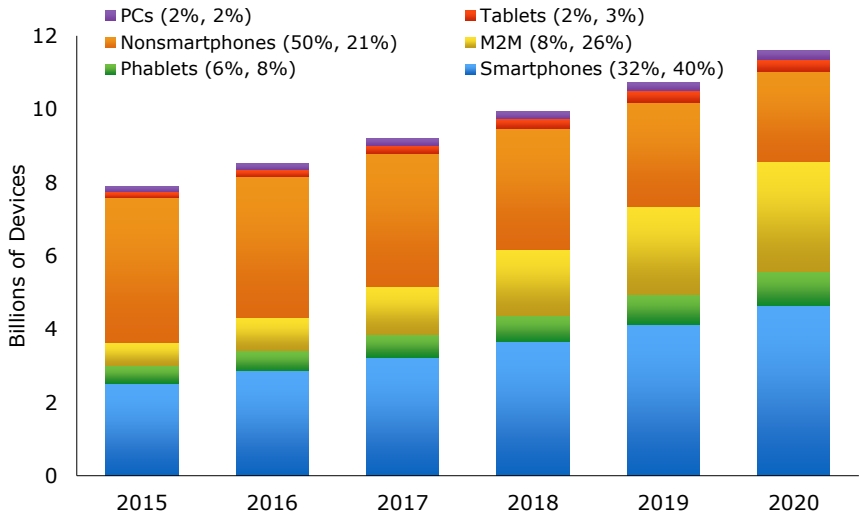


Figure 1.2 Global mobile devices and connections growth [2].

The application focus is shifting from desktop to individual and mobile devices. In 2015, global mobile devices and connections are up to 7.9 billion. More than half a billion (563 million) of them were added in 2015 [2]. Globally, as shown in Figure 1.2, mobile devices and connections will grow to 11.6 billion by 2020 at a CAGR (Compound Annual Growth Rate, CAGR) of 8 percent. By 2020, there will be 8.2 billion hand-held or personal mobile-ready devices [2].

- **The shifting design paradigm.**

Programmable accelerators are gaining popularity in embedded systems. The Application-Specific Instruction-set Processor (ASIP) emerges as an attractive hardware/software co-design solution [13, 14, 7, 15]. Figure 1.3 shows the relationship of design abstraction and design productivity. Unlike the traditional RTL design methodology, in which the basic design entity is the Finite State Machine (FSM), the HW/SW co-design approach is to replace these FSMs with programmable processing elements.

The ASIP-based processor becomes an attractive design point for two main reasons. First, programmability can lower the cost and risk of RTL design. Software features are added to the design flow. Making a change to software is cheaper and faster than that to hardware. Functionality can be constructed in an iterative approach, and problems can be fixed as soon as they are found. Second,

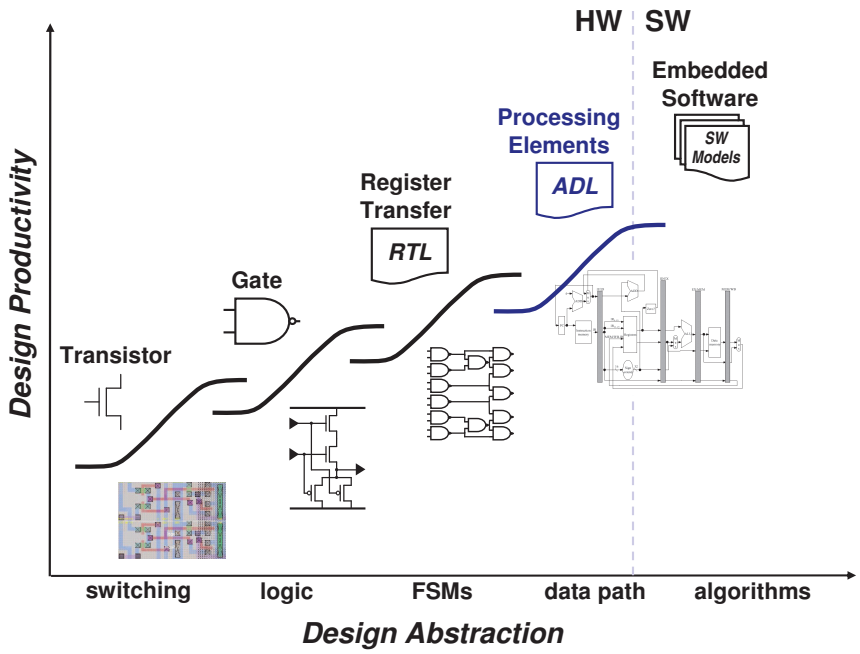


Figure 1.3 The programmable accelerator design discontinuity [3]

programmable systems can improve the design productivity. The architect can build up a more complex system in a shorter time based on a coarser block datapath other than a state machine. since the Instruction Set Architecture (ISA) and the clock rate characterize the performance and behavior of a datapath. Programmers can reliably predict the performance and behavior of a software application using computer science instead of physics.

1.2 Motivation

These trends are attracting the focus and interests from both academia and industry to explore Application-Specific Instruction-set Processors (ASIPs) for the emerging computer vision applications. However, the traditional development of ASIP is a very demanding and complex task involving instruction set, micro-architecture, RTL, compiler, and debugger design.

The motivation of this research is to study an efficient ASIP design methodology and to provide a complete design framework to support ASIP-based embedded vision processor design. The design framework is based on an Archi-

tecture Description Language (ADL) and provides the processor architect with a seamless design flow that covers the SW interface with an open source vision library, HW/SW partition, ASIP programming model, HW/SW co-simulation, and RTL implementation. Furthermore, on the basis of analyzing the vision algorithms, we provide some design considerations/guidelines to help the processor architect in designing an efficient embedded vision processor. Our thinking include algorithms consideration, applications understanding, architecture consideration, and design reuse consideration. As reference implementations, we demonstrate the effectiveness of the proposed framework using two real-life applications, Haar-like features with AdaBoost classifier and Histogram of Oriented Gradients (HOG) features with Support Vector Machine (SVM) classifier for object detection.

1.3 Contribution

The contribution of this dissertation is summarized as follows.

1. A complete framework is proposed to support the ASIP-based embedded vision processor design.

The proposed design framework covers from the high-level ASIP programming model to the low-level micro-architecture. It is aimed to facilitate the application-to-architecture mapping at the architect level. To expedite the design cycle, our ASIP design methodology is integrated with high-level abstraction Architecture Description Language (ADL) and commercial design tool. The framework also provides an interface with open source computer vision library, for easily maintain the evolution of vision algorithms.

2. Some design thinking are provided as references for vision processor architects.

These thinking include algorithms consideration, applications understanding, architecture consideration, and design reuse consideration.

3. A thorough analysis on representative object detection techniques and the proposed hardware-friendly object detection algorithms.

We conduct a comprehensive study on several representative object detection techniques, especially the ones using hand-crafted features with machine learning classifiers, e.g. Haar-like features with AdaBoost classifier and HOG features with SVM classifier. The study aims to identify the computational intensive tasks, extract the key computationally tasks, and exploit the data locality properties within the algorithm. The thorough analysis establishes a solid foundation for the design of object detection processors.

We careful reschedule the dataflow within the AdaBoost-based learning algorithm and HOG-based algorithm. The new dataflow is aimed at reducing the memory bandwidth requirements and redundant computations. Also, the simplified AdaBoost algorithm and HOG algorithm are introduced to minimize the hardware costs and reduce the mathematical complexity. The modified algorithms are evaluated on several popular faces and human datasets. The evaluation results show that the modified algorithms still maintain the high accuracy as well as the original algorithms.

4. As demonstrations, we show the effectiveness of the proposed framework using two real-life applications, Haar-like features with AdaBoost classifier and Histogram of Oriented Gradients (HOG) features with Support Vector Machine (SVM) classifier for object detection.

Two synthesized ASIP-based processors are designed and implemented for AdaBoost algorithm and HOG algorithm. To meet the real-time requirements, we propose several special functional units to accelerate the computationally intensive tasks in these object detection algorithm. The special functional units include integral image calculation array, pipeline Haar-like feature computation, and parallel histogram generation etc. To achieve low-power consumption, an

efficient classifier data storage mechanism for AdaBoost and on-the-fly SVM classification are proposed to minimize the data movement. With proper application-to-architecture mapping, the proposed ASIPs achieve high throughput in a small area, power, and energy footprint.

A full comparison between the proposed ASIP and the conventional embedded processors and hard-wired designs is carried out in terms of performance, area, and power consumption. The proposed ASIP exhibits an advantage in terms of both chip area-efficiency and power-efficiency when compared to embedded processors, and shows a competitive computational performance when compared to hard-wired designs. To the best of our knowledge, there is no similar work for AdaBoost algorithm and HOG algorithm using ASIP design methodology so far.

1.4 Thesis Organization

The rest of this dissertation is organized as follows.

Chapter 2 introduces the background knowledge of embedded vision techniques and the conventional hardware architecture of vision processing engine.

Chapter 3 demonstrates the proposed ASIP design framework for embedded vision and our considerations on vision processors design.

Chapter 4 focus on the design and evaluation of the object detection processor with Haar-like features and AdaBoost classifier.

Chapter 5 is dedicated to the design and evaluation of the object detection processor with HOG features and SVM classifier.

Chapter 6 concludes the thesis, with future work perspectives.

Chapter 2

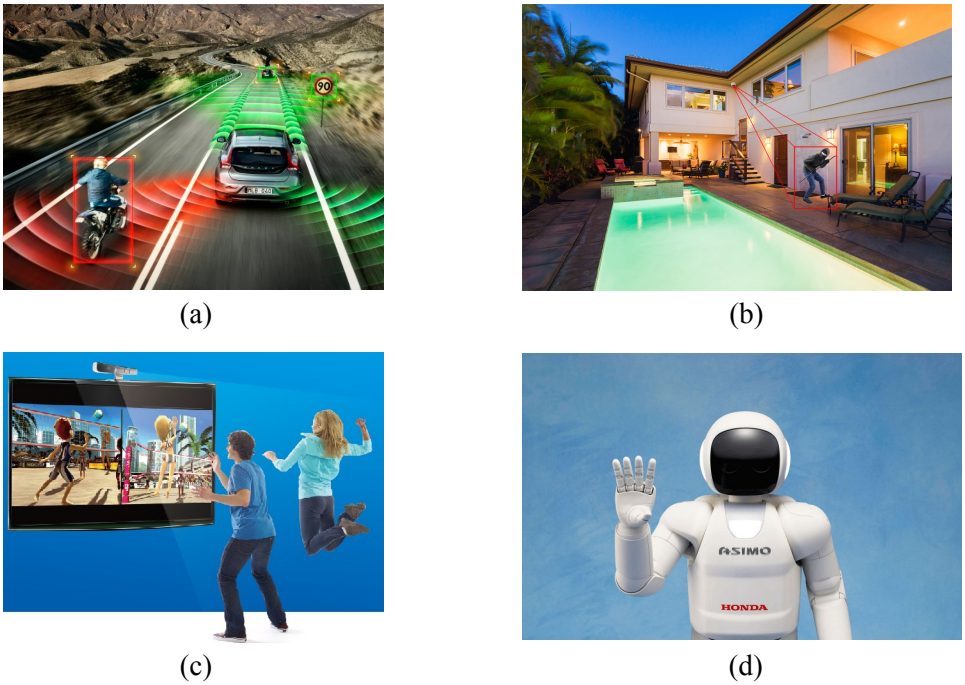
Background of Embedded Vision Techniques

2.1 Embedded Vision Applications

Computer vision is a discipline that studies the acquisition, processing, analysis, and understanding of real-world visual information [16]. It began in the late 1960s but has made recent rapid progress owing to the advances of technology in both computing and computer vision theory [17].

Embedded systems are computer systems with dedicated functions within mechanical or electrical systems [18]. They range from portable devices like mobile phones and tablets to large stationary installations such as factory controllers and video surveillance, and are often constrained by real-time computing, power consumption, and cost.

Embedded vision is a technology that incorporates practical computer vision capabilities into embedded devices to create widely deployable applications through visual means. This field is gaining popularity in home, industry, agriculture, and medical applications etc. Figure 2.1 shows some embedded vision applications.



(a) (b) (c) (d)

Figure 2.1 Embedded vision applications: (a) Advanced Driver Assistance Systems (ADAS); (b) Video surveillance; (c) Game machine; (d) Robotics.

In the following, we give a brief introduction of some specific embedded vision applications, including object detection, Advanced Driver Assistance Systems (ADAS), video surveillance, and gesture recognition.

Object detection: Object detection is a common and essential process in almost all the computer vision systems. It requires analyze a scene and recognize all of the constituent objects [16]. Furthermore, the outputs of object detection might serve as inputs of object tracking or object recognition etc. Popular applications include face detection, pedestrian detection, vehicle detection, and traffic sign detection etc. In some vision applications, some sort of object detection steps are required in the processing flow. For example, corner detection is usually applied in a movement tracking application to identify some key points in an image and then looks for them in subsequent frames.

ADAS: Advanced Driver-Assistance Systems (ADAS) are aimed to enhance the vehicle systems to improve safety and offer better driving [19, 20]. To avoid

collisions and accidents, the system provide techniques that alert the driver to potential problems. ADAS relies on multiple data sources and techniques, such as pedestrian, lane, vehicle, traffic sign, cyclist, and animal detection in the path of the vehicle.

Video surveillance: Video surveillance is the monitoring of behavior and activities through the visual mean for the purpose of protecting people. With the growing numbers of surveillance cameras and IP cameras, video surveillance systems are becoming more and more popular. Sending large amounts of visual information collected by video surveillance systems to the cloud would require high network bandwidth. Extracting the meaningful visual information and only transmitting the interesting cases, e.g., frames with detected human, for further analysis is an efficient way to ease the network traffic.

Gesture recognition: Gesture recognition enables humans using simple gestures to communicate or interact with the devices without physically touching them. Currently, gesture recognition focus in the hand gesture recognition and emotion recognition. It is gaining popularity in the field of consumer electronics and industrial. For consumer electronics, such as gaming and virtual reality, gesture recognition provide a direct and natural way to interact with the machine. For industrial applications, gesture recognition is a better choice to control machines while touching them might be dangerous or impractical.

2.2 Machine Learning for Object Detection

2.2.1 Vision Pipeline

Although vision applications are many and varied, the vision system follows more or less the same flow to analyze and process the visual data. This flow is called vision pipeline as shown in Figure 2.2. The vision pipeline consist of four major steps: image pre-processing, Region-of-Interest (ROI) extraction, extracted ROI processing, and decision making. In the image pre-processing

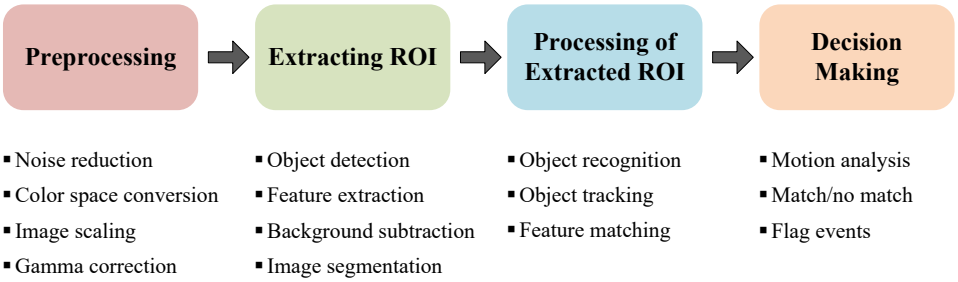


Figure 2.2 Vision pipeline.

step, algorithms are with regular computations and simple data-level parallelism. In the following steps, the computation becomes irregular with complex data structures and data-level parallelism. At the end of the pipeline, the processing becomes standard and common.

2.2.2 Object Detection

Object detection is at the heart of nearly all the computer vision systems. Figure 2.3 shows the general flow for object detection. The flow is based on machine learning techniques aiming at specific object recognition tasks. It typically consists of low-complexity pre-processing, versatile feature extraction, and classification with trained classifiers. In the image pre-processing stage, it usually involves color space conversion, image scaling, illumination adjustment and histogram equalization for successive input frames.

Feature extraction is the process of extracting key characteristics of an image. Typical image features include edges, lines, corners, shapes, and intensity gradients etc. A high quality feature should be invariant to viewpoint, scale, orientation, illumination, and noise. Feature extraction quality usually related with the algorithm's computational demand. In the next section, we give an introduction to various features for object detection.

As for classification, several popular classifiers are developed, such as boosting [21], Support Vector Machine (SVM) [22], and K-nearest neighbor [23], which are based on machine learning techniques. These techniques use previous data to predict information of the new data. The classifier is divided into supervised

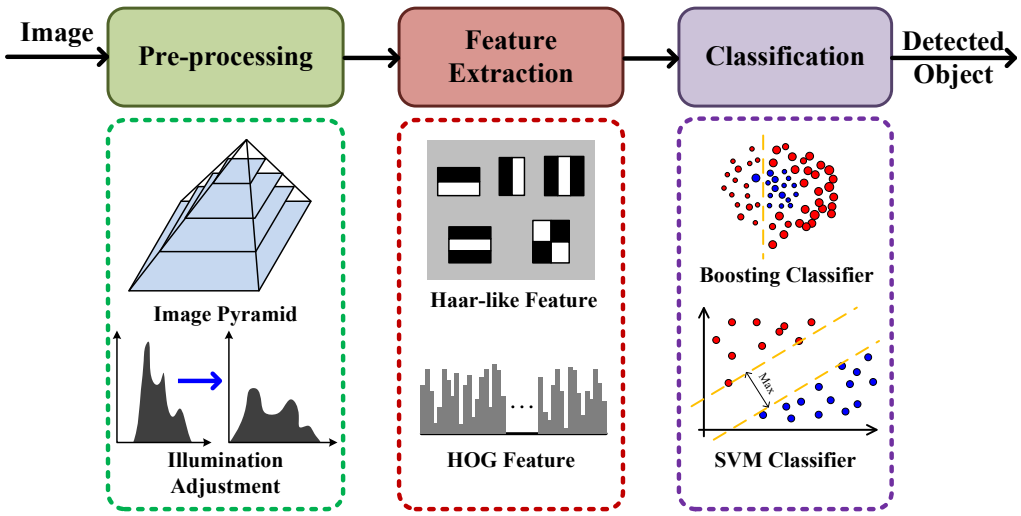


Figure 2.3 General flow for object detection.

or unsupervised category. AdaBoost is a supervised learning algorithm based on decision trees. The AdaBoost classifier combines a set of weak learner decision trees to improve the accuracy of classification [21]. SVM is also a supervised learning algorithm used to classify feature descriptors. The SVM classifier creates the maximum distance (margin) from two classes to achieve superior classification performance [22]. Details about the AdaBoost classifier and SVM classifier will be introduced in Section 4.1 and Section 5.1, respectively.

2.3 Overview of Feature Extraction Algorithm

In this section, we introduce various well-known feature extractions from low quality but efficient Harris Corner detector to high quality and computationally intensive Convolutional Neural Networks (CNN). Figure 2.4 shows the evolution of feature extraction algorithms. Note that details of Haar-like feature and Histogram of Oriented Gradients (HOG) feature are not include in this section, but introduced in Section 4.1 and Section 5.1, respectively.

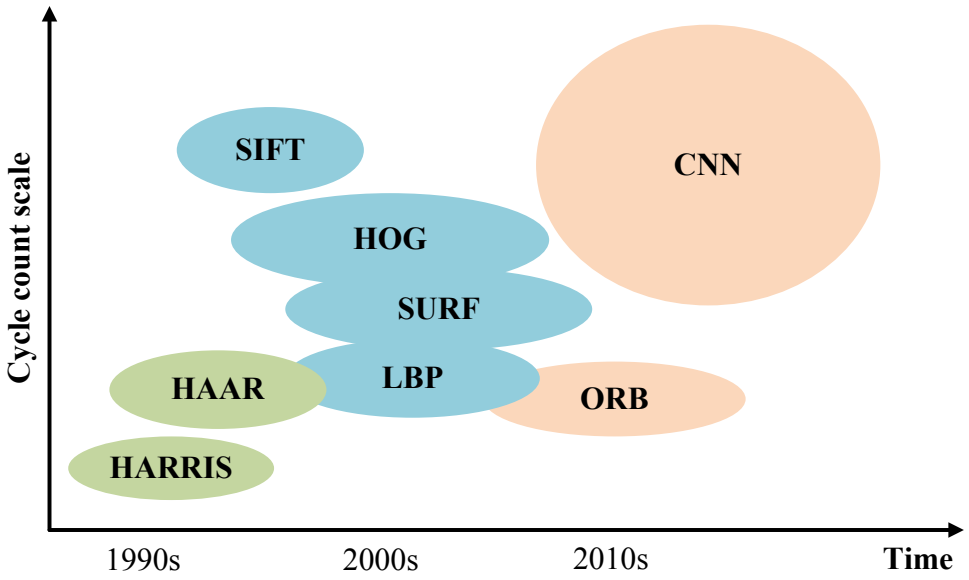


Figure 2.4 Feature extraction algorithm.

2.3.1 Harris Corner

The history of image features originates from the corner detection. Corners in an image are regions with large variations. The basic idea of corner detection is to find the variations by looking at intensity values within a small window. Harris corner [24] is one of the widely used corner detector. It finds the difference in intensity in all directions using a rectangle window or Gaussian window and considers the small shifts by applying Taylor Expansion. This allow the Harris corner detector being robust to rotations. However, Harris corner detector is not scale-invariant. Corner detection is frequently used in motion detection, video tracking, image mosaicing, and object recognition.

2.3.2 Scale-Invariant Feature Transform (SIFT)

Scale-invariant Feature Transform (SIFT) [25] is used to detect and describe local features in an image. SIFT descriptor is invariant to scaling, orientation, illumination variation, and affine distortion. It is widely used in applications including object recognition, tracking, image stitching, and robotic navigation.

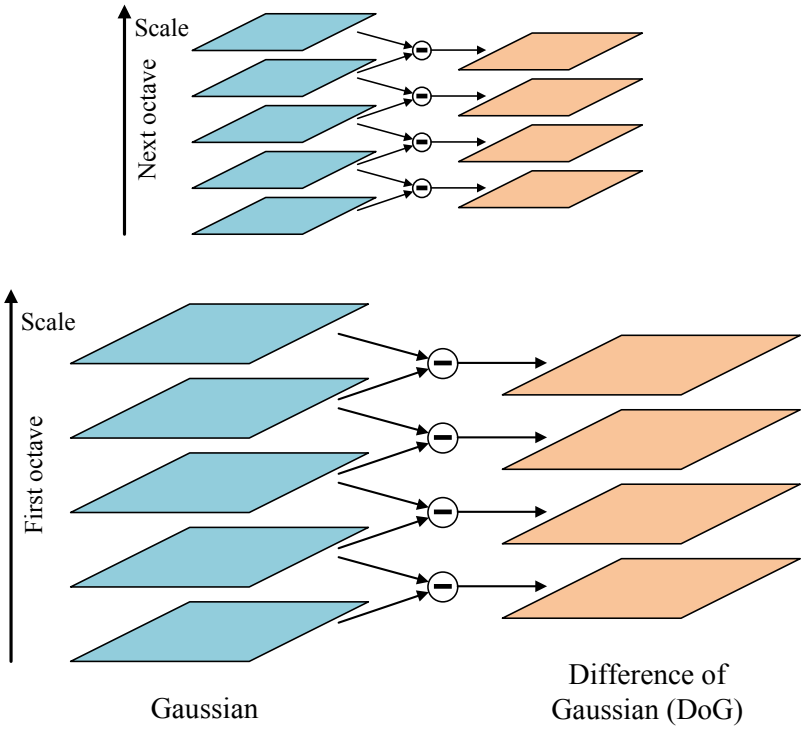


Figure 2.5 DoG pyramid generation in SIFT.

