**ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY**

# SYSTEM ON CHIP IMPLEMENTATION OF NEW INFORMATION HIDING METHOD

**M.Sc. THESIS**

**Utku ESEN**

**Department of Electronics and Communication Engineering**

**Electronics Engineering Program**

**JUNE 2018**

**ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY**

# SYSTEM ON CHIP IMPLEMENTATION OF NEW INFORMATION HIDING METHOD

**M.Sc. THESIS**

**Utku ESEN**
**(504131222)**

**Department of Electronics and Communication Engineering**

**Electronics Engineering Program**

**Thesis Advisor: Assoc. Prof. Dr. Sıddıka Berna Örs YALÇIN**

**JUNE 2018**

**İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ**

# YENİ BİR VERİ GİZLEME YÖNTEMİNİN GELİŞTİRİLMESİ VE YONGADA SİSTEM ÜZERİNDE GERÇEKLENMESİ

**YÜKSEK LİSANS TEZİ**

**Utku ESEN**
**(504131222)**

**Elektronik ve Haberleşme Mühendisliği Anabilim Dalı**

**Elektonik Mühendisliği Programı**

**Tez Danışmanı: Doç. Dr. Sıddıka Berna Örs Yalçın**

**HAZİRAN 2018**

Utku ESEN, a M.Sc. student of ITU Graduate School of Science Engineering and Technology 504131222 successfully defended the thesis entitled “SYSTEM ON CHIP IMPLEMENTATION OF NEW INFORMATION HIDING METHOD”, which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

| | | |
|---|---|---|
| **Thesis Advisor :** | **Assoc. Prof. Dr. Sıddıka Berna Örs YALÇIN**<br>Istanbul Technical University | ..... |
| **Jury Members :** | **Assoc. Prof. Dr. Osman Kaan Erol**<br>Istanbul Technical University | ..... |
| | **Asts. Prof. Dr. Faik Başkaya**<br>Boğaziçi University | ..... |

**Date of Submission :** **4 May 2018**
**Date of Defense :** **4 June 2018**

## FOREWORD

Turkey has been among developing countries for a long time. This situation exists because of its economy, industry and manufacturing power. If we want to increase this status to improved or rich country level, we should consume our energy on behalf of fields like electronics and computer science. These fields will be very popular in the future because of information age. Therefore, we should arrange our life style and government rules according to information age's needs. Thus, Especially for electronic engineers, we should improve our software and hardware development skills. In order to apply this improvement, we should follow and implement state of the art works in literature. I decided to apply for master education for this reason five years ago. I thought in order to improve my country, I should improve my job skills at first. After I started to my master education, I searched one weakness about electronic applications in Turkey. Then, I saw that we should improve our hardware development skills immediately. Thus, I start to work on hardware system design especially for embedded systems.

In this thesis, firstly we aimed at creating new data hiding method then learning hardware design methodology. This is significant because, if someone want to create hardware system, he or she has to learn as well as software. If you want to start to learn hardware system design, you should learn FPGA design because you can create your hardware by software on these devices.

The other way for improving your country is passing on being global. If you are just focus on national events and don't follow the world issues, you can't benefit yourself, you just stay at the same status. Your country also stay at the same place. In order to be global or world wide, the best way for students is to apply for student exchange programs like Erasmus. Therefore, during the thesis period, I applied for Erasmus student exchange program. Thanks to Erasmus, I met with many valuable people, improved my foreign language skills and learned new cultures. I saw that the people are same wherever they come from. Thus, If you have facility to go to exchange program in your university, you should apply for it and see different cultures. I have to say thanks to some people because of their help for this thesis. First, I feel myself lucky because I met my advisor, Sıddıka Berna Örs Yalçın. She directed me very friendly and consciously. I have to say gratitude thanks to other advisor, Stefano Mattoccia. He helped me like his students during my Erasmus period. I should say special thanks to Mehmet Can Döşlü, Uğur Başaran, Erdem Özcan and my family, because they always supported me to write this thesis.

June 2018

Utku ESEN
(Electronics and Communication Engineer)

**TABLE OF CONTENTS**

**Page**

**ABBREVIATIONS**

**FPGA** : Field programable gate arrays
**SOC** : System on chip
**HDL** : Hardware description language
**IP** : Intellectual property
**HLS** : High level synthesis
**CPU** : Computer processor unit
**GPU** : Graphical processor unit
**GNU** : GNU is not unix
**ARM** : Advanced risk machine
**ZED** : Zynq evaluation development
**VGA** : Video graphics array
**HDMI** : High-definition multimedia interface
**PSNR** : Peak signal to noise ratio
**TV** : Television
**DWT** : Discrete wavelet transform
**DCT** : Discrete cosine transform
**LSB** : Least significant bit
**NMI** : Neighbor mean interpolation
**SDT** : Scrambling degree theory
**ROM** : Read only memory
**RAM** : Random access memory
**DDR** : Double data rate
**LPDDR** : Low power double data rate
**ECC** : Error-correction code
**SRAM** : Synchronous random access memory
**OCM** : On chip memory
**NOR** : Not OR(operation)
**NAND** : Not AND(operation)
**i2c** : Inter-integrated circuit
**SPI** : Serial peripheral interface
**UART** : A universal asynchronous receiver-transmitter
**DMA** : Direct memory access
**I/O** : Input and output
**IEEE** : The institute of electrical and electronics engineers
**MAC** : Media access control
**PTP** : The precision time protocol
**GMII** : Gigabit media independent interface
**RGMII** : Reduced gigabit media independent interface
**SGMII** : Serial gigabit media independent interface
**USB** : Universal serial bus
**OTG** : On-the-go
**EHCI** : The enhanced host controller interface

**CAN** : A controller area network
**UTMI+** : USB 2.0 transceiver macrocell interface
**ULPI** : UTMI+ low pin interface
**OSI** : Open systems interconnection
**PHY** : Physical layer of the OSI model
**PS** : Processing system
**PL** : Programmable logic
**AMBA** : The ARM advanced microcontroller bus architecture
**AXI** : Advanced extensible interface
**CLB** : Configurable logic blocks
**LUT** : Look-up table

**SYMBOLS**

| | |
|---|---|
| **dB** | **:** Desibel |
| **C** | **:** One of programming language |
| **C++** | **:** One of programming language |
| **P** | **:** A set called the plaintext space |
| **C** | **:** A set called the ciphertext space |
| **K** | **:** A set called the key space |
| $\mathbf{e_K}$ | **:** Encryption rules |
| $\mathbf{d_K}$ | **:** Decryption rules |
| $\mathbf{K}(\mathbf{x},\mathbf{y})$ | **:** Represent coordinate of pixel |
| **Ds** | **:** Scrambling degree |
| **DSF** | **:** Distance scrambling factor |
| **GSF** | **:** Gray scrambling factor |
| $\mathbf{E}(\mathbf{x})$ | **:** Expected value |
| $\sigma^2$ | **:** Variance |
| **A** | **:** Main point in pixel symmetry |
| **O** | **:** Origin point in pixel symmetry |
| **B** | **:** Symmetry point in pixel symmetry |

## LIST OF TABLES

**Page**

## LIST OF FIGURES

**Page**

**SYSTEM ON CHIP IMPLEMENTATION**
**OF**
**NEW INFORMATION HIDING METHOD**

## SUMMARY

Information security systems are a part of daily life nowadays. For these reason, their applications are getting more and more significant. Steganography and Cryptography are most important fields of it. In this thesis, you can be aware of what Steganography and Cryptograhy are, what their applications and methods are, especially Steganography methods applied on spatial domain. Then, new data hiding method which is improving neighbor mean interpolation method is proposed. Thanks to proposed method, data hiding process is applied according to a key like cryptography approach. Proposed method can be summarized like that changing scan path of embedding process according to symmetry map which is calculated by pixel symmetry.

In addition to explaining new method, the method is also tested on personal computer and embedded hardware. Proposed method is firstly created with Matlab software on desktop computer. Then, proposed method is applied on Zynq embedded system which has hybrid processor architecture. Embedded system application is tested on AVNET's ZedBoard. In order to create this system, firstly development environment of embedded Linux operating system is created on Linux operating system running on personal computer. Then, Embedded Linux run on ZedBoard. In addition, OV7670 camera sensor is integrated with this system and image frame coming from the camera can be writen to memory directly. this frame also can be read for VGA output by using FPGA part of Zynq. In fact, proposed method's data hiding proses is applied by creating software for ARM based processor part of Zynq.

Thanks to this thesis, you can also learn installation of Xilinx development environment, installation of cross-compiling environment for embedded Linux systems, writing device drivers for embedded Linux systems, usage of i2c module inside processor part of Zynq and applying image processing application by creating user space application on embedded Linux operating system.

In this thesis, there are six chapters. In chapter 1, you can infer purpose of writing this thesis and what information security systems are. Then you can examine digital information hiding and scrambling methods in chapter 2. New data hiding method is explained in chapter 3. The implementations about proposed method are showed in chapter 4. Experiments and their results are inferred in chapter 5 and in last chapter conclusions are cited.

Comparing results of previous and proposed methods, you can infer same visual quality and similar computation number. The image which is used for covering, cover image, and the image which is produced after data hiding process, stego image, look similar to each other. You can understand it according to calculation PSNR value between cover and stego images. PSNR results are always stay higher than 35 dB. This result shows that data hiding process is imperceptible.

**YENİ BİR VERİ GİZLEME YÖNTEMİNİN GELİŞTİRİLMESİ**
**VE**
**YONGADA SİSTEM ÜZERİNDE GERÇEKLENMESİ**

## ÖZET

Bilgi güvenlik sistemleri artık gündelik hayatımızın birer parçası. Bu tez çalışmasında bilgi güvenlik sistemlerinin sınıflandırılmasından, amaçlarından, kullanım alanlarından ve birbirlerine göre farklılıklarından bahsedilmiştir. Bu sistemler içerisinde dijital veri gizleme yöntemlerinin neler olduğundan, özelliklede uzamsal uzayda uygulanan yöntemlerin detaylarına değinilmiştir. Daha sonra komşu ortalamalı ara değerlendirme yöntemi kullanılarak resmin içerisine veri gizleme yöntemine yapılan geliştirmeler detaylı bir şekilde okuyucuya aktarılmıştır. Bu geliştirme sayesinde veri gizleme işlemini, şifreleme biliminin yöntemine benzer şekilde anahtar yapısına bağlı olarak yapılması sağlanmıştır. Yeni yöntem; uzamsal uzayda görüntü içerisine veri gizlerken izlenilen işlem yolunun veya sırasının, genelde resim çerçevesinin sol üst köşesinden başlayarak sağa ve aşağıya doğru satır-sütun taraması yapılarak izlenen zig-zag yolun, şifreleme bilimine uygun olarak şifreleme anahtarı yapısına bağlı bir biçimde karıştırılması ve bu sayede tek bir iterasyonda dahi, şifreleme anahtarını bilmeyen kişilerin gizlenen mesaja ulaşamamasını sağlayan bir sistem olarak nitelendirilebilir. Yeni yöntem sayesinde kullanıcılar hem steganografinin yani gizleme biliminin hem de kriptografi yani şifreleme biliminin özelliklerinin iç içe bulunduğu bir yöntem ile haberleşmelerinin güvenliğini arttırabileceklerdir.

Bu tez çalışmasında, geliştirilen yeni yöntemin gerçek bir uygulama ile test edilmesi de sağlanmıştır. Önerilen yöntem ilk olarak masaüstü bilgisayar üzerinde Matlab yazılımı kullanılarak test edilmiştir. Bu kapsamda veri gizleme ve gizlenmiş veriyi çıkartma işlemlerinin yapıldığı programlar geliştirilmiştir. Bu programlar ekler bölümünde okuyucuyla paylaşılmıştır. Ayrıca gömülü sistem üzerinde aynı uygulamanın koşturulması sağlanmıştır. AVNET'in Zynq işlemci mimarisine geliştirmeler yapılması için ürettiği ZedBoard geliştirme kartı üzerinde önerilen yöntem uygulamaya alınmıştır. Bu kapsamda Linux işletim sisteminin ZedBoard üzerinde koşturulması için gerekli geliştirme ortamı kurulmuş, daha sonra kart üzerinde Linux işletim sistemi koşturulmuştur. Ayrıca, OV7670 kamera sensörü sisteme entegre edilip, kameradan görüntünün alınıp kartın üzerindeki belleğe yazılması, bellekteki görüntünün VGA protokolü ile dışarıda bulunan bir VGA monitörde yansıtılması sağlanmıştır. Burada belirtilen işlemlerin tümü, Zynq mimarisisnin içinde bulunan programlanabilir kapı dizileri kullanılarak gerçekleştirilmiştir. Tüm bunlara ek olarak, VGA çıkışına gönderilecek resim çerçevesi üzerinde, Zynq mimarisi içerisinde bulunan ARM tabanlı çift çekirdekli A9 işlemciler koşturularak, bu makalede yenilik unsuru olarak önerilen, piksel simetrisi kullanarak resim çerçevesinin içerisine veri gizleme yönteminin uygulanması sağlanmıştır.

Bizler bu çalışma sayesinde Xilinx firmasının ürettiği programlanabilir kapı dizileri ile çalışabilmek için gerekli olan geliştirme ortamının kurulmasını, Linux işletim sisteminin gömülü sistem üzerinde koşturulması için gerekli olan çapraz derleme geliştirme ortamının kurulmasını, Linux işletim sisteminde özel çevresel birimlerinin kullanılması için gerekli olan sürücü yazılımlarının Linux çekirdeği içerisinde nasıl kodlandığını, yine Linux işletim sisteminde kabuk katmanında I2C donanımının nasıl kullanıldığı ve görüntü işleme için gerekli olan altyapının nasıl kontrol edileceğini, daha önceden yüksek seviyeli programlama dilleri C veya C++ kullanılarak oluşturulmuş progranabilir kapı dizileri modüllerinde bulunan problemlerin nasıl çözülmesi gerektiğini öğrenmiş olduk.

Bu tez altı ana başlık altında incelenmektedir. İlk bölümde bilgi güvenliği sistemlerinden ve bu tezin amacından bahsedilmiştir. Bilgi güvenliği sistemleri güvenliği sağlayan ve güvenliği sınayan sistemler olarak iki gruba ayrılmıştır. Bu tezin içinde bulunduğu güvenliği sağlayan sistemler içerisinde; şifreleme, gizleme ve fligran oluşturma bilim alanlarının neler olduğundan bahsedilmiştir. Bu bilim alanlarının birbirlerine göre amaçları ve farklılıkları nelerdir açıklanmıştır.

İkinci bölümde veri gizleme yöntemlerinden, özelliklede uzamsal uzayda kullanılan temel yöntemlerden ve görüntü karıştırma algoritmalarından bahsedilmiştir. Bu algoritmalardan tez için önem teşkil eden komşu ortalamalı ara değerlendirme yöntemi detaylı bir şekilde anlatılmıştır. Sonrasında görüntü karıştırma algoritmalarının değerlendirlmesinde kullanılan karıştırma derecesi yöntemi anlatılarak tezin yenilik unsuru olan, piksel simetrisi kullanarak gizleme yolunun karıştırılması işleminin, diğer karıştırma yöntemleri ile karşılaştırılabilmesini sağlamak için teorik bir alt yapı okuyucuda oluşturulmuştur.

Üçüncü bölümde veri gizleme için önerilen yeni yönteme değinilmiştir. Öncelikle daha önce kullanılan ara değerlendirme yöntemlerinin arasından niçin komşu ortalamalı ara değerlendirme yönteminin seçildiğine değinilmiştir. Daha sonra piksel simetrisi olarak isimlendirilen yeni yöntemin, aslında birçoğumuzun geometriden hatırlayabileceği simetri kavramının resim çerçevesinin oluşturulmasını sağlayan temel yapıtaşları yani pikseller üzerinde nasıl kullanıldığı anlatılmıştır. Yeni yöntemin uygulanması ile ilgili örnekler verilmiştir.

Dördüncü bölümde önerilen yöntemin bilgisayar ve gömülü sistem üzerinde gerçeklenen uygulamalarının nasıl yapıldığından bahsedilmiştir. Öncelikle Matlab üzerinde yapılan uygulamanın detayları anlatılmıştır. Daha sonra Linux işletim sistemi ve Zynq mimarisi üzerinde yapılan uygulamanın detaylarına geçilmiştir. Bu noktada Zynq mimarisinin önemli noktaları vurgulanmıştır. FPGA'lerin kullanım alanlarından ve özelliklerinden bahsedilmiştir. Bilgisayar mimarisinde çevresel birimlerin çip içi haberleşmesinde sıklık ile kullanılan AXI protokolünden ve bu protokolün değişik amaçlar için oluşturulan versiyonlarından bahsedilmiştir. Daha sonra, OV7670 kamera sensörünün özelliklerine ve video çıkışı olarak sunduğu YUV/YCbCr ve VGA protokolünde kullanılan RGB renk uzaylarına değinilmiştir. Son olarak, Zynq mimarisinin FPGA kısmında kurulan donanım bileşenleri ve yazılım mimarisi incelenmiştir.

Beşinci bölümde, dördüncü bölümde bahsedilen uygulamaların çalışma esnasında elde edilen çıktılarından bahsedilmiştir. Veri gizlemesi yapılmış resim sonuçları okuyucuya sunulmuştur. Daha sonra gömülü sistem üzerinde yapılan uygulamanın

sonuçları, uygulama sırasında kullanılan kontrol bilgisayarındaki Linux terminalinden elde edilen ekran görüntüleri gösterilerek okuyucuya sunulmuştur.

Son olarak altıncı bölümde test sonuçlardan yola çıkarak tezin değerlendirilmesinin yapıldığı sonuç bölümü yazılmıştır. FPGA de gerçekleştirilen devrelerin, FPGA içinde bulunan kaynakların ne kadarını tükettiği tablolar ile gösterilmiştir. Uygulamalardan elde edilen sonuçların olumlu ve olumsuz yönlerine değinilmiştir. Gelecekte yapılabilecek geliştirmelerden bahsedilmiştir.

Ekler bölümünde, dördüncü bölümde bahsedilen gerçeklemelerin nasıl yapıldığını anlatan teknik detaylar verilmiştir. Ek-A'da öncelikle geliştirme ortamının kurulumu anlatılmıştır. Xilinx geliştirme araçlarının kurulumu, Linux kaynak kodlarının nasıl indirileceği ve Minicom seri haberleşme programının nasıl yükleneceğine değinilmiştir. Daha sonra Xilinx Linux işletim sisteminin ZedBoard geliştirme kartı üzerinde koşturulması için gerekli teknik adımlar anlatılmıştır. İlk adım sistem yükleyicisinin nasıl oluşturulacağından, U-Boot programının Zynq mimarisininde bulunan ARM işlemcisi için nasıl derleneceğinden, Linux çekirdeğinin, kök dosya sisteminin ve donanım sürücülerinin nasıl derleneceğinden ve son olarak ZedBoard üzerinde ilk Xilinx Linux'un koşturulması için geliştirme kartı ve test bilgisayarına yapılması gereken adımlardan bahsedilmiştir. Ek-B bölümünde Linux işletim sistemi için yüklenebilir donanım sürücülerinin oluşturulması anlatılmıştır. Donanım sürücülerinde Major ve Minor sayılarının neleri ifade ettiğine değinilmiştir. Linux işletim sistemi içerisinde kesme yapılarının nasıl oluşturulduğu ve Linux kullanıcı katmanında oluşturulan programların nasıl derlenmesi gerektiğinden bahsedilmişitir. Ek-C bölümünde Matlab programında yeni yöntemin çalıştığını göstermek amacıyla oluşturulan programların kodlarına yer verilmiştir. Bu amaçla önce veri gizleme uygulamasına ait kod, simetri noktalarının hesaplanması için oluşturulan fonksiyon ve gizlenen veriyi çözme işleminin yapıldığı kod okuyucuyla paylaşılmıştır. Son olarak Ek-D bölümünde, önerilen yöntemin aynı zamanda gömülü sistemlerde uygulamak için uygun olduğunu göstermek amacıyla oluşturulan, ZedBoard üzerinde Linux işletim sistemi kullanılarak koşturulan C kodları okuyucuya sunulmuştur. İlk olarak FPGA'de oluşturulan özel donanımların Linux işletim sistemi ile kontrolünü sağlayan donanım sürücüsünün kodu, daha sonra I2C ile kamera konfigürasyonunu sağlayan ve resim çekme işlemini gerçekleştiren Linux kullanıcı katmanı programlarının C kodları ve son olarak da veri gizleme işleminin yapıldığı programın C kodu okuyucular ile paylaşılmıştır.

Önerilen yeni yöntemin sonuçları ile bir önceki yöntemin sonuçları karşılaştırıldığında benzer görsel özelliklerin ve hesaplama basitliğinin yakalanmış olduğu gözlenmiştir. Üzerinde değişiklik yapılacak olan kaplama resmi ile içerisine veri gizlenmiş saklı resimin, stego resim, arasında hesaplanan PSNR değerlerinin her bir örnek çalışma için 35dB'in üzerinde kaldığı gözlenmiştir. Bu durum veri gizleme işlemi esnasında yapılan değişikliklerin gözle görünemeyecek seviyede olduğunu göstermektedir. Ayrıca şifreleme anahtarı kullanılarak yapılan gizleme sayesinde güvenlik seviyesi bir önceki yönteme göre çok daha güçlü bir hale getirilmiştir. Şifreleme anahataı kullanılarak karıştırılan gizleme sırasının, oldukça karmaşık ve takip edilmesi zor bir hale geldiği, karıştırma derecesi teoremi kullanılarak teorik olarak kanıtlanmıştır. Bu tezin çıktısı olarak şifreleme ve gizleme bilimlerine uygun, gerçek dünyada uygulanabilir, basit ama güçlü ve gerçeklemelerle uygulanabilirliği kanıtlanmış yeni bir veri gizleme yöntemi elde edilmiştir.

# 1. INTRODUCTION

Electronic security systems are getting more and more significant part of our daily life. Many sample applications about these systems can be given such as; banking and military security systems [1–3], secured communication applications on mobile phones [4] and encrypted TV broadcasts [5]. All of these example applications include many cryptography methods in order to provide security. On the other hand, many systems try to break these security systems in order to obtain their valuable information. Therefore, scientist need to find another solution to provide security for their valuable systems. With the help of our ancestors, finding new security methods for digital security systems is very straightforward because even if they couldn't utilize any electronic systems, they found very clever methods, which are also suitable for implementation on electronic systems. As a result, digital steganography and watermarking techniques can be mentioned. These techniques aim at hiding communication or becoming aware of risky situations like disturb the data and information theft. Thus, implementation of such systems are getting more and more significant nowadays and will be in the future. Figure 1.1 shows an example of steganography application.

## 1.1 Purpose of Thesis

As it is told before, digital steganography and watermarking applications are playing key role among electronic security systems. Therefore, in this thesis, we purpose to improve one of the existing digital steganography and watermarking methods in order to provide more secured and invisible information hiding technique. Then, proposed method is implemented on hybrid processor system, Xilinx Zynq, within Linux operating system. In fact, we aim to create an embedded system infrastructure for image processing applications, especially for information hiding techniques in order to hide photographer information inside his/her photograph art. Thus, we aimed to learn Zynq architecture and Zynq Evaluation Development Board (ZedBoard)

specifics, hardware design methodology with Vivado HLS software, basics of Linux operating system and writing device driver for custom IPs (Intellectual Properties), embedded system design with Vivado software, OV7670 camera sensor specifics and development of new information hiding method with Matlab software.

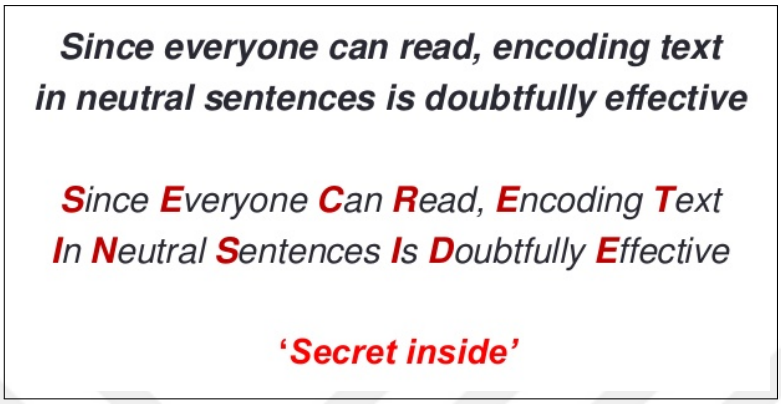


**Figure 1.1** : An example of steganography application.

## 1.2 Information Security Systems

Information security systems can be divided into two main sections, which are safety providing systems [6] and safety testing or breaking systems [7]. In this thesis, safety providing systems are mainly focused on , especially on digital information hiding methods.

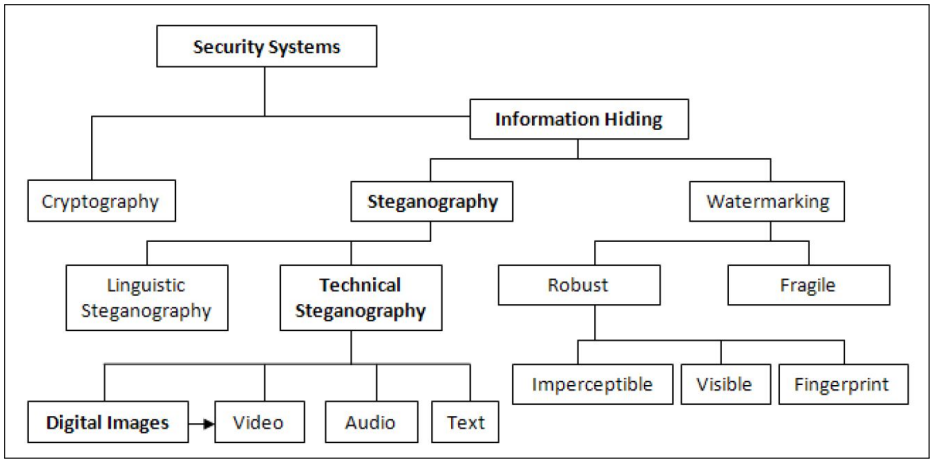


**Figure 1.2** : Classification of safety providing systems [17].

As you can see from Figure 1.2, safety providing systems can be analyze in two sections. These are cryptography and information hiding. Information hiding also

can be divided in two subsections, which are steganography and watermarking. In order to understand more correctly to this thesis aim, differences between security providing techniques must be showed. However, firstly these techniques and what are their purposes must be explained.

### 1.2.1 Cryptography

Cryptography is that all techniques which are utilized for converting data from comprehensible state to inexplicable state. It includes many scientific areas such as mathematics, computer science and electrical engineering [8]. Any data or information can be encrypted such as document [9], speech [10], image [11] or video stream [12].

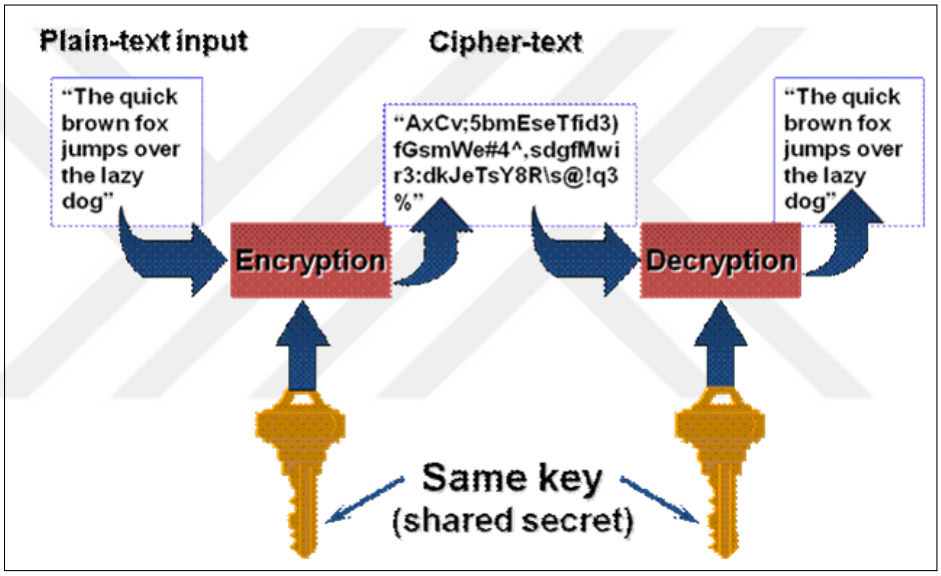


**Figure 1.3** : Five-tuples of modern cryptosystems.

Modern cryptosystem is defined by using the following five-tuples [13], which are showed in Figure 1.3;

1. P - a set called the plaintext space
2. C - a set called the ciphertext space
3. K - a set called the key space
4. $e_K$ - encryption rules
5. $d_K$ - decryption rules

Plaintext is converted to ciphertext by using encryption rules and key at the message sender side, then ciphertext is converted by using decryption rules and the same key

at the receiver side in modern cryptosystems. The purpose of modern cryptosystem is that just person or system who knows key can figure out the plaintext from ciphertext according to encryption key. Even if the person or system can obtain the ciphertext, if they don't know encryption key, they should not figure out the plaintext. Thus, it is acceptable that everybody knows encryption and decryption rules, message channel and there is a encrypted communication between sender and receiver but just sender and receiver knows the encryption key and valuable information in modern cryptosystems.

### 1.2.2 Information hiding

Information hiding means concealing any type of data like file, document, speech, image or video within another data. Information hiding techniques are divided by two main fields, which are steganography and watermarking.

#### 1.2.2.1 Steganography

The word steganography combines the Greek words steganos and graphien. Steganos means covered, concealed or protected. Graphien means writing. The history of steganography techniques are spreading for a long time scale. It was starting with Herodotus. Histiaeus sent a message to his vassal, Aristagoras, by shaving the head of his most trusted servant, "marking" the message onto his scalp, then sending him on his way once his hair had regrown, with the instruction, When thou art come to Miletus, bid Aristagoras shave the head and look thereon [14]. It is represented in Figure 1.4.

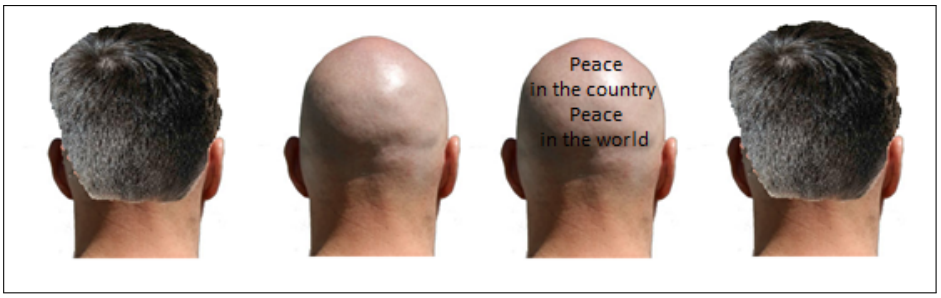


**Figure 1.4** : Representation of historical steganography example.

Digital image steganography is a method which is applied in computer science for hiding information inside a digital image. Three basic elements are utilized in order to hide information. These are cover image, binary message and stego image. Cover image is an image wanted to embed or hide our binary message. Binary message can

be any kind of data like text, speech, image or video represented by binary domain. Stego image is a result of steganography process, which includes both cover image and binary message. Stego image must be very similar to cover image in order to provide security in communication. These basic elements are showed in data hiding diagram in Figure 1.5. In steganography systems, the most significant factor is median of sent message must be indistinguishable. For instance, when message is embedded on an image, it should not be figured out by others [15–17].

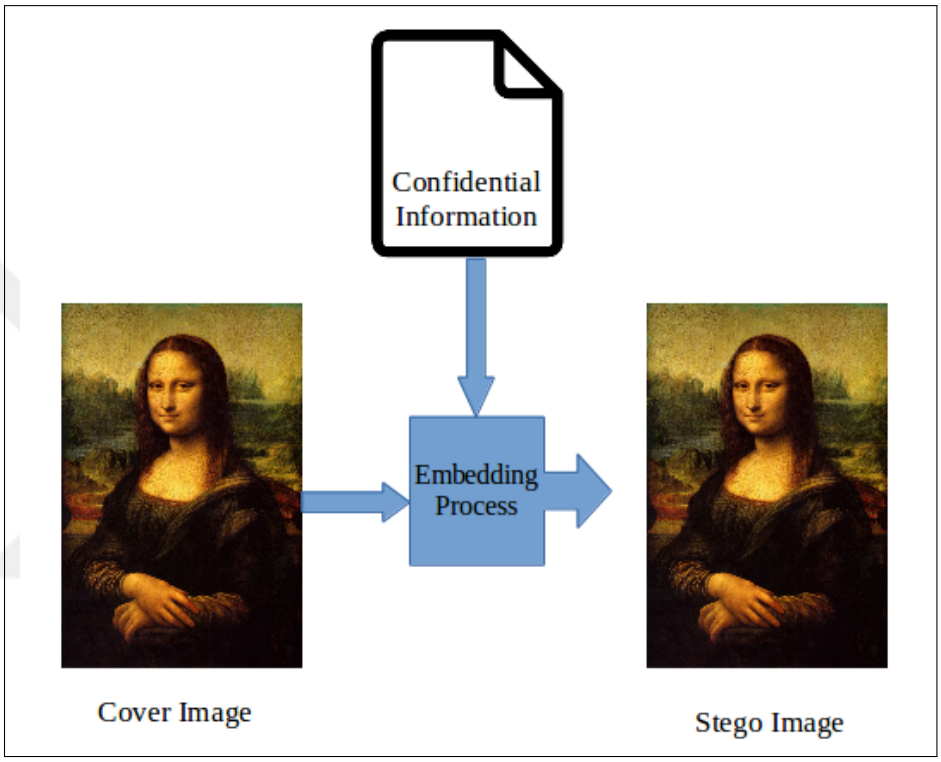


**Figure 1.5** : Data hiding diagram.

#### 1.2.2.2 Watermarking

Watermarking is another information hiding technique, which purposes to mainly identify ownership of the information [18]. Watermarking has often been used in our daily life. For instance, copyright protection, source tracking (different recipients get differently watermarked content), broadcast monitoring (Television news often contains watermarked video from international agencies), video authentication, software crippling on screen casting programs to encourage users to purchase the full version to remove it and content management on social networks [18]. These are showed in Figure 1.6. The main idea of watermarking technique is to protect owner of information from any changes and theft.


**Figure 1.6** : Watermarking examples, left (Copyright protection), middle (Broadcast monitoring) and right (Source tracking).

### 1.2.3 Comparison of information security systems

Comparing purposes of cryptography, steganography and watermarking, it can be said that cryptography wants to change information to inexplicable shape, steganography wants to hide communication channel and watermarking wants to protect ownership rights. Cryptography techniques apparently differ from both steganography and watermarking techniques. However, steganography and watermarking techniques can be similar to each other.

# 2. DIGITAL INFORMATION HIDING METHODS

## 2.1 Steganography Methods

Steganography methods can be classified in two sections with respect to which domain used during the modification of cover image [19]. If some changes are made on frequency domain coefficients, this kind of methods are called by frequency domain information hiding methods. Otherwise, if modifications are applied on spatial domain coefficients, these methods are named after spatial domain information hiding methods.

Compering information hiding methods about their property, some parameters become significant. These parameters are robustness, perceptibility and capacity [18].

Information hiding technique is called "fragile" if it fails to be detectable after the slightest modification [18]. Fragile methods are commonly used for tamper detection. Information hiding technique is called semi-fragile if it resists benign transformations, but fails detection after malignant transformations. Semi-fragile techniques are commonly used to detect malignant transformations.

Information hiding technique is called robust if it resists a designated class of transformations like rotation, cropping and scaling. Robust watermarks may be used in copy protection applications to carry copy and no access control information. Information hiding technique is called imperceptible if the original cover signal and the marked signal are perceptually indistinguishable.

Information hiding technique is called perceptible if its presence in the marked signal is noticeable (e.g. Digital On-screen Graphics like a Network Logo, Content Bug, Codes, Opaque images) [18]. Otherwise, if hiding process can not be realized, this technique is called imperceptible. Capacity represents maximum number of bit can be embedded inside cover image [18]. It generally depends on directly image size. However, some methods can be effected from visual characteristics of cover image.

### 2.1.1 Frequency domain methods

While spatial techniques involve manipulation of pixels of the cover image, frequency domain techniques involve manipulation of coefficients of the cover image [20]. The coefficients are obtained by transforming the cover image in time domain to a frequency domain through a specific transformation function. Since the manipulation of images is involved, spatial domain techniques in spite of their good fidelity criteria exhibit poor tolerance towards a wide range of external attacks which is not desirable. Further, since spatial techniques involve manipulation of pixels, pixel level modification may not be suited for images which may cost severely on the content of image which is not a comprehensible event to a very small extent. The transform of a signal is just another form of representing the signal. It does not change the information content present in the signal. Hence, the frequency domain transforms is utilized and in specific the multi resolution properties of certain transforms like discrete wavelet transform (DWT), contourlet transform and the robustness properties of certain transforms like discrete cosine transform (DCT) as you can see in Figure 2.1.

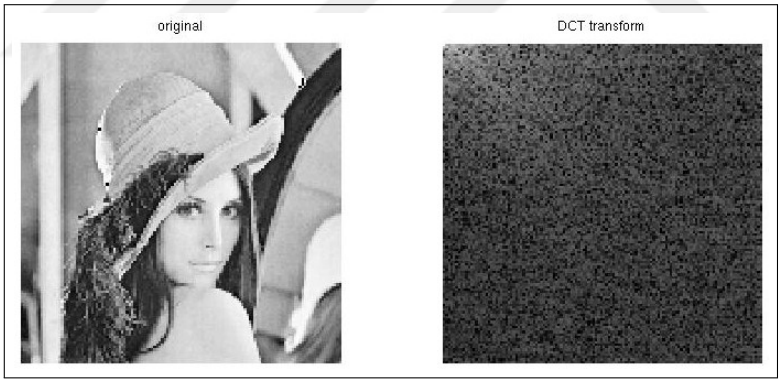


**Figure 2.1** : Example of DCT transform.

### 2.1.2 Spatial domain methods

While working on spatial domain, some changes are made on pixel values [20]. Generally, these changes are expected to be invisible. Thus, there must be some pixel value changing techniques which should keep image quality and integrity. Therefore, humankind has found several techniques. However, two of them represent main idea of other techniques. These two fundamental techniques are changing least significant bit (LSB) of all pixels and modifying interpolation values of image frame [15].

