FPGA-Based Hardware Implementation of Image Processing Algorithms for Real-Time Vehicle Detection Applications

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Peng Li

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Hua Tang

Octorbor, 2012

Acknowledgements

There are many people that have earned my gratitude for their contribution to my time in graduate school. Thank you to Dr. Hua Tang for providing me with the direction opportunity to work on the VLSI implementation of the digital image processing algorithms, which is the area I am very interested in. Thank you to my committee, Dr. Debao Zhou and Dr. Jing Bai, for their invaluable support and guidance in finishing the thesis. Thank you to Dr. Mohammed Hasan, I learned a lot from his digital signal processing class and digital image processing class. Thank you to Dr. Imran Hayee, I learned a lot from his modern communication class. Thank you to Dr. Jiann-Shiou Yang, I learned a lot from his digital control class.

Much of the financial resources for this thesis were provided through the Northland Advanced Transportation System Research Laboratory (NATSRL), University of Minnesota, Duluth Campus.

Dedication

This thesis is dedicated to my mother,

Meiji Zhang,

who leads my way when I need help,

and to my father,

Jie Li,

who introduced me to the joy of electrical and computer engineering from birth.

Abstract

It is well known that vehicle tracking processes are very computationally intensive. Traditionally, vehicle tracking algorithms have been implemented using software approaches. The software approaches have a large computational delay, which causes low frame rate vehicle tracking. However, real-time vehicle tracking is highly desirable to improve not only tracking accuracy but also response time, in some ITS (Intelligent Transportation System) applications such as security monitoring and hazard warning. For this purpose, this thesis makes an attempt to design a hardware based system for real-time vehicle detection, which is typically required in the complete tracking system. The vehicle detection systems capture pictures using a camera in real-time and then we apply several image processing algorithms, such as Fixed Block Size Motion Estimation (FBSME), Reconfigurable Block Size Motion Estimation (RBSME), Variable Block Size Motion Estimation (VBSME) and Mixtures of Gaussian, to process these images in real-time for vehicle detection.

We first propose the Very-Large-Scale Integration (VLSI) implementation for RBSME algorithm, which supports arbitrary block size motion estimation. Experiment results show that the proposed architecture achieves the flexibility of adjustable block size at the expense of only 5% hardware overhead compared to the traditional design.

We then propose a low-power VLSI implementation for the VBSME algorithm, which employs a fast full-search block matching algorithm to reduce power consumption, while preserving the optimal solutions [1] . The fast full-search algorithm is based on the comparison of the current minimum Sum of Absolute Difference (SAD) to a conservative lower bound so that unnecessary SAD calculations can be eliminated. We first experimentally decide on the specific conservative lower bound and then implement the fast full-search algorithm in Field-Programmable Gate Array (FPGA). To the best of our knowledge, this is the first time that a fast full-search block matching algorithm is explored to reduce power consumption in the context of VBSME, and designed in hardware. Experiment results show that the proposed hardware implementation can

[1] *The optimal solutions* are the results generated by full search block matching algorithm.

save power consumption by 45% compared to conventional VBSME designs based on the non-fast full-search algorithms.

At last, we propose an System-on-a-Chip (SoC) architecture for an Mixture of Gaussian (MoG) based image segmentation algorithm. The MoG algorithm for video segmentation application is computational intensive. To meet real-time requirement of high frame rate high resolution video segmentation tasks, we present a hardware implementation of the MoG algorithm. Moreover, we integrated the hardware IP into an SoC architecture, so that some key parameters, such as learning rate and threshold, can be configured on-line, which makes the system extremely flexible to adapt to different environments. The proposed system has been implemented and tested on Xilinx XtremeDSP Video Starter Kit Spartan-3ADSP 3400A Edition. Experiment results show that under a clock frequency of 25MHz, this design meets the real-time requirement for Video Graphics Array (VGA) resolution (640×480) at 30 frame-per-second (fps).

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Automatic real-time vehicle tracking in Intelligent Transportation System (ITS) applications require real-time vehicle detection first. Among many existing technologies for real-time vehicle detection such as microwave and radar, the camera-based approach is more popular today. In camera-based systems, real-time vehicle detection is equivalent to real-time image segmentation from image processing point of view. In this thesis, we apply several image processing algorithms and propose the corresponding hardware implementations for camera-based real-time vehicle detection. These algorithms include Fixed Block Size Motion Estimation (FBSME), Reconfigurable Block Size Motion Estimation (RBSME), Variable Block Size Motion Estimation (VBSME), and Mixtures of Gaussian. In the following, we give a brief introduction to these algorithms.

1.1 Reconfigurable Block Size Motion Estimation

Motion estimation technologies has been widely used in video coding systems for data compression. FBSME based on the full search block matching algorithm can be considered as the most popular method for practical motion estimation due to its good quality and regular computation for relatively low complexity of hardware implementation. Other than for video coding, motion estimation is in fact a powerful technique that goes beyond and allows for video sequence analysis. In computational vision and video surveillance applications, motion estimation can be used for camera motion detection, scene identification, object detection, object tracking and object classification. We refer

the readers to [2] on how motion estimation can be used for vehicle tracking.

An illustration of the FBSME algorithm is shown in Fig. 1.1. Suppose that the image size is $M \times S$ (in pixels) and the block size is $n \times n$, then a total of $\frac{M \times S}{n \times n}$ blocks are defined for each image frame (for illustration purpose, Fig. 1.1 shows only 16 blocks for each frame). With respect to two consecutive images, shown as "current frame" (say frame number N) and "previous frame" (frame number $N-1$) in Fig. 1.1, reference block A (the dark shaded region) in "previous frame" can be considered as a moved version of block A in "current frame" and block A is in the neighborhood area of the A. This neighborhood, called search area, is defined by parameter, p, in four directions (up, down, right, and left) from the position of block A. The value of p is determined by the frame rate and object speed. If the frame rate is high, as assumed in the proposed tracking system, then parameter p can be set relatively small since the block A is not expected to move dramatically within a short period of time. In the search area of $(2p+n) \times (2p+n)$, there are a total of $(2p+1) \times (2p+1)$ possible candidate blocks for A and the block A in "current frame" (N) that gives the minimum matching value is the one that originates from block A in "previous frame" $(N-1)$. The most commonly used matching criterion, Sum of Absolute Difference (SAD), is defined as follows:

$$SAD(u,v) = \sum_{i=1}^{n} \sum_{j=1}^{n} |S(i+u, j+v) - R(i,j)|, -p \leq (u,v) \leq p \tag{1.1}$$

where $R(i,j)$ is the reference block of size $n \times n$, $S(i+u, j+v)$ is the candidate block within the search area and (u,v) represents the block motion vector. The motion vectors computed from block matching contain information on how and where each block in the image moves within two consecutive image frames.

We found that vehicle tracking normally requires the flexibility of adjustable block size for motion estimation. In order to achieve the optimal performance, the block size is determined by the resolution of the scene, the vehicle size and the distance of the vehicle from the camera. Motivated by this, we first propose to build a hardware architecture to support RBSME in this thesis.

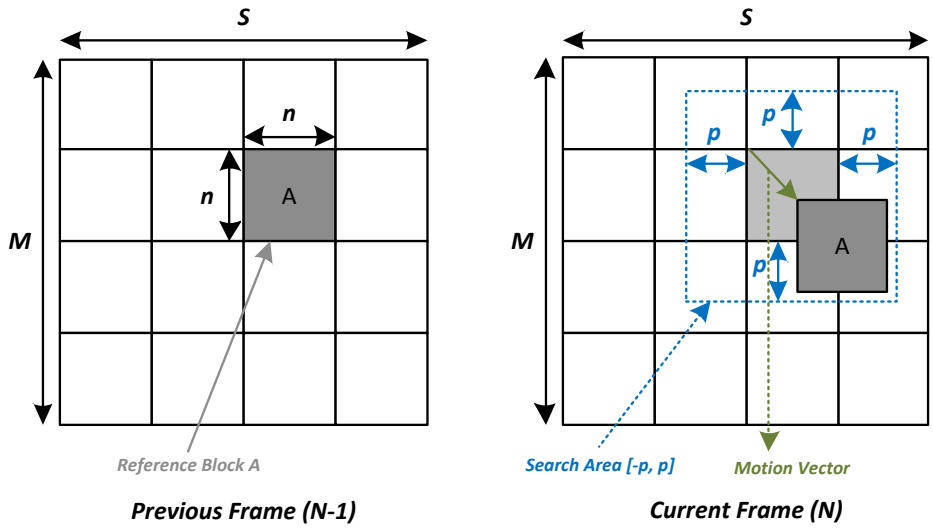

Figure 1.1: FBSME-based full search block matching.

1.2 Variable Block Size Motion Estimation

Although the RBSME-based image segmentation gives us more degree of freedom compared to the FBSME, we found that the segmentation results can be further improved by using the VBSME, which is adopted in the latest video coding standard H.264/AVC [3]. It contains a number of new features that allow it to compress videos much more effectively than the older standards and to provide more flexibility for applications in a wide variety of network environments. One of the key features is the VBSME, in which block sizes range from 4×4 (pixels), 4×8, 8×4, 8×8, 8×16, 16×8 to 16×16. This enables more accurate segmentation of the moving regions.

Research on VBSME has become a hot topic in the past years. One area of the research focuses on algorithm design for efficient and fast VBSME (i.e. reduced computation load with minor quality loss or without quality loss) [4] [5]. The other area focuses on hardware implementation of VBSME with or without fast algorithms [6] [7].

In [8], the authors propose a one-dimensional (1-D) VLSI architecture with 16 Processing Elements (PEs) for full-search VBSME. The main contribution of this architecture is that it can process more Motion Vectors (MVs) than a conventional 1-D architecture in the same number of clock cycles by incorporating a shuffling mechanism

within each PE. However, the performance of this architecture is limited by the 1-D structure. It can support VBSME with only low frame-rate and low resolution requirements. Moreover, the overlapped pixel data between two neighboring candidate blocks in the search area can not be reused, which increases the system's memory bandwidth. The power consumption of the design is very high compared to most other reported designs. The 1D architecture is also used in [9] [10].

In [6], the authors propose a Sums of Absolute Difference (SAD) tree based architecture for VBSME. The processing speed of this architecture is relatively high and it can support (High-Definition TeleVision) HDTV 720p at 30fps with a clock frequency of 108MHz. However, the data reuse scheme adopted by this architecture requires 208kb (kilo bits) on-chip Static Random-Access Memory (SRAM), which is 3 to 10 times of those needed in other designs for VBSME. The corresponding memory bitwidth and bandwidth [1] are very large as well. Overall, the power consumption of this design is expected to be relatively high compared to others.

In [11], the authors propose a memory-efficient hardware architecture for full-search VBSME, which consists of 16 arrays of 16 $\times$ 16 PEs each. The main contribution of this architecture is that it can save 98% of on-chip memory access with only 27% of memory overhead compared with the popular Level C data reuse method [6]. However, this architecture is very area-consuming, which takes 453k logic gates and 23.52kb of on-chip memory.

In [12], the authors propose a reconfigurable VLSI architecture to support VBSME. This architecture could support HDTV 720p at 45fps with a clock frequency of 180MHz. The main contribution of this paper is that it supports a meander-like scan format for a high data reuse of the search area. However, as a price this architecture needs 16 separate dual-port SRAMs, which causes high memory bitwidth and bandwidth and large die area.