SIFT descriptor is generated through two steps, feature points detection and description. The detection stage is to locate the feature points, while the description stage is to characterizes the feature. In the first stage, SIFT generates an image pyramid using iterative Gaussian blurring. Then, the Difference-of-Gaussian (DoG) images are taken from two adjacent Gaussian-blurred images per octave. Figure 2.5 shows the generation of DoG pyramid in SIFT. The candidate feature points are taken as maxima/minima of the DoG pyramid that occur at adjacent scales. In the second stage, the descriptor is formed by computing the gradient at each pixel in a small region (e.g., 16x16 window) around of detected feature points. To reduce the effects of contrast and illumination variations, the generated 128-D vector is normalized unit length as the final SIFT descriptor.

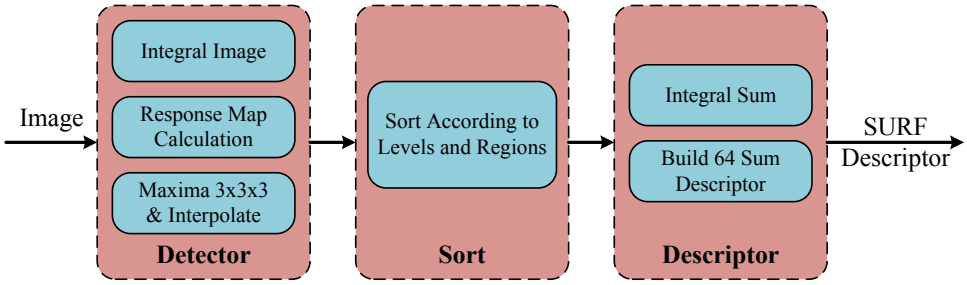


Figure 2.6 Speeded Up Robust Features (SURF) descriptor generation.

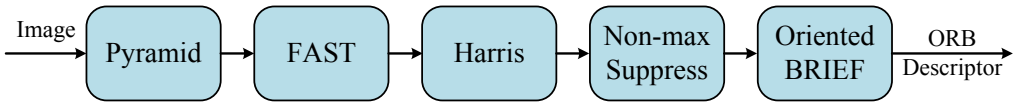


Figure 2.7 Oriented FAST and rotated BRIEF (ORB) descriptor generation.

2.3.3 Speeded Up Robust Features (SURF)

Speeded Up Robust Features (SURF) [26] are well-known feature extraction and partly inspired by Scale-Invariant Feature Transform (SIFT) descriptor. Similar to SIFT, SURF is also scale-invariant and rotation-invariant. It generates a smaller feature descriptor than SIFT with less computation. SURF descriptors have been utilized to locate and recognize objects, such as faces or human, to track objects and to extract points of interest.

SURF uses a Hessian-based measure and operates in a scale space to find the feature point locations. Figure 2.6 indicates the SURF descriptor generation. To speed up the feature extraction, integral images are introduced for image convolutions. For different scales, the algorithm apply multiple sizes of box filters (e.g., sizes with 9x9, 15x15, 21x21, 27x27) to find the interest points. Then, a non-maximum suppression is applied to filter the strongest signal in a 3×3×3 neighborhood area. Finally, the feature descriptor is generated basing on the sum of Haar wavelets response, which is gradient-like computations, for an oriented region.

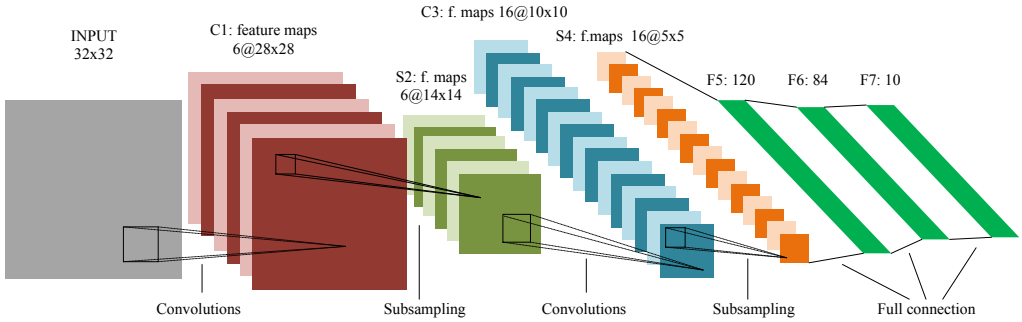


Figure 2.8 A representative CNN architecture – LeNet5 [4].

2.3.4 Oriented FAST and Rotated BRIEF (ORB)

Oriented FAST and rotated BRIEF (ORB) detector [27] is a fast robust local feature detector, which builds on the FAST keypoint detector and BRIEF (Binary Robust Independent Elementary Features) descriptor. It is developed for a fast and efficient alternative to SIFT or SURF.

Figure 2.7 shows the generation of ORB descriptor. ORB modifies FAST detector as scale invariant to find keypoints at each scale of the image pyramid. Once the keypoints are detected, ORB apply Harris corner measure to find top N points among the detected keypoints based on a threshold. To improve the rotation invariance, ORB utilizes the intensity centroid to compute the orientation of image patch. Then, BRIEF descriptors are computed on the rotated patch. The output binary string is referred as an ORB descriptor.

2.3.5 Convolutional Neural Networks (CNN)

A Convolutional Neural Network (CNN) consists of multiple computation layers. These computation layers are typically divided into three categories: convolutional layers, pooling layers, and fully-connected layers. A representative CNN architecture is shown in Figure 2.8. A convolutional layer applies filers on input images or input feature maps from the previous layer to extract visual characteristics or essential information and generate a higher-level abstraction as the output feature maps. The convolutional layer performs high-dimensional convolutions with highly intensive computation. Each convolutional layer is

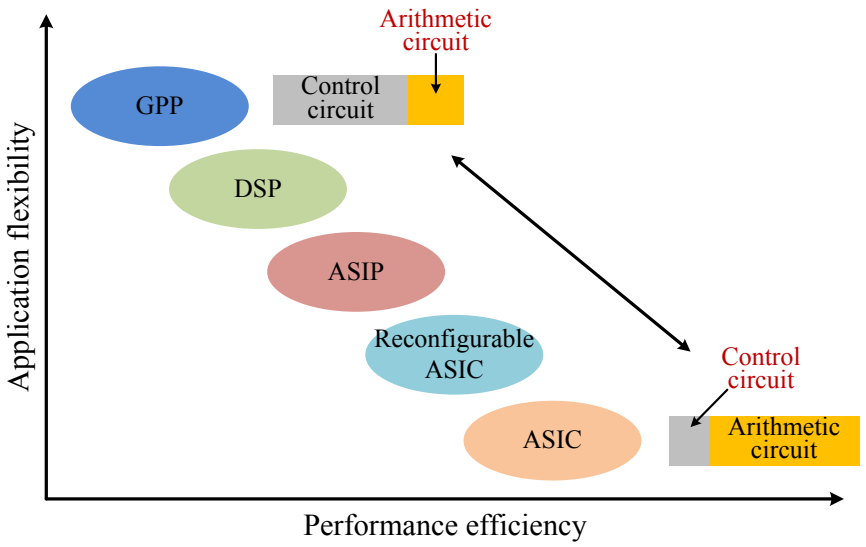


Figure 2.9 Hardware architecture with different levels of efficiency and flexibility.

usually followed by a pooling layer, which downsamples the output feature maps of the previous convolutional layer. Finally, fully-connected layers are introduced for classification purposes. To achieve superior performance, modern CNNs have the tendency to deploy a deep hierarchy of layers.

2.4 Hardware Architecture

The hardware architecture for vision applications falls into the category of General Purpose Processors (GPPs), Digital Signal Processors (DSPs), Application-Specific Instruction-set Processors (ASIPs), Application Specific Integrated Circuits (ASICs), and reconfigurable ASICs. The relationship of performance efficiency and application flexibility among GPPs, DSPs, ASIPs, ASICs and reconfigurable ASICs is shown in Figure 2.9.

The state-of-the-art GPPs are equipped with many advanced hardware techniques, e.g. multi issue ALU, out-of-order execution logic, and complicated branch prediction mechanism, to deliver higher performance. Intel Pentium 4 is a classical processor with many features of today's GPPs. Figure 2.10 shows the Intel Pentium 4 processor microarchitecture. Pentium 4 features hyper pipeline technology, which enables the software to run in a very deep instruction pipeline

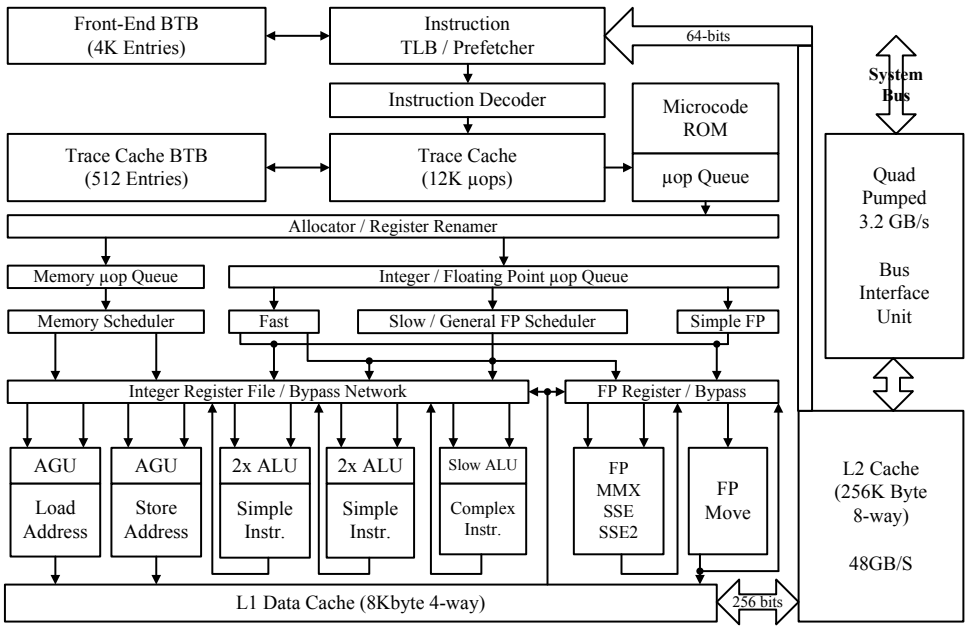


Figure 2.10 Intel Pentium 4 processor microarchitecture [5].

(20 stages) to achieve an industry-leading clock rate. Pentium 4 supports a new range of clock speeds. For higher performance, the frequently used ALU instructions can be executed at double the core clock. GPPs have the best application flexibility, but with the lowest performance efficiency against other types of hardware architecture.

Figure 2.11 depicts the architecture of TI TMS320C64x+ [5]. TMS320C64x+ is an 8-way VLIW DSP with a two-level memory architecture. It has independent instruction memory and data memory. The architecture contains two identical datapaths, each of which contains 32 32-bit general-purpose registers and four functional units (L: arithmetic unit; S: branch unit; M: multiply unit; D: load/store unit). The instruction fetch, instruction dispatch, and instruction decode units can deliver up to eight 32-bit instructions to the functional units every clock cycle. The execution of the instruction depends on the predication condition. The instruction set architecture contains instructions to accelerate the DSP applications and video applications.

Application-Specific Instruction-set processors (ASIPs) are processors in which the instruction set and hardware logic that customized toward certain

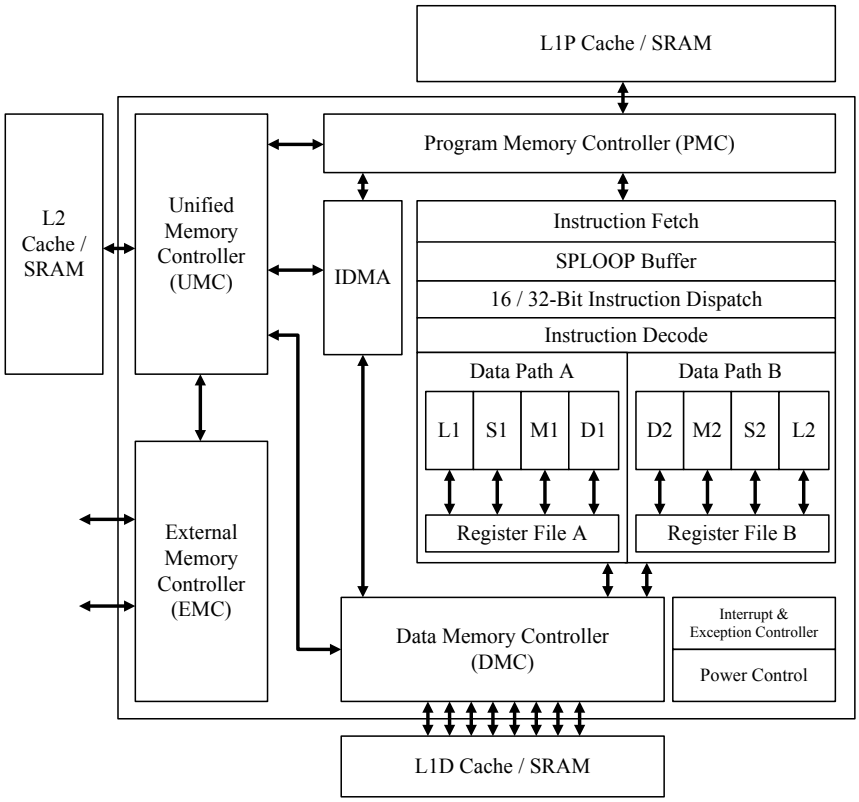


Figure 2.11 TI TMS320C64x+ block diagram [6].

applications. They combine the performance efficiency of a dedicated hardware and the flexibility of a software programmable processor. As shown in Figure 2.9, ASIPs can offer better performance and lower power consumption than GPPs/DSPs and provide higher flexibility than ASICs. Figure 2.12 gives an example of vision ASIP. The ASIP is designed for connected components labeling in embedded vision [7]. In [7], the parallel image slicing algorithms are analyzed. Then, a highly specialized processor architecture is proposed to execute thresholding, labeling and feature extraction. In the ASIP, a customized memory, special functional units, and the related special instructions are added to speed up the connected components labeling.

Application-Specific Integrated Circuit (ASIC) is an integrated circuit customized for a specific application. The functional units can execute in parallel or in a deep pipeline fashion and utilize special registers for internal communication. Therefore, the ASIC can offer the best datapath efficiency. However, the fixed

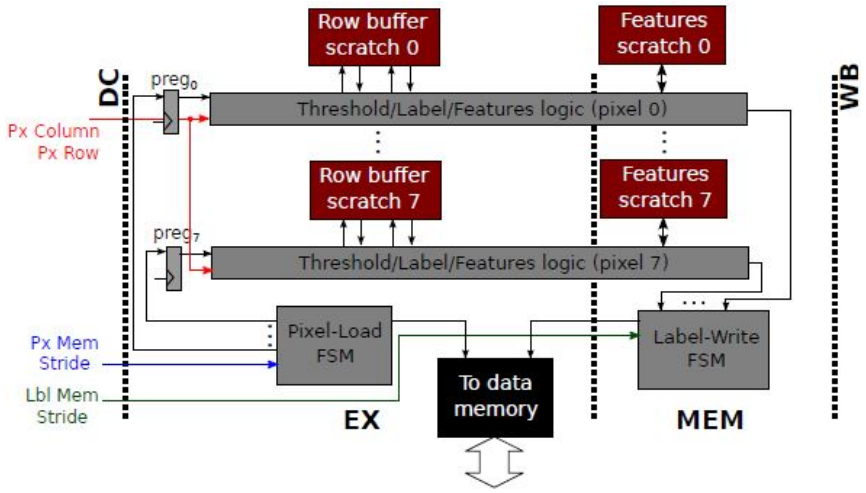


Figure 2.12 ASIP architecture for connected labeling in embedded vision [7].

control of the functional units make the ASIC with the worse flexibility. Even though some reconfigurable ASICs integrate more hardware resource, such as multiplexers, arithmetic logics and control registers, to lift the flexibility of ASIC approach. But the design cost is too high with a very limited flexibility.

2.5 Summary

In this chapter, we introduce the background knowledge of embedded vision techniques for two main parts. The first part shows the object detection based on machine learning fundamentals, including feature extraction and classification. Several features such as Harris corner, SIFT, SURF, ORB, and CNN are introduced. In the second part, we introduce the hardware architecture for vision applications. The pros and cons of GPPs, DSPs, ASIC, and ASIPs are discussed.

Chapter 3

ASIP Design Methodology

3.1 Design Challenges

With the rapid advance of technology, microprocessors are gaining more and more powerful computing capability. However, design and implement of a computer vision processor remain a very challenging task. The main challenges of vision processor design include:

- **High performance**: Vision applications are typically sophisticated and quite diverse with parallel and specialized input, such as image sensors and 2D/3D cameras. They have extremely high computation requirements. For example, to construct an image pyramid for a VGA image (640x480 pixels) in the image pre-processing step, it requires 10-15 million instructions. When the real-time constraints are taken into consideration, the computation is up to 300-450 millions of instruction per second. Furthermore, the processing in the later of the vision pipeline would requires more computation for the advanced but complex vision tasks. A high performance computation engine with high memory bandwidth are necessary to meet the high computation demanding.

- **Low power**: To run these computationally intensive algorithms on mobile devices, it demands low power as a critical requirement. A mobile device is

typically powered by batteries. Higher power consumption would require a large capacity battery. It comes with higher heat dissipation, leading to complex cooling systems with expensive packaging.

- **Flexibility**: Embedded vision applications are quite diverse and constantly evolving. It requires a certain degree of programmability to ease the maintenance, debugging, and refinement of the applications. The programmability makes contribution on improving the productivity and reduce the time-to-market pressure.

3.2 Application-Specific Instruction-set Processors (ASIPs)

The Application-Specific Instruction-set Processor (ASIP) is based on processor architecture. It combines a software programmable processor and application-specific hardware components. The design focus of an ASIP is aimed at specific performance and specific flexibility with low costs for solving problems in a specific domain.

Compared with General Purpose Processors (GPPs), Digital Signal Processors (DSPs) and Application-Specific Integrated Circuits (ASICs), Application-Specific Instruction-set Processors (ASIPs) emerge as promising solutions to offer the trade-offs between performance, power and flexibility requirements required by vision applications. ASIPs are processors in which the instruction set and hardware logic that customized toward certain applications. Similar to ASICs approach, functional units of ASIPs can execute in parallel and utilize special registers for internal communication. Therefore, ASIPs can provide data-path efficiency closeness to ASICs and better performance than GPPs/DSPs. Similar to GPPs/DSPs approach, the control of functional units is not fixed but taken over by an instruction decoder and a program. This enables them adaptable for different applications or variations of an algorithm. Thus, ASIPs can offer higher flexibility than ASICs. Also, owing to the highly utilized and dedicated

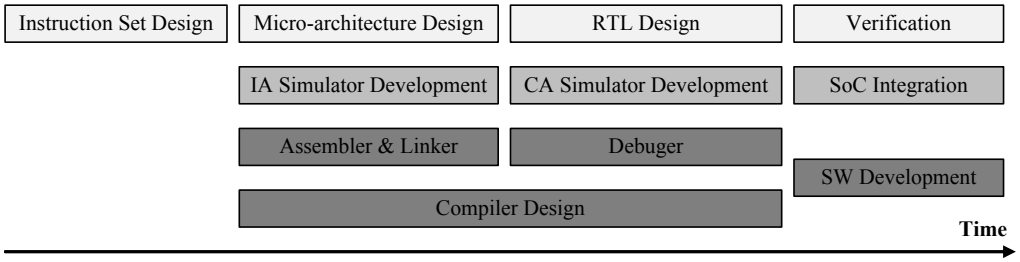


Figure 3.1 Traditional processor design flow.

data-paths with relatively low control overhead, ASIP becomes one of the most energy efficient solutions for programmable vision applications. Furthermore, from an application developer's perspective, the programmability offers shorter time-to-market with a lower risk.

3.3 Traditional Processor Design Flow

The design of a processor is a very demanding and complex task, which requires the design of the instruction set, micro-architecture, RTL, and a set of development tools, such as a simulator, an assembler, a linker, and a compiler. It is expensive to develop and maintain these development tools even for the specialist. Figure 3.1 shows the traditional processor design flow. The design is processed in sequence. The points of this design flow are summarized as below.

- RTL-based architecture design and optimization are time-consuming and expensive.
- Manually design of simulators is complex, tedious, and error-prone.
- The features of compilers are not clear in the architecture definition phase.
- The discontinuities exist between tools and models.
- Verification, software development, and SoC integration happen at the end of the design phase. It is too late to expose the performance bottlenecks in this stage.

This high-cost and inefficient design flow is unacceptable today, with the ever-growing complexity of SoC design and the time-to-market pressure.

3.4 ASIP Design Methodology

In order to facilitate the implementation of the ASIP-based processor for embedded vision applications, an efficient design methodology is necessary. Currently, there are two general ways for ASIP implementation. One is template processor based design with Hardware Description Language (HDL); Another is the custom architecture with Architecture Description Language (ADL). In the former approach, the design of an ASIP is through combining a template processor with some specialized modules. These modules are included in the component library. MeP [28] is one of these examples. This approach contributes to lower the design complexity and shorter the turnaround cycle. However, the fixed architecture of the template processor provided by the library would result in a lower design flexibility. In the latter approach, the processor architecture is captured at a higher level abstraction, which would reduce the design efforts and make the verification simpler. LISA [29] and nML [30] are good examples of this approach. Even though the ADL provides a higher level abstraction, ASIP design starts from scratch is still less efficient.

In our view, combining the template processor-based approach and ADL-based approach can bring in a more efficient ASIP design flow. In this section, we introduce our ASIP design methodology and the proposed design framework. Our ASIP design flow combines Architecture Description Language (ADL) based design and template processor based design.

3.4.1 ASIP Design Framework

Figure 3.2 shows our ASIP design methodology. It contains two design chains: hardware (HW) design chain and software (SW) design chain. The High-Level Synthesis (HLS) tool is at the heart of the design methodology. The dotted box represents the basic step of ASIP design, and the dark rectangle indicates

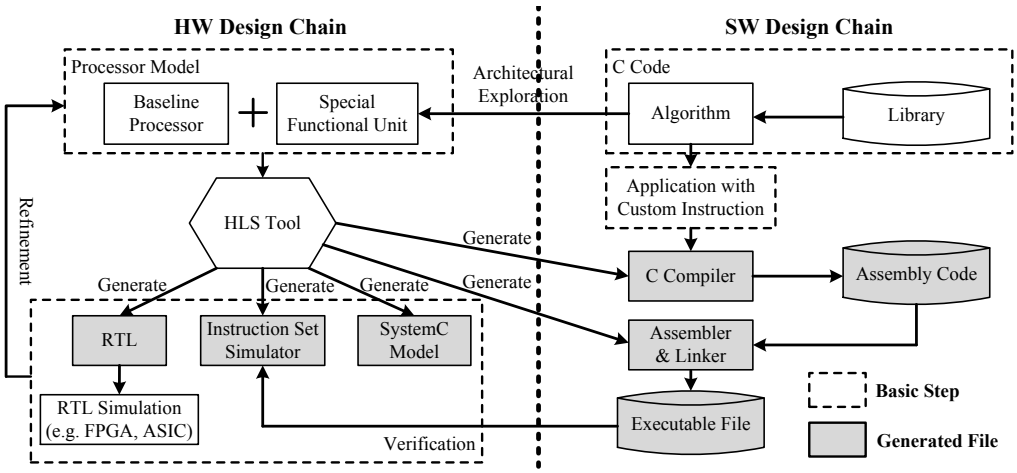


Figure 3.2 ASIP design methodology. It contains two design chains: hardware design chain and software design chain.

software development tool which automatically generated by HLS tools, e.g., Synopsys ASIP Designer [31].

Before the ASIP design starts, we capture the algorithms (see Section 4.2 and Section 5.2) in C using custom models and libraries such as OpenCV [32], which is an open source library for computer vision applications. Also, the algorithms are refined to use fixed-point data arithmetic.

