ENHANCED GAUSSIAN BACKGROUND MODELING ALGORITHM AND IMPLEMENTATION IN FPGA FOR REAL-TIME MOVING OBJECT DETECTION IN SURVEILLANCE VIDEO

GE GUO

September 2014

TECHNICAL REPORT
NO. 295

ENHANCED GAUSSIAN BACKGROUND MODELING ALGORITHM AND IMPLEMENTATION IN FPGA FOR REAL-TIME MOVING OBJECT DETECTION IN SURVEILLANCE VIDEO

Ge Guo

Department of Geodesy and Geomatics Engineering
University of New Brunswick
P.O. Box 4400
Fredericton, N.B.
Canada
E3B 5A3

September 2014

© Ge Guo, 2014

PREFACE

This technical report is a reproduction of a thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering in the Department of Geodesy and Geomatics Engineering, September 2014. The research was supervised by Dr. Yun Zhang (Department of Geodesy and Geomatics Engineering) and Professor Mary E. Kaye (Department of Electrical and Computer Engineering), and funding was provided by the Canada Research Chair (CRC) Program and the Atlantic Innovation Fund (AIF). The software and lab equipment used in this research was provided by Canadian Microelectronics Corporation Microsystems.

As with any copyrighted material, permission to reprint or quote extensively from this report must be received from the author. The citation to this work should appear as follows:

Guo, Ge (2014). *Enhanced Gaussian Background Modeling Algorithm and Implementation in FPGA for Real-Time Moving Object Detection in Surveillance Video.* M.Sc.E. thesis, Department of Geodesy and Geomatics Engineering, Technical Report No. 295, University of New Brunswick, Fredericton, New Brunswick, Canada, 108 pp.

ABSTRACT

A real-time solution of moving object detection (MOD) in surveillance video was explored in this work motivated by the practical need of real-time automated video analysis system. The core element of a moving object detection process is its background modeling algorithm in the content of surveillance and road monitoring applications. By reviewing and analyzing previous works, single Gaussian (SG) background modeling algorithm was selected and enhanced. Then a circuit that performs MOD with enhanced SG algorithm was designed and implemented in a Virtex6 FPGA of a ML605 evaluation board with other hardware components. The experiment results showed that the proposed MOD system could perform real-time MOD in a video of 1280×720p@30fps. It outperforms the software experiments/implementations and the state-of-art FPGA-based implementations.

DEDICATION

To my most supportive parents, Yawei Guo and Suyi Lv.

To my most beloved one, Sixian Zhang.

ACKNOWLEDGEMENTS

I would like to express my respect and sincere gratitude to my supervisors Professor Yun Zhang and Professor Mary Kaye for their helpful suggestions and guidance during the whole course of this work.

Table of Contents

ABSTRACT ii

DEDICATION iii

ACKNOWLEDGEMENTS iv

Table of Contents v

List of Tables viii

List of Figures ix

I. Introduction 1

1.1. Background and Motivation 1

1.2. Real-Time Definition 7

1.3. Problem Statement 8

1.4. Problem Breakdown and Thesis Organization 8

II. Literature Review 10

2.1. Review of the Alternative Hardware Platforms 10

2.1.1. DSP Platform Review 10

2.1.2. FPGA Platform Review 14

2.1.3. Summary 18

2.2. Review of Milestone Statistical Background Modeling Algorithms 19

2.2.1. Common Ground of Statistical Background Modeling Algorithms 19

2.2.2. Single Gaussian Algorithm 20

2.2.3. Mixture of Gaussian Algorithm 22

2.2.4. Kernel Density Estimation (KDE) 24

III. Solution Development 27

3.1. Algorithm Selection and Improvement 27
3.1.1. Memory Related Considerations 27
3.1.2. Algorithms Analysis and Comparison 30
3.1.3. SG Algorithm Enhancement 30
3.2. Pre-knowledge and System Features 51
3.2.1. Pre-mentioned Facts 51
3.2.2. Transferring Video Data in Real-Time 52
3.2.3. Pipelining Structure 53
3.2.4. System Feature Extraction 54
3.3. Proposed Design 55
3.3.1. System Level Descriptions 55
3.3.2. Functional Blocks Descriptions 59
IV. Implementation and Evaluation 68
4.1. Real-Time MOD Design Implementation 68
4.1.1. Circuit Implementation with EDA tools 68
4.1.2. Hardware Connection 69
4.2. Experiment Result 70
4.2.1. Real-Time Performance 70
4.2.2. Detection Quality 71
4.3. Comparison with Other Works 72
4.3.1. Comparison with other Software Implementation Work 72
4.3.2. Comparison with other FPGA Implementation Work 73
V. Conclusion and Future Work 76
5.1. Achievements 76
5.2. Improvements 76
5.3. Future Work 77
Bibliography 79

Appendix I – Implementation Note 85

FPGA Resource Utilization 85

Design Files and Schematics Description 85

Glossary 107

Curriculum Vitae

List of Tables

Table 1 -- Three-Level Image/Video Operations Characteristics 5

Table 2 -- Parameters and Their Average Bits on Data Bus 30

Table 3 -- Optimized SG Constants and Maximum JC Coefficients of Each Data Set 37

Table 4 -- Optimized Enhanced SG Constants and Maximum JC Coefficients 46

Table 5 -- Comparison of Parameter Optimization between SG and Enhanced SG 47

Table 6 -- Video Preprocessing Logic Blocks 60

Table 7 -- IP Cores' Classification in the Design 69

Table 8 -- Indirect Efficiency Comparison between Software and FPGA Implementation 73

Table 9 -- Real-Time Performance Comparison with Other FPGA Imlementations 74

Table 10 -- FPGA Resource Utilization of the Design 85

List of Figures

Figure 1 -- Three Steps in Automated Video Analysis....2

Figure 2 – Data Transformation within Automated Video Analysis System & Input / Output Data Relationship of 3 Levels of Image/Video Operations 4

Figure 3-- Typical Digital Signal Processing System Diagram....11

Figure 4-- Memory Interface at PHY 28

Figure 5 -- MPMC Interface Diagram....29

Figure 6 – Targeted Frames and Their Ground Truth 33

Figure 7 -- K-a-JC Surface of 'Time of Day' Data Set....35

Figure 8 -- K-a-JC Surface of 'Bootstrap' Data Set 36

Figure 9 -- K-a-JC Surface of 'Waving Trees' Data Set....37

Figure 10 -- Targeted Frame, Ground Truth, and SG Segmented Result in 'Time of Day' 38

Figure 11 -- Targeted Frame, Ground Truth, and SG Segmented Result in 'Bootstrap' 38

Figure 12 -- Targeted Frame, Ground Truth, and SG Segmented Result in 'Waving Trees' 38

Figure 13 -- Pixel Intensity, Mean, and Classification Boundaries at [118, 147]....39

Figure 14 -- Pixel Intensity, Mean, and Classification Boundaries at [98, 14]....41

Figure 15 -- Pixel Intensity, Mean, and Classification Boundaries at [35, 1]....42

Figure 16 – Details of Start-up Period of Figure 15 42

Figure 17 -- Th-JC curve of 'Time of Day' Data Set 45

Figure 18 -- Th-JC curve of 'Bootstrap' Data Set 45

Figure 19 -- Th-JC curve of 'Waving Trees' Data Set 46

Figure 20 -- Pixel Intensity, Mean, and Enhanced Classification Boundaries at [118, 147] 48

Figure 21 -- Pixel Intensity, Mean, and Enhanced Classification Boundaries at [98, 14] 48

Figure 22 -- Pixel Intensity, Mean, and Enhanced Classification Boundaries at [35, 1] 49

Figure 23 – Details of Start-up Period of Figure 22 49

Figure 24 -- Truth Ground, Original, and Enhanced Segmentation of 'Time of Day' Data set....50

Figure 25 -- Truth Ground, Original, and Enhanced Segmentation of 'Bootstrap' Data set 50

Figure 26 -- Truth Ground, Original, and Enhanced Segmentation of 'Waving Trees' Data set 51

Figure 27 -- Transferring Data of a Frame (3-by-3) 52

Figure 28 -- Virtual Timing Diagram of Transferring Video Data 52

Figure 29 -- Displaying a Frame from Data Stream 53

Figure 30 -- System Work Flow Diagram 56

Figure 31 -- System Data Flow Diagram 57

Figure 32 -- System Structure Diagram 58

Figure 33 -- Video Preprocessing Pipeline Diagram 59

Figure 34 -- First Major Format Change in Video Preprocessing Pipeline 60

Figure 35 -- XVSI Signals' Timing Diagram 62

Figure 36 -- Second Major Format Change in Video Preprocessing Pipeline 63

Figure 37 -- Enhanced SG Background Modeling Pipeline 64

Figure 38 -- Enhanced SG Algorithm Logic 64

Figure 39 -- Display Preparation Pipeline 66

Figure 40 -- Hardware Connection Diagram 70

Figure 41 -- Field Test: Detection of a Moving Ball 71

Figure 42 -- Field Test: Detection of a Waving Hand 72

Figure 43 -- Assembly View of the Design in XPS 87

Figure 44 -- Graphic View of the Design in XPS 88

Figure 45 -- Enhanced SG Pipeline Top View in BPS 89

Figure 46 – Enhanced SG Logic Block inside View in BPS 90

Figure 47 -- RGB to Gray Logic Block inside View in BPS 91

Figure 48 -- Absolute Operation Logic Block inside View in BPS 91

Figure 49 -- Classification Logic Block inside View in BPS 92

I. Introduction

1.1. Background and Motivation

Stationary cameras are often used in video surveillance and road monitoring systems. With the increasing concern of public safety, numerous stationary cameras are deployed across modern cities to cover as much area as possible. However, despite that so many cameras are deployed and they can run twenty-four-seven, the usage of the captured videos is rather primitive. According to a survey of 43 rail transit agencies in U.S., 2011 [1], the most common usage of surveillance video was 24-hours recording, and nearly one-half of the agencies did not monitor their cameras regularly, or at all, because of the personnel costs. A similar situation has also been a concern in UK, "With more than a million CCTV (Closed-Circuit Television) cameras in the UK alone, they are becoming increasingly difficult to manage," quoted from the New Scientist magazine [2]. "It is simple: we have so many cameras to capture video, but so few pairs of eyes." "If the technology takes off it could put an end to a longstanding problem that has dogged CCTV almost from the beginning." Indeed, if only there are extra pairs of "eyes" that can fill the vacancy caused by lack of man power to watch and analyze numerous video streams twenty-four-seven.

An automated video analysis system could be the extra pairs of "eyes". This kind of system can analyze the video stream(s) automatically without or with little people's attention and report only suspicious events to the professionals. ***With the assistance of automated video analysis system, the area covered by the surveillance cameras can be effectively monitored at last but only if the analysis is done in real-time.*** One benefit of analyzing video in real-time is the fast response to suspicious events. But there is another

reason for the system has to be real-time to be practical. That reason is there is no time to wait for post-processing in a twenty-four-seven working system. The detailed definition of real-time is introduced in section 1.2.

There are three key steps in an automated video analysis system identified by Yilmaz, Javed, and Shah [3]. Quoted from their work, these three steps are:

1) "Detection of interesting moving object."
2) "Tracking of such objects from frame to frame."
3) "Analysis of object tracks to recognize their behavior."

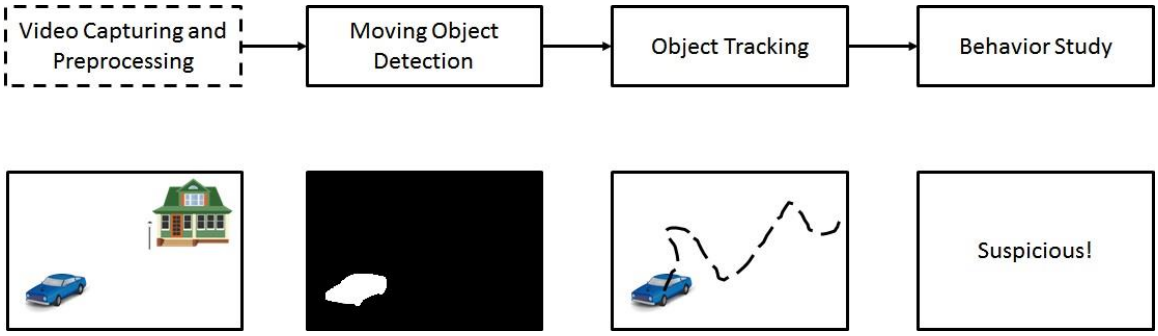

Figure 1 -- Three Steps in Automated Video Analysis

Figure 1 shows a simple example of an automated video analysis system. In this figure, three steps are connected together as a complete video processing chain.

In the first step, moving object detection (MOD) gives a bi-level image (image segmented into only two areas) from the current frame of video, where the region of moving object(s) (i.e., foreground) is labeled with 'white' and the region of other objects (i.e., background) is labeled with 'black'. Since this study is interested in moving object(s), the 'white' region is the 'region of interest' (ROI). Though the output is only bi-level, it remains a full size image, i.e., each pixel in the original video frame has a correspondent pixel in the segmented image.

In the second step, moving object tracking exploits information in the ROI and produces a trajectory of the moving car. This step usually involves three jobs in order. The first job is to tag the detected object(s) with image features (e.g., size, shape, texture, color, etc.) within the ROI of the original frame. The second job is to locate identified object(s) with the ROI's geometric properties. And the third job is to generate an up-to-date trajectory with up-to-date location information of the individual object. Note the trajectory is also a feature of the moving car.

In the third step, a "behavior study" deems the car's behavior "suspicious" and sends information accordingly. This step could be implemented with a pattern recognition system. A "suspicious event" could be defined as patterns of irregular trajectories. The output of the third step could be control signals triggering pre-defined actions. A pre-defined action here is sending information and could be anything like sending an alarm to authorities, recording the detected event, etc. In fact, this step can be implemented in various ways. However, in principle, the "behavior study" abstracts certain behavioral knowledge of the moving object(s) from the information provided by the previous step. And usually it also generates control signals according to the knowledge. Projection from knowledge to control signals is rather straight forward. If knowledge-to-control is taken as the last part of the system, the output of this step is control signals. Otherwise the output is behavioral knowledge of the moving object(s).

Input/output data between each step of the system is shown in Figure 2 (a).

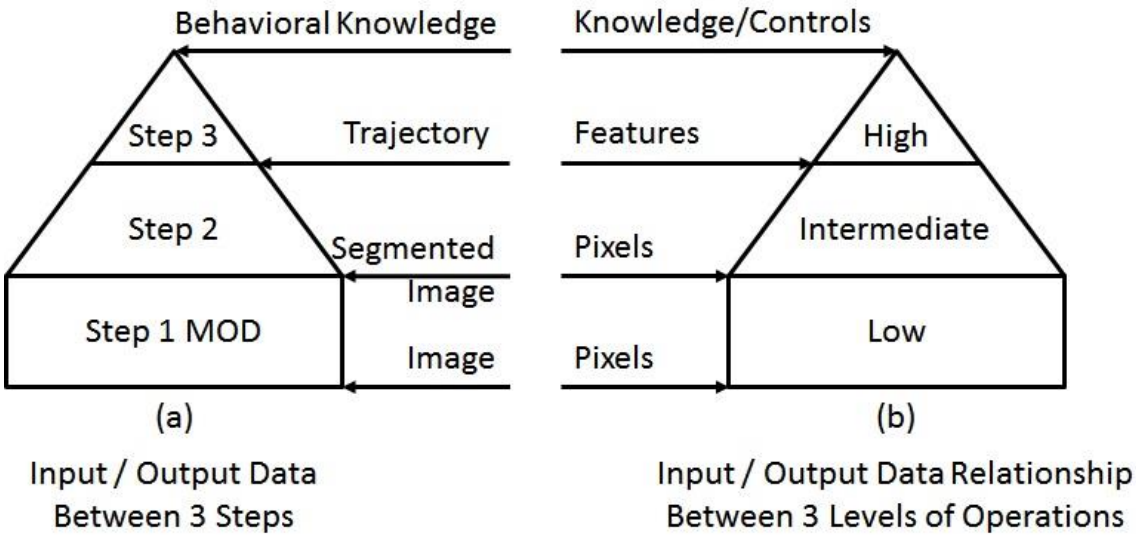

Figure 2 – Data Transformation within Automated Video Analysis System & Input / Output Data Relationship of 3 Levels of Image/Video Operations

Many research works can be found for each of the three steps that improve some algorithms in some way. But only a very small portion of the works takes real-time performance as the primary concern in modifying, designing, or implementing their algorithms. ***For the automated video analysis system to be practical, the author has taken improving the real-time performance as the primary focus in the thesis.***

To determine which step will yield the most improvement, one needs to know the computational characteristics of the three steps and then a comparison is made. In getting the computational characteristics, one can either review tons of compatible previous works for each step or learn them from an established and well-defined model.

Figure 2 (b), illustrates the input/output data relationship between successive three levels of image/video operations. These three levels have been classified traditionally in image/video processing [4, 5, 6, 7], namely low level, intermediate level, and high level.

There is an obvious one-to-one correspondence between the three steps in Figure 2 (a) and the three levels in Figure 2 (b). One can learn the computational characteristics of each from this model if it is well defined.

By reviewing the works of [4, 5, 6, 7], the characteristics of the three levels' image/video operations are summarized as follows: The low level operations are usually regular, high-bandwidth demanding, and data-intensive, the high level operations are more irregular, low-bandwidth demanding, and control-intensive, and feature extraction operations are in-between at an intermediate level. Table 1 lists these characteristics. Compared to that of higher levels, ***the low-level operations are usually considered as bottlenecks of efficiency within the image/video processing chain***, because they demand the most computational resources in terms of the quantity of computations and consumption of memory bandwidth [8] due to the amount of data to be processed. ***Correspondingly, the first step, moving object detection (MOD) is the major concern in improving real-time performance.***

Operation Level	Operation Structure	Memory Bandwidth Consumption	Data/Control
Low	Regular	High	Data Intensive
Intermediate	Regular	Intermediate	Less Data Intensive
High	Irregular	Low	Control Intensive

Table 1 -- Three-Level Image/Video Operations Characteristics

In terms of the moving object detection step, background subtraction is recognized as the most common approach for video captured by stationary cameras [3]. This is because the modern background subtraction approach is able to model the changes in the background [3]. The ability of modeling changes in background is due to state-of-art

background modeling algorithms used in the background subtraction approach. Thus the performance of the moving object detection with a stationary camera highly depends on its background modeling performance if the background subtraction technique is applied. This point has been mentioned by other researches as well. Toyoma, et al. [9] pointed out that background subtraction is the 'common element' in surveillance systems with stationary cameras and also point out that the background modeling is the most difficult part in background subtraction. Cheung and Kamath [10], also clearly indicate that background subtraction is a common approach in moving object detection and the 'heart' of any background subtraction technique is the construction of a statistical model that describes the background. ***In summary, background modeling is the core element of moving object detection, if background subtraction is applied with a stationary camera, which is the most common setting for a video surveillance system.***

With the focus on the background modeling within moving object detection, intensive research effort has been devoted to improving the robustness and adaptability of background modeling to date. Reviewing previous works [11, 12, 13] in the order of published date; the trend of increasing sophistication is evident in performing the statistical modeling for every pixel in a video. Modern cameras can provide video with resolution of several million pixels and can transfer several dozens of frames per second. ***With increasing sophistication in algorithms, real-time performance is not likely to be achieved on the conventional personal computers (PC) because most of CPUs are not created or specialized for massive data intensive workload*** [8]. And this issue has been reported by many other works. For example, Staffer and Grimson [12] implemented their proposed algorithm on an SGI O2 workstation and only get 11-13 frames per second (fps)

for a video of very low resolution at 160×120 pixels (p). Kristensen, Hedberg, and Jiang, et al. [14] also implemented the Staffer and Grimson's algorithm on a PC with an AMD4400+ processor and got only 4-16 fps for a video of 352×288p. Roshan and Zhang [15] examined the software implementation of several moving object detection algorithms and the experiment results also indicated that sophisticated background modeling algorithms are very time consuming running on conventional PC platform. Finding a feasible platform and implementing a suitable algorithm is the right way to improve real-time moving object detection.

1.2. Real-Time Definition

There are three common interpretations of 'real-time', namely 'real-time in the perceptual sense', 'real-time in the software programming sense', and 'real-time in the signal processing sense' [8]. Interpretation in the signal processing sense is used in this thesis, where 'real-time' means completing processing in the time available between successive input samples [16]. In the content of video processing, it means the algorithm must complete processing a frame between the start of a frame and start of the successive frame.

Note that the above definition does not give any specific number of how much time is available between two frames. It is because that time depends on the frame rate of the video. In other words, 'real-time in the signal processing sense' does not define exactly how fast the real-time performance is.

But how fast should it be to meet the desired frame rate? For an average multimedia display device, the screen needs to update at 30 fps (frame per second) for humans to

perceive continuous motion. Thus we consider executing the algorithm at 30 fps is the appropriate frame rate for real-time performance.

1.3. Problem Statement

In summary, the need of real-time automated video analysis motivates this research project. In investigating improvement of real-time performance, the MOD step becomes the major concern. Since a background modeling algorithm is the core element of the background subtraction approach that is most commonly used in MOD, the conventional implementation of background modeling algorithms is briefly reviewed. With increasing sophistication in background modeling algorithms, real-time performance is not likely to be achieved on conventional PCs because most of the CPUs used are not created or specialized for massive data intensive workloads.

Therefore, the aim of this thesis is to develop a real-time background modeling solution for moving object detection for a stationary camera using an alternative platform to the PC platform. The frame rate of the video captured by the camera should be no less than 30 fps. This goal is motivated by the need of automated video analysis in surveillance systems.

1.4. Problem Breakdown and Thesis Organization

Since the PC platform is infeasible for attempting real-time solution, the first issue was to decide on an alternative computing platform. The second issue was to select a suitable background modeling algorithm. Once the decisions were made, the next step was to devise a real-time moving object detection solution based on the selections. Then the solution was implemented and evaluated.

For solving the first issue, two types of hardware platforms were studied, namely DSP-based platforms and FPGA-based platforms. Each platform is introduced briefly and presented with implementation examples. For solving the second issue, several milestone background modeling algorithms are investigated, namely Single Gaussian Algorithm, Mixture of Gaussian Algorithm, and Kernel Density Estimation.

The decision of the platform was made based on the general performance of the presented examples. The decision of the algorithm involved some consideration of hardware features. Once the algorithm was selected, the algorithm was also improved to consider more hardware features in developing the real-time moving object detection solution. The design of the solution is presented by both high level and low level descriptions.

The proposed solution was implemented and evaluated. Evaluation includes experimental results and comparison with state-of-art works.

The contents of this these are organized in the chapters listed as following.

- *Chapter 1: Introduction of this thesis including background, motivation, and problem statement.*
- *Chapter 2: Literature review of hardware platforms and background modeling algorithms.*
- *Chapter 3: Development of the real-time moving object detection solution.*
- *Chapter 4: Implementation and evaluation of the proposed solution.*
- *Chapter 5: Conclusion of the thesis and future works.*

II. Literature Review

This chapter reviews two subjects. The first subject is alternative hardware platforms. The second subject is statistical background modeling algorithms.

The first review examines two types of platform, namely a DSP-based platform, and a FPGA-based platform. Both platforms considered can be used in embedded system.

The second review examines three algorithms, namely Single Gaussian Modeling (SG), Mixture of Gaussian Modeling (MOG), and Kernel Density Estimation Modeling (KDE).

2.1.Review of the Alternative Hardware Platforms

As discussed in section 1.1, the PC platform is not likely to support real-time background modeling algorithms, alternative hardware platforms are required. Considering that surveillance applications are likely to employ embedded systems, two types of hardware platforms that can be used to develop embedded systems are examined in this section. They are DSP-based platforms and FPGA-based platforms.

2.1.1. DSP Platform Review

DPS Introduction

"DSP stands for Digital Signal Processor. It is a specialized microprocessor with an architecture optimized for the operational needs of digital signal processing.

Digital signal processing algorithms typically require a large number of mathematical operations to be performed quickly and repeatedly on a series of data samples. Signals (perhaps from audio or video sensors) are converted from analog to digital, manipulated digitally, and then converted back to analog form. Figure 3 shows a typical digital

processing system. Note many digital signal processing applications have constraints on latency; that is, for the system to work, the digital signal processing operation must be completed within some fixed time, and deferred (or batch) processing is not viable." quoted from [17].

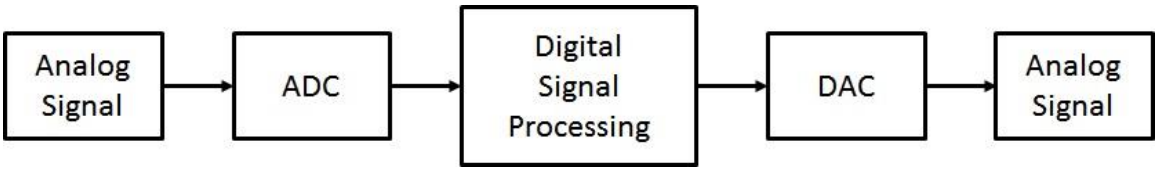

Figure 3-- Typical Digital Signal Processing System Diagram

Video signal processing algorithms are one type of digital signal processing algorithms. However, only high-performance DSPs are capable of meeting real-time requirements [8]. The challenge comes from the fact that the real-time video data throughput is very high as mentioned in section 1.1. The answers to this challenge are specific architectural enhancements addressing the data/computation throughput barrier in the new high-performance DSPs. The following discussed features are considered most useful for real-time video and image processing.

DSPs have been optimized for repetitive computation kernels with special hardware addressing modes like circular or modulo addressing mode. Such hardware addressing modes allows circular buffers to be implemented without having to constantly test for wrapping by software. Saving software operations is equal to saving time. This is especially beneficial for low level image/video processing operations such as convolution that contain intensive inner loops.

DSPs also have highly parallel architectures in a general sense. Most of DSPs have multiple functional units and VLIW/SIMD features. These features allow multiple

operations to be performed by a single long instruction and also multiple data are manipulated by a single instruction. These features can be used to exploit the inherit parallelism in image/video processing.

In addition, DSPs have been designed with high memory bandwidth in mind. On-chip DMA controllers, multilevel caches, and the buses connecting all the components together allow efficient data transferring between memories and devices. DMA controllers, particularly, access system memories independently and transfer data from and to the memories on the demand of processing units. With DMA taking care of the data transferring, processing units save time from less reading and writing operations. With more time that can be used in data processing, higher data throughput is allowed in the system. This is definitely very favored by real-time video processing because high data throughput is the challenge in the first place.

Besides the above three features, modern DSPs also have some other good features that are beneficial to general digital signal processing including the video signal processing. For example, DSPs often use memory architectures like Harvard architecture or a modified von Neumann architecture that are able to fetch multiple data and/or instructions at the same time. Another feature or trend perhaps, is that much higher frequency is allowed than before. New high-performance DSPs can work off frequency at the order of GHz, for example Davinci video processors [18] are able to work over 1 GHz. The features mentioned above make DSP a viable option for inclusion in a real-time video processing system.

DSP-based Implementations

DSP-based platforms have been found to be particularly popular in image filtering implementations. The reason is most likely that the looping computation structure of image filtering fits the DSPs architecture very well. Among all sorts of image filtering problems, non-linear filtering is relatively more challenging. One research group has consistently shown a single-chip high-performance DSP's capability in real-time non-linear filtering [19, 20, 21, 22, 23]. In [20, 21, 22], it was shown that by using a DSP-based platform, a real-time video rate (25 fps) edge-preserving, non-linear filtering could be achieved for 176×144p, so called Quarter Common Image Format (QCIF). High-performance DSPs were used in these implementations: TMS320C6701 DSP running at 167 MHz was used in [20] and TMS320C6711 DSP running at 150 MHz was used in [21, 22].

For moving object detection, however, the DSP-based platform is not often used. And most implementations use optical flow algorithms over the background modeling ones. For example, Iketani, Kuno, and Shimada et al. [24] employed 9 TMS320C40 DSPs with each performing a single operation of the optical flow algorithm in a video surveillance system and the system was able to run with the video resolution of 1024×256p at 15 fps. It is only very recently, with the newly available high-performance DSPs, implementations using background modeling algorithms have emerged. Published in December, 2013, a team implemented their background modeling algorithm on TI's high-performance TMS320DM642 DSP and achieved the performance of 352×288@14 fps [25].

