2018

SYSTEM-ON-A-CHIP (SOC)-BASED HARDWARE ACCELERATION FOR HUMAN ACTION RECOGNITION WITH CORE COMPONENTS

Amin Safaei
University of Windsor

SYSTEM-ON-A-CHIP (SOC)-BASED HARDWARE ACCELERATION FOR HUMAN ACTION RECOGNITION WITH CORE COMPONENTS

by

Amin Safaei

A Dissertation
Submitted to the Faculty of Graduate Studies
through the Department of Electrical and Computer Engineering
in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy
at the University of Windsor

Windsor, Ontario, Canada

2018

System-on-a-chip (SoC)-based hardware Acceleration for Human Action Recognition with Core Components

by

Amin Safaei

APPROVED BY:

W. Shen, External Examiner
University of Western Ontario

D. Wu
School of Computer Science

M. Ahmadi
Department of Electrical and Computer Engineering

E. Abdel-Raheem
Department of Electrical and Computer Engineering

Dr. Q. M. J. Wu, Advisor
Department of Electrical and Computer Engineering

25 April, 2018

Declaration of Co-Authorship / Previous Publication

I. Co-Authorship Declaration

I hereby declare that this dissertation incorporates material that is result of joint research, as follows: This dissertation incorporates the outcomes of joint research undertaken in collaboration with Dr. Yimin Yang and Mr. Thangarajah Akilan under the supervision of professor Jonathan Wu.

The collaboration is covered through Chapters 3, 4, 5 and 6 of the dissertation. In all cases, the key ideas, primary contributions, experimental designs, data analysis and interpretation, were performed by the author, and the contribution of the collaborator was primarily through the provision of valuable suggestions for the representation of ideas, for the analysis of the results for the experiments carried out and editorial activities throughout the process of dissemination of the work.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my dissertation, and have obtained written permission from the only co-author to include the above material in my dissertation.

I certify that, with the above qualification, this dissertation, and the research to which it refers, is the product of my own work.

II. Declaration of Previous Publication

This thesis includes 4 original papers that have been previously published/submitted in peer reviewed journals and conferences, as follows:

thesis Chapter	Publication title/full citation	Publication status
Chapter 3	A. Safaei and Q. M. J. Wu, Y. Yang, "System-on-a-chip (SoC)-based hardware acceleration for foreground and background identification," Journal of the Franklin Institute, Volume 355, Issue 4, 2018, Pages 1888-1912 ©2017 The Franklin Institute.Reprinted, with permission,	published
Chapter 4	A. Safaei, Q. M. J. Wu and T. Akilan, "System-on-chip-based hardware acceleration for human detection in 2D/3D scenes," 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Banff, AB, 2017, pp. 1041-1045. ©2018 IEEE.Reprinted, with permission	published
Chapter 5	A. Safaei and Q. M. J. Wu, "System-level design for human action recognition in 3D scenes," 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Banff, AB, 2017, pp. 548-553. ©2018 IEEE.Reprinted, with permission	published
Chapter 6	A. Safaei and Q. M. J. Wu, Y. Yang, T. Akilan, "System-on-a-Chip (SoC)-based Hardware Acceleration for an Online Sequential Extreme Learning Machine (OS-ELM)," IEEE transactions on computer-aided design of integrated circuits and systems.	Submitted

I certify that I have obtained a written permission from the copyright owners to include the above published materials in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University

of Windsor.

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owners to include such materials in my thesis.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

Today, the implementation of machine vision algorithms on embedded platforms or in portable systems is growing rapidly due to the demand for machine vision in daily human life. Among the applications of machine vision, human action and activity recognition has become an active research area, and market demand for providing integrated smart security systems is growing rapidly. Among the available approaches, embedded vision is in the top tier; however, current embedded platforms may not be able to fully exploit the potential performance of machine vision algorithms, especially in terms of low power consumption. Complex algorithms can impose immense computation and communication demands, especially action recognition algorithms, which require various stages of preprocessing, processing and machine learning blocks that need to operate concurrently. The market demands embedded platforms that operate with a power consumption of only a few watts. Attempts have been mad to improve the performance of traditional embedded approaches by adding more powerful processors; this solution may solve the computation problem but increases the power consumption. System-on-a-chip field-programmable gate arrays (SoC-FPGAs) have emerged as a major architecture approach for improving power efficiency while increasing computational performance. In a SoC-FPGA, an embedded processor and an FPGA serving as an accelerator are fabricated in the same die to simultaneously improve power consumption and performance. Still, current SoC-FPGA-based vision implementations either shy away from supporting complex and adaptive vision algorithms or operate at very limited resolutions due to the immense communication and computation demands.

The aim of this research is to develop a SoC-based hardware acceleration workflow for the realization of advanced vision algorithms. Hardware acceleration can improve performance for highly complex mathematical calculations or repeated functions. The

performance of a SoC system can thus be improved by using hardware acceleration method to accelerate the element that incurs the highest performance overhead. The outcome of this research could be used for the implementation of various vision algorithms, such as face recognition, object detection or object tracking, on embedded platforms.

The contributions of SoC-based hardware acceleration for hardware-software co-design platforms include the following: (1) development of frameworks for complex human action recognition in both 2D and 3D; (2) realization of a framework with four main implemented IPs, namely, foreground and background subtraction (foreground probability), human detection, 2D/3D point-of-interest detection and feature extraction, and OS-ELM as a machine learning algorithm for action identification; (3) use of an FPGA-based hardware acceleration method to resolve system bottlenecks and improve system performance; and (4) measurement and analysis of system specifications, such as the acceleration factor, power consumption, and resource utilization.

Experimental results show that the proposed SoC-based hardware acceleration approach provides better performance in terms of the acceleration factor, resource utilization and power consumption among all recent works. In addition, a comparison of the accuracy of the framework that runs on the proposed embedded platform (SoC-FPGA) with the accuracy of other PC-based frameworks shows that the proposed approach outperforms most other approaches.

to my

family for their unconditional love, support and encouragement

Acknowledgments

I would like to thank my supervisor, Dr. Q. M. Jonathan Wu, for all of his support throughout the entire program as well as his patience and belief in my abilities. I thank my committee members, Dr. M. Ahmadi, Dr. E. Abdel-Raheem and Dr. D. Wu, for their valuable suggestions and insight. I thank the external examiner, Dr. W. Shen, for all of his suggestions and help in improving the quality of this dissertation.

I would like to thank my family, who has been with me always with unconditional love and support. My mother has been my support throughout my life in every aspect and put stupendous effort into my upbringing. I express my love and respect for my father, who always taught me to aim higher. I thank God for blessing me with such wonderful parents. I thank my loving younger sister for being there beside my parents and taking care of them and always supporting and encouraging me.

I would like to take this opportunity to express my gratitude to my mentors whom I met during my high school and undergraduate studies for motivating me to carry on the pursuit of knowledge. As learning is a never-ending process that permeates throughout life, I humbly offer my gratitude to Mother Nature for making me a part of it and for helping me to realize it.

I would like to thank our departmental graduate secretary, Ms. Andria Ballo, for all of her help during my studies at the University of Windsor. I thank the Student Health Services at the University of Windsor for helping me to learn a better lifestyle during these strenuous years. My colleagues and friends in the CVSS Laboratory have been helpful, supportive and the source of several good memories I have made during my studies. I thank the city of Windsor for the friendly environment it has offered and for being my home away from home.

Table of Contents

List of Tables

List of Figures

List of Acronyms

APU	Application Processing Unit
ARM	Advanced RISC Machines
AMBA	Advanced Microcontroller Bus Architecture
AMD	Advanced Micro Devices
ASIC	Application Specific Integrated Circuit
BC	Bayesian Classification
BMC	Background Models Challenge
CB	Code Book
CAN	Controller Area Network
CPU	Central Processing Unit
CPLD	Complex Programmable Logic Devices
CMOS	Complementary Metal Oxide Semiconductor
DSP	Digital Signal Processor
DMA	Direct Memory Access
DBS	Deep Brain Simulation
ECV	Embedded Computer Vision
ELM	Extreme Learning Machine
FSM	Finite State Machine

FPGA	Field Gate Programmable Arrays
FIFO	First In First Out
GPU	Graphics Processing Unit
GigE	Gigabit Ethernet
GMM	Gaussian Mixture Model
GPIO	General Purpose Input/Output
GOPS	Billions Of Operations Per Second
HOG	Histogram Of Oriented Gradients
HOF	Histogram Oriented Flow
HPS	Hard Processor System
HBP	Hardware Based Profiling
HDMI	High Definition Multimedia Interface
HODG	Histogram Oriented Depth Gradients
IC	Integrated Circuit
IP	Intellectual Property
I2C	Inter Integrated Circuit
ILP	Instruction Level Parallelism
ISP	Image Signal Processing
ITRS	International Technology Roadmap For Semiconductors

LBP	Local Binary Pattern
LCD	Liquid Crystal Display
MOG	Mixture Of Gaussian
MIMD	Multiple Instruction Multiple Data
MS-SSIM	Multiscale Structural Similarity Index
NN	Neural Network
MHI	Motion History Image
MMHI	Motion Modified History Image
NP	Noise Perturbation
NZMS	Non Zero Motion Block Similarity
OpenCV	Open Computer Vision
OPENCL	Open Computing Language
OS-ELM	Online Sequential Extreme Learning Machine
PC	Personal Computer
PS	Processing System
PL	Programmable Logic
RBF	Radial Basis Function
ROI	Region Of Interest
RPU	Real Time Processing Unit
RTL	Register Transfer Level

RVFL	Random Vector Functional Link
SD	Secure Digital
SDK	Software Development Kit
SBP	Software Based Profiling
SOC	System On Chip
SVM	Support Vector Machine
SPI	Serial Peripheral Interface
SATA	Serial AT Attachment
SLFN	Single Hidden Layer Feed Forward Neural Network
SIMD	Single Instruction Multiple Data
SIFT	Scale Invariant Feature Transform
SOC-FPGA	System On Chip Field Programmable Gate Arrays
USB	Universal Serial Bus
UART	Universal Asynchronous Receiver-Transmitter
VM	VuMeter
VCA	Video Content Analysis
VDMA	Video Direct Memory Access
VLSI	Very Large Scale Integration

Chapter 1

Introduction

1.1 Embedded Computer Vision

An embedded system is a computer system that is optimized and designed to operate in scenarios in which space is limited and the power consumption is required to be low. Such devices include mobile phones, digital cameras, dental scanner and smart watches. In addition, the embedded approach allows a reduction in cost compared with traditional PC-based systems. For example, the budget required to set up a PC-based system for a computer vision application is approximately $1700. The peripherals required for this system consist of a camera, a lens, and a cable. Meanwhile, for an embedded system with the same throughput, the cost would be $300 [1].

There are two main approaches available for implementing computer vision algorithms. The first is the traditional approach, which consists of the integration of a typical camera through a GigE or USB interface that is connected to a PC and a software development kit (SDK) that is provided by the manufacturer for accessing the image or video stream (Figure 1.1). The second approach is to develop an embedded system, in which the entire system is integrated on a single board (Figure 1.2). Some well-known single-board embedded vision platforms include the Raspberry Pi (Odroid), Toradex and Advantech systems (Figure 1.3).

Among the applications of embedded systems, embedded computer vision is becoming an active and fast-growing area. Embedded computer vision is defined as the implementation of visual understanding techniques in an embedded system for gaining a better understanding of visual scenes. It has a wide variety of applications,

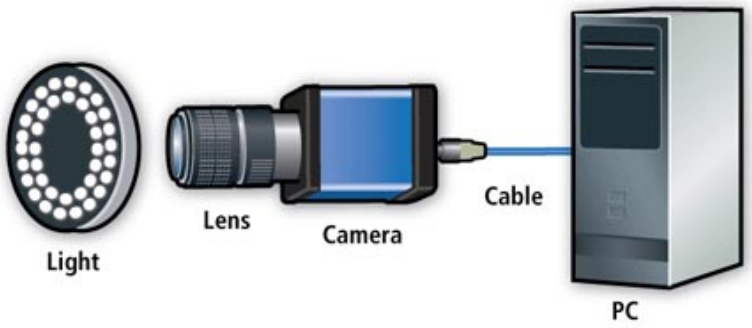


Figure 1.1 – Traditional PC based vision system [2].

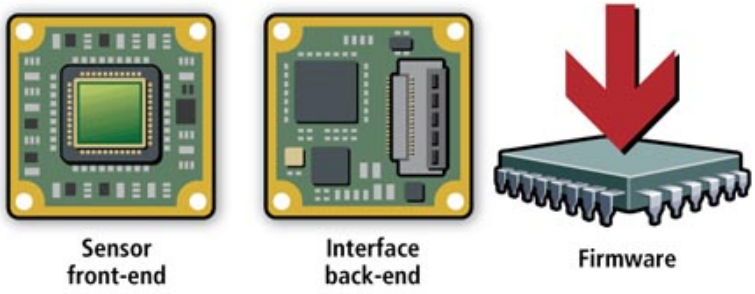


Figure 1.2 – Camera module for embedded vision [2].

such as autonomous vehicles, smart cameras [3] [4], industrial vision cameras, sensor networks and security systems (surveillance systems).

Although significant research has been done on algorithms for computer and machine vision, the implementation of current algorithms in hardware is still in a fairly early stage. It should also be considered that the complexity of vision algorithms, in addition to the high communication requirements imposed by the need to read the stream and display the results, makes the implementation of vision algorithms notoriously difficult. Moreover, in vision algorithms, the many parallel operations drive the pixel computation complexity well into the range of many billions of operations per second (GOPs), resulting in considerable power consumption. Meanwhile, one of the major limitations of embedded platforms, especially portable systems, is power consumption; generally, most embedded platforms are required to consume less than 1 W. The requirements of high performance, low power consumption and low cost

pose massive challenges in architecting embedded vision platforms.

1.2 Advancements in Modern Computing

Advancements in modern computing have pursued the processing of as much information as possible while maintaining or decreasing the execution time. Meanwhile, the main goal of semiconductor manufacturers is to shrink the size of transistors in order to pack more transistors onto a chip. Reducing the size of transistors not only minimizes the area occupied on a chip but also allows more tasks to be performed in a shorter time. As a result, devices can become more compact due to the smaller sizes of the processing units and the peripheral components. The level of integration is expected to double every two years, according to the International Technology Roadmap for Semiconductors (ITRS) [5] and Moores law. Very-large-scale integration (VLSI) [6] is one common approach that allows manufacturers to create an integrated circuit (IC) by combining billions of transistors into a single chip. This approach is not restricted to the fabrication of central processing units (CPUs); it can also be used to fabricate other electronic components. The traditional approach for increasing the speed of operation is to increase the frequency of operation; however, the performance improvement due to such an increase in frequency may not lead to a truly efficient implementation.

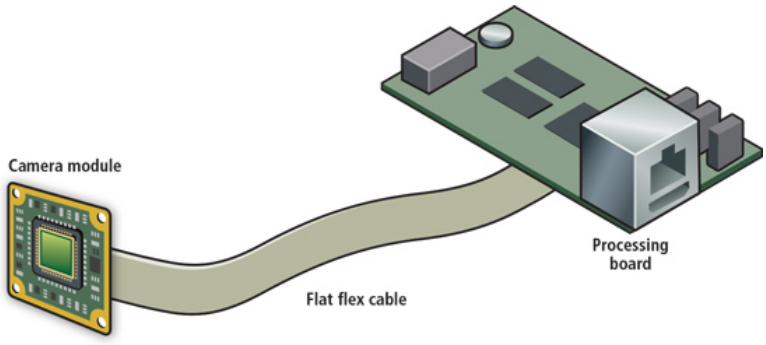


Figure 1.3 – Single-board embedded vision [2].

The typical limitations of design are the "power wall", the "memory wall", and the "instruction-level parallelism wall". An implementation with increased performance incurs an increase in power consumption, causing it to encounter the power wall. The second problem is related to the speed of memory operation. Increasing the clock rate of the processor will not be effective if the memory operates at a slower clock rate. The last limitation is related to the ability to find instructions that can be executed in parallel. When the clock rate is increased, more instructions can be executed; however, since there are always dependencies in each execution, finding instructions in a program that can be run in parallel is a difficult task.

Recent integration techniques allow manufacturers to fabricate many electronic components on a single chip, called a system-on-a-chip (SoC). With this approach, all related parts and components for the execution of specific functions can fabricated into a single chip [7]. The goal of integrating all components on a single chip is to achieve faster operation. This approach can overcome the above mentioned limitations to improve the performance of the system while keeping it running efficiently in terms of power consumption and resource utilization. The SoC approach can be used to fabricate multiple separate processing cores in the same die in order to speed up operation or can be used to fabricate a single core with an accelerator in order to improve processing and reduce power consumption.

1.3 Hardware Acceleration

The main approach for system-level design consists of separate approaches for the development of hardware and software. At the software level, it includes tools such as compilers that are optimized to speed up the execution of functions by generating machine code that is executed with more parallel instructions. Customized libraries such as Boost [8], OpenCL [9] and CUDA [10] generate machine code for parallel instructions. Optimization at the software level is outside the scope of this thesis.

However, at the hardware level, domain-specific approaches provide adequate solutions for high-performance and low-power applications. Domain-specific approaches provide customized hardware architectures that are customized for certain applications. Such a customized accelerator can operate at low power while offering better performance for a specific function. As a result, rather than the function relying on the CPU to execute an excess workload, it instead passes this workload to the accelerator in order to achieve the desired performance and reduce the growing power budget. Various hardware implementations can be used as accelerators. One such implementation is a digital signal processor (DSP) [11] [12], which is an efficient tool for processing streaming data. The architecture is optimized to process high-speed data as it traverses the system. A second type of accelerator is a graphics processing unit (GPU) [13] [14]. GPUs offer massive amounts of parallel computing potential, which can be used to accelerate the portions of a computer vision pipeline in which parallel processing is performed on pixel data. However, GPUs face challenges with regard to power consumption and consequently cannot be considered for use in all applications. Another approach is to develop a specialized processing chip called an application-specific integrated circuit (ASIC) [15] to execute specific functions to expedite processing. However, the cost of this method is very high. An alternative approach is to use a field-programmable gate array (FPGA) [16]. Instead of incurring the high cost and long lead time of a custom ASIC to accelerate a system, a designer can use an FPGA to implement a reprogrammable solution for hardware acceleration. With millions of programmable gates, hundreds of I/O pins, and compute performance in the trillions of multiply-accumulates/sec (tera-MACs), high-end FPGAs offer the greatest potential for high performance in a vision system. Unlike a CPU, which must time-slice or multi-thread tasks as they compete for compute resources, an FPGA has the advantage of being able to simultaneously accelerate multiple portions of a function pipeline.

1.4 SoC-FPGA Hardware Acceleration

A traditional hardware acceleration system is implemented with two chips: an accelerator and a host processor. As discussed above, depending on the type of application, the accelerator could be an FPGA, a GPU or a DSP that functions as a co-processor with the host processor. In this architecture, both streaming traffic and algorithm-intrinsic traffic occupy the interconnect and system memory. The host processor transfers the input stream to the shared memory; when the co-processor is triggered, the contents of the shared memory are processed by the co-processor, and the output of the algorithm is returned to the shared memory. In the final stage, the host processor reads the content of the shared memory and transfers it to the output port. All transactions are controlled and monitored by the host processor, which is inefficient and leads to high overhead. A more efficient approach is to use a SoC-FPGA [17] [18] [19] [20], which contains a processor and an FPGA fabricated on the same chip. Among the available SoC-FPGAs provided by different vendors, Xilinx SoC-FPGAs (Zynq-7000 SoC, Zynq UltraScale+ MPSoC) [21] and Intel (Altera) SoC-FPGAs (Stratix 10 SoC, Arria 10 SoC, Arria V SoC, Cyclone V SoC) [18] are popular because of their good performance and ease of utilization. These SoC-FPGAs include an ARM-based hard processor system (HPS), consisting of a processor, peripherals, and memory, that interfaces with the FPGA fabric via a high-bandwidth interconnect backbone. It combines the performance and power savings of hard intellectual property (IP) with the flexibility of programmable logic.

1.5 Motivation

As discussed above, because of the complexity of vision algorithms as well as the high communication requirements for reading the stream and displaying the results, the implementation of vision algorithms is difficult. We observe a gap between the mar-

ket demand in terms of power, speed of operation, and complexity and what current embedded computer vision solutions can deliver. At the algorithm level, researchers focus on the development of advanced and complex adaptive vision algorithms for the complex analysis that is required to employ machine learning principles for updating the algorithm parameters. In addition, for most implementations, predefined parameters (offline training) are used for machine learning algorithms, whereas at present, it is becoming necessary for the implemented machine learning blocks to be able to be trained online. Moreover, the availability of high-definition streams is imposing high operational requirements in terms of computation and communication for analyzing and updating the parameters of vision algorithms. Meanwhile, due to recent improvements in 3D depth cameras, the machine vision industry has been completely revolutionized. Depth cameras offer several advantages over traditional intensity sensors; depth cameras can operate at low light levels, provide calibrated scale estimates, produce color- and texture-invariant results, and resolve silhouette ambiguities in poses. As a result, most recently proposed algorithms operate with 3D data and require additional communication and computation in order to fuse the RGB and depth data.

In this research, we focus on a type of vision algorithm for which there is high market demand and that is also an active research area among researchers: human action and activity recognition. Most of the recently proposed algorithms for this purpose require the computation of complex mathematical models and have high communication interface requirements for streaming the data. Since one application of human action recognition is in security systems, it would be desirable to develop an embedded platform in which the entire system is integrated into a single board, which could be mounted on a camera for use in an online security system or a smart camera product.

The motivation for this dissertation is to investigate how to accelerate and implement a vision (action recognition) algorithm on an embedded platform. The proposed

approach enables not only reliable system design but also efficient resource utilization to achieve the optimal solution. The workflow of the design methods consists of the following:

- Develop a framework for complex human action recognition in a 3D context.
- Profile and analyze the sophisticated software-level operation of each algorithm and module.
- Design the corresponding hardware accelerator and optimize it for the mathematical model of each component.
- Find an optimal solution by balancing the system performance, energy consumption and resource allocation.
- Verify the operation of each component and the entire framework.

System performance, power consumption, and resource utilization are all important factors in our embedded platform.

1.6 Research Purpose and Difficulties

The aim of this research to propose a framework for human action recognition that can be synthesized and implemented on embedded platforms. An additional goal is to develop an efficient accelerator for each module in order to speed up operation to improve the performance of the system while considering constraints related to energy consumption and on-chip resource utilization. As discussed above, complex mathematical computations with extensive memory and communication interface needs are required for vision algorithms, especially for action recognition. As previously discussed, hardware acceleration can enable significant improvements in the efficiency of intensive mathematical calculations or repeated functions, such as the calculation of complex algorithms and loops. The performance of a SoC system can be improved

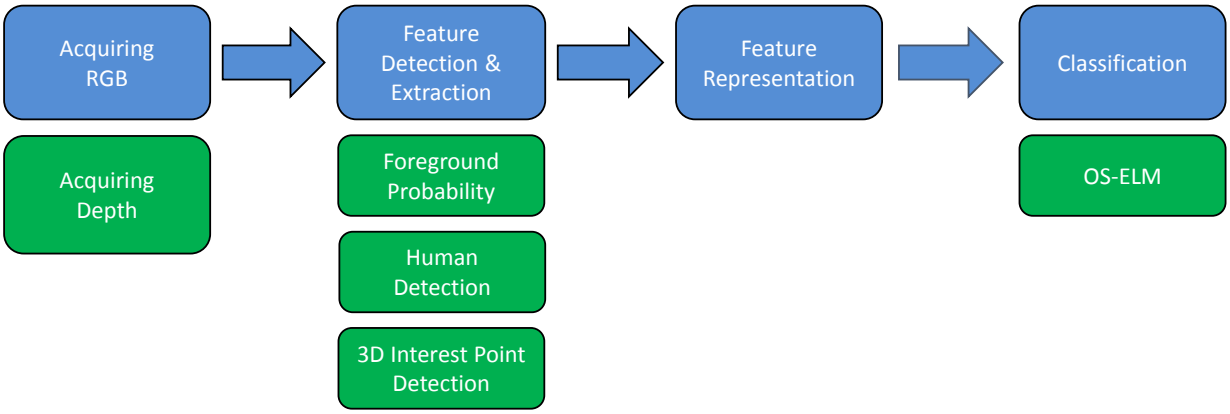


Figure 1.4 – The proposed framework.

by accelerating the element that incurs the highest calculation overhead. As a result, it is necessary to identify the function in each module that makes the operation slow and speed up that function with an appropriate accelerator circuit at the RTL level.

The proposed framework (Figure 1.4) was developed for complex human action recognition or the Recognition of Actions "In the Wild" [22]. The parameters that make recognition difficult are the following:

- Unknown camera motion
- Unknown camera viewpoint
- Lighting and shadow
- Undefined number of participants

For simple action recognition, the actions and motions of interest occur in a constrained environment, without the abovementioned variations. For better accuracy, depth information is also included; however, the framework is capable of working with both 2D and 3D streams. Since the camera viewpoint and camera motion affect the results of point-of-interest detection, two stages of preprocessing are included (foreground probability and human detection) to reject invalid points of interest.

The Harris Corners 3D [22] [23] [24] and Hessian Corners 3D [25] [26] operators are used as point-of-interest detection functions, and for later representation, the Bag of Visual Words (BOW) approach is used as a feature descriptor. Finally, in the last stage, an efficient machine learning algorithm that can recognizes the type of action is required. The online sequential extreme learning machine (OS-ELM) [27] [28] approach is a modified version of the ELM [29] [30] [31] [32] [33] approach that works with chunks of data instead of passing whole frames of data. The advantages of this method are, reduction resource utilization which is also more efficient for real-time application. The Xilinx Zynq 7000 SoC-FPGA [34] is considered as the platform for the implementation of the proposed framework. As represented in figure 1.5, all modules are synthesized and implemented in the FPGA in order to accelerate their operation.

The four major challenges hindering the embedded realization of the proposed framework are as follows:

- The need for sufficient storage space to maintain and update the model.
- The use of appropriate bandwidth and access time to update the framework parameters.
- Concurrent processing of and access to the modules aligned with the input stream.
- An efficient interface for connecting to the camera that satisfies the bandwidth requirement.

1.7 Organization of the Thesis

The remainder of this dissertation is organized as follows. Chapter 2 presents a brief overview of the basic concepts related to embedded vision as well as computer and

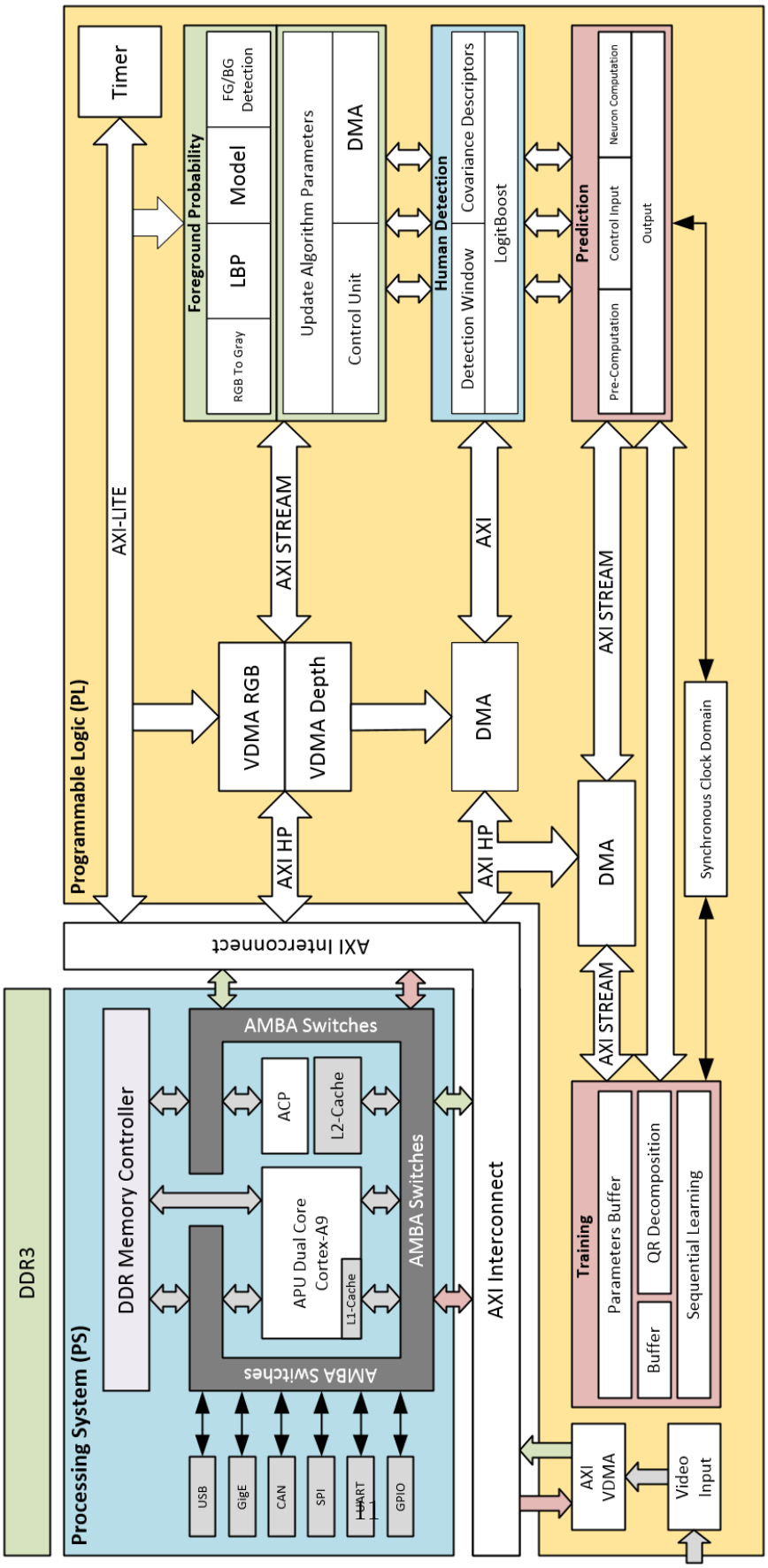


Figure 1.5 – The proposed framework implementation.

machine visionThe proposed approach for hardware acceleration for foreground and background identification (foreground probability) and the communication interface used to capture the stream and pass it to the memory block are discussed in chapter 3. The second stage of preprocessing, for identifying humans in the data stream and the appropriate circuit for speeding up operation, is discussed in chapter 4. Chapter 5 discusses point-of-interest detection, the feature descriptor modules, and the results of support vector machine (SVM)-based action recognition. Finally, the approach for the implementation of the OS-ELM approach in hardware, the operation of the prediction and training blocks and how these two blocks can be synced, and a comparison with the resource utilization and performance of the SVM method are discussed in chapter 6, followed by the presentation of conclusions and the suggestion of topics for future work in chapter 7.

References

[1] "Digilent Nexys Video Artix-7 FPGA: Trainer Board for Multimedia Applications," https://www.xilinx.com/products/boards-and-kits/1-cfdwic.html, accessed: 2018-04-18.

[2] "Integration of vision in embedded systems," https://www.vision-systems.com/articles/print/volume-22/issue-1/features/integration-of-vision-in-embedded-systems.html, accessed: 2018-04-18.

[3] B. Senouci, I. Charfi, B. Heyrman, J. Dubois, and J. Miteran, "Fast prototyping of a soc-based smart-camera: a real-time fall detection case study," *Journal of Real-Time Image Processing*, vol. 12, no. 4, pp. 649–662, 2016.

[4] P.-J. Lapray, B. Heyrman, and D. Ginhac, "Hdr-artist: an adaptive real-time smart camera for high dynamic range imaging," *Journal of Real-Time Image Processing*, vol. 12, no. 4, pp. 747–762, 2016.

[5] "ITRS 2.0 PUBLICATION," http://www.itrs2.net/itrs-reports.html, accessed: 2018-03-16.

[6] C. Mead and L. Conway, *Introduction to VLSI systems*. Addison-Wesley Reading, MA, 1980, vol. 1080.

[7] "Introduction of System-on-Chip," http://en.wikipedia.org/wiki/System-on-a-chip., accessed: 2018-03-16.

[8] "BOOST," http://www.boost.org/., accessed: 2018-03-16.

[9] "OpenCL: the open standard for parallel programming of heterogeneous systems," https://www.khronos.org/opencl/., accessed: 2018-03-16.

[10] "CUDA," https://developer.nvidia.com/cuda-zone., accessed: 2018-03-16.

[11] "Digital Signal Processor," https://en.wikipedia.org/wiki/Digital_signal_processor., accessed: 2018-03-16.

[12] "Digital Signal Processor (DSP)," http://www.ti.com/processors/dsp/overview.html., accessed: 2018-03-16.

[13] "Graphics Processing Unit (GPU)," https://en.wikipedia.org/wiki/Graphics_processing_unit., accessed: 2018-03-16.

[14] "Graphics Processing Unit (GPU)," http://www.nvidia.ca/object/gpu.html., accessed: 2018-03-16.

[15] N. Einspruch, *Application specific integrated circuit (ASIC) technology.* Academic Press, 2012, vol. 23.

[16] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field-programmable gate arrays.* Springer Science & Business Media, 2012, vol. 180.

[17] "Intel SoCs," https://www.altera.com/products/soc/overview.html, accessed: 2018-03-16.

[18] "Intel SoC FPGA Family," https://ca.mouser.com/new/altera/altera-socFPGA/, accessed: 2018-03-16.

[19] "ASIC, ASSP, SoC, FPGA What's the Difference?" https://www.eetimes.com/author.asp?section_id=216&doc_id=1322856, accessed: 2018-03-16.

[20] "Microsemi System on Chip FPGA," https://www.microsemi.com/products/fpga-soc/soc-fpgas, accessed: 2018-03-16.

[21] "SoCs, MPSoCs and RFSoCs," https://www.xilinx.com/products/silicon-devices/soc.html, accessed: 2018-03-16.

[22] S. Hadfield, K. Lebeda, and R. Bowden, "Hollywood 3d: What are the best 3d features for action recognition?" *International Journal of Computer Vision*, vol. 121, no. 1, pp. 95–110, 2017.

[23] I. Laptev and T. Lindeberg, "Space-time interest points," in *IN ICCV*, 2003, pp. 432–439.

[24] S. Hadfield and R. Bowden, "Hollywood 3d: Recognizing actions in 3d natural scenes," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE, 2013, pp. 3398–3405.

[25] G. Willems, T. Tuytelaars, and L. Van Gool, "An efficient dense and scale-invariant spatio-temporal interest point detector," in *Computer Vision–ECCV 2008*. Springer, 2008, pp. 650–663.

[26] P. R. Beaudet, "Rotationally invariant image operators." in *International Joint Conference on Pattern Recognition*, 1978, pp. 579–583.

[27] H. T. Huynh and Y. Won, "Regularized online sequential learning algorithm for single-hidden layer feedforward neural networks," *Pattern Recognition Letters*, vol. 32, no. 14, pp. 1930 – 1935, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865511002297

[28] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks," *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1411–1423, 2006, cited By 736.

[29] G. Huang, G.-B. Huang, S. Song, and K. You, "Trends in extreme learning machines: A review," *Neural Networks*, vol. 61, pp. 32–48, 2015.

[30] J. Cao, Z. Lin, G.-B. Huang, and N. Liu, "Voting based extreme learning machine," *Information Sciences*, vol. 185, no. 1, pp. 66–77, 2012.

[31] J. Tang, C. Deng, and G.-B. Huang, "Extreme learning machine for multilayer perceptron," *IEEE transactions on neural networks and learning systems*, vol. 27, no. 4, pp. 809–821, 2016.

[32] X.-Z. Wang, R. Wang, and C. Xu, "Discovering the relationship between generalization and uncertainty by incorporating complexity of classification," *IEEE transactions on cybernetics*, vol. 48, no. 2, pp. 703–715, 2018.

[33] Z. Huang, Y. Yu, J. Gu, and H. Liu, "An efficient method for traffic sign recognition based on extreme learning machine," *IEEE transactions on cybernetics*, vol. 47, no. 4, pp. 920–933, 2017.

[34] "Xilinx Zynq 7000 SoC-FPGA," https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html, accessed: 2018-03-16.

Chapter 2

Background

2.1 System-on-a-Chip (SoC)

2.1.1 Evolution

A system-on-a-chip (SoC) is a fabricated integrated circuit (IC) that consists of several preverified blocks called intellectual properties (IPs). These IP cores could be DSP blocks; memory blocks, such as DMA or VDMA; interface blocks, such as USB, SPI or I2C; embedded processor cores; or other digital or analog blocks. The first SoC was developed for the Microma watch in 1974; it was developed on a single Intel 5810 complementary metal-oxide-semiconductor (CMOS) chip to function as a driver for a liquid crystal display (LCD) to control the timing function [1]. With the development of technology for transistor manufacturing, the number of transistors that can be fabricated on a single chip has increased, allowing more components and blocks to be fabricated in a single IC. Figure 2.1 presents the general SoC architecture, and Figure 2.2 presents the modern SoC on which we developed and synthesized our framework.