The above four designs all employ the full-search block matching algorithm to find the optimal solutions of MVs at the expense of power consumptions. In [7], a fast algorithm based on 4-step search [13] to reduce the computational load of VBSME

[1] In [6], *Memory bitwidth* is defined as the number of bits which the hardware has to access from memory in each cycle during motion estimation, and *Memory bandwidth* as the number of bits which the hardware has to access from memory for motion estimation of a reference block. We use the same definitions in this paper.

and the corresponding hardware architecture are proposed for low power VBSME in H.264/AVC. The power consumption is only 16.72mW for real-time encoding of CIF videos at 30fps in high quality mode, rendering the proposed algorithm and design more suitable to mobile applications. However, unlike the previous VBSME designs, the optimal solutions of MVs are not guaranteed in [7]. In fact, many more fast algorithms can be used to reduce the computational load at the cost of non-optimal solutions of MVs [14].

The objective of this thesis is to employ a fast yet full-search block matching algorithm to reduce power consumption in VBSME while preserving the optimal solutions. The basic idea is to eliminate unnecessary SAD computation by using a conservative lower bound of SAD (defined as LSAD). As long as the computation load of LSAD is less than that of the skipped SAD, net power saving can be achieved. We are interested in the fast full-search block matching algorithms that have been previously used in the FBSME, which can guarantee the optimal MVs [15] [16] [17]. In this paper, these algorithms are extended for VBSME. To the best of our knowledge, this is the first time that a fast full-search block matching algorithm is explored to reduce power consumption in the context of VBSME, and designed in hardware. In most other implementations for VBSME, reducing power consumption is mostly attempted by using a fast but non-full-search block matching algorithm, which can not guarantee the optimal solutions of MVs [7].

For the proposed design for VBSME employing a fast full-search block matching algorithm, how much power can be saved depends on the LSAD and the detailed hardware architecture to implement it. We implemented the fast full-search algorithm for VBSME based on the traditional serial-input hardware architecture [18], to which significant revisions are made to accommodate VBSME. It is shown in this thesis that proposed approach can reduce power consumption by 45% without any quality loss.

1.3 Mixture of Gaussian

Although VBSME-based image segmentation algorithm gives more accurate results compared to FBSME and RBSME, one issue of these motion estimation based image segmentation algorithms is that they are not stable under camera jitter [2]. Thus, we turn

to alternative image segmentation algorithms to improve stability. Conventional image segmentation techniques including non-adaptive methods such as background subtraction and adaptive methods such as frame difference (FD). The non-adaptive methods have almost been abandoned because their need for manual initialization [19]. Without re-initialization, background noise accumulates over time. In another word, they are not suitable for highly automated surveillance environments or applications.

Besides FD, there are several other adaptive video segmentation methods, such as median filter (MF), linear predictive filter (LPF), MoG, and Kernel Density Estimation (KDE) [1]. A comparison among these methods has been made by Jiang et al. [1] in terms of performance, memory requirement, segmentation quality, and hardware complexity. We cite their results in Table 1.1. It can be seen that MoG has the best trade-off among these adaptive segmentation algorithms.

Table 1.1: Comparison of Difference Adaptive Video Segmentation Algorithms [1].

	FD	**MF**	**LPF**	**MoG**	**KDE**
Algorithm Performance	Fast	Fast	Medium	Medium	Slow
Memory Requirement	1 Frame	50 - 300 Frames	1 Frame	1 Frame of k Gaussian	n Frames of k Gaussian
Segmentation Quality	Worst	Low	Acceptable	Good	Best
Hardware Complexity	Very low	Medium	Low to medium	Low	High

Although sophisticated video segmentation algorithm development is a hot topic in the research community, only a few research groups work on the hardware implementation of the algorithms to meet real-time requirement of high frame rate high resolution video segmentation tasks. In our software implementation of an 3-mixture MoG algorithm on an Intel T4400 Dual-Core processor, the performance is about 5 second per frame with the VGA resolution (640×480), which can not even meet the real-time requirement of a 1fps video.

The latest work about hardware implementation of the MoG algorithm so fast as we know is from Jiang et al. [1]. In [1], the MoG algorithm is translated into hardware and implemented in Xilinx VirtexII pro Vp30 FPGA platform. It can support VGA resolution (640 × 480) at 25 fps in real-time. The authors also present a variety of memory access reduction schemes, which results in more than 70% memory bandwidth reduction. As a result, their design meets the real-time requirement of high frame rate high resolution segmentation task with a relatively low hardware complexity.

However, the design in [1] lacks flexibility. Some key parameters in the MoG algorithm, such as learning rate and threshold, have to be setup before generating the FPGA bit stream file. This means that every time if the users want to change the parameters, they have to re-program the FPGA, which is a time consuming process. Moreover, some commonly used components in embedded system, such as Universal Asynchronous Receiver/Transmitter (UART) and Inter-Integrated Circuit (I^2C), are not involved in their design, which also limits the application of their system.

As we know, video segmentation usually plays a preprocessing role for other applications, such as video surveillance, object detection, and object tracking. Thus, implementing the video segmentation algorithm into a hardware Intellectual Property (IP) will be more convenient than implementing it into an entire system such as [1]. The main reason is that a hardware IP can be easily integrated into an SoC architecture so long as it meets the specified bus standard. The SoC architecture has many advantages. First of all, it is easier to be updated for other applications. We can design other hardware IPs for applications based on the same bus standard and integrate them into the same system. Second, based on the SoC architecture, the parameter configuration task will be easier, which can be performed on-line via Micro Control Unit (MCU) in the SoC. Third, some commonly used components in embedded system, such as UART

and I^2C, can be easily integrated into the SoC. In a word, the SoC architecture makes the overall system more flexible.

Based on the above discussion, we implemented the MoG algorithm into a hardware IP, and integrated it into an SoC architecture. Similar to [1], we made some modifications to the original MoG algorithm. However, we did not modify the original algorithm in a same way as [1]. This is mainly because the Xilinx FPGA in our platform has more resources than the one in [1]. As a result, some algorithm modifications which will degrade segmentation performance are not used in our design. In another word, the modified MoG algorithm in our system are closer to the original MoG algorithm than the one in [1].

1.4 Summary

The rest of this thesis is organized as follows:

- Chapter 2 presents the VLSI implementation for RBSME. It includes the detailed architecture and experimental results.
- Chapter 3 presents the low-power VLSI implementation for VBSME. It includes the details of the fast full-search block matching algorithm and its power consumption saving estimation, the hardware implementation for VBSME based on the serial-input hardware architecture, and the experimental results.
- Chapter 4 presents the SoC architecture for video segmentation based on MoG algorithm. It includes the original MoG algorithm and a modified version for hardware implementation, the structure of the MoG hardware IP and the overall SoC architecture, and the experimental results.
- Chapter 5 presents a final discussion of the works presented in the thesis.

Chapter 2

Reconfigurable Block Size Motion Estimation

2.1 Hardware Architecture

The proposed conceptual VLSI architecture is shown in Fig. 2.1, which includes a *Reconfigurable Systolic Array*, a *Reconfigurable Search Area Control Unit*, a *Parallel Adder*, and a *Best Match Selection Unit (BMSU)*. This VLSI architecture is developed from the architecture in [20] originally designed for FBSME, which is chosen as the basis architecture for RBSME after a careful comparison of several possible hardware architectures [6] in terms of programmability, complexity and applicability. In Fig. 2.1, m, n refers to the block size in pixels, and p refers to the search range in pixels. R_{in}, S_{in} represents the input pixels of the reference block and the search area respectively.

The detailed structure of *Reconfigurable Systolic Array* is shown in Fig. 2.2. The *Two-Input Multiplexer* is controlled by the outputs of *Decoder M* or *Decoder D*, and the *Fifteen-Input Multiplexer* is controlled by the outputs of *Decoder N* so that the block size m and n can be arbitrarily adjusted in the range of $(2, 16)$ (we set a minimum block size of 2×2 in the design). The truth table of *Decoder M* is shown in Table 2.1 (sm has 14 bits due to the minimum block size constraints). The truth table of *Decoder N* is shown in Table 2.2. Note that different from *Decoder M*, it is a binary decoder. The truth table of *Decoder D* is the same as *Decoder M*, except the different inputs and outputs (see Table 2.1). The hardware structure of a *PE Unit* is shown in Fig. 2.3.

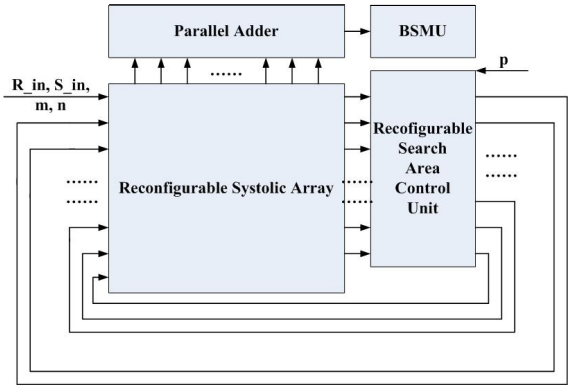

Figure 2.1: Conceptual hardware architecture.

RSR, *L1*, *L2* and *L3* are four registers. *ADC* is used to calculate absolute difference value.

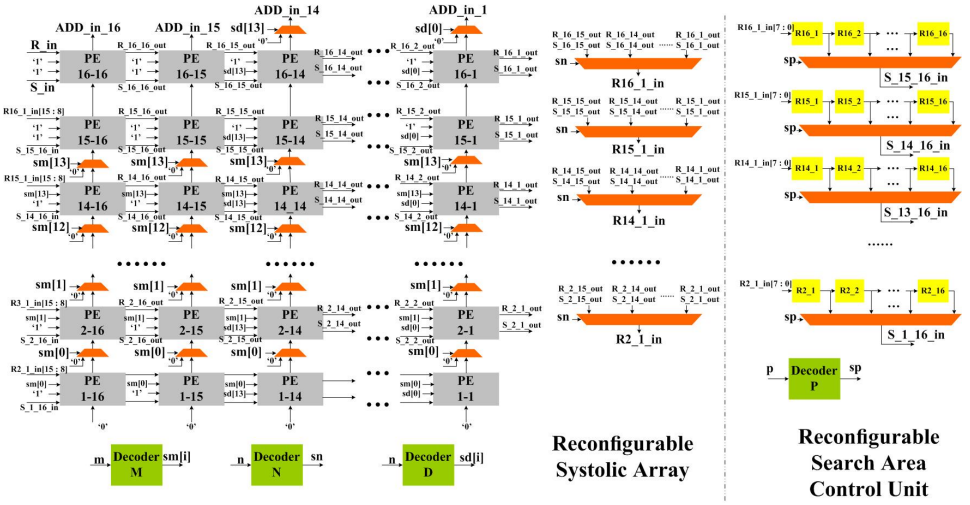

Figure 2.2: Block diagram of the hardware architecture for RBSME.

The detailed structure of *Reconfigurable Search Area Control Unit* is shown in Fig. 2.2 as well. The *Sixteen-Input Multiplexer* is controlled by the outputs of *Decoder P*, so that the search range parameter p can be arbitrarily adjusted in the range of $(1, 16)$. More details of the hardware architecture design can be found in [21].