### 2.1.2.1 LSB technique

LSB changing technique is a common and very simple method for information hiding [21]. In this technique, LSB of all pixels in an image are changed according to binary message. In order to apply this technique, firstly confidential message should be converted into binary stream, then all LSB of color channels are modified according to this binary stream through whole pixels of the cover image. LSB technique is represented in Figure 2.2.

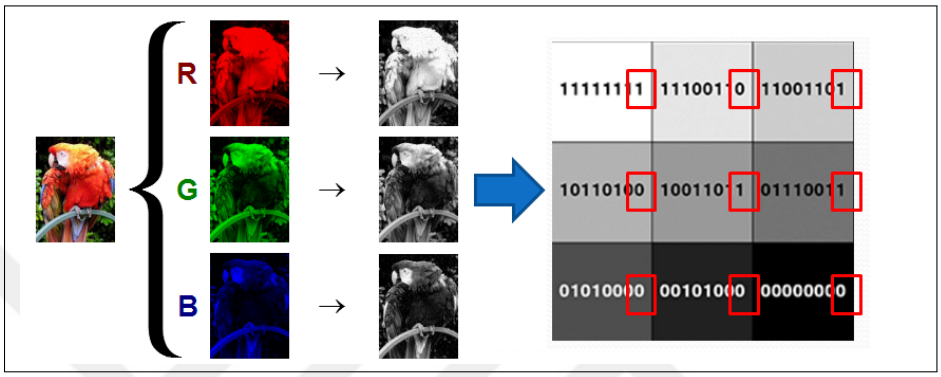


**Figure 2.2** : Representation of LSB technique.

### 2.1.2.2 Neighbor mean interpolation (NMI) method

Image interpolation process is used for calculating unknown pixel values by utilizing known pixel values of image. It is one of the major operations about creating or producing digital images from camera sensor [22]. On the other hand, general process is computed on screens of electronic devices, like rotation, enlarging and reducing images, need to utilize these methods [23]. In fact, image interpolation can be used for data hiding applications [15].

There are many methods in literature for image interpolation process. Giving some examples for traditional interpolation methods, the nearest neighbor [24], linear [25], bilinear [26] and neighbor mean interpolation methods can be mentioned [15]. The nearest neighbor and linear interpolation methods are cited by simple interpolation methods and they are used for re-sampling [25]. Nearest neighbor interpolation method suffers from normally unacceptable aliasing effects with regard to enlarging and reducing images. Bilinear interpolation method determines the grey level from the weighted average of the four closest pixels to the specified input coordinates and it assigns a value to the output coordinates. This method generates an image

that has a smoother appearance than nearest neighbor. In fact, bilinear interpolation method requires three or four times higher computation time than the nearest neighbor method [15]. Neighbor mean interpolation method is similar to bilinear interpolation method. However, this method has less blurring and greater image resolution.


**Figure 2.3** : Image pixels.

In neighbor mean interpolation method, unknown pixel values are calculated by using mean of two or three the neighbor pixels. In example image on Figure 2.3, $K(1,1)$, $K(1,3)$, $K(1,5)$, $K(3,1)$, $K(3,3)$ and $K(3,5)$ are known pixel values and shown by green color. $K(1,2)$, $K(1,4)$, $K(2,1)$, $K(2,2)$, $K(2,3)$, $K(2,4)$, $K(2,5)$, $K(3,1)$ and $K(3,4)$ are inter values which are wanted to be calculated and they are shown by yellow color. Value of $K(2,1)$ is calculated by meaning known values of $K(1,1)$ and $K(3,1)$. Value of $K(1,2)$ is calculated by meaning known values of $K(1,1)$ and $K(1,3)$. Value of $K(2,2)$ is calculated with equation 2.1 [15].

$$K(2,2) = \frac{K(1,1)+K(1,2)+K(2,1)}{3} \tag{2.1}$$

These processes can be repeated for $K(1,4)$, $K(2,3)$, $K(2,4)$, $K(2,5)$, $K(3,1)$ and $K(3,4)$ by following zig-zag way. The calculated pixels are showed below sequentially; $K(1,2)$, $K(2,1)$, $K(2,2)$, $K(1,4)$, $K(2,3)$, $K(2,4)$, $K(2,5)$, $K(3,2)$, and $K(3,4)$. Embedding binary message inside interpolation values in this technique is

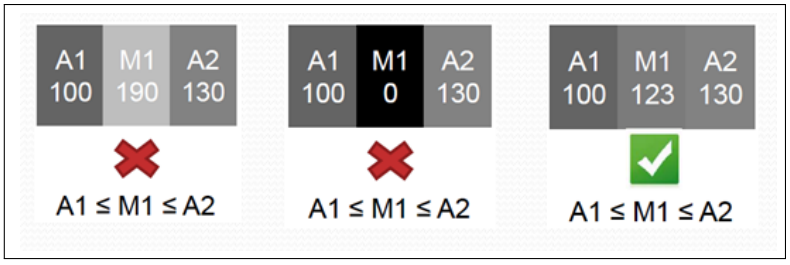


**Figure 2.4** : Interpolation must be invisible.

main concept of this thesis. In order to provide invisible hiding, interpolation value

must be kept between neighbor pixel values as you can see from Figure 2.4. Therefore, it must be calculated how many bits can be embedded onto interpolation value. Answer of this question is represented by "n" and it is calculated according to neighbor pixel values of interpolation value with the help of equations 2.2, 2.3 and 2.4. Then, n number of bit are read from binary message stream and add on interpolation value.

$$n(1,2) = \lfloor \log_2 |K(1,2) - K(1,1)| \rfloor \tag{2.2}$$

$$n(2,1) = \lfloor \log_2 |K(2,1) - K(1,1)| \rfloor \tag{2.3}$$

$$n(2,2) = \lfloor \log_2 |K(2,2) - K(1,1)| \rfloor \tag{2.4}$$

There is an example given in Figure 2.5 about data hiding proccess.

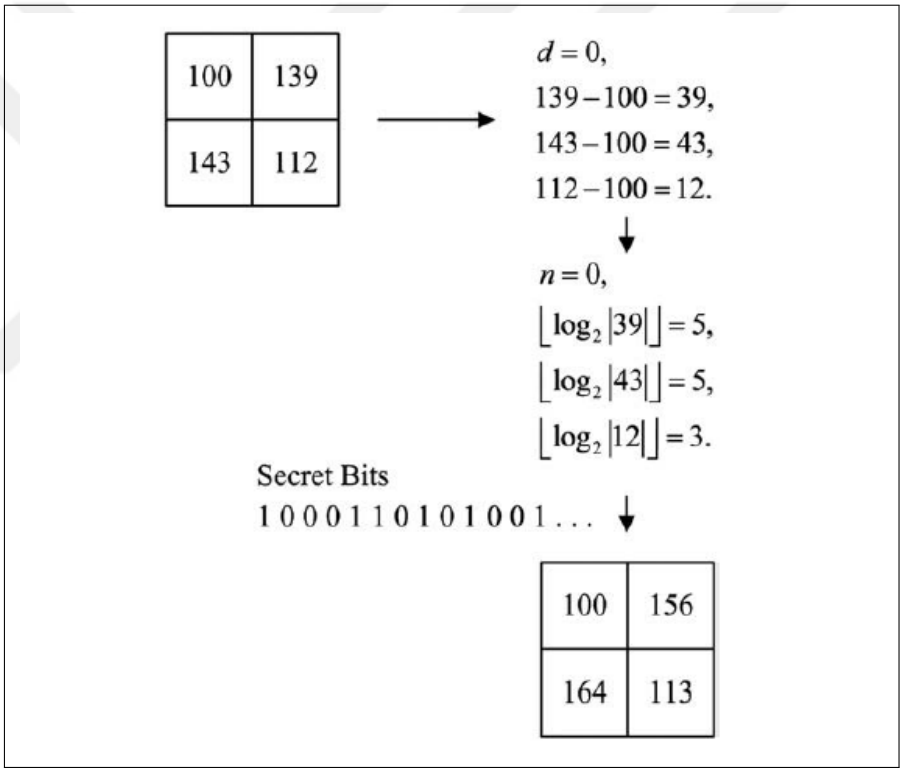


**Figure 2.5** : Data hiding proccess example [15].

In data extracting process, similar to hiding process, interpolation values must firstly be calculated then it is calculated how many number of bit can be added each interpolation values. Then, differences between interpolation values of stego image and calculated interpolation values of each interpolation pixels are measured. Finally, binary message is created according to number of added bit and difference of each pixels. This process starts from left - top side of image and follows same path with data hiding process. There is an example given in Figure 2.6 about data extracting process.

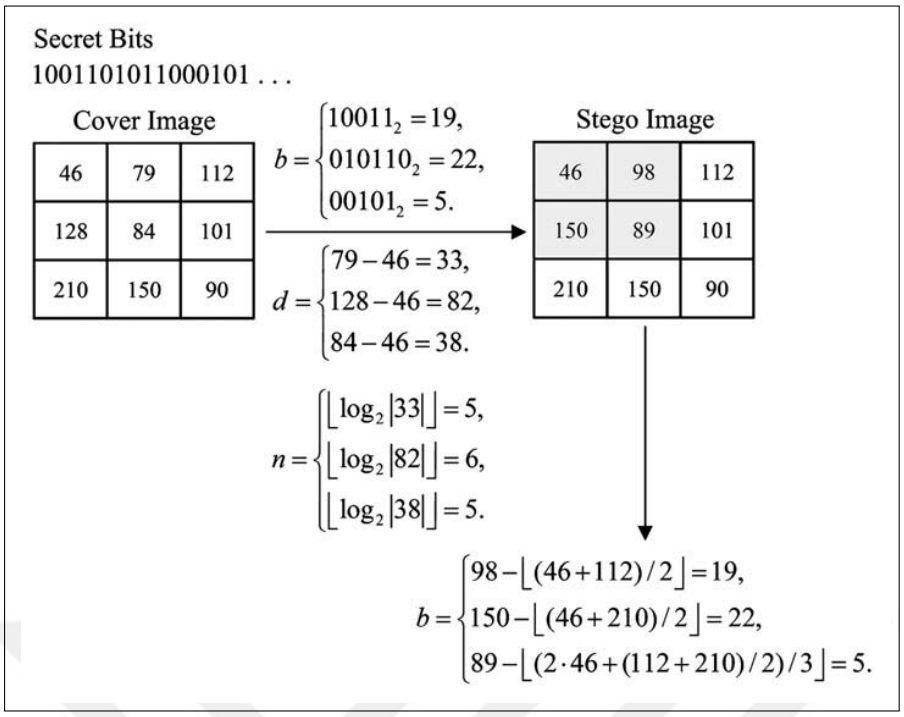


**Figure 2.6** : Data extracting proccess example [15].

## 2.2 Image Scrambling Techniques

Image scrambling techniques are utilized in cryptography and steganography fields. Scrambling methods are generally applied for two ways, which are changing embeding path and the other one is scrambling plaintext according to encryption key or table in steganography field. These techniques are compared according to Scrambling Degree Theory (SDT) in order to show their performans differences. Thus, firstly SDT is mentioned. Then, Rubik's Cubic Algorithm, Arnold Transformation and Sudoku Puzzle scrambling methods are explained in this section.

### 2.2.1 Scrambling degree theory

According to the concept of the scrambling transformation, the scrambling of position space is the original image pixel position has been moved essentially, If the pixel has been moved farther away comparing to the original image pixels, the degree of scrambling is higher [27]. While scrambling does not change the original image pixel gray level, but can change the image of the visual effects. Scrambling the image compare to the original image more "chaos ", indicating that the scrambling algorithm

is more effective. A more "chaos" image should be intuitive visual was "chaotic", and the overall distribution of relatively uniform gray image. According to the analysis above, the degree of image scrambling is not only considering the distance to move the image pixels, but also considering the intuitive visual effect of image. Using the formula of Ds below can objectively shows the image scrambling degree.

$$DS = DSF \times GSF \tag{2.5}$$

DSF (Distance scrambling factor) is determined by the distance of the pixel movement, GSF (Gray scrambling factor) is calculated by quantifying the intuitive visual data.

In general, the image pixel position to move the more further away, the greater the degree of scrambling is, so it can be used to represent the scrambling degree by moving distance. Move away from here by pixel to calculate the mean and variance of DSF.

Definition 1: Assumes that the image $A_{M\times N}$ scrambled into the image $A'_{M\times N}$ , a pixel position (x, y) is mapped to scrambled images (x', y') position, then the move distance of pixels is:

$$\mathrm{d(x,y)} = \sqrt{(x-x')^2 + (y-y')^2} \tag{2.6}$$

The mean moves distance of whole image is:

$$E(d) = \frac{1}{M \times N} \sum_{x=1}^{M} \sum_{y=1}^{N} \mathrm{d(x,y)} \tag{2.7}$$

Obviously, when each pixel in the image does not move, it means when the image does not scrambling, $E(d)$ is the minimum value $E_min(d) = 0$; when the image of each pixel move diagonal distance, $E(d)$ is Max.

$$E_{max}(d) = \sqrt{(M-1)^2 + (N-1)^2} \tag{2.8}$$

The distance of pixel scrambled moved higher means that the degree of movement is greater. Therefore, DSF can be calculated by

$$DSF(A,A') = E(d)/E_{max}(d) \tag{2.9}$$

According to the previous analysis that if the image is divided into the same size and do not re-place sub-block, then scrambling the image average gray level of each sub-block closer to the image "chaos " the greater the degree. As a result, the following method can be used to calculate GSF.

Divied Scrambling image and original image into $k \times k$ pixels in size and do not overlap sub-block, set sub-image $B_{m\times n}(i,j)$ stand for the image of the $(m,n)$ sub-blocks, then the sub-image $B_{m\times n}(i,j)$ of gray level $E(B_{m\times n})$ is:

$$E(B_{m\times n}) = \frac{1}{k\times k}\sum_{i=1}^{k}\sum_{j=1}^{k} B_{m\times n}(i,j) \tag{2.10}$$

There are $(Mk)\times(Nk)$ sub-block after block of the image, such as mean and variance of equation gray showed below by formula 2.11 and 2.12.

$$E(E(B'_{m\times n})) = \frac{1}{(M/k)\times(N/k)}\sum_{m=1}^{M/k}\sum_{n=1}^{N/k} E(B'_{m\times n}) \tag{2.11}$$

$$\sigma^2 = \frac{1}{(M/k)\times(N/k)}\sum_{m=1}^{M/k}\sum_{n=1}^{N/k}(E(B'_{m\times n}) - E(E(B'_{m\times n})))^2 \tag{2.12}$$

Described as a random variable variance relative to the mean wave, if the variance is smaller, the wave degree is smaller, the variance is larger, and the wave degree is larger. Therefore, the gray level sub-image can be used to describe the variance of degree of image scrambling. If the variance is smaller, the difference between the average gray code between is smaller, is means the degree of "chaos" is higher.

If the sub-scrambling image means gray level variance is $\sigma^2_{new}$, the original image of the sub-image intensity mean and variance is $\sigma^2_{org}$ org, then the value of intuitive visual GSF is:

$$GSF = \frac{\sigma^2_{org}}{\sigma^2_{new}} \tag{2.13}$$

According to the equation above, the more GSF larger, the image more "chaos" , which compare to the original image, and the scrambling better.

### 2.2.2 Rubik's cubic algorithm

Rubik's cubic was invented in 1974 as a famous wisdom game. In the beginning, it is a cubic with 6 different colors in each side (6 faces) as shown in Figure 2.7.

Rubik's cubic possesses 6 faces and can be divided into $54(6faces\times 3\times 3)$ elements. In the beginning, the hidden data (treated similar as an image) will be partitioned into different unit block size such as pixel based, $3\times 3$ pixels based, or other $n\times n$ pixels based. Then, 54 units will be selected sequentially and transformed into 6 faces according to the six faces of a Rubik's cubic by designated an index number as shown

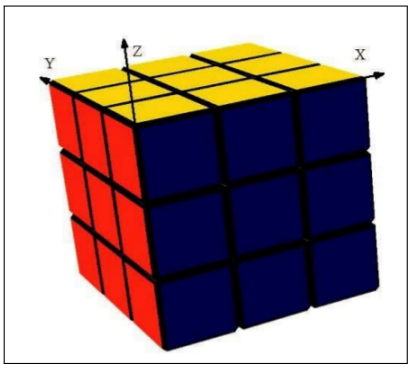


**Figure 2.7** : A Rubik's Cubic indexed with direction parameter.

in Figure 2.8 and Figure 2.9. Therefore, an image can be partitioned into a lot of different 54 units of blocks and formed a lot of different Rubik's cubic. To apply the Rubik's cubic for image data hiding, the basic process unit can be one pixel, small block, or macrocell (large block) is compared to the traditional Rubik's Cubic. For example, an image can be partition by pixels to fit and associated with each of the small cubic of a Rubik's Cubic. Therefore, 54 pixels totally can be fit into the Rubik's Cubic and each pixel represents a small block. An image can also be partitioned based on $3 \times 3$, i.e. 9 pixels, as a small block. Thus, 54 $3 \times 3$ blocks can be fit into the Rubik's Cubic and each $3 \times 3$ block represents a small block of the Rubik's Cubic. Each Rubik's cubic can be assigned a different random number for performing rotation to scramble the sequence of original 54 units.

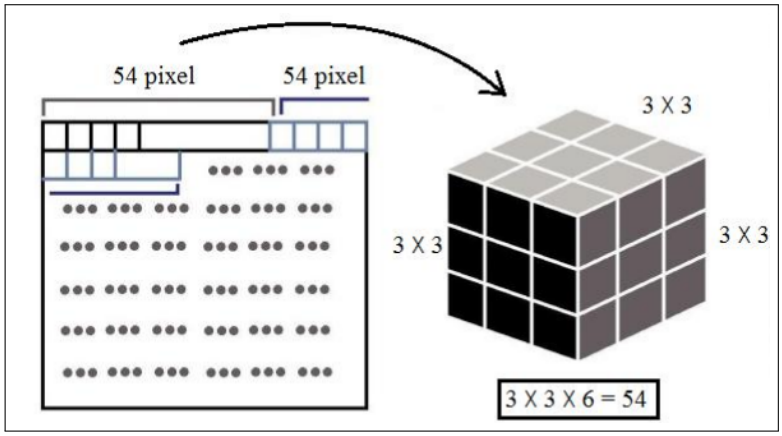


**Figure 2.8** : Mapping of Rubik's Cubic and image.

In the proposed data hiding, the data hiding process is performed from left to right and then top to bottom in the cover image, i.e., horizontally, with the covert information. In the proposed scheme, some parameters are utilized for controlling the process of data scrambling and data embedding listed below.

- Macrocell parameter $M_p$: It is used to specify scrambling is either pixel or block based.
- Hiding method parameter $H_p$ : Specify which data hiding is used.
- Rotation parameter $R_p$ : Specifies number of rotation of Rubik's cubic block and its direction.
- Rotation regulation parameter $R_r$ : Specifies all of the macrocells use the same or different rotation parameter for performing scrambling.

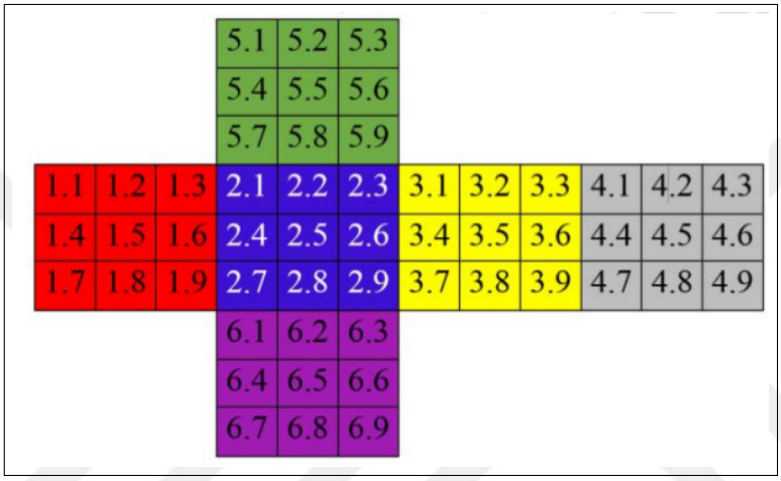


**Figure 2.9** : Corresponding index of Rubik's Cubic.

Proposed data hiding approach is implemented by the following procedure:

1. Define the required $M_p$ , $H_p$ , $R_p$ and $R_r$ parameters.
2. Hidden data is encrypted by the cipher system in order to strengthen the data security.
3. The encrypted data is scrambled by applying the Rubik's Cubic rotation.
4. The scrambled data is embedded into the cover image to obtain the stego-image.

The hidden data can be extracted by performing the above steps reversely.

### 2.2.3 Arnold transformation

Images are composed of discrete units called pixels. A pixel is the basic unit representing some color value, which when taken together form the image. The image is a $m \times n$ matrix, where m represents the number of rows of pixels and n the number of

columns of pixels, and each entry in the matrix being a numeric value that represents a given color. For example, consider the $175 \times 175$ image of a caffeine molecule in Figure 2.10.

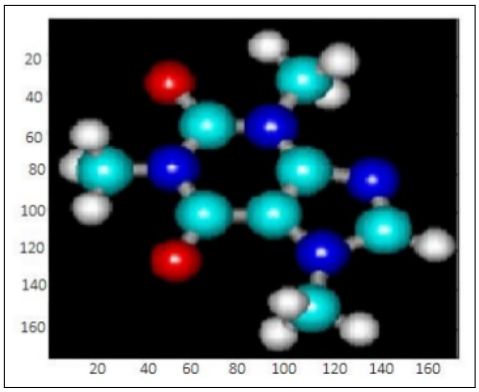


**Figure 2.10** : $175 \times 175$ image of a caffeine molecule.

Let X be the image matrix shown below, it is possible to examine selected entries in X. The numeric entries represent some color value. The mapping known as Arnolds Cat Map is named after the mathematician Vladimir I. Arnold, who first illustrated it using a diagram of a cat. It is a simple and elegant demonstration and illustration of some of the principles of chaos namely, underlying order to an apparently random evolution of a system.

Arnold's cat map is the transformation $\Gamma \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x+y \\ x+2y \end{bmatrix} \text{ mod } n$ Where mod is the modulo of the $\begin{bmatrix} x+y \\ x+2y \end{bmatrix}$

For understanding the mechanism of the transformation better, it can be decomposed into elemental pieces.

1. Shear in the x-direction by a factor of 1.
   $\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x+y \\ y \end{bmatrix}$
2. Shear in the y-direction by a factor of 1.
   $\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x \\ x+y \end{bmatrix}$
3. Evaluate modulo.
   $\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \end{bmatrix} \text{ mod } n$

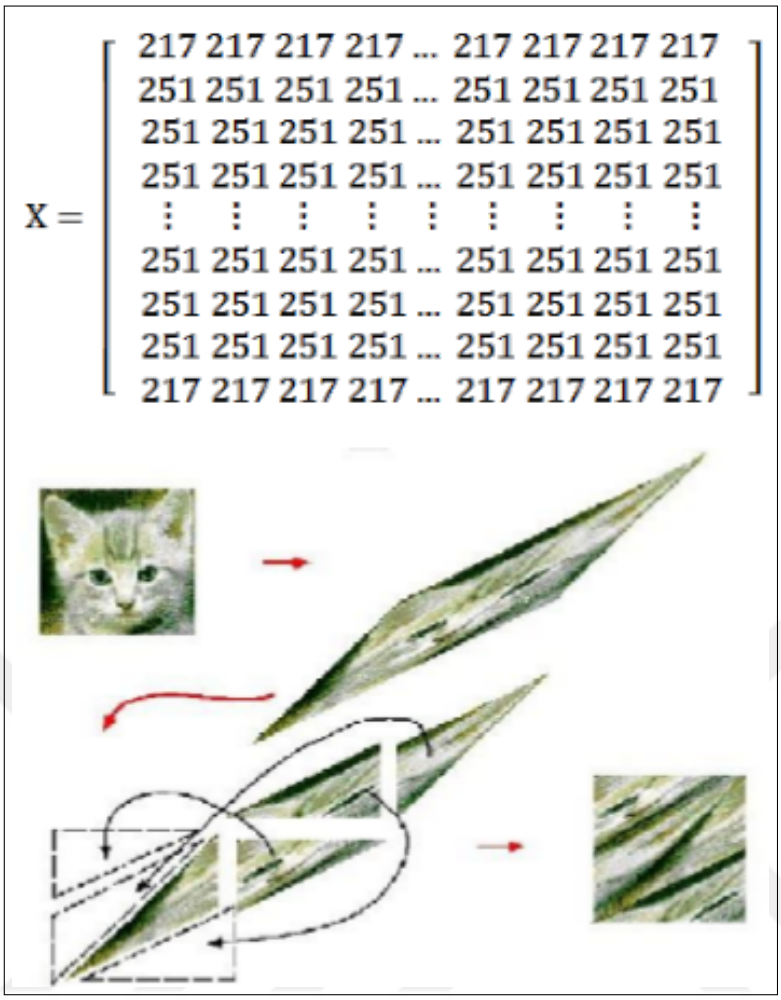


**Figure 2.11** : Visuals illustrating the steps.

Figure 2.11 shows the shearing in the x and y directions, followed by modulo operation and then the reassembly of the image.

### 2.2.4 Using sudoku puzzle

In 2011, Zou, Tian, Xia, and Song introduced an image scrambling method using Sudoku puzzle [28]. This method securely scrambles images making them appear to contain no information. The proposed method uses pairs of Sudoku puzzles to map original and scrambled images. The method takes a pair of Sudoku puzzles and modifies it so there is a 1-1 relationship between the digits of the puzzles. It adds the digits corresponding to column number in front each of the digits for the puzzle corresponding to the original image. It does the same with row numbers to the puzzle for the scrambled image. It then scrambles the image by taking a pixel in the original image, locating the digit entry in the Sudoku puzzle in the same place as the pixel, and

moving it to the corresponding digit in the other puzzle. The proposed method takes advantage of the Sudoku rule to create this 1-to-1 correspondence between puzzles. The method also take benefits from the large number of Sudoku solutions to provide security against unscrambling attempts.

The scrambling algorithm for this method is divided into four parts: Sudoku pair selection, Sudoku pair preparation, image marking and mapping, and bit scrambling. This discussion also specified how to unscramble the image.

#### 2.2.4.1 Sudoku pair selection

The first step is the Sudoku puzzle pair selection. In this step, one must simply make pairs of Sudoku puzzles. The pairs can be of any size and there can be any number of pairs. Having many different pairs can be beneficial to improve the security of the method.

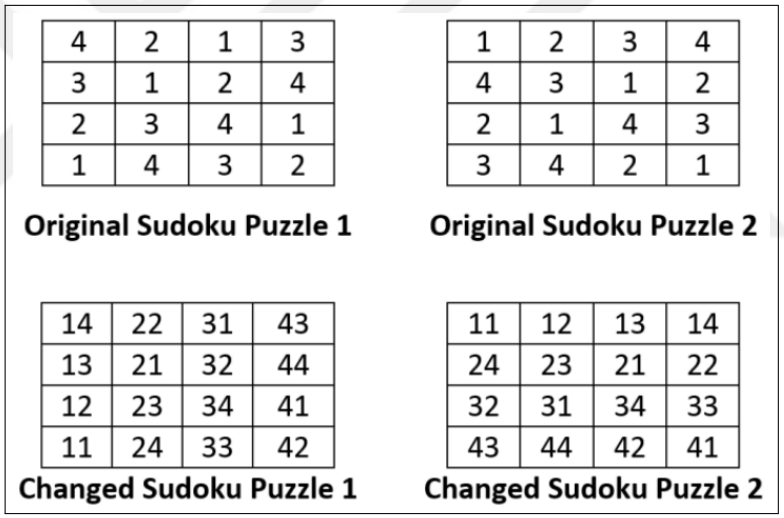


**Figure 2.12** : Sudoku Pair Preparation.

#### 2.2.4.2 Sudoku pair preparation

In the second part, it need to establish 1-to-1 relations between the puzzles in each pair. This can be done by adding a prefix to each of the digit entries in order to make them all unique. This way there is exactly one of each entry in the first puzzle for each entry in the second. Then modify the entries in the first puzzle with the formula $NewValue = OldValue + Column \times 10^{Digits}$ , Where digits is the number of digits in the puzzle. Note that this formula simply adds a row prefix to each entry. Figure 2.12 is given for exemplifying this process.

#### 2.2.4.3 Sudoku image marking and mapping

The third part uses these prepared pairs of Sudoku puzzles to establish a relation between the original image and the scrambled one. This part is sub-divided into two sub-parts: block scrambling and sub-block scrambling.

#### 2.2.4.4 Sudoku block scrambling

In block scrambling, use the Sudoku pairs to scramble blocks of the same size in the original image. In this step the first Sudoku puzzle in a pair is used to mark the pixel positions of the original image. Then place that pixel in the equivalent entry for the second Sudoku puzzle. The following steps and figure explain the process in more detail:

1. For the $i^{th}$ pixel in the original image block $p_i$ , take the $i^{th}$ entry in the first Sudoku puzzle $a_i$.
2. Locate the entry in the second Sudoku puzzle such that $b_j = a_i$.
3. Set the $j^{th}$ pixel in the scrambled image to be $s_j = p_i$.
4. Repeat these steps until all pixels in the block have been processed.

#### 2.2.4.5 Sudoku sub-block scrambling

In sub-block scrambling, they take each scrambled block and break it up into smaller sub-blocks, then repeat the same process from block scrambling with these smaller blocks as it is shown in Figure 2.13.

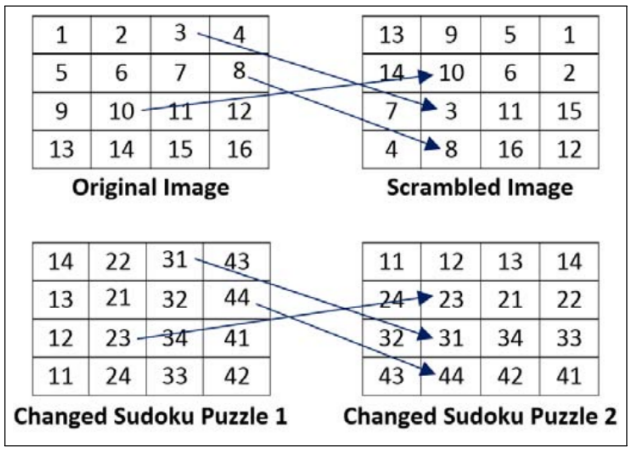


**Figure 2.13** : Block scrambling using Sudoku Pairs.

#### 2.2.4.6 Sudoku bit scrambling

After the third part, the image is not sufficiently scrambled and still appears to show some information in the scrambled image and in the histogram. For this reason, it need the fourth part: bit scrambling. In this part, take the bits of the image and modify them so it is possible to treat them like a 2-D grid. To do this first flatten the scrambled image into a 1-D grid by connecting rows to each other. The grids length is P, where P is the number of pixels. For each pixel in the grid, create a column containing its binary representation, giving us a 2-D grid of size $8 \times P$. There are at most 8 rows because pixel values range between 0 and 255. Then reshape the grid into a square of size $M \times M$, where M is the floor of square root of $8 \times P$. This is performed by reshaping by going through entries row-by-row and adding them to the square grid. Then perform the same puzzle pair scrambling process to this grid and obtain new pixel values in the image. This process is shown in Figure 2.14.

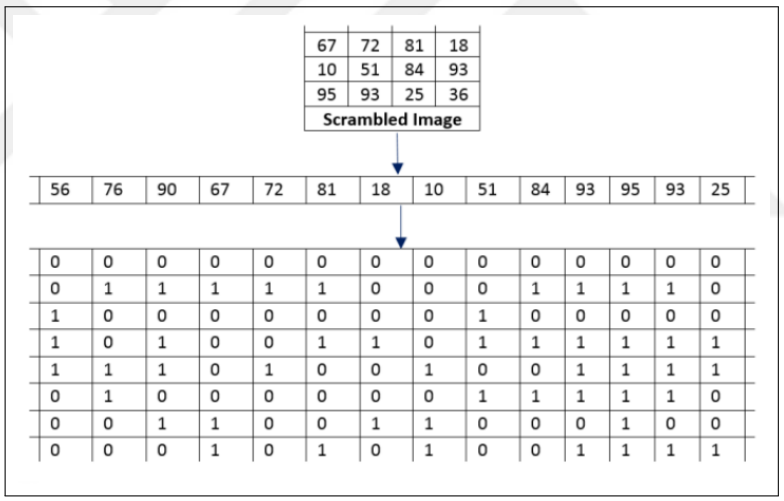


**Figure 2.14** : Bit scrambling matrix generation.

# 3. PROPOSED METHOD

In this chapter, we propose a new information hiding technique which is very similar to information hiding with neighbor mean interpolation method. With the help of new technique, binary message can be embedded according to a key like cryptography techniques. In order to obtain more secured information hiding system, a new embedding method which is named by pixel symmetry is created. Thanks to this method, algorithm which is more secured, has same image quality and similar computation number is provided.

## 3.1 The Purpose Of Proposed Method

In order to provide robust information hiding method, there is an indispensable parameter which is named by visual quality. Visual qualitiy is significant parameter of steganography field, which is described by visual similarity between stego image and cover mage according to human eye. In mathematics field, visual similarity represents by PSNR value. If stego image and cover image are not visually similar,

Original

Pixel Symmetry, PSNR = 40,977dB

**Figure 3.1** : PSNR value of Pixel Symmetry (sufficient visual quality).

visual similarity is insufficient for hiding that there is a embedding process on the stego image. As a result, PSNR value is lower than $35dB$. Otherwise, the stego image is sufficient about visual quality. Proposed method satisfies, if the visual quality of stego image is higher than $35dB$ like previous method [15].

As you can see from Figure 3.1, Figure 3.2 and Figure 3.3, there are three methods applied on original image for data hiding. These methods are LSB5, LSB6 and Pixel Symmetry. LSB5 method modify fifth bit of pixels and LSB6 method modify sixth bit of pixels according to plaint text message. Pixel Symmetry method is proposed in this thesis and it is explained in detail in Subsection 3.3.2. As a result of figures, while PSNR value is decreasing, visual quality of stego image is also decreasing. Therefore, proposed method must provide PSNR value higher than $35dB$ in order to get sufficient visual quality.

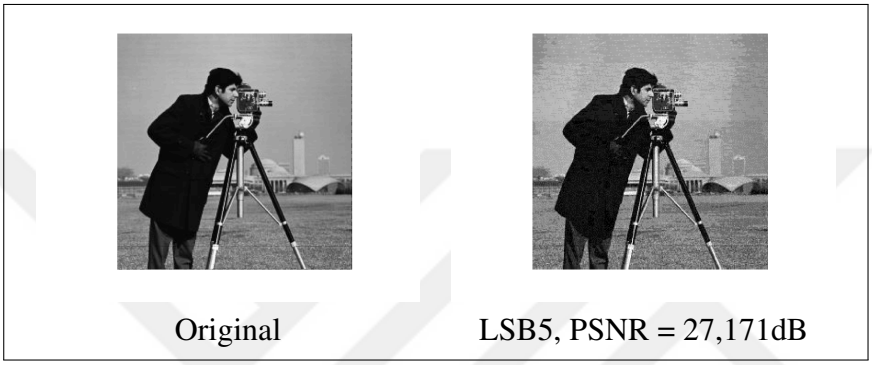


**Figure 3.2** : PSNR value of LSB5 (insufficient visual quality).

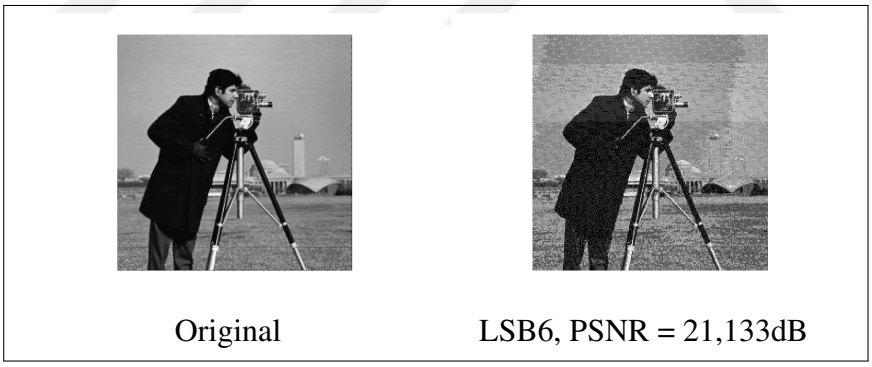


**Figure 3.3** : PSNR value of LSB6 (insufficient visual quality).

As it is mentioned before, binary message is embedded inside each interpolation values of cover image, starting from left-top side of image and following zig-zag path with neighbor mean interpolation method. However, people can follow many different scan paths as they are shown in Figure 3.4.

The first idea of the thesis is to transform embedding path to more complex one than scan paths showed in Figure 3.4 according to a cryptographic key as it is shown in Figure 3.5. If the key is changed, the embedding path should be changed. On the

other hand, the second idea is the transformation must be simple in order to keep computation number similar.

