University of Calgary

PRISM Repository **https://prism.ucalgary.ca**

The Vault Open Theses and Dissertations

2013-01-25

A Secure, Low Power and Low Hardware Digital Watermarking System for Real-time Surveillance Video Authentication

Roy, Sonjoy Deb

Roy, S. D. (2013). A Secure, Low Power and Low Hardware Digital Watermarking System for Real-time Surveillance Video Authentication (Master's thesis, University of Calgary, Calgary, Canada). Retrieved from https://prism.ucalgary.ca. doi:10.11575/PRISM/25003

http://hdl.handle.net/11023/469

Downloaded from PRISM Repository, University of Calgary

UNIVERSITY OF CALGARY

A Secure, Low Power and Low Hardware Digital Watermarking System for Real-time Surveillance Video Authentication

by

Sonjoy Deb Roy

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

JANUARY, 2013

© Sonjoy Deb Roy 2013

Abstract

Real-time video authentication is a major concern for surveillance video cameras. In this thesis, we present a secure, low power and low hardware video watermarking system that inserts an invisible semi-fragile watermark into the video stream for real-time surveillance video authentication during the commonly used MPEG-4 encoding process with insignificant video quality degradation. Our watermarking system was prototyped on an Altera® Cyclone FPGA device and was evaluated to ensure the achievement of high processing speed, low power and low hardware goals.

In addition, we propose a general threat model for video watermarking system intended for real-time surveillance video authentication. Grain – 128a, a secure stream cipher, was chosen as a source of low hardware cryptographically secure pseudorandom number generator to generate a secure watermark sequence. The system was also assessed based on different recognized watermarking attacks and the proposed threat model to provide evidence for the achievement of the security goals.

Acknowledgements

First and foremost, I offer my sincerest gratitude to my supervisor Dr. Orly Yadid-Pecht, who has supported me throughout my graduate research with her patience and experience. I also gratefully acknowledge my co-supervisor Dr. Michael John Jacobson for his guidance and contribution in a crucial period. Without both of them this research would not have been completed. Their encouragement and effort provided me the opportunity to develop my knowledge and research skills.

It is a pleasure to express my gratitude to the supervisory committee who kindly agreed to review this dissertation. This research was supported by NSERC, iCORE and CMC Microsystems.

I convey special acknowledgement to my colleagues Hamid, Lior, Chika, Qing and Yonathan in Integrated Sensors Intelligent Systems (ISIS) laboratory at University of Calgary for their selfless help. I would like to thank all the stuffs of the Department of Electrical and Computer Engineer, especially to Pauline Cummings. I am grateful to my friends and relatives, who provided me the strength and the inspiration to continue my study in a country far away from my family. I also convey my appreciation to my friend Sharmila for her moral support.

Words fail me to express my gratefulness to my family for all their love and encouragement. My parents, grandparents and my sister have given me their unequivocal support throughout, as always, for which my mere expression of thanks likewise does not suffice. Without their inspiration, I may never have overcome this long journey in my studies.

To

My lovely sisters and brothers,

Niki, Mithila, Tonmoy, Trisha, Joyeeta and

Aindrila

Table of Contents

Abstract ii
Acknowledgements iii
Dedication iv
Table of Contents v
List of Tables ix
List of Figures and Illustrations x
List of Abbreviations xiii

CHAPTER ONE: INTRODUCTION 1
1.1 Problem Statement 2
1.2 Objective 4
1.3 Thesis Organization 6

CHAPTER TWO: BACKGROUND 9
2.1 Digital Watermarking 9
2.2 Types of Watermarking 9
2.2.1 Processing Domain 9
2.2.2 Robustness Level 10
2.2.3 Human Perception 10
2.3 Types of Watermarking Implementation 11
2.4 Requirements for Surveillance Video Watermarking 12
2.5 Reasons for Semi-fragile Watermarking 14
2.6 Past Works on Hardware Based Video Watermarking 15
2.7 Watermarking Algorithm for current work 18
2.8 Video Encoding and Watermarking 23
2.9 Overview of the Proposed Video Watermarking System for Real-time Authentication 24
2.10 Why CMOS Image Sensor Based Camera? 26
2.11 Background on Random Number Generator 26
2.11.1 True Random Number Generator 27
2.11.2 Pseudorandom Number Generator (PRNG) 27
2.11.3 Cryptographically Secure Pseudorandom Number Generator (CSPRNG) 34

CHAPTER THREE: HARDWARE WATERMARKING SYSTEM FOR MJPEG VIDEO AUTHENTICATION 45
3.1 Introduction 45
3.2 Contribution 46
3.3 Proposed Digital Video Watermarking 47
3.3.1 Video Compression 48
3.3.2 Watermark Generation 51
3.3.3 Watermark Embedding 55
3.4 Hardware Architecture Design 57
3.4.1 Video Compressor 58
3.4.2 Watermark Generator 61
3.4.3 Watermark Embedder 62

3.4.4 Control Unit 64
3.4.5 FPGA-based Prototyping 66
3.5 Experimental Results 66
3.5.1 Methodology of Verification 66
3.5.2 Analysis of Watermarked Video Quality 67
3.5.3 Hardware Cost and Power Consumption: 71
3.5.4 Processing Speed 72
3.5.5 Robustness Analysis: 74
3.5.6 Security issues 76
3.6 Comparison with Existing Research 78
3.7 Summary 80

CHAPTER FOUR: LOW POWER AND LOW HARDWARE REAL-TIME MPEG-4 VIDEO WATERMARKING SYSTEM 81
4.1 Introduction 81
4.2 Contribution 82
4.3 Watermarking in MPEG-4 Video Encoding Framework 83
4.4 Hardware Architecture 85
4.4.1 CMOS Based Video Camera 85
4.4.2 Colorspace Conversion Block 85
4.4.3 Input Frame Buffer 86
4.4.4 Frame Sorter 86
4.4.5 2D DCT and Quantization 86
4.4.6 De-quantization and 2D Inverse DCT 86
4.4.7 Output Buffer 88
4.4.8 Motion Estimation 88
4.4.9 Motion Compensation 88
4.4.10 Predictive Difference Block 88
4.4.11 Watermark generator 89
4.4.12 Watermark Embedder 89
4.4.13 Controller 90
4.4.14 Entropy Coder 90
4.5 Architecture Prototyping 92
4.5.1 MATLAB® Prototyping 92
4.5.2 MODELSIM® Simulation 94
4.5.3 FPGA Prototyping 94
4.6 Performance Analysis 96
4.6.1 Hardware Cost 96
4.6.2 Power Consumption 96
4.6.3 Processing Speed 97
4.6.4 Video Quality Degradation (PSNR analysis) 100
4.6.5 Tamper Detection 102
4.7 Comparison with Other Works 106
4.8 Summary 108

CHAPTER FIVE: IMPROVED SECURITY, LOW POWER AND LOW COST HARDWARE WATERMARKING SYSTEM FOR REAL-TIME SURVEILLANCE VIDEO AUTHENTICATION....109
5.1 Introduction....109
5.2 Contributions....110
5.3 Proposed Video Watermarking system with Improved Security....112
5.4 Attacks on digital video watermarks....112
5.4.1 Watermarking Attacks....113
5.4.2 General threat model for surveillance video watermarking....115
5.4.3 Cryptographic vulnerability of past works....119
5.5 Goals for Proposed Watermarking System....122
5.5.1 Security Goals....123
5.5.2 Implementation Goals....124
5.6 Low Power, Low Hardware CSPRNG as Secure Watermark Sequence generator....124
5.6.1 Choice of a Stream Cipher....125
5.6.2 Grain 128a: Best security and Low Hardware Cost Stream Cipher....126
5.7 Hardware Implementation....128
5.7.1 MODELSIM® Simulation....129
5.7.2 FPGA Prototyping....129
5.7.3 Hardware Implementation Result....131
5.7.4 Watermarked Video Quality Assessment....134
5.8 Proposed Watermarking system Assessment....135
5.8.1 Implementation Goals achievement....135
5.8.2 Security Goals achievement....138
5.8.3 Sample Video Detection....144
5.9 Summary....150

CHAPTER SIX: CONCLUSION....151
6.1 Summary and Achievements....151
6.2 Future Works....153

BIBLIOGRAPHY....154

DISSEMINATION....166

APPENDIX A: MATLAB SIMULATION CODES....167
A.1. Main function....167
A.2. Compression, watermarking and de-compression....171
A.3. Watermark embedding....172
A.3.1. Compression (DCT, quantization and zigzag)....174
A.3.2. Decompression (de-quantization and inverse 2D DCT)....175
A.4. Decoding watermarked video....176
A.4.1. Watermark detection and generate detection map....177
A.5. Plot PSNR curves....178

APPENDIX B: VERILOG/VHDL CODES....180

B.1. Watermarking System with MPEG encoder 180
B.1.1. Testbench 180
B.1.2. Top-module 183
B.1.3. Encoder module 187
B.1.4. Watermark embedder unit 189
B.1.5. 2D DCT and 2D Inverse DCT 189
B.2. Watermark Generator 189
B.2.1. Gollmann Cascaded Filtered Feedback with Carry Shift Register (GCF-FCSR) module 189
B.2.2. Filtered FCSR module 190
B.2.3. Test-bench for GCF-FCSR module 193
B.3. MPEG-4 Motion Estimation System 195
B.3.1. Top-module 195
B.3.2. Motion estimation controller 197
B.3.3. Motion estimation module 201
B.3.4. Controller of Three Step Search (TSS) module 203
B.3.5. Memory addressing module 206
B.3.6. Sum of absolute difference (SAD) module 212
B.3.7. Address conversion unit 214
B.3.8. Test Bench 214
B.4. Grain-128a stream cipher 216

List of Tables

Table 3.1 FPGA synthesis result for MJPEG video watermarking system implementation 71

Table 3.2 Comparison of our MJPEG video watermarking system with other published works on video watermarking 79

Table 4.1 States of the FSM of the controller unit 93

Table 4.2 FPGA synthesis result for MPEG-4 video watermarking system implementation 98

Table 4.3 Estimated initial latency and processing speed (for 320 × 240 pixels) 98

Table 4.4 Estimated watermarking speed (frame/sec) for different video resolutions at 40 MHz of clock frequency 100

Table 4.5 Comparison of our MPEG-4 video watermarking system with other published works on hardware video watermarking 107

Table 5.1 Design performance of eSTREAM stream ciphers (0.13 μm standard cell CMOS) 132

Table 5.2 FPGA synthesis result for the improved security MPEG-4 video watermarking system implementation 133

Table 5.3 Estimated processing speed (for 1280x720 pixels video) 134

Table 5.4 Comparison of our improved security MPEG-4 video watermarking system with other published hardware video watermarking works 137

Table 5.5 Effects of possible attacks on watermarked video 140

List of Figures and Illustrations

Figure 2.1 DCT transformation and dimension reduction by zigzag ordering 21

Figure 2.2 Organization of DCT coefficient matrix in the zigzag order 22

Figure 2.3 Determination of P(*i*) matrix a) Previously processed DCT block Bp(*i*) b) Currently processed DCT block Bc(*i*) c) WM(*i*) block; d) P(*i*) Block. 22

Figure 2.4 Overview of the proposed hardware video watermarking system for real-time video authentication. 25

Figure 2.5 Feedback with Carry Shift Register (FCSR) - Fibonacci configuration 31

Figure 2.6 Feedback with Carry Shift Register (FCSR) - Galois configuration..... 31

Figure 2.7 Filtered FCSR (F-FCSR)..... 33

Figure 2.8 Gollmann Cascaded F-FCSR (GCF-FCSR)..... 33

Figure 2.9 Principles of encrypting b bits with a stream cipher 39

Figure 2.10 Principles of encrypting b bits with a block cipher 40

Figure 2.11 Traditional modes of operation of block cipher: (a) ECB (b) CBC (c) CFB (d) OFB..... 42

Figure 2.12 Block cipher in counter mode..... 42

Figure 3.1 Proposed video watermarking system 48

Figure 3.2 Block diagram of hybrid DPCM/DCT coding scheme 51

Figure 3.3 Structure of the primitive watermark sequence..... 53

Figure 3.4 Block diagram of the proposed watermark generator 54

Figure 3.5 Dataflow of proposed watermarking process during MJPEG encoding process..... 57

Figure 3.6 System level architecture of the proposed watermarking system 58

Figure 3.7 Hardware architecture of MJPEG video encoder integrated with watermarking unit (F_{ori} is the original frame, F_{ref} is the reference frame) 60

Figure 3.8 System level architecture of the watermark generator unit 62

Figure 3.9 Hardware architecture of the watermark embedder unit 63

Figure 3.10 States of Finite State Machine (FSM) of controller unit of the designed video watermarking system with integrated MJPEG encoder system 65

Figure 3.11 FPGA based implementation of the proposed video watermarking system.. 68

Figure 3.12 Examples of watermarked video streams (a) Original video frame (b) MJPEG video frame (c) Watermarked video frame. 68

Figure 3.13 PSNR comparisons for static scene 70

Figure 3.14 PSNR comparisons for dynamic scene 70

Figure 3.15 Timing diagram for the processing (watermarking and compression) of a single 256 × 256 pixels video frame (I frame). MB stands for macro-block of 8 × 8 pixels. 73

Figure 3.16 Tamper detection of video sample-1: (a) Original video (b) Watermarked video (c) Tampered video (segment removal) (d) Detection map 76

Figure 3.17 Tamper detection of video sample-2: (a) Original video (b) Watermarked video (c) Tampered video (cover-up attack) (d) Detection map 76

Figure 4.1 System level hardware architecture of proposed MPEG-4 video watermarking system for CMOS image sensor based video camera 87

Figure 4.2 System level architecture of watermark generator unit 90

Figure 4.3 System level architecture of watermark embedder unit 91

Figure 4.4 Finite State Machine (FSM) of the controller unit 92

Figure 4.5 Design flow of the architecture prototyping 95

Figure 4.6 PSNR comparison for MPEG-4 encoded (without watermark embedded) video vs original video sequence 102

Figure 4.7 PSNR comparison for MPEG-4 encoded (with watermark) video vs original video sequence 103

Figure 4.8 Tamper Detection of Cover-up attack: (a) Original video (b) Watermarked and MPEG-4 encoded video frame (c) Cover-up attack on watermarked and MPEG-4 encoded frame (d) Detection map. Sample video resolution is 320x240 pixels 104

Figure 4.9 Tamper Detection of Segment- Removal attack: (a) Original video (b) Watermarked and MPEG-4 encoded video frame (c) Segment-removal attack on watermarked and MPEG-4 encoded frame (d) Detection map. Sample video resolution is 320x240 pixels 105

Figure 4.10 Tamper Detection of the blurring effect: (a) Original video (b) Watermarked and MPEG-4 encoded video frame (c) Blurring on watermarked and MPEG-4 encoded frame (shown by the red rectangle) (d) Detection map. Sample video resolution is 320x240 pixels 105

Figure 5.1 Proposed watermarking system with improved security..... 113

Figure 5.2 An overview of Grain-128a stream cipher's key-stream generator 128

Figure 5.3 Hardware architecture of the video watermarking system (watermark embedder and watermark generator)..... 130

Figure 5.4 PSNR comparison between watermarked video and un-watermarked MPEG-4 video 136

Figure 5.5 Detection of segment-removal on a 960 × 720 pixels resolution video sample: (a) Original video (b) Watermarked and MPEG-4 encoded Video frame (c) Watermarked MPEG-4 Video frame is tampered (d) Tamper Detection Map . 146

Figure 5.6 Detection of Cover-up attack on a 960 × 720 pixels resolution video sample: (a) Original video (b) Watermarked and MPEG-4 encoded Video frame (c) watermarked MPEG-4 video frame is tampered (d) Tamper Detection Map ... 147

Figure 5.7 Detection of sharpening and blurring effect on a 960 × 720 pixels resolution video sample: (a) Original video (b) Watermarked and MPEG-4 encoded Video frame (c) tampered by sharpening and blurring (shown in red rectangle) (d) Tamper Detection Map..... 148

Figure 5.8 a) original video frame, b) watermarked video (quantization with quality level Q = 50), c) Compressed after watermarking with quality level, Q = 90, d) Detection map does not detect any tampering, e) Compressed after watermarking with quality level, Q = 10, f) Detection map detects tampering..... 149

List of Abbreviations

Abbreviation	Definition
WM	Watermark
RNG	Random number generator
TRNG	True random number generator
PRNG	Pseudorandom number generator
CSPRNG	Cryptographically secure pseudorandom number generator
SR	Shift register
LFSR	Linear feedback shift registers
FCSR	Feedback with carry shift register
F-FCSR	Filtered feedback with carry shift register
GCF-FCSR	Gollmann cascaded filtered feedback with carry shift register
AES	Advanced Encryption Standard
DES	Data Encryption Standard
NIST	National Institue of Standards and Technology
FIPS	Federal Information Processing Standard
CFB	Counter Feedback
OFB	Output Feedback
CTR	Counter
JPEG	Joint Photographic Experts Group
MJPEG	Motion Joint Photographic Experts Group
MPEG	Moving Picture Experts Group
PSNR	Peak Signal to Noise Ratio
MSE	Mean Squared Error

Chapter One: **Introduction**

In the past, duplicating art work was quite complicated and required a high level of expertise for the counterfeit to look like the original. However, in the digital world this is not true anymore. A fast migration has occurred from analog media to digital media during the recent two decades. In this process analog cameras have been replaced by digital cameras. The digital media has introduced a lot of new features for copying, editing and storage of multimedia contents. Sharing and storage of digital video data has become much easier and faster due to the advances in electronics and rapid growth of digital signal and multimedia processing. Unfortunately, these progress are also associated with various security threats and attacks i.e. image or video copying, tampering, ownership theft, unauthorised playback etc. [1]-[4]. Using widely available editing software tools a digital video sequence can be easily altered or manipulated. These attacks can be performed in the twinkling of an eye using various digital devices. This issue becomes more significant when the particular video sequence has to be used as evidence. In such a case we need to prove that video data is original and reliable. Hence the authentication of digital media information is an important matter of concern. In order to maintain authenticity, integrity and security of digital video contents authentication systems are necessary.

Digital watermarking, a data hiding technique, is considered as one of the key authentication methods [1]-[5]. Today, digital video watermarking techniques are widely used in various imager or video applications such as copyright protection, data integrity, copy control, content authentication, etc. [1]-[7]. Digital watermark is used in

fingerprinting to track back a malicious user. Watermark is also used in a copy control system with watermarking capability to prevent unauthorized copying [2], [5]. In copyright protection, digital watermark can protect the ownership information of the contents. For video authentication, watermarking can ensure that the original content has not been altered or manipulated

Because of its potential applications, current digital watermarking techniques have focused on multimedia data and in particular on video content. Over the past few years, researchers have investigated the embedding process of visible or invisible digital watermarks into raw digital video [7], uncompressed digital video both on software [7]-[11] and hardware platforms [12]-[15]. Most of these works are on watermarking for the copyright protection application. This dissertation focuses on a secure and low power hardware based digital video watermarking system for real-time authentication of surveillance video stream.

1.1 Problem Statement

Surveillance cameras are video cameras used for the purpose of observing an area or facility. They are often connected to a recording device or IP network, and may be watched and analysed by a security guard or law enforcement officer using video analysis software. The security cameras are usually installed in such a place which can be assumed as unreachable to the general people. Any sort of tampering with the camera devices will set the alarm on and notify the law enforcement officer to take immediate action. The surveillance cameras and the video footage monitoring and control rooms are out of reach from the attackers. But the link between the camera and the control room is

vulnerable. This link can be wireless or wired depending on the setup and the facility where it is being used. The attacker may intercept this link and tamper with the video sequences transmitted over the link. Therefore this link is the main concern for the security threats. Thus, we need real-time authentication of the captured video stream in the surveillance cameras. Moreover, real-time authentication can be beneficial in forensic applications as well. The video streams and images captured by a forensic investigation officer or an under-cover security agent on field may also require real-time authentication to prove the integrity of the evidence in court.

Today, digital watermarking is a commonly used authentication technique for multimedia information. Software implementation of watermarking techniques is not a difficult task. However, for the above mentioned problem, where we need real-time authentication of the captured video stream, a hardware based video watermarking system integrated within a surveillance video camera to enable the authentication of the captured video stream in real-time is the most suitable solution. The basic idea is to watermark the captured video footage within the surveillance camera during the encoding process before the transmission of the video stream from the camera unit. The watermarked video stream will be transmitted to the control room where the detection algorithm will be run on the received video stream using a software program to detect any alteration or manipulation on the watermarked video stream. In addition, such a hardware video watermarking system inside a video camera, used in on-field investigations or secret missions lead by security agents, may also be used to enable real-time authentication of the captured video streams.

There are some potential challenges in designing a hardware video watermarking system for real-time authentication. The hardware watermarking system itself should be secure enough so that it cannot be compromised by the hacker within the life time of the particular security camera, inside which the watermarking system will be integrated. The security of the watermarking system is dependent on the cryptographic properties of the watermark sequence and the resilience of the watermark embedding algorithm itself. Apart from the security, the watermarking system should also maintain some implementation constraints such as low power consumption, small area and low hardware usage. Moreover, to incorporate the video watermarking system with the contemporary and commonly used MPEG – 4 encoding standard the watermarking system should also maintain a high throughput (watermarking rate) to watermark the standard resolution video in real-time, which is another challenge.

1.2 Objective

The objective of this dissertation is to present the design and implementation of a secure, low power and low cost hardware based video watermarking system which is able to insert an invisible and semi-fragile watermark into the video stream, captured by a surveillance video camera. The designed watermarking system should be of low power consumption, low hardware cost and small in size to be integrated with the encoder system within a CMOS image sensor based video camera system. These properties are desirable in order to enable the real-time authentication of the captured video stream without significant increase in the hardware resources and power consumption. It should also be secure enough to protest all kinds of security threats such as potential

cryptographic attacks on the watermark sequence or watermarking attacks on the watermarked video stream.

As a proof of the design concept, the hardware video watermarking system is designed using Verilog Hardware language and then implemented and tested on an ALTERA® Cyclone FPGA device.

The main objectives of this thesis are:

1. To present an efficient design and implementation of a hardware digital video watermarking system which can be integrated with an MPEG-4 video encoding system for the authentication of video stream captured by CMOS image sensor based surveillance cameras.
2. To verify that the designed watermarking system works efficiently without significant increase in overall system power consumption and hardware utilization.
3. To ensure that the watermarking system is suitable for real time video watermarking maintaining sufficient throughput (watermarking rate), which can be easily incorporated with the state-of-the-art MPEG-4 video standards. To prove that the watermarking system solution also features insignificant video quality degradation due to the watermarking process.
4. To show that the watermarking system enables the real-time detection of certain potential attacks, i.e. cover-up attack, cropping, segment removal, sharpening, blurring etc. on the watermarked video stream and at the same time is capable to allow certain legitimate modification like mild lossy compression to enable the

storage of the captured video stream in compressed format for using as forensic evidence later.

5. To make sure that the watermarking system is secure against known cryptographic attacks so that the watermarking system cannot be compromised by the attacker even with a modern computer.

1.3 Thesis Organization

This thesis is organized in six separate chapters.

In **Chapter 1**, the motivation and objective of the thesis is presented with the overview of the proposed video watermarking system for real-time surveillance video authentication.

Following is **Chapter 2**, where background and past works on hardware based video watermarking are summarized. This chapter also presents the watermarking algorithm selected for the implementation in our hardware video watermarking system for surveillance video stream authentication.

To achieve the objectives of this thesis the entire work was divided into three different parts. These parts are described in **Chapter 3**, **Chapter 4** and **Chapter 5** respectively.

Chapter 3 presents the design and implementation of a digital video watermarking system integrated with a non-standard MJPEG encoding system to prove the concept of real-time video watermarking using the designed hardware watermarking system during the video encoding process. The performance and resource utilization of the designed system was also compared with other research works on the similar field.

After proving the concept, it is important to verify that the designed concept works with the state-of-the-art. Thus, we designed the video watermarking system to integrate with the state-of-the-art MPEG-4 encoding system. **Chapter 4** presents the design and implementation of the proposed video watermarking system integrated with an MPEG-4 encoder system, which features motion estimation and motion compensation based prediction of video frames to provide advanced and better compression than the MJPEG encoding system does. It was verified that the designed system can be incorporated with the contemporary and commonly used MPEG-4 encoding system without significant increase in power dissipation and logic gate requirements.

As mentioned before, a video watermarking system intended for surveillance video authentication should itself be secure enough. In **Chapter 5** specific security goals are set in addition to the implementation goals for the proposed video watermarking system to ensure the required resilience of the video watermarking system considering the surveillance video authentication in mind. A standardized, efficient and secure stream cipher was chosen as a source of cryptographically secure pseudorandom watermark sequence generator to meet up the security demands for the desired video watermarking system. A general threat model for a video watermarking system for real-time surveillance video application is also proposed here. Finally, the improved design of the video watermarking system with the secure watermark generator was integrated with a commonly used MPEG-4 video encoder and implemented on an ALTERA® Cyclone FPGA. The designed system was assessed on the basis of the implementation constraints and on different recognized watermarking attacks and proposed threat model to ensure the achievement of security goals.

Chapter 6 concludes the thesis by summarizing the overall works and also proposes some future works which can be performed with the same concept and application.

Chapter Two: **Background**

2.1 Digital Watermarking

Digital Watermarking is the process of embedding an additional, identifying information within a host multimedia object, such as text, audio, image or video. By adding a digital watermark to the multimedia content, it is possible to detect hostile alterations as well as to verify the integrity and the ownership of the digital media.

2.2 Types of Watermarking

Watermarking techniques can be divided into different categories according to various criterions [16].

2.2.1 Processing Domain

According to the domain in which video watermarking is performed, watermarking processing methods can be classified into two categories: spatial domain and frequency domain [1]. In the spatial domain, directly applying minor changes to the values of the pixels in a minor way is mainly used. This technique makes the embedded information hardly noticeable to the human eye. For example, pseudo-random watermarking works by a simple addition of a small amplitude pseudo-noise signal to the original media data. In the frequency domain, the object first goes through a certain transformation, DCT or discrete wavelet transforms (DWT), the watermark is embedded in the transform coefficients and then it is inversely transformed to receive the watermarked data. The frequency domain methods are more robust than the spatial domain techniques [17].

2.2.2 Robustness Level

According to the level of robustness, watermarking methods can be categorized into three main types: Fragile, Semi-fragile and Robust.

A *fragile* watermark is one which enables the detection of all kinds of modifications in the watermarked image. For image integrity applications, fragile watermarks are commonly used so that even the slightest change in the image can be detected. Conversely, a watermark is called *robust* if it can resist any sort of transformations in the watermarked image. Robust watermarks are designed to resist attempts to remove or destroy the watermark from the image. Robust watermarks are typically used for copyright protection [5], [18]-[20].

A *semi-fragile* watermark allows certain legitimate modifications like mild lossy compression of the watermarked images, but does not allow malicious transformations like cropping, segment removal, cover-up, blurring, sharpening etc. For the authentication of surveillance video footage, semi-fragile watermarking can be used [5], [18]-[20].

2.2.3 Human Perception

According to human perception, digital watermarking can be categorized into two different types: visible and invisible. Different applications have different requirements. Sometimes a certain application requires a watermark to be visible, so that the embedded watermark appears visible to a casual viewer. Visible watermarks can be used in applications like copyright protection, broadcaster logo, etc. On the other hand, applications like content authentication require invisible watermarks [5].

2.3 Types of Watermarking Implementation

Two kinds of watermark implementations exist: software watermarking and hardware watermarking. A watermarking system can be implemented on either software or hardware platforms, or some combinations of the two.

In software implementation, the watermarking scheme can simply be implemented in a PC environment. In order to embed the watermark in software, the image or video contents have to be captured by the camera and stored in an external memory. The watermarking algorithm's operations can be performed as machine code software running on an embedded processor. By programming the code and making use of available software tools, it can be easy to design and implement any watermarking algorithm at various level of complexity. Over the last decade, numerous software implementations of watermarking algorithms for relatively low data rate signals (like audio and image data) have been invented [7]-[11].

While the software approach has the advantage of flexibility, computational limitations may arise when attempting to utilize these watermarking methods for video signals or in portable devices. For real time solutions hardware (HW) implementations are more reliable. These enable the addition of watermarks during the read-out phase of the digital camera. The image or video contents will already have a certain level of security. Therefore, there is a strong incentive to apply hardware-based implementation for real-time watermarking of video streams [12]. The hardware-level design offers several distinct advantages over the software implementation in terms of low power consumption, reduced area and reliability. It enables the addition of a tiny, fast and potentially cheap watermark embedder as a part of portable consumer electronic devices.

Such devices can be a digital camera, camcorder or other multimedia devices, where the multimedia data are watermarked at the origin. On the other hand, hardware implementations of watermarking techniques require flexibility in the implementation of both computational and design complexity.

2.4 Requirements for Surveillance Video Watermarking

Different watermarking applications have specific requirements. Therefore, there is no universal requirement to be satisfied by all watermarking techniques. According to *Shoshan et al.* [18] the requirements for video watermarking depend on some criteria such as visibility, transparency, robustness level, security, complexity, throughput, etc. Real-time surveillance video watermarking demands some specific requirements according to these criteria. These are mentioned below:

1. ***Invisibility:*** Watermark should be imperceptible and invisible to a human observer.
2. ***Transparency:*** Watermark insertion should not degrade the quality of the underlying video stream. The *Peak Signal to noise Ratio* (PSNR) of the watermarked video stream should be high enough without any significant visual artifacts.
3. ***Semi-fragile***: Watermarking for the authentication of surveillance video should be semi-fragile because semi-fragile watermarking enables the detection of malicious transformations like cropping, segment removal, cover-up, blurring, sharpening etc. on the watermarked video streams but allows certain modifications such as mild lossy compression. In surveillance video footage mild compression should

be allowed because the authorized person may want to store the captured video footage in compressed format for using as evidence in court.

4. ***Security***: The watermarking system must be truly secure so that knowing the exact algorithms for embedding and detecting the watermark does not help an unauthorized party to extract or estimate the inserted watermark into the video stream captured by the surveillance video camera. The watermark cannot be added or removed by an unauthorized user. It is very important that to make sure that the watermarking system itself is secure against all possible cryptographic attacks so that it cannot be compromised by the attacker even with a modern computer.
5. ***Oblivious***: The watermark algorithm should be oblivious. That implies it should be possible for the authorized person to regenerate the watermark information for running the detection algorithm on the received watermarked video stream in the control room without knowing the original multimedia data captured by the surveillance camera.
6. ***Low complexity***: The watermark embedding and extracting should have low computational complexity. As the watermarking system is intended to be integrated within a surveillance video camera the watermarking algorithm should be simple enough to allow the implementation in hardware platform easily without significant increase in hardware resource and power consumption.
7. ***Compressed domain processing***: It should be possible to incorporate the watermark into compressed video stream. This will allow the watermarking

system to be integrated with the encoder system inside the camera to watermark the video stream during the encoding process in real-time.

8. ***High Throughput and Constant bit-rate***: The watermark should not increase the size of the compressed host video data and the throughput of the watermarking system should be high enough to adapt with the state-of-the-art video encoding process and to allow high resolution video watermarking maintaining standard video frame rate.

2.5 Reasons for Semi-fragile Watermarking

Surveillance video camera footage is used to monitor the live activities of the facility covered by the video camera and the captured video may also be used later as evidence. Hence, the integrity and originality of the captured video is very important. Potential manipulations such as cropping, removal of segments, cover-up attack, blurring and sharpening on the original image or video will harm the surveillance video footage since this will destroy the original information from the captured video or image. These may fool the monitoring person in the control room or may question the integrity of the evidence to be used later. Therefore, these sorts of attacks should not be allowed in surveillance video footage.

However, mild compression should be allowed on surveillance video footage, because for using as evidence in investigation procedure, the authorized person may want to store the captured video footage in compressed format. Moreover, mild compression of the transmitted video footage will not serve the purpose of the attackers because this will

not hide any information from the original video stream. The monitoring person in the control room will still be able to understand what is going on in the scenario.

Thus, in authenticating the surveillance video footage semi-fragile watermarking is the best choice since semi-fragile watermark enables the detection of malicious transformations like cropping, segment removal, cover-up attack, sharpening, blurring etc. but allows certain legitimate modification like mild compression of the watermarked images or video stream.

Semi-fragile watermarking scheme chosen for this work is embedded in the frequency domain, yielding some advantages. In practical video storage and distribution systems, video sequences are stored and transmitted in a compressed format and during compression the image is transformed from the spatial domain to the frequency domain. Working on a compressed rather than an uncompressed video stream will be beneficial for video authentication application in terms of less computation, low power and low hardware requirements. Therefore, semi-fragile watermark embedding and detecting directly in the compressed video stream in frequency domain can minimize computationally demanding operations and optimize the hardware system implementation.

2.6 Past Works on Hardware Based Video Watermarking

Contrary to still image watermarking techniques, new problems and new challenges have emerged in video watermarking applications. *Shoshan et al.* [1] and *Li et al.* [5] presented an overview of the various existing video watermarking techniques and showed their features and specific requirements, possible applications, benefits and

drawbacks. In the past few years, a great deal of research effort has been focused on efficient watermarking systems implementation using hardware platforms. They vary in terms of applications and implementation platforms.

For example, *Strycker et al.* [13] proposed a well-known video watermarking scheme, called *Just Another Watermarking System* (JAWS), for television (TV) broadcast monitoring and implemented the system on a Philips's Trimedia TM-1000 Very Long Instruction Word (VLIW) processor. The experimental results proved the feasibility of watermarking in a professional TV broadcast monitoring system.

Mathai et al. [14]-[15] presented an Application Specific Integrated Circuits (ASIC) implementation of the JAWS watermarking algorithm using 1.8V, 0.18μm complementary metal oxide semiconductor (CMOS) technology for real-time video stream embedding. With a core area of 3.53 mm^2 and operating frequency of 75MHz, that chip implemented watermarking of raw digital video stream at a peak pixel rate of over 3 Mpixels/s while consuming only 60 mW power.

Maes et al. [21] presented the Millennium watermarking system for copyright protection of DVD Video and some specific issues such as watermark detector location and copy generation control were also addressed in their work.

Mohanty et al. [22] presented a concept of secure digital camera (SDC) with a built-in invisible-robust watermarking and encryption facility. Also another watermarking algorithm and corresponding VLSI architecture that inserts a broadcasters logo (a visible watermark) into video streams in real-time was presented [23] by the same group.

In [24] *Jeong et al.* presented an FPGA prototype for HAAR-wavelet based real time video watermarking for real-time based copyright protection system which can be useful in movie production and broadcasting companies.

A new real-time watermarking VLSI architecture for spatial and transform domain was presented by *Tsai et al.* [16]. A real-time video watermarking system using DSP and VLIW processors was presented that embeds the watermark using fractal approximation by *Petitjean et al.* in [25].

In general, digital watermarking techniques proposed so far for media authentication are usually designed to be visible or invisible-robust or invisible-fragile watermarks according to the level of required robustness [4]-[5]. Each of the schemes is equally important due to its unique applications.

There have been very few works on hardware based video watermarking for the surveillance application. In [26] *Schyndel et al.* presented a watermarking system based on spread spectrum watermarking generated by the array construction method. They focused on surveillance video watermarking.

Our group has been working on the hardware architecture of a semi-fragile and invisible watermarking system for a few years. In our previous works, hardware architecture of a watermarking system was designed and implemented for still image authentication based on JPEG encoding process by *Shoshan et al.* [18], [27]. Later on *Roy et al.* reported the design and implementation of hardware based invisible and semi-fragile video watermarking system for video authentication in [28]-[29], where non-standard MJPEG and state-of-the-art MPEG-4 encoding process were chosen respectively. In [30] *Roy et al.* reported the hardware based watermarking solutions for

CMOS based image sensors (for image and video). Starting from the JPEG based watermarking design, through an MJPEG one, the watermarking solution for the commonly used MPEG encoded video was also reported there. In all these works, we concentrated on the hardware implementation of an invisible semi-fragile watermarking system for authentication of surveillance camera footage. Our watermarking scheme is based on the DCT domain and optimized for hardware implementation.

In this thesis, however, we focus on designing a hardware based invisible and semi-fragile watermarking system with improved security that still maintains low power and low hardware utilization for real-time surveillance video authentication. The motivation here is to integrate the video watermarking system within a surveillance video camera for real time watermarking in the source end. We propose, for the first time, a general threat model for a video watermarking system for real-time surveillance video authentication. The designed video watermarking system was assessed on the basis of different recognized watermarking attacks and the given threat model to provide evidence for the achievement of security goals. Our watermarking system requires fewer hardware resources, has higher processing speed and improved security when compared to other's work in the past.

2.7 Watermarking Algorithm for current work

The watermark algorithm used in our work was originally designed by *Shoshan et al.* in [18]. After simple adjustments this algorithm was used in our works [28]-[30]. The developed video watermarking system demonstrated in this thesis provides better security with small increase in system power consumption and hardware resources when

compared to other's work in the past [14]-[26]. Grain – 128a, a standardized, efficient and secure stream cipher, has been chosen as a source of cryptographically secure pseudorandom watermark sequence generator to meet the security demands for this secure video watermarking system. The watermarking algorithm is Discrete Cosine Transform (DCT) based and the type of watermark being used here is invisible and semi-fragile; this algorithm will be used in authentication process of surveillance video footage. The algorithm is briefly summarized below:

1. The original image (each frame in case of a video sequence) is divided into 8×8 blocks, the DCT coefficients are determined for each 8×8 block and the resulting 8×8 DCT values are reorganized into a 1×64 block following a zigzag order (Figure 2.1 and Figure 2.2).
2. According to the algorithm, a certain number of cells (N) in each 8×8 DCT block are modified. As the number of cells to be watermarked (N) in an 8×8 block increases, the robustness of the watermarking also increases. On the other hand, such an increase reduces the video frame quality because of the reduction in the originality of the video frame. For our application we will change only two cells from each 8x8 block (N = 2).
3. The values of the currently processed DCT block (B_C) are considered along with the values of its adjacent previously processed DCT block (B_P) and the watermark data block (W_M), in order to choose which cells are to be modified and in what way. A secret pseudo-random sequence, serves as the watermark data.

4. A binary matrix P(i) is constructed based on the following AND operation (equation 2.1):

$$\boldsymbol{P(i) = W(B_c, i).B_c(i).B_P(i).} \qquad \textbf{(2.1)}$$

Here, i stands for the position of the DCT coefficient according to the zigzag order; $B_C(i)$ and $B_P(i)$ are the DCT data values of two adjacent blocks B_C and B_P respectively and $W_M(B_C,i)$ is the watermark bit (generated by a pseudorandom number generator).

5. Assuming N = 2 and following the zigzag order, the first two non-zero cells (blackened in Figure 2.3) in the matrix P indicate the two cells or two coefficients which are going to be modified in B_P. The indices of the coefficients to be modified are stored in $P_{indC}(l)$. Here, l = 1, … N(=2). Thus we get N(=2) mid-frequency coefficients in the B_C block and those are the cells that will be modified. Modification is done only in the LSB of those two coefficients.

6. Following the zigzag order, the cells are alternately divided into N (=2) groups such that the first cell belongs to the first group, the next cell to the second group and so forth until the DCT coefficient C_0 (DC coefficient) is reached. Two groups are marked by different backgrounds in Fig. 2(d). The bits in each group are XOR'ed to determine the value of the register $P_{valC}(l)$ (Equation (2.2).

$$\boldsymbol{P_{valC}(l) = \underset{i = l, l+N, l+2N}{\overset{i < P_{indC}(N)}{XOR}} (P(i)).} \qquad \textbf{(2.2)}$$

7. The LSB of the coefficients $B_P(P_{indC}(l))$ will be flipped if the value of $P_{valC}(l)$ is 1, otherwise it remains same.

8. The detection procedure is similar to that of the embedding procedure. The input is the watermarked image or frame in compressed format. The matrix P is created in the same manner as for the embedding process. The modified cells are identified and instead of modifying the cells, a comparison is made to verify that they are indeed equal to the expected modified value. Taking advantage of the software platform, additional processing can aid detection.

Each color video frames is converted from RGB to YC_bC_r color-space. Y stands for Luminance (Brightness) and C_b and Cr are the two chrominance components. During the watermarking of individual color video frames, only Y components are taken into account. Moreover, in watermarking video stream, only the reference (I) frame will be watermarked in a group of pictures (GOPs) but not the following P frames, since P-frames will be predicted from the reference (I) frame of the corresponding GOPs.

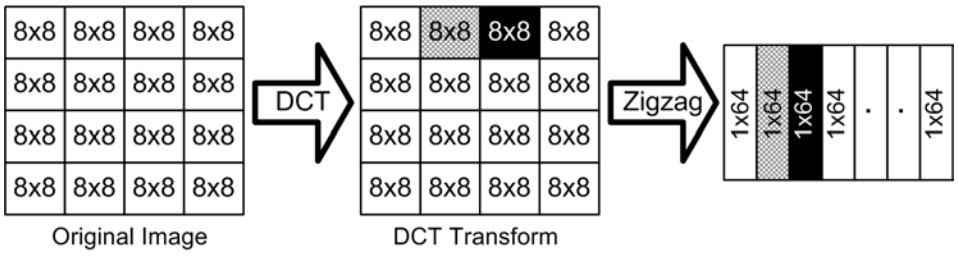


Figure 2.1 DCT transformation and dimension reduction by zigzag ordering

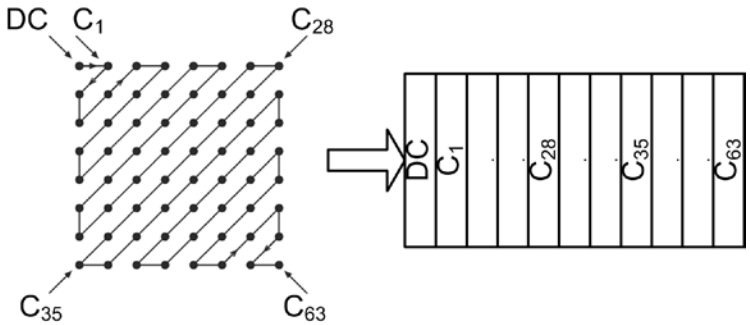


Figure 2.2 Organization of DCT coefficient matrix in the zigzag order

301	-1	4	0	3	-3	0	-1
19	1	-4	3	0	0	0	-1
7	9	1	-4	2	0	0	1
19	-6	-8	5	-1	-1	0	0
-6	3	0	-2	-1	0	0	1
6	-5	0	2	1	1	0	0
-5	0	1	1	0	-2	0	0
1	-1	0	0	-2	0	0	0

(a) B_P

299	-1	-6	2	5	-3	0	0
5	5	-7	-3	0	1	0	0
-26	11	2	-4	-2	1	3	1
-8	-4	-8	-5	0	0	1	0
-1	3	6	1	0	0	0	0
6	1	2	1	1	0	0	0
2	0	0	-1	1	0	0	0
1	2	-1	1	1	1	1	0

(b) B_C

0	0	1	0	0	0	1	1
0	1	0	0	0	1	1	1
0	1	0	0	1	1	1	1
1	1	1	0	0	1	1	1
1	1	1	0	0	1	1	1
1	0	0	0	0	0	0	1
1	0	0	1	0	0	1	1
1	0	0	1	0	0	1	1

(c) WM Sequence

0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	0	1	0	0	1
1	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0

(d) P Block

Figure 2.3 Determination of P(*i*) matrix a) Previously processed DCT block Bp(*i*) b) Currently processed DCT block Bc(*i*) c) WM(*i*) block; d) P(*i*) Block.