Recently developed SoCs provide many IPs, such as a central processing unit (CPU) as the key block, a memory controller, memory storage blocks, and additional processing units that can act as co-processors or accelerators, such as application processing units (APUs), real-time processing units (RPUs) and graphics processing units (GPUs). Cores such as the ARM Cortex-A53 [4] are commonly used as APUs, and ARM Cortex-R5 [5] and ARM Mali-400 MP2 [6] cores can serve as RPUs and GPUs, respectively . A SoC also has a system bus for communication among the IPs. Several types of buses have been developed by different vendors; among them,

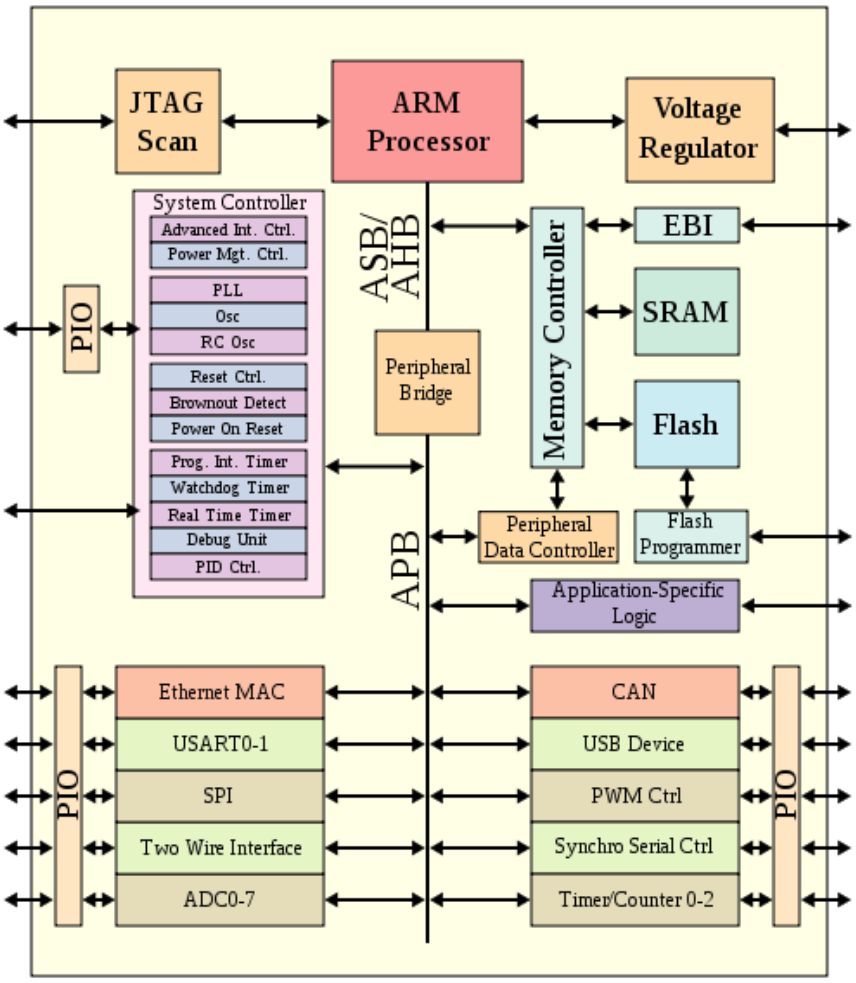


Figure 2.1 – General SoC architecture [2].

the AXI [7] buses developed based on the ARM AMBA [8] [9] protocol are some of the best known. Depending on the type of application, a SoC may also offer some general-purpose input/output (GPIO) or high-speed connectivity, such as through SATA3.1, PCI Express, SATA, PS-GTR interfaces, or general connectivity, such as through GigE, CAN or UART interfaces. In addition to the abovementioned components, recent SoCs often include reconfigurable logic blocks (SoC FPGAs). Such a reconfigurable block allows designers to synthesize soft IP cores, as well, such as soft core processors (MicroBlaze [10]), DSP functions, video codecs (H.265/H.264) or

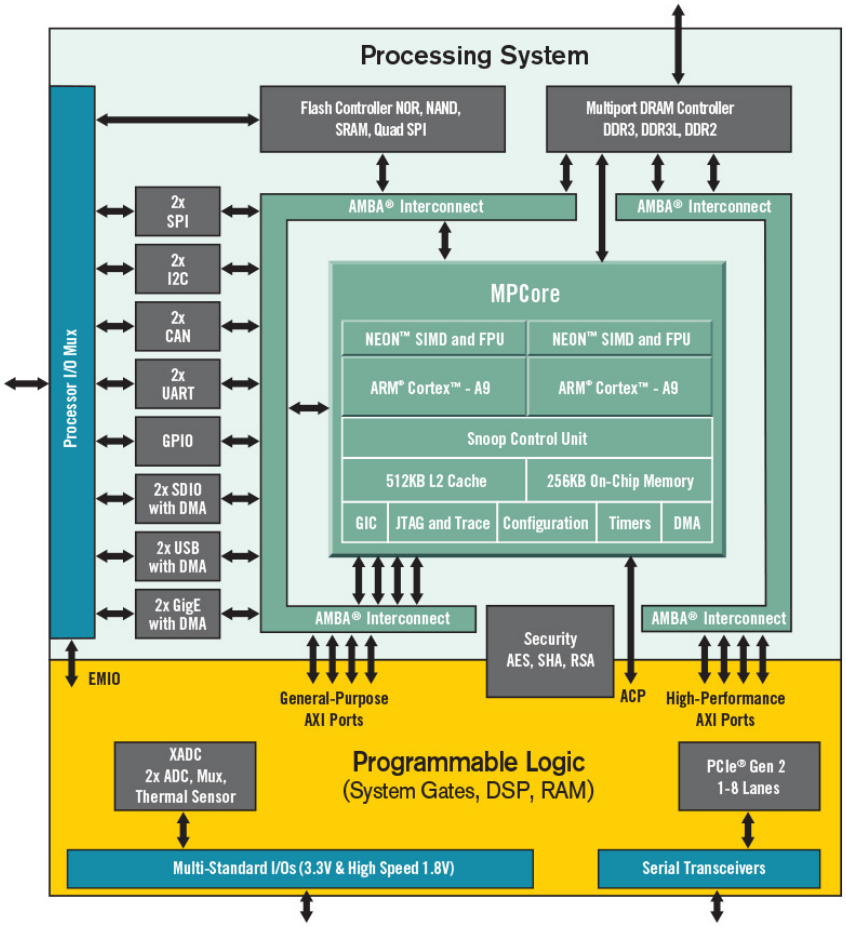


Figure 2.2 – Zynq-7000 SoC [3].

customized logic coded by the user. This provides more flexibility for designers to use the same SoC for different applications.

2.1.2 Design Trends

The recent trends in SoC design are generally focused on the following items:

- **Multi-Core Processor SoCs:** Integrating more cores can increase the processing capabilities of a SoC while also providing a more acceptable level of power consumption. This approach is currently considered as a common solu-

tion for modern design, especially for applications that require complex computations. In this approach, it is not required that all cores be the same. Fabricating cores such as APUs, RPUs and GPUs in the same die is common.

- **Power Efficiency:** As the density increases, the power consumption also increases. To cope with this problem, some solutions have been proposed in which idle cycles are introduced for some blocks to save power. Another approach is to manage the clock distribution; instead of using multiple clock sources, the system could operate on a single clock source, and the operation of each block could be controlled based on a different phase of the clock.

- **Reconfigurable Blocks:** As mentioned above, some recent SoC components (SoC FPGAs) include reconfigurable blocks to provide better flexibility for designers. SoC-FPGAs have recently become popular in image and video processing applications since most recent applications of this type have been developed for portable systems, making SoC-FPGAs an excellent option for implementing these algorithms.

2.1.3 Embedded Vision on a SoC

The algorithms that have been developed for machine vision applications consist of the components represented in the following pipeline; however, depending on the type of application, many of these components may be included or excluded. Because of the complexity of the algorithms, the processors are often optimized for the compute-intensive portions of the software workload. However, this approach is not applicable for embedded vision systems or portable systems. Moreover, power limitations restrict the deployment of machine vision algorithms in embedded systems, especially when the input stream is of full-HD resolution. These restrictions could be overcome by combining embedded processors with specialized hardware accelerators.

Some solutions have been developed in academia and industry and are provided

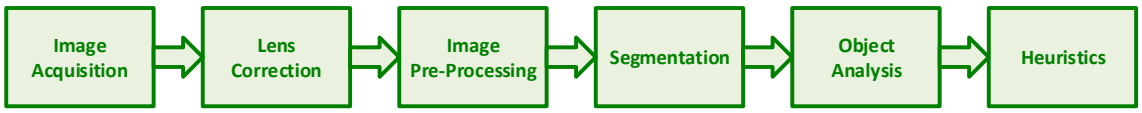


Figure 2.3 – Machine vision framework pipeline.

with ADI ADSP-BF60x [7] and TI DaVinci [11]. In these platforms, the high-level analytics are executed and controlled by a processor, while the computations for the machine vision algorithms are performed by accelerators. However, these platforms were developed for basic vision algorithms, such as vision filters, and advanced vision algorithms, which, as mentioned above, consist of multiple blocks of much higher complexity and irregularity of execution, have been left behind. In traditional approaches, custom hardware (accelerator) implementations are used for preprocessing, and the remaining computations are executed on the host processor. Some processors that support instruction-level parallelism (ILP) can speed up operation; however, the communication bandwidth between the custom hardware and the processor restricts the level of optimization. In recent solutions, an accelerator is integrated into the system as a co-processor; as a result, the accelerator can access streaming data from the system memory and system I/O and consequently lower the system traffic. Among the available platforms for the realization of the custom hardware approach, SoC-FPGA-based solutions are highly popular due to their balance between performance and power efficiency. Although ASICs provide better power consumption, 5x-12x [12] less than FPGAs, FPGAs are very efficient for both prototyping and real deployment due to their much lower design and development costs.

2.2 AXI Interface

AXI is the communication interface protocol developed based on the ARM AMBA [8] [9] [7] protocol for microcontrollers in 1996. The first version of AXI, AMBA 3.0, was introduced in 2003. The second version, AMBA 4.0, was released in 2010 and is

known as AXI4. AXI4 is the main bus interface protocol for Xilinx Zynq SoC-FPGAs. The AXI4 protocol includes three types of interfaces:

- **AXI4:** Developed for high-performance memory-mapped communication.
- **AXI4-Lite:** Developed for simple, low-throughput memory-mapped communication.
- **AXI4-Stream:** Developed for high-speed streaming data.

An AXI interface operates in both master and slave modes, represented by IP cores that exchange information with each other. Master and slave IPs can be connected using a structure called an interconnect block. The AXI Interconnect IP includes AXI-compliant master and slave interfaces and can be used to establish communications between one or more AXI masters and slaves. AXI4 and AXI4-Lite interfaces consist of the following five different channels:

- Read Address Channel
- Write Address Channel
- Read Data Channel
- Write Data Channel
- Write Response Channel

Transactions can pass in both directions between the master and slave simultaneously and can operate at different transfer sizes. An AXI4-Lite interface can transfer only 1 data transaction, whereas the burst operation in an AXI interface can transfer up to 256 data transactions. Figure 2.4 shows how an AXI4 read transaction uses the read address and read data channels.

As represented in the figures 2.4 and 2.5 , an AXI4 interface provides separate data and address channels for read and write operations. The advantage of using

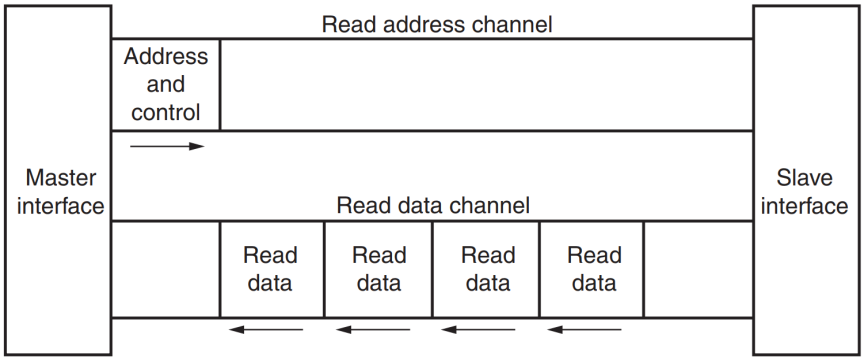


Figure 2.4 – AXI Architecture of Reads [13].

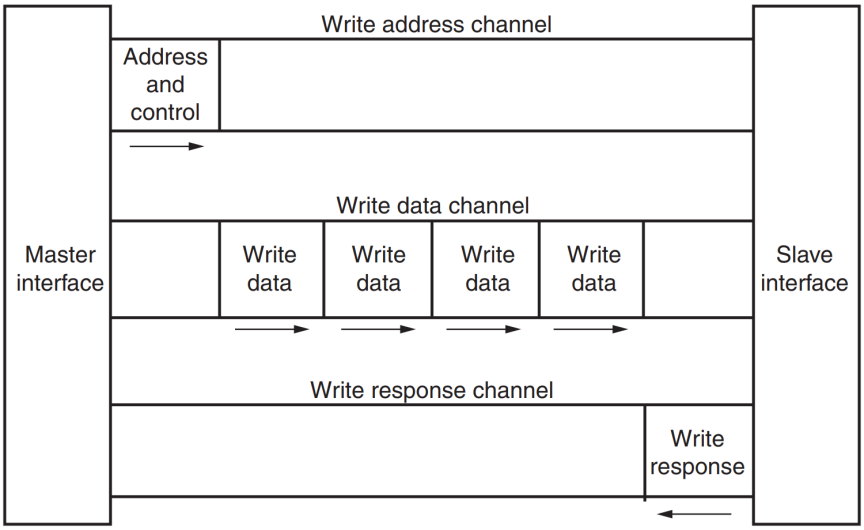


Figure 2.5 – AXI Architecture of Writes [13].

separate channels is that doing so enables simultaneous, bidirectional data transfer. An AXI4 transaction requires a single address and can burst up to 256 words of data. AXI interfaces provide a variety of features that allow a system to achieve high data throughput. These features include data upsizing and downsizing, multiple outstanding addresses, and out-of-order transaction processing. Also, AXI4 allows the use of a different clock for each AXI master-slave pair. Moreover, the AXI protocol allows the insertion of register slices to aid in timing closure.

The difference between AXI4-Lite and AXI4 is the bursting operation, which is not supported in AXI4-Lite. An AXI4-Stream interface operates with a single channel

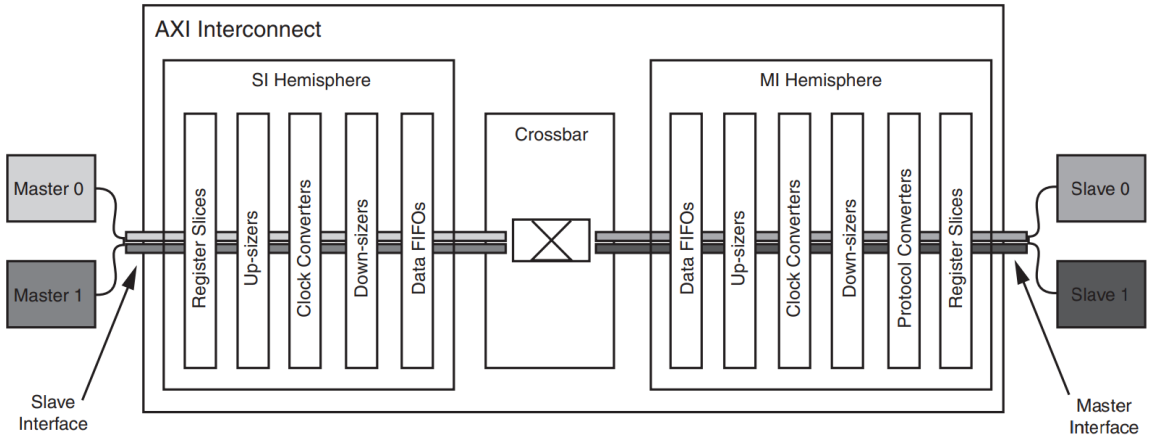


Figure 2.6 – AXI Interconnnect [13].

for the transmission of streaming data. The operation of AXI4-Stream is modeled after that of the write data channel of AXI4. Unlike AXI4 interfaces, AXI4-Stream interfaces can burst an unlimited amount of data. The additional features of AXI-Stream include the splitting, merging, interleaving, upsizing, and downsizing of the stream. Notably, the ability to reorder the transferred streams, which is available in AXI4 and AXI4-Stream, is not applicable in AXI4-Stream.

2.2.1 Xilinx AXI Infrastructure IP

A Xilinx SoC-FPGA provides common IPs that are used in most designs. In this section, we briefly discuss the main IPs that are used in many AXI-based systems. The AXI Interconnect IP is utilized to connect one or more AXI master devices to one or more slave devices (Figure 2.6).

The AXI Interconnect module consists of slave and master submodules and other functional blocks, such as registers, a FIFO block, a clock converter, and upsizers and downsizers, which manage the transactions between the master and slave. The slave accepts write and read transactions at the request of the master, and the crossbar module controls the traffic on all channels between the various devices connected to the slave and master interfaces. An AXI interconnect can operate in different modes

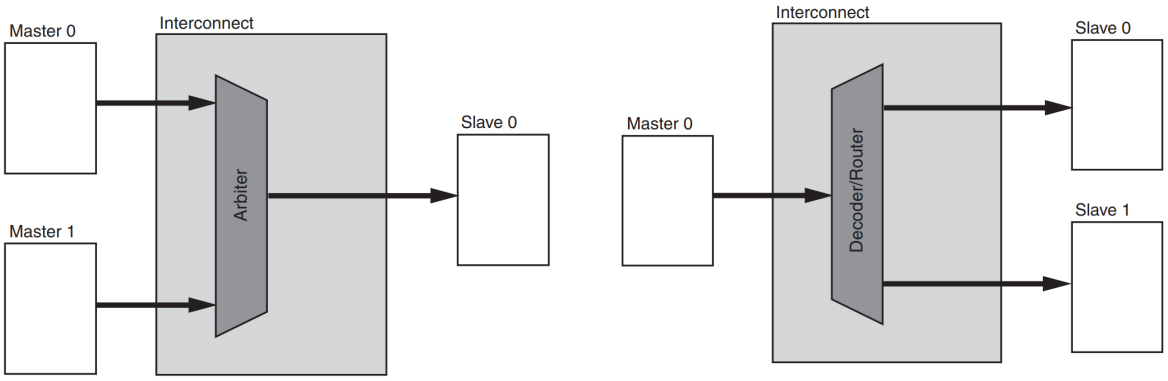


Figure 2.7 – N-to-1 AXI Interconnect (left), 1-to-N AXI Interconnect (right) [13].

depending on the architecture of the design. The modes in which an AXI interconnect can operate are as follows:

- Pass-Through
- Conversion Only
- N-to-1 Interconnect
- 1-to-N Interconnect
- N-to-M Interconnect

In the pass-through mode, the transaction passes only between one master IP and one slave IP; as a result, no conversion is required. The second mode, the conversion-only mode, again involves communication between one master IP and one slave IP; the following types of conversion can be employed by the interconnect:

- Data width conversion
- Clock rate conversion
- AXI4-Lite slave adaptation

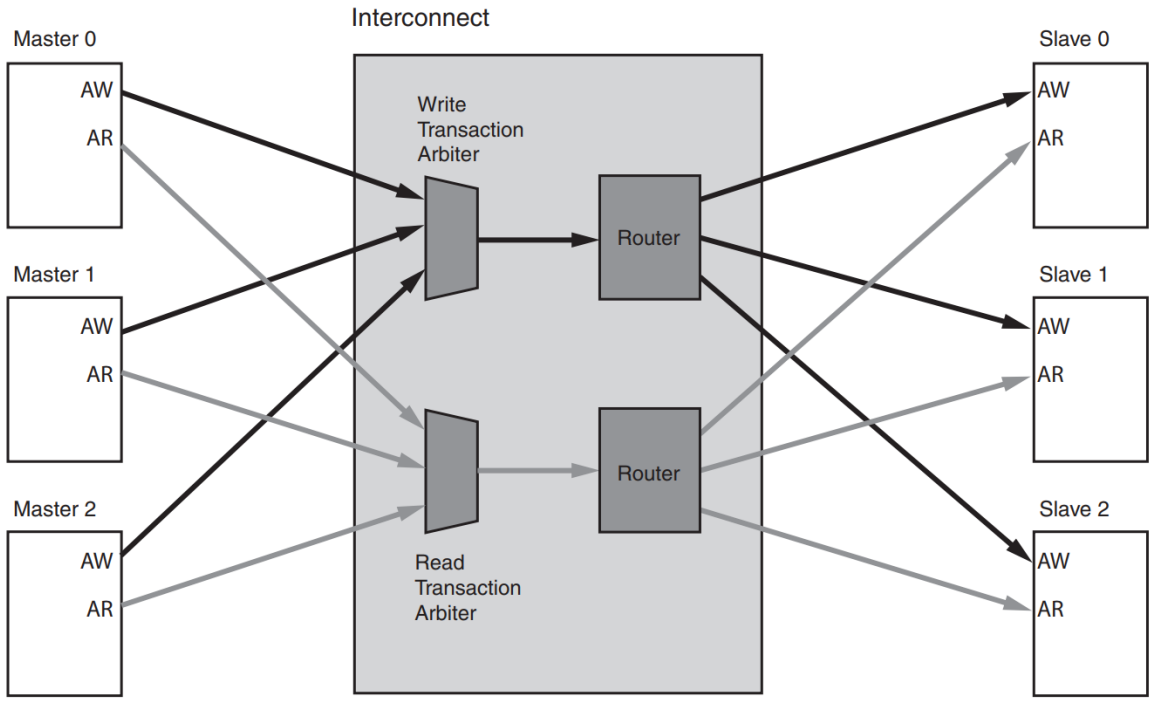


Figure 2.8 – N-to-M Interconnect [13].

- AXI-3 slave adaptation
- Pipelining, such as a register slice or data channel FIFO

The third and fourth modes are used when multiple master devices are required to communicate with one slave (N to 1) or when a single master is required to communicate with multiple slaves (1 to N). A typical example of the third mode is the operation of a memory controller, and a typical example of the fourth mode is when a processor is required to communicate with multiple peripherals. The arbiter and decoder/router are required to control the operation of N-to-1 and 1-to-N communication. The last mode is used when multiple IPs are required to communicate with each other. In this mode, the arbiter and decoder work together to control the transaction(s).

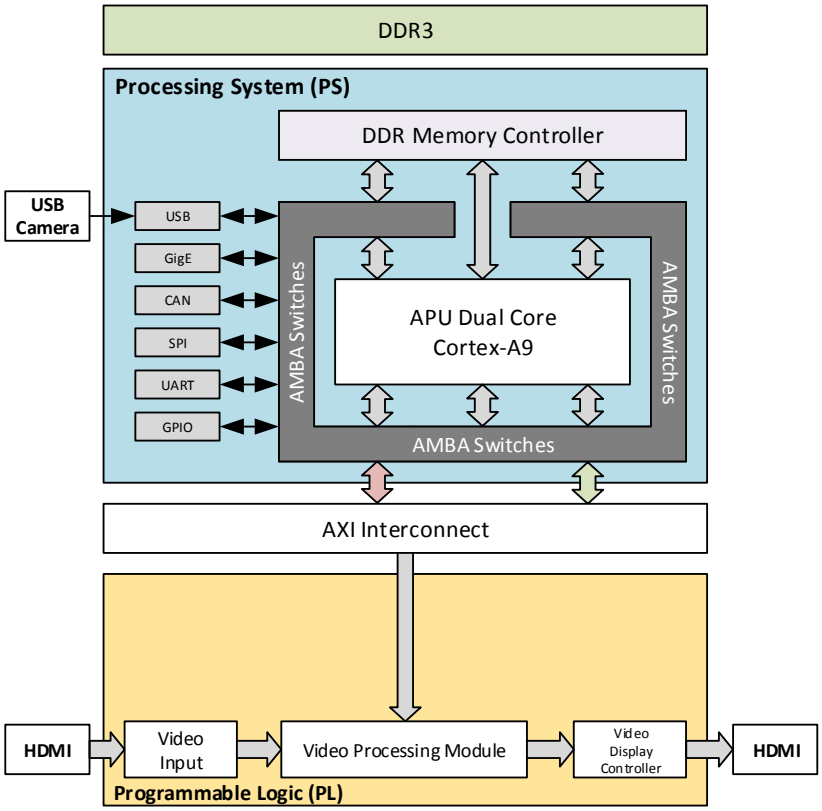


Figure 2.9 – Direct streaming.

2.3 Streaming Architectures

There are two main approaches available for the implementation of machine vision algorithms on embedded platforms. The first approach is "direct streaming". The input stream arrives directly from the communication link; the vision algorithm is directly employed, and the result is passed to the output interface. This approach offers the simplest and most efficient processing; however, it requires the design of components that can process frames in real time (Figure 2.9). In the second approach, which is called "frame-buffer streaming", the input stream is first stored in a memory block (VDMA). The video processing block reads each frame from memory and, after

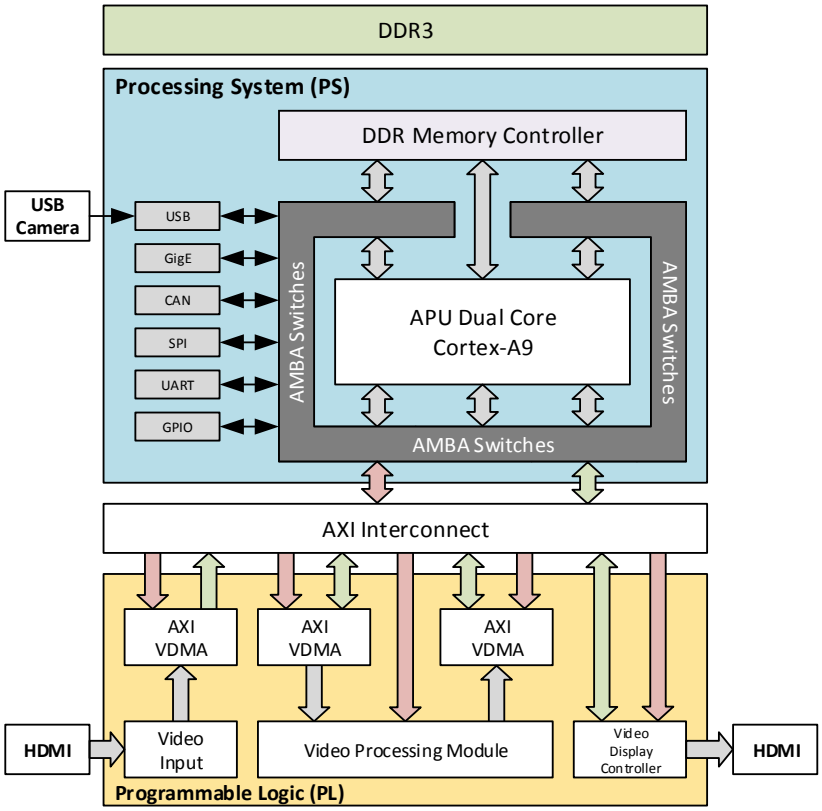


Figure 2.10 – Frame-buffer streaming.

processing it, stores it again in another block of memory (VDMA). The display of the results is controlled by another component that has access to this second block of memory. Therefore, there is no additional overhead for the video processing block. This approach offers more decoupling between the video rate and the processing speed of the video components. However, it requires a sufficient memory bandwidth for reading and writing frames from and to the memory (Figure 2.10).

2.4 System Profiling

Recently, the utilization of embedded SoC systems has grown rapidly as processing capabilities have increased. In addition, the market is demanding simpler and more efficient designs. In an efficient design, the final product should have the optimal partitioning of hardware and software. For this reason, tools such as profilers plays an important role in determining which function or component is the bottleneck in a system. Profilers assist designers in creating a proper balance between hardware and software in a system.

Since most of a SoC consists of embedded processors and accelerators, such as FPGAs and GPUs, finding the bottleneck function in the processor framework that slows the operation is achieved by executing the same function with an accelerator, and how the operation is optimized is essential. There are three main approaches available for system profiling: software-based profiling (SBP), hardware-based profiling (HBP), and FPGA-based profiling.

In SBP, function performance is evaluated using tools coded in the programming language of the system. The profiler keeps track of all processes running in the processor and helps the designer to identify the functions that make the operation slow. Profilers in this category include IDE tools such as Visual Studio.

Tools of the second type (HBP) use the counters integrated in the chips. Most common processors, such as Sun UltraSPARC processors, Intel Pentium processors, and Advanced Micro Devices (AMD) processors use dedicated counters to measure the execution of target functions. The advantages of this approach are that it does not interfere with the operation of the processor and data are collected during run time.

In the last approach, reconfigurable logic is used to measure performance. This approach can generally be used to measure the performance of soft core processors, such as MicroBlaze [10] and Nios II [14], which are synthesized on FPGAs, or hard

core processors, such as ARM processors, which are fabricated in SoC-FPGAs. The profiling components could be counters or timers that are synced with the processor clock to measure the clock cycles required to compute a specific function.

References

[1] "Computer History Museum," http://www.computerhistory.org/siliconengine/digital-watch-is-first-system-on-chip-integrated-circuit/., accessed: 2018-03-16.

[2] "System On a Chip," https://en.wikipedia.org/wiki/System _ on _ a _ chip, accessed: 2018-03-16.

[3] "Zynq-7000 Programmable SoC Product Advantages," https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html, accessed: 2018-03-16.

[4] "ARM Cortex-A53," https://developer.arm.com/products/processors/cortex-a/cortex-a53, accessed: 2018-03-16.

[5] "ARM Cortex-R5," https://developer.arm.com/products/processors/cortex-r/cortex-r5, accessed: 2018-03-16.

[6] "ARM Mali-400 MP2," https://www.arm.com/products/graphics-and-multimedia/mali-gpu, accessed: 2018-03-16.

[7] "AMBA AXI4 Interface Protocol," https://www.xilinx.com/products/intellectual-property/axi.html, accessed: 2018-03-16.

[8] "AMBA Introduction," https://developer.arm.com/products/architecture/amba-protocol, accessed: 2018-03-16.

[9] "AMBA SPECIFICATIONS," https://www.arm.com/products/system-ip/amba-specifications, accessed: 2018-03-16.

[10] "Xilinx MicroBlaze," https://www.xilinx.com/products/design-tools/microblaze.html, accessed: 2018-03-16.

[11] "TIs processors leading the way inembedded analytics," http://www.ti.com.cn/cn/lit/wp/spry280/spry280.pdf, accessed: 2018-03-16.

[12] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 26, no. 2, pp. 203–215, 2007.

[13] "AMBA AXI4 Interface Protocol," https://www.xilinx.com/products/intellectual-property/axi.html, accessed: 2018-03-16.

[14] "Nios II Processor," https://www.altera.com/products/processors/overview.html, accessed: 2018-03-16.

Chapter 3

System-On-a-Chip (SoC)-Based Hardware Acceleration For Foreground And Background Identification

3.1 Introduction

Embedded vision has experienced significant technological breakthroughs in recent years by virtue of the development of enabling technologies for the implementation of different and complex image and video processing algorithms with improved performance, higher efficiency, and reduced cost. Among these technologies, hardware acceleration has been critical to the development of such systems because it allows certain functions to be performed more efficiently than is possible in software running on a more general-purpose processor.

For the implementation of a vision algorithm, two main approaches are available. First, microprocessors (μPs) and digital signal processors (DSPs) offer all-software solutions that can meet the requirements of most applications, along with a complete set of development tools. Second, hardware acceleration, in the form of field-programmable gate arrays (FPGAs) or graphics processing units (GPUs), can perform specific tasks that are not suitable for μPs or DSPs. However, GPUs tend to have high power consumption, making them challenging to deploy in embedded environments.

Recent advances in silicon technology have brought to the market FPGA and microprocessor technology with sufficient resources to implement computationally de-

manding algorithms. In the new generation of system-on-a-chip field-programmable gate arrays (SoC-FPGAs), both a microprocessor and an FPGA are included on a single chip. Such a device consumes less power and can be incorporated into smaller systems, offering an attractive platform for embedded applications and making SoC-FPGA technology well suited for embedded systems from the perspectives of performance and cost.

With the recent increase in processing capabilities, machine vision algorithms are being increasingly implemented in embedded systems. Embedded vision, defined as the implementation and execution of visual analysis and computer vision algorithms using embedded systems [1], is an active topic in the field of embedded platforms that covers a variety of applications, such as autonomous vehicles, smart cameras [2] [3], industrial vision cameras, sensor networks and security systems (surveillance systems). The surveillance market alone is estimated to have more than tripled in the past decade, from $11.5 billion in 2008 to $37.5 billion in 2015 [4].

Most embedded vision applications feature common top-level requirements. First, a sequence is sampled at a number of spatio-temporal locations, and then, local feature descriptors are extracted. As the next step, the local features are encoded into a holistic descriptor, and finally, a discriminative classifier is used to identify likely categories. The main challenge in embedded vision systems is the tradeoff between power consumption and performance, where the power consumption becomes more important when the device is battery powered.

Some vision algorithms have been developed for specific streams and environments. Thus, the structure of such an algorithm requires constant updating of the parameters, and the quality of the output result is determined by the specific input to the algorithm. The type of application that is most appropriate for this method is vision filtering. Another type of algorithm that is currently of interest to researchers is adaptive vision algorithms, which are defined in terms of a model that is updated by tracking visual streams. Such algorithms are capable of executing complex tasks

with different types of inputs and in different types of environments, but they require considerable bandwidth and high-performance hardware. Most adaptive vision algorithms are based on machine learning principles, such as support vector machines (SVMs), extreme learning machines (ELMs) [5] [6] [7] and mixtures of Gaussians (MoGs), which require training steps; this could be another task for development.

The emergence of commercial full-HD cameras and depth sensors and the market desire for high-resolution processing have given rise to a need for a platform that can capably perform complex computations requiring many billions of operations per second (GOPS). As a result, high-resolution processing requires considerable power; however, for an embedded platform powered by a battery, the power consumption must be less than 1 W, which could pose a challenge for high-quality vision processing applications. The main challenges facing the implementation of adaptive vision algorithms on hardware, particularly when the input stream is of HD resolution, are as follows:

- The use of a storage space that can maintain and update the model;
- The use of an appropriate bandwidth and access time to update the model parameters;
- Concurrent processing and access to the model aligned with the input stream; and
- An efficient interface for connecting to the camera that satisfies the bandwidth requirement.

As mentioned above, traditional implementations are mainly restricted to filtering operations such as illumination or color enhancement or edge extraction. In most algorithms, these procedures require information only on the current frame and consequently have fairly low data storage and bandwidth requirements. However, such traditional approaches are not applicable for tasks that require adaptive

algorithms. New, efficient approaches are required that offer high performance in a portable system that is battery powered and consumes little power to enable the embedded deployment of adaptive vision algorithms. Memory traffic and strict alignment by considering an efficient buffering module are also required.

In vision algorithms, background and foreground identification is a common task in video content analysis (VCA). Although many background identification methods have been proposed in the last decade, algorithms with certain capabilities have still not been well studied. These capabilities include the ability to distinguish between foreground and background in imagery captured by a non-stationary camera, the ability to achieve sufficient robustness in identifying shadows, the ability to work with both 2D and 3D streams, the ability to be implemented in hardware and the ability to use a hardware accelerator to improve performance.

This chapter introduces a systematic approach that uses layer-based background subtraction in 2D/3D scenes to address the challenges of adaptive vision algorithms. The first step of the system-level analysis consists of identifying the bandwidth requirements and optimizing the architecture for bandwidth reduction. In addition, in this work, a communication-centric architecture is used to separate the algorithm data from the stream data, thereby allowing the algorithm data to be compressed for transfer and simplifying the multiple interconnections of IP and heterogeneous nodes.

The proposed system and algorithms were implemented as part of a study on human activity recognition on the Zynq-7000 SoC-FPGA platform. The system was evaluated on complex and challenging 3D datasets for human action recognition [8] [9] on a real-time 1080p, 30 Hz video stream. The proposed architecture is 300 faster than an ARM Cortex-A9 for computing local binary patterns (LBPs), which is one of the most computationally demanding functions; in addition, the architecture consumes 156 mW of on-chip power.

Organization: We first review relevant related works and then further explain the

background and motivation of our work. Our approach is discussed in more detail in Section 3.4. In Section 3.5, we discuss the results of the implementation and evaluation and how we validated our approach. Finally, in the last section, we present our conclusions.

3.2 Related Work

Research in the area of embedded vision has focused on two subjects: the platform and hardware, mostly with regard to the method of implementation, and the algorithm implemented in the hardware and the mathematics behind it. This section reviews these two subjects.

Constraints concerning budget and time to market are always the main restrictions facing embedded vision applications, particularly for full-HD resolution. Using new processors that were developed for fast processing is not an efficient solution, as this approach may solve the processing problem but also increases the total cost. Furthermore, for an embedded platform, the ability to customize a system after it has been delivered to market for a specific application is essential. Therefore, an efficient solution could be to combine processors with reconfigurable hardware to serve as accelerators. The AVNET ZedBoard [10] and Cyclone V [11] are two well-known platforms of this type. They can pass computationally intense kernels into hardware accelerators, while the high-level execution is performed on a processor. Using a reconfigurable device as an accelerator enables the system to change its application by modifying the logic of the reconfigurable part.

Various techniques have been proposed for detecting dynamic regions of a scene; a relatively simple approach is to compare two consecutive frames. The foreground and background are defined by identifying the foreground object and comparing the current frame against a background model. Most approaches require the model to be updated when the scene changes. Consequently, an adaptive algorithm may be

applied to different streams in different environments, such as different streams of a surveillance security system or different frames of an action movie. The background model must therefore be updated to accommodate scene changes. In [12] and [13], the background model is updated using a Kalman filter, whereas the approach in [14] uses a Wiener filter, and the method proposed in [15] uses a median filter.