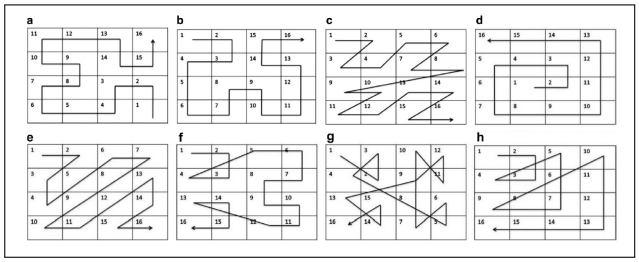


**Figure 3.4** : Scan paths in literature. (a) Moore, (b) Hilbert, (c) z-scan, (d) s-scan, (e) d-scan, (f) b-scan, (g) x-scan and (h) a-scan [29].

$$\begin{bmatrix} x & 1 & x & 4 & x & 7 & x \\ 2 & 3 & 5 & 6 & 8 & 9 & x \\ x & 10 & x & 13 & x & 16 & x \\ 11 & 12 & 14 & 15 & 17 & 18 & x \\ x & 19 & x & 22 & x & 25 & x \\ 20 & 21 & 23 & 24 & 26 & 27 & x \\ x & 28 & x & 29 & x & 30 & x \end{bmatrix}$$

Default embedding path

$$\begin{bmatrix} x & 22 & x & 19 & x & 25 & x \\ 17 & 15 & 14 & 12 & 11 & 27 & x \\ x & 13 & x & 10 & x & 16 & x \\ 8 & 6 & 5 & 3 & 2 & 18 & x \\ x & 4 & x & 1 & x & 7 & x \\ 26 & 24 & 23 & 21 & 20 & 9 & x \\ x & 29 & x & 28 & x & 30 & x \end{bmatrix}$$

Scrambled embedding path according to Key_1= [(2,2)(2,2)(2,2)]

$$\begin{bmatrix} x & 25 & x & 22 & x & 19 & x \\ 17 & 12 & 14 & 18 & 11 & 15 & x \\ x & 16 & x & 13 & x & 10 & x \\ 8 & 3 & 5 & 9 & 2 & 6 & x \\ x & 7 & x & 4 & x & 1 & x \\ 26 & 21 & 23 & 27 & 20 & 24 & x \\ x & 30 & x & 29 & x & 28 & x \end{bmatrix}$$

Scrambled embedding path, Key_2= [(3,2)(2,2)(4,5)]

**Figure 3.5** : Default and scrambled embbeding paths.

In order to change embedding path according to a key, the scrambling methods which are explained in Chapter 2 are investigated. However, all methods scramble image by appliying scrambling process on blocks of the image. Therefore, DS values of all these methods are not adequate for appliying them one iteration. At the sum, we proposed a new image scrambling method which is modified especially for data hiding with neighbor mean interpolation method. As you can remember from Chapter 2,

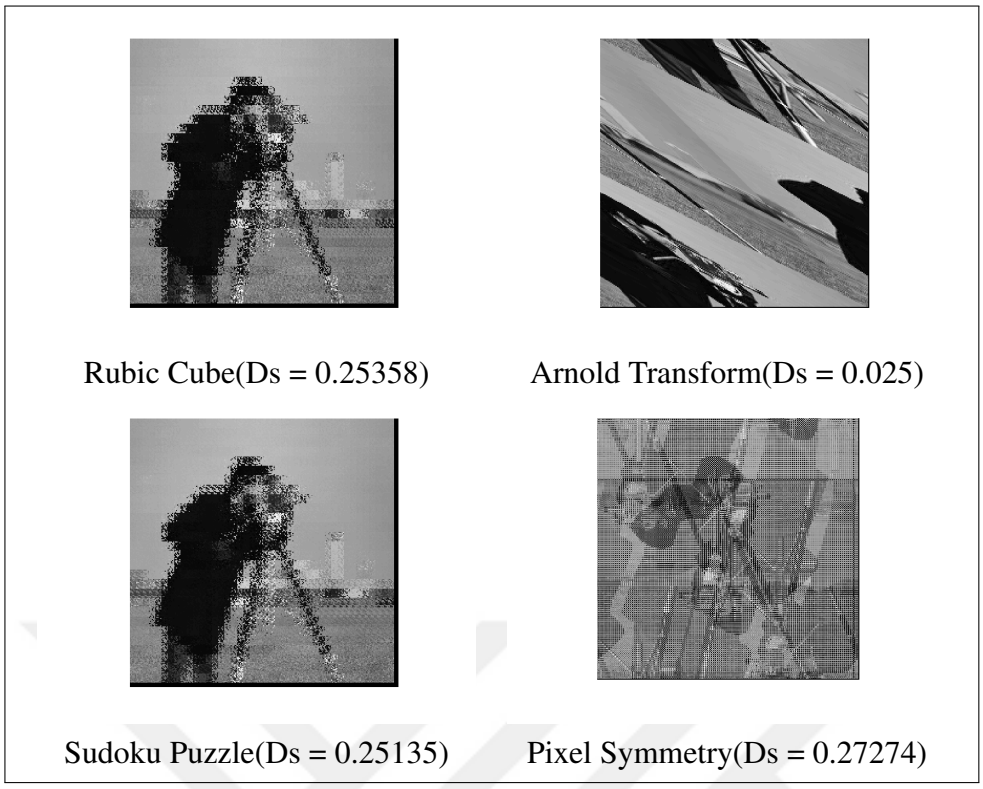


**Figure 3.6** : Results of scrambling methods after one iteration.

DS value is going 1 when stego and cover images are same. Therefore, DS value should be close to 1 as much as possible. Results in Figure 3.6 show that DS value of previous methods are 0.25358 for Rubic Cube, 0.025 for Arnold Transform and 0.25135 for Sudoku Puzzle. On the other hand, DS value of the proposed method is 0.27274. Thus, the proposed method has similar DS value according to previous methods, which shows the proposed method can be applied for scrambelling process like other methods. The proposed method is utilized for embedding path scrambelling process rather than previous methods because its DS value is a little bit more higher than other methods.

### 3.2 Choosing Embedding Method

There are several embedding methods in literature. four of them is mentioned in Chapter 2. Among the four methods, two of them are DWT and DCT transforms in frequency domain, the others are LSB and NMI techniques in spatial domain. Because of complexity, frequency domain approaches are not implemented in this thesis. LSB is a very simple method but its embedding capacity is less than NMI as you can infer

from Table 3.1 and Figure 3.7. Embedding capacity is explained by maximum number of bit can be embedded inside cover image in Chapter 2. As you can see from Figure 3.7, NMI capacity of all images are bigger than LSB capacities weather or not their frequency characteristic is different because NMI capacity is affected from gradients number in the cover image. Gradients number is total number of vertical and horizantal derivatives on an image [30].

**Table 3.1** : Capacity of cover images

| Image | Oktay | Cameraman | Sezen |
|---|---|---|---|
| Size(Pixel) | 256x256x3 | 256x256x3 | 256x256x3 |
| NMI Capacity(Bit) | 766299 | 994662 | 819145 |
| LSB Capacity(Bit) | 196608 | 196608 | 196608 |

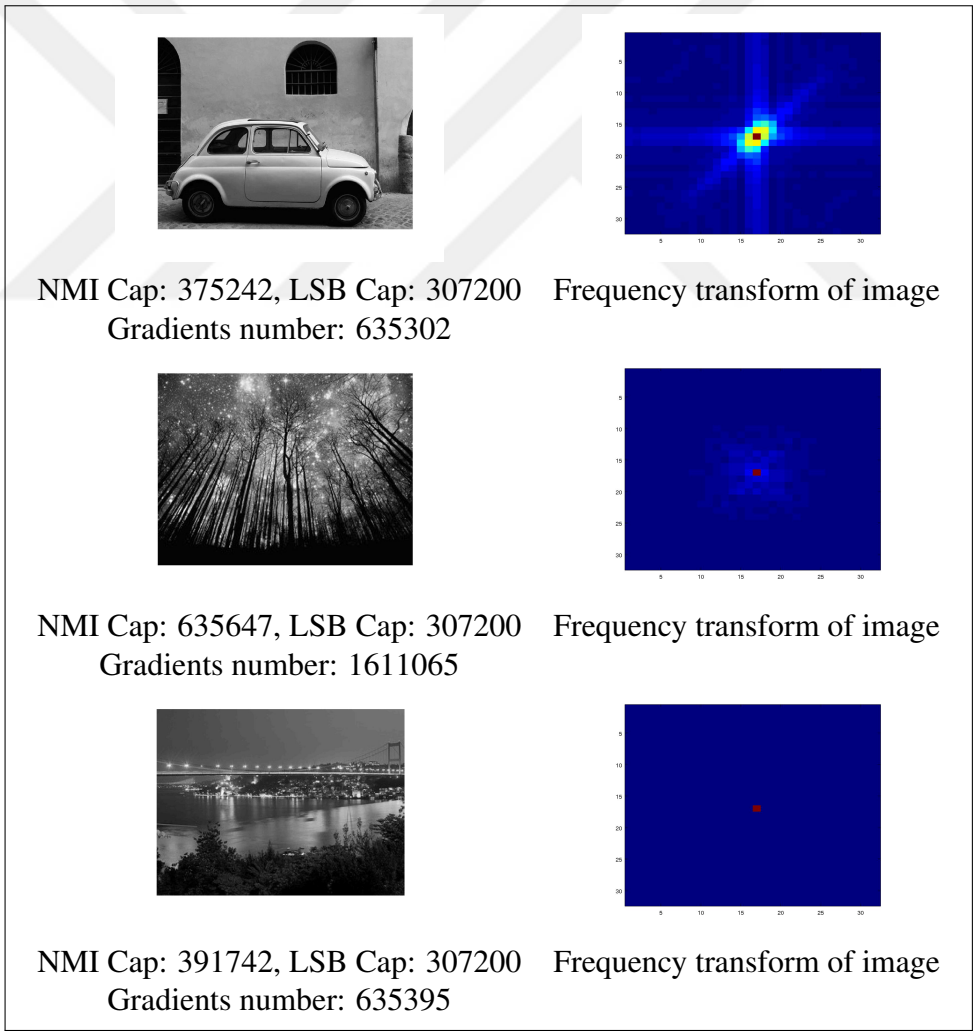


**Figure 3.7** : Embedding capacities of image according to their frequency characteristic.

If there are not many gradients in the cover image, data hiding process should not be applied on this cover image in order to keep hidden communication. Otherwise, third person can infer that there is a hidden communication or hiding process on the stego image as you can see from Figure 3.8. Gradients number is not checked in LSB technique. On the other hand, gradients number is significant for embedding process of NMI technique because if gradients number is not sufficient for hiding bits of plaintext, embedding proccess is not applied on this cover image as it is mentioned in next sections in Chapter 3. Therefore, NMI is selected for embedding method in this thesis implementation.

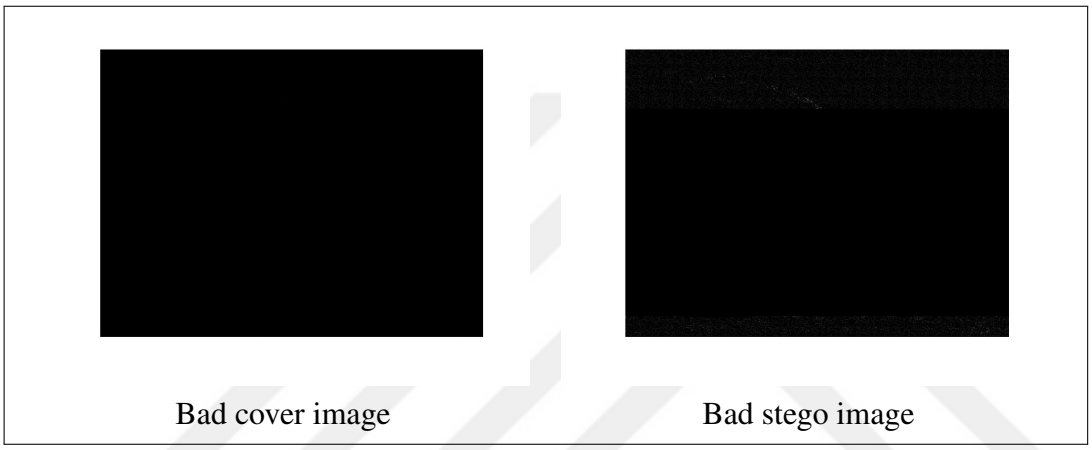


**Figure 3.8** : Bad covering in hiding process.

### 3.3 Procedure Of New Information Hiding Method

Some definetions such as pixel symmetry, symmetry key, pixel areas and image regions must be defined before explaining new embedding method.

#### 3.3.1 Definition of pixel area and image region

Image regions represent $2x2$ pixel block of image. The first image region is started with top-left corner of image and it is shifted right two pixels in each steps. If the region reaches at the end of the row, column number is added by 2 and the process continues same. Example image regions can be seen in Figure 3.9 [31]. Pixel areas are slots in image regions. There are three pixel areas in one image region. These areas are shown in Figure 3.10. These pixel areas are used for calculating pixel symmetry in proposed method.

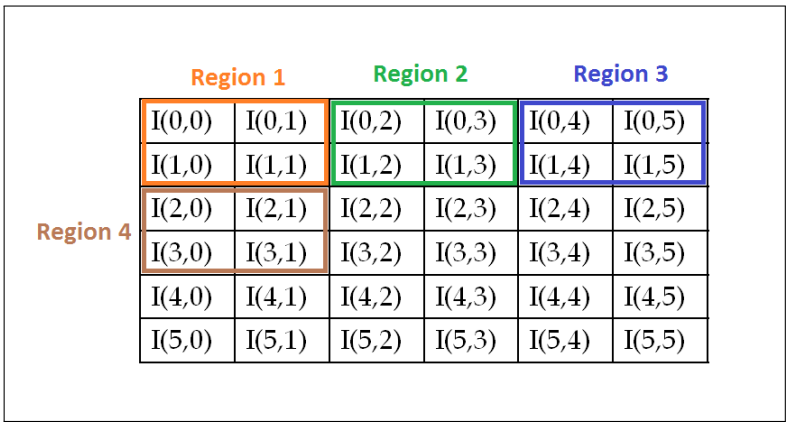


**Figure 3.9** : Image regions in frame [31].

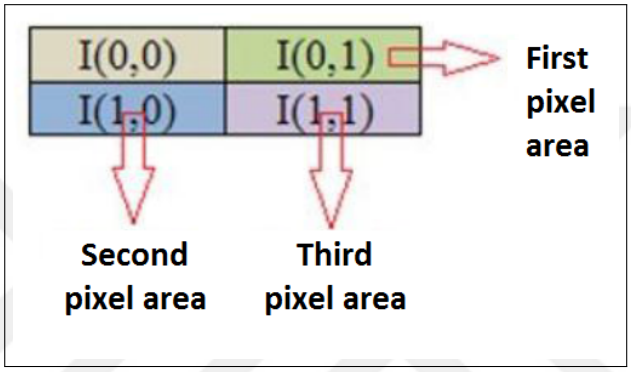


**Figure 3.10** : Pixel areas in image region.

### 3.3.2 Definition of symmetry, pixel symmetry and symmetry key

Symmetry can be found in various forms in electronic applications [32]. Such as, symmetry by point, symmetry by line and symmetry by rotation. Pixel symmetry is utilized similarly symmetry by point approach. In symmetry by point method; There are at least three points, these are named by;

1. Main point(A)
2. Symmetry point(B)
3. Origin point(O)

All points are located on the same line, symmetry point and main point have same distance from origin point but these points have different place on the line. Symmetry point is also called reflection of main point according to origin point. All points are shown in Figure 3.11.

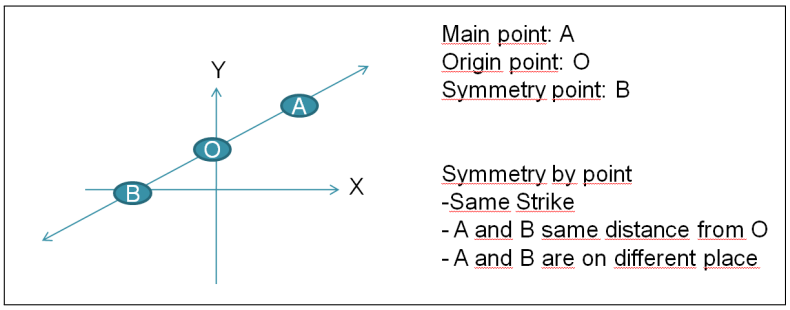


**Figure 3.11** : Symmetry description.

Let coordinates of A is $(a_x, a_y)$, coordinates of B is $(b_x, b_y)$, coordinates of O is $(o_x, o_y)$, image widht is W, image height is H, $d_1$ is distance between A and O on X coordinates and $d_2$ is distance between O and B. Equations 3.1, 3.2, 3.3, 3.4, 3.5 and 3.6 show measuring coordinates of B from coordinates of A and O.

$$d_1 = o_x - a_x \tag{3.1}$$

$$d_2 = d_1 \tag{3.2}$$

$$b_x = o_x + d_2 \tag{3.3}$$

$$b_x = (o_x + (o_x - a_x)) \mod H \tag{3.4}$$

$$b_x = (2 \times o_x - a_x) \mod H \tag{3.5}$$

Similar calculations are made for measuring $b_y$ as it is shown in equation 3.6.

$$b_y = (2 \times o_y - a_y) \mod W \tag{3.6}$$

While row and column numbers of pixels are supposed to be coordinates in a plane, symmetry coordinate of these pixels can be calculated according to coordinate of another pixel . This method is called pixel symmetry. Three origin pixels are utilized for proposed method in order to calculate all symmetry of pixel areas. These three origin coordinates represent symmetry key. As it is mentioned before, there are three pixel areas. Symmetry point of each pixel areas is calculated by their origin point or pixel. Overflow means that coordinates of symmetry point is less than 0 or more than image sizes. If there is an overflow, image size is added on result coordinates in case coordinate of symmetry point less than 0. Otherwise, image size is subtracted in case coordinate of symmetry point is more than image size. This operation is named modulo in mathematics. Thanks to this method, overlapping among symmetry points

can be prevented, which can be named by injective function in mathematics. Some pixel symmetry calculation examples are given in Figures 3.12 and 3.13.

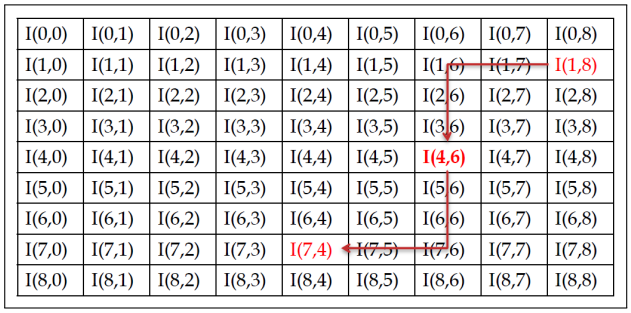


**Figure 3.12** : Calculating pixel symmetry of $I(1,8)$ according to $I(4,6)$.

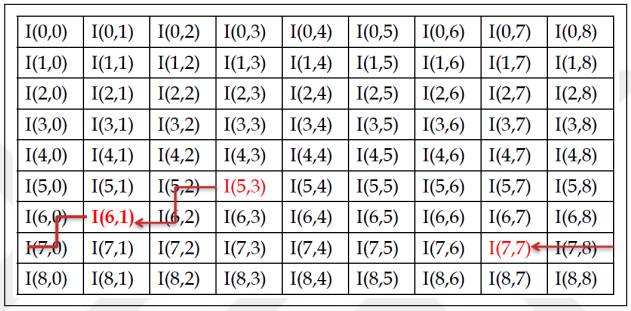


**Figure 3.13** : Calculating pixel symmetry of $I(5,3)$ according to $I(6,1)$.

### 3.3.3 New embedding method

Similar to information hiding with neighbor mean interpolation method, interpolation values and bit number added on interpolation values are firstly calculated as it is explained in detail in Chapter 2. Then, pixel symmetry of that pixel is calculated during the embedding process. Then, embedding process is applied on symmetry point. Example calculation process can be seen in Figure 3.14 [31].

In this example, symmetry key is $Y5 = (1,2)$, $Y7 = (2,1)$ and $Y7 = (2,1)$. It means that symmetry of first pixel area is calculated by $Y5$, symmetry of second pixel area is calculated by $Y7$ and symmetry of third pixel area is calculated by $Y7$. Result of calculating pixel symmetry can be easily seen in the top-middle frame. Then, embedding bit numbers according to neighbor pixels of interpolation values is measured. Then, pixel symmetry of $Y1$ is calculated in order to embed bit stream (binary message) inside cover image. the result of this process is $Y10$. Then move $Y10$ and get $n10$ amount of bit from starting point of bit stream. Then, convert this binary number to decimal number and add on $Y10$ value. Then, move to $Y2$ and apply same

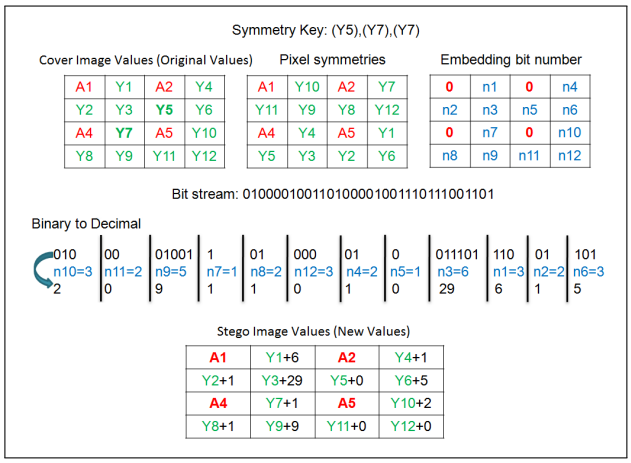


**Figure 3.14** : Example of embedding process in proposed method.

process for it. This path is continued towards this order; $Y1$, $Y2$, $Y3$,..., $Y10$, $Y11$ and $Y12$. However, thanks to proposed method, the embedding process is following in this order; $Y10$, $Y11$, $Y9$, $Y7$, $Y8$, $Y12$, $Y4$, $Y5$, $Y3$, $Y1$, $Y2$ and $Y6$. In fact, this order is produced differently for each symmetry keys.

Data hiding and extracting processes are shown in Figure 3.15.

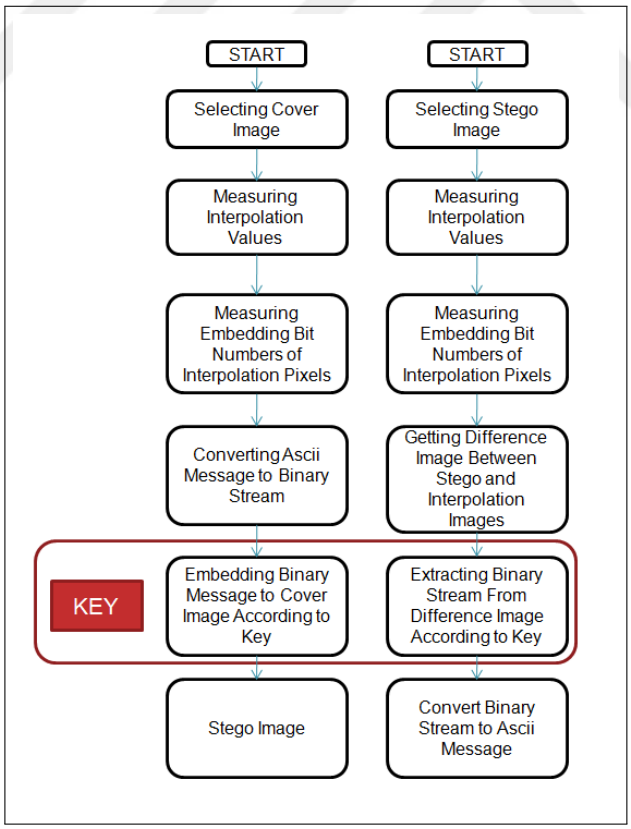


**Figure 3.15** : Flowchart of information hiding and extracting processes.

# 4. IMPLEMENTATION

In this section, all implementations about proposed method are mentioned. First implementation is applied by Matlab software on personal computer in order to show that proposed method works well. The second implementation is made on AVNET's ZedBoard in order to show that proposed method can be applied on embedded system.

## 4.1 Matlab Implementation

Matlab is suitable software for testing and applying new signal processing methods. It is widely used in signal processing applications. Therefore, Matlab is firstly used for proposed method works well. All application specific functions are written by us like measure_ symmetry_ point rather than used ready for use functions of Matlab. Ready for use functions of matlab only used for basic process like image read, write and show process.

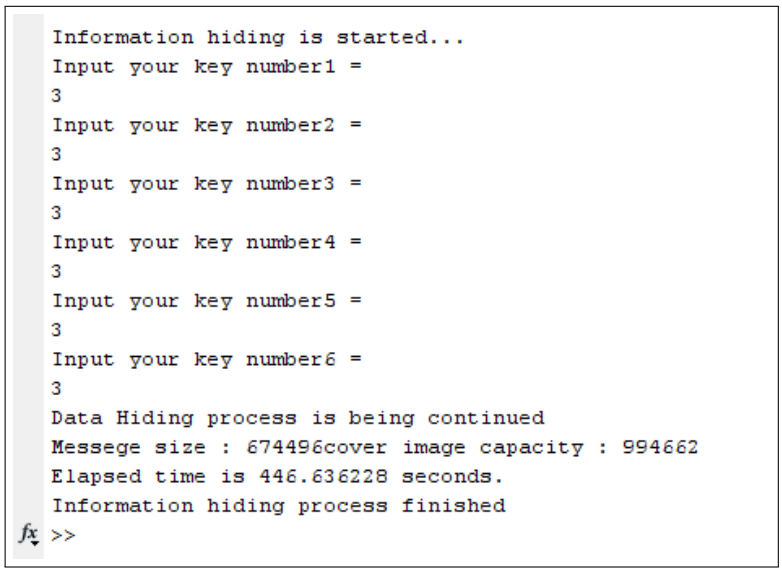


**Figure 4.1** : Matlab interface of information hiding application.

While Matlab program is running, symmetry key is firstly entered by user. User enters 6 numbers in this step. These numbers must be between 1 and image weight or height as it is shown in Figure 4.1. Then, information inside "plaintext.txt" file is hidden and

embedded inside "cover_ image.png". If user wants to change cover image, user has to change "cover_ image.png" file inside the application folder. If user wants to change message, user has to change "plaintext.txt" file inside the application folder.

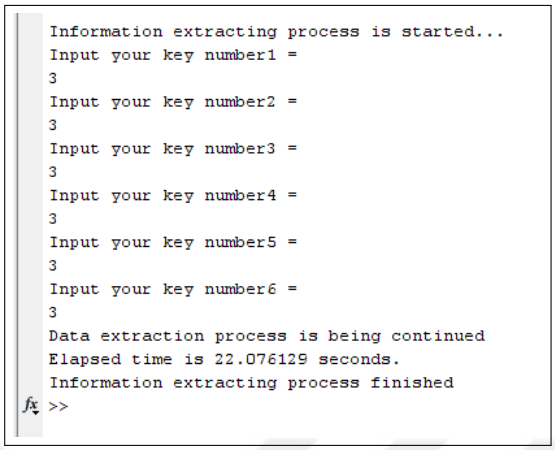


**Figure 4.2** : Matlab interface of information extracting application.

If user wants to extract stego image, user has to run extraction application with Matlab. Similar to hiding application, it first wants to enter 6 numbers of symmetry key as it is shown in Figure 4.2. Then, extracting process starts. At the end of extraction process, extracted message open with Matlab editor. User can also find extracted message in "deltaplaintext.txt" file inside the application folder. Both applications can run for RGB and gray scale images. All Matlab codes are given in appendix A.1.1 and A.1.3. Codes include hiding and extracting programs.

## 4.2 System On Chip Implementation

In order to prove that proposed method can be applied on embedded systems, AVNET's ZedBoard is selected because there is a Zynq processor based on it. Zynq based processor provides to create an infrastructure for image processing system on hybrid processor. In this section; Zynq processor architecture, ZED Board specifics, our embedded system design, software architecture of data hiding processes and drivers created for Linux Operating system is mentioned.

### 4.2.1 Field programmable gate arrays (FPGAs)

FPGAs are re-configurable devices which create hardware systems with the help of hardware description languages (HDLs) like Verilog HDL, VHDL or System C. Xilinx

and Altera are two prominent companies which produce FPGA chips. Their FPGA architectures are different but their devices can be programed by same HDL. However, if there is a platform specific hardware in your custom IP, you should modify your HDL design according to architecture of brand you use. On the other hand, there is an another median to create hardware for FPGAs by using higher level languages then HDLs like C or C++. High Level Synthesis (HLS) is used for this reason. Hardware systems can be programmed by using HLS programs like Xilinx's Vivado HLS, Intel HLS or Mathworks's Matlab.

According to FPGA Frontiers's recent white paper, New Applications in Reconfigurable Computing, Hybrid processor architecture, which means FPGA and processor system with together, will be very popular among computer architectures in the future [33]. The processors which have FPGA part can accelerate functions or programs by using hardware based application specific custom circuit inside their FPGA rather than running all process just on software based processor. As an existing example of hybrid computer architecture, Xilinx's Zynq has already been utilized in many applications. For instance, data science [34], speech [35] and image processing [36]. In fact, after one of the prominent processor manufacturer, Intel, shelling out $16.7 billion in December 2015 to buy Altera, they announced that they would start to produce Xeon computer processor units (CPUs) with FPGA accelerator within five years or early [33]. As a result, even desktop application developers should learn to program FPGAs as well as graphical processor units (GPUs)in order to accelerate their applications.

### 4.2.2 Linux operating system

Linux or GNU/Linux is explained by Wikipedia like that it is an Unix-like computer operating system assembled under the model of free and open-source software development and distribution. It was first released on 17 September 1991 by Linus Torvalds. Then it has been contributed by world wide community like GNU (GNU is not UNIX) project. GNU project was founded by Richard Stallman. The combination of GNU software and the Linux kernel is commonly known as Linux or less frequently GNU/Linux [37]. Linux was originally designed for personal computers based on Intel x86 architecture, but has since been ported to more platforms than any other operating

system [37].Nowadays, there are many applications which utilize Linux operating system on various platforms, especially Advance Risk Machine (ARM) based.

### 4.2.3 Zynq architecture [38]

The Zynq-7000 family is based on the Xilinx All Programmable SoC architecture. These products integrate a feature-rich dual-core or single-core ARM Cortex-A9 based processing system (PS) and 28 nm Xilinx programmable logic (PL) in a single device. The ARM Cortex-A9 CPUs are the heart of the PS and also include on-chip memory, external memory interfaces, and a rich set of peripheral connectivity interfaces.

#### 4.2.3.1 Processing system (PS)

ARM Cortex-A9 Based Application Processor Unit (APU):

- 2.5 DMIPS/MHz per CPU
- CPU frequency: Up to 1 GHz
- Coherent multiprocessor support
- ARMv7-A architecture
- TrustZone security
- Thumb-2 instruction set
- Jazelle RCT execution Environment Architecture
- NEON media-processing engine
- Single and double precision Vector Floating Point Unit (VFPU)
- CoreSight and Program Trace Macrocell (PTM)
- Timer and Interrupts
- Three watchdog timers
- One global timer
- Two triple-timer counters Caches
- 32 KB Level 1 4-way set-associative instruction and data caches (independent for each CPU)

- 512 KB 8-way set-associative Level 2 cache (shared between the CPUs)
- Byte-parity support

On-Chip Memory:

- On-chip boot ROM
- 256 KB on-chip RAM (OCM)
- Byte-parity support

External Memory Interfaces:

- Multiprotocol dynamic memory controller
- 16-bit or 32-bit interfaces to DDR3, DDR3L, DDR2, or LPDDR2 memories
- ECC support in 16-bit mode
- 1GB of address space using single rank of 8-, 16-, or 32-bit-wide memories
- Static memory interfaces
- 8-bit SRAM data bus with up to 64 MB support
- Parallel NOR flash support
- ONFI1.0 NAND flash support (1-bit ECC)
- 1-bit SPI, 2-bit SPI, 4-bit SPI (quad-SPI), or two quad-SPI (8-bit) serial NOR flash

8-Channel DMA Controller:

- Memory-to-memory, memory-to-peripheral, peripheral-to-memory and scatter-gather transaction support

I/O Peripherals and Interfaces:

- Two 10/100/1000 tri-speed Ethernet MAC peripherals with IEEE Std 802.3 and IEEE Std 1588 revision 2.0 support
- Scatter-gather DMA capability
- Recognition of 1588 rev. 2 PTP frames
- GMII, RGMII, and SGMII interfaces
- Two USB 2.0 OTG peripherals, each supporting up to 12 Endpoints

- USB 2.0 compliant device IP core
- Supports on-the-go, high-speed, full-speed, and low-speed modes
- Intel EHCI compliant USB host
- 8-bit ULPI external PHY interface
- Two full CAN 2.0B compliant CAN bus interfaces
- CAN 2.0-A and CAN 2.0-B and ISO 118981-1 standard compliant
- External PHY interface
- Two SD/SDIO 2.0/MMC3.31 compliant controllers
- Two full-duplex SPI ports with three peripheral chip selects
- Two high-speed UARTs (up to 1 Mb/s)
- Two master and slave I2C interfaces
- GPIO with four 32-bit banks, of which up to 54 bits can be used with the PS I/O (one bank of 32b and one bank of 22b) and up to 64 bits (up to two banks of 32b) connected to the Programmable Logic
- Up to 54 flexible multiplexed I/O (MIO) for peripheral pin assignments

Interconnect:

- High-bandwidth connectivity within PS and between PS and PL
- ARM AMBA AXI based
- QoS support on critical masters for latency and bandwidth control

#### 4.2.3.2 Programmable logic (PL)

Configurable Logic Blocks (CLB):

- Look-up tables (LUT)
- Flip-flops
- Cascadeable adders

Serial Transceivers:

- Up to 16 receivers and transmitters

- Supports up to 12.5 Gb/s data rates

JTAG Boundary-Scan:

- IEEE Std 1149.1 Compatible Test Interface

36 Kb Block RAM:

- True Dual-Port
- Up to 72 bits wide
- Configurable as dual 18 Kb block RAM

DSP Blocks:

- 18 x 25 signed multiply
- 48-bit adder/accumulator
- 25-bit pre-adder

Programmable I/O Blocks:

- Supports LVCMOS, LVDS, and SSTL
- 1.2V to 3.3V I/O
- Programmable I/O delay and SerDes

PCI Express Block:

- Supports Root complex and End Point configurations
- Supports up to Gen2 speeds
- Supports up to 8 lanes

Two 12-Bit Analog-to-Digital Converters:

- On-chip voltage and temperature sensing
- Up to 17 external differential input channels
- One million samples per second maximum conversion rate

Figure 4.3 shows the architecture of Zynq processing system.

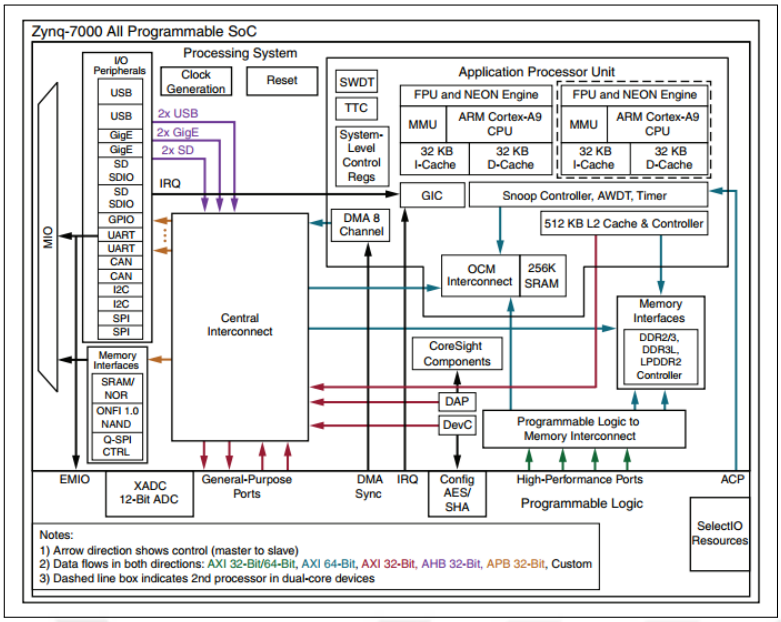


**Figure 4.3** : Zynq Architecture.

### 4.2.4 ZedBoard details [39]

The ZedBoard enables hardware and software developers to create or evaluate Zynq-7000 All Programmable SoC designs. The expandability features of this evaluation and development platform make it ideal for rapid prototyping and proof-of-concept development. The ZedBoard includes Xilinx XADC, FMC (FPGA Mezzanine Card), and Digilent Pmod compatible expansion headers as well as many common features used in system design. ZedBoard enables embedded computing capability by using DDR3 memory, Flash memory, gigabit Ethernet, general purpose I/O, and UART technologies. ZedBoard is shown in Figure 4.4.

Processor:

- Zynq-7000 AP SoC XC7Z020-CLG484-1

Memory:

- 512 MB DDR3
- 256 Mb Quad-SPI Flash
- 4 GB SD card

Communication:

- Onboard USB-JTAG Programming

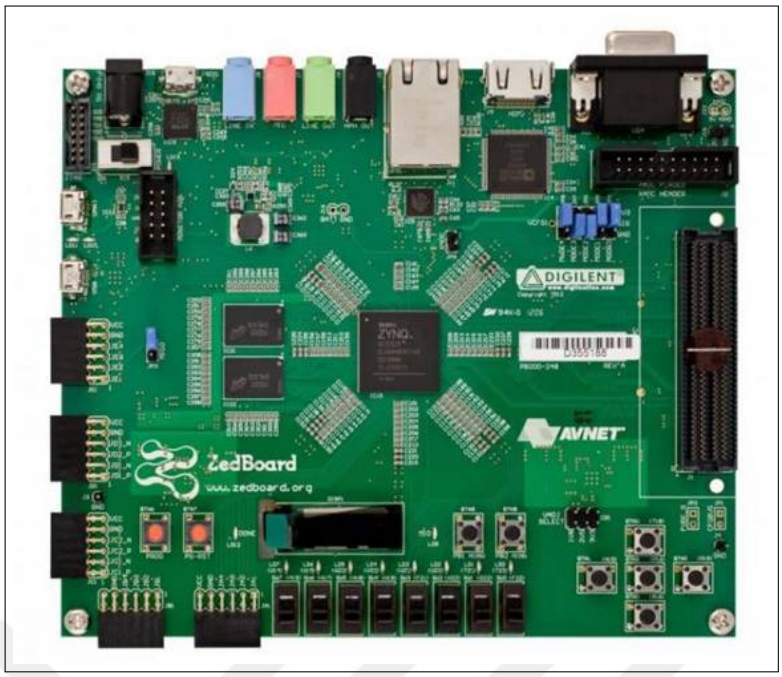


**Figure 4.4** : AVNET's ZedBoard.

- 10/100/1000 Ethernet
- USB OTG 2.0 and USB-UART

Expansion connectors:

- FMC-LPC connector (68 single-ended or 34 differential I/Os)
- 5 Pmod$^{TM}$ compatible headers (2x6)
- Agile Mixed Signaling (AMS) header

Clocking:

- 33.33333 MHz clock source for PS
- 100 MHz oscillator for PL

Display:

- HDMI output supporting 1080p60 with 16-bit, YCbCr, 4:2:2 mode color
- VGA output (12-bit resolution color)
- 128x32 OLED display

Configuration and Debug:

- Onboard USB-JTAG interface

- Xilinx Platform Cable JTAG connector

General Purpose I/O:

- 8 user LEDs
- 7 push buttons
- 8 DIP switches

The block diagram of Zedboard is shown in Figure 4.5.