2.8 Video Encoding and Watermarking

Raw video frames captured by a modern digital video camera contain a vast amount of information (data bits) which may require high memory consumption for storage or high bandwidth to transmit. Hence the captured image or video frames are compressed using various encoding methods before the transmission or storage. In image compression only spatial redundancy within the image is reduced by encoding process whereas in video compression, spatial redundancy within a single frame and temporal redundancy among consequent frames are reduced using different encoding methods.

Today, nearly all commonly used video compression methods (encoding standards) apply discrete cosine transform (DCT) for spatial redundancy reduction. Watermarking algorithm adapted for our current work also inserts the watermarking information into the original image or video in DCT domain. Since, an encoding system itself requires a remarkable amount of hardware resources and power consumption within the whole camera system it will be good idea to integrate the watermarking system with the encoding system of the video camera. This will allow the watermarking system to fit in the overall system successfully without a significant increase in the amount of hardware resources, size of the system and power consumption. In this way, integrating the watermarking system within the surveillance video cameras can be beneficial.

JPEG is a state-of-the-art encoding standard for still image compression and Motion JPEG is a nonstandard method of compressing a sequence of video frames using JPEG [31]. MJPEG is popular for applications such as video capture, PC-based video editing and security surveillance. Despite the fact that MJPEG provides less compression

than MPEG encoding standards, it is easy to implement MJPEG format in hardware with relatively low computational complexity and low power dissipation. This is the reason that was chosen in our previous work of designing hardware watermarking system for video authentication [28].

MPEG developed several MPEG-x video encoding standards i.e. MPEG-1, MPEG-2, MPEG-4, MPEG-7, etc. for coding video and audio. But today, MPEG-4 is widely used for communication and storage of digital video [32]. MPEG-4 has rapidly become one of the mainstream exchangeable video formats in the Internet today because it has high and flexible compression rate, low bit rate, and higher efficiency while providing superior visual quality. Microsoft, Real Networks, and Apple support the MPEG-4 standard and have already embedded MPEG-4 decoders into many of their products [32], [33]. This gave the motivation to consider MPEG-4 as the target video compression method in this work of secure, low power and low cost hardware video watermarking system.

2.9 Overview of the Proposed Video Watermarking System for Real-time Authentication

The overview of the proposed secure, low power and low hardware cost video watermarking scheme is shown in Figure 2.4. It consists of a CMOS image sensor based Video Camera integrated with the proposed video watermarking module and the MPEG-4 encoder inside it. The watermark is inserted during the encoding process. The watermarked and encoded video stream is transmitted through the link to the control room for monitoring the surveillance video footage. A secret key is utilized to initialize a

pseudorandom sequence generator to generate a cryptographically secure and pseudorandom watermark sequence in the watermarking module. This secret key is a private key chosen by the manufacturer of the camera and is shared only with the authorized person responsible for monitoring and authenticating the camera footage. In the control room the watermark sequence is regenerated using the secret key. The received video stream is decoded and analyzed by running the detection algorithm using the regenerated watermark sequence. The detection algorithm is similar to the watermarking algorithm. A detection map is generated in the process showing the detected tampered blocks in the watermarked video stream.

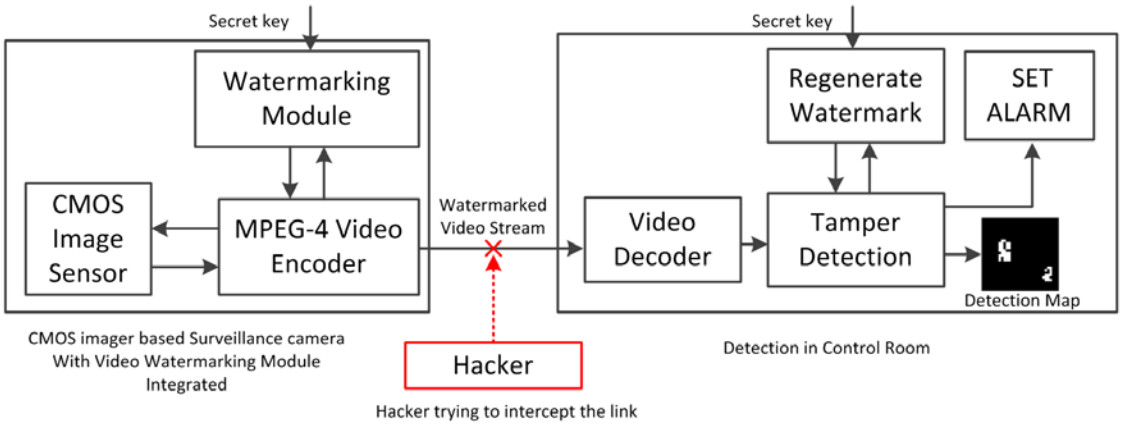


Figure 2.4 Overview of the proposed hardware video watermarking system for real-time video authentication.

2.10 Why CMOS Image Sensor Based Camera?

This work focuses on the design and implementation of a hardware watermarking system for real-time authentication of surveillance video footage. The designed watermarking system is intended to be integrated with the MPEG-4 encoder system inside the camera, satisfying the constraints of small area, low power consumption and low hardware cost. Recent developments of CMOS image sensors feature on-chip integration of different components on the same die of the image sensor with low power consumption, low cost and small area. Therefore it will be possible to integrate the watermark system on the same die with the image sensor and image or video encoder inside the CMOS image sensor based camera.

2.11 Background on Random Number Generator

A random number generator (RNG) is a fundamental primitive in cryptographic and security related systems. An RNG uses a nondeterministic source (i.e. entropy source) along with some processing function (i.e. entropy distillation process) to produce randomness. However, some physical sources (i.e. date/time vectors) are quite predictable. In addition, the production of high quality random numbers may be too time consuming, which makes it undesirable when a large number of random numbers are needed. To produce large quantities of random sequences, pseudorandom number generators (PRNGs) may be preferable [34]. Random number generators fall into three categories:

1. True Random Number Generator (TRNG)

2. Pseudo Random Number Generator (PRNG)
3. Cryptographically Secure Random Number Generator (CSPRNG)

2.11.1 True Random Number Generator

True random number generators (TRNGs) produce random data based on unpredictable physical sources, such as electrical resistor noise, or oscillator phase noise [35]. They are characterized by the fact that their output cannot be reproduced. For instance, if we flip a coin 100 times and record the resulting sequence of 100 bits, it will be virtually impossible for anyone on earth to generate the same 100 bit sequence. The chance of success is $1/2^{100}$, which is an extremely small probability. Another characteristic of a TRNG is unpredictability of the output sequence e.g. after observing 100 coin flips, the chance of guessing the 101st is still 50%. TRNGs are based on physical processes. Examples include coin flipping, rolling of dice, semiconductor noise, clock jitter in digital circuits and radioactive decay [36].

2.11.2 Pseudorandom Number Generator (PRNG)

A Pseudorandom Number Generators (PRNGs) produce a pseudorandom sequence with suitable period and statistical quality. In general a good random sequence should have low cross correlation values between different samples and no repeating patterns. PRNGs use a deterministic method to generate random numbers, employing a (truly random) seed to initialize the algorithm. The output sequences are computed from an initial seed value. Often they are computed recursively in the following way:

s_0 = seed

$s_{i+1} = f(s_i)$, $i = 0,1, \ldots$

A generalization of this are generators of the form $s_{i+1} = f(s_i, s_{i-1}, \ldots, s_{i-t})$, where t is a fixed integer. A well-known example is the linear congruential generator, described by

s_0 = seed

$s_{i+1} \equiv as_i + b \bmod m, \quad i = 0,1, \ldots$

where, a, b, m are integer constants. Note that PRNGs are not random in a true sense because they are completely deterministic. A widely used example is the *rand()* function used in ANSI C. It has the parameters:

$s_0 = 12345$

$s_{i+1} \equiv 1103515245s_i + 12345 \bmod 231$, $i = 0,1, \ldots$

A common requirement of PRNGs is that they possess good statistical properties, meaning their output approximates a sequence of true random numbers. There are many statistical tests, which can verify the statistical behavior of PRNG sequences [34], [37], [38].

In cryptography, sometimes a pseudorandom sequence is used to encode data into ciphertext (i.e. in stream cipher). However, there are many applications for pseudorandom numbers outside cryptography. For instance, many types of simulations or testing, e.g., of software or of VLSI chips, need pseudorandom data as input.

In generating watermarks for still images or video frames, a pseudorandom sequence is a suitable choice. True random sequences are not considered here, because they cannot be regenerated; in a watermarking application the same random sequence is required during the detection process. In addition, it is much easier computationally to

produce a pseudorandom sequence by a finite state machine (FSM). Although the sequence generated by the FSM will be periodic eventually, if the period of sequence is long enough then it can be considered as a pseudorandom sequence for digital watermarking application.

There are a number of methods to produce pseudorandom sequences and a variety of PRNGs have been proposed in [39]-[41]. For hardware digital watermarking system we are concerned about hardware implemented pseudorandom number generators to generate a pseudorandom watermark sequence. Shift register (SR) based PRNGs provide a fast method for generating a wide variety of pseudorandom sequences [39]-[41]. A few such PRNGs are described next, that have been used for watermarking in the past due to their efficiency in hardware. However, as we will discuss later, pseudo randomness is not enough for a secure watermarking method; additional cryptographic properties are also required. It has been shown that none of the following shift register based PRNGs satisfy these properties, but we nevertheless describe them briefly for convenience.

Linear Feedback Shift Register (LFSR)

Linear feedback shift registers (LFSR) are the simplest shift register based PRNG, used to design fast random generators. However, they have some limiting properties which make them useless for digital watermarking. Even if the feedback scheme of the LFSR is unknown it can be recovered from only *2n* output bits of an LFSR of length *n* by using the highly efficient Berlekamp-Massey algorithm [41]-[42], [87].

Feedback with Carry Shift Register (FCSR)

An FCSR is comprised of a main shift register M, an adder and a carry register C. After each clock the data in the SR is shifted one bit to the right and the new bit is calculated according to the previous register state and that of the carry register. In order to calculate the value of the new bit, some of the SR bits are tapped. The adder calculates the sum of the tapped bits along with the value stored in the carry register from the previous clock [43]. The structure described above is called the Fibonacci implementation (Figure 2.5), where the feedback is a function of the tapped bits.

An alternative structure is the Galois implementation, where the addition is done in parallel for all tapped bits [44]. Each tapped bit employs a full adder and a single bit carry register. Figure 2.6 represents the structure of the Galois FCSR. The operation of the Galois implementation can be formalized as follows [43], [44]:

$$m_i(t+1) = (m_{i+1}(t) + q_{i+1}(m_0(t) + c_{i+1}(t))) \bmod 2$$

$$ec_{i+1}(t+1) = q_{i+1}(m_{i+1}(t) + m_0(t) + c_{i+1}(t)) \operatorname{div} 2$$

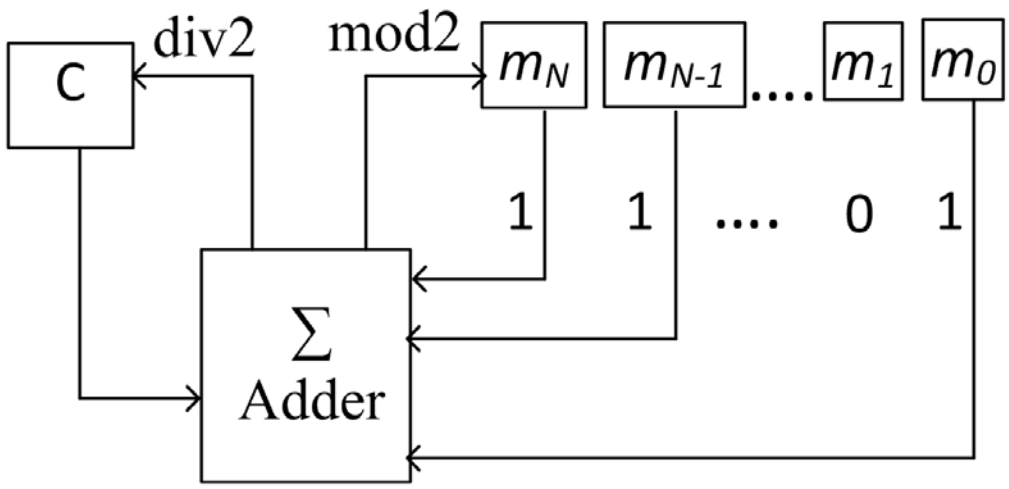


Figure 2.5 Feedback with Carry Shift Register (FCSR) - Fibonacci configuration

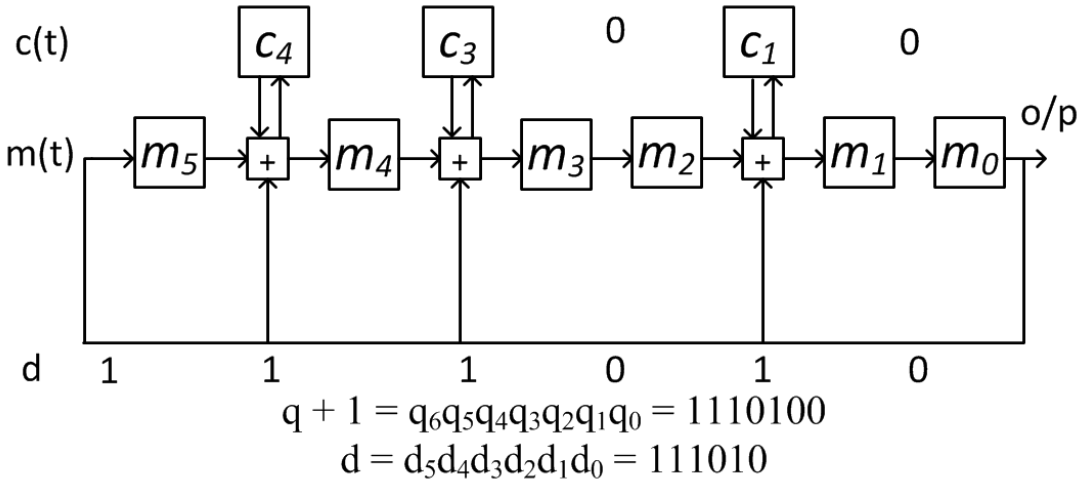


Figure 2.6 Feedback with Carry Shift Register (FCSR) - Galois configuration

Here, $m_i(t)$ is the state of the ith bit of the main register at time t, q_i is the value of the connection integer for bit i, $c(t)$ is the state of the carry register at time t and $0 \leq i \leq N-1$. The output of the generator at time t is $m_0(t)$. N is the size of the main register. The parameter q is the connection integer and it indicates which bits in the register should be tapped. q is a negative prime number and it follows that,

$$\mathbf{2^N \leq |q| \leq 2^{N+1}} \tag{2.3}$$

$$\boldsymbol{q + 1 = \sum_{i=1}^{N} 2^i q_i}. \tag{2.4}$$

$q_i = 1$ means that the bit is tapped. Let us consider another integer d which is related with q according to the following equation,

$$\boldsymbol{d = \frac{(|q|+1)}{2} = \sum_{i=0}^{N-1} 2^i d_i} \ . \tag{2.5}$$

Hence, $2^{N-1} \leq d < 2^N$. The position of '1' in the binary representation of d indicates the tapped bit. In order to achieve the maximal period of the generated sequence, q must be a prime number with 2 as a primitive root [41], [43]. That is, $(2^{q-1}-1)$ is divided by q. In that case, the maximum period T would be equal to $(|q|-1)$.

Filtered Feedback with Carry Shift Register (F-FCSR)

In order to increase complexities in the output sequence one of the common method is to combine different PRNG architectures. As a result the filtered FCSR (F-FCSR) has been proposed. The output sequence is linearly filtered (a linear Boolean function like XOR) in order to make it robust to cryptanalysis [42]. Figure 2.7 represents the structure of a Filtered FCSR.

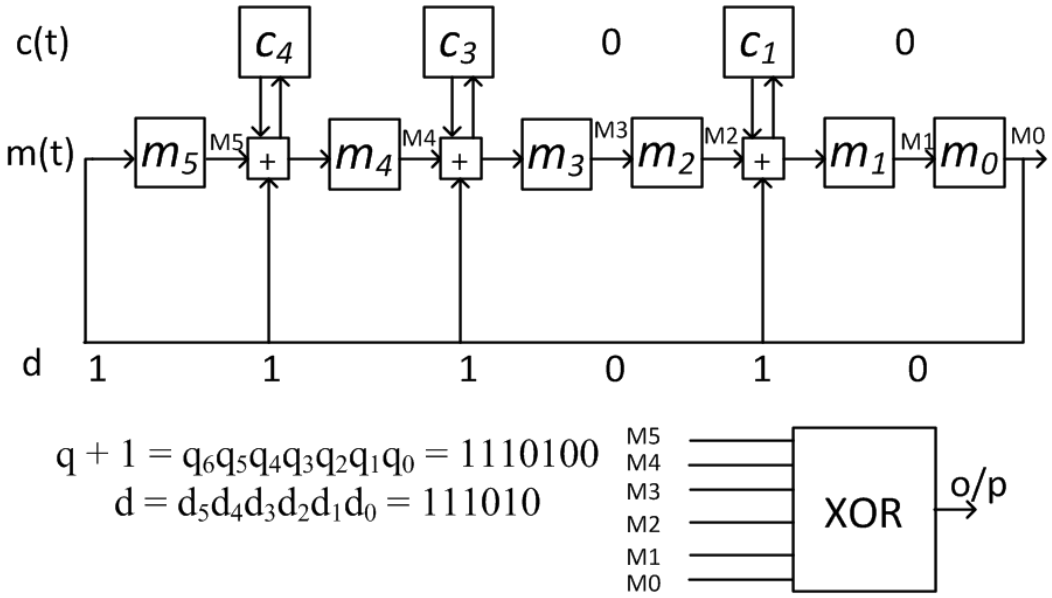


Figure 2.7 Filtered FCSR (F-FCSR)

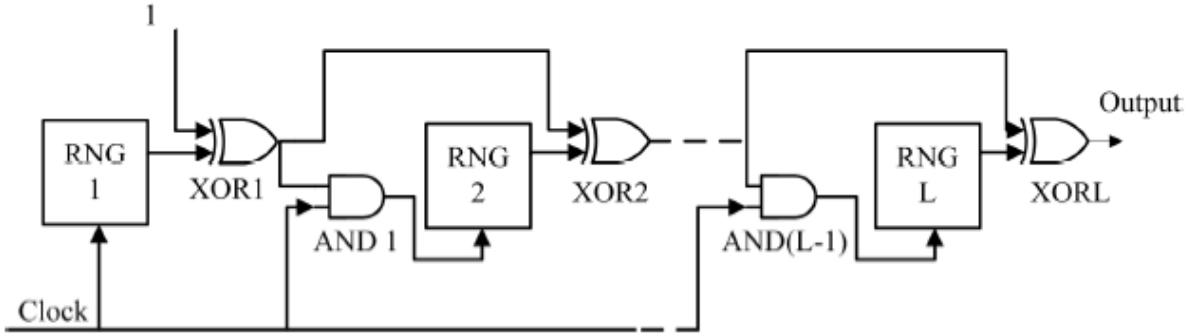


Figure 2.8 Gollmann Cascaded F-FCSR (GCF-FCSR)

Gollmann Cascaded Filtered Feedback with Carry Shift Register (GCF-FCSR)

Shift register based PRNGs are cascaded by making the registers' clocks controlled by their predecessors. The Gollmann cascaded configuration, introduced in [41], [45], is demonstrated in Figure 2.8. The important feature of this method of cascading is the use of the XOR function for the coupling of the register clock. Gollmann Cascaded F-FCSR was proposed to achieve a high level of modularity for the designer [43]. Theoretically, the period of the cascaded generator is the product of the periods of the different cores. When identical cores are used the period of the output sequence will be,

$$T = (T')^L$$

Here, T is the period of the entire cascaded structure, T' is the period of a single core and L is the number of the cores in the cascaded structure.

2.11.3 Cryptographically Secure Pseudorandom Number Generator (CSPRNG)

Cryptographically secure PRNGs (CSPRNGs) are PRNGs which have been specially designed to be resilient to certain cryptographic attacks and unpredictable in nature. The need for the unpredictability nature is unique to cryptography. For example, the security an encryption cipher entirely depends on a cryptographically secure key stream, which is used to encrypt a pain-text. This key stream should be unpredictable in nature so that one cannot predict or replicate the key stream even after observing an infinitely long sequence of that key stream.

In all other situations where pseudorandom numbers are needed in computer science or engineering, unpredictability is not a requirement. The requirements of an

ordinary PRNG are also satisfied by a cryptographically secure PRNG, but the reverse is not true. CSPRNG requirements fall into two groups: first, that they pass statistical randomness tests; and secondly, that they hold up well under serious cryptographic attacks, even when part of their initial or running state becomes available to an attacker [36], [38].

1. Every CSPRNG should satisfy the next-bit test. That is, given the first k bits of a random sequence, there is no polynomial-time algorithm that can predict the $(k+1)^{th}$ bit with probability of success better than 50%. In [46] it was proved that a generator passing the next-bit test will pass all other polynomial-time statistical tests for randomness. Informally, this means that given n output bits of the key stream,

   $$s_i, s_{i+1}, \ldots, s_{i+n-1},$$

   it is computationally infeasible to compute the subsequent bits,

   $$s_{i+n}, s_{i+n+1}, \ldots$$

   Another property of CSPRNG is that given the above sequence, it should be computationally infeasible to compute any preceding bits,

   $$s_{i-1}, s_{i-2}, \ldots$$

2. Every CSPRNG should withstand *state compromise extensions* attack. If part or all of its state has been revealed (or guessed correctly), it should be impossible to reconstruct the stream of random numbers prior to the revelation. Additionally, if there is an entropy input while running, it should be infeasible to use knowledge of the input's state to predict future conditions of the CSPRNG state.

Most PRNGs are not suitable for use as CSPRNGs. The outputs appear random to assorted statistical tests, but they do not resist determined reverse engineering. Moreover, for most PRNGs, when their state has been revealed, all past random numbers can be determined. CSPRNGs are designed explicitly to resist this type of cryptanalysis.

Some classes of CSPRNG designs include the following:

1. ***Number Theoretic Designs:***

   These designs are based on mathematical problems thought to be hard. Such algorithms are slow and impractical for many applications.

   a) ***Blum Blum Shub***

   The Blum Blum Shub algorithm has a security proof, based on the difficulty of the quadratic residuosity problem defined with respect to a large composite integer modulus. The only known way to solve that problem is to factor the modulus, therefore, it is generally regarded that the difficulty of integer factorization provides a conditional security proof for the Blum Blum Shub algorithm. However the algorithm is very inefficient and hence impractical for many applications unless very high security is needed [47].

   b) ***Blum-Micali***

   The Blum-Micali algorithm has an unconditional security proof based on the difficulty of the discrete logarithm problem but is also very inefficient [48].

2. ***Specific CSPRNG Designs:***

There are a number of practical PRNGs that have been designed specifically to be cryptographically secure, including:

a) ***Yarrow algorithm*** – used in FreeBSD, OpenBSD and Mac OS X [49].

b) ***Fortuna algorithm –*** Successor to Yarrow algorithm [38].

c) ***CryptGenRandom*** – Microsoft's Cryptographic Application Programming Interface Function [50].

d) ***Marsenne Twister –*** Used in Monte Carlo simulations in a number of fields including simulating complex biochemical pathways, photon migration, genome coalescence, cellular biology and computational finance. It is the default random number generator in *Python, Ruby, PHP, MATLAB® and C++*. It may take long time to get the initial random sequence [51]-[52]. Specific designs of CSPRNG mentioned above are application specific and are mainly implemented and used in software platforms.

3. ***Design based on Cryptographic Primitives***

Fast, efficient and hardware based CSPRNGs can be designed from cryptographic primitives such as stream ciphers, block ciphers (running in counter or output feedback mode) or cryptographic hash functions. These primitives are described next.

Stream Cipher

A stream cipher is a symmetric key cipher where each plaintext bit is XOR'ed with a pseudorandom bit from a key stream (Figure 2.9). There are synchronous stream ciphers where the key stream depends only on the key and asynchronous ones where the key stream also depends on the ciphertext [53]. The key stream used for encrypting the plaintext should be cryptographically secure and unpredictable in nature, otherwise ciphertexts can be decrypted easily. Hence, the key stream generated by a stream cipher is required to be a cryptographically secure pseudorandom sequence.

In the case of software-oriented stream ciphers, arguably the best investigated ones are RC4 [54], [55] and SEAL [56]. In the case of hardware-oriented ciphers, there is a wealth of LFSR-based algorithms. Many proposed ciphers have been broken. Among the best-studied ones are the A5/1 and A5/2 algorithms which are used in GSM mobile networks for voice encryption between cell phones and base stations. Neither of the two ciphers is recommended based on today's understanding of cryptanalysis. For 3GPP mobile communication, a different cipher A5/3 (also named KASUMI) is used, but it is a block cipher [53].

The pessimistic outlook on the state-of-the-art in stream ciphers changed with the eSTREAM project. It was a multi-year effort (2004 -2008), organized by the European Network of Excellence for Cryptology (EU ECRYPT), to promote the design of efficient and compact stream ciphers suitable for widespread adoption. The eSTREAM project had the explicit goal to advance the state-of-the-art of knowledge about stream cipher design [57]. The eSTREAM portfolio ciphers fall into two profiles. Profile 1 contains stream ciphers more suitable for software applications with high throughput requirements.

Profile-2 stream ciphers are particularly suitable for hardware applications with restricted resources such as limited storage, gate count, or power consumption (eg. *Grain v1, MICKEY 2.0 and Trivium*). Algorithms in this profile were intended to be efficient in terms of the physical resources required when implemented in hardware [58].

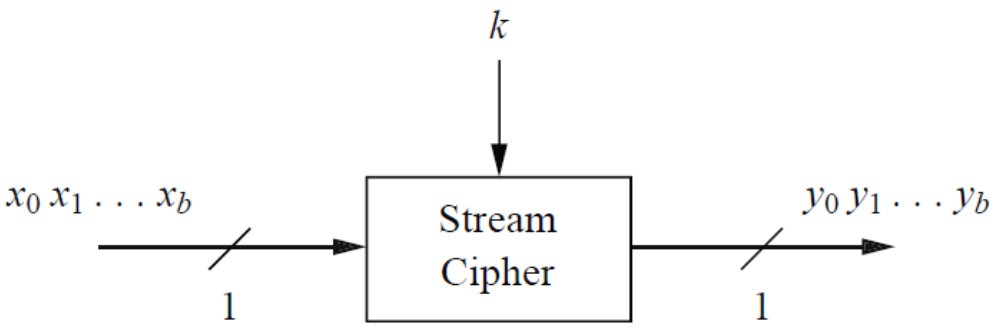


Figure 2.9 Principles of encrypting b bits with a stream cipher

Block Cipher

Block ciphers encrypt an entire block of plaintext bits at a time with the same key. The encryption of any plaintext bit in a given block should depend on every other plaintext bit in the same block. The vast majority of block ciphers either have a block length of 128 bits (16 bytes) such as the advanced encryption standard (AES), or a block length of 64 bits (8 bytes) such as the data encryption standard (DES) or triple DES (3DES) algorithm (Figure 2.10).

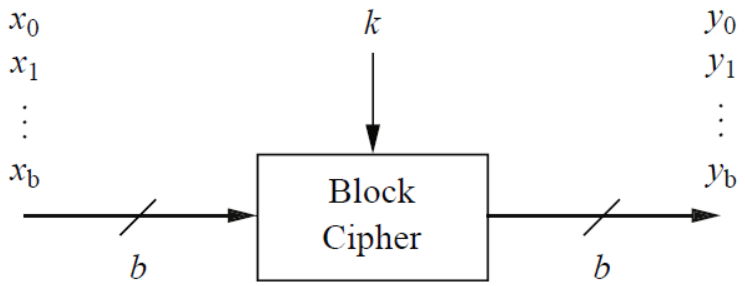


Figure 2.10 Principles of encrypting b bits with a block cipher

There are several modes of operation in Block Ciphers [61]. They are:

2.11.3.1.1 Electronic Code Book mode (ECB)

In ECB mode, a plaintext is segmented and encryption function is applied on each of the segment separately. No initialization vector (IV) is used here (Figure 2.11a) [61].

2.11.3.1.2 Cipher Block Chaining mode (CBC)

In CBC mode, output of one block cipher is fed into the other block cipher along with the next block of plaintext (message). Here, plaintext is XOR'ed with the previous cipher block's output and then the encryption function is applied (Figure 2.11b) [61].

2.11.3.1.3 Cipher Feedback mode (CFB)

In CFB mode, the cipher blocks are also chained as in CBC mode, but the output is produced in a different manner [61]. For each block, the cipher produced is just XOR'ed with the message (Figure 2.11c).

2.11.3.1.4 Output Feedback mode (OFB)

In this mode the IV is repeatedly encrypted to get a stream of random bytes. Unlike the other modes described before in OFB no part of the plaintext is ever given as an input to the block cipher. This makes this mode very similar to a stream cipher, where

a stream cipher produces a stream of random bytes and these random strings are XOR'ed with the plaintext to generate the cipher (Figure 2.11d) [61].

2.11.3.1.5 Counter mode (CTR)

The counter (CTR) mode is a bit different from the other modes defined above. It takes in an IV, and in each iteration the value of the IV incremented by one gets encrypted. The ciphertext is produced by XORing the encryption results with the plaintext blocks [53], [61] (Figure 2.12).

Good examples of block ciphers are – DES (Data Encryption Standard), IDEA (International Data Encryption Algorithm), RC5, and AES (Advanced Encryption Standard). DES (also known as Lucipher) is considered to be the first civilian block cipher, developed at IBM in 1970, based on the work done by Horst Feistal. A revised version of the algorithm was adopted as a US Government Federal Information Processing Standard [59]. DES has a block size of 64 bit and key size of 56 bit. DES has been superseded as a US Federal Standard by the AES [60]. It was adopted by National Institute Standards and Technology (NIST) in 2001 after a 5-year public competition. AES was developed by Joan Daemen and Vincent Rijmen, and known as Rijndael. AES has a fixed block size of 128 bits and a key size of 128, 192 or 256 bits.

Details of all the modes of operation of block ciphers can be found in [53], [61]. Among them the latter three modes use the block cipher as a building block for a stream cipher. Hence, a secure block cipher can be converted into a CSPRNG by running it in counter (CTR) mode or output feedback (OFB) mode, but in terms of hardware resources block ciphers are more expensive than the other cryptographic primitives.

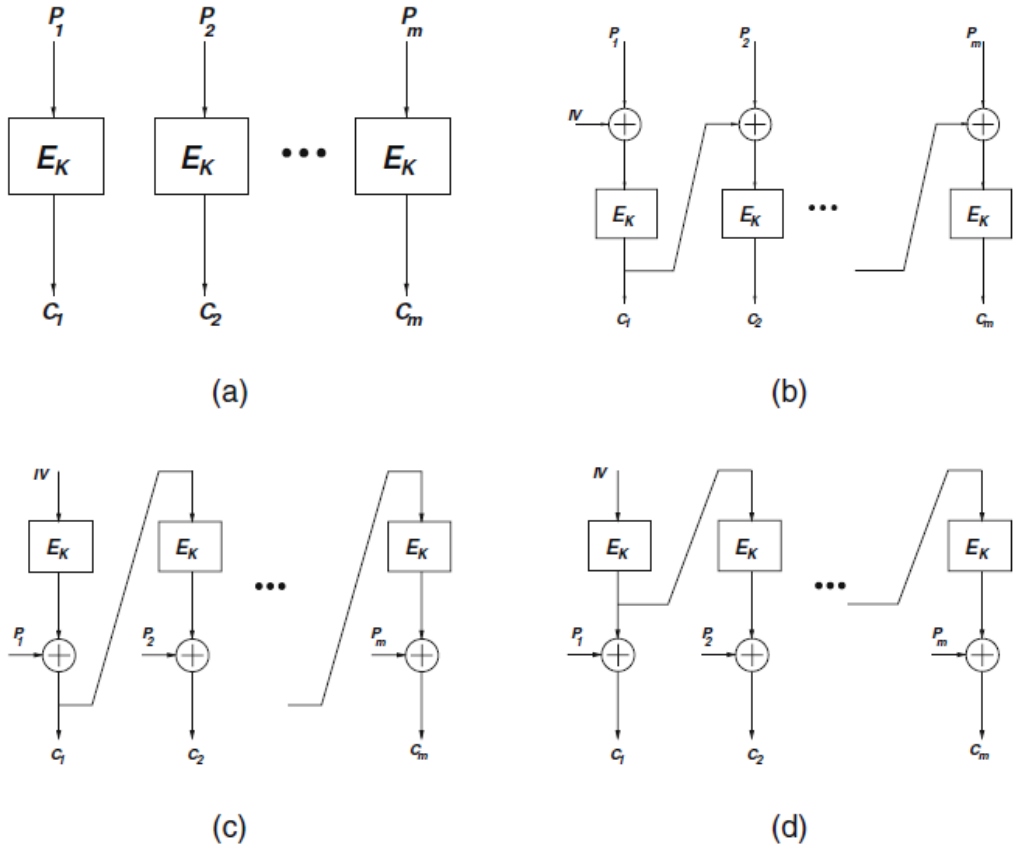


Figure 2.11 Traditional modes of operation of block cipher: (a) ECB (b) CBC (c) CFB (d) OFB

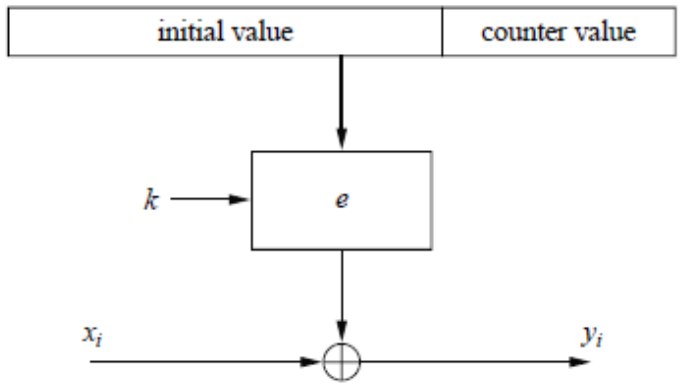


Figure 2.12 Block cipher in counter mode

Cryptographic Hash Function

A cryptographic hash function is a regular hash function; that is, an algorithm that takes an arbitrary block of data and returns a fixed-size bit string, the (cryptographic) hash value, but with the additional cryptographic property that an accidental or intentional change to the data will (with very high probability) change the hash value. The data to be hashed are often called the “message”, and the hash values are sometimes called the message digest or simply digest. The ideal cryptographic hash function has four main or significant properties:

1. it is easy to compute the hash value for any given message
2. it is infeasible to generate a message that has a given hash
3. it is infeasible to modify a message without changing the hash
4. it is infeasible to find two different messages with the same hash

Recognized hash functions are MD4, MD5, SHA-1 and SHA-2. Cryptographic hash functions have many information security applications, notably in digital signatures, message authentication codes (MACs), and other forms of authentication. They can also be used as ordinary hash functions, to index data in hash tables, for fingerprinting, to detect duplicate data or uniquely identify files, and as checksums to detect accidental data corruption. Indeed, in information security contexts, cryptographic hash values are sometimes called (digital) fingerprints, checksums, or just hash values, even though all these terms stand for functions with rather different properties and purposes [62].

A cryptographically secure hash of a counter may also act as a good CSPRNG. This is done by combining a (secret) random seed with a counter and hashing it.

However, there has been little study on this case and some authors warned against such use of hash functions [53], [62].

Chapter Three: **Hardware Watermarking System for MJPEG Video Authentication**

3.1 Introduction

In this chapter, we present a hardware implementation of a digital watermarking system that can insert invisible, semi-fragile watermark information in compressed video streams in real time. The watermark embedding is processed in the discrete cosine transform (DCT) domain. The designed watermarking system was integrated with an MJPEG video encoder (a non-standard video encoder) system to prove the concept of real-time video watermarking using the designed video watermarking system during the video encoding process. The hardware architecture was implemented on an Altera® Cyclone field programmable gate array (FPGA) for overall performance evaluation. To achieve high performance, the proposed system architecture employs pipeline structure and uses parallelism. Experimental results showed that such a hardware-based video authentication system using this watermarking technique features minimum video quality degradation and can withstand certain potential attacks i.e. cover-up attack, cropping, segment removal on video sequences. Furthermore, the proposed hardware based watermarking system features low power consumption, low cost implementation, high processing speed and reliability [28].

3.2 Contribution

The followings are the contributions described in this chapter:

1. To present an efficient hardware-based concept of a digital video watermarking system capable to insert invisible and semi-fragile watermark into compressed video stream.
2. To verify the performance of the designed watermarking system after being integrated with an MJPEG video compressor unit. The assessment shows that the designed watermarking system features low power consumption, efficient and low cost implementation and high processing speed. The system also features minimum video quality degradation and can withstand certain potential attacks i.e. cover-up attack, cropping, segment removal on video sequences.
3. The design was achieved via combined parallel hardware architecture with a simplified design approach of each of the components. This enhanced the suitability of this hardware watermarking approach to fit easily in devices that require high tampering resistance such as surveillance cameras and video protection apparatus.

The motivation here is to integrate the video watermarking system with a surveillance video camera for real time watermarking in the source end. This work presents the first semi-fragile watermarking system for video streams with hardware architecture.

3.3 Proposed Digital Video Watermarking

In this section, the details of the watermarking process of the proposed digital video watermarking system are provided. Figure 3.1 illustrates the general block diagram of the proposed system which is comprised of four main modules: a video camera, video compression unit, watermark generation and watermark embedding units.

According to the watermarking algorithm described in Chapter 2, the watermark embedding is performed in the Discrete Cosine Transform (DCT) domain. There are several advantages of doing this. DCT is used in the most popular image/video compression formats including JPEG, MJPEG, MPEG-x and H.26x. This allows the integration of both watermarking and compression into a single system. Compression is divided into three elementary phases: DCT transformation, quantization and Huffman encoding. Embedding the watermark after quantization makes the watermark robust to the DCT compression with a quantization of equal or lower degree used during the watermarking process. Another advantage of this approach is that in image or video compression the image or frames are first divided into 8×8 blocks. By embedding the watermark into each 8×8 DCT block, tamper localization and better detection ratios are achieved [18].

Each of the video frames is divided into 8×8 pixel blocks and DCT and quantization are performed on each of those 8×8 pixel blocks. After that the quantized DCT coefficients are passed to the watermark embedder unit in 8×8 blocks. The watermark generator unit, initialized by a predefined secret key, produces a pseudorandom watermark sequence. The watermark embedder module inserts the watermark data into the quantized DCT coefficients for each video frame. The

watermarked and quantized DCT coefficients of each video frame are encoded by the entropy coder. Finally, the compressed and encoded video frames with the embedded watermark information are read out of the system.

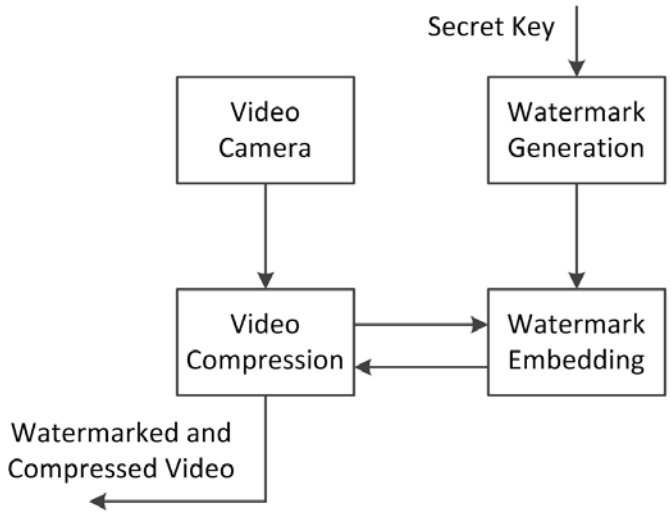


Figure 3.1 Proposed video watermarking system

3.3.1 Video Compression

Currently all popular standards for video compression, namely MPEG-x (ISO standard) and H.26x formats (ITU-T standard), use the same basic hybrid coding schemes that apply the principle of motion-compensated prediction and block-based transform coding using DCT [30]. In this chapter, we have focused on implementing our video watermarking system with non-standard MJPEG video encoder due to the simplicity of its implementation. However, MPEG-2 video compression standard is described below as a representative of the video encoding standards which feature DCT-based advanced compression methods.

Generally, a video sequence is divided into multiple Group of Pictures (GOPs), representing sets of video frames which are neighbouring in display order. An encoded MPEG-2 video sequence is made up of two types of frame-encoded pictures: Intra-frames (I frames) and Inter-frames (P frames or B frames). P frames are forward prediction frames and B frames are bidirectional prediction frames. Within a typical sequence of an encoded GOP, P frames may be 10% of the size of I frames and B frames are about 2% of I frames. Two types of redundancies are possible in video frames: temporal redundancy and spatial redundancy. MPEG-2 video compression technique reduces these redundancies to compress the images.

Within a GOP, the temporal redundancy among the video frames is reduced by applying temporal differential pulse code modulation (DPCM). The major video coding standards like H.261, H.263, MPEG-1, MPEG-2, MPEG-4 and H.264 are all based on the Hybrid DPCM/DCT CODEC, which incorporates motion estimation and motion compensation function, a transform stage and an entropy encoder [32]-[33]. This has been illustrated in Figure 3.2 an input video frame F_n is compared with a reference frame (previously encoded) F'_{n-1} and a motion estimation function finds a region in F'_{n-1} that matches the current macro-block in F_n. The offset between the current macro-block position and the chosen reference region is a motion vector, d_k. Based on this d_k, a motion compensated prediction F''_n is generated and it is then subtracted from the current macro-block to produce a residual or prediction error, e [32]. For properly decoding this motion vector, d_k has to be transmitted as well.

The spatial redundancy in the prediction error, e (also called the displaced frame difference) of the predicted frames and the I frame is reduced by the following

operations: each frame is split into blocks of 8×8 pixels which are compressed using the DCT followed by quantization (Q) and entropy coding (Run-level-coding and Huffman coding) (Figure 3.2).

Hardware implementation of MPEG-2 standard is not that simple. For simplifying the implementation of the video compressor module, non-standard Motion JPEG (MJPEG) video encoding technique is considered instead of MPEG-2 in this experiment. The MJPEG standard was developed by Xinph.org Foundation for Theora encoders (based on VP3 made by On2 Technologies) [31] to compete efficiently with MPEG encoders. The encoding process, performed on the raw data, is similar in both MPEG-2 and MJPEG. The only difference is the motion compensated prediction which is used in MPEG to encode the inter-frames (P or B frames).