Table 2.1: Truth Table of Decoder M (D)

m (n)	**sm[13 : 0] (sd[13 : 0])**	**m (n)**	**sm[13 : 0] (sd[13 : 0])**
16	11 1111 1111 1111	8	11 1111 0000 0000
15	11 1111 1111 1110	7	11 1110 0000 0000
14	11 1111 1111 1100	6	11 1100 0000 0000
13	11 1111 1111 1000	5	11 1000 0000 0000
12	11 1111 1111 0000	4	11 0000 0000 0000
11	11 1111 1110 0000	3	10 0000 0000 0000
10	11 1111 1100 0000	2	00 0000 0000 0000
9	11 1111 1000 0000	-	-

Table 2.2: Truth Table of Decoder N (P)

n (p)	**sn[3 : 0] (sp[3 : 0])**	**n (p)**	**sn[3 : 0] (sp[3 : 0])**
2(1)	0000	10(9)	1000
3(2)	0001	11(10)	1001
4(3)	0010	12(11)	1010
5(4)	0011	13(12)	1011
6(5)	0100	14(13)	1100
7(6)	0101	15(14)	1101
8(7)	0110	16(15)	1110
9(8)	0111	- (16)	- (1111)

2.2 Experiment

We implemented the proposed design in Xilinx Spartan-3A DSP XC3SD3400A FPGA. The design results show that the critical path delay of the proposed architecture is $6.7ns$. We also implement the original architecture for FBSME [20] with the same 16×16 PE array, which gives the same critical path delay. This is because all decoders and multiplexers in the design are pre-configured before motion estimation and there is no dynamic delay for the multiplexers and decoders. We evaluated the hardware resource of the original architecture for FBSME and the proposed architecture for RBSME, and

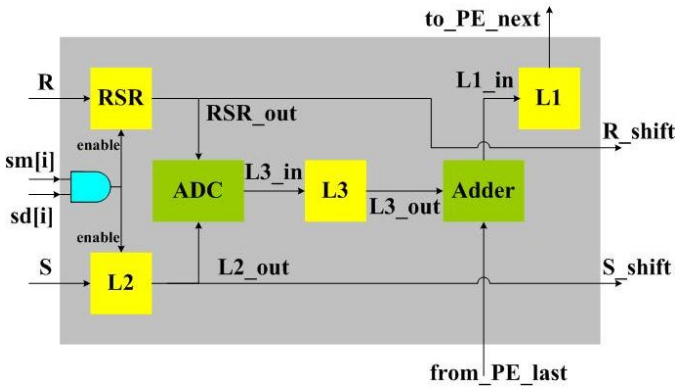

Figure 2.3: Block diagram of a PE Unit.

the results are shown in Table 2.3.

We also evaluated the two hardware architectures implemented in standard cell IBM 0.13μm CMOS technology and the results are shown in Table 2.4. The hardware overhead of the proposed architecture for RBSME is only 5% compared to the original one for FBSME with the same critical path delay.

Table 2.3: System resource of FBSME architecture and RBSME architecture in Xilinx FPGA

-	**FBSME**	**RBSME**	**Overhead**
Slices	11,942	13,790	15.5%
Slice Flip Flops	13,256	13,256	0
4-Input LUTs	18,268	22,017	20.5%

Table 2.4: Performance of FBSME architecture and RBSME architecture in IBM 0.13μm CMOS technology

-	**FBSME**	**RBSME**	**Overhead**
Critical Path (ns)	3.37	3.37	0
Area (μm^2)	771,991	813,632	5.4%

Chapter 3

Variable Block Size Motion Estimation

3.1 Fast Full-Search Block Matching Algorithm

Classic fast full-search block matching algorithms without quality loss have been presented in [15] [16] [17]. Their basic idea is to eliminate unnecessary computation of SAD by using the LSAD. Based on the algorithms presented in these papers, there are quite a few options to determine the LSAD. However, different LSADs have different disabling rate for SAD calculation, and the computation load of these LSADs are also different. If we define the power consumption of SAD calculation as P_{SAD}, the disabling rate of SAD calculation by using the fast full-search algorithm as $d\%$ and the power consumption of LSAD calculation as P_{LSAD}, then the overall power consumption P of a fast full-search algorithm can be expressed as follows.

$$P = P_{SAD} * (1 - d\%) + P_{LSAD} \tag{3.1}$$

And the power saving rate (PSR) is,

$$PSR = \frac{P_{SAD} - P}{P_{SAD}} = 1 - \frac{P}{P_{SAD}}$$

$$= d\% - \frac{P_{LSAD}}{P_{SAD}} \tag{3.2}$$

It can be seen from equation (3.2) that, regardless of the hardware architecture for fast full-search block matching, we can enhance *PSR* by either increasing d or decreasing P_{LSAD}. Both d and P_{LSAD} are dependent on the specific LSAD. However, a higher d usually leads to a higher P_{LSAD} and this tradeoff makes it difficult to determine the specific LSAD to maximize *PSR*. To find the best LSAD, we must use an experimental approach. In the following, we first discuss some options for LSAD, and then test their disabling rates for SAD calculation in FBSME based on the 4 × 4 block. Then we will extend the fast full-search algorithm to VBSME, and test their disabling rates in VBSME. Finally, we will estimate the power consumption of calculating these LSADs, and determine the LSAD that maximizes *PSR*.

3.1.1 Lower Bound of SAD and their disabling rates in FBSME

SAD is defined as follows [15] [16] [17]:

$$SAD(u,v) = \sum_{i=1}^{m}\sum_{j=1}^{n} |S(i+u, j+v) - R(i,j)|$$

$$-p \le u, v \le p \tag{3.3}$$

where R is reference block, *(i, j)* the coordinates of the pixels of the reference block, S target block, (u, v) the displacement, *m, n* the block size in pixels, and finally p the search range in pixels. As this paper focuses on lower power design for VBSME, and *4* × *4* is the basic block size in VBSME, the following discussion assumes $m = n = 4$.

The *triangular inequality* applied to equation (3.3) leads to:

$$SAD(u,v)_{m=n=4} \ge LSAD_a(u,v) =$$

$$|\sum_{i=1}^{4}\sum_{j=1}^{4} S(i+u, j+v) - \sum_{i=1}^{4}\sum_{j=1}^{4} R(i,j)| \tag{3.4}$$

It can be seen that $LSAD_a$ is a conservative lower bound of SAD. Let's define current minimum SAD as $SAD_{cur,min}$, we can eliminate SAD calculation for the candidate block at search positions *(u, v)* if $LSAD_a(u,v) \ge SAD_{cur,min}$.

Furthermore, based on the idea in [17], we can also define other LSADs as follows:

$$LSAD_b(u,v) = |\sum_{i=1}^{2}\sum_{j=1}^{4} S(i+u,j+v) - \sum_{i=1}^{2}\sum_{j=1}^{4} R(i,j)|$$
$$+|\sum_{i=3}^{4}\sum_{j=1}^{4} S(i+u,j+v) - \sum_{i=3}^{4}\sum_{j=1}^{4} R(i,j)| \quad (3.5)$$

$$LSAD_c(u,v) = |\sum_{i=1}^{2}\sum_{j=1}^{2} S(i+u,j+v) - \sum_{i=1}^{2}\sum_{j=1}^{2} R(i,j)|$$
$$+|\sum_{i=1}^{2}\sum_{j=3}^{4} S(i+u,j+v) - \sum_{i=1}^{2}\sum_{j=3}^{4} R(i,j)|$$
$$+|\sum_{i=3}^{4}\sum_{j=1}^{2} S(i+u,j+v) - \sum_{i=3}^{4}\sum_{j=1}^{2} R(i,j)|$$
$$+|\sum_{i=3}^{4}\sum_{j=3}^{4} S(i+u,j+v) - \sum_{i=3}^{4}\sum_{j=3}^{4} R(i,j)| \quad (3.6)$$

$$LSAD_d(u,v) = \sum_{i=1}^{4}|\sum_{j=1}^{4} S(i+u,j+v) - \sum_{j=1}^{4} R(i,j)| \quad (3.7)$$

By applying the *triangular inequality*, we can derive

$$LSAD_a(u,v) \le LSAD_b(u,v) \le LSAD_c(u,v) \le SAD(u,v) \quad (3.8)$$

and

$$LSAD_a(u,v) \le LSAD_b(u,v) \le LSAD_d(u,v) \le SAD(u,v) \quad (3.9)$$

The above two equations show that $LSAD_a(u,\ v)$, $LSAD_b(u,\ v)$, $LSAD_c(u,\ v)$, and $LSAD_d(u,\ v)$ are all conservative lower bounds of $SAD(u,\ v)$. Moreover, $LSAD_b(u,\ v)$ is tighter to $SAD(u,\ v)$ than $LSAD_a(u,\ v)$. $LSAD_c(u,\ v)$ and $LSAD_d(u,\ v)$ are tighter to $SAD(u,\ v)$ than $LSAD_b(u,\ v)$. However, it is hard to tell which is tighter to $SAD(u,\ v)$ between $LSAD_c(u,\ v)$ and $LSAD_d(u,\ v)$. We tested the disabling rates $d\%$ of these four LSADs based on ten CIF video sequences downloaded from [22]. The results are shown in Table 3.1. Finally, it should be noted that there could be many other definitions of LSADs and the reason that we only discuss these four is that they have regular expressions and are easier for hardware implementation.

Table 3.1: Disabling rates of the four LSADs in 4 × 4 FBSME (search area [-16, 16]).

Sequence (Frames)	$LSAD_a$	$LSAD_b$	$LSAD_c$	$LSAD_d$
Akiyo (300)	88.0%	91.0%	94.2%	92.6%
Claire (494)	88.7%	92.8%	95.6%	94.5%
Coastguard (300)	73.3%	85.9%	90.0%	93.3%
Container (300)	79.4%	86.2%	90.0%	91.8%
Foreman (300)	85.6%	91.1%	95.3%	93.5%
Hall Monitor (300)	86.6%	90.8%	94.8%	92.9%
Mobile (300)	74.9%	84.5%	91.3%	90.1%
Mother & Daughter (300)	84.5%	90.0%	94.2%	93.0%
Salesman (449)	84.3%	89.8%	94.5%	92.2%
Silent (300)	86.1%	91.1%	95.4%	92.9%
Average	83.1%	89.3%	93.5%	92.7%

3.1.2 Disabling rates in VBSME

To support VBSME in H.264/AVC, reusing the SADs of *4×4* blocks to derive the SADs of larger blocks is efficient [6] [7] [8] [9] [10] [12] [14]. In other words, full-search VBSME is based on *4×4* FBSME. Thus, how to eliminate search positions of a *4×4* sub-block is the key problem when we extend the fast full-search algorithm from FBSME to VBSME. We can not adopt the same criterion used in FBSME for VBSME, because the *4×4* sub-block is also a component of other larger blocks. For example, in Fig. 3.1, Block1 (*4×4*) is a component of Block17 (*4×8*), Block25 (*8×4*), Block33 (*8×8*), Block37 (*8×16*), Block39 (*16×8*), and Block41 (*16×16*).