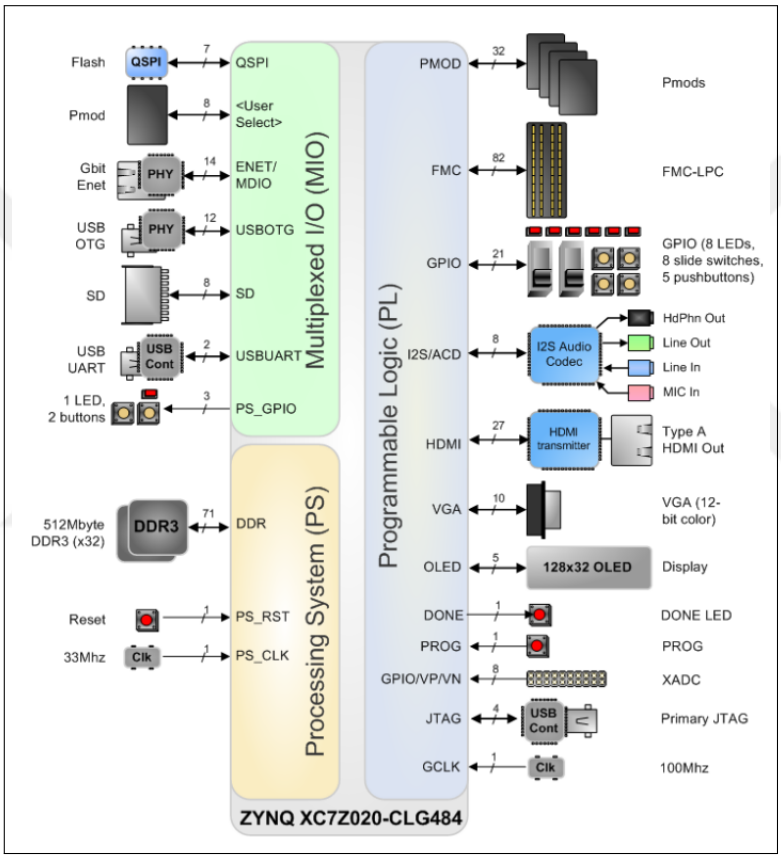


**Figure 4.5** : ZedBoard hardware block diagram.

### 4.2.5 Advanced extensible interface (AXI) Protocol

In computer architecture, peripherals around processor must communicate with each other or processor according to a bus protocol. Bus is a communication system which provide data transfer between peripherals or computers. AXI is a one of these bus protocols which is provided by Xilinx. Xilinx has adopted the AXI protocol for IP cores beginning with it's Spartan-6 and Virtex-6 FPGA devices [40]. AXI is part of ARM Advanced Microcontroller Bus Architecture (AMBA), a family of micro

controller buses first introduced in 1996 [40]. The first version of AXI was first included in AMBA 3.0, released in 2003. AMBA 4.0, released in 2010, includes the second version of AXI, AXI4. There are three types of AXI4 interfaces:

- AXI4—for high-performance memory-mapped requirements.
- AXI4-Lite—for simple, low-throughput memory-mapped communication (for example, to and from control and status registers).
- AXI4-Stream—for high-speed streaming data.

#### 4.2.5.1 AXI4

AXI4 is utilized for memory-mapped based IP cores. According to Xilinx document, AXI reference guide [40], AXI4 provides separate data and address connections for readings and writings, which allow simultaneous, bidirectional data transfer. AXI4 requires a single address and then bursts up to 256 words of data. The AXI4 protocol describes a variety of options that allow AXI4-compliant systems to achieve very high data throughput. Some of these features, in addition to bursting, are: data upsizing and downsizing, multiple outstanding addresses, and out-of-order transaction processing.

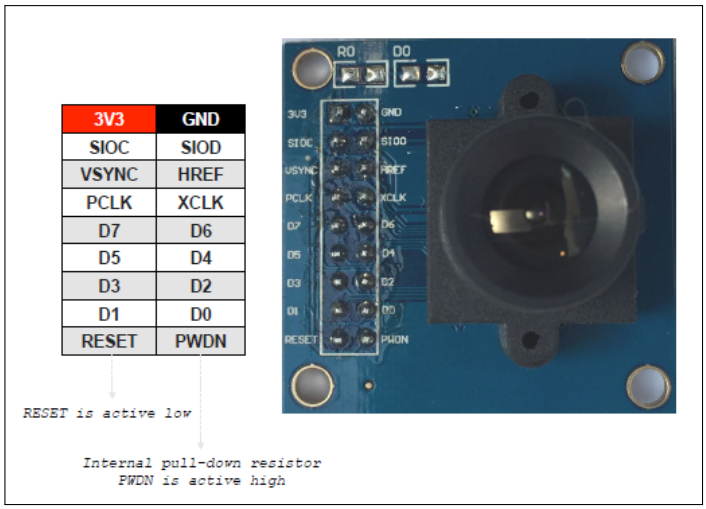


**Figure 4.6** : Pinout of OV7670.

#### 4.2.5.2 AXI4-Lite

AXI4-Lite is similar to AXI4 with some exceptions, the most notable of which is that bursting, is not supported [40]. It is suitable for controlling significant registers of IP cores.

#### 4.2.5.3 AXI4-Stream

The AXI4-Stream protocol defines a single channel for transmission of streaming data. The AXI4-Stream channel is modeled after the Write Data channel of the AXI4. Unlike AXI4, AXI4-Stream interfaces can burst an unlimited amount of data [40].

If you create IP core with AXI interface, you can use this IP core with all platforms which support AXI protocol. In Zynq architecture, communication between PS and PL is made by AXI protocols like Spartan-6 and Virtex-6 families. All kind of AXI protocols are used in our system according to needs and purpose of the IP cores.

### 4.2.6 OV7670 camera sensor [41]

The OV7670 camera chip image sensor is a low voltage CMOS device that provides the full functionality of a single-chip VGA camera and image processor in a small footprint package. The OV7670 provides full-frame, subsampled or windowed 8-bit images in a wide range of formats, controlled through the Serial Camera Control Bus (SCCB) interface. Pinout of OV7670 camera sensor is shown in Figure 4.6.

This camera module has an image array capable of operating at up to 30 frames per second (fps) in VGA with complete user control over image quality, formatting and output data transfer. All required image processing functions, including exposure control, gamma, white balance, color saturation, hue control and more, are also programmable through the SCCB interface. In addition, OmniVision sensors use proprietary sensor technology to improve image quality by reducing or eliminating common lighting/electrical source of image contamination, such as fixed pattern noise (FPN), smearing, blooming, etc., to produce a clean, fully stable color image. The camera sensor has these features;

- Pixel size : 3.6um x 3.6 um
- Resolution : 640x480 (VGA), 320x240 (QVGA), etc
- Frame rate : 30 fps
- Color format : RGB, YUV (4:2:2) and YCbCr (4:2:2)
- Scan mode : progressive
- Output : parallel (16 bit for YCbCr)

- Power supply : max 3.0 V
- Programming : I2C (7 bit address 0x21)

| Pin | Type | Description |
|---|---|---|
| VDD/3V3 | Supply | Power supply 3.3V |
| GND | Supply | Ground level |
| SIOC/SCL | Input | I2C clock |
| SIOD/SDA | Input/Output | I2C data |
| VSYNC | Output | Vertical synchronization |
| HREF | Output | Horizontal synchronization |
| PCLK | Output | Pixel clock |
| XCLK | Input | System clock |
| D0-D7 | Output | Video parallel output |
| RESET | Input | Reset (Active low) |
| PWDN | Input | Power down (Active high) |

**Figure 4.7** : Pin description of OV7670.

Pin description can be found in Figure 4.7. The OV7670 takes as input the XCLK clock signal minimum 10 Mhz and maximum 48 Mhz. The output pixel clock (PCLK) is programmable by SSCB interface (I2C). The synchronous (with PCLK) 8-bit data output D[7..0], VSYNC and HREF signal encode the image content. All video signals are shown in Figure 4.8. OV7670 camera sensor is used with YCbCr output format.

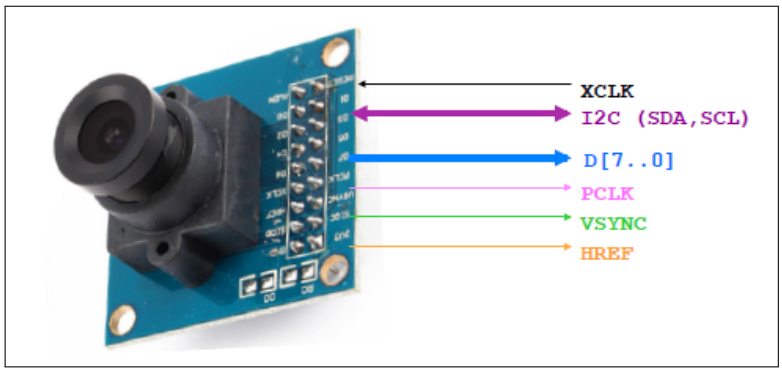


**Figure 4.8** : Control pins of OV7670.

### 4.2.7 YUV/YCbCr (4:2:2) and RGB color space

YUV/YCbCr (4:2:2) and RGB are color space in computer science. These color space are used for represent digital image on computers. YUV/YCbCr (4:2:2) enables a more compact encoding with respect to RGB color space because one pixel is represented with 16-bit in YUV/YCbCr (4:2:2) color space while it is represented with 24-bit in

VGA color space. Grayscale component is Luma (Y) in YUV/YCbCr (4:2:2) color space. Color channels are subsampled. Resolution of CbCr and UV components are halved (W/2xH). However, all color channels - red, green, blue, - have same size in RGB color space. Difference between these color spaces are shown in Figure 4.9.

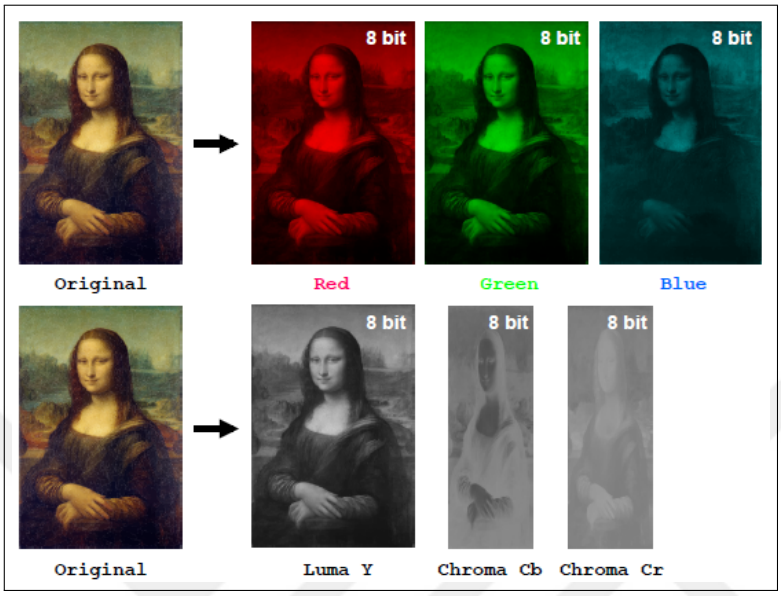


**Figure 4.9** : RGB color space vs YUV/YCbCr (4:2:2) color space.

### 4.2.8 Embedded system design

In this section, our image processing infrastructure created on zynq based hybrid processing system is mentioned. The aim of this thesis is also creating image processing infrastructure on embedded system and using Linux OS on it because with the help of Linux libraries, image processing applications can be applied more quickly than bare-metal applications.

Bare-metal application, or it is also called by standalone, means creating embedded system without operating system. At the beginning step of the creation software system, the system which runs with operating system can be hard to implement because installation of operating system development environment is harder than installation of bare-metal development environment in FPGA design flow. However, after understand installition of development environment, complex applications can be more easily and quickly created than bare-metal approach. Creation of our hardware design is introduced by improvement of hardware design of Computer Vision Laboratory in Bologna University. The laboratory people created a hardware design for their FPGA

applications about image processing. All process are implemented inside FPGA. This hardware design includes three main process, which are acquisition of image frame from camera sensor to memory, applying some computer vision (CV) filters on image frame and showing results with VGA output. This system was working with bare-metal applications. Therefore, first improvement was porting this system to Linux operating system. In order to run this system with Linux operating system, Linux development environment is installed some problems about Ip cores are fixed, device drivers for all IP cores and user space applications are coded. Before explain all of these steps, our hardware design must be explained in details.

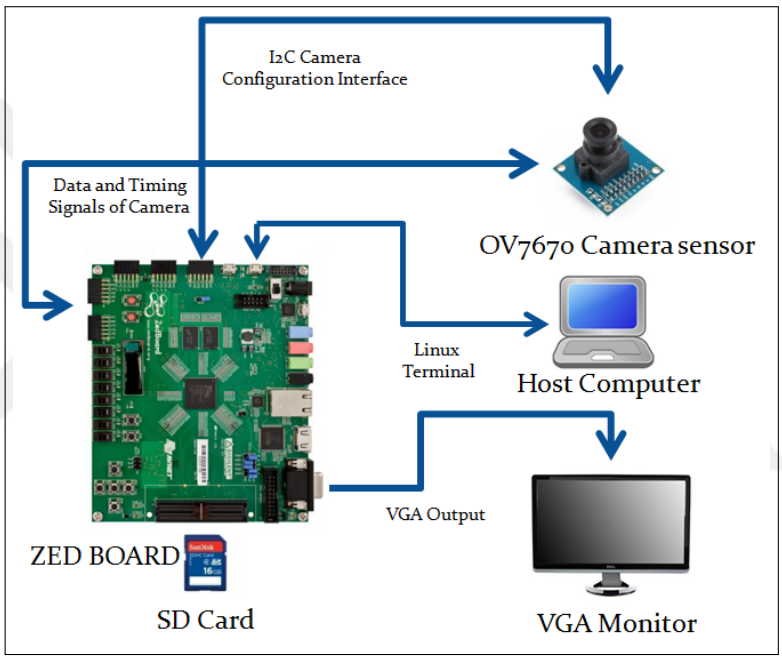


**Figure 4.10** : Hardware componenets of our embedded system.

Our hardware has some external device like camera and monitor. This device is controlled by peripherals of processor and custom IP cores inside PL. Hardware components can be seen in Figure 4.10. Quality of OV7670 camera is controlled by I2C peripheral of PS. Image frames are captured from OV7670 camera sensor with PMOD ports of ZedBoard. Image processing results are showed with VGA output of Zedboard. On the other hand, ZedBoard is connected to host computer via micro-USB port of ZedBoard in order to establish terminal connection of Linux operating system.

As it is mentioned before, hardware system mainly has three significant parts, which are acquisition of image frame, apply some computer vision filters and showing results with VGA output. You can see block diagram of hardware system in Figure 4.11.

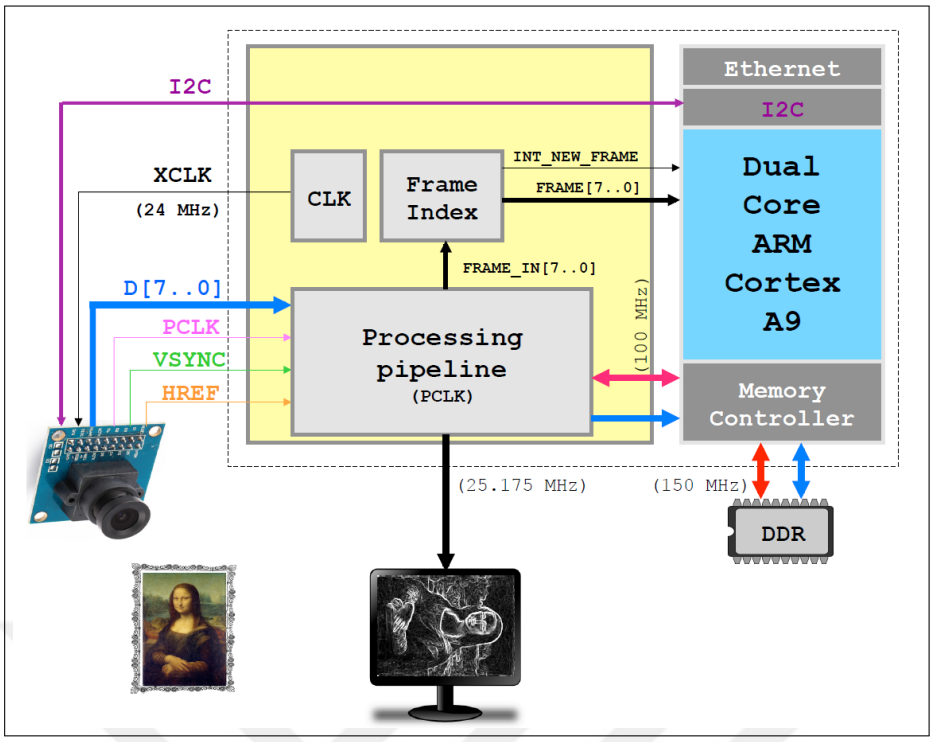


**Figure 4.11** : Block diagram of hardware system.

#### 4.2.8.1 Processing pipeline

Processing pipeline is constructed by three parts. These parts are front-end, computer vision algorithms and memory manager. OV7670 output protocol is converted into AXI-Stream protocol at Front-end part. Then, computer vision algorithms are applied on AXI-Stream data stream at computer vision algorithm part. Lastly, memory manager part controls all process about memory management. All parts are shown in detail in Figures 4.12, 4.13 and 4.14.

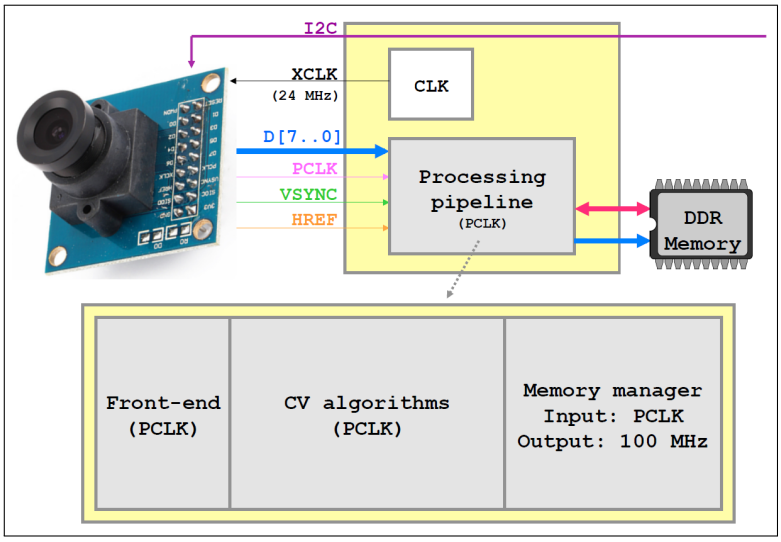


**Figure 4.12** : Processing pipeline.

There are three steps in front-end part. These are converting camera sensor's signals which are HREF, VSYNC AND PCLK to frame_ valid (FV) and line_ valid (LV) in OV7670 interface box. Then, Data (D)[7..0] is converted to AXI-Stream in OV7670 LF_ VALID to_ AXIS box. After signal is converted AXI-Stream, grayscale image frame is created by pitching only one out of two bytes of AXI-Stream data. Then, it is applied on input of computer vision filter by using again AXI-Stream.

There are several convolutions based on filters in computer vision part. These filters can find horizontal and vertical edges on an image frame according to position of switches on ZedBoard. Nevertheless, these filters are not be used because grayscale image is needed to apply data hiding process. Therefore, we are keeping switches position at zero in order to write grayscale image to memory and this grayscale image is directly written to memory.

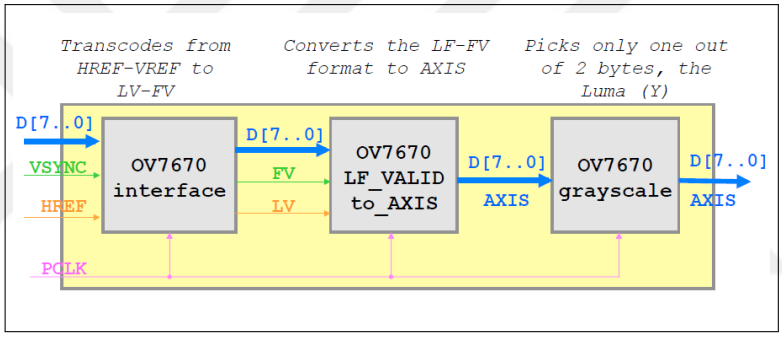


**Figure 4.13** : Front-end in detail.

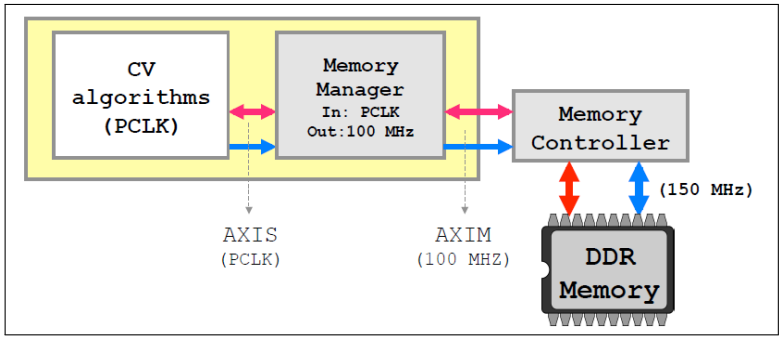


**Figure 4.14** : Memory manager in detail.

Input of memory manager part is connected to upstream of computer vision pipeline. Output of memory manager part is connected to memory controller (HP0) of the Zynq/PS with 64 bit AXI4 interface. These part writes data on frame buffer inside DDR memory. Memory addresses (frame buffer) can be configured by ARM processor via AXI4-Lite protocol.

The Zynq processor has a 4 GB address space. The Zed Board contains 2x256 MB DDR3 memory devices. The overall DDR memory available (ZedBoard) is 512 MB. The design includes a frame buffer in DDR memory. The number of frame is configurable via AXI4-Lite protocol. Maximum 256 frames can be arranged however typically 8 frames is utilized in our application. Images are stored in consecutive locations starting from the base of the frame buffer. A valid address for the frame buffer is 0x10000000 as you can see from Figure 4.15. Image size is configurable. The CPU efficiently keep the the last images stored in the frame buffer by means of an

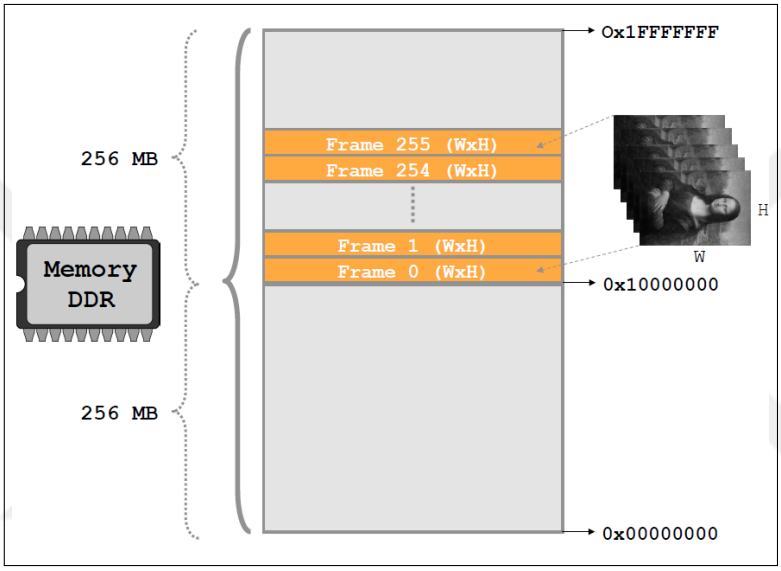


**Figure 4.15** : Memory map of our hardware design.

interrupt-driven approach. Therefore, Linux driver for memory manager must handle interrupt. Thus, there is an another module, which is shown in Figure 4.16. The name

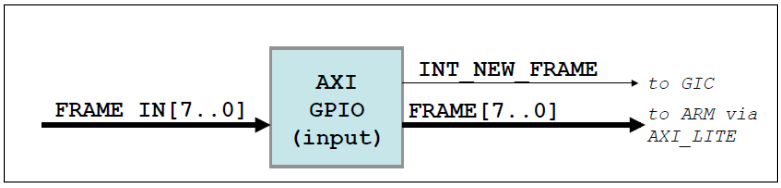


**Figure 4.16** : Frame indexer in detail.

of this module is frame indexer. This module contains the index of the last frame completely written in memory. Once an image is completely written in memory, the frame idexer is filled, by the memory writer, with the index of the frame buffer element. This operation triggers an interrupt for the ARM processor. The interrupt handler reads via AXI4-Lite the index and performs appropriate operations accordingly. The Frame

Index is mapped as peripheral in the address space and based on a standard AXI4_ GPIO module. The frame buffer is also a crucial for displaying images on a standard monitor by means of the standard VGA interface.

### 4.2.9 Software architecture

Linux operating system is used for controlling hardware. Adding costum hardware on system and configuring or utilizing it with Linux operating system, you should create Linux device driver and user application respectively. General methodology of creating new hardware application with Linux operating system is shown in Figure 4.17.

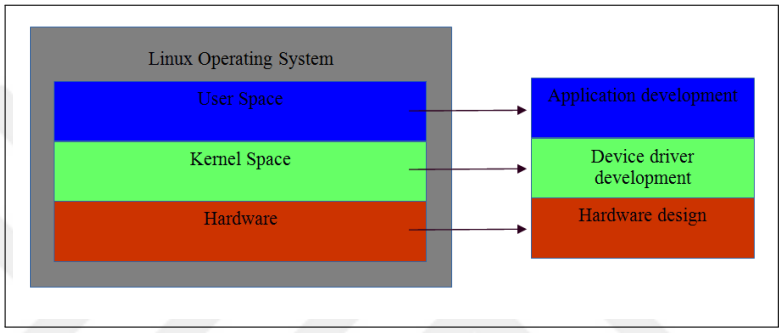


**Figure 4.17** : General methodology of creating new hardware application with Linux.

In order to create Linux device driver, new loadable kernel module is created because utilizing loadable kernel module is more efficient way for embedded systems. As you know that embedded systems have limited resources. Therefore, in order to decrease booting time and provide more efficient memory usage, loadable kernel module can be inserted and removed according to the needs. All code of linux device drivers which are created by us can be reached from Appendix A.2.1.

After device driver is ready for use, Linux user space application must be created for developing software system. Device driver can be controlled from user space application in order to configure hardware. In fact, other device drivers provided by Linux kernel like ethernet, usb or i2c drivers can be utilized easily. All codes of user space applications are shown in Appendix A.2.1, A.2.2, A.2.3 and A.2.4.

# 5. EXPERIMENTS AND RESULTS

## 5.1 Matlab Experiments And Results

Application is accomplished by four example image which are oktay, cameraman, sezen and lena. These image sizes are defined 256$x$256 pixels. Subsampling is not applied therefore stego image size become 512$x$512 in the end of hiding process. In


**Figure 5.1** : Cover images

order to measure PSNR, cover image and stego image must be in same size. Thus, cover images which have 256$x$256 pixels are enlarged to 512$x$512 by "imresize" function of Matlab. Bicubic interpolation is utilized by default for this process. Then PSNR values between stego image and cover image can be measured. PSNR results are obtained generally higher than 35$dB$. Cover images are shown in Figure 5.1, stego images are shown in Figure 5.2 and the PSNR results are given in Table 5.1.

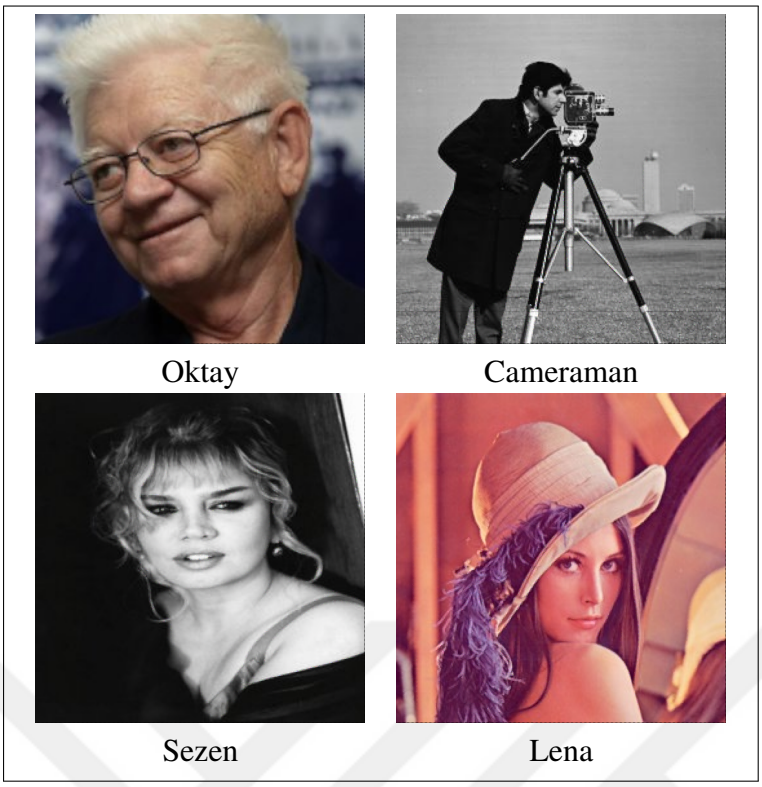


**Figure 5.2** : Stego images

**Table 5.1** : PSNR results between stego and cover images

| Image | Oktay | Cameraman | Sezen | Lena |
|---|---|---|---|---|
| Size (Pixel) | 256*x*256 | 256*x*256 | 256*x*256 | 256*x*256 |
| Capacity (Bit) | 766299 | 994662 | 819145 | 952652 |
| Added Bit Number (Bit) | 674496 | 674496 | 674496 | 674496 |
| PSNR (*dB*) | 45.7416 | 40.3941 | 43.9661 | 41.4403 |

## 5.2 System On Chip Experiments And Results

The system on chip application is running on ZedBoard. Firstly, device drivers created for custom hardwares are inserted after booting of Linux on ZedBoard. Then, camera sensor is configured by user space application which uses i2c module of PS. Then, data hiding process is run by user space application in order to hide text file inside image was written on DDR memory. Thus, "plaintext.txt" file is read from SD card. Then, user space application takes a photo according to last image indexer which is updated by each interrupt of frame_ indexer. In order to get symmetry key from user, Linux terminal is utilized. Then, all data hiding process is applied on processor part of Zynq. After data hiding process is finished, user space application saves stego image to SD

card. Finally, message can be extracted from stego image in host computer by reading from SD card. SD card files are shown in Figure 5.3.

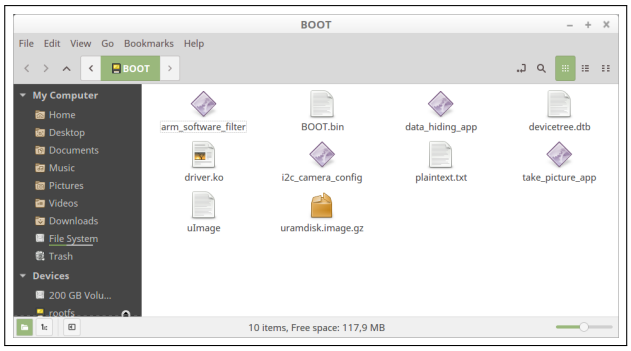


**Figure 5.3** : Sd card files for booting Linux on ZED Board.

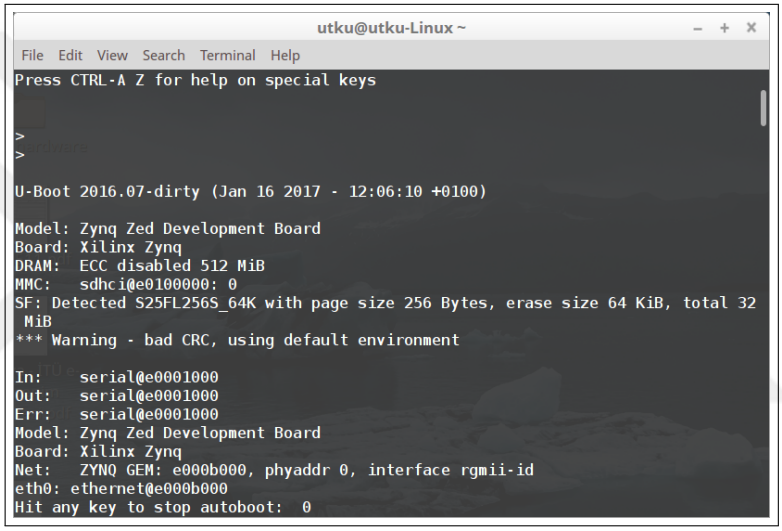


**Figure 5.4** : U-boot interface in terminal.

Text file can be hidden inside a camera frame in this example. Our system works properly because hidden data from stego image produced by our system on chip architecture can be extracted on host computer program properly. Some results of system on chip application can be viewed in Figures 5.4, 5.5, 5.6, 5.9, 5.10, 5.11 and 5.12. However, there are some errors about synchronization sometimes about image acquisitioning system. This situation happens because of reset control of front-end part in hardware system. This problem can be solved by fixing reset status of front-end part in hardware system. Figures 5.14, 5.15, 5.16 and 5.17 show consumption of resource in FPGA. As you can understand from these figures,there are many resource can be utilized for accelerating data hiding application by hardware accelerator created inside FPGA. Therefore, data hididng process can be applied inside FPGA in the future.