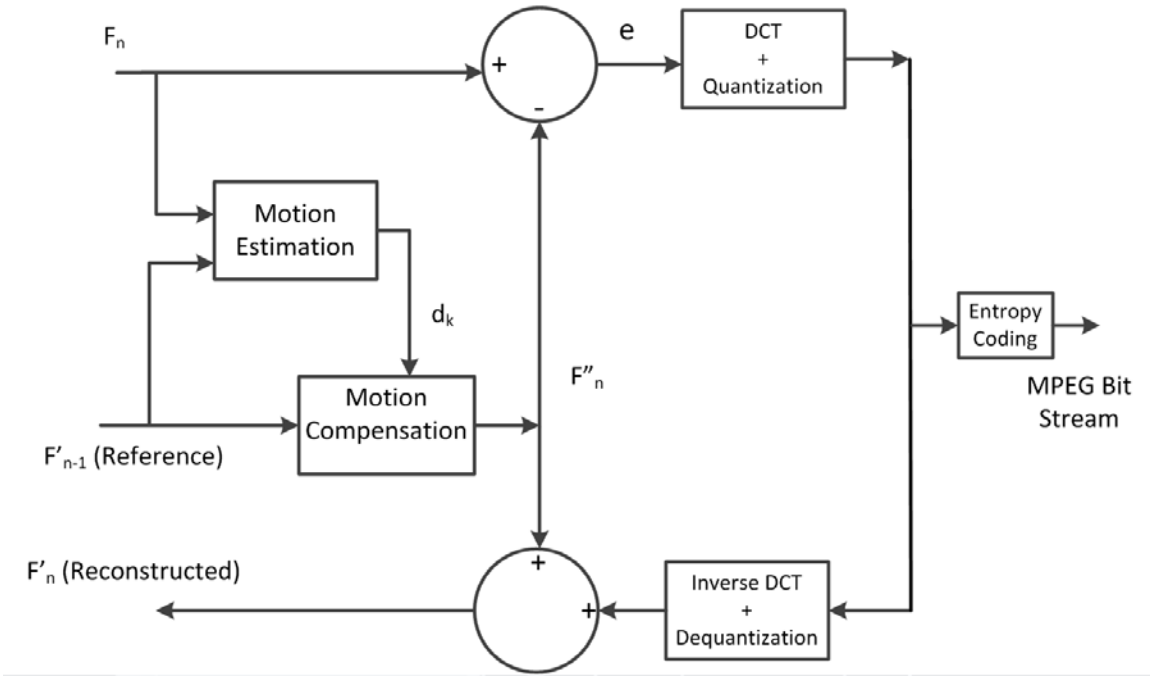


Figure 3.2 Block diagram of hybrid DPCM/DCT coding scheme [32]-[33]

3.3.2 Watermark Generation

Simple watermark data can be easily cracked. Therefore, it is essential to XOR the primitive watermark sequence with a pseudorandom sequence generated by a cryptographically pseudorandom number generator (CSPRNG). This ensures that the watermark sequence to be embedded into each video frame is unpredictable. In this chapter, we describe our earlier proof-of-concept implementation that used a non-cryptographically secure but hardware efficient PRNG. In subsequent chapters, we describe our improved version that does use a CSPRNG.

There are different available approaches to convert a primitive watermark into a secret pattern [15] - [16]. According to the recommendation by *J. Dittman et al.* [35], a primitive watermark pattern can be defined as a meaningful identifying sequence for each

video frame in case of video watermarking. In our design, the unique meaningful watermark information for each video frame contains the time, date, camera ID and frame serial number (which is related to its creation) as shown in Figure 3.3. This will establish a unique relationship of the video stream frames with the time instant, the specific video camera and the frame number. Any manipulation, such as frame exchange, cut and substitution will be detected by the specific watermark. This will generate a different watermark for every frame (time-varying) because of the instantaneously changing serial number and time. A 64-bit long binary sequence, a_i is chosen as a primitive watermark sequence in our system. The reason for choosing 64-bit long sequence is to match the size of a single 8×8 DCT coefficient block.

The block diagram of the proposed watermark generator is shown in Figure 3.4. A secret watermark sequence is generated by performing expanding, scrambling and modulation on the primitive watermark sequence a_i. Two secret binary keys are used in generating the watermark sequence. *Key 1* is used for initializing the scrambling operation and *Key 2* is used as a seed for the PRNG which generates a pseudorandom sequence.

Initially, the 64-bit long primitive watermark sequence, a_i is expanded (a_i'). It is expanded by a factor C_r. For watermarking a single 256 × 256 pixels video frame, the value of the factor C_r will be (256 × 256 / 64) or, 1024. Expanding is performed to match the size of the video frame with the expanded primitive watermark sequence, a_i'. Expanded sequence is stored in a buffer. Scrambling is performed on the expanded sequence a_i'. This is actually a sequence of XOR operations performed among the contents of the expanded primitive watermark sequence stored in the buffer. *Key 1*

initiates the scrambling process by specifying two different addresses (*Add1* and *Add2*) of the buffer for performing the XOR operation in between them. The basic purpose of scrambling is to complicate the primitive watermark sequence. Thus, the expanded and scrambled sequence c_i is obtained.

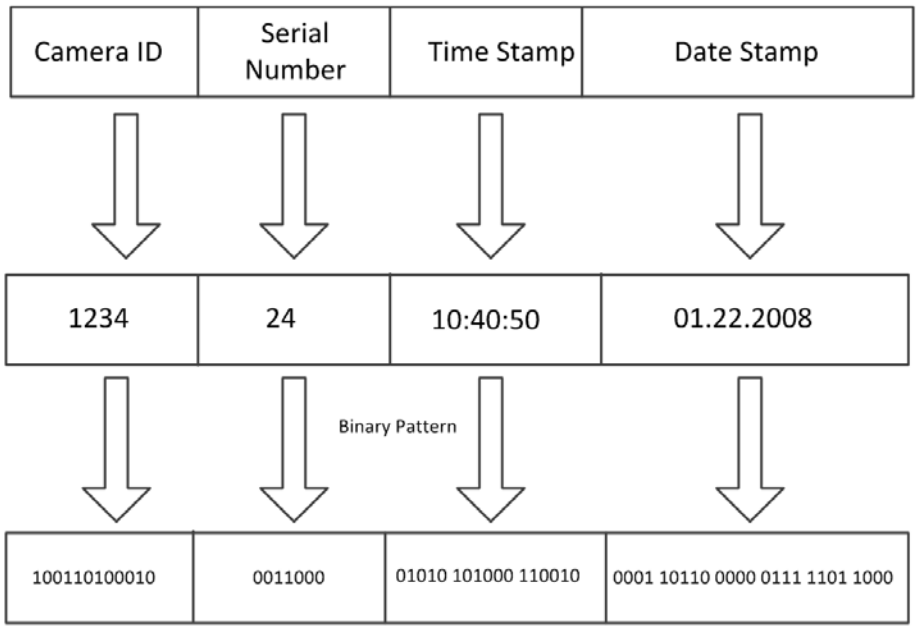


Figure 3.3 Structure of the primitive watermark sequence

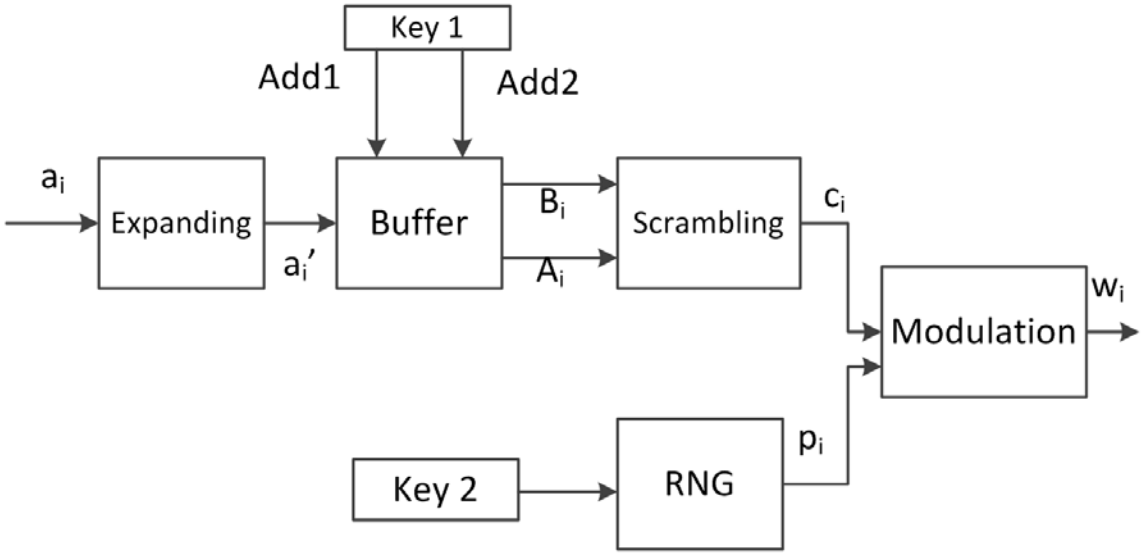


Figure 3.4 Block diagram of the proposed watermark generator

After that, the expanded and scrambled primitive watermark sequence, c_i is XOR'ed by the PRNG generated pseudorandom sequence p_i to generate the secret watermark sequence w_i. The pseudo randomness of the sequence makes it difficult to detect, estimate and manipulate the pseudorandom watermark sequence for the attacker.

The PRNG used in our system is a Gollmann Cascaded Filtered Feedback with Carry Shift Register (GCF-FCSR), presented by *Li et al.* in [43]. The length of the PRNG key (*Key2*), which is the initializing seed, is chosen to generate a pseudorandom sequence of a period larger than the size of the video frame to be watermarked so that the watermark information does not repeat inside a single frame. In our system we have chosen a 25-bit long binary key to initialize the PRNG.

3.3.3 Watermark Embedding

Watermark embedding is performed only on I frames (Intra-frames) since B or, P frames are predicted from I frames. If all of I, B or, P frames are watermarked, the watermarked data of previous frame and the one of the current frame may accumulate, resulting in visual artifacts (called 'drift' or 'error accumulation') during decoding procedures [9]. To avoid such an issue, within each GOP of video stream, only I frame is identified to be watermarked.

The watermarking algorithm should be hardware friendly so that it can be implemented with high throughput and minimum resources. For this purpose, the implementation of the watermarking algorithm must support pipeline processing so that two or more 8×8 blocks inside a single video frame or more than one frame can be watermarked simultaneously without the penalty of high latency. The implementation should also support parallel processing to perform more than one operation simultaneously. These features can aid in increasing the processing speed of watermarking system.

The watermark embedding approach used in this work was originally developed by *Shoshan et al.* [18] and *Nelson et al.* [64]. This watermarking algorithm, capable of inserting a semi-fragile invisible watermark into a compressed image in the DCT frequency domain, was modified and applied in watermarking of video frames. In general, for each DCT block of a video frame, N cells are identified as 'watermarkable' and are modified by the generated watermark sequence. In our design we will change only two cells from each 8×8 block. The chosen cells contain non-zero DCT coefficient values and are found in the mid-frequency range. This algorithm is detailed in Chapter 2.

The proposed watermarking process is performed during the MJPEG video encoding process. This is described briefly as below (Figure 3.5):

1. Split I frame data into 8 × 8 blocks.
2. For each 8×8 block perform DCT, quantization and zigzag scanning to generate quantized DCT coefficients in zigzag order.
3. Identify two watermarkable cells for each 8×8 block and calculate the modification value for each selected cell.
4. Modify the identified watemarkable DCT coefficients according to the modification values.
5. Perform de-quantization and inverse DCT on each 8×8 block of watermarked coefficients to reconstruct the watermarked I frame data (*I'*).
6. Buffer the reconstructed watermarked I frame (*I'*).
7. The prediction error is determined by taking differences between the current P frame and the reconstructed watermarked reference frame (*I'*).
8. Perform DCT, quantization and zigzag scanning on the prediction error to generate compressed P frame data.
9. Perform entropy coding on each 8×8 block of compressed P frame data and watermarked compressed I frame data to generate compressed and watermark embedded video stream.

To avoid computationally demanding operations and to simplify the hardware implementation, watermarking is performed during non-standard MJPEG video encoding process.

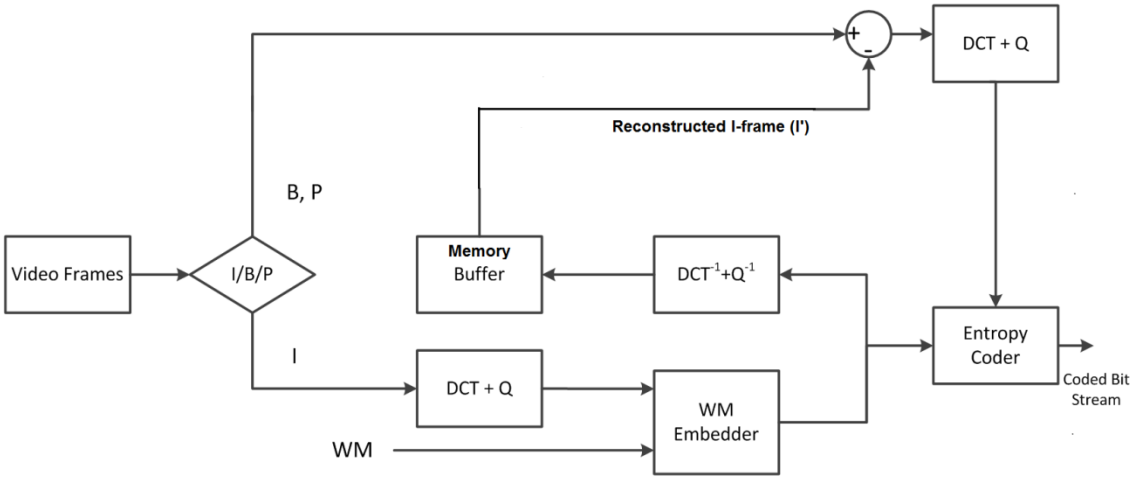


Figure 3.5 Dataflow of proposed watermarking process during MJPEG encoding process

3.4 Hardware Architecture Design

There exists a wide range of available techniques to implement the peripheral blocks of the proposed video watermarking system. Here, the focus is on simplifying the process as much as possible to make it fit easily within existing video processing circuitry with a small increase in system power consumption and hardware resources. The system level architecture of the video watermarking system is demonstrated in Figure 3.6. Basically, the proposed system architecture includes six modules: video camera, video compressor, watermark generator, watermark embedder, control unit and memory. Among them video compressor unit, watermark generator, watermark embedder and controller unit were implemented on the FPGA. Video streams captured by the camera were stored in the memory buffer (onboard SRAM).

The hardware implementation for the complete design is developed using Verilog HDL. At first, the captured video frame is temporarily stored in a memory buffer and each 8×8 block of the video frame data is processed by the video compressor unit sequentially. The watermark embedder unit inserts a pseudorandom watermark sequence, generated by the watermark generator unit, into I frames and stores the watermarked video frame in memory. The control unit is responsible for synchronizing the operations and the data flow in the entire system.

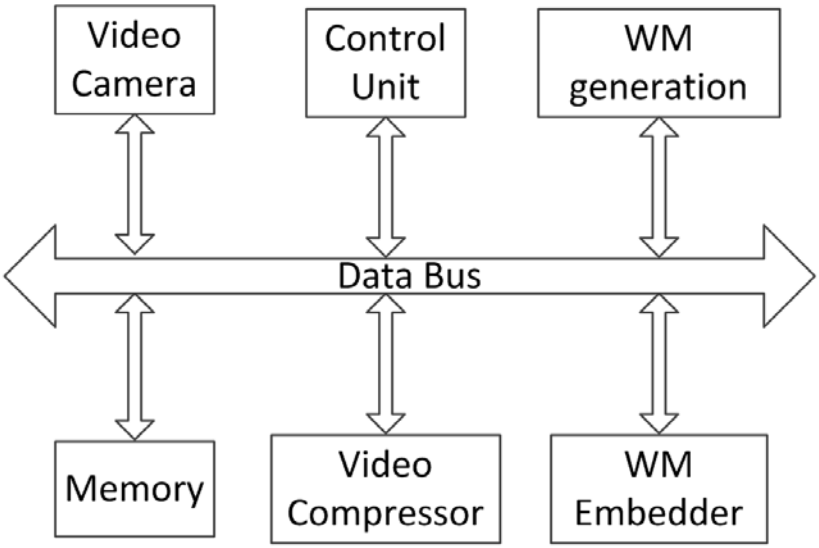


Figure 3.6 System level architecture of the proposed watermarking system

3.4.1 Video Compressor

For hardware implementation of the video compressor module, the non-standard Motion JPEG (MJPEG) video encoding technique was chosen because MJPEG offers several significant advantages for hardware implementation [65]. Even though MJPEG

provides less compression than commonly used MPEG-x encoding standard, it is easy to implement MJPEG encoding process in hardware with relatively low computational complexity and low power dissipation. MJPEG is supported by many video data players like VLC Media Player [66], MPlayer [67] and so on. Therefore, for the evaluation of the designed watermarking system, the MJPEG compression standard was chosen. Figure 3.7 presents the hardware architecture of the MJPEG video compressor. However, in the following chapter, we focus on validating the designed watermarking system with the contemporary and state-of-the-art MPEG-4 video encoder.

Depending on the type of the frames, video data (in 8×8 blocks) is compressed (DCT and quantization) either directly (for I frames) or after determining the prediction difference between P frame and the reconstructed reference frame (*I'*). The compressed P frames (prediction difference) are passed to the entropy coder but the compressed I frames are fed to watermark embedder. After the watermark embedding process is completed, the watermarked and compressed I – frames are also passed to the entropy coder. Encoded P – frames and encoded watermarked I – frames form MJPEG watermarked data stream.

After encoding and watermarking process, another data flow containing compressed P – frames and watermarked compressed I – frames are passed to the decoder. Here decompression (de-quantization and inverse DCT) is performed on each 8x8 block of data. The decompressed I frame data is stored in the memory buffer as the reconstructed watermarked I frame (or reconstructed reference frame) and is utilized as the reference frame for the encoding process of the following P frames. The reconstructed reference frame data, delayed by the bypass buffer, is combined with the decompressed P

frame data to form reconstructed P frame data. Two multiplexers are used to select I/P frame data stream during the compression and decompression process.

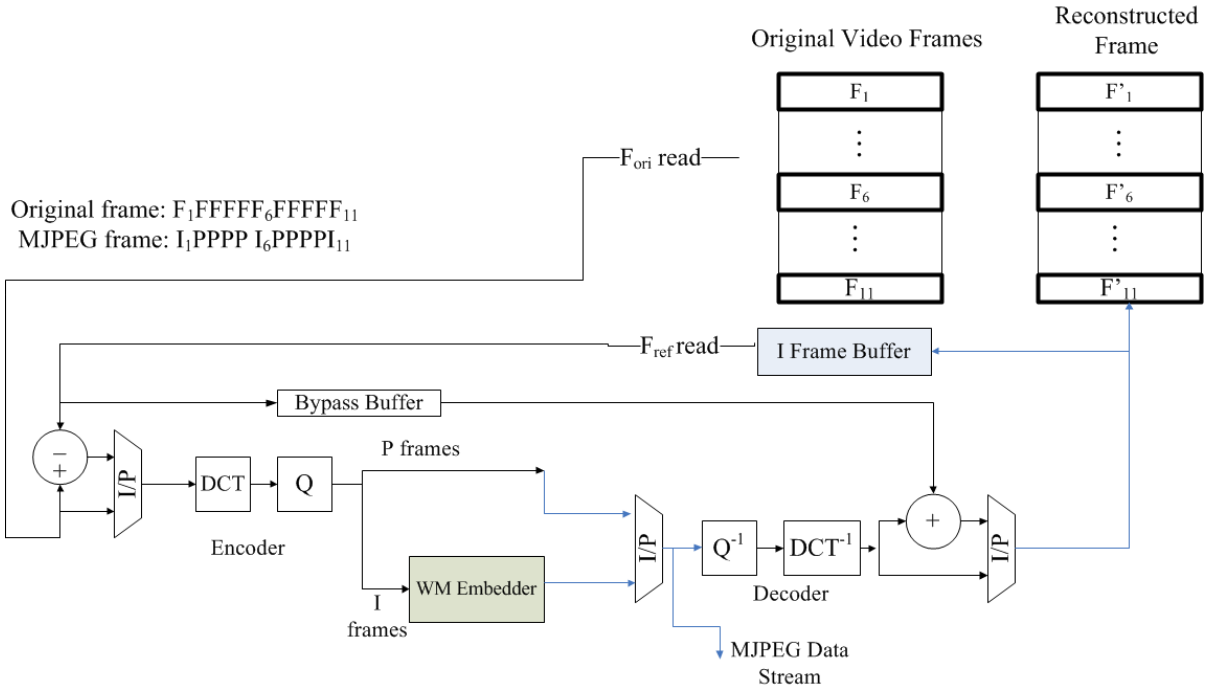


Figure 3.7 Hardware architecture of MJPEG video encoder integrated with watermarking unit (F_{ori} is the original frame, F_{ref} is the reference frame)

3.4.2 Watermark Generator

Figure 3.8 demonstrates the hardware architecture of the watermark generator. The expanding process is performed by providing consecutive clock signal so that expanded watermark sequence can be generated by reading the primitive watermark sequence (a_i) out for c_r times. Expanded sequence (a_i') is stored in memory buffer.

Scrambling is done by using the secret digital key *Key1*, which has two parts. The two different parts initiate two different counters. In each state of the counters two readings (addressed by *Add1* and *Add2*) come out from the buffer to perform the XOR operation between them. Thus the scrambled watermark sequence, c_i is generated. Different choice of *Key1* can start the counters with different states and can generate different corresponding addresses; hence we will get different patterns of c_i.

A pseudorandom sequence p_i, generated by the proposed Gollmann Cascaded Filtered Feedback with Carry Shift Register (GC-FFCSR) PRNG [43], seeded with secret binary key (*Key2)* of 25 bits long, is used to XOR the expanded and scrambled watermark sequence c_i. Finally, the generated secret and pseudorandom watermark sequence, w_i is embedded into the video streams by the watermark embedder unit.

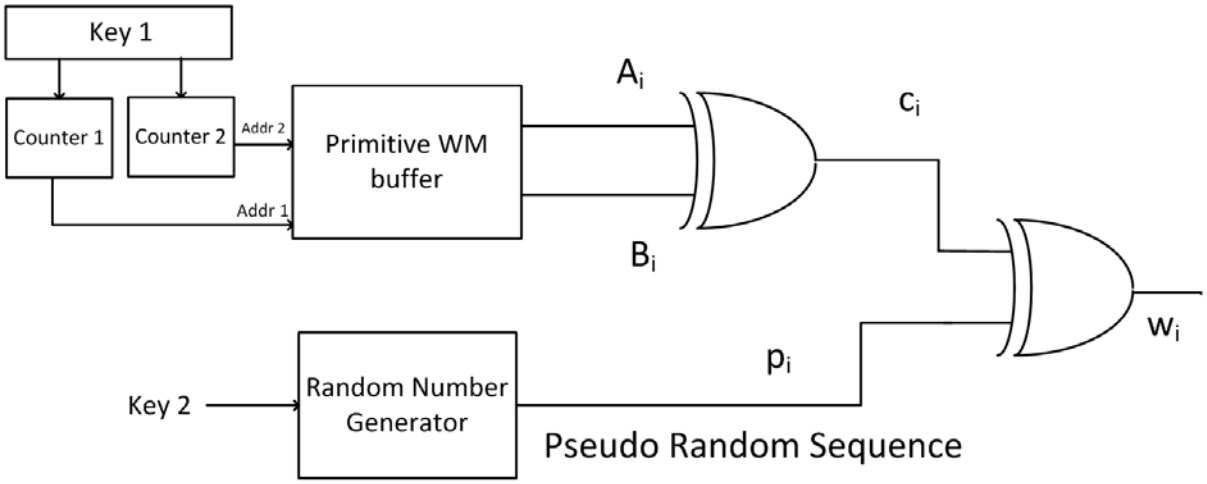


Figure 3.8 System level architecture of the watermark generator unit

3.4.3 Watermark Embedder

The schematic of the watermark embedder unit is presented in Figure 3.9. The watermark embedder works in two phases (calculation and embedding). When considering a cycle of embedding the watermark in a single 8×8 block, each phase takes one block processing cycle. Two blocks are processed simultaneously in a pipelined manner so that the embedding process only requires one block processing cycle for each 8×8 block.

The watermarkable cells are identified in each 8×8 block according to the algorithm detailed in Chapter 2. An increase in the number of watermarkable cells (N) reduces the video frame quality because of the reduction in the originality of the video frame. In our design we have maintained $N=2$. A block that produces less than $N(=2)$ cells is considered to be unmarked and disregarded. The design of the watermark embedder unit was taken from *Shoshan et al.* [18].

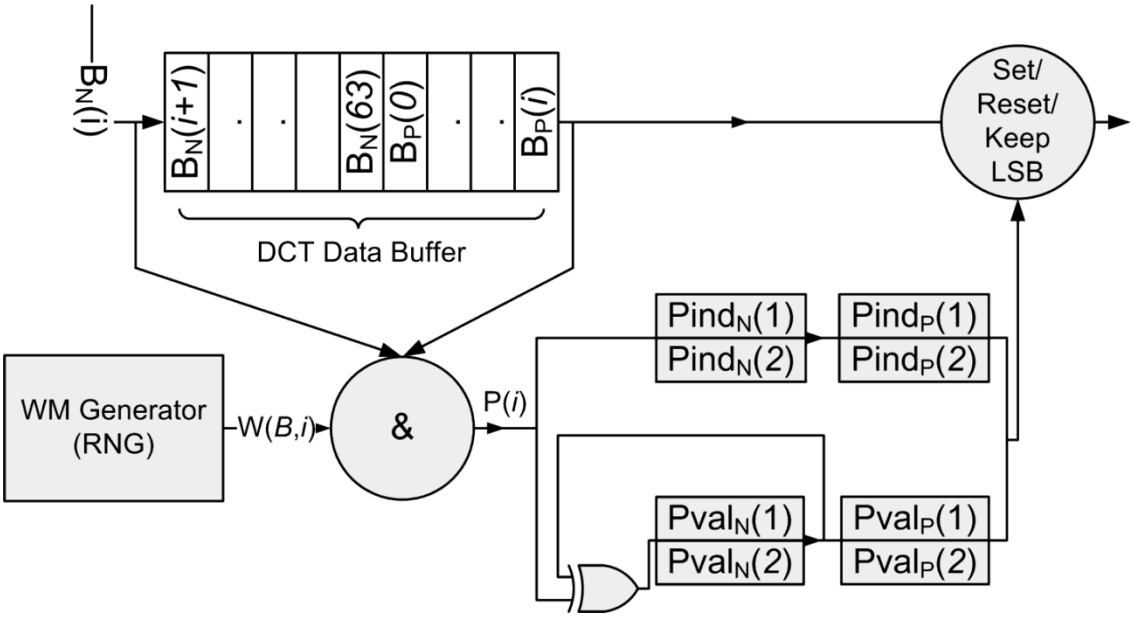


Figure 3.9 Hardware architecture of the watermark embedder unit [18]

3.4.4 Control Unit

The control unit generates control signals to synchronize the overall process in the whole system. As shown in Figure 3.10, it is implemented as a finite state machine (FSM) with four main states.

1. **S0:** In this initial state, all the modules stay idle till the signal **Start** is set to high or ‘1’. Depending on the **Select** signal, different processing steps are applied to the video frames. When **Select** is set to ‘1’, the control state will move to state S1 for intra-frames (I frames) or directly jump to the state S2 for inter-frames (P frames) if **Select** is ‘0’.
2. **S1:** In this state, watermark embedding is performed. The intra-frame blocks read from the memory and the generated watermark sequence are added together by activating the watermark embedder module in this state. Once the watermarking is completed for the block, **Blk** signal will be set to ‘1’ and the FSM moves to the state S2.
3. **S2:** In the state S2, the watermarked intra-frame data from the watermark embedder module or the unmodified inter-frame sequences from the memory are encoded. The signal **Blk** remains ‘0’ until the encoding of the current block is completed. When finished, the encoded and watermarked blocks are fed to the next state S3.
4. **S3:** In this stage, the watermarked and compressed video frame blocks are written back to the memory. When all the blocks of a frame are encoded the control signal **Img** changes to ‘1’ and the FSM goes back to the state S0 for considering the next frame. If encoding of all the blocks of the current frame is not finished,

the system goes back to the state S1 or S2 depending on the type of the current frame (inter-frame or intra-frame).

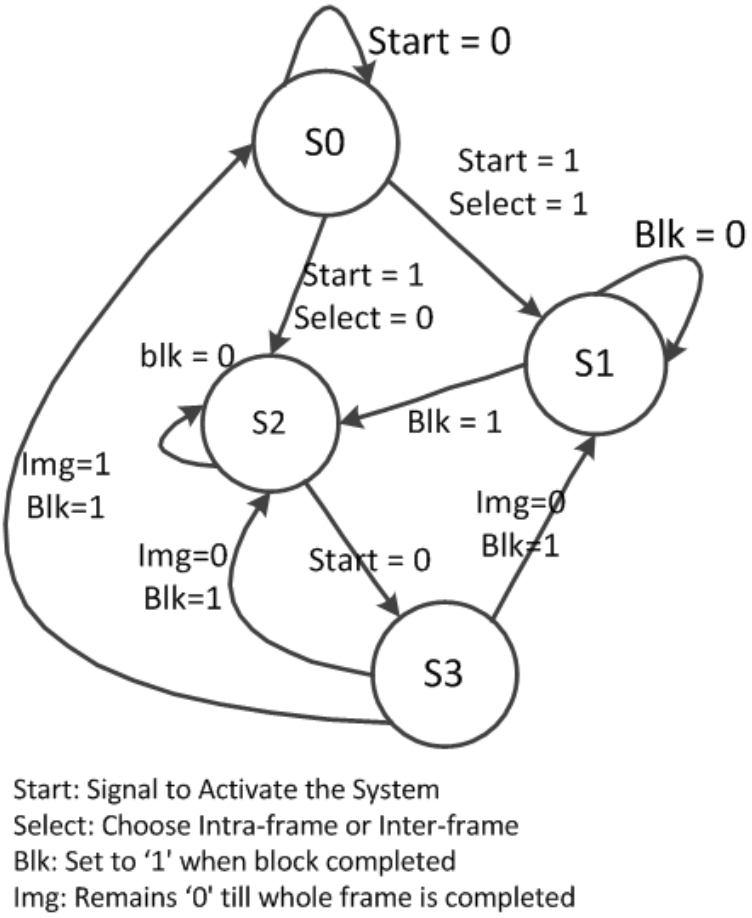


Figure 3.10 States of Finite State Machine (FSM) of controller unit of the designed video watermarking system with integrated MJPEG encoder system

3.4.5 FPGA-based Prototyping

Each module in the proposed digital video watermarking system, including the MJPEG compressor, watermark generator and watermark embedder has been implemented and tested individually, and then integrated together to obtain the final system architecture. The proposed architecture was first modeled in Verilog HDL and the functional simulation of the HDL design was performed using the Mentor's ModelSim® tool. Finally, the system design was synthesized to Altera® Cyclone EP1C20 FPGA device using Altera® Quartus II design software. The block diagram of the FPGA implementation of the whole system is shown in Figure 3.11. The synthesis report provides the hardware usage of the individual units and is shown in Table 3.1.

3.5 Experimental Results

3.5.1 Methodology of Verification

In order to evaluate the performance of the hardware-based video watermarking system properly, the algorithm was tested with two representative grey-scale video clips: A "dynamic" scene, which has significant high-frequency patterns and a "static" scene, which has several homogenous (low spatial frequency) areas. The video clips were of 256 × 256 pixels/frame resolution and were captured at a rate of 25 frames/s by a video camera.

For each video stream, a comparison was performed between two sets of experimental results: original video stream vs MJPEG video stream and original video data vs watermarked video stream. The comparisons were quantified using the standard

video quality metric: Peak Signal to Noise Ratio (PSNR), which is a well-known quantitative measure in multimedia processing used to determine the fidelity of a video frame and the amount of distortion found in it, as suggested by *Piva et al.* [4] and *Strycker et al.* [13]. The PSNR, measured in decibels (dB), is computed using Equation 3.1.

$$PSNR = 10\log_{10}\left(\frac{255}{MSE}\right) \tag{3.1}$$

$$MSE = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left[f(m,n) - k(m,n)\right]^2 \tag{3.2}$$

Here, *255* is the maximum pixel value in the grey-scale image and MSE is the average mean-squared error, as defined in Equation 3.2. Here, *f* and *k* are the two compared images of M × N pixels (256 × 256 pixels in this experiment).

3.5.2 Analysis of Watermarked Video Quality

The results of the experiment performed with the implemented system are presented in Figure 3.12, which contains the original sample frames, the compressed frames and the atermarked frames. Here, the presented results are achieved for the case that only two DCT coefficients in each block are changed. In terms of PSNR, the quality of the watermarked frames is maintained above ~35dB, measured consistently with no visually perceptible artefacts.

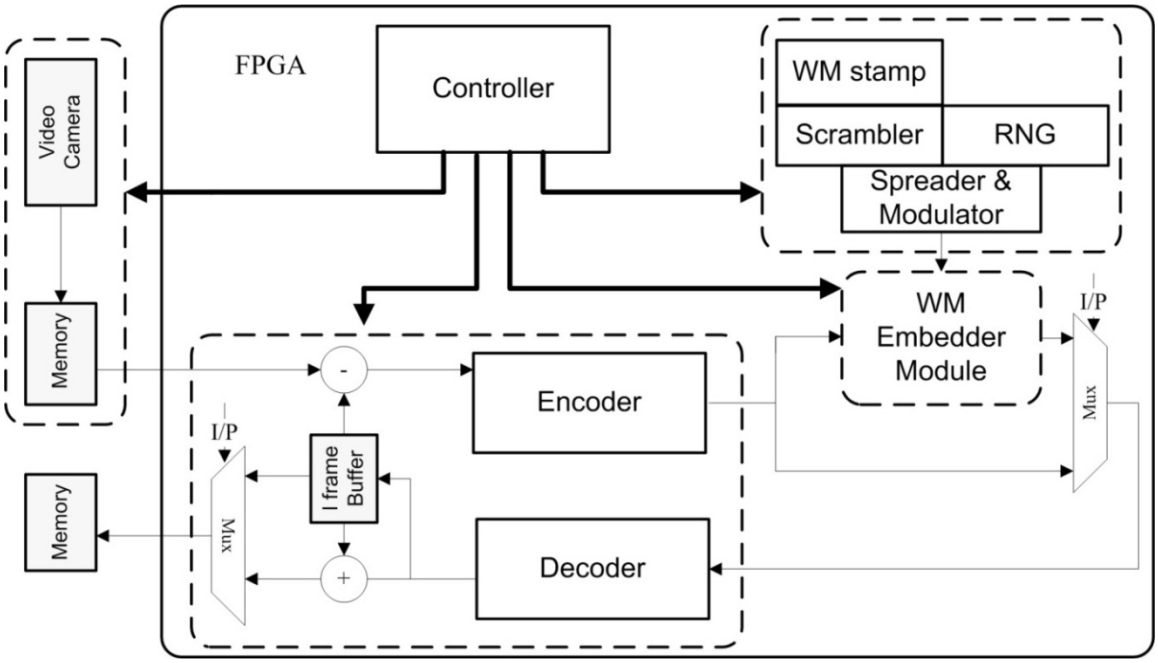


Figure 3.11 FPGA based implementation of the proposed video watermarking system

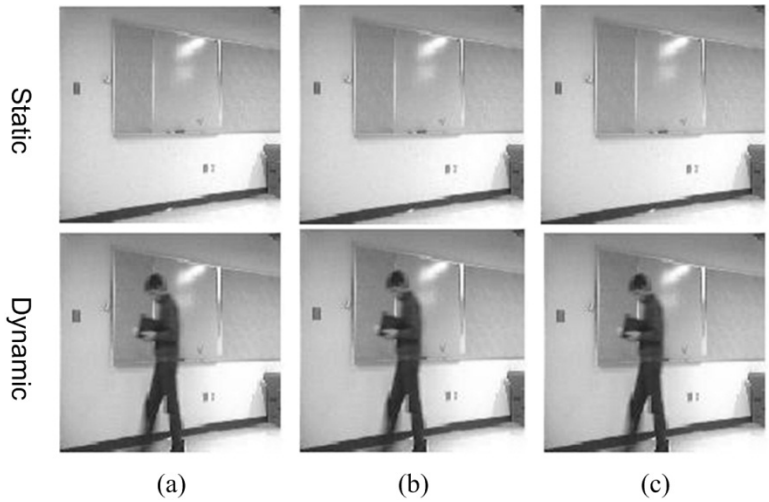


Figure 3.12 Examples of watermarked video streams (a) Original video frame (b) MJPEG video frame (c) Watermarked video frame.

Three sets of Group of pictures (GOPs) consisting of three frames (one I and two P frames) were experimented for both static and dynamic video clips. As described above, the inter-frames (P frames) are encoded based on the intra-frames (I frames) and thus the PSNR values of the intra-frames are expected to be higher than that of the inter-frames, as demonstrated in Figure 3.13 and Figure 3.14. That is why we get the 'hill' behavior at the 1st, 4th, 7th fame which are I frames. On the other hand, watermark embedding did not degrade the quality of the watermarked compressed video streams compared to the un-watermarked compressed video (Figure 3.13 and Figure 3.14). Therefore, the proposed hardware implementation of video watermarking system features minimum degradation of video quality. This is because we have chosen only two DCT coefficients to be watermarked in each 8×8 DCT coefficient block during the watermarking process. Increasing the number of coefficients to be watermarked will degrade the quality of the video.

The performance of the FPGA implemented video watermarking system with MJPEG encoder was evaluated in terms of hardware cost, power consumption, processing speed and security issues.

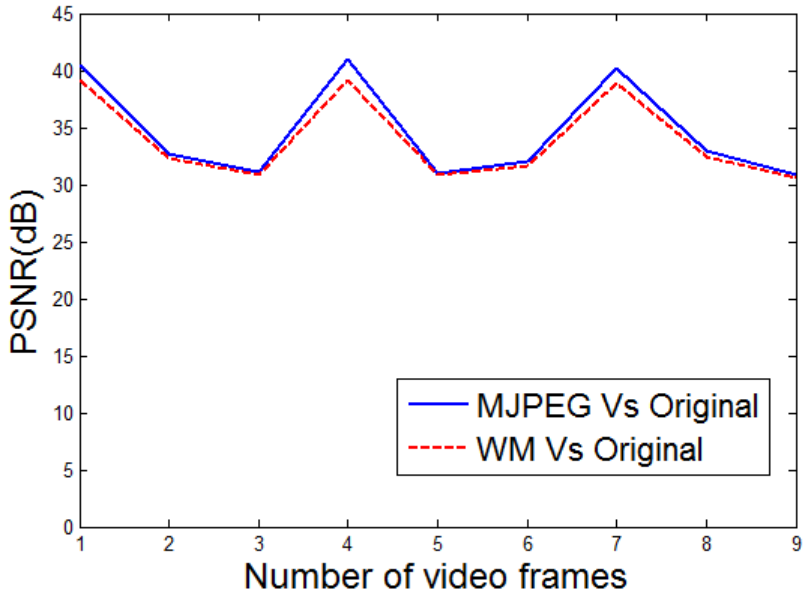


Figure 3.13 PSNR comparisons for static scene

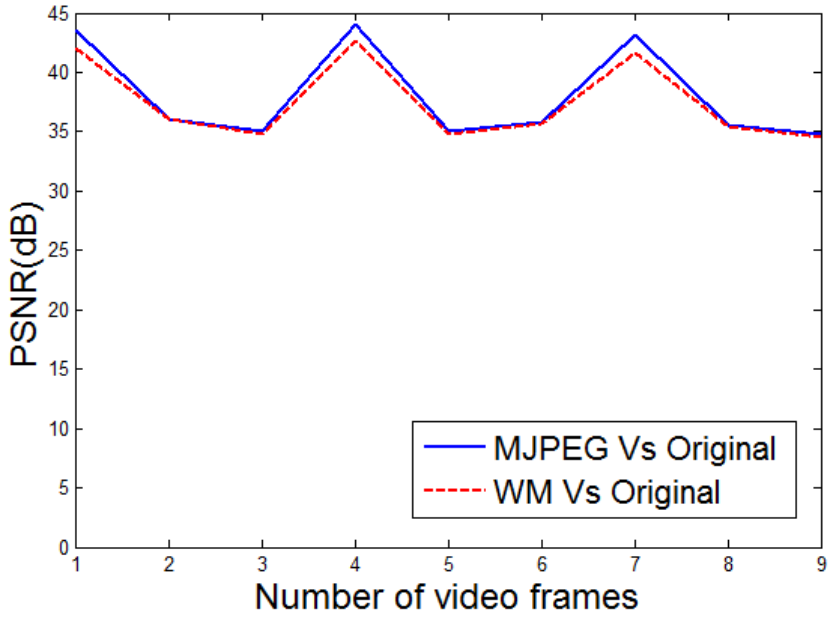


Figure 3.14 PSNR comparisons for dynamic scene

3.5.3 Hardware Cost and Power Consumption:

The hardware resources utilized by different units are given in Table 3.1. The results clearly indicate that the addition of the watermark generator and watermark embedder units caused only 4.99% increase in logic elements usage and 9.28% increase in memory resources utilized compared to those of the MJPEG video compressor unit alone. In general, any device that is large enough for the implementation of the MJPEG video compressor will be able to accommodate the additional hardware required for the watermark generator and watermark embedder units.

Table 3.1 FPGA synthesis result for MJPEG video watermarking system implementation

Components	**Logic Cells**	**Memory Bits**	**Initial Latency (cycles)**	**Power Consumption**
MJPEG Video Compressor	8822	13540	188	260 mW
Watermark Generator	309	512	64	
Watermark Embedder	132	744	64	10 mW
Overall Architecture	9263	14796	320	270 mW

Power consumption is an important concern in hardware implementation of a VLSI system and needs to be kept low as much as possible. Using the built in power analyzer tool of Altera® Quartus II software the power consumption of the synthesized video watermarking system was measured and it was found that the MJPEG video compressor integrated with the watermarking unit consumes 270 mW power in total, whereas the MJPEG video compressor itself consumes 260 mW power. So with an addition of only 10 mW power consumption the video watermarking unit (watermark generator and watermark embedder) can be easily integrated with the MJPEG video compressor unit.

3.5.4 Processing Speed

The data of each video frame is processed in 8×8 block wise. Each 8×8 data block is passed through the compression (DCT and quantization) unit, the watermark embedder unit and the decompression (de-quantization and inverse DCT) unit sequentially in a pipelined manner. From the timing diagram of Figure 3.15 the processing time of a single I frame can be determined.

For the first 8×8 macro-block of a frame it takes 320 clock cycles (Latency). Then for all the later 8×8 blocks it takes 64 cycles each (Throughput). The watermark generator takes 64 clock cycles to generate the watermark data which is completed at the initial 64 clock cycles of the frame processing period and then watermark data is stored in a memory buffer. Hence the watermark generator unit does not contribute to the initial latency of the frame processing period. The pipelining architecture and parallelism of the designed system helped in achieving this high throughput regardless of the initial latency

period. Processing of P frame will require less time as they are not watermarked. If we consider a video frame of N× M pixel resolution, the time it would take to watermark one frame of the video is defined by equation

$$T = \left[Latency + \left(\frac{MN}{8x8} - 1\right) Throughput\right] \frac{1}{Clock\ Frequency} \quad \textbf{(3.3)}$$

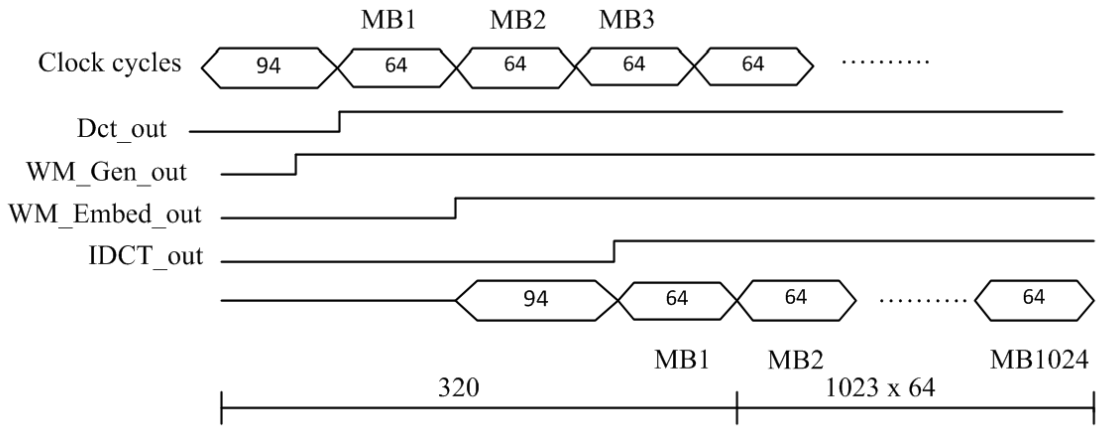


Figure 3.15 Timing diagram for the processing (watermarking and compression) of a single 256 × 256 pixels video frame (I frame). MB stands for macro-block of 8 × 8 pixels.