If we eliminate the search position *(u, v)* of Block1 only when Block1's current minimum *SAD* (defined as $SAD_{cur,min,1}$) is less than its conservative lower bound at position *(u, v)* (defined as $LSAD_1(u, v)$), we might lose the optimal MV for Block17. This is because $SAD_{min,1} + SAD_{min,2} \leq SAD_{min,17}$, where $SAD_{min,X}$ represents the global minimum *SAD* for Block *X*. Although Block1's search position at *(u, v)* is not the optimal MV for Block1 when $SAD_{cur,min,1} < LSAD_1(u, v)$, it might be the optimal MV for Block17, Block25, Block33, Block37, Block39, and/or Block41 as all these blocks have Block1 as a component. Therefore, to eliminate the SAD calculation at search

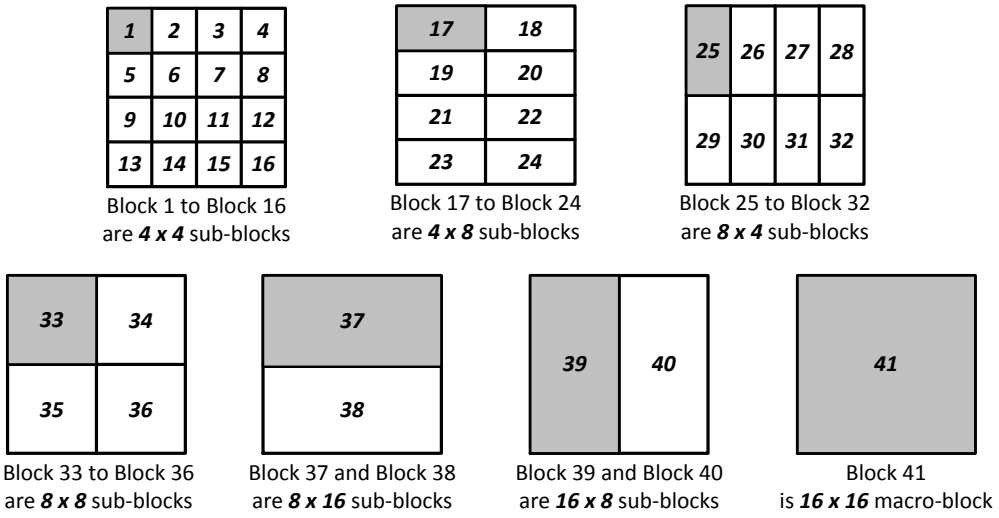

Figure 3.1: The partitions of 16 × 16 macro-block in VBSME.

position *(u, v)* of Block1 (defined as $SAD_1(u, v)$), we must make sure that we do not need to calculate SAD for any of the blocks of which the *4×4* sub-block is a component. This can be represented as the following [23]:

$$
\begin{aligned}
&if\ and\ only\ if \\
&\quad SAD_{cur,min,41} \le LSAD_{41}(u,v)\ \ and \\
&\quad SAD_{cur,min,39} \le LSAD_{39}(u,v)\ \ and \\
&\quad SAD_{cur,min,37} \le LSAD_{37}(u,v)\ \ and \\
&\quad SAD_{cur,min,33} \le LSAD_{33}(u,v)\ \ and \\
&\quad SAD_{cur,min,25} \le LSAD_{25}(u,v)\ \ and \\
&\quad SAD_{cur,min,17} \le LSAD_{17}(u,v)\ \ and \\
&\quad SAD_{cur,min,1} \le LSAD_{1}(u,v) \\
&Eliminate\ the\ calculation\ of\ SAD_1(u,v)
\end{aligned}
\tag{3.10}
$$

By using this approach, we can guarantee the optimal solutions of MVs for all the 41 blocks. The necessities of testing LSAD for multiple blocks in equation (3.10) for VBSME suggest that the closeness of LSAD approximation to SAD is very important as it may significantly affect the rate of disabling SAD computations for the 16 *4×4*

blocks. Fortunately, a very important point here is that $LSAD_{17}(u,v)$ to $LSAD_{41}(u,v)$ are derived by summing the lower bounds of their corresponding *4×4* sub-blocks. For example,

$$LSAD_{17}(u,v) = LSAD_1(u,v) + LSAD_2(u,v)$$

$$LSAD_{41}(u,v) = \sum_{i=1}^{16} LSAD_i(u,v)$$

This way of computing LSAD for larger blocks gives a great advantage compared to computing LSAD directly from those bigger blocks themselves, as the former way gives a tighter approximation to SAD. Note that this natural way of computing lower bounds for larger blocks in the context of VBSME can be also viewed as an application of the sub-region partitioning approach in [16] [17] for FBSME, by which a tighter bound to *SAD* can be obtained.

With the above described extensions of the fast full-search block matching algorithm from FBSME to VBSME, we finally obtain the disabling rates of the four LSADs defined in Section II.A and the results are shown in Table 3.2.

Table 3.2: Disabling rates of the four LSADs in VBSME (search area [-16, 16]).

Sequence (Frames)	$LSAD_a$	$LSAD_b$	$LSAD_c$	$LSAD_d$
Akiyo (300)	72.6%	76.9%	82.1%	79.3%
Claire (494)	74.6%	80.0%	84.7%	82.8%
Coastguard (300)	47.3%	61.1%	66.8%	74.4%
Container (300)	57.3%	63.6%	67.7%	70.8%
Foreman (300)	70.3%	77.6%	84.5%	81.2%
Hall Monitor (300)	71.1%	76.8%	84.2%	80.2%
Mobile (300)	49.8%	61.3%	71.6%	69.3%
Mother & Daughter (300)	65.7%	72.2%	79.0%	77.1%
Salesman (449)	66.0%	73.0%	81.2%	76.9%
Silent (300)	68.5%	75.1%	83.0%	78.2%
Average	64.3%	71.8%	78.5%	77.0%

3.1.3 Power Saving Rate Estimation

In equation (3.1), PSR depends on P_{SAD}, d, and P_{LSAD}. We have experimentally evaluated d in the previous sub-section and we evaluate P_{SAD} and P_{LSAD} in this part. We define the power consumption of calculating one addition in hardware as P_{Add}, and the power consumption of calculating one absolute difference is estimated to be $2P_{Add}$ [24]. We also define the power consumption of calculating $LSAD_a$ as P_{LSADa}, $LSAD_b$ as P_{LSADb}, $LSAD_c$ as P_{LSADc}, and $LSAD_d$ as P_{LSADd}. We first evaluate P_{SAD}. P_{SAD} for the optimal MV of a 4×4 sub-block is estimated as follows:

$$P_{SAD} = (2p+1)^2 \cdot (16 \cdot 2P_{Add} + 15 \cdot P_{Add})$$

$$= (188p^2 + 188p + 47) \cdot P_{Add} \tag{3.11}$$

where p refers to the search range in pixels. $(2p + 1)^2$ in equation (3.11) represents the total number of candidate blocks for one reference block. To calculate SAD of each candidate block, there needs 16 absolute difference calculations and 15 additions (see equation (3.3)).

P_{LSADa} is estimated as follows:

$$P_{LSADa} = (2p+1) \cdot (15 + 6 \cdot 2p) \cdot P_{Add}$$

$$+15 \cdot P_{Add} + (2p+1)^2 \cdot 2P_{Add}$$

$$= (32p^2 + 50p + 32) \cdot P_{Add} \tag{3.12}$$

The first term in equation (3.12), $(2p + 1) \cdot (15 + 6 \cdot 2p) \cdot P_{Add}$, corresponds to the power consumption of calculating the first term in equation (3.4) for all candidate blocks in the search area of one reference block. The search area contains $(2p + 1)$ rows. In each row, there are 15 additions for the first candidate block in the row and 6 additions for each of the other $2p$ candidate blocks (note that overlapped calculations between adjacent candidate blocks in the search window can be reused to save power). The second term in equation (3.12), $15 \cdot P_{Add}$, corresponds to the power consumption of calculating the second term in equation (3.4), which only needs to be calculated once for one reference

block. The last term in equation (3.12), $(2p + 1)^2 \cdot 2P_{Add}$, corresponds to the power consumption of $(2p + 1)^2$ absolute difference calculations required for all candidate blocks in the search area of one reference block. Similarly, we can estimate P_{LSADb}, P_{LSADc}, and P_{LSADd} as follows:

$$P_{LSADb} = (2p+1) \cdot (14 + 5 \cdot 2p) \cdot P_{Add}$$

$$+14 \cdot P_{Add} + (2p+1)^2 \cdot 5P_{Add}$$

$$= (40p^2 + 58p + 33) \cdot P_{Add} \tag{3.13}$$

$$P_{LSADc} = (2p+1) \cdot (12 + 6 \cdot 2p) \cdot P_{Add}$$

$$+12 \cdot P_{Add} + (2p+1)^2 \cdot 11P_{Add}$$

$$= (68p^2 + 80p + 35) \cdot P_{Add} \tag{3.14}$$

$$P_{LSADd} = (2p+1) \cdot (12 + 3 \cdot 2p) \cdot P_{Add}$$

$$+12 \cdot P_{Add} + (2p+1)^2 \cdot 11P_{Add}$$

$$= (56p^2 + 74p + 35) \cdot P_{Add} \tag{3.15}$$

Based on the above discussions and the derived disabling rates in Table 3.2, we can now use equation (3.1) to estimate the *PSR* of these four LSADs. The results are shown in Table 3.3. It can be seen that $LSAD_b$ has the highest average *PSR*. As a result, we use $LSAD_b$ as the lower bound in the fast full-search block matching algorithm for VBSME. The detailed hardware implementation of the fast algorithm is presented next.

3.2 Hardware Architecture

We implemented the fast full-search algorithm for VBSME based on the traditional serial-input hardware architecture [18] previously used in FBSME, on which we need to

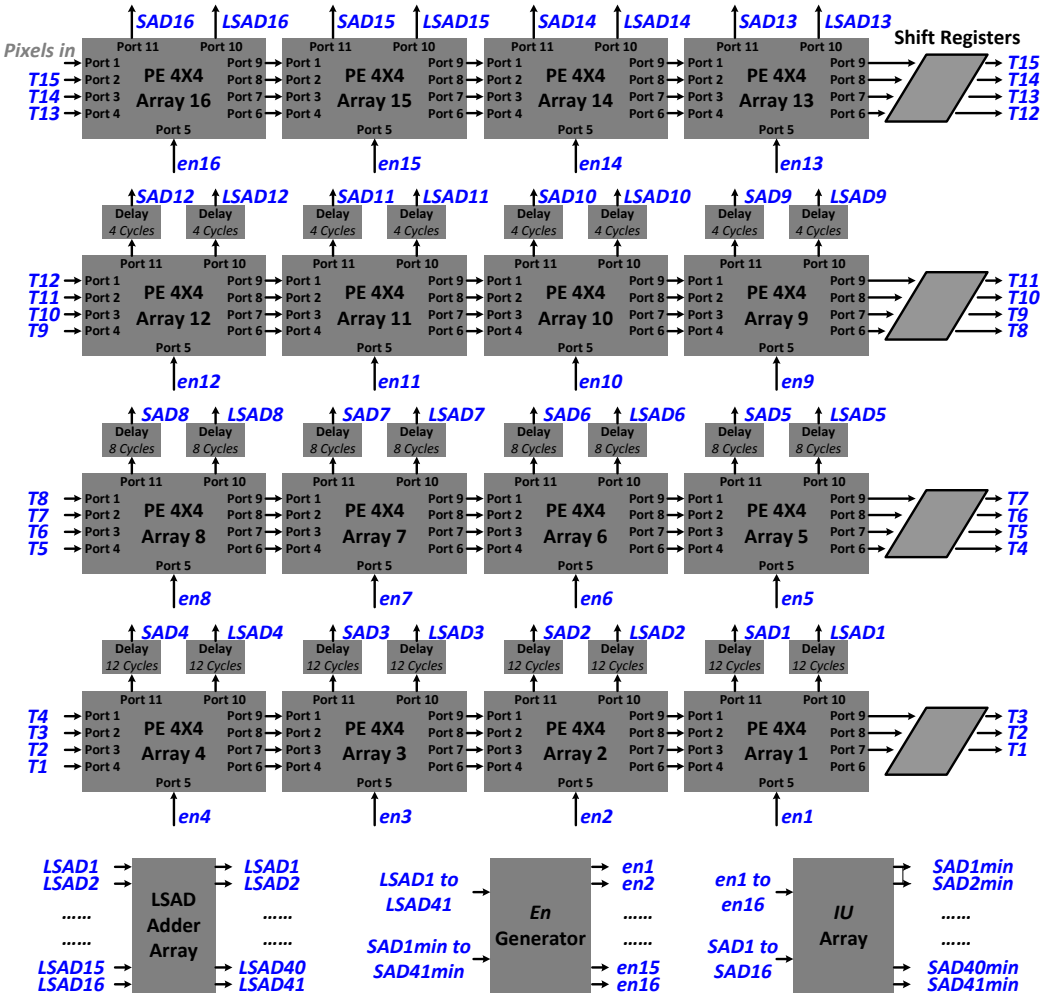

Figure 3.2: Proposed low-power architecture to calculate VBSME.