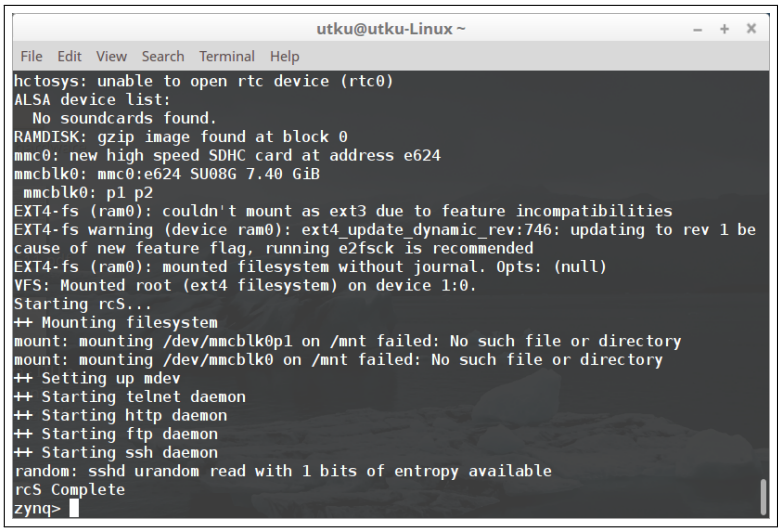


**Figure 5.5** : After boot process finished, we can type command on terminal.

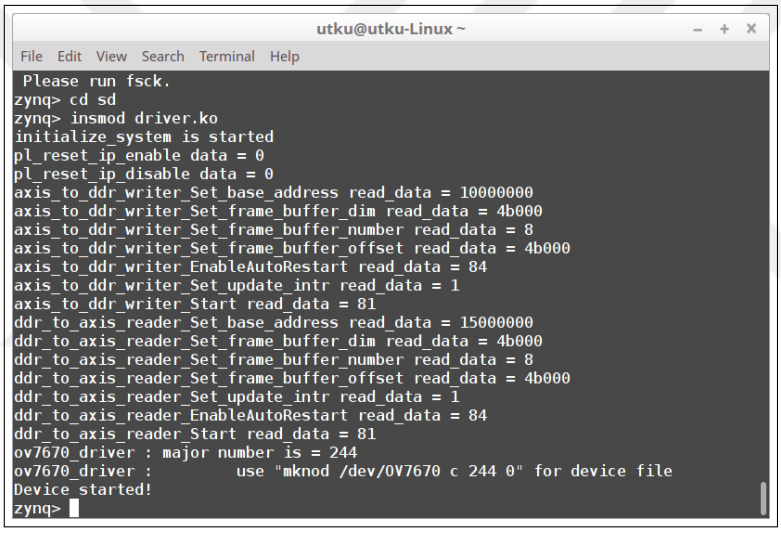


**Figure 5.6** : Insert driver module step.

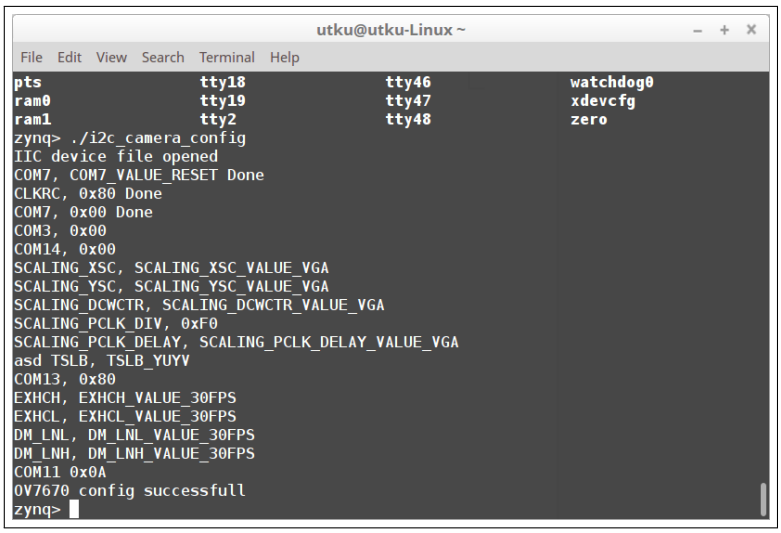


**Figure 5.7** : Camera configuration with i2c.

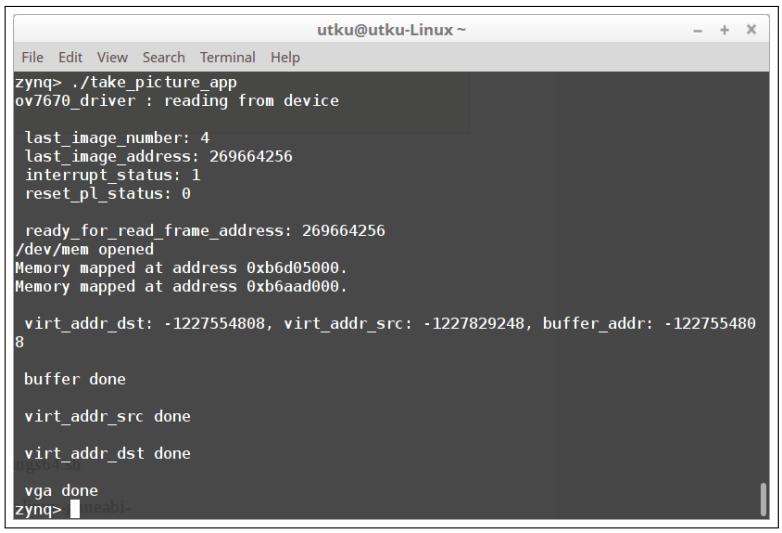


**Figure 5.8** : Take picture application results.

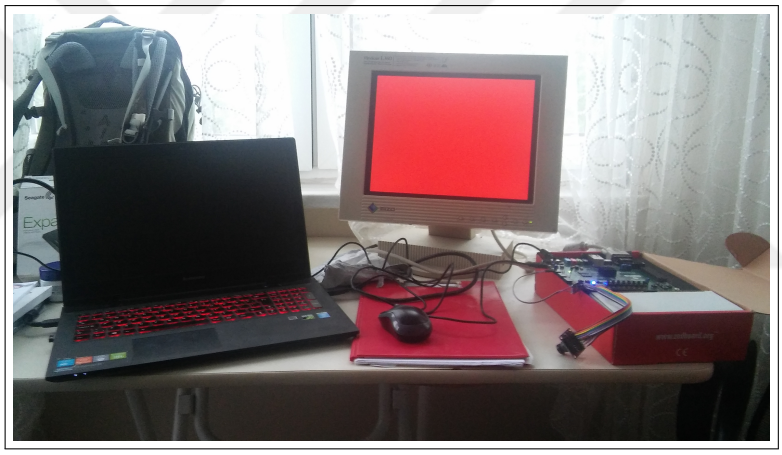

**Figure 5.9** : SOC system setup at the beginning.


**Figure 5.10** : SOC system setup after information hiding application.


**Figure 5.11** : SOC system setup without synchronization error.

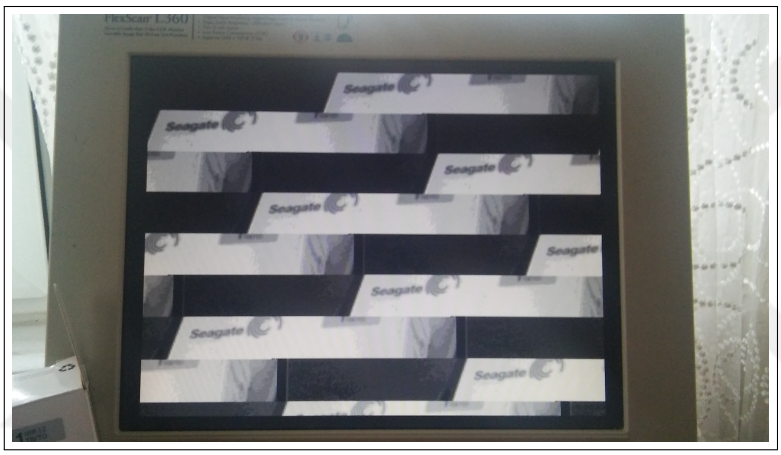


**Figure 5.12** : SOC system setup with synchronization error.

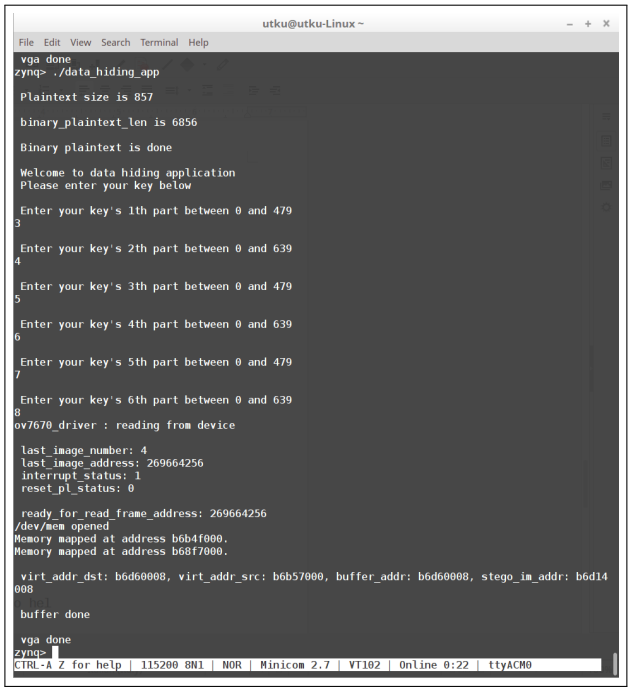


**Figure 5.13** : Information hiding application results.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 3819 | 0 | 53200 | 7.18 |
| LUT as Logic | 3421 | 0 | 53200 | 6.43 |
| LUT as Memory | 398 | 0 | 17400 | 2.29 |
| LUT as Distributed RAM | 20 | 0 | | |
| LUT as Shift Register | 378 | 0 | | |
| Slice Registers | 5093 | 0 | 106400 | 4.79 |
| Register as Flip Flop | 5093 | 0 | 106400 | 4.79 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 10 | 0 | 26600 | 0.04 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

**Figure 5.14** : Resource consumption of our FPGA implementation-1.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Block RAM Tile | 4 | 0 | 140 | 2.86 |
| RAMB36/FIFO* | 2 | 0 | 140 | 1.43 |
| RAMB36E1 only | 2 | | | |
| RAMB18 | 4 | 0 | 280 | 1.43 |
| RAMB18E1 only | 4 | | | |

**Figure 5.15** : Resource consumption of our FPGA implementation-2.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| DSPs | 13 | 0 | 220 | 5.91 |
| DSP48E1 only | 13 | | | |

**Figure 5.16** : Resource consumption of our FPGA implementation-3.

| Ref Name | Used | Functional Category |
|---|---|---|
| FDRE | 4514 | Flop & Latch |
| LUT3 | 1184 | LUT |
| LUT6 | 820 | LUT |
| LUT5 | 713 | LUT |
| LUT2 | 643 | LUT |
| LUT4 | 555 | LUT |
| LUT1 | 361 | LUT |
| FDCE | 342 | Flop & Latch |
| SRL16E | 330 | Distributed Memory |
| CARRY4 | 321 | CarryLogic |
| FDPE | 148 | Flop & Latch |
| BIBUF | 130 | IO |
| FDSE | 89 | Flop & Latch |
| SRLC32E | 48 | Distributed Memory |
| RAMD32 | 28 | Distributed Memory |
| OBUF | 17 | IO |
| IBUF | 16 | IO |
| DSP48E1 | 13 | Block Arithmetic |
| MUXF7 | 10 | MuxFx |
| RAMS32 | 8 | Distributed Memory |
| BUFG | 5 | Clock |
| RAMB18E1 | 4 | Block Memory |
| RAMB36E1 | 2 | Block Memory |
| PS7 | 1 | Specialized Resource |
| MMCME2_ADV | 1 | Clock |

**Figure 5.17** : Resource consumption of our FPGA implementation-4.

# 6. CONCLUSIONS

In this thesis, information hiding inside a digital image process is made by neighbor mean interpolation method, which is the better than other interpolation methods for steganography applications. Then, in addition to this method, five tubles of cryptography is added by designing pixel symmetry algorithm and using it in process which is adding message's bit on interpolation pixels. As a result of experiments, the proposed embedding method holds the PSNR value above as $35dB$ similar to main reference of this thesis [15]. Therefore, The most significant property of steganography, a large amount of secret data can be embedded while keeping a very high visual quality, it is achieved as well as getting same speed and computation like previous studies.

Number of key bits are $6 * 10 = 60$-bit in this application. However, this number can be increased by using pixel symmetry process recursively. it means that another symmetry measurement for first symmetry measurement result can be applied. On the other hand, proposed method is implemented on embedded system. Information hiding method worked properly but video acquisition part of hybrid system implementation sometimes lost its synchronization. Therefore, this problem can be fixed in the future. In addition, there are many resource which can be used for creating custom IPs in order to accelerate some process inside FPGA part of Zynq because the implementation don't consume all resource of PL as you can see in Figures 5.14, 5.15, 5.16 and 5.17.

## REFERENCES


[1] **Rivest, R.L., Shamir, A. and Adleman, L.M.**, (1978). A Method for Obtaining Digital Signatures And Public Key Cryptosystems, *Communications of The ACM*, *21*(2),pp. 120 – 126.

[2] **Zhu, D.**, (2002). Security Control in Inter-Bank Fund Transfer, *Journal of Electronic Commerce Research*, Vol. 3, No: 1, pp. 15 – 22.

[3] **U. S. Army Department of Defense**, (2013). Electronic Security Systems, *Unified Facilities Criteria*, 4-021-02.

[4] **Url-1**, `https://play.google.com/store/apps/details?id=org.thoughtcrime.securesms&hl=EN`, visiting date: (25.04.2017).

[5] **Chen, T.H, Chen, Y.C., Shih, W.K. and Wei, H.W.**, July (2011). An Efficient Anonymous Authentication Protocol for Mobile Pay-TV, *Journal of Network and Computer Applications, Elsevier*, Volume 34, Issue 4, pp. 1131 .. 1137.

[6] **Daemen, J. and Rijmen, V.**, September 3, (1999). AES Proposal: Rijndael, *AES Algorithm Submission*.

[7] **Gilbert, H. and Peyrin, T.**, (2010). Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations, *FSE 2010*, LNCS 6147, pp. 365 – 383.

[8] **Url-2**, `https://en.wikipedia.org/wiki/Cryptography`, visiting date: (11.02.2017).

[9] **Mathura, N. and Bansode, R.**, (2016). AES Based Text Encryption Using 12 Rounds with Dynamic Key Selection, *7th International Conference on Communication*, Computing and Virtualization.

[10] **Farsana, F. J. and Gopakumar, K.**, September (2016). A Novel Approach for Speech Encryption: Zaslavsky Map as Pseudo Random Number Generator, *6th International Conference on Advances In Computing & Communications, ICACC 2016*, pp. 6 – 8, Cochin, India.

[11] **Pak, C. and Huang, L.**, (2017). A New Color Image Encryption Using Combination of The 1D Chaotic Map, *Elsevier, Signal Processing*, 138, pp. 129 – 137.

[12] **Xiao, C., Wang, L., Zhu, M. and Wang, W.**, (2016). A Resource-Efficient Multimedia Encryption Scheme for Embedded Video Sensing System Based on Unmanned Aircraft, *Elsevier, Journal of Network and Computer Applications*, 59, pp. 117 – 125.

[13] **Stinson, D.R.**, (2006). Cryptography Theory And Practice, *Discrete Mathematics and Its Applications*, pp. 1 – 25.

[14] **Url-3**, `https://en.wikipedia.org/wiki/Steganography`, visiting date: (12.03.2017).

[15] **Ki-Hyun Jung, Kee-Young Yoo**, (2009). Data Hiding Method Using Image Interpolation, *Computer Standarts & Interfaces 31*, pp. 465 – 470.

[16] **Mishra, M., Mishra, P. and Adhikary, M.C.**, (2012). Digital Image Data Hiding Techniques: A Comparative Study, *ISSN-0974-715X, ANSVESA*, 7(2), pp. 105 – 115.

[17] **Cheddad, A., Condell, J., Curran, K. and Kevitt, P.**, March (2010). Digital Image Steganography: Survey And Analyses of Current Methods, *Signal Processing, Volume 90, Issue 3*, pp. 727 – 752.

[18] **Url-4**, `https://en.wikipedia.org/wiki/Digital_watermarkin`, visiting date: (12.03.2017).

[19] **Joshi, S.V., Bokil, A., Jain, N.A. and Koshti, D.**, September (2012). Image Steganography Combination of Spatial and Frequency Domain, *International Journal of Computer Applications(0975-8887)*, Volume 53, No.5.

[20] **Manoharan, S.J.**, July (2013). An Efficient Reversible Data Embedding Approach in Medical Images for Health Care Management, *http://shodhganga.inflibnet.ac.in:8080/jspui/handle/10603/10113*, Chapter 3, pp 30 – 36.

[21] **Johnson, N.F. and Jajodia, S.**, (1998). Exploring Steganography: Seeing The Unseen, *Computer Practices*, pp. 26 – 34.

[22] **Gunturk, B.K., Glotzbach, J., Altunbasak, J., Schafer, R.W. and Mersereau, R.M.**, January (2005). Demosaicking: Color Filter Array Interpolation, *IEEE Signal Processing Magazine*, pp. 44 – 54.

[23] **Unser, M., Thĕvenaz, P. and Yaroslavsky, L.**, October (1995). Convolution-Based Interpoaltion for Fast, High-Quality Rotation of Images, *IEEE Transactions on Image Processing*, Vol. 4, N0: 10.

[24] **Olivier, R. and Hanqianq, C.**, (2012). Nearest Neighbor Value Interpolation, *International Journal of Advaned Computer Science And Applications(IJACSA)*, Vol. 3, No: 4.

[25] **Lehmann, T.M., Gonner, C. and Spitzer, K.**, (1999). Survey:Interpolation Methods in Medical Image Processing, *IEEE Transactions on Medical Imaging, 18*(11), pp. 1049 – 1075.

[26] **Fahmy, S.A.**, (2008). Generalised Parallel Bilinear Interpolation Architecture for Vision Systems, *International Conference on Reconfugurable Computing and FPGAs*, pp. 331 – 336.

[27] **Li, M., Liang, T. and He, Y.**, (2013). Arnold Transform Based Image Scrambling Method, *3rd International Conference on Multimedia Technology(ICMT 2013)*, pp. 1309 – 1316.

[28] **Modak, P.M. and Pawar, V.**, (2015). A Comprehensive Survey on Image Scrambling Techniques, *International Journal of Science and Research (IJSR)*, pp. 814 – 818.

[29] **Ramalingam, B., Amirtharajan, R. and Rayappan, J.B.B.**, (2016). Multiplexed Stego Path on Reconfigurable Hardware: A Novel Random Approach, *Elsevier, Computers and Electrical Engineering, 55*, pp. 153 – 163.

[30] **Url-5**, `https://en.wikipedia.org/wiki/Image_gradient`, visiting date: (22.05.2018).

[31] **Esen, U., Örs Yalçın, S.B.**, (2015). Data Hiding Method Using Image Interpolation And Pixel Symmetry, *IEEE 9th International Conference on Electrical and Electronics Engineering (ELECO)*, pp. 776 – 779.

[32] **Reddy, H.C., Khoo, I.H. and Rajan, P.K.**, Third Quarter (2003). 2-D Symmetry: Theory And Filter Design Applications, *IEEE Circuits And Systems Magazine*.

[33] **FPGA Frontiers**, (2017).New Applications in Reconfigurable Computing.

[34] **Sklyarov, V., Skliarova, I., Rjabov, A. and Sudnitson, A.**, (2015). Zynq-based System for Extracting Sorted Subsets from Large Data Sets, *Journal of Microelectronics, Electronic Components and Materials*, pp 142 – 152, Vol. 45, No. 2.

[35] **Lita, A.I., Ionescu, L.M., Mazare, A.G., Serban, G. and Lita, I.**, (2016). Real Time System for Instrumental Sound Extraction And Recognition, *39th International Spring Seminar on Electronics Technology (ISSE)*.

[36] **Padmanabha, M., Schott, C., Robler, M., Kriesten, D. and Heinkel, U.**, (2016). ZYNQ Flexible Platform for Object Recognition & Tracking, *13th Workshop on Positioning, Navigation and Communications (WPNC)*.

[37] **Url-6**, `https://en.wikipedia.org/wiki/Linux`, visiting date: (22.11.2016).

[38] **Xilinx**, September (2016). Zynq-7000 All Programmable SoC Overview.

[39] **AVNET**, June (2012). Getting started with Zed board.

[40] **Xilinx**, March 7, (2011) . AXI Reference Guide, UG761 (v13.1).

[41] **OmniVision**, August (2006). OV7670/OV7171 CMOS VGA (640x480) CameraChip Sensor with OmniPixel Technology.

[42] **Salzman, P.J., Burian, M. and Pomerantz, O.**, May 18, (2007). The Linux Kernel Module Programming Guide.

**APPENDICES**

## APPENDIX A.1: Creation of Development Environment

### A.1.1: Installation of Xilinx Development Environment

#### A.1.1.1: Installation of Xilinx tools

1. Visit and download appreciate version of Vivado Hlx : All OS Installer Single-File from “https://www.xilinx.com/support/download.html" website. In this thesis, 2016.2 version is used

2. After file download extract the compressed file inside a folder.

3. Open terminal and change direction to extracted files folder. Then insert “sudo ./xsetup” command.

4. After installation gui is opened, confirm all license agreements, select “/opt/Xilinx” for installation directory, select “System Edition” for installation and don’t forget to select “Software Development Kit” from list. Click all “Next” buttons and wait for installation is completed.

5. Open “.bashrc” file inside terminal. Then, insert below commands at the end of file. If your Vivado version is different than 2016.2, you must modify below commands according to your Vivado version.

   export SWT_GTK3 = 0

   export PATH = $/opt/Xilinx/Vivado/2016.2/bin/ : /opt/Xilinx/SDK/2016.2/bin/ : /opt/Xilinx/Vivado_HLS/2016.2/bin/ : /home/utku/Desktop/thesis_project_files/u-boot-xlnx/tools/ : \$PATH$

6. Change direction to “opt/Xilinx/Vivado/2016.2” in terminal. Then, type “./settings64.sh” command. Then change direction to “opt/Xilinx/SDK/2016.2” in terminal. Then, type the same command “./settinga64.sh”.

7. Close all terminals. Then, open a new terminal and try to type “vivado”, “vivado_hls” and “xsdk” commands separately in order to prove that your installation is properly completed.

#### A.1.1.2: Installation of Xilinx Linux sources

Xilinx Linux is a official linux kernel from Xilinx. It has been created for running on architectures which has been found on specifically Xilinx devices such as Zynq, MicroBlaze and PowerPC. In order to run Xilinx Linux on these architectures; "linux-xlnx" kernel sources files, "u-boot-xlnx" U-boot source files and "device-tree-xlnx" device tree sources files must be fetched. Thus, firstly git must be installed. Then, these folders can be fetched by typing below commands.

1. sudo apt install git

2. mkdir xilinx_linux, cd xilinx_linux
3. git clone https://github.com/Xilinx/linux-xlnx
4. git clone https://github.com/Xilinx/u-boot-xlnx
5. git clone https://github.com/Xilinx/device-tree-xlnx

#### A.1.1.3: Installation of Minicom

Minicom is a lightweight serial communication program which is running on Linux terminal.

1. Open a new terminal and type “sudo apt-get install minicom” command.
2. In order to run Minicom, type “sudo minicom -s” command.
3. Select “Serial port setup” item from first list.
4. Configure minicom as being in below

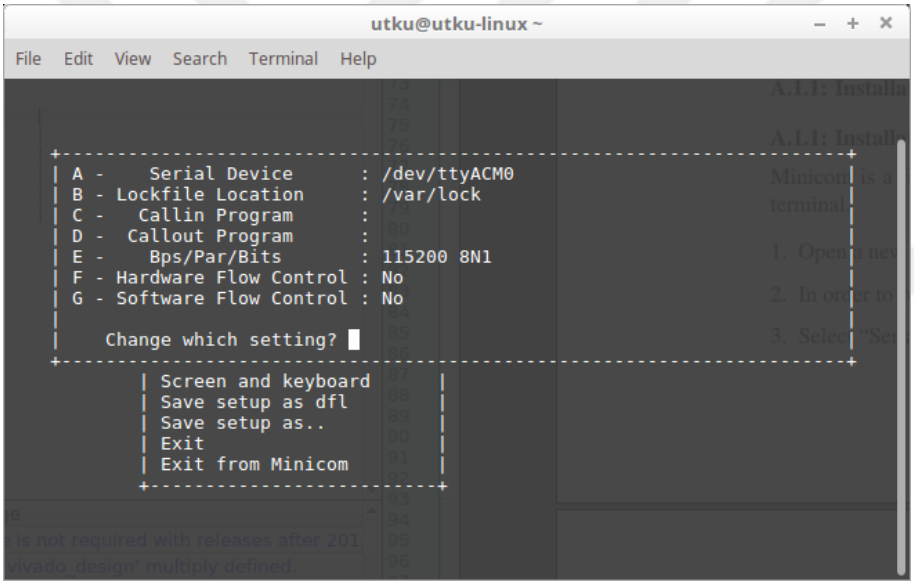


**Figure A.1** : Minicom configuration.

5. After configurate minicom, select "Save setup as dfl" item from list.
6. Select "Exit" from list. Then serail communication will be started.
7. While opening the minicom, "sudo minicom -w" command can be typed after configuration is made in order to connect directly.

### A.1.2: Running Xilinx Linux on ZedBoard

In order to run Linux on ZedBoard, there are several files which must be created by developer. These files are BOOT.bin, FSBL.elf, uBoot.elf, devicetree.dtb, uImage, uramdisk.image.gz or root file system and system.bit files. These files can be copied to sdcard then user can boot Linux and run applications. Please follow steps of this guide respectively.

#### A.1.2.1: Creating first step boot loader(FSBL)

FSBL is a baremetal application which is utilized for fetching U-boot files from Flash to RAM.

1. Follow this way for creating FSBL in Xilinx SDK. File → New → Application Project
2. Insert project name “FSBL” and click on “Next”.
3. Select “Zynq FSBL” from below list and click on “Finish”.

#### A.1.2.2: Building U-Boot

1. In order to boot linux on ZedBoard, there must be a boot loader bare-metal application. U-Boot is ready for use bare-metal application for this reason. U-Boot can be utilized on many platforms. However, it must be compiled in order to run on zynq processor. Therefore, firstly u-boot must be cross-compiled on a host PC for zynq, which means ARM processor. Thus, some packages must be installed by typing below commands.

   sudo apt-get install libssl-dev

   sudo apt-get install gcc-arm-linux-gnueabi

   sudo apt-get install g++-arm-linux-gnueabi

   sudo dpkg –add-architecture i386

   sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386 libz-dev:i386

2. below changes must be done in "/opt/Xilinx/Vivado/2016.2/.settings64-Vivado.sh" file. Make these lines comment by adding "#" character at the beggining of the lines.

   #if [ -n "$LD_LIBRARY_PATH" ]; then

   # export LD_LIBRARY_PATH=/opt/Xilinx/Vivado/2016.2/lib/lnx64.o

   :$LD_LIBRARY_PATH

   #else

   # export LD_LIBRARY_PATH=/opt/Xilinx/Vivado/2016.2/lib/lnx64.o

   #fi

3. Write these commands in <Xilinx_linux folder dir>/Xilinx_linux/u-boot-xlnx/ folder on terminal.

   source /opt/Xilinx/Vivado/2016.2/settings64.sh

   export CROSS_COMPILE=arm-xilinx-linux-gnueabi-

   export ARCH=arm

   make zynq_zed_config

   make

4. After compiling finished, change name of generated “u-boot” file to “u-boot.elf” from /u-boot-xlnx/ folder.

#### A.1.2.3: Creating BOOT.bin file

1. Open Xilinx SDK by writing “xsdk” command on terminal.
2. GUI of creating BOOT.bin file can be opened by "Tools → Create Boot Image".
3. You should click on “Add” from Boot Image Partitions section. Then, you should insert FSBL file path information to “File path” information so click on “Browse” button and enter FSBL/Debug folder. Then, select "FSBL.elf" file.
4. Click on “Add” button again and click on “Browse” button again. Then, enter /design1_wrapper_hw_platform and select xxxxx_wrapper.bit file. Click on “OK” button. Then, click on “OK” again.
5. Click on “Add” button and click on “Browse” button. Enter “xilinx_linux/u-boot-xlnx/” folder and select "u-boot.elf" file. Then click on “OK” button.
6. Click on “Browse” button from “Output Path” and select "<project name>.sdk" folder. Create a new file with “Boot_bin” name. Enter “Boot_bin” folder and click on “OK” button.
7. Click on “Browse” button from “Output BIF file path” and click on “OK” button.
8. Click on “Create Image” button.

You can find your "BOOT.bin" file in "Boot_bin" folder.

#### A.1.2.4: Building the kernel

1. Install some packages by typing these commands

   sudo apt-get install libncurses5-dev libncursesw5-dev

   sudo apt-get install u-boot-tools

2. Before compilation, if you want to change kernel configurations, you can type this command on terminal.

   make menuconfig

   Thanks to this command you can configure your linux kernel with an interface.

3. Write these commands in order to compile linux kernel.

   Export PATH = $/opt/Xilinx/Vivado/2016.2/bin/ : /opt/Xilinx/SDK/2016.2/bin/ : /opt/Xilinx/Vivado\_HLS/2016.2/bin/ : / < path\_of\_xilinx\_linux\_folder > /xilinx\_linux/u - boot - xlnx/tools : / < path\_of\_xilinx\_linux\_folder > /xilinx\_linux/linux\_xlnx/$ :$PATH

   source /opt/Xilinx/Vivado/2016.2/settings64.sh

   export CROSS_COMPILE=arm-xilinx-linux-gnueabi-

   export ARCH=arm

   make clean

   make xilinx_zynq_defconfig

   make UIMAGE_LOADADDR=0x8000 uImage

After compilation finished, you can find uImage file in ..<path of xilinx_linux/linux_xlnx/arch/arm/boot/" path.

#### A.1.2.5: Building the root file system

Ready-for-use file system which is provided by Xilinx will be used. Therefore, follow below steps.

1. Enter this web site "http://www.wiki.xilinx.com/Build+and+Modify+a+Rootfs" and download "arm_ramdisk.image.gz" file by finding it among the "Prebuilt Images". This file is only for Zynq AP SOC(ARM) systems.
2. Change downloaded file name by "uramdisk.image.gz".

#### A.1.2.6: Building the Linux device tree

"devicetree.dtb" file can be automatically created. Thus, follow below steps.

1. Source Xilinx design tools

   source /opt/Xilinx/Vivado/2016.2/settings64.sh
2. Run HSM

   hsm
3. Open HDF file

   open_hw_design <design_name>.hdf

   You can find .hdf file inside your vivado project folder's .sdk folder.
4. Set repository path set_repo_path <path to device-tree-xlnx repository>

   This path must be like ". . . /xilinx_linux/device-tree-xlnx"
5. Create software design and setup CPU You should enter processor_name for ZynqMP psu_cortex53_0, for Zynq ps7_cortexa9_0 and for Microblaze microblaze_0.

   create_sw_design device-tree -os device_tree -proc <processor_name>
6. Generate DTS/DTSI files to folder

   generate_target -dir <folder_name>
7. modify "pl.dtsi" file inside generated folder as you can see below in order to give all costum IPs a spesific name. While Linux kernel module try to write and read some data inside these IPs, device driver will use these names.

   Exp 1: VDMA_axis_to_ddr_writer_0: axis_to_ddr_writer@43c00000 {

   Exp 2: axi_gpio_pl_reset_0: axi_gpio_pl_reset@41210000 {
8. In order to create "devicetree.dtb" file, dtc program which is provided in ". . . /xilinx_linux/linux-xlnx/scripts/dtc" must be used. If you are in your .dts file path, you can utilize these commands for generate "devicetree.dtb" file in terminal.

   source /opt/Xilinx/Vivado/2016.2/settings64.sh

export CROSS_COMPILE=arm-xilinx-linux-gnueabi-

export ARCH=arm

../../linux-xlnx/scripts/dtc/dtc -I dts -O dtb -o devicetree.dtb system.dts

#### A.1.2.7: First boot with Xilinx Linux on ZedBoard

1. In order to prepare Zedboard for booting Linux, we should arrange some jumpers position according to boot from sdcard. Therefore you should change position of jumper 9 and 10 on Zedboard to “3V3” position.

2. arrange sdcard partition according to Linux file system so we can use “Gparted(Gnome partition editor)” program for this reason. In order to download this program you can write this command on terminal.

   sudo apt-get install gparted

3. Run gparted program by typing this command.

   sudo gparted /dev/mmcblk0

   Note: Your sdcard location can be changed according to your system. Check it from “df” command. Then you can find your sdcard’s file path from Filesystem column.

4. Create one partition on your sdcard. The partition must be "FAT32", its size can be all memory on sdcard and its name is "BOOT". Then you can close gparted program.

5. Copy all files we need for booting linux to sdcard. Therefore, you should copy "BOOT.bin" from ...<your_vivado_project_path>/<your_vivado project_name>.sdk/boot_bin folder, "devicetree.dtb" from <your_vivado_project_path>/my_dts/ folder, "uImage" from <xilinx_linux_path>/xilinx_linux/linux-xlnx/arch/arm/boot/ folder and "uramdisk.image.gz" from your download folder.

## APPENDIX B.1: Linux Loadable Kernel Module

### B.1.1: Creating Example Loadable Kernel Module

This section describes how to create new loadable kernel module, how you can insert it to Linux kernel and remove during run time. In order to insert loadable kernel module you can use "insmod" command and to remove you can use "rmmod" command in linux terminal. There is an example loadable kernel module in below. In order to create loadable kernel module, you should create basically "init" and "exit" functions of module. These example modules just print “Hello world” after the module is inserted and print "Goodbye world" text on terminal after module is removed.

If you are trying this loadable kernel module different than Xilinx Linux kernel, you may not see any output because of your distribution’s print permission level. You can see this module results with typing this command on terminal.

dmesg

Example loadable kernel module code:

```
/*
 *  hello.c - The simplest kernel module.
 */
#include <linux/module.h> /* Needed by all modules */
#include <linux/kernel.h> /* Needed for KERN_INFO */

int init_module(void)
{
  printk(KERN_INFO "Hello world.\n");

  /*
   * A non 0 return means init_module failed; module can't be loaded.
   */
  return 0;
}

void remove_module(void)
{
  printk(KERN_INFO "Goodbye world.\n");
}

module_init(init_module);
module_exit(remove_module);
```

#### B.1.1.1:Compiling kernel modules

Kernel modules need to be compiled a bit differently from regular userspace apps. Former kernel versions require us to care much about these settings, which are usually stored in Makefiles. Although hierarchically organized, many redundant settings accumulated in sublevel Makefiles and made them large and rather difficult to maintain. Fortunately, there is a new way of doing these things, called kbuild, and the build

process for external loadable modules is now fully integrated into the standard kernel build mechanism. there is an example "Makefile" below.

---

obj-m += hello-1.o

all:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

---

From a technical point of view just the first line is really necessary, the "all" and "clean" targets were added for pure convenience. Now you can compile the module by issuing the command make.

### B.1.2:Device Drivers in Linux [42]

One class of module is the device driver, which provides functionality for hardware like a TV card or a serialport. On unix, each piece of hardware is represented by a file located in /dev named a device file which provides the means to communicate with the hardware. The device driver provides the communication on behalf of a user program. So the es1370.o sound card device driver might connect the /dev/sound device file to the Ensoniq IS1370 sound card. A userspace program like mp3blaster can use /dev/sound without ever knowing what kind of sound card is installed.

#### B.1.2.1:Major and minor numbers

Let's look at some device files. Here are device files which represent the first three partitions on the primary master IDE hard drive:

```
# ls -l /dev/hda[1-3]
brw-rw----  1 root  disk  3, 1 Jul  5  2000 /dev/hda1
brw-rw----  1 root  disk  3, 2 Jul  5  2000 /dev/hda2
brw-rw----  1 root  disk  3, 3 Jul  5  2000 /dev/hda3
```

**Figure B.1** : Major and minor numbers of devices.

Notice the column of numbers separated by a comma? The first number is called the device's major number. The second number is the minor number. The major number tells you which driver is used to access the hardware. Each driver is assigned a unique major number; all device files with the same major number are controlled by the same driver. All the above major numbers are 3, because they're all controlled by the same driver.

The minor number is used by the driver to distinguish between the various hardware it controls. Returning to the example above, although all three devices are handled by the same driver they have unique minor numbers because the driver sees them as being different pieces of hardware.

Devices are divided into two types: character devices and block devices. The difference is that block devices have a buffer for requests, so they can choose the best order in which to respond to the requests. This is important in the case of storage

devices, where it's faster to read or write sectors which are close to each other, rather than those which are further apart. Another difference is that block devices can only accept input and return output in blocks (whose size can vary according to the device), whereas character devices are allowed to use as many or as few bytes as they like. Most devices in the world are character, because they don't need this type of buffering, and they don't operate with a fixed block size. You can tell whether a device file is for a block device or a character device by looking at the first character in the output of ls -l. If it's "b" then it's a block device, and if it's "c" then it's a character device. The devices you see above are block devices. Here are some character devices (the serial ports):

```
crw-rw----  1 root  dial 4, 64 Feb 18 23:34 /dev/ttyS0
crw-r-----  1 root  dial 4, 65 Nov 17 10:26 /dev/ttyS1
crw-rw----  1 root  dial 4, 66 Jul  5  2000 /dev/ttyS2
crw-rw----  1 root  dial 4, 67 Jul  5  2000 /dev/ttyS3
```

**Figure B.2** : Informations of character devices.

If you want to see which major numbers have been assigned, you can look at /usr/src/linux/Documentation/devices.txt.

When the system was installed, all of those device files were created by the mknod command. To create a new char device named "coffee" with major/minor number 12 and 2, simply do mknod /dev/coffee c 12 2. You don't have to put your device files into /dev, but it's done by convention. Linus put his device files in /dev, and so should you. However, when creating a device file for testing purposes, it's probably OK to place it in your working directory where you compile the kernel module. Just be sure to put it in the right place when you're done writing the device driver.

I would like to make a few last points which are implicit from the above discussion, but I'd like to make them explicit just in case. When a device file is accessed, the kernel uses the major number of the file to determine which driver should be used to handle the access. This means that the kernel doesn't really need to use or even know about the minor number. The driver itself is the only thing that cares about the minor number. It uses the minor number to distinguish between different pieces of hardware.

By the way, when I say "hardware", I mean something a bit more abstract than a PCI card that you can hold in your hand. Look at these two device files:

```
% ls -l /dev/fd0 /dev/fd0u1680
brwxrwxrwx   1 root  floppy   2,  0 Jul  5  2000 /dev/fd0
brw-rw----   1 root  floppy   2, 44 Jul  5  2000 /dev/fd0u1680
```

**Figure B.3** : Informations of block devices.

By now you can look at these two device files and know instantly that they are block devices and are handled by same driver (block major 2). You might even be aware that these both represent your floppy drive, even if you only have one floppy drive. Why two files? One represents the floppy drive with 1.44 MB of storage. The other is the same floppy drive with 1.68 MB of storage, and corresponds to what some people call a "superformatted" disk. One that holds more data than a standard formatted floppy. So here's a case where two device files with different minor number actually represent the same piece of physical hardware. So just be aware that the word 'hardware' in our discussion can mean something very abstract.

There is an example character device code below:

```
/*
* chardev.c: Creates a read-only char device that says how many times
* you've read from the dev file
*/
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <asm/uaccess.h>
/* for put_user */
/*
* Prototypes - this would normally go in a .h file
*/
int init_module(void);
void cleanup_module(void);
static int device_open
(struct inode *, struct file *);
static int device_release
(struct inode *, struct file *);
static ssize_t device_read
(struct file *, char *, size_t, loff_t *);
static ssize_t device_write
(struct file *, const char *, size_t, loff_t *);
#define SUCCESS 0
#define DEVICE_NAME "chardev"
#define BUF_LEN 80
/* Dev name as it appears in /proc/devices
*/
/* Max length of the message from the device */
/*
* Global variables are declared as static,
* so are global within the file.
*/
static int Major;
static int Device_Open = 0;
static char msg[BUF_LEN];
static char *msg_Ptr;
/*Major number assigned to our device driver */
/*Is device open?
* Used to prevent multiple access to device
The msg the device will give when asked */
static struct file_operations fops = {
.read = device_read,
.write = device_write,
.open = device_open,
.release = device_release
};
/*
* This function is called when the module is loaded
*/
int init_module(void)
{
Major = register_chrdev(0, DEVICE_NAME, &fops);
if (Major < 0) {
printk(KERN_ALERT
"Registering char device failed with %d\n", Major);
return Major;
}
```

```
printk(KERN_INFO
"I was assigned major number %d. To talk to\n", Major);
printk(KERN_INFO
"the driver, create a dev file with\n");

printk(KERN_INFO
"'mknod /dev/%s c %d 0'.\n", DEVICE_NAME, Major);
printk(KERN_INFO
"Try various minor numbers. Try to cat and echo to\n");
printk(KERN_INFO
"the device file.\n");
printk(KERN_INFO
"Remove the device file and module when done.\n");
return SUCCESS;
}
/*
* This function is called when the module is unloaded
*/
void cleanup_module(void)
{
/*
* Unregister the device
*/
int ret = unregister_chrdev(Major, DEVICE_NAME);
if (ret < 0)
printk(KERN_ALERT
"Error in unregister_chrdev: %d\n", ret);
}

static int device_open
(struct inode *inode, struct file *file)
{
static int counter = 0;
if (Device_Open)
return -EBUSY;
Device_Open++;
sprintf(msg,
"I already told you %d times Hello world!\n", counter++);
msg_Ptr = msg;
try_module_get(THIS_MODULE);
return SUCCESS;
}
/*
* Called when a process closes
* the device file.
*/
static int device_release
(struct inode *inode, struct file *file)
{
Device_Open--;
module_put(THIS_MODULE);
return 0;
}

static ssize_t device_read
(struct file *filp,
char *buffer,
size_t length,
loff_t * offset)
```

```
{

int bytes_read = 0;
if (*msg_Ptr == 0)
return 0;
/*
* Actually put the data into the buffer
*/
while (length && *msg_Ptr) {
/*
* The buffer is in the user data segment,
* not the kernel
* segment so "*" assignment won't work.
* We have to use
* put_user which copies data from
* the kernel data segment to
* the user data segment.
*/
put_user(*(msg_Ptr++), buffer++);
length--;
bytes_read++;
}
/*
* Most read functions return
* the number of bytes put into the buffer
*/
return bytes_read;
}
/*
* Called when a process writes to
* dev file: echo "hi" > /dev/hello
*/
static ssize_t
device_write
(struct file *filp,
const char *buff,
size_t len, loff_t * off)
{
printk(KERN_ALERT
"Sorry, this operation is not supported.\n");
return -EINVAL;
}
```

### B.1.3:Interrupt Handling With Linux

There are two types of interaction between the CPU and the rest of the computer's hardware. The first type is when the CPU gives orders to the hardware, the other is when the hardware needs to tell the CPU something. The second, called interrupts, is much harder to implement because it has to be dealt with when convenient for the hardware, not the CPU. Hardware devices typically have a very small amount of RAM, and if you don't read their information when available, it is lost.