Algorithms have been proposed in [16], [17], [18], [19], and [20] to identify the background model using static methods. The work in [17] is based on the static component of the color signal, whereas the optimized approach proposed in [18] is robust against shadows, and the algorithms presented in [19] and [20] use Gaussians to model the pixel information. Most work has focused on the MoG model, which was proposed by Stauffer and Grimson [20]. However, the main problem with the MoG approach is the tradeoff between the speed with which the model adapts to background changes and its stability. In [21], an improved speed was demonstrated without compromising model stability. Tuzel et al. [22] used recursive Bayesian estimation to estimate the probability distribution of the mean and covariance. This approach yields a multi-layer model and estimates the number of required layers. However, most of these approaches rely on color and intensity information and thus fail to detect a foreground object when the foreground and background are of similar color and pattern.

To address this problem, Heikkila et al. [23] proposed a robust approach based on discriminative texture features that are encoded based on LBP histograms to capture background statistics. The advantage of the LBP technique is that it analyzes local pixel values, making it robust to local changes in illumination. However, if this technique is employed on a surface with large uniform regions, the output is not efficient; for example, in an area of a scene that contains shadows or highlighting, the output result is poor. Some studies, such as [24] and [15], have used a shadow removal technique as post-processing, whereas [15] used zero mean normalized cross-correlation to identify shadow pixels. The photometric invariant color approach proposed by Hu et

al. [24] was applied in the RGB space to find the intensity of the pixel changes.

The work discussed above demonstrates that foreground and background identification is an active area in VCA; many researchers have contributed to developing adaptive algorithms with high processing rates. However, few researchers have focused on developing a hardware circuit for implementing the entire process in an embedded system to achieve a real-time system for embedded vision applications. Embedded vision applications with a large frame size require a dedicated platform and efficient architecture for their implementation. In [25] and [26], the implementation of the MoG technique on a GPU is discussed, where the main constraint is the low-power nature of the embedded system. Viable methods in the area of adaptive vision that use a reconfigurable platform have been proposed in [27], [28], [29], and [30], but the existing proposed architecture is not designed for full-HD resolution, and the reconfigurable part is used only for the co-processor. This approach results in significant synchronization overhead and unnecessary traffic for transferring the stream and operational data throughout the system memory. Furthermore, the current proposed architecture is not efficient for applications that require an adaptive vision algorithm.

Among the available foreground and background identification algorithms, the MoG approach has been studied in greater depth than other approaches [31] [32] [33] [34] [35]. The method proposed in [33] and [34] operates at a low resolution (120×120); the architecture proposed in [31] and [32] achieves optimized utilization by avoiding square root operations, thereby affecting the output results; and the architecture proposed in [31] is targeted at a resolution of 640×480. The solution proposed in [35] can operate at HD resolution and reduces the amount of traffic by compressing the Gaussian parameters, but that work remains at the simulation level and lacks actual real-time FPGA execution. The architecture proposed in [36] can process video frames with dimensions of 1024×1024 at 38 fps and, when implemented on a Virtex-II, can reach 39.8 fps. The architecture presented in [29] [25] follows the method proposed in [36]; this architecture achieves improved memory throughput by

employing a memory reduction scheme. The work reported in [37] is based on the OpenCV MoG algorithm and permits processing of a frame size of 1920 × 1080 at 22 fps. The improved circuit of [37] has been discussed in [38].

In summary, most approaches target low resolutions and/or take shortcuts to manage computational complexity. The inclusion of depth information has not been discussed, but this approach could also improve performance, albeit at the cost of increased processing requirements.

The methodology proposed in this work is intended to overcome the computational and communication challenges of embedded vision applications and improve the output results by considering depth information. Our methodology employs an algorithm/architecture co-design approach that uses photometric invariant color, depth data and LBPs to distinguish backgrounds from foregrounds. It outperforms the current approaches and is capable of using depth information. Furthermore, it addresses the challenges of system-level integration.

3.3 Background

This section discusses the contextual information related to this study and methodologies that have been used to implement adaptive foreground and background identification algorithms in embedded vision systems.

Vision algorithms can be categorized into two classes: filter-based approaches, such as Canny and Sobel edge detection, and adaptive methods such as MoG. Filter-based approaches mostly focus on incoming frames, whereas adaptive methods work across frames. Algorithms such as k-means are required to encode the features and information of the previous frames and use them to update the background model. This strategy requires a complex architecture and significant memory traffic, but it can address complex tasks with high quality.

Our approach is based on a successful adaptive vision method that was initially

Table 3.1 – Notations to be used in describing the proposed method

Notation	Meaning
t	Time
I	Input image, average RGB value
I_{min}	Minimum RGB vectors
I_{max}	Maximum RGB vector
$I = \{I^t\}_{t=1...N}$	Image sequence
D	Input depth
$D = \{D^t\}_{t=1...N}$	Depth sequence
g_c	Gray level of center pixel
g_p	Gray level of neighboring pixels
LBP	Primitive texture operator
$S(x)$	Sign function
R	Radius of a circle centered at (x, y)
n	Noise parameter
ω	Weight factor, the probability of a mode belonging to the background or foreground
ω_{max}	Maximum weight factor
l	Number of layers to which a mode can be applied

proposed by Heikkila et al. [23] and Yao et al. [39], although in our methodology, depth information is also considered. For a 3D scene, we apply the LBP approach while incorporating photometric invariant color measurements. Through the integration of photometric invariant color information and depth data, the framework overcomes the limitations of the LBP approach when applied to a surface with no or poor texture or to shadow boundary regions. Additionally, the framework utilizes adaptive weight parameters that make the algorithm robust for complex streams such as wavering tree branches. The proposed method also employs layer-based processing that can handle background changes in a scene due to the addition or removal of stationary objects, making the algorithm more efficient for moving objects or for a stream that exhibits sudden background changes.

3.3.1 Local Binary Patterns

An LBP operator is a primitive texture operator that labels the pixels in an image by thresholding the neighborhood of each pixel using the center value and considering the result in terms of a binary value [39]. Consider a pixel, x, in an image, I, that has a gray level of g_c, and let g_p represent the gray value of the neighboring pixels. The LBP label for pixel x is defined as

$$LBP_P(x) = \sum_{p=0}^{P-1} S(g_p - g_c + n)2^p \quad S(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{3.1}$$

$$x_p = x + Rcos(2\pi p/P) \quad y_p = y + Rsin(2\pi p/P) \tag{3.2}$$

where R is the radius of a circle centered at (x, y) and n is a noise parameter that is used to make the LBP robust to noise. For instance, in a uniform area, the effect of noise can reduce the accuracy of the output result; however, with a sufficiently high value of n, the LBP operator will not be affected by noise-related fluctuations in the pixel values.

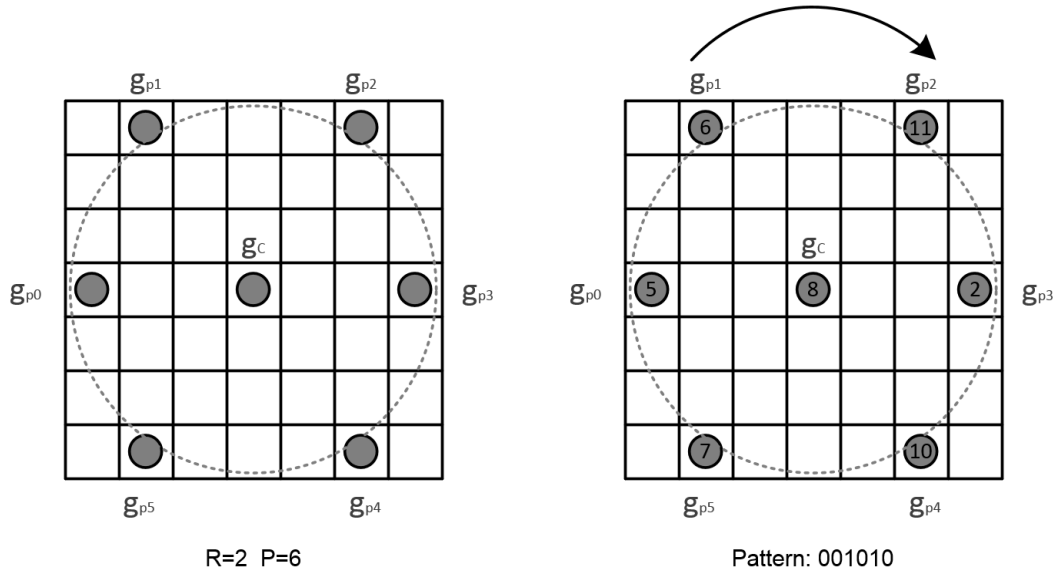


Figure 3.1 – Examples of the LBP operator.

In this study, we implement LBPs with a set of P binary values to reduce the memory and computational requirements. The implementation results demonstrate that LBPs have several advantages that make them useful for background modeling. They are robust to grayscale variations and can tolerate global and local changes in illumination. Additionally, LBP features can be computed very quickly, and such calculations do not require the determination of many parameters, which is important from a hardware implementation perspective. To improve the performance of the system, the method can be implemented as a complete pipeline.

In this work, an input color image is converted into a grayscale image, and the LBP operator is then applied to the resulting image for the input frame, using separate 3D data channels for the grayscale image and the depth data.

3.3.2 Photometric Invariant Color

The LBP descriptor is an excellent visual descriptor that can perform well in most video processing applications. However, this type of background modeling is not effi-

cient when applied to a surface that has a texture in common with the background, such as a wall, a floor or clothes. In such a case, the method may fail to distinguish the foreground object from the background. To cope with this problem, one common solution is to use the normalized RGB color space to account for changes in illumination. However, this solution is not efficient in dark regions because dark pixels have higher uncertainty than bright pixels. The solution in our framework is to use photometric invariant color information in the RGB space, which is robust to illumination changes such as shadows and highlights. According to [40], when the value of a pixel changes because of illumination changes, this change in value has a certain distribution with respect to the origin of the RGB space. In this approach, information on the angles of the differences of the foreground and background in RGB space with respect to the origin as well as the minimal and maximal background pixels is included in the learning procedure.

3.3.3 Background Modeling

To quantify the statistical information regarding a scene, including texture, color and depth information, a mathematical model is required. Let $\mathbf{I} = \{I^t\}_{t=1\ldots N}$ represent an image sequence, and let $\mathbf{D} = \{D^t\}_{t=1\ldots N}$ represent the corresponding depth information, where the superscripted t denotes the time. The mathematical model of the background has the form of a matrix $\mathbf{M} = \{M^t(x)\}_x$, which represents the statistical background information at time t for all pixels x belonging to the image grid. The model also consists of a list of modes that are defined based on the observed data, where for each mode, nine components are defined:

$$M^t(x) = \{m_k^t(x)_{k=1\ldots K}\} \tag{3.3}$$

$$m_k = \{I, I_{max}, I_{min}, D, LBP_{RGB}, LBP_{Depth}, \omega, \omega_{max}, \ell\} \tag{3.4}$$

in which I represents the average RGB value; I_{max} and I_{min} denote the estimated maximum and minimum RGB vectors, respectively; the parameter D is the depth

value; LBP_{RGB} and LBP_{Depth} denote the results of applying the LBP operator to the RGB-space data and the depth data, respectively; ω denotes the probability of the mode belonging to the background or the foreground; ω_{max} denotes the maximum weight that has been achieved in the past; and the parameter ℓ denotes the number of layers to which the mode can be applied, where 0 indicates that the mode is not reliable for any layer.

3.3.4 Background Model Update

When a new frame arrives, the model must be updated. Once the values of I_t and D_t have been obtained, LBP_{RGB} and LBP_{Depth} are computed. Then, it is necessary to determine which mode is associated with the new input. By computing the proposed distance measure (Equation 3.13), the closest mode is obtained, and all components of the selected mode are updated by means of Equations 3.5-3.10. If the output of the distance measure calculation is greater than a specific threshold and the number of current active modes is not greater than the maximum number of modes (K), a new mode may be created. In this case, all components are defined with their initial values. Figure 3.2 illustrates the model update procedure.

$$I^t_{min} = min(I^t, (1 + \delta I^{t-1})) \tag{3.5}$$

$$I^t_{max} = max(I^t, (1 - \delta I^{t-1})) \tag{3.6}$$

$$I^t = (1 - \lambda) I^{t-1} + \lambda I^t \tag{3.7}$$

$$D^t = (1 - \alpha) D^{t-1} + \alpha D^t \tag{3.8}$$

$$LBP^t_{RGB} = (1 - \lambda) LBP^{t-1}_{RGB} + \lambda LBP^t_{RGB} \tag{3.9}$$

$$LBP^t_{Depth} = (1 - \alpha) LBP^{t-1}_{Depth} + \alpha LBP^t_{Depth} \tag{3.10}$$

$$\omega^t = \begin{cases} (1 - \overline{\alpha_\omega})\omega^{t-1} + \overline{\alpha_\omega} \quad With \quad \overline{\alpha_\omega} = \alpha_\omega(1 + \tau\omega^{t-1}_{max}) & Mode = True \\ (1 - \overline{\alpha_\omega})\omega^{t-1} \qquad With \quad \overline{\alpha_\omega} = \frac{\alpha_\omega}{1 + \tau\omega^{t-1}_{max}} & Mode = False \end{cases} \tag{3.11}$$

$$\omega_{max} = max(\omega^{t-1}, \omega^t) \tag{3.12}$$

The parameters α, $\overline{\alpha_\omega}$, λ, τ and δ are learning rate values that are defined to ensure that the process avoids the maximum. The result for the background model is used to compute the distance measure, which is discussed in the next section.

3.3.5 Distance Measure

The proposed distance measure, which integrates texture, color and depth information, is defined as

$$\begin{aligned} M = {} & \omega_1 M_{text}(LBP^{t-1}_{RGB}, LBP^t_{RGB}) \\ & + \omega_2 M_{Depth}(LBP^{t-1}_{Depth}, LBP^t_{Depth}) \\ & + \omega_3 M_{Color}(I^{t-1}, I^t) \\ & \omega_1 + \omega_2 + \omega_3 = 1 \end{aligned} \tag{3.13}$$

where the ω parameters are weight values that indicate the contribution of each component to the overall distance. The texture and depth distances are defined as

$$M_{text,depth} = (LBP^{t-1}, LBP^t) = \tfrac{1}{P} \sum_{p=1}^{P} M_{0/1}(LBP^{t-1}_p, LBP^t_p) \tag{3.14}$$

$$M_{0/1} = \begin{cases} 0 & |x - y| < T \\ 1 & otherwise \end{cases} \tag{3.15}$$

Based on the changes in pixel values [40] in response to changes in illumination along the axis, the angle between two frames is used to measure the color distance.

$$M_{color}(I^{t-1}, I^t) = M_{angle}(I^{t-1}, I^t) + M_{range}(I^{t-1}, I^t) \tag{3.16}$$

$$M_{angle}(I^{t-1}, I^t) = 1 - \exp^{-\tau\theta} \quad \tau = 100 \tag{3.17}$$

$$\theta = \sqrt{1 - (\frac{\vec{I}^{t-1}.\vec{I}^t}{\|\vec{I}^t\| \ \|\vec{I}^{t-1}\|})^2} \tag{3.18}$$

$$M_{range}(I^{t-1}, I^t) = \begin{cases} 0 & \hat{I}^t_{min} < I^t < \hat{I}^t_{max} \\ 1 & otherwise \end{cases} \tag{3.19}$$

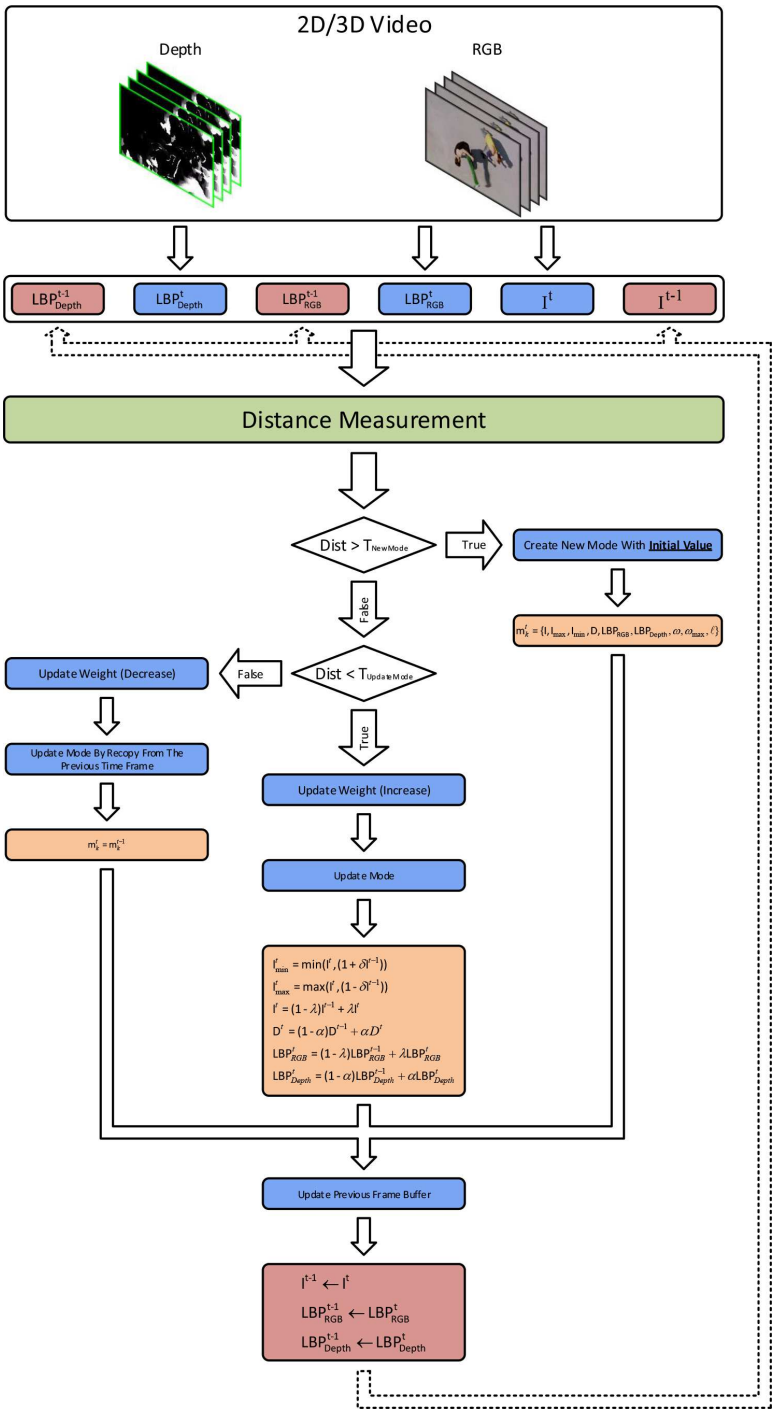


Figure 3.2 – Background model update procedure.

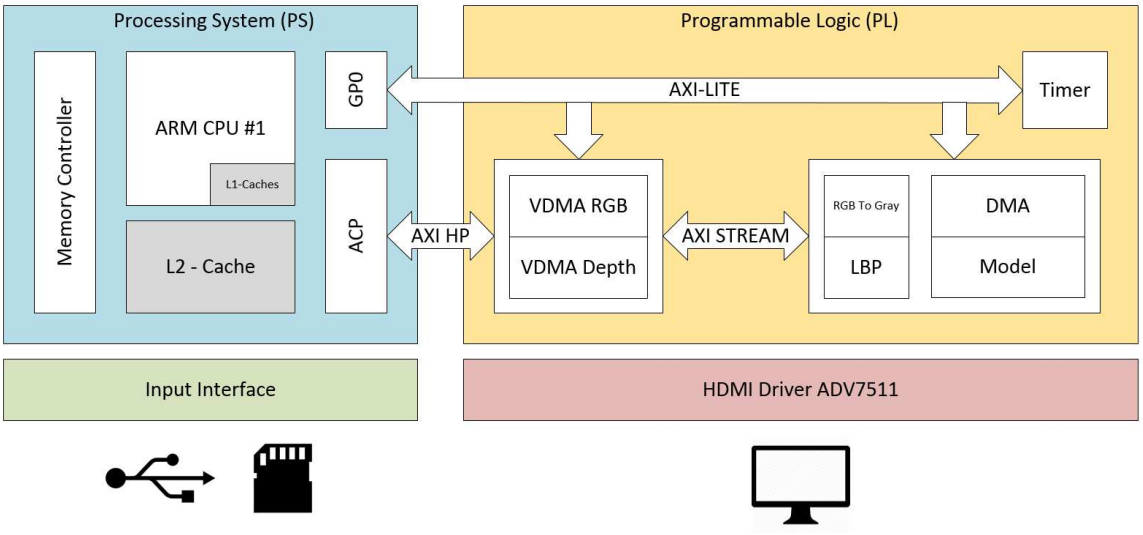


Figure 3.3 – SoC-FPGA block diagram showing video devices in the FPGA fabric that accelerate vision algorithms.

$$\hat{I}^t_{min} = min(\mu I^t, I^t_{min}) \quad \mu = 0.5 \tag{3.20}$$

$$\hat{I}^t_{max} = max(\eta I^t, I^t_{max}) \quad \eta = 0.5 \tag{3.21}$$

The output of the distance measure calculation defines whether a pixel belongs to the foreground or the background. If the value exceeds the defined threshold, the pixel is considered to be foreground, whereas if it is below the threshold, it is considered to be background. In the next section, the details of the implementation based on the presented mathematical operations will be discussed.

3.4 Approach

The design procedure for an embedded vision system consists of system-level exploration, computation and communication and, finally, system-level integration. The block diagram of the developed SoC-FPGA circuits is shown in figure 3.3. Efficient implementation is required for the placement of all dependencies and design options. Therefore, it is efficient to start from the abstract level to analyze and resolve the

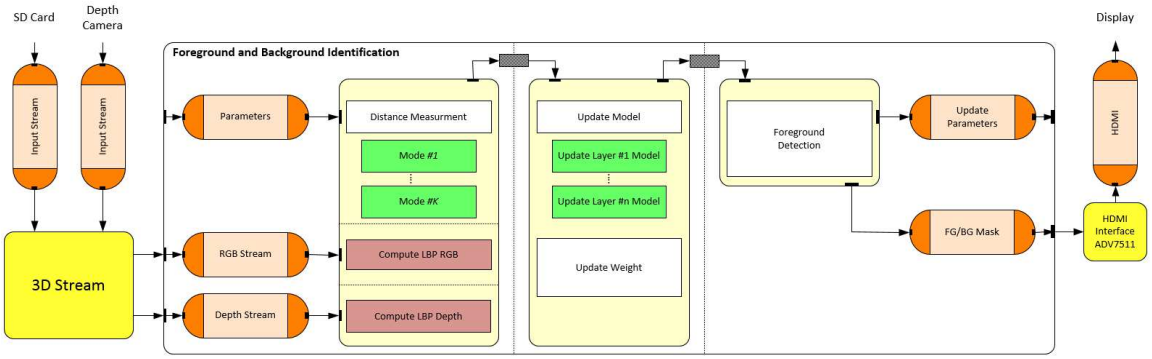


Figure 3.4 – Algorithm specification model, including coarse-grained parallelism.

problems at hand. Such an approach is helpful for evaluating and adapting both the algorithm and the architecture with respect to each other. Our design is constructed in a modular manner and then merged to form a functional adaptive foreground and background identification system. In the SoC-FPGA Zynq system, the processing system (PS) transfers streams to the programmable logic (PL) part, where the actual adaptive vision algorithm is implemented.

Figure 3.4 illustrates the abstract-level model of the algorithm using SpecC. A stream is captured from the camera or storage memory and transferred to the logic circuit; after processing, the result is passed to the HDMI interface. The model represents the concurrent processing and pipeline stages as well as the interfaces between blocks.

The pipeline stages consist of *LBP computation*, *distance measure computation*, *background model update* and, finally, *foreground detection*, all of which operate on each new stream.

The communication in the system consists of two main channels: the first channel contains the model parameters, and the second channel consists of the stream information. According to the SoC architecture, different protocols may be used. Because the device targeted in this work is Zynq, the main protocol for data transfer is AMBA AXI, which will be discussed in more detail later.

3.4.1 System Profile

An efficient solution is required to identify bottlenecks in the proposed system. Table 3.2 presents the computational demand of the algorithm in units of the number of operations per second (GOPS) and the communication demand for different resolutions. The specification model provides system-level profiling; in this work, we consider the SoC model proposed by Dömer et al. [41]. According to the result of the profiler, the background subtraction algorithm is computationally demanding, and the computational complexity makes it prohibitively expensive for software implementation; conversely, it is an excellent candidate for hardware implementation. The result of the profiler for communication also shows that a high communication volume is needed to update the algorithm parameters (Read/Write). One solution to this problem of high computational and communication demand is to reduce the resolution of the input stream, but considering the market demand for high accuracy, this is not a feasible solution. One effective solution could be to reduce the precision (bit width) for data traffic, as most of the traffic attributes are considered in the quality/bandwidth exploration of the algorithm parameters. The circuit illustrated in figure 3.5 can be used for precision adjustment. The multiscale structural similarity index (MS-SSIM) [42] is used to evaluate the precision by considering the structural similarity and comparing it against a ground truth. We investigate four types of scenes, namely, simple, medium, moderate and complex: in the simple and medium cases, the foreground changes; in the moderate cases, there are changes in

Table 3.2 – Computational and communication demands

Frame Size	Computation (GOPS)	Bandwidth (MB/s)
1920 × 1080	26.2	9460
1280 × 960	13.1	5830

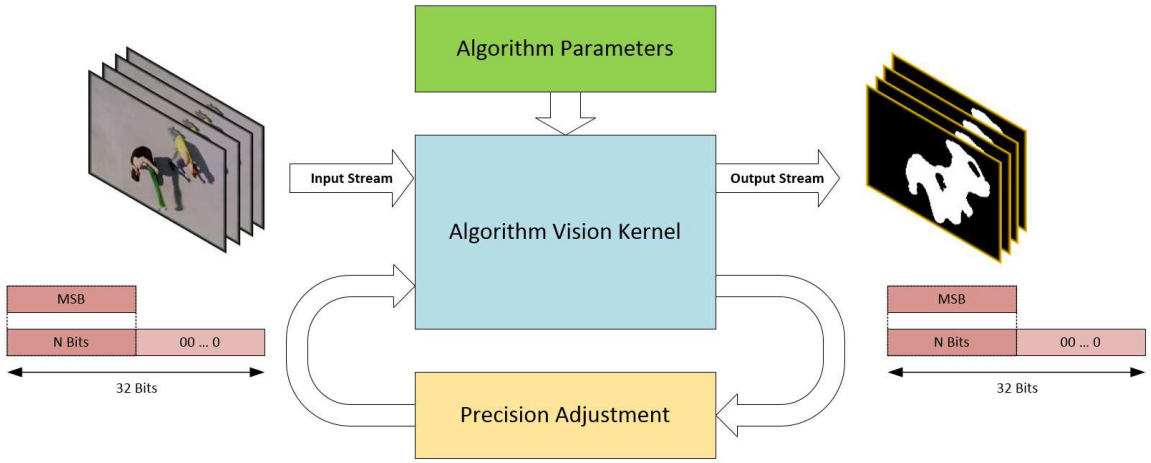


Figure 3.5 – Quality/bandwidth exploration for algorithm parameters.

both the foreground and background; and in the complex case, we consider videos from human action datasets that include objects suddenly moving in or out of frame. The experimental results show that the proposed architecture retains an MS-SSIM of 97% when the number of bits is reduced to 14 for all parameters of the model, except for M_{angle}, for which the number of bits can only be reduced to 16.

3.4.2 Computational Optimization

As discussed previously, the proposed model has high computational demands, and effective implementation requires different levels of optimization. The initial register-transfer level (RTL) design consists of three pipeline stages with parallel updating of the algorithm parameters, LBP computation and foreground detection. Further optimization is achieved by employing (i) algorithm tuning, (ii) unrolled loop identification, and (iii) deep pipelining techniques. Algorithm tuning modifies the proposed architecture to be more compatible with the target system. By identifying an unrolled loop, we can improve the utilization of the massive computational resources of the FPGA, as is further discussed in Section 4.5. In the deep pipelining stage, the individual coarse-level pipeline stages are further broken down into finer micro-pipeline

stages, such as in the LBP computation stage.

3.4.3 Communication Optimization

Two main types of traffic are involved in the proposed algorithm: one related to the input stream, (RGB+Depth)/Output, and the other related to the algorithm's intrinsic parameters. Figure 3.6 illustrates how the traffic of the algorithm is separated into two parts. The input stream traffic is related to the RGB and depth data of the raw data stream that is transferred from the input interface, where the input interface may be an RGB and depth-sensing camera [43] [44] or storage memory such as an SD card. This streaming traffic is related to the I/O interface and is defined based on the availability of the interface in the embedded platform, which is completely independent of the algorithm selected to achieve the desired functionality. By contrast, the algorithm-intrinsic data is related to the functionality of the algorithm. Even when different algorithms produce the same output, their different internal data structures may result in different degrees of algorithm-intrinsic traffic.

Some algorithms, such as mean shift [45] and MoG, require a complete history of the last n frames of data, which requires significant bandwidth and, consequently, high power consumption. By separating the input stream data and the algorithm-intrinsic data, it can be ensured that the tradeoff between quality and bandwidth is independent of the streaming pixels. To achieve this benefit, it is necessary to identify the traffic in the specification; the proposed architecture can also work with separate traffic.

For background identification, the streaming data consist of RGB pixels, depth pixels, gray pixels, probability and the background/foreground mask. For the LBP computation, it is necessary to transform the RGB stream into grayscale. Although it is possible to have separate LBP operators for each of the channels, this approach increases the processing requirements without affecting the accuracy. In this study, the input stream is transferred by the first processor (PS1) to the shared memory

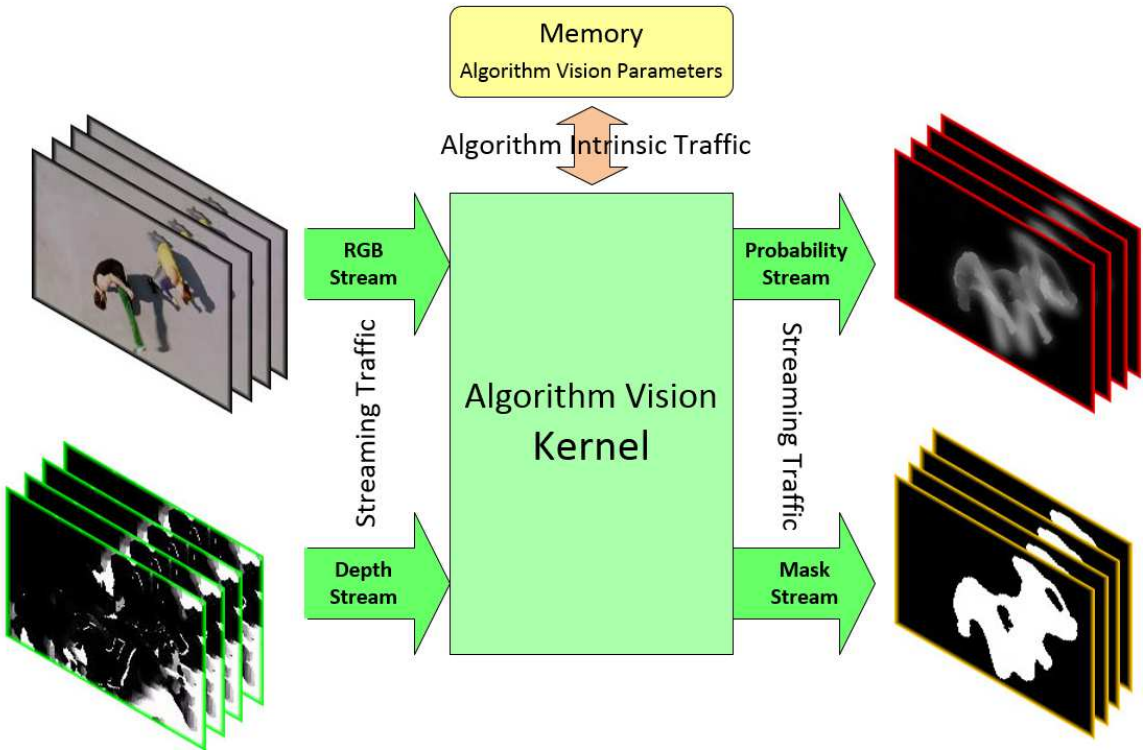


Figure 3.6 – Input/Output stream traffic and algorithm-intrinsic traffic.

(L2 cache), and then, it is transferred to the two blocks of the VDMA to store the RGB and depth data. The luminance value for each pixel is determined by the circuit implemented based on Equation 3.22. The luminance values are stored in 8-bit blocks in the RAM. The minimum mathematical operation, based on a shift operation instead of division, is considered in this transformation equation.

$$y = \left[\left[\begin{bmatrix} R & G & B \end{bmatrix} \times \begin{bmatrix} 66 \\ 129 \\ 25 \end{bmatrix} + 128 \right] >> 8 \right] + 16 \tag{3.22}$$

3.4.4 Communication-centric Architecture

Given the optimization procedure discussed in Sections 3.4.2 and 3.4.3, in which the architecture is optimized by separating the data into two domains, it is also

necessary to define separate clock domains to operate the stream data and algorithm-intrinsic data independently. The purpose of these separate clock domains is to achieve data alignment between the streaming and algorithm-intrinsic traffic. Figure 3.7(a) presents a block diagram of the architecture; as previously discussed, it consists of two clock domains: a computational domain set by the streaming data clocking the adaptive vision kernel and a communication domain set by the interconnect for the operational data. A separate data path is used to separate data access using individual ports, whereas the streaming data are processed through internal and external ports. For an external port, an efficient solution could be to use a dedicated system platform port, such as a USB or SD interface, to read the stream and an HDMI interface to visualize the stream. For an internal port, an AXI interface is used.

The SoC-FPGA that we use in this work is interfaced with an AMBA AXI version 4 to interconnect PS to PL and vice versa. The AXI interface consists of four 64/32-bit high-performance AXI (HP AXI) slave interfaces, two 32-bit AXI master interfaces, two 32-bit AXI slave interfaces, a 64-bit AXI accelerator coherency port (ACP) and an extended multiplexed I/O (EMIO) interface.

In this study, we use 2 HP AXI interfaces for stream traffic to connect the processor to the vision kernel and video capture modules, which require memory-mapped communication. To control each block, an AXI master interface is used to connect the vision kernel to the processor. The development platform (Zynq-7000 ZC702 Evaluation Board [46]) that we use in this work has multiple common interfaces, such as USB and SD memory readers and an HDMI driver (ADV7715).

To read and write back the vision kernel's intrinsic parameters, a dedicated DMA that is connected to the DDR3 is used. It operates in circular mode and automatically restarts when a new frame enters. The advantage of this technique is that it eliminates unnecessary synchronization with the processor.

Since the input stream and kernel operate in different clock domains, any misalignment between the input pixel and kernel data affect the entire process. To address

this problem, the initial alignment is synchronized when the first frame is received.

3.4.5 Loop Optimization

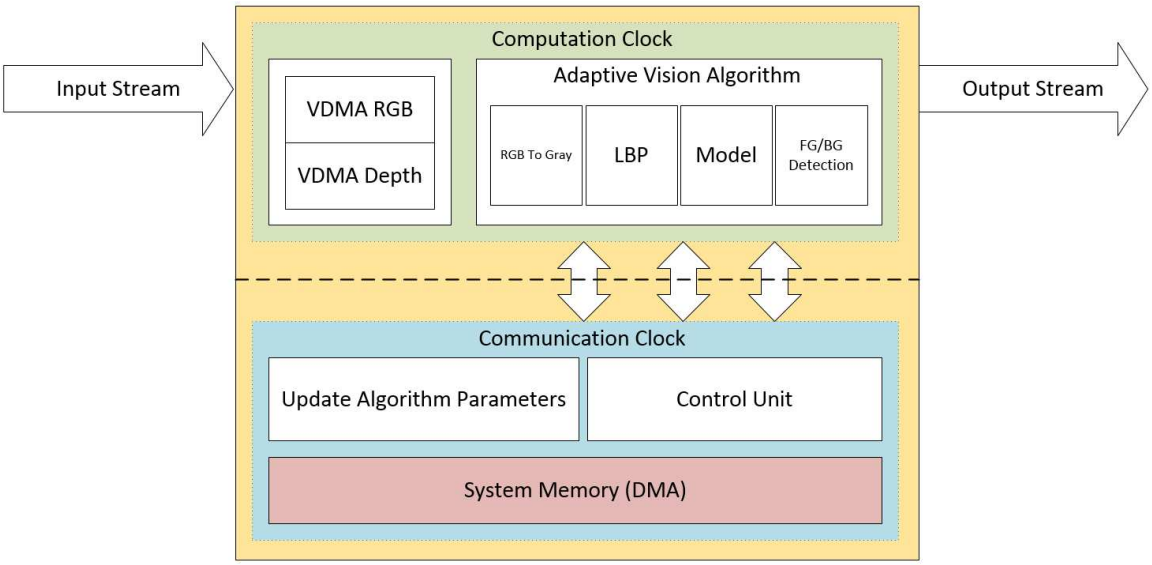
In the proposed vision algorithm (Section 3.3), the majority of the computational load is associated with the LBP computation. Since the center pixel must be compared with all pixels and the comparison results encoded in a binary representation (Equation 3.1), the latency in this step is considerable.

By identifying an unrolled loop, such as the loop in algorithm 1, we can improve the utilization of the massive computational resources of the FPGA.