For the present case, in which real-time video streams of size 256 × 256 pixels/frame are used, the number of cycles it would take to watermark one frame would be 65,792 clock cycles. Thus, the processing time it takes to watermark each frame is 1.6 ms (607 frames/s) at a clock frequency of 40 MHz. For video streams of 640 × 480 pixels/frame, the processing time would be 7 ms/frame (130 frames/sec). This means that, if this watermarking system is employed in practical applications (e.g. in a surveillance video system) with an input video of 30 frame/sec, the implemented video

watermarking system can watermark the video stream in real time. The reason of this high frame rate is the pipelining architecture of the watermarking system along with the video encoder.

3.5.5 Robustness Analysis:

As stated before, the watermarking system described in this paper is designed for invisible semi-fragile watermarking which is allowed to withstand certain legitimate manipulations i.e. mild lossy compression of images, but is capable to reject malicious changes i.e. cover-up attack, cropping segment removal, etc. To demonstrate the robustness of the watermark, two sample video sequences were embedded with watermark according to the proposed algorithm. The cover-up attack is applied to the watermarked sample video. The tampered video is analyzed by the watermark detector, which outputs a detection map. The detection map is used to indicate which blocks of the image are suspected as inauthentic.

The results of the tamper detection of the video sequences are presented in Figure 3.16 and Figure 3.17. Only one frame of video samples is shown here as an example. The following tampering was done in the first video sequence (Figure 3.16).

1. The book in the hand of the person was removed by copying the contents of adjacent blocks onto the blocks where the book was in the original frame. To an innocent observer the original existence of the book in the video frames is visually un-detectable.

2. The segment containing the electrical outlet on the wall under the white board was also removed. Hence the electrical outlet was also undetectable in the video frames.

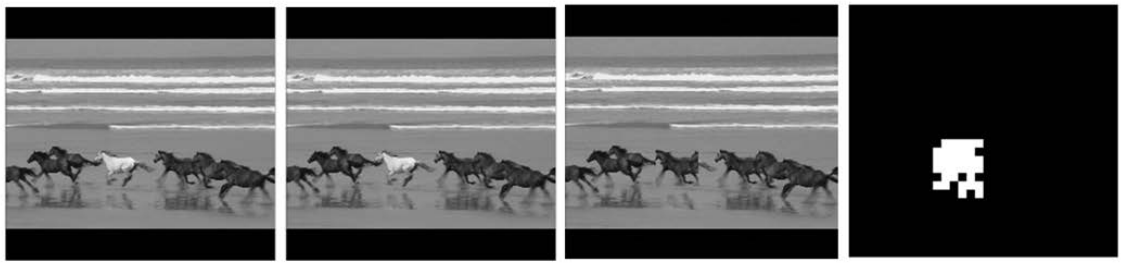
In the second video sequence (Figure 3.17), the portion of the white horse was covered-up by a black horse, so for an innocent observer it seems that there was no white horse in the frame. The detection map successfully illustrated the blocks of the tampered video frame which are suspected as inauthentic.

It is important to understand that the proposed watermarking system embeds the watermark into MJPEG video frames and enables the detection of tampered video frames. The detection algorithm is run in software program by using a similar algorithm utilized for the watermark embedding process as described in Chapter 2. The tampering or modifications can be easily noticed in the detection map created by the watermark detector and presented in Figure 3.16(d) and Figure 3.17(d).

Since the watermarking is performed after the quantization of the DCT coefficients, the video watermarking system allows mild lossy compression on the watermarked video. The algorithm allows an equal or lower degree of compression that was performed before the watermarking process.

a) b) c) d)

Figure 3.16 Tamper detection of video sample-1: (a) Original video (b) Watermarked video (c) Tampered video (segment removal) (d) Detection map



a) b) c) d)

Figure 3.17 Tamper detection of video sample-2: (a) Original video (b) Watermarked video (c) Tampered video (cover-up attack) (d) Detection map

3.5.6 Security issues

The randomness of the watermark sequence is mainly dependent on the statistical property of the pseudorandom sequence generated by the PRNG module since the primitive watermark is XOR'ed by the pseudorandom sequence. Hence it is necessary that the pseudorandom sequence should have statistical properties as close as possible to a true random sequence [43]. A true random binary sequence has equal distribution of 1's

and 0's. Another indication of the randomness of the sequence is the auto-correlation values of the sequence. This feature is crucial for the resistance of the sequence to correlation attacks [43].

In order to evaluate the statistical quality of the pseudorandom number sequence generated by the Gollmann cascade F-FCSR PRNG module, designed in [43], two tests were carried out. First, a comparison between the auto-correlation values of the proposed Gollmann Cascade F-FCSR PRNG and F-FCSR PRNG (which is well studied and known to have good statistical properties [39]) with a similar size, was done. The second evaluation was done using the statistical test suite available through the National Institute of Standards and Technology (NIST) [34]. NIST provides a comprehensive tool to verify the statistical quality of a pseudorandom sequence through various tests. These tests check the properties of the sequence and compare the results with the expected values of a true random sequence [34]. The tests were all successful and proved that the proposed PRNG meets the standards compared to other published PRNGs, while providing a simple design methodology [43].

However, the cryptographic property of GCF-FCSR is completely dependent on F-FCSR. Recently in [68]-[69] *Hell et al*. presented a severe cryptanalytic attack on the F-FCSR stream cipher family. Breaking of F-FCSR made the design of Gollmann Cascaded F-FCSR vulnerable since F-FCSR is the primary building block of GCF-FCSR. Moreover, the secret key used for initializing the GC-FFCSR is 25 bits long only. This is not long enough to withstand a brute-force attack. Hence, we require a new CSPRNG for our hardware based video watermarking system, a PRNG that has been specially designed to be resilient to certain cryptographic attacks. This problem is to be addressed

in Chapter 5 of this dissertation where we have focused on designing a video watermarking system with improved security.

3.6 Comparison with Existing Research

The proposed hardware-based MJPEG video watermarking system features minimum video frame quality degradation with no visually perceptible artifacts and high PSNR values. However, the complexity of the proposed algorithm is lower since only the intra-frames are modified and the results shown are achieved with only two DCT coefficients in each 8 × 8 block being changed. Furthermore, the watermark embedding that is designed to be performed directly in the compressed video streams minimizes computationally demanding operations. Table 3.2 presents a comparative perspective with other published hardware-based video watermarking system. The proposed watermarking system is the first that employs invisible semi-fragile watermark approach in the frequency domain (DCT) for video streams with fewer logic gates and higher processing speed compared to other works in [13], [15]-[16], [21], [23] and [25].

Table 3.2 Comparison of our MJPEG video watermarking system with other published works on video watermarking

Research Works	Design Type	Type of WM	Video Standard	Processing Domain	Chip Statistics		
					Logic Elements (Kilo gates)	Clock Frequency and processing speed	PSNR
Strycker et al. [13]	DSP board	Invisible-Robust	-	Spatial	N/A	100 MHz	N/A
Mathai et al. [15]	Custom IC-0.18µm	Invisible-Robust	-	Wavelet	N/A	75 MHz @ 30 fps (320 x 320)	40 dB
Tsai and Wu [16]	Custom IC	Robust	Raw Video	Spatial	N/A	N/A	N/A
Maes et al. [21]	FPGA Custom IC	Robust	MPEG	Spatial	17 14	N/A	N/A
Petitjean et al. [25]	FPGA Board DSP Board	Invisible-Robust	MPEG	Fractal	N/A	50 MHz takes 6 *us* 250 MHz takes 118 *us*	N/A
Mohanty et al. [23]	FPGA	Visible	MPEG-4	DCT	28.3	100 MHz @ 43 fps (320 x 240)	Around 30 dB
This MJPEG WM work [28]	FPGA	Semi-fragile	MJPEG	DCT	9.263	40 MHz @ 607 fps (256 x 256) 40 MHz @ 130 fps (640x 480)	44 dB

3.7 Summary

The Design of the hardware architecture of a novel digital video watermarking system to authenticate video stream in real time is presented in this chapter. To demonstrate the fact that the designed video watermarking system works well with a video encoder system we have validated the designed system with non-standard MJPEG video encoder system. It was also demonstrated that the designed watermarking system is capable to withstand certain potential threats without significant video frame quality degradation. However, the designed hardware based watermarking system also features low power consumption, low cost implementation, high processing speed and reliable watermarking on video sequence.

However, the described video watermarking system needs to be validated with the state-of-the-art MPEG-4 video encoding standard, which features motion estimation and motion compensation based prediction of video frames to provide advanced and better compression than the non-standard MJPEG video encoding standard does. The challenge here is to demonstrate that the proposed video watermarking system is capable to watermark a video stream during the commonly used MPEG-4 encoding process in real-time without significant increase in power dissipation and logic gate requirements. Chapter 4 concentrates on this issue.

Moreover, there are some security flaws in the Gollmann Cascaded F-FCSR, used as the PRNG to generate the watermark sequence in the system. A cryptographically secure PRNG is necessary for our hardware based video watermarking system. These security issues are discussed in Chapter 5, where we have focused on designing an improved security video watermarking system.

Chapter Four: **Low Power and Low Hardware Real-time MPEG-4 Video Watermarking System**

4.1 Introduction

This work presents the design and implementation of a low power and low cost hardware watermarking system that can insert an invisible and semi-fragile watermark into state-of-the-art MPEG-4 encoded video streams captured by a CMOS image sensor based surveillance video camera. The watermark insertion is processed in the Discrete Cosine Transform (DCT) domain. The designed video watermarking system is intended to be built inside a CMOS image sensory based camera system for enabling the real-time authentication of the surveillance video footage. The watermarking system works efficiently with the commonly used MPEG-4 encoding system without high penalty in system power consumption (only 10mW) and hardware resources (0.8% increase in logic cells and 0.2% increase in memory). It maintains a high throughput to watermark a standard resolution MPEG-4 video stream in real-time. This hardware watermarking system also features insignificant video quality degradation and it enables the detection of certain potential attacks such as cover-up, cropping, sharpening, blurring and segment removal, etc. on the original video stream. The proposed watermarking system was prototyped on an Altera® Cyclone FPGA system as a proof of concept [29].

The watermarking method and implementation described in this chapter represents the next step in our goal of a hardware efficient and secure real-time video watermarking system. We move from MJPEG watermarking to MPEG-4, and show that we can maintain a similar level of performance and hardware efficiency. In this chapter,

we still use an insecure PRNG. Incorporating a cryptographically secure PRNG is the final component to be included, and is described in the next chapter.

4.2 Contribution

We focus on the design and implementation of a hardware based real-time MPEG-4 video watermarking system which can be integrated within the CMOS image sensor based camera system to enable the real-time authentication of surveillance video footage, with insignificant increase in system power consumption and hardware utilization. The main contributions of the current work are as follows:

1. To present an efficient design and implementation of a hardware digital video watermarking system which can be integrated with a commonly used MPEG-4 video encoding system for a CMOS imager based sensory system.
2. To verify that the designed watermarking system works efficiently without high penalty in system power consumption and hardware resources. To show that the designed watermarking system consumes low power and minimum additional logic and memory resources than that of MPEG-4 video encoder system.
3. To ensure that the watermarking system is suitable for real time video (with sufficient frame rate), which can be easily incorporated with state-of-the-art MPEG-4 video standards.
4. To show that the watermarking will feature insignificant video quality degradation and will be able to withstand certain potential attacks, i.e. cover-up attack, cropping, segment removal, blurring etc. on the video stream.

4.3 Watermarking in MPEG-4 Video Encoding Framework

We have implemented our proposed digital watermarking system with a simple profile MPEG-4 encoder in hardware, to demonstrate that the designed video watermarking system works successfully when integrated with a MPEG-4 encoder system.

An MPEG-4 encoder system consists of various advanced video encoding features for improved video compression. To simplify the MPEG-4 encoder implementation, we have taken some concepts from the Xilinx® MPEG-4 Part 2 Simple Profile Encoder (MPEG-4 Encoder) core, which is a fully functional VHDL design implemented on a Xilinx FPGA [70]-[72]. The implemented MPEG-4 encoder has the following basic features:

1. Inter frame prediction
2. Inter frame Motion estimation and compensation
3. Fixed (8×8) block size motion prediction
4. Integer pixel motion vector search

It does not have advanced features of video encoding like the following:

1. Intra frame motion prediction
2. Variable block size motion prediction
3. Half/Quarter pixel motion vector search and etc.

The entire watermarking process during MPEG-4 encoding is detailed below:

1. RGB video frames are converted to YC_bC_r frames.
2. Y component frames are separated from C_b and C_r frames.
3. Only I frames of the Y components are watermarked, not the P frames.
4. I frames (reference frame) of the Y components are passed to DCT-Quantization unit, where they are split into 8×8 blocks, subjected to 2D DCT process, quantization process and zigzag scanning. After that, I frames are passed to the watermark insertion unit in 8×8 blocks, where a watermark sequence is inserted into each 8×8 DCT blocks, stored in a buffer and passed to the entropy coder.
5. Watermarked I frames are processed through 2D Inverse DCT (IDCT) and de-quantization process for reconstruction. The reconstructed I frames are stored in reconstructed reference frame buffer.
6. P frames and the reconstructed reference frame (I') are used in motion estimation (ME) process for the determination of motion vectors (MV). Motion vectors are stored in buffer and used for generating the motion compensated reference frame (I''). Stored MV's are passed to the entropy coder.
7. Prediction error or prediction differences (PE) are estimated from the block based differences between the current P frame and motion compensated I'' frame. PE's are processed through 2D DCT and quantization. Then they are passed to the entropy coder.
8. Entropy coding is performed of the quantized DCT coefficients of P frames and watermarked I frames using MVs and thus watermarked MPEG-4 bit stream is generated.

9. For C_b and C_r frames, step-4 to step-8 are followed except the watermarking of I frames in step-4.

Watermarking is embedded only in I frames but not in the P frames. This is because P frames are already predicted from the reference I frames, so to embed the watermark into I frames is sufficient.

4.4 Hardware Architecture

The system level hardware architecture for the proposed MPEG-4 video watermarking system is shown in Figure 4.1. The proposed system architecture includes the following modules:

4.4.1 CMOS Based Video Camera

A CMOS image sensor based video camera captures the video footage and sends the video frames to the MPEG-4 watermarking system.

4.4.2 Colorspace Conversion Block

RGB video frames are converted to Y, C_b, C_r components. Y, Cb, Cr component video frame data are stored in the memory buffer and separately sent to the MPEG-4 encoder. Only Y component frames are MPEG-4 encoded and watermarked; C_r and C_b component frames are only encoded but not watermarked. The conversion is processed using the following relations [73]-[74]:

$$\boldsymbol{Y = 0.299R + 0.587G + 0.114B} \tag{4.1}$$

$$\boldsymbol{C_b = 0.564(B - Y) + 128} \tag{4.2}$$

$$\boldsymbol{C_r = 0.731(R - Y) + 128} \tag{4.3}$$

Colorspace conversion block requires six adders and five multipliers. Usually in MPEG-4 4:2:0 color sampling is chosen, but for simplicity we chose 4:4:4. Hence we did not require to down-sample the C_r and C_b components.

4.4.3 Input Frame Buffer

The input frame buffer stores the incoming video frames from the video camera after the conversion of the video frame from RGB colorspace to YC_bC_r colorspace. The frame data are read out to the MPEG-4 encoder system in 8×8 blocks. Onboard SRAM on an Altera® Cyclone FPGA board is used as the input frame buffer.

4.4.4 Frame Sorter

The frame sorter recognizes I frame data and P frame data and directs them into two different streams. It handles 8-bits of data. It is basically a counter which keeps track of the frame count and thus recognizes I frames and P frames in the video sequence.

4.4.5 2D DCT and Quantization

This block performs 2D DCT operation on each 8×8 block data of I frames and P frames. After that, the quantization operation is performed on each of the 8×8 blocks. The output data is stored temporarily in a 64×1B data buffer for storing 64 DCT coefficients after the compression (DCT and Quantization) of I or P frames.

4.4.6 De-quantization and 2D Inverse DCT

This block performs the de-quantization operation on 8×8 block data of watermarked I frame data stored in output buffer. After that 2D Inverse DCT operation is

performed on each of the 8×8 blocks. The output data is stored temporarily in a 1x64 data buffer for storing 64 DCT coefficients after the DCT and quantization operation.

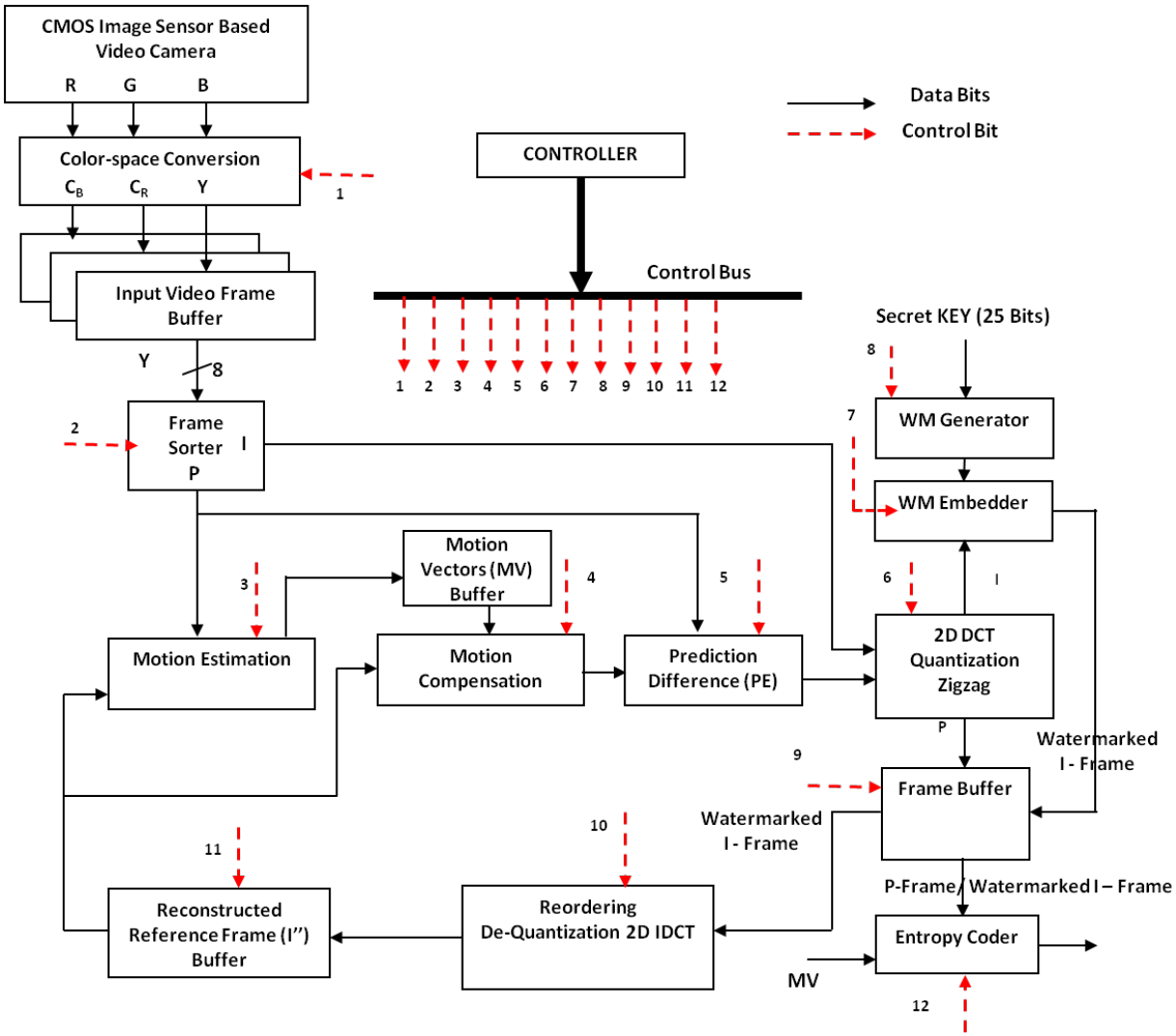


Figure 4.1 System level hardware architecture of proposed MPEG-4 video watermarking system for CMOS image sensor based video camera

4.4.7 Output Buffer

This a 64×1B data buffer which stores the watermarked I frame data or compressed P frame (prediction difference) data in 8×8 block manner.

4.4.8 Motion Estimation

Motion estimation is the process of estimating motion vectors that describe the transformation from one 2D image to another, usually from adjacent frames in a video sequence. Different block based search algorithms can be used for this process. The three step search (TSS) algorithm is used in our prototype implementation for lower computational complexity. It uses P frame data and reconstructed reference frame data to estimate motion vectors. Motion vectors are stored into the onboard SRAM.

4.4.9 Motion Compensation

Motion Compensation is the process where the index or positions of the contents of reference frame are adjusted according to the motion vectors corresponding to the macro-blocks of the current frame. Motion compensation is done by shifting the index of the current macro-block according to the motion vectors.

4.4.10 Predictive Difference Block

These are two 8 bit subtracter blocks. Predictive difference (PE) is calculated between current P frame data and motion compensated reference frame (I’’) data using this subtracter block.

4.4.11 Watermark generator

The watermark generator used here is similar to the one that is used in previous chapter. The only difference is that, here we use the primitive watermark sequence directly without scrambling it. The primitive watermark sequence used here also consists of a meaningful data i.e. time, date, camera ID and frame serial number and is XOR'ed directly with a pseudorandom sequence generated by a PRNG. Since we are not scrambling the primitive watermark sequence, we do not need the extra key for the scrambling operation. As in the previous chapter, a Gollman Cascaded Filtered Feedback with Carry Shift Register (GCF-FCSR) is used as the PRNG. The secret initialization key of the PRNG used here is also 25-bits long. We have used this PRNG for the purpose of demonstrating feasibility and hardware efficiency, although this PRNG is still cryptographically insecure. A cryptographically secure PRNG will be included in the next chapter.

4.4.12 Watermark Embedder

A schematic view of the hardware architecture for the watermark embedder unit is presented in Figure 4.3. The watermark embedder performs the steps of the watermark algorithm summarized in Section III. It works in two phases (calculation and embedding). When considering a cycle of embedding the watermark in one 8×8 block of pixels, each phase takes one block cycle. Two blocks are processed simultaneously in a pipelined manner so that the embedding process only requires one block cycle.

The design of the watermark embedder supports pipelining architecture so that two or more 8×8 macro-blocks inside a single video frame or more than one frame can be

watermarked simultaneously. This feature has a great impact on increasing the watermarking speed; hence watermark embedding process does not affect the video frame rate and ensures the real-time watermarking of the video frames.

4.4.13 Controller

The control unit generates control signals to synchronize the overall system's functionalities. As shown in Figure 4.4, it is implemented using a finite state machine (FSM) with eight different states in it. Table 4.1 shows the functions of different states of the finite state machine used in the controller unit.

4.4.14 Entropy Coder

It is implemented using a Huffman encoder. It uses the watermarked and compressed I frame data 8×8 blocks or compressed P frame data 8×8 blocks and motion vector data. Final output is the watermarked and MPEG-4 encoded bit stream.

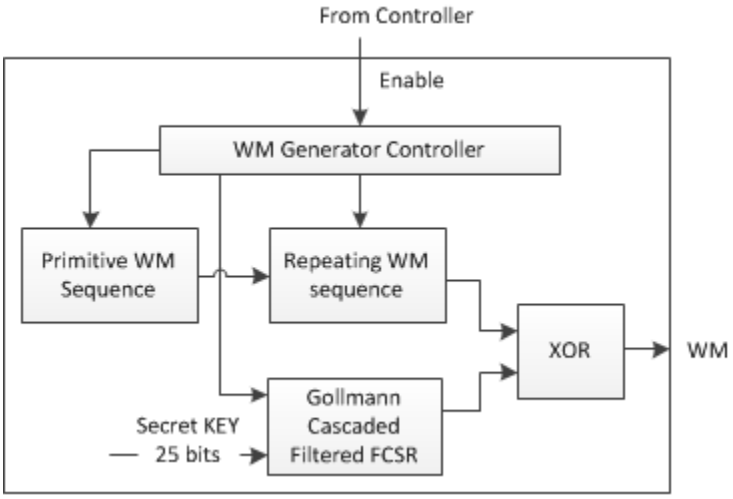


Figure 4.2 System level architecture of watermark generator unit

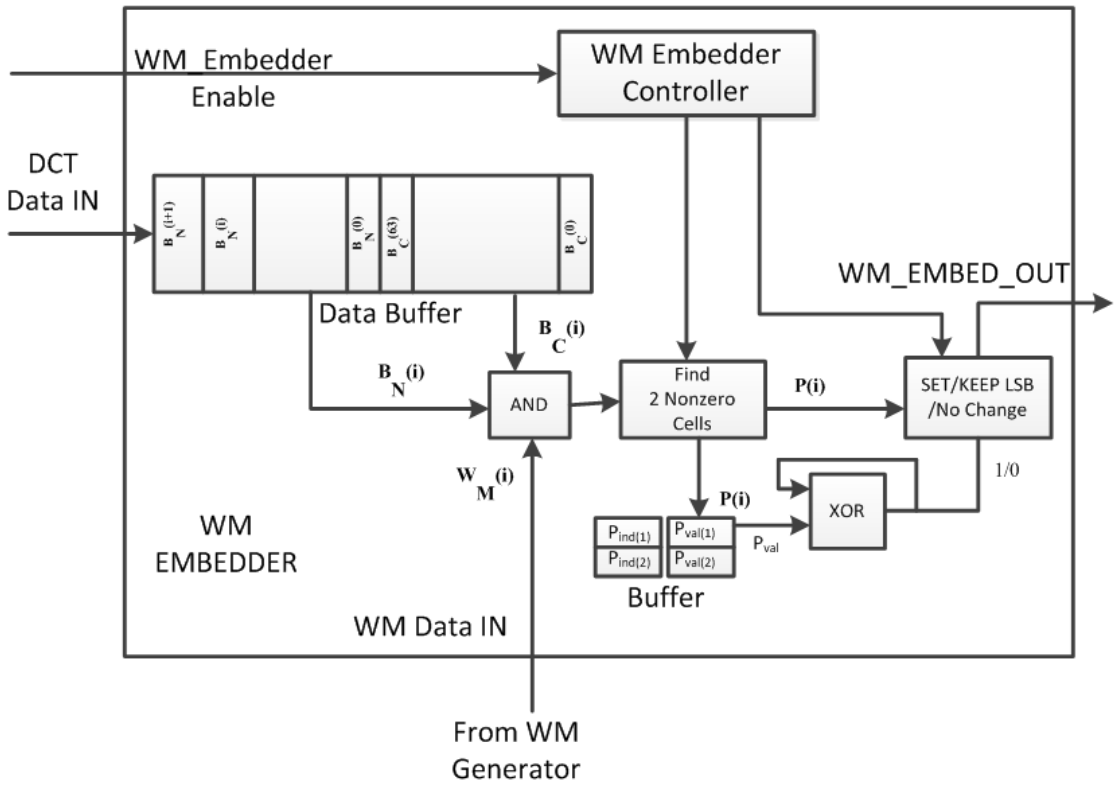


Figure 4.3 System level architecture of watermark embedder unit

4.5 Architecture Prototyping

Each unit in the designed video WM system with MPEG-4 encoder (as described in Section 4.4 and illustrated in Figure 4.1) has been simulated, implemented and tested individually, and then integrated together to obtain the final system architecture. The entire design flow of the architecture prototyping is shown in Figure 4.5.

4.5.1 MATLAB® Prototyping

To verify the functionality of the algorithms and effectiveness of the WM system, a prototype was built in MATLAB®, a high-level language and interactive environment for numerical computation, visualization, and programming. The prototype was simulated and tested with video sequences to verify the ability of the WM system.

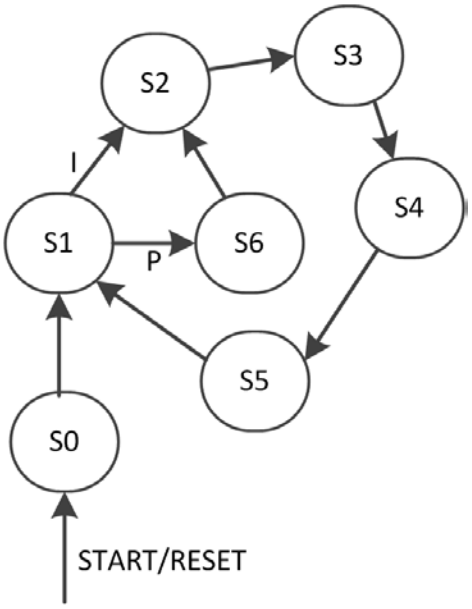


Figure 4.4 Finite State Machine (FSM) of the controller unit

Table 4.1 States of the FSM of the controller unit

States	Functions
S0	Perform Colorspace conversion from RGB to YCbCr.
S1	Sort I frame and P frame.
S2	Initiate WM generator and initiate GCFFCSR to generate pseudo-random sequence; Initiate performing 2D DCT, Quantization and Zigzag scanning.
S3	Initiate WM embedding.
S4	Initiate performing entropy coding and generate watermarked MPEG-4 bit stream.
S5	Initiate performing inverse zig-zag ordering, de-quantization and 2D Inverse DCT process of the watermarked and compressed I frame 8×8 data blocks.
S6	Initiate Motion Estimation process and store Motion Vector data in SRAM.
S7	Start Motion Compensation process.
S8	Perform prediction difference (PE) calculation.

4.5.2 MODELSIM® Simulation

The VLSI architecture of the designed video WM system was described using Verilog HDL. Each unit of the entire WM system with MPEG-4 encoder was designed using Verilog HDL to build sub-module. Different Test-bench was created to verify the functionality of the units separately.

Overall WM system was built by integrating all the Verilog described sub-modules together. Test-bench was created to verify the functionality of the overall system. To verify the functionality we have used MODELSIM® (an advanced HDL simulation and debugging software). The design of the hardware system was debugged using the timing diagram generated by the MODELSIM®.

4.5.3 FPGA Prototyping

The Verilog HDL description of the overall video watermarking system with MPEG-4 encoder was compiled and synthesized using Altera® Quartus - II design and synthesis software. The generated synthesis report is shown in Table 4.2. Finally, an Altera® EP1C20 Cyclone FPGA chip was programmed by using Altera® Quartus II using HDL description of overall WM system with MPEG-4 encoder.

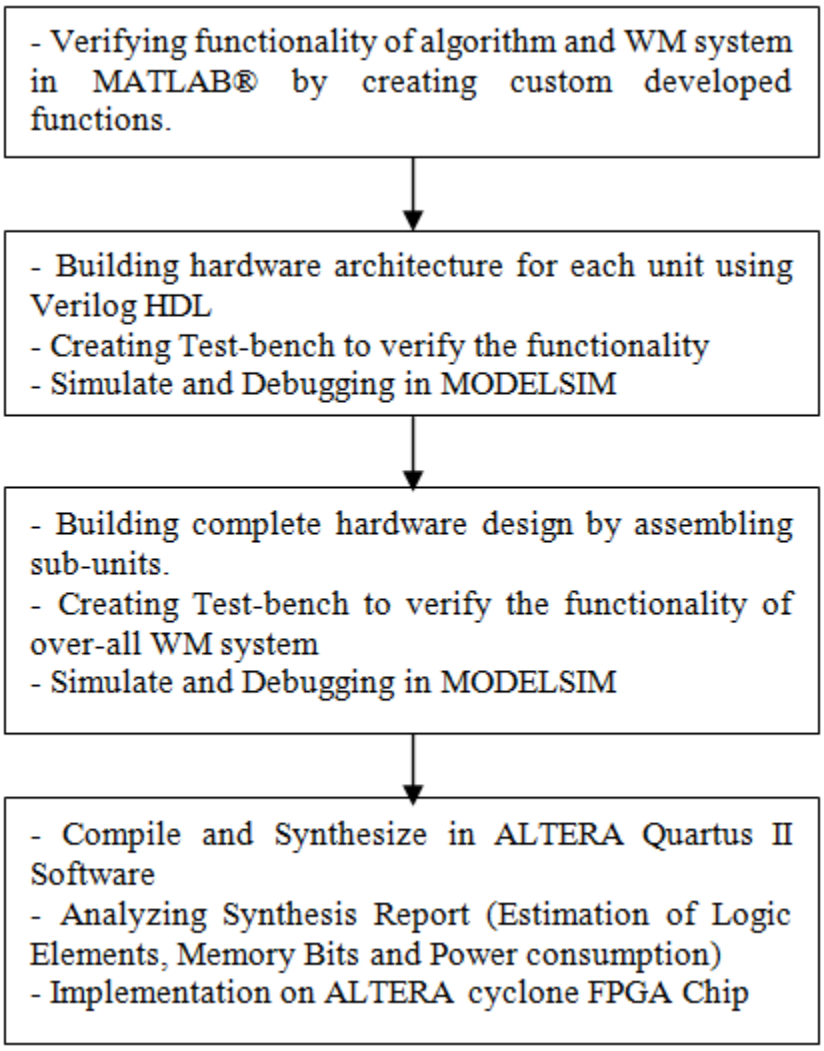


Figure 4.5 Design flow of the architecture prototyping

4.6 Performance Analysis

The performance of the implemented MPEG-4 video WM system was evaluated in terms of hardware cost, power consumption, processing speed, video quality degradation and sample attack detection.

4.6.1 Hardware Cost

The FPGA synthesis report is shown in Table 4.2. The results clearly indicate that the addition of the watermark generator and watermark embedder units caused very small increase in logic cells (0.8% increase) and in memory resource usage (0.2% increase) when compared to the logic cells and memory requirement of the MPEG-4 video encoder unit itself. Any device that is large enough for the implementation of the MPEG video encoder module would be able to accommodate the additional hardware required for the watermark generator and watermark embedder system for enabling the feature of authentication of captured video footage.

4.6.2 Power Consumption

The power consumption of the entire video watermarking system with MPEG video encoder implemented on the FPGA device was analyzed using the Powerplay Power Analyzer Tool of Altera® Quartus II. The power analysis result is also shown in Table 4.2. The overall MPEG-4 video encoder system integrated with the watermarking system (watermark generator and watermark embedder unit) consumes 650 mW of power, whereas the watermarking system (watermark generator and embedder) itself consumes only 10 mW power only 1.5% of the power dissipated by the overall

architecture). This ensures that the addition of the watermarking system does not significantly affect the total power consumption.

Another important fact is that using higher resolution video will not affect the power consumption and the hardware utilization of the watermarking system because the watermarking scheme operates on a single 8×8 block processing at a time regardless of the video resolution. According to the watermarking algorithm, buffering two 8×8 blocks is sufficient to run the system irrespective of the entire image size. This feature ensures that the proposed system will work successfully on any standard resolution of video stream i.e. 320 × 240, 640 × 480, 1280 × 720 (HD video).

4.6.3 Processing Speed

The data of each frame is processed based on a single 8×8 block at a time. Only I frames will be watermarked but not the P frames. Moreover, only the prediction of P frames encounters the motion estimation and compensation process.

In the MPEG-4 video watermarking system, processing of I frames (which are to be encoded and watermarked), each of the macro-block (8 x 8 pixels) passes through the following operations: DCT, Quantization, zigzag scanning, watermark insertion, reordering and entropy coding. For the initial 8×8 macro-block of a particular I frame, excluding the watermark insertion process it takes 252 clock cycles of processing time. For all the later 8×8 blocks, it takes 64 cycles each (throughput). The WM embedder takes 64 clock cycles for inserting the watermark into each 8×8 block. As a whole, including the watermarking the processing of the initial 8×8 block of a particular I frame requires around 320 clock cycles. The WM generator takes 64 clock cycles latency to

generate the WM data, which is completed by the processing of I frames (252 clock cycles) and after that WM sequence is generated at a speed of 1bit/cycle. Hence the WM generator does not contribute to the initial latency of each I frame processing period.

Table 4.2 FPGA synthesis result for MPEG-4 video watermarking system implementation

Components	**Logic Cells**	**Memory bits (including buffers)**	**Power consumption**
MPEG-4 Video Encoder	54K	614 KB	640 mW (approx.)
Watermark Generator	309	512	10mW
Watermark Embedder	132	744	
Overall Architecture	55 K	615KB	650 mW (approx.)

Table 4.3 Estimated initial latency and processing speed (for 320 × 240 pixels)

Components	**Initial Latency* (clock-cycles)**	**Total Time** (Clock-Cycles)**	**Processing Speed (fps)**
Simple profile MPEG-4 Video Encoder	I frames: 252 P frames: 1890	76,988 757,890	52
Watermark Generator	64	N/A	519
Watermark Embedder	64	N/A	
Overall Architecture (MPEG-4 Encoder with Watermarking)	I frames: 320 P frame: 1890	77,056 757,890	52

* Processing time for the initial 8×8 block.

** Processing time for the whole frame (320×240 pixels in this case).

Processing of P frames does not include watermarking procedure because each of the P frames is predicted from the preceding I frame, which is already watermarked. Processing of P frames includes the following: motion estimation, motion compensation, DCT, quantization, zigzag scanning, entropy coding. Motion estimation and compensation process require a large amount of time during the processing of P frame. Hence the processing of a single P frame will require longer time than that of a single I frame. The processing time of a single video frame of N× M pixels resolution can be estimated by the following equation:

$$\mathbf{T} = \left[\mathbf{Latency} + \left(\frac{\mathbf{MN}}{\mathbf{8\times 8}} - \mathbf{1}\right)\mathbf{Throughput}\right]\frac{\mathbf{1}}{\mathbf{clock\_frequency}} \qquad \textbf{(4.4)}$$

For a video stream of 320 × 240 pixels resolution, the number of clock cycles it takes to watermark and to encode a single I frame will be 77,056. Thus, at a clock frequency of 40 MHz the time for processing a single I frame is 1.9 ms (519 fps).

In the implementation of a simple profile MPEG-4 encoder in our work, the processing time for encoding a single P frame is 757,890 clock-cycles. Thus the frame rate is 52 fps (at a clock frequency 40 MHz). Hence, it is clear that the processing time of P frames dominates the entire processing time of the MPEG-4 encoding of video streams. Table 4.3 demonstrates the initial latency required for different units of the system. It also shows that at a clock frequency of 40 MHz, the watermarking rate is 519 fps and encoded video frame rate is 52 fps.

Table 4.4 shows how the watermarking (frame) rate changes for higher resolution of video sequences. If this MPEG-4 video watermarking system is integrated within a surveillance video camera system with a standard video at 30 fps, the implemented

watermarking system can watermark the video stream in real time even for higher resolution video sequences i.e. 640 x 480, 1280 x 720 (HD). The pipelining architecture and parallelism of the designed system helped in achieving this high throughput after the initial latency state.

Table 4.4 Estimated watermarking speed (frame/sec) for different video resolutions at 40 MHz of clock frequency

Video Resolution	Clock Cycles required	Watermarking Speed (frame/sec)
320 × 240	77056	519
640 × 480	307456	130
1280 × 720 (HD)	921856	43

4.6.4 Video Quality Degradation (PSNR analysis)

In order to evaluate the performance of the watermarking system for MPEG-4 encoded video, the algorithm was tested with a color video clip which has significant high-frequency patterns. The video stream was of 320 × 240 pixels/frame resolution and it was captured by a video camera at a frame rate of 30 fps.

For this sample video stream, a comparison was made between two sets of experimental results: original video frames vs MPEG-4 encoded video frames and original video frames vs watermarked MPEG-4 video frames using the designed MPEG-4 video watermarking system. The comparisons were performed using the standard video quality metric: Peak Signal to Noise Ratio (PSNR), measured in decibels (dB) and is computed using the following equations:

$$\boldsymbol{PSNR = 10\log_{10}\left(\frac{255}{MSE}\right)} \tag{4.5}$$

$$\mathbf{MSE} = \frac{\mathbf{1}}{\boldsymbol{MN}}\sum_{\boldsymbol{m=0}}^{\boldsymbol{M-1}}\sum_{\boldsymbol{n=0}}^{\boldsymbol{N-1}}[\boldsymbol{f(m,n)} - \boldsymbol{k(m,n)}]^{\mathbf{2}} \tag{4.6}$$

Here, f and k are the two frames (of $M \times N$ pixels) in comparison. We have tested video clips of 320 × 240 pixels.

In terms of PSNR, the quality of the watermarked frames is maintained above 40dB, measured consistently with no visually perceptible artifacts. Four sets of video group of pictures (GOPs) consisting of five consecutive frames (one I and four P frames) in each of them were tested.

As stated before, P - frames are predicted and encoded based on I frame in each of the GOPs, hence the PSNR value of a single I frame is expected to be higher than that of the following P frames in each of the GOPs. Figure 4.6 demonstrates the PSNR comparison between MPEG-4 encoded (without watermark) video frames vs original video frames. The 1st, 6th, 11th and 16th fame are I frames. Their PSNR values are higher (~43 db) than those of the following P frames as expected. But the PSNR of the P frames are gradually decreasing from frame to frame.

Figure 4.7 demonstrates the PSNR comparison between watermarked MPEG-4 encoded video frames vs original video frames. It is clear that watermark embedding did not degrade the quality of the video streams compared to the MPEG-4 encoded video streams without watermark. The PSNR of the P frames remained same since the watermarking did not change the P frames. On the other hand, the PSNR of the I frames are slightly less than those in Figure 4.6 but still above 40 dB which proves that the watermarking did not have significant degradation in the quality of the video stream.

4.6.5 Tamper Detection

Our MPEG-4 video watermarking system is designed for watermarking CMOS image sensor based surveillance camera video footage for authentication purposes. It involves invisible semi-fragile watermarking which permits certain legitimate changes like mild lossy compression in the video stream, but is capable to detect malicious changes i.e. cover-up attack, cropping, segment removal, blurring, etc.