Under Linux, hardware interrupts are called IRQ's (InterruptRequests). There are two types of IRQ's, short and long. A short IRQ is one which is expected to take a very short period of time, during which the rest of the machine will be blocked and no other interrupts will be handled. A long IRQ is one which can take longer, and during which

other interrupts may occur (but not interrupts from the same device). If at all possible, it's better to declare an interrupt handler to be long.

When the CPU receives an interrupt, it stops whatever it's doing (unless it's processing a more important interrupt, in which case it will deal with this one only when the more important one is done), saves certain parameters on the stack and calls the interrupt handler. This means that certain things are not allowed in the interrupt handler itself, because the system is in an unknown state. The solution to this problem is for the interrupt handler to do what needs to be done immediately, usually read something from the hardware or send something to the hardware, and then schedule the handling of the new information at a later time (this is called the "bottom half") and return. The kernel is then guaranteed to call the bottom half as soon as possible – and when it does, everything allowed in kernel modules will be allowed.

The way to implement this is to call "request_irq()" to get your interrupt handler called when the relevant IRQ is received. This function receives the IRQ number, the name of the function, flags, a name for "/proc/interrupts" and a parameter to pass to the interrupt handler. Usually there is a certain number of IRQs available. How many IRQs there are is hardware-dependent. The flags can include "SA_SHIRQ" to indicate you're willing to share the IRQ with other interrupt handlers (usually because a number of hardware devices sit on the same IRQ) and "SA_INTERRUPT" to indicate this is a fast interrupt. This function will only succeed if there isn't already a handler on this IRQ, or if you're both willing to share.

Then, from within the interrupt handler, we communicate with the hardware and then use "queue_work()" "mark_bh(BH_IMMEDIATE)" to schedule the bottom half.

### B.1.4:Compiling User Space Application for Xilinx Linux

After you create your design file, in this case let us assume that its name is "hello.c", you can cross-compile it with below steps.

1. Open a terminal and call Xilinx source with the help of below command.

   source /opt/Xilinx/Vivado/2016.2/settings64.sh

2. Select the tool chain by typing below command.

   export CROSS_COMPILE=arm-xilinx-linux-gnueabi-

3. Select desired platform by typing below command

   export ARCH=arm

4. Cross compile your design with the help of below command

   arm-xilinx-linux-gnueabi-gcc hello.c -o hello_app

## APPENDIX C.1: Matlab Implementation of Proposed Method

### C.1.1: Matlab Code of Information Hiding Process

```
clc; clear all; close all;
tic
disp('Information hiding is started...');
%% encryption block 1
original_data = imread('input_image_3.jpg');%imread('greens.jpg');
% key image is selected from database
[w h c] = size(original_data);  % sizes of key image are obtained
 %% adding k parameter
    % Converting Plaintext ascii to binary
    % Reshape binary plaintext to 1 dimension
    key = zeros(6,1);
     i = 1;
    while i < 7
        if mod(i,2) == 1     % x axis
            key(i) = input(['Input your key number',...
      num2str(i),' = \n']);
            if (key(i)< 1) || (key(i) > 2*w -1)
                disp('Your key number is wrong!');
                disp(['Key number must be between 1 and ',...
        num2str(2*w-1)]);
                disp('Please input true number');
            else
                i = i + 1;
            end
        else % y axis
            key(i) = input(['Input your key number',...
      num2str(i),' = \n']);
            if (key(i)< 1) || (key(i) > 2*h -1)
                disp('Your key number is wrong!');
                disp(['Key number must be between 1 and ',...
        num2str(2*h-1)]);
                disp('Please input true number');
            else
                i = i + 1;
            end
        end
    end
    disp('Data Hiding process is being continued');
%% encryption block 2
cipher_image = zeros(w * 2-1, h * 2-1, c);
% cipher text is created from sizes of key
for k = 1 : 1 : c
    for i = 1 : 1 : w
        for j = 1 : 1 : h
            % key pixels are including to ciphertext
            cipher_image((i-1)*2+1, (j-1)*2+1, k) = ...
      original_data(i,j,k);
        end
    end
```

```
end
% figure,imshow(uint8(cipher_image));

%% ecryption block 3
% interpolation is being processed

for k = 1 : 1 : c
    for i = 1 : 2 : w*2-2
        for j = 1 : 2 : h*2-2
            % (a, b+1)
            cipher_image(i, j+1, k) = ...
      floor((cipher_image(i, j, k) +...
      cipher_image(i, j+2, k))/2);
            % (a+1, b)
            cipher_image(i+1, j, k) = ...
      floor((cipher_image(i, j, k) +...
      cipher_image(i+2, j, k))/2);
            % (a+1, b+1)
            cipher_image(i+1, j+1, k) = ...
      floor(((cipher_image(i, j, k) +...
                cipher_image(i, j+1, k) +...
                cipher_image(i+1, j, k)) / 3));
        end
    end
end
interpolation_image = cipher_image;
% imwrite(uint8(interpolation_image), 'interpolation_image.png');
% figure,imshow(uint8(cipher_image));
%% encryption block 4
% measureing size of each pixel's plaintext part

% figure,imshow(uint8(derivative_data));
bit_number = zeros(w * 2, h * 2, c);
total_bit = 0;
for k = 1 : 1 : c
    for i = 1 : 2 : (w-1) * 2
        for j = 1 : 2 : (h-1) * 2
            % key pixels are including to ciphertext
            % (a, b+1)
            difference_1 = abs(cipher_image(i+1, j, k) - ...
      cipher_image(i, j, k));
            for l = 1 : 1 : 8
                if(2^l>= difference_1)
                    bit_number(i+1, j, k) = l;
                    break;
                end
            end
            total_bit = total_bit + l;
            % (a+1, b)
            difference_1 = abs(cipher_image(i, j+1, k) - ...
      cipher_image(i, j, k));
            for l = 1 : 1 : 8
                if(2^l>= difference_1)
                    bit_number(i, j+1, k) = l;
                    break;
                end
            end
            total_bit = total_bit + l;
            % (a+1, b+1)
```

```
            difference_1 = abs(cipher_image(i+1, j+1, k) - ...
        cipher_image(i, j, k));
            for l = 1 : 1 : 8
                if(2^l>= difference_1)
                    bit_number(i+1, j+1, k) = l;
                    break;
                end
            end
            total_bit = total_bit + l;
        end
    end
end
max_plaintext_size = floor(total_bit / 8);
%  figure,imshow(uint8(bit_number*10));
imwrite(uint8(bit_number),'bit_number.png');

%% encryption block 5
% Converting Plaintext ascii to binary
% Reshape binary plaintext to 1 dimension

file = fopen('plaintext.txt','r');
plaintext = dec2bin(fread(file),8);
% [char(bin2dec(plaintext))]
fclose(file);
[plain_w,plain_h] = size(plaintext);
if plain_w * plain_h > total_bit
    disp('there is an error about selecting plaintext,...
    plaintext is very long');
else
    disp(['Message size : ', num2str(plain_w * plain_h),...
  ' Cover image capacity : ', num2str(total_bit)]);
    long_plaintext = ...
  dec2bin((zeros(ceil(total_bit/plain_h),1)),plain_h);
    long_plaintext(1:plain_w,:) = plaintext;
    resized_one_dim_plaintext = char((zeros(1,(total_bit))));
    l = 1;
    for i = 1 : plain_h : plain_w * plain_h
        resized_one_dim_plaintext(i:i+plain_h-1) = ...
    long_plaintext(l,:);
        l = l+ 1;
    end
    for i = plain_w * plain_h+1 : plain_h : ceil(total_bit/plain_h)
        resized_one_dim_plaintext(i:i+plain_h-1) =...
    dec2bin(0,plain_h);
    end
    %     resized_one_dim_plaintext = ...
  %   char(reshape(long_plaintext, 1, []));
    %     asd = reshape(resized_one_dim_plaintext, [],plain_h);
    save('all_data.mat',...
  'resized_one_dim_plaintext',...
  'long_plaintext');

    %% encrytion block 6 && block 7
    % adding certain bits of binary plaintext to interpolation values
    bit_start = 1;
    coordinates = zeros(3,2);
    for a = 1: 2: w * 2-2
        for b = 1: 2: h * 2-2
            for k = 1: 1: c
```

```
                if(bit_start > plain_w * plain_h )
                    continue;
                end
                % measuring coordinates of areas on shifting block
                coordinates(1,:) = [a, b+1];     % area 1
                coordinates(2,:) = [a+1, b];     % area 2
                coordinates(3,:) = [a+1, b+1];     % area 3
                for p = 1: 1: 3 % make this procces for 3 areas

                    [i,j] = measure_symmetry_point(...
            coordinates(p,1),...
            coordinates(p,2),...
            key,2*w-1,2*h-1);

                    if(bit_number(i,j) > 1)
                        bit_finish = bit_start + bit_number(i,j)-1;
                        if  bit_finish > plain_w * plain_h
                            break;
                        else
                            cipher_image(i,j,k) = ...
                cipher_image(i,j,k) + ...
                            bin2dec(resized_one_dim_plaintext(...
                bit_start:bit_finish));
                            bit_start = bit_finish + 1;
                        end
                    elseif(bit_number(i,j) == 1)
                        cipher_image(i,j,k) = cipher_image(i,j,k)...
              + bin2dec(...
              resized_one_dim_plaintext(bit_start));
                        bit_start = bit_start + 1;
                    else
                        continue;
                    end
                end
            end
        end
    end
    imwrite(uint8(cipher_image), 'StegoImage_1.png');
    %      asd = imread('cipher_image.png');
    %      sum(sum(abs(uint8(cipher_image) - asd)))

    imshow(original_data);  % key image is showed
    figure,imshow(uint8(cipher_image));
    figure,imshow(uint8(abs(cipher_image - ...
  interpolation_image).*100));
    %sum(sum(abs(cipher_image - interpolation_image)));

end
toc
disp('Information hiding process finished');
```

### C.1.2: Matlab Code of Calculating Symmetry Point of Pixel Function

```
function [new_coordinate_x, new_coordinate_y] = ...
measure_symmetry_point(input_x, input_y, key, w, h)
    if(mod(input_x, 2) == 1)&&(mod(input_y,2) == 0) % 1. area
        state_input = 0;
    elseif (mod(input_x, 2) == 0)&&(mod(input_y,2) == 1) % 2. area
        state_input = 1;
    else % 3. area
        state_input = 2;
    end
    input_x = input_x -1;
    input_y = input_y -1;
    switch( state_input )
%%%%%%%%%%%%%% CASE 0 %%%%%%%%%% AREA 1 %%%%%%%%%%%%%
        case 0
            new_coordinate_x = 2 * key(1) - (input_x);
            if new_coordinate_x < 0
              new_coordinate_x = new_coordinate_x + (w);
              if new_coordinate_x < 0
                 new_coordinate_x = new_coordinate_x + (w);
              end
            end
            if new_coordinate_x > w-1
              new_coordinate_x = new_coordinate_x - (w);
              if new_coordinate_x > w-1
                new_coordinate_x = new_coordinate_x - (w);
              end
            end

             new_coordinate_y = 2 * key(2) - (input_y);
            if new_coordinate_y < 0
              new_coordinate_y = new_coordinate_y + (h);
              if new_coordinate_y < 0
                new_coordinate_y = new_coordinate_y + (h);
              end
            end
            if new_coordinate_y > h-1
              new_coordinate_y = new_coordinate_y - (h);
               if new_coordinate_y > h-1
                  new_coordinate_y = new_coordinate_y - (h);
                end
            end
%%%%%%%%%%%%%%%%%%%%%% CASE 1 %%%%%%%%%%%%%%%% AREA 2 %%%%%%%
        case 1
             new_coordinate_x = 2 * key(3) - (input_x);
            if new_coordinate_x < 0
              new_coordinate_x = new_coordinate_x + (w);
              if new_coordinate_x < 0
                new_coordinate_x = new_coordinate_x + (w);
              end
            end
            if new_coordinate_x > w-1
              new_coordinate_x = new_coordinate_x - (w);
              if new_coordinate_x > w-1
                  new_coordinate_x = new_coordinate_x - (w);
                end
```

```
            end
             new_coordinate_y = 2 * key(4) - (input_y);
            if new_coordinate_y < 0
              new_coordinate_y = new_coordinate_y + (h);
              if new_coordinate_y < 0
                  new_coordinate_y = new_coordinate_y + (h);
                end
            end
            if new_coordinate_y > h-1
              new_coordinate_y = new_coordinate_y - (h);
               if new_coordinate_y > h-1
                  new_coordinate_y = new_coordinate_y - (h);
                end
            end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% CASE 2 %%%%%%%%%% AREA 3 %%%%%
        case 2
             new_coordinate_x = 2 * key(5) - (input_x);
            if new_coordinate_x < 0
              new_coordinate_x = new_coordinate_x + (w);
              if new_coordinate_x < 0
                  new_coordinate_x = new_coordinate_x + (w);
                end
            end
            if new_coordinate_x > w-1
              new_coordinate_x = new_coordinate_x - (w);
               if new_coordinate_x > w-1
                  new_coordinate_x = new_coordinate_x - (w);
                end
            end
             new_coordinate_y = 2 * key(6) - (input_y);
            if new_coordinate_y < 0
              new_coordinate_y = new_coordinate_y + (h);
              if new_coordinate_y < 0
                  new_coordinate_y = new_coordinate_y + (h);
                end
            end
            if new_coordinate_y > h-1
              new_coordinate_y = new_coordinate_y - (h);
              if new_coordinate_y > h-1
                  new_coordinate_y = new_coordinate_y - (h);
                end
            end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        otherwise
            disp('error !');
    end
    new_coordinate_x = new_coordinate_x +1;
    new_coordinate_y = new_coordinate_y +1;

end
```

### C.1.3: Matlab Code of Information Extracting Process

```
clc; clear all; close all;

disp('stego_image extracting process is started...');
file = fopen('results/stego_image.dat','r');
stego_image_temp = fread(file );
fclose(file);
stego_image = reshape(stego_image_temp, [480 640]);
[w h c] = size(stego_image);
imwrite(uint8((stego_image)), 'stego_image.bmp', 'bmp');
figure, imshow(uint8((stego_image)));
title('stego image');

interpolation_image = zeros(w, h, c);
interpolation_image = stego_image;
%% first area
interpolation_image(1:2:w-2, 2:2:h-2,:) = ...
uint8(floor((stego_image(1:2:w-2, 1:2:h-2,:) +...
 stego_image(1:2:w-2, 3:2:h,:))/2));
%% second area
interpolation_image(2:2:w-2, 1:2:h-2,:) = ...
uint8(floor((stego_image(1:2:w-2, 1:2:h-2,:) +...
 stego_image(3:2:w, 1:2:h-2,:))/2));
%% third area
interpolation_image(2:2:w-2, 2:2:h-2,:) = ...
uint8(floor((stego_image(1:2:w-2, 1:2:h-2,:) +...
 stego_image(3:2:w, 3:2:h,:))/2));

bit_number = zeros(w, h, c);
total_bit = 0;
for k = 1 : 1 : c
   for i = 1 : 2 : w - 2
       for j = 1 : 2 : h - 2
           difference_1 = abs(interpolation_image(i, j+1, k) - ...
            interpolation_image(i, j, k));
           bit_number(i, j+1, k) = ...
            measure_bit_number(difference_1);
           total_bit = total_bit + bit_number(i, j+1, k);

           difference_2 = abs(interpolation_image(i+1, j, k) - ...
            interpolation_image(i, j, k));
           bit_number(i+1, j, k) = ...
            measure_bit_number(difference_2);
           total_bit = total_bit + bit_number(i+1, j, k);

           difference_3 = abs(interpolation_image(i+1, j+1, k) - ...
            interpolation_image(i, j, k));
           bit_number(i+1, j+1, k) = ...
            measure_bit_number(difference_3);
           total_bit = total_bit + bit_number(i+1, j+1, k);
       end
   end
end
total_bit
```

```
%% DATA EXTRACTION
diff_image = uint8(abs(stego_image - interpolation_image));
key = zeros(6,1);
i = 1;
while i < 7
    if mod(i,2) == 1     % x axis
        key(i) = input(['Input your key number',num2str(i),' = ']);
        if (key(i)< 1) || (key(i) > w -1)
            disp('Your key number is wrong!');
            disp(['Key number must be between 1 and ',num2str(w-1)]);
            disp('Please input true number');
        else
            i = i + 1;
        end
    else % y axis
        key(i) = input(['Input your key number',num2str(i),' = ']);
        if (key(i)< 1) || (key(i) > h -1)
            disp('Your key number is wrong!');
            disp(['Key number must be between 1 and ',num2str(h-1)]);
            disp('Please input true number');
        else
            i = i + 1;
        end
    end
end

disp('Data extraction process is being continued');
binary_data = char((zeros(1,(total_bit))));
bit_start = double(1);
coordinates = zeros(3,2);
for k = 1: 1: c
    for a = 1: 2: w-1
        for b = 1: 2: h-1
            % measuring coordinates of areas on shifting block
            coordinates(1,:) = [a, b+1];    % area 1
            coordinates(2,:) = [a+1, b];    % area 2
            coordinates(3,:) = [a+1, b+1];    % area 3
            for p = 1: 1: 3 % make this procces for 3 areas
                [i,j] = measure_symmetry_point(
                            coordinates(p,1),
                            coordinates(p,2),
                            key,w,h
                        );
                if(bit_number(i,j) > 1)
                    eklenecek_deger = (bit_number(i,j))-1;
                    bit_finish = double(bit_start + eklenecek_deger);
                    if  bit_finish > total_bit
                        break;
                    else
                        binary_data(bit_start:bit_finish) = ...
                        dec2bin(diff_image(i,j,k), bit_number(i,j));
                        bit_start = bit_finish + 1;
                    end
                elseif(bit_number(i,j) == 1)
                    binary_data(bit_start) = ...
                    dec2bin(diff_image(i,j,k),1);
                    bit_start = bit_start + 1;
                else
                    continue;
```

```
                end
            end
        end
    end
end

character_size = ceil(bit_finish / 8)
resized_two_dim_data = (zeros(character_size,1));
char_number = 1;
for i = 1 : 8 : bit_finish-7
    resized_two_dim_data(char_number) = bin2dec(binary_data(i:i+7));
    char_number = char_number + 1;
end
file = fopen('delta_planintext.txt','w');
fwrite(file,((resized_two_dim_data)));
fclose(file);
open('delta_planintext.txt');
disp('Data extracting process is completed');
```