Table 3.3: *PSR* of the four LSADs in VBSME (search area [-16, 16]).

Sequence (Frames)	$LSAD_a$	$LSAD_b$	$LSAD_c$	$LSAD_d$
Akiyo (300)	55.0%	55.0%	45.5%	48.9%
Claire (494)	57.0%	58.1%	48.1%	52.4%
Coastguard (300)	29.7%	39.2%	30.2%	44.0%
Container (300)	39.7%	41.7%	31.1%	40.4%
Foreman (300)	52.7%	55.7%	47.9%	50.8%
Hall Monitor (300)	53.5%	54.9%	47.6%	49.8%
Mobile (300)	32.2%	39.4%	35.0%	38.9%
Mother & Daughter (300)	48.1%	50.3%	42.4%	46.7%
Salesman (449)	48.4%	51.1%	44.6%	46.5%
Silent (300)	50.9%	53.2%	46.4%	47.8%
Average	46.7%	49.9%	41.9%	46.6%

make significant revisions to accommodate VBSME. The advantage of the serial-input hardware architecture is that it saves power consumption of data access from VBSME core to the SRAM (which stores the search area pixels), though at the expense of slightly lower processing speed. The overall architecture for low-power VBSME design is shown in Fig. 3.2. It consists of 16 *PE 4 × 4 Arrays*, a *LSAD Adder Array*, an *En Generator*, an *IU array*, and *Shift Registers*. Each *PE 4 × 4 Array* is used to calculate SAD and LSAD of a *4 × 4* sub-block. *Shift Registers* in each row contain (*2p - 17*) registers, where *p* stands for search area in pixels. *LSAD Adder Array* is used to compose and synchronize 41 LSADs from the 16 basic 4 × 4 sub-blocks' LSADs. Since the architecture of this module is quite straight forward, we will not show the detail. *En Generator* is used to generate 16 enable signals to enable *SAD* calculation based on the principle shown in expression (3.10). *IU Array* is used to calculate 41 current minimum *SADs* of the 41 blocks defined in VBSME.

3.2.1 Processing Element Array

The architecture of a *PE 4 × 4 Array* is shown in Fig. 3.3. It mainly consists of 16 *PEs*, a *Sel Signal Generator*, an *enLSAD Signal Generator*, and summation circuits

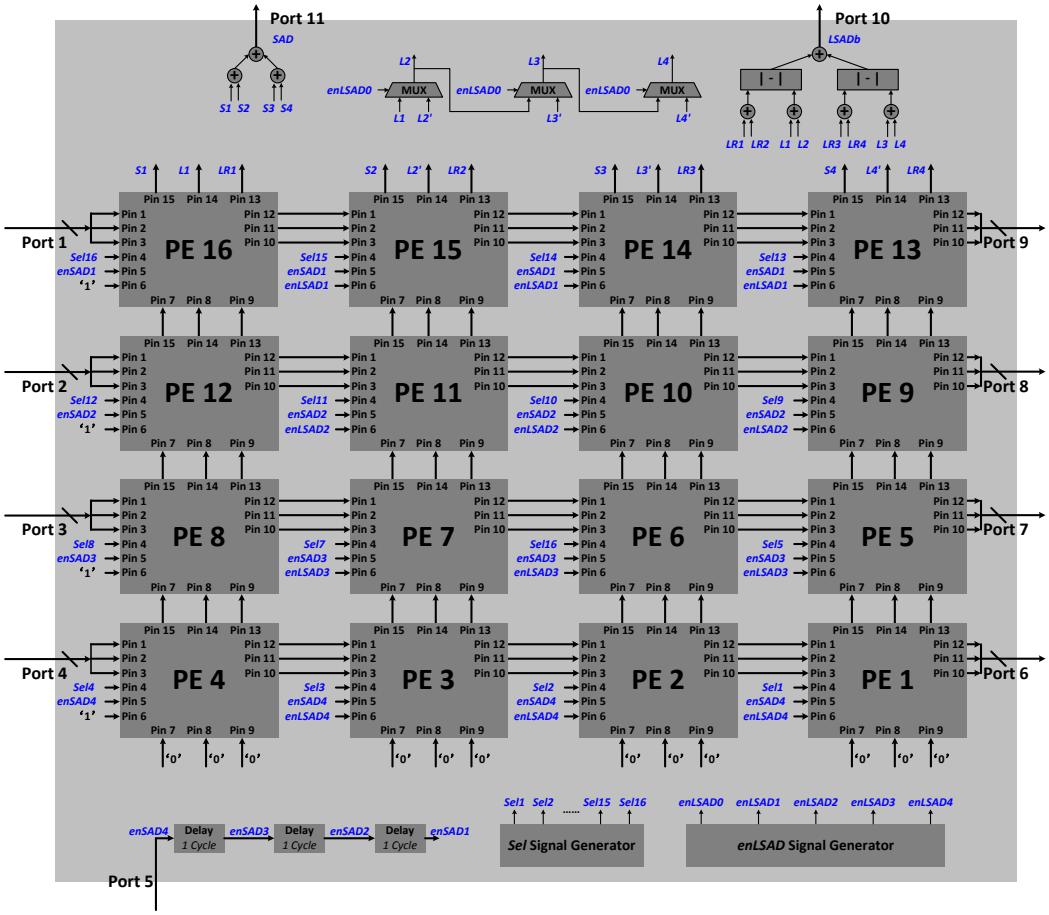

Figure 3.3: The architecture of PE 4 × 4 Array.

to calculate SAD and $LSAD_b$. *Sel Signal Generator* is designed to improve the PE utilization and the circuits can be found in [18].

3.2.2 Integration Unit Array

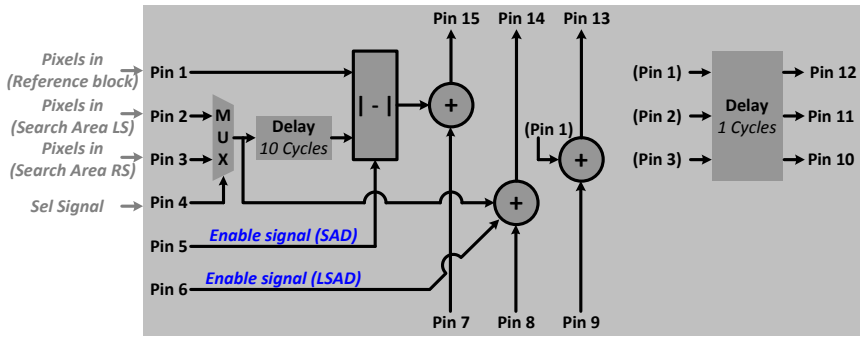

Figure 3.4: The architecture of PE.

The architecture of *PE* is shown in Fig. 3.4, which is not only based on the design in [20] and [18], but also combine the fast full-search block matching algorithm. The adder connected to *Pin 14* is used to calculate part of $LSAD_b$ for pixels in the search area. The adder connected to *Pin 13* is used to calculate part of $LSAD_b$ for pixels in the reference block. Note that two enable signals from *Pin 5* and *Pin 6* are used to eliminate unnecessary calculation of absolute difference and addition operations to save power. The 10-cycle delay after *MUX* will be explained in Part C.

The architectures of *IU Array* and *IU* are shown in Fig. 3.5. In Fig. 3.5 (a), 16 4×4 sub-blocks' current minimum SAD (*SAD1min* to *SAD16min*) are calculated directly from their current SAD (*SAD1* to *SAD16*) by the corresponding *IU*. The other 25 blocks' current minimum SAD (*SAD17min* to *SAD41min*) are calculated based on their corresponding sub-blocks' current SAD. The relationship between them has been shown in Fig. 3.1. The architecture of *IU* is shown in Fig. 3.5 (b). Its functions is shown as follows,

$$
\begin{aligned}
&if\ (Pin\ 1 = 1\ and\ Pin\ 2 = 1)\{\\
&\quad Pin\ 5 = 1;\\
&\quad Pin\ 6 = Pin\ 3 + Pin\ 4;\\
&\quad Pin\ 7(new) = min(Pin\ 6,\ Pin\ 7(old));\}\\
&else\{\\
&\quad Pin\ 5 = 0;\\
&\quad Pin\ 7(new) = Pin\ 7(old);\}
\end{aligned}
$$

3.2.3 State-Flow of the Processing Scheme

A state-flow of the proposed architecture is shown in Fig. 3.6. The initialization part takes 256 clock cycles. The 16 *PE 4×4 Array* has a 15-stage pipeline. The SAD and LSAD results are sent to *IU Array* and *LSAD Adder Array* respectively. Both of them have 8-stage pipeline. Then, *SADmin 1* to *41* generated by *IU Array* and *LSAD 1* to *41* generated by *LSAD Adder Array* will be sent to *En Generator*, which has 2-stage pipeline. *en 1* to *16* generated by *En Generator* will be sent back to both *PE 4×4 Array* and *IU Array*. Note that these *en* signals are delayed by 15 cycles before sending to *IU Array* to synchronize the results from 16 *PE 4×4 Array*. Moreover, for the same candidate block in a search window, we must get its LSAD several cycles before we decide whether or not to calculate its SAD. The number of cycles is determined by the sum of pipeline stage of *IU Array* (or *LSAD Adder Array*) and *En Generator*. That explains the 10-cycle delay in Fig. 3.4.

Finally, the total number of clock cycles to compute the optimal MV for one Macro-Block (MB) for the VBSME design based on the serial-input hardware architecture structure is:

$$256 + (2p+1)^2 + 15 + 8 + 2 = (2p+1)^2 + 281 \tag{3.16}$$

where p stands for the search area in pixels. If $p = 16$, the result is 1370.