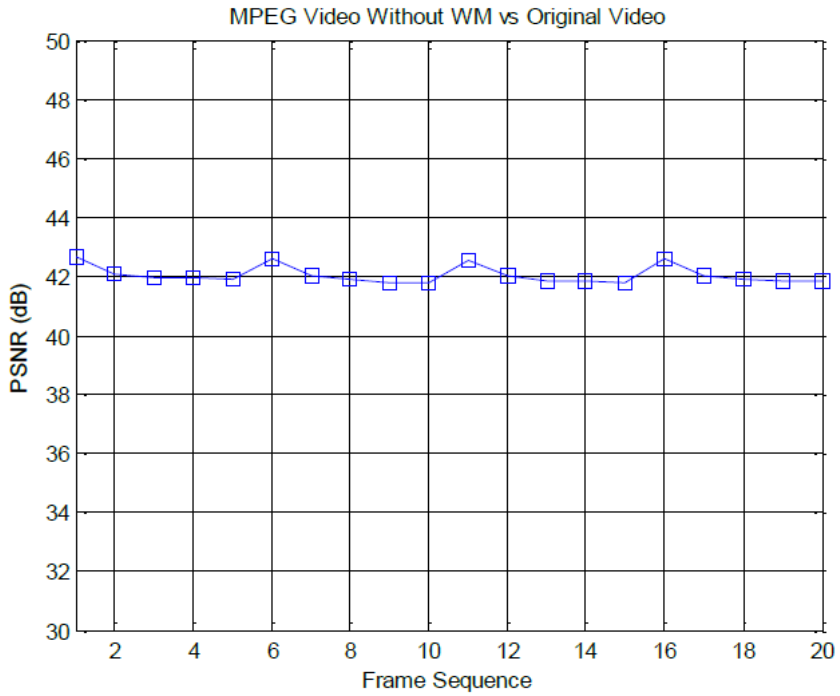


Figure 4.6 PSNR comparison for MPEG-4 encoded (without watermark embedded) video vs original video sequence

To demonstrate the ability of the designed watermarking system to detect these kinds of attacks, sample video sequence was embedded with watermark using the implemented MPEG-4 video watermarking system according to the proposed algorithm. The cover-up attack is applied to the watermarked sample video. The tampered video is

analyzed by the watermark detection algorithm, which outputs a detection map. The detection map indicates which blocks of the image are detected as inauthentic.

The results of the tamper detection of the video sequences are presented in Figure 4.8. Only one frame of the sample video stream is shown here as an example. The portion of the white horse in the video frame was covered-up by a black horse, so for an innocent observer it seems that there was no white horse in the frame. The detection map successfully illustrated the detection of the tampered blocks in the video frame which are suspected as inauthentic (Figure 4.8d)

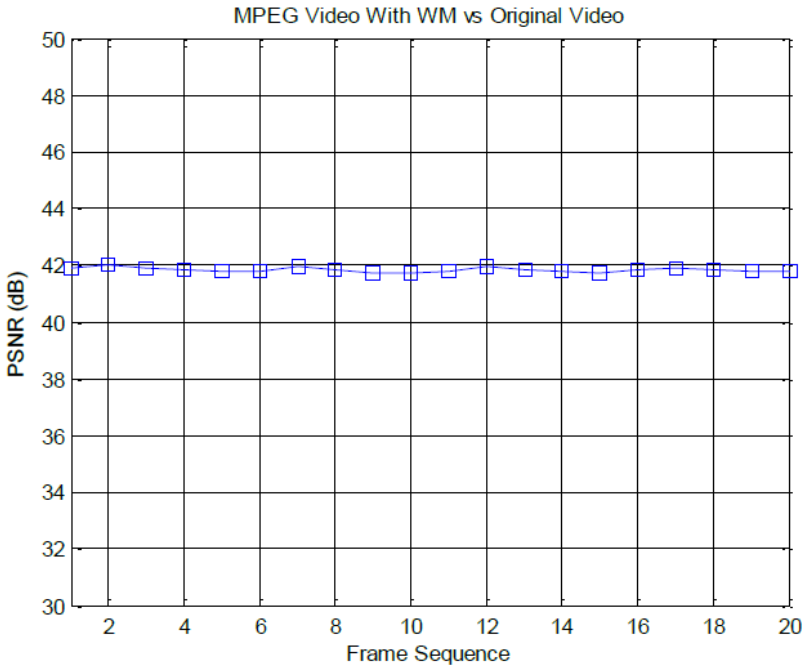


Figure 4.7 PSNR comparison for MPEG-4 encoded (with watermark) video vs original video sequence

Figure 4.9 also demonstrates another sample video of a parking lot, captured by a security camera. The video sample was also watermarked and MPEG-4 encoded using the designed hardware MPEG-4 video watermarking system. From the lower-left corner of the video frame a segment of a black car was removed (Figure 4.9c). This segment-removal was also successfully detected in the detection map illustrated in Figure 4.9d.

Similarly, Figure 4.10 shows the detection of blurring. A portion of the watermarked MPEG-4 video frame was blurred. This modified the watermark embedded in the frame and enabled the detection of the blurring effect in the detection map. Our MPEG-4 video watermarking system presented here allows mild lossy compression of the watermarked video. This ability is demonstrated in the following chapter, where we have assessed the proposed watermarking system based on different watermarking attacks and a proposed threat model for real-time surveillance video authentication.

Figure 4.8 Tamper Detection of Cover-up attack: (a) Original video (b) Watermarked and MPEG-4 encoded video frame (c) Cover-up attack on watermarked and MPEG-4 encoded frame (d) Detection map. Sample video resolution is 320x240 pixels

a) b) c) d)

Figure 4.9 Tamper Detection of Segment- Removal attack: (a) Original video (b) Watermarked and MPEG-4 encoded video frame (c) Segment-removal attack on watermarked and MPEG-4 encoded frame (d) Detection map. Sample video resolution is 320x240 pixels

a) b) c) d)

Figure 4.10 Tamper Detection of the blurring effect: (a) Original video (b) Watermarked and MPEG-4 encoded video frame (c) Blurring on watermarked and MPEG-4 encoded frame (shown by the red rectangle) (d) Detection map. Sample video resolution is 320x240 pixels

4.7 Comparison with Other Works

Table 4.5 presents a comparison of our MPEG-4 video watermarking system with other published works on hardware-based video watermarking system [14], [23]-[25]. Our MPEG-4 based hardware video watermarking system employs invisible semi-fragile watermarking into the video stream captured by a CMOS image sensor based surveillance camera. The watermarking process is performed in the frequency domain (DCT) which enabled us to integrate the designed watermarking system with the contemporary and commonly used MPEG-4 encoding system inside a CMOS image sensor based camera. Moreover, the implementation of the MPEG-4 video watermarking system was done with fewer hardware resources and higher watermarking speed compared to other published works in [14], [23]-[25].

Our designed watermarking system is capable of watermarking the video stream during the MPEG-4 video encoding process before the transmission of the video stream from the CMOS camera. Hence this work clearly is a big step forward from our previous work in the same field [24], where we focused on watermarking a non-standard MJPEG video stream. Another important fact is that, the target application for our current MPEG-4 video watermarking system is to watermark the surveillance video camera footage to enable the authentication feature of the captured video stream whereas the works presented in [14], [23]-[25] are focused on copyright protection of the video.

Table 4.5 Comparison of our MPEG-4 video watermarking system with other published works on hardware video watermarking

Research Works	Design Type	Type of WM	Application	Video Standard	Processing Domain/method	Design Statistics		
						Logic Elements	Clock Frequency and watermarking speed (fps)	PSNR
Mathai et al. [14]	Custom IC-0.18µm	InvisibleRobust	Copyright Protection	-	Wavelet	N/A	75 MHz @ 30 fps (320 x 320)	40 dB
Jeong et al. [24]	FPGA	Robust	Copyright Protection	DVD quality	HAAR Wavelet	-	720 x 480	-
Petitjean et al. [25]	FPGA & DSP Board	InvisibleRobust	Copyright Protection	MPEG	Fractal	N/A	50 MHz takes 6 *us* 250 MHz takes 118 *us*	N/A
Mohanty et al. [23]	FPGA	Visible	Copyright Protection	MPEG-4	DCT	28.3K (Total system) 4000 (Watermarking system)	100 MHz @ 43 fps (320×240)	30 dB
Our MJPEG watermarking system [28]	FPGA	Invisible Semi-fragile	Surveillance Video Authentication	MJPEG	DCT	9.263 K (Total system with MJPEG) 400 (Watermarking system)	40 MHz @ 607 fps (256 × 256) 103 fps (640× 480)	44 dB
This work (MPEG-4 WM system) [29]	FPGA	Invisible Semi-Fragile	Surveillance Video Authentication	MPEG-4	DCT	55 K (Total system with MPEG-4) 400 (Watermarking system)	40 MHz@ 519 fps (320 × 240) 130 fps (640 × 480) 43 fps (1280 × 720)	42 dB

4.8 Summary

In this chapter, a hardware based digital watermarking system for real-time MPEG-4 video watermarking was presented. It was demonstrated that the designed watermarking system can be integrated with the state-of-the-art MPEG-4 video encoding system inside a CMOS image sensor based camera without high penalty in the overall system power consumption and hardware cost. The video watermarking system is capable of watermarking in real-time at a sufficient watermarking speed maintaining the standard frame rate even with the high video resolutions supported by state-of-the-art MPEG-4 video encoding standards. It was confirmed that the designed video watermarking system integrated with the MPEG-4 encoding system is able to withstand certain level of potential threats i.e. cover-up attack, cropping segment removal, blurring etc. with insignificant video quality degradation.

The following chapter discusses the required security and resilience of the watermarking system against known cryptographic and watermarking attacks in the context of real-time surveillance video authentication without significant increase in system power consumption and foot print.

Chapter Five: **Improved Security, Low Power and Low Cost Hardware Watermarking System for Real-time Surveillance Video Authentication**

5.1 Introduction

In this chapter we present the design and implementation of a hardware video watermarking system. The system described here offers improved security compared to other similar systems, including those presented in the previous chapters, but still maintains low power and low hardware utilization. The designed system inserts an invisible and semi-fragile watermark into the video streams captured by a surveillance video camera during the MPEG-4 encoding process and also maintains high throughput and insignificant video quality degradation. The Grain – 128a stream cipher has been chosen as a source of a hardware efficient and cryptographically secure pseudorandom number generator (CSPRNG) to generate secure watermark sequence to meet the security demands for the desired video watermark system. For enabling the reliable and secure real-time authentication of surveillance video footage, the designed watermarking system is intended to be integrated inside a surveillance video camera unit. A general threat model for video watermarking system for such application is also proposed. The designed watermarking system was prototyped on an Altera® Cyclone FPGA system as a proof of concept and was assessed on the basis of different recognized watermarking attacks and proposed threat model to ensure the achievement of security goals and implementation constraints [75].

5.2 Contributions

In the previous chapters, we have focused on validating our watermarking system with the video encoder system without high penalty in system power consumption and hardware utilization for real-time video authentication. In Chapter 3, a non-standard MJPEG encoder system was chosen as a proof-of-concept to simplify the implementation of the system [28]. In Chapter 4, our watermarking system was incorporated with the contemporary MPEG - 4 encoder system to demonstrate the compatibility of the watermarking system with state-of-the-art video encoding standard [29]. However, in both of these works the security of the watermark system was not considered. The watermark generator used in these works was based on a Gollmann Cascaded Filtered Feedback with Carry Shift Register (GCF-FCSR) PRNG, which is not cryptographically secure enough to withstand potential cryptographic attacks.

In this chapter, we focus on designing a hardware based watermarking system with improved security but still maintaining low power and low hardware utilization for real-time surveillance video authentication. We propose, for the first time, a general threat model for a video watermarking system for real-time surveillance video authentication. Specific security goals have been selected in addition to the implementation goals to ensure the required resilience of the video watermarking system. The designed video watermarking system was assessed on the basis of different recognized watermarking attacks and the given threat model to provide evidence for the achievement of security goals. The watermarking system described here, requires fewer

hardware resources, has higher processing speed and improved security when compared to other's work in the past. The main contributions of this chapter are as follows:

1. Specific security goals have been selected for the proposed video watermarking system, considering the real-time surveillance video authentication application in mind. A general threat model for a video watermarking system for such application is proposed.
2. Grain – 128a, a standardized and secure stream cipher has been chosen as a source of a hardware efficient and cryptographically secure pseudorandom number generator (CSPRNG) to generate secure watermark sequence to meet the security demands for the desired video watermarking system.
3. Finally, the designed video watermarking system with the cryptographically secure pseudorandom watermark sequence generator was integrated with an MPEG-4 video encoder and implemented on an Altera® Cyclone FPGA. The designed system was assessed on the basis of the implementation constraints and on different recognized watermarking attacks and proposed threat model to ensure the achievement of security goals.

5.3 Proposed Video Watermarking system with Improved Security

The overview of the proposed watermarking system with improved security is shown in Figure 5.1. It consists of a CMOS image sensor based video camera, commonly used MPEG-4 encoder, secure watermark generator and watermark embedder. A cryptographically secure pseudorandom number generator (CSPRNG) is used as the watermark generator to generate a secure and pseudorandom watermarking sequence. A secret key initiates the CSPRNG. This secret key is a private key chosen by the manufacturer of the camera. The same key is used to regenerate the watermark information to authenticate the video streams after analyzing the watermarked MPEG-4 video footage in the detection process. In the detection process a detection map is generated to show the detected tampered blocks in the watermarked MPEG-4 video. The details of the data flow and integration of the watermarking system with the MPEG-4 encoder system were described in the previous chapter [28]-[29].

5.4 Attacks on digital video watermarks

In parallel with the development of research in watermarking schemes different ways of attacking watermarks are also being developed. The effectiveness of different attacks depends on the robustness and the application of the watermarking algorithms. In this section we discuss known attacks on watermarking schemes and propose an attack model applicable for surveillance video cameras.

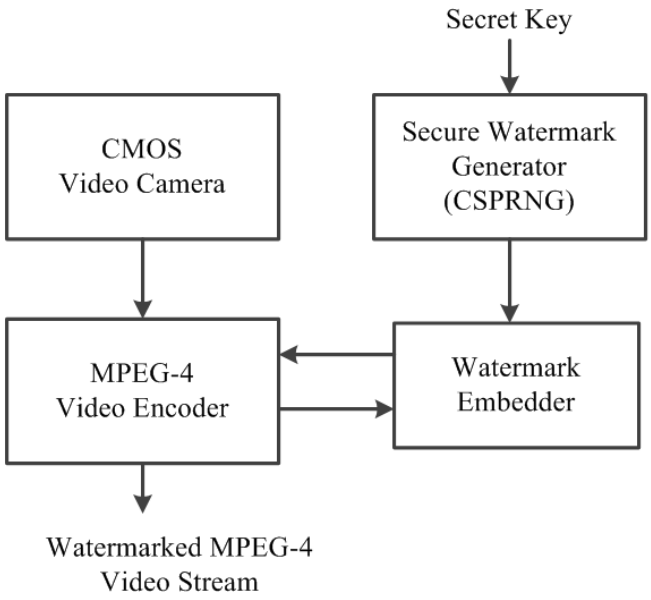


Figure 5.1 Proposed watermarking system with improved security

5.4.1 Watermarking Attacks

In *Voloshynovskiy et al.* [76], several state-of-the-art attacks are discussed. The wide class of existing attacks is categorized into four classes of attacks: removal, geometric, cryptographic and protocol attacks. A new direction of attacks based on watermarking estimation exploiting the statistics of watermarking data and original data with or without the knowledge of watermarking technology is also discussed in the same paper. The analysis concentrates on copyright protection of still images.

Jayamalar et al. [77] also categorized digital watermarking attacks as follows: subtractive, distortive, additive, filtering, cropping, compression, rotation and scaling, statistical averaging, multiple watermarking attacks, etc. Most of these attacks coincide with the attacks discussed in [76].

Hollimen et al. in [78] describe a class of attacks on certain block-based oblivious watermarking schemes. They show that watermarking techniques that embed watermarks into a host image in a block-wise independent fashion are vulnerable. This attack is possible for both fragile and robust watermarking schemes.

With copyright protection applications, the purpose of the attacks is to remove the watermark from the data or to modify the watermark signal in the data so that it becomes undetectable and the owner's information from the original data is lost. This has to be done without degrading the original information in the data otherwise the purpose of the attack is not served.

On the other hand, with authentication in surveillance cameras, the goal of the watermarking scheme is that if any sort of alteration is done in the watermarked data, the watermark detector will be unable to detect the watermark, indicating that the original watermarked data has been tampered. Hence, the attacker's goal is to modify the captured data by the surveillance camera without altering the watermark.

Therefore, the attacks on robust watermarking are not necessarily effective on semi-fragile watermarking targeted to authentication purposes. In this case, the attacker might try to extract the watermark information from the watermarked data and use that within their false data to make it authenticated. The followings attacks are possible in this case: cryptographic attack and estimation based copy attack, discussed in [76] and [79].

5.4.2 General threat model for surveillance video watermarking

Our work on hardware video watermarking is focused on the surveillance video authentication application. Such a watermarking system can be used within surveillance video cameras utilized for securing a facility such as museums, airports, public transportation, parking lots, shopping malls, etc. We describe a general threat model for attacking the hardware watermarking system of the surveillance video camera applicable in these scenarios.

A single surveillance camera installed at a place will cover a certain range of the entire facility. The main idea of the surveillance is that if anyone tries to commit a crime, the event will be captured by that security camera and the footage will be transmitted to the control room. The law enforcement officer in the control room will take action after observing the video or with the help of efficient image/video processing software, which can detect suspicious motions in the video footage. Our threat model is based on the following assumptions:

1. The hardware watermarking system is integrated into the security camera. Usually the security cameras are installed in a safe place. If the attacker wants to change the position of the camera or try to damage it physically, the security alarm will notify the law enforcement unit. This ensures the safety of the embedded watermarking system as well. Hence, we assume that the watermarking system and camera unit are physically safe. Similarly the base station is also assumed to be physically secure and inaccessible to the attackers. This means that the video footage captured by the cameras and the watermark sequence generated by the

pseudorandom number generator inside the camera unit are unable to be hacked before the transmission of the video from the camera.

2. The watermarking algorithm is public, i.e. known to the attackers. Only the secret key, required to both generate and verify the watermarks, is assumed to be unknown to the attackers.
3. The attackers have complete access to the communication link between the camera and the base station. They can intercept watermarked video, block transmission to the base station, and insert their own material to be transmitted to the base station.

If the above assumptions are true, the attacker is unable to tamper with the physical system, but he can control the transmission link (wired or wireless). His goal is to replace the original video footage with a tampered or false video footage. The attacker will be able replace or tamper the video sequence without being detected by the watermark detection process if he can regenerate the same watermark sequence used in the watermarking process and use that sequence within the tampered video sequence.

The attacker has two possible ways of attacking the watermarking system: offline attack and online or real-time attack. We discuss both of these below.

Offline Attack

An offline attack refers to an approach in which the attacker computes validly-watermarked fraudulent video offline. This may involve collecting a series of watermarked video frames for a long period of time. The attacker might be able to extract an extended watermark sequence based on a statistical analysis of those video frames. The statistical analysis of the video frames to extract the original watermark sequence embedded into the frames might be similar to the estimation based attack on the watermarked frames, described in [76]. The collected video frames can be a long video sequence of a static scene depending on the algorithm of the statistical analysis.

After extracting the extended watermark sequence the attacker must determine how to embed a new watermark sequence into his own video in such a way that it will pass the verification algorithm performed in the base station. The watermark sequence extracted from the video will not suffice, as the corresponding sequence of pseudorandom bits generated by the watermarking algorithm will not repeat, assuming the secret key and algorithm are chosen appropriately. However, the attacker may be able to use this information to mount an attack on the pseudorandom number generator, enabling him or her to predict watermark sequences and forge valid watermarked video.

Online Attack

An online attack means the attacker has to perform all the computation for generating the watermark sequence in real-time. In this attack he performs a brute force attack on the current watermarked video frame. He may perform the attack in a block-wise manner to reduce the computation. The attacker requires access to un-watermarked

versions of the image blocks he wishes to attack - the feasibility of this assumption is addressed below.

A single watermarked video frame can be divided into different watermarked pixel blocks of a particular size (i.e. 8 × 8 pixels). To tamper with a particular region of a video frame the attacker performs the brute-force attack only for that particular watermarked block of the frame.

The attacker guesses every possible permutation of the original watermark sequence of that particular block. Then using the watermarking algorithm he applies each of the guessed watermark sequences to the original un-watermarked video frame. By comparing the results to the watermarked version of the block, the attacker can determine the correct watermarked sequence used for that block.

The attacker inserts a watermark into a fraudulent video frame block, using the watermark sequence obtained in the previous step. He then transmits this new block to the base station in place of the original. The attacker must apply this process repeatedly if he wants to modify more than one block. Thus, he can generate a false version of the video frame that is still properly authenticated by the watermark.

There are some challenges for the attacker to face in this attack. He may take advantage of the knowledge of the watermarking algorithm since the algorithm is assumed to be public. However, the original un-watermarked video frame will not be available in general, since the video stream transmitted from the camera will already be watermarked and the original captured data inside the camera is not directly accessible. The attacker might use the following options to get the original un-watermarked video:

1. The attacker can set his own customized video camera at the same elevation and almost at the same physical position of that particular security camera. This will allow them to capture almost the same video footage as that of the original security camera. Thus they can get the un-watermarked video without disturbing the original security camera.
2. The attacker can perform a de-noising operation on the watermarked video frames [76]; but this is only applicable if the watermark can be considered as an additive noise added to the original video frame.

Another challenge is the computational power and time available for performing a brute-force attack. Because this is a real-time attack, the time available to perform the necessary computations is extremely limited. This issue is analyzed further in section 5.8.

5.4.3 Cryptographic vulnerability of past works

The resilience of the watermarking techniques against geometric attacks, removal attacks or protocol attacks depends on the strength of the watermarking algorithm. Resilience against a cryptographic attack depends on the cryptographic algorithm used in generating the watermark information. To resist the cryptographic attack on the secret watermark information, different works on watermarking schemes have used different cryptographic properties.

In [22], *Mohanty et al.* used AES (Advanced Encryption Standard) to encrypt the watermark information. A similar method was utilized in another of their works [23] of inserting a broadcaster's logo (a visible watermark) into video streams. AES is a popular block cipher, standardized by NIST in 2001 [60]; however, it requires a large number of

logic gates to implement on a hardware platform and it also consumes relatively a large amount of power.

In [18], *Shoshan et al.* used a Linear Feedback Shift Register (LFSR) as a pseudorandom number generator which was used to generate the watermark sequence. However, LFSRs are vulnerable to algebraic attacks and to correlation attacks. Moreover, an LFSR output sequence can be predicted efficiently by the Barlekamp-Massey algorithm described in [87]. Feedback with Carry Shift Register (FCSR), introduced by *Goresky and Klapper* [44] were thought to solve the problems associated with LFSR because its transition function is quadratic and hence resilient to algebraic and correlation attacks. In [89], *Shoshan et al.* used FCSR in Galois mode as the pseudorandom number generator (PRNG) to encrypt the primitive watermarking sequence in the CMOS imager watermarking based system.

However, FCSR sequences are also predictable like LFSR and therefore not suitable for cryptography in original form. An attacker can synthesize the FCSR, given a small amount of key-stream using the algorithm described in [80]-[81]. In order to remove this weakness of FCSR-based generators, *Arnault et al.* proposed filtering the state of the FCSR with a linear function and thus developed Filtered FCSR (F-FCSR) structure [42]. According to [42] this method of filtering the internal state is secure since the FCSR structure is not related to a linear property.

In our works [28]-[29], we used Gollmann Cascaded F-FCSR (GCF-FCSR) as the PRNG, which was designed in [57] based on F-FCSR structure. Here, F-FCSR blocks are cascaded. Gollmann Cascaded F-FCSR was proposed to achieve a high level of modularity for the designer. The cryptographic property of GCF-FCSR is completely

dependent on F-FCSR. Apart from some flaws in the initial propositions, corrected in a later stage, there was no known weakness in the core of this algorithm. However, recently in [68]-[69] *Hell et al.* presented a severe cryptanalytic attack on the F-FCSR stream cipher family. That attack requires a few Mbytes of received sequence and the complexity is low enough to allow the attack to be performed on a single PC within seconds.

Breaking of F-FCSR made the design of Gollmann Cascaded F-FCSR vulnerable since F-FCSR is the primary building block of GCF-FCSR. Hence, a new cryptographically secure PRNG for our hardware based video watermarking system was required. In addition, the secret key used for the GC-FFCSR in our previous works [28]-[29] was 25-bits long only. This is not long enough to withstand a brute-force attack. For the application of surveillance video footage authentication, we need to choose the binary key of the watermark generator system be long enough to withstand brute-force attacks for at least 2-3 years of period, because the surveillance camera units are not expected to be replaced by a newer model within 2-3 years.

Cryptographically Secure PRNGs (CSPRNGs) are PRNGs which have been specially designed to be resilient to certain cryptographic attacks. In addition to statistical randomness, CSPRNGs provide unpredictability in the sense that an adversary cannot efficiently gain an advantage in guessing parts of the output sequence even when in possession of previous outputs or the internal state of the generator [36]-[38]. Cryptographic ciphers are considered to be suitable to be used as a cryptographically secure PRNG, because the output sequence generated by an encryption cipher is also

pseudorandom in nature and at the same time cryptographically secure enough to withstand cryptographic attacks [36].

For our work on hardware based secure video watermarking system, a stream cipher was selected as a source of a cryptographically secure PRNG to generate the secret watermark sequence. For the surveillance video watermarking application, the pseudorandom sequence generator should not only be cryptographically secure but also should maintain some specific performance requirements, in particular low hardware cost and low power consumption.

5.5 Goals for Proposed Watermarking System

The proposed watermarking system is designed to be integrated into a security video camera for authenticating the video footage to detect any sort of forgery in real-time. Hence the watermarking system has to maintain some features in its design. The system is designed to be integrated within a video camera; hence it should maintain some implementation limitations. The system should be small enough to be integrated within the camera. The power consumption and the hardware utilization should be low enough to enable the integration. On the other hand, the system is built to authenticate the video footage to ensure the security, hence the watermarking system itself should maintain a high security standard so that the system does not become vulnerable to known watermarking or cryptographic threats. Taking these two issues into account, we list goals for the design of the proposed hardware watermarking system. We can categorize them by security goals and implementation goals.

5.5.1 Security Goals

The resilience of the watermarking system depends on the resilience of the watermarking generator and the watermark embedding algorithm. As mentioned before, the watermark generator is a cryptographically secure PRNG (CSPRNG) which will generate a pseudorandom bit sequence. This sequence will be used as the watermark information. The CSPRNG will be initialized by a secret key. The same key will be used in the detection of the watermarking. The followings are the security goals:

a. The CSPRNG should be resilient enough to withstand all known cryptographic attacks. Hence it should be well studied in terms of security, and should be standardized as well.

b. The key-length must be sufficiently large enough so that it is not feasible for a modern computer to exhaustively search the key. We will assume that this key will remain unchanged for at least 2-3 years and within this period of time the key should not be cracked successfully by a modern computer. According to NIST recommendation, a minimum of 80-bits of security should be provided until 2010. Between 2011 and 2030, a minimum of 112-bits of security is recommended. Thereafter, at least 128 bits of security will be required [8], [82].

c. The watermarking algorithm for the authentication of surveillance video footage is semi-fragile. The algorithm is robust against certain legitimate modification like mild lossy compression, but will be able to detect cropping, segment removal, sharpening, blurring and cover-up attacks. It should also detect watermark removal attack, noise filtering attack successfully. In addition, we will assume

that the proposed watermarking system will be robust against the proposed threat model discussed in section 5.4.

5.5.2 Implementation Goals

Since we want the watermarking system to be integrated within the camera system, the watermarking system should be small in size, have low hardware cost and low power consumption. The watermark embedder implements the watermarking algorithm. The watermark generator is a CSPRNG. The watermark embedder will remain unchanged as in [28]-[29]. Therefore, the watermarking generator will play an important role in the size, hardware cost and power consumption of the whole watermarking system. Hence our goals for the implementation are to select a suitable CSPRNG which will maintain the following constraints:

i. Minimum logic gate counts
ii. Low power consumption
iii. High processing speed (throughput)

5.6 Low Power, Low Hardware CSPRNG as Secure Watermark Sequence generator

Our sole cryptographic requirement is the choice of a cryptographically secure pseudorandom number generator (CSPRNG) that is efficient on a hardware platform. As mentioned in Chapter 2, the designs of CSPRNGs based on cryptographic primitives are fast, efficient and can be implemented on hardware. Stream ciphers, block ciphers

(operating in counter mode or output feedback mode) and cryptographic hash function are suitable for CSPRNG [53], [83].

In our system we concentrate on hardware generated CSPRNG, maintaining the implementation constraints discussed before. A secure block cipher can be converted into a CSPRNG by running it in counter (CTR) mode or output feedback (OFB) mode, but in terms of hardware resources block ciphers are more expensive than the other cryptographic primitives. Hence, they are not suitable for our application. A cryptographically secure hash of a counter may also act as a good CSPRNG. However, there has been little study on this case and some authors warned against such use of hash functions [53]. On the other hand, stream ciphers are small in size, simple to implement on hardware and fast in processing. Therefore, they are suitable to be used in applications with little computational resources i.e. cell phones or other embedded devices [84]. Stream ciphers require fewer gates (or smaller chip area) than a block cipher for the same data rate. Thus, the CSPRNG of a stream cipher is the most suitable for our application.

5.6.1 Choice of a Stream Cipher

Even though many stream ciphers have been proposed over the years, there are considerably fewer well-investigated ones. The security of many proposed stream ciphers is unknown, and many stream ciphers have been broken. This pessimistic outlook on the stream ciphers was changed when the eSTREAM project advanced the state-of-the-art of knowledge on stream cipher design [57]. As mentioned in Chapter 2, the eSTREAM project announced two profiles of stream ciphers, one containing stream ciphers more suitable for software applications with high throughput requirements and another

containing stream ciphers (e.g. *Grain v1, MICKEY 2.0 and Trivium*), particularly suitable for hardware applications with restricted resources such as limited storage, gate count, or power consumption. Algorithms in the profile were intended to be efficient in terms of the physical resources required when implemented in hardware, as is required for our application. Table 5.1 has been prepared from the hardware performance results obtained from the eSTREAM project report in [58]. Table 5.1 illustrates the comparison among the three eSTREAM stream ciphers along with the AES block cipher [85]-[86] in terms of hardware resources and performance. The primary metrics for hardware implementation performance comparisons are area or gate equivalents, power consumption and maximum frequency. Although according to the eSTREAM portfolio, the three winner stream ciphers, Grain, Trivium and Mickey, are all suitable for different application from different perspective, but when compared with the primary metrics, it is clear that Grain-80 and Grain-128 are the better options for hardware implementation for 80 bits and 128 bits respectively.

5.6.2 Grain 128a: Best security and Low Hardware Cost Stream Cipher

Grain is notable for its extremely small hardware representation. In the eSTREAM portfolio, the final 80 bit version of the Grain family, also known as Grain v1, was modern in the sense that it allows for public Initialization Vectors (IV). However, it only uses 80-bit keys. Recognizing the emerging need for 128-bit keys, *Hell et al.* (2006) proposed Grain-128 supporting 128-bit keys and 96-bit IVs. The design is akin to that of 80-bit Grain, but noticeably, the nonlinear parts of the cipher have smaller degrees than their counterparts in Grain v1. Hence, a new version Grain-128a was proposed [84]. With

a low gate count, low power consumption and a small chip area, Grain-128a is very well-suited for hardware environments. There is no 128 bit cipher offering the same security as Grain-128a and a smaller gate count in hardware [84].

5.6.2.1 Design Details

Grain-128a consists of a mechanism that produces a pre-output stream (pseudorandom key-stream). The key-stream generator, which we will use as a CSPRNG, is constructed using three main building blocks, namely a Linear Feedback Shift Register (LFSR), a Nonlinear Feedback Shift Register (NFSR) and a pre-output function (Figure 5.2). There are 256 memory elements in the two shift registers, which represent the state of the pre-output generator.

Before the key-stream is generated the cipher must be initialized with the key and the Initialization Vector (IV). The 128 NFSR bits are initialized by the key, the first 96 LFSR elements are loaded with the IV bits and the last 32 bits of the LFSR are filled with ones. Then, the cipher is clocked 256 times without producing any key-stream. Instead the pre-output function is fed back and XOR'ed with the input, both to the LFSR and to the NFSR. Details of the operation are given in [84].

5.6.2.2 Cryptanalysis

Excellent hardware performance is of little use if the cipher is not secure. Grain-128a is resistant against several cryptanalytic attacks e.g. linear approximation, algebraic attack, time-memory-data trade-off attack, fault-attack, side-channel and weak key-IV pairs attack [84]. Moreover, according to NIST recommendation, between 2011 and 2030, a minimum of 112-bits of security is recommended and thereafter, at least 128 bits

of security will be required [8], [82]. The key length of Grain-128a is 128 bits. Therefore, it will not be possible to brute force the key within the three-year period required for our application.

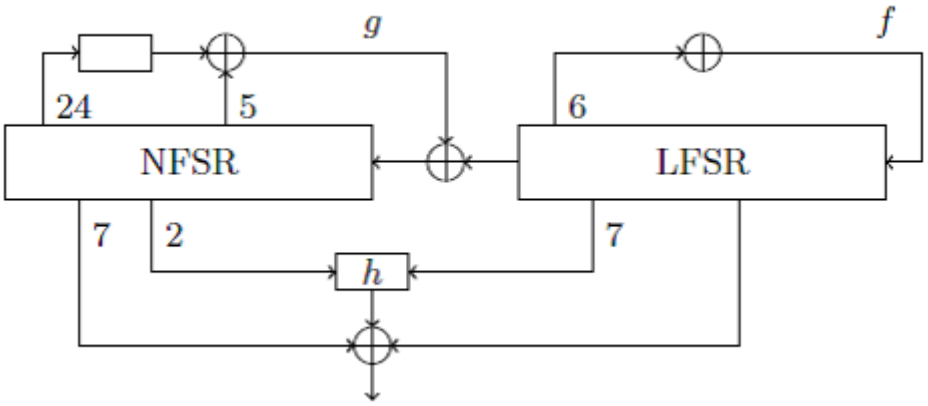


Figure 5.2 An overview of Grain-128a stream cipher's key-stream generator [77]

5.7 Hardware Implementation

We have developed a proof-of-concept hardware implementation of the video watermarking system. The implementation was done in two steps:

1. Implementation of the watermarking system (with Grain-128a as watermark sequence generator). This step validates the updated design of the watermarking system in hardware platform. We implemented the Grain-128a cipher with the proposed watermarking system according to the watermarking algorithm detailed before. The architecture of the watermarking system is shown in Figure 5.3.
2. Implementation of the complete watermarking system with MPEG video encoder. As frequently stated before, the designed watermarking system is for the video surveillance purpose. Thus in this step we integrated our video watermarking

system with the MPEG encoder, as it would be inside the surveillance video camera module.

5.7.1 MODELSIM® Simulation

Verilog HDL was used to describe the hardware architecture of the overall watermarking system (watermark embedder with Grain-128a) within the MPEG encoder system. Each of the modules in the design was simulated separately using MODELSIM (HDL simulation software) and all of them passed the simulation individually.

5.7.2 FPGA Prototyping

The Verilog HDL description of the overall system was compiled and synthesized into an Altera® EP1C20 Cyclone FPGA board and the synthesis report was analyzed in Altera® Quartus II software.

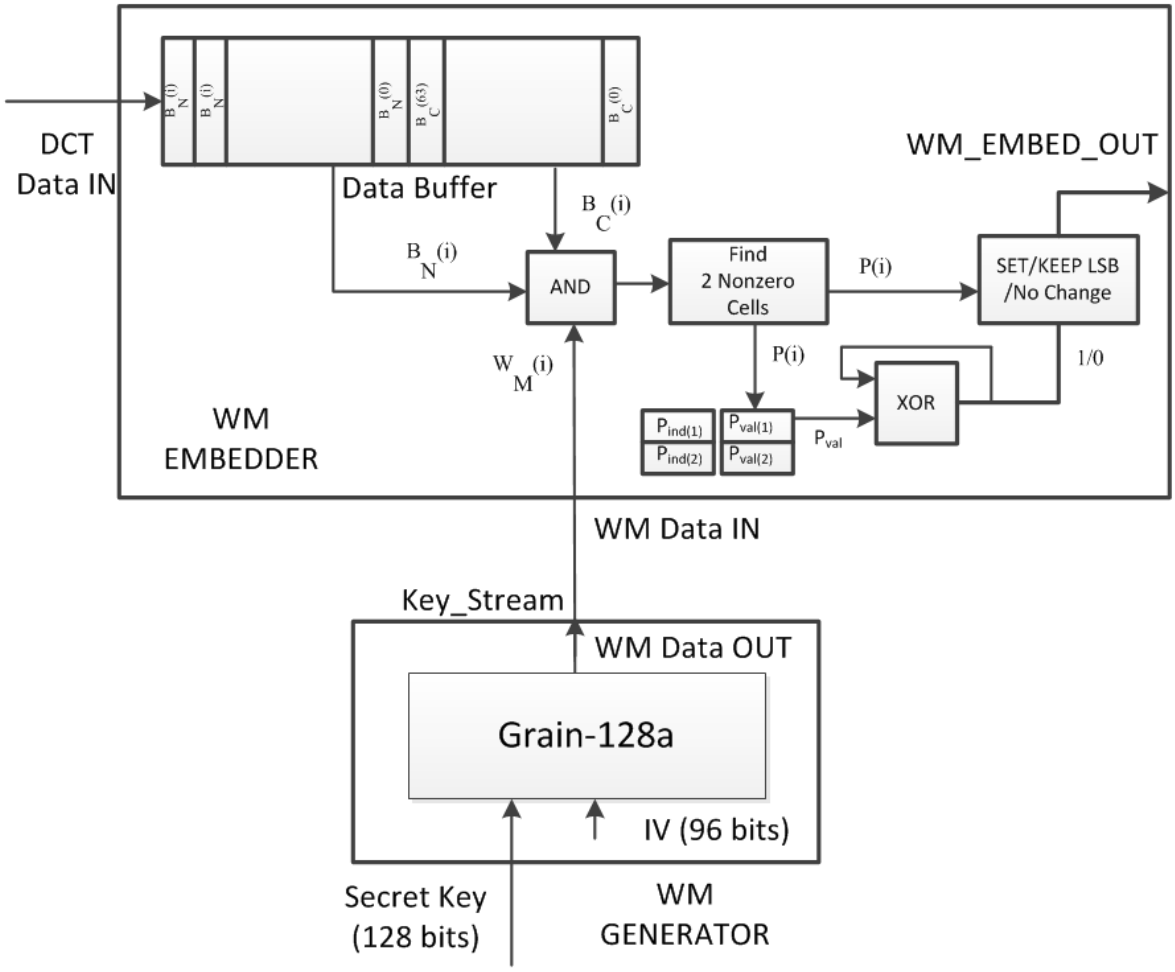


Figure 5.3 Hardware architecture of the video watermarking system (watermark embedder and watermark generator)

5.7.3 Hardware Implementation Result

The hardware resources used by the different modules are given in Table 5.2. The results indicate that the watermark generator (Grain-128a) utilized 516 logic elements and 290 register bits of the FPGA system. If we compare this with our previous design in [28] and [29], where the watermark generator required 309 logic elements and 512 register bits, we see that there is a small increase in the logic elements due to the design of Grain-128a but this increase is affordable for improved security. The increase in logic elements due to the watermarking system is only 1.2% of the total logic elements required for the overall system with the MPEG – 4 encoder.

In the previous design of watermark generator, there was 512 register bits utilization [28]-[29], which has been reduced to 290 bits because of the modification in the design of the watermark generator. The resource utilization of the watermark embedder did not change as expected since we have not changed the design. The overall power consumption of the watermarking system (watermark generator and watermark embedder) is 100 mW, which is increased from before. However, this increase in the power consumption is only 15% when compared with the total video watermarking system along with the MPEG - 4 encoding system with which it will be integrated within the video camera. Any device that is large enough for the implementation of the MPEG - 4 video encoder unit, would be able to accommodate the additional hardware required for the watermark system designed here.

From Table 5.2, we see that the maximum frequency allowed for the designed watermarking system with Grain-128a is 250 MHz which is higher than the maximum frequency allowed for the MPEG - 4 encoder system. This means the watermarking

system can synchronize with the overall video encoding system perfectly. Table 5.3 shows the initial latency and frame rate of the complete system. Initial latency for the watermark generator (Grain-128a) is 256 clock cycles and for the watermark embedder 64 clock cycles. After initialization, the throughput rate of the watermarking system is 1 bit/clock-cycle. The initial latency will only occur when the camera is turned on for the first time, so this latency will not have any effect on the captured frame and hence it will not affect the frame rate of the video sequence. It is able to maintain 43.4 fps for 1280 × 720 pixels resolution MPEG-4 encoded video stream. So it is clear that the watermarking system satisfies the required processing speed for MPEG-4 encoding even with high resolution video frames.

Table 5.1 Design performance of eSTREAM stream ciphers (0.13 µm standard cell CMOS)

Design	**Key Length**	**Load/ Initializing Cycle**	**NAND Gate Equivalents**	**Total Power @ 10 MHz (Mw)**	**Max. Clock Frequency (MHz)**
Trivium	80	1314	2580	175.1	327.9
Mickey-128	128	417	5039	310.7	413.2
Mickey2(80)	80	261	3188	196.5	454.5
AES [85]*	128	50	5398	-	131.2
AES [86]*	128	1016	3400	-	80
Grain-80	80	321	1294	109.4	724.6
Grain-128	128	513	1857	167.7	925.9
Grain-128a [84]**	128	256	2133	-	-

* Results are for different CMOS processes. Power could not be scaled reliably between different processes and libraries.

** Grain-128a is latest 128 bit version in Grain family, developed after the eSTREAM project ended [84].

Table 5.2 FPGA synthesis result for the improved security MPEG-4 video watermarking system implementation

Components	**Logic Elements***	**Register bits (Memory buffers)**	**Power consumption**	**Maximum Frequency (MHz)**
Simple profile MPEG-4 Video Encoder*	54K***	614 KB	640 mW (approx.)	40
Grain-128a (Watermark Generator)	516	290	100mW**	250
Watermark Embedder	132	744		
Overall Architecture	55 K	615 KB	740 mW (approx.)	40

* Logic element is the main building block in Altera® FPGAs.

** The Watermarking system itself (watermark generator and watermark embedder) requires 648 logic elements and 938 register bits as a whole, power consumption is approximately 100 mW.

*** Was estimated from the Simple Profile MPEG-4 encoder system implementation based on the design in our work [29].

Table 5.3 Estimated processing speed (for 1280x720 pixels video)

Components	**Initial Latency (clock-cycles)**	**Frame/sec @ 40MHz**
Simple profile MPEG-4 Video Encoder	256 (For I frames)*	43.4
Watermark Generator	257**	43.4
Watermark Embedder	64	

* Initial latency (For I frames) was estimated from the Simple Profile MPEG-4 encoder system implementation based on the design in our work [29].

** Although watermark generator (Grain-128a) has an initial latency of 257 clock cycles, this will only occur when the camera will be turned on for the first time, so this latency will not affect the average frame rate of the video.

5.7.4 Watermarked Video Quality Assessment

The video quality degradation due to the watermarking process was measured by peak signal to noise ratio (PSNR) using the following equations,

$$PSNR = 10\log_{10}\left(\frac{255}{MSE}\right) \tag{5.1}$$

$$MSE = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left[f(m,n) - k(m,n)\right]^2 \tag{5.2}$$

The size of each video frame tested was 960×720 pixels. In Figure 5.4, PSNR curves for both watermarked video and un-watermarked MPEG-4 encoded video are plotted for three different video samples. It is clear that for each video sample, the PSNR curve for watermarked video almost coincides with the PSNR curve for un-watermarked MPEG-4

video which implies that the PSNR degradation due to the watermarking for each video sample is insignificant.