2.1.2. FPGA Platform Review

FPGA Architecture

FPGA stands for Field-Programmable Gate Array. It is an integrated circuit designed to be configured or reconfigured into a custom circuit by the user after manufacturing hence "field-programmable". Its basic architecture is an array of logic blocks (that logically are equivalent to a combination of logic gates and flip-flops) and I/O pads with a network of programmable interconnects [26] hence "gate array".

Each logic block (called Configurable Logic Block (CLB) or Logic Array Block (LAB), depending on FPGA vendor) has the potential to be configured into many types of logic and can be connected to many other logic blocks and I/O pads via routing channels (networks) by programming the interconnects (switch box, whenever a vertical and horizontal channel intersect). Modern FPGA families can provide over 10 thousand logic blocks (e.g., Virtex6 FPGA family [27]). Considering the combination of these logic blocks, FPGAs today are capable of providing countless possibilities of implementations in terms of circuit logic and interfaces. In addition to the above capabilities, modern FPGAs also have included higher level functionalities, (e.g., multipliers, generic DSP blocks, embedded processors, high speed I/O logic, embedded memories, etc.) fixed into silicon. Having these common functions embedded into the silicon, reduced the area required and gave those functions increased speed compared to building them from primitives.

FPGA Programming

To define the behavior of the FPGA, the hardware description language (HDL) or schematic design can be used.

In practice, design files (in HDL or schematic) that represent desired circuit logic are provided to EDA tools along with constraint files. EDA tools translate the high-level design files to a binary file that is ready to be downloaded onto a targeted FPGA device to configure it.

The above process appears to be similar to that of compiling source code, written in software programming language, into executable code in machine language. Nevertheless, there is a significant difference. FPGA programming aims to implement custom digital circuits on the FPGA. Software programming, on the other hand, aims to produce executable code that can run on a processor.

EDA tools for programming FPGAs vary from vendor to vendor. However, they follow a similar road map and translate users' design step by step. Usually there are 6 steps involved.

1. Synthesis: This step translates the high-level circuit design to a netlist that describes the circuit by listing its instances and their connectivity. An "instance" can be anything from a simple register to a complex digital circuit.
2. Translation: This step "unfolds" the hierarchical instances in the previous netlist so only primitives are instanced in the new netlist. A "primitive" is an element that cannot be further unfolded such as a register or a logic gate.
3. Map: This step maps the primitives of the input netlist onto the physical resources (e.g., CLB, I/O pads, DSP blocks, etc.) in a targeted FPGA device and generates a new netlist that describes the circuit design physically, whereas the previous netlists do it logically.

4. Place and Route: This step defines how device resources are located and interconnected inside an FPGA. The definition is added to the previous physical netlist.
5. Programming File Generation: This step uses the physical netlist to generate a binary file for FPGA device configuration.
6. FPGA Configuration: This step downloads the binary file to the FPGA device to configure it.

FPGA Advantages

A digital circuit implemented in an FPGA is customizable. This nature is beneficial to implement video processing algorithms in many ways.

First, one can design glue logic on FPGA to connect it to useful devices in video processing like a camera board, an off-chip memory, video display devices, etc.

Second, one can optimize a video processing circuit in the FPGA to give the best possible data throughput. For example, one can exploit different levels of parallelism inherent in an image/video processing algorithm by using multiple identical functional units (e.g., multiple arithmetic logic units or even multiple microprocessors) in parallel or putting all necessary functional blocks in a chain running in synchronization (e.g., pipelining) or a hybrid of the former two.

Third, one can use custom memory configurations and/or addressing techniques in an FPGA to exploit efficiency in data locality of video data. Note video data is three-dimensional data and memories are initially designed for one-dimensional data only.

In addition, one can accurately define data processing time in an FPGA-based design. Data in a digital circuit is transferred via registers. Registers are known for synchronizing circuit's operation at the edge of the clock signals. A register in the data path introduces one clock delay for transferring data. Delay is known to the designer in simple circuits or can be accurately estimated by EDA tools.

In summary, FPGA-based design can interface multiple devices easily and achieve high data throughput both in the inner circuit and memory bandwidth. Plus its data processing time is very predictable. These advantages are very useful for all kinds of video processing implementations.

FPGA-based Implementations

FPGA-based platforms have been used to implement many video processing algorithms particularly in low-level image/video operations, e.g., image filtering operations, edge detection, moment calculation, Hough transform, image/video compression, etc.

In many cases, an FPGA platform has the potential to meet or exceed the performance of a single DSP or multiple DSPs. Since non-linear filtering examples have been presented in section 2.1.1, we also present FPGA-based non-linear filtering implementations here. An example of such an implementation was shown in [28] and the reported results showed that this implementation allowed an image of 512×512p to be filtered within 98 ms (≈11 fps) when the FPGA works off 8 MHz or within 23 ms (≈43 fps) when the FPGA works off 33.3 MHz. Another encouraging example is that of a fuzzy morphological filter implementation [29] which achieved a performance of 179 fps for 512×512p.

For moving object detection using background modeling algorithms, several FPGA-based examples also have been found. Two teams have consistently worked on the real-time solution of moving object detection. A team at Lund University, Sweden, presented a validated solution that is able to perform real-time moving object detection in video of 640×480p@25fps [30, 14, 31]. Another team at University of Napoli Federico II, Italy, presented their validated solution that is capable of real-time moving object detection in video of 1280×720p@20fps [32, 33, 34]. Kryjak, Komorkiewicz, and Gorgon in another separate work gave a performance of 640×480p@60fps [35].

2.1.3. Summary

From the above review of DSP-based platforms and FPG-based platforms, the following two conclusions are drawn:

1. For image filtering implementations, FPGA-based and DSP-based platforms seem to be equally used. However, in general, the FPGA-based platforms can outperform DSP-based ones significantly for this application.
2. In moving object detection implementations, FPGA-based platforms are preferred by researchers. The achieved real-time performance shows that FPGA-based platforms have the potential to outperform the DSP-based ones.

Based on the above conclusions, an FPGA-based platform was chosen to develop and implement the solution. Specifically, for this thesis, the Xilinx ML605 evaluation board (for details of ML605 please refer to [36]) was selected for its abundant available resources and potential capabilities of performing real-time video processing. The ML605 board has

a Virtex6-xc6vlx240t FPGA (for more information about this FPGA model please refer to [27]) on the board.

2.2.Review of Milestone Statistical Background Modeling Algorithms

Elhabian, El-Sayed, and Ahmed [37] classified the background modeling algorithms into two categories, recursive and non-recursive. The recursive method mainly contains two types of algorithms, recursive filters [38], and parametric statistical modeling [11, 12]. The non-recursive method mainly contains non-recursive filters [39, 40, 9, 41] and non-parametric statistical modelling techniques [13].

Cheung and Kamath [10] also surveyed background modeling algorithms based on a similar classification and also compared their performance. They remarked that statistical modeling algorithms generally outperform the filter ones.

A recent survey by Bouwmans [42] classified the background modeling algorithms into 5 categories. He remarked that the most frequently used models are the statistical in nature due to their robustness in changing environment, such as illumination change in a day.

Based on the previous systematic reviews in this field, the focus in this chapter is on three milestone statistical background modeling algorithms, namely single Gaussian modeling [11], a mixture of Gaussian modeling [12], and kernel density estimation [13].

2.2.1. Common Ground of Statistical Background Modeling Algorithms

Before describing details of each selected algorithm, it is helpful to have pre-knowledge of the common ground of the statistical background modeling algorithms that were investigated.

As mentioned in the Chapter 1, the result of background modeling algorithms is foreground and background segmentation at each frame of the video. But unlike other segmentation approaches that classify pixels of an image based on the information in the spatial domain, background modeling does it in the temporal domain at each individual spatial position. A pixel is defined as the intensity value of a position in the (x, y) plane at the frame number t. Then the set of all the pixels in a video is $\{I_{(x,y,t)} | x, y, t \in N, 1 \leq x \leq H, 1 \leq y \leq W, t > 1\}$ where I represents the intensity value, H is the height of the frame, W is the width. The members of $I_{(x,y,t)}$ are independent to each other. At a given position (x_0, y_0), $\{I_{(x_0,y_0,t)} | t \in N, t > 1\}$ is a time series. A statistical background modelling algorithm estimates the statistical properties of every time series on-the-fly and generates the corresponding classification time series accordingly, e.g. $\{fb_{(x_0,y_0,t)} | t \in N, t > 1\}$, where fb is either 1 i.e. foreground or 0 i.e. background. When every position in the (x, y) plane of the current frame is classified either as foreground or background, the entire frame is segmented i.e. $\{fb_{(x,y,t)} | x, y \in N, 1 \leq x \leq H, 1 \leq y \leq W, t = t_0\}$.

Most statistical background modeling algorithms have two sets of operations, namely model estimation, and foreground/background classification. Each algorithm investigated below is organized in this manner.

2.2.2. Single Gaussian Algorithm

Wren, Azarbayejani, and Darrell, et. al. [11] introduced a single Gaussian (SG) modelling algorithm in a gesture recognition system. However, the formalization of the algorithm is generalized in later work by Bouwmans et al. [43]. In the SG algorithm, the time series $\{I_{(x,y,t)} | x, y, t \in N, 1 \leq x \leq H, 1 \leq y \leq W, t > 1\}$ is modeled by a single

Gaussian. Then the probability of observing an intensity value at any position of a frame is defined as follow:

$$P\left(I_{(x,y,t)}\right) = \eta(I_{(x,y,t)}, \mu_{(x,y,t)}, \sigma^2_{(x,y,t)})$$

Where η is the Gaussian probability density function, $\mu_{(x,y,t)}$ is the estimation of the mean and $\sigma^2_{(x,y,t)}$ is the estimation of the variance.

The estimation principle of the SG algorithm is:

$$\mu_{(x,y,t)} = a \cdot \mu_{(x,y,t-1)} + (1-a) \cdot I_{(x,y,t)}$$

$$\sigma^2_{(x,y,t)} = a \cdot \sigma^2_{(x,y,t-1)} + (1-a) \cdot \left(I_{(x,y,t)} - \mu_{(x,y,t)}\right)^2$$

where a is the forgetting rate, a number smaller than but very near to 1.

The classification principle is:

$$fb_{(x,y,t)} = \begin{cases} 0, background, if \left|I_{(x,y,t)} - \mu_{(x,y,t)}\right| \leq K \cdot \sqrt{\sigma^2_{(x,y,t)}} \\ 1, foreground, if \left|I_{(x,y,t)} - \mu_{(x,y,t)}\right| > K \cdot \sqrt{\sigma^2_{(x,y,t)}} \end{cases}$$

where K is used to adjust the sensitivity of the foreground detection. In essence it decides the credible interval of the observations being background.

As simple as SG is, it is considered as a milestone algorithm for the initial introduction of statistical tools into this field. The algorithm is characterized as being suitable to deal with a stable environment with moderate illumination change [42]. A sudden change of background objects, or in illumination, will cause miss-classification at the beginning, but will be adapted gradually thanks to its on-the-fly estimating nature.

2.2.3. Mixture of Gaussian Algorithm

The mixture of Gaussian modelling (MOG) algorithm was proposed by Friedman and Russell [44] for a traffic surveillance system. In their algorithm, each time series at a position uses a mixture of three Gaussians of which each corresponds to road, vehicle, or shadow. This version of MOG initializes the Gaussian models using an off-line Estimate Maximum (EM) algorithm. The Gaussian models are heuristically labeled by the rule: The Gaussian model with the darkest mean value is shadow, of the remaining two, the one with the larger variance is labeled as vehicle, and the other is labeled as road. From this point, the label of each Gaussian model remains fixed. For classification, each pixel is labeled according to its Gaussian model's label. The model maintenance is the incremental EM algorithm.

Stauffer and Crimson [12] generalized the above idea by using a mixture of k Gaussians. Compared to the MOG proposed by Friedman and Russell [44], this new version MOG has a greater flexibility in number of Gaussian and distribution's label of each is not fixed. The probability of observing the current intensity value is given by the weighted average of probabilities in k Gaussian models. This probability is given by the following formula (for simplicity, the index x and y are omitted in the formula):

$$P(I_t) = \sum_{i=1}^{k} \omega_{i,t} \cdot \eta(I_{i,t}, \mu_{i,t}, \sigma_{i,t}^2)$$

where k is the number of Gaussians, $\omega_{i,t}$ is the weight of the i^{th} Gaussian model with the mean estimation $\mu_{i,t}$ and variance estimation $\sigma_{i,t}^2$. Note these k Gaussian models are descending ordered following the criterion ratio of $\omega_{i,t}/\sigma_{i,t}$ at time t.

The estimation principle has two cases and the case selected for each situation depends on if the "matching condition" is true for one of the models or none of them. The "matching condition" is:

$$\left|I_t - \mu_{i,t}\right| \leq K \cdot \sqrt{\sigma_{i,t}^2}$$

where K is a constant to adjust the credible interval of the observations that are matched to a Gaussian model. The first case is that a match is found, i.e. the "matching condition" is true for one of the k Gaussians. Note if the "matching condition" is true for more than one Gaussian, only the first match is adopted. The second case is that no match is found.

If it is the first case, the estimation principle of models' weights is:

$$\omega_{i,t} = a \cdot \omega_{i,t-1} + (1 - a) \cdot M_{i,t}$$

where a is forgetting rate and $M_{i,t}$ is 1 for the Gaussian model which is matched and 0 for the rest of them. After this estimation, the weights are normalized to guarantee that the sum of all the weights is 1, i.e. $\sum_{i=1}^{k} \omega_{i,t} = 1$.

The $\mu_{i,t}$ and $\sigma_{i,t}^2$ parameters for unmatched Gaussian models remain the same, while the parameters of the model which matches the new observation are updated as follows:

$$\mu_{i,t} = \rho \cdot \mu_{i,t-1} + (1 - \rho) \cdot I_t$$

$$\sigma_{i,t}^2 = \rho \cdot \sigma_{i,t}^2 + (1 - \rho) \cdot (I_t - \mu_{i,t})^2$$

where:

$$1 - \rho = (1 - a) \cdot \eta(I_{i,t}, \mu_{i,t-1}, \sigma_{i,t-1}^2)$$

If it is the second case, the least Gaussian model, the k^{th} model is replaced with a new one with:

$$\begin{cases} \omega_{k,t} = Low\ Priority\ Weight \\ \mu_{k,t} = I_t \\ \sigma_{k,t}^2 = Large\ Initial\ Variance \end{cases}$$

As to the classification principle, whether the latest observation is foreground or background depends on which Gaussian model the latest observation falls into. The first B Gaussian models are defined as "background Gaussian models". B is calculated as follow:

$$B = \underset{b}{\operatorname{argmin}} \sum_{i=1}^{b} \omega_{i,t} > Th$$

where Th is a proper threshold that divides the first B and rest of the Gaussian models. This division is based on the sum of the first b models' weights, $\sum_{i=1}^{b} \omega_{i,t}$.

The classification time series is generated as follow:

$$fb_t = \begin{cases} 0, background, if\ the\ I_t\ matches\ one\ of\ the\ first\ B\ models \\ 1, foreground, if\ the\ I_t\ matches\ none\ of\ the\ first\ B\ models \end{cases}$$

The MOG algorithm is considered as a milestone because it introduced parametric multi-models into this field. Compared to the SG algorithm, the MOG is more adaptive for outdoor scenes since some periodical background change can fit in more than one Gaussian model and still be classified as background.

2.2.4. Kernel Density Estimation (KDE)

Elgammal, Harwood, and Davis [13] proposed an algorithm estimating the probability density function using the kernel estimator for N intensity samples $\{x_1, x_2, \dots, x_N\}$. The probability of observing the latest pixel intensity value I_t is considered given by the following formula:

$$P(I_t) = \frac{1}{N}\sum_{i=1}^{N} K(I_t - x_i)$$

where $K(\cdot)$ is the kernel. These samples are initialized with N observations from a time window of W. The window contains the W latest pixel intensity observations, $\{I_{t-W+1}, I_{t-W+2}, \dots, I_t\}$. Either $N = W$ or the N samples are $N/2$ pairs of consecutive observations in the time window. Elgammal et al. suggested that kernel $K(\cdot)$ should be a normal destribution fucntion $\mathcal{N}(0, \sigma^2)$ and the initial two sets of samples should be initilized. One set is for the "short term" estimation, the other is for the "long term" estimation.

The estimation principle in essence is altering the oldest pair of elements in the samples with the latest pair of consecutive observations. As mentioned above, there are two types of estimation:

- Short term estimation uses a smaller time window to initialize a smaller set of samples. The altering is performed with a selective mechanism so that only if the latest observation is classified as background, the consecutive pair it belongs to will be adopted into the samples.
- Long term estimation uses a larger time window to initialize a larger set of samples. And the samples are altered blindly.

The classification principle is:

$$fb = \begin{cases} 0, background, if\ P_{long}(I_t) < Th\ and\ P_{short}(I_t) < Th \\ 1, foreground, if\ P_{long}(I_t) \geq Th\ or\ P_{short}(I_t) \geq Th \end{cases}$$

where Th is a manually set global threshold. $P_{long}(I_t)$ is the probability of observing I_t calculated with the long term samples, $P_{short}(I_t)$ is that with the short term samples.

In addition to the new technique for background modeling, a false detection reduction strategy and a shadow reduction strategy are also introduced by Elgammal et al. The former is based on local neighborhood pixels and the latter is based on color scaling.

KDE initially introduced non-parametric modeling into background modeling. Elgammal et al. claimed that this algorithm is more sensitive in detecting a moving target than the parametric ones and less affected by the presence of the target in the scene. Bouwmans [42] also commented that KDE is more adapted for outdoor scenes where dynamic backgrounds appear but less suited for illumination changes.

III. Solution Development

This chapter describes the road map of developing the real-time MOD solution. The ML605 FPGA-based platform was selected for the proposed solution based on the discussion in section 2.1. Based on the review in section 2.2 and memory considerations, this chapter discusses the algorithm selection and its enhancement. The features of the system are then discussed and the overall design is introduced in the end of this chapter.

3.1.Algorithm Selection and Improvement

In section 2.1.3, an FPGA-based platform was selected for the real-time MOD solution. There are different concerns for implementing a background modeling algorithm on an FPGA than on the conventional PC-based platform. The primary concern is memory related. In this section, the memory related considerations will be discussed first, then specific considerations for each of the three algorithms. The SG algorithm is selected and is discussed further. In the end, the SG algorithm is enhanced before being mapped into the design of the solution.

3.1.1. Memory Related Considerations

Memory Storage Requirements Consideration

Three statistical background modeling algorithms and their common ground were introduced in section 2.2.1. In background estimation, current pixel intensities with either the digital value of the latest statistical parameters or historical pixel intensities are used. This indicates that memory storage is required for keeping these data during the background estimation. Given that multiple parameters are required for each pixel's

background estimation and the number of pixels of each frame is on the order of a million for HD videos, the required memory amount could easily exceed that available by the on-chip memory (e.g., Virtex6-xc6vlx240t FPGA has a maximum of 15 Mb [27]). An off-chip DDR3 SDRAM was used to meet the memory storage requirement since it is the fastest and largest memory available on the ML605 board.

Memory Bandwidth Consumption Consideration

SDRAM, like many other generic memory systems, was originally designed for one-dimensional data access and thus cannot properly address the spatial locality necessary for two-dimensional or three-dimensional image and video data [8]. Figure 4 shows that there is only one port that interfaces the FPGA chip with SDRAM at the physical layer (PHY).

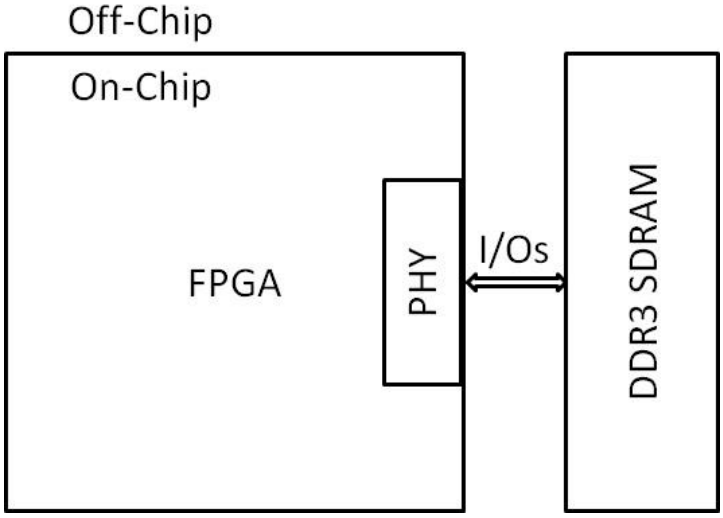

Figure 4-- Memory Interface at PHY

Custom memory addressing schemes were needed to allow efficient memory access of multi-dimensional data. Moreover, arithmetic operations prefer simultaneous access to all needed data. To get multiple pieces of data from memory, and to write multiple parameters

back to memory, a multi-port memory interface is needed. Given that designing a memory controller from primitives is a non-trivial work, a customizable multi-port memory controller (MPMC) IP core [45] was used in the design to meet the memory interface preference. An IP (intellectual property) core is a block of logic or data that is used in FPGA-based circuit design. It can be designed by the user, provided by FPGA vendor, or provided by third party. Figure 5 shows a simplified diagram of a multi-port memory controller that interfaces other FPGA Logics with off-chip memory.

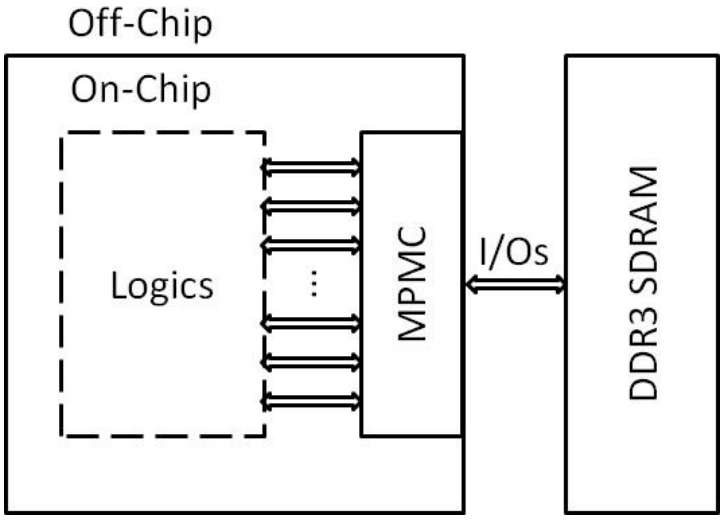

Figure 5 -- MPMC Interface Diagram

The MPMC can provide several video frame buffer connectors (VFBC, for more information please refer to [45]) that can access two-dimensional data separately and efficiently. Such a property is favored in video processing; however, certain limitations are attached. There are a limited number of ports available to be configured as VFBC and each VFBC port has a fixed size data bus (number of bits for transferring data). In other words, there are a limited number of bits available to represent background model parameters for each pixel. This number is 32 for the MPMC IP core.

3.1.2. Algorithms Analysis and Comparison

Given that only 32 bits are available, the number of parameters to be represented becomes the issue since more parameters means more loss of precision in calculations. This is extremely important for background estimation for each pixel which is an on-line iterative process and losing precision excessively may neutralize the effect of this process. In this context, we prefer a fewer number of background modelling parameters with greater precision. Table 2 lists the name and number of parameters for each algorithm discussed in section 2.2.

Table 2 -- Parameters and Their Average Bits on Data Bus

Algorithm	Parameters	Number	Number of bits for each parameter (average)
SG	Mean Variance	2	16
MOG	Mean×3 Variance×3 Weight×3 Order×3 (For 3 Gaussian Models)	12	2-3
KDE	Samples	32 (max)	1

Observed from Table 2, the SG algorithm has the smallest number of parameters with the greatest precision per parameter so the SG algorithm was selected for further discussion in section 3.1.3.

3.1.3. SG Algorithm Enhancement

In this section, an enhanced version of the selected SG algorithm is proposed. Firstly, the SG algorithm is evaluated after optimization. Then the enhanced SG algorithm is

proposed based on the analysis of the evaluation. Finally, the enhanced SG algorithm is also evaluated by comparing its results with the optimized SG algorithm's results.

SG Evaluation

The SG algorithm was evaluated by optimizing the constants a and K then continuing with observing the segmentation results using the optimized constants. For optimization, a measurement of the similarity between the segmentation result and "ground truth" is used, namely the Jaccard Similarity coefficient (JC coefficient, please see section below). The bigger the JC coefficient is the more similar the segmentation result to the "ground truth". The optimum pair of a and K gives the maximum JC coefficient for a given frame. The wallflower data set [9] provides several sets of image sequences and each set has one image with a hand segmented "ground truth". More details of "ground truth" provided in wallflower data set are discussed below.

Step1: SG Constants Optimization

JC coefficient:

The JC coefficient is named after Paul Jaccard (18 November 1868 in Sainte-Croix - 9 May 1944 in Zurich) who was a professor of botany and plant physiology at the ETH Zurich. The coefficient was initially included in a biology book [46]. Rosin and Ioannidis [47] referred to [46] and interpreted the JC coefficient as a measurement in pixel-based evaluation of image thresholding for change detection, where the coefficient is calculated with three values, namely true positive (TP), false positive (FP), and false negative (FN). The JC coefficient formula was given as follows:

$$JC = \frac{TP}{TP + FP + FN}$$

Elhabian et. al. [37] referred to [47] and regarded this measurement as a pixel-based method for quantifying the performance of image segmentation in moving object detection.

In moving object detection, the specifications of the TP, the FP, and the TN are:

- TP: the number of foreground pixels correctly detected; It is equivalent to the number of pixels that are foreground both in the segmentation result and in "ground truth".
- FP: the number of background pixels incorrectly detected as foreground (also known as false alarms); It is equivalent to the number of pixels that are foreground in the segmentation result but background in "ground truth".
- FN: the number of background pixels incorrectly detected; It is equivalent to the number of pixels that are background in the segmentation result but foreground in "ground truth".

Wallflower data set:

As mentioned above, the wallflower data set provides several sets of image sequences and each set has an image with a hand segmented ground truth. Three sets of image sequences were selected, namely 'Time of Day', 'Bootstrap', and 'Waving Trees'. 'Time of Day' has 5890 images and the 1850th image is provided with a ground truth of foreground/background segmentation. 'Bootstrap' has 3055 images and the 300th image is provided with a ground truth. 'Waving Trees' has 287 images and the 248th is provided with a ground truth. The images with the provided 'ground truth' are referred to as 'targeted

frames' in the following sections since only these frames' segmentation results have reference to compare with and were used in evaluating the algorithm. Figure 6 shows these special images with their provided 'ground truth'.

Figure 6 – Targeted Frames and Their Ground Truth

Optimization Principle:

For each pair of a and K there is a segmentation result of the targeted frame in each data set. So the JC coefficient can be calculated accordingly. The optimization principle is simply to find the maximum JC coefficient in a-K plane and retrieve the pair of a and K as the optimized constants of the SG algorithm for the data set the targeted frame belongs to.

Optimization Result:

Figure 7 shows the relationship between the a-K plane and the JC coefficient of the targeted frame (i.e. 1850^{th} frame) in data set 'Time of Day'. Figure 7 shows that the maximum JC coefficient is 0.61649, and the correspondent pair of $[a, K]$ is$[0.993, 2.3]$.