After the hot spots are identified, the time critical codes are moved from the baseline processor (See Section 3.5.3) to a special functional unit (See Section 4.3.2 and Section 5.3.2). In this stage, we use the nML processor description language [30] to model ASIP architecture (See Section 4.3.1 and Section 5.3.1), which can capture a wide spectrum of processor architecture. The hardware customizations typically include adding pipeline stages, special registers, or local memories, adding special functional units and associated acceleration instructions (See Section 4.3 and Section 5.3). Based on these architecture description, the HLS tools can automatically generate a software development kit. It consists of a C compiler, assembler, linker, instruction set simulator, and SystemC model. With these tools, the new special instructions (See Section 4.3.3 and Section 5.3.3) are called from the software, and the modified algorithm is executed and tested effectively. Also, the ASIP model is translated automatically

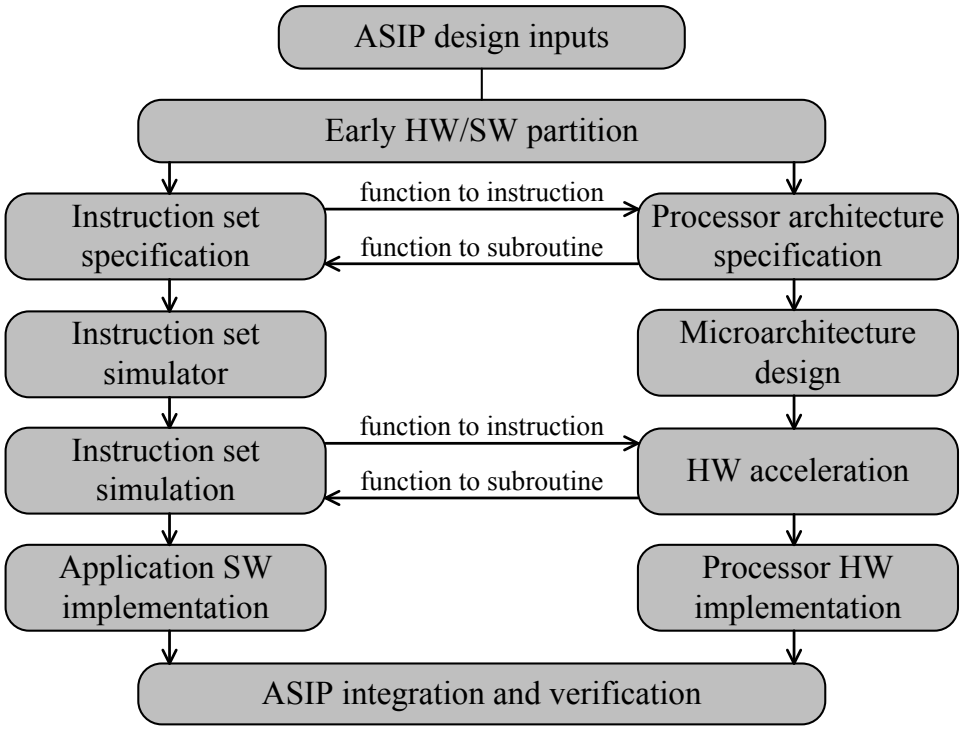


Figure 3.3 Hardware software co-design.

into Register-Transfer Level (RTL) code. This allows for fast feedback on the clock frequency, gate count, and power consumption. With these feedback information, we can easily refine and optimize the ASIP architecture.

3.4.2 Hardware/Software Co-design

The ASIP is a device that combines hardware and software. The hardware and software of an ASIP are tightly related to accommodating the applications. Therefore, HW/SW co-design methodology is necessary for designing an ASIP. In an ASIP design, hardware implementation means the design of new instructions performing a specific function; software implementation means that a specific function is designed as a subroutine using the available instructions. HW/SW co-design moves functions or tasks between software and hardware to satisfy the design requirements.

Figure 3.3 shows our ASIP HW/SW co-design flow. The HW/SW co-design starts from design space exploration down to the ASIP integration and

verification. At the very beginning of an ASIP design, applications are carefully analyzed. On the basis of the analysis, an instruction set architecture and the allocations of functions to hardware or software can be proposed. The most used functions will be assigned to hardware; while the seldom used functions are implemented as software subroutines.

3.4.3 Software Interface

To further facilitate the ASIP design for embedded vision, the proposed ASIP design framework provides a software interface with an open source computer vision library (OpenCV) [32], which includes many low-level vision and machine learning algorithms for a wide range computer vision application development. In our ASIP design framework, the software shares the same primitive data type (e.g., points, size, rectangles and scalar tuples etc.), data structure, and matrix structure with the OpenCV framework. This allows us quickly to add and optimize functions to follow the evolution of the vision algorithms.

3.4.4 ASIP Programming Interface

To further explain the ASIP design flow, the detail of our ASIP programming interface is given in Figure 3.4. Figure 3.4 (a) shows the instruction format of our ASIP. The instruction length is fixed to be 32-bit. It contains six programmable fields: (i) opcode fields (*opcode_1*, *opcode_2*), which are required by instruction decoder for instruction types identification; (ii) register fields (*reg_dst*, *reg_src*1, *reg_src*2), which indicate three optional registers (two source registers and one destination register) for operands of the instruction; (iii) immediate value field (*imm*7), which is a 7-bit immediate number or reserved for identifying types of special instructions. Figure 3.4 (b)-(e) demonstrate an example to customize a special instruction, diffv, which performs image gradient computation. Figure 3.4 (b) describes the C code of image gradient computation. The architecture description language (nML) for the special instruction (diffv)

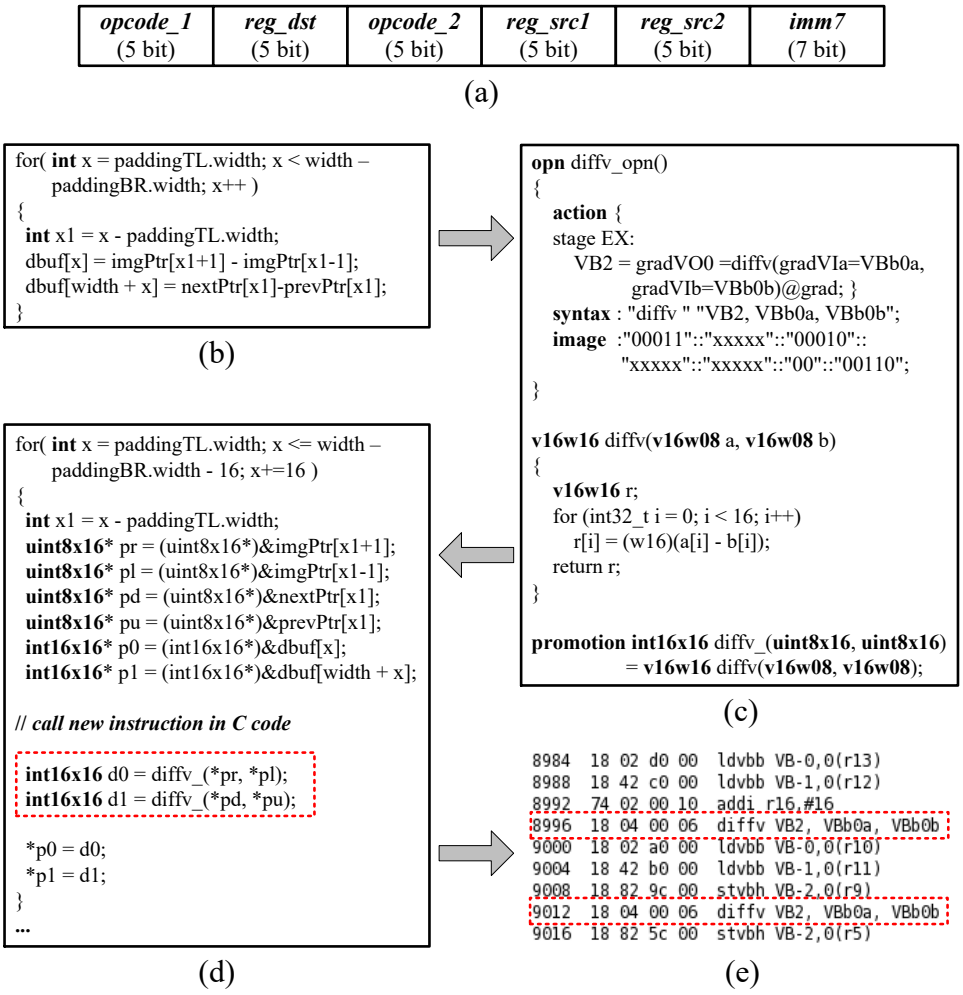


Figure 3.4 An example of special instruction customization: (a) Instruction format. (b) C code of image gradient computation. (c) Architecture description language for the new special instruction. (d) Call new special instruction in C code. (e) Generated machine code.

is given in Figure 3.4 (c). In Figure 3.4 (d), the new custom instruction is called in the application C code. Figure 3.4 (e) shows the generated machine code.

3.5 Design Considerations

To demonstrate the efficiency of our ASIP design framework, we carried out two embedded vision processor designs using the proposed framework. The algorithms are the Haar-like feature with AdaBoost classifier and HOG feature

with SVM classifier. These algorithms are two representative object detection algorithms. The former one is efficient for face detection, while the latter one is popularly used for pedestrian detection. Face detection and pedestrian detection are the active research fields in object detection study. These two algorithms are with different computation and memory access patterns. Even though the difference exists, we can find different levels of design reuse in the two ASIPs design. The efficient and practical of our ASIP design methodology can be revealed through the two ASIPs design. This is our original intention to provide two design references. In this section, we will introduce our thinkings on vision processors design.

3.5.1 Algorithm Consideration

In Section 2.3, we give an overview of feature extraction algorithms. There are two kinds of features: hand-crafted features and learned features. The hand-crafted features were designed by experts in the field by a hand-crafted flow; while the learned-features were directly learned from the training data and the whole system is trained through an end-to-end approach. There are many popular hand-crafted features, such as Histogram of Oriented Gradients (HOG) and Scale-Invariant Feature Transform (SIFT). These features are image gradient-based features and motivated by the observation that pedestrians are sensitive to edges (i.e., gradients) in an image. For the learned-features, they are used in a form of machine learning known as deep learning. Convolutional Neural Network (CNN) is a well-known learned-feature used in image and vision processing field.

Compared with learned-features, hand-crafted features show a higher energy efficiency with a reduced performance accuracy. One of the reasons is hand-crafted features required less amount of computation and data movement. Compared with hand-crafted features, learned-features offer better accuracy on a variety of tasks. They map the input into a higher dimensional space. The high accuracy comes at the cost of high computational complexity. No-free-lunch

theorem from machine learning field indicates that any learning techniques cannot perform universally better than another learning technique [33]. For example, Neural networks prone to over-fitting and perform worse than a linear classifier (e.g., SVM) in the classification of linearly-separable data. Therefore, we should not search for a generous algorithm, but a suitable algorithm under the constraints of a specific application.

In this research, we focus on the hand-crafted features. We applied two representative algorithms, the Haar-like feature with AdaBoost classifier and HOG feature with SVM classifier, to our ASIP design framework. The former one is on behalf of the decision trees based algorithms. The latter one is on behalf of the image gradient based algorithms; it robust to variations in illumination and noise. Both algorithms can be applied to multiple object detection, such as the face, pedestrian, vehicle, and traffic sign detection [34–38] etc. But the Haar-like feature is efficient for face detection, while HOG feature is efficient for pedestrian detection.

The common features of the above algorithms can be used to construct other state-of-the-art algorithms. For example, the integral image calculation within AdaBoost algorithm is also the fundamental of Speeded Up Robust Features (SURF). The image gradient computation is the necessary step to generate the SIFT feature. Further, the HOG descriptor is the basis of Deformable Parts Models (DPM) algorithm, which is the best hand-crafted feature based object detection technique so far.

3.5.2 Understanding the Application

Understanding the application means design space exploration at the system level. The first step in our ASIP design flow is the application design space exploration. The data structure, computation pattern, memory access pattern, and the opportunities of parallelization are exploited and analyzed before the ASIP design. In Section 4.2 and Section 5.2, we perform the Design Space Exploration (DSE) to understand the vision applications. Based on the profiling information,

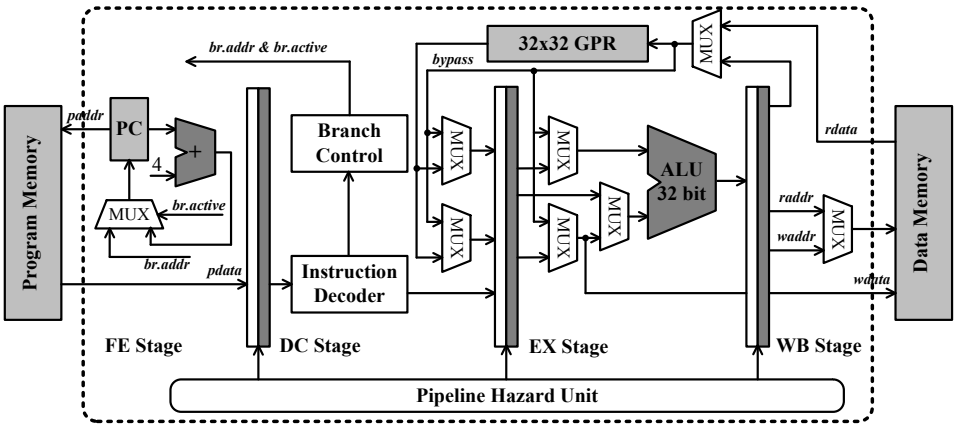


Figure 3.5 Architecture of baseline processor. A 32-bit RISC processor with four stage pipelines: fetch, decode, execute and write back stage.

the critical path in the algorithm, the function coverage, the performance requirements, and the early hardware/software partition are available.

3.5.3 Architecture Consideration

The microarchitecture of the vision processors is based on in-house TCT processor [39], which is a 32-bit Harvard architecture RISC processor. Other starting point architectures, such as a VLIW, were considered. For VLIW architecture, the compiler design cost and software design cost could be higher. Without a very good compiler knowledge, it would be troublesome to select the VLIW architecture. By contrast, the RISC features the simple and short instructions, which can reduce the power and area of the instruction decoder. Also, this feature can significantly reduce design and verification complexity.

The TCT processor architecture is shown in Figure 3.5. TCT processor contains four stage pipelines, local instruction and data memories, 32 general purpose registers, integer multiplier and multi-cycle integer divider. The four pipeline stages are instruction fetch (FE) stage, instruction decode (DC) stage, execute and memory access (EX) stage and write back (WB) stage. Pipeline hazard unit is adopted to handle the data hazard problems. TCT processor has been shown to have comparable performance to that of ARM9 processor [40].

In the proposed ASIP (See Section 4.3.1 and Section 5.3.1), TCT processor handles task controlling and data transmission between special function unit and memory, while special function unit dealing with intensive computation tasks. The architectures of the proposed ASIP are introduced in Section 4.3 and Section 5.3.

We observe that the data are organized and operated in a symmetric manner in most feature extraction techniques. For these operations, the data-level parallelism can be more efficient than the instruction-level parallelism of the traditional scalar instructions. The focus of the vision processor design would be data-level parallelism. In the two reference ASIP designs, we introduce Single-Instruction Multiple-Data (SIMD) architecture to exploit the data-level parallelism. As shown in Section 4.4 and Section 5.4, the SIMD architecture can result in 21x speed-up for Haar-like feature value calculation and 101x speed-up for cell histogram generation.

Low power design is to minimize the dynamic power and the static power. Eliminating redundant operations is to minimize the dynamic power; while reducing the circuit size is to minimize the static power. In a modern processor, the data movement consumes more power than computation [41]. In order to achieve power efficient vision processing, we need to exploit dataflows to support parallel processing with minimal data movement or design efficient memory architecture to minimize the memory transaction. A mix data storage mechanism (See Section 4.2.2), cell-based histogram generation (See Section 5.2.1), and on-the-fly SVM classification (See Section 5.2.2) are introduced to reduce the memory transaction and enable low power consumption.

3.5.4 Design Reuse Consideration

Reusability is a key principle need to considered in hardware design. Maximize the design reuse can lower the design cost and make the design more practical. We show the different levels of design reusabilities in the two ASIP designs (See Section 4.3 and Section 5.3). the first level is the reuse of baseline processor

architecture. The baseline processor, TCT processor, serves as a template processor in our design methodology. In Section 4.3.1, the ASIP shares the same instruction and data memory with TCT processor and the pipeline is extended to accommodate the applications. In Section 5.3.1, the ASIP the same instruction, data memory, and pipeline with the baseline processor. The well-verified baseline processor provides a solid foundation for the ASIP design. The second level is the reuse of the CPU-enhancing instructions. Zero overhead loop controlling instructions (See Section 4.3.3 and Section 5.3.3), which minimizes the loop controlling overhead, are introduced in both ASIPs. The minimum/maximum value selection and conditional move instructions are also such kinds of instructions. These CPU-enhancing instructions are not limited in the two reference ASIPs and can be applied to all the ASIP-based processors. The third level is the reuse of customized HW blocks. Figure 2.3 illustrates the general flow for object detection. In the image pre-processing step, color space conversion, image pyramid, and illumination normalization are common features used in nearly all the object detection algorithms. The customized instructions (rgb2gray, hzip/vtip and sqrtv) for these functions can be reused for other object detection algorithms. The functional blocks introduced in the two reference ASIPs can also be applied to many of the hand-crafted feature based vision algorithms. For example, the integral image is an important and convenient approach to accelerate the feature computation in vision algorithms. It can be used to compute the Haar-like feature in AdaBoost-based object detection algorithm, and also can be used to compute the Speeded Up Robust Features (SURF) in the corresponding detection occasions. The gradient computation is a common operation for image gradient based features, such as HOG feature and SIFT feature.

3.6 Summary

In this chapter, we introduce the traditional processor design flow. Then, we demonstrate our ASIP design methodology and the proposed ASIP design

framework. The proposed framework is based on ADL and a baseline processor, it contains a hardware/software co-design chain. Finally, we give our consideration about the ASIP design, including the algorithm consideration, architecture consideration, and design reuse consideration.

Chapter 4

Object Detection Processor with Haar-like Feature and AdaBoost Classifier

4.1 AdaBoost-Based Learning Algorithm

The object detection method using AdaBoost as part of a learning algorithm was first proposed by Viola and Jones [42]. In order to select the most prominent features among a large number of features, AdaBoost makes use of a small number of weak classifiers, which are used for a single visual feature extraction and then boosted to construct a strong classifier. Moreover, the cascade of strong classifiers makes classification procedure even more efficient. The cascade structure is shown in Figure 4.1. This simple-to-complex cascade structure, which follows the principle of coarse-to-fine search [43], can quickly reject the non-object like samples and spend more computation time on more object promising region.

The most popular weak classifiers used with AdaBoost algorithm are Haar-like features. Examples of Haar-like features are shown in Figure 4.2. These features consist of several black and white rectangles. They can be viewed as filters that can detect the presence or absence of certain visual information,

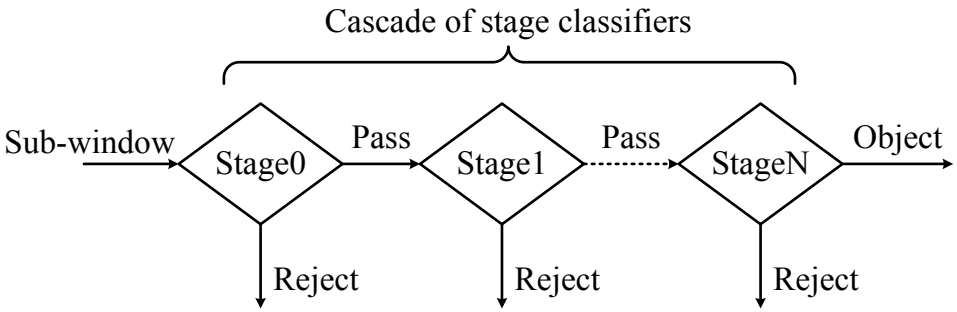


Figure 4.1 Cascade structure of AdaBoost algorithm. Each stage contains a small number of weak classifiers. The structure can quickly reject non-object like sub-windows in early stages.

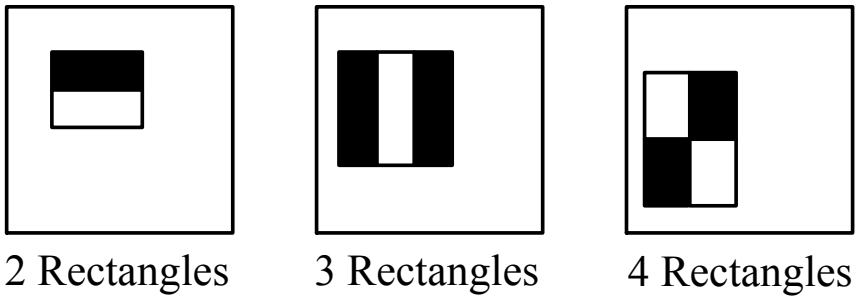


Figure 4.2 Examples of Haar-like features. These features are consisted of black and white rectangles.

e.g., an edge or a line feature, in an image. The computation of Haar-like features involves calculating the difference of brightness between black and white rectangle regions. If the difference exceeds a feature threshold, the weak classifier returns true. Otherwise, the weak classifier outputs false. This approach is used to determine if an edge or a line feature exists in the image or not.

To calculate sum of the pixel value (brightness) in black and white rectangles in a rapid manner, a concept of the integral image was introduced.

$$II(x,y) = \sum_{x' \leq x, y' \leq y} I(x^{'}, y^{'}) \tag{4.1}$$

where $I(x,y)$ and $II(x,y)$ are pixel values of the original image and integral image, respectively. Each pixel location (x,y) in the integral image holds the

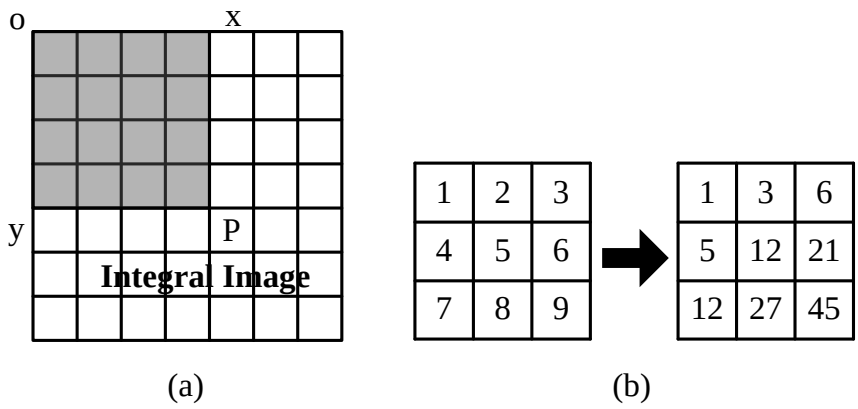


Figure 4.3 (a) Basic concept of integral image. Pixel P contains sum of all the shaded pixels. (b) An example of integral image generation. A 3x3 image and its integral image.