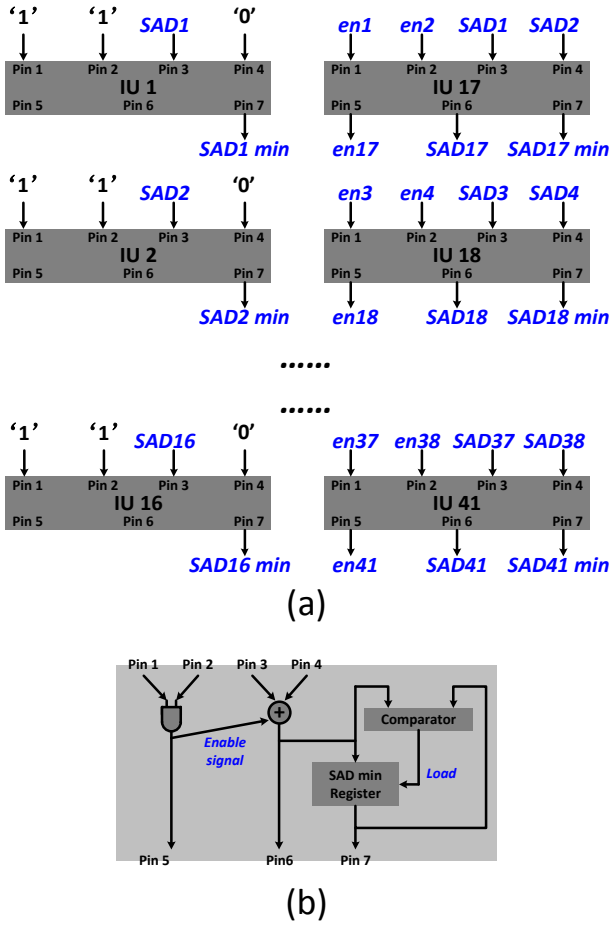

Figure 3.5: (a). IU Array. (b). Basic architecture of IU.

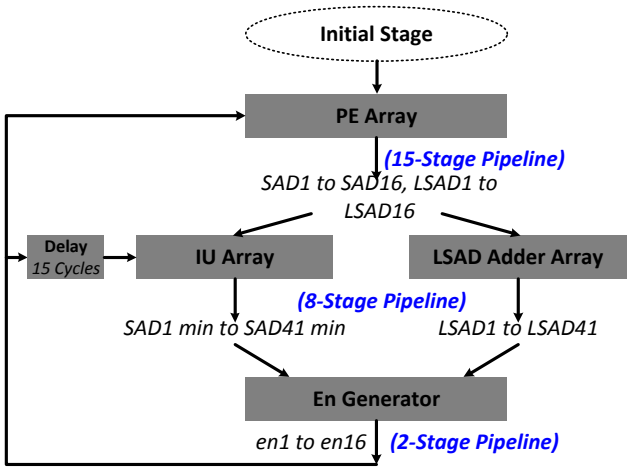

Figure 3.6: State-flow of the processing scheme.

3.3 Experiment and Performance Analysis

The proposed architecture has been implemented and verified in Xilinx Spartan-3A DSP FPGA Video Starter Kit [25]. Design report from Xilinx ISE shows that the critical path delay of the proposed architecture is 5.373ns. The overall design takes 17k Slices, 15k Slice Flip Flops, 23k 4-Input LUTs and 20k bits SRAMs. The FPGA prototype has been tested with ten video sequences presented in Section II. To measure how much power consumption can be saved, we also implement the design without full-search fast VBSME algorithm. We measure the power consumptions of these two designs by Xilinx XPower, the results is shown in Table 3.4. The results show that the proposed design can save power consumption by almost 45% compared to the conventional design while preserving the optimal solutions of motion vectors. It can be seen that power saving rate in Table 3.4 is a little lower than the one shown in Table 3.3. This is because some hardware components, such as *Shift Registers*, *IU array*, *En Generator* and *LSAD Adder Array*, also consume power. Their power consumptions are ignored by equation (3.2), but evaluated by Xilinx XPower.

Finally, we made a comparison of the proposed implementation to the other reported hardware architecture designs for VBSME. The results are shown in Table 3.5. The one in [12] is a high performance VLSI implementation for full-search VBSME without fast algorithm. It is a good example for those designs which focus on the optimal

Table 3.4: Power consumption of proposed and conventional architecture for VBSME (search area [-16, 16], FPGA working at 25MHz)

Sequence (Frames)	Proposed (mW)	Conventional (mW)	Power Saving Rate
Akiyo (300)	16.36	33.07	50.5%
Claire (494)	15.44	33.31	53.7%
Coastguard (300)	21.53	33.07	34.9%
Container (300)	20.87	33.02	36.8%
Foreman (300)	16.20	33.02	51.0%
Hall Monitor (300)	16.66	33.32	50.0%
Mobile (300)	21.51	33.29	35.4%
Mother & Daughter (300)	18.16	33.21	45.3%
Salesman (449)	17.77	33.26	46.6%
Silent (300)	17.13	33.19	48.4%
Average	18.16	33.18	45.2%

solution and high performance without quality loss. The other one in [7] is an extremely low-power VLSI implementation for fast VBSME, however, it can not guarantee the optimal solution. It is a good example for those designs using fast algorithm to focus on low-power VBSME implementation with some quality loss. The VLSI implementation presented in this paper can not only reduce power consumption, but also preserve the optimal solution. To the best of our knowledge, this is the first time that a fast full-search block matching algorithm is explored to reduce power consumption in the context of VBSME. Compared to [12], the proposed design saves power consumption by 41%. Although the one in [7] can save power consumption by 77% compared to our design, it can not guarantee the optimal solutions. It should be noted that our current design is implemented on FPGA and the power saving would be improved if a similar CMOS technology was used [23].

One main drawback of our design seems to be the large demand for hardware resources compared to other VBSME designs, which is 12.5% larger than [12], and 37.2% larger than [7]. Moreover, in terms of throughput, it can be seen that the operation cycles of our design are larger than the other two designs.

Table 3.5: Performance comparison of different hardware implementations for VBSME based on CIF videos (block size 16 × 16 and search range [-16, 16]).

Architecture	**[7]**	**[12]**	**This Paper**
Power (mW)	4.2	31.0	18.2
Optimal MV	No	Yes	Yes
Gate Count (K)	131.2	160	180
Operation Cycles (Per MB, *p = 16*)	568	1129	1370
Highest Resolution Supported	576i,30fps @55MHz	720p,45fps @180MHz	720p,45fps @180MHz

Chapter 4

Mixture of Gaussian

4.1 Mixture of Gaussian Algorithm for Video Segmentation

The MoG algorithm for video segmentation is first proposed by Stauffer and Grimson [19]. This algorithm considers the values of a pixel at a particular position *(x, y)* of an image over time t as a "pixel process", and the recent history of the pixel is modeled by a mixture of K Gaussian distributions. The probability of observing the pixel is

$$P(X_t) = \sum_{i=1}^{K} \omega_{i,t} \cdot \eta(X_t, \mu_{i,t}, \Sigma_{i,t}) \tag{4.1}$$

where X_t stands for the incoming pixel at time t (or frame t), K is the number of Gaussian distribution, and η is a Gaussian probability density function. Moreover, $\omega_{i,t}$ is the weighting factor, $\mu_{i,t}$ is the mean value and $\Sigma_{i,t}$ is the covariance matrix of the i_{th} Gaussian at time t.

A match is defined as the incoming pixel within J (usually set to 2.5) times the standard deviation off the center. If a match is found, ω_i, μ_i, and δ_i ($\Sigma_{i,t} = \delta_i^2 \cdot I$) is updated as follows,

$$\omega_{i,t+1} = (1 - \alpha) \cdot \omega_{i,t} + \alpha \tag{4.2}$$

$$\mu_{i,t+1} = (1 - \beta) \cdot \mu_{i,t} + \beta \cdot X_t \tag{4.3}$$

$$\delta^2_{i,t+1} = (1-\beta)\cdot\delta^2_{i,t} + \beta\cdot(X_t-\mu_{i,t})^T\cdot(X_t-\mu_{i,t}) \tag{4.4}$$

else,

$$\omega_{i,t+1} = (1-\alpha)\cdot\omega_{i,t} \tag{4.5}$$

$$\mu_{i,t+1} = \mu_{i,t} \tag{4.6}$$

$$\delta^2_{i,t+1} = \delta^2_{i,t} \tag{4.7}$$

The portion of the Gaussian distributions belonging to the background is defined to be

$$B = argmin_b(\sum_{i=1}^{b}\omega_i > T) \tag{4.8}$$

where T, the threshold, is a measure of the minimum portion of the data that could be accounted for by the background.

To translate the MoG algorithm into hardware, it is necessary to do some modification based on the original algorithm. The modified algorithm should have almost the same segmentation performance as the original one, but be more suitable for hardware implementation. A pseudocode of the modified algorithm is show as follows,

========== Step 1.==========
for *i = 1 to K*
    t(i) = $(\mu_i - X_t)^2 - J^2\cdot\delta_i^2$;

========== Step 2.==========
Find the minimum t(i), define r = arg min[t(i)];
if *t(r) > 0*
    then *match = 0 (no match);*
    else *match = 1 (match);*

========== Step 3.==========

if *(not match)*

then { $\mu_1 = X_t$; $\delta_1^2 = 25$; }

else {

$\mu_r = \mu_r + \beta \cdot (\mu_r - X_t)$;

$\delta_r^2 = \delta_r^2 + \beta \cdot [(\mu_r - X_t)^2 - \delta_r^2]$;

$\omega_r = \omega_r + \alpha \cdot (1 - \omega_r)$;

for *i = 1 to K*

if *(i != r)*

then $\omega_i = \omega_i - \alpha \cdot \omega_r$; }

========== Step 4.==========

if *(not match)*

then *Current pixel is foreground;*

else if *[r < (K - B)]*

then *Current pixel is foreground;*

else *Current pixel is background;*

========== Step 5.==========

Sort ω_i *in ascending order;*

B = 0; Sum_ω *= 0; i = K - 1;*

While *(*Sum_ω *< T)* {

Sum_ω *+=* ω_{i--};

B++; }

4.2 Hardware Implementation of the MoG Algorithm

The proposed hardware implementation of the modified MoG algorithm with three mixtures is shown in Fig. 4.1. The registers J^2, α, β, and T are used to control global algorithm parameters, and the blocks labeled *Step1* to *Step5* in Fig. 4.1 are the hardware implementations of the modified MoG algorithm presented in Section II. Their

detailed circuits can be found from Fig. 4.2 to Fig. 4.6 respectively. The corresponding pseudocode presented in Section II explains how these circuits work.

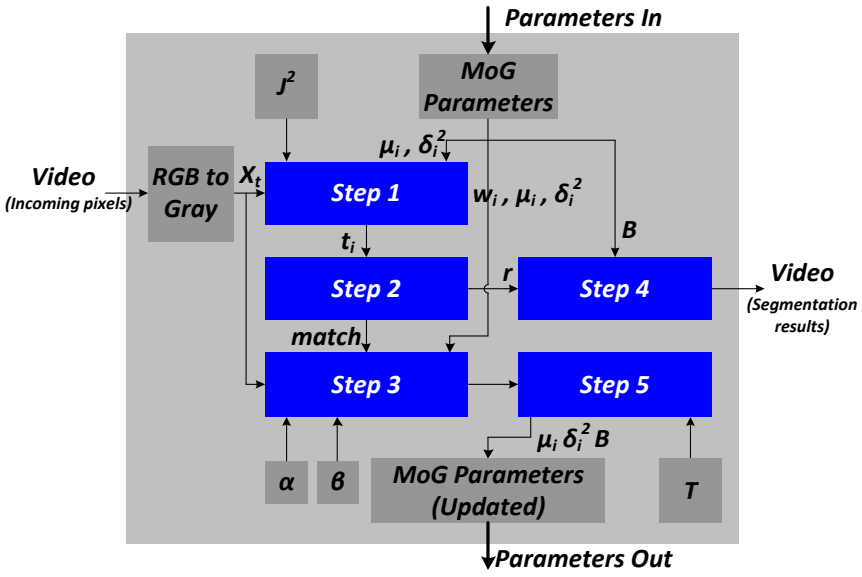

Figure 4.1: Overall architecture of the MoG IP.