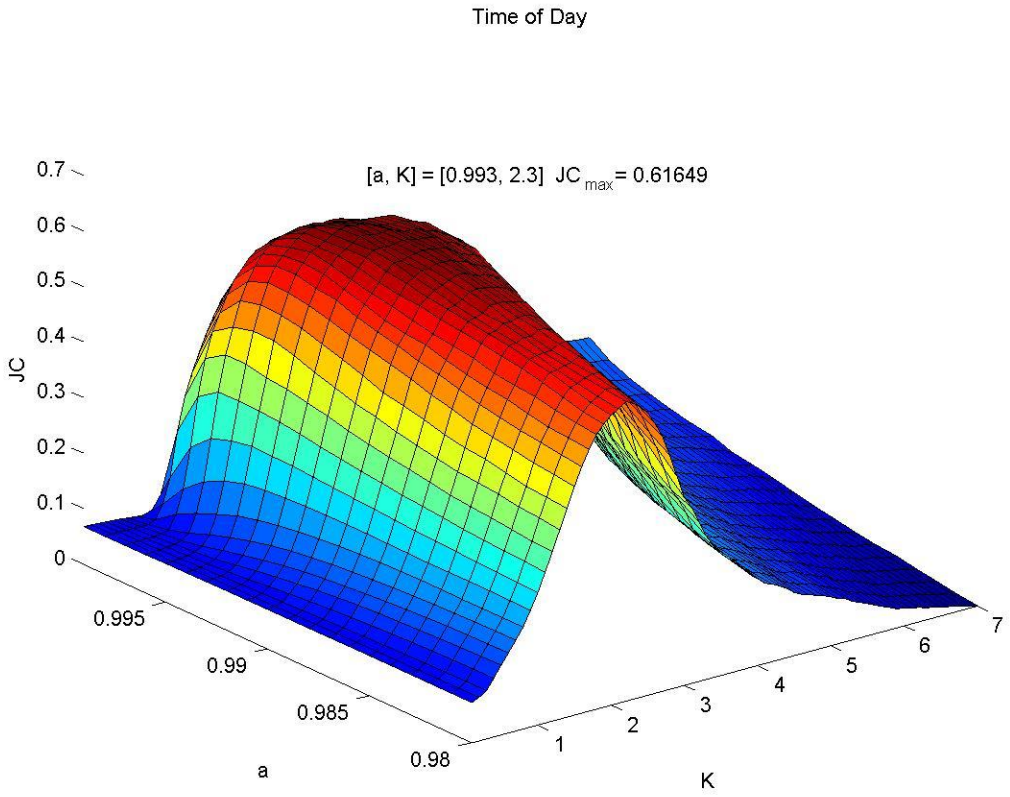

Figure 7 -- K-a-JC Surface of 'Time of Day' Data Set

Similarly, for the 'Bootstrap' data set, the maximum JC coefficient is 0.28235, correspondent $[a, K]$ is $[0.99, 0.6]$ (Figure 8); for 'Waving Trees' data set, the JC coefficient is 0.61317, correspondent $[a, K]$ is $[0.994, 1.8]$ (Figure 9). The optimized constants $[a, K]$ and maximum JC coefficients of each target frame in every data set are listed in Table 3.

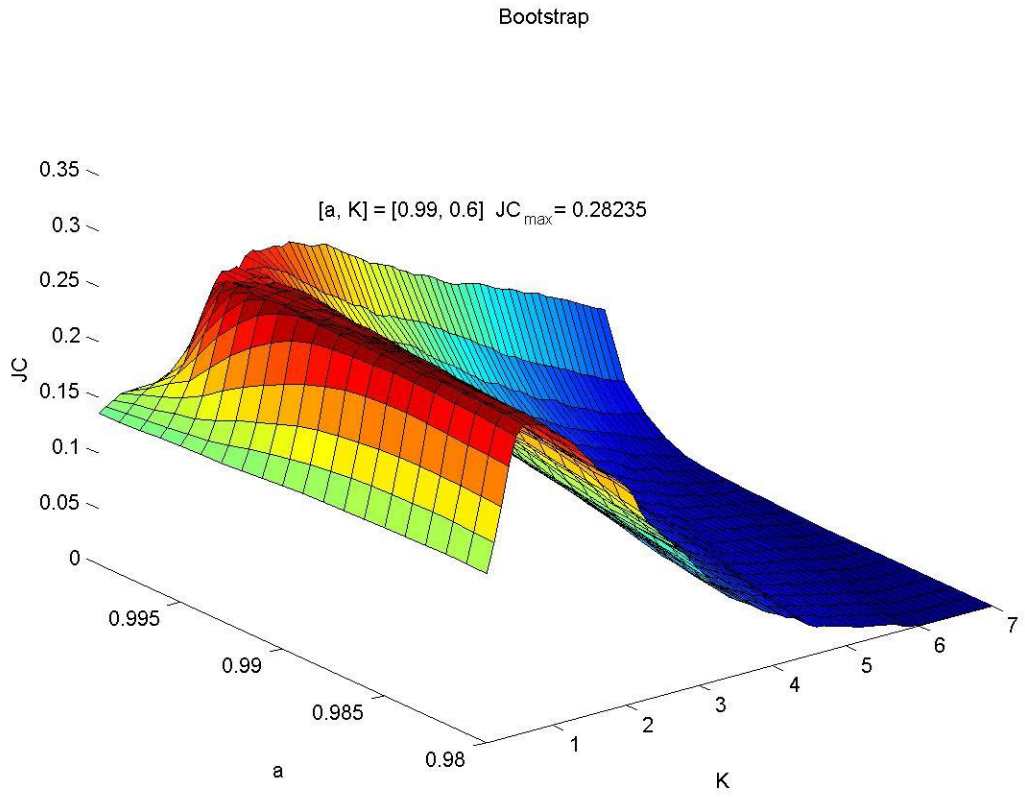

Figure 8 -- K-a-JC Surface of 'Bootstrap' Data Set

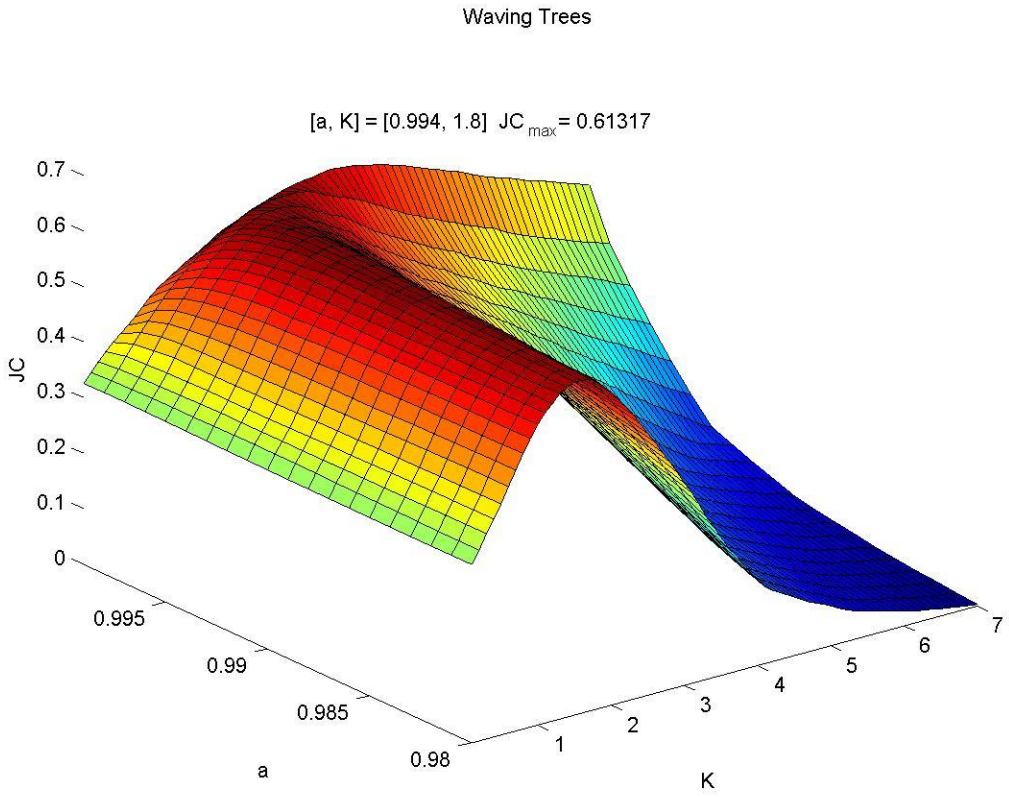

Figure 9 -- K-a-JC Surface of 'Waving Trees' Data Set

Table 3 -- Optimized SG Constants and Maximum JC Coefficients of Each Data Set

Data set	SG algorithm Constants	JC coefficients of the target frame
Time of Day	$a = 0.993$ $K = 2.3$	0.61649
Bootstrap	$a = 0.99$ $K = 0.6$	0.28235
Waving Trees	$a = 0.994$ $K = 1.8$	0.61317

Step2: Observing the Optimized Segmentation Results

Figure 10, Figure 11, and Figure 12 show the targeted frames, their ground truth, and its optimized segmentation result by the SG algorithm.

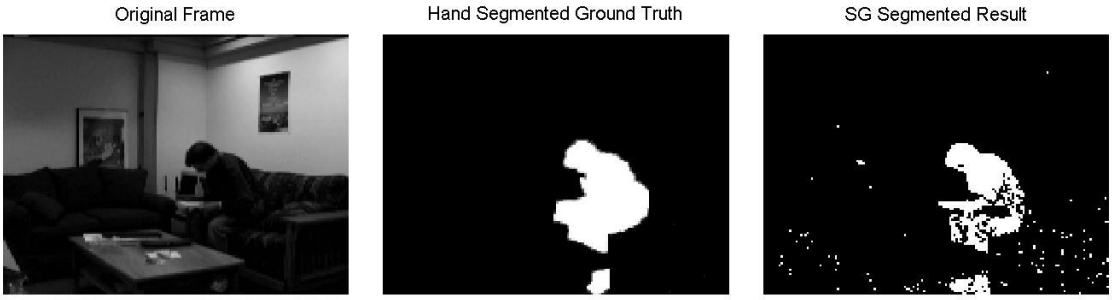

Figure 10 -- Targeted Frame, Ground Truth, and SG Segmented Result in 'Time of Day'

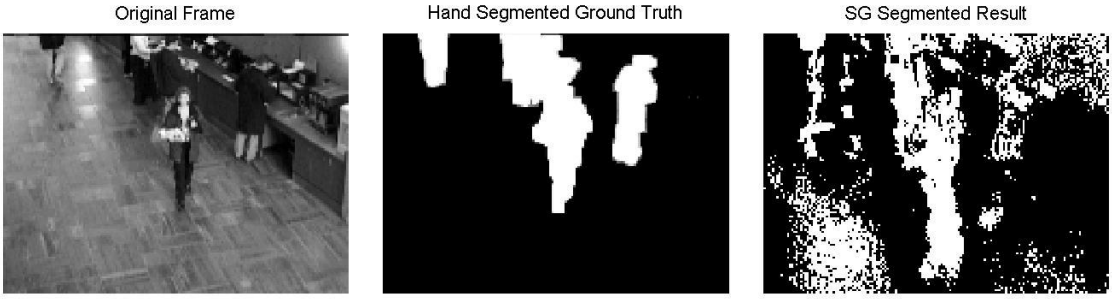

Figure 11 -- Targeted Frame, Ground Truth, and SG Segmented Result in 'Bootstrap'

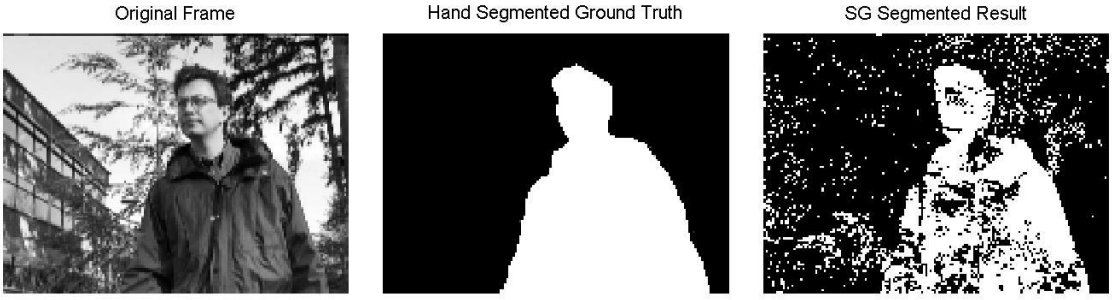

Figure 12 -- Targeted Frame, Ground Truth, and SG Segmented Result in 'Waving Trees'

By observing the segmentation results, the most significant defect in the results is false positive (FP). Since they appear as white dots in the segmentation results, it is referred to as ‘salt’ noise as well.

Noise Analysis and Reducing

Step1: Noise Analysis

'Salt’ noise, as motioned above, is false positive in nature. To analyze the cause of this sort of positives, the SG algorithm process was observed at several individual positions in the (x, y) plane of the ‘Time of Day’ data set.

Figure 13 shows the sequences of pixel intensity ($I_{(x,y,t)}$), mean value ($\mu_{(x,y,t)}$), and classification boundaries ($\mu_{(x,y,t)} \pm K \cdot \sqrt{\sigma^2_{(x,y,t)}}$) at position($x = 118, y = 147$).

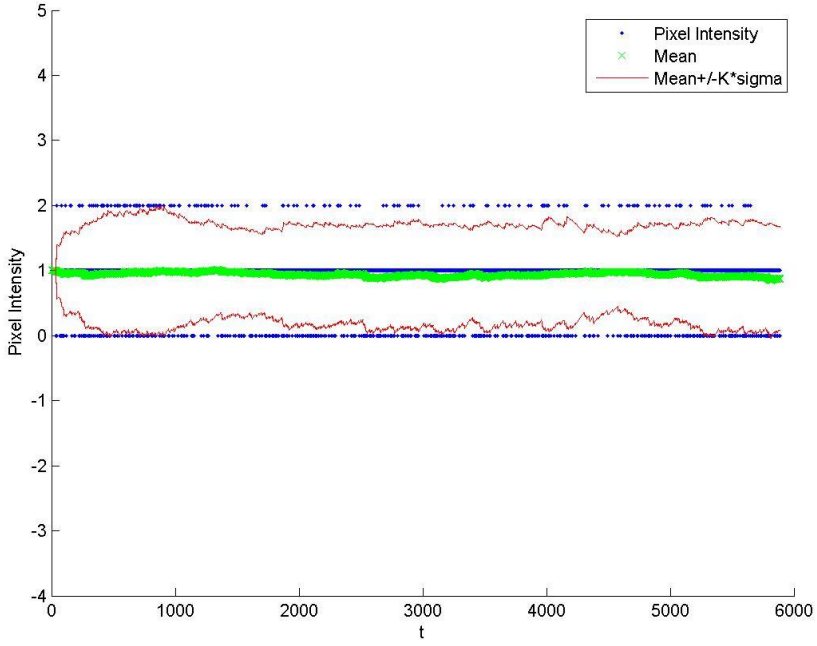

Figure 13 -- Pixel Intensity, Mean, and Classification Boundaries at [118, 147]

As observed from Figure 13, pixel intensity ($I_{(x,y,t)}$) vibrates strictly between 0 and 2. This fact suggests that pixels at this position are actually all background for every frame. Yet many pixel intensities are observed beyond the boundaries ($\mu_{(x,y,t)} \pm K \cdot \sqrt{\sigma^2_{(x,y,t)}}$) and are classified as positives. So the positives are all false positives. The total number of false positives is 704.

The reason why so many FP are observed here is because the classification boundaries ($\mu_{(x,y,t)} \pm K \cdot \sqrt{\sigma^2_{(x,y,t)}}$) are too near to the mean values ($\mu_{(x,y,t)}$). And this is mainly due to the low amplitude of the intensity vibration (i.e. ±1). As a consequence, in the SG algorithm, the variance ($\sigma^2_{(x,y,t)}$) (or standard derivation ($\sqrt{\sigma^2_{(x,y,t)}}$)) is too small to widen the boundaries to include the vibration itself.

Figure 14 shows the sequences of pixel intensity ($I_{(x,y,t)}$), mean value ($\mu_{(x,y,t)}$), and classification boundaries ($\mu_{(x,y,t)} \pm K \cdot \sqrt{\sigma^2_{(x,y,t)}}$) at position($x = 98, y = 14$).

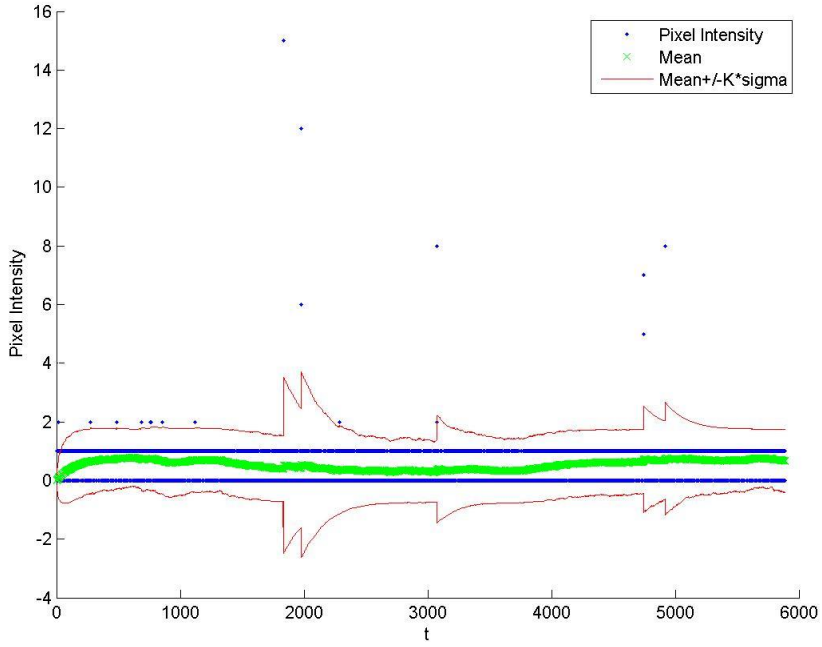

Figure 14 -- Pixel Intensity, Mean, and Classification Boundaries at [98, 14]

At this position, classification boundaries successfully distinguished some foreground pixels that are far from the mean values. But they also classified some pixels near to the mean values (i.e., with distance smaller than 2) as foreground. Such classifications are very likely to be false positives.The total number of positives is 37.

The misclassification issue here is mainly because of the rareness of some background intensities. Note that the nature of the SG algorithm is to classify the rare event as foreground by assuming the moving object appears rarely against the background. But this assumption cannot exclude the situation like 'rare background intensities'.

Figure 15 shows the sequences of pixel intensity ($I_{(x,y,t)}$), mean value ($\mu_{(x,y,t)}$), and classification boundaries ($\mu_{(x,y,t)} \pm K \cdot \sqrt{\sigma^2_{(x,y,t)}}$) at position($x = 35, y = 1$).

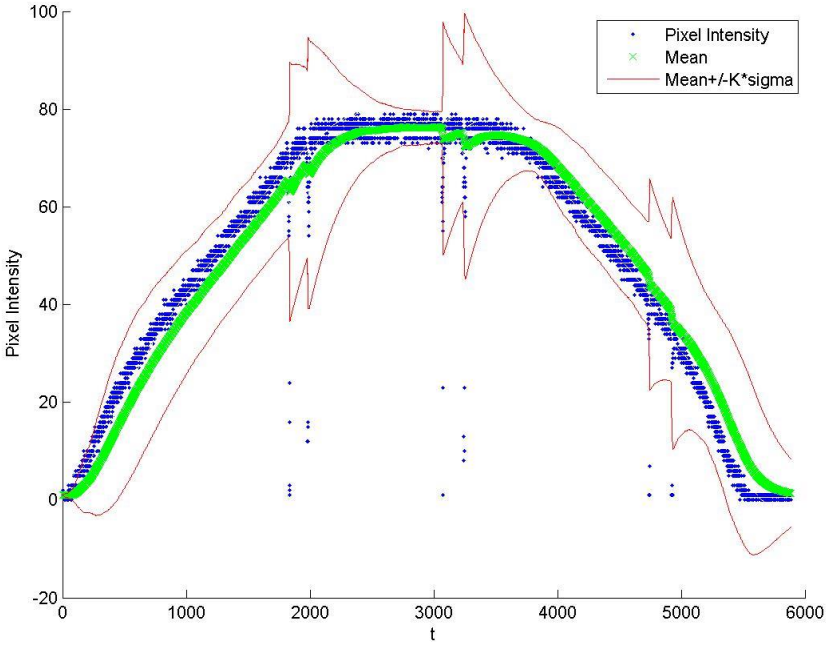

Figure 15 -- Pixel Intensity, Mean, and Classification Boundaries at [35, 1]

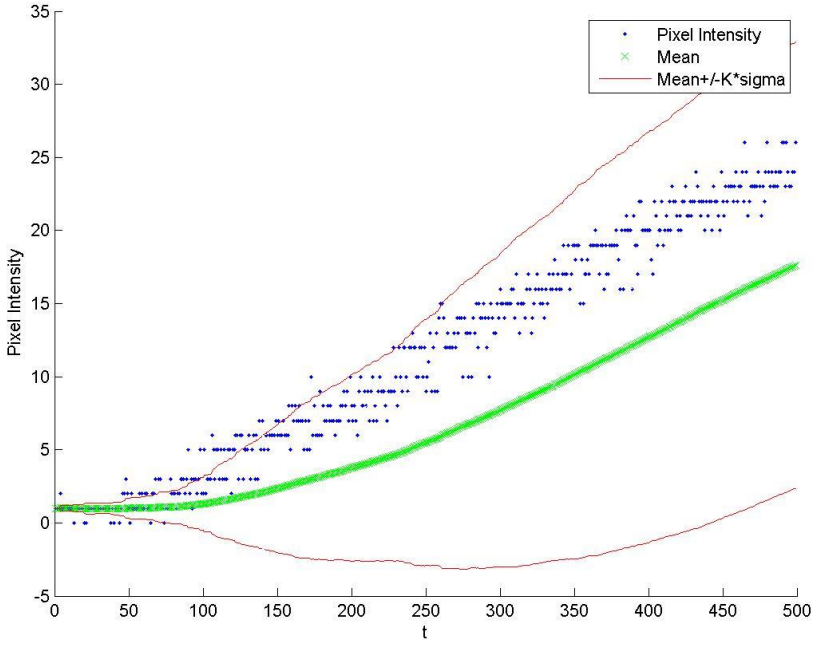

Figure 16 – Details of Start-up Period of Figure 15

At this position, the boundaries most ideally included the background pixels that are near the mean values and excluded foreground pixels that are far from them. However there are still some misclassifications at the beginning of the sequences (see Figure 16). This is because the SG algorithm needs some time to adapt the intensity variance. There are 110 positives in total.

From the above analysis, three causes of false positives were concluded:

1. Small variance ($\sigma^2_{(x,y,t)}$) due to small vibration of the background intensities.
2. Rareness of some background intensities.
3. Start-up time to adapt the vibration of the pixel intensities.

All these three cause a narrow width between the classification boundaries ($\mu_{(x,y,t)} \pm K \cdot \sqrt{\sigma^2_{(x,y,t)}}$) making it a more difficult classification problem.

Step2: Noise Reduction

To address the issue mentioned above and reduce the salt noise in segmentation, the following enhanced SG algorithm that can widen the coverage between boundaries is proposed:

The estimation principle is unchanged:

$$\mu_{(x,y,t)} = a \cdot \mu_{(x,y,t-1)} + (1-a) \cdot I_{(x,y,t)}$$

$$\sigma^2_{(x,y,t)} = a \cdot \sigma^2_{(x,y,t-1)} + (1-a) \cdot \left(I_{(x,y,t)} - \mu_{(x,y,t)}\right)^2$$

where a is the forgetting rate, a number smaller than but very near to 1.

The classification principle is changed:

$$fb_{(x,y,t)} = \begin{cases} 0, background, if\ |I_{(x,y,t)} - \mu_{(x,y,t)}| \leq max(Th, K \cdot \sqrt{\sigma^2_{(x,y,t)}}) \\ 1, foreground, if\ |I_{(x,y,t)} - \mu_{(x,y,t)}| > max(Th, K \cdot \sqrt{\sigma^2_{(x,y,t)}}) \end{cases}$$

where K and Th are used to adjust the sensitivity of the foreground detection. Th makes sure that the width between the boundaries is no less than $2 \cdot Th$ around the mean value ($\mu_{(x,y,t)}$).

Enhanced SG Evaluation

Similar to the evaluation of SG (SG **Evaluation** in section 3.1.3), the constants of the enhanced SG algorithm were optimized, and then the optimized segmentation results were observed. The optimized segmentation results of the enhanced SG algorithm were also compared to that of the original optimized SG algorithm.

Step1: Optimization

Here the optimized a and K inherited from previous Step1: SG Constants Optimization were used. So only the Th needs to be optimized in this section. The optimization principle is finding the maximum JC coefficient by varying Th then retrieving the correspondent Th which is the optimum one.

Figure 17 shows the $Th - JC$ relationship of the targeted frame in the 'Time of Day' data set. Also the maximum JC coefficient and the optimum Th are marked in it. The similar content in 'Bootstrap' is shown in Figure 18 and that in 'Waving Trees' is shown in Figure 19.

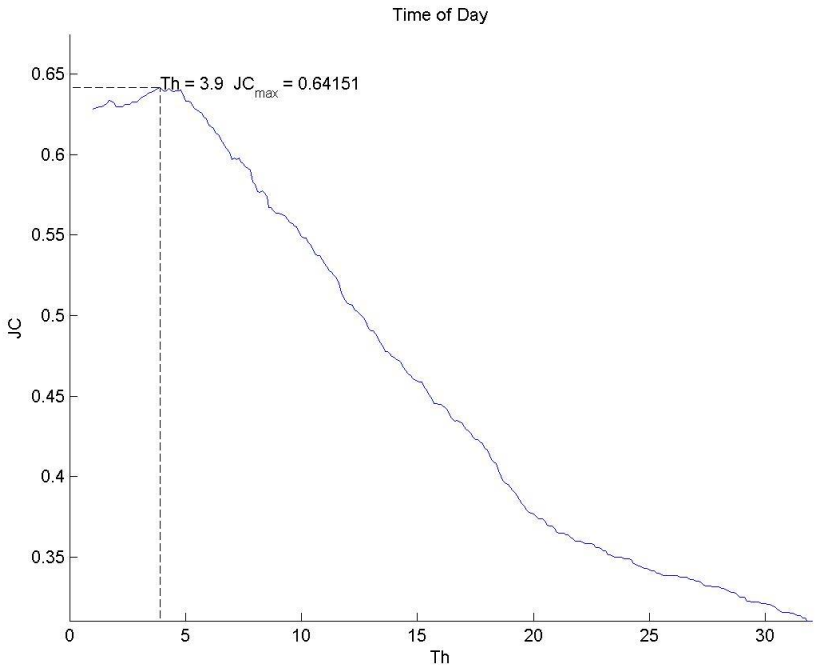

Figure 17 -- Th-JC curve of 'Time of Day' Data Set

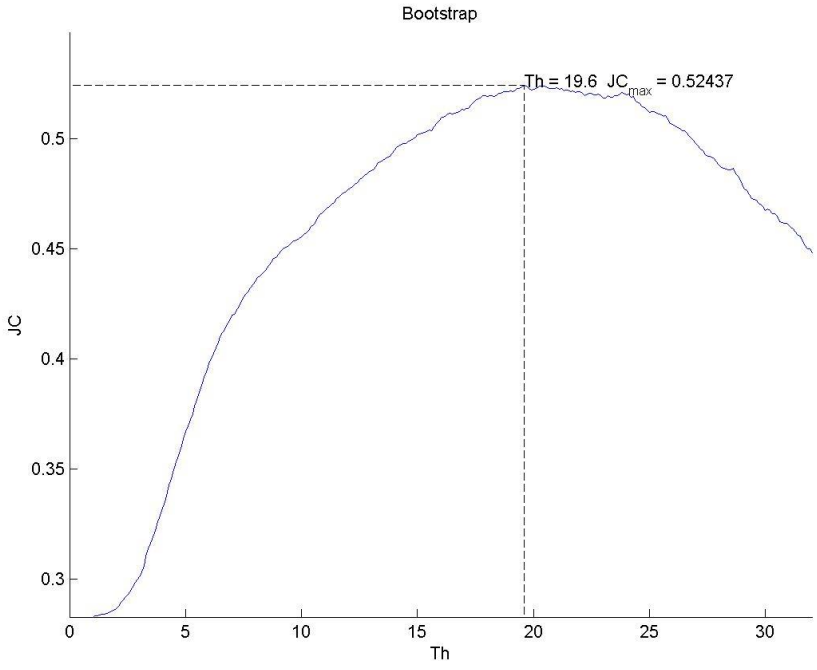

Figure 18 -- Th-JC curve of 'Bootstrap' Data Set

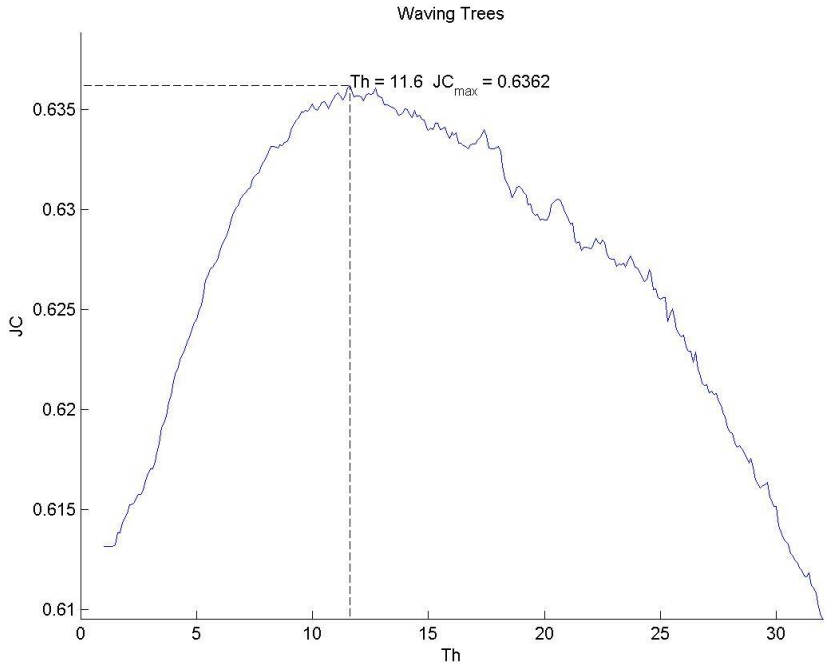

Figure 19 -- Th-JC curve of 'Waving Trees' Data Set

The optimization results were generalized in Table 4.