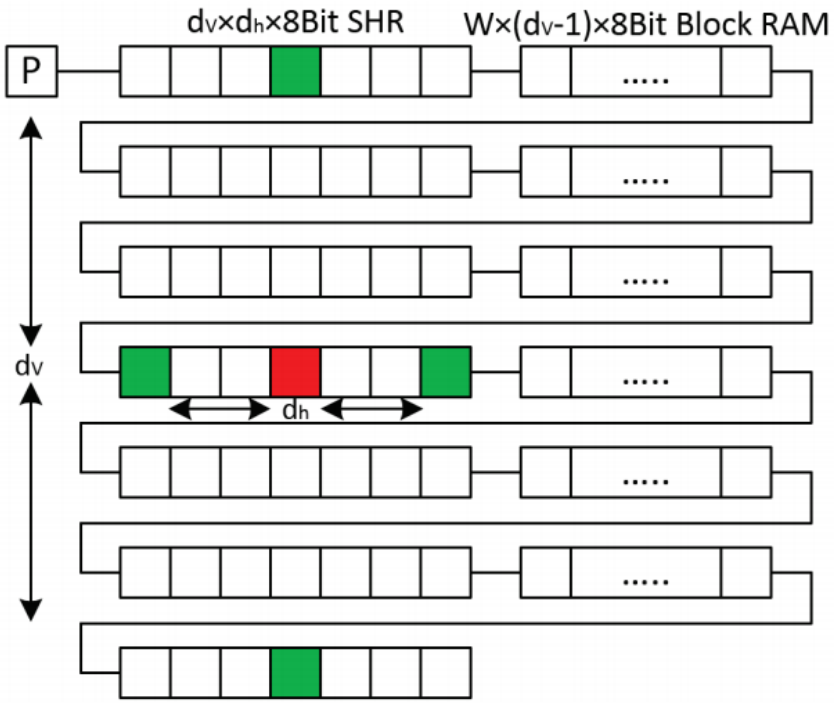
The various types of loops can be classified as follows:

- **Irrelevant:** If the loop index is not used in any function of array I, the loop index is considered irrelevant to array I.
- **Independent:** If the union of the data spaced on array A is completely separable along the loop index, the loop index is said to be independent of array I.
- **Dependent:** If the union of the data spaced on array A is not separable along the loop index, the loop index is said to be dependent on array I.

Algorithm 1 for the LBP computation consists of a single loop, and all variables depend on the loop iterator, p. Although the results of each operation are dependent on each other, the operation as a whole can be processed in parallel. An appropriate circuit for algorithm 1 is illustrated in figure 3.7(b) and consists of a $d_v \times d_h \times 8$-bit shift register that is accessible by each cell in parallel and a $W \times (d_v - 1) \times 8$-bit block of RAM. The values of the parameters d_v and d_h depend on the LBP model used, with the values in this work being 8 and 7, respectively. W is related to the width of the input frames. By using the proposed circuit depicted in figure 3.7(b), the LBP



(a)



(b)

Figure 3.7 – (a) Architecture with two clock domains. (b) The LBP computation circuit.

Algorithm 1 LBP Computation

1: **function** LBP(I_{center})
2: $I_{LBP} \leftarrow 0$
3: **for** $p \leftarrow 1$ to P_{nLBP} **do**
4: $x_p \leftarrow x + R \times cos(\frac{2\pi p}{P_{nLBP}})$
5: $y_p \leftarrow y + R \times sin(\frac{2\pi p}{P_{nLBP}})$
6: $I_{neighbor} \leftarrow$ **FuncShift**(I_{center}, x_p, y_p)
7: $I_{difference} \leftarrow$ **FuncDifference**$(I_{center}, I_{neighbor}, n, 2^p)$
8: $I_{LBP} \leftarrow I_{LBP} + I_{difference}$
9: **end for**
10: **return** I_{LBP}
11: **end function**

descriptor of a pixel of interest can be computed in two clock cycles. The values of the pixels of interest and their neighbors require only one clock cycle to update.

3.4.6 System Integration

A traditional embedded vision system is implemented with two chips: an accelerator and a host processor. Depending on the type of application, the accelerator could be an FPGA, GPU or DSP that works as a co-processor for the host processor. In this architecture, both the streaming traffic and the algorithm-intrinsic traffic occupy the interconnect and system memory. The host processor transfers the input stream to the shared memory; when the co-processor is triggered, the contents of the shared memory are processed by the co-processor, and the output of the algorithm is returned to the shared memory. In the final stage, the host processor reads the content of the shared memory and transfers it to the output port. All transactions are controlled and monitored by the host processor, which is inefficient and leads to high overhead.

A more efficient approach is to use the new SoC-FPGA known as the Zynq-7000, which contains a processor and an FPGA fabricated on the same chip. In this architecture, the shared memory between the host processor and the accelerator (co-processor) is an L2 cache with low latency. The ACP is the interface between the AXI4-Stream interconnect and the host processor (ARM CPU). The ACP is a 64-bit AXI slave interface on the snoop control unit (SCU), which provides an asynchronous cache-coherent access point directly from the PL to the host processor subsystem. The ACP provides a low-latency path between the PS and the accelerator implemented in the PL. The host processor needs only to initialize the cache, after which the input stream (RGB and depth data) is transferred in a completely independent manner to the VDMA in the PL. As a result, more cycles remain available on the host processor for higher-level processing.

3.5 Experimental Results

This section discusses the results of the implementation of the proposed architecture on the Xilinx Zynq platform, including the performance, power consumption and resource utilization, and compares these results with recent work.

3.5.1 Algorithm Implementation and Evaluation

The proposed hardware acceleration implementation of the background and foreground identification algorithm was synthesized and implemented (Figure 3.8) on a Zynq-7000 XC7Z020-CLG484 AP Xilinx SoC with a Vivado 2015.2 synthesizer. The SoC has two ARM Cortex-A9 processor cores (PS part) and a PL Artix-7 FPGA (PL part). The software for the processor is written in C, and the other modules are written in VHDL. Both VDMA RGB and depth capabilities are offered by the IPs that are provided in Xilinx Vivado. In addition, an embedded version of Linux, PetaLinux [47] [48], which is customized for Xilinx SoCs, is used to control the op-

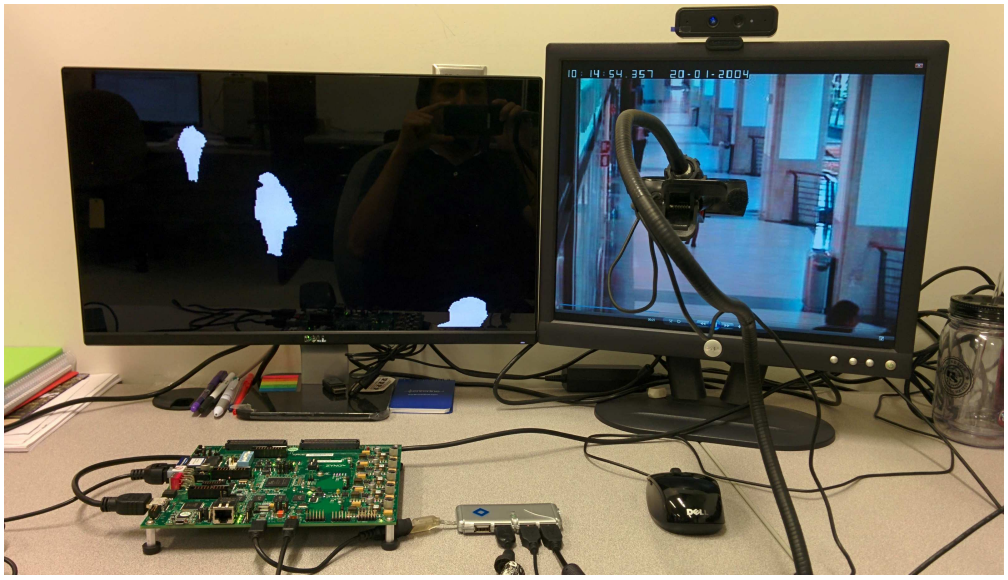


Figure 3.8 – System illustration.

eration of the processor. As mentioned in Section 4.4, AMBA AXI version 4 is used internally to interconnect the processor cores with the FPGA part. The platform is also equipped with two off-chip memory interfaces, DDR3 and LPDDR2. The ARM processor reads a raw stream from the SD card or the USB interface, and by sharing it in the L2 cache, the stream is made accessible to the PL part for further processing. The RGB and depth data are stored separately in two blocks of the VDMA for further processing with the background subtraction kernel. After processing, the output result is passed to the HDMI display chip (ADV7715). As discussed previously, we adopted a communication-centric architecture that requires two clock sources; the clock module gives rise to a 100 MHz clock that creates 120 MHz and 148 MHz signals for communication and computation, respectively.

The kernel parameters receive and update the model parameters via a DMA that connects to the DDR3 memory via the AXI stream and works in circular mode. For all control signals, an AXI-Lite interface is used to configure and initialize all modules and submodules in the PL part.

We compared the proposed algorithm with other previously proposed implementa-

tions on Virtex-5 (XC5V1X50), Virtex-4 (XC4VFX12), and Virtex-II Pro (XC2PRO30) Xilinx FPGAs. The synthesizer for the Virtex-4 and Virtex-5 FPGAs was created using XST, whereas Synplify was used for the Virtex-II Pro. The evaluation was performed based on each IC's utilization of the available resources and a comparison of the background identification results achieved on data used in previous studies. We also tested the proposed system using datasets of 3D videos that were generated by IR cameras or through stereo vision [43] [44]. The power consumption of the proposed architecture and the acceleration factor are discussed in the next section.

3.5.2 Resource Utilization and Evaluation

Table 3.3 summarizes the resource utilization results and compares them with those of previous studies. We tested our proposed system using most of the available datasets and new datasets for human action recognition. Some of these datasets, such as the Hollywood 3D [8] and HON4D [9] datasets, include depth data. We tested our hardware on the KTH [49], Wallflower [14], Background Models Challenge (BMC) [50] [51] and Weizman [52] datasets, which are invaluable because they provide benchmarks for comparing computational approaches. Of these datasets, the Wallflower [14], HON4D and Hollywood datasets focus on human action recognition.

We also tested our hardware using several typical and challenging scenarios from highway and crowd datasets, namely, the PETS2009 [53], PETS2017 [54], MOT2015 [55], MOT2016 [56] and TUD [57] datasets, containing scenes from surveillance systems capturing human activities. The results show that the algorithm can appropriately subtract the foreground from the background in both a highway environment and a crowded environment (Figure 3.9).

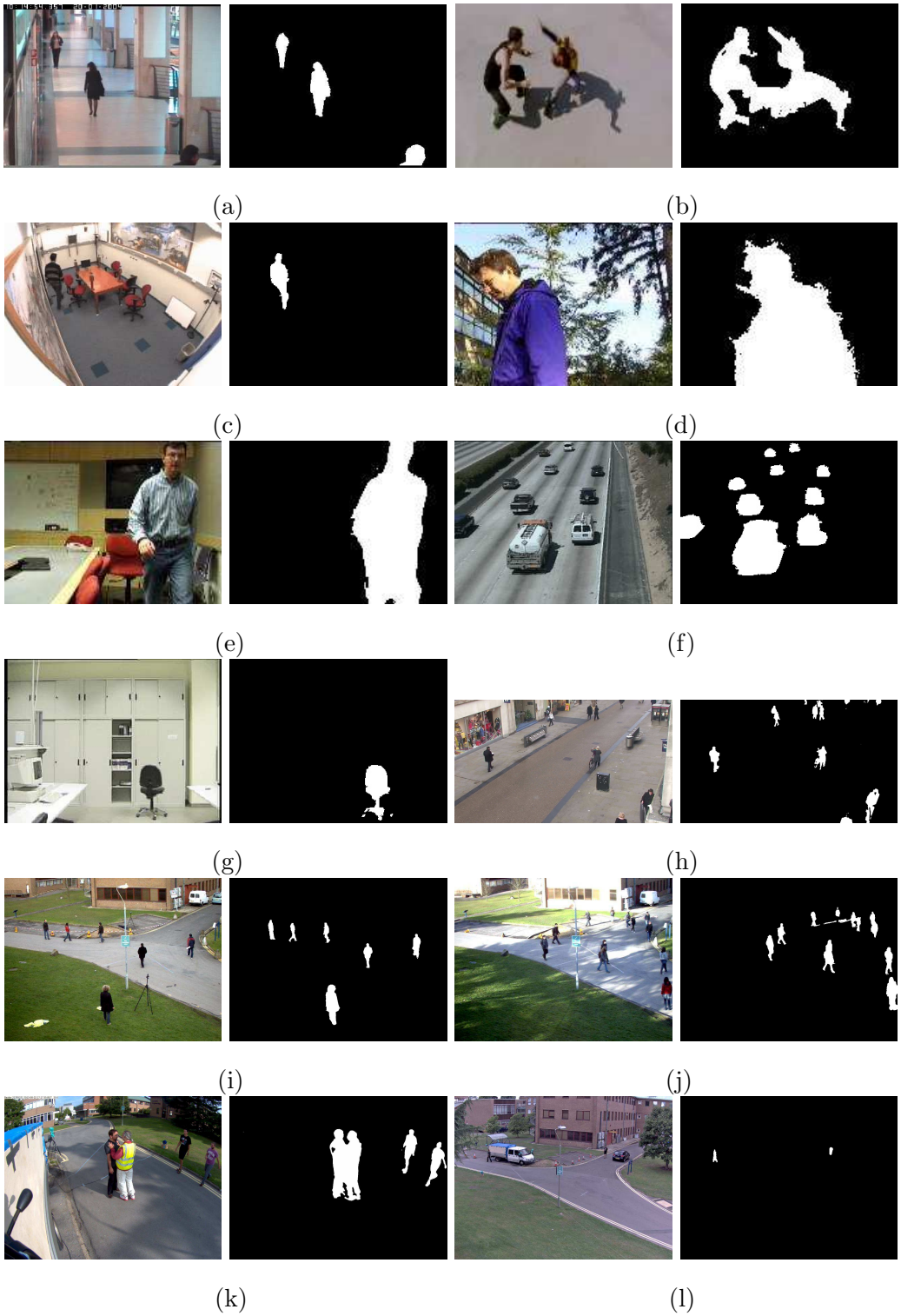


Figure 3.9 – Results of the implemented algorithm on examples from various datasets: (a) Ref [23], (b) Ref [8], (c)(d)(e)(f)(g) Ref [14], (h) Ref [55], (i)(j) Ref [53] and (k)(l) Ref [54].

Table 3.3 – Consumed FPGA resources

Target FPGA	LUT	Flip Flop	Slice	DSP-MULT
Virtex-II [58]	3394/27932 (12%)	N.A.	N.A.	7/136 (5%)
Virtex-4 [59]	6583/10944 (60%)	1131/10944 (10%)	3824/5472	N.A.
Virtex-4 [59]	6451/10944 (59%)	1132/10944 (10%)	3770/5472	N.A.
Virtex-5 [60]	1572/28800 (5%)	0/28800	N.A.	N.A.
Virtex-5 [61]	1066/28800 (4%)	0/28800	346/7200	10/48 (21%)
Proposed System	**1051/53200 (2%)**	**80/106400** (0.1%)	**1131/159600 (1%)**	**1/220 (0.5%)**

3.5.3 Qualitative Evaluation

We compared the results of our proposed method with those of other approaches: GMM [20], codebook [40], LBP [23] and εLBP [62]. It should be noted that these other approaches were tested on a PC platform, whereas our work was tested on an embedded platform with limited memory space for defining float and double variables as well as floating point operations. To fairly evaluate and compare the accuracy of the different approaches, we used a qualitative evaluation formula. The output of this formula (F-score) was taken as the accuracy. The formula is defined as

$$F = \frac{2 \times TP}{2 \times TP + FN + FP} \tag{3.23}$$

where TP, FP, and FN are the numbers of true positives (correctly identified foreground pixels), false positives (background pixels identified as foreground pixels), and false negatives (foreground pixels identified as background pixels), respectively. Among the datasets on which we tested our proposed method, some were originally developed for other computer vision applications, such as tracking and action recognition. Consequently, these datasets do not contain ground-truth information for background subtraction that can be used for evaluation. Traditionally, the Wallflower dataset [14] has been used for benchmarking. However, the BMC dataset [50] [51] has also recently been used for benchmarking for complicated scenarios. The results

of comparisons on these datasets are summarized in two tables. Table 3.4 compares the proposed method with the other approaches on the Wallflower dataset [14]. Table 3.5 compares the results obtained on the BMC dataset [50]. Table 3.5 also presents the range of F-scores obtained for each method.

Table 3.4 – Comparison of all methods on the Wallflower dataset [14]

Algorithm	GMM [20]	Codebook [40]	LBP [23]	εLBP [62]	**Proposed Method**
Bootstrap	45%	66.5%	61.1%	76.6%	**78.2%**
Camouflage	97%	98.6%	89.7%	98.3%	**98.8%**
Light Switch	28%	31.8%	58.2%	51.8%	**78.2%**
Time of Day	91.8%	90.9%	76%	90.5%	**92.1%**
Waving Trees	89.3%	93.8%	73.8%	76.0%	**94.5%**

Table 3.5 – Comparison of all methods on the BMC dataset [50] [51]

Algorithm	F-score
Naive Approach (NA)	55% - 85%
Gaussian Mixture Model (GMM1) [20] [63]	75%
Gaussian Mixture Model (GMM2) [64] [65]	93%
Bayesian Classification (BC) [66]	91.8%
Codewords and Codebooks (CB) [40]	94%
VuMeter (VM) [67]	80% - 85%
Proposed Method	**86% - 91%**

From Table 3.4, it can be seen that our proposed method outperforms the other classical background subtraction algorithms on the Wallflower dataset [14]; however, the results presented in Table 3.5 show that one of the Gaussian mixture model methods (GMM2), Bayesian classification (BC) [66] and the codewords and codebooks (CB) approach [40] outperform our method. Nevertheless, it should be noted

that our proposed method was tested and implemented on an embedded platform, and the main concerns for an embedded platform are resource utilization and power consumption.

3.5.4 Power Consumption

The power consumption of the proposed architecture was measured using a Xilinx X-Power Analyzer after the implementation of the entire system on the Zynq platform. For our approach, the on-chip power was measured to be 1.747 W, where 91% of the total power is related to the dynamic power. The processor consumes the majority of the power (1.529 W), whereas the remaining blocks consume 9% of the total power (0.156 W). The total power consumption for the vision kernel is 9 mW. Although a direct power comparison is difficult because of the different assumptions and stages of the implementation, a comparison with recent work regarding background subtraction presented in [35], which reported a value of 1.766 mW, shows that the proposed architecture offers much higher power efficiency.

3.5.5 Acceleration Factor

An embedded system that uses a second chip on the platform as an accelerator can be quantitatively evaluated in terms of the acceleration factor, which is defined based on the time needed for the host processor to execute a single operation compared with the execution of the same operation when processed by the accelerator. In this work, as discussed previously, one of the most resource-consuming operations is the LBP computation; therefore, we studied the acceleration factor for this operation for a single frame. The measurement circuit is illustrated in figure 3.3. The circuit consists of a timer that is coded in the PL block. For the first measurement, the ARM processor transfers the contents of a single frame to the cache, and before the LBP command is performed, it triggers the timer. For the second time measurement, the

same operation is executed by the PL block. For a single frame at HD resolution, we measured an acceleration factor of 300. This testing algorithm for measuring the acceleration factor was proposed by Xilinx [68] for the evaluation of hardware acceleration. The results of a performance comparison between the proposed embedded system and a software-based system are summarized in Table 3.6. The software implementation was realized with 1 thread and 16 threads using Visual Studio 2013 with Boost 1.53.0. The results based on the performance metric show that 66.8% of the operations are associated with the main function, and the remainder are performed as part of the initialization to run the software on the operating system. The main function consists of the following tasks: *GetNextImage*, *InitBGS*, *Process*, and *GetForegroundImage* or *GetMaskImage*. Figure 3.10 summarizes each task and the functions called in each task. According to the results of the profiler, 56% of the operations are dedicated to the read frame. This operation is more efficiently implemented on the embedded platform using VDMA. In addition, individual sequences of all of the tasks and functions for the *Process* task (*ComputeLBP*, *DistanceMeasurement*, and *UpdateBGS*), which need to be implemented sequentially, can be run in parallel. Overall, our SoC-FPGA-based implementation achieves a speedup of up to 11.24x over a software implementation with 1 thread. It also achieves a 9.04x speedup over a software implementation with 16 threads.

Table 3.6 – Performance comparison with a 3.40 GHz CPU, in ms

Function	1 thd	16 thd	SoC-FPGA
GetNextFrame	30	30	1.86
Process	12	4.3	0.96
GetForegroundImage	5	5	1.36
Total	**47**	**39.3**	**4.18**

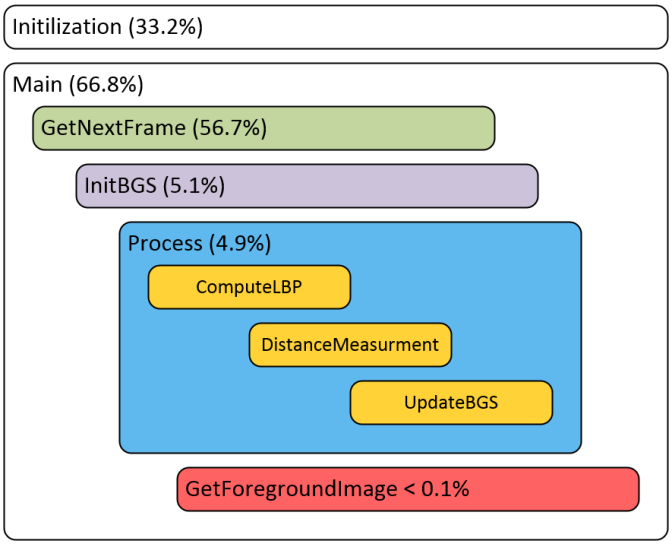


Figure 3.10 – Software performance metrics.

3.6 Conclusion

In this chapter, a SoC-FPGA-based video processing system is presented. The proposed algorithm employs parallel processing and a pipelined architecture to perform real-time foreground and background identification. In addition, a technique for traffic separation is proposed to enable easy implementation and execution of the adaptive algorithm on the embedded platform. The parameters of the algorithm can be modified and updated without interfering with the main operation. For example, the threshold value for the distance measure can be changed while the entire process is being performed. The system can be used with 2D and 3D data, and it can be applied to low-resolution videos. The experimental results for all tested datasets show that the system can be used in the pre-processing step for most embedded vision applications, such as face recognition, object tracking and human action and activity recognition, to remove unimportant parts of a scene to enable better processing.

In addition, the proposed architecture is analyzed to identify the required number of bits for the kernel to allow the system to operate at a high quality and bandwidth. The results of the power consumption analysis show that the proposed architecture

can be implemented for most embedded vision applications based on battery-powered devices. Since the Zynq platform has two ARM processors, future work could investigate the use of both ARM processors in addition to the accelerator to improve performance: one processor could manage the control signals, and the other processor could transfer raw data (frames) to the VDMA. However, this would increase the total power consumption of the system. The motivation for this work was to develop a system for human action recognition, and foreground and background identification could be one of the first pre-processing stages for this task. By identifying the unimportant parts of a scene (background), we can extract better and more important features, thereby improving the recognition accuracy.

3.7 Summary

This chapter presents the hardware acceleration of a real-time adaptive background and foreground identification algorithm in a SoC, including the capture, processing and display stages.

The foreground and background subtraction module is the one of the main components of the human action and activity recognition framework; in this framework, it is considered to constitute the first stage of pre-processing. The framework is discussed in more detail in chapter 1. This module produces a mask image that defines pixels with rich information. Foreground and background subtraction are common components in most machine vision frameworks and algorithms. Background subtraction algorithms required high computational effort. Moreover, with the popularization of high-definition video, the amount of data available to be processed has also increased substantially, posing massive computational and communication demands, which make the implementation and operation of such algorithms in embedded systems more challenging. Hardware acceleration through specialization has thus received renewed interest in recent years; such acceleration has generally been

implemented using two chips, with the image signal processing (ISP) being performed by a DSP, a GPU or an FPGA and the video content analytics (VCA) being executed by a processor. GPUs consume a substantial amount of power; thus, it is challenging to deploy them in embedded environments. However, the new generation of SoC-FPGAs, which are fabricated with both a microprocessor and an FPGA on a single chip, consume less power and can be built into smaller systems, thereby offering an attractive platform for embedded applications.

The algorithm discussed in this chapter can be performed in either 2D or 3D space. The proposed module uses photometric invariant color, depth data and local binary patterns (LBPs) to distinguish background from foreground. The system uses minimal cell resources, an elastically pipelined architecture is used to compensate for variations in processing time, and each pipeline stage is optimized to use the available FPGA primitives. Additionally, the communication-centric architecture used in this approach simplifies the implementation of embedded vision algorithms.

The next chapter discusses a human detection module that serves as the second stage of preprocessing in order to restrict the main processing to areas in which humans are present. The output of the foreground and background subtraction module provides the foreground probability that is used as one of the elements of the feature vector for human detection.

References

[1] B. D. Technology, "Implementing vision capabilities in embedded systems," 2012.

[2] B. Senouci, I. Charfi, B. Heyrman, J. Dubois, and J. Miteran, "Fast prototyping of a soc-based smart-camera: a real-time fall detection case study," *Journal of Real-Time Image Processing*, vol. 12, no. 4, pp. 649–662, 2016.

[3] P.-J. Lapray, B. Heyrman, and D. Ginhac, "Hdr-artist: an adaptive real-time smart camera for high dynamic range imaging," *Journal of Real-Time Image Processing*, vol. 12, no. 4, pp. 747–762, 2016.

[4] E. Publications:. (2014) Video surveillance market: global forecast and analysis 2011-2016. [Online]. Available: https://www.electronics.ca

[5] Y. Yang and Q. J. Wu, "Multilayer extreme learning machine with subnetwork nodes for representation learning," *IEEE transactions on cybernetics*, vol. 46, no. 11, pp. 2570–2583, 2016.

[6] J. Cao, Z. Lin, G.-B. Huang, and N. Liu, "Voting based extreme learning machine," *Information Sciences*, vol. 185, no. 1, pp. 66–77, 2012.

[7] J. Cao, W. Wang, J. Wang, and R. Wang, "Excavation equipment recognition based on novel acoustic statistical features," *IEEE Transactions on Cybernetics*, vol. PP, no. 99, pp. 1–13, 2017.

[8] S. Hadfield and R. Bowden, "Hollywood 3d: Recognizing actions in 3d natural scenes," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, June 2013, pp. 3398–3405.

[9] O. Oreifej and Z. Liu, "Hon4d: Histogram of oriented 4d normals for activity recognition from depth sequences," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, June 2013, pp. 716–723.

[10] Zedboard. (2016) Avnet product brief. [Online]. Available: http://zedboard.org

[11] C. V. (2016) Cyclone v soc development kit. [Online]. Available: https://www.altera.com

[12] C. Ridder, O. Munkelt, and H. Kirchner, "Adaptive background estimation and foreground detection using kalman-filtering," in *Proceedings of International Conference on recent Advances in Mechatronics*. Citeseer, 1995, pp. 193–199.

[13] K. Karmann and A. V. Brandt, "Moving object recognition using an adaptive background memory," in *Time-Varying Image Processing and Moving Object Recognition*. V. Capellini, Ed. Amsterdam, The Netherlands: Elsevier, 1990, pp. 289–307.

[14] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers, "Wallflower: Principles and practice of background maintenance," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 1. IEEE, 1999, pp. 255–261.

[15] J. C. S. Jacques, C. R. Jung, and S. R. Musse, "Background subtraction and shadow detection in grayscale video sequences," in *XVIII Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'05)*. IEEE, 2005, pp. 189–196.

[16] C.-C. Chiu, M.-Y. Ku, and L.-W. Liang, "A robust object segmentation system using a probability-based background extraction algorithm," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 4, pp. 518–528, 2010.

[17] T. Horprasert, D. Harwood, and L. S. Davis, "A statistical approach for real-time robust background subtraction and shadow detection," 1999, pp. 1–19.

[18] R. Rodriguez-Gomez, E. J. Fernandez-Sanchez, J. Diaz, and E. Ros, "Fpga implementation for real-time background subtraction based on horprasert

model," *Sensors*, vol. 12, no. 1, p. 585, 2012. [Online]. Available: http://www.mdpi.com/1424-8220/12/1/585

[19] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland, "Pfinder: real-time tracking of the human body," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 780–785, Jul 1997.

[20] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, vol. 2. IEEE, 1999.

[21] D.-S. Lee, "Effective gaussian mixture learning for video background subtraction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 5, pp. 827–832, 2005.

[22] O. Tuzel, F. Porikli, and P. Meer, "A bayesian approach to background modeling," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)-Workshops*. IEEE, 2005, pp. 58–58.

[23] M. Heikkila and M. Pietikainen, "A texture-based method for modeling the background and detecting moving objects," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 4, pp. 657–662, 2006.

[24] J.-S. Hu and T.-M. Su, "Robust background subtraction with shadow and highlight removal for indoor surveillance," *EURASIP Journal on Applied Signal Processing*, vol. 2007, no. 1, pp. 108–108, 2007.

[25] P. Carr, "Gpu accelerated multimodal background subtraction." in *DICTA*, 2008, pp. 279–286.

[26] V. Pham, P. Vo, V. T. Hung *et al.*, "Gpu implementation of extended gaussian mixture model for background subtraction," in *Computing and Communication*

Technologies, Research, Innovation, and Vision for the Future (RIVF), 2010 IEEE RIVF International Conference on. IEEE, 2010, pp. 1–4.

[27] J. Xu, W. Wolf, J. Henkel, S. Chakradhar, and T. Lv, "A case study in networks-on-chip design for embedded video," in *Proceedings of the conference on Design, automation and test in Europe-Volume 2.* IEEE Computer Society, 2004, p. 20770.

[28] T. Lv, J. Xu, W. Wolf, I. B. Ozer, S. T. Chakradhar *et al.*, "A methodology for architectural design of multimedia multiprocessor socs," *IEEE design & test of computers*, no. 1, pp. 18–26, 2005.

[29] G. Chen, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and W. Wolf, "Energy savings through compression in embedded java environments," in *Proceedings of the tenth international symposium on Hardware/software codesign.* ACM, 2002, pp. 163–168.

[30] K. Swaminathan, G. Lakshminarayanan, and S.-B. Ko, "High speed generic network interface for network on chip using ping pong buffers," in *Electronic System Design (ISED), 2012 International Symposium on.* IEEE, 2012, pp. 72–76.

[31] K. Appiah and A. Hunter, "A single-chip fpga implementation of real-time adaptive background model," in *Proceedings. 2005 IEEE International Conference on Field-Programmable Technology, 2005.* IEEE, 2005, pp. 95–102.

[32] J. Schlessman, M. Lodato, B. Ozer, and W. Wolf, "Heterogeneous mpsoc architectures for embedded computer vision," in *2007 IEEE international conference on multimedia and expo.* IEEE, 2007, pp. 1870–1873.

[33] M. Wójcikowski, R. Żaglewski, and B. Pankiewicz, "Fpga-based real-time implementation of detection algorithm for automatic traffic surveillance sensor network," *Journal of Signal Processing Systems*, vol. 68, no. 1, pp. 1–18, 2012.

[34] M. Happe, E. Lübbers, and M. Platzner, "A self-adaptive heterogeneous multi-core architecture for embedded real-time video object tracking," *Journal of real-time image processing*, vol. 8, no. 1, pp. 95–110, 2013.

[35] K. Ratnayake and A. Amer, "Embedded architecture for noise-adaptive video object detection using parameter-compressed background modeling," *Journal of Real-Time Image Processing*, pp. 1–18, 2014.

[36] H. Jiang, H. Ardo, and V. Owall, "Hardware accelerator design for video segmentation with multi-modal background modelling," in *2005 IEEE International Symposium on Circuits and Systems*. IEEE, 2005, pp. 1142–1145.

[37] M. Genovese, E. Napoli, and N. Petra, "Opencv compatible real time processor for background foreground identification," in *2010 International Conference on Microelectronics*. IEEE, 2010, pp. 467–470.

[38] M. Genovese and E. Napoli, "Fpga-based architecture for real time segmentation and denoising of hd video," *Journal of Real-Time Image Processing*, vol. 8, no. 4, pp. 389–401, 2013.

[39] J. Yao and J.-M. Odobez, "Multi-layer background subtraction based on color and texture," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2007, pp. 1–8.

[40] K. Kim, T. H. Chalidabhongse, D. Harwood, and L. Davis, "Real-time foreground–background segmentation using codebook model," *Real-time imaging*, vol. 11, no. 3, pp. 172–185, 2005.

[41] R. Dömer, A. Gerstlauer, J. Peng, D. Shin, L. Cai, H. Yu, S. Abdi, and D. D. Gajski, "System-on-chip environment: A specc-based framework for heterogeneous mpsoc design," *EURASIP Journal on Embedded Systems*, vol. 2008, p. 5, 2008.

[42] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, vol. 2. Ieee, 2003, pp. 1398–1402.

[43] A. Safaei and Q. J. Wu, "Evaluating 3d hand motion with a softkinetic camera," in *Multimedia Big Data (BigMM), 2015 IEEE International Conference on*. IEEE, 2015, pp. 290–291.

[44] A. Safaei and M. Jahed, "3d hand motion evaluation using hmm," *Journal of Electrical and Computer Engineering Innovations*, vol. 1, no. 1, pp. 11–18, 2013.

[45] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 5, pp. 603–619, 2002.

[46] *ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC*, Xilinx, 9 2015, rev. 1.5.

[47] Xilinx. (2017) Petalinux tools. [Online]. Available: www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html

[48] ——. (2017) Wiki xilinx. [Online]. Available: www.wiki.xilinx.com/PetaLinux

[49] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: a local svm approach," in *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, vol. 3, Aug 2004, pp. 32–36 Vol.3.

[50] A. Vacavant, T. Chateau, A. Wilhelm, and L. Lequièvre, "A benchmark dataset for outdoor foreground/background extraction," in *Asian Conference on Computer Vision*. Springer, 2012, pp. 291–300.

[51] "Background Models Challenge (BMC)," http://bmc.iut-auvergne.com/, accessed: 2017-03-38.

[52] M. Blank, L. Gorelick, E. Shechtman, M. Irani, and R. Basri, "Actions as space-time shapes," in *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, vol. 2, Oct 2005, pp. 1395–1402 Vol. 2.

[53] J. Ferryman and A. Shahrokni, "Pets2009: Dataset and challenge," in *Performance Evaluation of Tracking and Surveillance (PETS-Winter), 2009 Twelfth IEEE International Workshop on*. IEEE, 2009, pp. 1–6.

[54] L. Patino, T. Cane, A. Vallee, and J. Ferryman, "Pets 2016: Dataset and challenge," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2016, pp. 1–8.

[55] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler, "Motchallenge 2015: Towards a benchmark for multi-target tracking," *arXiv preprint arXiv:1504.01942*, 2015.

[56] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler, "Mot16: A benchmark for multi-object tracking," *arXiv preprint arXiv:1603.00831*, 2016.

[57] M. Andriluka, S. Roth, and B. Schiele, "People-tracking-by-detection and people-detection-by-tracking," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1–8.

[58] F. Kristensen, H. Hedberg, H. Jiang, P. Nilsson, and V. Öwall, "An embedded real-time surveillance system: implementation and evaluation," *Journal of Signal Processing Systems*, vol. 52, no. 1, pp. 75–94, 2008.

[59] J. Hiraiwa, E. Vargas, and S. Toral, "An fpga based embedded vision system for real-time motion segmentation," in *Proceedings of 17th International Conference on Systems, Signals and Image Processing. Brazil*, 2010.

[60] M. Genovese, E. Napoli, and N. Petra, "Opencv compatible real time processor for background foreground identification," in *Microelectronics (ICM), 2010 International Conference on*, Dec 2010, pp. 467–470.

[61] M. Genovese and E. Napoli, "Fpga-based architecture for real time segmentation and denoising of hd video," *Journal of Real-Time Image Processing*, vol. 8, no. 4, pp. 389–401, 2013.

[62] S. Liao, G. Zhao, V. Kellokumpu, M. Pietikäinen, and S. Z. Li, "Modeling pixel process with scale invariant local patterns for background subtraction in complex scenes," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 1301–1306.

[63] E. Hayman and J.-O. Eklundh, "Statistical background subtraction for a mobile observer." in *ICCV*, vol. 1, 2003.

[64] Z. Zivkovic, "Improved adaptive gaussian mixture model for background subtraction," in *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, vol. 2. IEEE, 2004, pp. 28–31.

[65] Z. Zivkovic and F. Van Der Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction," *Pattern recognition letters*, vol. 27, no. 7, pp. 773–780, 2006.

[66] L. Li, W. Huang, I. Y. Gu, and Q. Tian, "Foreground object detection from videos containing complex background," in *Proceedings of the eleventh ACM international conference on Multimedia*. ACM, 2003, pp. 2–10.

[67] Y. Goya, T. Chateau, L. Malaterre, and L. Trassoudaine, "Vehicle trajectories evaluation by static video sensors," in *Intelligent Transportation Systems Conference, 2006. ITSC'06. IEEE*. IEEE, 2006, pp. 864–869.

[68] *A Zynq Accelerator for Floating Point Matrix Multiplication Designed with Vivado HLS*, Xilinx, 1 2016, rev. 2.0.