## APPENDIX D.1: Embedded System Implementation of Proposed Method

### D.1.1: Device Driver of Hardware System for Linux

```
/*
 * driver.c
 *
 *  Created on: June 26, 2018
 *      Author: utku
 */
//////////////////////////////////// Libraries ////////////////
#include <linux/module.h>
#include <linux/kernel.h>

#include <linux/fs.h>      // file operations structure.
//Which of course allows use to open/close, read/write to device
#include <linux/cdev.h>      // This is a char driver;
//makes cdev available
#include <linux/semaphore.h>  // Used to access semaphores;
//sychronization behaviors
#include <asm/uaccess.h>     // copy_to_user; copy_from_user

#include <linux/io.h>
//  iowrite8   iowrite32  iounmap
#include <linux/interrupt.h>
// irqreturn_t  NO_IRQ  request_irq  free_irq
#include <linux/ioport.h> //needed for resource struct
// request_mem_region  struct resource   release_mem_region
#include <linux/of.h>
#include <linux/of_irq.h>
#include <linux/of_address.h>
#include <linux/delay.h>
#include <linux/slab.h>
#include <asm/io.h>

/////////////// User definitions ////////////////////////////////
//#define DEBUG_MODE
#define LED_DELAY 500000
#define DEVICE_NAME "OV7670"

// FRAME CONSTANTS
#define FRAME_BUFFER_DIM_WRITER 307200
#define FRAME_BUFFER_BASE_ADDR_WRITER 0x10000000
#define FRAME_BUFFER_NUM_WRITER 8

#define FRAME_BUFFER_DIM_READER 307200
#define FRAME_BUFFER_BASE_ADDR_READER 0x10000000
#define FRAME_BUFFER_NUM_READER 8

// XAXIS_TO_DDR_WRITER
#define XAXIS_TO_DDR_WRITER_AXILITES_ADDR_AP_CTRL   \
                    0x00
#define XAXIS_TO_DDR_WRITER_AXILITES_ADDR_GIE     \
```

```
                          0x04
#define XAXIS_TO_DDR_WRITER_AXILITES_ADDR_IER     \
                           0x08
#define XAXIS_TO_DDR_WRITER_AXILITES_ADDR_ISR      \
                          0x0c
#define XAXIS_TO_DDR_WRITER_AXILITES_ADDR_BASE_ADDRESS_DATA     \
          0x10
#define XAXIS_TO_DDR_WRITER_AXILITES_BITS_BASE_ADDRESS_DATA     \
          32
#define XAXIS_TO_DDR_WRITER_AXILITES_ADDR_FRAME_BUFFER_DIM_DATA     \
       0x18
#define XAXIS_TO_DDR_WRITER_AXILITES_BITS_FRAME_BUFFER_DIM_DATA     \
       32
#define XAXIS_TO_DDR_WRITER_AXILITES_ADDR_FRAME_BUFFER_OFFSET_DATA   \
  0x20
#define XAXIS_TO_DDR_WRITER_AXILITES_BITS_FRAME_BUFFER_OFFSET_DATA\
     32
#define XAXIS_TO_DDR_WRITER_AXILITES_ADDR_FRAME_BUFFER_NUMBER_DATA\
     0x28
#define XAXIS_TO_DDR_WRITER_AXILITES_BITS_FRAME_BUFFER_NUMBER_DATA\
       8
#define XAXIS_TO_DDR_WRITER_AXILITES_ADDR_UPDATE_INTR_DATA      \
          0x30
#define XAXIS_TO_DDR_WRITER_AXILITES_BITS_UPDATE_INTR_DATA   \
             1

// XDDR_TO_AXIS_READER
#define XDDR_TO_AXIS_READER_AXILITES_ADDR_AP_CTRL   \
                        0x00
#define XDDR_TO_AXIS_READER_AXILITES_ADDR_GIE       \
                         0x04
#define XDDR_TO_AXIS_READER_AXILITES_ADDR_IER      \
                          0x08
#define XDDR_TO_AXIS_READER_AXILITES_ADDR_ISR       \
                        0x0c
#define XDDR_TO_AXIS_READER_AXILITES_ADDR_BASE_ADDRESS_DATA \
            0x10
#define XDDR_TO_AXIS_READER_AXILITES_BITS_BASE_ADDRESS_DATA   \
           32
#define XDDR_TO_AXIS_READER_AXILITES_ADDR_FRAME_BUFFER_DIM_DATA\
          0x18
#define XDDR_TO_AXIS_READER_AXILITES_BITS_FRAME_BUFFER_DIM_DATA\
          32
#define XDDR_TO_AXIS_READER_AXILITES_ADDR_FRAME_BUFFER_OFFSET_DATA\
       0x20
#define XDDR_TO_AXIS_READER_AXILITES_BITS_FRAME_BUFFER_OFFSET_DATA\
       32
#define XDDR_TO_AXIS_READER_AXILITES_ADDR_FRAME_BUFFER_NUMBER_DATA\
       0x28
#define XDDR_TO_AXIS_READER_AXILITES_BITS_FRAME_BUFFER_NUMBER_DATA\
       8
#define XDDR_TO_AXIS_READER_AXILITES_ADDR_UPDATE_INTR_DATA \
            0x30
#define XDDR_TO_AXIS_READER_AXILITES_BITS_UPDATE_INTR_DATA \
             1

//////////////////// Global Variables //////////////////////////////
static struct character_device {
  char last_image_number;
```

```
  char *last_image_address;
  char interrupt_status;
  char reset_pl_status;
  // writer base address
  u32 image_writer_base_address;
  // reader base address
  u32 image_reader_base_address;
  //struct semaphore sem;
} ov7670_device;

static struct cdev *mcdev;
static int major_number;
static int ret;

static dev_t dev_num;
void __iomem *reg_ptr_tmp;

void __iomem *ov7670_reg;
static  struct device_node *ov7670_node;
static  struct resource ov7670;
static  unsigned int irq_ov7670;


void __iomem *reg_ptr_tmp;

void __iomem *pl_reset_ip_reg;
static  struct device_node *pl_reset_ip_node;
static  struct resource pl_reset_ip;
static bool initialize_pl_reset_ip_bool = false;
static int pl_is_on_reset = 0;
#define RESET_ENABLED 0
#define RESET_DISABLED 1

void __iomem *axis_to_ddr_writer_reg;
static  struct device_node *axis_to_ddr_writer_node;
static  struct resource axis_to_ddr_writer;

void __iomem *ddr_to_axis_reader_reg;
static  struct device_node *ddr_to_axis_reader_node;
static  struct resource ddr_to_axis_reader;

void __iomem *frame_buffer_start_address;

///////////////// User Functions ////////////////////////
static int __init ov7670_driver_init(void);
static void __exit ov7670_driver_exit(void);
int driver_entry(void);

int initialize_platform(void);
int initialize_components(void);
void platform_release(void);
int initialize_interrupts(void);
int initialization_after_reset(void);

void wait(void);

u32 read_data;
// initialize ov7670 gpio ip
int initialize_ov7670(void);
```

```
// initialize ov7670 interrupt
int initialize_ov7670_interrupt(void);
irqreturn_t irq_default_primary_handler(int irq, void *dev_id);
// ov7670 interrupt service routine
irqreturn_t ov7670_irq_handler(int irq, void *dev_id);
//release_ov7670 during module exit
void release_ov7670(void);

int initialize_pl_reset_ip(void);
int pl_reset_ip_enable(void);
int pl_reset_ip_disable(void);
int pl_reset_ip_keep_milisecond(int msec);
int is_pl_reset_ip_on_reset(void);
//release_ov7670 during module exit

int initialize_axis_to_ddr_writer(void);
int Configure_axis_to_ddr_writer(void);

void axis_to_ddr_writer_Set_base_address
(void __iomem *int_ptr, u32 data);
void axis_to_ddr_writer_Set_frame_buffer_dim
(void __iomem *int_ptr, u32 data);
void axis_to_ddr_writer_Set_frame_buffer_number
(void __iomem *int_ptr, u32 data);
void axis_to_ddr_writer_Set_frame_buffer_offset
(void __iomem *int_ptr, u32 data);
void axis_to_ddr_writer_EnableAutoRestart
(void __iomem *int_ptr, u32 data);
void axis_to_ddr_writer_Set_update_intr
(void __iomem *int_ptr, u32 data);
void axis_to_ddr_writer_Start
(void __iomem *int_ptr, u32 data);

int initialize_ddr_to_axis_reader(void);
int Configure_ddr_to_axis_reader(void);

void ddr_to_axis_reader_Set_base_address
(void __iomem *int_ptr, u32 data);
void ddr_to_axis_reader_Set_frame_buffer_dim
(void __iomem *int_ptr, u32 data);
void ddr_to_axis_reader_Set_frame_buffer_number
(void __iomem *int_ptr, u32 data);
void ddr_to_axis_reader_Set_frame_buffer_offset
(void __iomem *int_ptr, u32 data);
void ddr_to_axis_reader_Set_update_intr
(void __iomem *int_ptr, u32 data);
void ddr_to_axis_reader_EnableAutoRestart
(void __iomem *int_ptr, u32 data);
void ddr_to_axis_reader_Start
(void __iomem *int_ptr, u32 data);


void release_pl_reset_ip(void);


int start_application(void);
void stop_application(void);

void interrupt_status_control
```

```
(struct character_device temp_device);
void enable_frame_buffer_interrupt(void);
void disable_frame_buffer_interrupt(void);

void reset_pl_status_control
(struct character_device temp_device);
void writer_base_addr_control
(struct character_device temp_device);
void reader_base_addr_control
(struct character_device temp_device);


//////// Loadable kernel module init function ///////////////
static int __init ov7670_driver_init(void)
{
  // response variable is used
  // for checking wheather process finish with correct or error
  int response = 0;
  printk(KERN_ERR "initialize_system is started\n");
  // initialize_system
  response = initialize_platform();
  if(response != 0)
  {
    return -1;
  }
  #ifdef DEBUG_MODE
    printk(KERN_ERR "initialize_system is done\n");
  #endif

  // start_application
  response = start_application();
  if(response != 0)
  {
    return -1;
  }
  #ifdef DEBUG_MODE
    printk(KERN_ERR "start_application is done\n");
  #endif

  response = driver_entry();
  if(response != 0)
  {
    return -1;
  }
  printk(KERN_ERR "Device started!\n");

  return 0;
}
/////////////////////////////////////////////////////////////

////////// Loadable kernel module exit function ///////////////
static void __exit ov7670_driver_exit(void)
{
  // response variable is used
  // for checking wheather process finish with correct or error
  stop_application();
  #ifdef DEBUG_MODE
    printk(KERN_INFO "stop_application is done\n");
  #endif
```

```
  // all_components_release
  platform_release();
  #ifdef DEBUG_MODE
    printk(KERN_INFO "all_components_release is done\n");
  #endif

  cdev_del(mcdev);
  unregister_chrdev_region(dev_num, 1);
  printk(KERN_ALERT "ov7670_driver : unloaded module \n");

}

int initialize_platform(void)
{
  // response variable is used
  // for checking wheather process finish with correct or error
  int response = 0;

  // initialize all used peripherals
  response = initialize_components();
  if(response != 0)
  {
    return -1;
  }
  #ifdef DEBUG_MODE
    printk(KERN_ERR
    "\nInitializations of all components are done.\n");
  #endif



  return 0;
}

////////////// Initialize all componenets ////////////////////
int initialize_components(void)
{
  // response variable is used
  // for checking wheather process finish with correct or error
  int response = 0;

  // initializing ov7670 gpio
  response = initialize_ov7670();
  // checking initialize led process is done
  if(response != 0)
  {
    return -1;
  }
  #ifdef DEBUG_MODE
    printk(KERN_ERR "ov7670 initialization is done\n");
  #endif

  // initializing initialize_pl_reset_ip gpio
  response = initialize_pl_reset_ip();
  // checking initialize initialize_pl_reset_ip process is done
  if(response != 0)
  {
    return -1;
```

```
  }

  // initializing axis_to_ddr_writer gpio
  response = initialize_axis_to_ddr_writer();
  // checking initialize axis_to_ddr_writer process is done
  if(response != 0)
  {
    return -1;
  }

  #ifdef DEBUG_MODE
    printk(KERN_INFO "axis_to_ddr_writer initialization is done\n");
  #endif

  // initializing ddr_to_axis_reader gpio
  response = initialize_ddr_to_axis_reader();
  // checking initialize ddr_to_axis_reader process is done
  if(response != 0)
  {
    return -1;
  }

  #ifdef DEBUG_MODE
    printk(KERN_INFO "ddr_to_axis_reader initialization is done\n");
  #endif

  return 0;
}

void platform_release(void)
{
  // release_ov7670
  release_ov7670();
  // release_pl_reset_ip
  release_pl_reset_ip();
}


int initialize_axis_to_ddr_writer(void)
{
  // response variable is used
  // for checking wheather process finish with correct or error
  int response = 0;

  // finding axis_to_ddr_writer node
  //according to name of axis_to_ddr_writer in devicetree.dts file
  axis_to_ddr_writer_node =
  (struct device_node *)of_find_node_by_name
    (NULL, "axis_to_ddr_writer");

  // creating resource for
  // axis_to_ddr_writer with the help of axis_to_ddr_writer node
  response = of_address_to_resource
    (axis_to_ddr_writer_node, 0, &axis_to_ddr_writer);
  if (response < 0)
  {
    printk(KERN_INFO
"error in of_address_to_resource axis_to_ddr_writer\n");
    return -1;
```

```
  }
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"of_address_to_resource axis_to_ddr_writer is done.\n");
  #endif

  // checking wheather axis_to_ddr_writer
  // address space is used or not
  if  (request_mem_region
  (axis_to_ddr_writer.start,
  resource_size(&axis_to_ddr_writer),
  "axis_to_ddr_writer") == NULL)
  {
    printk(KERN_INFO
"error in request_mem_region axis_to_ddr_writer\n");
    return -1;
  }
  #ifdef DEBUG_MODE
    // If there is not error,
    //show the request_mem_region of axis_to_ddr_writer
    printk(KERN_INFO
"request_mem_region axis_to_ddr_writer DONE. \n");
  #endif


  // axis_to_ddr_writer address space is allocating
  axis_to_ddr_writer_reg = (void __iomem *)of_iomap
                           (axis_to_ddr_writer_node, 0);
  if (!axis_to_ddr_writer_reg)
  {
    printk(KERN_INFO "could not allocate iomap\n");
    release_mem_region
            (axis_to_ddr_writer.start,
             resource_size(&axis_to_ddr_writer));
    return -1;
  }
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"axis_to_ddr_writer registers are allocated. \n");
  #endif

  return 0;
}

int initialize_ddr_to_axis_reader(void)
{
  // response variable is used
  // for checking wheather process finish with correct or error
  int response = 0;

  // finding ddr_to_axis_reader node
  // according to name of ddr_to_axis_reader in devicetree.dts file
  ddr_to_axis_reader_node =
(struct device_node *)of_find_node_by_name
(NULL, "ddr_to_axis_reader");

  // creating resource for
  // ddr_to_axis_reader with the help of ddr_to_axis_reader node
  response = of_address_to_resource
```

```
        (ddr_to_axis_reader_node, 0, &ddr_to_axis_reader);
  if (response < 0)
  {
    printk(KERN_INFO
"error in of_address_to_resource ddr_to_axis_reader\n");
    return -1;
  }
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"of_address_to_resource ddr_to_axis_reader is done.\n");
  #endif

  // checking wheather ddr_to_axis_reader
  // address space is used or not
  if  (request_mem_region
        (ddr_to_axis_reader.start,
        resource_size(&ddr_to_axis_reader),
        "ddr_to_axis_reader") == NULL
    )
  {
    printk(KERN_INFO
"error in request_mem_region ddr_to_axis_reader\n");
    return -1;
  }
  #ifdef DEBUG_MODE
    // If there is not error,
    //show the request_mem_region of ddr_to_axis_reader
    printk(KERN_INFO
"request_mem_region ddr_to_axis_reader DONE. \n");
  #endif


  // ddr_to_axis_reader address space is allocating
  ddr_to_axis_reader_reg =
    (void __iomem *)of_iomap(ddr_to_axis_reader_node, 0);
  if (!ddr_to_axis_reader_reg)
  {
    printk(KERN_INFO "could not allocate iomem\n");
    release_mem_region
            (ddr_to_axis_reader.start,
             resource_size(&ddr_to_axis_reader)
        );
    return -1;
  }
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"ddr_to_axis_reader registers are allocated. \n");
  #endif

  return 0;
}


////////////////// WAIT FUNCTION ////////////////////////////

void wait(void)
{
  volatile int i, Delay;
  for (i = 0; i < 50; i++)
```

```
  {
    for (Delay = 0; Delay < LED_DELAY; Delay++);
  }

}

////////////////// ov7670 Initialization ////////////////////
int initialize_ov7670(void)
{
  // response variable is used
  // for checking wheather process finish with correct or error
  int response = 0;
  #ifdef DEBUG_MODE
    printk(KERN_INFO "\n\n initialize_ov7670\n\n");
  #endif
  ov7670_device.interrupt_status = 0;
  ov7670_device.last_image_address = 0;
  ov7670_device.last_image_number = 0;
  ov7670_device.reset_pl_status = 0;
  // finding ov7670 node according to
  // name of ov7670 in devicetree.dts file
  ov7670_node =
(struct device_node *)of_find_node_by_name
(NULL, "axi_gpio_frame_intr");

  // creating resource for ov7670 with the help of ov7670 node
  response = of_address_to_resource(ov7670_node, 0, &ov7670);
  if (response < 0)
  {
    printk(KERN_INFO
    "error in of_address_to_resource ov7670\n");
    return -1;
  }
  #ifdef DEBUG_MODE
    printk(KERN_INFO
    "of_address_to_resource ov7670 is done.\n");
  #endif

  // checking wheather ov7670 address space is used or not
  if  (
        request_mem_region
            (ov7670.start,
             resource_size(&ov7670),
             "ov7670") == NULL
    )
  {
    printk(KERN_INFO
    "error in request_mem_region ov7670\n");
    return -1;
  }
  #ifdef DEBUG_MODE
    // If there is not error, show the request_mem_region of ov7670
    printk(KERN_INFO "request_mem_region ov7670 DONE. \n");
  #endif


  // ov7670 address space is allocating
  ov7670_reg = (void __iomem *)ioremap_nocache
    (ov7670.start, resource_size(&ov7670));
```

```
  if (!ov7670_reg)
  {
    printk(KERN_INFO "could not allocate iomem\n");
    release_mem_region(ov7670.start, resource_size(&ov7670));
    return -1;
  }
  #ifdef DEBUG_MODE
    printk(KERN_INFO "ov7670 registers are allocated. \n");
  #endif


  // ov7670 gpio ip is input
  reg_ptr_tmp = (void __iomem *)(ov7670_reg +  0x0004);
  iowrite8(255 , reg_ptr_tmp);
  //gpio tri-> this tell me the ov7670 are output
  #ifdef DEBUG_MODE
    printk(KERN_INFO "ov7670 are made outputs. \n");
  #endif


  return 0;

}

///////////////// Initialize all interrupts ////////////////////
int initialize_interrupts(void)
{
  // response variable is used
  // for checking wheather process finish with correct or error
  int response = 0;

  // initialize ov7670 interrupt
  response = initialize_ov7670_interrupt();
  if(response != 0)
  {
    return -1;
  }
  #ifdef DEBUG_MODE
    printk(KERN_INFO "Initialize ov7670 interrupt is done. \n");
  #endif

  return 0;
}
////////////////// Initialize ov7670 interrupts ////////////////////
int initialize_ov7670_interrupt(void)
{
  // response variable is used
  // for checking wheather process finish with correct or error
  int response = 0;

#ifdef  USE_LIKE_NESTED_INTERRUPT
  ov7670_interrupt_happening = false;

#endif

  // find ov7670 irq resource
  irq_ov7670 = of_irq_to_resource(ov7670_node, 0, &ov7670);
  if (irq_ov7670 == NO_IRQ)
```

```
  {
    printk(KERN_ERR "ov7670: of_irq_to_resource failed\n");
    release_mem_region(ov7670.start, resource_size(&ov7670));
    iounmap(ov7670_reg);
    of_node_put(ov7670_node);
    return -1;
  }
  #ifdef DEBUG_MODE
    printk(KERN_INFO "ov7670 interrupt resource found. \n");
  #endif

  // Enabling interrupt for ov7670 on the side of
  // processor and general interrupt controller
  // Connecting interrupt resource to interrupt handler
  response =
    request_threaded_irq(
        irq_ov7670,
        irq_default_primary_handler,
        ov7670_irq_handler ,
        IRQF_TRIGGER_RISING ,
        "ov7670_interrupt",
        &ov7670
    );

  if (response < 0)
  {
    printk(KERN_ERR
"unable to request IRQ %d : %d\n", irq_ov7670, response);

    free_irq(irq_ov7670, ov7670_reg);
    release_mem_region
    (ov7670.start, resource_size(&ov7670));
    iounmap(ov7670_reg);
    of_node_put(ov7670_node);
    return -1;
  }
  //irq_set_irq_type (irq_ov7670, IRQ_TYPE_EDGE_RISING);

  enable_frame_buffer_interrupt();

  return 0;
}
//////////////////////


////////////////// irq_default_primary_handler ///////////////
irqreturn_t irq_default_primary_handler(int irq, void *dev_id)
 {
         return IRQ_WAKE_THREAD;
 }

///////////// OV7670 interupt handler ////////////////////////
irqreturn_t ov7670_irq_handler(int irq, void *dev_id)
{
  // writing frame buffer number
  u32 frame_number, sendAble_frame_number;
  static  void __iomem *reg_ptr_tmp;
  // Buttons interrupt is cleared
  reg_ptr_tmp = (void __iomem *)(ov7670_reg +  0x0120);
```

```
  // GPIO IP interrupt clean register address
  iowrite32(1 , reg_ptr_tmp);
  //writing 1 to interrupt cleaning register

  #ifdef  USE_LIKE_NESTED_INTERRUPT
    ov7670_interrupt_happening = true;

  #endif
  reg_ptr_tmp = (void __iomem *)(ov7670_reg);
  // GPIO IP interrupt clean register address
  frame_number = ioread32(reg_ptr_tmp);
  if(frame_number == 0)
  {
    sendAble_frame_number = FRAME_BUFFER_NUM_WRITER -1;
  }
  else
  {
    sendAble_frame_number = frame_number -1;
  }
  ov7670_device.last_image_number = sendAble_frame_number;
  ov7670_device.last_image_address =
(char*)FRAME_BUFFER_BASE_ADDR_WRITER +
(FRAME_BUFFER_DIM_WRITER * sendAble_frame_number);

  #ifdef DEBUG_MODE
    printk(KERN_INFO
"fraNum=%d, send=%d \n", frame_number, sendAble_frame_number);
  #endif

  #ifdef  USE_LIKE_NESTED_INTERRUPT
    ov7670_interrupt_happening = false;

  #endif

  return IRQ_HANDLED;
}
//////////////////////////////////////////////////////////////////////

///////////////// release_ov7670 ////////////////////////////
void release_ov7670(void)
{
  release_mem_region  (
        ov7670.start,
        (unsigned long)resource_size(&ov7670)
    );
  iounmap(ov7670_reg);
  of_node_put(ov7670_node);
  free_irq(irq_ov7670, ov7670_reg);
}

//////////////////// initialize_pl_reset_ip ////////////////////
int initialize_pl_reset_ip(void)
{

  // response variable is used
  //for checking wheather process finish with correct or error
  int response = 0;
  #ifdef DEBUG_MODE
    printk(KERN_INFO "\n\n initialize_pl_reset_ip\n\n");
```

```
  #endif

  // finding pl_reset_ip node
  // according to name of pl_reset_ip in devicetree.dts file
  pl_reset_ip_node =  (struct device_node *)
    of_find_node_by_name(NULL, "axi_gpio_pl_reset");

  // creating resource for
  // pl_reset_ip with the help of pl_reset_ip node
  response = of_address_to_resource
               (pl_reset_ip_node, 0, &pl_reset_ip);
  if (response < 0)
  {
    printk(KERN_INFO
"error in of_address_to_resource pl_reset_ip\n");
    return 1;
  }
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"of_address_to_resource pl_reset_ip is done.\n");
  #endif

  // checking wheather pl_reset_ip address space is used or not
  if  (request_mem_region
           (pl_reset_ip.start,
            resource_size(&pl_reset_ip),
            "pl_reset_ip") == NULL
       )
  {
    printk(KERN_INFO "error in request_mem_region pl_reset_ip\n");
    return 1;
  }
  #ifdef DEBUG_MODE
    // If there is not error,
    // show the request_mem_region of pl_reset_ip
    printk(KERN_INFO "request_mem_region pl_reset_ip DONE. \n");
  #endif


  // pl_reset_ip address space is allocating
    pl_reset_ip_reg = (void __iomem *)ioremap
        (pl_reset_ip.start, resource_size(&pl_reset_ip));

  if (!pl_reset_ip_reg)
  {
    printk(KERN_INFO "could not allocate iomem\n");
    release_mem_region
(pl_reset_ip.start, resource_size(&pl_reset_ip));
    return 1;
  }
  #ifdef DEBUG_MODE
    printk(KERN_INFO "pl_reset_ip registers are allocated. \n");
  #endif

  // pl_reset_ip are made outputs
  reg_ptr_tmp = (void __iomem *)
  (pl_reset_ip_reg +  0x0004);
  iowrite8(0 , reg_ptr_tmp);
  //gpio tri-> this tell me the pl_reset_ip are output
```

```
  #ifdef DEBUG_MODE
    printk(KERN_INFO "pl_reset_ip are made outputs. \n");
  #endif

  initialize_pl_reset_ip_bool = true;

  return 0;

}

////////////////////////// pl_reset_enable ///////////////////////
int pl_reset_ip_enable(void)
{
  u32 data;
  if ( initialize_pl_reset_ip_bool )
  {
    pl_is_on_reset = 1;
    reg_ptr_tmp = (void __iomem *)(pl_reset_ip_reg);
    iowrite32(RESET_ENABLED, reg_ptr_tmp);
    //gpio tri -> this tell me the pl_reset_ip are output
    reg_ptr_tmp =
            (void __iomem *)ioremap_nocache(
                pl_reset_ip.start,
                resource_size(&pl_reset_ip)
            );
    if (!pl_reset_ip_reg)
    {
      return 1;
    }
    data = ioread32(pl_reset_ip_reg);
    printk(KERN_INFO "pl_reset_ip_enable data = %u \n", data);
    ov7670_device.reset_pl_status = 1;
    return 0;
  }
  else return 1;
}

////////////////////////// pl_reset_disable ///////////////////////
int pl_reset_ip_disable(void)
{
  if ( initialize_pl_reset_ip_bool )
  {
    u32 data;
    reg_ptr_tmp = (void __iomem *)(pl_reset_ip_reg);
    iowrite32(RESET_DISABLED, reg_ptr_tmp);
    //gpio tri -> this tell me the pl_reset_ip are output
    pl_is_on_reset = 0;
    data = ioread32(reg_ptr_tmp);
    printk(KERN_INFO "pl_reset_ip_disable data = %u \n", data);
    ov7670_device.reset_pl_status = 0;
    return 0;
  }
  else return 1;
}

////////////////////////// pl_reset_keep_for ///////////////////////
int pl_reset_ip_keep_milisecond(int msec)
{
  if (initialize_pl_reset_ip_bool)
```

```
  {
    pl_reset_ip_enable();
    msleep(msec);
    pl_reset_ip_disable();
    return 0;
  }
  else return 1;
}

//////////////////// is_pl_on_reset /////////////////////
int is_pl_reset_ip_on_reset(void)
{
  return pl_is_on_reset;
}

////////////////////// release_pl_reset_ip ////////////////////
void release_pl_reset_ip(void)
{
  release_mem_region(
        pl_reset_ip.start,
        (unsigned long)resource_size(&pl_reset_ip)
    );
  iounmap(pl_reset_ip_reg);
  of_node_put(pl_reset_ip_node);
}

int start_application(void)
{
  // response variable is used
  //for checking wheather process finish with correct or error
  int response = 0;

  // Reset PL
  //response = pl_reset_ip_keep_milisecond(1000);
  if(response != 0)
  {
    return -1;
  }
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"pl_reset_ip_keep_milisecond is done.\n");
  #endif

  // Configure WRITER
  response = Configure_axis_to_ddr_writer();
  if(response != 0)
  {
    return -1;
  }
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"Configure_axis_to_ddr_writer is done.\n");
  #endif

  // Configure READER
  response = Configure_ddr_to_axis_reader();
  if(response != 0)
  {
    return -1;
```

```
  }
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"Configure_ddr_to_axis_reader is done.\n");
  #endif

  // initialize all used peripherals
  response = initialize_interrupts();
  if(response != 0)
  {
    return -1;
  }
  #ifdef DEBUG_MODE
    printk(KERN_ERR
"\nInitializations of all interrupts are done.\n");
  #endif
  return 0;
}

int Configure_axis_to_ddr_writer(void)
{
  #ifdef DEBUG_MODE
    printk(KERN_INFO "\n\nConfigure_axis_to_ddr_writer\n\n");
  #endif
  axis_to_ddr_writer_Set_base_address
    (axis_to_ddr_writer_reg, FRAME_BUFFER_BASE_ADDR_WRITER);
  axis_to_ddr_writer_Set_frame_buffer_dim
    (axis_to_ddr_writer_reg, FRAME_BUFFER_DIM_WRITER);
  axis_to_ddr_writer_Set_frame_buffer_number
    (axis_to_ddr_writer_reg, FRAME_BUFFER_NUM_WRITER);
  axis_to_ddr_writer_Set_frame_buffer_offset
    (axis_to_ddr_writer_reg, FRAME_BUFFER_DIM_WRITER);
  axis_to_ddr_writer_EnableAutoRestart
    (axis_to_ddr_writer_reg, 0x80);
  axis_to_ddr_writer_Set_update_intr
    (axis_to_ddr_writer_reg, 1);
  axis_to_ddr_writer_Start
    (axis_to_ddr_writer_reg, 0);
  return 0;
}
void axis_to_ddr_writer_Set_base_address
(void __iomem *int_ptr, u32 data)
{
  // axis_to_ddr_writer_Set_base_address
  reg_ptr_tmp = (void __iomem *)
    (int_ptr +
    XAXIS_TO_DDR_WRITER_AXILITES_ADDR_BASE_ADDRESS_DATA);
  iowrite32(data , reg_ptr_tmp);
  ov7670_device.image_writer_base_address = data;
  //gpio tri-> this tell me the ov7670 are output
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"axis_to_ddr_writer_Set_base_address %x\n", data);
  #endif
  read_data = ioread32(reg_ptr_tmp);
  printk(KERN_INFO
"axis_to_ddr_writer_Set_base_address read_data = %x\n",
    read_data);
}
```

```
void axis_to_ddr_writer_Set_frame_buffer_dim
(void __iomem *int_ptr, u32 data)
{
  // axis_to_ddr_writer_Set_frame_buffer_dim
  reg_ptr_tmp = (void __iomem *)
  (int_ptr +
  XAXIS_TO_DDR_WRITER_AXILITES_ADDR_FRAME_BUFFER_DIM_DATA);
  iowrite32(data , reg_ptr_tmp);
  //gpio tri-> this tell me the ov7670 are output
  #ifdef DEBUG_MODE
    printk(KERN_INFO
    "axis_to_ddr_writer_Set_frame_buffer_dim %x\n", data);
  #endif
  read_data = ioread32(reg_ptr_tmp);
  printk(KERN_INFO
  "axis_to_ddr_writer_Set_frame_buffer_dim read_data = %x\n",
  read_data);
}
void axis_to_ddr_writer_Set_frame_buffer_number
(void __iomem *int_ptr, u32 data)
{
  // axis_to_ddr_writer_Set_frame_buffer_number
  reg_ptr_tmp = (void __iomem *)
    (int_ptr +
    XAXIS_TO_DDR_WRITER_AXILITES_ADDR_FRAME_BUFFER_NUMBER_DATA);
  iowrite32(data , reg_ptr_tmp);
  //gpio tri-> this tell me the ov7670 are output
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"axis_to_ddr_writer_Set_frame_buffer_number %x\n", data);
  #endif
  read_data = ioread32(reg_ptr_tmp);
  printk(KERN_INFO
"axis_to_ddr_writer_Set_frame_buffer_number read_data = %x\n",
  read_data);
}
void axis_to_ddr_writer_Set_frame_buffer_offset
(void __iomem *int_ptr, u32 data)
{
  // axis_to_ddr_writer_Set_frame_buffer_offset
  reg_ptr_tmp = (void __iomem *)
    (int_ptr +
    XAXIS_TO_DDR_WRITER_AXILITES_ADDR_FRAME_BUFFER_OFFSET_DATA);
  iowrite32(data , reg_ptr_tmp);
  //gpio tri-> this tell me the ov7670 are output
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"axis_to_ddr_writer_Set_frame_buffer_offset %x\n", data);
  #endif
  read_data = ioread32(reg_ptr_tmp);
  printk(KERN_INFO
"axis_to_ddr_writer_Set_frame_buffer_offset read_data = %x\n",
    read_data);
}
void axis_to_ddr_writer_EnableAutoRestart
(void __iomem *int_ptr, u32 data)
{
  // axis_to_ddr_writer_EnableAutoRestart
  reg_ptr_tmp = (void __iomem *)
```

```
    (int_ptr +  XAXIS_TO_DDR_WRITER_AXILITES_ADDR_AP_CTRL);
  iowrite32(data , reg_ptr_tmp);
  //gpio tri-> this tell me the ov7670 are output
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"axis_to_ddr_writer_EnableAutoRestart %x\n", data);
  #endif
  read_data = ioread32(reg_ptr_tmp);
  printk(KERN_INFO
"axis_to_ddr_writer_EnableAutoRestart read_data = %x\n",
    read_data);
}
void axis_to_ddr_writer_Set_update_intr
(void __iomem *int_ptr, u32 data)
{
  // axis_to_ddr_writer_Set_update_intr
  reg_ptr_tmp = (void __iomem *)
    (int_ptr +
    XAXIS_TO_DDR_WRITER_AXILITES_ADDR_UPDATE_INTR_DATA);
  iowrite32(data , reg_ptr_tmp);
  //gpio tri-> this tell me the ov7670 are output
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"axis_to_ddr_writer_Set_update_intr %x\n", data);
  #endif
  read_data = ioread32(reg_ptr_tmp);
  printk(KERN_INFO
"axis_to_ddr_writer_Set_update_intr read_data = %x\n", read_data);
}
void axis_to_ddr_writer_Start
(void __iomem *int_ptr, u32 data)
{
  u32 temp;
  // axis_to_ddr_writer_Start
  reg_ptr_tmp = (void __iomem *)
    (int_ptr +  XAXIS_TO_DDR_WRITER_AXILITES_ADDR_AP_CTRL);
  temp = ioread32(reg_ptr_tmp) & 0x80;
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"TEMP axis_to_ddr_writer temp = %x\n", temp);
  #endif
  data = temp |0x01;
  iowrite32(data, reg_ptr_tmp);
  //gpio tri-> this tell me the ov7670 are output
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"axis_to_ddr_writer_Start %x\n", data);
  #endif
  read_data = ioread32(reg_ptr_tmp);
  printk(KERN_INFO
"axis_to_ddr_writer_Start read_data = %x\n", read_data);
}

int Configure_ddr_to_axis_reader(void)
{
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"\n\nConfigure_ddr_to_axis_reader\n\n");
  #endif
```

```
  ddr_to_axis_reader_Set_base_address
(ddr_to_axis_reader_reg, FRAME_BUFFER_BASE_ADDR_READER);
  ddr_to_axis_reader_Set_frame_buffer_dim
(ddr_to_axis_reader_reg, FRAME_BUFFER_DIM_READER);
  ddr_to_axis_reader_Set_frame_buffer_number
(ddr_to_axis_reader_reg, FRAME_BUFFER_NUM_READER);
  ddr_to_axis_reader_Set_frame_buffer_offset
(ddr_to_axis_reader_reg, FRAME_BUFFER_DIM_READER);
  ddr_to_axis_reader_Set_update_intr
(ddr_to_axis_reader_reg, 0x01);
  ddr_to_axis_reader_EnableAutoRestart
(ddr_to_axis_reader_reg, 0x80);
  ddr_to_axis_reader_Start
(ddr_to_axis_reader_reg, 0);
  return 0;
}
void ddr_to_axis_reader_Set_base_address
(void __iomem *int_ptr, u32 data)
{
  // axis_to_ddr_writer_Set_base_address
  reg_ptr_tmp = (void __iomem *)
    (int_ptr +
    XDDR_TO_AXIS_READER_AXILITES_ADDR_BASE_ADDRESS_DATA);
  iowrite32(data , reg_ptr_tmp);
  ov7670_device.image_reader_base_address = data;
  //gpio tri -> this tell me the ov7670 are output
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"ddr_to_axis_reader_Set_base_address %x\n", data);
  #endif
  read_data = ioread32(reg_ptr_tmp);
  printk(KERN_INFO
"ddr_to_axis_reader_Set_base_address read_data = %x\n",
    read_data);
}
void ddr_to_axis_reader_Set_frame_buffer_dim
(void __iomem *int_ptr, u32 data)
{
  // axis_to_ddr_writer_Set_base_address
  reg_ptr_tmp = (void __iomem *)
    (int_ptr +
    XDDR_TO_AXIS_READER_AXILITES_ADDR_FRAME_BUFFER_DIM_DATA);
  iowrite32(data , reg_ptr_tmp);
  //gpio tri -> this tell me the ov7670 are output
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"ddr_to_axis_reader_Set_frame_buffer_dim %x\n", data);
  #endif
  read_data = ioread32(reg_ptr_tmp);
  printk(KERN_INFO
"ddr_to_axis_reader_Set_frame_buffer_dim read_data = %x\n",
    read_data);

}
void ddr_to_axis_reader_Set_frame_buffer_number
(void __iomem *int_ptr, u32 data)
{
  // axis_to_ddr_writer_Set_base_address
  reg_ptr_tmp = (void __iomem *)
```

```
    (int_ptr +
    XDDR_TO_AXIS_READER_AXILITES_ADDR_FRAME_BUFFER_NUMBER_DATA);
  iowrite32(data , reg_ptr_tmp);
  //gpio tri-> this tell me the ov7670 are output
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"ddr_to_axis_reader_Set_frame_buffer_number %x\n", data);
  #endif
  read_data = ioread32(reg_ptr_tmp);
  printk(KERN_INFO
"ddr_to_axis_reader_Set_frame_buffer_number read_data = %x\n",
    read_data);

}
void ddr_to_axis_reader_Set_frame_buffer_offset
(void __iomem *int_ptr, u32 data)
{
  // axis_to_ddr_writer_Set_base_address
  reg_ptr_tmp = (void __iomem *)
    (int_ptr +
    XDDR_TO_AXIS_READER_AXILITES_ADDR_FRAME_BUFFER_OFFSET_DATA);
  iowrite32(data , reg_ptr_tmp);
  //gpio tri-> this tell me the ov7670 are output
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"ddr_to_axis_reader_Set_frame_buffer_offset %x\n", data);
  #endif
  read_data = ioread32(reg_ptr_tmp);
  printk(KERN_INFO
"ddr_to_axis_reader_Set_frame_buffer_offset read_data = %x\n",
    read_data);

}
void ddr_to_axis_reader_Set_update_intr
(void __iomem *int_ptr, u32 data)
{
  // axis_to_ddr_writer_Set_base_address
  reg_ptr_tmp = (void __iomem *)
    (int_ptr +
    XDDR_TO_AXIS_READER_AXILITES_ADDR_UPDATE_INTR_DATA);
  iowrite32(data , reg_ptr_tmp);
  //gpio tri-> this tell me the ov7670 are output
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"ddr_to_axis_reader_Set_update_intr %x\n", data);
  #endif
  read_data = ioread32(reg_ptr_tmp);
  printk(KERN_INFO
"ddr_to_axis_reader_Set_update_intr read_data = %x\n",
    read_data);

}
void ddr_to_axis_reader_EnableAutoRestart
(void __iomem *int_ptr, u32 data)
{
  // axis_to_ddr_writer_Set_base_address
  reg_ptr_tmp = (void __iomem *)
(int_ptr +  XDDR_TO_AXIS_READER_AXILITES_ADDR_AP_CTRL);
  iowrite32(data , reg_ptr_tmp);
```

```
  //gpio tri -> this tell me the ov7670 are output
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"ddr_to_axis_reader_EnableAutoRestart %x\n", data);
  #endif
  read_data = ioread32(reg_ptr_tmp);
  printk(KERN_INFO
"ddr_to_axis_reader_EnableAutoRestart read_data = %x\n",
    read_data);

}
void ddr_to_axis_reader_Start
(void __iomem *int_ptr, u32 data)
{
  u32 temp;
  // axis_to_ddr_writer_Start
  reg_ptr_tmp = (void __iomem *)
(int_ptr +  XDDR_TO_AXIS_READER_AXILITES_ADDR_AP_CTRL);
  temp = ioread32(reg_ptr_tmp) & 0x80;
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"TEMP ddr_to_axis_reader_Start temp = %x\n", temp);
  #endif
  data = temp |0x01;
  iowrite32(data, reg_ptr_tmp);
  //gpio tri -> this tell me the ov7670 are output
  #ifdef DEBUG_MODE
    printk(KERN_INFO
"ddr_to_axis_reader_Start %x\n", data);
  #endif
  read_data = ioread32(reg_ptr_tmp);
  printk(KERN_INFO
"ddr_to_axis_reader_Start read_data = %x\n", read_data);

}


void stop_application(void)
{

}

static int device_open
(struct inode *inode, struct file *filp) {
  /*
  if(down_interruptible(&ov7670_device.sem) != 0) {
    printk(KERN_ALERT
    "ov7670_driver : could not lock device
    during open \n");
    return -1;
  }
  */
  //printk(KERN_ALERT "ov7670_driver : opened device \n");
  return 0;
}
static ssize_t device_read
    (struct file *filp,
    char* bufStoreData,
    size_t bufCount,
```

```
    loff_t* curOffset)
{

  printk(KERN_ALERT
"ov7670_driver : reading from device \n");
  printk(KERN_ALERT
"writer_base : %x \n", ov7670_device.image_writer_base_address);
  printk(KERN_ALERT
"reader_base : %x \n", ov7670_device.image_reader_base_address);
  ret = copy_to_user
(bufStoreData, &ov7670_device, bufCount);
  return ret;
}

static ssize_t device_write
    (struct file *filp,
    const char* bufSourceData,
    size_t bufCount,
    loff_t* curOffset)
{
  struct character_device temp_device;
  //printk(KERN_ALERT "ov7670_driver :
  //writing to device \n");
  if(sizeof(temp_device) != bufCount) {
    printk(KERN_ALERT
"ov7670_driver : writing to device failed!
Please be sure your data size is correct. \n");
  }
  ret = copy_from_user
(&temp_device + *curOffset, bufSourceData, bufCount);
  if(ret < 0) {
    printk(KERN_ALERT
"ov7670_driver : writing to device failed!
Please be sure your data is correct.
ret = %d, bufCount = %d\n", ret, bufCount);
  }
  reset_pl_status_control(temp_device);
  interrupt_status_control(temp_device);
  writer_base_addr_control(temp_device);
  reader_base_addr_control(temp_device);
  //temp_device.reset_pl_status;

  return ret;
}

void interrupt_status_control
(struct character_device temp_device){
  if(temp_device.interrupt_status) {
    enable_frame_buffer_interrupt();
  }
  else{
    disable_frame_buffer_interrupt();
  }
}
void enable_frame_buffer_interrupt(void){
  //Enable ov7670 to generate interrupts
  //writing in their own register
  reg_ptr_tmp = (void __iomem *)(ov7670_reg +  0x0128);
  iowrite32(255 , reg_ptr_tmp);
```

```
  // GPIO Ip interrupt enable register.
  // Maybe here that is needed*(u32 *)
  reg_ptr_tmp = (void __iomem *)(ov7670_reg +  0x011C);
  iowrite32(0xffffffff , reg_ptr_tmp);
  // GPIO Ip global interrupt enable register
  ov7670_device.interrupt_status = 1;
}
void disable_frame_buffer_interrupt(void){
  //disable ov7670 to generate interrupts
  // writing in their own register
  reg_ptr_tmp = (void __iomem *)(ov7670_reg +  0x0128);
  iowrite32(0 , reg_ptr_tmp);
  // GPIO Ip interrupt disable register.
  // Maybe here that is needed*(u32 *)
  reg_ptr_tmp = (void __iomem *)(ov7670_reg +  0x011C);
  iowrite32(0x00000000 , reg_ptr_tmp);
  // GPIO Ip global interrupt disable register
  ov7670_device.interrupt_status = 0;
}

void reset_pl_status_control
(struct character_device temp_device){
  if(temp_device.reset_pl_status) {
    if(!ov7670_device.reset_pl_status) {
      pl_reset_ip_enable();
    }
  }
  else{
    if(ov7670_device.reset_pl_status) {
      pl_reset_ip_disable();
      initialization_after_reset();
    }
  }
}

void writer_base_addr_control
(struct character_device temp_device){
  axis_to_ddr_writer_Set_base_address
    (axis_to_ddr_writer_reg,
    temp_device.image_writer_base_address);
}

void reader_base_addr_control
(struct character_device temp_device){
  ddr_to_axis_reader_Set_base_address
    (ddr_to_axis_reader_reg,
    temp_device.image_reader_base_address);
}

static int device_close
(struct inode *inode, struct file *filp) {
  //up(&ov7670_device.sem);
  //printk(KERN_ALERT "ov7670_driver :
  //closed device \n");
  return 0;
}

static struct file_operations fops = {
  .open = device_open,
```

```
  .release = device_close,
  .write = device_write,
  .read = device_read
};

 int driver_entry(void) {

  ret = alloc_chrdev_region
  (&dev_num, 0, 1, DEVICE_NAME);
  if(ret < 0) {
    printk(KERN_ALERT
"ov7670_driver : failed to allocate a major number \n");
    return ret;
  }
  major_number = MAJOR(dev_num);
  printk(KERN_ALERT
"ov7670_driver : major number is = %d \n",major_number);
  printk(KERN_ALERT
"ov7670_driver : \tuse
\"mknod /dev/%s c %d 0\" for device file \n",
DEVICE_NAME, major_number);

  mcdev = cdev_alloc();
  // create our cdev structure,
  // initialized our cdev
  mcdev->ops =&fops;
  mcdev->owner = THIS_MODULE;

  ret = cdev_add(mcdev, dev_num, 1);
  if(ret < 0) {
    printk(KERN_ALERT
"ov7670_driver : unable to add cdev to kernel \n");
    return ret;
  }

  return 0;
}

 int initialization_after_reset(void) {
  // Configure WRITER
  Configure_axis_to_ddr_writer();
  // Configure READER
  Configure_ddr_to_axis_reader();

    return 0;
 }

////////////////////////////////////////////////////////////////////
MODULE_LICENSE("GPL");
MODULE_AUTHOR ("Utku Esen");
MODULE_DESCRIPTION("OV7670 Linux platform driver for ZedBoard");
module_init(ov7670_driver_init);
module_exit(ov7670_driver_exit);
////////////////////////////////////////////////////////////////////
```