Table 4 -- Optimized Enhanced SG Constants and Maximum JC Coefficients

Data set	Enhanced SG algorithm constants	JC coefficient of the target frame
Time of Day	$a = 0.993$ $K = 2.3$ $Th = 3.9$	0.64151
Bootstrap	$a = 0.99$ $K = 0.6$ $Th = 19.6$	0.52437
Waving Trees	$a = 0.994$ $K = 1.8$ $Th = 11.6$	0.6362

Step2: Observation and Comparison

Table 5 combines Table 3 and Table 4 and lists the improvement of JC coefficients introduced by the proposed enhancement on the SG algorithm.

Table 5 -- Comparison of Parameter Optimization between SG and Enhanced SG

Data set	SG Constants	JC by SG	Enhanced SG Constants	JC by Enhanced SG	JC Improvement
Time of Day	$a = 0.993$ $K = 2.3$	0.61649	$a = 0.993$ $K = 2.3$ $Th = 3.9$	0.64151	0.02502
Bootstraps	$a = 0.99$ $K = 0.6$	0.28235	$a = 0.99$ $K = 0.6$ $Th = 19.6$	0.52437	0.24202
Waving Trees	$a = 0.994$ $K = 1.8$	0.61317	$a = 0.994$ $K = 1.8$ $Th = 11.6$	0.6362	0.02303

Figure 20 shows the sequences of pixel intensity ($I_{(x,y,t)}$), mean value ($\mu_{(x,y,t)}$), original classification boundaries ($\mu_{(x,y,t)} \pm K \cdot \sqrt{\sigma^2_{(x,y,t)}}$), and enhanced classification boundaries ($\mu_{(x,y,t)} \pm max(Th, K \cdot \sqrt{\sigma^2_{(x,y,t)}})$). With the new enhanced boundaries, all the pixels were classified correctly as background. The number of false positives decreases from 704 (in Figure 13) to 0. In Figure 21, the new boundaries suppressed the rare background pixels that were false positives. The total number of positives decreases from 37 (in Figure 14) to 7. In Figure 22 and Figure 23, the enhanced classification boundaries reduced misclassification in the initialization stage of the algorithm. The total number of positives decreases from 110 (in Figure 15) to 65.

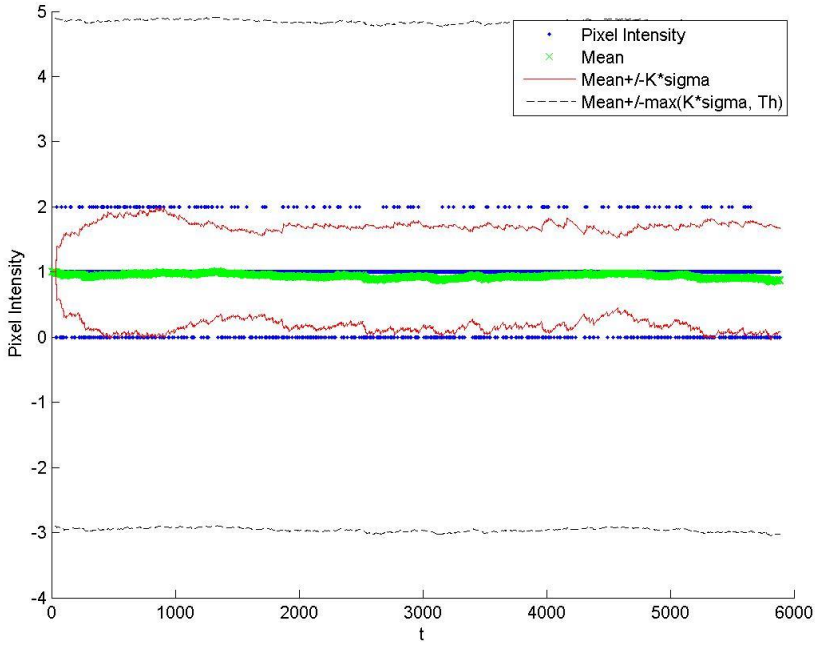

Figure 20 -- Pixel Intensity, Mean, and Enhanced Classification Boundaries at [118, 147]

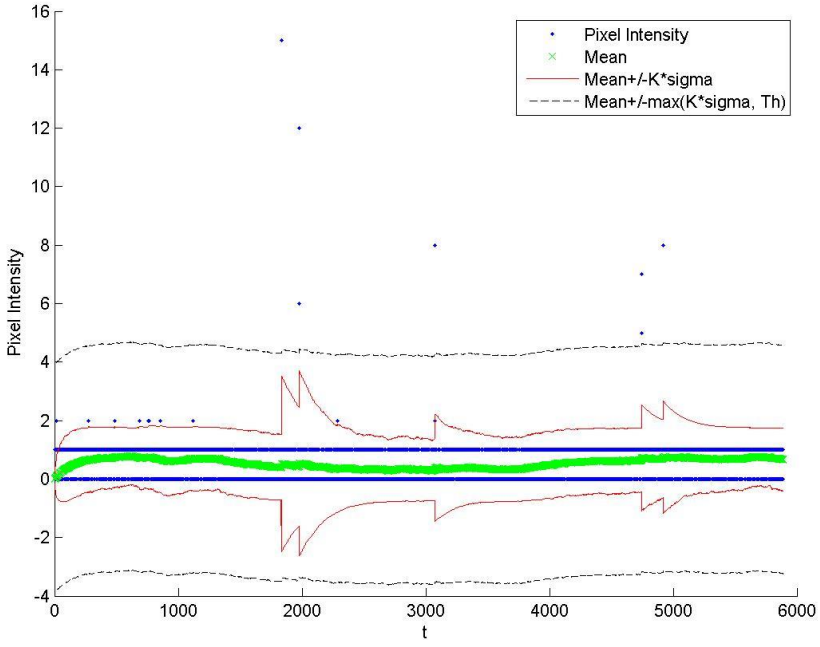

Figure 21 -- Pixel Intensity, Mean, and Enhanced Classification Boundaries at [98, 14]

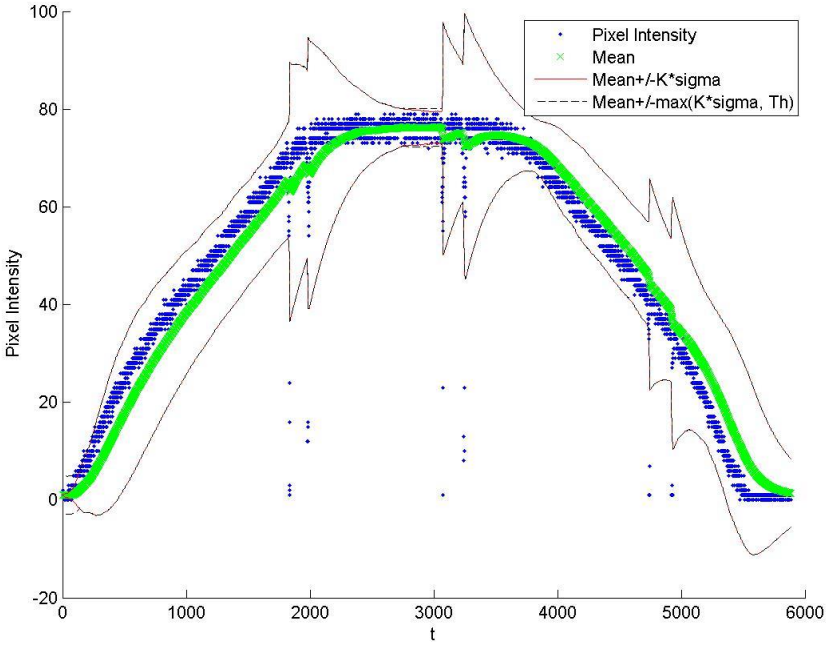

Figure 22 -- Pixel Intensity, Mean, and Enhanced Classification Boundaries at [35, 1]

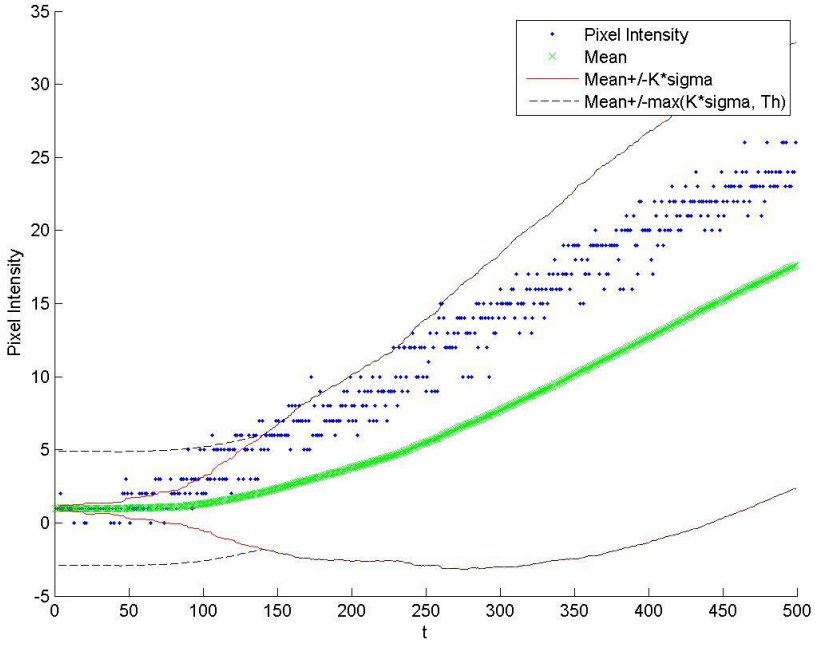

Figure 23 – Details of Start-up Period of Figure 22

Figure 24, Figure 25, and Figure 26 show the ground truth, the optimized SG segmentation, and the optimized enhanced SG segmentation of the targeted frame of each data set. The segmentation results from our proposed algorithm have better quality than that from the original SG algorithm in terms of less false positives.

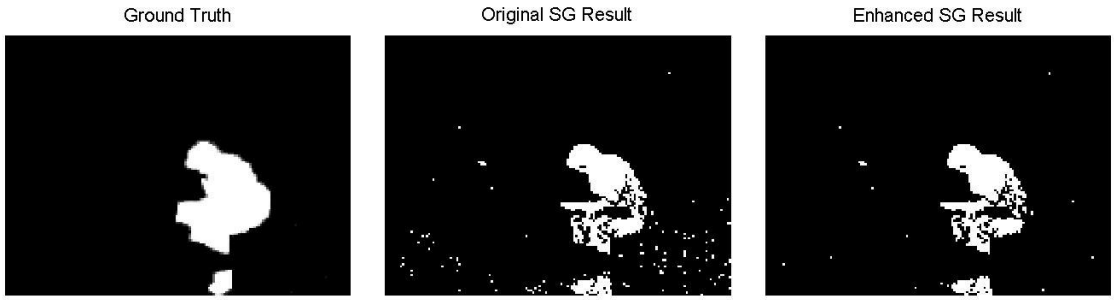

Figure 24 -- Truth Ground, Original, and Enhanced Segmentation of 'Time of Day' Data set

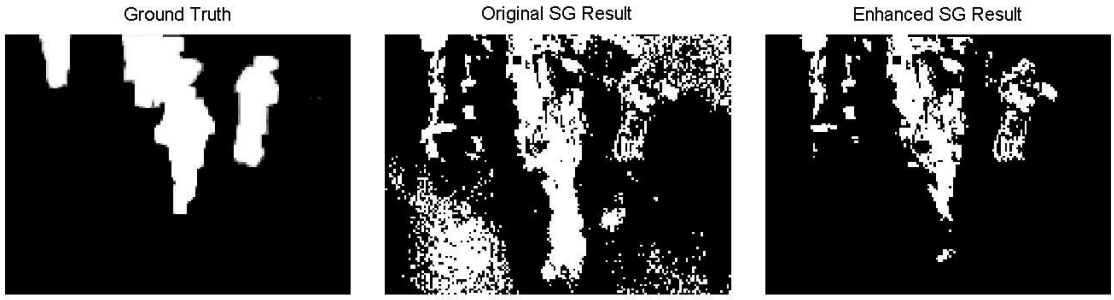

Figure 25 -- Truth Ground, Original, and Enhanced Segmentation of 'Bootstrap' Data set

Figure 26 -- Truth Ground, Original, and Enhanced Segmentation of 'Waving Trees' Data set

In summary, the proposed Enhanced SG algorithm efficiently reduces the number of false positives and increases the JC coefficient. More importantly, better quality of the segmentation results is perceived. So this enhanced SG algorithm was chosen to implement in the FPGA-base Real-Time MOD Design.

3.2.Pre-knowledge and System Features

3.2.1. Pre-mentioned Facts

In previous sections of this chapter, three solution related facts that are mentioned:

1. FPGA based platform is used for designing.
2. Enhanced SG background modelling algorithm is to be mapped in the design.
3. Off-Chip memory and multi-port memory controller are necessary in the design.

The decision of using the FPGA based platform was made in section 2.1.3. The second decision of using enhanced SG algorithm was made in section 3.1.3. In discussion of memory related issues in section 3.1.1, the necessity of off-chip memory and multi-port memory controller was concluded. Figure 4 and Figure 5 in section 3.1.1 show diagrams of off-chip memory and multi-port memory controller.

3.2.2. Transferring Video Data in Real-Time

This project is about real-time video processing, so the system is to interface with a real-time video source and to transfer real-time segmentation results. It becomes important to understand transferring video data in real-time.

Modern cameras tend to have high resolution so each frame usually has the number of pixels in the order of millions. Since it is impractical to transfer millions of data at once, the common strategy is to transfer the data in the manner of one pixel after another in the group of lines. Figure 27 shows the signal transmitting principle of image sensors (e.g., CCD or CMOS). Figure 28 shows a virtual timing example of transferring video data.

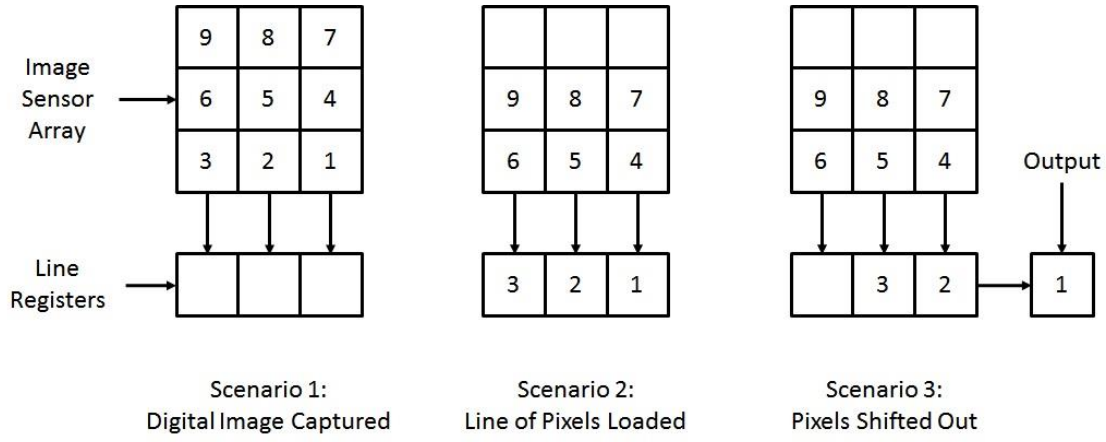

Figure 27 -- Transferring Data of a Frame (3-by-3)

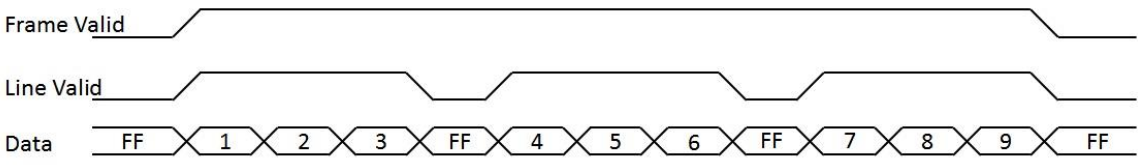

Figure 28 -- Virtual Timing Diagram of Transferring Video Data

The virtual timing example in Figure 28 indicates that the transferred video data is a data stream. In Figure 28, scenario 1 in Figure 27 happens between each high 'Frame

Valid', scenario 2 happens between the falling edge and rising edge of 'Line valid', scenario 3 happens during each high 'Line Valid' when 'Frame Valid' is also high.

The video data transferred to the display device is also data stream. The most common principle of displaying video data on the screen is: updating the frame on the screen from left to right pixel by pixel within each line, and from top to bottom line by line within each frame. Figure 29 shows the principle of displaying a frame on a screen.

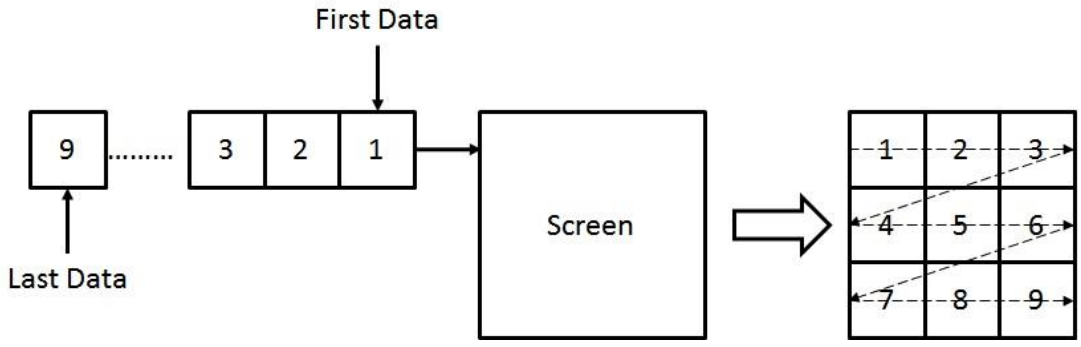

Figure 29 -- Displaying a Frame from Data Stream

The fact that video data is transferred in stream indicates that the input and output of the system are streams of pixel data.

3.2.3. Pipelining Structure

Section 3.1.1 denotes that a MPMC (multi-port memory controller) is needed and each port can provide a VFBC (video frame buffer connector) that has efficient 2-D data addressing functionality. Yet section 3.2.2 indicates that real-time video data is transferred as a data stream. These two facts are not contradictory. The former fact tells us that 2-D data (e.g., a frame of video) can be written or read efficiently by custom memory addressing logic (e.g., MPMC and VFBC). The latter fact explains how to transfer 2-D data via a data

bus in general. In summary, each port of the MPMC is used to address 2-D data via a VFBC; and at each port the 2-D data is transferred in a data steam.

The adopted enhanced SG background modeling algorithm is pixel-based. It requires that the system perform the same model estimation and classification operations on every pixel. The model estimation and classification operations can be further decomposed to a sequence of arithmetic operations. As mentioned in section 3.1.1, multiple inputs are required for performing the estimation and classification operations on any single pixel and multiple outputs are generated. In summary, the logic to perform the enhanced SG background modeling algorithm shall have multiple input ports and multiple output ports; and in essence it performs a sequence of arithmetic operations. If the ports of the background modeling logic interact with the MPMC ports directly, the logic shall have multiple incoming data streams and outgoing data streams and perform a sequence of arithmetic operations in between.

It is widely recognized that the pipeline structure increases the throughput of the system when it performs a sequence of operations on data streams. Since the goal is a real-time MOD solution that deals with high volume data throughput, the pipeline structure is surely favoured in designing the background modeling logic or any other needed logic. Pipelines used in the proposed design are introduced in section 3.3.2.

3.2.4. System Feature Extraction

In above introduction, five solution related facts are discussed as listed:

1. FPGA-based platform is used for designing.
2. Enhanced SG background modelling algorithm is used in the solution.

3. Off-Chip memory and multi-port memory controller are needed in the design.
4. Real-time video data is transferred in data streams.
5. Pipeline structure is favoured in the design.

Based on these facts and their discussion, the system features are extracted as below:

1. Three major groups of off-chip devices are required by the system:
   a) off-chip memory
   b) video source devices
   c) display devices
2. Three major function blocks are to be designed on the FPGA:
   a) Video Pre-Processing
   b) Enhanced SG Background Modeling
   c) Displaying Preparation
3. An MPMC core is also to be implemented on the FPGA to communicate with the off-chip memory and on-chip function blocks

3.3.Proposed Design

3.3.1. System Level Descriptions

Real-time MOD system design is introduced in this section as the solution to our problem stated in section 1.3. The design is described from three overlapped aspects. Together they embody the system features extracted in section 3.2.4. These three aspects are:

1. System Work Flow
2. System Data Flow

3. System Structure

System Work Flow

The work flow of the design contains three major functions between video source and display device. They are namely video preprocessing, enhanced SG background modeling, and display preparation. Figure 30 shows the system work flow diagram.

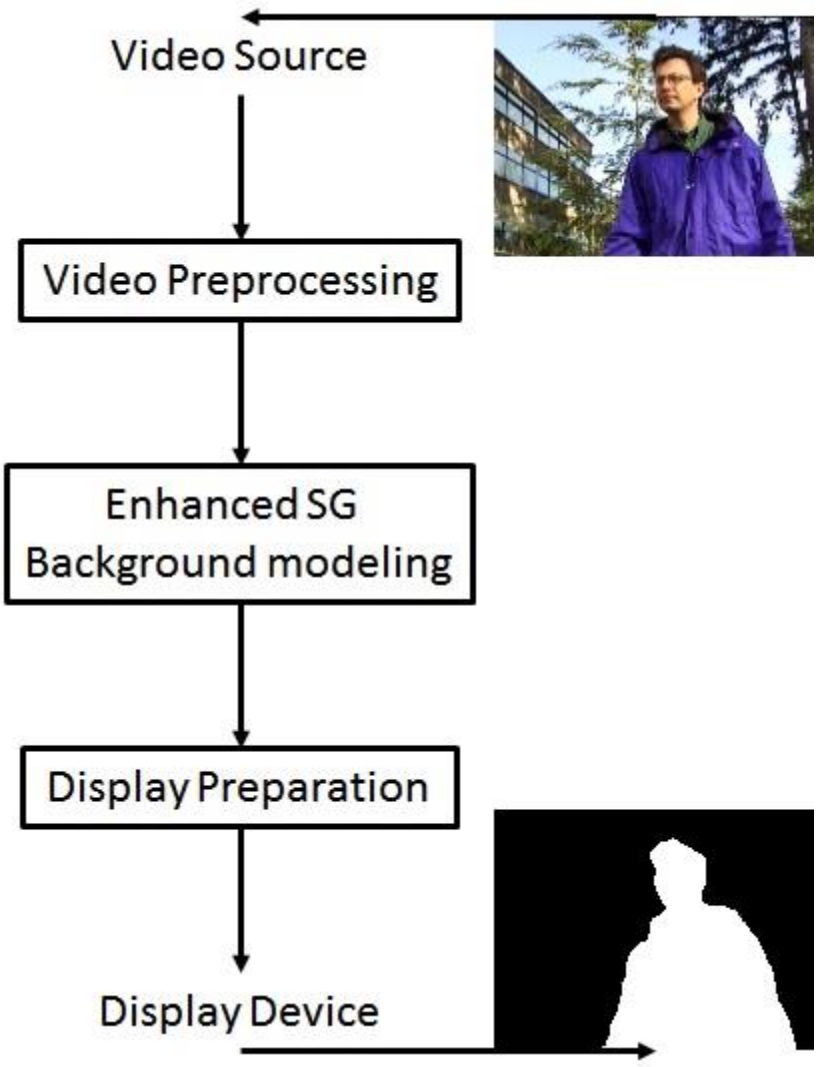

Figure 30 -- System Work Flow Diagram

The video preprocessing block is mainly to prepare the raw image data into the form that can be further processed by the following block. It includes a series of image processing routines (e.g., Bayer conversion, gamma correction, etc.). On the other end, the display preparation block prepares the processed data into the form that meets the requirement of the display devices. In between is the background modeling block that preforms the enhanced SG algorithm.

System Data Flow

The system data flow describes how data is transferred between the functional blocks. Figure 31 shows the system data flow diagram. The utility of the multi-port memory is visualized in this diagram. The memory controller plays a key role in interconnecting functional blocks. The functional blocks have a pipeline structure as discussed in section 3.2.3. More details of the pipelines are introduced in section 3.3.2.

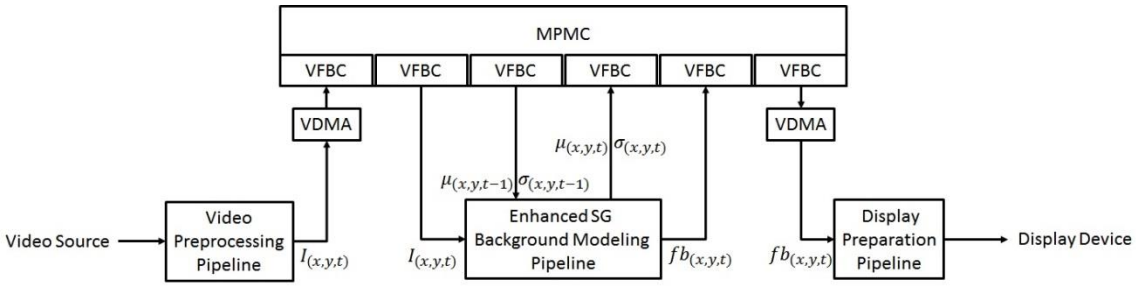

Figure 31 -- System Data Flow Diagram

System Structure

The system structure describes the overall connection between all the major components of the real-time MOD design, including the FPGA functional logics and off-chip devices. Figure 32 shows the diagram of the system structure.