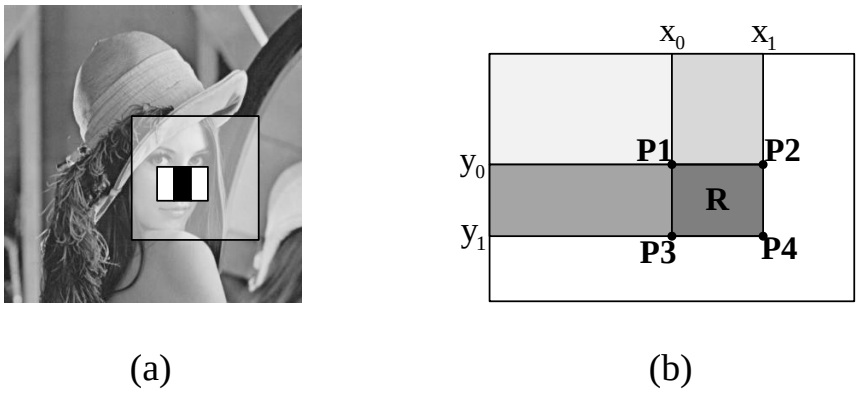


Figure 4.4 (a) Feature computation over image. Step from left to right, from top to down. (b) Rectangle sum calculation. Sum of rectangle R can be calculated using the corner integral image values of the rectangle: P4-P3-P2+P1.

sum of all the pixel values to the left and above that location in the original image [42]. Figure 4.3 illustrates the concept of the integral image.

$$
\begin{aligned}
R(x_0, y_0, x_1, y_1) &= \sum_{x_0 \le x' \le x_1, y_0 \le y' \le y_1} I(x', y') \\
&= \sum_{x' \le x_1, y' \le y_1} I(x', y') - \sum_{x' \le x_1, y' \le y_0} I(x', y') \\
&- \sum_{x' \le x_0, y' \le y_1} I(x', y') + \sum_{x' \le x_0, y' \le y_0} I(x', y') \\
&= II(x_1, y_1) - II(x_1, y_0) - II(x_0, y_1) + II(x_0, y_0)
\end{aligned}
\tag{4.2}
$$

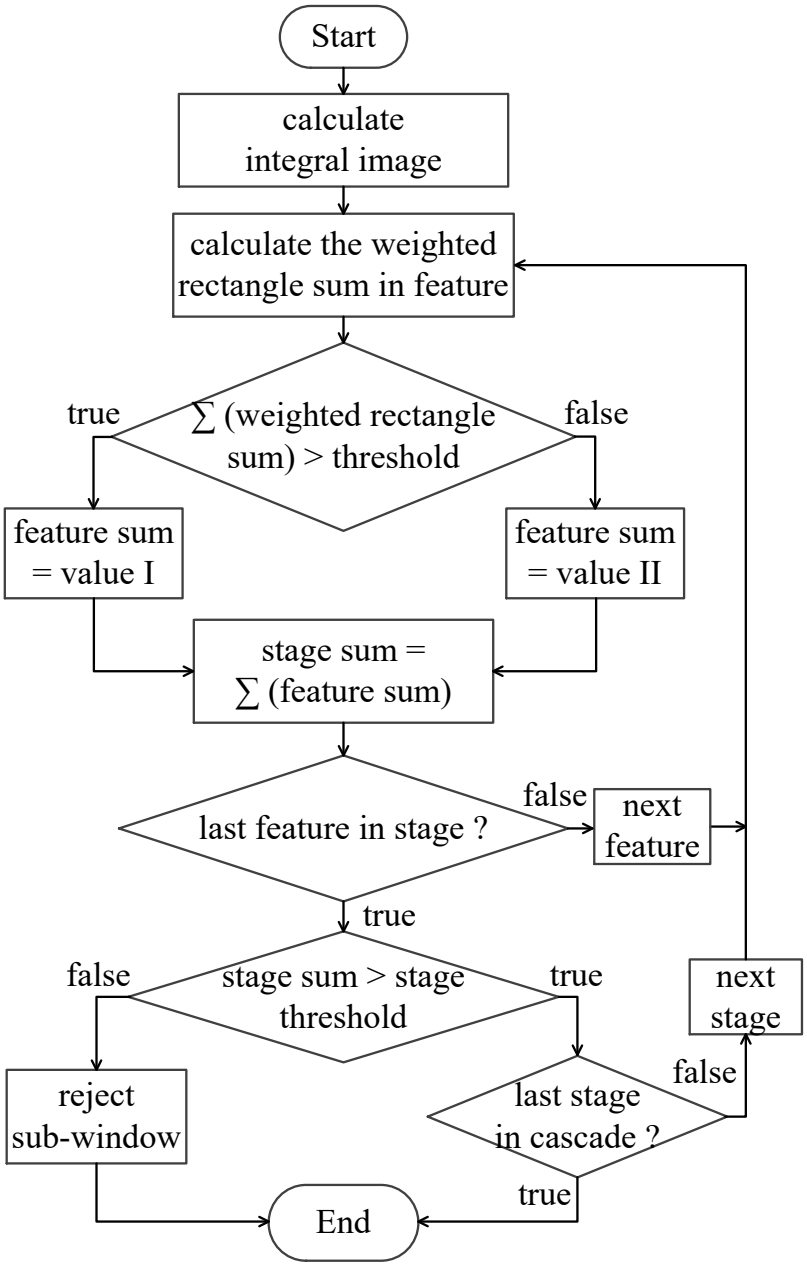


Figure 4.5 Classification procedure of AdaBoost Algorithm. Value I, value II, threshold and stage threshold were determined during the machine learning training phase.

As shown in (4.2), sum of pixel values in rectangle $R(x_0, y_0, x_1, y_1)$, the region between (x_0, y_0) and (x_1, y_1), can be computed through two additions and two subtractions using four corner integral image values of the rectangle. The basic idea of feature sum calculation is shown in Figure 4.4.

For AdaBoost algorithm, the detection is performed on a rectangle region of the original image, which called a sub-window. The detection algorithm scans the entire image with the sub-window to detect an object. The size of sub-window can vary with applications. For example, Viola and Jones used 24 x 24 pixels of sub-window for face detection. For objects in the image which are larger than the sub-window, Viola and Jones scale the sub-window to detect large-sized objects [42].

After calculation and accumulation of all the Haar-like features in a stage, a stage threshold is used to determine if the processing flow advances to the next stage or not. The classification procedure of AdaBoost algorithm is shown in Figure 4.5. The input of the algorithm is a detected image and a pre-trained cascade classifier data. The output is the found locations of each detected object and a scale value which represents how large or small the object was.

4.2 Algorithm Analysis

4.2.1 Hot Spots Identification

As a starting point, we use OpenCV [32] to develop the detection program for software profiling, to find the most critical computation in the AdaBoost algorithm. The classifier data originates in OpenCV, which includes 22 cascade stages with a total number of 2135 weak classifiers. These classifier data are pre-trained by fixed size frontal faces (20 x 20 pixels). Table 4.1 shows the number of Haar-like features in each stage. Figure 4.6 reports the profiling result.

To further find the bottleneck of AdaBoost algorithm, we make a deep analysis of Run Classifier function, which performs Haar-like feature values calculation. It takes almost 80% of the processing time. Algorithm 1 demonstrates the pseudo code of Run Classifier function. The main loop (lines 1-24) processes a sub-window at a time. The inner loop (lines 2-20) of Run Classifier function is the kernel computing Haar-like feature values. To deal with all the weak

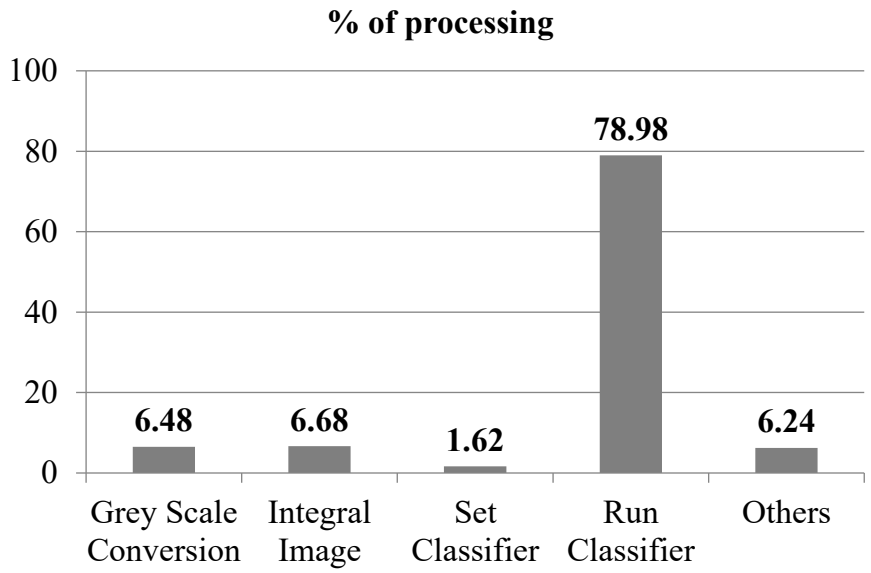


Figure 4.6 Distribution of workload among AdaBoost algorithm. The highest computational task, Run Classifier function, takes almost 80% of the total cycle counts.

classifiers within a sub-window and all the sub-windows within an input frame, the inner for loop repeats tens of thousands of times that makes it becomes the time critical codes. The bottleneck comes from two aspects. One is the limited memory bandwidth between the classifier module and memories, which store integral image and classifier data. Another is the repeated rectangle sum calculation operations.

For our object detection engine, we focus on implementing the most computationally intensive parts on ASIP: grey scale conversion, image scaling, integral image calculation and Haar-like feature value calculation. All other computations are performed in pure software.

4.2.2 Memory Bottleneck

4.2.2.1 Image Scaling

Generally, in sliding windows approach, there are two methods to detect large-sized objects: sub-window upscaling and input image downscaling. Figure 4.7 demonstrates the two image scaling methods. Viola and Jones suggest enlarging

Algorithm 1 Pseudo code of Run Classifier function

```
Input: Integral image and classifier data
for all stages do
    for all weak classifiers do
        Load classifier data for 1st rectangle;
        Load integral image for 1st rectangle;
        Compute 1st weighted rectangle sum;
        Load classifier data for 2nd rectangle;
        Load integral image for 2nd rectangle;
        Compute 2nd weighted rectangle sum;
        if has_3rd_rectangle then
            Load classifier data for 3rd rectangle;
            Load integral image for 3rd rectangle;
            Compute 3rd weighted rectangle sum;
        end if
        Accumulate the weighted rectangle sum;
        if sum < classifier threshold then
            Set stage sum to a pre-trained value;
        else
            Set stage sum to another pre-trained value;
        end if
    end for
    if stage sum < stage threshold then
        Return current stage number;
    end if
end for
```

the sub-window [42]. Since Haar-like features only contain black and white rectangles, sub-window upscaling linearly would not result in losing of data. However, for the sub-window upscaling method, data access to the integral image would become a bottleneck, due to the memory bandwidth. One potential solution is employing the input image downscaling method, keeping the sub-window in a fixed-size and storing the integral image values in register arrays. Note that such approach might affect the detection rate.

Thus, we evaluate the detection rate of image downscaling method and make a comparison with the sub-window upscaling method. For image scaling, we use bilinear interpolation algorithm [44], in which the pixel value in the new image is generated from four closest neighborhood pixel values of the computed

Table 4.1 Number of weak classifiers in each stage

Stage	**# of Weak Classifiers**	**Stage**	**# of Weak Classifiers**	**Stage**	**# of Weak Classifiers**
0	3	8	56	16	140
1	16	9	71	17	160
2	21	10	80	18	177
3	39	11	103	19	182
4	33	12	111	20	211
5	44	13	102	21	213
6	50	14	135	**Total**	2135
7	51	15	137		

pixel location. The detection is performed on MIT + CMU frontal faces image data set [45]. This data set contains 130 images with 507 labeled frontal faces. Both downscale and upscale factors are set to 1.2.

The experimental result is shown in Figure 4.8. The graph shows the Receiver Operating Characteristic (ROC) curves of image downscaling and sub-window upscaling methods. The ROC curve shows a relationship of detection rate against false positive rate at different threshold settings [46]. The detection rate is the number of accurately detected objects divide by the number of labeled objects. The false positive rate is the number of incorrectly detected sub-windows divide by the total number of sub-windows scanned.

In Figure 4.8, the solid line and the dotted line indicate ROC curves of the sub-window upscaling method and image downscaling method using bilinear interpolation, respectively. In the lower false positive rate region, image downscaling method shows better detection rate than sub-window upscaling method, while in the higher false positive rate region, sub-window upscaling method gains superior performance. However, the performance gap between them is small. The fact indicates that image downscaling method would not affect the detection performance too much and still maintain a considerable detection rate.

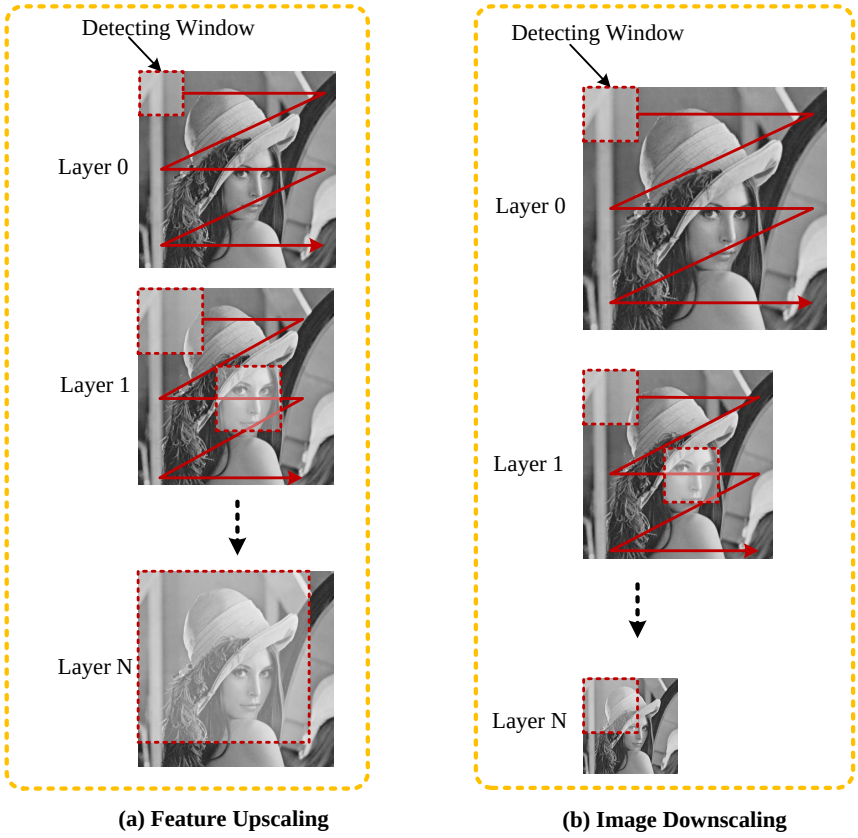


Figure 4.7 Sub-window upscaling method and image downscaling method.

4.2.2.2 Integral Image Calculation

When applying image downscaling method, an integral image is generated for each scanned sub-window. These operations are both memory and computation intensive, due to the overlapping of sub-window resulting in reloading of input pixels and redundant arithmetic operations. Consequently, an efficient computation approach is necessary.

Once the integral image values are stored in register arrays, a benefit of utilizing the integral image of current sub-window to calculate the integral image of next sub-window is available. This is based on the fact that the first $h-1$ rows of next sub-window integral image can be computed:

$$\hat{I}(x,y) = \tilde{I}(x+1,y),\ for\ 1 \leq x \leq h-1 \tag{4.3}$$

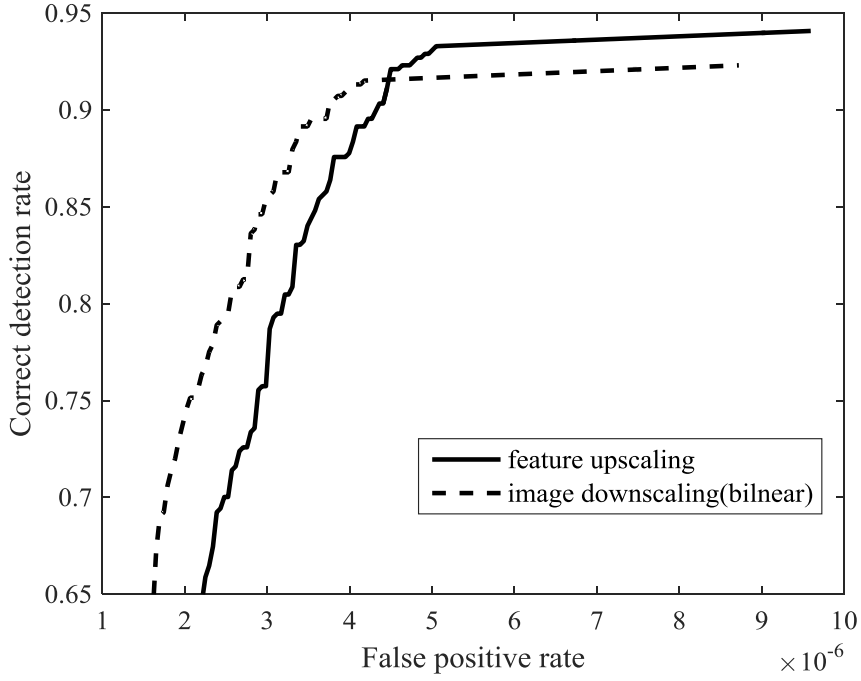


Figure 4.8 ROC curves of feature upscaling method and image downscaling method.

$$\begin{aligned} \hat{II}(x,y) &= \sum_{i=1,j=1}^{x,y} \hat{I}(i,j) = \sum_{x'=2,y'=1}^{x+1,y} \tilde{I}(x^{'},y^{'}) \\ &= \sum_{x'=1,y'=1}^{x+1,y} \tilde{I}(x^{'},y^{'}) - \sum_{y'=1}^{y} \tilde{I}(1,y^{'}) \\ &= \tilde{II}(x+1,y) - \tilde{II}(1,y) \end{aligned} \tag{4.4}$$

where $\tilde{I}(x,y)$ and $\tilde{II}(x,y)$ are pixel value and integral image value of current sub-window, respectively. $\hat{I}(x,y)$ and $\hat{II}(x,y)$ are the pixel value and integral image value of next sub-window, respectively. The last row of the next integral image can be computed:

$$S(y) = \sum_{j=1}^{y} \hat{I}(h,j) \tag{4.5}$$

$$\begin{aligned}\hat{II}(h,y) &= \sum_{i=1,j=1}^{h-1,y} \hat{I}(i,j) + \sum_{j=1}^{y} \hat{I}(h,j) \\ &= \tilde{II}(h,y) - \tilde{II}(1,y) + S(y)\end{aligned} \tag{4.6}$$

where $S(y)$ is the cumulative row sum of row h of next sub-window. (4.4) and (4.6) imply that the integral image of next sub-window can be calculated in following steps. First, subtracting first row from all the remaining rows in register array which stores integral image; Second, shifting one row up in the register array; Third, computing cumulative row sum of the new input row; Last, adding the cumulative row sum to the row next to last to form the last row of register array.

4.2.2.3 Execution Rate of Cascade Stages

To find out which part of classifier data is memory accessing intensive, we conduct an experiment on testing the execution rate of each cascade stage. The experimental conditions are the same as described in Section 4.2.2.1. The result is shown in Figure 4.9. The graph shows that execution rate of classifiers in earlier stages is much higher than latter stages. Note that the execution rate of the first stage is 1, which means the weak classifiers in the first stage are always been executed. The average number of weak classifiers applied to a sub-window is 47.8, 75.0% of them belongs to the first 5 stages. Keeping the most frequently used data in registers is an effective way to reduce the data transfer time. From Table 4.1, note that early stages contain fewer weak classifiers. The fact indicates that we can store first few stages in registers at an affordable hardware resource price. In this paper, classifier data of the first 5 stages is stored in register files instead of memory for high-speed processing.

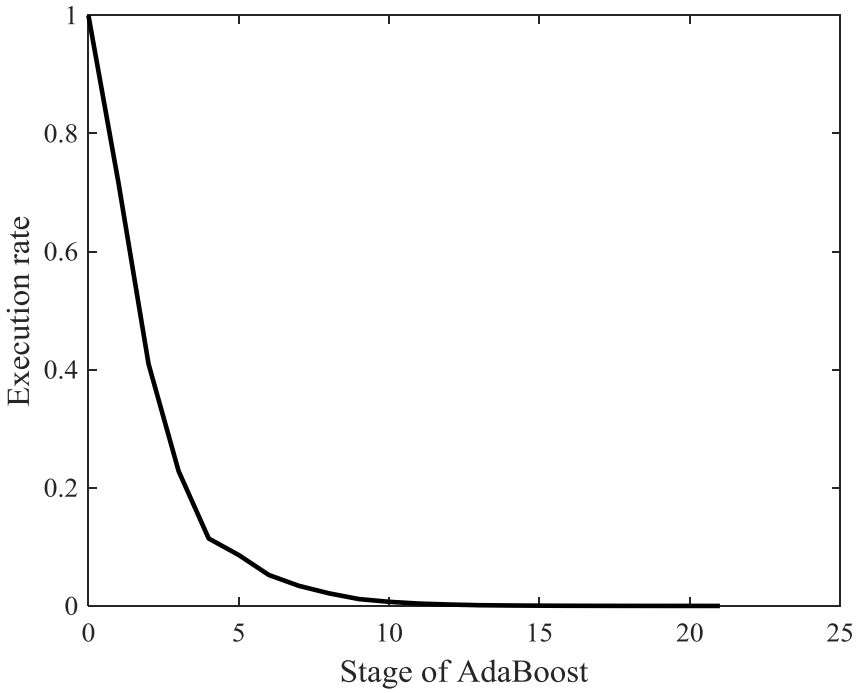


Figure 4.9 Execution rate of each cascade stage. Note that the execution rate of early stages is much higher than latter stages.

4.2.3 Parallelism Analysis

AdaBoost algorithm exhibits a mass of parallelism at different levels, including both coarse-grained and fine-grained parallelism. Figure 4.10 illustrates the available parallelism.

- Image pyramid level: To detect large-sized objects, detection is performed on each image layer. The processing of each image layer can be executed in parallel.
- Image partition level: Large input images can be divided into several partitions. The computation of each partition is independent and can be processed simultaneously.
- Weak classifiers level: In AdaBoost algorithm, as shown in Table 4.1, there are tens or hundreds of weak classifiers in each stage. A total of weak classifiers are several thousands. These weak classifiers are independent and can be processed in parallel.

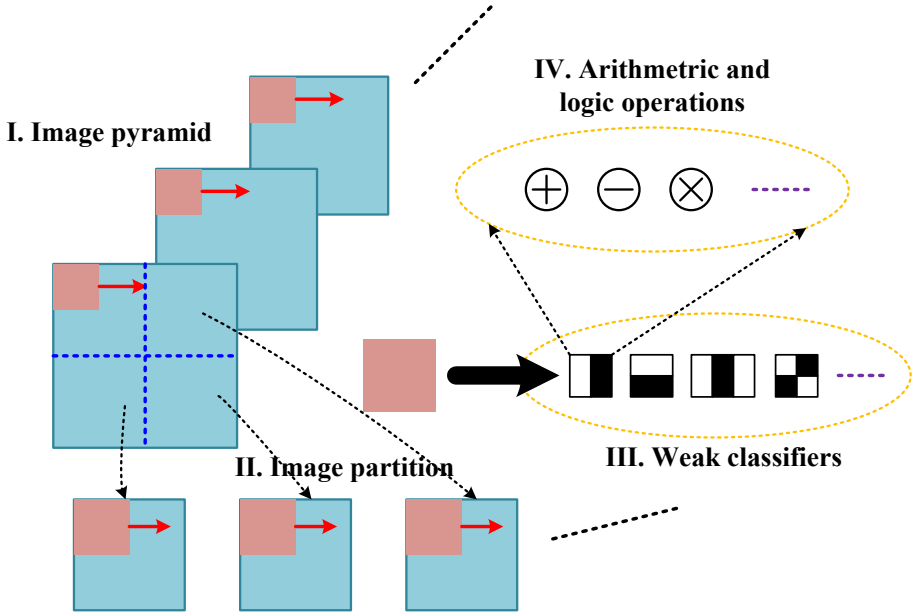


Figure 4.10 Available parallelism in AdaBoost algorithm.

- Arithmetic operations level: The calculation of Haar-like features are performed on different data with the same operation. Thus, these operations can be executed concurrently.

The parallelization schemes of image pyramid and image partition belong to course-grained parallelism, while the parallelization schemes of weak classifiers and arithmetic operations belong to fine-grained parallelism. Considering that a single processor is not suitable for exploiting course-grained parallelism, we devote to exploiting fine-grained parallelism in this paper.