Chapter 4

System-on-Chip-Based Hardware Acceleration for Human Detection in 2D/3D Scenes

4.1 Introduction

Human detection is very important for many embedded computer vision (ECV) applications, such as surveillance and security systems, that human action and activity recognition. Most previously proposed methods follow four steps: first, a number of spatiotemporal locations are sampled. Then, a local feature extractor extracts features around points of interest, and local features are encoded into a single comprehensive descriptor. Finally, a discriminative classifier is employed to find the most likely category. Since hardware resources (cells) are limited, the implementation of all these steps is challenging and becomes more challenging when an algorithm is required for real-time applications. Although the new generation of system-on-chip (SoC) field gate programmable arrays (FPGA) has more resources, finding an effective balance between speed (real time) and stability is still difficult. To address this problem, preprocessing is required to identify the point of interest in the frame and reject unimportant parts, e.g., detecting the part of the scene in which humans are present and removing the remaining parts.

This chapter presents an embedded computer vision system for human detection, which is part of the overall framework for developing an embedded vision system for recognizing human actions. We define two preprocessing steps for action recognition:

the first is foreground and background identification (Figure 4.1), which was discussed in chapter 3. Then, based on the results of this first step, the second module identifies humans present in the scene. The feature extraction process and the classifier used after the second preprocessing module to identify the type of action will be covered in the next chapter. The proposed method for human detection takes advantage of covariance mapping and an integral image in a sub-region of the entire 2D/3D image feature space. The video content is fused from the spatial, temporal, and depth domains to consider appearance, foreground, and depth data. The proposed implementation runs on the Xilinx SoC Zynq-7000 (XC7Z020). The Zynq-7000 integrates a dual core ARM processor and an Artix-7 FPGA in a single device. The ARM processor has on-chip and external memory interfaces and supports peripheral connectivity. The processor reads raw data from the USB interface or SD card storage and passes it to the programmable logic. The detection algorithm that is implemented in the programmable logic unit is applied to the raw data.

The remainder of this chapter is organized as follows. Previous work related to human detection and the proposed algorithm are discussed in Sections 4.2 and 4.3, respectively. The design of the video processing modules for the embedded video processing algorithm is presented in Section 4.4, and the implementation and evaluation results are discussed in Section 4.5. The conclusions are presented in Section 4.6.

4.2 Related Work

Human detection is the one of the key components of most computer vision applications. It is always a complicated problem because of human body articulation, clothing color, and environmental conditions such as light, shadow and oscillation. Most algorithms require a complete view of each human subject, and sometimes, when part of the body is occluded by another object, such an algorithm fails to recognize a human who is present in a scene.

Human detection algorithms can be grouped into two categories. Algorithms of the first type are called shape models. In such a model, the parts of the body are represented by combinations of joints in a geometric model, and classifiers such as AdaBoost are trained on these body parts and are then employed for human detection. The main drawback of this approach is that it is not applicable to low-resolution frames. In addition, this approach involves many mathematical operations, which impedes its implementation speed.

Methods of the second type are called appearance methods. In such a method, a scene is divided into sub-regions, and a separate detector is applied to each sub-region. Most recent approaches, in the forms of both algorithms and system-level implementations, attempt to use appearance-based methods [1] [2] [3]. Appearance methods rely on extracted features that are encoded with feature descriptors and employ classification methods to identify humans.

Among the available feature descriptors that have been proposed, local self-similarities [4], contour-based methods [5] [6], gradient-based methods (HOG) [7] [8] and Haar-like features are all commonly used for human detection. Among the available classification algorithms, principal component analysis (PCA), support vector machine (SVM) classifiers, and combined learning approaches such as AdaBoost are widely used for human detection. Experimental results prove that among the available feature descriptors, HOG descriptors tend to offer the best classification performance. Human identification on the basis of shape texture information was discussed in [9] [10]. The results of the studies presented in [11] [9] [10] indicate that an appearance-based method with HOG descriptors outperforms the shape texture approach.

Various approaches based on different descriptors and classifiers have been proposed. In [12], Haar wavelet descriptors were used with a support vector machine (SVM) classifier. The method proposed in [13] is based on a densely sampled histogram of gradients descriptor and an SVM classifier, whereas in [14], a histogram of optical flows is used. The algorithms used in these approaches yield acceptable re-

sults; however, their computation processes are too complicated to implement directly using a simple FPGA circuit.

Image and video processing applications typically require considerable data processing and sometimes complex mathematical operations, requiring a high-performance CPU or even a multi-core system. However, recent portable vision applications, which have strict requirements regarding power efficiency, cost, and stability, provide designers with an embedded solution. At present, the options for implementing computer vision algorithms on hardware include DSPs, FPGAs, mobile PC processors, and SoC FPGAs. Of course, each of these implementations is highly application-dependent. However, most embedded vision implementations for human detection involve the following tasks: (a) image acquisition, to capture input frames; (b) preprocessing, to reduce noise and generate a mask image to identify the region of interest (RoI); (c) feature extraction and encoding; and (d) high-level processing or recognition analysis for human detection and identification.

4.3 Algorithm

As previously mentioned, because of the resource limitations of embedded systems, an algorithm that requires few mathematical operations is required. In our proposed method, we use covariance mapping in each sub-region of the 2D/3D image feature space. These sub-regions are defined by the foreground and background identification module discussed in chapter 3.

4.3.1 Foreground Identification

The output of the foreground and background identification module specifies the area of the screen that is rich in information. The operation of this module, as discussed in the previous chapter, is based on the foreground probability and a threshold value. By defining a specific threshold, unimportant parts can be identified and removed

from the scene. Figure 4.1 presents the result of the first stage of preprocessing. The output of this module is a foreground probability value that is used as one of the elements of the feature vector, as discussed in Section 4.3.2, and restricts the movement of the detection window to the area (ROI) defined by the mask image.

4.3.2 Covariance Descriptors

Efficient detection algorithms rely on feature selection. Good features should be informative and discriminative. However, in an embedded system application, hardware resources are limited. Therefore, the detection algorithm should be computationally efficient and require few resources. In this study, we use a covariance matrix to fuse multiple features and employ an integral image technique to achieve fast computation with fewer resources. Consider pixel p in image I in a 3D space with dimension $W \times H$. We can extract each pixel location $p = (\mathrm{x}, \mathrm{y}, \mathrm{z})$, where z represents depth information. Let f represent the defined features for each point; thus, the total number of feature vectors is $W \times H \times f$. The feature vector f is defined as follows:

$$\begin{aligned} f \quad = \{ \quad & |\nabla G_x|, |\nabla G_y|, \sqrt{\nabla G_x^2 + \nabla G_y^2}, arctan\frac{|\nabla G_y|}{|\nabla G_x|} \\ & |\nabla D_x|, |\nabla D_y|, \sqrt{\nabla D_x^2 + \nabla D_y^2}, arctan\frac{|\nabla D_y|}{|\nabla D_x|} \\ & F_P \quad \} \end{aligned} \tag{4.1}$$

where ∇G_x and ∇G_y are the first-order intensity derivatives of the horizontal and vertical changes, respectively, and $\arctan\frac{|\nabla G_y|}{|\nabla G_x|}$ is the orientation of the edge. F_P is the foreground probability value for each point. The probability value is computed using the foreground and background identification modules. ∇D_x and ∇D_y are the first-order depth derivatives of the horizontal and vertical changes, respectively. In the most of the previously proposed methods, the authors used the second-order derivatives results to make the results more informative. However, since the primary concern with the hardware implementation is the reduction of the computation oper-

ation, we use the results of the foreground identification module, which identify the ROI. Figure 4.1 shows the results of foreground and background identification of the three input streams. There are three scenarios: only a human, only a car on a highway, and both a human and a car. This preprocessing step restricts the result of the human detection operation to the area that has the maximum probability foreground value. By identifying the ROI, the covariance matrix C_R and mean vector μ_R can be defined as follows:

$$\begin{aligned} C_R &= \frac{1}{|R|-1}\sum_{x\in R}(f(x)-\mu)((f(x)-\mu)^T \\ \mu_R &= \frac{1}{R}\sum_{x\in R}(f(x)) \end{aligned} \tag{4.2}$$

An integral image technique is used to improve the speed of the covariance computation. This technique involves intermediate image representation for fast computation of the ROI. Each pixel of the ROI is the sum of all pixels inside the upper left corner of the region of interest and the pixel of interest.

$$\text{Integral Image} = \sum_{x<x\prime,y<y\prime} I(x,y) \tag{4.3}$$

The (x, y) element in equation 4.2 can be rewritten as follows:

$$C_R(x,y) = \frac{1}{|R|-1}\sum_{x,y\in R}(f(x)-\mu(x))((f(y)-\mu(y))^T \tag{4.4}$$

By expanding the mean vector, equation 4.4 can be rewritten as follows:

$$\begin{aligned} C_R(x,y) = \frac{1}{|R|-1}\Bigg[&\sum_{x,y\in R}(f(x)f(y)) \\ &-\frac{1}{n}\sum_{x\in R}f(x)\frac{1}{n}\sum_{y\in R}f(y)\Bigg] \end{aligned} \tag{4.5}$$

(a) (b) (c)

Figure 4.1 – Input Stream (a), foreground probability (b) and mask image (c).

According to equation 4.5, we multiply two features (first term = Q) and the sum of each feature (second term = P) to compute the covariance matrix.

$$P(x\prime, y\prime, i) = \sum_{x<x\prime, y<y\prime} f(x,y,i) \quad i = 1 \ldots f \tag{4.6}$$

$$Q(x\prime, y\prime, i, j) = \sum_{x<x\prime, y<y\prime} f(x,y,i) f(x,y,j) \quad i,j = 1 \ldots f \tag{4.7}$$

$$P_{x,y} = \left[P(x,y,1) \ldots P(x,y,f)\right]^T \tag{4.8}$$

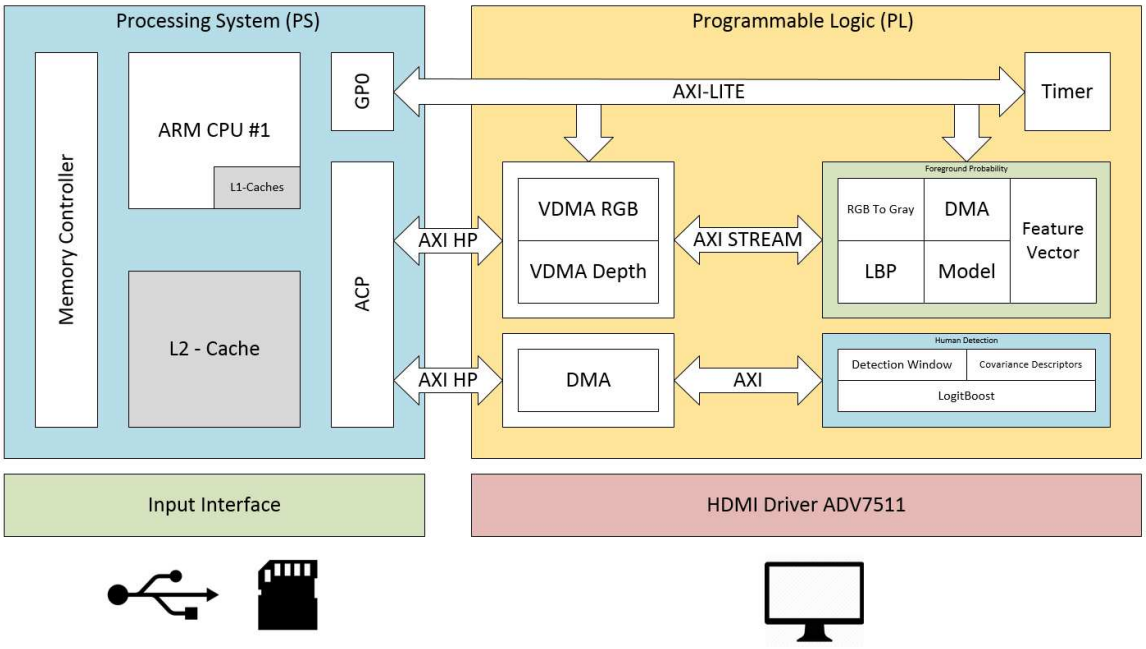


Figure 4.2 – Block diagram of the proposed human detection system.

$$Q_{x,y} = \begin{bmatrix} Q_{x,y,1,1} & \cdots & Q_{x,y,1,f} \\ & \vdots & \\ Q_{x,y,f,1} & \cdots & Q_{x,y,f,f} \end{bmatrix} \tag{4.9}$$

The computation result in [15] shows that by constructing an integral image, the covariance of any ROI can be computed in O(d^2) time. Considering the region bounded by (1,1) and (x', y'), the covariance could be defined as follows:

$$C_{R(1,1,x',y')} = \frac{1}{|n|-1}\left[Q_{x',y'} - \frac{1}{n}P_{x',y'}P^T_{x',y'}\right] \tag{4.10}$$

Since the covariance matrix is not a vector, we use the method proposed in [16]. This method uses Riemannian geometry, which focuses on the space of symmetric positive definite matrices. In this approach, the covariance matrix of the feature vector is

mapped into Euclidean tangent space.

$$h : \mathrm{X} \rightarrow \mathrm{x} \quad = \quad h(\mathrm{X}) \quad h(\mathrm{x}) = \mathrm{vec}_{\mu_l}(\log_{\mu_l}(\mathrm{X})) \tag{4.11}$$

$$\mathrm{vec}_z(\mathrm{y}) \quad = \quad \mathrm{upper}(\mathrm{Z}^{-\frac{1}{2}}\mathrm{y}\mathrm{Z}^{-\frac{1}{2}}) \tag{4.12}$$

$$\log \mathrm{Z} \quad = \quad \mathrm{Z}^{\frac{1}{2}} \log(\mathrm{Z}^{-\frac{1}{2}}\mathrm{y}\mathrm{Z}^{-\frac{1}{2}})\mathrm{Z}^{\frac{1}{2}} \tag{4.13}$$

$$\log(\Sigma) \quad = \quad \mathrm{U} \log(\mathrm{D})\mathrm{U}^T \rightarrow \quad \Sigma = \mathrm{UDU}^{\mathrm{T}} \tag{4.14}$$

Here, x is a point in the vector space x$\in R^m$, X is a point on the Riemannian manifold X$\in M$, Σ is the eigenvalue decomposition of the symmetric matrix, log(D) is a diagonal matrix, and *upper* refers to the upper triangular part of the matrix.

4.3.3 LogitBoost Algorithm

LogitBoost is a machine learning algorithm that applies established logistic regression techniques to the AdaBoost method [17]. LogitBoost combines multiple weak classifiers that return true or false. The output is used to construct strong classifiers. In the detection step, we use the LogitBoost algorithm that was trained on the features extracted from the ROI and then compare it with the input image feature to detect a human.

4.4 Hardware Implementation

A block diagram of the SoC FPGA implementation of the human detection algorithm is illustrated in Figure 4.2. The details of the implementation of each module is discussed in this section. All the modules were written in VHDL and synthesized with a Vivado 2015.2. The software executed by the ARM processor was written in C and compiled with the Xilinx SDK 2015.2.

Algorithm 2 LogitBoost

1: $F_{i,k} \leftarrow 0$

2: $p_{i,k} \leftarrow \frac{1}{K}$

3: $k \leftarrow 0 \cdots K-1$

4: $i \leftarrow 1 \cdots N$

5: **if** $y_i = k$ **then**

6: $\quad r_{i,k} = 1$

7: **else**

8: $\quad r_{i,k} = 0$

9: **end if**

10: **for** $m \leftarrow 1$ **to** M **do**

11: $\quad$ **for** $k \leftarrow 0$ **to** $K-1$ **do**

12: $\quad\quad \omega_{i,k} = p_{i,k}(1-p_{i,k})$

13: $\quad\quad z_{i,k} = \frac{r_{i,k}-p_{i,k}}{p_{i,k}(1-p_{i,k})}$

14: $\quad\quad F_{i,k} = F_{i,k} + \frac{K-1}{k}(f_{i,k} - \frac{1}{K}\sum_{k=0}^{K-1} f_{i,k})$

15: $\quad$ **end for**

16: $\quad p_{i,k} = \frac{\exp F_{i,k}}{\sum_{s=0}^{K-1} \exp F_{i,k}}$

17: **end for**

4.4.1 Input Frame

In this implementation, we use a depth camera (SoftKinect) [18] that can simultaneously generate RGB and depth data. This camera also includes a UV map generator that can merge depth and RGB data without calibration. The camera connects directly to the USB interface on the Xilinx board. We also use videos stored on an SD card. The processor system (PS) unit transfers the RGB and depth data to the two block VDMAs, which are accessible by the programmable logic (PL) unit.

4.4.2 The AMBA AXI Interface

The SoC-FPGA that we use in this work is interfaced with an AMBA AXI version 4 to interconnect the processor with programmable logic and vice versa. The AXI interface consists of four 64/32-bit high-performance AXI (HP AXI) slave interfaces, two 32-bit AXI master interfaces, two 32-bit AXI slave interfaces, a 64-bit AXI accelerator coherency port (ACP) and an extended multiplexed I/O (EMIO) interface. In this study, we use 2 HP AXI interfaces for stream traffic that connect the processor to the video processing and video capture modules that require memory-mapped communication. To control each block, an AXI master interface is used to connect the modules to the processor.

4.4.3 Video Processing

The raw data is first converted from RGB to grayscale, and the RGB results of each pixel are stored in a line buffer (depth, 32 bits) that contains 24 bits of RGB data and 8 bits of luminance data. The luminance calculation requires a clock cycle. The output of the RGB to grayscale function is transferred to the LBP module to identify the foreground probability. The details of the hardware implementation with an LBP with six neighbors are discussed in [19] and chapter three. If pixels do not satisfy the probability threshold, they will be masked (False), and pixels that satisfy

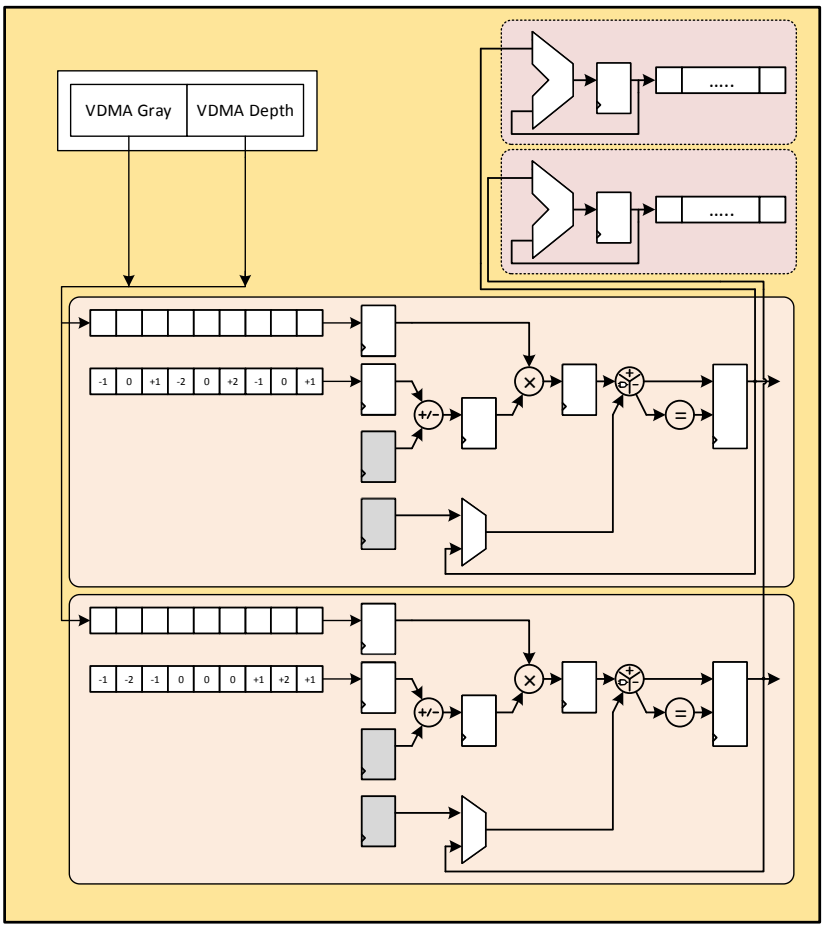


Figure 4.3 – RTL level first-order derivatives computation.

the threshold are unmasked (True). The masked image is stored in the RAM block used to identify the ROI. To determine the first-order derivatives, we use three lines of a 3-stage shift register that contains the pixels of interest with eight neighbors and a Sobel operator applied to all nine pixels to determine the gradient in both the horizontal and vertical directions (Figure 4.3). To calculate a square root, we use the square and square root operations implemented in the arithmetic unit of the VIVADO IP generator, which requires a single clock cycle for execution. To avoid a resource-heavy division operation for determining θ, we use the quantized gradient orientation. By defining the quantization number d, the threshold is given

by $Q_d = tan^{-1}((112 - 32d) \times \pi)$. Consequently, the value of the quantized gradient orientation is determined as follows:

$$\begin{aligned} Q_{d+1} \leq arctan\frac{|\nabla G_y|}{|\nabla G_x|} \leq Q_d \\ tan\, Q_{d+1} \leq \frac{|\nabla G_y|}{|\nabla G_x|} \leq tan\, Q_d \\ |\nabla G_x| \times tan\, Q_{d+1} \leq |\nabla G_y| \leq |\nabla G_x| \times tan\, Q_d \end{aligned} \tag{4.15}$$

The feature vector results for pixels encoded by the covariance matrix were explained in Section 4.3.2, and as discussed, we used the feature of the integral image technique to improve the computation speed (equations 4.3, 4.4 and 4.5).

The results of the covariance matrix stored in the RAM block are used for human detection by the LogitBoost classifiers, which are pre-generated by offline learning. The weak classifiers are generated by the features that appear frequently in human body data, called positive samples, and features that appear infrequently in the images, called negative samples. FN, which is the feature vector in the ROI, is obtained from N-th training times. The strong classifier can be generated using the results of the weak classifiers as follows:

$$\mathrm{S_{Classifier}} = \{\mathrm{W_{Classifier\ 1}}, \mathrm{W_{Classifier\ 2}}, \cdots, \mathrm{W_{Classifier\ n}}\} \tag{4.16}$$

The last step uses a detection window in the input image to compare the features of the input image with the trained value to detect a human. Here, we use the results of the ROI, which is determined using the foreground probability results. Rather than moving the detection window in the whole scene, it is only moved within the region defined by the masked image.

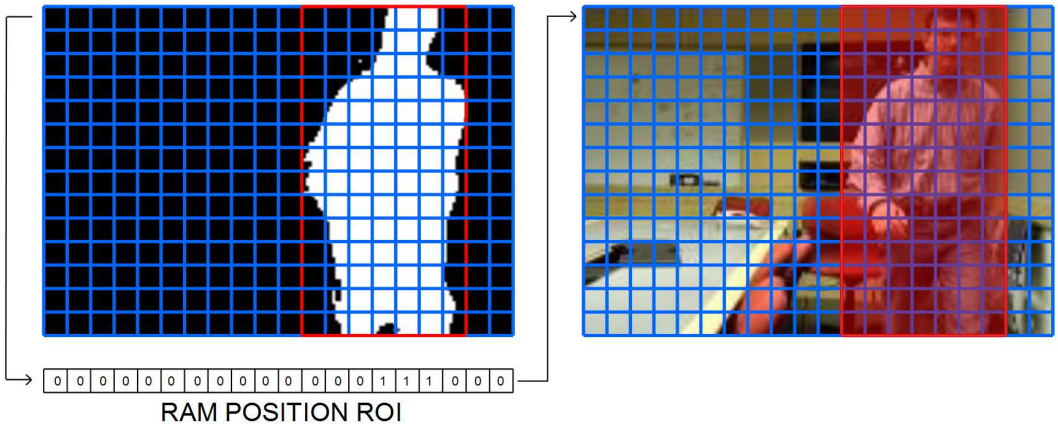


Figure 4.4 – The masked image and ROI.

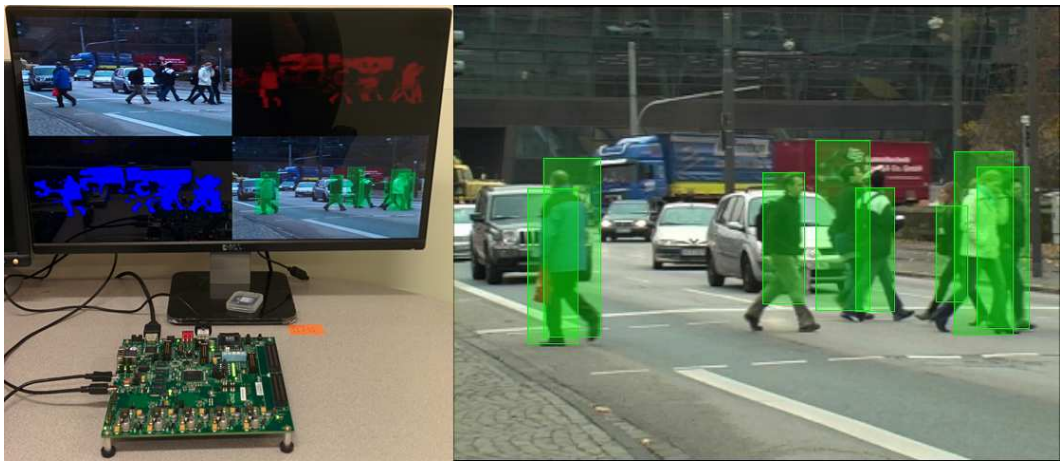


Figure 4.5 – System illustration (left) and the result of human detection (right).

4.5 Implementation Results and Evaluation

The human detection process described in Section 4.3 was implemented on a Xilinx XC7Z020 board equipped with an XC7Z020-1CLG484C device. Table 4.1 summarizes the results of the implementation and compares them to those of previous implementations. We tested the proposed system with two modern datasets with depth data that are used for human action and activity recognition, i.e., Hollywood [20] and HON4D [21]. We tested our hardware using the NICTA [22] pedestrian database for benchmarking. The results of the comparison yielded an 83.4 % detection rate. By

(a) (b) (c)

Figure 4.6 – Input Stream (a), foreground probability (b) and the result of human detection (c).

including depth information, the algorithm could distinguish between shadows and humans in the scenes (Figure 4.6). Table 4.2 summarizes the results of the performance analysis for the proposed implementation on hardware- and software-based systems. The software-based implementation was compiled with visual studio 2013 with boost 1.53.0 and was tested in 1-thread and 16-thread environments. Comparing the software- and hardware-based implementations shows that our proposed system achieves speed improvements of up to 20.5x in 1-thread and 6.72x in 16-thread environments.

Table 4.1 – Utilized Resources

Target FPGA	LUT	Flip Flop	DSP48E	BRAM
Altera Stratix II [23]	37940	66990	120	N.A
Altera Cyclone IV [24]	34403	N.A	N.A	348
Spartan 3 [25]	28616	N.A	N.A	100
Virtex 5 [26]	17383	2070	N.A	36
Proposed System (Zynq)	**30235**	**15017**	**80**	**126**

Table 4.2 – Performance analysis; Software-based implementation on Intel CPU (3.40 GHz) and SoC FPGA

Module	1-Thread	16-Thread	**SoC FPGA**
Foreground Probability	40ms	14ms	**1.91ms**
Human Detection	77ms	24.30ms	**3.79ms**
Total	**117ms**	**38.3ms**	**5.7ms**

4.6 Conclusion

In this chapter, an SoC FPGA-based video processing system is presented. The proposed algorithm employs parallel processing and pipeline architecture to perform human detection. The system can be used with both 2D and 3D data, and it works with low-resolution video data. The experimental results for all tested datasets indicate that the proposed method can be used as a preprocessing stage in human action recognition algorithms to remove unimportant parts of a scene.

4.7 Summary

A system-on-a-chip field-programmable gate array (SoC-FPGA)-based video processing platform for human detection in complex scenes is discussed. This chapter details the hardware-based implementation of a human detection algorithm for 2D/3D scenes, including the capture, video processing, and display stages. The proposed method is implemented by extending a previously proposed method that uses features extracted from the Riemannian manifold of region covariance matrices computed from 2D data. The proposed method considers both 2D and depth data (3D). The LogitBoost classifier is employed to detect humans. The proposed implementation uses minimal resources and employs a pipeline technique for better performance

and operation.

References

[1] D. Geronimo, A. M. Lopez, A. D. Sappa, and T. Graf, "Survey of pedestrian detection for advanced driver assistance systems," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 7, pp. 1239–1258, 2010.

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

[3] P. Viola, M. J. Jones, and D. Snow, "Detecting pedestrians using patterns of motion and appearance," in *null*. IEEE, 2003, p. 734.

[4] E. Shechtman and M. Irani, "Matching local self-similarities across images and videos," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE, 2007, pp. 1–8.

[5] M. Leordeanu, M. Hebert, and R. Sukthankar, "Beyond local appearance: Category recognition from pairwise interactions of simple features," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE, 2007, pp. 1–8.

[6] J. Shotton, A. Blake, and R. Cipolla, "Multiscale categorical object recognition using contour fragments," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 7, pp. 1270–1281, 2008.

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

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

[9] M. Enzweiler and D. M. Gavrila, "Monocular pedestrian detection: Survey and experiments," *IEEE transactions on pattern analysis and machine intelligence*, vol. 31, no. 12, pp. 2179–2195, 2009.

[10] D. M. Gavrila and S. Munder, "Multi-cue pedestrian detection and tracking from a moving vehicle," *International journal of computer vision*, vol. 73, no. 1, pp. 41–59, 2007.

[11] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 304–311.

[12] C. Papageorgiou and T. Poggio, "A trainable system for object detection," *International Journal of Computer Vision*, vol. 38, no. 1, pp. 15–33, 2000.

[13] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. Computer Society Conf. on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, June 2005, pp. 886–893.

[14] N. Dalal, B. Triggs, and C. Schmid, "Human detection using oriented histograms of flow and appearance," in *Proceedings of the 9th European Conference on Computer Vision - Volume Part II*, ser. ECCV'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 428–441.

[15] O. Tuzel, F. Porikli, and P. Meer, "Region covariance: A fast descriptor for detection and classification," in *Proceedings of the 9th European Conference on Computer Vision - Volume Part II*, ser. ECCV'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 589–600.

[16] W. M. Boothby, *An introduction to differentiable manifolds and Riemannian geometry*. Gulf Professional Publishing, 2003, vol. 120.

[17] J. Friedman, T. Hastie, R. Tibshirani *et al.*, "Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)," *The annals of statistics*, vol. 28, no. 2, pp. 337–407, 2000.

[18] A. Safaei and Q. M. J. Wu, "Evaluating 3d hand motion with a softkinetic camera," in *Multimedia Big Data (BigMM), 2015 IEEE International Conference on*, April 2015, pp. 290–291.

[19] A. Safaei and Q. M. J. Wu, "A system-level design for foreground and background identification in 3d scenes," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2016, pp. 2571–2574.

[20] S. Hadfield and R. Bowden, "Hollywood 3d: Recognizing actions in 3d natural scenes," in *Proc. Computer Society Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2013, pp. 3398–3405.

[21] O. Oreifej and Z. Liu, "Hon4d: Histogram of oriented 4d normals for activity recognition from depth sequences," in *Proc. Computer Society Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2013, pp. 716–723.

[22] G. Overett, L. Petersson, N. Brewer, L. Andersson, and N. Pettersson, "A new pedestrian dataset for supervised learning," in *Proc. of IEEE Symp. on Intelligent Vehicles*, June 2008, pp. 373–378.

[23] R. Kadota, H. Sugano, M. Hiromoto, H. Ochi, R. Miyamoto, and Y. Nakamura, "Hardware architecture for hog feature extraction," in *2009 Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, Sept 2009, pp. 1330–1333.

[24] K. Mizuno, Y. Terachi, K. Takagi, S. Izumi, H. Kawaguchi, and M. Yoshimoto, "Architectural study of hog feature extraction processor for real-time object de-

tection," in *2012 IEEE Workshop on Signal Processing Systems*, Oct 2012, pp. 197–202.

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

[26] 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*, Dec 2011, pp. 1–8.

Chapter 5

System-Level Design for Human Action Recognition in 3D Scenes

5.1 Introduction

Significant technological breakthroughs in embedded vision have been achieved in recent years due to the development of technologies that enable the implementation of various complex image and video content analysis (VCA) algorithms with improved performance, higher efficiency, and reduced cost.

Among the applications of computer vision, human action and activity recognition is expected to become increasingly important and receive increasing attention due to its wide applicability. Rapid advancements in camera technology and memory storage capabilities have made high-resolution videos readily available to everyone and for various types of applications. The applications involving human action recognition can be divided in two main subcategories: applications in which real-time operation and recognition are essential and applications in which real-time operation is not mandatory. Among applications of the second type, video indexing is currently a common application for which operations can be performed offline. Applications in the first category include smart security and severance systems, care for elderly people and assistance for sick people. In these applications, real-time operations are essential.

The complexity of human action recognition algorithms is very high, and such algorithms typically involve suboperations that require massive resources for com-

putation and communication. These suboperations include image acquisition, image pre-processing, feature extraction and classification. Most of the currently proposed methods rely on software-based implementations. This means that all processing and classification operations are executed in a loop of sequential operations. Even if there are multiple cores available for processing, the operations must still be executed sequentially. The drawback of this method of implementation is its inherent delay in making fast and correct decisions, which might be critical depending on the type of application.

To cope with this problem, it is necessary to make the operations as parallel as possible in order to speed up their execution. Hardware acceleration has played a key role in the development of such systems by allowing some functions to be performed more efficiently than is possible with software running on a general-purpose processor.

To utilize a hardware accelerator, the tasks must be divided between the host processor and the accelerator device. Signal processing tasks, which require massive computational operations, are assigned to accelerators such as digital signal processors (DSP), graphics processing units (GPU) or field-programmable gate arrays (FPGAs), while video content analysis and control operations are assigned to the host processor.

The problem of the operation speed can be addressed by executing operations with parallel instruction code and utilizing a GPU; however, for applications in which integration and portability are important, such as smart security cameras, GPUs are not an efficient solution. In addition, the high power consumption of GPUs is another drawback for battery-powered devices. Another solution is to utilize a DSP as an accelerator; however, DSPs are efficient only for specific mathematical functions. Using an FPGA as an accelerator can provide both lower power consumption and the capability of parallel operations; however, for better integration, a new generation of SoC-FPGAs fabricated with both a microprocessor and FPGA on a single chip has been introduced. These SoC-FPGAs consume less power and can be integrated into smaller systems, thereby offering an attractive platform for embedded computer

vision applications.

This chapter presents a hardware acceleration scheme for a VCA application, i.e., human action and activity recognition in a SoC, including the capture, processing and display stages. The framework, as discussed before, consists of the following submodules: foreground probability estimation (foreground and background subtraction), human detection, and interest point detection via support vector machine (SVM) classification. The remainder of this chapter is organized as follows. In Section 5.2, we discuss the algorithms proposed for human action activity recognition and the advantages and disadvantages of each method. Additionally, we present related work on the implementation of vision algorithms in embedded systems. In Section 5.3, we introduce our framework and provide a mathematical description of the proposed framework. Our method of implementing the proposed algorithm is discussed in Section 5.4, and the resources utilized and the evaluation of the implementation are discussed in Section 5.5. Finally, in the last section, we present our conclusions.

5.2 Related Work

Human action recognition has recently become an active topic in the area of computer vision. Many algorithms have been proposed in recent years, and these methods can be classified as model-based or appearance-based methods. In the model-based methods, the pose of the human body is recovered from the input stream and the type of action is defined based on pose estimations. This approach is based on biologically plausible methods. Although the performance of this approach is acceptable and provides useful information, because variability exists in human motion, high-dimensional models, which are extremely challenging to implement in hardware, are required.

Appearance-based methods represent human action based on image features and motion information extracted from 2D and 3D image sequences. The advantages

of this approach are that it relies on only motion characteristics and it does not require a specific model for implementation. Moreover, the mathematical models of the appearance-based methods are easier to implement than those of the model-based methods. Separable-linear filters or spatio-temporal Harris corners are two popular examples of interest point detection. Descriptors are used to encode interest points in several ways, such as SIFT, SURF, FREAK, BRIEF, ORB, pixel gradients, and Jet descriptors. Background identification can also be included in terms of context by taking advantage of this approach to remove unimportant parts of an image.

Currently, embedded vision platforms are composed of DSPs, FPGAs, GPU, mobile PC processors, and SoC-FPGAs. Each of these implementations is highly application dependent; however, most embedded vision applications share some common modules:

Capture: The input stream that is captured by 2D/3D sensors and then transferred to memory. For applications that are not real time or that can work offline, the input stream can be transferred from stored memory, such as an SD card.

Low-level processing: Basic enhancement, such as noise reduction and lens distortion, is employed.

Interest point detection and Feature descriptor: Interest points, such as edges, corners, and blobs, are detected and encoded by feature descriptors.

Segmentation (optional): Segmentation depends on the type of application and is commonly used for biomedical imaging.

High-level processing: High-level processing provides the recognition result.

5.3 Algorithm

When streamed in a constrained environment or background, extracted features are likely to select robust features that are highly correlated with the event of interest. However, in most video streams, in which the environment and camera position

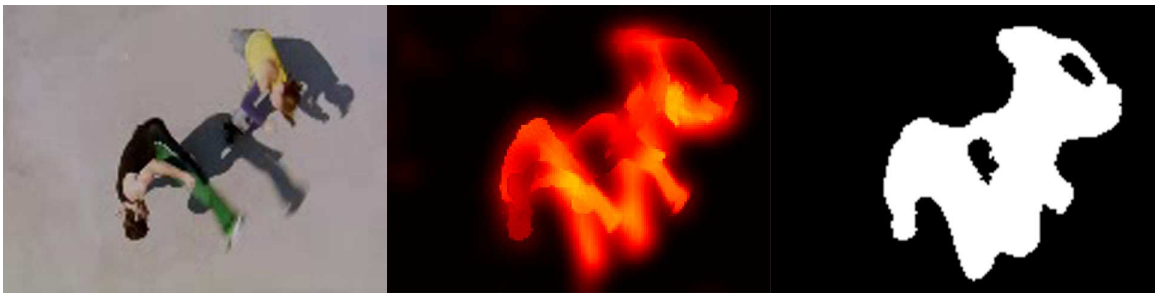


Figure 5.1 – Input stream (left), foreground probability (middle) and mask image (right).