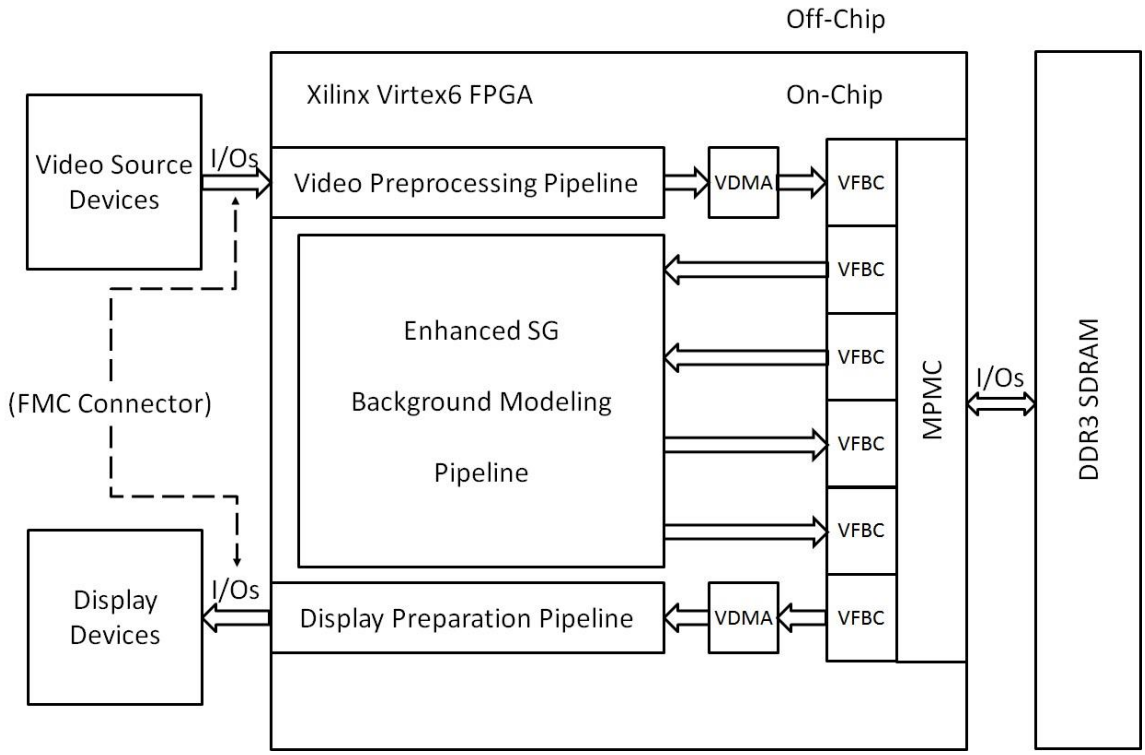

Figure 32 -- System Structure Diagram

Note, as mentioned in section 2.1.3, the ML605 FPGA evaluation board was assigned to use in developing the solution. Considering the connectivity of the given board, an Avnet Dual Image Sensor FMC Module [48] was used to bridge the video source devices and display devices to the Xilinx Virtex6 FPGA via the FMC connector on the ML605 board. FMC (FPGA Mezzanine Card) is an ANSI/VITA standard that defines I/O mezzanine modules with connection to an FPGA or other device with reconfigurable I/O capability [49]. The video source devices mainly include an OmniVision 1 MP camera kit with ribbon Cable [50]. The display devices mainly include a TFP410 [51] on the FMC board and a monitor that has DVI signal adapter.

3.3.2. Functional Blocks Descriptions

The proposed design carries out all the necessary functions between a camera's output and display device's input. Three major function blocks with pipeline structure were included in the design. This section gives an introduction of these three functional blocks with more detail.

Video Preprocessing Pipeline

Figure 33 shows a diagram of the logic blocks related to video preprocessing pipeline.

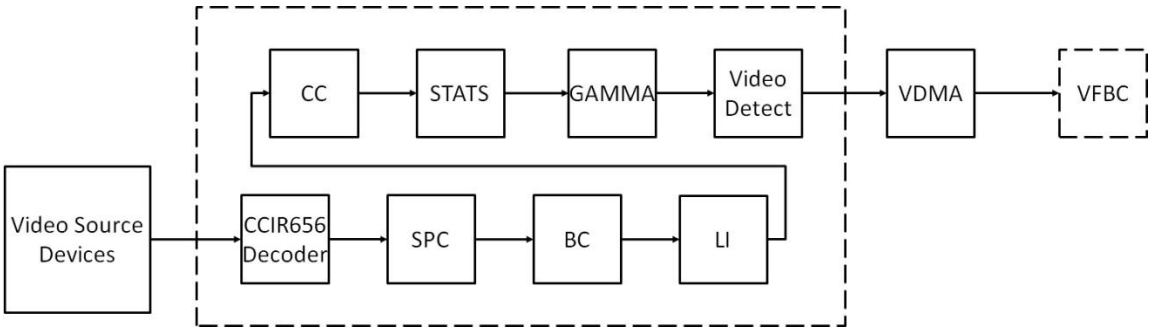

Figure 33 -- Video Preprocessing Pipeline Diagram

The main functions of each logic block in the video preprocessing pipeline are listed in Table 6. These logic blocks are legacy IP cores of a reference design [52] provided by Xilinx, Inc.

Table 6 -- Video Preprocessing Logic Blocks

Logic Block Name	Main Function
CCIR656 Decoder	Decode the CCIR656 codes embedded with the data conveyed by CAM_DATA to generate VBLAK/HBLANK/ACTIVE_DATA
Stuck Pixel Correction (SPC)	Adaptive median filter of pixels with a configurable threshold
Brightness and Contrast Control (BC)	Brightness: Global offset control by way of adder Contrast: Global digital gain stage using a multiplier
Linear Interpolation (LI)	Linear interpolation of RGB color components from Bayer pattern color filter array (CFA)
Color Balance Control (CC)	Individual color gains for red, green, and blue components
Image Statistics (STATS)	Calculates global maximums and minimums for each color component
Gamma Correction (GAMMA)	Gamma Correction implemented with a look up table (LUT)
Video Detect	To detect the resolution of the input video data

Within the video preprocessing pipeline, there are two major format changes of video stream signals. The first change is that the camera output signals become Xilinx Video Stream Interface (XVSI) signals in the CCIR656 Decoder logic block. Figure 34 shows the details of this change.

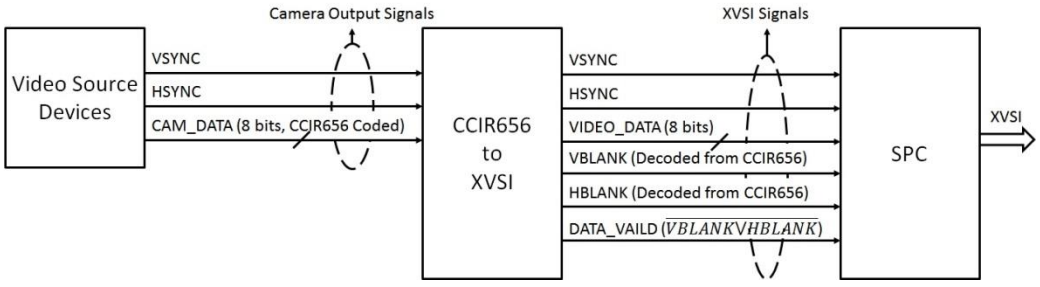

Figure 34 -- First Major Format Change in Video Preprocessing Pipeline

CCIR656 is a simple protocol for streaming uncompressed signals. The camera of the video source devices is configured to work in CCIR656 mode to embed CCIR656 synchronization codes into the video data stream. In the CCIR656 mode, the end of the last line/frame is indicated by the 4-byte long EAV (end of active video) code and the start of the new line/frame is indicated by the 4-byte long SAV (start of active video) code. By decoding the CCIR656 synchronization codes, the timing of HBLAK/VBLANK are determined since these two signals should be active high between active lines/frames. As to the timing of DATA_VALID, it is the result from "NOR" logic of VBLANK and HBLANK ($\overline{HBLANK \wedge VBLANK}$).

The V/H-SYNC, V/H-BLANK, DATA_VALID, and VIDEO_DATA signals are all included in the XVSI. Figure 35 shows XVSI signals' timing diagram. For more details of XVSI signals please refer to [52, p. 59].

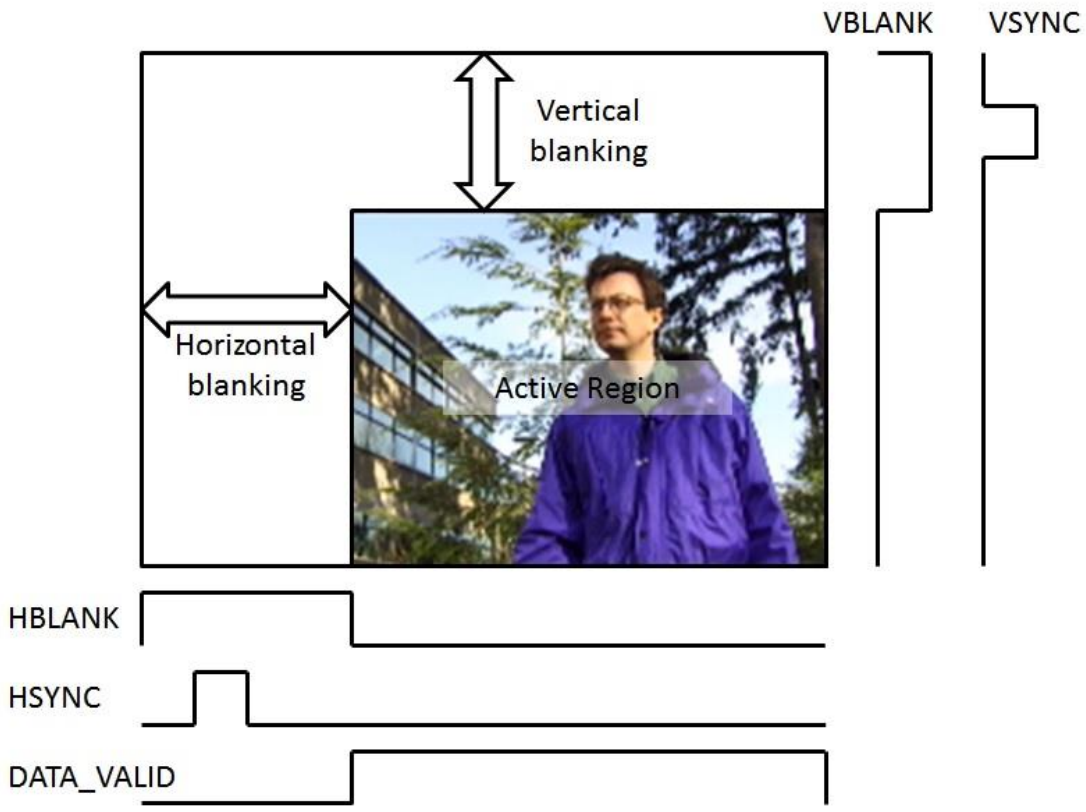

Figure 35 -- XVSI Signals' Timing Diagram

The second change is that the VIDEO_DATA turns to 24 bits from 8 bits in the LI logic block. Figure 36 shows this change. The new 24 bits data bus conveys the 24 bits RGB color pixel data that is interpolated from pixel data of the raw output from the Bayer-filter camera. A Bayer filter mosaic is a color filter array (CFA) for arranging RGB color filters on a square grid of photosensors. The raw output of Bayer-filter cameras is referred to as a Bayer pattern image. Since each pixel in this raw output is filtered to record only one of three colors, the data from each pixel cannot fully specify each of the red, green, and blue values on its own, represented by 8 bits in this case. To obtain a full-color image, demosaicing algorithms can be used to interpolate a set of complete red, green, and blue values for each pixel, represented by 24 bits in this case.

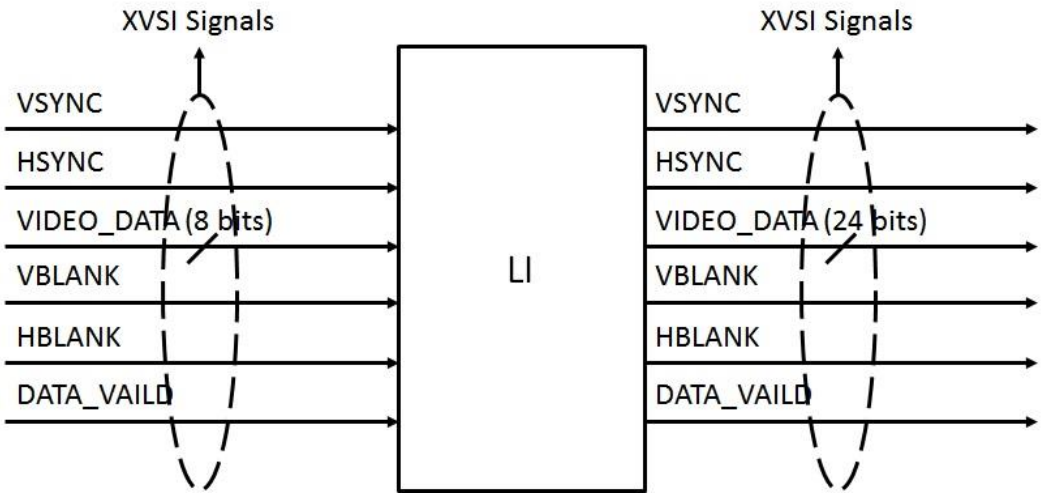

Figure 36 -- Second Major Format Change in Video Preprocessing Pipeline

The output signals of the pipeline go to the VDMA (shown in Figure 33). "VDMA" stands for video direct memory access. This type of logic blocks bridges XVSI signals to VFBC (writing video data to memory) or the opposite way (reading video data from memory, discussed in section Display Preparation Pipeline). The major function of VDMA is to issue the command packet required by the VFBC. The command packet provides the information of the video resolution and the data stream direction (write/read). For the VDMA in Figure 33, the data stream direction is from VDMA to VFBC. Once the VFBC has received the command packet at the beginning of each frame, it becomes ready to transfer the valid incoming data to the off-chip memory via the MPMC. Valid data is indicated by the DATA_VALID signal.

Enhanced SG Background Modeling Pipeline

Figure 37 shows the logic blocks and their signal connections in the enhanced SG background modelling pipeline.

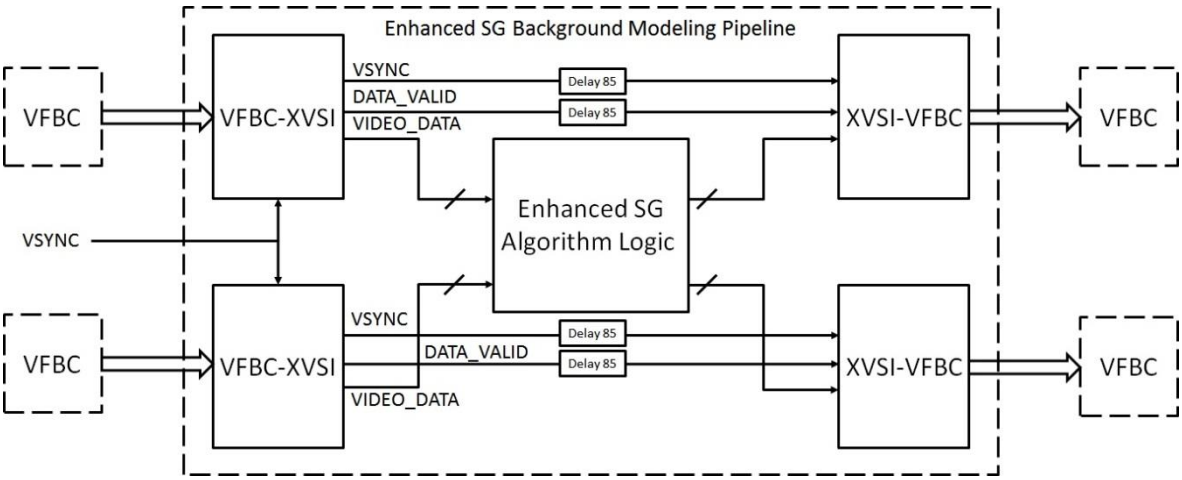

Figure 37 -- Enhanced SG Background Modeling Pipeline

The "VFBC-XVSI" logic blocks require a frame of video data from the VFBC port when a pulse on VSYNC happens and reform the signals into the XVSI format. Note since HSYNC, VBLANK, and HBLANK are not functional in other logics, they are omitted in the figure. "XVSI-VFBC" blocks on the other hand, start to write a frame of data after a pulse of VSYNC and reformat the XVSI signals into VFBC acceptable signals. The "Enhanced SG Algorithm Logic" performs the background modelling with the input data streams and outputs the streams of model estimation result and classification result. Figure 38 shows the diagram of the inside logics of the Enhanced SG Algorithm Logic block.

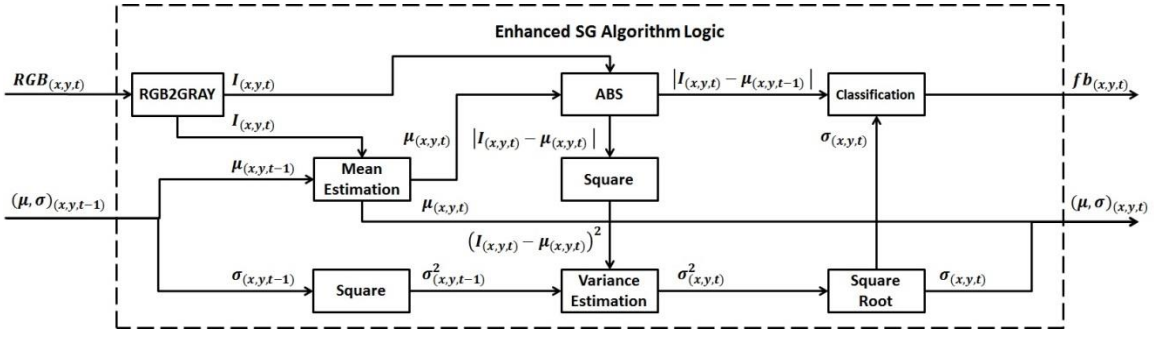

Figure 38 -- Enhanced SG Algorithm Logic

Note that certain delays can be introduced by each operation within the algorithm logic and some delay units are inserted on the data path for synchronization purpose. The total delay from input data streams to output data streams is 85 clock cycles at 40MHz. Details of the delay within the algorithm logic are omitted for simplicity. Some key logic blocks are introduced by their function:

RGB2GRAY

RGB2GRAY performs the function of converting RGB color pixel ($RGB_{(x,y,t)}$) data into gray level intensity ($I_{(x,y,t)}$). The formula of color-to-gray conversion is:

$$I_{(x,y,t)} = 0.2989 \cdot R_{(x,y,t)} + 0.5870 \cdot G_{(x,y,t)} + 0.1140 \cdot B_{(x,y,t)}$$

Mean Estimation

The function is to estimate the mean ($\mu_{(x,y,t)}$) of the Gaussian model of the input pixel. The formula of mean estimation is:

$$\mu_{(x,y,t)} = a \cdot \mu_{(x,y,t-1)} + (1 - a) \cdot I_{(x,y,t)}$$

Variance Estimation

The function is to estimate the variance of the Gaussian model of the input pixel. The formula of the variance estimation is:

$$\sigma^2_{(x,y,t)} = a \cdot \sigma^2_{(x,y,t-1)} + (1 - a) \cdot \left(I_{(x,y,t)} - \mu_{(x,y,t)}\right)^2$$

Classification

The function of this block is to classify the pixel either as foreground or background, the formula is:

$$fb_{(x,y,t)} = \begin{cases} 0, background, if\ |I_{(x,y,t)} - \mu_{(x,y,t)}| \leq max(Th, K \cdot \sqrt{\sigma^2_{(x,y,t)}}) \\ 1, foreground, if\ |I_{(x,y,t)} - \mu_{(x,y,t)}| > max(Th, K \cdot \sqrt{\sigma^2_{(x,y,t)}}) \end{cases}$$

The above logic blocks with others together perform the complete function of the Enhanced SG algorithm. Note that it is the square root of the variance, the standard deviation that is stored in the off-chip memory. This is because, with limited memory bandwidth, this manner can extend the precision of the variance.

Display Preparation Pipeline

Figure 39 shows a diagram of logic blocks in the display preparation pipeline.

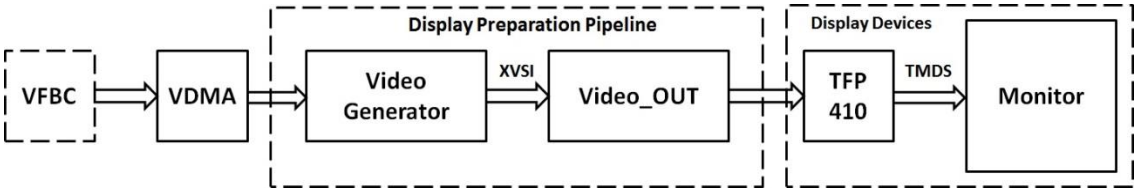

Figure 39 -- Display Preparation Pipeline

The “Video Generator” logic block transfers video data via XVSI to the "Video_OUT" block. Signals from the VDMA block contain video data and data valid signals but have no vertical and horizontal synchronization or blanking signals. So the main function of “Video Generator” is to reconstruct some of these missing signals that are required by the “Video_OUT” logic block. The “Video_OUT” logic block further reforms the incoming signals into TFP410 acceptable signals. TFP410 is a semiconductor product of Texas Instrument. TFP410 converts the "universal input" signals to TMDS signals [51]. TMDS (Transition-minimized differential signaling) is a technology for transmitting high-speed

serial data and is used by the DVI and HDMI video interfaces [53]. For more detail of "Video Generator" and "Video_OUT" logic blocks, please refer to [52].

IV. Implementation and Evaluation

4.1.Real-Time MOD Design Implementation

4.1.1. Circuit Implementation with EDA tools

The “on-chip” part of the design, or the circuit design, was implemented with Xilinx EDA tools and downloaded to the FPGA on the ML605 evaluation board.

In the proposed circuit, each logic block in both the video preprocessing pipeline and display preparation pipeline was implemented by creating an IP core. The enhanced SG background modeling pipeline was implemented by creating a single IP core. In addition, there were other IP cores like the MPMC provided by the design tool. There are four major types of IP cores involved in the proposed design:

1. System Generator IP core
2. BPS (BEECube Platform Studio) IP core
3. Custom EDK IP core
4. Generic EDK IP core

System Generator IP cores are designed using the Xilinx System Generator (XSG) software for this project for the Xilinx Virtex6 FPGA. This Xilinx tool allows the user to build a digital signal processing system using Simulink with the Xilinx Blockset. Each generated IP core is represented by a netlist (NGC file) and is imported to Xilinx Platform Studio (XPS). BSP cores are similar to the System Generator IP cores. BSP also allows users to build system in Simulink with both the Xilinx Blockset and the BSP Blockset. Its own block set is relatively at a higher level. For more information please refer to [54]. The

generated IP core is represented by a netlist as well. Custom EDK IP cores are created in XPS using an HDL template. In this design such IP cores were defined in VHDL. Generic EDK IP cores (e.g., MPMC) are provided with Xilinx tools. A Classification of IP cores introduced in section 3.3 is listed in Table 7.

Table 7 -- IP Cores' Classification in the Design

IP Core Types	IP Cores in Proposed Design
System Generator	Stuck Pixel Correction (SPC), Brightness and Contrast Control (BC), Linear Interpolation (LI), Color Balance Control (CC), Image Statistics (STATS), Gamma Correction (GAMMA)
BSP	Enhanced SG Background Modelling Pipeline
Custom EDK	CCIR656 Decoder, Video Detect, Video Generator, Video_OUT
Generic EDK	MPMC

All the IP cores involved in the design were assembled in Xilinx Platform Studio (XPS). The rest of the procedures follow the road map of FPGA programing introduced in section 2.1.2.

4.1.2. Hardware Connection

The “off-chip” components were carried by multiple devices. Other than the Omnivision camera and the monitor, there is an ML605 board and an FMC board. Figure 40 illustrates the connection between hardware devices.

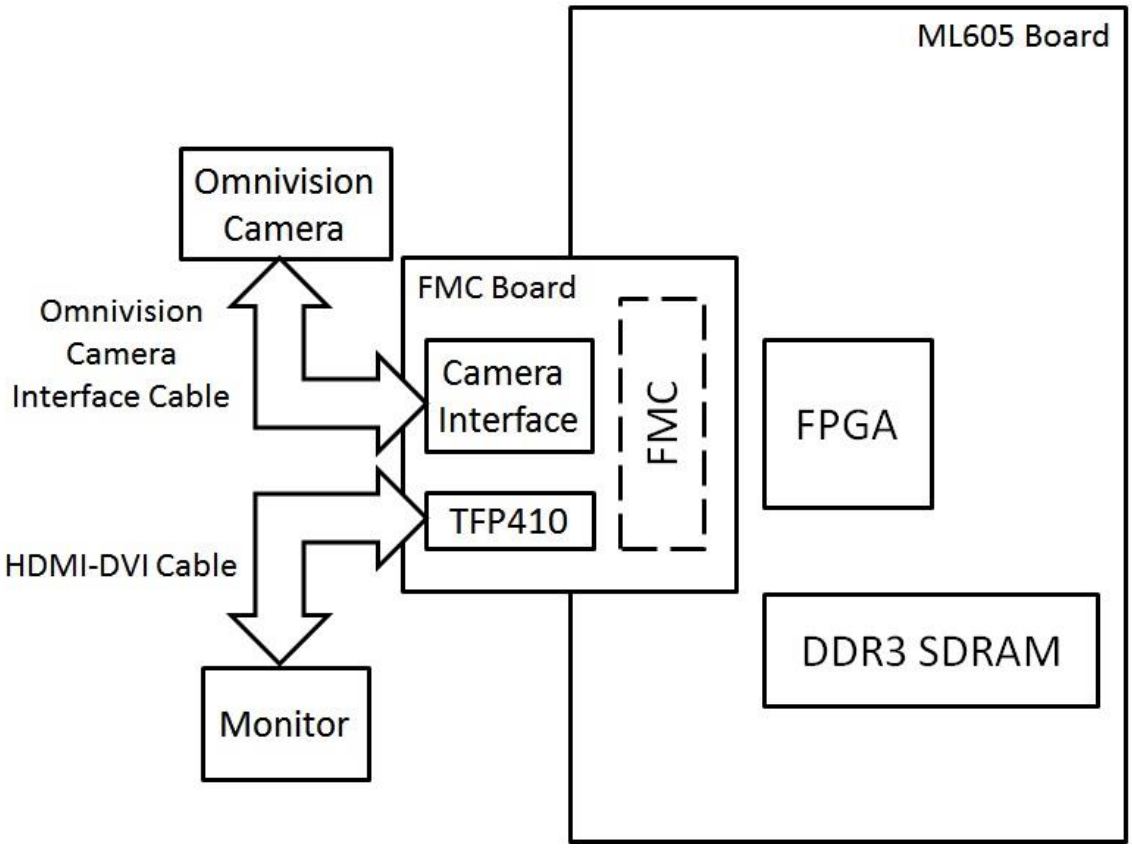

Figure 40 -- Hardware Connection Diagram

4.2.Experiment Result

4.2.1. Real-Time Performance

In the implemented design, the Omnivision camera was configured to work in 1280×720p@30fps. In this resolution and frame rate, the camera works with the clock frequency at 40 MHz. The video preprocessing pipeline and enhanced SG background modeling pipeline synchronize with the camera working frequency. For the display preparation pipeline, the working frequency is 72.5 MHz, because the monitor works in 1280×720p@60fps. The memory bandwidth required by the enhanced SG background

modeling pipeline is 0.64 GB/s (0.64 GB/s = 32 bit × 4 ×40 MHz). The total bandwidth requirement is 1.29 GB/s (1.29 GB/s = 32bit ×5×40 MHz+32bit×1×72.5 MHz).

Referring to the real-time definition in section 1.1, the real-time MOD system achieved real-time moving object detection for a HD (1280×720p) video of 30 fps.

4.2.2. Detection Quality

Figure 41 and Figure 42 show the quality of the foreground detection of the design implemented on the FPGA. As we can observe from the sample pictures, the detection result is very clear with very little noise and very high resolution.