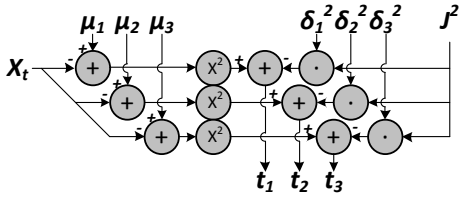

Figure 4.2: Circuits in step1 (3-stage pipeline).

4.3 The SoC Architecture

To integrate the hardware implementation of the MoG algorithm into an SoC architecture, we need to use standard interfaces of the SoC system. Since our platform is based on Xilinx Spartan3A-DSP FPGA, we need to use three groups of interfaces. The first one is the video bus interface [25], which includes *incoming pixels* and *segmentation results*. The second one is the Processor Local Bus (PLB) interface [26], which connects

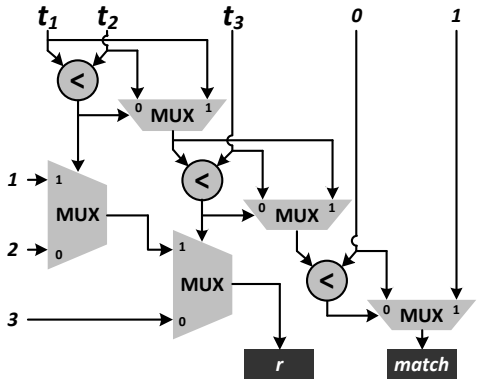

Figure 4.3: Circuits in step2 (6-stage pipeline).

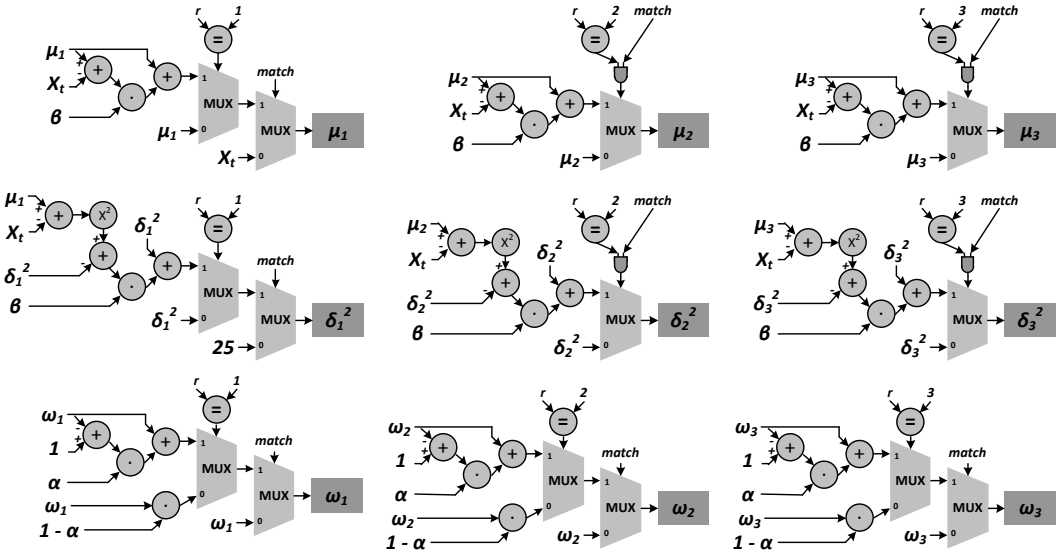

Figure 4.4: Circuits in step3 (5-stage pipeline).

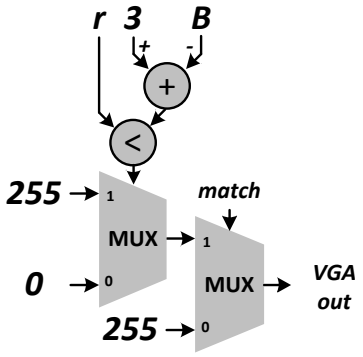

Figure 4.5: Circuits in step4 (4-stage pipeline).

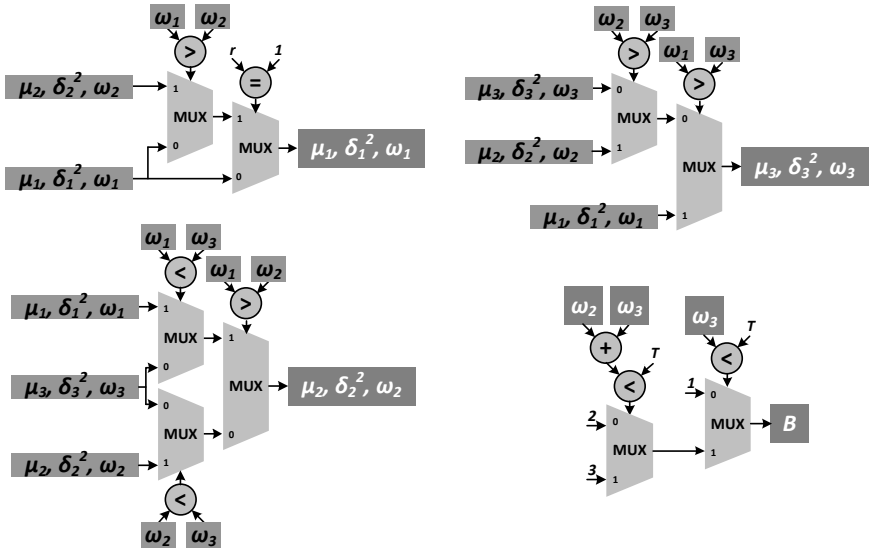

Figure 4.6: Circuits in step5 (3-stage pipeline).

the registers J^2, α, β, and T to the Xilinx MCU *MicroBlaze* [27]. The third one is the Video Frame Buffer Controller (VFBC) bus interface [28], which connects the MoG IP to Xilinx multi-port memory controller (MPMC) [28] to read the model parameters (*Parameters In* in Fig. 4.1) and update the model parameters (*Parameters Out* in Fig. 4.1). The overall SoC architecture is shown in Fig. 4.7. All the IPs except our *MoG IP* are Xilinx's products, and are available from Xilinx. For more detailed information about these IPs, please visit Xilinx's website [29].

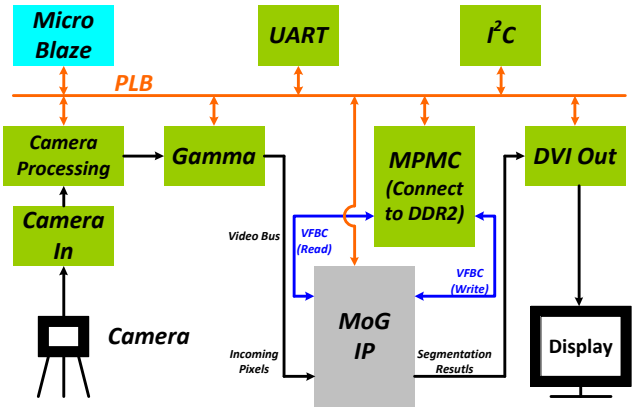

Figure 4.7: SoC architecture of the proposed design.

We use 8 bits for each register in the *MoG IP*. The bit width of the *VFBC* bus is set to 64, which is the maximum bit width of the bus. The mean parameter needs a minimum of 8 bits to represent its value (8-bit integer). The variance parameter also needs a minimum of 8 bits to represent its value (8-bit integer). The weight parameter needs a minimum of 4 bits to represent its value (4-bit fractional). As a result, we can only support at most 3 mixtures for the Gaussian model based on this hardware platform. These 64 bits are assigned for each pixel model as follows. 8 bits of them are for the mean parameter μ of each mixture in the model, and 8 bits are for the variance parameter δ^2 of each mixture in the model. All of these bits are used for integer part. No fraction part is used for the mean parameter or the variance parameter. 4 bits are for the weight parameter ω of each mixture in the model. All of these bits are used for fraction part. No integer part is used for the weight parameter. As a result, for the three-mixture Gaussian model, $3 \times (8 + 8 + 4) = 60$ bits are used to restore the mean,

variance, and weight parameters. The last 4 bits of the *VFBC* bus are used for the parameter B. For a typical 640×480 image, to store all the related parameters for the three-mixtures Gaussian model, we need $640 \times 480 \times 64 \approx 2.35$ Mega Bytes memory.

The proposed SoC design works under three clock domains. The components connected to *PLB* work under the 62.5MHz clock domain. The components connected to *Video Bus* and *VFBC* work under the 25MHz clock domain. The *DDR2* SRAM (not shown in Fig. 4.7) connected to *MPMC* work under the 125MHz clock domain. Some components connected both to *PLB* and *Video Bus*, such as *Camera Processing* and the *MoG IP*, have two clock domains. *MPMC* is the only component which works under all of the three clock domains.

The main advantage of this SoC architecture is that most of the components are connected to *MicroBlaze* via *PLB*. These components can communicate with each other via *MicroBlaze*, which makes the system very flexible. For example, if we want to control or reset the learning rate α of the *MoG IP* via *UART*, we can send commands to *UART* first. Then *UART* will send these commands to *MicroBlaze* via *PLB*. Once *MicroBlaze* get these commands, it will configure the value of register α to the desired one via *PLB*. This process is independent from the video processing, which means that the configuration process can be performed on-line. In the *MoG IP*, the configurable parameters are J^2, learning rate α and β, and the threshold T. Users can configure these parameters on-line to their most idea values by an experiment approach.

4.4 Experiment and Performance Analysis

The proposed design has been implemented and verified in Xilinx Spartan-3A DSP FPGA Video Starter Kit shown in Fig. 4.8 [25]. Design report from Xilinx ISE shows that the critical path delay of the proposed design is 7.766ns. The overall design takes 14k Slices, 12k Slice Flip Flops, 15k 4-Input LUTs, 77 BRAMs, 11 DSP48As, and 2 DCMs. The FPGA design summary is shown in Table 4.1. A hardware complexity of different blocks in the system is shown in Table 4.2. The system's parameters are shown in Table 4.3. This system can process one pixel per clock cycle due to the pipeline structure. If the video's resolution is 640×480, it can support up to $\frac{25 \times 10^6}{640 \times 480} = 81 fps$. The throughput in Table 4.3 is computed by multiplying the *Video Bus clock frequency*

and the bit width of the memory bus. In our design, we use 64 bits to restore a 3 mixtures of Gaussian model, and 8 bits for the input image. Thus the bit width of the memory bus is totally 72 bits, and the throughput is $\frac{25\times10^6\times72}{1024^2\times8} \approx 214$ Mega Bytes.