5.8 Proposed Watermarking system Assessment

In this section, we assess the proposed secure, low power and low hardware cost video watermarking system with the Grain-128a key-stream generator as the watermark sequence generator, to verify the achievement of both implementation and security goals set for the proposed system. We also provide samples of detection of tampering done in video sequences watermarked using the proposed video watermarking system.

5.8.1 Implementation Goals achievement

The implementation results detailed in the previous section clearly indicate that the logic elements and memory utilization for the designed watermarking system are affordable in comparison with the utilization of MPEG-4 encoder system. Although the watermarking system with Grain 128a requires approximately 100 mW of power consumption, this is only 15% of the overall system power consumption. For featuring better security within the system we can easily afford this power consumption in the surveillance video camera system. Table 5.4 illustrates the comparison of our improved security video watermarking system with other research works on video watermarking. Our current watermarking system is clearly ahead of others in terms of logic utilization and processing speed. Our system also uses Grain 128a, a stream cipher with 128 bit binary private key, as a source of CSPRNG to generate secure and pseudorandom watermark sequence for the application. This combination helped us to maintain the

security and low hardware cost at the same time. Moreover, the designed watermarking system is also capable of maintaining a high processing speed (43 fps) with a high resolution video of 1280×720 pixels with 40 MHz clock frequency.

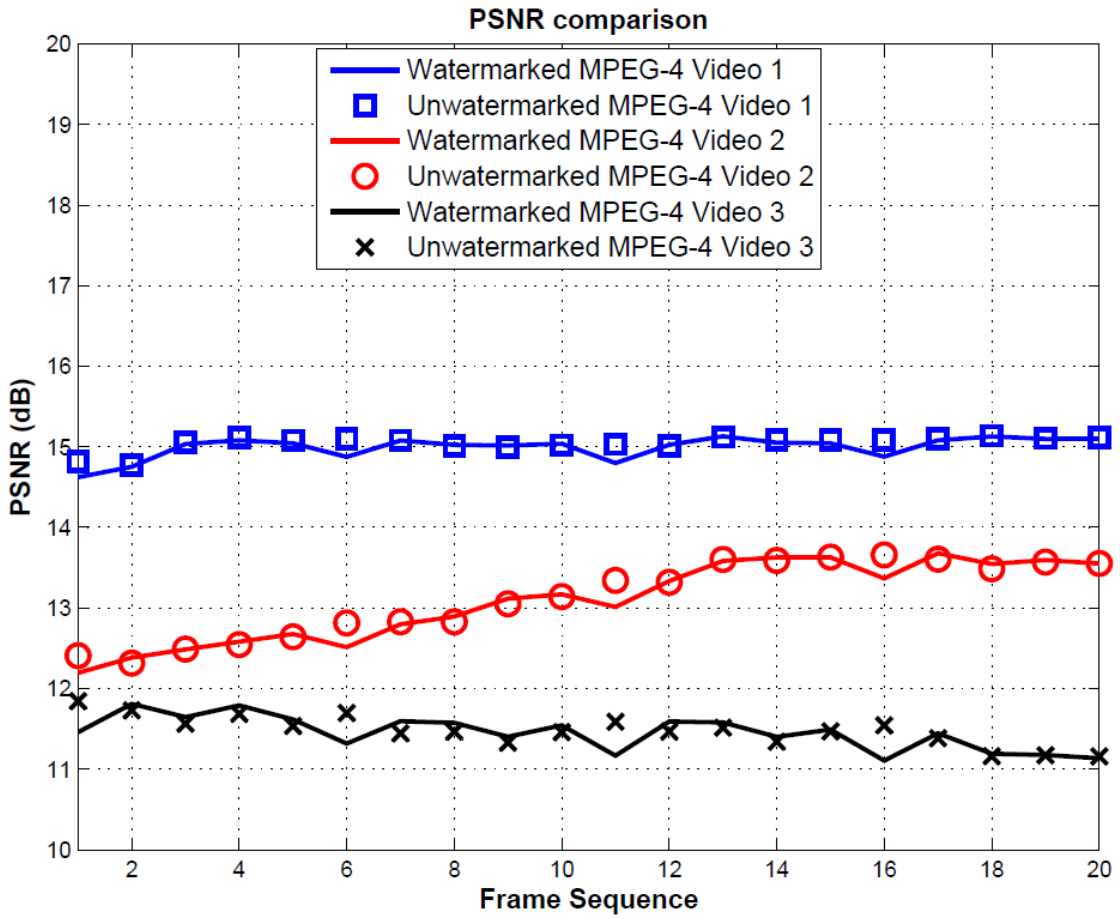


Figure 5.4 PSNR comparison between watermarked video and un-watermarked MPEG-4 video

Table 5.4 Comparison of our improved security MPEG-4 video watermarking system with other published hardware video watermarking works

Research Works	Design Platform	Type of WM	WM Gen. Security	Application	Video Standard	Processing Domain	Design Statistics	
							Logic Elements	Clock Freq. and proc. speed
Strycker et al. [40]	DSP board	Invisible Robust	-	Copyright Protection	-	Spatial	N/A	100 MHz
Maes et al. [14]	FPGA	Robust	-	Copyright Protection	MPEG	Spatial	17 K	N/A
Petitjean et al. [15]	FPGA & DSP Board	Invisible Robust	-	Copyright Protection	MPEG	Fractal	N/A	50 MHz takes 6 *us* 250 MHz takes 118 *us*
Mohanty et al. [13]	FPGA	Visible	Advanced Encryption Standard (AES)	Copyright Protection	MPEG	DCT	28.3 K (Total) 4000 (WM System)	100 MHz; 43 fps (320 x 240)
Schyndel et al. [16]	-	Invisible Fragile	-	Surveillance	-	Spread Spectrum	-	-
MJPEG Watermarking [28]	FPGA	Invisible Semi-fragile	Gollmann Cascaded F-FCSR	Surveillance Video Authentication	MJPEG	DCT	9.2 K (Total) 400 (WM System)	40 MHz; 607 fps (256 x 256) 130 fps (640x 480)
MPEG-4 Watermarking [29]	FPGA	Invisible Semi-fragile	Gollmann Cascaded F-FCSR	Surveillance Video Authentication	MPEG-4	DCT	54 K (Total) 400 (WM System)	40 MHz; 519 fps (320 x 240), 130 fps (640 x 480) 43 fps (1280 x 720)
Current Work	FPGA	Invisible Semi-fragile	Grain 128a Stream cipher	Surveillance Video Authentication	MPEG-4	DCT	54 K (Total) 648 (WM System)	40 MHz; 43 fps (1280 x 720)

5.8.2 Security Goals achievement

In this section, we assess the proposed watermarking system according to the three security goals set in section 5.5.1.

5.8.2.1 Resilience against cryptographic attacks

The proposed watermarking system consists of the watermark generator and the watermark embedder. We have used the output key-stream generated from Grain-128a stream cipher as the watermark sequence for the watermarking system. The resilience of the watermarking system depends on the cryptographic properties of the Grain-128a stream cipher.

The cryptanalysis of the Grain-128a stream cipher is detailed in [84]. As stated above, Grain-128a can withstand all known cryptanalytic attacks. Grain - *v*1 stream cipher is one of the winners in hardware stream cipher category in eSTREAM project and Grain-128a is the updated version of the Grain - *v*1 in eSTREAM competition. Thus, it has been subjected to extensive cryptanalysis, and to date no further attacks have been found to break it.

5.8.2.2 Secret Key Length

The secret key of the watermark generator is private and is only shared with the authorized person responsible for verifying the watermarked video. The manufacturer of the surveillance video camera will set a unique key for each of the cameras, where the watermarking systems will be integrated inside. We assume that the camera installed for surveillance purposes in a public facility will not be changed for two or three years. The secret key length for the watermark sequence generator should be long enough so that it cannot be compromised by an attacker using brute force during this time. According to

the NIST recommendations, between 2011 and 2030, a minimum of 112-bits of security is recommended and thereafter, at least 128 bits of security will be required [8], [82]. In our design, we use Grain-128a for the sequence generator which will be initialized using a 128 bit key. Hence this will require a brute force attack of complexity 2^{128}, which is considered to be out of reach even for the modern digital computing techniques for the foreseeable future [8], [82].

5.8.2.3 Detectability of watermarking attacks

Our watermarking system is based on an algorithm which is content dependent and allows increased security features without degrading image quality. The watermark is semi-fragile, and thus should be able to withstand legitimate modification like mild lossy compression, but should also be able to detect any sort of cover-up, segment-removal attacks, cropping, sharpening and blurring from the watermarked video.

Table 5.5 shows the result of different attacks on watermarked video frames using our watermarking system. Figure 5.5, Figure 5.6 and Figure 5.7 illustrate the effects of segment-removal, cover-up and sharpening-blurring attacks respectively on three different video streams watermarked using our watermarking system.

According to the algorithm detailed in Chapter 2, the watermark is inserted into two mid-frequency DCT coefficients in each 8×8 DCT block. It is not processed in the spatial domain. Even a pixel manipulation in an image or video frame can be detected successfully using the watermark detection process. Determination of these two coefficients in each 8×8 DCT block is performed depending on each 8×8 watermark sequence block and two successive 8×8 DCT blocks. The watermark sequence used here is also pseudorandom in nature. This algorithm features non-deterministic block-wise

dependency, non-deterministic choice of the watermarked cells within each block and frequency domain watermark insertion. These three features explain the ability of the watermarking algorithm to enable the detection of above mentioned attacks on the watermarked video frames.

The non-deterministic block-wise dependency of the watermarking scheme ensures the detection of cropping, segment-removal and cover-up attacks which basically modify the original image or video frame locally with respect to the neighboring blocks inside the video frame. The modification of two mid-frequency DCT coefficients using the watermark sequence ensures the detection of sharpening, blurring or filtering attacks on the original video frame because, sharpening is considered as the high pass filtering and blurring is the low pass filtering of the image. The non-deterministic choice of the watermarked coefficients makes the algorithm resilient against the estimation based watermarking attacks described in [76].

Table 5.5 Effects of possible attacks on watermarked video

Attacks attempted	**Effect on watermark**	**Decision**	**Expectation Fulfilled?**
Mild compression	No effect	No Change	Yes
Blurring	Affected	Tampered	Yes
Sharpening	Affected	Tampered	Yes
Cropping	Affected	Tampered	Yes
Segment Removal	Affected	Tampered	Yes
Cover-up	Affected	Tampered	Yes

Hollimen et al. [78] reported a class of attacks on certain block-based oblivious watermarking schemes. Watermarking techniques that embed a watermark into a host image in a block-wise independent fashion are vulnerable to this attack. Our watermarking scheme is DCT block based but is not block wise independent. For embedding the watermark information into the image or video frames, the embedding procedure is dependent on two adjacent 8x8 DCT blocks and the pseudorandom watermark sequence block. This non-deterministic block-wise dependency resists the counterfeiting attacks.

Since the watermark insertion is performed after DCT transformation and quantization, this algorithm is robust to the quantization of equal or lower degree. This implies that any compression on the watermarked image with a quantization ratio equal or less than the ratio during watermarking process will not affect the watermark.

5.8.2.4 Assessment based on proposed threat model

As mentioned before, the watermarking system will be integrated within the surveillance video camera, and according to our design the watermarking system will be combined with the MPEG-4 video encoder. Modern digital video cameras can capture and encode the video streams at a standard frame rate of at least 30 fps even for high resolution video (i.e. 1280 × 270 pixels SD video). At a frame rate of 30 fps each frame will take 33.33 ms of time to be read out from the camera. Therefore, an attacker would have less than 33.33 ms time limit to perform an attack on the watermarked video stream to manipulate a single video frame.

Offline Attack

In an offline attack, the attacker collects a series of watermarked video frames for a long period of time. Then he needs to extract an extended watermark sequence through a statistical analysis of those video frames. The collected video frames can be a long video sequence of a static scene depending on the algorithm of the statistical analysis. The watermarking algorithm used in our watermarking system features non-deterministic block-wise dependency and non-deterministic choice of the watermarked cells within each block. The non-deterministic nature of the algorithm makes it difficult to extract the watermark sequence.

After extracting the extended watermark sequence the attacker tries to determine a valid watermark sequence to inject into the false version of the video frame to make it authenticated. As mentioned above, this involves breaking the CSPRNG using a known output sequence, which we assume Grain-128a is able to resist. The other possibility is to wait for the watermark bit sequence to repeat. Grain-128a, which uses a 128-bit key, will not repeat before 2^{128} bits of output watermark sequence. A video frame of 1280 by 720 pixels resolution will require $(1280 \times 720) \approx 2^{20}$ bits of watermark sequence. For a video stream of 30 fps, the estimated time required for the repetition of the watermark sequence is,

$$T = \frac{\left({2^{128}}/{2^{20}} \right) frames}{30\ frames/sec} \approx \frac{2^{108}}{2^{5}} sec$$

$$= \frac{2^{105}}{(60 \times 60 \times 24 \times 365)}\ years \approx 2^{75}\ \ years.$$

clearly infeasible in practice.

Online Attack

In an online attack the attacker performs a brute force attack on the current watermarked video frame in real-time. In our system, watermark embedding is performed on 8×8 DCT blocks. Therefore, the attacker has to perform the attack at least on a single DCT block. As mentioned before, at a frame rate of 30 fps the attacker only has 33.33 ms for applying the attack.

Assuming that the attacker performs the attack on a single 8×8 DCT block, and has the corresponding un-watermarked block, he has to guess a watermark sequence of 64 bits for embedding into that single block. Thus there are 2^{64} possible watermark sequences to test for that DCT block and the attacker has to embed each sequence to check whether the guess is right or not. Embedding the watermark into a single 8×8 DCT block requires 64 or 2^4 clock cycles. So, the estimated computational time required is,

$$T \approx 2^{64} \times 2^4 = 2^{68} clock\ cycles.$$

For a high end modern CPU of 10 GHz processing speed, this amount of calculation would take,

$$T = \frac{2^{68}}{(10\ GHz)} = \frac{2^{68}}{(10 \times 10^9)} sec = \frac{2^{68}}{10^{10}}$$

$$\approx \frac{2^{68}}{(2^3)^{10}} = 2^{38} sec = \frac{2^{38}}{(60 \times 60 \times 24 \times 365)}\ years$$

$$\approx \frac{2^{38}}{2^{25}} = 2^{13}\ years.$$

So, clearly the attacker would not be able to perform the attack on the watermarked video in real-time. This provides some evidence that the designed watermarking system is resilient against the online threat model presented in this paper.

5.8.3 Sample Video Detection

The video watermarking system designed here involves invisible semi-fragile watermarking which permits certain legitimate change like mild compression of the video stream, but is capable to detect malicious changes i.e. cover-up attack, cropping, segment removal, blurring etc. as stated before. To demonstrate the ability of the secure video watermarking system to detect these kinds of attacks, video samples were embedded with watermarks using our secure video watermarking system. Sample video resolution is 720x960pixels.

Figure 5.5 illustrates a video sample of 960 by 720 pixels resolution. The video sample was also watermarked using the designed secure hardware video watermarking system during the MPEG-4 encoding process (Figure 5.5a). A single segment was removed from the watermarked MPEG-4 video frame (Figure 5.5c). This segment-removal was also successfully detected in the detection map illustrated in Figure 5.5d. Only one frame of the sample video stream is shown here as an example.

The cover-up attack was applied to the watermarked sample video (Figure 5.6). The tampered video was analyzed by the watermark detection algorithm, which outputs a detection map. The detection map indicates which blocks of the image are detected as inauthentic. In the original video frame there is a yellow color vehicle in the picture which is copied to another portion of the same frame to look like there are two yellow

color vehicles in the pictures. The detection map successfully illustrated the detection of the tampered blocks in the video frame which are suspected as inauthentic.

The detection of sharpening and blurring effect on a single watermarked video sample of 960 by 720 pixels resolution was demonstrated in Figure 5.7.

In Figure 5.8, we have demonstrated the ability of the watermarking system to allow mild compression on the watermarked video. Figure 5.8a shows the watermarked video of resolution 1280 by 720 pixels. During the watermarking process quantization was performed with a quality level, Q = 50. Later, further compression was performed on the watermarked video with a quality level, Q = 90 shown in Figure 5.8c and with a quality level, Q = 10 shown in Figure 5.8e. In other words, in Figure 5.8c, quantization ratio used for further compression was less than the quantization ratio used during watermarking of that frame, and in Figure 5.8e, quantization ratio used for further compression was higher than the quantization ratio used during the watermarking. Corresponding detection maps are shown in Figure 5.8d and Figure 5.8f respectively, which clarify that the compression of the watermarked video at a higher degree than the compression during the watermarking is detected as an alteration of the original video frame, but the compression of the watermarked video at a lower degree is permitted by the designed semi-fragile watermarking system.

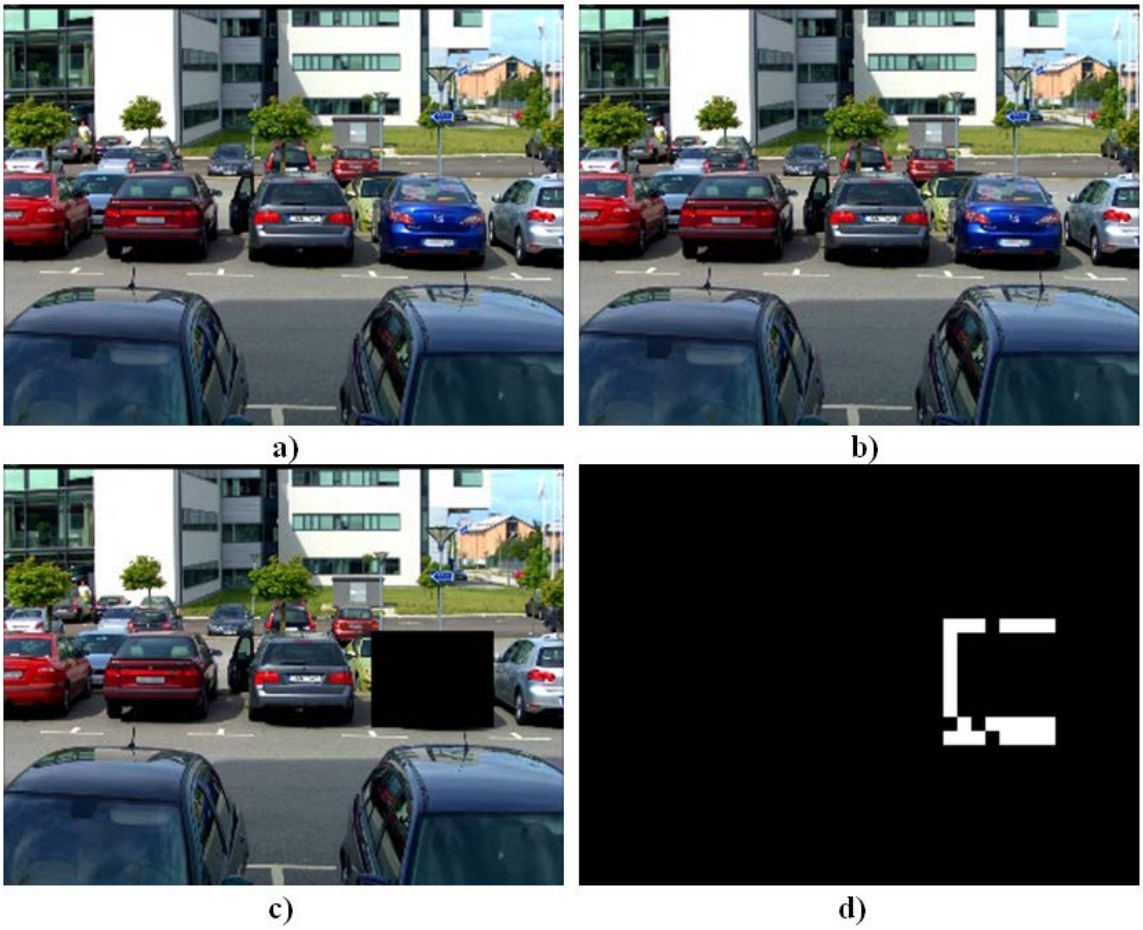


Figure 5.5 Detection of segment-removal on a 960 × 720 pixels resolution video sample: (a) Original video (b) Watermarked and MPEG-4 encoded Video frame (c) Watermarked MPEG-4 Video frame is tampered (d) Tamper Detection Map

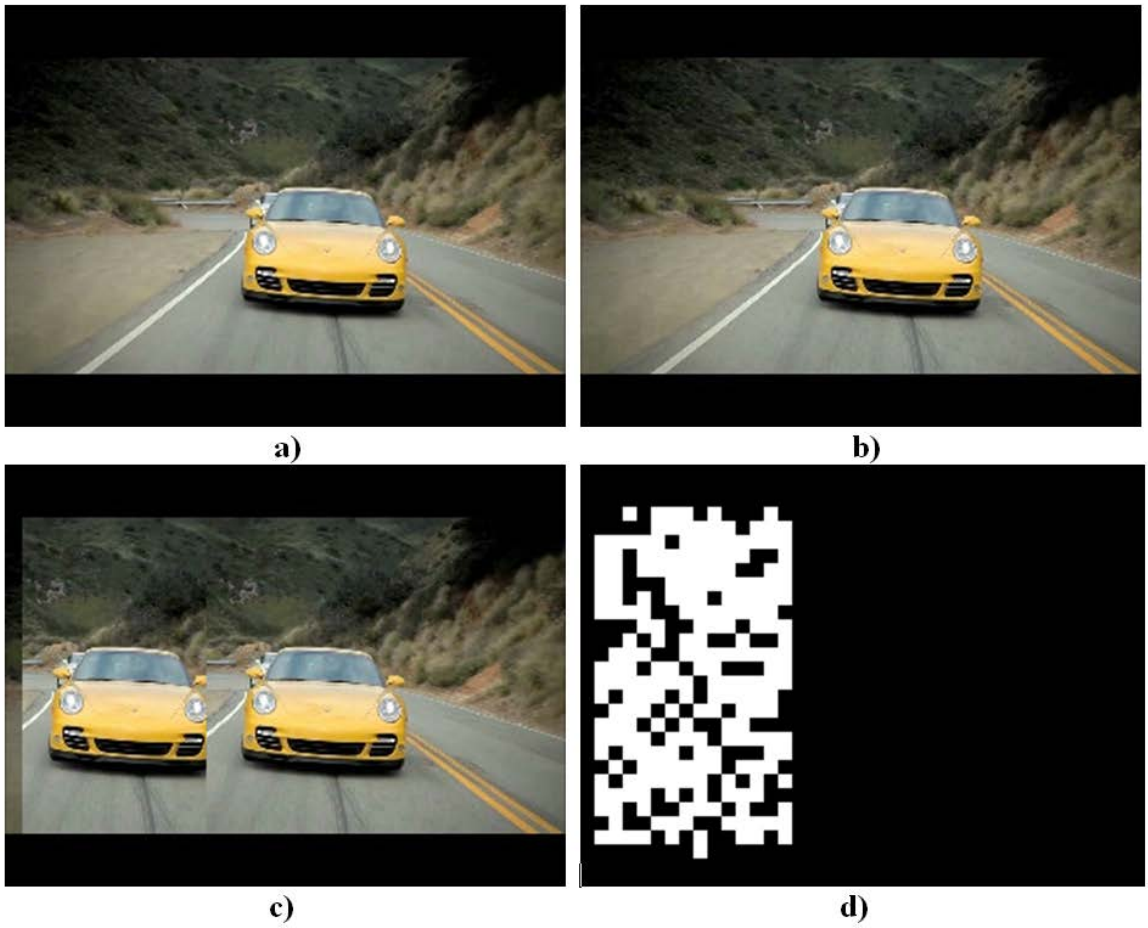


Figure 5.6 Detection of Cover-up attack on a 960 × 720 pixels resolution video sample: (a) Original video (b) Watermarked and MPEG-4 encoded Video frame (c) watermarked MPEG-4 video frame is tampered (d) Tamper Detection Map

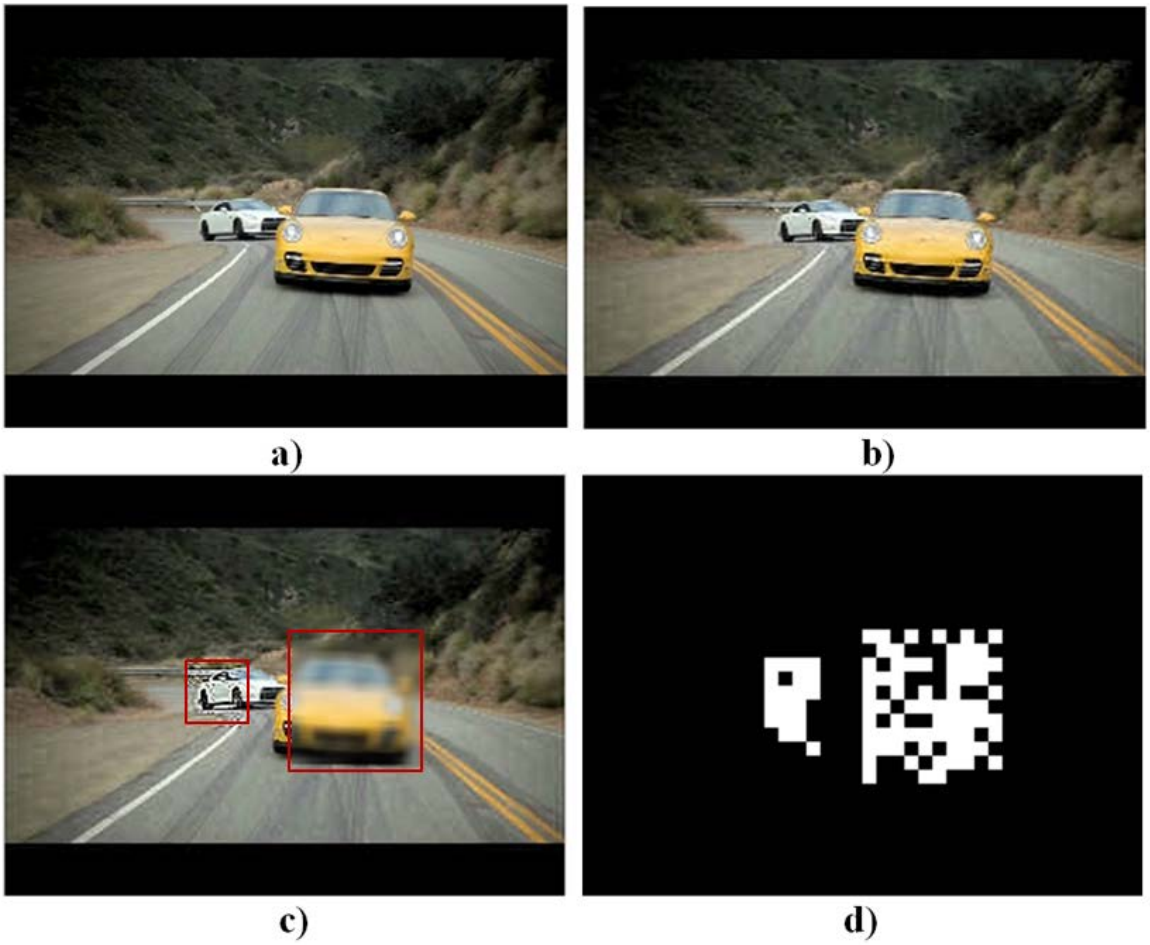


Figure 5.7 Detection of sharpening and blurring effect on a 960 × 720 pixels resolution video sample: (a) Original video (b) Watermarked and MPEG-4 encoded Video frame (c) tampered by sharpening and blurring (shown in red rectangle) (d) Tamper Detection Map

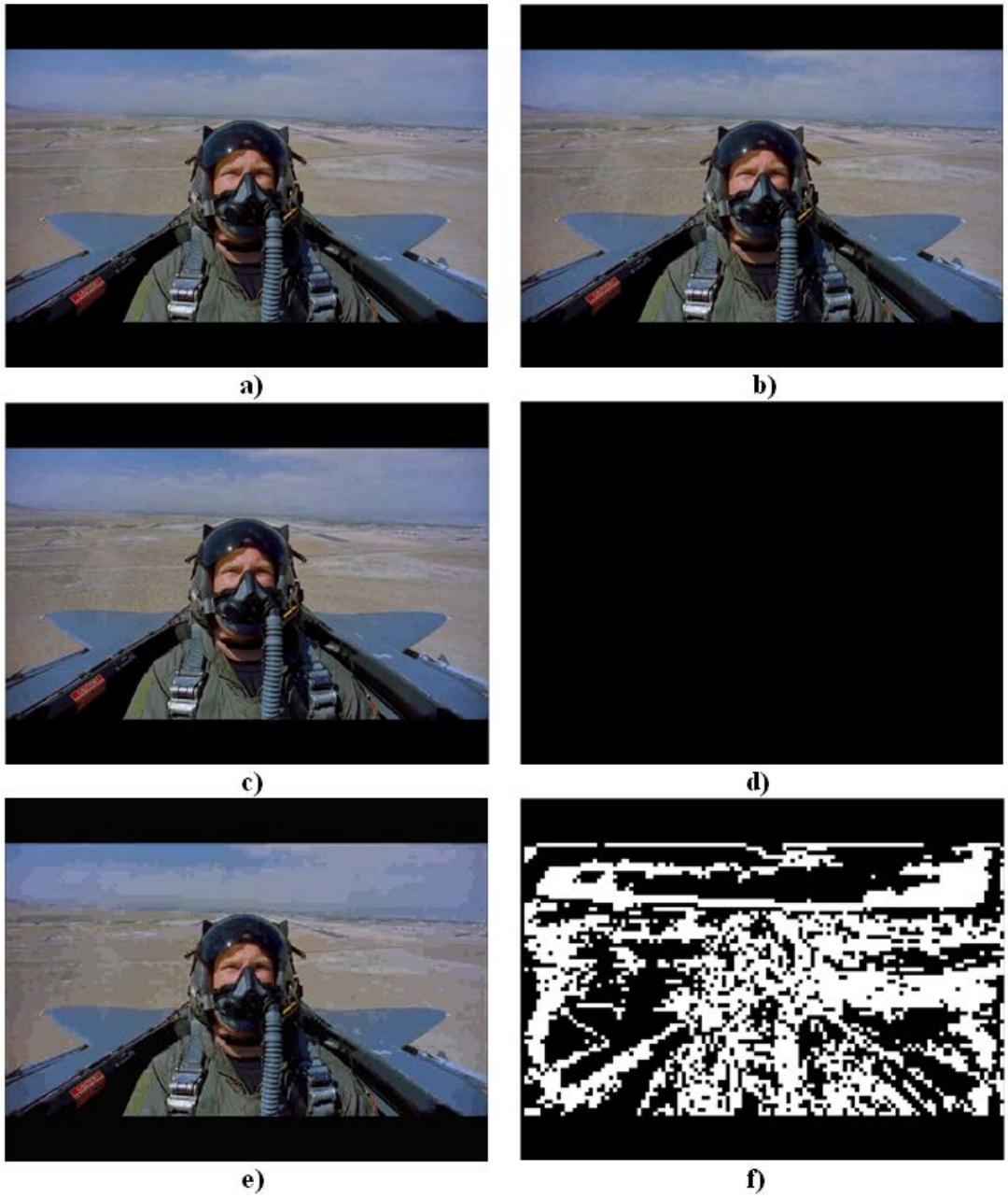


Figure 5.8 a) original video frame, b) watermarked video (quantization with quality level Q = 50), c) Compressed after watermarking with quality level, Q = 90, d) Detection map does not detect any tampering, e) Compressed after watermarking with quality level, Q = 10, f) Detection map detects tampering

5.9 Summary

In this work, the design and implementation of a secure and low hardware cost video watermarking system for surveillance video cameras is proposed. The system is designed to be integrated within the surveillance video camera to watermark the captured video footage before transmission through the link. The purpose of the designed watermarking system is to watermark the surveillance video footage in real-time, so that tampering or forgery on the captured video footage can be detected in real-time.

Our watermarking system was designed to fulfill two different sets of goals: security goals and implementation goals. The Grain-128a stream cipher was used as a source of pseudorandom sequence generator to generate a secure watermark sequence which made the watermarking system resilient against possible cryptographic attacks. A general threat model was also proposed for our application. The watermarking system was assessed based on different recognized and possible watermark attacks and proposed threat model. The hardware design of the watermarking system also maintained low power, low hardware cost with satisfactory processing speed which was the implementation goals. In comparison with other works on hardware video watermarking system, our watermarking system features better security, high processing speed and low hardware cost.

Chapter Six: **Conclusion**

6.1 Summary and Achievements

Contrary to existing solutions on video watermarking, where robust watermarking systems are mainly used for copyright protection, a secure, low power and low hardware cost video watermarking system, capable of inserting invisible and semi-fragile watermark into an MPEG-4 encoded video stream in real-time, has been presented in this thesis. The designed video watermarking system is developed to be integrated with the state-of-the-art MPEG-4 video encoding system inside a CMOS image sensory based camera system for enabling the real-time authentication of the surveillance video stream footage. Such a system inside a video camera may also be used in on-field investigations or secret missions lead by security agents to prove the integrity of the video streams for forensic purposes.

An FPGA based prototype of the hardware architecture of the overall system has been developed. The performance results indicate that this system is suitable for surveillance applications that require compactness together with a high level of security. It can be used as part of an ASIC as well. The following are the achievements of this dissertation:

1. The proposed video watermarking system was integrated with a non-standard MJPEG encoding system in hardware platform to prove the concept of effective and efficient video watermarking during the video encoding process for real-time authentication of the video stream.

2. The hardware video watermarking system was modified and integrated with a contemporary and commonly used MPEG-4 encoding system and performance was analyzed to prove that the developed watermarking system can be efficiently incorporated with the state-of-the-art MPEG-4 encoder without significant increase in system power consumption and hardware utilization to watermark the video stream in real-time during the encoding process of the video streams.
3. To use the developed watermarking system in real-time surveillance video footage authentication application, it was required that the watermarking system itself is secure enough against known cryptographic attacks and resilient against recognized watermarking attacks. A set of security goals and implementation goals were selected for the proposed watermarking system. The Grain-128a, a standardized, efficient and secure stream cipher, was chosen as a source of hardware efficient and cryptographically secure pseudorandom number generator to generate secure watermark sequence.
4. For the first time, a general threat model for a video watermarking system for real-time surveillance video authentication was proposed. The designed system was assessed on the basis of recognized watermarking attacks and the given threat model to provide evidence for the achievement of security goals.
5. The improved security design of the video watermarking system with the secure watermark generator was integrated with a commonly used MPEG-4 video encoder and implemented on an Altera® Cyclone FPGA. The overall hardware video watermarking system maintained satisfactory processing (watermarking) speed for high resolution MPEG - 4 video watermarking and insignificant video

quality degradation with small increase in system power consumption and hardware utilization.

Our watermarking system requires fewer hardware resources, has higher processing speed and improved security when compared to other's work in the past.

6.2 Future Works

Although the security of our video watermarking system is clearly an improvement, the analysis is still ad hoc in the sense that informal arguments were used to argue security against known attacks only. We have no formal proof, even under cryptographic assumptions, that the system is completely secure. Rigorous formal modeling of the watermarking threats is beyond the scope of this thesis, but would be a good subject for further research. If such a model were developed, designing a hardware efficient watermarking system that satisfies the model would also be an interesting project.

We have mentioned that our video watermarking system may also be used to prove the integrity of the video streams for forensic purposes. However, this application of the designed watermarking system will require further analysis.

Moreover, an Altera® Cyclone FPGA based prototype was developed for the designed secure video watermarking system. In future an ASIC implementation of our video watermarking system integrated with a CMOS image sensory system may also be considered.

Bibliography

[1] Y. Shoshan, A. Fish, X. Li, G. A. Jullien, and O. Yadid-Pecht, “VLSI watermark implementations and applications,” *International Journal on Information Technologies and Knowledge*, vol. 2, no. 4, pp. 379–386, June 2008.

[2] V. M. Potdar, S. Han, and E. Chang, “A survey of digital image watermarking techniques”, in Proc. of the *IEEE International Conference on Industrial Informatics*, 2005, pp. 709–716.

[3] G. Doerr and J. L. Dugelay, "A guide tour of video watermarking," *Signal Process Image Commun*, vol. 18, pp. 263-282, 2003.

[4] A. Piva, F. Bartolini, and M. Barni, “Managing copyright in open networks,” *IEEE Trans. Internet Computing*, vol. 6, no. 3, pp. 18–26, May/June 2002.

[5] X. Li, Y. Shoshan, A. Fish, G. A. Jullien, and O. Yadid-Pecht, “Hardware implementations of video watermarking,” *International Book Series on Information Science and Computing*, no. 5, pp. 9–16, June 2008.

[6] I. J. Cox, J. Kilian, F. T. Leighton, and T. Shamoon, “Secure spread spectrum watermarking for multimedia,” *IEEE Trans. Image Process.*, vol. 6, no. 12, pp. 1673–1687, Dec. 1997.

[7] M. P. Saraju, "Digital watermarking: A tutorial review," *Dept of Computer Science and Engineering, University of South Florida*, Technical report, 1999. Available: http://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/WMSurvey1999Mohanty.pdf

[8] E. Barker, W. Barker, W. Burr, W. Polk and M. Smid, "NIST Special Publication 800-57," *NIST Special Publication*, vol. 800, pp. 1-142, 2007.

[9] F. Hartung, and B. Girod, “Watermarking of uncompressed and compressed video,” *IEEE Trans. Signal Processing*, vol. 66, no. 3, pp. 283 – 302, May 1998.

[10] T. L. Wu and S. F. Wu, "Selective encryption and watermarking of MPEG video," in *International Conference on Image Science, Systems, and Technology, CISST*, 1997.

[11] A. Shan and E. Salari, "Real-time digital video watermarking," in *Consumer Electronics, 2002, ICCE. 2002 Digest of Technical Papers. International Conference on, 2002*, pp. 12-13.

[12] L. Qiao, and K. Nahrstedt, “Watermarking methods for MPEG encoded video: towards resolving rightful ownership,” in *Proc. IEEE Int. Conf. Multimedia Computing and Systems*, June 1998, pp. 276–285.

[13] L. De Strycker, P. Termont, J. Vandewege, J. Haitsma, A. Kalker, M. Maes and G. Depovere, "Implementation of a real-time digital watermarking process for broadcast monitoring on a TriMedia VLIW processor," in *Vision, Image and Signal Processing, IEE Proceedings-*, 2000, pp. 371-376.

[14] N. J. Mathai, D. Kundur and A. Sheikholeslami, "Hardware implementation perspectives of digital video watermarking algorithms," *Signal Processing, IEEE Transactions on*, vol. 51, pp. 925-938, 2003.

[15] N. J. Mathai, A. Sheikholesami, and D. Kundur, “VLSI implementation of a real-time video watermark embedder and detector,” in *Proc. of IEEE Intl. Symposium on Circuits and Systems*, vol. 2, 2003, pp. 772–775.

[16] T. H. Tsai, and C. Y. Wu, “An implementation of configurable digital watermarking systems in MPEG video encoder,” in *Proc. of IEEE Intl. Conf. on Consumer Electronics*, 2003, pp. 216–217.

[17] S. J. Lee and S. H. Jung, "A survey of watermarking techniques applied to multimedia," in *proc. of IEEE International Symposium on Industrial Electronics*, 2001, pp. 272-277.

[18] Y. Shoshan, A. Fish, G. A. Jullien, and O. Yadid-Pecht, “Hardware implementation of a DCT watermark for CMOS image sensors,” *IEEE International Conference on Electronics, Circuits, and Systems*, 2008, pp. 368-371.

[19] S. Saha, D. Bhattacharyya, S. K. Bandyopadhyay, “Security on Fragile and Semi-fragile Watermarks Authentication”, *International Journal of Computer Application*, vol. 3, no. 4, pp. 23 – 27, 2010.

[20] E.T. Lin, C. I. Podilchuk, and E. J. Delp, “Detection of image alterations using semifragile watermarks”, in *proc. of International Society for Optics and Photonics Electronic Imaging*, 200, pp. 152-163.

[21] M. Maes, T. Kalker, J. P. Linnartz, J. Talstra, G. Depoyere, and J. Haitsma, “Digital watermarking for DVD video copy protection,” *IEEE Signal Process. Mag.*, vol. 17, no. 5, pp. 47–57, Sep. 2000.

[22] S. P. Mohanty, “A Secure Digital Camera Architecture for Integrated Real-Time Digital Rights Management”, *Journal of Systems Architecture*, vol. 55, no. 10-12, pp. 468-480, 2009.

[23] S. P. Mohanty and E. Kougianos, “Real-Time Perceptual Watermarking Architectures for Video Broadcasting”, *Journal of Systems and Software*, vol. 84, no. 5, pp. 724-738, May 2011.

[24] Y.-J. Jeong, K.-S. Moon, J.-N. Kim, “Implementation of Real Time Video Watermark Embedder Based on Haar Wavelet Transform Using FPGA”, in *Proceedings of the Second International Conference on Future Generation Communication and Networking Symposia (FGCNS)*, 2008, pp. 63 – 66.

[25] G. Petitjean, J. L. Dugelay, S. Gabriele, C. Rey, J. Nicolai, “Towards Realtime Video Watermarking for Systems-On-Chip”, in *Proceedings of the IEEE International Conference on Multimedia and Expo* (Vol. 1), 2002, pp. 597–600.

[26] R. V. Schyndel, "A Hardware-Based Surveillance Video Camera Watermark." *IEEE International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, 2010, pp. 343-348.

[27] A. Fish, Y. Shoshan and O. Yadid-Pecht, “Digital Watermarking CMOS Sensors.” US Patent 7 688 994, Mar. 30, 2010.

[28] S. D. Roy, X. Li, Y. Shoshan, A. Fish and O. Yadid-Pecht, “Hardware Implementation of a Digital Watermarking System for Video Authentication”, *IEEE Transactions of Circuits and Systems for Video Technology*, vol. PP, no. 99, pp. 1, 2012. (Accepted on April 2012).

[29] S. D. Roy and O. Yadid-Pecht “Low Power and Low Hardware MPEG-4 Video Watermarking System for CMOS Image Sensor Based Surveillance Cameras”, in *IEEE Sensors Journal.* (Revised version submitted, 2012)

[30] S. D. Roy and O. Yadid-Pecht, "Design and implementation of hardware based watermarking solutions for CMOS image sensors," in *New Circuits and Systems Conference (NEWCAS), 2012 IEEE 10th International*, 2012, pp. 341-344.

[31] Theora.org. Available: http://www.theora.org/, (Accessed on: March 8, 2012)

[32] I. E. G. Richardson, *H. 264 and MPEG-4 video compression: Video Coding for Next-generation Multimedia*, New Jersey: John Wiley & Sons, 2004.

[33] K. Jack, *Video demystified: a handbook for the digital engineer*, 5th ed., VA 24085: Elsevier, 2011.

[34] A. Rukhin, J. Soto, J. Nechvatal, M. Smid and E. Barker, *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, National Institute of Standards and Technology (NIST) special publication 800-22, Gaithersburg, MD, USA, 2001.

[35] P. Peris-Lopez, E. San Millan, J. C. A. van der Lubbe and L. A. Entrena, "Cryptographically secure pseudo-random bit generator for RFID tags", in *International Conference for Internet Technology and Secured Transactions*, 2010.

[36] T.W. Cusick, C. Ding, A. Renvall, "Stream Ciphers and Number Theory", Revised Ed., Gulf Professional Publishing, 2004.

[37] D. E. Knuth, "The Art of Computer Programming, vol. 2", 2nd ed., Reading, Massachusetts: Addison-Wesley, 1981.