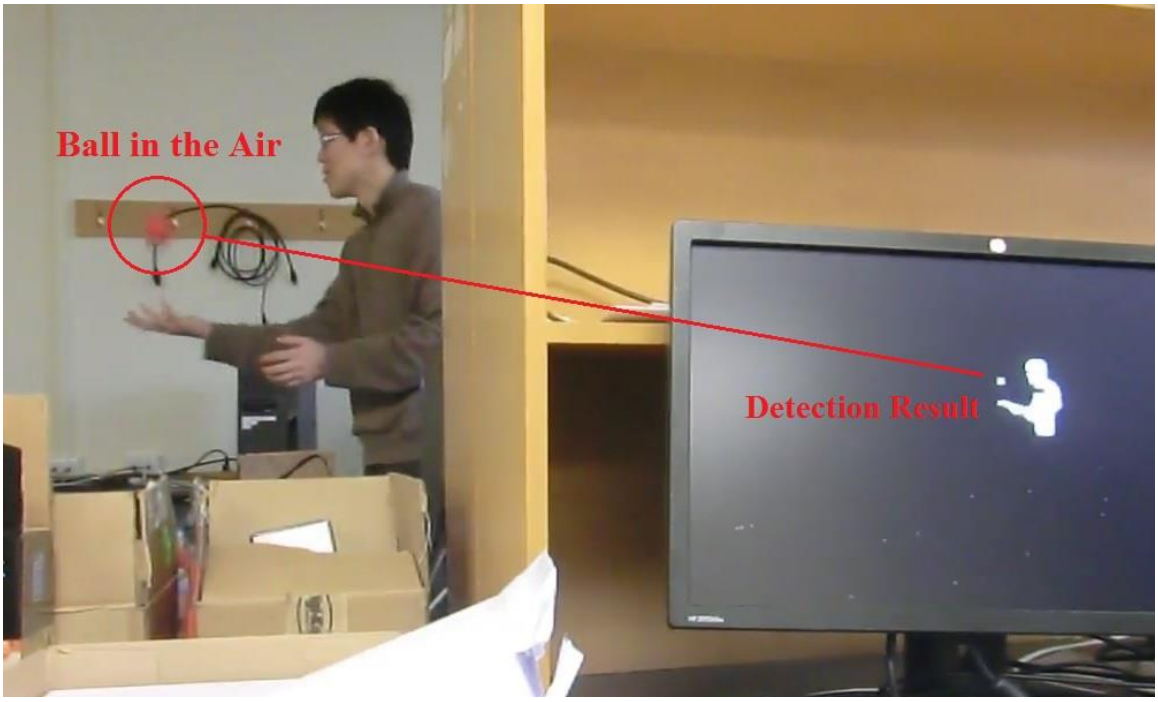

Figure 41 -- Field Test: Detection of a Moving Ball

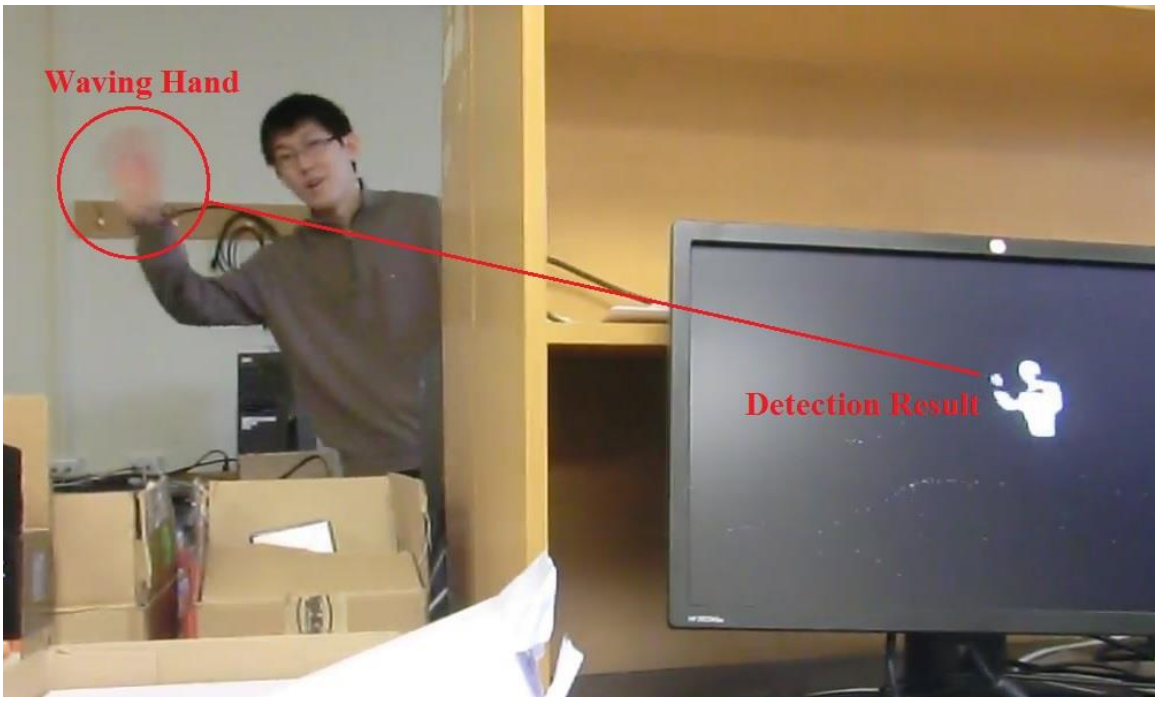

Figure 42 -- Field Test: Detection of a Waving Hand

4.3.Comparison with Other Works

4.3.1. Comparison with other Software Implementation Work

In the end of section 1.1, several examples of software implementation of background modeling algorithms are provided. Their performance suggests that the conventional PC platform is not suitable for real-time moving object detection. In this section, the efficiency of the proposed FPGA implementation is compared indirectly to that of the software examples. Since the video resolution, video frame rate, and specific background modeling algorithm vary from case to case, the indicator of 'second per pixel' was used to unify the performance measurement of these examples. Table 8 shows the indirect comparison mentioned above. The proposed FPGA implementation has a much lower value of second

per pixel which indicates a much higher efficiency performance than the software implementations.

Table 8 -- Indirect Efficiency Comparison between Software and FPGA Implementation

Case Name	Implementation Purpose	Efficincy Performance (Second per Pixel)
Staffer and Grimson [12]	MOG implementation with software	434.03×10^{-8}
Kristensen et al. [14]	MOG validation with software for hadware implemenation	98.64×10^{-8}
Roshan and Zhang [15]	Algorithm comparison with software (Only the background modeling algorithm is considered in this table.)	came1: 9816.67×10^{-8} cam2: 8584.87×10^{-8}
Proposed FPGA Implementation	Enhanced SG hadware implemenation	3.62×10^{-8}

4.3.2. Comparison with other FPGA Implementation Work

Jiang al. [30] designed an FPGA architecture that adopted the MOG algorithm proposed by Staffer and Grimson [12] and implemented it in a latter work [14] where real-time MOD of 320x240@25fps video is achieved. With the effort of reducing the memory bandwidth consumption, a higher resolution was achieved in [30]. The achieved real-time performance was 640x480@25fps. However, the memory bandwidth consumption reduction "came at the cost of segmentation quality", quoted from [55].

Genovese al. [32] adopted the optimized version of MOG from OpenCV (Open source Computer Vision) software library and implemented it on an FPGA. However, no segmentation result was shown in this initial work but the maximum frequency that background modeling logic can work at is given, as 47 MHz. With the effort of optimizing

the circuit logic in [56], the maximum frequency went up to 50.5 MHz without the pipeline structure and 90.0 MHz with the pipeline structure [33]. This number continued to go up in their latest work [34] where the highest frequency achieved was 189.3 MHz. These numbers are equal to the potential pixel rates that the background modeling logic can handle. However, the best achieved experimental result is 1280x720p@20fps on a Virtex4 FPGA device [34].

Kryjak al. [35] implemented a customized background modeling algorithm and achieved a real-time performance of 640x480@60fps. The system worked at 25 MHz and the maximum frequency working frequency was 119MHz.

To compare real-time performance of the proposed solution with the above works, three indicators were used, namely best real-time performance in implementation, its working frequency, and the maximum working frequency allowed. These three indicators of real-time performance are commonly used among similar works.Table 9 compares these three indicators of the proposed solution to the others.

Table 9 -- Real-Time Performance Comparison with Other FPGA Implementations

Name	Best Real-Time Performance of Implementation	Working Frequency	Maximum Working Frequency
Jiang al.	640x480p@25fps (Xilinx VirtexII vp 30)	16 MHz	83 MHz
Genovese al.	1280x720p@20fps (Virtex4 xc4vfx12)	N/A	189.3 MHz (Virtex6 xc6vlx75t)
Kryjak al.	640x480p@60fps (Spartan6 XC6SLX45T)	25 MHz	119 MHz
Proposed	1280x720p@30fps (Virtex-6 XC6VLX240T)	40 MHz	200 MHz

The comparison showed that the proposed solution outperformed other solutions in terms of resolution, working frequency, and maximum frequency allowed.

V. Conclusion and Future Work

5.1.Achievements

In this thesis, a real-time MOD system was designed (see section 3.3) and implemented (see section 4.1) on an FPGA-based platform. The system employs the enhanced SG background modeling algorithm proposed in section 3.1. This algorithm outperforms the original SG background modeling algorithm in terms of reducing false positives. The implementation of the proposed design is able to perform real-time MOD in a video of 1280×720p@30fps. The segmentation results of the experiment clearly showed the areas of moving object(s) in the video with very little noise (see Figure 41 and Figure 42 in section 4.2).

In summary the proposed system is able to perform real-time MOD in high resolution video with decent segmentation quality. Thus makes it a viable solution to the efficiency bottle neck of real-time automated video analysis system.

5.2.Improvements

Comparing to previous works (shown in Table 9), the proposed solution made improvements in resolution, frame rate, working frequency (increased by 60%), and maximum working frequency allowed (increased by 6.2%). The improvements are coherent with the figures listed in Table 9.

5.3.Future Work

Certain compromises were made in the trade-off between data precision and algorithm sophistication. The SG algorithm was chosen to maintain the data precision so that the background modeling could be performed normally. This trade-off comes from the limitations of the number of bits in the data bus of a memory port and the number of these memory ports of the memory controller.

A new memory controller could be used to address this issue. An example is an AXI-based system which is able to handle 24 video streams of 1290x1080p@60fps by using AXI-mpmc, AXI-VDMA, and other AXI assisting IP cores [57]. These numbers are very encouraging simply by considering what has been achieved with 5 video streams of 1280x720p@30fps and 1 video stream of 1280×720@60fps. If more video streams are able to be handled, a more sophisticated background modeling algorithm could be implemented in the future design.

To adopt AXI-MPMC and replace the original MPMC, the entire bus system needs to be replaced with AXI interfaces and AXI interconnections. AXI stands for Advanced eXtensible Interface. AXI is a part of AMBA, Advanced Microcontroller Bus Architecture. It is an interface specification rather than a bus specification. There are three types of AXI interface supported by Xilinx Virtex6 FPGA so far, namely AXI-4 Full, AXI-4 Streaming, and AXI-4 Lite. The AXI interfaces are connected by AXI interconnection IP cores.

One major concern of this replacement is whether the legacy IP cores are still reusable in the future design. So far, several documents have been found that might be related to this issue. The AXI reference guide [58] introduces how to bridge PLB (Processor Local

Bus) IP cores to an AXI interconnection. The Xilinx application note [59] introduces a way to bridge XVSI with AXI-4 Stream protocol and vice versa.

However, there are still other questions left to be answered. For example, in implementing background modelling algorithm logic in BEECube, is the AXI-4 Stream supported in BEECube 4.0? While the circuit design is changed, how much work needs to be done with software programming? To answer these questions could be a starting point of the future work.

Bibliography

[1] M. D. Schulz and S. Gilbert, "TCRP Synthesis 90: Video Surveillance Uses by Rail Transit Agencies," The National Academies Press, Washington, D.C., 2011.

[2] J. Hogan, "Smart software linked to CCTV can spot dubious behaviour," *New Scientist,* p. 4, 12 July 2003.

[3] A. Yilmaz, O. Javed and M. Shah, "Object tracking: A survey," *ACM Comput. Surv.,* vol. 38, no. 4, Dec 2006.

[4] K. Dong, M. Hu, Z. Ji and B. Fang, "Research on Architectures for High Performance Image Processing," in *Proceedings of the Fourth International Workshop on Advanced Parallel Processing Technologies*, 2001.

[5] A. C. Downton and D. Crookes, "Parallel Architectures for Image Processing," *Electronic & Communication Engineering Journal,* vol. 10, no. 3, pp. 139-151, June 1998.

[6] G. L. Foresti, C. Micheloni, L. Snidaro, P. Remagnino and T. Ellis, "Active Video-Based Surveillance System: The low-level image and video processing techniques needed for implementation," *Signal Processing Magazine, IEEE,* vol. 22, pp. 25-37, Mar 2005.

[7] C. Soviany, "Embedding Data and Task Parallelism in Image Processing Applications," *Ph.D. Dissertation of Delft University of Technology,* 2003.

[8] N. Kehtarnavaz and M. Gamadia, Real-Time Image and Video Processing: From Research to Reality, 1 ed., A. C. Bovik, Ed., Morgan & Claypool, 2006.

[9] K. Toyoma, J. Krumm, B. Brummitt and B. Meyers, "Wallflower: Principles and Practice of Background Maintenance," *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on,* vol. 1, pp. 255-261, 1999.

[10] S.-C. S. Cheung and C. Kamath, "Robust techniques for background subtraction in urban traffic video," *Proc. SPIE 5308, Visual Communications and Image Processing 2004,* vol. 5308, pp. 881-892, 7 Jan 2004.

[11] C. Wren, A. Azarbayejani, T. Darrell and A. Pentland, "Pfinder:Real-Time Tracking of the Human Body," *Pattern Analysis Machine Intelligence,* vol. 19, no. 7, pp. 780-785, Jul 1997.

[12] C. Stauffer and W. Grimson, "Adaptive background mixture models for real-time tracking," *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.,* vol. 2, pp. 2246-2252, 1999.

[13] A. Elgammal, D. Harwood and L. Davis, "Non-parametric model for background subtraction," *Computer Vision — ECCV 2000,* vol. 1843, pp. 751-767, 2000.

[14] F. Kristensen, H. Hedberg, H. Jiang, P. Nilsson and V. Öwall, "Surveillance System: Implementation and Evaluation," *Journal of Signal Processing Systems,* vol. 52, no. 1, pp. 75-94, 1 Jul 2008.

[15] A. Roshan and Y. Zhang, "A comparison of moving object detection methods for real-time moving object detection," *Proc. SPIE 9076, Airborne Intelligence, Surveillance, Reconnaissance (ISR) Systems and Applications XI,* vol. 9076, no. 09, 9 Jun 2014.

[16] N. Kehtarnavaz, Real-Time Digital Signal Processing: Based on the TMS320C6000, 1 ed., Newnes, 2004.

[17] Wikimedia Foundation, Inc., "Digital signal processor," Wikimedia Foundation, Inc., [Online]. Available: http://en.wikipedia.org/wiki/Digital_signal_processor. [Accessed 27 Jun 2014].

[18] Texas Instruments, Inc., "DaVinci Video Processors," Texas Instruments, Inc., [Online].
Available:http://www.ti.com/lsds/ti/dsp/video_processors/overview.page.
[Accessed 17 6 2014].

[19] V. I. Ponomaryov, F. J. Gallegos-Funes, O. B. Pogrebnyak and L. Nino-de-Rivera, "Real-time image filtering with retention of small-size details and complex noise mixture," *Proceedings of SPIE-IS&T Electronic Imaging Conference on Real-Time Imaging,* vol. 4666, pp. 30-41, 4 March 2002.

[20] V. I. Ponomaryov, F. J. Gallegos-Funes and L. Nino-de-Rivera, "Real-time processing scheme based on RM estimators," *Proceedings of SPIE-IS&T Electronic Imaging Conference on Real-Time Imaging,* vol. 5012, pp. 37-48, 14 April 2003.

[21] V. I. Ponomaryov, A. J. Rosales and F. J. Gallegos-Funes, "Real-time color imaging using the vectorial order statistics filters," *Proceedings of SPIE-IS&T Electronic Imaging Conference on Real-Time Imaging,* vol. 5297, pp. 35-44, 18 May 2004.

[22] F. J. Gallegos-Funes and V. I. Ponomaryov, "Real-time image filtering scheme based on robust estimators in presence of impulsive noise," *Journal of Real-Time Imaging,* vol. 10, no. 2, pp. 69-80, April 2004.

[23] V. I. Ponomaryov, R. Sansores-Pech and F. J. Gallegos-Funes, "Real-time 3D ultrasound imaging," *Proceedings of SPIE-IS&T Electronic Imaging Conference on Real-Time Imaging,* vol. 5671, pp. 19-29, 17 Mar 2005.

[24] A. Iketani, Y. Kuno, N. Shimada and Y. Shirai, "Real-time Surveillance System Detecting Persons in Complex Scenes," *Journal of Real-Time Imaging,* vol. 7, no. 5, pp. 433-446, October 2001.

[25] G. Li, L. Ma, Y. Liu and W. Liu, "Moving object detection based on Mixture of Gaussian fusing spatial-temporal information," *Image and Signal Processing (CISP), 2013 6th International Congress on,* vol. 1, pp. 154 - 159, Dec 2013.

[26] J. F. Wakerly, Digital Design: Principals and Practices, 4 ed., Englewood Cliffs, NJ: Prentice Hall, 2005.

[27] Xilinx, Inc., "Virtex-6 Family Overview," Jan 2012. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf. [Accessed 15 Jul 2014].

[28] J. Sivaswamy, Z. A. Salcic and K. L. Ling, "A Real-Time Implementation of Nonlinear Unsharp Masking with FPLDs.," *Journal of Real-Time Imaging,* vol. 7, no. 2, pp. 195-202, April 2001.

[29] N. Gupta and P. Sinha, "FPGA implementation of fuzzy morphological filters," *Proc. SPIE 5297, Real-Time Imaging VIII,* vol. 5297, pp. 220-230, 18 May 2004.

[30] H. Jiang, H. Ardö and V. Öwall, "Hardware accelerator design for video segmentation with multi-modal background modelling," *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on,* vol. 2, pp. 1142-1145, 23-26 May 2005.

[31] H. Jiang, H. Ardö and V. Öwall, "A Hardware Architecture for Real-Time Video Segmentation Utilizing Memory Reduction Techniques," *Circuits and Systems for Video Technology, IEEE Transactions on,* vol. 19, no. 2, pp. 226 - 236, Feb 2009.

[32] M. Genovese, E. Napoli and N. Petra, "OpenCV compatible real time processor for background foreground identification," *Microelectronics (ICM), 2010 International Conference on,* pp. 487-470, Dec 2010.

[33] 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, Dec 2013.

[34] M. Genovese and E. Napoli, "ASIC and FPGA Implementation of the Gaussian Mixture Model Algorithm for Real-Time Segmentation of High Definition Video," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 22, no. 3, pp. 537-547, 18 Mar 2014.

[35] T. Kryjak, M. Komorkiewicz and M. Gorgon, "Real-time moving object detection for video surveillance system in FPGA," *Design and Architectures for Signal and Image Processing (DASIP), 2011 Conference on,* pp. 1-8, Nov 2011.

[36] Xilinx, "Virtex-6 FPGA ML605 Evaluation Kit," Xilinx, [Online]. Available: http://www.xilinx.com/products/boards-and-kits/EK-V6-ML605-G.htm. [Accessed 3 2014].

[37] S. Y. Elhabian, K. M. El-Sayed and S. H. Ahmed, "Moving Object Detection in Spatial Domain using Background Removal Techniques," *Recent Patents on Computer Science,* vol. 1, pp. 32-54, 2008.

[38] N. McFarlane and C. Schofield, "Segmentation and tracking of piglets in images," *Machine Vision and Applications,* vol. 8, no. 3, pp. 187-193, 1 May 1995.

[39] D. Koller, J. Weber, T. Huang, J. Malik, G. Ogasawara, B. Rao and S. Russell, "Towards Robust Automatic Traffic Scene Analysis in Real-Time," *Pattern Recognition, 1994., Proceedings of the 12th IAPR International Conference on,* vol. 1, pp. 126-131, Oct 1994.

[40] R. Cutler and L. Davis, "View-based Detection and Analysis of Periodic Motion," *Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on,* vol. 1, pp. 495-500, Aug 1998.

[41] A. Monnet, A. Mittal, N. Paragios and V. Ramesh, "Background Modeling and Subtraction of Dynamic Scenes," *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on,* vol. 2, pp. 1305-1312, 13-16 Oct 2003.

[42] T. Bouwmans, "Recent Advanced Statistical Background Modeling for Foreground Detection - A Systematic Survey," *Recent Patents on Computer Science,* vol. 4, pp. 147-170, Sep 2011.

[43] T. Bouwmans, F. El Baf and B. Vachon, "Statistical Background Modeling for Foreground Detection: A Survey," in *Handbook oPattern Recognition and Computer Vision*, 4 ed., World Scientific Publishing, 2010, pp. 181-199.

[44] N. Friedman and S. J. Russell, "Image segmentation in video sequences: a probabilistic approach," *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence(UAI'97),* pp. 175-181, 1997.

[45] Xilinx, Inc., "LogiCORE IP Multi-Port Memory Controller (v6.05.a)," 2011. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/mpmc/v6_05_a/mpmc.pdf. [Accessed 15 Jul 2014].

[46] P. H. Sneath and R. R. Sokal, Numerical Taxonomy: The Principles and Practice of Numerical Classification, W. H. Freeman & Co (Sd), 1973.

[47] P. L. Rosin and E. Ioannidis, "Evaluation of global image thresholding for change detection," *Pattern Recognition Letters,* vol. 24, no. 14, pp. 2345-2356, Oct 2003.

[48] Avnet, Inc., "Dual Image Sensor FMC Module," [Online]. Available: http://www.em.avnet.com/en-us/design/drc/Pages/Dual-Image-Sensor-FMC-Module.aspx. [Accessed 15 Jul 2014].

[49] Wikimedia Foundation, Inc., "FMC – FPGA Mezzanine Card," [Online]. Available:http://en.wikipedia.org/wiki/FMC_%E2%80%93_FPGA_Mezzanine_Card. [Accessed 15 Jul 2014].

[50] Avnet, Inc, "OmniVision 1 MP Camera Kit with Ribbon Cable," [Online]. Available: http://www.em.avnet.com/en-us/design/drc/Pages/OmniVision-1-MP-Camera-Kit-with-Ribbon-Cable.aspx. [Accessed 15 Jul 2014].

[51] Texas Instruments, Inc., "TFP410 TI PanelBus Digital Transmitter Datasheet," 2001. [Online]. Available: http://www.ti.com/lit/ds/symlink/tfp410.pdf. [Accessed 15 Jul 2014].

[52] Avnet, Inc., "Spartan®-6 Industrial Video Processing Kit – EDK Reference Design Tutorial," 2010.

[53] Wikimedia Foundation, Inc., "Transition-minimized differential signaling," [Online]. Available: http://en.wikipedia.org/wiki/Transition-minimized_differential_signaling. [Accessed 15 Jul 2014].

[54] BEEcube, Inc, "BEEcube Platform Studio," 2012. [Online]. Available: http://www.beecube.com/downloads/BEEcube_Platform_Studio_BPS.pdf. [Accessed 3 7 2014].

[55] H. Jiang, V. Öwall and H. Ardö, "Real-Time Video Segmentation with VGA Resolution and Memory Bandwidth Reduction," *Video and Signal Based Surveillance, 2006. AVSS '06. IEEE International Conference on,* pp. 104-109, Nov 2006.

[56] M. Genovese and E. Napoli, "An FPGA-based Real-time Background Identification Circuit for 1080p Video," *Signal Image Technology and Internet Based Systems (SITIS), 2012 Eighth International Conference on,* pp. 330-335, Nov 2012.

[57] Xilinx, Inc., "Designing High-Performance Video Systems in 7 Series FPGAs with the AXI Interconnect," 14 Apr 2014. [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp741-high-performance-video-AXI-interconnect.pdf. [Accessed 24 Jul 2014].

[58] Xilinx, Inc., "AXI Reference Guide (UG761)," 7 Mar 2011. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf. [Accessed 23 Jul 2014].

[59] Xilinx, Inc., "Bridging Xilinx Streaming Video Interface with the AXI4-Stream Protocol," 1 Feb 2012. [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp521_XSVI_AXI4.pdf. [Accessed 23 Jul 2014].

Appendix I – Implementation Note

This appendix is provided as a complement to Chapter IV. It is very useful for the readers who intend to continue to work on this topic with FPGA in the lab.

FPGA Resource Utilization

Some important items of FPGA resource utilization are summarized in Table 10. The figures shown by this table indicate that there is still very much potential in Virtex-6 XC6VLX240T FPGA on the ML605 board.

Table 10 -- FPGA Resource Utilization of the Design

Slice Logic Utilization	Used	Available	Utilization
Number of occupied slices	8,850	37,680	23%
Number of RAMB36E1	87	461	20%
Number of RAMB18E1	4	832	1%
Number of DSP48E1s	19	768	2%

Design Files and Schematics Description

As mentioned in section 4.1.1, the design was completed in Xilinx Platform Studio (XPS). One of the most important files in XPS is the Microprocessor Hardware Specification (MHS) file. An MHS file defines the configuration of the embedded processor system, and includes the following:

- Bus architecture
- Peripherals
- Processor
- System Connectivity
- Address space

The MHS file is shown in the end of this appendix after some visualization of the design in the XPS. Figure 43 shows the assembly view of the design in XPS and Figure 44 the graphic view. These two figures give a general overview of the design, however not much details of each IP core could be given. The details of connection between each IP core and its configurations can be retrieved in the MHS file. As to the inside of each IP core, depending on its origin, it is intentionally hidden or can be shown in the platform where it is designed.

Most of the IP cores were designed in Xilinx EDA tools, so routines to examine these IP cores can be found in Xilinx documents. One exception is the enhanced SG background modeling pipeline, named as gaussian_model_test_0 in the XPS design view. This IP core, as mentioned in section 4.1.1, is designed in a third party tool, named BSP (BEECube Platform Studio) v4.0. BEEcube is a spin out from the University of California, Berkeley, where founders conducted decades of leading research on the FPGA-based Berkeley Emulation Engine (BEE) platforms and development environments. In this project, the BSP v4.0 was used to design the enhanced SG background modeling pipeline and generate an IP core accordingly that targets the ML605 evaluation board. Figure 45 shows the top view of the enhanced SG background modeling pipeline. Figure 46 shows the inside logic of the enhanced SG background modeling logic block, named Enhanced_SG_Delay_85 in Figure 45, the core of this pipeline. Although the other logic blocks in Figure 45 are also complex and important, they are BSP blocksets that are predefined in the Simulink environment and are not shown in this appendix.

Within the enhanced SG logic block (Figure 46), there are four logic blocks shown by their masks, namely RGB2GRAY_Delay_3 (Figure 47), ABS_UFix_16_8_Delay1 (Figure

48), Classification_Delay_1_1, and Classification_System_Delay_1_2 (both have the same logic as shown in Figure 49). Functionally speaking, their names are "RGB to Gray Logic block", "Absolute Operation Logic Block", and two of the "Classification Logic Block". The RGB to Gray logic block generates the gray level pixel intensity from the input RGB pixel data ($RGB_{(x,y,t)} \rightarrow I_{(x,y,t)}$). The Absolute Operation Logic Block generates the absolute difference ($\left|I_{(x,y,t)} - \mu_{(x,y,t)}\right|$) between two inputs. The Classification Logic Block compares the two inputs by performing the abstraction of them and indicates the result by the sign of the outcome and this sign is used in labeling the pixel later. Figures and MHS file are listed after this point.