Figure 5.2 – The results of human detection in three environments.

change, feature points are sparse. To address this problem, our methodology detects interest points in a region defined by two preprocessing steps, i.e., foreground and background identification and human detection. The output of these preprocessing steps identifies a region with rich information, and regions that do not meet the minimum probability are filtered out. The mathematical operations and hardware implementation of these two preprocessing steps have been presented in chapter 3 and 4 (Figures 5.1 & 5.2). Next, the detected points are encoded using a well-known technique, i.e., bag of visual words (BOW). Finally, SVM is employed to recognize the type of action.

5.3.1 Interest Point Detection

The results of the abovementioned preprocessing steps used to identify the ROI can also be used to extract interest points in the spatio-temporal domain. Here, we use ex-

tended detection algorithms based on the Harris corners method proposed by Laptev and Lindeberg [1] and the Hessian points method proposed by William et al. [2].

Harris Corners: The Harris corners method, which was extended into the spatio-temporal domain by Laptev and Lindeberg [1], is widely used for interest point detection in time-based applications. Hadfield et al. [3] [4] included depth data, which was determined based on equation 5.1. The interest point operator works in the spatio-temporal domain with strong intensity variations along three axes.

$$I_z = \frac{I_x}{D_x} + \frac{I_y}{D_y} + \frac{I_t}{D_t} \tag{5.1}$$

$$\mu = g(\sigma^2, \tau^2) * \begin{bmatrix} I_x I_x & I_x I_y & I_x I_t & I_x I_z \\ I_x I_y & I_y I_y & I_y I_t & I_y I_z \\ I_x I_t & I_y I_t & I_t I_t & I_t I_z \\ I_x I_z & I_y I_z & I_t I_z & I_z I_z \end{bmatrix} \tag{5.2}$$

$$H = det(\mu) - k\, trace^3(\mu) \tag{5.3}$$

Hessian Points: Williams et al. [2] defined a Hessian matrix as an extension of the method proposed by Beaudet [5]. The Hessian matrix was determined based on second-order intensity derivatives, where the strength of the interest point is defined by equation 5.4. The gradients along the z-axis, as defined in the Harris operator, are calculated using the relationship between the stream intensity and the depth.

$$\mu\prime = g(\sigma^2, \tau^2) * \begin{bmatrix} I_{xx} & I_{xy} & I_{xt} & I_{xz} \\ I_{xy} & I_{yy} & I_{yt} & I_{yz} \\ I_{xt} & I_{yt} & I_{tt} & I_{tz} \\ I_{xz} & I_{yz} & I_{tz} & I_{zz} \end{bmatrix} \tag{5.4}$$

5.3.2 Feature Descriptors

The results of interest point detection must be encoded by feature descriptors to generate temporal and spatial translation invariants. One successful feature descriptor,

i.e., BOW, was extended by Hadfield [3] [4] to recognize action in a 3D scene. BOW comprises a concatenation of a histogram of oriented gradients (HOG), a histogram-oriented flow (HOF), and histogram-oriented depth gradients (HODG).

$$\rho(x, y, t) = \{HOG(I(x, y, t)), HOF(I(x, y, t)), HODG(D(x, y, t))\} \tag{5.5}$$

5.3.3 Classification

We use multi-class SVM with a kernel radial basis function (RBF) to classify the type of action. Equations 5.6 and 5.7 represent the decision function and kernel, respectively.

$$sgn(\omega^T \phi(x) + b) = sgn(\sum_{i=1}^{\iota} y_i \alpha i K(x_i, x) + b) \tag{5.6}$$

$$K(x_i, x_j) = exp(-\gamma \|x_i - x_j\|^2) \tag{5.7}$$

5.4 Hardware Implementation

To design an embedded vision system, the exploration, computation, and communication functions are designed separately before integration. A block diagram of the developed SoC-FPGA circuit is shown in figure 5.3. Our design is built modular-wise and then merged to allow functional human activity recognition. In the SoC-FPGA Zynq system, the processing system (PS) is used to transfer the stream to the programmable-logic (PL) component, where the actual recognition algorithm is implemented. The architecture captures a data stream from a camera or stored memory and transfers it to a logic circuit. Then, after processing the result passed to an HDMI interface, the model represents concurrent processing, pipeline stages, and interface between blocks.

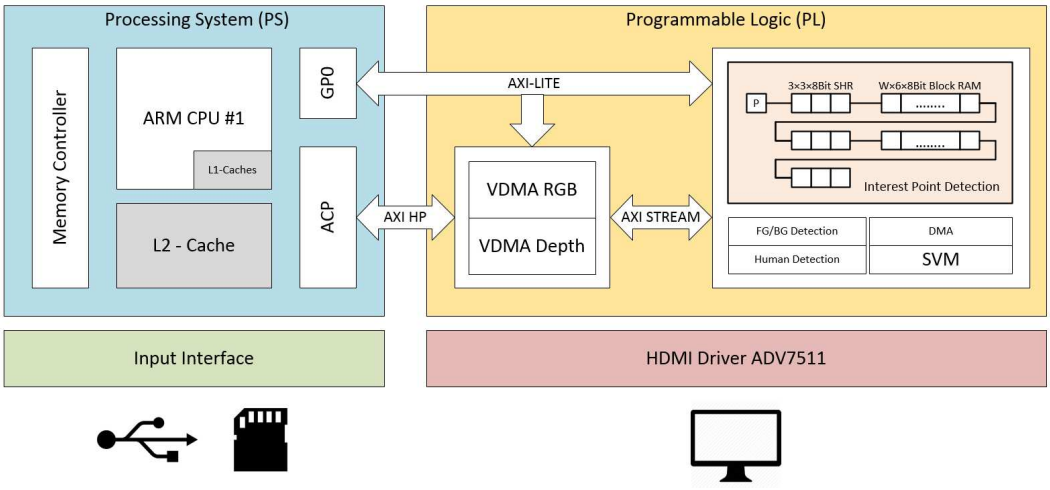


Figure 5.3 – SoC FPGA block diagram showing video devices in the FPGA fabric, which accelerates vision algorithms.

5.4.1 Communication

The proposed architecture separates data into two domains, and it is necessary to define separate clock domains to work with the stream data and the intrinsic-algorithm data independently. The separate clock domains provide data alignment between the streaming and algorithm-intrinsic traffic. We employ a communication-centric architecture to resolve this problem, which also provides a framework for algorithm-intrinsic data access. The first clock domain is the computation domain, which is set by streaming data clocking the vision kernel. The second is the communication domain, which is set by the interconnection of operational data. Separate data paths are used to separate data access using individual ports, where streaming data uses the internal and external ports. For the external port, an efficient solution is obtained by utilizing the systems own port platform (i.e., a USB or SD interface) to read a data stream and an HDMI interface to illustrate the stream. For the internal port, the AXI interface is used. The SoC-FPGA we use in this study is interfaced with AMBA AXI version 4 to connect the PS and PL. In this study, we use two HP AXI interfaces for stream traffic. They connect the processor to the vision-kernel and video-capture

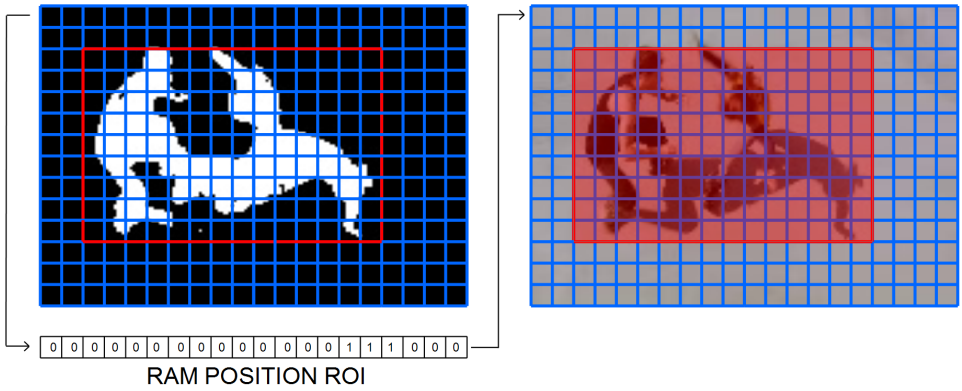


Figure 5.4 – The masked image and ROI window.

modules, which require memory-mapped communication. To control each block, an AXI lite interface is used to connect the vision kernel to the processor. The development platform has multiple common interfaces, such as USB and SD memory readers and an HDMI driver (ADV7715). A dedicated DMA connected to the DDR3 is used to read-back and write-back vision-kernel-intrinsic parameters. The DMA operates in circular mode and automatically restarts when a new frame is input. The advantage of this technique is that it eliminates unnecessary synchronization with the processor. Since the input stream and kernel operate in different clock domains, any misalignment between an input pixel and the kernel data affects the entire process; therefore, the initial alignment is synced when the first frame is input to address this problem.

5.4.2 Computation

The output of foreground and background identification and human detection determines the ROI (Figure 5.4) that can be used for interest point detection in the spatio-temporal domain (equations 5.2 and 5.5). The Harris corners method relies on the values of the intensity gradients along the spatial and temporal dimensions and the gradient of the depth stream. The Hessian point method relies on the values of the second-order intensity derivatives. The values of the intensity gradients along

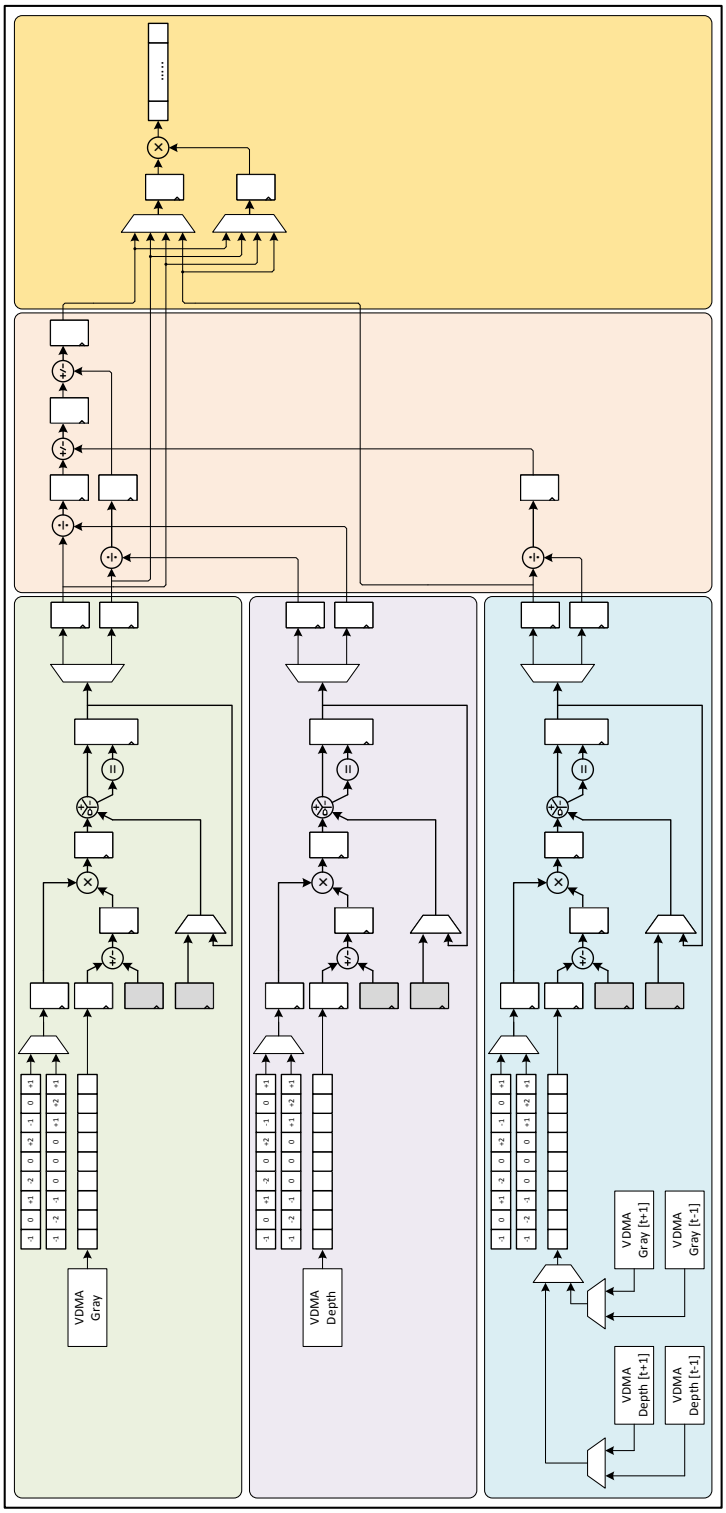


Figure 5.5 – RTL representation interest point detection, Harris Corners 3D.

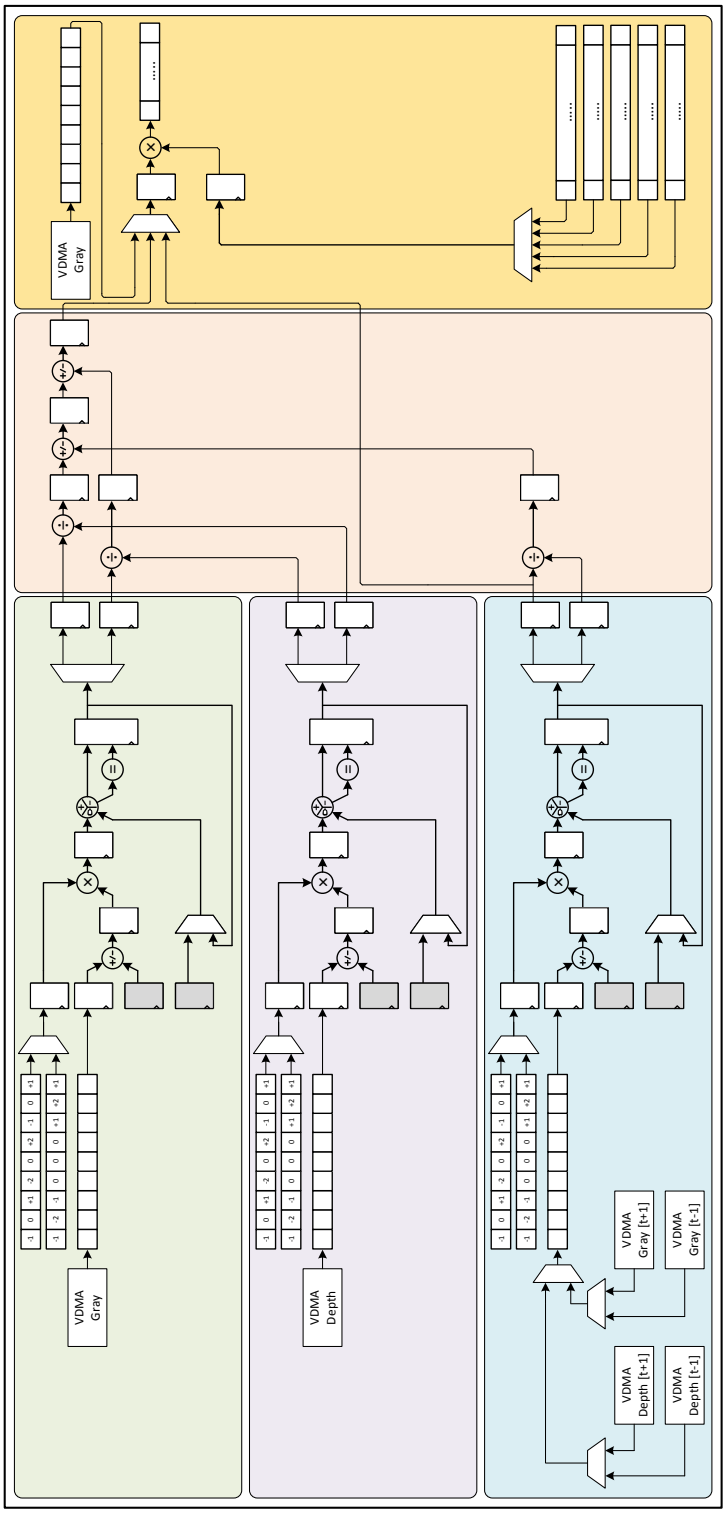


Figure 5.6 – RTL representation interest point detection, Hessian Corners 3D

the x- and y-axes are obtained by the circuit shown in figures 5.3, 5.5 and 5.6. The implemented circuit consists of a 3 × 3 × 8-bit shift register that is accessible by each cell in parallel and a w × 2 × 8-bit block of RAM that stores the results of each cell multiplied by each cell of a 3 × 3 × 8-bit Sobel operator (Figure 5.6). By serializing the multiplication, one multiplier can be saved. The result of the square root is extracted by an arithmetic unit, IP, and the quantized gradient orientation is used to avoid resource-expensive division to determine θ.

The values of the intensity and depth gradients in the temporal domain are obtained by storing the results of three consecutive frames. The value of I_z (equation 5.2) depends on the values of the intensity gradients I_x, I_y, and I_t, and the gradient depths D_x, D_y, and D_t can be determined after the values of I_t and D_t are obtained. The value of the intensity variation in equation 5.2 is computed via the IP arithmetic unit.

The convolution of a first-order Sobel filter kernel with another first-order Sobel filter gives the kernel for a second-order filter, which is given as follows.

$$\begin{aligned} K_{xx} &= (K_x^{sobel} * K_x^{sobel}) \quad K_{xy} = (K_x^{sobel} * K_y^{sobel}) \\ K_{xz} &= (K_x^{sobel} * K_z^{sobel}) \quad K_{xt} = (K_x^{sobel} * K_t^{sobel}) \end{aligned} \tag{5.8}$$

The circuit shown in figure 5.6, which consists of 5 × 5 × 8-bit shift registers that are accessible by each cell in parallel and a w × 4 × 8-bit block of RAM, can be used to determine the second-order derivatives. To make the circuit applicable in the border of the image, the dimensions of the image are extended to ((w + 4) × (h + 4)) by adding dummy values along the image borders. The determinant of the matrix is computed via an arithmetic unit. After employing the Harris and Hessian operators on the interest points, the points that meet the minimum threshold are considered salient points. Then, the result of the saliency information is encoded by a feature descriptor. As explained in Section 5.3, a BOW is used in this study. For the BOW, it is necessary to compute the HOG, HODG, and HOF, and because the values of the HOG and HODG are computed in the ROI during preprocessing, the

only remaining operation is to filter out the interest points that are not considered for the saliency map. This filtering can be performed using a mask filter generated during the comparison of the results of the outputs of the Harris and Hessian operators according to the threshold and computing the HOF. In this study, to implement the HOF, we use the method proposed by Gultekin and Saranli [6]. Here, consider E(x, y) as the brightest pixel in the coordinate (x, y). This can be determined as follows:

$$E_x u + E_y v + E_t = 0, \tag{5.9}$$

where $u = \frac{dx}{dt}$ and $v = \frac{dy}{dt}$ are the scalar horizontal and vertical components of the optical flow vectors, respectively. Equations 5.10 and 5.11 compute u and v, and the values of E_x, E_y, and E_t are computed by first-order differentiation of eight pixels. The average values of the u and v optical flows are estimated using a 3 × 3 weight matrix. The circuit shown in 5.7 used to compute HOF.

$$u_{n+1} = \overline{u}_n - E_x \frac{E_x \overline{u}_n + E_y \overline{v}_n + E_t}{\alpha^2 + E_x^2 + E_y^2} \tag{5.10}$$

$$v_{n+1} = \overline{v}_n - E_x \frac{E_x \overline{u}_n + E_y \overline{v}_n + E_t}{\alpha^2 + E_x^2 + E_y^2} \tag{5.11}$$

$$E_x = \begin{bmatrix} -\frac{1}{4} & \frac{1}{4} \\ -\frac{1}{4} & \frac{1}{4} \end{bmatrix} \times I_{t-1} + \begin{bmatrix} -\frac{1}{4} & \frac{1}{4} \\ -\frac{1}{4} & \frac{1}{4} \end{bmatrix} \times I_t \tag{5.12}$$

$$E_y = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} \\ -\frac{1}{4} - & \frac{1}{4} \end{bmatrix} \times I_{t-1} + \begin{bmatrix} \frac{1}{4} & \frac{1}{4} \\ -\frac{1}{4} & -\frac{1}{4} \end{bmatrix} \times I_t \tag{5.13}$$

$$E_t = \begin{bmatrix} -\frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{4} & -\frac{1}{4} \end{bmatrix} \times I_{t-1} + \begin{bmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{bmatrix} \times I_t \tag{5.14}$$

$$\overline{u}, \overline{v} = \begin{bmatrix} \frac{1}{12} & \frac{1}{6} & \frac{1}{12} \\ \frac{1}{12} & -1 & \frac{1}{6} \\ \frac{1}{12} & \frac{1}{6} & \frac{1}{12} \end{bmatrix} \tag{5.15}$$

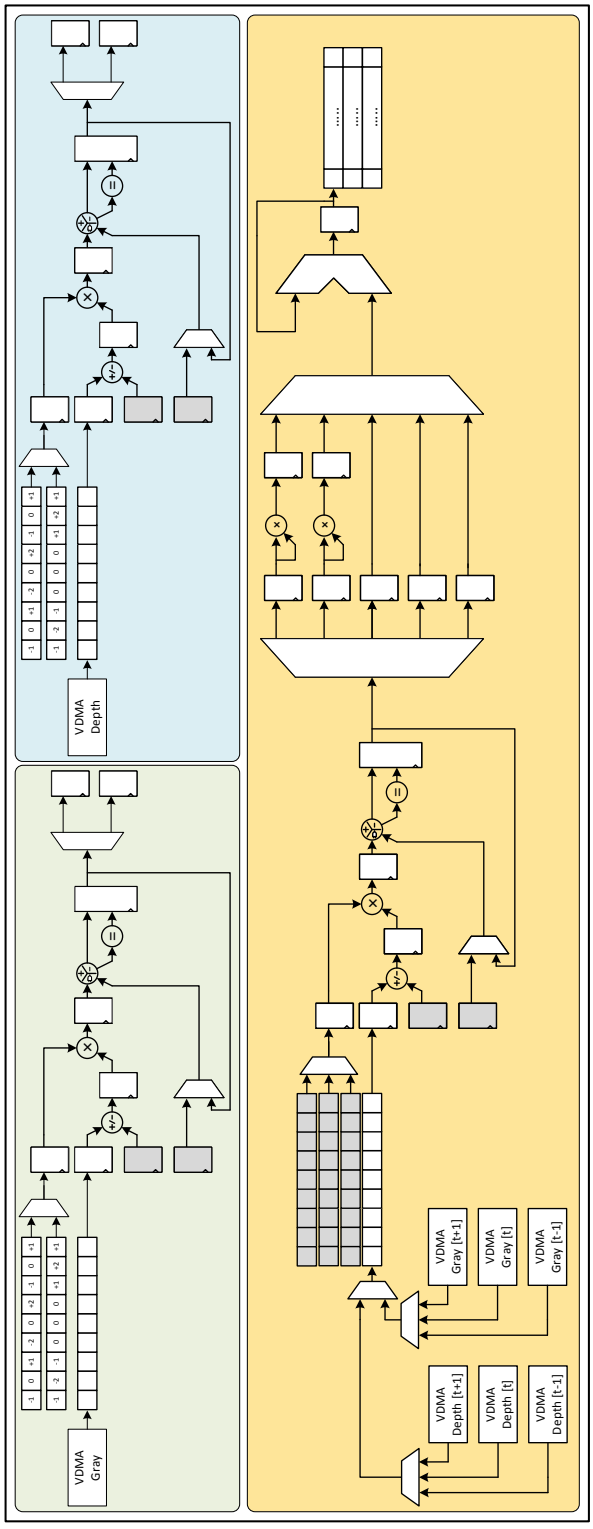


Figure 5.7 – RTL representation HOF computation.

5.4.3 Classification

In this work, to implement SVM, we use the methodology proposed in [7]. Algorithm 5.1 represents the steps of the SVM classifier. It accepts input as a vector, and the output defines the type of action. The classifier consists of a few steps and loops

Algorithm 3 SVM

1: $\overline{V} = \{$ Vector to classify $\}$
2: **for** $x_i \leftarrow 1$ **to** $\overline{V}$ **do**
3: $\quad c_i \leftarrow theclassof\mathbf{x}$
4: $\quad k_i \leftarrow K(x_i, x)$
5: $\quad$ **for** All c class **do**
6: $\quad\quad$ **if** $c_i \neq c_j$ **then**
7: $\quad\quad\quad d = index \ \ of \ \ the \ \ decision$
8: $\quad\quad\quad S_d = S_d + y_{d,i} \times \alpha_{d,i} \times k_i$
9: $\quad\quad$ **end if**
10: $\quad$ **end for**
11: **end for**
12: **for** All decision d **do**
13: $\quad$ **if** $S_d - b_d > 0$ **then**
14: $\quad\quad V_i = V_i + 1$
15: $\quad$ **else if**
16: $\quad\quad$ **then**$V_j = V_j + 1$
17: $\quad$ **end if**
18: **end for**
19: c_n with the highest V_n is selected.

that are repeated for each input vector. The first loop, which is called the main loop has an additional inner loop, called the sum loop, and the second loop in the main sequence is called caparison loop. The main loop consumes the most time, and

it has some inner loops and performs complicated mathematical operations, such as multiplication and exponentiation. To parallelize the operation to improve the speed, the shared data S are duplicated and then merged at the end of the process. As a result, each duplicated shared data S can run in parallel with others. The iteration of the sum loop is defined based on the number of classes; therefore, it is independent and can run in parallel. By contrast, the sum loop depends on the vote counter, which can be run in parallel by initially duplicating and then merging at the end of each iteration. These modifications enable the SVM to be run partially in parallel. Some of the constant parameters, such as w_i and the bias b, can be loaded by the processor.

5.5 Implementation Results and Evaluation

The proposed human action recognition methodology was implemented with a Xilinx XC7Z020 development board equipped with a Zynq SoC XC7Z020-1CLG484C. The clocks for the system were a 140 MHz for the FPGA and 866 MHz for the ARM processor. All the modules for the programmable logic part were written by VHDL and synthesized by Xilinx Vivado 2015.2. The ARM processor software was written in C and developed in Xilinx SDK 2015.2. We tested the proposed system on two modern datasets with depth data that are used for human action and activity recognition, i.e., Hollywood [3] and HON4D [8]. The results are presented in tables 5.1, 5.2 and 5.3. It should be considered The highest accuracy was reached on a PC with floating-point variables and operations, whereas in this work, we used fixed-point variables. Additionally, we compared our results with other methods on a challenging dataset [9] and summarized the correctly classified rates in table 5.4.

Table 5.1 – FPGA Consumed Resources

Target FPGA	LUT	Flip Flop	DSP48E	BRAM
Virtex-II [10][1]	40960	40960	N.A	160
Kintex-7 [11]	123708	N.A	320	265
Virtex-6 [11]	197025	N.A	168	247
Proposed System (Zynq)	**36962**	**42115**	**108**	**96**

[1] Consumed resources considered based on summation of all implementation.

Table 5.2 – Average precision per class on the 3D action dataset [4]

Action	4D Ha	Proposed Method	4D He	Proposed Method
NoAction	12.9	12.9	12.2	13.0
Run	22.4	23.2	15.9	17.3
Punch	4.8	5.0	2.9	4.6
Kick	4.3	4.5	4.2	4.5
Shoot	17.2	19.0	18.9	19.1
Eat	5.3	5.3	3.6	5.5
Drive	69.3	73.0	25.6	27.0
UsePhone	8.0	8.4	14.7	15.6
Kiss	10.0	10.8	8.5	9.1
Hug	4.4	4.6	3.5	8.6
StandUp	7.6	7.9	7.0	7.7
SitDown	4.2	5.5	4.5	5.0
Swim	5.5	7.2	7.8	8.4
Dance	10.5	14.6	4.2	4.9
Average	13.3	**14.4**	9.8	**10.7**

Table 5.3 – The performance of our method on MSR Action 3D dataset [8] compared to previous approaches.

Method	Accuracy %
HON4D + D_{disc} [8]	88.89
HON4D [8]	85.85
Jiang et al. [12]	88.20
Jiang et al. [13]	86.50
Yang et al. [14]	85.52
Dollar [15] + BOW	72.40
STIP [16] + BOW	69.57
Vieira et al. [17]	78.20
Klaser et al. [18]	81.43
Proposed Method	**87.20**

5.6 Conclusion

In this study, we have presented a SoC-FPGA-based video processing system. The proposed algorithm employs parallel processing and pipeline architecture to perform real-time human action recognition. The experimental results for all tested datasets indicate that the proposed method improves the average precision compared to conventional methods. In the future, we will focus on improving the hardware architecture by using both cores of the ARM processor. In addition, we plan to implement more complex feature descriptors.

Table 5.4 – The performance of our method on challenging human action recognition dataset [9], compared to previous approaches.

Method	Accuracy %
SVM on local features [9]	71.7
Cascade of filters on volumetric features [19]	62.9
SVM on MHI [20]	63.5
SVM 2K on MHI & MMHI [21]	65.3
SVM on HWT of MHI & Hist. of MHI [22]	70.9
SVM on MGD & Hist. of MHI [23]	80.3
SVM on spatio-temporal feature [15]	81.2
Learning on spatial-temporal words [24]	81.5
KNN on NZMS [25]	86.0
Proposed Method	**88.4**

5.7 Summary

This chapter discusses a system-on-a-chip field-programmable gate array (SoC-FPGA)-based real-time video processing platform for human action recognition. It presents the details of a hardware implementation for real-time human action recognition in 3D scenes, including capture, processing, and display. The proposed platform was implemented by adding a two-stage preprocessing step to improve the saliency map results and utilizing the inherent parallelism of FPGAs. Appropriate circuits for the parallelized execution of Harris and Hessian interest point detection and support vector machine classifiers are discussed.

References

[1] I. Laptev and T. Lindeberg, "Space-time interest points," in *IN ICCV*, 2003, pp. 432–439.

[2] G. Willems, T. Tuytelaars, and L. Van Gool, "An efficient dense and scale-invariant spatio-temporal interest point detector," in *Computer Vision–ECCV 2008*. Springer, 2008, pp. 650–663.

[3] S. Hadfield and R. Bowden, "Hollywood 3d: Recognizing actions in 3d natural scenes," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE, 2013, pp. 3398–3405.

[4] S. Hadfield, K. Lebeda, and R. Bowden, "Hollywood 3d: What are the best 3d features for action recognition?" *International Journal of Computer Vision*, pp. 1–16, 2016.

[5] P. R. Beaudet, "Rotationally invariant image operators." in *International Joint Conference on Pattern Recognition*, 1978, pp. 579–583.

[6] G. K. Gultekin and A. Saranli, "An fpga based high performance optical flow hardware design for computer vision applications," *Microprocessors and Microsystems*, vol. 37, no. 3, pp. 270–286, 2013.

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

[8] O. Oreifej and Z. Liu, "Hon4d: Histogram of oriented 4d normals for activity recognition from depth sequences," in *Proc. Computer Society Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2013, pp. 716–723.

[9] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: A local svm approach," in *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, vol. 3. IEEE, 2004, pp. 32–36.

[10] K. Sehairi, C. Benbouchama, F. Chouireb *et al.*, "Real-time implementation of human action recognition system based on motion analysis," in *Artificial Intelligence and Computer Vision*. Springer, 2017, pp. 143–164.

[11] X. Ma, J. R. Borbon, W. Najjar, and A. K. Roy-Chowdhury, "Optimizing hardware design for human action recognition," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–11.

[12] J. Wang, Z. Liu, Y. Wu, and J. Yuan, "Mining actionlet ensemble for action recognition with depth cameras," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 1290–1297.

[13] J. Wang, Z. Liu, J. Chorowski, Z. Chen, and Y. Wu, "Robust 3d action recognition with random occupancy patterns," in *Computer vision–ECCV 2012*. Springer, 2012, pp. 872–885.

[14] X. Yang, C. Zhang, and Y. Tian, "Recognizing actions using depth motion maps-based histograms of oriented gradients," in *Proceedings of the 20th ACM international conference on Multimedia*. ACM, 2012, pp. 1057–1060.

[15] P. Dollár, V. Rabaud, G. Cottrell, and S. Belongie, "Behavior recognition via sparse spatio-temporal features," in *Visual Surveillance and Performance Evaluation of Tracking and Surveillance, 2005. 2nd Joint IEEE International Workshop on*. IEEE, 2005, pp. 65–72.

[16] I. Laptev, "On space-time interest points," *International journal of computer vision*, vol. 64, no. 2-3, pp. 107–123, 2005.

[17] A. Vieira, E. Nascimento, G. Oliveira, Z. Liu, and M. Campos, "Stop: Space-time occupancy patterns for 3d action recognition from depth map sequences," *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pp. 252–259, 2012.

[18] A. Klaser, M. Marszałek, and C. Schmid, "A spatio-temporal descriptor based on 3d-gradients," in *BMVC 2008-19th British Machine Vision Conference*. British Machine Vision Association, 2008, pp. 275–1.

[19] Y. Ke, R. Sukthankar, and M. Hebert, "Efficient visual event detection using volumetric features," in *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, vol. 1. IEEE, 2005, pp. 166–173.

[20] H. Meng, N. Pears, and C. Bailey, "Recognizing heuman actions based on motion information and svm," in *Intelligent Environments, 2006. IE 06. 2nd IET International Conference on*, vol. 1. IET, 2006, pp. 239–245.

[21] H. Meng, N. Pears, and C. Bailey, "Human action classification using svm_2k classifier on motion features," in *International Workshop on Multimedia Content Representation, Classification and Security*. Springer, 2006, pp. 458–465.

[22] H. Meng, N. Pears, and C. Bailey, "Motion information combination for fast human action recognition." in *VISAPP (2)*, 2007, pp. 21–28.

[23] H. Meng, N. Pears, and C. Bailey, "A human action recognition system for embedded computer vision application," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE, 2007, pp. 1–6.

[24] J. C. Niebles, H. Wang, and L. Fei-Fei, "Unsupervised learning of human action categories using spatial-temporal words," *International journal of computer vision*, vol. 79, no. 3, pp. 299–318, 2008.

[25] C. Yeo, P. Ahammad, K. Ramchandran, and S. S. Sastry, "Compressed domain real-time action recognition," in *Multimedia Signal Processing, 2006 IEEE 8th Workshop on*. IEEE, 2006, pp. 33–36.

Chapter 6

System-on-a-Chip (SoC)-based Hardware Acceleration for an Online Sequential Extreme Learning Machine (OS-ELM)

6.1 Introduction

Neural networks (NNs) are a powerful class of machine learning algorithms that are used in a wide range of applications. The complexity of these algorithm depends on the number of samples. As a result, for applications that require high accuracy, the number of computational operations required is also quite high. Consequently, when such an application is embedded, as in the case of an embedded vision, automotive or security (surveillance) system that must operate in real time, concerns such as power management, computational efficiency, and efficient resource (cell) utilization become more important.

One of the active research areas related to NNs concerns extreme learning machines (ELMs), which are well known for their computational efficiency and their good performance for high-volume data processing. The ELM approach is based on the theory of random vector functional link (RVFL) networks [1]; however, ELMs generally outperform RVFL networks [2]. An ELM is a single-hidden-layer feed-forward neural classifier in which the hidden layer does not need to consist of identical neurons. Figure 6.1 depicts the structure of an ELM, which consists of a random hidden layer with nonlinear feature mapping to the output. Its advantages include a low training complexity, a fast learning speed, the ability to use different types of activa-

tion functions and a well-known representation ability. Currently, many researchers are attempting to develop machine learning algorithms with better learning rates and improved performance; however, few researchers have focused on developing a hardware implementation to execute such an algorithm in its entirety. This has motivated considerable research on developing systems for utilizing parallel computing platforms such as field-programmable gate arrays (FPGAs) [3] [4] [5] and graphics processing units (GPUs) [6] or specialized multi-core microprocessors, whose architectures are optimized for multiple computational operations.

Implementations of ELMs on GPU platforms have recently been discussed [7]; however, GPUs face challenges with regard to power consumption [8] and are consequently difficult to deploy in embedded environments. Hence, a new generation of system-on-a-chip (SoC) FPGA-based devices has emerged as an attractive platform for embedded applications. Such devices include both a microprocessor and an FPGA on a single chip [9] [10], consume less power and can be integrated into smaller systems.

In this work, we propose a specialized SoC hardware implementation and design approach for embedded online sequential ELM (OS-ELM) classification. The OS-ELM approach is based on the ELM approach but is optimized for efficiency in real-time applications such as human action recognition, in which classification must be performed every time a new frame arrives. In addition, for use in embedded platforms, the proposed system must consume little power and must often operate with limited available resources. The presented design was developed as part of a research program on the development of embedded systems for human action recognition; related studies concerning the pre-processing and feature extraction stages have been discussed in previous chapters, whereas in this chapter, we mainly focus on the final step, namely, classification. Although in the previous chapter, classification via the SVM approach was presented, the structure of SVM classifiers and their limitations with regard to real-time operation make the OS-ELM approach a more efficient solu-

tion for classification. We also discuss how to reduce the hardware complexity of the OS-ELM classifier by utilizing the processor and the FPGA together and improving both the area and power requirements through an efficient matrix decomposition for ELM/OS-ELM training. Moreover, the proposed design attempts to improve the efficiency of matrix multiplication by dividing the necessary operations into subtasks to enable pipelined operation and the use of ping-pong memory.