[38] R. McEvoy, J. Curran, P. Cotter, and C. Murphy, "Fortuna: Cryptographically Secure Pseudo-Random Number Generation in Software and Hardware", in Irish Signals and Systems Conference, 2006, pp.457-462.

[39] F. Arnault, T. Berger and A. Necer, "A new class of stream ciphers combining LFSR and FCSR architectures," Progress in Cryptology—INDOCRYPT 2002, pp. 22-33, 2002.

[40] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator," *ACM Trans. Modeling and Comp. Sim.*, vol. 8, no. 1, pp. 3-30, 1998.

[41] B. Schneier, “Applied Cryptography”, 2nd Ed., New York: Wiley, 1996.

[42] F. Arnault and T. P. Berger, "Design and properties of a new pseudorandom generator based on a filtered FCSR automaton," IEEE Trans. on Computers, vol. 54, no.11, pp. 1374-1383, Nov. 2005.

[43] X. Li, Y. Shoshan, A. Fish, and G. A. Jullien, “A simplified approach for designing secure random number generators in HW,” *IEEE International Conference on Electronics, Circuits, and Systems*, 2008, pp. 372-375.

[44] M. Goresky and A. M. Klapper, "Fibonacci and Galois Representations of Feedback-With-Carry Shift Registers," *IEEE Trans. Information Theory*, vol. 48, no.11, pp. 2826-2836, Nov. 2002.

[45] D. Gollmann, "Pseudo random properties of cascade connections of clock controlled shift registers," in *Advances in Cryptology*ypt 1985, pp. 93-98.

[46] A. C. Yao, “Theory and applications of trapdoor functions”, in *Proc. of the 23rd IEEE Symposium on Foundations of Computer Science*, 1982, pp. 80-91.

[47] P. Peris-Lopez, E. San Millan, J. C. A. van der Lubbe and L. A. Entrena, “Cryptographically secure pseudo-random bit generator for RFID tags”,

in *International Conference for Internet Technology and Secured Transactions*, 2010.

[48] M. Blum and S. Micali, “How to generate cryptographically strong sequences of pseudorandom bits”, *SIAM Journal on Computing*, vol. 13, no. 4, pp. 850-864, 1984.

[49] J. Kelsey, B. Schneier and N. Ferguson, “Yarrow-160: Notes on the design and analysis of the yarrow cryptographic pseudorandom number generator”, in *Selected Areas in Cryptography*, 2000, pp. 13-33.

[50] L. Dorrendorf, Z. Gutterman and B. Pinkas, “Cryptanalysis of the random number generator of the windows operating system”, *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 1, pp. 10, 2009.

[51] M. Matsumoto and T. Nishimura, “Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator”, *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 8, no. 1, pp. 3-30, 1998.

[52] M. Saito and M. Matsumoto, "SIMD-oriented fast Mersenne Twister: a 128-bit pseudorandom number generator," *Monte Carlo and Quasi-Monte Carlo Methods 2006*, pp. 607-622, 2008.

[53] C. Paar and J. Pelzl, “Understanding Cryptography: A Textbook for Students and Practitioners”, illustrated ed., Springer-Verlag, 2010.

[54] P. Prasithsangaree and P. Krishnamurthy, “Analysis of energy consumption of RC4 and AES algorithms in wireless LANs”, in *IEEE Global Telecommunications Conference, 2003. GLOBECOM'03, vol. 3*, 2003, pp. 1445-1449 vol. 3.

[55] R. L. Rivest, “The RC4 Encryption Algorithm,” *RSA Data Security, Inc.*, March 12, 1992. (Proprietary).

[56] P. Rogaway and D. Coppersmith, “A software-optimized encryption algorithm”, J. Cryptol., vol. 11, no. 4, pp. 273-287, 1998.

[57] http://www.ecrypt.eu.org/stream/ (last accessed on Dec 19, 2012).

[58] T. Good and M. Benaissa, “Hardware performance of eStream phase-III stream cipher candidates”, in *Proc. of Workshop on the State of the Art of Stream Ciphers (SACS’08)*, 2008.

[59] FIPS, PUB, “197: Data Encryption Standard (DES),” *National Institute for Standards and Technology*, vol. 26, 2001.

[60] N. FIPS, "197: Announcing the advanced encryption standard (AES)," *Information Technology Laboratory, National Institute of Standards and Technology*, vol. 5, no. 4, 2001.

[61] D. Chakraborty and F. R. Henríquez, "Block Cipher Modes of Operation from a Hardware Implementation Perspective," *Cryptographic Engineering*, pp. 321-363, 2009.

[62] P. Rogaway and T. Shrimpton, “Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance”, in *Proceedings of FastSoftware Encryption 2004*, pp. 371-388.

[63] J. Dittmann, T. Fiebig, R. Steinmetz, S. Fischer and I. Rimac, “Combined video and audio watermarking: embedding content information in multimedia data”, in

Proc. SPIE 3971, Security Watermarking Multimedia Contents II, 2000, pp. 455-464.

[64] G. R. Nelson, G. A. Jullien, and O. Yadid-Pecht, "CMOS image sensor with watermarking capabilities," in *Proc. IEEE Int. Symp. on Circuits and Systems* (ISCAS''05), vol. 5, , Kobe, Japan, May 2005, pp. 5326-5329.

[65] A. Filippov, "Encoding high-resolution Ogg/Theora video with reconfigurable FPGAs," in *Plugging into High-Volume Consumer Products, Xcell Journal, Xcell Publications*, Issue 53, Second Quarter 2005, pp. 19-21 , Xilinx Inc. Available: http://www.xilinx.com/publications/archives/xcell/Xcell53.pdf , [accessed on August 17, 2012]

[66] http://www.videolan.org/vlc/features.php?cat=video, [accessed on: March 8, 2012]

[67] http://www.mplayerhq.hu/design7/info.html, [accessed on: April 25, 2011].

[68] M. Hell and T. Johansson, "Breaking the stream ciphers f-fcsr-h and f-fcsr-16 in real time," *Journal of Cryptology.*, vol. 24, no. 3, pp. 427-445, 2011.

[69] M. Hell and T. Johansson, "Breaking the F-FCSR-H stream cipher in real time," *Advances in Cryptology-ASIACRYPT 2008*, pp. 557-569, 2008.

[70] MPEG-4 Simple Profile Encoder v1.2 Data Sheet, Available: http://www.xilinx.com/support/documentation/ip_documentation/mpeg_4_encoder_ds511.pdf [accessed on: May 8, 2012]

[71] MPEG-4 Simple Profile Encoder v1.2 Product Brief, Available: http://www.xilinx.com/support/documentation/ip_documentation/mpeg_4_encoder_ds521_prod_brf.pdf [accessed on: May 8, 2012]

[72] MPEG-4 Simple Profile Encoder v1.2 User Guide, Available: http://www.xilinx.com/support/documentation/ip_documentation/mpeg_4_encoder_ug219.pdf [accessed on: May 8, 2012]

[73] J. Chen, U.V. Koc and K. J. R. Liu, "Design of Digital Video Coding Systems: A Complete Compressed Domain Approach", New York: Marcel Dekker, 2002.

[74] W. Cai, "FPGA Prototyping of a Watermarking Algorithm for MPEG-4", *Master's Thesis, Department of Engineering Technology, University of North Texas*, Spring 2007.

[75] S. D. Roy, M. J. Jacobson and O. Yadid-Pecht, "Improved Security Low Power and Low Cost Hardware Watermarking System for Surveillance Video Cameras," in preparation for *Journal of Low Power Electronics and Applications* (2012).

[76] S. Voloshynovskiy, S. Pereira, T. Pun, J.J. Eggers and J.K. Su, "Attacks on digital watermarks: classification, estimation based attacks, and benchmarks", *IEEE Communications Magazine*, vol. 39, no. 8, pp. 118-126, 2001.

[77] T. Jayamalar and V. Radha, "Survey on Digital Video Watermarking Techniques and attacks on Watermarks," *International Journal of Engineering and Technology*, vol. 2, pp. 6963-6967, 2010.

[78] M. Holliman and N. Memon, "Counterfeiting attacks on oblivious block-wise independent invisible watermarking schemes," *Image Processing, IEEE Transactions on*, vol. 9, pp. 432-441, 2000.

[79] M. Kuttera, S. Voloshynovskiya and A. Herrigela, "The watermark copy attack," in *Proc. SPIE Electronic Imaging ' 00, Security and Watermarking of Multimedia Content II*, vol. 3971, pp.371 -380, 2000.

[80] F. Arnault, T. P. Berger and A. Necer, "Feedback with carry shift registers synthesis with the Euclidean algorithm," *Information Theory, IEEE Transactions on*, vol. 50, no. 5, pp. 910-917, 2004.

[81] A. Klapper and J. Xu, "Register synthesis for algebraic feedback shift registers based on non-primes," *Designs, Codes and Cryptography*, vol. 31, no. 3, pp. 227-250, 2004.

[82] E. B. Barker and J. M. Kelsey, "Recommendation for Random Number Generation using Deterministic Random Bit Generators (Revised)", *US Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Security Division, Information Technology Laboratory*, 2007.

[83] A. Young and M. Yung, "Malicious cryptography: Exposing Cryptovirology", illustrated ed., Wiley, 2004.

[84] M. Agren, M. Hell, T. Johansson and W. Meier, "Grain-128a: a new version of Grain-128 with optional authentication", *International Journal of Wireless and Mobile Computing*, vol. 5, no. 1, pp. 48-59, 2011.

[85] A. Satoh, S. Morioka, K. Takano and S. Munetoh, "A compact Rijndael hardware architecture with S-box optimization," *Advances in Cryptology—ASIACRYPT 2001*, pp. 239-254, 2001.

[86] M. Feldhofer, J. Wolkerstorfer and V. Rijmen, "AES implementation on a grain of sand", in Information Security, IEE Proceedings, 2005, pp. 13-20.

[87] F. G. Gustavson, "Analysis of the Berlekamp-Massey linear feedback shift-register synthesis algorithm," IBM Journal of Research and Development, vol. 20, no. 3, pp. 204-212, 1976.

[88] Yonatan Shoshan, “An Imaging System with Watermarking and Compression capabilities,” M.Sc Thesis, Dept. of Electrical and computer Engineering, University of Calgary, Calgary, Canada, 2009.

[89] Y. Shoshan, S. D. Roy, A. Fish, G. Jullien and O. Yadid-Pecht, “Hardware Implementation of Digital Watermarking for CMOS Image Sensor Based Camera”. (Submitted)

[90] www.opencores.org (Last accessed on January 20, 2013)

Dissemination

[1] S. D. Roy, X. Li, Y. Shoshan, A. Fish and O. Yadid-Pecht, “Hardware Implementation of a Digital Watermarking System for Video Authentication”, *IEEE Transactions of Circuits and Systems for Video Technology*, vol. PP, no. 99, pp. 1, 2012. (Accepted on April 2012).

[2] S. D. Roy and O. Yadid-Pecht “Low Power and Low Hardware MPEG-4 Video Watermarking System for CMOS Image Sensor Based Surveillance Cameras”, in *IEEE Sensors Journal*. (Revised version submitted, 2012).

[3] S. D. Roy and O. Yadid-Pecht, "Design and implementation of hardware based watermarking solutions for CMOS image sensors," in *New Circuits and Systems Conference (NEWCAS), 2012 IEEE 10th International*, 2012, pp. 341-344.

[4] S. D. Roy, M. J. Jacobson and O. Yadid-Pecht, “Improved Security Low Power and Low Cost Hardware Watermarking System for Surveillance Video Cameras,” in preparation for *Journal of Low Power Electronics and Applications* (2012).

APPENDIX A: MATLAB SIMULATION CODES

A.1. Main function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
% Date:           10/10/2012
% File:           Roy_watermark_color_video.m
% Description:
% This program simulates the watermarking for MPEG-4 encoding
% with color video samples.
% Written By:     S. D. Roy
% Updated:        on Nov 15, 2012
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
clear all;
close all;
clc;

%% Read Video File %%
% obj=mmreader('ISL2.avi');
% obj=mmreader('Wildlife.wmv')
% obj=mmreader('Nissan2.wmv');
obj=mmreader('parking.wmv');
% obj=mmreader('Horse.wmv');
% obj=mmreader('cctv.wmv');
% obj=mmreader('Car.mp4');
% obj=VideoReader('Car.mp4');
a=read(obj);
% frames=get(obj,'numberOfFrames');
% implay(a);

s = 0;
for i = 41:1:60   % 20 frame sequences will be read
% for i = 141:1:150   % 20 frame sequences will be read
P = a(:,:,:,i);
s = s +1;


% b(:,:,s) = double(rgb2gray(P)); %

ycc = rgb2ycbcr(P);
y(:,:,s) = ycc(:,:,1);
cb(:,:,s) = ycc(:,:,2);
cr(:,:,s) = ycc(:,:,3);

b(:,:,s) = double(y(:,:,s));

% I(k).colormap = [gray];
end
[row col] = size(y(:,:,1));
```

```
%% DCT + WM Embedding + IDCT
q=1;
[I(:,:,1) A1] = dct_wm_idct(b(:,:,1),row,col,q);
[I(:,:,6) A2] = dct_wm_idct(b(:,:,6),row,col,q);
[I(:,:,11) A3]= dct_wm_idct(b(:,:,11),row,col,q);
[I(:,:,16) A4]= dct_wm_idct(b(:,:,16),row,col,q);

% % DCT+ICT
% I(:,:,1) = dct_no_wm_idct(b(:,:,1));
% I(:,:,6) = dct_no_wm_idct(b(:,:,6));
% I(:,:,11) = dct_no_wm_idct(b(:,:,11));
% I(:,:,16) = dct_no_wm_idct(b(:,:,16));
%
% imshow(uint8(I(:,:,1)));

%% Motion Estimation and Compensation
%%% I1, I6, I11, I16 frames will be used as reference (I) frame in
Motion
%%% Estimation
for j = 1:5:16   %j = 51:5:66
      % I-frames
    imgI = I(:,:,j);  %imgI = b(:,:,j-50);
    for k = 1:4

            % P-frames
            imgP = b(:,:,j+k); %imgP = b(:,:,j+k-50);

            mbSize = 8; % Macroblock size
            p = 4; % Search Parameter block-size for Motion
estimation
            [vectors] = MES_TSS(imgP, imgI, mbSize, p);
            vec = vectors';
            %% motion compensated Video + Prediction
            imgComp = motionComp(imgI, vec, mbSize);
            imgDiff = imgP - imgComp; %% prediction deff, e

            %% DCT of Prediction Diff (e)
            imgDiff_dct = im2dct(imgDiff,4);


            %% IDCT of Prediction Diff (e)
            imgDiff_idct = dct2im(imgDiff_dct,4,row,col);

            %% reconstruction of predicted frame
            imgR(:,:,j+k) = imgComp  + imgDiff_idct; %imgR(:,:,j+k-
50) = imgDiff + imgComp;

            %% Predicted Video w/o motion compensation
            %  imgDiffPI = imgP - imgI;
            %  imgR1(:,:,j+k) = imgDiffPI + imgI;

    end
```

```
    imgR(:,:,j)= imgI; %   imgR(:,:,j-50)= imgI;

end

% Play Watermarked and Motion compensated compressed Video img1
img1 = uint8(imgR);  %% WM MPEG Video

%% PSNR Calculation
% for i = 1:20
%     db(i) = PSNR(b(:,:,i)/255, double(uint8(imgR(:,:,i)))/255);
% end
% plot(db, 's-');
% axis([1 20 0 50]); xlabel('Frame Sequence'); ylabel('PSNR (dB)');
% Title('MPEG Video WM vs Original Video')

%% Cover-up/ Segment-removal/ copy-crop attack

for i =  1:20
%
      I3(:,:,i) = imgR(:,:,i);
      cb_mod(:,:,i) = cb(:,:,i);
      cr_mod(:,:,i) = cr(:,:,i);

      %% Segment-addition in Nissan Video
      % I3(331:365,726:780,i)= zeros(35,55);
      % cb_mod(331:365,726:780,i)= 128*ones(35,55);
      % cr_mod(331:365,726:780,i)= 128*ones(35,55);

      %% Blurring and Sharpening on Nissan

      gaussianFilter = fspecial('gaussian', [8, 8], 150); % For
Blurring
      sharpFilter = fspecial('unsharp'); % For Sharpening

      A = I3(95:165,165:240,i);
      B = cb_mod(95:165,165:240,i);
      C = cr_mod(95:165,165:240,i);

      As = I3(105:145,105:135,i);
      Bs = cb_mod(105:145,105:135,i);
      Cs = cr_mod(105:145,105:135,i);

      % Blurr
      A = imfilter(A, gaussianFilter, 'symmetric', 'conv');
      B = imfilter(B, gaussianFilter, 'symmetric', 'conv');
      C = imfilter(C, gaussianFilter, 'symmetric', 'conv');
      % Sharp
      As = imfilter(As, sharpFilter, 'replicate');
      Bs = imfilter(Bs, sharpFilter, 'replicate');
      Cs = imfilter(Cs, sharpFilter, 'replicate');

      I3(95:165,165:240,i) = A;
```

```
      cb_mod(95:165,165:240,i) = B;
      cr_mod(95:165,165:240,i) = C;

      I3(105:145,105:135,i)= As;
      cb_mod(105:145,105:135,i) = Bs;
      cr_mod(105:145,105:135,i) = Cs;

      %% Cover-up attack in CCTV video %
      % I3(175:208,1:33,i) = I3(175:208,286:318,i);
      % cb_mod(175:208,1:33,i) = cb(175:208,286:318,i);
      % cr_mod(175:208,1:33,i) = cr(175:208,286:318,i);

      %% segment-removal-1
      % I3(175:208,1:33,i) = zeros(34,33);
      % cb_mod(175:208,1:33,i) = 128*ones(34,33);
      % cr_mod(175:208,1:33,i) = 128*ones(34,33);
      %
end

%% Detection
s=0;
for i=1:5:16
      s=s+1;
      I_D(:,:,s) = decode_WM_Video(I3(:,:,i),q);
end

encoded_y = img1; % Encoded Video
tampered_y = uint8(I3); % Tampered Video

%% Re-assemble Y (Watermarked and compressed), Cb, Cr components

s=0;
for i = 1:20
      s = s+1;
      ycc2 = reshape([y(:,:,s)  cb(:,:,s) cr(:,:,s)],[row col 3]);
      ycc2_encoded = reshape([encoded_y(:,:,s)  cb(:,:,s)
cr(:,:,s)],[row col 3]);
      ycc2_tampered = reshape([tampered_y(:,:,s)  cb_mod(:,:,s)
cr_mod(:,:,s)],[row col 3]);
      rgb = ycbcr2rgb(ycc2);
      rgb_encoded = ycbcr2rgb(ycc2_encoded);
      rgb_tampered = ycbcr2rgb(ycc2_tampered);
      rgb_video(:,:,:,s) = rgb;
      rgb_video_encoded(:,:,:,s) = rgb_encoded;
      rgb_video_tampered(:,:,:,s) = rgb_tampered;
end

%% Play Video
implay(rgb_video); % Original Video
implay(rgb_video_encoded); %MPEG-4 encoded video
implay(rgb_video_tampered); %Tampered Video
implay((I_D));  %% Detected Video
```

A.2. Compression, watermarking and de-compression

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
% Date:          10/10/2011
% File:          dct_wm_idct.m
% Description:
% Performs compression, watermarking and then decompression
% Written By:    S. D. Roy
% Updated:       on Nov 15, 2012
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%

function [Y A] = dct_wm_idct(I,row,col,q)

% quantization factor
nc = 2; %% no of coeff to be changed during WM Embedding

%% Load WM data to A
Total = (row*col)/64;
B = load('WM.txt');
index =0;

for i =1:Total
    for j=1:64  % 8x8 Block-wise
        index = index+1;
        A(j,i) = B(index);
    end
end

%% Perform DCT and Embed WM into I frame
[J H] = embed_WM(I,A,q,nc);

%% Perform decompression
Y = dct2im(J,q,row,col);  %%% Decompressed WM Image
```

A.3. Watermark embedding

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
% Date:          10/10/2012
% File:          embed_WM.m
% Description:
% This program runs the watermarking algorithm and inserts WM
% into video frames in 8x8 block-wise
% Written By:     S. D. Roy
% Updated:        on Nov 15, 2012
% I = Input frame, A = WM sequence, q = Quality,
% nc = no of cells watermarked in each 8x8 block
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%

function [y H] = embed_WM(I,A,q,nc)
% Step 1
J = im2dct(I,q);
[row col] = size(I);

% Step 3
NumOfC = nc;
fun = @findh;
Neg = J < 0;
% % Jfun(Neg) = -Jfun(Neg);
% % Jfun(:,2:end) = bitand(Jfun(:,1:end-1),Jfun(:,2:end));
% % Jfun = and(J,A);
Jfun = J;
Jfun(:,2:end) = and(J(:,1:end-1),J(:,2:end));
Jfun(:,1) = 1;

H = blkproc(Jfun,[64 1],fun);
H = H - 1;
% Going through the block to embed the WM

S = zeros(NumOfC,60/NumOfC);
M = zeros(NumOfC,60/NumOfC);

P = zeros(1,60);
Sm = 0:NumOfC:60-NumOfC;

r = row/8;
c = col/8;
for m = 2:r  %%32
    b = m;
    for n = 2:c  %%32
        b = m + (n - 1)*r  ; %%32;
        P(1:end) = 0;
        P(1:H(NumOfC,b)) =
Jfun(1:H(NumOfC,b),b).*A(1:H(NumOfC,b),b);
        for s = 1:NumOfC
```

```
            S(s,:) = P(Sm+s);
% %             if CheckParity(J(H(s,b) + 1,b)) ~=
mod(sum(S(s,:)),2)
            if CheckLSB(J(H(s,b) + 1,b)) ~= mod(sum(S(s,:)),2)
                J(H(s,b) + 1,b) = toggle2(J(H(s,b) + 1,b));
            end % parity check
        end % s loop
    end
end

y = J;

%% This function Checks LSB of the DCT Coefficients

function y = CheckLSB(x)

      C = bitget(uint16(abs(x)),1:16);
      y = C(1);

% This function flips the bits of LSB according to the
% values determined in watermarking algorithm

function j = toggle(x)

      if mod(x,2)
        x = x - 1;
      else
        x = x + 1;
      end

      if x == 0
        x = 2;
      end

      j = x;
```

A.3.1. Compression (DCT, quantization and zigzag)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
% Date:           6/6/2011
% File:           im2dct.m
% Description:    This function compresses the image;
% transforms the image to dct coefficients in 8x8 block-wise;
% quantizes the DCT values using the JPEG standard quantization
matrix and
% It reorders the DCT coefficients in zigzag pattern,
% Input quality decides the amount of information lost.
% Written By:     S. D. Roy
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
function y=im2dct(x ,quality)

% Quality level determines the compression ratio
if (quality <50) quality = (50/quality);
else quality = ((100-quality)/50);
end
% Standard JPEG quantization matrix
Q = [16 11 10 16 24 40 51 61;
      12 12 14 19 26 58 60 55;
      14 13 16 24 40 57 69 56;
      14 17 22 29 51 87 80 62;
      18 22 37 56 68 109 103 77;
      24 35 55 64 81 104 113 92;
      49 64 78 87 103 121 120 101;
      72 92 95 98 112 100 103 99]*quality;
Q = round(Q);

order = [1  9  2  3  10 17 25 18 11 4  5  12 19 26 33 41 34 27 20 13
6  7  14 21 28 35 42 49 57 50 43 36 29 22 15 8  16 23 30 37 44 51 58
59 52 45 38 31 24 32 39 46 53 60 61 54 47 40 48 55 62 63 56 64];

fun = @dct2; % Perform 2D DCT
J = blkproc(x,[8 8],fun);
J = blkproc(J, [8 8],'round(x./P1)',Q); %quantization the
coefficients

%reorder the coefficients with the zigzag  pattern

J = im2col(J, [8 8], 'distinct');  %Break 8x8 blocks into columns
J = J(order, : );   %Reorder column elements in zig-zag format
y = J;
```

A.3.2. Decompression (de-quantization and inverse 2D DCT)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
% Date:           6/6/2011
% File:           dct2im.m
% Description:
% dct2im function decompresses the image;
% It reorders the DCT coefficients in regular pattern from
% zigzag pattern, de-quantizes using the JPEG standard
% quantization matrix and transforms the dct coefficients
% back in spatial domain in 8x8 block-wise;
% Written By:     S. D. Roy
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%

function y=dct2im(J ,quality,row,col)

% Quality level determines the compression ratio
if (quality <50) quality = (50/quality);
else quality = ((100-quality)/50);
end
% Standard JPEG quantization matrix
Q = quality * [16 11 10 16 24 40 51 61;
    12 12 14 19 26 58 60 55;
    14 13 16 24 40 57 69 56;
    14 17 22 29 51 87 80 62;
    18 22 37 56 68 109 103 77;
    24 35 55 64 81 104 113 92;
    49 64 78 87 103 121 120 101;
    72 92 95 98 112 100 103 99];
Q = round(Q);

%% Reorderes in regular pattern from Zig-zag pattern

inv_order = [1 3 4 10 11 21 22 36 2 5 9 12 20 23 35  37 6 8 13 19 24
34 38 49 7 14 18 25 33 39 48 50 15 17 26 32 40 47 51 58 16 27 31 41
46 52 57 59 28 30 42 45 53 56 60 63 29 43 44 54 55 61 62 64];
J = J(inv_order, : );   % Reorder column elements back from zig-zag
format
J = col2im(J,[8 8],[row col],'distinct');        % Arranges 64
columns into 8x8 blocks
J = blkproc(J, [8 8],'x.*P1',Q); %quantizes the coefficients
fun = @idct2;
I = blkproc(J,[8 8],fun);

% % y = I + 128;
y = I;
```

A.4. Decoding watermarked video

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
% Date:           10/10/2012
% File:           detect_WM.m
% Description:
% This program loads the regenerated watermarks and runs the
detection
% algorithm
% Written By:     S. D. Roy
% Updated:        on Nov 15, 2012
% I = Input frame, q = Quality,
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%

function Y = decode_WM_Video(I,q)

[row col] = size(I);
nc = 2;
Total =(row*col)/64;
%% Load regenerated WM data to A
B = load('WM.txt');
k = 0;
for i =1:Total
    for j=1:64
        k = k+1;
        A(j,i) = B(k);
    end
end

%% Perform Detection
[C LOC] = detect_WM(I,A,q,nc);
% % disp(C);
r = row/8;
c = col/8;
LOC = reshape(LOC,r,c);

for i = 1:r
    for j = 1:c
       LOC1(8*(i-1)+1:8*i, 8*(j-1)+1:8*j) = LOC(i,j);
    end
end
Y = LOC1;
```

A.4.1. Watermark detection and generate detection map

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
% Date:           10/10/2012
% File:           detect_WM.m
% Description:
% This program runs the detection algorithm and generates a
detection map
% Written By:     S. D. Roy
% Updated:        on Nov 15, 2012
% I = Input frame, A = WM sequence, q = Quality,
% nc = no of cells watermarked in each 8x8 block
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%

function [C LOC] = detect_WM(I,A,q,nc)
% Step 1
J = im2dct(I,q);
[row col] = size(I);
% Step 3
NumOfC = nc;
fun = @findh;
% % Neg = J < 0;
% % Jfun(Neg) = -Jfun(Neg);
% % Jfun = [Jfun(:,1),bitand(Jfun(:,1:end-1),Jfun(:,2:end))];
% % Jfun = and(J,A);
Jfun = J;
Jfun(:,2:end) = and(J(:,1:end-1),J(:,2:end));
Jfun(:,1) = 1;

H = blkproc(Jfun,[64 1],fun);
H = H - 1;
% Going through the block to embed the WM
HD = zeros(size(H));

S = zeros(NumOfC,60/NumOfC);
M = zeros(NumOfC,60/NumOfC);

P = zeros(1,60);
Sm = 0:NumOfC:60-NumOfC;
fun = @binxor;

r = row/8;
c = col/8;
for m = 2:r   %%32
    b = m;
    for n = 2:c   %%32
        b = m + (n - 1)*r ;%%  32;
        P(1:end) = 0;
        P(1:H(NumOfC,b)) =
Jfun(1:H(NumOfC,b),b).*A(1:H(NumOfC,b),b);
```

```
        for s = 1:NumOfC
            S(s,:) = P(Sm+s);
mod(sum(S(s,:)),2);
            HD(s,b) = CheckLSB(J(H(s,b)+1,b)) ~= mod(sum(S(s,:)),2);
        end % s loop
    end
end

C = 1 - sum(sum(HD))/4096;
LOC = sum(HD)>0;
```

A.5. Plot PSNR curves

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
% Date:           10/10/2011
% File:           PSNR_plot.m
% Description:
% This program plots PSNR curves for watermarked video
% and unwatermarked video
% Written By:     S. D. Roy
% Updated:        on Nov 15, 2012
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%

function PSNR_plot(imgWM, img_noWM, img_Original)

% img_Original = Original video frame
% imgWM = watermarked MPEG-4 video
% img_noWM = un-watermarked MPEG-4 video

% PSNR of watermarked MPEG-4 video
for i = 1:20
      db1(i) = PSNR(img_Original(:,:,1,i)/255,
(uint8(imgWM(:,:,i)))/255);
end

% PSNR of un-watermarked MPEG-4 video
for i = 1:20
      db2(i) = PSNR(img_Original(:,:,1,i)/255,
(uint8(img_noWM(:,:,i)))/255);
end

x=1:20;
% Plot PSNR of watermarked MPEG-4 video
plot(x,db1, 'b:','LineWidth',2,'MarkerSize',10);
hold on;
% Plot PSNR of un-watermarked MPEG-4 video
plot(x,db2, 'rs','LineWidth',2,'MarkerSize',10);
grid on;
axis([1 20 5 25]); xlabel('Frame Sequence'); ylabel('PSNR (dB)');
```

```
Title('PSNR comparison')


function decibels = PSNR(A,B)

% PURPOSE: To find the PSNR (peak signal-to-noise ratio) between two
%          intensity images A and B, each having values in the
interval
%          [0,1]. The answer is in decibels (dB).

max2_A = max(max(A));
max2_B = max(max(B));
min2_A = min(min(A));
min2_B = min(min(B));

if max2_A > 1 | max2_B > 1 | min2_A < 0 | min2_B < 0
   error('input matrices must have values in the interval [0,1]')
end

err = A - B;
decibels = 20*log10(1/(sqrt(mean(mean(err.^2)))));
disp(sprintf('PSNR = +%5.2f dB',decibels))
```

APPENDIX B: VERILOG/VHDL CODES

B.1. Watermarking System with MPEG encoder

B.1.1. Testbench

```
/*********************************************************
Date:           15/8/2011
File:           wm_top_tb.v
Description:
Testbench for mpeg video watermarking module
By:     S.D.Roy
*********************************************************/

`resetall
`timescale 1ns/10ps

module wm_top_tb;

parameter clk_period = 10;
parameter FR_SIZE = 12; //size in bits

reg clk;
reg rst;

wire sim_start;
//original and decoded frames
wire [FR_SIZE+2:0] ori_fra_rd_add;
reg [FR_SIZE+2:0] dec_fra_wr_add;
reg dec_fra_wren;
wire [7:0] ori_fra_dat;
wire [7:0] dec_fra_dat;
wire [7:0] dec_fra_dat_mux;

//mjpeg internal wires
wire [2:0] indexi;
wire [2:0] indexj;
reg  [2:0] indexj_reg;
wire [FR_SIZE-1:0] fra_rd_add,fra_wr_add;

// frame_memory:original frame
// ram1 - ram5: store original video frames
wire fra_wren;

/*Begin*/
// generate signals
// generate clk and pulse rst signals
initial
begin
  $dumpfile("Power_Video_WM.vcd");
  $dumpvars;
```

```
clk <= 0;
rst <= 0;
#5 rst <= 1'b1;
#5 rst <= 1'b0;
end

always #(clk_period/2) clk = ~clk;

//instantiate mjpeg processing
wm_top U2
        (
        .clk(clk),
        .rst(rst),
        .sim_start(sim_start),
        .indexi(indexi),
        .indexj(indexj),
        .fra_rd_add(fra_rd_add),
        .fra_wr_add(fra_wr_add),
        .ori_fra_dat(ori_fra_dat),
        .fra_wren(fra_wren),
        .dec_fra_dat(dec_fra_dat)
        );

// original 3 frames and decoded 3 frames memory

                  frame_rom ori_fra_rom(
                               .address(ori_fra_rd_add[13:0]),
                               .clock(clk),
                               .q(ori_fra_dat)
                                );



                  frame_ram dec_fra_ram(
                               .address(dec_fra_wr_add[13:0]),
                               .clock(clk),
                               .data(dec_fra_dat_mux),
                               .wren(dec_fra_wren),
                               .q()
                                );

 assign ori_fra_rd_add[14:12] = (indexi - 3'd1);
 assign ori_fra_rd_add[11:0] = fra_rd_add;

 always @ (posedge clk or posedge rst)
    if (rst)
      indexj_reg <= 0;
    else
      indexj_reg <= indexj;

 always @ (posedge clk or posedge rst)
```

```
    begin
      if (rst)
        begin
        dec_fra_wren <= 0;
        dec_fra_wr_add <= 0;
        end
      else begin
        dec_fra_wren <= fra_wren;
        dec_fra_wr_add[14:12] <= indexj - 3'd1;
        dec_fra_wr_add[11:0] = fra_wr_add - 12'h1;
        end
     end

reg sim_start_reg;
reg [2:0] control_out;

always @ (posedge clk or posedge rst)
       if (rst)
               control_out <= 3'h0;
       else
               control_out <= 3'h1;

always @ (posedge clk or posedge rst)
      if (rst)
            sim_start_reg <= 1'b0;
      else
            sim_start_reg <= control_out[0];

//control assignments to be configured through the JTAG
assign sim_start = (control_out[0] && (sim_start_reg == 1'b0)) ?
1'b1 : 1'b0;
assign dec_fra_dat_mux = dec_fra_dat;

endmodule
```

B.1.2. Top-module

```
/*******************************************************
Updated on:            15/8/2011
File:                  wm_top_tb.v
Description:
Topmodule for mpeg video watermarking module
By:        S.D.Roy */
*******************************************************/

//`timescale 1ns/10ps

module wm_top
(
clk,
rst,
sim_start,
indexi,
indexj,
fra_rd_add,
fra_wr_add,
fra_wren,
ori_fra_dat,
dec_fra_dat
);
parameter FR_SIZE = 12;
input clk;
input rst;
input sim_start;
input [7:0] ori_fra_dat;
output [2:0] indexi,indexj;
output [FR_SIZE-1:0] fra_rd_add;
output [FR_SIZE-1:0] fra_wr_add;
output fra_wren;
output [7:0] dec_fra_dat;

//external wires
reg [2:0] indexi, indexj;
wire [FR_SIZE-1:0] fra_rd_add;
wire [FR_SIZE-1:0] fra_wr_add;
wire fra_wren;
wire [7:0]  dec_fra_dat;

//internal wires and signals
// mjpeg encoder_decoder wires
reg rdy_in,rdy_in_reg;
wire [7:0] in_fra_dat;
wire out_fra_en;
wire [7:0] out_fra_dat;
reg  [7:0] out_fra_dat_d;
wire [7:0]  ref_fra_dat;
```

```
// frame_memory:ref_fra_ram
wire rd_cntr_en, wr_cntr_en;             // enable signals
reg [FR_SIZE:0] rd_cntr, wr_cntr;        // ram address generator
wire ram1_wren1,ram1_rden1,ram1_rden2;   // write enable
wire ram2_wren1,ram2_rden1,ram2_rden2;   //read enable)
wire [7:0] ref_fra_dat1,ref_fra_dat2;
wire [7:0] ref_fra_dat1_ram1,ref_fra_dat1_ram2;
wire [7:0] ref_fra_dat2_ram1,ref_fra_dat2_ram2;
wire a,b;

/*begin*/
// once complete the coding (i-1)th frame and start the ith frame,
representing indexi
// once complete the decoding (i-1)th frame and start the ith frame,
representing indexj
// rdy_in for encoder
always @ (posedge clk or posedge rst)
   if (rst)
         rdy_in <= 0;
   else
   begin
      if (sim_start)
            rdy_in<= 1'b1;
      else
            rdy_in <= rdy_in;
   end

always @ (posedge clk or posedge rst)
   if (rst)
         rdy_in_reg <= 0;
   else
         rdy_in_reg <= rdy_in;

//ram rdaddress and wraddress generation
reg rd_cntr_en_reg, wr_cntr_en_reg;

always @ (posedge clk or posedge rst)
begin
      if (rst)
      begin
            rd_cntr_en_reg <= 1'b0;
            wr_cntr_en_reg <= 1'b0;
      end
      else
      begin
            rd_cntr_en_reg <= rd_cntr_en;
            wr_cntr_en_reg <= wr_cntr_en;
      end
end

assign rd_cntr_en = sim_start ? 1'b1: (indexi == 3'd4) ? 1'b0 :
rd_cntr_en_reg;
```

```
assign wr_cntr_en = (rd_cntr[FR_SIZE-1:0] == 12'h17d) ? 1'b1:
(indexj == 3'd4 && fra_wr_add == 12'd0) ? 1'b0 : wr_cntr_en_reg;
assign fra_wren = wr_cntr_en_reg;

always @ (posedge clk or posedge rst)
   begin
   if (rst)
       begin
       rd_cntr <= 0;
       end
   else begin
       if (rd_cntr_en_reg == 1'b1)
          rd_cntr <= rd_cntr + 1;
         else
            rd_cntr <= 0;
       end
   end

always @ (posedge clk or posedge rst)
   begin
   if (rst)
       begin
       wr_cntr <= 0;
       end
   else begin
       if (wr_cntr_en_reg == 1'b1)
          wr_cntr <= wr_cntr + 1;
         else
            wr_cntr <= 0;
       end
   end
  // generate indexi and indexj
 always @ (posedge clk or posedge rst)
    begin
    if (rst)
        begin
        indexi <= 3'd1;
        end
    else if (rd_cntr[FR_SIZE-1:0] == 12'hfff)
         begin
         indexi <= indexi + 3'd1;
         end
    else indexi <= indexi;
end

always @ (posedge clk or posedge rst)
   begin
   if (rst)
       begin
       indexj <= 3'd1;
       end
   else if (wr_cntr[FR_SIZE-1:0] == 12'hfff)
```

```
        begin
        indexj <= indexj + 3'd1;
        end
   else indexj <= indexj;
end
//assignments
assign fra_rd_add =
{rd_cntr[11:9],rd_cntr[5:3],rd_cntr[8:6],rd_cntr[2:0]};
assign fra_wr_add =
{wr_cntr[11:9],wr_cntr[5:3],wr_cntr[8:6],wr_cntr[2:0]};

/* mjpeg encoder_decoder*/
always @ (posedge clk or posedge rst)
     if (rst)
        out_fra_dat_d <= 0;
     else
        out_fra_dat_d <= out_fra_dat;

assign in_fra_dat = (indexi == 3'd1 ) ? ori_fra_dat :  (ori_fra_dat
- ref_fra_dat1);
assign dec_fra_dat = (indexj == 3'd1 ) ? out_fra_dat_d :
(out_fra_dat_d + ref_fra_dat2);
assign ref_fra_dat = out_fra_dat;
/*module instantiations */
//encoder_decoder
encoder
encoder_embedder_decoder (
      .clk(clk),
      .ena(1'b1),
      .rst(rst),
      .dstrb(rdy_in),
      .indexi(indexi),
      .din(in_fra_dat),
      .dout(out_fra_dat),
      .douten(out_fra_en)
      );
// frame_memory:ref_fra_ram1
// ram selection b/w write and read models
// indexj==1: just store the first decoded video frames of a GOP
// indexj==i: read it out as for reference frame twice, one for
encoder and one for decoder.

          ref_frame_memory ref_fra_ram1(
              .address_a(fra_wr_add),
                    .address_b(fra_rd_add),
                    .clock(clk),
                    .data_a(ref_fra_dat),
                    .data_b(),
                    .wren_a(ram1_wren1),
                    .wren_b(1'b0),
                    .q_a(ref_fra_dat2_ram1),
                    .q_b(ref_fra_dat1_ram1)
```

```
            );
assign ram1_wren1 = rst ? 1'b0 : (indexj == 3'd1 && wr_cntr_en_reg
== 1'b1) ? 1'b1 : 1'b0;
//assign ram1_rden1 = rst ? 1'b0 : (indexi == 3'd2 ) ? 1'b1 :
ram1_rden1;
//assign ram1_rden2 = rst ? 1'b0 : (indexj == 3'd2 ) ? 1'b1 :
ram1_rden2;
assign a = (fra_rd_add) ? 1'b1 : 1'b0;
assign b = (fra_wr_add) ? 1'b1 : 1'b0;
assign ref_fra_dat1 = (indexi == 3'd2 && a == 1'b0) ? 8'h00 :
ref_fra_dat1_ram1;
assign ref_fra_dat2 = (indexj == 3'd2 && b == 1'b0) ? 8'h00 :
ref_fra_dat2_ram1;

endmodule
```

B.1.3. Encoder module

```
/*************************************************************
Updated on:            15/8/2011
File:                  encoder.v
Description:
Top level for the encoding blocks - DCT, Zigzag, Quantizer
and WM embedder
By:                    S.D.Roy
*************************************************************/


module encoder(
      clk,
      ena,
      rst,
      dstrb,
      indexi,
      din,
      dout,
      douten
);
      parameter coef_width = 11;
      parameter di_width = 8;
      parameter do_qnr_width = 12;

      input clk;                      // system clock
      input ena;                      // clock enable
      input rst;                      // active low asynchronous
reset

      input dstrb;
// data-strobe. Present dstrb 1clk-cycle before data block
      input [di_width-1:0] din;
      input indexi;
      output [7:0] dout;    // output image data after WM embedding
```

```
      output        douten;  // data output enable
      wire douten;
      wire fdct_doe, qnr_doe;
      wire [11:0] fdct_dout;

// Hookup FDCT & ZigZag module
      fdct #(coef_width, di_width, 12)
      fdct_zigzag(
            .clk(clk),
            .ena(ena),
            .rst(rst),
            .dstrb(dstrb),
            .din(din),
            .dout(fdct_dout),
            .douten(fdct_doe)
      );
// WM Embedder
      wire WM_data_in;
      wire [do_qnr_width-1:0] WM_out;
      wire wm_doe;

      embedder_RNG #(.COEFF_SIZE(do_qnr_width))
      wm_embedder (
            .clk(clk),
            .rst(rst),
            .data_valid(fdct_doe),
            .indexi(indexi),
            .serial_data_out(fdct_dout),
            .WM_out(WM_out), // Watermarked data output
            .WM_data_in(WM_data_in),     // The pseudo-random WM
sequence
            .douten(wm_doe)
            );


      wire [di_width-1:0] dout;
      reg fdct_doe_d;
      ifdct #(do_qnr_width, di_width)
      zigzag_idct(
            .clk(clk),
            .rst(rst),
            .dstrb(wm_doe),
            .din(WM_out),
            .dout(dout),
            .douten(douten)
      );
endmodule
```

B.1.4. Watermark embedder unit

Watermark embedder module was borrowed from [88] where the source code is also available. This module was modified and incorporated to our system

B.1.5. 2D DCT and 2D Inverse DCT

2D DCT module was borrowed from www.opencores.org [90].
They provide IPCores and open source codes which may be used and distributed without restriction.