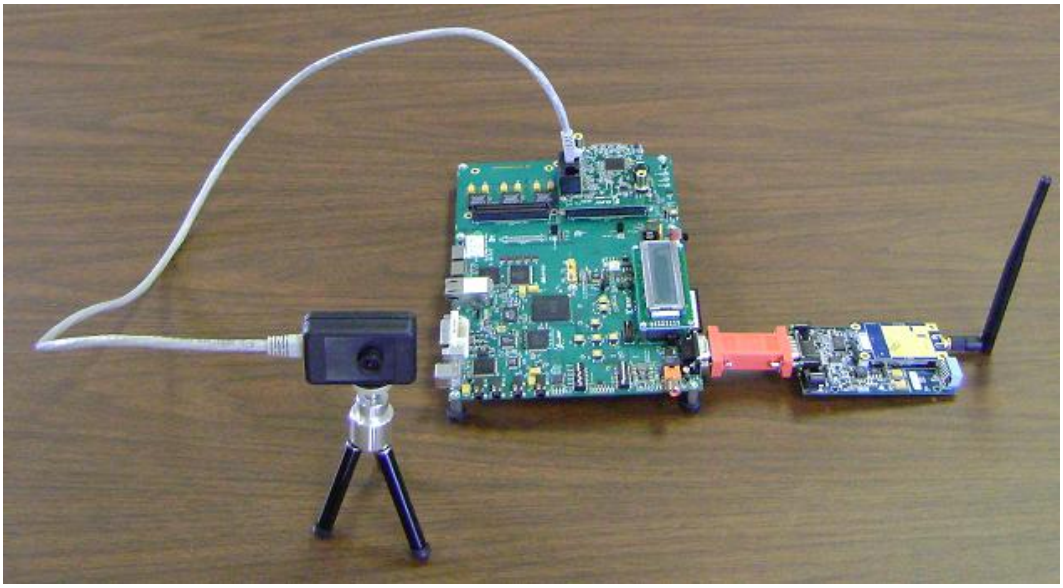

Figure 4.8: Hardware system.

An sample output of the proposed video segmentation system is shown in Fig. 4.9.

Figure 4.9: Segmentation result ($\alpha = \beta = 0.0625$).

Finally, we made a comparison of the proposed design with the one proposed by Jiang et al [1]. The results are shown in Table 4.4. Note that the theoretically maximum

Table 4.1: Design Summary.

Logic	No. of Used	Utilization
Slice Flip Flops	12,410	26%
4-input LUTs	15,921	33%
Occupied Slices	14,265	60%
DCMs	2	25%
BRAMs	77	61%
DSP48As	11	8%

Table 4.2: Hardware Complexity of Different Blocks.

Logic Block	Flip Flops	4-Input LUTs	BRAMS
Camera In	10	0	0
Camera Processing	992	1073	3
Gamma	95	62	3
MPMC	7040	7933	43
DVI out	29	1	0
UART	146	134	0
I^2C	382	494	0
MicroBlaze	1676	2639	4
MOG IP	1278	2481	0

Table 4.3: System Parameters.

Resolution	640 × 480
Throughput	214 MB/s
Frame rate	30 fps
No. of Gaussian	3
PLB clock frequency	62.5 MHz
Video Bus clock frequency	25 MHz
DDR2 clock frequency	125 MHz

frame rate of our design is 81fps for the image resolution 640×480, which is much higher than the one proposed by Jiang et al [1]. This is mainly because our design uses totally 21-stage pipeline, which has much shorter critique path than the one proposed by Jiang et al [1]. Basically, the two systems have almost the same performance. Our system consumes more hardware resources than [1]. However, our system is more flexible than [1] at the expense of more hardware resources. For further development work such as video surveillance, object detection, and object tracking, our system is a better platform due to the SoC architecture.

Table 4.4: System Comparison with Jiang et.al [1].

	Jiang et.al [1]	**The proposed design**
No. of Slices	6,107	14,265
No. of Flip Flops	4,273	12,410
No. of DCMs	5	2
No. of BRAMs	84	77
Resolution	640 × 480	640 × 480
Frame rate	25 fps	30 fps
Throughput	170MB/s	214MB/s
No. of Gaussian	3	3

Chapter 5

Conclusion and Discussion

In this thesis, we first present a reconfigurable VLSI architecture for RBSME. It is developed from the traditional hardware architecture for FBSME and achieves the flexibility of adjustable block size at the expense of only 5% hardware overhead. The main benefit of the proposed architecture is the support for RBSME, which makes it flexible to accommodate different applications. Even for a specific application, such as object tracking in video surveillance, the proposed hardware architecture allows experiments of the block size on the fly for optimized performance.

Then we propose a new VLSI implementation of VBSME in H.264/AVC. The main contribution of the proposed design is the VLSI implementation of a fast full-search block matching algorithm to save power consumption while preserving optimal solutions of MV, which has not been explored in the context of VBSME in literature. We first determined the specific LSAD to be used in the fast full-search algorithm and then propose the implementation of the algorithm based on the serial-input hardware architecture. The power consumption of our design is very competitive compared to other VBSME designs, especially those giving optimal solutions of MV. For high-end applications, the proposed design can support HDTV 720p (1280×720) at 45fps in real-time under 180MHz clock frequency and the power consumption is 131 mW.

In the last part of this thesis, we present a hardware implementation of the MoG algorithm for video segmentation application and the corresponding SoC architecture. The main contribution of the proposed design is that the SoC architecture makes the system more flexible to adapt to different environments and easier to develop other

video processing IPs for the applications such as video surveillance, object detection, and object tracking. The proposed design can support VGA resolution (640×480) at 30fps in real-time under 25MHz clock frequency. Future work should focus on hardware platform custom design for the real-time image segmentation application, and focus on hardware implementations of more sophisticated algorithms which have better segmentation performance, such as KDE.

References

[1] H. Jiang, H. Ardo, and V. Owall, "A hardware architecture for real-time video segmentation utilizing memory reduction techniques," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 19, no. 2, pp. 226–236, 2009.

[2] Y. Zheng, "Design of a hardware based vehicle tracking system," *Master's Thesis, Department of Electrical and Computer Engineering, University of Minnesota Duluth*, 2008.

[3] "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification," 2003.

[4] E. Qaralleh and T. S. Chang, "Fast variable block size motion estimation by adaptive early termination," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 16, no. 8, pp. 1021–1026, 2006.

[5] S.-C. Hsia and P.-Y. Hong, "Fast variable block motion estimation based on adaptive search for H.264/AVC system," in *Proc. of 8th IEEE International Conference on Computer and Information Technology*, pp. 41–46, 2008.

[6] C. Chen, S. Chien, Y. Huang, T. Chen, T. Wang, and L.G.Chen, "Analysis and architecture design of variable block-size motion estimation for H.264/AVC," *IEEE Trans. on Circuits and Systems I*, vol. 53, no. 2, pp. 578–593, 2006.

[7] T. Chen, Y. Chen, S. Tsai, S. Chien, and L. Chen, "Fast algorithm and architecture design of low-power integer motion estimation for H.264/AVC," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 17, no. 5, pp. 568–577, 2007.

[8] S. Yap and J. McCanny, "A VLSI architecture for variable block size video motion estimation," *IEEE Trans. on Circuits and Systems II*, vol. 51, no. 7, pp. 384–389, 2004.

[9] C.-M. Ou, C.-F. Le, and W.-J. Hwang, "An efficient VLSI architecture for H.264 variable block size motion estimation," *IEEE Trans. Consumer Electronics*, vol. 51, no. 4, pp. 1291–1299, 2005.

[10] L. Deng, W. Gao, M. Z. Hu, and Z. Z. Ji, "An efficient hardware implementation for motion estimation of AVC standard," *IEEE Trans. Consumer Electronics*, vol. 51, no. 4, pp. 1360–1366, 2005.

[11] C. Kao and Y. Lin, "A high-performance and memory-efficient architecture for H.264/AVC motion estimation," in *IEEE Conf. ICME*, 2008.

[12] W. Cao, H. Hou, J. Tong, J. Lai, and H. Min, "A high-performance reconfigurable VLSI architecture for VBSME in H.264," *IEEE Trans. Consumer Electronics*, vol. 54, no. 3, pp. 1338–1345, 2008.

[13] L. Po and W. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 313–317, 1996.

[14] G. Callic, S. Lpez, O. Sosa, J. Lopez, and R. Sarmiento, "Analysis of fast block matching motion estimation algorithms for video super-resolution systems," *IEEE Trans. Consumer Electronics*, vol. 54, no. 3, pp. 1430–1438, 2008.

[15] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Trans. on Image Processing*, vol. 4, no. 1, pp. 105–107, 1995.

[16] M. Brunig and W. Niehsen, "Fast full-search block matching," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 11, no. 2, pp. 241–247, 2001.

[17] S. Mattoccia, F. Tombari, L. Stefano, and M. Pignoloni, "Efficient and optimal block matching for motion estimation," in *IEEE Conf. ICIAP*, 2007.

[18] Y. Yeh and C. Lee, "Cost-effective VLSI architectures and buffer size optimization for full-search block matching algorithms," *IEEE Trans. on VLSI Systems*, vol. 7, no. 3, pp. 345–358, 1999.

[19] C. Stauffer and W. Grimson, "Adaptive background mixture models for real-time tracking," in *Proc. IEEE Conf. Compu. Vis. Pattern Recognit.*, pp. 246–252, 1999.

[20] C. Hsieh and T. Lin, "VLSI architecture for block-matching motion estimation algorithm," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 2, no. 2, pp. 169–175, 1992.

[21] P. Li and H. Tang, "Vlsi architecture design for reconfigurable block size motion estimation," in *Proc. of IEEE International Conference on Consumer Electronics*, pp. 439–440, 2010.

[22] *http://trace.eas.asu.edu/yuv/index.html.*

[23] P. Li and H. Tang, "A low-power VLSI implementation for full-search variable block size motion estimation in H.264/AVC," in *Proc. of IEEE International Symposium on Circuits and Systems*, pp. 2972–2975, 2010.

[24] V. Do and K. Yun, "A low-power VLSI architecture for full-search block-matching motion estimation," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 8, no. 4, pp. 393–398, 1998.

[25] "Spartan-3A DSP FPGA video starter kit user guide," *Xilinx Inc*, 2008.

[26] "Processor Local Bus (PLB) v4.6 (v1.04a) - Product Specification," *Xilinx Inc*, 2009.

[27] "Microblaze processor reference guide," *Xilinx Inc*, 2009.

[28] "Multi-port memory controller (MPMC) (v5.04.a) - Product Specification," *Xilinx Inc*, 2009.

[29] *http://www.xilinx.com.*

Appendix A

Acronyms

Care has been taken in this thesis to minimize the use of acronyms, but this cannot always be achieved. This appendix contains a table of acronyms and their meaning.

A.1 Acronyms

Table A.1: Acronyms

Acronym	Meaning
1-D	One-Dimensional
AVC	Advanced Video Coding
BMSU	Best Match Selection Unit
CMOS	Complementary Metal Oxide Semiconductor
CIF	Common Intermediate Format
DSP	Digital Signal Processor
FBSME	Fixed Block Size Motion Estimation
FD	Frame Different
FPGA	Field-Programmable Gate Array
fps	frame-per-second
HDTV	High-Definition TeleVision

Continued on next page

Table A.1 – continued from previous page

Acronym	Meaning
I^2C	Inter-Integrated Circuit
ITS	Intelligent Transportation System
IU	Integration Unit
kb	kilo bits
KDE	Kernel Density Estimation
LPF	Low-Pass Filter
LSAD	Lower bound of Sum of Absolute Difference
MCU	Micro-Control Unit
MF	Mean Filter
MHz	Mega-Hertz
MoG	Mixture of Gaussian
MV	Motion Vector
PE	Processing Element
PLB	Processor Local Bus
PSR	Power Saving Ratio
RBSME	Reconfigurable Block Size Motion Estimation
SAD	Sum of Absolute Difference
SoC	System-on-a-Chip
SRAM	Static Random-Access Memory
UART	Universal Asynchronous Receiver/Transmitter
VBSME	Variable Block Size Motion Estimation
VFBC	Video Frame Buffer Controller
VGA	Video Graphics Array
VLSI	Very Large Scale Integrated