4.3 Object Detection Processor Architecture

4.3.1 ASIP Architecture

This section presents overall architecture of the proposed ASIP and data transferring between memory and ASIP engine. Figure 4.11 illustrates a block diagram of the proposed ASIP architecture. The main custom hardware components are vector registers, a register array, storage elements for classifier data and

ASIP engine, which are shown with darked blocks in Figure 4.11. To increase the parallelism, the pipeline of the proposed ASIP is extended to seven stages (FE: fetch stage, DC: decode stage, EX1−EX5: five execute stages). ASIP engine includes integral image calculation (IIC) unit, Haar-like feature value calculation (HFVC) unit, color space conversion (CSC) unit and image resize (RSZ) unit. IIC unit includes hardware component for integral image extraction (IIE) and cumulative row sum (CRS) calculation. IIC unit can perform under two modes: initial mode and update mode. The initial mode uses only input pixels to compute the integral image, while the update mode makes use of the integral image of current sub-window to calculate the integral image of next sub-window. Thus, there are two kinds of inputs for ASIP engine: input image and integral image. The input image is stored in data memory. A single memory access returns a 32-bit value, which includes four-pixel values. IIC unit utilizes these inputs (through interface *in0*) to compute integral image values and saves (through interface *out0*) the integral image values to integral image register arrays. 20 x 20 x 17 bit is needed to store the entire integral image of a sub-window (20 x 20 pixels). When in update mode, ASIP engine reads integral image through interface *in1* (340 bit) and writes the updated integral image through interface *out1* (340 bit). To perform Haar-like feature values calculation, integral image values are directly loaded from integral image register arrays (through interface *in2*). CSC unit and RSZ unit load pixels through interface *in3* (64 bit) and save the outputs through interface *out2* (64 bit).

According to the analysis in Section 4.2.2.3, classifier data of first 5 stages is stored in register files for high-speed processing, while the others are stored in memory. These classifier data serve as input address of the integral image register arrays. To support 8-way integral image value extraction in parallel, the width of the interface is 8x32 bit. Note that since the size of the rectangle is constrained by the size of sub-window (20 x 20 pixels), 32 bit is enough to hold the rectangle information (left-corner position, width, and height).

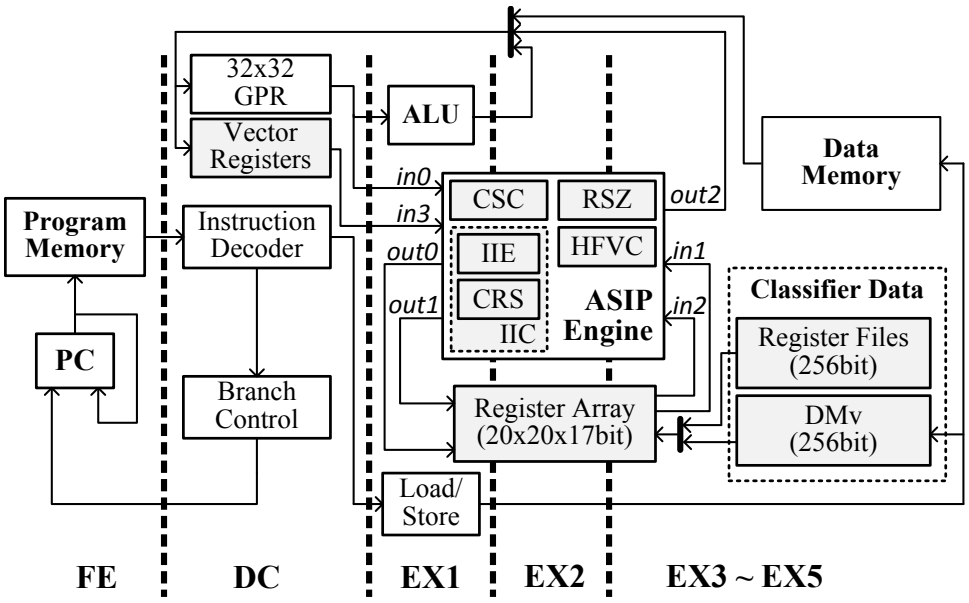


Figure 4.11 Overall architecture of the proposed ASIP.

4.3.2 Custom Hardware Component

In this section, we introduce special function units to accelerate integral image calculation and Haar-like feature value calculation.

Figure 4.12 shows the architecture for cumulative row sum calculation. The architecture consists of five full adder trees and a line buffer. The full adder trees compute the cumulative row sum for each row of sub-window. The line buffer stores intermediate cumulative row sum. The cumulative row sum calculation is based on a five-stage pipeline to be operated at a high clock frequency. Figure 4.13 describes a block diagram of the integral image calculation architecture. The integral image update process is presented in Section 4.2.2.2.

Since the integral image values of a sub-window are stored in register arrays, it is possible to access all integral image values simultaneously. Figure 4.14 illustrates the integral image value extraction architecture. It mainly contains a 2-D register array, row multiplexers, column multiplexers and line buffers. The architecture firstly reads (x_0, y_0, w, h) which represents the top-left corner position and the size of a rectangle in a feature. The bit with of x_0, y_0, w and h is $8bit$. Then, the row multiplexer selects two rows of the integral image

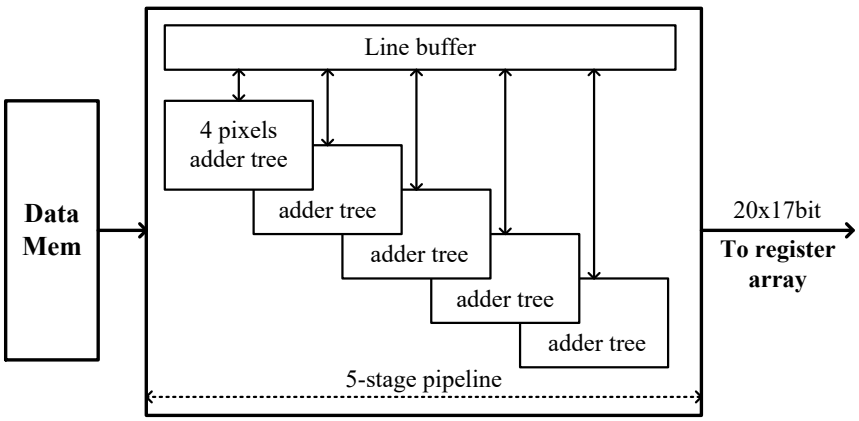


Figure 4.12 Pipelined cumulative row sum calculation architecture.

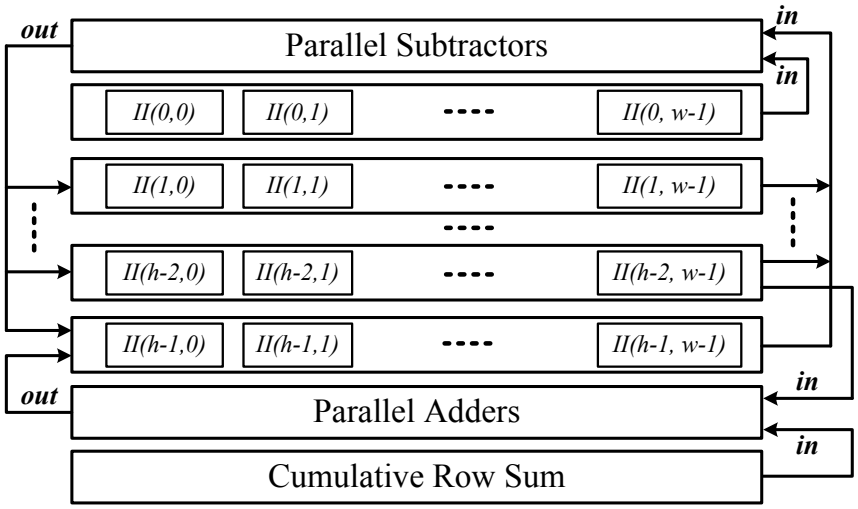


Figure 4.13 Integral image calculation architecture.

from register array using $(y_0, y_0 + h)$. Next, based on the selected two rows integral image, the column multiplexer extracts integral image values at four corners of the rectangle using $(x_0, x_0 + w)$. After that, these integral values are sent to Haar-like feature calculation unit. There are eight sets of row-column multiplexers in the architecture, which can handle eight rectangles at a time.

The repeated arithmetic operations are known to be suitable for parallel processing architectures [47]. SIMD is one of the most suitable architecture for exploiting data-level parallelism. In this paper, SIMD architecture is introduced to accelerate the Haar-like feature value calculation.

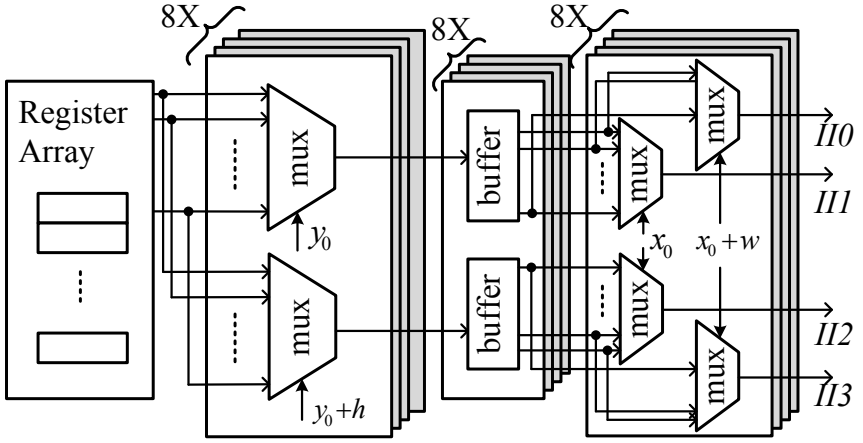


Figure 4.14 Integral image extraction architecture.

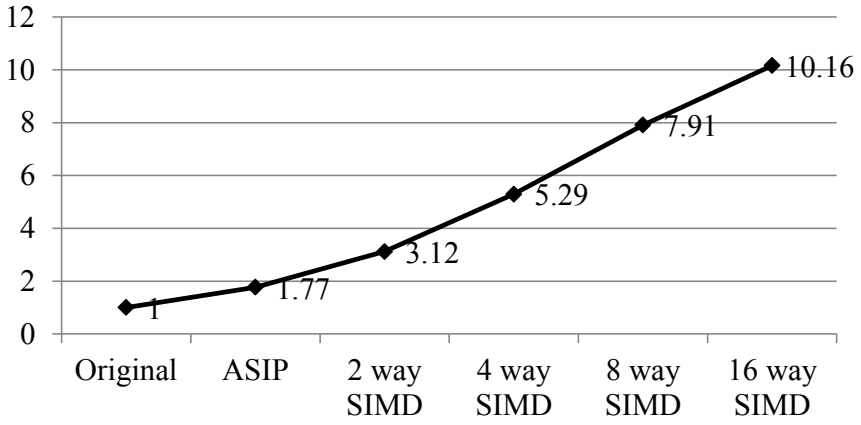


Figure 4.15 Performance gain under different parallelism.

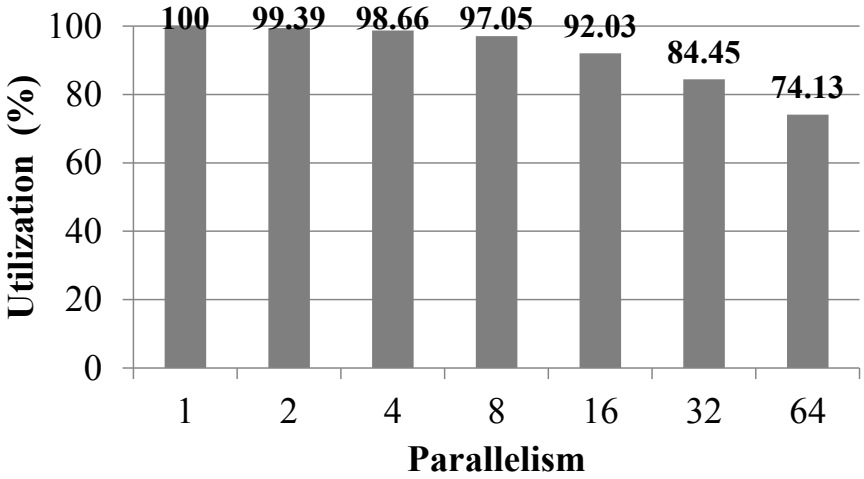


Figure 4.16 Hardware resource utilization rate under different parallelism.

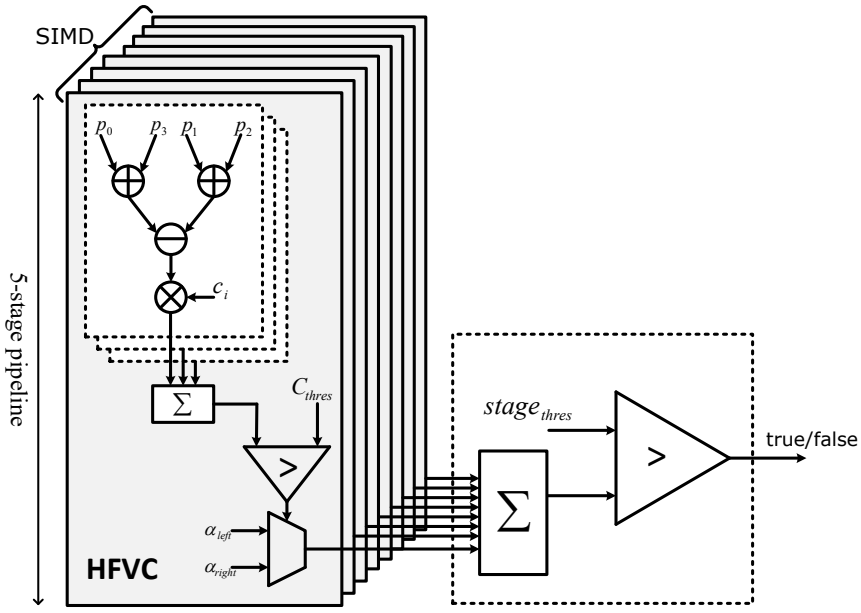


Figure 4.17 Haar-like feature value calculation architecture.

To find out how many ways of SIMD is reasonable for Haar-like feature calculation, we evaluate the performance gain under different parallelism. The result of performance gain comparison is shown in Figure 4.15. As can be seen, when applying wider SIMD, the expected speed up can be obtained. Compared to the pure software implementation, 1.7x speed-ups can be obtained by ASIP with custom instructions. For 16-way SIMD with custom instructions, the speed-ups can achieve 10x.

However, due to the unbalanced structure of the cascade classifier structure, higher data-level parallelism may induce lower hardware resource utilization. The comparison of hardware resource utilization under different data-level parallelism is shown in Figure 4.16. For example, if the first stage contains 9 features, for 1-parallelization, the hardware utilization rate is 100%. However, for 2-parallelization, the utilization rate becomes 90%. Moreover, the chip area cost increases as the parallelization increases. In this design, to guarantee the hardware resource utilization rate can achieve over 95%, an 8-way SIMD architecture is adopted for Haar-like feature value calculation.

Table 4.2 Special instruction in the proposed ASIP (AdaBoost)

Type	Special Instructions	Function Description
A	do	Zero overhead loop
	min/max	minimum/maximum value selection
	selz/seln	Conditional move
B	rgb2gray	Color space conversion
	hzip/vtip	Image resize
C	viic	Integral image calculation
	vvarc	Variance calculation
	viie	Integral image value extraction
	vhfvc	Haar-like feature value calculation
	vreginit	Set classifier data registers

The custom hardware component for Haar-like feature calculation is shown in Figure 4.17. The input is from the integral image extraction unit.

4.3.3 Instruction Set Extensions

To accelerate the AdaBoost algorithm on an ASIP, the associated acceleration instruction need to be added as well as special function units. Table 4.2 shows an overview of the special instructions for the proposed ASIP. The custom instructions include three categories: Type A, e.g. do instruction, which allows performing fast looping and can be applied to most of C applications; Type B, e.g. rgb2gray, hzip and vtip instruction, which performs color space conversion and image resizing, are widely used in digital image processing; Type C, e.g. viic and vhfvc instruction, which performs integral image and Haar-like feature calculation, are customized for object detection using AdaBoost algorithm.

4.4 Experimental Results

This section presents results obtained for implementation of the proposed ASIP described in Section 4.3. The results indicate that the proposed ASIP is

competitive with implementations based on other hardware architectures, e.g. general purpose embedded processors, DSPs and ASICs.

In the proposed ASIP, all the architecture customizations were implemented using nML [30], which is a high-level architecture description language. This is aimed to shorten the development time of ASIPs.

Our ASIP is synthesized using a 90nm standard CMOS technology under worst case conditions with Synopsys Design Compiler [48]. The implementation results are summarized in Table 4.4. The ASIP can achieve a maximum clock frequency of 250MHz, with a silicon area of $1.91mm^2$. The power consumption is estimated to be 198mW [49]. The maximum performance is 32fps on VGA video with 91% detection accuracy (with $5x10^{-6}$ false positive rate, shown in Figure 4.8) when the image scale factor is set to 1.2.

For comparison purpose, simulation results for the proposed ASIP are presented, but also for a pure software implementation of the algorithm running on baseline RISC processor. Table 4.3 shows the cycle counts comparison for the AdaBoost algorithm on the baseline processor and the proposed ASIP. The results presented here were obtained from cycle-accurate simulations performed using the Synopsys ASIP Designer [31]. Utilizing all the special function units presented in Section 4.3, the resulting speed-ups are 30x for integral image calculation of a sub-window (20 x 20 pixels) in initial mode, 303x in update mode, 21x for Haar-like feature values calculation. Compared to the optimized pure software implementation of AdaBoost algorithm on baseline RISC processor, the number of total cycle counts could be reduced by a factor of 13.7x using the proposed ASIP.

Figure 4.18 shows parts of the detection results. The target object is frontal human face. The detection is performed on MIT + CMU faces data set [45]. The shown results are tested on the images affected by illumination and noise and the ones with different rotation of faces.

Table 4.5 shows the comparison with software implementations of AdaBoost algorithm when mapping on different embedded processors. Application executed by ARM946 [50] is compiled by ARM C Compiler and evaluated under ARM

Table 4.3 Cycle counts comparison for AdaBoost algorithm on baseline processor and proposed ASIP

Function	**Baseline Processor**	**ASIP**	**Speed-UP**
Integral Image	3333	111	30.6x
Integral Image Update	3333	11	303.4x
Run Classifier	83,173,439	3,852,962	21.5x
Color Space Conversion	6,825,759	664,099	10.2x
Image Resize	10,506,970	1,777,624	5.9x
Total Cycle Counts	105,304,390	7,652,396	13.7x

Table 4.4 ASIP implementation results (AdaBoost)

Parameter	**ASIP**
Technology	90nm
Frequency	250MHz
Supply Voltage	1.0V
Area	1.91 mm^2
Power Consumption	198mW
Image Resolution	VGA(640x480)
Image Scale Factor	1.2
Frames per Second	32
Detection Accuracy	91%

RealView Development Suite (RVDS) environment [56]. Application executed by TMS320C64+ [51] is compiled by TI Compiler with level 3 optimization and evaluated under TI Code Composer Studio (CCS) environment [57]. Compared with software implementations on ARM946, our ASIP shows 32x, 10x and 6.8x better throughput, area efficiency, and power efficiency, respectively. Compared with TMS320C64+, our ASIP has 7x, 224x and 18.8x more throughput, area efficient and power efficient, respectively.

Table 4.6 shows the comparison with previous hardware implementations, which use the AdaBoost algorithm for object detection. Compared with [52], [54], our ASIP shows better throughput. Compared with [55], our frame rate

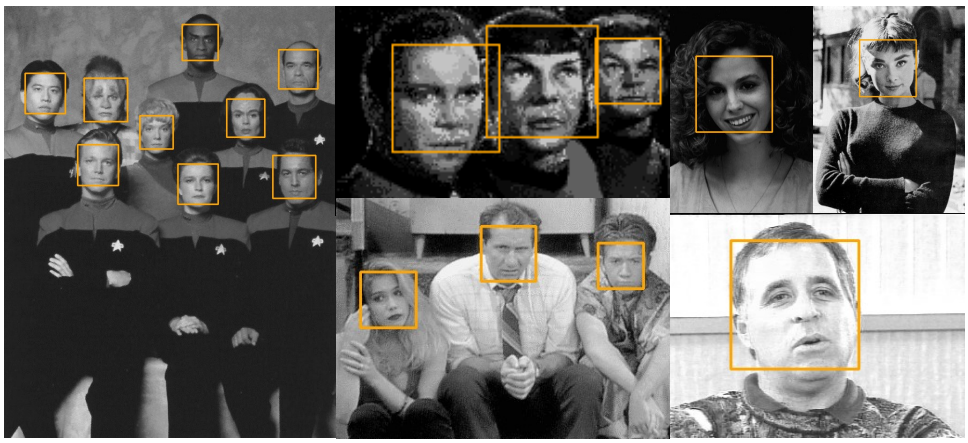


Figure 4.18 Sample of frontal face detection results. Images affected by illumination, noise and with different rotation of faces.

Table 4.5 Comparison with software implementations on embedded processors (AdaBoost)

Parameter	**ARM946E-S** [50]	**TMS320C64+** [51]	**TCT Processor**	**Proposed ASIP**
Technology	90nm	90nm	90nm	90nm
Frequency	441MHz	625MHz	250MHz	250MHz
Area	$0.613mm^2$ (w/o cache)	$64mm^2$ [a] (with cache)	$0.108mm^2$ (w/o cache)	$1.91mm^2$ (w/o cache)
Power Consumption	41.9mW	562.5mW	5.6mW	198mW
Cycle Counts	439,221,618	128,896,336	105,304,390	7,652,396
Frame Rate	1.00fps	4.84fps	2.38fps	32fps
Area Efficiency (fps/KGates)	0.0046	0.00021	0.621	0.0471
Power Efficiency (mW/fps)	41.9	116.2	2.35	6.19

[a] Chip area is estimated from the PBGA package.

is not an advantage, but the performance is competitive by taking the image resolution into account. Compared with hard-wired designs [52],[53],[54],[55], our ASIP has 12.4x, 0.88x, 4.1x and 31x more area efficiency, respectively.

Table 4.6 Performance comparison with previous works (AdaBoost)

Parameter	Hanai [52]	Tsai [53]	Hiromoto [54]	Kyrkou [55]	This Work
Technology	90nm	40nm	65nm	65nm	90nm
Frequency	54MHz	220MHz	160.9MHz	800MHz	250MHz
Gate Count (2NAND)	525K [a]	1340K	2600K [b]	22M [a]	680K
Image Resolution	320x240	640x480	640x480	320x240	640x480
Frame Rate	8fps	72fps	30fps	133fps	32fps
Detection Accuracy	81%	90%	92.8%	95%	91%
Area Efficiency (fps/KGates)	0.0038 [c]	0.0537	0.0115	0.00152 [c]	0.0471

[a] Gate count is estimated from the number of transistors.
[b] Gate count is estimated from the number of LUTs and register bits.
[c] For comparison, the resolution is up-scaled to VGA.

Moreover, compared with hard-wired designs, our ASIP with a programmable RISC processor, which can be reused by other algorithms, while the hard-wired design hardly applies to other applications.

4.5 Related Work

The AdaBoost-based learning algorithm shows high performance in detection speed and accuracy [42]. However, the computational cost still remains too high to support embedded applications. In order to achieve faster detection, some works on hardware implementation have been proposed [52, 53, 58, 54, 55].

A versatile recognition processor was proposed in [52]. Three techniques in the architecture and circuit level were introduced to handle the cascade classifier and the Haar-like features. A classifier data cache was introduced to reduce on-chip SRAM size. A Haar-like feature coordinates decoder was used to reduce

the on-chip SRAM size and access. A Haar-like feature value extractor was proposed to improve throughput. For QVGA image, the proposed processor achieves frame rate of 8fps. However, it is difficult to meet the requirement of real-time object detection.

Tsai et al. [53] proposed a vision processor for versatile automotive applications. Similar to [52], a classifier data compression technique, which supports non-tilted Haar-like feature types, was adopted to reduce on-chip memory size. Also, a 4-bank on-chip classifier memory was used to reduce memory bandwidth and classifier data access. To improve the throughput, four Haar-like feature values were calculated in parallel.

Cho et al. [58]. chose a FPGA as a target device instead of an ASIC. They used hardware design techniques such as input image scaling, integral image generation for each sub-window instead of the whole input image, pipelined processing, as well as triple classifier processing in parallel to accelerate the processing speed of the proposed face detection system. For VGA video, the detection rate can achieve 15fps. However, they do not mention the detection accuracy of the proposed system.