B.2. Watermark Generator

B.2.1. Gollmann Cascaded Filtered Feedback with Carry Shift Register (GCF-FCSR) module

```
/*****************************************************************
***
--    File: gcffcsr.v
--    Implemented By:        S.D. Roy (sdroy@ucalgary.ca)
--    Organization:    ISIS, Dept. ECE, U of Calgary.
--    Feb 10, 2011
-------------------------------------------------------
--    Description:
--    Filtered FCSR (13 bit), used in GC-FFCSR module.
-------------------------------------------------------
*****************************************************************
***/


module gffcsr (
            clk,
            rst,
            enable,
            mkey, ckey,
            gffcsr_out
      );

            input clk, rst, enable;
            input [25:0] mkey;
            input [23:0] ckey;
            output reg gffcsr_out;

            wire [12:0] mkey_1;
            wire [12:0] mkey_2;

            wire [11:0] ckey_1;
            wire [11:0] ckey_2;
```

```
            wire ffcsr_out1, ffcsr_out2, gcffcsr_out;

    integer fidin;

    assign mkey_1 = mkey[13:0];
    assign mkey_2 = mkey[25:13];

    assign mcey_1 = ckey[11:0];
    assign mcey_2 = ckey[23:12];

    assign out1 = (1 ^ ffcsr_out1);
    assign gcffcsr_out =  out1 ^ ffcsr_out2;
    assign clk_1 = clk;
    assign clk_2 = clk & out1;

//cascading two F-FCSR of 13 bits key-length
    ffcsr F1(clk_1, rst, enable, mkey_1, ckey_1, ffcsr_out1);
    ffcsr F2(clk_2, rst, enable, mkey_2, ckey_2, ffcsr_out2);


endmodule
```

B.2.2. Filtered FCSR module

```
/*****************************************************************
***
--    File: ffcsr.v
--    Implemented By:        S.D. Roy (sdroy@ucalgary.ca)
--    Organization:    ISIS, Dept. ECE, U of Calgary.
--    Feb 10, 2011
-------------------------------------------------------
--    Description:
--    Filtered FCSR (13 bit), used in GC-FFCSR module.
-------------------------------------------------------
*****************************************************************
***/

//`timescale  1ps/1ps


module ffcsr (
                clk,
                rst,
                enable,
                mkey,
                ckey,
                                ffcsr_out
                );

parameter   q = 2851; //(12,10,9,6,3)    q + 1 = 1062     1011 0010
0100
```

```
parameter   d = 13'b 010110010010;//(10,8,7,4,1) d=531     101 1001
0010
//parameter  T= (q-1)/2;
parameter W = 13;


input clk,rst,enable;
input [W-1:0] mkey;
input [W-2:0] ckey;

//output ffcsr_rdy_out;
output  ffcsr_out;
reg ffcsr_out;
//external wires
//reg ffcsr_rdy_out;


// internal wires
reg  [W-1:0] mstate;
wire [W-1:0] mstate_N;
reg  [W-2:0] cstate;
wire [W-2:0] cstate_N;
wire fit_out;


// begin
always @(posedge clk) begin
  if(rst)
    begin
       mstate<= mkey[W-1:0];
       cstate<= ckey[W-2:0];
//       ffcsr_rdy_out <= 0;
      ffcsr_out <= 1'hx;
    end
  else if (enable)
       begin
       mstate <= mstate_N;
       cstate <= cstate_N;
       //ffcsr_rdy_out <= rdy;
       ffcsr_out <= fit_out;
       end
            else begin
       mstate<=mstate;
       cstate<=cstate;
       //ffcsr_rdy_out <= ffcsr_rdy_out;
       ffcsr_out <= ffcsr_out;
       end

end


// Define the FCSR and Filter function
```

```
assign mstate_N[0]=mstate[1]^d[0]&cstate[0]^d[0]&mstate[0];
assign mstate_N[1]=mstate[2]^d[1]&cstate[1]^d[1]&mstate[0];
assign mstate_N[2]=mstate[3]^d[2]&cstate[2]^d[2]&mstate[0];
assign mstate_N[3]=mstate[4]^d[3]&cstate[3]^d[3]&mstate[0];
assign mstate_N[4]=mstate[5]^d[4]&cstate[4]^d[4]&mstate[0];
assign mstate_N[5]=mstate[6]^d[5]&cstate[5]^d[5]&mstate[0];
assign mstate_N[6]=mstate[7]^d[6]&cstate[6]^d[6]&mstate[0];
assign mstate_N[7]=mstate[8]^d[7]&cstate[7]^d[7]&mstate[0];
assign mstate_N[8]=mstate[9]^d[8]&cstate[8]^d[8]&mstate[0];
assign mstate_N[9]=mstate[10]^d[9]&cstate[9]^d[9]&mstate[0];
assign mstate_N[10]=mstate[11]^d[10]&cstate[10]^d[10]&mstate[0];
assign mstate_N[11]=mstate[10]^d[11]&cstate[11]^d[11]&mstate[0];
//assign mstate_N[12]=d[12]&mstate[0];
//assign mstate_N[10]=d[10]&mstate[0];
assign mstate_N[12]=1'b0;

assign
cstate_N[0]=(mstate[1]&cstate[0]^cstate[0]&mstate[0]^mstate[1]&mstat
e[0]) & d[0];
assign
cstate_N[1]=(mstate[2]&cstate[1]^cstate[1]&mstate[0]^mstate[2]&mstat
e[0]) & d[1];
assign
cstate_N[2]=(mstate[3]&cstate[2]^cstate[2]&mstate[0]^mstate[3]&mstat
e[0]) & d[2];
assign
cstate_N[3]=(mstate[4]&cstate[3]^cstate[3]&mstate[0]^mstate[4]&mstat
e[0]) & d[3];
assign
cstate_N[4]=(mstate[5]&cstate[4]^cstate[4]&mstate[0]^mstate[5]&mstat
e[0]) & d[4];
assign
cstate_N[5]=(mstate[6]&cstate[5]^cstate[5]&mstate[0]^mstate[6]&mstat
e[0]) & d[5];
assign
cstate_N[6]=(mstate[7]&cstate[6]^cstate[6]&mstate[0]^mstate[7]&mstat
e[0]) & d[6];
assign
cstate_N[7]=(mstate[8]&cstate[7]^cstate[7]&mstate[0]^mstate[8]&mstat
e[0]) & d[7];
assign
cstate_N[8]=(mstate[9]&cstate[8]^cstate[8]&mstate[0]^mstate[9]&mstat
e[0]) & d[8];
assign
cstate_N[9]=(mstate[10]&cstate[9]^cstate[9]&mstate[0]^mstate[10]&mst
ate[0]) & d[9];
assign
cstate_N[10]=(mstate[11]&cstate[10]^cstate[10]&mstate[0]^mstate[11]&
mstate[0]) & d[10];
```

```
//assign
cstate_N[11]=(mstate[12]&cstate[11]^cstate[11]&mstate[0]^mstate[12]&
mstate[0]) & d[11];
assign cstate_N[11]=1'b0;

// output ffcsr_out
assign fit_out = mstate[1]^mstate[4]^mstate[7]^mstate[8]^mstate[10];


endmodule
```

B.2.3. Test-bench for GCF-FCSR module

```
/*****************************************************************
***
--    File: tb_Gollmann_ffcsr.v
--    By:         S.D. Roy (sdroy@ucalgary.ca)
--    Feb 10, 2011
--    Organization:    ISIS, Dept. ECE, U of Calgary.
-------------------------------------------------------
--    Description:
--    Testbench for Gollmann Cascaded Filtered FCSR (26 bit)
-------------------------------------------------------
*****************************************************************
***/
`timescale  1ps/1ps

module tb_Gollmann_ffcsr ();

      reg clk, rst, enable;
             reg [25:0] mkey;
             reg [23:0] ckey;
      wire gcffcsr_out  ;

      integer fidin;

    gffcsr top(
                 clk,
          rst,
          enable,
                 mkey, ckey,
                 gffcsr_out
              );

      //ffcsr    F1(clk_s,    rst_s,    enable_s,    mkey_s,    ckey_s,
ffcsr_out1);
      //ffcsr    F2(clk_s2,    rst_s,    enable_s,    mkey_s,    ckey_s,
ffcsr_out2);

      //assign out1 = (1 ^ ffcsr_out1);
      // assign gcffcsr_out =  out1 ^ ffcsr_out2;
      //assign clk = clk_s & out1;
```

```
      always begin
              clk <= 0;
              #5 ;
              clk <= 1;
              #5 ;
      end


      initial begin

              mkey <= 26'b 00000100000000000100000000;
              ckey <= 24'b 100000000000010000000000;
              // store the bit-stream in a text file (as a Memory
buffer)
              fidin = $fopen("D:/gcffcsr_data.txt");

              rst <= 0;
              enable <= 0;
              #10;
              rst <= 1;
              #10;
              rst <= 0;
              #10;
              enable <= 1;
              #40620000;

              end

              always @(posedge clk) begin
                   $fdisplay(fidin,"%d",gcffcsr_out);
      end



endmodule
```

B.3. MPEG-4 Motion Estimation System

B.3.1. Top-module

```
/*********************************************************
Date:           1/8/2011
File:           Topmodule.v
Description:
This is the topmodule for Motion Estimation Module
Written By:     S. D. Roy
*********************************************************/

//`timescale  1ps/1ps

module Topmodule(
      clock, reset, en_MotionEstimation, i1_next_3, j1_next_3,
      count_i_3, count_j_3,
      //i1_initial_3, j1_initial_3
);

parameter CLK_PERIOD = 10;
parameter DATA_WIDTH = 8 ;
parameter ADD_WIDTH = 26 ;
parameter MEM_SIZE = 16384;


input clock, reset;
input en_MotionEstimation;
output reg i1_next_3, j1_next_3,  count_i_3,      count_j_3;      //
      i1_initial_3, j1_initial_3;
//output wire i1_next_3, j1_next_3,     count_i_3,   count_j_3;   //
      i1_initial_3, j1_initial_3;

wire en_MotionES1, en_MotionES2, en_MotionES3;
//wire en_Motion_chip1;
wire  en_SAD, SAVE_SAD, DEL_SAD;
wire en_MemAddSys; //reg en_MemAddSys; //

//wire en_Mem_current, en_Mem_ref, en_MV_cal;
wire   [10:0]   i1_next_1,   j1_next_1,   i1_next_2,   j1_next_2;   //,
i1_next_3, j1_next_3 ;

wire [10:0] step1, step2, step3;
wire [10:0] i1_initial_1, j1_initial_1, i1_initial_2, j1_initial_2,
i1_initial_3, j1_initial_3;
wire [10:0] count_i_1, count_j_1, count_i_2, count_j_2; //count_i_3,
count_j_3;

//assign i1_initial_1 = 10'd 0;
//assign j1_initial_1 = 10'd 0;
```

```
MotionES_chip CHIP1(
      clock,
      reset,
      en_MotionES1,
      step1 ,
      i1_next_1, j1_next_1,
      count_i_1, count_j_1,
      i1_initial_1, j1_initial_1
      );

MotionES_chip CHIP2(
      clock,
      reset,
      en_MotionES2,
      step2 ,
      i1_next_2, j1_next_2,
      count_i_2, count_j_2,
      i1_initial_2, j1_initial_2
      );

MotionES_chip CHIP3(
      clock,
      reset,
      en_MotionES3,
      step3 ,
      i1_next_3, j1_next_3,
      count_i_3, count_j_3,
      i1_initial_3, j1_initial_3
      );

MotionES_Controller_Top MC_TOP(
  clock,
  reset, en_MotionEstimation,
  en_MotionES1, en_MotionES2, en_MotionES3, step1, step2, step3,
  count_i_1, count_j_1, count_i_2, count_j_2, count_i_3, count_j_3,
  i1_next_1, j1_next_1, i1_next_2, j1_next_2, i1_next_3, j1_next_3,
  i1_initial_1,     j1_initial_1,     i1_initial_2,     j1_initial_2,
i1_initial_3, j1_initial_3
  );

/*
initial begin
  clock <= 1'b0;
  reset <= 1'b1;
  en_MotionEstimation <= 0;
  //step <= 10'd 4;

end

always  #(CLK_PERIOD/2) clock = ~clock;
```

```
initial begin
  #15 reset <= 0;


  //#5 en_MemAddSys <= 1;
  #1000000;
end

always@(posedge clock) begin
  if (!reset) en_MotionEstimation <= 1;
end
*/


endmodule
```

B.3.2. Motion estimation controller

```
/*******************************************************************
*
Date:           1/8/2011
File:           MotionES_Controller_Top.v
Description:
This module controls the overall motion estimation process.
This is the top controller module of the Motion Estimation Module.
Written By:     S. D. Roy
*******************************************************************
**/
//`timescale  1ps/1ps

module MotionES_Controller_Top(
  clock,
  reset, en_controller,
  en_MotionES1, en_MotionES2, en_MotionES3, step1, step2, step3,
  count_i_1, count_j_1, count_i_2, count_j_2, count_i_3, count_j_3,
  i1_next_1, j1_next_1, i1_next_2, j1_next_2, i1_next_3, j1_next_3,
  i1_initial_1,     j1_initial_1,     i1_initial_2,     j1_initial_2,
i1_initial_3, j1_initial_3
  );

input clock,reset, en_controller; // en_MV_cal;
input [10:0] count_i_1, count_j_1, count_i_2, count_j_2, count_i_3,
count_j_3;
input [10:0] i1_next_1, j1_next_1, i1_next_2, j1_next_2, i1_next_3,
j1_next_3;

output [10:0] step1, step2, step3;
reg [10:0] step1, step2, step3;

output en_MotionES1, en_MotionES2, en_MotionES3;
wire en_MotionES1, en_MotionES2, en_MotionES3;
```

```
output [10:0] i1_initial_1, j1_initial_1;
wire [10:0] i1_initial_1, j1_initial_1;

output [10:0] i1_initial_2, j1_initial_2;
wire [10:0] i1_initial_2, j1_initial_2;

output [10:0] i1_initial_3, j1_initial_3;
wire [10:0] i1_initial_3, j1_initial_3;


reg [10:0] count_630;
reg [11:0] count_1203;


wire  en_save_endpoint, en_save_startpoint, save_sign;

reg [20:0] add_start;
reg [10:0] mem_start_i ; //[0:1199];

reg [20:0] add_end;
reg [10:0] mem_end_i1 ; //[0:1199];

reg [10:0] i_1, j_1, i_2, j_2, i_3, j_3 ;
reg   [10:0]   i_1_latch1,   j_1_latch1,   i_1_latch2,   j_1_latch2,
i_1_latch3, j_1_latch3, i_final, j_final;
wire [10:0] motion_vector_col, motion_vector_row;

assign save_sign = (reset)? 0: ( (en_controller)? ((count_630 == 0)?
1: 0):0 );

assign  en_save_startpoint  =  (reset)?  0:  (  (en_controller)?     ((
count_1203 >= 1 && count_1203 <= 1200)? 1 :0 ) : 0);
assign  en_save_endpoint  =  (reset)?  0:  (  (en_controller)?     ((
count_1203 >= 4 && count_1203 <= 1203)? 1 :0 ) : 0);
assign en_MotionES1 = en_save_startpoint;
assign en_MotionES2 = (reset)? 0: ( (en_controller)?   (( count_1203
>= 2 && count_1203 <= 1201)? 1 :0 ) : 0);
assign en_MotionES3 = (reset)? 0: ( (en_controller)?   (( count_1203
>= 3 && count_1203 <= 1202)? 1 :0 ) : 0);

always @(posedge clock) begin
  if (reset) begin
    i_1 <= 0;
    j_1 <= 0;
    i_1_latch1 <= 0; j_1_latch1 <= 0; i_1_latch2 <= 0; j_1_latch2 <=
0; i_1_latch3 <= 0; j_1_latch3 <= 0;
  end
  else if (en_MotionES1) begin
    i_1 <= (save_sign)? count_i_1 : i_1;
    j_1 <= (save_sign)? count_j_1 : j_1;
```

```
    i_1_latch1 <= (save_sign)? i_1 : i_1_latch1;
    j_1_latch1 <= (save_sign)? j_1 : j_1_latch1;

    i_1_latch2 <= (save_sign)? i_1_latch1 : i_1_latch2;
    j_1_latch2 <= (save_sign)? j_1_latch1 : j_1_latch2;

    i_1_latch3 <= (save_sign)? i_1_latch2 : i_1_latch3;
    j_1_latch3 <= (save_sign)? j_1_latch2 : j_1_latch3;


  end
end

always @(posedge clock) begin
  if (reset) begin
    i_2 <= 0;
    j_2 <= 0;
  end
  else if (en_MotionES2) begin
    i_2 <= (save_sign)? count_i_2 : i_2;
    j_2 <= (save_sign)? count_j_2 : j_2;
  end
end


initial begin
step1 <= 4;
step2 <= 2;
step3 <= 1;
end


assign i1_initial_1 = 0;
assign j1_initial_1 = 0;

assign i1_initial_2 = (reset)? 0: ((count_630 == 629)? (i1_next_1 -
i_1) : i1_initial_2);
assign j1_initial_2 = (reset)? 0: ((count_630 == 629)? (j1_next_1 -
j_1) : j1_initial_2);

assign i1_initial_3 = (reset)? 0:((count_630 == 629)? (i1_next_2 -
i_2) : i1_initial_3);
assign j1_initial_3 = (reset)? 0:((count_630 == 629)? (j1_next_2 -
j_2) : j1_initial_3);

always @(posedge clock) begin
  if (reset) begin
    count_630 <= 0;
  end
  else if (en_controller) begin

      if (count_630 < 629) begin
```

```
        count_630 <= count_630 + 1;
       end
       else count_630 <= 0;
  end
end

/*always @(posedge clock) begin
  if (reset) begin
    count_1203 <= 0;
  end
  else count_1203 <= count_1203 ;
end */

always @(posedge save_sign ) begin
  if (en_controller) begin
    //if (save_sign) begin
      if (count_1203 < 1203) begin
        count_1203 <= count_1203 + 1;
      end
      else count_1203 <= 1;
    //end
  end
end

/* always @(posedge clock)  begin
  if (reset) begin
    add_start <= 0;
  end
  else if (en_save_startpoint && save_sign) begin
    //if (save_sign) begin
    mem_start_i <= count_i_1;   //[add_start_i]
    add_start <= add_start + 1;
    //end
   end
end */

always @(posedge clock)  begin
  if (reset) begin
    //mem_end_i1 <= 0;   //[add_start_i]
    i_final <= 0; j_final <= 0;
  end
  else if (en_save_endpoint && save_sign) begin
    i_final <= i1_next_3;   //[add_start_i]
    j_final <= j1_next_3 ;
  end

end

assign motion_vector_col = (reset)? 0: ((en_save_endpoint)? (i_final
- i_1_latch3):0);
assign motion_vector_row = (reset)? 0: ((en_save_endpoint)? (j_final
- j_1_latch3):0);
```

```
endmodule
```

B.3.3. Motion estimation module

```
/*****************************************************************
Date:           1/8/2011
File:           MotionES_chip.v
Description:
This module performs three step search (TSS) algorithm to find out
the match in neighbouring macro-blocks (8x8). We need three
MotionES_chip module running in parallel to implement TSS algorithm
with high throughput.
Written By:     S. D. Roy
*******************************************************************
/

//`timescale  1ps/1ps

module MotionES_chip(
      clock,
      reset,
      enable,
      step ,
      i1_next, j1_next,
      count_i, count_j,
      i1_initial, j1_initial
      );

input clock, reset, enable;


//input en_MemAddSys;
input [10:0] step, i1_initial, j1_initial;



output [10:0] i1_next, j1_next, count_i, count_j;

wire [10:0] i1_next, j1_next;

wire   [10:0]   count_r,   count_c,   count_i1,   count_j1,   count_r1,
count_c1, count_i, count_j;
wire [7:0] Data_current,  Data_ref;
wire [19:0] Add_current, Add_ref;
wire   en_SAD,   SAVE_SAD,   DEL_SAD,   DEL_SADMEM,   en_MemAddSys,
en_Mem_current, en_Mem_ref;

controller_TSS CTRL(
  clock,
  reset, enable,
```

```
  en_SAD, SAVE_SAD, DEL_SAD, DEL_SADMEM,
  en_MemAddSys, //step, // i1_initial_1, j1_initial_1,
  en_Mem_current, en_Mem_ref  // , en_MV_cal
  );


 MemAddSys MAS1(
  clock,
  reset,
  en_MemAddSys,
  count_r, count_c, count_i1, count_j1, count_r1, count_c1, count_i,
count_j,
  i1_initial, j1_initial,
  step
  );

SAD SAD1(
  clock,  reset,  en_SAD,
  Data_current,  Data_ref, count_i1, count_j1, i1_next, j1_next,
  SAVE_SAD, DEL_SAD, DEL_SADMEM
  );

mem_addr_conv MAC_current(
      count_r,
      count_c,
      Add_current
      );

mem_addr_conv MAC_ref(
      count_r1,
      count_c1,
      Add_ref
      );

Mem_current MC1(
      clock,
      reset,
      en_Mem_current,
      Data_current,
      Add_current);

Mem_ref MR1(
      clock,
      reset,
      en_Mem_ref,
      Data_ref,
      Add_ref);


endmodule
```

B.3.4. Controller of Three Step Search (TSS) module

```
/*****************************************************************
Date:           1/8/2011
File:           controller_TSS.v
Description:
This module controls all the processing of inside each
MotionES_chip module. Bbasically it controls the
operations for each steps of the Three Step Search (TSS) algorithm.
Written By:     S. D. Roy
*******************************************************************
/

//`timescale  1ps/1ps

module controller_TSS(
  clock,
  reset, enable,
  en_SAD, SAVE_SAD, DEL_SAD, DEL_SADMEM,
  en_MemAddSys, //step, // i1_initial_1, j1_initial_1,
  en_Mem_current, en_Mem_ref // en_MV_cal
  //i1_next_1, j1_next_1    // for MemAddSys
  );

input clock, reset, enable;
  //input [10:0] i1_next_1, j1_next_1;
output     en_SAD,   DEL_SAD,   DEL_SADMEM,   SAVE_SAD,   en_MemAddSys,
en_Mem_current, en_Mem_ref; //, en_MV_cal ; //
   //, i1_initial_1, j1_initial_1;

reg    en_SAD,    DEL_SAD,    DEL_SADMEM,    SAVE_SAD,    en_MemAddSys,
en_Mem_current, en_Mem_ref, en_MV_cal ; //
reg [10:0] step;  //, i1_initial_1, j1_initial_1;
//////////////////////
reg [7:0] count_cntr;
reg [9:0] cntr_630;
/////////////// Counter for Enabling diff modules in Motion Chip 1
////////
always @(posedge clock) begin
    if(reset) begin
      count_cntr <= 0;
    end
    else if (enable) begin
      if (count_cntr < 70) begin
      count_cntr <= count_cntr + 1;
      end
      else begin
      count_cntr <= 1;
      end
    end
end
```

```
always @(posedge clock) begin
  if (reset) begin
    cntr_630 <= 0;
  end
  else if (enable) begin

      if (cntr_630 < 630) begin
       cntr_630 <= cntr_630 + 1;
      end
      else cntr_630 <= 1;
  end
end

always @(posedge clock) begin
    if (reset) begin
     // en_Motion_chip1 <= 0;
      en_MemAddSys <= 0;
      //en_MV_cal <= 0;
    end
    else if (enable) begin
     // en_Motion_chip1 <=1;
      en_MemAddSys <= 1;
      //en_MV_cal <= 1;
    end
end


always @(posedge clock) begin
  if(reset) begin
    en_Mem_current <=0;
    en_Mem_ref <=0;
  end
else if (enable) begin
    if ( count_cntr<64) begin   // count_cntr >0 &&
      en_Mem_current <=1;
      en_Mem_ref <=1;
    end
    else if (count_cntr >63 && count_cntr<70) begin
      en_Mem_current <=0;
      en_Mem_ref <=0;
    end
    else begin
      en_Mem_current <= 1;
      en_Mem_ref <= 1;
    end
  end
end

always @(posedge clock) begin
    if (reset) begin
      en_SAD <= 0;
    end
```

```
    else begin
      if ( count_cntr < 65) begin   // count_cntr >0 &&
       en_SAD <= 1;
      end
      else if (count_cntr >64 && count_cntr<70) begin
       en_SAD <= 0;
      end
      else begin
       en_SAD <= 1;
      end
    end
end

always @(posedge clock) begin
    if (count_cntr == 68) begin
      DEL_SAD <= 1;
    end
    else begin
      DEL_SAD <= 0;
    end
end

always @(posedge clock) begin
    if (reset) begin
      SAVE_SAD <= 0;
    end
    else if (count_cntr == 65) begin
      SAVE_SAD <= 1;
    end
    else begin
      SAVE_SAD <= 0;
    end
end

always @(posedge clock) begin
    if (reset) begin
      DEL_SADMEM <= 0;
    end
    else if (cntr_630 == 628) begin
      DEL_SADMEM <= 1;
    end
    else begin
      DEL_SADMEM <= 0;
    end
end
endmodule
```

B.3.5. Memory addressing module

```
/*********************************************************
Date:           1/8/2011
File:           MemAddSys.v
Description:
This module generates appropriate address for memory location
to read out the values in buffers.
Written By:     S. D. Roy
*********************************************************/

//`timescale  1ps/1ps

module MemAddSys(
  clock,
  reset,
  en_MemAddSys,
  count_r, count_c, count_i1, count_j1, count_r1, count_c1, count_i,
count_j,
  i1_initial, j1_initial,
  step
  );

input clock, reset, en_MemAddSys;
input [10:0] step;
input [10:0] i1_initial, j1_initial;
output [10:0] count_r;
output [10:0] count_c;
//output [10:0] count_i;
//output [10:0] count_j;
output [10:0] count_r1;
output [10:0] count_c1;
output [10:0] count_i1, count_j1;
output [10:0] count_i, count_j;

//assign i1_initial = 10'd 3;
//assign j1_initial = 10'd 3;


////////////
reg [7:0] count_70;
wire count_70_done;
wire count_8_done;
//reg count_8_done;


reg [10:0] count_c;
reg [10:0] count_r;
wire [10:0] count_c1;
wire [10:0] count_r1; // reference block
reg [10:0] count_i;
reg [10:0] count_j;
```

```
reg [10:0] count_i1, count_j1;  // reference block
//////////////////////////
reg [10:0] del_c, del_r;
wire [3:0] case_count;
//reg [10:0] count_ca, count_ra;

//////////////
reg en_count_70;
reg [4:0] count_9  ;  // counts upto 70x9 = 630 clockcycles
wire change_i;
wire en_count_c, en_count_r;
reg [6:0] count_40; // for (70*9) * 40 clock cycles in row.. count_j
will increase after (70*9)*40 cycles
wire change_j;

always @(en_MemAddSys) begin
if (reset) en_count_70 <= 0;
else en_count_70 <= en_MemAddSys;
end


always @(posedge clock) begin
  if (reset) begin
    count_70 <= 0;
   // en_count_70 <= 0;
  end
    else if (en_count_70 == 1) begin
      if (count_70 < 69) begin
        count_70 <= count_70 + 1;
      end
      else begin
        count_70 <= 0;
      end
    end
    else begin
      count_70 <= 0;
    end

end

assign count_70_done = (reset)? 0:((count_70 == 69)?1:0);

//////////////////////////////////////////////

assign  en_count_c  =  (reset)?  0:((en_count_70  &&  count_70  >=  0  &&
count_70 < 64)?1:0);
assign en_count_r = en_count_c;

always @(posedge clock)begin
  if (reset) begin
    count_c <= 0;
    //en_count_c <= 0;
```

```
  end
  else begin
    //en_count_c <= 1;
    if (en_count_c) begin
      if (count_c < count_i + 7) begin
        count_c <= count_c + 1;
      end
      else begin
        count_c <= count_i; // load count_i
      end
    end
    else count_c <= count_i;

  end
end
assign count_8_done = (reset)? 0:((count_c % 8 == 7 || count_70 ==
69)?1:0);

always @(posedge clock) begin
  if (reset) begin
    count_r <= 0;
    //en_count_r <= 0;
  end
  else begin
    //en_count_r <= 1;
      if (count_8_done) begin
        if (count_r < count_j + 7) begin
         count_r <= count_r +1;

        end
        else if (count_70_done) begin
                count_r <= count_j;
        end
      end
   end
end

always @(posedge clock) begin
  if (reset) begin
    count_9  <=0;
  end
  else if (en_count_70 && count_70_done) begin
    if (count_9 < 8) begin
      count_9 <= count_9 + 1;
    end
    else begin
      count_9 <= 0;
    end
  end
  else count_9 <= count_9;
end
```

```
assign change_i = (reset)? 0 : ( (count_70 == 68 && count_9 == 8)? 1
: 0);


always @(posedge clock) begin
  if (reset) begin
    count_i <= 0;
  end
  else if (change_i) begin
    if (count_i < 312) begin
      count_i <= count_i + 8;
    end
    else begin
      count_i <= 0;
    end
  end
end


always @(posedge clock) begin
  if (reset) begin
    count_40  <=0;
  end
  else if (en_count_70 && change_i) begin
    if (count_40 < 39) begin
      count_40 <= count_40 + 1;
    end
    else begin
      count_40 <= 0;
    end
  end
  else count_40 <= count_40;
end

assign change_j = (reset)? 0: ((change_i && count_40 == 39) ? 1: 0);

always @(posedge clock) begin
  if (reset) begin
    count_j <= 0;
  end
  else if (change_j) begin
    if (count_j < 232) begin    // Change 232 to your desired value
      count_j <= count_j + 8;
    end
    else begin
      count_j <= 0;
    end
  end
end
```

```
//assign step = 10'd 4;
assign case_count = count_9;

always @(case_count) begin
case (case_count)
  0: begin
      del_c <= 0;
      del_r <= 0;
  end
  1: begin
      del_c <= 10'd 0 - step;
      del_r <= 10'd 0 - step;
  end
  2: begin
    del_c <= 0;
     del_r <= 10'd 0 - step;
  end
  3: begin
     del_c <= step;
     del_r <= 10'd 0 - step;
  end
  4: begin
     del_c <= 10'd 0 - step;
     del_r <= 0;
  end
  5: begin
     del_c <= step;
     del_r <= 0;
  end
  6: begin
     del_c <= 10'd 0 - step;
     del_r <= step;
  end
  7: begin
     del_c = 0;
     del_r = step;
  end
  8: begin
     del_c = step;
     del_r = step;
  end
  default: begin
    del_c <= 0;
    del_r <= 0;
  end

endcase
end


assign count_c1 = (reset)? 0 : ( (!en_count_c)? 0:(   ((count_i +
i1_initial + del_c) < 10'd 0 || (count_i + i1_initial + del_c) >
```

```
10'd 312 || (count_j + del_r + j1_initial) < 10'd 0 || (count_j +
del_r + j1_initial) > 10'd 232) ? (count_c + i1_initial) : (count_c
+ i1_initial + del_c)) );
assign count_r1 = (reset)? 0 : ( (!en_count_r)? 0:(   ((count_i +
i1_initial + del_c) < 10'd 0 || (count_i + i1_initial + del_c) >
10'd 312 || (count_j + del_r + j1_initial) < 10'd 0 || (count_j +
del_r + j1_initial) > 10'd 232) ? (count_r + j1_initial) : (count_r
+ j1_initial + del_r)) );
//assign count_c1 = (reset)? 0 : ( (!en_count_c)? 0:(  ((i1_initial
+ del_c) < 10'd 0 || (i1_initial + del_c) > 10'd 312 || (del_r +
j1_initial)  <  10'd  0  ||  (del_r  +  j1_initial)  >  10'd  232)  ?
(i1_initial) : ( i1_initial + del_c)) );
//assign count_r1 = (reset)? 0 : ( (!en_count_r)? 0:(  ((i1_initial
+ del_c) < 10'd 0 || (i1_initial + del_c) > 10'd 312 || (del_r +
j1_initial)  <  10'd  0  ||  (del_r  +  j1_initial)  >  10'd  232)  ?
(j1_initial) : (j1_initial + del_r)) );



always  @(count_i,  del_c,  en_count_c,  count_j,  del_r,  en_count_r)
begin
  if (reset) count_i1 <= 0;
  else if (en_count_c) begin
    if  ((count_i  +  i1_initial  +  del_c)  <  10'd  0  ||  (count_i  +
i1_initial + del_c) > 10'd 312 || (count_j + del_r + j1_initial) <
10'd 0 || (count_j + del_r + j1_initial) > 10'd 232) begin

      count_i1 <= count_i + i1_initial;
      //count_j1 <= count_j;
    end
    else begin

      count_i1 <= count_i + del_c + i1_initial;
      // count_j1 <= count_j + del_r;

    end
  end
 // else count_i1 <= count_i1;
end

always  @(count_i,  del_c,  en_count_c,  count_j,  del_r,  en_count_r)
begin

  if (reset) count_j1 <= 0;
  else if (en_count_c) begin
    if  ((count_i  +  i1_initial  +  del_c)  <  10'd  0  ||  (count_i  +
i1_initial + del_c) > 10'd 312 || (count_j + del_r + j1_initial) <
10'd 0 || (count_j + del_r + j1_initial) > 10'd 232) begin

      //count_i1 <= count_i;
      count_j1 <= count_j + j1_initial;
    end
```

```
    else begin

      //count_i1 <= count_i + del_c;
      count_j1 <= count_j + del_r + j1_initial;

    end
  end
  //else count_j1 <= count_j1;
end

endmodule
```

B.3.6. Sum of absolute difference (SAD) module

```
/*************************************************************
Date:           1/8/2011
File:           SAD.v
Description:
This module calculates the Sum of Absolute Difference (SAD) between
the current macro-block and the neighboring macro-block in
consideration to find a match for estimating motion vectors
Written By:     S. D. Roy
*************************************************************/

//`timescale  1ps/1ps

module SAD(
  clock,  reset,  en_SAD,
  Data_current,  Data_ref, count_i1, count_j1, i1_next, j1_next,
  SAVE_SAD, DEL_SAD, DEL_SADMEM
  );

input clock, reset, en_SAD, DEL_SAD, DEL_SADMEM, SAVE_SAD;
input [7:0] Data_current;
input [7:0] Data_ref;
input [10:0] count_i1, count_j1;
output [10:0] i1_next, j1_next;

parameter cost = 320*240;  // Choosing a  SAD minimum cost = row*col
//output [10:0] SAD_out;
//output finish_SAD, ACK_SAD;
//reg finish_SAD, ACK_SAD;

reg [14:0] SAD_out;
wire [7:0] diff;
wire [7:0] abs_diff;
reg [14:0] SAD_mem;

//wire [10:0] i1, j1;
reg [10:0] i1_next, j1_next;
```

```
//assign i1 = 8;
//assign i1 = 8;
//reg [7:0] count_SAD;

always @(posedge clock) begin
  if (reset) begin
    SAD_out <= 0;
  end
  else if (en_SAD ==1) begin

    SAD_out <= SAD_out + abs_diff;
  end
  else if (DEL_SAD == 1) begin
    SAD_out <= 0;
  end
  else begin
    SAD_out <= SAD_out;
  end

end

assign diff =(Data_current - Data_ref);
assign abs_diff = (reset)? 0: ((diff[7] ==1) ?(~diff + 1) : diff );
//assign sum = sum + abs_diff;
//assign cost = sum / 64;
//assign cost_fin = (cntr_cost ==64)? 1 : 0 ;

always @(posedge clock) begin
    if (reset) begin
      SAD_mem <= cost;
      i1_next <= 0;
      j1_next <= 0;
    end
    else if (SAVE_SAD) begin
      if (SAD_mem >= SAD_out) begin
        SAD_mem <= SAD_out;
        i1_next <= count_i1;
        j1_next <= count_j1;
      end
      else begin
        SAD_mem <= SAD_mem;
        i1_next <= i1_next;
        j1_next <= j1_next;
      end
    end
    else if (DEL_SADMEM) begin
      SAD_mem <= cost;
    end
    else  SAD_mem <= SAD_mem;
  end

endmodule
```

B.3.7. Address conversion unit

```
/*************************************************************
Date:           1/8/2011
File:           mem_addr_conv.v
Description:
Performs Conversion of memory address to
address suitable memory location. Used both for
current frame memory buffer and reference frame memory buffer
Written By:     S. D. Roy
*****************************************************************/
//`timescale  1ps/1ps

module mem_addr_conv(
      addr,
      addc,
      add_mem);

      parameter ADDWIDTH = 20;
      parameter row_num = 30;
      parameter col_num = 40;

      input [10:0] addr;
      input [10:0] addc;
      output [ADDWIDTH-1:0] add_mem;

      wire [ADDWIDTH-1:0] add_mem;

      assign  add_mem  =  ((addr/8)*col_num*64)  +  ((addc/8)*64)  +
((addr%8)*8 + (addc%8));

endmodule
```

B.3.8. Test Bench

```
/************************************************************
Date:           1/8/2011
File:           test_MotionEstimation.v
Description:
This is the Testbench made for Motion Estimation Module
Written By:     S. D. Roy
************************************************************/

`timescale  1ps/1ps

module test_MotionEstimation();

parameter CLK_PERIOD = 10;
parameter DATA_WIDTH = 8 ;
parameter ADD_WIDTH = 26 ;
```

```
parameter MEM_SIZE = 16384;


reg clock, reset;
reg en_MotionEstimation;

wire [10:0] i1_next_3, j1_next_3 ;
wire [10:0] count_i_3, count_j_3;

Topmodule top(
      clock,
      reset,
      en_MotionEstimation,
      i1_next_3, j1_next_3,
      count_i_3, count_j_3
);

initial begin
  clock <= 1'b0;
  reset <= 1'b1;
  en_MotionEstimation <= 0;
  //step <= 10'd 4;

end

always  #(CLK_PERIOD/2) clock = ~clock;

initial begin
  #15 reset <= 0;


  //#5 en_MemAddSys <= 1;
  #1000000;
end

always@(posedge clock) begin
  if (!reset) en_MotionEstimation <= 1;
end

endmodule
```

B.4. Grain-128a stream cipher

```
--    File: grain128a.vhd
--    Modified By:     S.D. Roy (sdroy@ucalgary.ca)
--    Organization:    ISIS, Dept. ECE, U of Calgary.
--    Algorithm developed by:
--    Martin Agren, Martin Hell, Thomas Johansson and Willi Meier []
--    Original code may be found in:
--    http://eeweb.poly.edu/faculty/karri/stream_ciphers/grain.html
-------------------------------------------------------
--    Description:
--    Grain-128a stream cipher as a source of CSPRG instead of GCF-
FCSR.
--    This module has replaced the gcffcsr.v module from the design
--    speed (1x mode)
--    Initialization with (256+64)=320 ClockCycle
--------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
--use ieee.std_logic_unsigned.all;

entity grain128a is
      port( clk, rst    : in std_logic;
            key    : in std_logic_vector(127 downto 0);
            IV     : in std_logic_vector(95 downto 0);
            o_vld  : out std_logic;
            z      : out std_logic);
end grain128a;


architecture do_it of grain128a is

signal nfsr, lfsr : std_logic_vector(127 downto 0); -- registers
signal lbit, nbit, z_tmp : std_logic; --feedback bit
type statetype is (key_initialize, skip, run);
type Encode is array (0 to 2) of statetype;
type Decode is array (statetype) of std_logic_vector(1 downto 0);
signal state_encode : Encode := (key_initialize, skip, run);
signal state_decode : Decode := ("00","01","10");
signal state : statetype;
signal count : integer;

begin

      -- lfsr feedback function s(i)
      lbit <= lfsr(96) xor lfsr(81) xor lfsr(70) xor lfsr(38) xor
lfsr(7) xor lfsr(0);
      --lbit <= lfsr(62) xor lfsr(51) xor lfsr(38) xor lfsr(23) xor
lfsr(13) xor lfsr(0);
      --lbit <= lfsr(17) xor lfsr(28) xor lfsr(41) xor lfsr(56) xor
lfsr(66) xor lfsr(79);
```

```
      -- nfsr feedback function b(i)
      nbit <= lfsr(0) xor nfsr(0) xor nfsr(26) xor nfsr(56) xor
nfsr(91) xor nfsr(96)
                  xor (nfsr(3) and nfsr(67)) xor (nfsr(11) and
nfsr(13)) xor (nfsr(17) and nfsr(18))
                  xor (nfsr(27) and nfsr(59)) xor (nfsr(40) and
nfsr(48)) xor (nfsr(61) and nfsr(65))
                  xor (nfsr(68) and nfsr(84)) xor (nfsr(88) and
nfsr(92) and nfsr(93) and nfsr(95))
                  xor (nfsr(22) and nfsr(24) and nfsr(25)) xor
(nfsr(70) and nfsr(78) and nfsr(82));

      z_tmp <= ((nfsr(12) and lfsr(8)) xor (lfsr(13) and lfsr(20))
xor (nfsr(95) and lfsr(42)) xor (lfsr(60) and lfsr(79)) xor
(nfsr(12) and nfsr(95) and lfsr(94))) xor lfsr(93) xor (nfsr(2) xor
nfsr(15) xor nfsr(36) xor nfsr(45) xor nfsr(64) xor nfsr(73) xor
nfsr(89) xor nfsr(93));

      -- z(i) = b(2) + b(15) + b(36) + b(45) + b(64) + b(73) + b(89)
+ s(93) + h(x)
      -- h(x) = x0x1 + x2x3 + x4x5 + x6x7 + x0x4x8
      -- b(12); s(8); s(13); s(20); b(95); s(42); s(60); s(79);
s(94);

      -- z_tmp <= nfsr(1) xor nfsr(2) xor nfsr(4) xor nfsr(10) xor
nfsr(31) xor nfsr(43) xor nfsr(56) xor lfsr(25) xor nfsr(63) xor
(lfsr(3) and lfsr(64))
            -- xor (lfsr(46) and lfsr(64)) xor (lfsr(64) and
nfsr(63)) xor (lfsr(3) and lfsr(25) and lfsr(46)) xor (lfsr(3) and
lfsr(46) and lfsr(64))
            -- xor (lfsr(3) and lfsr(46) and nfsr(63)) xor (lfsr(25)
and lfsr(46) and nfsr(63)) xor (lfsr(46) and lfsr(64) and nfsr(63));

      z <= z_tmp; -- output key stream of grain-128a

--lfsr
process(rst, clk)
begin
if (rst = '1') then
      --on reset key initialization
      lfsr <= "01111111111111111111111111111111" & IV;
elsif(clk'event and clk='1') then
      if (state = key_initialize) then
            lfsr(127 downto 0) <= (z_tmp xor lbit) & lfsr(127 downto
1);
      else
            lfsr(127 downto 0) <= lbit & lfsr(127 downto 1);
      end if;
end if;
end process;
```

```
--nfsr
process(rst, clk)
begin
if (rst = '1') then
      --on reset key initialization
      nfsr <= key;
elsif(clk'event and clk='1') then
      if (state = key_initialize) then
            nfsr(127 downto 0) <= (z_tmp xor nbit) & nfsr(127 downto
1);
      else
            nfsr(127 downto 0) <= nbit & nfsr(127 downto 1);
      end if;
end if;
end process;

--state machine
process(rst, clk)
begin
if (rst = '1') then
      state <= key_initialize;
      count <= 0;
      o_vld <= '0';
elsif(clk'event and clk='1') then
      case state is
            when key_initialize =>
                  count <= count + 1;
                  if (count = 255) then
                        state <= skip;
                        count <= 0;
                        o_vld <= '0';
                  elsif (count = 319) then
                        state <= run;
                        count <= 0;
                        o_vld <= '1';
                  end if;
                  when others =>
      end case;
end if;
end process;


end do_it;
```