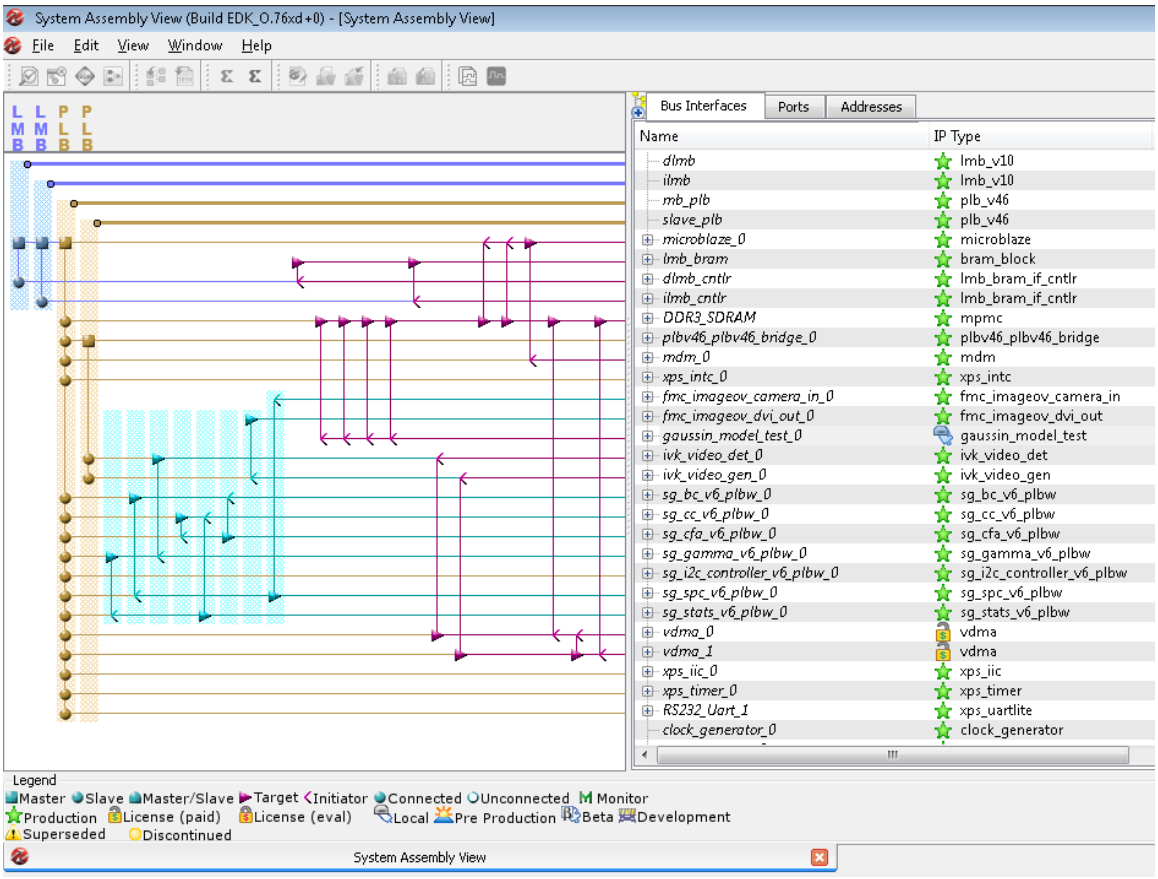

Figure 43 -- Assembly View of the Design in XPS

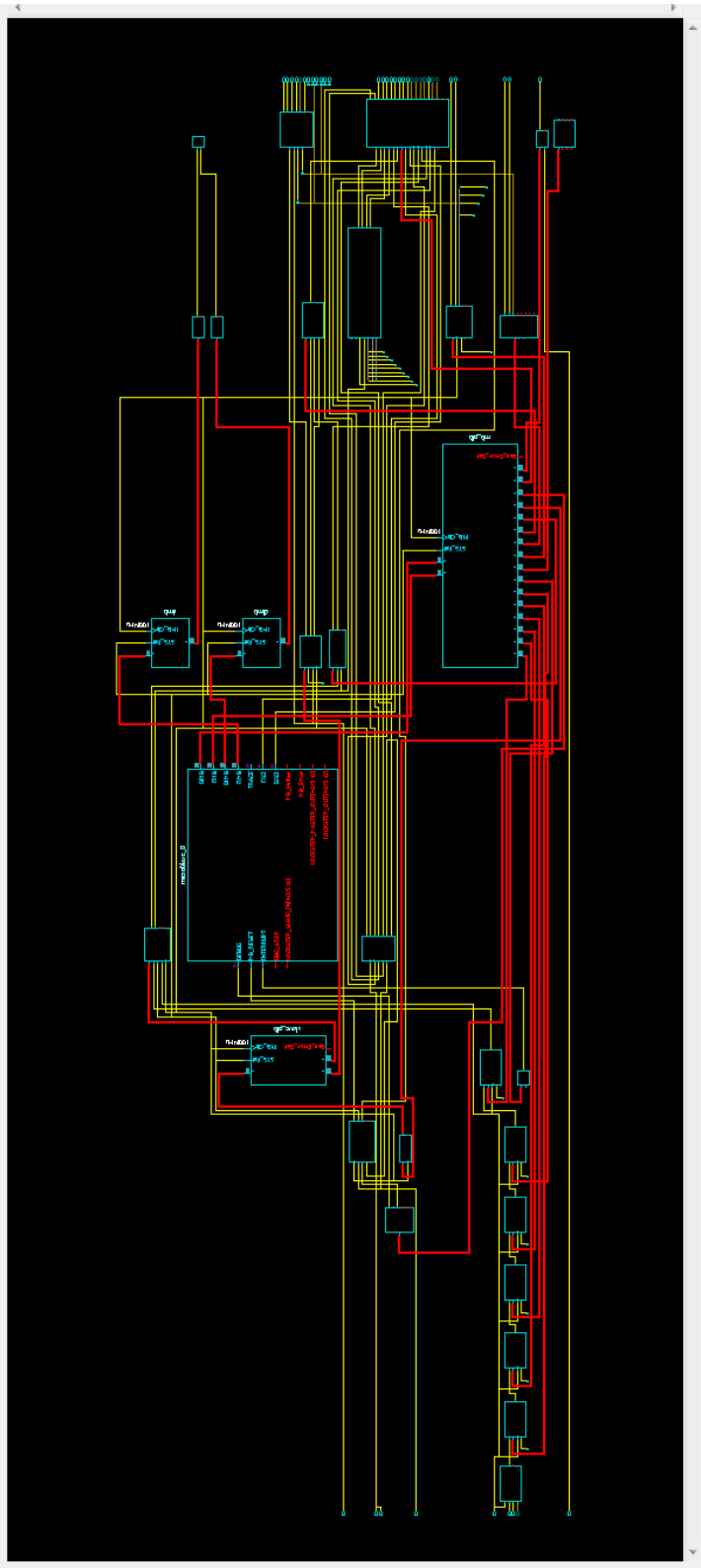

Figure 44 -- Graphic View of the Design in XPS

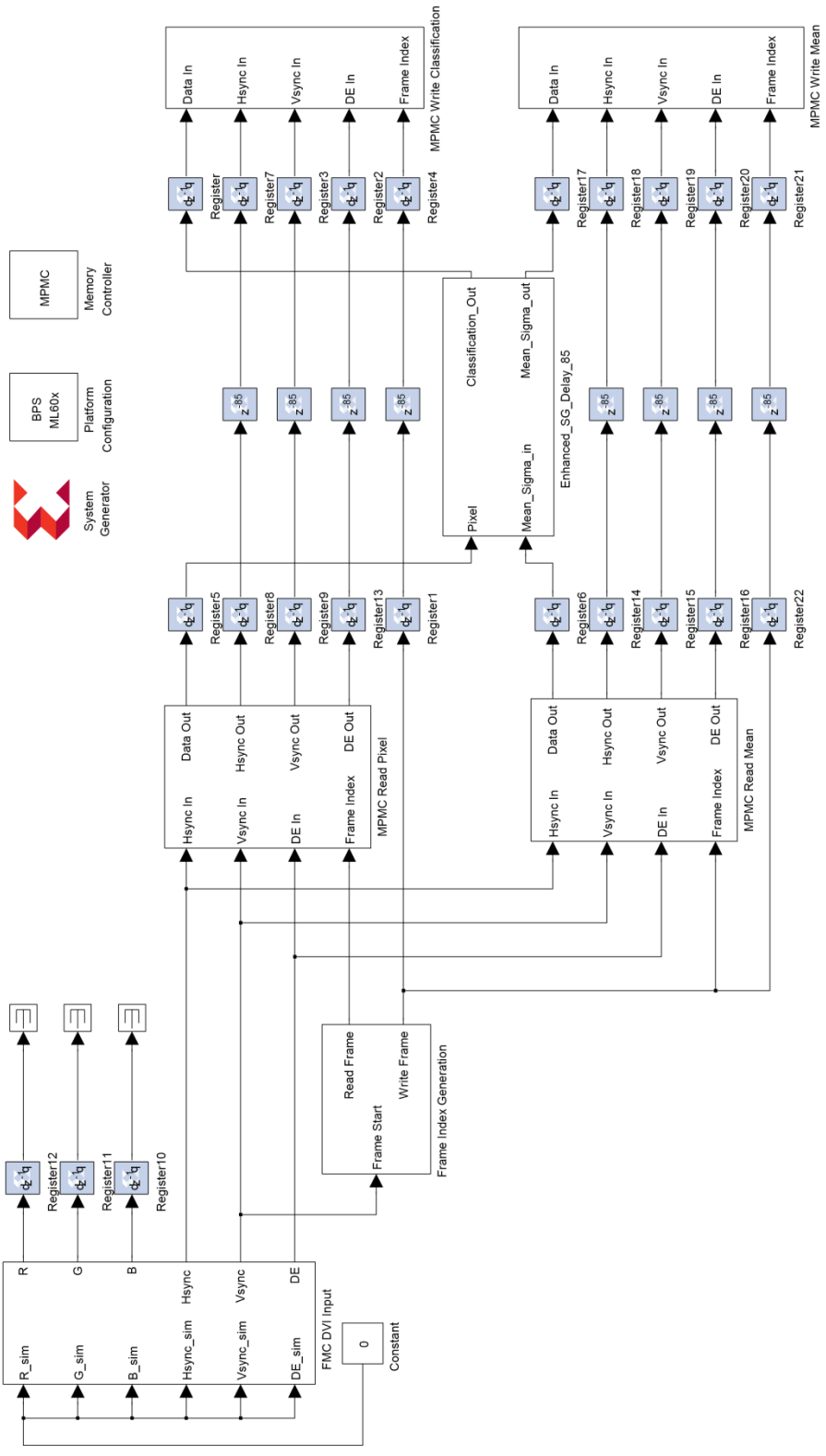

Figure 45 -- Enhanced SG Pipeline Top View in BPS

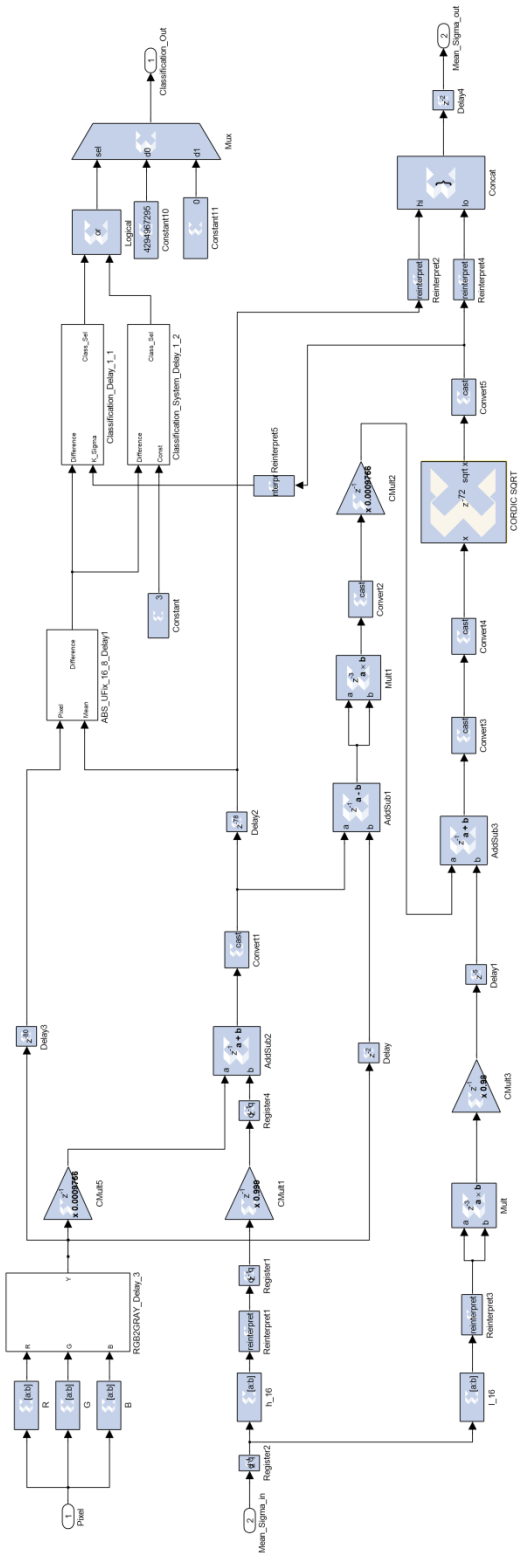

Figure 46 – Enhanced SG Logic Block inside View in BPS

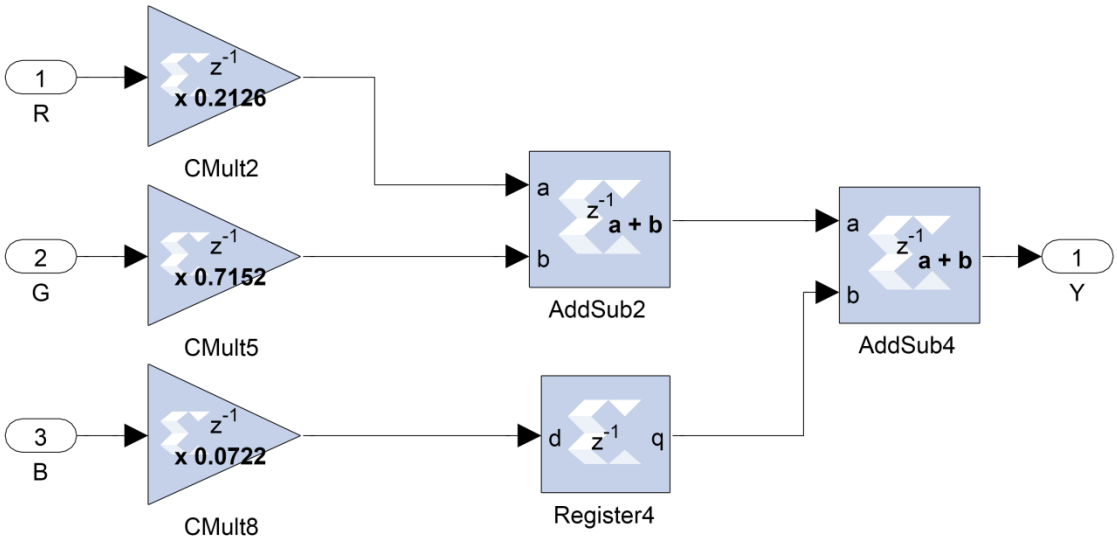

Figure 47 -- RGB to Gray Logic Block inside View in BPS

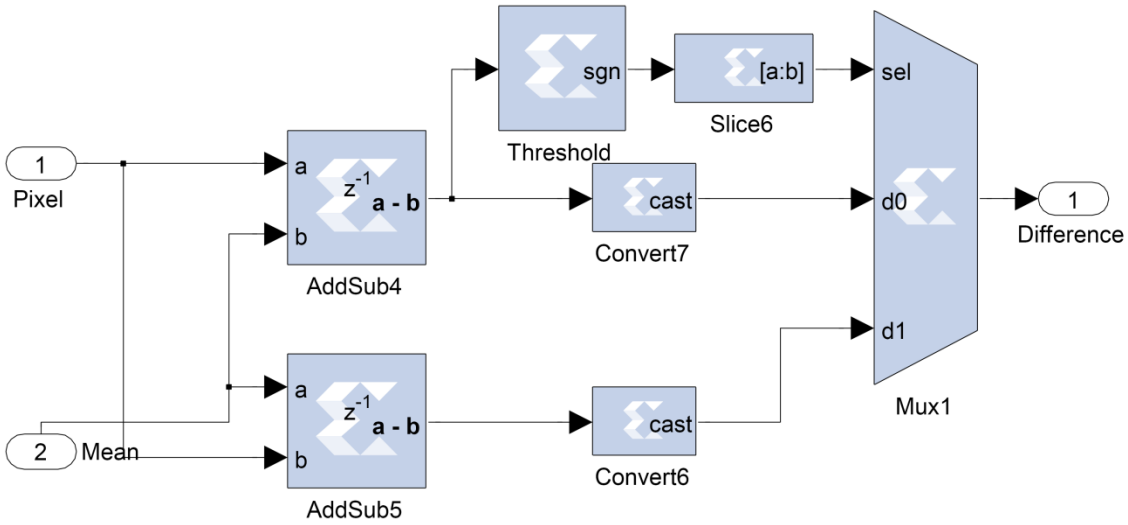

Figure 48 -- Absolute Operation Logic Block inside View in BPS

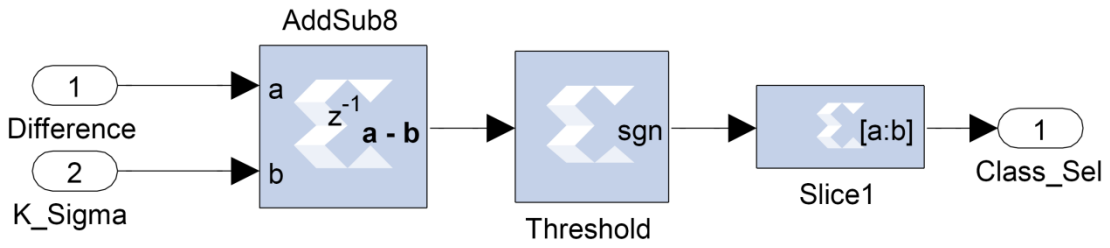

Figure 49 -- Classification Logic Block inside View in BPS

```
    # MHS file
    # Created by Base System Builder Wizard for Xilinx EDK 13.3 Build EDK_O.40d
    # Thu Nov 10 14:52:00 2013
    # Target Board:  Xilinx Virtex 6 ML605 Evaluation Platform Rev D
    # Family:    virtex6
    # Device:    xc6vlx240t
    # Package:   ff1156
    # Speed Grade:  -1
    # Processor number: 1
    # Processor 1: microblaze_0
    # System clock frequency: 100.0
    # Debug Interface: On-Chip HW Debug Module
     PARAMETER VERSION = 2.1.0

    # dcm_0_rst & fmc2_enable & fmc1_enable
     PORT LED_0_pin = fmc1_enable, DIR = O
     PORT LED_1_pin = fmc2_enable, DIR = O
     PORT LED_2_pin = dcm_0_rst, DIR = O
    #  PORT  fpga_0_LEDs_8Bit_GPIO_IO_pin  =  fpga_0_LEDs_8Bit_GPIO_IO_pin,
DIR = IO, VEC = [0:7]
     PORT fpga_0_RS232_Uart_1_RX_pin = fpga_0_RS232_Uart_1_RX_pin, DIR = I
     PORT fpga_0_RS232_Uart_1_TX_pin = fpga_0_RS232_Uart_1_TX_pin, DIR = O
     PORT                    fpga_0_DDR3_SDRAM_DDR3_Clk_pin                    =
fpga_0_DDR3_SDRAM_DDR3_Clk_pin, DIR = O
     PORT                  fpga_0_DDR3_SDRAM_DDR3_Clk_n_pin                    =
fpga_0_DDR3_SDRAM_DDR3_Clk_n_pin, DIR = O
     PORT                    fpga_0_DDR3_SDRAM_DDR3_CE_pin                     =
fpga_0_DDR3_SDRAM_DDR3_CE_pin, DIR = O
     PORT                  fpga_0_DDR3_SDRAM_DDR3_CS_n_pin                     =
fpga_0_DDR3_SDRAM_DDR3_CS_n_pin, DIR = O
     PORT                  fpga_0_DDR3_SDRAM_DDR3_ODT_pin                      =
fpga_0_DDR3_SDRAM_DDR3_ODT_pin, DIR = O
```

```
    PORT fpga_0_DDR3_SDRAM_DDR3_RAS_n_pin = fpga_0_DDR3_SDRAM_DDR3_RAS_n_pin, DIR = O
    PORT fpga_0_DDR3_SDRAM_DDR3_CAS_n_pin = fpga_0_DDR3_SDRAM_DDR3_CAS_n_pin, DIR = O
    PORT fpga_0_DDR3_SDRAM_DDR3_WE_n_pin = fpga_0_DDR3_SDRAM_DDR3_WE_n_pin, DIR = O
    PORT fpga_0_DDR3_SDRAM_DDR3_BankAddr_pin = fpga_0_DDR3_SDRAM_DDR3_BankAddr_pin, DIR = O, VEC = [2:0]
    PORT fpga_0_DDR3_SDRAM_DDR3_Addr_pin = fpga_0_DDR3_SDRAM_DDR3_Addr_pin, DIR = O, VEC = [12:0]
    PORT fpga_0_DDR3_SDRAM_DDR3_DQ_pin = fpga_0_DDR3_SDRAM_DDR3_DQ_pin, DIR = IO, VEC = [31:0]
    PORT fpga_0_DDR3_SDRAM_DDR3_DM_pin = fpga_0_DDR3_SDRAM_DDR3_DM_pin, DIR = O, VEC = [3:0]
    PORT fpga_0_DDR3_SDRAM_DDR3_Reset_n_pin = fpga_0_DDR3_SDRAM_DDR3_Reset_n_pin, DIR = O
    PORT fpga_0_DDR3_SDRAM_DDR3_DQS_pin = fpga_0_DDR3_SDRAM_DDR3_DQS_pin, DIR = IO, VEC = [3:0]
    PORT fpga_0_DDR3_SDRAM_DDR3_DQS_n_pin = fpga_0_DDR3_SDRAM_DDR3_DQS_n_pin, DIR = IO, VEC = [3:0]
    PORT fpga_0_clk_1_sys_clk_p_pin = CLK_S, DIR = I, SIGIS = CLK, DIFFERENTIAL_POLARITY = P, CLK_FREQ = 200000000
    PORT fpga_0_clk_1_sys_clk_n_pin = CLK_S, DIR = I, SIGIS = CLK, DIFFERENTIAL_POLARITY = N, CLK_FREQ = 200000000
    PORT fpga_0_rst_1_sys_rst_pin = sys_rst_s, DIR = I, SIGIS = RST, RST_POLARITY = 1
   # FMC
    PORT fmc_ipmi_i2c_scl = xps_iic_0_Scl, DIR = IO
    PORT fmc_ipmi_i2c_sda = xps_iic_0_Sda, DIR = IO
   # FMC-IMAGEOV - I2C
    PORT fmc_imageov_i2c_scl_pin = fmc_imageov_i2c_scl, DIR = O
    PORT fmc_imageov_i2c_sda_pin = fmc_imageov_i2c_sda, DIR = IO
    PORT fmc_imageov_i2c_rst_pin = fmc_imageov_i2c_rst, DIR = O
   # FMC-IMAGEOV - Video Clock Synthesizer
    PORT fmc_imageov_video_clk_pin = display_clk, DIR = I, SIGIS = CLK, CLK_FREQ = 74250000, BUFFER_TYPE = BUFR
   # FMC-IMAGEOV - Camera 1
    PORT fmc_imageov_cam1_pwdn_pin = fmc_imageov_cam1_pwdn, DIR = O
    PORT fmc_imageov_cam1_rst_pin = fmc_imageov_cam1_rst, DIR = O
    PORT fmc_imageov_cam1_clk_pin = vid_in_clk, DIR = I, SIGIS = CLK, CLK_FREQ = 40000000, BUFFER_TYPE = BUFR
    PORT fmc_imageov_cam1_frame_valid_pin = fmc_imageov_cam1_frame_valid, DIR = I
    PORT fmc_imageov_cam1_line_valid_pin = fmc_imageov_cam1_line_valid, DIR = I
```

```
 PORT fmc_imageov_cam1_data_pin = fmc_imageov_cam1_data, DIR = I, VEC = [9:0]
 # FMC-IMAGEOV - DVI output
 PORT fmc_imageov_dvi_reset_n_pin = fmc_imageov_dvi_reset_n, DIR = O
 PORT fmc_imageov_dvi_clk_pin = fmc_imageov_dvi_clk, DIR = O
 PORT fmc_imageov_dvi_de_pin = fmc_imageov_dvi_de, DIR = O
 PORT fmc_imageov_dvi_vsync_pin = fmc_imageov_dvi_vsync, DIR = O
 PORT fmc_imageov_dvi_hsync_pin = fmc_imageov_dvi_hsync, DIR = O
 PORT fmc_imageov_dvi_data_pin = fmc_imageov_dvi_data, DIR = O, VEC = [11:0]

BEGIN microblaze
 PARAMETER INSTANCE = microblaze_0
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_FPU = 1
 PARAMETER C_DEBUG_ENABLED = 1
 PARAMETER HW_VER = 8.20.a
 PARAMETER C_USE_MMU = 0
 PARAMETER C_USE_ICACHE = 1
 PARAMETER C_USE_DCACHE = 1
 PARAMETER C_CACHE_BYTE_SIZE = 65536
 PARAMETER C_ICACHE_LINE_LEN = 8
 PARAMETER C_ICACHE_BASEADDR = 0x90000000
 PARAMETER C_ICACHE_HIGHADDR = 0x9fffffff
 PARAMETER C_ICACHE_ALWAYS_USED = 1
 PARAMETER C_DCACHE_BYTE_SIZE = 65536
 PARAMETER C_DCACHE_LINE_LEN = 8
 PARAMETER C_DCACHE_BASEADDR = 0x90000000
 PARAMETER C_DCACHE_HIGHADDR = 0x9fffffff
 PARAMETER C_DCACHE_ALWAYS_USED = 1
 BUS_INTERFACE DPLB = mb_plb
 BUS_INTERFACE IPLB = mb_plb
 BUS_INTERFACE DEBUG = microblaze_0_mdm_bus
 BUS_INTERFACE DLMB = dlmb
 BUS_INTERFACE ILMB = ilmb
 BUS_INTERFACE IXCL = microblaze_0_IXCL
 BUS_INTERFACE DXCL = microblaze_0_DXCL
 PORT MB_RESET = mb_reset
 PORT INTERRUPT = microblaze_0_Interrupt
END

BEGIN plb_v46
 PARAMETER INSTANCE = mb_plb
 PARAMETER HW_VER = 1.05.a
 PORT PLB_Clk = clk_100_0000MHzMMCM0
 PORT SYS_Rst = sys_bus_reset
```

```
END

BEGIN lmb_v10
 PARAMETER INSTANCE = ilmb
 PARAMETER HW_VER = 2.00.b
 PORT LMB_Clk = clk_100_0000MHzMMCM0
 PORT SYS_Rst = sys_bus_reset
END

BEGIN lmb_v10
 PARAMETER INSTANCE = dlmb
 PARAMETER HW_VER = 2.00.b
 PORT LMB_Clk = clk_100_0000MHzMMCM0
 PORT SYS_Rst = sys_bus_reset
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = dlmb_cntlr
 PARAMETER HW_VER = 3.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00001fff
 BUS_INTERFACE SLMB = dlmb
 BUS_INTERFACE BRAM_PORT = dlmb_port
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = ilmb_cntlr
 PARAMETER HW_VER = 3.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00001fff
 BUS_INTERFACE SLMB = ilmb
 BUS_INTERFACE BRAM_PORT = ilmb_port
END

BEGIN bram_block
 PARAMETER INSTANCE = lmb_bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = ilmb_port
 BUS_INTERFACE PORTB = dlmb_port
END

BEGIN xps_uartlite
 PARAMETER INSTANCE = RS232_Uart_1
 PARAMETER C_BAUDRATE = 9600
 PARAMETER C_DATA_BITS = 8
```

```
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_ODD_PARITY = 0
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BASEADDR = 0x84000000
 PARAMETER C_HIGHADDR = 0x8400ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT RX = fpga_0_RS232_Uart_1_RX_pin
 PORT TX = fpga_0_RS232_Uart_1_TX_pin
END

BEGIN mpmc
 PARAMETER INSTANCE = DDR3_SDRAM
 PARAMETER C_NUM_PORTS = 8
# PARAMETER C_MMCM_EXT_LOC = MMCM_ADV_X0Y9
 PARAMETER C_MEM_TYPE = DDR3
 PARAMETER C_MEM_PARTNO = MT4JSF6464HY-1G1
 PARAMETER C_MEM_ODT_TYPE = 1
 PARAMETER C_MEM_REG_DIMM = 0
 PARAMETER C_MEM_CLK_WIDTH = 1
 PARAMETER C_MEM_ODT_WIDTH = 1
 PARAMETER C_MEM_CE_WIDTH = 1
 PARAMETER C_MEM_CS_N_WIDTH = 1
 PARAMETER C_MEM_DATA_WIDTH = 32
 PARAMETER HW_VER = 6.05.a
 PARAMETER C_PIM0_BASETYPE = 6
 PARAMETER C_PIM1_BASETYPE = 6
 PARAMETER C_PIM2_BASETYPE = 6
 PARAMETER C_PIM3_BASETYPE = 6
 PARAMETER C_PIM4_BASETYPE = 6
 PARAMETER C_PIM5_BASETYPE = 6
 PARAMETER C_PIM6_BASETYPE = 2
 PARAMETER C_PIM7_BASETYPE = 1
# PARAMETER C_PIM4_BASETYPE = 0
 PARAMETER C_RD_DATAPATH_TML_MAX_FANOUT = 8
 PARAMETER C_MEM_NDQS_COL0 = 3
 PARAMETER C_MEM_NDQS_COL1 = 1
 PARAMETER C_MEM_DQS_LOC_COL0 =
0x00000000000000000000000000020100
 PARAMETER C_MEM_DQS_LOC_COL1 =
0x00000000000000000000000000000003
 PARAMETER C_MPMC_BASEADDR = 0x90000000
 PARAMETER C_MPMC_HIGHADDR = 0x9FFFFFFF
 PARAMETER C_VFBC0_RDWD_FIFO_DEPTH = 1024
 PARAMETER C_VFBC1_RDWD_FIFO_DEPTH = 1024
 PARAMETER C_VFBC2_RDWD_FIFO_DEPTH = 1024
```

```
    PARAMETER C_VFBC3_RDWD_FIFO_DEPTH = 1024
    PARAMETER C_VFBC4_RDWD_FIFO_DEPTH = 1024
    PARAMETER C_VFBC5_RDWD_FIFO_DEPTH = 1024
    PARAMETER C_PI0_RD_FIFO_TYPE = DISABLED
    PARAMETER C_PI1_RD_FIFO_TYPE = DISABLED
    PARAMETER C_PI2_RD_FIFO_TYPE = DISABLED
    PARAMETER C_PI3_RD_FIFO_TYPE = SRL
    PARAMETER C_PI3_WR_FIFO_TYPE = DISABLED
    PARAMETER C_PI4_RD_FIFO_TYPE = SRL
    PARAMETER C_PI4_WR_FIFO_TYPE = DISABLED
    PARAMETER C_PI5_RD_FIFO_TYPE = SRL
    PARAMETER C_PI5_WR_FIFO_TYPE = DISABLED
    PARAMETER C_XCL7_B_IN_USE = 1
    BUS_INTERFACE VFBC0 = vdma_0_XIL_VFBC
    BUS_INTERFACE VFBC1 =
gaussin_model_test_0_gaussin_model_test_MPMC_Write_Classification_MPMC_VFB
C_vfbc
    BUS_INTERFACE VFBC2 =
gaussin_model_test_0_gaussin_model_test_MPMC_Write_Mean_MPMC_VFBC_vfbc
    BUS_INTERFACE VFBC3 =
gaussin_model_test_0_gaussin_model_test_MPMC_Read_Pixel_MPMC_VFBC_vfbc
    BUS_INTERFACE VFBC4 =
gaussin_model_test_0_gaussin_model_test_MPMC_Read_Mean_MPMC_VFBC_vfbc
    BUS_INTERFACE VFBC5 = vdma_1_XIL_VFBC
    BUS_INTERFACE SPLB6 = mb_plb
    BUS_INTERFACE XCL7 = microblaze_0_IXCL
    BUS_INTERFACE XCL7_B = microblaze_0_DXCL
    PORT VFBC4_Wd_Data_BE =
gaussin_model_test_0_gaussin_model_test_MPMC_Read_Mean_MPMC_VFBC_vfbc_
Wd_DataByteEn
    PORT VFBC3_Wd_Data_BE =
gaussin_model_test_0_gaussin_model_test_MPMC_Read_Pixel_MPMC_VFBC_vfbc_
Wd_DataByteEn
    PORT VFBC2_Wd_Data_BE =
gaussin_model_test_0_gaussin_model_test_MPMC_Write_Mean_MPMC_VFBC_vfbc_
Wd_DataByteEn
    PORT VFBC1_Wd_Data_BE =
gaussin_model_test_0_gaussin_model_test_MPMC_Write_Classification_MPMC_VFB
C_vfbc_Wd_DataByteEn
    PORT VFBC1_Cmd_Clk = ivk_video_det_0_XIL_WD_VDMA_wd_clk
    PORT VFBC1_Wd_Clk = ivk_video_det_0_XIL_WD_VDMA_wd_clk
    PORT VFBC1_Rd_Clk = ivk_video_det_0_XIL_WD_VDMA_wd_clk
    PORT VFBC2_Cmd_Clk = ivk_video_det_0_XIL_WD_VDMA_wd_clk
    PORT VFBC2_Wd_Clk = ivk_video_det_0_XIL_WD_VDMA_wd_clk
    PORT VFBC2_Rd_Clk = ivk_video_det_0_XIL_WD_VDMA_wd_clk
```

```
 PORT VFBC3_Cmd_Clk = ivk_video_det_0_XIL_WD_VDMA_wd_clk
 PORT VFBC3_Wd_Clk = ivk_video_det_0_XIL_WD_VDMA_wd_clk
 PORT VFBC3_Rd_Clk = ivk_video_det_0_XIL_WD_VDMA_wd_clk
 PORT VFBC4_Cmd_Clk = ivk_video_det_0_XIL_WD_VDMA_wd_clk
 PORT VFBC4_Wd_Clk = ivk_video_det_0_XIL_WD_VDMA_wd_clk
 PORT VFBC4_Rd_Clk = ivk_video_det_0_XIL_WD_VDMA_wd_clk
 PORT MPMC_Clk0 = clk_200_0000MHzMMCM0
 PORT MPMC_Clk_200MHz = clk_200_0000MHz
 PORT MPMC_Rst = sys_periph_reset
 PORT MPMC_Clk_Mem = clk_400_0000MHzMMCM0
 PORT MPMC_Clk_Rd_Base = clk_400_0000MHzMMCM0_nobuf_varphase
 PORT MPMC_DCM_PSEN = MPMC_DCM_PSEN
 PORT MPMC_DCM_PSINCDEC = MPMC_DCM_PSINCDEC
 PORT MPMC_DCM_PSDONE = MPMC_DCM_PSDONE
 PORT DDR3_Clk = fpga_0_DDR3_SDRAM_DDR3_Clk_pin
 PORT DDR3_Clk_n = fpga_0_DDR3_SDRAM_DDR3_Clk_n_pin
 PORT DDR3_CE = fpga_0_DDR3_SDRAM_DDR3_CE_pin
 PORT DDR3_CS_n = fpga_0_DDR3_SDRAM_DDR3_CS_n_pin
 PORT DDR3_ODT = fpga_0_DDR3_SDRAM_DDR3_ODT_pin
 PORT DDR3_RAS_n = fpga_0_DDR3_SDRAM_DDR3_RAS_n_pin
 PORT DDR3_CAS_n = fpga_0_DDR3_SDRAM_DDR3_CAS_n_pin
 PORT DDR3_WE_n = fpga_0_DDR3_SDRAM_DDR3_WE_n_pin
 PORT DDR3_BankAddr = fpga_0_DDR3_SDRAM_DDR3_BankAddr_pin
 PORT DDR3_Addr = fpga_0_DDR3_SDRAM_DDR3_Addr_pin
 PORT DDR3_DQ = fpga_0_DDR3_SDRAM_DDR3_DQ_pin
 PORT DDR3_DM = fpga_0_DDR3_SDRAM_DDR3_DM_pin
 PORT DDR3_Reset_n = fpga_0_DDR3_SDRAM_DDR3_Reset_n_pin
 PORT DDR3_DQS = fpga_0_DDR3_SDRAM_DDR3_DQS_pin
 PORT DDR3_DQS_n = fpga_0_DDR3_SDRAM_DDR3_DQS_n_pin
END

BEGIN clock_generator
 PARAMETER INSTANCE = clock_generator_0
 PARAMETER C_CLKIN_FREQ = 200000000
 PARAMETER C_CLKOUT0_FREQ = 100000000
 PARAMETER C_CLKOUT0_PHASE = 0
 PARAMETER C_CLKOUT0_GROUP = MMCM0
 PARAMETER C_CLKOUT0_BUF = TRUE