The proposed system and algorithms have been implemented on a SoC FPGA (Zynq 7000) platform as part of a research program on human activity recognition. The system was evaluated on two complex and challenging 3D human action recognition datasets, i.e., Hollywood [11] [12] and HON4D [13], with resolutions of 800×600 and 1280×720. The results of the OS-ELM and ELM implementations demonstrate that the hardware reduction of the OS-ELM implementation reduces the utilization of FPGA logic resources by 19-46%. Moreover, the proposed architecture is 200x times faster than an ARM Cortex-A9 for the matrix inversion computation, which is one of the most computationally demanding functions, and consumes 157 mW of on-chip power.

This chapter is organized as follows. Section 6.2 provides some background on ELM/OS-ELM classifiers and discusses related work. Section 6.3 details the proposed hardware architecture for ELM/OS-ELM processing and the hardware optimization method. Section 6.4 presents the results of experimental performance evaluations and a comparison with related works. The performance of the proposed algorithm on human action recognition datasets is discussed in Section 6.5. Finally, conclusions are drawn in Section 6.6.

6.2 Related Work

The fixed training time, possibility of parallel operation and temporal requirements of ELM classifiers make them good candidates for hardware implementation. Be-

cause of the specific structure of NNs designed for applications that require online training and classification, general processors are not efficient for the implementation of such NNs. To address this problem, various hardware accelerators based on FPGAs, GPUs, VLSI/ASICs or SoC FPGAs have recently been proposed to improve the performance of NN designs. Early NN implementations utilized parallel computing architectures, such as SIMD [14] [15] or MIMD [16] [17] [18]. In [19] [20], parallel implementations of the learning process were discussed. Neuro-computers are another class of platforms designed based on computing devices such as digital signal processors (DSPs) [21] [22] [23] or neuro-processors, as discussed in [24] [25]. VLSI/ASIC (analog or digital) implementations represent another approach, although analog implementations have disadvantages in terms of precision and data storage and also lack flexibility, whereas digital implementations [26] [27] offer better accuracy and can handle standard neural computations. Due to the difficulty of design and the cost of production, the VLSI/ASIC approach is not popular; however, in a recent work [28], a compact low-power chip was used to execute the ELM algorithm. Several works have attempted to improve the speed of classification by means of the data acceleration offered by GPUs [6] [29] [30] [29]; however, as discussed above, the power consumption of GPUs poses a challenge for embedded platforms. Among these approaches, configurable hardware devices such as FPGAs and complex programmable logic devices (CPLDs) have been attracting increasing attention from researchers because of their good performance, high energy efficiency, rapid development cycle and simple coding. In addition, the reprogrammable nature of FPGAs/CPLDs allows the current design to be easily mapped to an improved device without any modification.

Several interesting hardware implementations have been reported for various applications [31] [32] [33] [34]. Among these, the algorithm developed by Decherchi et al. [33] reduces the complexity of a trained ELM for implementation. A silicon-based ELM implementation has been discussed by Basu et al. [31]. In [33], an FPGA implementation was discussed in which the training-phase calculations were performed

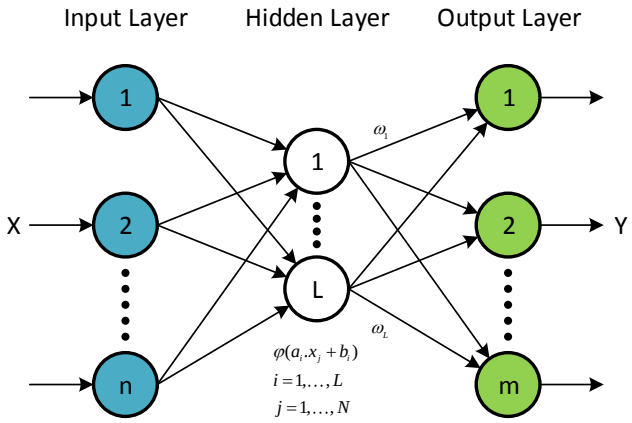


Figure 6.1 – ELM architecture.

on a PC. The works presented in [4] [5] discussed both the practical and theoretical implementation of ELMs in FPGAs using a finite state machine (FSM) approach, and finally, a recent work on OS-ELM implementation was presented in [5].

6.2.1 Extreme Learning Machine (ELM)

The ELM approach is mainly based on a learning theory for single-hidden-layer feed-forward neural networks (SLFNs) in which the hidden layer need not consist of identical neurons. Traditionally, gradient-descent-based methods have been used for feed-forward NNs, in which all parameters must be tuned, necessitating a long processing time. By contrast, an ELM contains only one hidden layer in which all of the layer parameters, weights and biases are randomly defined. Inverse operations can be used to determine the output weights that link the hidden layer to the output layer.

In this section, we briefly discuss the fundamental theory of ELMs. Further details can be found in [35] [36] [37] [38] [39].

Let us consider a training dataset X consisting of N arbitrary distinct sample pairs $(x_j, y_j), j = 1 \ldots N$, where $x_j = [x_{j1}, x_{j2}, \ldots, x_{jn}]^T \in R^n$ is the j_{th} input vector and $y_j = [y_{j1}, y_{j2}, \ldots, y_{jm}]^T \in R^m$ is the j_{th} target vector. Then, an SLFN with L nodes in the hidden layer, an activation function $\varphi(x)$ and an output function $f(x)$

can be defined as follows:

$$f_L(x) = \sum_{i=1}^{L} \omega_i \varphi_i(x) = \varphi(x)\omega \tag{6.1}$$

$$y_j = \sum_{i=1}^{L} \omega_i \varphi(a_i.x_j + b_i), \quad j = 1, \ldots, N \tag{6.2}$$

where ω_i is the weight vector connecting the i^{th} hidden node to the output nodes and $\varphi(x) = [\varphi_1(x), \ldots, \varphi_L]$ is the "nonlinear feature mapping" of the ELM. The vector $\varphi_i(x)$ is the output of the i^{th} hidden node, where $a_i = [a_{i1}, a_{i2}, \ldots, a_{in}]^T$ is the set of weights connecting the input layer to this hidden node and $b_i = [b_{i1}, b_{i2}, \ldots, b_{in}]^T$ is the bias term. The ELM training process consists of two steps. First, the hidden node parameters (a, b) are randomly defined to map the input data into the feature space. The mapping function could be any activation function [35]; however, the sigmoid function is commonly used.

$$\varphi_i(x) = g(a_i, b_i, x) \tag{6.3}$$

$$g(a, b, x) = \frac{1}{1 + e^{-a.x+b}} \tag{6.4}$$

The second step is to find the values of the weights ω_i that connect the hidden nodes to the output nodes, which are obtained by minimizing the convex cost:

$$\min_{\omega \in R^{L \times m}} \|\varphi\omega - y\|^2 \tag{6.5}$$

$$\varphi = \begin{bmatrix} \varphi(x_1) \\ \vdots \\ \varphi(x_N) \end{bmatrix}_{N \times L} = \begin{bmatrix} \varphi_1(x_1) & \cdots & \varphi_L(x_1) \\ \vdots & \ddots & \vdots \\ \varphi_1(x_N) & \cdots & \varphi_L(x_N) \end{bmatrix}_{N \times L} \tag{6.6}$$

$$\omega = \begin{bmatrix} \omega_1^T \\ \vdots \\ \omega_L^T \end{bmatrix}_{L \times m} = \begin{bmatrix} \omega_{11} & \cdots & \omega_{1m} \\ \vdots & \ddots & \vdots \\ \omega_{L1} & \cdots & \omega_{Lm} \end{bmatrix}_{L \times m} \tag{6.7}$$

$$y = \begin{bmatrix} y_1^T \\ \vdots \\ y_N^T \end{bmatrix}_{N \times m} = \begin{bmatrix} y_{11} & \cdots & y_{1m} \\ \vdots & \ddots & \vdots \\ y_{N1} & \cdots & y_{Nm} \end{bmatrix}_{N \times m} \tag{6.8}$$

where y and φ are the training data and the hidden layer output matrix, respectively, and $\|.\|$ is the Euclidean norm. Each row of the matrix φ represents the hidden layer feature mapping with respect to the i^{th} input x_i, and each column of φ represents the output of the i^{th} hidden node with respect to the inputs $x_1, x_2, \cdots, x_N$. To solve equation 6.5, suppose that the number of hidden neurons is equal to the number of samples; then, the matrix φ is square and invertible. By inverting the matrix φ, the output ω can be obtained. However, in practice, the number of samples will typically be greater than the number of hidden nodes; in this case, the matrix φ is not square, and equation 6.5 is not solvable. To resolve this problem, since a and b are fixed, equation 6.5 can be regarded as a linear system and thus can be rewritten as

$$\omega = \varphi^{\dagger} y \tag{6.9}$$

where $\varphi^{\dagger}$ is the Moore–Penrose pseudoinverse of the matrix φ, which can be obtained as follows:

$$\varphi^{\dagger} = (\varphi^T \varphi)^{-1} \varphi^T \rightarrow \text{If } \varphi^T \varphi \text{ is nonsingular} \tag{6.10}$$

$$\varphi^{\dagger} = \varphi^T (\varphi \varphi^T)^{-1} \rightarrow \text{If } \varphi \varphi^T \text{ is nonsingular} \tag{6.11}$$

Since the training process requires only three calculation steps, the training time can be extremely fast, which is necessary for applications that require real-time training.

6.2.2 ELM Computations

As discussed with regard to equation 6.9, to obtain the matrix $\varphi^{\dagger}$, matrix inversion, transposition and multiplication are required. Among these three operations, the inversion operation is the most challenging. Various approaches to matrix decomposition have been proposed, among which the *Cholesky*, *LU* and QR methods are the most common. The Cholesky and LU methods are mainly used for positive and nonsingular square matrices, whereas the QR method can be used for any kind of matrix. In the QR method, a given square matrix φ is decomposed into an orthogonal matrix

Q and a triangular matrix R, $\varphi = QR$; then, the inverse matrix can be computed as $\varphi^{-1} = (Q.R)^{-1} = R^{-1}.Q^t$. For a nonsquare matrix, the pseudo-inverse of φ can be found as $\varphi^{\dagger} = R^{-1}.Q^t$. Various approaches have been proposed for computing QR decompositions, including the *Gram-Schmidt*, *Householder transformation* and *Givens rotation* methods. Among them, the method of Givens rotations is the most popular because of its stability, whereas the Gram-Schmidt method is not efficient when run on hardware [40]. In the modified Gram-Schmidt (MGS) model, the stability problem is solved. In addition, compared with the original algorithm, it requires a square root operation in the last step; as a result, it requires fewer operations and less resources [41].

$$\varphi = [\varphi_1|\varphi_2|\cdots|\varphi_n] \tag{6.12}$$

$$u_1 = \varphi_1 \tag{6.13}$$

$$u_2 = \varphi_2 - \frac{\langle \varphi_2, u_1 \rangle}{\langle u_1, u_1 \rangle} u_1 \tag{6.14}$$

$$u_n = \varphi_n - \frac{\langle \varphi_n, u_1 \rangle}{\langle u_1, u_1 \rangle} u_1 - \ldots \frac{\langle \varphi_n, u_{n-1} \rangle}{\langle u_{n-1}, u_{n-1} \rangle} u_{n-1} \tag{6.15}$$

$$e_1 = \frac{u_1}{\| u_1 \|} \quad e_2 = \frac{u_2}{\| u_2 \|} \quad \ldots\ldots\ldots \quad e_n = \frac{u_n}{\| u_n \|} \tag{6.16}$$

$$\varphi = \underbrace{[e_1|e_2|\cdots|e_n]}_{\mathbf{Q}} \underbrace{\begin{bmatrix} \varphi_1 \cdot e_1 & \varphi_2 \cdot e_1 & \ldots & \varphi_n \cdot e_1 \\ 0 & \varphi_2 \cdot e_2 & \ldots & \varphi_n \cdot e_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \varphi_n \cdot e_n \end{bmatrix}}_{\mathbf{R}} \tag{6.17}$$

$$\varphi^{-1} = (Q.R)^{-1} = R^{-1}.Q^t \quad \text{If } \varphi \text{ is a square matrix} \tag{6.18}$$

$$\varphi^{\dagger} = (Q.R)^{-1} = R^{-1}.Q^t \quad \text{If } \varphi \text{ is a non-square matrix} \tag{6.19}$$

6.2.3 Online Sequential ELM (OS-ELM)

Recently, in many artificial intelligence applications, online learning algorithms have begun to be used for time-sensitive tasks, such as deep brain simulation (DBS) and

human activity and action recognition. In such applications, not all of the training data are available before the training process begins. One well-known and efficient online machine learning algorithm is the OS-ELM algorithm [42] [43], which is a variation of the ELM approach in which data may be received in a one-by-one or chunk-by-chunk manner, where the chunks may be of fixed or varying sizes. The OS-ELM algorithm consists of two phases: the initialization phase and the sequential learning phase.

Initialization

In this phase, the initial hidden layer output matrix needs to be defined based on the initial chunk of training data. Let us consider a small chunk X_0 of the training dataset X.

$$X_0 = \{(x_i, y_i)\}_{i=1}^{N_0} \tag{6.20}$$

$$X_0 = \{(x_i, y_i) | x_i \in R^n, y_i \in R^m, i = 1 \dots N_0\} \tag{6.21}$$

$$\varphi_0 = \begin{bmatrix} \varphi(x_1) \\ \vdots \\ \varphi(x_{N_0}) \end{bmatrix}_{N_0 \times L} = \begin{bmatrix} \varphi_1(x_1) & \cdots & \varphi_L(x_1) \\ \vdots & \ddots & \vdots \\ \varphi_1(x_{N_0}) & \cdots & \varphi_L(x_{N_0}) \end{bmatrix}_{N_0 \times L} \tag{6.22}$$

$$\min_{\omega \in R^{L \times m}} \|\varphi_0 \omega_0 - y_0\|^2 \tag{6.23}$$

$$\varphi_0^\dagger = (\varphi_0^T \varphi_0)^{-1} \varphi_0^T \quad \eta_0 = \varphi_0^T \varphi_0 \tag{6.24}$$

$$\omega_0 = \varphi_0^\dagger y_0 = \eta_0^{-1} \varphi_0^T y_0 \tag{6.25}$$

Sequential Learning Phase

When a new chunk of data is received, all parameters must be updated. Let us consider another data chunk, denoted by X_1.

$$X_1 = \{(x_i, y_i)\}_{i=N_0+1}^{N_0+N_1} \tag{6.26}$$

$$X_1 = \{(x_i, y_i) | x_i \in R^n, y_i \in R^m, i = (N_0 + 1) \dots (N_0 + N_1)\} \tag{6.27}$$

$$\varphi_1 = \begin{bmatrix} \varphi(x_{N_0+1}) \\ \vdots \\ \varphi(x_{N_0+N_1}) \end{bmatrix}_{N_1 \times L} = \begin{bmatrix} \varphi_1(x_{N_0+1}) & \cdots & \varphi_L(x_{N_0+N_1}) \\ \vdots & \ddots & \vdots \\ \varphi_1(x_{N_0+N_1}) & \cdots & \varphi_L(x_{N_0+N_1}) \end{bmatrix}_{N_1 \times L} \tag{6.28}$$

$$\min_{\omega \in R^{L \times m}} \left\| \begin{bmatrix} \varphi_0 \\ \varphi_1 \end{bmatrix} \omega_1 - \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} \right\|^2 \tag{6.29}$$

$$\omega_1 = \eta_1^{-1} \begin{bmatrix} \varphi_0 \\ \varphi_1 \end{bmatrix}^T \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} \quad \eta_1 = \begin{bmatrix} \varphi_0 \\ \varphi_1 \end{bmatrix}^T \begin{bmatrix} \varphi_0 \\ \varphi_1 \end{bmatrix} \tag{6.30}$$

$$\eta_1 = \begin{bmatrix} \varphi_0 \\ \varphi_1 \end{bmatrix}^T \begin{bmatrix} \varphi_0 \\ \varphi_1 \end{bmatrix} = \begin{bmatrix} \varphi_0^T & \varphi_1^T \end{bmatrix} \begin{bmatrix} \varphi_0 \\ \varphi_1 \end{bmatrix} = \eta_0 + \varphi_1^T \varphi_1 \tag{6.31}$$

$$\begin{aligned} \begin{bmatrix} \varphi_0 \\ \varphi_1 \end{bmatrix}^T \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} &= \varphi_0^T T_0 + \varphi_1^T T_1 \\ &= \eta_0 \eta_0^{-1} \varphi_0^T y_0 + \varphi_1^T y_1 \\ &= \eta_0 \omega_0 + \varphi_1^T y_1 \\ &= (\eta_1 - \varphi_1^T \varphi_1)\omega_0 + \varphi_1^T y_1 \\ &= \eta_1 \omega_0 - \varphi_1^T \varphi_1 \omega_0 + \varphi_1^T y_1 \end{aligned} \tag{6.32}$$

By substituting equation 6.32 into equation 6.30, the output is obtained as follows:

$$\begin{aligned} \omega_1 &= \eta_1^{-1}(\eta_1 \omega_0 - \varphi_1^T \varphi_1 \omega_0 + \varphi_1^T y_1) \\ &= \omega_0 + \eta_1^{-1} \varphi_1^T (y_1 - \varphi_1 \omega_0) \end{aligned} \tag{6.33}$$

where η_1 is given by

$$\eta_1 = \eta_0 + \varphi_1^T \varphi_1 \tag{6.34}$$

The general formulas are given by

$$X_{k+1} = \{(x_i, y_i)\}_{i=(\Sigma_{j=0}^{k} N_j)+1}^{\Sigma_{j=0}^{k+1} N_j} \tag{6.35}$$

$$\eta_{k+1} = \eta_k + \varphi_{k+1}^T \varphi_{k+1} \tag{6.36}$$

$$\eta_{k+1}^{-1} = \eta_k^{-1} - \eta_k^{-1} \varphi_{k+1}^T (I + \varphi_{k+1} \eta_k^{-1} \varphi_{k+1}^T)^{-1} \varphi_{k+1} \eta_k^{-1} \tag{6.37}$$

By considering η_{k+1}^{-1} with p_{k+1}, we obtain

$$p_{k+1} = p_k - p_k \varphi_{k+1}^T (I + \varphi_{k+1} p_k \varphi_{k+1}^T)^{-1} \varphi_{k+1} p_k \tag{6.38}$$

$$\omega_{k+1} = \omega_k + p_{k+1} \varphi_{k+1}^T (y_{k+1} - \varphi_{k+1} \omega_k) \tag{6.39}$$

6.3 System Architecture

As mentioned before, most recently emerging real-world applications are time-sensitive and require the algorithm parameters to be updated when new data arrive. As a result, both the training and prediction blocks must work simultaneously. Since each block has a different mathematical model with a different latency, each block operates in a different clock domain. In this work, the proposed hardware architecture is conceived as an IP core that can be integrated into a complex design as an additional peripheral or can operate as a standalone module. The proposed implementation was synthesized and run on a Xilinx Zynq SoC 7000. The advantage of the Xilinx Zynq SoC is that it combines dual-core ARM processors (PS unit) and a programmable logic block (PL unit) in a single chip. The hardware structure and external data interface are based on the following five main blocks (Figure 6.2).

- **Training module:** This module performs OS-ELM training to obtain the weight parameters using the QR decomposition method. It consists of two submodules, one for the *initialization phase* and one for the *sequential learning phase*.

- **Prediction module:** This module classifies the input stream based on the weights and biases obtained by the training module. The prediction module

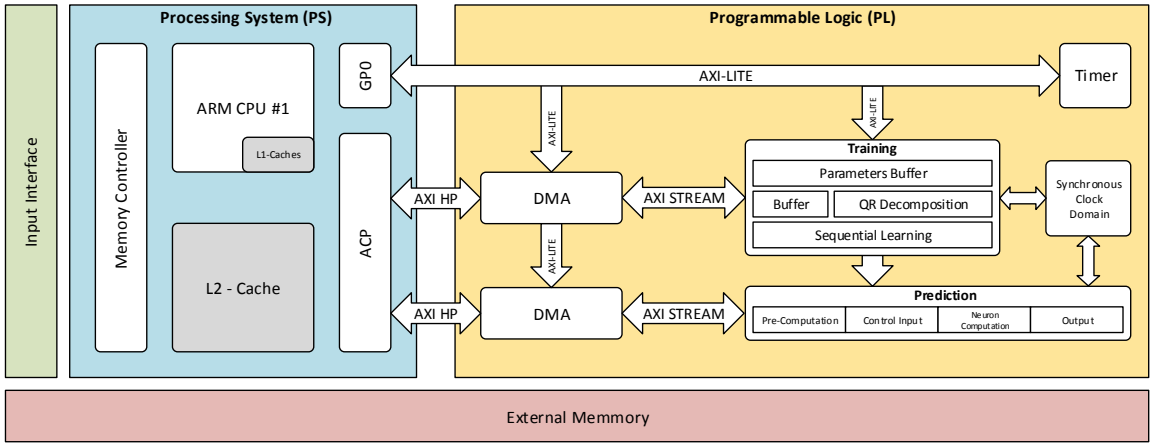


Figure 6.2 – Block diagram of the proposed system.

consist of four submodules: a pre-computation submodule, a control-input submodule, a neuron computation submodule and an output submodule.

- **Memory:** The memory consists of DMA and a shared memory L2 cache to store the OS-ELM parameters.

- **Communication:** The AMBA AXI version 4 protocol is the main interface for the communication of the target IP with other peripherals or processors. The AXI interface consists of four 64-/32-bit high-performance AXI (HP AXI) slave interfaces, two 32-bit AXI master interfaces, two 32-bit AXI slave interfaces, a 64-bit AXI accelerator coherency port (ACP) and an extended multiplexed I/O (EMIO) interface.

- **Clock domain synchronization:** Since the training and prediction modules operate in different clock domains, synchronization between them is required.

In a traditional accelerator system, separate chips, such as DSPs, FPGAs and GPUs, work along with the host processor as co-processors. The host processor manages the control signals and transfers data to the shared memory. Each co-processor is triggered by the host processor to execute its operation on the shared data. The

output of the algorithm is then sent back to the shared memory. In the final stage, the host processor reads the content of the shared memory and transfers it to the output port. All transactions are controlled and monitored by the host processor. However, this approach is inefficient and leads to high overhead. In the new architecture used in this study, the shared memory between the host processor and the accelerator is an L2 cache with low latency. An ACP serves as the interface between the AXI4-Stream interconnect and the host processor (ARM CPU). The ACP is a 64-bit AXI slave interface on the snoop control unit (SCU), which provides an asynchronous cache-coherent access point directly from the PL unit to the host processor subsystem. The ACP provides a low-latency path between the PS unit and the accelerator implemented in the PL unit. The host processor needs only to initialize the cache, after which the input data are transferred in a completely independent manner to the DMA in the PL unit. As a result, more cycles remain available on the host processor for higher-level processing.

6.3.1 Training Module

The proposed OS-ELM implementation operates in two clock domains. The first clock domain is used to train the OS-ELM. As mentioned before, the training process consists of two phases: the initialization phase and the sequential learning phase. In the initialization phase, after the input data are transferred from the shared memory to the DMA, the circuit illustrated in figure 6.3 is used to execute the MGS algorithm to find the matrices Q and R. The circuit consists of three shift registers that store the results of the operations u_i, $< u_i.u_i >$ and $< u_i.\varphi_n >$. Each cell of the shift registers is updated in parallel, which saves clock cycles for further operations. After the update, the content of each cell is passed to the temporary register for division and subtraction. The obtained result u_i is sent back to the second shift register to compute u_{i+1}. MUX_1 is used to control the input to the second shift register since during the initialization phase, it is necessary for φ_1 to be passed directly to SHR_2.

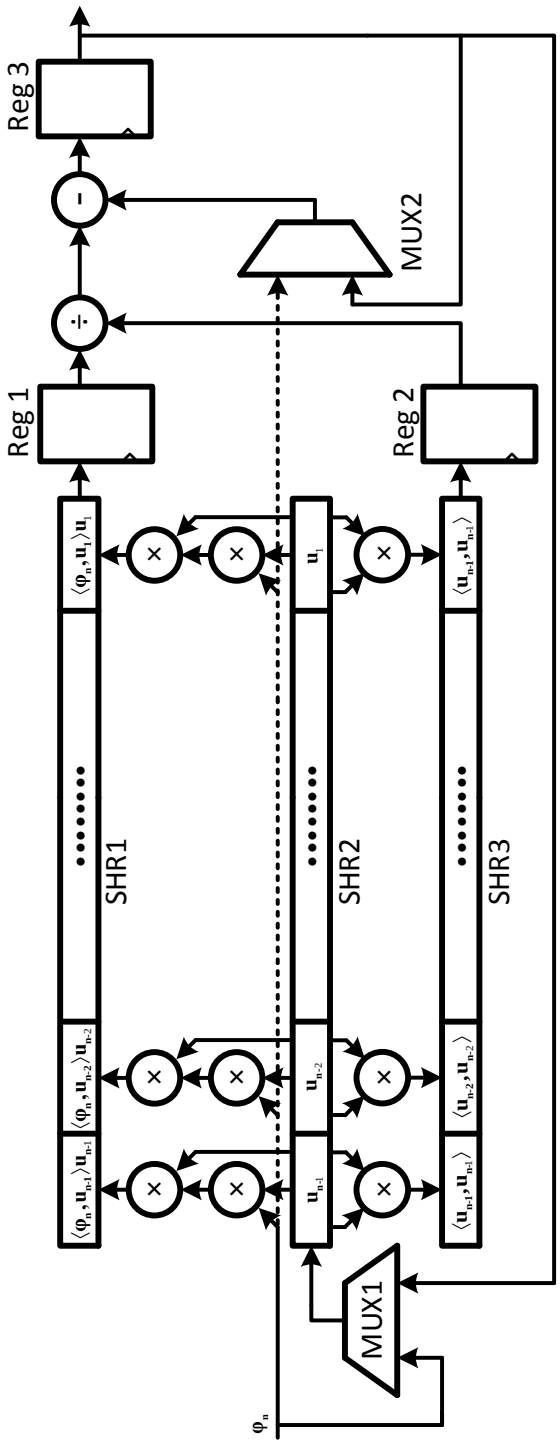


Figure 6.3 – Schematic diagram of the MGS-based Q and R matrix computation.

MUX_2 controls the input for the subtraction operation. $\frac{\langle \varphi_n, u_1 \rangle}{\langle u_1, u_1 \rangle} u_1$ is subtracted from φ_n first, and the remaining terms are then subtracted (equation 6.15). The circuit depicted in figure 6.3 generates $\{u_1, u_2, ..., u_n\}$ as output, which can be used to find $[e_1|e_2|...|e_n]$. A square root operation is required to find e_i, and it is implemented by an arithmetic unit IP in Vivado. The arithmetic unit generates each element of the matrix Q as its output. The matrix Q is transferred to the shared L2 memory cache, and the host processor computes the matrix R and, finally, $\varphi^{\dagger}$. The second phase is the sequential learning phase. When new observation data arrive, it is necessary to recompute the hidden layer output matrix φ and the output weight matrix ω. Equations 6.35 to 6.39 represent this procedure. By considering equations 6.38 and 6.39, we find that multiple matrix operations may affect the performance of the algorithm in terms of speed and latency. Consequently, the performance of the algorithm could be improved by improving the computation of equations 6.38 and 6.39. For a more thorough analysis and understanding, we rewrite equations 6.38 and 6.39 using equations 6.41 and 6.42 below.

$$A = \varphi_{k+1} \quad B = p_k \quad C = p_{k+1} \quad D = \omega_k \quad E = y_{k+1} \tag{6.40}$$

$$C = B - BA^T(I + ABA^T)^{-1}AB \tag{6.41}$$

$$F = D + CA^T(E - AD) \tag{6.42}$$

Equations 6.41 and 6.42 can be divided into suboperations T_x for pipelining as follows:

$$C = B - \overbrace{\underbrace{BA^T}_{T_1}(I + A\underbrace{BA^T}_{T_1})^{-1}\underbrace{AB}_{T_1}}^{T_{5-6}} \tag{6.43}$$

$$\overbrace{\underbrace{BA^T}_{T_1}(I + A\underbrace{BA^T}_{T_1})^{-1}\underbrace{AB}_{T_1}}^{T7}$$

$$\underbrace{(I + A\underbrace{BA^T}_{T_1})}_{T_2}$$

$$\underbrace{(I + A\underbrace{BA^T}_{T_1})^{-1}}_{T_3}$$

$$\underbrace{(I + A\underbrace{BA^T}_{T_1})^{-1}\underbrace{AB}_{T_1}}_{T_4}$$

$$F = D + \overbrace{\underbrace{CA^T}_{T_8}(E - \underbrace{AD}_{T_1})}^{T_9} \tag{6.44}$$

$$\overbrace{\underbrace{CA^T}_{T_8}(E - \underbrace{AD}_{T_1})}^{T_{10}}$$

$$\underbrace{(E - \underbrace{AD}_{T_1})}_{T_2}$$

By dividing each equation in this way, the matrix operations can be pipelined at different levels for high throughput. At the subsystem level, equations 6.43 and 6.44 consist of three functional blocks responsible for the execution of different operations: *multiplication*, *summation* and *inversion*. Ping-pong memory is introduced between the blocks to parallelize these functions (Figure 6.4).

Vectorization is employed for each matrix operation within each of the functional blocks to facilitate a pipelined multiply-and-accumulate architecture. Operations of this type can be performed using either fixed-point or floating-point variables; for floating-point variables in particular, decimal point alignment, rounding and exponent normalization are required, and these additional operations add extra clock cycles. Consider the implementation of $w = x_1y_1+x_2y_2+x_3y_3$ such that all of the variables are fixed-point variables. With the application of a multiply-and-accumulate architecture, each operation can be implemented by one DSP for pipelined calculations. However, for floating-point operations, as mentioned above, additional clock cycles are required, which leads to a significant degradation in efficiency. Therefore, in this work, we define all variables as fixed-point variables for better performance. This architecture works by vectorizing the operations and defining an intermediate variable $z = x_1y_1 + x_2y_2$. This means that instead of calculating one individual w or z, it can more efficiently

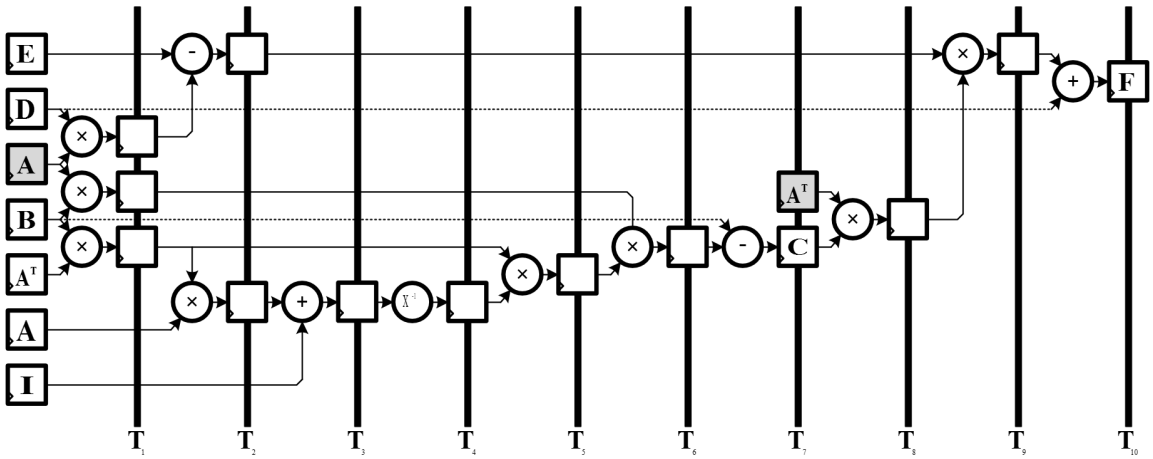


Figure 6.4 – Sequential phase computation equations pipelined architecture

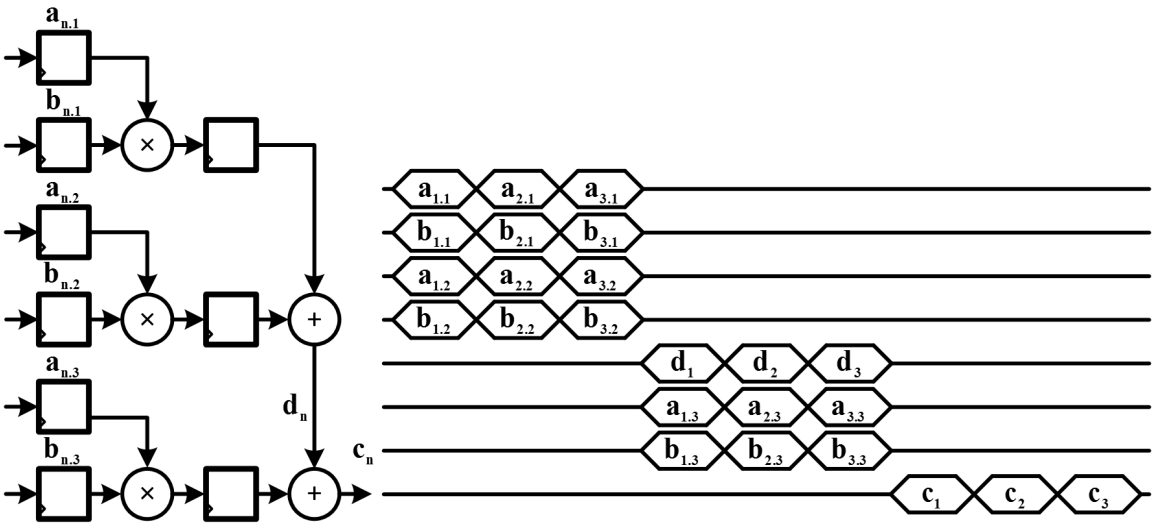


Figure 6.5 – Pipelined architecture for vectorized floating-point/fixed-point multiply-and-add operations

calculate a vector of both (Figure 6.5):

$$z_1 = x_{1,1}y_{1,1} + x_{1,2}y_{1,2}$$

$$z_2 = x_{2,1}y_{2,1} + x_{2,2}y_{2,2}$$

$$z_3 = x_{3,1}y_{3,1} + x_{3,2}y_{3,2}$$

$$\cdots\cdots\cdots$$

$$w_1 = z_1 + x_{1,3}y_{1,3}$$

$$w_2 = z_2 + x_{2,3}y_{2,3}$$

$$w_3 = z_3 + x_{3,3}y_{3,3}$$

The output of the multiply-and-accumulate module can then be passed to the previously discussed QR decomposition block for matrix inversion. The performance validation will be discussed in the Section 6.4.

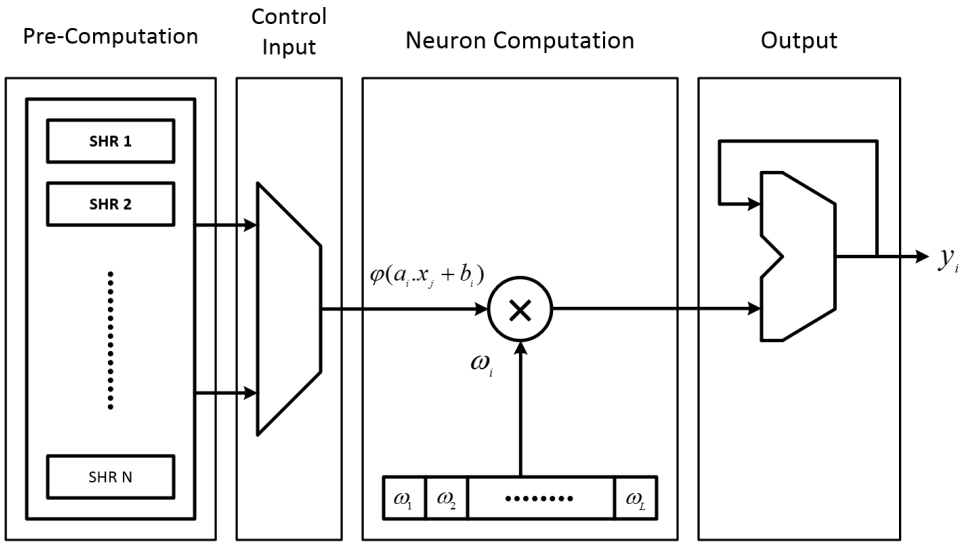


Figure 6.6 – Prediction circuit.

6.3.2 Prediction Module

After training, once all parameters have been obtained, the circuit depicted in figure 6.6, which operates in the second clock domain, is used for prediction. This circuit consists of four main submodules: the *pre-computation* submodule, the *control-input* submodule, the *neuron computation* submodule and the *output* submodule. The first stage consists of multiple shift registers, each of which generates a vector for the i^{th} neuron:

$$[(a_i.x_1 + b_i), (a_i.x_2 + b_i), \ldots, (a_i.x_N + b_i)]. \tag{6.45}$$

The input to each neuron is selected by a MUX, and the output of each neuron is computed as $\varphi_i = g(a_i, b_i, x_{j=1\ldots N})$ and is passed to the output stage to compute $y_j = \sum_{i=1}^{L} \omega_i \varphi(a_i.x_j + b_i), \quad j = 1, \ldots, N$.