A hybrid execution model incorporating parallel processing with sequential processing modules was presented in [54]. The early stages of the cascade classifier is executed in parallel, which are frequently used in the algorithm, while the subsequent stages are mapped to sequential processing modules. However, upon different training data set, experiments need to be conducted to split the parallel and sequential stages every time.

Kyrkou et al. [55] proposed a parallel classification engine using a systolic array implementation approach. This architecture can parallelize integral image computation and be applied to different application scenarios by training different classifiers. Also, they used both image downscaling and sub-window upscaling to perform multi-scale detection. However, such an approach would result in a high consumption of hardware resource.

4.6 Summary

In this chapter, we proposed an ASIP-based processor for object detection using AdaBoost-based learning algorithm. This approach utilizes Haar-like features as weak classifiers for pattern classification to determine the existence of an object. The main contributions of this work are threefold. First, the most computational intensive arithmetic function and the memory bandwidth bottleneck are identified by thorough analysis of the detection algorithm. Second, Single Instruction Multiple Data (SIMD) architecture is adopted to fully exploiting data-level parallelism within the algorithm. An efficient integral image and Haar-like feature calculation architecture is proposed to improve the throughput. Also, an efficient classifier data storage mechanism is introduced to meet the built-in nature, coarse-to-fine search approach, of the AdaBoost algorithm. Third, with proper application-to architecture mapping, the proposed ASIP exhibits an advantage in terms of both chip area efficiency and power efficiency when compared to embedded processors, and shows better chip area efficiency when compared to hard-wired designs.

Chapter 5

Object Detection Processor with HOG Feature and SVM Classifier

5.1 Histogram of Oriented Gradients

This section gives an overview of the original HOG-based object detection algorithm. In the HOG algorithm, sliding window approach is applied to detect objects. HOG features are collected over the detection window. There are two computation elements (cell and block) in HOG feature extraction. Figure 5.1 (a) illustrates the relationship of cell, block and detection window. The detected image is divided into 8x8 pixels spatial regions called cells. Every 2x2 cells form a block. The HOG descriptor is extracted from the overlapping blocks within a detection window and then passed to a pre-trained linear SVM classifier for object/non-object classification [59]. Details about HOG algorithm are described as below.

5.1.1 Gradient Computation

The input image is converted to a gradient image with a simple centered 1-D mask $[-1, 0, 1]$. Image gradients are computed as below.

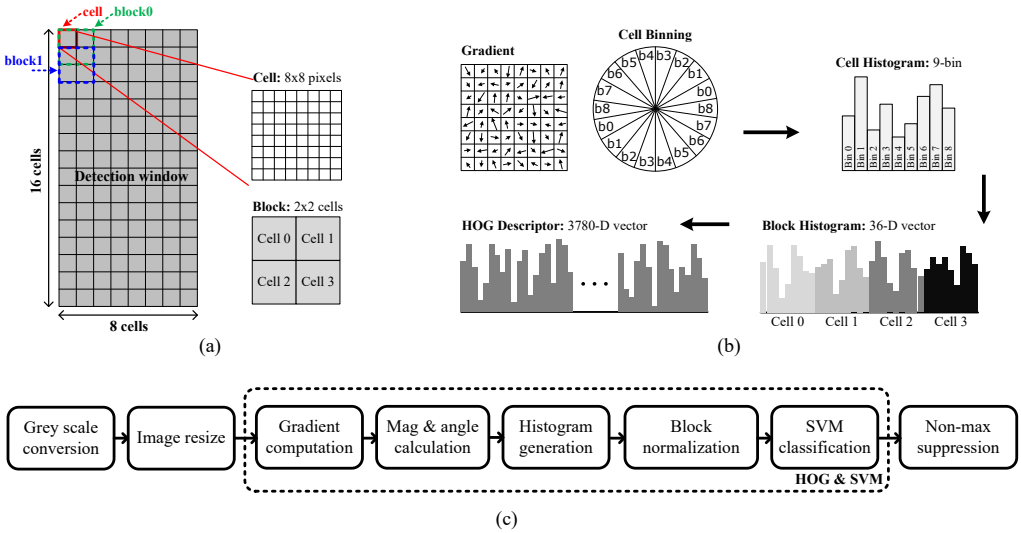


Figure 5.1 (a) Basic concept of cell and block. The detection window is divided into non-overlapping 8x8 pixels spatial regions called cells. The size of detection window is 64x128 pixels. It consists 8x16 cells. Every 2x2 cells form a block. (b) HOG descriptor generation. The orientation is divided into 9-bins in the range of $0° - 360°$. Each cell generates a 9-bin histogram. HOG descriptor is a 3780-D vector. (c) HOG-based object detection flow.

$$\begin{cases} f_x = f(x+1, y) - f(x-1, y) \\ f_y = f(x, y+1) - f(x, y-1) \end{cases} \tag{5.1}$$

where $f(x, y)$ is the pixel value of the input image at position (x, y). f_x and f_y are horizontal gradient and vertical gradient, respectively. Then, for each pair of f_x and f_y, the magnitude $m(x, y)$ and orientation $\theta(x, y)$ can be given as

$$\begin{cases} m(x, y) = \sqrt{f_x^2 + f_y^2} \\ \theta(x, y) = \arctan \dfrac{f_y}{f_x} \end{cases} \tag{5.2}$$

5.1.2 Histogram Generation and Block Normalization

Within a cell, each pixel calculates a weighted vote, and the votes are accumulated into orientation bins. Figure 5.1 (b) shows the binning diagram. The orientation is divided into 9 bins in the range of $0° - 360°$. The weighted vote

is calculated according to the difference between the gradient orientation and the histogram bin edge. The voting weight is written as

$$\alpha = \frac{b \cdot \theta}{\pi} - floor(\frac{b \cdot \theta}{\pi} - 0.5) \tag{5.3}$$

where b is 9, the total histogram bins of a cell. To reduce aliasing, both values of two neighboring bins are updated. The updated values are given as

$$\begin{cases} m_{current} = m \times (1 - \alpha) \\ m_{next} = m \times \alpha \end{cases} \tag{5.4}$$

where $m_{current}$ and m_{next} are updated values for current bin and next bin, respectively. Also for anti-aliasing, votes of each cell are bilinearly interpolated to the neighboring cells. Furthermore, a Gaussian spatial window is applied to each pixel before accumulating orientation votes. Thus, the final votes can be written as

$$\begin{cases} vote_0 = weight_{gauss} \times weight_{ip} \times m_{current} \\ vote_1 = weight_{gauss} \times weight_{ip} \times m_{next} \end{cases} \tag{5.5}$$

where $weight_{gauss}$ and $weight_{ip}$ are Gaussian spatial weight and bilinearly interpolation weight, respectively.

To reduce the impact of local variances in illumination and foreground-background contrast, the block histograms are normalized as

$$\vec{V} = \frac{\vec{v}}{\sqrt{||\vec{v}||^2 + \varepsilon^2}} \tag{5.6}$$

where $||\vec{v}||^2$ ($||\vec{v}||^2 = v_1^2 + v_2^2 + ... + v_{36}^2$) is the energy of block histogram $\vec{v}$ and ε is a small constant to avoid division by zero.

A 36-D feature is obtained from four cell histograms within one block. For a 64x128 pixels detection window, all normalized histograms from 105 blocks are concatenated together, generating a 3780-D HOG descriptor. Figure 5.1 (b) illustrates the HOG descriptor generation.

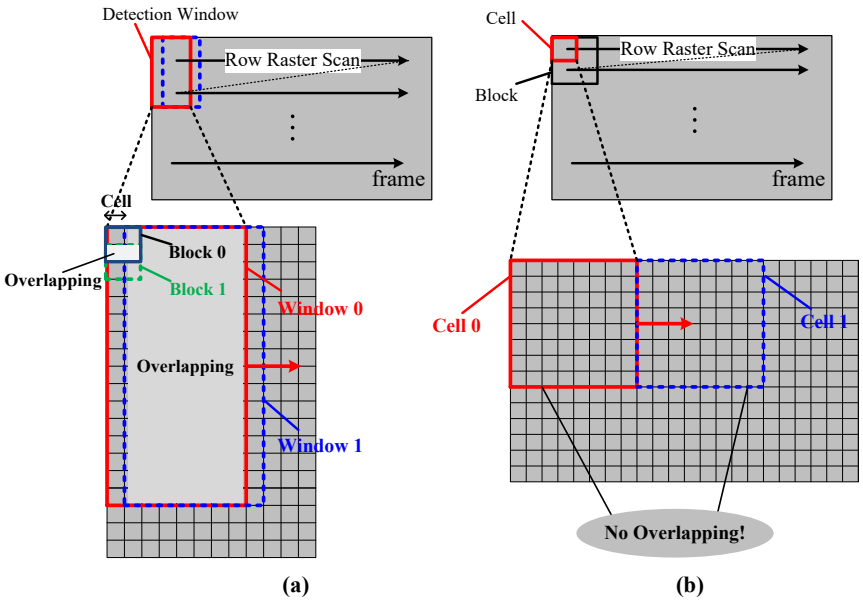


Figure 5.2 (a) Window-based scanning method. Two kinds of overlapping: detection windows overlapping and blocks overlapping. (b) Cell-based scanning method. No cell overlapping between neighboring cells.

5.1.3 SVM Classification

Finally, the HOG descriptor is passed to a linear SVM classifier for final classification. The SVM classifier creates the maximum distance (margin) from two classes to achieve superior classification performance [22]. The final output can be written as

$$s = \vec{W} \times \vec{V} + s_0 \tag{5.7}$$

where $\vec{W}$ is the pre-trained SVM coefficients, s_0 is an offset. The classification output s is referred as a score, and then is compared with a threshold to make a decision whether or not a detection window contains an object. The HOG-based object detection flow is shown in Figure 5.1 (c).

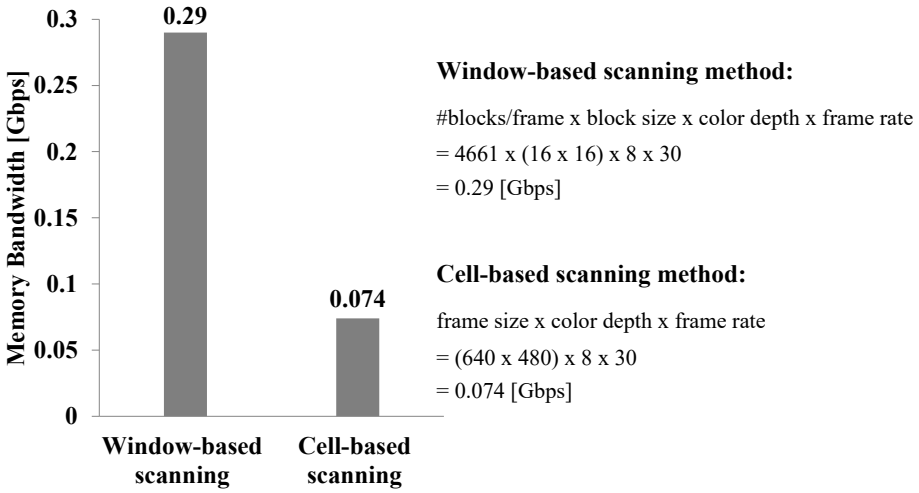


Figure 5.3 Memory bandwidth requirements analysis. Only blocks overlapping is considered in window-based method.

5.2 Hardware-Friendly HOG Algorithm

5.2.1 Cell-Based Scanning Method

Figure 5.2 (a) depicts the window-based scanning method used in original HOG algorithm. The detection window scans the entire input frame in row raster scanning manner. The window stride is 8 pixels (cell width or height) in both horizontal and vertical direction. Within each detection window, HOG feature is extracted from the overlapping blocks. Two kinds of overlapping come with window-based scanning approach: detection windows overlapping and blocks overlapping. These overlapping result in a high memory bandwidth requirement, as well as an extra on-chip memory.

Consequently, a data reuse scheme is desirable to reduce the memory bandwidth requirement. In this work, we adopt cell-based scanning method for object detection. Figure 5.2 (b) shows the cell-based scanning method. HOG features are collected from cell-based scanning approach. In contrast to window-based scanning method, no cell overlapping occurs between neighboring cells.

To better understand the benefit of cell-based scanning method. We analysis the memory bandwidth requirements for both scanning methods. For window-based scanning method, the detection window overlapping can be eliminated

by an on-chip memory. Thus, we only consider the block overlapping in the analysis. Figure 5.3 gives the analysis result. The window-based approach requires a memory bandwidth of 0.29 Gbps for VGA input, while cell-based approach requires 0.074Gbps. A 4x memory bandwidth reduction is available with the cell-based method. This is owing to cell sharing for block histogram generation, which can avoid reloading pixels for the next block.

5.2.2 On-The-Fly SVM Calculation

In window-based scanning method, HOG features are extracted from 105 blocks within a detection window. Then, the HOG features are multiplied with the corresponding SVM weights and accumulated as the output score. In this work, an on-the-fly SVM calculation approach is carried out. Once the HOG feature of a block is extracted, it is immediately used for partial score computation so that it is never buffered or recomputed. This approach can reduce on-chip memory. Figure 5.4 shows the basic idea of on-the-fly SVM calculation. A block feature is shared with 105 detection windows maximally, but at different positions within each window. For example, in Figure 5.4, the block feature is the first feature for Window B and the 105th feature for Window A. The block feature is multiplied and accumulated with the SVM weights of each window that enclose that block (e.g., Window N). All computations required that block HOG feature must be completed before discarding it.

5.2.3 HOG Algorithm Simplification

This section presents the simplified HOG algorithm for hardware implementation. To reduce mathematical complexity and minimize the hardware cost, the following techniques are introduced in this work.

- Gradient magnitude calculation using bitwise verification method [60] and histogram bin computation without actual gradient orientation.
- Approximation weighted voting for orientation and spatial anti-aliasing.

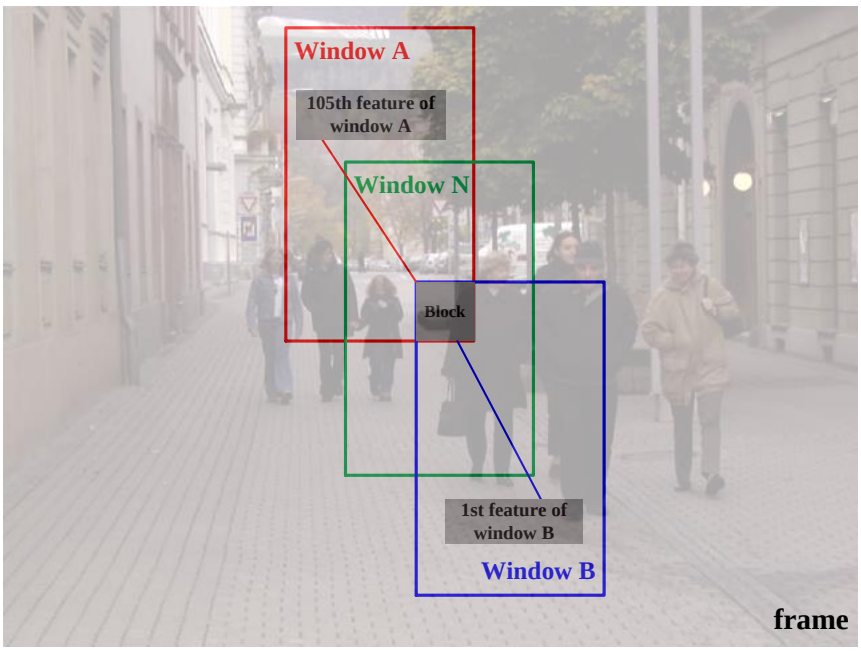


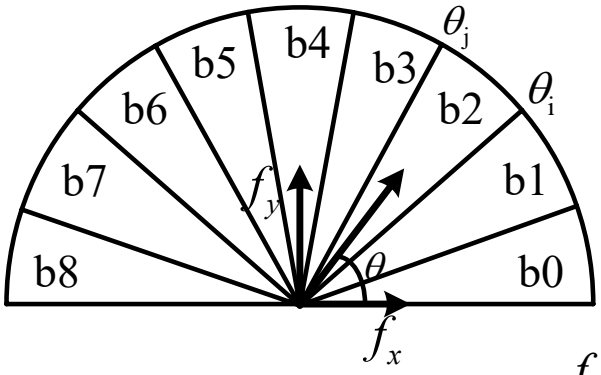
Figure 5.4 Basic idea of on-the-fly SVM calculation. A block feature belongs to 105 detection windows maximally. The pedestrian image is from [8].

- Block histogram normalization with a fast inverse square root calculation.

5.2.3.1 Gradient Computation

As shown in (5.2), square root operation is required for gradient magnitude calculation. However, the square root is expensive in hardware implementation. In this work, a bitwise verification method [60] is adopted for low-cost implementation to approximate $m(x, y)$. This method is a variation of the traditional "subtract and shift" division algorithm [61]. For an unsigned 32-bit integer, 16 iterations are needed to find the approximation of the square root. Each iteration can be carried out by shifts and adds only, requiring no multiplications.

For orientation binning, since the orientation is simply applied to select the histogram bin, the actual orientation is not necessary to compute. As shown in Figure 5.5, the boundary (e.g., θ_i and θ_j) of each bin is known as constants. Therefore, histogram bin can be computed through the simple inequality discriminant rather than the complex actual orientation calculation.



$$\tan\theta_i < \tan\theta \le \tan\theta_j \qquad \tan\theta = \frac{f_y}{f_x}$$

$$f_x \cdot \tan\theta_i < f_y \le f_x \cdot \tan\theta_j$$

Figure 5.5 Histogram bin calculation without the actual value of gradient orientation. Note that θ_i and θ_j are constants. In this example, the pixel is assigned to bin 2.

5.2.3.2 Histogram Generation

To reduce aliasing, magnitudes are bilinearly interpolated between the neighboring bins in both orientation and position [59]. For orientation anti-aliasing, the vote is a function of the gradient magnitude and the orientation at the pixel (shown in (5.4)). However, according to Section 5.2.3.1, $\theta(x, y)$ is not calculated accurately. Therefore, in this paper, α in (5.4) is set to a reasonable constant (0.5) for weighted magnitude calculation. For spatial anti-aliasing, the weighted vote is computed through a bilinear interpolation method and Gaussian weighting method (shown in (5.5)). Since both the interpolation weight and Gaussian weight are fixed values at a certain position. To reduce the multiplications for weighting, the weights are approximated to a fixed value, which can be expressed as a power of two. Thus, a low-cost bit shift operations can replace the expensive multiplications for anti-aliasing.

Table 5.1 Parameters of HOG algorithm

Algorithm Parameter	**Specification**
Detection window size	64 x 128 pixels
Cell size	8 x 8 pixels
Block size	16 x 16 pixels
Window stride	8 pixels, vertically and horizontally
Block stride	8 pixels, vertically and horizontally
# of orientation bins	9 bins ($0° - 360°$)
Normalization method	L2-norm
SVM weights and threshold	Default value in OpenCV
Size of HOG descriptor	3780=(7x15 blocks)x(2x2 cells)x(9 bins)

5.2.3.3 Block Normalization

According to Section 5.1.2, the block normalization is done by dividing the 36-D block histogram by its energy (L2-norm). To avoid square root and division operations, Newton's method is introduced to approximate the inverse square root [62]. The approximation value is obtained by the iteration calculation as below.

$$d_{i+1} = d_i \cdot (3 - d_i^2 \cdot ||\vec{v}||^2)/2 \tag{5.8}$$

where d_{i+1} is the approximation value of $1/\sqrt{||\vec{v}||^2 + \varepsilon^2}$. One iteration consists of three multiplications, a subtraction and a shift operation. The initial value d_0 is important to reduce the number of iterations. In this paper, we use the magic number (i.e. 0x5f3759df) [63] to obtain d_0. The initial value can be expressed as

$$d_{0f} = 0x5f3759df - (||\vec{v}||_f^2 >> 1) \tag{5.9}$$

where d_{0f} and $||\vec{v}||_f^2$ are values of d_0 and $||\vec{v}||^2$ in IEEE 754 format, respectively. With the proper initial value, only one iteration of Newton's method can yield a approximation value with adequate accuracy.

Table 5.2 Optimized fixed-point bit-width

Parameter	Sign	Integer	Fraction
Gradient	1	8	0
Gradient magnitude	0	9	0
Gradient orientation	0	4	0
Cell histogram	0	12	0
HOG feature	0	0	8
SVM coefficient	1	0	7
Classification buffer	1	3	10

5.2.4 Performance Evaluation

Note that the simplification might affect the detection rate of HOG algorithm. Therefore, in this section, we evaluate the detection rate of simplified HOG algorithm with cell-based scanning method and make a comparison with the original HOG algorithm with window-based scanning method. As a starting point, we use OpenCV [32] to develop the detection program. Table 5.1 summarizes the parameters used in both HOG algorithms. The optimized fixed-point bit-width for simplified HOG algorithm is shown in Table 5.2. The detection is performed on INRIA pedestrian image dataset [8] and MIT pedestrian dataset [64]. The INRIA dataset contains 288 positive images (include 1126 person) and 453 negative images. The labeled person is usually standing against a wide variety variations. The MIT dataset contains 924 images (include 924 person). The labeled pedestrian is in front or back views with a limited range of poses.

The experimental result is shown in Figure 5.6. The graph shows the precision and recall curves of the modified HOG algorithm and the original HOG algorithm. The PR curve shows a relationship of precision against recall at different threshold settings [46]. Precision and recall are defined as

$$\begin{cases} precision = \dfrac{TP}{TP + FP} \\ recall = \dfrac{TP}{TP + FN} \end{cases} \tag{5.10}$$

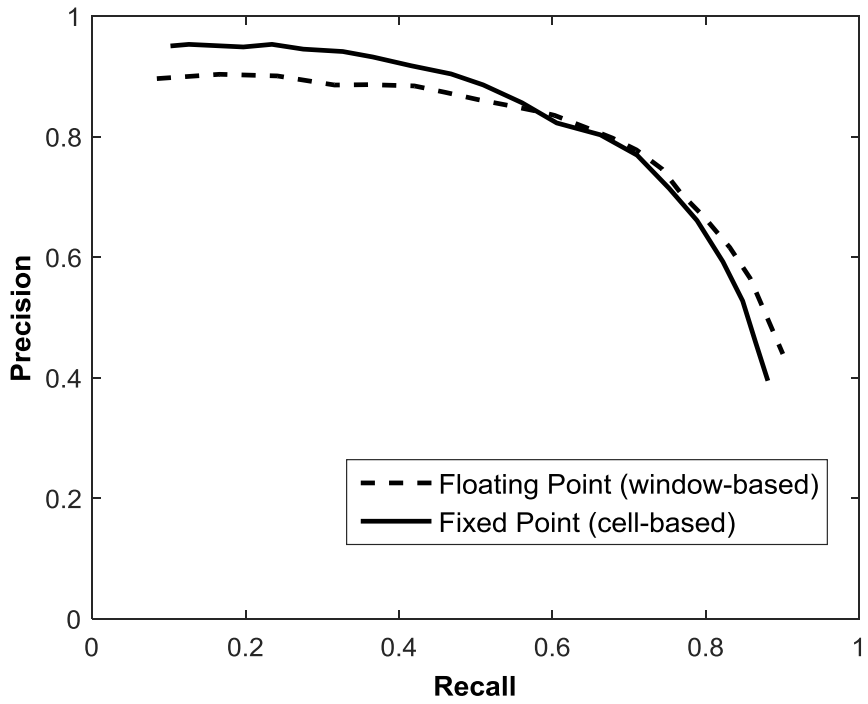


Figure 5.6 PR curves of window-based scanning method and cell-based scanning method (INRIA dataset).

where TP, FP and FN are true positives, false positives, and false negatives, respectively. True positives and false positives are the numbers of items correctly or incorrectly labeled as the positive class. False negatives are items incorrectly labeled as the negative class. Thus, precision is defined as a proportion of detected windows that are objects, and recall means the proportion of objects that are detected [65].