 PARAMETER C_CLKOUT1_FREQ = 200000000
 PARAMETER C_CLKOUT1_PHASE = 0
 PARAMETER C_CLKOUT1_GROUP = MMCM0
 PARAMETER C_CLKOUT1_BUF = TRUE
 PARAMETER C_CLKOUT2_FREQ = 400000000
 PARAMETER C_CLKOUT2_PHASE = 0
 PARAMETER C_CLKOUT2_GROUP = MMCM0
```

```
 PARAMETER C_CLKOUT2_BUF = TRUE
 PARAMETER C_CLKOUT3_FREQ = 400000000
 PARAMETER C_CLKOUT3_PHASE = 0
 PARAMETER C_CLKOUT3_GROUP = MMCM0
 PARAMETER C_CLKOUT3_BUF = FALSE
 PARAMETER C_CLKOUT3_VARIABLE_PHASE = TRUE
 PARAMETER C_PSDONE_GROUP = MMCM0
 PARAMETER C_EXT_RESET_HIGH = 1
 PARAMETER HW_VER = 4.03.a
 PARAMETER C_CLKOUT4_FREQ = 200000000
 PORT CLKIN = CLK_S
 PORT CLKOUT0 = clk_100_0000MHzMMCM0
 PORT CLKOUT1 = clk_200_0000MHzMMCM0
 PORT CLKOUT2 = clk_400_0000MHzMMCM0
 PORT CLKOUT3 = clk_400_0000MHzMMCM0_nobuf_varphase
 PORT PSCLK = clk_200_0000MHzMMCM0
 PORT PSEN = MPMC_DCM_PSEN
 PORT PSINCDEC = MPMC_DCM_PSINCDEC
 PORT PSDONE = MPMC_DCM_PSDONE
 PORT RST = sys_rst_s
 PORT LOCKED = Dcm_all_locked
 PORT CLKOUT4 = clk_200_0000MHz
END

BEGIN mdm
 PARAMETER INSTANCE = mdm_0
 PARAMETER C_MB_DBG_PORTS = 1
 PARAMETER C_USE_UART = 1
 PARAMETER HW_VER = 2.00.b
 PARAMETER C_BASEADDR = 0x84400000
 PARAMETER C_HIGHADDR = 0x8440ffff
 BUS_INTERFACE SPLB = mb_plb
 BUS_INTERFACE MBDEBUG_0 = microblaze_0_mdm_bus
 PORT Debug_SYS_Rst = Debug_SYS_Rst
END

BEGIN proc_sys_reset
 PARAMETER INSTANCE = proc_sys_reset_0
 PARAMETER C_EXT_RESET_HIGH = 1
 PARAMETER HW_VER = 3.00.a
 PORT Slowest_sync_clk = clk_100_0000MHzMMCM0
 PORT Ext_Reset_In = sys_rst_s
 PORT MB_Debug_Sys_Rst = Debug_SYS_Rst
 PORT Dcm_locked = Dcm_all_locked
 PORT MB_Reset = mb_reset
```

```
 PORT Bus_Struct_Reset = sys_bus_reset
 PORT Peripheral_Reset = sys_periph_reset
END

BEGIN plbv46_plbv46_bridge
 PARAMETER INSTANCE = plbv46_plbv46_bridge_0
 PARAMETER HW_VER = 1.04.a
 PARAMETER C_NUM_ADDR_RNG = 1
 PARAMETER C_BRIDGE_BASEADDR = 0x86200000
 PARAMETER C_BRIDGE_HIGHADDR = 0x8620ffff
 PARAMETER C_RNG0_BASEADDR = 0xc2000000
 PARAMETER C_RNG0_HIGHADDR = 0xc203ffff
 BUS_INTERFACE MPLB = slave_plb
 BUS_INTERFACE SPLB = mb_plb
 PORT MPLB_Clk = clk_100_0000MHzMMCM0
END

BEGIN plb_v46
 PARAMETER INSTANCE = slave_plb
 PARAMETER HW_VER = 1.05.a
 PORT PLB_Clk = clk_100_0000MHzMMCM0
 PORT SYS_Rst = sys_bus_reset
END

BEGIN fmc_imageov_camera_in
 PARAMETER INSTANCE = fmc_imageov_camera_in_0
 PARAMETER HW_VER = 2.01.a
 PARAMETER C_DATA_WIDTH = 8
 BUS_INTERFACE XSVI_VIDEO_OUT =
fmc_imageov_camera_in_0_XSVI_VIDEO_OUT
 PORT clk = vid_in_clk
 PORT io_frame_valid_i = fmc_imageov_cam1_frame_valid
 PORT io_line_valid_i = fmc_imageov_cam1_line_valid
 PORT io_data_i = fmc_imageov_cam1_data
END

BEGIN fmc_imageov_dvi_out
 PARAMETER INSTANCE = fmc_imageov_dvi_out_0
 PARAMETER HW_VER = 2.01.a
 PARAMETER C_DATA_WIDTH = 24
 BUS_INTERFACE XSVI_VIDEO_IN = ivk_video_gen_0_XSVI_VIDEO_OUT
 PORT io_dvi_clk = fmc_imageov_dvi_clk
 PORT io_dvi_de = fmc_imageov_dvi_de
 PORT io_dvi_vsync = fmc_imageov_dvi_vsync
 PORT io_dvi_hsync = fmc_imageov_dvi_hsync
```

```
 PORT io_dvi_data = fmc_imageov_dvi_data
 PORT io_dvi_reset_n = fmc_imageov_dvi_reset_n
 PORT clk = display_clk
 PORT reset = fmc_imageov_dvi_rst
# PORT ce = net_vcc
 PORT oe = fmc2_enable
END

# PORT oe = fmc1_enable
BEGIN ivk_video_det
 PARAMETER INSTANCE = ivk_video_det_0
 PARAMETER HW_VER = 2.01.a
 PARAMETER C_GEN_XSVI_OUT = 0
 PARAMETER C_GEN_WD_VDMA = 1
 PARAMETER C_GEN_FSYNC = 1
 PARAMETER C_XSVIO_DATA_WIDTH = 32
 PARAMETER C_XSVII_DATA_WIDTH = 24
 PARAMETER C_BASEADDR = 0xc2020000
 PARAMETER C_HIGHADDR = 0xc202ffff
 BUS_INTERFACE SPLB = slave_plb
 BUS_INTERFACE XIL_WD_VDMA = ivk_video_det_0_XIL_WD_VDMA
 BUS_INTERFACE XSVI_VIDEO_IN =
sg_gamma_v6_plbw_0_XSVI_VIDEO_OUT
 PORT reset = sys_bus_reset
 PORT clk = vid_in_clk
 PORT fsync_o = ivk_video_det_0_fsync
 PORT SPLB_Clk = clk_100_0000MHzMMCM0
END

BEGIN ivk_video_gen
 PARAMETER INSTANCE = ivk_video_gen_0
 PARAMETER HW_VER = 2.01.a
 PARAMETER C_GEN_FSYNC = 1
 PARAMETER C_GEN_RD_VDMA = 1
 PARAMETER C_VIDEO_INTERFACE = 2
 PARAMETER C_XSVI_DATA_WIDTH = 24
 PARAMETER C_VDMA_DATA_WIDTH = 32
 PARAMETER C_BASEADDR = 0xc2000000
 PARAMETER C_HIGHADDR = 0xc200ffff
 BUS_INTERFACE SPLB = slave_plb
 BUS_INTERFACE XIL_RD_VDMA = ivk_video_gen_0_XIL_RD_VDMA
 BUS_INTERFACE XSVI_VIDEO_OUT = ivk_video_gen_0_XSVI_VIDEO_OUT
 PORT reset = net_gnd
 PORT clk = display_clk
 PORT fsync_o = ivk_video_gen_0_fsync_o
```

```
 PORT SPLB_Clk = clk_100_0000MHzMMCM0
END

BEGIN vdma
 PARAMETER INSTANCE = vdma_0
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_MPMC_BASEADDR = 0x90000000
 PARAMETER C_MPMC_HIGHADDR = 0x9fffffff
 PARAMETER C_USE_FSYNC = 1
 PARAMETER C_GEN_RESET = 1
 PARAMETER C_NUM_FSTORES = 5
 PARAMETER C_CROP_ENABLE = 0
 PARAMETER C_BASEADDR = 0xcb420000
 PARAMETER C_HIGHADDR = 0xcb42ffff
 BUS_INTERFACE SPLB = mb_plb
 BUS_INTERFACE XIL_WD_VDMA = ivk_video_det_0_XIL_WD_VDMA
 BUS_INTERFACE XIL_WD_MGENLOCK = vdma_0_XIL_WD_MGENLOCK
 BUS_INTERFACE XIL_VFBC = vdma_0_XIL_VFBC
 PORT fsync = ivk_video_det_0_fsync
END

BEGIN vdma
 PARAMETER INSTANCE = vdma_1
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_DMA_TYPE = 1
 PARAMETER C_USE_FSYNC = 1
 PARAMETER C_MPMC_BASEADDR = 0x90000000
 PARAMETER C_MPMC_HIGHADDR = 0x9fffffff
 PARAMETER C_NUM_FSTORES = 5
 PARAMETER C_GEN_RESET = 0
 PARAMETER C_CROP_ENABLE = 0
 PARAMETER C_BASEADDR = 0xcb400000
 PARAMETER C_HIGHADDR = 0xcb40ffff
 BUS_INTERFACE SPLB = mb_plb
 BUS_INTERFACE XIL_RD_VDMA = ivk_video_gen_0_XIL_RD_VDMA
 BUS_INTERFACE XIL_RD_SGENLOCK1 = vdma_0_XIL_WD_MGENLOCK
 BUS_INTERFACE XIL_VFBC = vdma_1_XIL_VFBC
 PORT fsync = ivk_video_gen_0_fsync_o
END

BEGIN xps_iic
 PARAMETER INSTANCE = xps_iic_0
 PARAMETER HW_VER = 2.03.a
 PARAMETER C_GPO_WIDTH = 3
 PARAMETER C_BASEADDR = 0x81600000
```

```
 PARAMETER C_HIGHADDR = 0x8160ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT Scl = xps_iic_0_Scl
 PORT Sda = xps_iic_0_Sda
 PORT Gpo = dcm_0_rst & fmc2_enable & fmc1_enable
END

# PORT Gpo = DCM & 0b0 & fmc1_enable
BEGIN sg_i2c_controller_v6_plbw
 PARAMETER INSTANCE = sg_i2c_controller_v6_plbw_0
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_BASEADDR = 0xce000000
 PARAMETER C_HIGHADDR = 0xce00ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT i2c_scl = fmc_imageov_i2c_scl
 PORT i2c_sda = fmc_imageov_i2c_sda
 PORT gpio_out8_o = 0b0 & fmc_imageov_dvi_rst & fmc_imageov_i2c_rst & 0b0 &
0b0 & fmc_imageov_cam1_rst & 0b0 & fmc_imageov_cam1_pwdn
 PORT sysgen_clk = clk_100_0000MHzMMCM0
 PORT splb_rst = net_gnd
END

BEGIN xps_timer
 PARAMETER INSTANCE = xps_timer_0
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BASEADDR = 0x83c00000
 PARAMETER C_HIGHADDR = 0x83c0ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT Interrupt = xps_timer_0_Interrupt
END

BEGIN xps_intc
 PARAMETER INSTANCE = xps_intc_0
 PARAMETER HW_VER = 2.01.a
 PARAMETER C_BASEADDR = 0x81800000
 PARAMETER C_HIGHADDR = 0x8180ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT Intr = xps_timer_0_Interrupt
 PORT Irq = microblaze_0_Interrupt
END

BEGIN sg_spc_v6_plbw
 PARAMETER INSTANCE = sg_spc_v6_plbw_0
 PARAMETER HW_VER = 3.01.a
 PARAMETER C_BASEADDR = 0xc1000000
```

```
 PARAMETER C_HIGHADDR = 0xc100ffff
 BUS_INTERFACE SPLB = mb_plb
 BUS_INTERFACE XSVI_VIDEO_IN = fmc_imageov_camera_in_0_XSVI_VIDEO_OUT
 BUS_INTERFACE XSVI_VIDEO_OUT = sg_spc_v6_plbw_0_XSVI_VIDEO_OUT
 PORT sysgen_clk = vid_in_clk
 PORT splb_rst = net_gnd
END

BEGIN sg_bc_v6_plbw
 PARAMETER INSTANCE = sg_bc_v6_plbw_0
 PARAMETER HW_VER = 3.01.a
 PARAMETER C_BASEADDR = 0xcfe20000
 PARAMETER C_HIGHADDR = 0xcfe2ffff
 BUS_INTERFACE SPLB = mb_plb
 BUS_INTERFACE XSVI_VIDEO_IN = sg_spc_v6_plbw_0_XSVI_VIDEO_OUT
 BUS_INTERFACE XSVI_VIDEO_OUT = sg_bc_v6_plbw_0_XSVI_VIDEO_OUT
 PORT sysgen_clk = vid_in_clk
 PORT splb_rst = net_gnd
END

BEGIN sg_cfa_v6_plbw
 PARAMETER INSTANCE = sg_cfa_v6_plbw_0
 PARAMETER HW_VER = 3.01.b
 PARAMETER C_BASEADDR = 0xc1020000
 PARAMETER C_HIGHADDR = 0xc102ffff
 BUS_INTERFACE SPLB = mb_plb
 BUS_INTERFACE XSVI_VIDEO_IN = sg_bc_v6_plbw_0_XSVI_VIDEO_OUT
 BUS_INTERFACE XSVI_VIDEO_OUT = sg_cfa_v6_plbw_0_XSVI_VIDEO_OUT
 PORT sysgen_clk = vid_in_clk
 PORT splb_rst = net_gnd
END

BEGIN sg_cc_v6_plbw
 PARAMETER INSTANCE = sg_cc_v6_plbw_0
 PARAMETER HW_VER = 3.01.b
 PARAMETER C_BASEADDR = 0xcfe00000
 PARAMETER C_HIGHADDR = 0xcfe0ffff
 BUS_INTERFACE SPLB = mb_plb
 BUS_INTERFACE XSVI_VIDEO_IN = sg_cfa_v6_plbw_0_XSVI_VIDEO_OUT
 BUS_INTERFACE XSVI_VIDEO_OUT = sg_cc_v6_plbw_0_XSVI_VIDEO_OUT
 PORT sysgen_clk = vid_in_clk
 PORT splb_rst = net_gnd
END
```

```
 BEGIN sg_stats_v6_plbw
  PARAMETER INSTANCE = sg_stats_v6_plbw_0
  PARAMETER HW_VER = 3.01.b
  PARAMETER C_BASEADDR = 0xc3600000
  PARAMETER C_HIGHADDR = 0xc360ffff
  BUS_INTERFACE SPLB = mb_plb
  BUS_INTERFACE XSVI_VIDEO_IN = sg_cc_v6_plbw_0_XSVI_VIDEO_OUT
  BUS_INTERFACE XSVI_VIDEO_OUT =
sg_stats_v6_plbw_0_XSVI_VIDEO_OUT
  PORT sysgen_clk = vid_in_clk
  PORT splb_rst = net_gnd
 END

 BEGIN sg_gamma_v6_plbw
  PARAMETER INSTANCE = sg_gamma_v6_plbw_0
  PARAMETER HW_VER = 3.01.c
  PARAMETER C_BASEADDR = 0xc3620000
  PARAMETER C_HIGHADDR = 0xc362ffff
  BUS_INTERFACE SPLB = mb_plb
  BUS_INTERFACE XSVI_VIDEO_IN = sg_stats_v6_plbw_0_XSVI_VIDEO_OUT
  BUS_INTERFACE XSVI_VIDEO_OUT =
sg_gamma_v6_plbw_0_XSVI_VIDEO_OUT
  PORT sysgen_clk = vid_in_clk
  PORT splb_rst = net_gnd
  PORT vsync_i = fmc_imageov_camera_in_0_XSVI_VIDEO_OUT_vsync
  PORT hsync_i = fmc_imageov_camera_in_0_XSVI_VIDEO_OUT_hsync
 END

 BEGIN gaussin_model_test
  PARAMETER INSTANCE = gaussin_model_test_0
  PARAMETER HW_VER = 1.00.j
  BUS_INTERFACE  gaussin_model_test_MPMC_Read_Pixel_MPMC_VFBC_vfbc
= gaussin_model_test_0_gaussin_model_test_MPMC_Read_Pixel_MPMC_VFBC_vfbc
  BUS_INTERFACE  gaussin_model_test_MPMC_Read_Mean_MPMC_VFBC_vfbc
= gaussin_model_test_0_gaussin_model_test_MPMC_Read_Mean_MPMC_VFBC_vfbc
  BUS_INTERFACE  gaussin_model_test_MPMC_Write_Mean_MPMC_VFBC_vfbc
= gaussin_model_test_0_gaussin_model_test_MPMC_Write_Mean_MPMC_VFBC_vfbc
  BUS_INTERFACE
gaussin_model_test_MPMC_Write_Classification_MPMC_VFBC_vfbc =
gaussin_model_test_0_gaussin_model_test_MPMC_Write_Classification_MPMC_VFB
C_vfbc
  PORT clk = ivk_video_det_0_XIL_WD_VDMA_wd_clk
  PORT gaussin_model_test_FMC_DVI_Input_DE =
ivk_video_det_0_XIL_WD_VDMA_wd_write
```

```
PORT gaussin_model_test_FMC_DVI_Input_HSYNC = net_gnd
PORT gaussin_model_test_FMC_DVI_Input_VSYNC = ivk_video_det_0_fsync
PORT gaussin_model_test_FMC_DVI_Input_BLUE = net_gnd
PORT gaussin_model_test_FMC_DVI_Input_GREEN = net_gnd
PORT gaussin_model_test_FMC_DVI_Input_RED = net_gnd
```

Glossary

AMBA.....Advanced Microcontroller Bus Architecture

ANSI ..... *American National Standards Institute*

AXI .....*Advanced eXtensible Interface*

BEE ..... *Berkeley Emulation Engine*

BPS ..... *BEECube Platform Studio*

CCTV .....*Closed-Circuit Television*

CFA.....*Color Filter Array*

CLB..... *Configurable Logic Block*

DMA .....*Direct Memory Access*

DSP .....*Digital Signal Processing*

DVI ..... *Digital Visual Interface*

EAV ..... *End of Active Video*

EDA .....*Electronic Design Automation*

FMC .....*FPGA Mezzanine Card*

FPGA .....*Field Programmable Gate Array*

HDL .....*Hardware Description Language*

HDMI.....*High-Definition Multimedia Interface*

IP .....*Intellectual Property*

KDE .....*Kernel Density Estimation*

LAB.....*Array Logic Block*

MOD ..... *Moving Object Detection*

MOG ..... *Mixture of Gaussian*

MPMC..... *Multi-Port Memory Controller*

PC..... *Personal Computer*

QCIF ..... *Quarter Common Image Format*

ROI..... *Region of Interest*

SAV..... *Start of Active Video*

SG ..... *Single Gaussian*

SIMD..... *Single Instruction Multiple Data*

VDMA ..... *Video Direct Memory Access*

VFBC ..... *Video Frame Buffer Connector*

VITA ..... *VMEbus International Trade Association*

VLIW ..... *Very Long Instruction Word*

XPS ..... *Xilinx Platform Studio*

XSG..... *Xilinx System Generator*

XVSI ..... *Xilinx Video Stream Interface*

Curriculum Vitae

Candidate's full name: Ge Guo

Universities attended: Dalian University of Technology, China, 2006-2011

Degrees awarded: B.ScE. in Electronic and Information Engineering, 2011

Diploma in English, 2011