6.3.3 Clock Domain Synchronization

When a signal is transferred between circuits in unrelated or asynchronous clock domains, it is necessary to synchronize this signal to the new clock domain before it

can be used. Since the training and prediction modules operate in two different clock domains and the updated algorithm parameters ω and φ need to be passed to the prediction circuit, synchronization between the two blocks is required. One common solution is to use a FIFO approach; however, a FIFO scheme requires considerable resources for implementation. Our solution is to use both control and data paths. As represented in figure 6.7, the control path is flop-synchronized, whereas a synced-in control signal is used to synchronize the data path. A controlled synchronizer MUX is used to allow the data path to cross between the clock domains. The first flip-flop in the new clock domain acts as a synchronization register. In asynchronous signal transfer, to minimize failures due to metastability, a sequence of flip-flops in the destination clock domain is used to resynchronize the signal to the new clock domain. These flip-flops provide additional time for a potentially metastable signal to resolve to a known value before the signal is used in the rest of the circuit. The timing slack in the synchronization register-to-register paths is the time available for a metastable signal to settle and is known as the available metastability settling time.

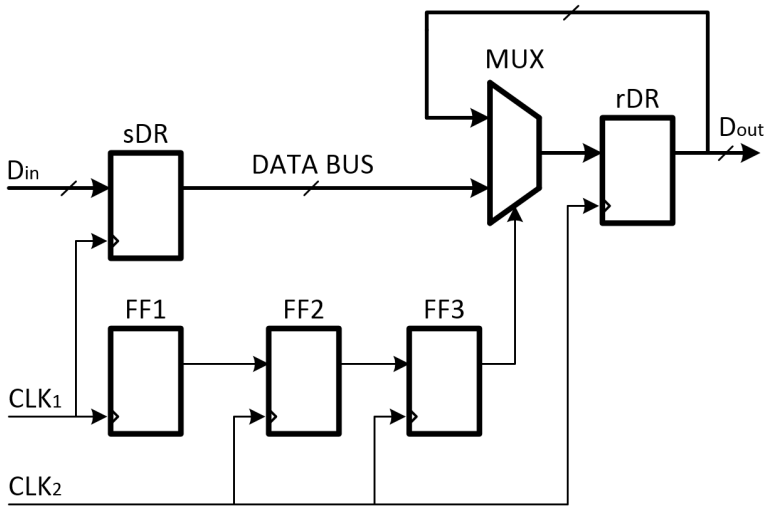


Figure 6.7 – Pipelined architecture for vectorized floating-point/fixed-point multiply-and-add operations

6.4 Performance Analysis and Evaluation

This section discusses the results of the implementation of the proposed architecture on the Xilinx Zynq platform. For the performance evaluation, we used a housing dataset that is commonly used to evaluate machine learning algorithms; it is available on the ELM website [44]. The housing dataset consists of 13 fields and 500 samples. Half of the dataset was used for training, and the rest was reserved for prediction (validation).

The performance of the proposed system was evaluated from four perspectives: *accuracy*, *hardware performance*, *system profile* and *hardware resources*. In terms of accuracy, the performances of several fundamental components, such as the matrix decomposition operation, were compared with those of a PC-based implementation. In addition, the number of bits required to achieve acceptable accuracy was considered. With regard to hardware performance, we investigated the performance of the system in terms of power consumption and the maximum frequency of operation.

For an efficient embedded design that uses an accelerator to achieve better performance, the different contributions from hardware and software must be considered. It is necessary to identify how hardware acceleration can improve the performance for a task that could be run by a processor as software. This is why profiling tools and algorithms have become so important, because they allow one to determine which part of an application constitutes the software bottleneck and how the performance can be improved through acceleration. With regard to the system profile, the performance of the proposed system was characterized in terms of the acceleration factor, and finally, the utilized hardware resources were assessed.

6.4.1 Accuracy Analysis

As mentioned, the matrix decomposition operation is the fundamental component that defines the performance and accuracy of the OS-ELM implementation. There-

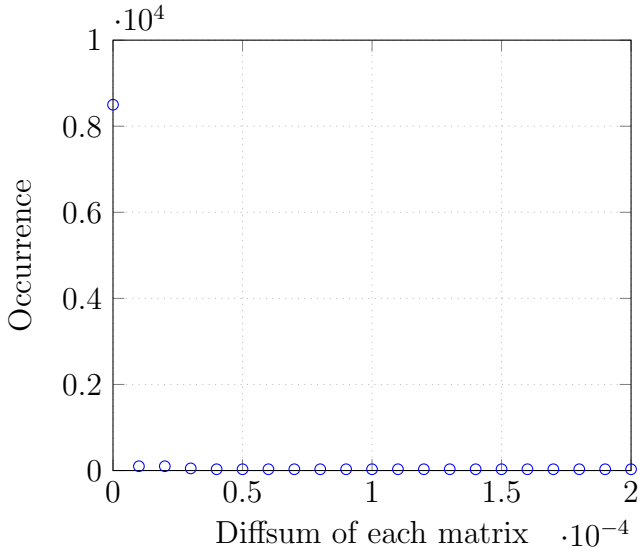


Figure 6.8 – Histogram of errors for 10,000 random 32 × 32 matrices.

fore, the matrix inversion operation must be carefully validated to ensure that the numerical errors are within an acceptable range. The matrix inversion output was compared with the output of the MATLAB `inv()` function, and for validation, the square root of the mean square error of the matrix elements was selected as the performance metric. This performance metric is defined as shown in equation 6.46, where the $x_{(i,j)}$ are the elements of the inverted matrix and the $y_{(i,j)}$ are the reference matrix elements.

$$Error = \sqrt{\frac{\Sigma_i \Sigma_j |x_{(i,j)} - y_{(i,j)}|^2}{\Sigma_i \Sigma_j |x_{(i,j)}|^2}} \tag{6.46}$$

The histogram plotted in figure 6.8 illustrates the matrix inversion performance. The function was tested on 10,000 random matrices, and the output of the module showed small errors close to zero. In some tests, the output of the module had larger errors; however, these errors remained within 0.01% of the data magnitude.

The second step of the accuracy analysis was to evaluate the performance of the OS-ELM algorithm with different numbers of hidden cells and different numbers of bits. In this scenario, the hardware implementation was evaluated through a direct comparison with an implementation of the OS-ELM algorithm that was coded in

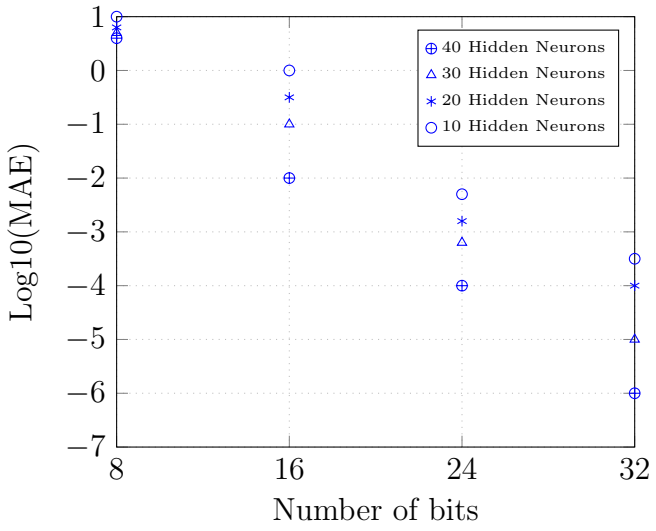


Figure 6.9 – Mean absolute validation error (MAE) versus bit length.

C++ in Visual Studio, with all variables treated as floating-point variables. The mean absolute error (MAE) is commonly used to evaluate the performance of an algorithm; it is obtained by applying a training dataset and then evaluating the results of the trained algorithm on a validation dataset. The algorithm was tested with 10, 20, 30 and 40 cells and with 8, 16, 24 and 32 bits. The evaluation results are presented in figure 6.9. According to this figure, better accuracy can be achieved by increasing the numbers of cells and bits. Since the Zynq SoC has a 32-bit architecture, acceptable accuracy could be achieved in our implementation by considering 32-bit variables with 30-40 hidden cells.

6.4.2 Hardware Performance Analysis

The hardware performance of the proposed architecture was evaluated in terms of the maximum clock frequency, the number of clock cycles required for computation and the power consumption.

For a thorough evaluation, we again considered variables with different numbers of bits. The number of clock cycles increases as the numbers of bits and hidden

cells increase. Figures 6.10 and 6.11 present the number of clock cycles required for computation as a function of the number of bits and as a function of the number of hidden cells, respectively. To compare the proposed architecture with other recently published OS-ELM/ELM implementations [5], we also include the results of three other implementations in figures 6.10 and 6.11. All of the previous implementations and the proposed architecture were tested on the same dataset.

The maximum operation frequency of the synthesized circuit depends on parameters such as the numbers of hidden cells and variable bits; as the number of bits increases, more clock cycles are required for multiplication, and the the number of DSP blocks utilized also increases. Figure 6.12 presents the results for the maximum frequency of operation.

The total power consumption analysis was divided into analyses of static and dynamic power. The power consumption was measured using the Xilinx X-Power Analyzer. The proposed architecture consumes 1.423 W of on-chip power, where 92% of the total power is related to the dynamic power consumption, the processor consumes the majority of the power (88%, or 1.152 W), and the remaining blocks consume the remaining 12% (0.157 W) of the total power.

Although a direct power comparison is difficult because of the different assumptions and structures of different implementations, for a recent OS-ELM/ELM implementation presented in [5], an average value of 1.575 W was reported for 32 bits and 40 hidden cells. A comparison with this result indicates that the proposed architecture offers much better power efficiency.

6.4.3 System Profile

The performance of the system was also characterized in terms of the acceleration factor. An embedded system that uses a second chip on the platform as an accelerator can be quantitatively evaluated in terms of an acceleration factor defined based on the time required for the host processor to execute a single operation compared with

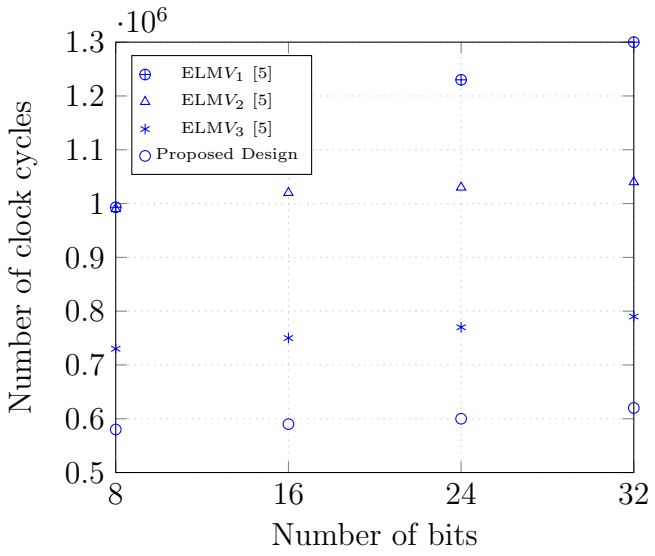


Figure 6.10 – Numbers of clock cycles needed to compute ELM results with 40 hidden cells and variables with different numbers of bits.

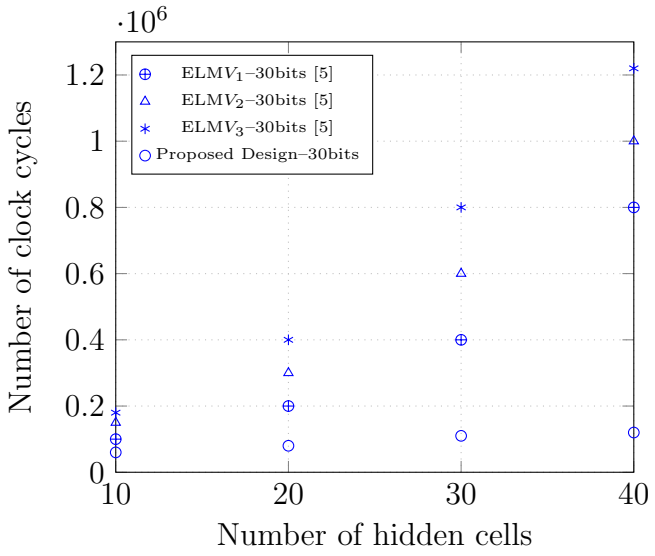


Figure 6.11 – Numbers of clock cycles needed to compute ELM results with different numbers of hidden cells and 30-bit fixed-point variables.

the time required for the execution of the same operation when processed by the accelerator. In this work, as discussed previously, one of the most resource-consuming operations is the matrix decomposition computation that is performed to obtain the

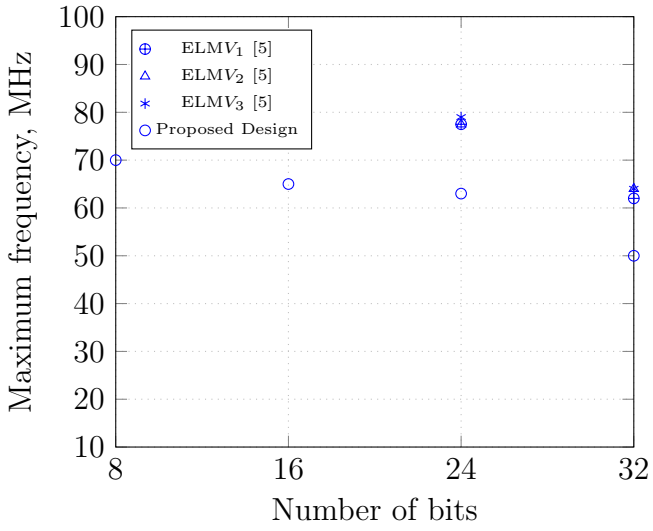


Figure 6.12 – Maximum frequency of operation (reference [5] provides no information about the maximum frequencies for 8-bit implementations).

hidden layer output φ; therefore, we investigated the acceleration factor for this operation for a single chunk of training data. The measurement circuit is illustrated in figure 6.2. The circuit includes a timer that is coded in the PL block. For the first measurement, the ARM processor transfers the contents of a single chunk of training data to the cache, and before the matrix inversion operation is performed, it triggers the timer. For the second measurement, the same operation is executed by the PL block. We measured an acceleration factor of 200x. This test algorithm for measuring the acceleration factor was proposed by Xilinx [45] for evaluating the effectiveness of hardware acceleration.

6.4.4 Hardware Resources

The resource utilization of both the training and prediction modules was evaluated (Table 6.1). Based on the internal blocks of the device, the analysis resources include look-up tables (LUTs), block RAM (BRAM) and DSP blocks (DSP48E). The resource utilization varies based on the bit length of the variables and the number of hidden

cells. The levels of LUT, BRAM and DSP48E utilization increase as the bit length increases; however, the trend for DSP48E utilization is different than that for LUT and BRAM utilization. The resource utilization analysis reveals that the levels of LUT and BRAM utilization increase in proportion to the bit length, whereas since the input to a DSP48E block has a fixed bit length (Figure 6.13), the level of DSP48E resource utilization remains constant until another unit is needed, at which time the resource consumption doubles. This means that when a greater bit length is necessary, more DSP48E blocks are added and synthesized to compose an arithmetic unit with a longer bit length. On the other hand, the utilization of DSP blocks is independent of the number of hidden cells; since the transfer of data from memory to the training and prediction modules is serial in nature, no additional resources are required as the number of hidden cells increases. By contrast, the RAM utilization grows as the number of hidden cells increases, but not proportionally. Instead, the increase in RAM utilization with an increasing number of hidden cells follows the same trend as the increase in DSP block utilization with increasing bit length. The Xilinx SoC FPGA has a new BRAM architecture. Each block of RAM has independent control over its address depth and can store up to 36 kbits (Figure 6.14). It can also be configured as either two independent 18 kb RAMs or one 36 kb RAM. In this case,

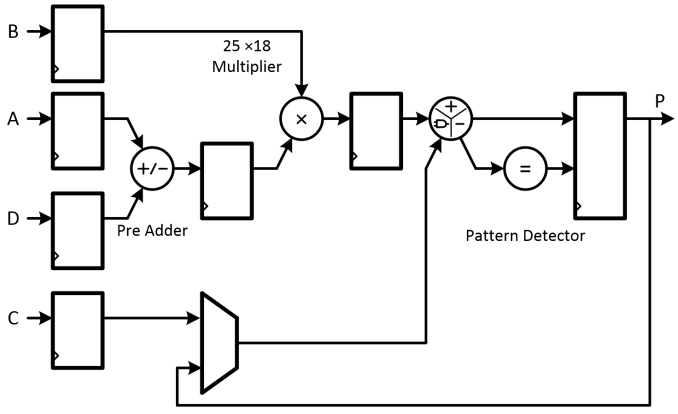


Figure 6.13 – DSP48E

Table 6.1 – Resource utilization percentages (LUTs, DSP48E, and BRAM) for general logic implementations

Look-Up Tables (LUTs) (%)							
				Bit length			
Algorithm	Family	Device	LUTs	8	16	24	32
$ELMV_1$ [5]	Virtex6	XC6VCX75TL-1L	11,640×4	N.A.	3259 (7%)	4190 (9%)	4656 (10%)
$ELMV_2$ [5]	Virtex6	XC6VCX75TL-1L	11,640×4	N.A.	3724 (8%)	4656 (10%)	6052 (13%)
$ELMV_3$ [5]	Virtex6	XC6VCX75TL-1L	11,640×4	N.A.	3724 (8%)	4656 (10%)	6052 (13%)
Proposed Method	Zynq 7000	XC7Z020-CLG484	53,200	2128 (4%)	3192 (6%)	4256 (8%)	5320 (10%)

DSP48E (%)							
				Bit length			
Algorithm	Family	Device	DSP48E	8	16	24	32
$ELMV_1$ [5]	Virtex6	XC6VCX75TL-1L	288	N.A.	16 (7%)	20 (9%)	23 (10%)
$ELMV_2$ [5]	Virtex6	XC6VCX75TL-1L	288	N.A.	18 (8%)	23 (10%)	29 (13%)
$ELMV_3$ [5]	Virtex6	XC6VCX75TL-1L	288	N.A.	18 (8%)	23 (10%)	29 (13%)
Proposed Method	Zynq 7000	XC7Z020-CLG484	220	9 (4%)	13 (6%)	17 (8%)	22 (10%)

Block RAM Logic (BRAM) (%)							
				Bit length			
Algorithm	Family	Device	BRAM	8	16	24	32
$ELMV_1$ [5]	Virtex6	XC6VCX75TL-1L	5.9 MB (156×36 kB)	N.A.	1684 (30%)	2246 (40%)	2864 (51%)
$ELMV_2$ [5]	Virtex6	XC6VCX75TL-1L	5.9 MB (156×36 kB)	N.A.	1235 (22%)	1684 (30%)	2246 (40%)
$ELMV_3$ [5]	Virtex6	XC6VCX75TL-1L	5.9 MB (156×36 kB)	N.A.	1740 (31%)	2302 (41%)	2976 (53%)
Proposed Method	Zynq 7000	XC7Z020-CLG484	4.9 MB (140×36 kB)	504 (10%)	907 (18%)	1310 (26%)	1814 (36%)

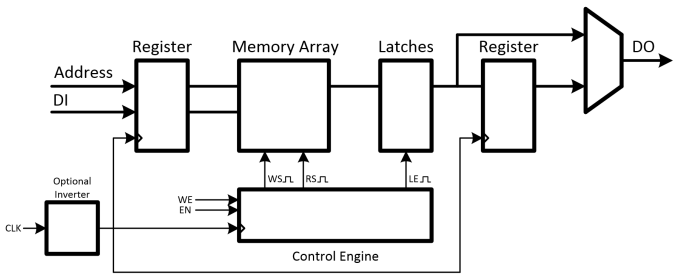


Figure 6.14 – Block RAM logic diagram

when the bit length surpasses the maximum, additional RAM blocks are needed to increase the bit length.

6.5 OS-ELMs and Real-Time Applications

In the previous section, we compared the performance of the proposed design with the performances of other recent proposals. However, a direct comparison of the results of our implementation with those of other works is difficult because previous works have considered different test datasets, different numbers of hidden layers, and different types of variables (fixed point or floating point). In addition, different solutions have been defined for different applications. As mentioned previously, the research presented in this study is mainly focused on embedded systems for human action recognition, which requires the system to be capable of both training and prediction. In this work, we tested our algorithm on two recent challenging datasets: 3D Hollywood [12] and HON4D [13]. The data format for this implementation consisted of

Table 6.2 – FPGA resources consumed for OS-ELM/ELM implementations

Algorithm	Target FPGA	LUT	DSP48E	BRAM
ELM	Zynq 7000 (XCZ020)	15488/53200 (29%)	81/220 (36%)	96×36kB/140×36kB (68.5%)
OS-ELM	Zynq 7000 (XCZ020)	12488/53200 (23%)	64/220 (29%)	52×36kB/140×36kB (37.1%)
Percentage of resource reduction		**-19%**	**-21%**	**-46%**

fixed-point variables with a 64-bit width. The classification accuracy was 12.2% for the 3D Hollywood dataset and 72.40% for the HON4D dataset. For comparison, the highest accuracies that have been achieved on these datasets are 13.3% for the 3D Hollywood dataset and 88.89% for the HON4D dataset. However, these highest accuracies were achieved on a PC with floating-point variables and operations, whereas in this work, we used fixed-point variables. Table 1 shows the resources utilized for the implementations of the ELM and OS-ELM algorithms. The results presented in table 6.2 show that chunk-by-chunk training can effectively improve the utilization of resources, especially memory.

6.6 Conclusion

In this work, a SoC FPGA-based OS-ELM algorithm is presented. Appropriate circuits are proposed and evaluated for both training and prediction in an embedded system with fixed-point variables of different bit widths and with different numbers of hidden cells. The proposed architecture efficiently utilizes both processor and accelerator resources (FPGA) to achieve better performance and accuracy. Analyses of accuracy and the acceleration factor prove that the proposed architecture for matrix inversion enables effective computation by means of programmable logic. In addition, the proposed architecture for matrix inversion and QR decomposition can be utilized as a separate IP for other applications, such as communication. Furthermore, the power consumption results show that the proposed architecture is appropriate for battery-powered portable systems.

The results of testing the algorithm for application in action recognition show that it achieves acceptable accuracy for real-time applications and the real-world challenges faced in computer vision and video processing.

Future work could include testing the proposed system with more datasets and evaluating the performance of the proposed circuits under different conditions, includ-

ing the use of floating-point variables and different numbers of hidden cells. Finally, the proposed architecture could be implemented for applications in which NNs are used for classification (object detection), regression and recognition (action recognition).

6.7 Summary

Machine learning algorithms, such as those for object classification in images, video content analysis (VCA), and human action recognition, are used to extract meaningful information from data recorded by image sensors and cameras. Among the existing machine learning algorithms, extreme learning machines (ELMs) and online sequential ELMs (OS-ELMs) are well known for their computational efficiency and performance when processing large datasets. The latter approach was derived from the ELM approach and is optimized for real-time application. However, OS-ELM classifiers are computationally demanding, and the existing state-of-the-art computing platforms are not efficient enough for embedded systems, especially for applications with strict requirements in terms of low power consumption, high throughput, and low latency. This chapter presents the implementation of an ELM/OS-ELM in a customized system-on-a-chip field-programmable gate array (SoC FPGA)-based architecture to ensure efficient hardware acceleration. The acceleration process comprises parallel extraction, deep pipelining and efficient shared memory communication.

References

[1] Y.-H. Pao, G.-H. Park, and D. J. Sobajic, "Learning and generalization characteristics of the random vector functional-link net," *Neurocomputing*, vol. 6, no. 2, pp. 163–180, 1994.

[2] E. Cambria, G.-B. Huang, L. L. C. Kasun, H. Zhou, C. M. Vong, J. Lin, J. Yin, Z. Cai, Q. Liu, K. Li *et al.*, "Extreme learning machines [trends & controversies]," *IEEE Intelligent Systems*, vol. 28, no. 6, pp. 30–59, 2013.

[3] Y. Xu, J. Jiang, J. Jiang, Z. Liu, and J. Xu, "Fixed-point evaluation of extreme learning machine for classification," in *Proceedings of ELM-2015 Volume 1*. Springer, 2016, pp. 27–38.

[4] M. Bataller-Mompeán, J. M. Martínez-Villena, A. Rosado-Muñoz, J. V. Francés-Víllora, J. F. Guerrero-Martínez, M. Wegrzyn, and M. Adamski, "Support tool for the combined software/hardware design of on-chip elm training for slff neural networks," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 3, pp. 1114–1123, 2016.

[5] J. V. Frances-Villora, A. Rosado-Muñoz, J. M. Martínez-Villena, M. Bataller-Mompean, J. F. Guerrero, and M. Wegrzyn, "Hardware implementation of real-time extreme learning machine in fpga: Analysis of precision, resource occupation and performance," *Computers & Electrical Engineering*, vol. 51, pp. 139–156, 2016.

[6] M. Van Heeswijk, Y. Miche, E. Oja, and A. Lendasse, "Gpu-accelerated and parallelized elm ensembles for large-scale regression," *Neurocomputing*, vol. 74, no. 16, pp. 2430–2437, 2011.

[7] S. Li, X. Niu, Y. Dou, Q. Lv, and Y. Wang, "Heterogeneous blocked cpu-gpu accelerate scheme for large scale extreme learning machine," *Neurocomputing*, 2017.

[8] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*. ACM, 2012, pp. 47–56.

[9] Xilinx. (2017, Aug.) Expanding the all programmable soc portfolio. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc.html

[10] Intel. (2017, Aug.) Intel socs: When architecture matters. [Online]. Available: https://www.altera.com/products/soc/overview.html

[11] S. Hadfield and R. Bowden, "Hollywood 3d: Recognizing actions in 3d natural scenes," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE, 2013, pp. 3398–3405.

[12] S. Hadfield, K. Lebeda, and R. Bowden, "Hollywood 3d: What are the best 3d features for action recognition?" *International Journal of Computer Vision*, pp. 1–16, 2016.

[13] O. Oreifej and Z. Liu, "Hon4d: Histogram of oriented 4d normals for activity recognition from depth sequences," in *Proc. Computer Society Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2013, pp. 716–723.

[14] X. Zhang, M. McKenna, J. P. Mesirov, and D. L. Waltz, "The backpropagation algorithm on grid and hypercube architectures," *Parallel Computing*, vol. 14, no. 3, pp. 317–327, 1990.

[15] S. Shams and J.-L. Gaudiot, "Implementing regularly structured neural networks on the dream machine," *IEEE transactions on neural networks*, vol. 6, no. 2, pp. 407–421, 1995.

[16] V. Kumar, S. Shekhar, and M. B. Amin, "A scalable parallel formulation of the backpropagation algorithm for hypercubes and related architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 10, pp. 1073–1090, 1994.

[17] R. Östermark, "A flexible multicomputer algorithm for artificial neural networks," *Neural Networks*, vol. 9, no. 1, pp. 169–178, 1996.

[18] A. Petrowski, "Choosing among several parallel implementations of the backpropagation algorithm," in *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, vol. 3. IEEE, 1994, pp. 1981–1986.

[19] S. K. Foo, P. Saratchandran, and N. Sundararajan, "Parallel implementation of backpropagation neural networks on a heterogeneous array of transputers," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 27, no. 1, pp. 118–126, 1997.

[20] H. Paugam-Moisy, "Optimal speedup conditions for a parallel back-propagation algorithm," *Parallel Processing: CONPAR 92VAPP V*, pp. 719–724, 1992.

[21] J. Kennedy and J. Austin, "A parallel architecture for binary neural networks," in *Proceedings of the 6th International Conference on Microelectronics for Neural Networks, Evolutionary & Fuzzy Systems (MICRONEURO97)*, 1997, pp. 225–232.

[22] U. A. Muller, A. Gunzinger, and W. Guggenbuhl, "Fast neural net simulation with a dsp processor array," *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 203–213, 1995.

[23] T. Nordström and B. Svensson, "Using and designing massively parallel computers for artificial neural networks," *Journal of Parallel and Distributed Computing*, vol. 14, no. 3, pp. 260–285, 1992.

[24] N. Avellana, A. Strey, R. Holgado, J. Fernandes, R. Capillas, and E. Valderrama, "Design of a low-cost and high-speed neurocomputer system," in *Microelectronics for Neural Networks, 1996., Proceedings of Fifth International Conference on.* IEEE, 1996, pp. 221–226.

[25] S. Shams and J.-L. Gaudiot, "Parallel implementations of neural networks," *International Journal on Artificial Intelligence Tools*, vol. 2, no. 04, pp. 557–581, 1993.

[26] W. Eppler, T. Fisher, H. Gemmeke, T. Becher, and G. Kock, "High speed neural network chip on pci-board," in *Proc. MicroNeuro*, 1997, pp. 9–17.

[27] J. Wawrzynek, K. Asanovic, and N. Morgan, "The design of a neuro-microprocessor," *IEEE Transactions on Neural Networks*, vol. 4, no. 3, pp. 394–399, 1993.

[28] E. Yao and A. Basu, "Vlsi extreme learning machine: A design space exploration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 1, pp. 60–74, 2017.

[29] M. Nabiyouni and D. Aghamirzaie, "A highly parallel multi-class pattern classification on gpu," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on.* IEEE, 2012, pp. 148–155.

[30] T. Jeowicz, P. Gajdo, V. Uher, and V. Snáel, "Classification with extreme learning machine on gpu," in *Intelligent Networking and Collaborative Systems (INCOS), 2015 International Conference on.* IEEE, 2015, pp. 116–122.

[31] A. Basu, S. Shuo, H. Zhou, M. H. Lim, and G.-B. Huang, "Silicon spiking neurons for hardware implementation of extreme learning machines," *Neurocomputing*, vol. 102, pp. 125–134, 2013.

[32] S. Decherchi, P. Gastaldo, R. S. Dahiya, M. Valle, and R. Zunino, "Tactile-data classification of contact materials using computational intelligence," *IEEE Transactions on Robotics*, vol. 27, no. 3, pp. 635–639, 2011.

[33] S. Decherchi, P. Gastaldo, A. Leoncini, and R. Zunino, "Efficient digital implementation of extreme learning machines for classification," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 59, no. 8, pp. 496–500, 2012.

[34] S. Decherchi, P. Gastaldo, R. Zunino, E. Cambria, and J. Redi, "Circular-elm for the reduced-reference assessment of perceived image quality," *Neurocomputing*, vol. 102, pp. 78–89, 2013.

[35] G. Huang, G.-B. Huang, S. Song, and K. You, "Trends in extreme learning machines: A review," *Neural Networks*, vol. 61, pp. 32–48, 2015.

[36] J. Cao, Z. Lin, G.-B. Huang, and N. Liu, "Voting based extreme learning machine," *Information Sciences*, vol. 185, no. 1, pp. 66–77, 2012.

[37] J. Tang, C. Deng, and G.-B. Huang, "Extreme learning machine for multilayer perceptron," *IEEE transactions on neural networks and learning systems*, vol. 27, no. 4, pp. 809–821, 2016.

[38] X.-Z. Wang, R. Wang, and C. Xu, "Discovering the relationship between generalization and uncertainty by incorporating complexity of classification," *IEEE transactions on cybernetics*, vol. 48, no. 2, pp. 703–715, 2018.

[39] Z. Huang, Y. Yu, J. Gu, and H. Liu, "An efficient method for traffic sign recognition based on extreme learning machine," *IEEE transactions on cybernetics*, vol. 47, no. 4, pp. 920–933, 2017.

[40] J. E. Gentle, *Matrix algebra: theory, computations, and applications in statistics*. Springer Science & Business Media, 2007.

[41] Å. Björck, "Solving linear least squares problems by gram-schmidt orthogonalization," *BIT Numerical Mathematics*, vol. 7, no. 1, pp. 1–21, 1967.

[42] H. T. Huynh and Y. Won, "Regularized online sequential learning algorithm for single-hidden layer feedforward neural networks," *Pattern Recognition Letters*, vol. 32, no. 14, pp. 1930 – 1935, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865511002297

[43] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks," *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1411–1423, 2006, cited By 736.

[44] Extreme Learning Machines (ELM) @ONLINE. http://www.ntu.edu.sg/home/egbhuang/index.html. Accessed: 2018-02-1.

[45] *A Zynq Accelerator for Floating Point Matrix Multiplication Designed with Vivado HLS*, Xilinx, 1 2016, rev. 2.0.

Chapter 7

Conclusions

7.1 Summary of the Thesis

This dissertation has described a SoC-based hardware acceleration implementation for use in accelerating computer vision algorithms in a SoC-FPGA system. As part of the design approach, four core components were developed for the implementation of a human action recognition algorithm.

Chapters 1 and 2 discussed the SoC concept and the advantages of using a SoC-FPGA accelerator in combination with other devices. The fundamental components, such as internal communication interfaces (AXI interfaces), needed to provide communication between each of the synthesized modules were covered. In addition, the proposed framework for human action and activity recognition, which was implemented on a Xilinx Zynq SoC FPGA, was discussed. The evaluation method used to measure the acceleration performance in terms of the acceleration factor was discussed in chapter 2.

In chapter 3, the first component of the framework constituting the first stage of pre-processing was discussed. The foreground and background subtraction module with adaptive updating based on photometric invariant color, depth data and texture information extracted by employing LBP operators was presented. The operation of the algorithm was sped up by implementing time-consuming functions, such as the LBP operations, on an FPGA. Various levels of optimization were also considered, such as reducing the number of bits required for variables, loop optimization and a communication-centric architecture. The implementation and evaluation results

showed that the proposed implementation achieved an acceleration factor of 300x for the LBP computations for a single frame and that the total power consumption for the whole algorithm was 156 mW.

In chapter 4, the second module of the framework was discussed. A human detection algorithm was proposed based on information extracted from both RGB and depth data and features encoded with a covariance descriptor. The implementation and evaluation results showed that the proposed implementation achieved better resource utilization compared with other implementations.

In chapter 5, two algorithms for interest point detection in 3D scenes were discussed: the Harris Corners and Hessian Points operators. An efficient solution for encoding features with the BOW approach was discussed. The implementation results were compared with other state-of-the-art implementations, and the comparisons revealed that the proposed framework outperformed the other approaches in terms of accuracy and resource utilization.

In chapter 6, a different method of classification was discussed as an alternative to the SVM method covered in chapter 5. An ELM is a single-hidden-layer neural network that provides fast training and prediction operations. OS-ELM is a variant of the ELM approach in which training can be performed frame by frame instead of requiring all frames simultaneously. The advantages of this approach are that it provides an ideal classifier for real-time application while utilizing fewer resources. The evaluation results in terms of the accuracy of the algorithm showed that the proposed approach outperformed the other approaches considered for comparison; moreover, between ELM and OS-ELM implementations, the resource utilization of the OS-ELM implementation was 46% lower.

The main challenges addressed in this dissertation in order to architect an efficient solution were as follows: (1) satisfying the immense computational demands of vision algorithms; (2) achieving the ability to execute different vision and machine learning algorithms on single platforms; and (3) integrating the entire process, from capturing

the input frames from an image sensor or video to preprocessing, processing and displaying the results, into a single platform. To properly address the challenges of embedded vision and pave the way toward the embedded realization of advanced vision processing, this dissertation has addressed the following design tasks:

- Developing and implementing core components (IPs) for advanced machine vision algorithms.
- Developing a framework based on the developed core components for advanced machine vision applications: human activity and action recognition.
- Profiling and analyzing the sophisticated software-level operations of each algorithm and component.
- Designing the corresponding hardware accelerator and optimizing it for the mathematical model of each component.
- Finding an optimal solution by balancing the system performance, energy consumption and resource allocation.
- Verifying the operation of each component and the entire framework.
- Each of the core components, namely, foreground and background subtraction, human detection, interest point and feature detection, and OS-ELM classification, could be utilized in other machine vision frameworks.

7.2 Future Work

This dissertation introduces new areas of research and exploration regarding SoC-FPGA acceleration in embedded vision that could be pursued in both academia and industry. The results of this research prove that remarkable opportunities are available for architecting and programming at the function level, enabling a shift from

optimizing individual applications to identifying and concentrating on common functions that are used in many applications and algorithms. Since this research has mainly focused on recognizing human actions by means of a visual sensor, a possible future extension of this approach could be to fuse data from visual and inertial sensors to improve the accuracy of the results.

In addition, a valuable avenue of study could be to compare the training times required for OS-ELM classifiers and other state-of-the-art machine learning algorithms such as DNNs and CNNs and to evaluate their performances. The proposed framework could be utilized for other image processing applications, such as face recognition or gesture recognition. Performance and accuracy evaluations for such applications are also highly recommended. Furthermore, the performance of the proposed framework with different interest point detectors and feature descriptors should be investigated.

Appendix A: IEEE Permission to Reprint

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of University of Windsor's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

Appendix B: Elsevier Permission to Reprint

In reference to Elsevier copyrighted material which is used with permission in this dissertation, the Elsevier does not endorse any of University of Windsor's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing Elsevier copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to the following links to learn how to obtain a License from RightsLink.

https://s100.copyright.com/AppDispatchServlet#formTop

https://www.elsevier.com/about/our-business/policies/copyright#Author-rights

Vita Auctoris

NAME	:	Amin Safaei
BIRTH YEAR	:	1983
BIRTH PLACE	:	IRAN
EDUCATION		
2018	:	**Doctor of Philosophy**
		Electrical and Computer Engineering
		University of Windsor, Windsor
2011	:	**Masters of Applied Science**
		Electrical and Computer Engineering
		Sharif University of Technology
2008	:	**Bachelors of Engineering**
		Electrical and Computer Engineering
		Islamic Azad University