In Figure 5.6 and Figure 5.7, the solid line, and the dashed line indicate PR curves of modified HOG algorithm and original HOG algorithm, respectively. Figure 5.6 and Figure 5.7 shows the PR curves on INRIA pedestrian dataset and MIT pedestrian dataset, respectively. In the lower recall region, the simplified HOG algorithm shows better precision, while in the higher recall region, the original HOG algorithm exhibits superior precision. This performance gap is very small. Thus, the simplified HOG algorithm with optimized fixed bit width would not decrease the performance of HOG algorithm.

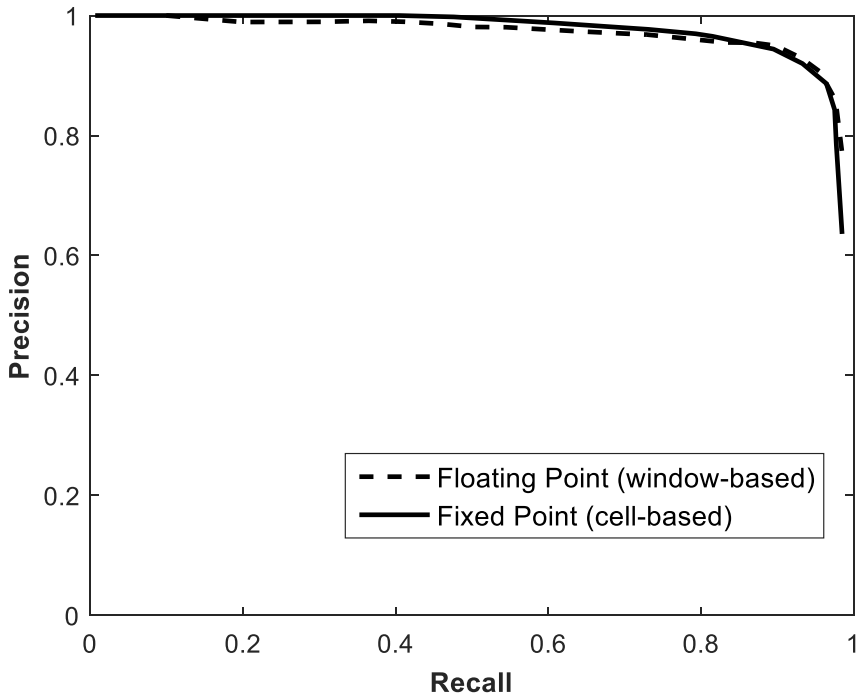


Figure 5.7 PR curves of window-based scanning method and cell-based scanning method (MIT dataset).

5.3 Architecture of Vision Processor

5.3.1 ASIP Architecture

As a first step, we profiled the detection program on the baseline processor, to find the most critical computation in HOG algorithm. Without specification, in the rest of the paper, HOG algorithm refers to the modified algorithm described in Section 5.2 and baseline processor refers to TCT processor. Figure 5.8 reports the profiling result. The top-5 computationally intensive functions take 99.96% of the total cycle counts. For our object detection engine, we focus on implementing the most computationally intensive functions on ASIP: gradient computation, magnitude and orientation calculation, histogram generation, block histogram normalization and SVM classification. All other computations are performed in pure software.

This section presents the overall architecture of the proposed ASIP and the data transferring between the ASIP engine and the memory. Figure 5.9 shows a

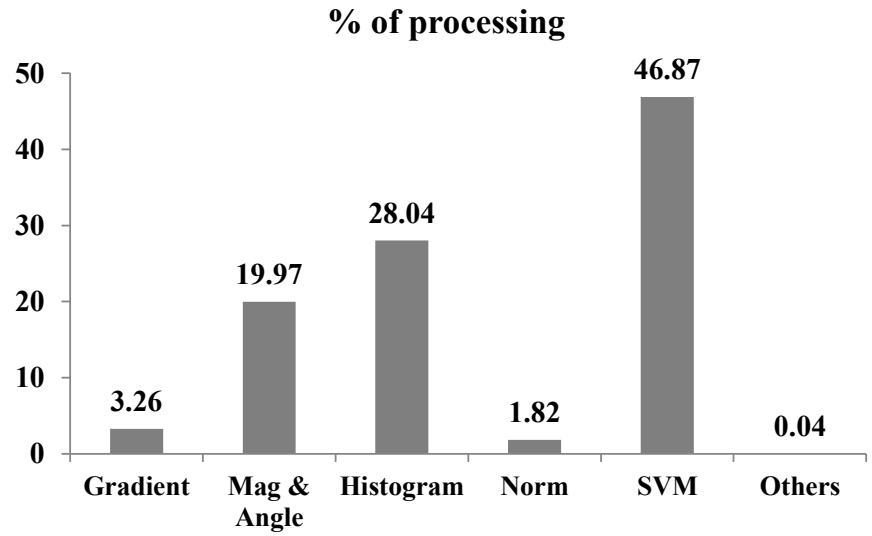


Figure 5.8 HOG algorithm profiling result. The top-5 computationally intensive functions take 99.96% of the total cycle counts.

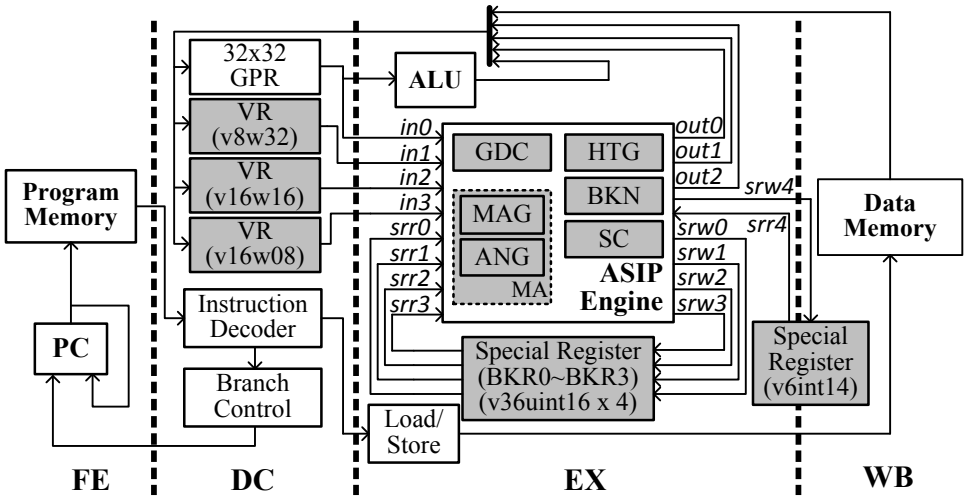


Figure 5.9 Overall architecture of the proposed ASIP. The ASIP engine consists of five special functional units: Gradient Computation (GDC), Magnitude and Angle Calculation (MA), Histogram Generation (HTG), Block Normalization (BKN), and SVM Classification (SC) unit.

block diagram of the proposed ASIP. The main custom hardware components are vector registers (VR) for Single Instruction Multiple Data (SIMD) operation, special registers for internal data communication and an ASIP engine. The ASIP engine consists of five special functional units: gradient computation (GDC) unit, magnitude, and angle calculation (MA) unit, histogram generation

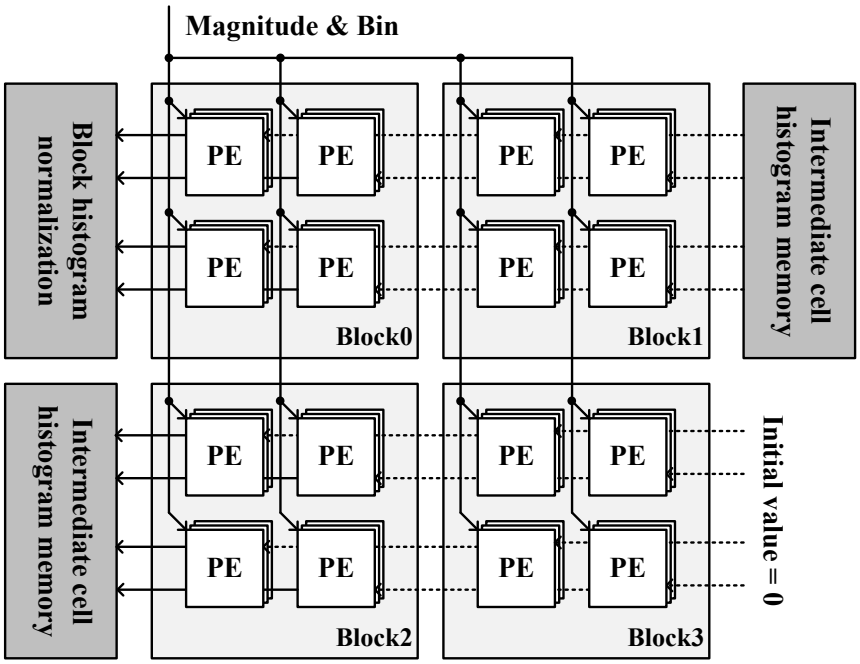


Figure 5.10 Block diagram of cell histogram generation. It is a 4-way architecture. Each PE is an 8-way SIMD architecture.

(HTG) unit, block normalization (BKN) unit and SVM classification (SC) unit. The special functional units are controlled by the instruction decoder and a program, sharing the same pipeline with the baseline processor.

To increase the parallelism, a mass of SIMD operations are adopted in the special functional units. These multiple data are loaded through interface $in1$ (8x32 bit), $in2$ (16x16 bit) and $in3$ (16x8 bit), and stored through interface $out1$ (8x32 bit) and $out2$ (16x16 bit). For GDC unit, a single memory access returns a 128-bit data, which includes 16-pixel values. GDC unit utilizes these inputs (through interface $in3$) to compute the gradients and saves the outputs (through interface $out2$) to the memory. The MA unit supports 16-way magnitude and orientation calculation. Gradients are loaded through $in2$. After computation, each pair of magnitude and orientation values are packed into a single 32-bit integer. Sixteen pairs of magnitude and orientation values are stored in memory (through interface $out1$, two cycles).

The HTG unit is an 8-way SIMD architecture. One memory access returns 256-bit (through interface $in1$) value, which consists eight pairs of magnitude

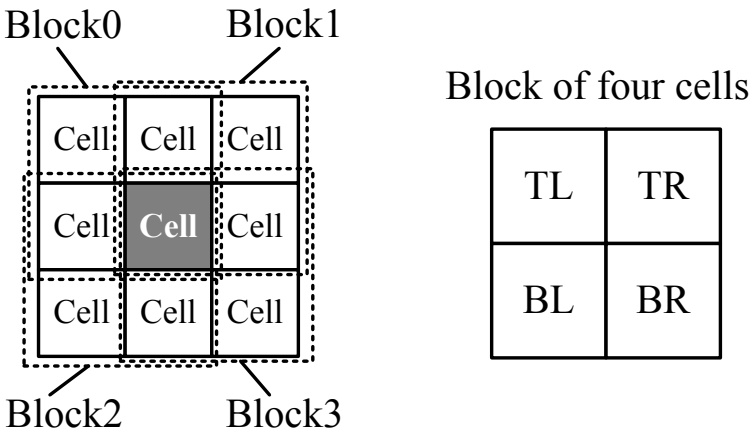


Figure 5.11 Each cell belongs to four blocks maximally. TL, TR, BL and BR represents top-left, top-right, below-left and below-right, respectively.

and orientation values. Each pair of magnitude and orientation generates two votes. These votes are voted to four blocks. Four special registers ($BKR0$, $BKR1$, $BKR2$ and $BKR3$, v36uint16) are customized for block histogram accumulation. The HTG unit utilizes interfaces ($srr0 - srr3$ and $srw0 - srw3$) to access the special registers. For BKN unit, two inputs loads through $in0$ and $srr3$, the block energy are stored in memory (through interface out0, 32 bit) and the normalized block histogram is saved to $BKR3$ (through interface $srw3$). To perform SVM classification, SVM coefficients (through interface $in2$) and the block histogram (through interface $srr3$) are multiplied and accumulated. The special register (v6int14) is used for buffering the intermediate results.

5.3.2 Custom Hardware Component

In this section, we introduce the special functional units to accelerate histogram generation, block histogram normalization, and SVM classification.

Figure 5.10 shows the architecture of histogram generation. For one cell is belong to four blocks maximally (as shown in Figure 5.11), a 4-way architecture is applied for histogram generation to reduce the interaction between different cells. Note that each cell could be in one of the four positions of a block. As shown in Figure 5.11, TL, TR, BL, and BR represents the position of a cell at top-left, top-right, below-left and below-right of a block, respectively. To

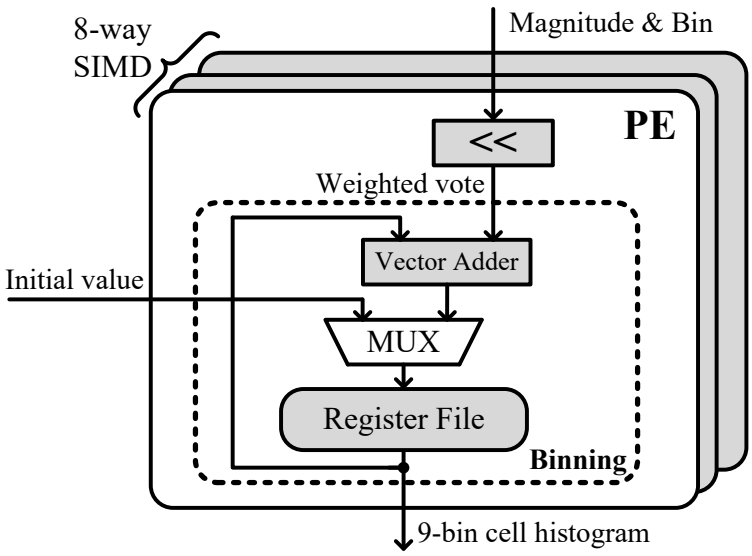


Figure 5.12 Architecture of processing element.

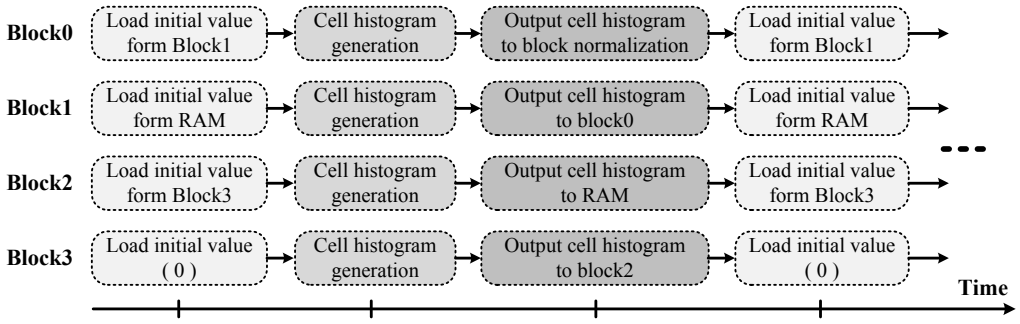


Figure 5.13 Behavior of each block.

generate 4-cell histograms in parallel, each way of block architecture consists of 4 processing elements (PEs). Each PE performs weighted voting and binning to generate the cell histogram. Figure 5.12 demonstrates the architecture of the PE. The PE contains a shifter, a vector adder, and a register file. Through the shifter, the weighted vote is generated from the input magnitude. Then, the weighted vote is accumulated to generate the cell histogram. Furthermore, to increase the parallelism, each PE is an 8-way SIMD architecture, which can process one row of cell histogram generation. In Figure 5.10, the solid arrow indicates the data flow of histogram generation unit, while the dashed arrow represents the initial value loading. Block 0 and Block 2 read the initial value from the neighboring blocks (Block 1 and Block 3). Block 1 reads the

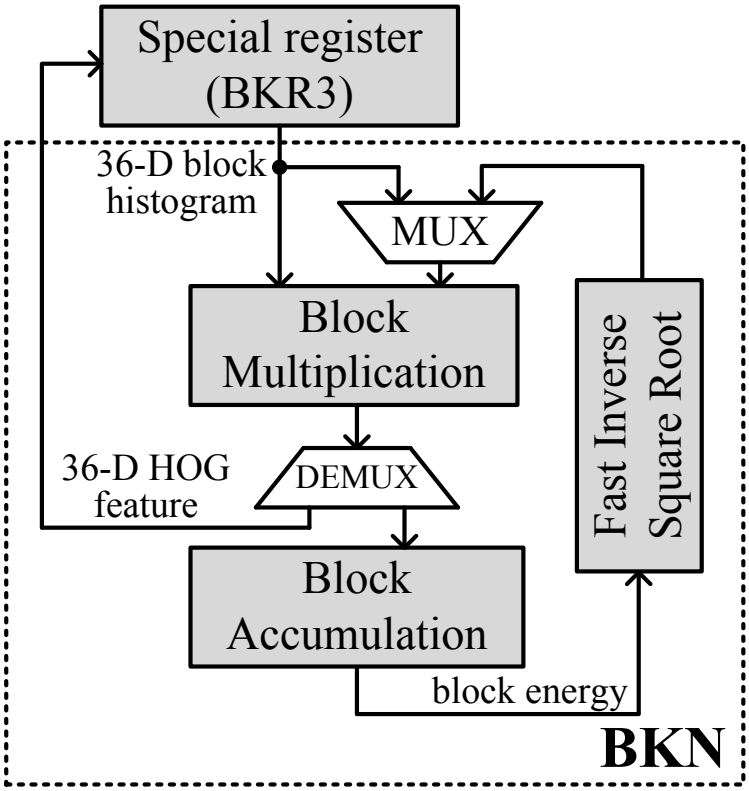


Figure 5.14 Architecture of block normalization.

initial value from memory, while Block 3 loads zero as the initial value. When finished the histogram generation, Block 0 sends the block histogram to the block histogram normalization unit. Figure 5.13 shows the behavior of each block.

Figure 5.14 shows the architecture of block histogram normalization. The BKN unit contains a block multiplication module, a block accumulation module, and a fast inverse square root (FISR) module. The block multiplication module along with the block accumulation module computes the block energy (sum of the squares of block histogram). Then, the block energy is used for normalized coefficient calculation through the FISR module. Newton's method is applied to approximate the inverse square root of the block energy (as discussed in Section 5.2.3.3). When the normalized coefficient is available, the block multiplication module is reused to normalize the block histogram. The normalized 36-D HOG feature is then stored in the special register (BKR3).

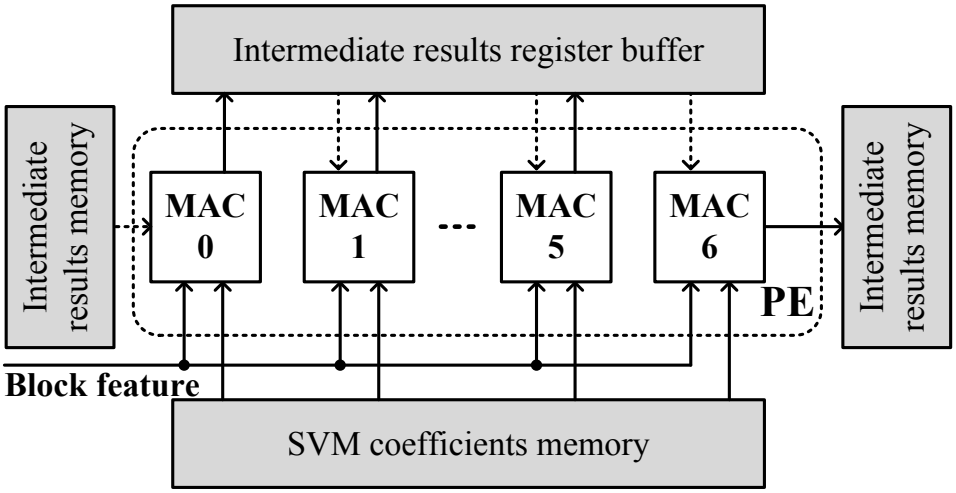


Figure 5.15 Architecture of on-the-fly SVM classification. The classification PE handles seven blocks of MAC operations.

Figure 5.15 presents the architecture of SVM classification. It mainly consists of a classification PE and a register buffer. The classification PE handles seven blocks of MAC operation (since the detection window includes seven blocks in the horizontal direction). Each MAC unit consists of multipliers and adders to compute the sum of the dot product of block feature and SVM coefficients. To exploit the data locality and minimize the data movement between the classification PE and the memory, the intermediate results of MAC operations are stored in the register buffer. The initial value of each MAC unit is from the previous MAC unit or memory. After MAC operation, MAC unit outputs the result to the register buffer or memory.

5.3.3 Instruction Set Extensions

To speed-up the HOG algorithm, the ASIP needs to add the associated acceleration instruction as well as special functional units. The special instructions in the proposed ASIP are summarized in Table 5.3.

Table 5.3 Special instruction in the proposed ASIP (HOG)

Special Instructions	**Function Description**
do	Zero overhead loop
min/max	Minimum/maximum value selection
selz/seln	Conditional move
diffv	Gradient computation
atanv	Orientation calculation
magv	Magnitude computation
sqrtv	Square root calculation
hist	Cell histogram generation
bknorm	Block histogram normalization
i2f/f2i/fmul/fsub	Fast inverse square root calculation
svmpe	SVM calculation

5.4 Experimental Results

In this section, we present the experimental results of ASIP design and make a comparison with other hardware architectures, e.g. in-house TCT processor, commercial embedded processors, and hard-wired designs.

In this work, we implemented the ASIP architecture customizations using nML language [30], which is a high-level architecture description language aiming to shorten the development time of ASIP design. The customizations were added to the base instruction set architecture of the TCT processor [39]. The HDL code for the proposed ASIP was generated using Synopsys ASIP designer [31] based on the nML description.

The ASIP was synthesized with Synopsys Design Compiler [48] using TSMC 90nm low power standard CMOS technology. Table 5.5 summarizes the implementation results. Our ASIP is able to achieve an operating frequency of 200MHz. The silicon area of the proposed ASIP is $1.31mm^2$, and the power consumption is estimated to be $47.8mW$ [49] (include dynamic and leakage power consumption at gate level). The throughput of the ASIP is 42fps on VGA video (estimation based on the processor frequency and the total cycle counts).

Table 5.4 Cycle counts comparison for HOG algorithm on baseline processor and proposed ASIP

Function	**Baseline Processor**	**ASIP**	**Speed-UP**
Gradient computation	9,825,298	474,281	20.7x
Magnitude & Orientation computation	60,210,747	399,280	150.8x
Histogram generation	84,537,585	833,776	101.4x
Block normalization	5,481,731	86,428	63.4x
SVM classification	141,292,456	2,896,986	48.8x
Total cycle count	301,469,649	4,783,417	63x

The cycle count is a meaningful performance comparison metric since it could better reflect the efficiency of hardware to perform certain calculations while being independent of the architecture clock frequency. In Table 5.4, cycle counts comparison for the modified HOG algorithm on the baseline processor and the proposed ASIP is presented. The listed cycle counts are obtained from cycle-accurate simulations performed using the Synopsys ASIP Designer [31] on VGA video. The proposed ASIP outperforms the baseline processor by a factor that ranges between 20.7x and 150.8x for the five functions. For histogram generation, block histogram normalization and SVM classification, the resulting speed-ups are 101.4x, 63.4x and 48.8x, respectively. With the proposed ASIP, 98.4% of the total cycle counts are reduced. Compared to the pure software implementation of HOG algorithm on baseline RISC processor, the total speed-up is 63x. Furthermore, we performed the detection on different standard videos. The cycle counts and throughput are summarized in Table 5.6.

To better evaluate the efficiency of the proposed ASIP, we make a comparison with software implementations of HOG algorithm when mapping on different embedded processors. We choose throughput, area efficiency, and power efficiency as the main comparison metrics. The comparison is shown in Table 5.7. The application executed by ARM968 is compiled by ARM C Compiler and evaluated under ARM RealView Development Suite (RVDS) environment [56]. The

Table 5.5 ASIP implementation results (HOG)

Parameter	**ASIP**
Technology	90nm
Frequency	200MHz
Supply Voltage	1.0V
Area	1.31 mm^2
Power Consumption	47.8mW
Image Resolution	VGA (640x480 pixels)
Throughput	42fps

Table 5.6 Throughput for different video standards

Standard	**Resolution**	**Cycle Count**	**Throughput**
QVGA	320x240 pixels	1,167,793	171fps
VGA	640x480 pixels	4,783,417	42fps
SVGA	800x600 pixels	7,527,679	26.6fps
1080HD	1920x1080 pixels	32,628,137	6.1fps

application executed by TMS320C64+ is compiled by TI Compiler with level 3 optimization and evaluated under TI Code Composer Studio (CCS) environment [57]. Our ASIP takes the least cycle counts to perform the HOG algorithm. This is not surprising because the ASIP integrates dedicated functional units for HOG algorithm. For the main comparison metrics, specifically, the proposed ASIP is 113x better in throughput, 36x better in area efficiency, and 122x better in power efficiency than ARM968 when executing HOG algorithm. Compared with TMS320C64+, our ASIP shows 15x, 750x and 184x better throughput, area efficiency, and power efficiency, respectively. Moreover, compared with the baseline processor, the proposed ASIP has a factor of 18.5x increased in the chip area, but with 63x throughput improvement.

Furthermore, a comparison with previous hardware implementations is shown in Table 5.8. These implementations are all hard-wired designs targeted on FPGA or ASIC, which use HOG feature for object detection. Compared with

Table 5.7 Comparison with software implementations on embedded processors (HOG)

Parameter	**ARM968E-S** [69]	**TMS320C64+** [51]	**TCT Processor**	**Proposed ASIP**
Technology	90nm	90nm	90nm	90nm
Frequency	470MHz	625MHz	200MHz	200MHz
Area	$0.42mm^2$ (w/o cache)	$64mm^2$ [a] (with cache)	$0.071mm^2$ (w/o cache)	$1.31mm^2$ (w/o cache)
Power Consumption	51.7mW	562.5mW	4.1mW	47.8mW
Cycle Counts	1,274,040,121	233,921,254	301,469,649	4,783,417
Frame Rate	0.37fps	2.67fps	0.66fps	42fps
Area Efficiency (fps/KGates)	0.0025	0.00012	0.026	0.090
Power Efficiency (mW/fps)	139.73	210.67	6.21	1.14

[a] Chip area is estimated from the PBGA package.

[54] and [66], our ASIP shows better area efficiency/power efficiency. Since the MAC operations are expensive in SVM classification. In [67], they used AdaBoost rather than SVM as the classifier, thus to reduce the chip area. To process the 1080HD video in real-time, [68] used a dual-core architecture for parallel HOG feature extraction. Compared with [67] and [68], our area efficiency/power efficiency is not an advantage, but our uniprocessor based ASIP still shows a comparable performance, especially, when the technology is taken into account. Moreover, the proposed processor maintains the flexibility of ASIPs for integration of evolving algorithms, while hard-wired designs are hardly adapted for integrating new features in the algorithm.

Table 5.8 Performance comparison with hard-wired designs (HOG)

Parameter	Hiromoto [54]	Negi [67]	Mizuno [66]	Takagi [68]	This Work
Technology	65nm	65nm	60nm	65nm	90nm
Frequency	167MHz	44.85MHz	40MHz	42.9MHz	200MHz
Gate Count (2NAND)	330K [a]	200K [a]	460K [a]	502K	464K
Power Consumption	N/A	N/A	196.9mW	99.5mW	47.8mW
Image Resolution	320x240	640x480	800x600	1920x1080	640x480
Frame Rate	38fps	62.5fps	72fps	30fps	42fps
Area Efficiency (fps/KGates)	0.0288 [b]	0.3125	0.2446 [b]	0.4034 [b]	0.090
Power Efficiency (mW/fps)	N/A	N/A	1.75 [b]	0.491 [b]	1.14

[a] Gate count is estimated from the number of LUTs and register bits.
[b] For comparison, the resolution is scaled to VGA.

5.5 Related Work

Most of the implementations of HOG-based object detection are on GPU platforms [70–72]. In [72], the HOG feature extraction on GPU achieves 100 fps for VGA video (640x480 pixels). However, GPU implementations consume significant power (e.g., Nvidia GeForce GTX 780 consumes 250W [73]), which cannot meet the low power requirement of embedded applications.

To balance the throughput and power consumption trade-offs, FPGA-based implementations have been proposed for real-time applications [74, 75, 54, 67, 66]. A hybrid FPGA-GPU architecture was proposed in [74]. HOG feature extraction was implemented on a FPGA and a CPU, while SVM classification was performed on a GPU. It achieved 10fps for SVGA (800x600 pixels) video. In [75], the effects of reduced bit-width on the performance of HOG algorithm was evaluated using

full-image evaluation methodology. Also, a hybrid FPGA-CPU architecture was proposed, and the computationally intensive parts of HOG algorithm were implemented on FPGA.

The entire HOG-based detector implemented on FPGA was proposed in [67]. A stream processing approach for the histogram generation was proposed to improve the throughput. AdaBoost classifier instead of SVM classifier was adopted to shorter the classification time. Similar to [67], a stream processing approach for histogram generation was used in [66]. Also, they presented a simultaneous SVM calculation architecture for fast classification. The object detector can achieve 72fps for SVGA video.

To minimize the power consumption, [68] and [76] chose ASICs as the hardware architecture. In [68], they used a dual-core architecture for parallel feature extraction. It can achieve 30fps for 1080HD video (1920x1080 pixels) with 99.5mW power consumption. In [76], they used reduced bit-width (4 bit) for SVM weights to reduce the cost of multiply and accumulate (MAC) operations. It can achieve 60fps for 1080HD video with 45.3mW power consumption. Although these implementations can provide powerful computing capability with low power consumption for object detection, they cannot offer the required flexibility, especially when new characteristics are needed to add in the application.

5.6 Summary

In this chapter, we proposed an ASIP-based processor for embedded vision with the Histogram of Oriented Gradients (HOG) algorithm. This algorithm generates a HOG descriptor from detection window with grids of overlapping blocks and combines with a Support Vector Machine (SVM) for classification to determine the existence of an object. The main contributions of this work are a) hardware algorithm co-design: Cell-based scanning approach with on-the-fly SVM calculation architecture are employed to reduce memory bandwidth requirements and redundant computations. Simplified HOG algorithm is introduced to minimize the hardware costs and reduce the mathematical complexity without

losing reliability. b) hardware-software co-design: With multiple parallelization techniques, the proposed ASIP exhibits an advantage in terms of both chip area efficiency and power efficiency when compared to embedded processors, and shows a competitive computational performance when compared to hard-wired designs. The proposed processor maintains the flexibility of ASIPs for a fast integration of evolving algorithms with an effective and a low-cost development flow.

Chapter 6

Conclusion and Future Works

6.1 Conclusion

The Application-Specific Instruction-set Processor (ASIP) is gaining popularity in the development of high-performance applications, with the ability to balance the performance-power-flexibility trade-offs. This thesis aims to study an efficient ASIP design methodology and to provide a complete framework to support the design of ASIP-based processors. The contribution of this dissertation is summarized as follows.

1. A complete framework is proposed to support the ASIP-based embedded vision processor design.

The proposed design framework covers from the high-level ASIP programming model to the low-level micro-architecture. It is aimed to facilitate the application-to-architecture mapping at the architect level. To expedite the design cycle, our ASIP design methodology is integrated with high-level abstraction Architecture Description Language (ADL) and commercial design tool. The framework also provides an interface with open source computer vision library, for easily maintain the evolution of vision algorithms.

2. Some design thinking are provided as references for vision processor architects.

These thinking include algorithms consideration, applications understanding, architecture consideration, and design reuse consideration.

3. A thorough analysis on representative object detection techniques and the proposed hardware-friendly object detection algorithms.

We conduct a comprehensive study on several representative object detection techniques, especially the ones using hand-crafted features with machine learning classifiers, e.g. Haar-like features with AdaBoost classifier and HOG features with SVM classifier. The study aims to identify the computational intensive tasks, extract the key computationally tasks, and exploit the data locality properties within the algorithm. The thorough analysis establishes a solid foundation for the design of object detection processors.

We careful reschedule the dataflow within the AdaBoost-based learning algorithm and HOG-based algorithm. The new dataflow is aimed at reducing the memory bandwidth requirements and redundant computations. Also, the simplified AdaBoost algorithm and HOG algorithm are introduced to minimize the hardware costs and reduce the mathematical complexity. The modified algorithms are evaluated on several popular faces and human datasets. The evaluation results show that the modified algorithms still maintain the high accuracy as well as the original algorithms.

4. As demonstrations, we show the effectiveness of the proposed framework using two real-life applications, Haar-like features with AdaBoost classifier and Histogram of Oriented Gradients (HOG) features with Support Vector Machine (SVM) classifier for object detection.

Two synthesized ASIP-based processors are designed and implemented for AdaBoost algorithm and HOG algorithm. To meet the real-time requirements, we propose several special functional units to accelerate the computationally intensive tasks in these object detection algorithm. The special functional units

include integral image calculation array, pipeline Haar-like feature computation, and parallel histogram generation etc. To achieve low-power consumption, an efficient classifier data storage mechanism for AdaBoost and on-the-fly SVM classification are proposed to minimize the data movement. With proper application-to-architecture mapping, the proposed ASIPs achieve high throughput in a small area, power, and energy footprint.

A full comparison between the proposed ASIP and the conventional embedded processors and hard-wired designs is carried out in terms of performance, area, and power consumption. The proposed ASIP exhibits an advantage in terms of both chip area-efficiency and power-efficiency when compared to embedded processors, and shows a competitive computational performance when compared to hard-wired designs. To the best of our knowledge, there is no similar work for AdaBoost algorithm and HOG algorithm using ASIP design methodology so far.

6.2 Future Works

In this research, we propose a complete framework for ASIP-based vision processor design and demonstrate the efficiency of the proposed framework using two real-life applications, Haar-like features with AdaBoost classifier and histogram of oriented gradients (HOG) features with Support Vector Machine (SVM) classifier for object detection. For the future work, we will focus on extending the ASIP design methodology to more widely used applications and integrating the ASIP design methodology with the multiprocessor system-on-chip (MPSoC) design.

First, we notice that many vision algorithms, such as the Histogram of Oriented Gradients (HOG) features, Scale-Invariant Feature Transform (SIFT), and Speeded Up Robust Feature (SURF), contains abundant common or similar operations. These common features can be described in high-level functional blocks or low-level computational operations. A single ASIP is able to support these common features through a reconfigurable datapath.

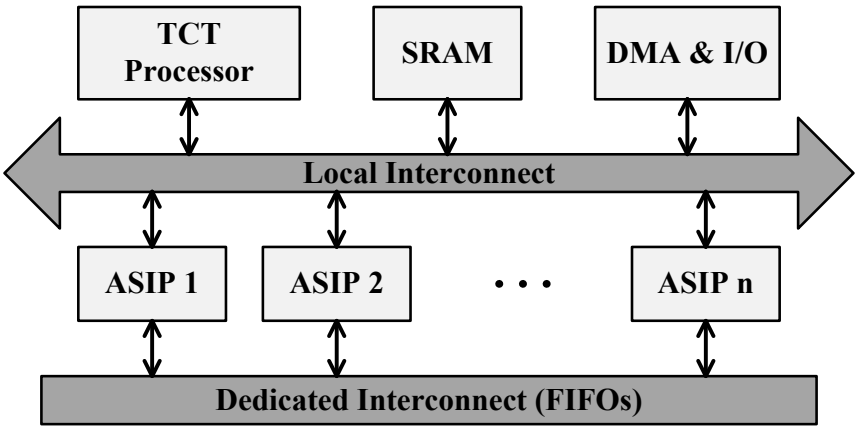


Figure 6.1 Multi-ASIP based embedded vision processor.

Second, the increasing video definition (e.g., Ultra-High-Definition, or 4K) and the evolving state-of-the-art computer vision algorithms (e.g., RCNN [77], Fast RCNN [78], Faster RCNN [79], YOLO [80], and SSD [81]) are bring in a heavy computation and mass of data movement for mobile and embedded platforms. ASIP-based MPSoC platforms are an attractive design solution to provide powerful computing capability and energy efficiency with relative low design cost for the future vision applications. Figure 6.1 shows a multi-ASIP based vision processor. We will focus on the improvement of ASIP design framework to support the ASIP-based MPSoC design.

Bibliography

[1] Rob van der Meulen. For a Trillion Sensor Road Map.

[2] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015–2020, Feb. 2016.

[3] Andrew Christopher Mihal. *Deploying concurrent applications on heterogeneous multiprocessors*. PhD thesis, University of California, Berkeley, 2006.

[4] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[5] Glenn Hinton, Dave Sager, Mike Upton, Darrell Boggs, et al. The microarchitecture of the Pentium® 4 processor. In *Intel Technology Journal*, 2001.

[6] Texas Instruments. TMS320C64x/C64x+ DSP CPU and instruction set reference guide. 2005.

[7] Juan Fernando Eusse, Rainer Leupers, Gerd Ascheid, Patrick Sudowe, Bastian Leibe, and Tamon Sadasue. A flexible ASIP architecture for connected components labeling in embedded vision applications. In *2014 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1–6, 2014.

[8] INRIA Person Dataset. http://pascal.inrialpes.fr/data/human/, 2015.

[9] Cisco Global Cloud Index: Forecast and Methodology, 2015–2020, June 2016.

[10] Bernard Marr. Big Data: 20 Mind-Boggling Facts Everyone Must Read.

[11] Josh Woodhouse. Big, big, big data: higher and higher resolution video surveillance.

[12] Rob van der Meulen. Gartner Says 6.4 Billion Connected Things Will Be in Use in 2016, Up 30 Percent From 2015.

[13] K. Keutzer, S. Malik, and A. R. Newton. From ASIC to ASIP: the next design discontinuity. In *2002 IEEE International Conference on Computer Design (ICCD)*, pages 84–90, Freiburg, Germany, Sept. 2002.

[14] Paolo Ienne and Rainer Leupers. *Customizable Embedded Processors: Design Technologies and Applications*. Morgan Kaufmann Publishers Inc., San Francisco, USA, 2007.

[15] Nico Mentzer, Guillermo Payá-Vayá, and Holger Blume. Analyzing the performance-hardware trade-off of an ASIP-based SIFT feature extraction. *Journal of Signal Processing Systems*, 85(1):83–99, 2016.

[16] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.

[17] Scott Krig. *Computer Vision Metrics: Survey, Taxonomy, and Analysis*. Apress, 2014.

[18] Steve Heath. *Embedded Systems Design*. Newnes, 2002.

[19] Willie D Jones. Building safer cars. *IEEE Spectrum*, 39(1):82–85, 2002.

[20] David Geronimo, Antonio M Lopez, Angel D Sappa, and Thorsten Graf. Survey of pedestrian detection for advanced driver assistance systems. *IEEE transactions on pattern analysis and machine intelligence*, 32(7):1239–1258, 2010.

[21] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[22] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[23] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[24] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244, 1988.

[25] David G Lowe. Object recognition from local scale-invariant features. In *1999 IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1150–1157, 1999.

[26] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: speeded up robust features. *2006 European Conference on Computer Vision (ECCV)*, pages 404–417, 2006.

[27] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *2011 IEEE International Conference on Computer Vision (ICCV)*, pages 2564–2571, 2011.

[28] Atsushi Mizuno, Kazuyoshi Kohno, Ryuichiro Ohyama, Takahiro Tokuyoshi, Hironori Uetani, Hans Eichel, Takashi Miyamori, Nobu Matsumoto, and Masataka Matsui. Design methodology and system for a configurable media embedded processor extensible to VLIW architecture. In *2002 IEEE International Conference on Computer Design (ICCD)*, pages 2–7, 2002.

[29] Andreas Hoffmann, Oliver Schliebusch, Achim Nohl, Gunnar Braun, Oliver Wahlen, and Heinrich Meyr. A methodology for the design of application specific instruction set processors (ASIP) using the machine description language LISA. In *2011 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 625–630, 2001.

[30] Andreas Fauth, Johan Van Praet, and Markus Freericks. Describing instruction set processors using nML. In *Proc. 1995 European Design and Test Conference*, pages 503–507, Paris, France, Mar. 1995.

[31] ASIP Designer, Synopsys Inc. http://www.synopsys.com/, 2015.

[32] Open Computer Vision Library. http://opencv.org/, 2014.

[33] David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

[34] P. Viola, M. J. Jones, and D. Snow. Detecting pedestrians using patterns of motion and appearance. In *2003 IEEE International Conference on Computer Vision (ICCV)*, pages 734–741, Nice, France, Oct. 2003.

[35] Cristiano Premebida, Gonçalo Monteiro, Urbano Nunes, and Paulo Peixoto. A lidar and vision-based approach for pedestrian and vehicle detection and tracking. In *2007 IEEE Intelligent Transportation Systems Conference*, pages 1044–1049, Bellevue, USA, Sept. 2007.

[36] R. Timofte, K. Zimmermann, and L. V. Gool. Multi-view traffic sign detection, recognition, and 3d localisation. In *2009 Workshop on Applications of Computer Vision (WACV)*, pages 1–8, Snowbird, USA, Dec. 2009.

[37] Oscar Déniz, Gloria Bueno, Jesús Salido, and Fernando De la Torre. Face recognition using histograms of oriented gradients. *Pattern Recognition Letters*, 32(12):1598–1603, 2011.

[38] Yuteng Zhou, Zhilu Chen, and Xinming Huang. A pipeline architecture for traffic sign classification on an FPGA. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 950–953, Lisbon, Portugal, May 2015.

[39] Mohammad Zalfany Urfianto, Tsuyoshi Isshiki, Arif Ullah Khan, Dongju Li, and Hiroaki Kunieda. A Multiprocessor SoC Architecture with Efficient Communication Infrastructure and Advanced Compiler Support for Easy Application Development. *IEICE Trans. Fundamentals*, E91-A(4):1185–1196, Apr. 2008.

[40] Hao Xiao, Tsuyoshi Isshiki, Arif Ullah Khan, Dongju Li, Hiroaki Kunieda, Yuko Nakase, and Sadahiro Kimura. A low-cost and energy-efficient multiprocessor system-on-chip for UWB MAC layer. *IEICE Trans. Inf. & Syst.*, E95-D(8):2027–2038, Aug. 2012.

[41] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. Understanding sources of inefficiency in general-purpose chips. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 37–47, 2010.

[42] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 511–518, Kauai, USA, Dec. 2001.

[43] Francois Fleuret and Donald Geman. Coarse-to-Fine Face Detection. *International Journal of computer vision*, 41(1-2):85–107, 2001.

[44] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition).* Prentice-Hall, Inc., Upper Saddle River, USA, 2006.

[45] H. A. Rowley, S. Baluja, and T. Kanade. Neural Network-Based Face Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, Jan. 1998.

[46] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann. The binormal assumption on precision-recall curves. In *20th International Conference on Pattern Recognition (ICPR)*, pages 4263–4266, Istanbul, Turkey, Aug. 2010.

[47] John L Hennessy and David A Patterson. *Computer Architecture: A Quantitative Approach*. Elsevier, 2011.

[48] Design Compiler, Synopsys Inc. http://www.synopsys.com/, 2016.

[49] Power Compiler, Synopsys Inc. http://www.synopsys.com/, 2016.

[50] ARM946E-S, ARM Inc. http://www.arm.com/, 2016.

[51] TMS320C64+, Texas Instruments Inc. http://www.ti.com/, 2016.

[52] Yuya Hanai, Yuichi Hori, Jun Nishimura, and Tadahiro Kuroda. A versatile recognition processor employing Haar-like feature and cascaded classifier. In *2009 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 148–149, San Francisco, USA, Feb. 2009.

[53] Y. M. Tsai, T. J. Yang, C. C. Tsai, K. Y. Huang, and L. G. Chen. A 69mW 140-meter/60fps and 60-meter/300fps intelligent vision SoC for versatile automotive applications. In *2012 Symposium on VLSI Circuits (VLSIC)*, pages 152–153, Honolulu, USA, June 2012.

[54] Masayuki Hiromoto, Hiroki Sugano, and Ryusuke Miyamoto. Partially parallel architecture for adaboost-based detection with haar-like features. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(1):41–52, Jan. 2009.

[55] C. Kyrkou and T. Theocharides. A Flexible Parallel Hardware Architecture for AdaBoost-Based Real-Time Object Detection. *IEEE Transactions on Very Large Scale Integration Systems*, 19(6):1034–1047, June 2011.

[56] RealView Development Suite, ARM Inc. http://www.arm.com/, 2016.

[57] Code Composer Studio, Texas Instruments Inc. http://www.ti.com/, 2016.

[58] Junguk Cho, Shahnam Mirzaei, Jason Oberg, and Ryan Kastner. FPGA-Based Face Detection System Using Haar Classifiers. In *Proc. ACM/SIGDA international symposium on Field Programmable Gate Arrays (FPGA 2009)*, pages 103–112, Monterey, USA, Feb. 2009.

[59] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893, San Diego, USA, June 2005.

[60] James Ulery. Computing integer square roots. www.azillionmonkeys.com/qed/ulerysqroot.pdf, 2016.

[61] David A Patterson and John L Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Elsevier, 2009.

[62] David H. Eberly. *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics*. Morgan Kaufmann, 2006.

[63] Chris Lomont. Fast inverse square root. *Tech-315 nical Report*, page 32, 2003.

[64] MIT pedestrian Dataset. http://cbcl.mit.edu/software-datasets/PedestrianData.html, 2015.

[65] David Martin Powers. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

[66] K. Mizuno, Y. Terachi, K. Takagi, S. Izumi, H. Kawaguchi, and M. Yoshimoto. Architectural study of HOG feature extraction processor for real-time object detection. In *2012 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 197–202, Quebec City, Canada, Oct. 2012.

[67] K. Negi, K. Dohi, Y. Shibata, and K. Oguri. Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm. In *2011 International Conference on Field-Programmable Technology (FPT)*, pages 1–8, New Delhi, India, Dec. 2011.

[68] Kosuke Mizuno, Kenta Takagi, Yosuke Terachi, Shintaro Izumi, Hiroshi Kawaguchi, and Masahiko Yoshimoto. A sub-100mW dual-core HOG accelerator VLSI for parallel feature extraction processing for HDTV resolution video. *IEICE Trans. Electronics*, 96(4):433–443, 2013.

[69] ARM968E-S, ARM Inc. http://www.arm.com/, 2016.

[70] T. Machida and T. Naito. GPU & CPU cooperative accelerated pedestrian and vehicle detection. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 506–513, Barcelona, Spain, Nov. 2011.

[71] Li Zhang and R. Nevatia. Efficient scan-window based object detection using GPGPU. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops)*, pages 1–7, Anchorage, USA, June 2008.

[72] Rodrigo Benenson, Markus Mathias, Radu Timofte, and Luc Van Gool. Pedestrian detection at 100 frames per second. In *2012 IEEE Conference*

on Computer Vision and Pattern Recognition (CVPR), pages 2903–2910, Providence, USA, June 2012.

[73] GeForce GTX 780, Nvidia Inc. http://www.nvidia.com/, 2016.

[74] Sebastian Bauer, Sebastian Köhler, Konrad Doll, and Ulrich Brunsmann. FPGA-GPU architecture for kernel SVM pedestrian detection. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops)*, pages 61–68, San Francisco, USA, June 2010.

[75] Xiaoyin Ma, Walid A Najjar, and Amit K Roy-Chowdhury. Evaluation and acceleration of high-throughput fixed-point object detection on FPGAs. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(6):1051–1062, 2015.

[76] Amr Suleiman and Vivienne Sze. An energy-efficient hardware implementation of HOG-based object detection at 1080HD 60 fps with multi-scale support. *Journal of Signal Processing Systems*, 84(3):325–337, 2016.

[77] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE conference on computer vision and pattern recognition (CVPR)*, pages 580–587, 2014.

[78] Ross Girshick. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[79] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems (NIPS)*, pages 91–99, 2015.

[80] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[81] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single shot multibox detector. In *2016 European Conference on Computer Vision (ECCV)*, pages 21–37, 2016.