### D.1.2: I2C Camera Configuration User Space Application for Linux

```
/*************************************************************/
/**
* @file i2c_test.c
* This file contains the linux example for i2c.
* Note:    10-bit addressing is not supported
* in the current linux driver.
* June 26, 2018
* Repeated start also not supported in the current linux driver.
* @note      None.
*************************************************************/
/***************************** Include Files ****************/
#include <fcntl.h>
#include <stdio.h>
#include <linux/i2c-dev.h>
#include <errno.h>
#include <string.h>
#include <sys/time.h>
/************************** Constant Definitions ************/
#define OV7670_SLAVE_ADDRESS 0x21
#define OV7670_SLAVE_ADDRESS_WRITE 0x42
#define OV7670_SLAVE_ADDRESS_READ 0x43
#define I2C_SLAVE_FORCE 0x0706
#define I2C_SLAVE_ADDRESS 0x0703     /* Change slave address */
#define I2C_FUNCS 0x0705 /* Get the adapter functionality */
#define I2C_RDWR  0x0707/* Combined R/W transfer (one stop only)*/
#define COM7 0x12
#define COM7_VALUE_RESET        0x80
#define MUX_ADDR                0x74
#define EEPROM_ADDR             0x54
#define MUX_CHANNEL_KINTEX7     0x04
#define PAGESIZE        16
#define DATA_REGISTER_CHANGED 1
#define DATA_REGISTER_NOT_CHANGED 0
#define INPUT_CLOCK_FREQUENCY 24//MHz
#define INTERNAL_CLOCK_FREQUENCY 24 //MHz
#define OV7670_SLAVE_ADDRESS 0x21
#define OV7670_SLAVE_ADDRESS_WRITE 0x42
#define OV7670_SLAVE_ADDRESS_READ 0x43
#define AEW 0x24
#define AEW_VALUE 0x95
#define AEB 0x25
#define AEB_VALUE 0x33
#define VPT 0x26
#define VPT_VALUE 0xE3
#define HAECC77 0xAA
#define HAECC77_VALUE_HISTOGRAM_AEC_ON 0x94
#define HAECC77_VALUE_AVERAGE_AEC_ON 0x00
#define CLKRC 0x11
#define CLKRC_VALUE_VGA 0x01
#define CLKRC_VALUE_NIGHTMODE_AUTO 0x80
#define COM7 0x12
#define COM7_VALUE_VGA 0x01
#define COM7_VALUE_VGA_COLOR_BAR 0x03
#define COM7_VALUE_QVGA 0x00
#define COM7_VALUE_RESET 0x80
```

```
#define COM3 0x0C
#define COM3_VALUE_VGA 0x00
#define COM3_VALUE_QVGA 0x04
#define COM14 0x3E
#define COM14_VALUE_VGA 0x00
#define SCALING_XSC 0x70
#define SCALING_XSC_VALUE_VGA 0x3A
#define SCALING_YSC 0x71
#define SCALING_YSC_VALUE_VGA 0x35
#define SCALING_DCWCTR 0x72
#define SCALING_DCWCTR_VALUE_VGA 0x11
#define SCALING_PCLK_DIV 0x73
#define SCALING_PCLK_DIV_VALUE_VGA 0xF0
#define SCALING_PCLK_DELAY 0xA2
#define SCALING_PCLK_DELAY_VALUE_VGA 0x02
#define COM17 0x42
#define COM17_VALUE_AEC_NORMAL_NO_COLOR_BAR 0x00
#define COM17_VALUE_AEC_NORMAL_COLOR_BAR 0x08
#define DBLV 0x6B
#define DBLV_VALUE_30FPS 0x0A
#define EXHCH 0x2A
#define EXHCH_VALUE_30FPS 0x00
#define EXHCL 0x2B
#define EXHCL_VALUE_30FPS 0x00
#define DM_LNL 0x92
#define DM_LNL_VALUE_30FPS 0x00
#define DM_LNH 0x93
#define DM_LNH_VALUE_30FPS 0x00
#define COM11 0x3B
#define COM11_VALUE_30_FPS 0x0A
#define COM11_VALUE_NIGHTMODE_AUTO 0xEA //Auto frame adjust
#define TSLB 0x3A
#define TSLB_YUYV 0x05
#define TSLB_YVYU 0x05
#define TSLB_UYVY 0x0D
#define TSLB_VYUY 0x0D
#define COM13 0x3D
#define COM13_YUYV 0x80
#define COM13_YVYU 0x89
#define COM13_UYVY 0x80
#define COM13_VYUY 0x89
#define COM16 0x41
#define COM16_VALUE_DENOISE_ON_EDGE_ENHANCEMENT_ON 0x38
#define SATCTR 0xC9
#define SATCTR_VALUE 0xc0 // 0xc0
#define HSTART 0x17
#define HSTART_VALUE_DEFAULT 0x11
#define HSTART_VALUE_VGA 0x13 //0x14
#define HSTOP 0x18
#define HSTOP_VALUE_DEFAULT 0x61
#define HSTOP_VALUE_VGA 0x01 //0x02
#define HREF 0x32
#define HREF_VALUE_DEFAULT 0x80
#define HREF_VALUE_VGA  0xb6 //0x24
#define VSTRT 0x19
#define VSTRT_VALUE_DEFAULT 0x03
#define VSTRT_VALUE_VGA 0x02 //0x03
#define VSTOP 0x1A
#define VSTOP_VALUE_DEFAULT 0x7B
```

```
#define VSTOP_VALUE_VGA 0x7a //0x7B
#define VREF 0x03
#define VREF_VALUE_DEFAULT 0x03
#define VREF_VALUE_VGA 0x0A
#define ABLC1 0xB1
#define ABLC1_VALUE 0x0C
#define THL_ST 0xB3
#define THL_ST_VALUE 0x82
#define WAITING_TIME 5000
/**************************** Type Definitions *************/
typedef unsigned char   Xuint8;
typedef char            Xint8;
/**< signed 8-bit */
typedef unsigned short  Xuint16;
/**< unsigned 16-bit */
typedef short           Xint16;
/**< signed 16-bit */
typedef unsigned long   Xuint32;
/**< unsigned 32-bit */
typedef long            Xint32;
/**< signed 32-bit */
typedef float           Xfloat32;
/**< 32-bit floating point */
typedef double          Xfloat64;
/**< 64-bit double precision floating point */
typedef unsigned long   Xboolean;
/**< boolean (XTRUE or XFALSE) */
typedef  Xuint8 AddressType;

/********* Macros (Inline Functions) Definitions *************/

/***************** Function Prototypes ******************/

static int IicConfigure(void);
int OV7670_reset_all_registers(void);
int OV7670_prescale_clock_and_PLL_control(void);
int OV7670_set_VGA_resolution_and_YUV_output(void);
int OV7670_change_order_for_YUV(void);
int OV7670_set_other_registers(void);
int OV7670_set_window_output_registers(void);
/************** Variable Definitions ******************/

/*
 * FD of the IIC device opened.
 */
int Fdiic;
Xuint8 BytesWritten;
Xuint8 WriteBuffer[2];

/************** Function Definitions ******************/

/*****************************************************/
/**
*
* Entry point for integration tests on IIC.
*
* @param     .
*
* @return    0 if successful else -1.
```

```
*
* @note      None.
*
*****************************************************/
int main()
{
    int Status;
    int TestLoops=1;
    // Open the device
    Fdiic = open("/dev/i2c-0", O_RDWR);
    if(Fdiic < 0)
    {
        printf("Cannot open the IIC device\n");

        return 1;
    }
  printf("IIC device file opened\n");
    Status = IicConfigure();
     if (Status)
    {
        printf("Cannot configured OV7670\n");
        close(Fdiic);
        return 1;
    }
    printf("OV7670 config successfull \n");
    close(Fdiic);

    return 0;
}

static int IicConfigure(void)
{
      /* Buffer to hold location address.*/

    int addr = OV7670_SLAVE_ADDRESS;
  if (ioctl(Fdiic, I2C_SLAVE_FORCE, addr) < 0)
    {
        printf("\n Unable to set the ov7670 address\n");
      printf(" %s\n", strerror(errno));
        return -1;
    }
  usleep(WAITING_TIME);
  OV7670_reset_all_registers();
  printf("\n Video quality must be bad \n");
  printf("\n OV7670_reset_all_registers done.
    Press a button.\n");
  OV7670_set_VGA_resolution_and_YUV_output();
  printf("\n OV7670_set_VGA_resolution_and_YUV_output done.
     Press a button.\n");
  OV7670_change_order_for_YUV();
  printf("\n OV7670_change_order_for_YUV done.
    Press a button.\n");
  OV7670_set_other_registers();
  printf("\n OV7670_set_other_registers done.
    Press a button.\n");
  OV7670_set_window_output_registers();
  printf("\n OV7670_set_window_output_registers done.
    Press a button to finish application.\n");
    return 0;
```

```
}

int OV7670_reset_all_registers(void)
{
  /*
   * Position the address pointer in EEPROM.
   */
  WriteBuffer[0] = COM7;
  WriteBuffer[1] = COM7_VALUE_RESET;
  BytesWritten = write(Fdiic, WriteBuffer, 2);
  if(BytesWritten != 2)
    {
      printf(
"Error COM7,  COM7_VALUE_RESET BytesWritten = %u\n",
             BytesWritten);
      printf(" %s\n", strerror(errno));
      return -1;
    }
  else
  {
    printf("COM7, COM7_VALUE_RESET Done\n");
  }
  usleep(WAITING_TIME);

  return 0;
}

int OV7670_prescale_clock_and_PLL_control(void)
{
  return 0;
}

int OV7670_set_VGA_resolution_and_YUV_output(void)
{
  WriteBuffer[0] = CLKRC;
  WriteBuffer[1] = 0x80;
  BytesWritten = write(Fdiic, WriteBuffer, 2);
  if(BytesWritten != 2)
    {
      printf("Error CLKRC, 0x80\n");
      printf(" %s\n", strerror(errno));
      return -1;
    }
    else
    {
      printf("CLKRC, 0x80 Done\n");
    }
  usleep(WAITING_TIME);
  WriteBuffer[0] = COM7;
  WriteBuffer[1] = 0x00;
  BytesWritten = write(Fdiic, WriteBuffer, 2);
  if(BytesWritten != 2)
    {
      printf("Error COM7, 0x00\n");
      printf(" %s\n", strerror(errno));
      return -1;
    }
    else
    {
```

```
    printf("COM7, 0x00 Done\n");
  }
usleep(WAITING_TIME);
WriteBuffer[0] = COM3;
WriteBuffer[1] = 0x00;
BytesWritten = write(Fdiic, WriteBuffer, 2);
if(BytesWritten != 2)
  {
    printf("Error COM3, 0x00\n");
    printf(" %s\n", strerror(errno));
    return -1;
  }
  else
  {
    printf("COM3, 0x00 \n");
  }
usleep(WAITING_TIME);
WriteBuffer[0] = COM14;
WriteBuffer[1] = 0x00;
BytesWritten = write(Fdiic, WriteBuffer, 2);
if(BytesWritten != 2)
  {
    printf("Error COM14, 0x00\n");
    printf(" %s\n", strerror(errno));
    return -1;
  }
  else
  {
    printf("COM14, 0x00 \n");
  }
usleep(WAITING_TIME);
WriteBuffer[0] = SCALING_XSC;
WriteBuffer[1] = SCALING_XSC_VALUE_VGA;
BytesWritten = write(Fdiic, WriteBuffer, 2);
if(BytesWritten != 2)
  {
    printf("Error SCALING_XSC, SCALING_XSC_VALUE_VGA\n");
    printf(" %s\n", strerror(errno));
    return -1;
  }
  else
  {
    printf("SCALING_XSC, SCALING_XSC_VALUE_VGA \n");
  }
usleep(WAITING_TIME);
WriteBuffer[0] = SCALING_YSC;
WriteBuffer[1] = SCALING_YSC_VALUE_VGA;
BytesWritten = write(Fdiic, WriteBuffer, 2);
if(BytesWritten != 2)
  {
    printf("Error SCALING_YSC, SCALING_YSC_VALUE_VGA\n");
    printf(" %s\n", strerror(errno));
    return -1;
  }
  else
  {
    printf("SCALING_YSC, SCALING_YSC_VALUE_VGA \n");
  }
usleep(WAITING_TIME);
```

```
  WriteBuffer[0] = SCALING_DCWCTR;
  WriteBuffer[1] = SCALING_DCWCTR_VALUE_VGA;
  BytesWritten = write(Fdiic, WriteBuffer, 2);
  if(BytesWritten != 2)
    {
      printf("Error SCALING_DCWCTR, SCALING_DCWCTR_VALUE_VGA\n");
      printf(" %s\n", strerror(errno));
      return -1;
    }
    else
    {
      printf("SCALING_DCWCTR, SCALING_DCWCTR_VALUE_VGA \n");
    }
  usleep(WAITING_TIME);
  WriteBuffer[0] = SCALING_PCLK_DIV;
  WriteBuffer[1] = 0xF0;
  BytesWritten = write(Fdiic, WriteBuffer, 2);
  if(BytesWritten != 2)
    {
      printf("Error SCALING_PCLK_DIV, 0xF0\n");
      printf(" %s\n", strerror(errno));
      return -1;
    }
    else
    {
      printf("SCALING_PCLK_DIV, 0xF0 \n");
    }
  usleep(WAITING_TIME);
  printf("\n Video quality must be change \n");
  printf("\n SCALING_PCLK_DIV done.Press a button.\n");
  //getchar();
  WriteBuffer[0] = SCALING_PCLK_DELAY;
  WriteBuffer[1] = SCALING_PCLK_DELAY_VALUE_VGA;
  BytesWritten = write(Fdiic, WriteBuffer, 2);
  if(BytesWritten != 2)
    {
      printf(
            "Error SCALING_PCLK_DELAY, SCALING_PCLK_DELAY_VALUE_VGA\n"
            );
      printf(" %s\n", strerror(errno));
      return -1;
    }
    else
    {
      printf(
            "SCALING_PCLK_DELAY, SCALING_PCLK_DELAY_VALUE_VGA \n"
            );
    }
  usleep(WAITING_TIME);
  return 0;
}

int OV7670_change_order_for_YUV(void)
{
  WriteBuffer[0] = TSLB;
  WriteBuffer[1] = TSLB_YUYV;
  BytesWritten = write(Fdiic, WriteBuffer, 2);
  if(BytesWritten != 2)
    {
```

```
      printf("Error asd TSLB, TSLB_YUYV\n");
      printf(" %s\n", strerror(errno));
      return -1;
    }
    else
    {
      printf("asd TSLB, TSLB_YUYV\n");
    }
  usleep(WAITING_TIME);
  printf("\n Video quality must be change \n");
  printf("\n TSLB_YUYV done.Press a button.\n");
  //getchar();
  WriteBuffer[0] = COM13;
  WriteBuffer[1] = 0x80;
  BytesWritten = write(Fdiic, WriteBuffer, 2);
  if(BytesWritten != 2)
    {
      printf("Error COM13, 0x80\n");
      printf(" %s\n", strerror(errno));
      return -1;
    }
    else
    {
      printf("COM13, 0x80 \n");
    }
  // here quality must be high
  usleep(WAITING_TIME);
  return 0;
}

int OV7670_set_other_registers(void)
{
  WriteBuffer[0] = EXHCH;
      WriteBuffer[1] = EXHCH_VALUE_30FPS;
      BytesWritten = write(Fdiic, WriteBuffer, 2);
      if(BytesWritten != 2)
          {
              printf("Error EXHCH, EXHCH_VALUE_30FPS\n");
        printf(" %s\n", strerror(errno));
              return -1;
          }
          else
          {
            printf("EXHCH, EXHCH_VALUE_30FPS \n");
          }
    usleep(WAITING_TIME);
    WriteBuffer[0] = EXHCL;
      WriteBuffer[1] = EXHCL_VALUE_30FPS;
      BytesWritten = write(Fdiic, WriteBuffer, 2);
      if(BytesWritten != 2)
          {
              printf("Error EXHCL, EXHCL_VALUE_30FPS\n");
        printf(" %s\n", strerror(errno));
              return -1;
          }
          else
          {
            printf("EXHCL, EXHCL_VALUE_30FPS \n");
          }
```

```
    usleep(WAITING_TIME);
    WriteBuffer[0] = DM_LNL;
      WriteBuffer[1] = DM_LNL_VALUE_30FPS;
      BytesWritten = write(Fdiic, WriteBuffer, 2);
      if(BytesWritten != 2)
          {
               printf("Error DM_LNL, DM_LNL_VALUE_30FPS\n");
        printf(" %s\n", strerror(errno));
               return -1;
          }
          else
          {
            printf("DM_LNL, DM_LNL_VALUE_30FPS \n");
          }
    usleep(WAITING_TIME);
    WriteBuffer[0] = DM_LNH;
      WriteBuffer[1] = DM_LNH_VALUE_30FPS;
      BytesWritten = write(Fdiic, WriteBuffer, 2);
      if(BytesWritten != 2)
          {
               printf("Error DM_LNH, DM_LNH_VALUE_30FPS\n");
        printf(" %s\n", strerror(errno));
               return -1;
          }
          else
          {
            printf("DM_LNH, DM_LNH_VALUE_30FPS \n");
          }
    usleep(WAITING_TIME);
    WriteBuffer[0] = COM11;
      WriteBuffer[1] = 0x0A;
      BytesWritten = write(Fdiic, WriteBuffer, 2);
      if(BytesWritten != 2)
    {
      printf("Error COM11 0x0A \n");
      printf(" %s\n", strerror(errno));
      return -1;
    }
    else
    {
      printf("COM11 0x0A\n");
    }
      usleep(WAITING_TIME);
  return 0;
}

int OV7670_set_window_output_registers(void)
{

  WriteBuffer[0] = HSTART;
  WriteBuffer[1] = HSTART_VALUE_VGA;
  BytesWritten = write(Fdiic, WriteBuffer, 2);
  if(BytesWritten != 2)
  {
    printf("Error HSTART HSTART_VALUE_VGA \n");
    printf(" %s\n", strerror(errno));
    return -1;
  }
  else
```

```
{
  printf("HSTART HSTART_VALUE_VGA\n");
}
usleep(WAITING_TIME);
printf("\n Video quality must be change to bad \n");
printf("\n HSTART_VALUE_VGA done.Press a button.\n");
//getchar();
WriteBuffer[0] = HSTOP;
WriteBuffer[1] = HSTOP_VALUE_VGA;
BytesWritten = write(Fdiic, WriteBuffer, 2);
if(BytesWritten != 2)
{
  printf("Error HSTOP HSTOP_VALUE_VGA \n");
  printf(" %s\n", strerror(errno));
  return -1;
}
else
{
  printf("HSTOP HSTOP_VALUE_VGA\n");
}
usleep(WAITING_TIME);
printf("\n Video quality must be change to good \n");
printf("\n HSTOP_VALUE_VGA done.Press a button.\n");
//getchar();
WriteBuffer[0] = HREF;
WriteBuffer[1] = HREF_VALUE_VGA;
BytesWritten = write(Fdiic, WriteBuffer, 2);
if(BytesWritten != 2)
{
  printf("Error HREF HREF_VALUE_VGA \n");
  printf(" %s\n", strerror(errno));
  return -1;
}
else
{
  printf("HREF HREF_VALUE_VGA\n");
}
usleep(WAITING_TIME);
printf("\n Video quality must be change to good \n");
printf("\n HREF_VALUE_VGA done.Press a button.\n");
//getchar();
WriteBuffer[0] = VSTRT;
WriteBuffer[1] = VSTRT_VALUE_VGA;
BytesWritten = write(Fdiic, WriteBuffer, 2);
if(BytesWritten != 2)
{
  printf("Error VSTRT VSTRT_VALUE_VGA \n");
  printf(" %s\n", strerror(errno));
  return -1;
}
else
{
  printf("VSTRT VSTRT_VALUE_VGA\n");
}
usleep(WAITING_TIME);
printf("\n Video quality must be change to good \n");
printf("\n VSTRT_VALUE_VGA done.Press a button.\n");
//getchar();
WriteBuffer[0] = VSTOP;
```

```
  WriteBuffer[1] = VSTOP_VALUE_VGA;
  BytesWritten = write(Fdiic, WriteBuffer, 2);
  if(BytesWritten != 2)
  {
    printf("Error VSTOP VSTOP_VALUE_VGA \n");
    printf(" %s\n", strerror(errno));
    return -1;
  }
  else
  {
    printf("VSTOP VSTOP_VALUE_VGA\n");
  }
  usleep(WAITING_TIME);
  WriteBuffer[0] = VREF;
  WriteBuffer[1] = VREF_VALUE_VGA;
  BytesWritten = write(Fdiic, WriteBuffer, 2);
  if(BytesWritten != 2)
  {
    printf("Error VREF VREF_VALUE_VGA \n");
    printf(" %s\n", strerror(errno));
    return -1;
  }
  else
  {
    printf("VREF VREF_VALUE_VGA\n");
  }
  usleep(WAITING_TIME);
  printf("\n There must be one vibration \n");
  printf("\n VREF done.Press a button.\n");
  //getchar();
  return 0;
}
```

### D.1.3: Taking Picture User Space Application for Linux

```
/*
 ============================================================
 Name        : take_picture.c
 Author      : UTKU ESEN
 Version     : June 26, 2018
 Copyright   : All free
 Description : take_pic linux user space app.
 ============================================================
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
#include <ctype.h>
#include <sys/mman.h>
#include <stdint.h>

#define FATAL do{ \
fprintf(stderr, "Error at line %d, file %s (%d)[%s]\n", \
    __LINE__, __FILE__, errno, strerror(errno));\
    exit(1);}while(0)

#define FRAME_W 640
#define FRAME_H 480
#define CAMERA_DEVICE "/dev/OV7670"
#define RAM_MEMORY_DEVICE "/dev/mem"
#define FRAME_BUFFER_DIM_WRITER 307200
#define FRAME_BUFFER_BASE_ADDR_WRITER 0x10000000
#define FRAME_BUFFER_BASE_ADDR_WRITER_TEMP 0x13000000
#define FRAME_BUFFER_NUM_WRITER 8
#define FRAME_BUFFER_DIM_READER 307200
#define FRAME_BUFFER_BASE_ADDR_READER 0x16000000
#define FRAME_BUFFER_NUM_READER 8
#define CAPTURED_IMAGE_FILE_NAME  "results/captured_image.dat"
#define MAP_SIZE_WRITER \
FRAME_BUFFER_DIM_WRITER * (FRAME_BUFFER_NUM_WRITER)
#define MAP_SIZE_READER \
FRAME_BUFFER_DIM_READER * (FRAME_BUFFER_NUM_READER)
#define MAP_MASK_WRITER (MAP_SIZE_WRITER-1)
#define MAP_MASK_READER (MAP_SIZE_READER-1)

static struct character_device {
  char last_image_number;
  char *last_image_address;
  char interrupt_status;
  char reset_pl_status;
  // writer base address
  int image_writer_base_address;
  // reader base address
  int image_reader_base_address;
  //struct semaphore sem;
} ov7670_device;

int main(void)
```

```
{
  // variables are being created
  int fd_camera, fd_mem;
  char input_char;
  unsigned int i, j;
  void *map_base_reader, *map_base_writer;
  void*virt_addr_src, *virt_addr_dst, *buffer_addr;
  unsigned long read_result, writeval;
  off_t source, destination;
  int access_type = 'w';
  int ready_for_read_frame_address;
  FILE  *fd_captured_image_save;

  // Creating new frame buffer in
  // virtual memory for copy one frame from camera
  buffer_addr = malloc(FRAME_BUFFER_DIM_WRITER);
  if(buffer_addr == 0)
  {
    printf("Error:  There is not enough memory in your system \n");
    fflush(stdout);
    FATAL;
  }

  // Opening camera config file
  fd_camera = open(CAMERA_DEVICE, O_RDWR);

  // Reading all camera informations
  read(fd_camera, &ov7670_device, sizeof(ov7670_device));
  ov7670_device.image_writer_base_address =
    FRAME_BUFFER_BASE_ADDR_WRITER_TEMP;
  ov7670_device.image_reader_base_address =
    FRAME_BUFFER_BASE_ADDR_READER;
  write(fd_camera, &ov7670_device, sizeof(ov7670_device));

  // Reading last frame address from all camera informations
  ready_for_read_frame_address =
    (int) ov7670_device.last_image_address;
  printf ("\n ready_for_read_frame_address: %p \n",
        ready_for_read_frame_address);
  fflush(stdout);

  // Opening Ram file
  if((fd_mem = open(RAM_MEMORY_DEVICE, O_RDWR|O_SYNC)) == -1)
  {
    printf("Problem reason is RAM_MEMORY cannot be opened\n");
    fflush(stdout);
    FATAL;
  }
  else
  {
    printf("/dev/mem opened\n");
  }
  fflush(stdout);

  // dst and src addresses are cited
  source      = FRAME_BUFFER_BASE_ADDR_WRITER;
  destination = FRAME_BUFFER_BASE_ADDR_READER;

  // Map Ram memory virtualy for writer
```

```
map_base_writer = mmap(
    0,
    MAP_SIZE_WRITER,
    PROT_READ | PROT_WRITE,
    MAP_SHARED,
    fd_mem,
    source
);

if(map_base_writer == (void*)-1)
{
  FATAL;

}
else
{
  printf("map_base_writer Memory mapped at address %p\n",
        map_base_writer);
}
fflush(stdout);

// Map Ram memory virtualy for reader
map_base_reader = mmap(
    0,
    MAP_SIZE_READER,
    PROT_READ | PROT_WRITE,
    MAP_SHARED,
    fd_mem,
    destination
);

if(map_base_reader == (void*)-1)
{
  FATAL;

}
else
{
  printf("map_base_reader Memory mapped at address %p\n",
      map_base_reader);
}

fflush(stdout);

// calculating src and dst virtual addresses
//virt_addr_dst =
// map_base_reader +(destination & MAP_MASK_READER);
virt_addr_dst = buffer_addr;
virt_addr_src = map_base_writer +
  ov7670_device.last_image_number*FRAME_BUFFER_NUM_WRITER;

// this address will be used
// after one frame is wrote on first buffer of vga
//first_frame_addr =
// map_base_reader +(destination & MAP_MASK_READER);
printf ("\n virt_addr_dst: %p,
      virt_addr_src: %p,
      buffer_addr: %p \n",
      virt_addr_dst,
```

```
      virt_addr_src,
      buffer_addr);
fflush(stdout);

// last frame wrote on ram is
// reading then writing on vga reader address
for(i = 0; i < (FRAME_BUFFER_DIM_READER); i++)
{
  *((unsigned char*)virt_addr_dst) =
         *((unsigned char*)virt_addr_src);
  virt_addr_dst++;
  virt_addr_src++;
}
printf ("\n buffer done \n");
fflush(stdout);
// after first frame is copied,
// we should copy other vga buffers from first vga_buffer
// because last frame wrote on ram from camera
// can be change during the writing process

// our source is vga buffer_0
virt_addr_src = buffer_addr;
printf ("\n virt_addr_src done \n");
  fflush(stdout);
// our destinations are from vga_buffer_1 to vga_buffer_7
virt_addr_dst = map_base_reader;
// this is showing vga_buffer_1
printf ("\n virt_addr_dst done \n");
    fflush(stdout);
// copy all vga_buffer from vga_buffer_0
for(i = 0; i < (FRAME_BUFFER_NUM_READER); i++)
{
  // our source is vga buffer_0
    virt_addr_src = buffer_addr;
      printf ("\n dst = %x,
                  virt_addr_dst: %p,
                  virt_addr_src: %p,
                  buffer_addr: %p \n",
                  destination,
                  virt_addr_dst,
                  virt_addr_src,
                  buffer_addr);
    fflush(stdout);

  for(j = 0; j < (FRAME_BUFFER_DIM_READER); j++)
  {
    *((unsigned char*)virt_addr_dst) =
            *((unsigned char*)virt_addr_src);
    virt_addr_dst++;
    virt_addr_src++;
  }

}
printf ("\n vga done \n");
      fflush(stdout);

////////////// write captured image to sdcard ////////////
// create or open existing original image data file
fd_captured_image_save = fopen(CAPTURED_IMAGE_FILE_NAME, "wb");
```

```
  if (fd_captured_image_save == NULL)
  {
    printf("Error opening or creating file!\n");
    exit(1);
  }

  // our src is buffer
  virt_addr_src = buffer_addr;

  // write all pixel values on file
  for(j = 0; j < (FRAME_W); j++)
  {
    for(i = 0; i < (FRAME_H); i++)
    {
      virt_addr_src = buffer_addr + i * FRAME_W + j;
      fprintf(fd_captured_image_save, "%c",
                *((unsigned char*)virt_addr_src));
    }

  }
  fclose(fd_captured_image_save);
  ////////////////////////////////////////////////////////
  // releasing virtual memory for writer
  if(munmap(map_base_writer, MAP_SIZE_WRITER) == -1)
  {
    FATAL;
  }

  // releasing virtual memory for reader
  if(munmap(map_base_reader, MAP_SIZE_READER) == -1)
  {
    FATAL;
  }

  // Release buffer
  free(buffer_addr);

  // Closing memory file
  close(fd_mem);
  printf ("\n continue to video stream press a key \n");
  fflush(stdout);
  getchar();

  ov7670_device.image_writer_base_address =
        FRAME_BUFFER_BASE_ADDR_WRITER;
  ov7670_device.image_reader_base_address =
        FRAME_BUFFER_BASE_ADDR_WRITER;
  write(fd_camera, &ov7670_device, sizeof(ov7670_device));


  close(fd_camera);

  return 0;
}
```

### D.1.4: Information Hiding User Space Application for Linux

```
/*
 ============================================================================
 Name        : data_hiding_just_arm.c
 Author      : UTKU ESEN
 Version     : June 26, 2018
 Copyright   : Your copyright notice
 Description : data hiding just arm
 ============================================================================
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
#include <ctype.h>
#include <sys/mman.h>
#include <math.h>

#define FATAL do{ \
fprintf(stderr, "Error at line %d, file %s (%d)[%s]\n", \
    __LINE__, __FILE__, errno, strerror(errno));\
    exit(1);}while(0)

#define CAMERA_DEVICE "/dev/OV7670"
#define RAM_MEMORY_DEVICE "/dev/mem"

#define PLAINTEXT_FILE_NAME \
"plaintext.txt"
#define BINARY_PLAINTEXT_FILE_NAME \
"results/binary_plaintext.txt"
#define ORIGINAL_IMAGE_FILE_NAME \
"results/original_image.dat"
#define STEGO_IMAGE_FILE_NAME \
"results/stego_image.dat"
#define INTERPOLATION_IMAGE_FILE_NAME \
"results/interpolation_image.dat"
#define BIT_NUMBER_IMAGE_FILE_NAME \
"results/bit_number_image.dat"

#define FRAME_W    640
#define FRAME_H    480
#define THRESHOLD 210

#define FRAME_BUFFER_DIM_WRITER 307200
#define FRAME_BUFFER_BASE_ADDR_WRITER 0x10000000
#define FRAME_BUFFER_BASE_ADDR_WRITER_TEMP 0x13000000
#define FRAME_BUFFER_NUM_WRITER 8

#define FRAME_BUFFER_DIM_READER 307200
#define FRAME_BUFFER_BASE_ADDR_READER 0x16000000
#define FRAME_BUFFER_NUM_READER 8

#define MAP_SIZE_WRITER \
FRAME_BUFFER_DIM_WRITER * (FRAME_BUFFER_NUM_READER)
```

```
#define MAP_SIZE_READER \
FRAME_BUFFER_DIM_READER * (FRAME_BUFFER_NUM_READER)

#define MAP_MASK_WRITER (MAP_SIZE_WRITER-1)
#define MAP_MASK_READER (MAP_SIZE_READER-1)

static struct character_device {
  char last_image_number;
  char *last_image_address;
  char interrupt_status;
  char reset_pl_status;
  // writer base address
  int image_writer_base_address;
  // reader base address
  int image_reader_base_address;
  //struct semaphore sem;
} ov7670_device;

////////////////// Global Variables //////////////////////
int key[3][2]={100, 120, 300,200, 10, 470};
void *map_base_reader, *map_base_writer;
void *virt_addr_src, *virt_addr_dst;
void *buffer_addr;
void *interpolation_im_addr;
void *embedding_map_addr;
void *stego_im_addr;
int *symmetry_map_addr_rows, *symmetry_map_addr_columns;
int fd_camera, fd_mem;

char *file_wrt_src_ptr, *process_wrt_src_ptr, *process_wrt_dst_ptr;
unsigned long read_result, writeval;
off_t source, destination;
int ready_for_read_frame_address;
int plaintext_len, binary_plaintext_len;

////////////////////// User Functions /////////////////////

void get_plaintext(void);
void convert_plaintext_to_binary_and_save_on_sdcard(void);
void get_key_from_user(void);
void create_buffers(void);


void read_last_image_number(void);
void initialize_RAM(void);
void init_virt_addr_write(void);
void init_virt_addr_read(void);
void copy_last_image_to_buffer(void);
void save_original_image_to_sd_card(void);
void run_data_hiding_app(void);
char measure_bit_number( char decimal_number);
void add_hide_data_on_interpolation_values(void);
void show_buffer_on_VGA(void);
void show_result_on_VGA(void);
void save_interpolation_image_to_sd_card(void);
void save_bit_number_image_to_sd_card(void);
void save_stego_image_to_sd_card(void);
void release_mems(void);
```

```
int main(void)
{
  // variables are being created
  char input_char;

  // get plaintext
  get_plaintext();

  convert_plaintext_to_binary_and_save_on_sdcard();



  // Create two buffer for original image and stego image
  create_buffers();

  // Read Last Image Number From Camera
  read_last_image_number();

    // initialize RAM
  initialize_RAM();

  // initialize virtual addresses of writer and reader
  init_virt_addr_write();
  init_virt_addr_read();

  ////////////////////////////// STEP 1 ////////////////////////
  // Copy last image to buffer
  copy_last_image_to_buffer();
  show_buffer_on_VGA();

  ////////////////////////////// STEP 2 ////////////////////////
  // Saving original image on sd_card
  save_original_image_to_sd_card();

  // key input from user
  get_key_from_user();

  ////////////////////////////// STEP 3 ////////////////////////
  // run data_hiding application
  run_data_hiding_app();

  // Saving STEGO image on sd_card
  save_stego_image_to_sd_card();

  ////////////////////// STEP 4 ////////////////////////////////
  // show result on VGA
  show_result_on_VGA();


  // Release buffer
  free(buffer_addr);

  // releasing virtual memory for writer
  if(munmap(map_base_writer, MAP_SIZE_WRITER) == -1)
  {
    FATAL;
  }

  // Release memories
```

```
  release_mems();

  return 0;
}

void get_plaintext(void)
{
  FILE *fd_plaintext;
  // open existing plaintext data file
  fd_plaintext = fopen(PLAINTEXT_FILE_NAME, "r");
  if (fd_plaintext == NULL)
  {
    printf("Error opening plaintext-1 file!\n");
    exit(1);
  }
  fseek(fd_plaintext, 0, SEEK_END);
  plaintext_len = ftell(fd_plaintext);

  printf("\n Plaintext size is %d \n", plaintext_len);
  fflush(stdout);

  // Go back to the beginning of file
  rewind(fd_plaintext);
  fclose(fd_plaintext);
}

void convert_plaintext_to_binary_and_save_on_sdcard(void)
{
  int i, j;
  FILE *fd_plaintext, *fd_binary_plaintext;
  // open existing plaintext data file
  fd_plaintext = fopen(PLAINTEXT_FILE_NAME, "r");
  if (fd_plaintext == NULL)
  {
    printf("Error opening plaintext-2 file!\n");
    exit(1);
  }

  char char_data, temp_data;
  char binary_data[8];
  // create or open existing plaintext data file
  fd_binary_plaintext = fopen(BINARY_PLAINTEXT_FILE_NAME, "w+");
  if (fd_binary_plaintext == NULL)
  {
    printf("Error opening binary plaintext file!\n");
    exit(1);
  }


  for(i = 0; i < plaintext_len; i++)
  {
    fscanf(fd_plaintext, "%c", &char_data);
    temp_data = char_data;
    //printf("\n char_data = %c %d\n", char_data, char_data);
    //fflush(stdout);
    for(j = 0; j < 8; j++)
    {
      if(temp_data >= (1<<(7 - j)))
      {
```

```
        fprintf(fd_binary_plaintext, "%d ", 1);
        temp_data = temp_data - (1<<(7 - j));
//        printf(" %d", 1);
//        fflush(stdout);
      }
      else
      {
        fprintf(fd_binary_plaintext, "%d ", 0);
//        printf(" %d", 0);
//        fflush(stdout);
      }
    }
    //printf("\n");
    //fflush(stdout);
//    scanf("%s", &key[0][0]);
  }
  binary_plaintext_len = ftell(fd_binary_plaintext);
  printf("\n binary_plaintext_len is %d \n",
           binary_plaintext_len);
  fflush(stdout);
  fclose(fd_plaintext);
  fclose(fd_binary_plaintext);

  printf("\n Binary plaintext is done \n");
  fflush(stdout);
}

void get_key_from_user(void)
{
  printf("\n Welcome to data hiding application");
  printf("\n Please enter your key below \n");

  printf("\n Enter your key's 1th part between 0 and 479 \n");
  scanf("%d", &key[0][0]);

  printf("\n Enter your key's 2th part between 0 and 639 \n");
  scanf("%d", &key[0][1]);

  printf("\n Enter your key's 3th part between 0 and 479 \n");
  scanf("%d", &key[1][0]);

  printf("\n Enter your key's 4th part between 0 and 639 \n");
  scanf("%d", &key[1][1]);

  printf("\n Enter your key's 5th part between 0 and 479 \n");
  scanf("%d", &key[2][0]);

  printf("\n Enter your key's 6th part between 0 and 639 \n");
  scanf("%d", &key[2][1]);
  fflush(stdout);
}

void create_buffers(void)
{
  // Creating new frame buffer in virtual memory for
  // copy one frame from camera
  buffer_addr = malloc(FRAME_BUFFER_DIM_WRITER);
  if(buffer_addr == 0)
  {
```

```
    printf(
        "Error: There is not enough memory in your system \n");
    fflush(stdout);
    FATAL;
  }

  // Creating interpolation_im_addr in virtual memory
  interpolation_im_addr = malloc(FRAME_BUFFER_DIM_WRITER);
  if(interpolation_im_addr == 0)
  {
    printf("Error: Insufficient memory \n");
    fflush(stdout);
    FATAL;
  }

  // Creating embedding_map buffer in virtual memory
  embedding_map_addr = malloc(FRAME_BUFFER_DIM_WRITER);
  if(embedding_map_addr == 0)
  {
    printf("Error: Insufficient memory\n");
    fflush(stdout);
    FATAL;
  }

  // Creating result frame buffer in virtual memory
  // for copy filter results
  stego_im_addr = malloc(FRAME_BUFFER_DIM_WRITER);
  if(stego_im_addr == 0)
  {
    printf("Error: Insufficient memory\n");
    fflush(stdout);
    FATAL;
  }

}

void read_last_image_number(void)
{
  // Opening camera config file
  fd_camera = open(CAMERA_DEVICE, O_RDWR);

  // Reading all camera informations
  read(fd_camera, &ov7670_device, sizeof(ov7670_device));
  ov7670_device.image_writer_base_address =
    FRAME_BUFFER_BASE_ADDR_WRITER_TEMP;
  ov7670_device.image_reader_base_address =
    FRAME_BUFFER_BASE_ADDR_READER;
  write(fd_camera, &ov7670_device, sizeof(ov7670_device));

  // showing all informations about camera to user
  printf ("\n last_image_number: %d",
        ov7670_device.last_image_number);
  printf ("\n last_image_address: %x",
        ov7670_device.last_image_address);
  printf ("\n interrupt_status: %d",
        ov7670_device.interrupt_status);
  printf ("\n reset_pl_status: %d \n",
        ov7670_device.reset_pl_status);
  printf ("\n image_writer_base_address: %x \n",
```

```
          ov7670_device.image_writer_base_address);
  printf ("\n image_reader_base_address: %x \n",
          ov7670_device.image_reader_base_address);
  fflush(stdout);

  // Reading last frame address from all camera informations
  ready_for_read_frame_address =
    (int) ov7670_device.last_image_number;
  printf ("\n last_image_number: %d \n",
        ready_for_read_frame_address);
  fflush(stdout);

}

void initialize_RAM(void)
{
  // Opening Ram file
  if((fd_mem = open(RAM_MEMORY_DEVICE, O_RDWR|O_SYNC)) == -1)
  {

    printf("Problem reason is RAM_MEMORY cannot be opened\n");
    fflush(stdout);
    FATAL;
  }
  else
  {
    printf("/dev/mem opened\n");
  }
  fflush(stdout);
}
void init_virt_addr_write(void)
{
  // dst and src addresses are cited
  source = FRAME_BUFFER_BASE_ADDR_WRITER;

  // Map Ram memory virtualy for writer
  map_base_writer = mmap(0, MAP_SIZE_WRITER,
              PROT_READ | PROT_WRITE,
              MAP_SHARED,
              fd_mem,
              source);
  if(map_base_writer == (void*)-1)
  {
    FATAL;
  }
  else
  {
    printf("Memory mapped at address %x.\n", map_base_writer);
  }
  fflush(stdout);
}
void init_virt_addr_read(void)
{
  // dst and src addresses are cited
  destination = ov7670_device.image_reader_base_address;

  // Map Ram memory virtualy for reader
  map_base_reader = mmap(0, MAP_SIZE_READER,
              PROT_READ | PROT_WRITE,
```

```
                MAP_SHARED,
                fd_mem,
                destination);
  if(map_base_reader == (void*)-1)
  {
    FATAL;

  }
  else
  {
    printf("Memory mapped at address %x.\n", map_base_reader);
  }

  fflush(stdout);
}
void copy_last_image_to_buffer(void)
{
  unsigned int i;
  unsigned char* intr_img_addr;
  // calculating src and dst virtual addresses
  //virt_addr_dst = map_base_reader +(destination & MAP_MASK_READER);
  virt_addr_dst = buffer_addr;
  virt_addr_src =
    map_base_writer +
    FRAME_BUFFER_DIM_WRITER * ready_for_read_frame_address;
  intr_img_addr = interpolation_im_addr;
  // this address will be used
  // after one frame is wrote on first buffer of vga
  // first_frame_addr =
  // map_base_reader +(destination & MAP_MASK_READER);
  printf ("\n virt_addr_dst: %p,
                virt_addr_src: %p,
                buffer_addr: %p,
                stego_im_addr: %p \n",
        virt_addr_dst,
        virt_addr_src,
        buffer_addr,
        stego_im_addr);
  fflush(stdout);


  ////////////////// STEP 1 ////////////////////////////
  // last frame wrote on ram is reading then writing on frame_buffer
  for(i = 0; i < (FRAME_BUFFER_DIM_READER); i++)
  {
    *((unsigned char*)virt_addr_dst) =
            *((unsigned char*)virt_addr_src);
    *((unsigned char*)intr_img_addr) =
            *((unsigned char*)virt_addr_src);

    virt_addr_dst++;
    virt_addr_src++;
    intr_img_addr++;
  }
  printf ("\n buffer done \n");
  fflush(stdout);
}
void save_original_image_to_sd_card(void)
{
```

```
  unsigned int i, j;
  FILE *fd_original_image_save;
  // create or open existing original image data file
  fd_original_image_save =
        fopen(ORIGINAL_IMAGE_FILE_NAME, "w+");
  if (fd_original_image_save == NULL)
  {
    printf("Error opening or creating file!\n");
    exit(1);
  }

  // our src is buffer
  // write all pixel values on file
  for(j = 0; j < (FRAME_W); j++)
  {
    for(i = 0; i < (FRAME_H); i++)
    {
      file_wrt_src_ptr =
                buffer_addr + i * FRAME_W + j;
      fprintf(fd_original_image_save, "%c",
                *file_wrt_src_ptr);
    }
  }

  // close original image file
  fclose(fd_original_image_save);
}
void run_data_hiding_app(void)
{
  unsigned int i, j, l;
  int status;
  char interpolated_im[2][2];
    char original_im[2][2];
    char emb_sizes_im[2][2];
  char * embedding_ptr, input;
  // source is last frame_buffer and
    // destination is filter result
  // calculate interpolations and find embedding sizes
  // for each interpolation pixels
  for(i = 0; i < (FRAME_H); i+=2)
  {
    for(j = 0; j < (FRAME_W); j+=2)
    {
      // Calculating region area
      status = 0;
      // Checking columns
      if(j == (FRAME_W - 2))
      {
        status += 1;
      }
      // Checking rows
      if(i == (FRAME_H - 2))
      {
        status += 2;
      }
      //printf ("\n status= %d \n", status);
        //fflush(stdout);
      process_wrt_dst_ptr =
            interpolation_im_addr + i * FRAME_W + j;
```

```
process_wrt_src_ptr =
      buffer_addr + i * FRAME_W + j;
embedding_ptr =
      embedding_map_addr + i * FRAME_W + j;
// Calculating interpolation values and
// embedding sizes according to pixels area
switch(status)
{
  //<---------------- H ---------------->
  ////////////////////////////////////    ^
  //                  //        // / \
  //                      //            //    |
  //                  //        //   |
  //    AREA 0        //    AREA 1   //   |
  //                  //        //   |
  //                  //        //  W
  //                  //        //   |
  ////////////////////////////////////   |
  //              //        //   |
  //       AREA 2       //    AREA 3   //   |
  //              //        //    \ /
  ////////////////////////////////////    v

  ////////////// AREA 0 ////////////////
  case 0:
    /////////////////////////// Values ///////
    original_im[0][0] =
                  *process_wrt_src_ptr;
    original_im[0][1] =
                  *(process_wrt_src_ptr+2);
    original_im[1][0] =
                  *(process_wrt_src_ptr+FRAME_W*2);
    original_im[1][1] =
                  *(process_wrt_src_ptr+FRAME_W*2 + 2);

    // known pixel is wrote
    interpolated_im[0][0] =
                  original_im[0][0];

    // first pixel area is calculated
    interpolated_im[0][1] =
                  (original_im[0][0] + original_im[0][1]) >> 1;

    // second pixel area is calculated
    interpolated_im[1][0] =
                  (original_im[0][0] + original_im[1][0]) >> 1;

    // third pixel area is calculated
    interpolated_im[1][1] =
                  (original_im[0][0] + original_im[1][1]) >> 1;

    ////////////////////// Embedding sizes //////////
    emb_sizes_im[0][0] = 0;

    // first pixel area is calculated

    emb_sizes_im[0][1] =
                  measure_bit_number
                  (abs(interpolated_im[0][0]
```

```
                - interpolated_im[0][1]));

    // second pixel area is calculated
    emb_sizes_im[1][0] =
                measure_bit_number
                (abs(interpolated_im[0][0]
                - interpolated_im[1][0]));

    // third pixel area is calculated
    emb_sizes_im[1][1] =
                measure_bit_number
                (abs(interpolated_im[0][0]
                - interpolated_im[1][1]));
    break;

  ////////// AREA 1 ///////////////////////
  case 1:
    ////////// Values //////////////
    original_im[0][0] =
                *process_wrt_src_ptr;
    original_im[0][1] =
                *(process_wrt_src_ptr+1);
    original_im[1][0] =
                *(process_wrt_src_ptr+FRAME_W);
    original_im[1][1] =
                *(process_wrt_src_ptr+FRAME_W + 1);
    // known pixel is wrote
    interpolated_im[0][0] =
                original_im[0][0];

    // first pixel area is calculated
    interpolated_im[0][1] =
                original_im[0][1];

    // second pixel area is calculated
    interpolated_im[1][0] =
                original_im[1][0];

    // third pixel area is calculated
    interpolated_im[1][1] =
                original_im[1][1];

    ////////////// Embedding sizes //////////////
    emb_sizes_im[0][0] = 0;

    // first pixel area is calculated
    emb_sizes_im[0][1] = 0;

    // second pixel area is calculated
    emb_sizes_im[1][0] = 0;

    // third pixel area is calculated
    emb_sizes_im[1][1] = 0;
    break;

  ////////////////// AREA 2 //////////////////
  case 2:
    ////////// Values //////////////
    original_im[0][0] =
```

```
                *process_wrt_src_ptr;
    original_im[0][1] =
                *(process_wrt_src_ptr+1);
    original_im[1][0] =
                *(process_wrt_src_ptr+FRAME_W);
    original_im[1][1] =
                *(process_wrt_src_ptr+FRAME_W + 1);
    // known pixel is wrote
    interpolated_im[0][0] =
                original_im[0][0];

    // first pixel area is calculated
    interpolated_im[0][1] =
                original_im[0][1];

    // second pixel area is calculated
    interpolated_im[1][0] =
                original_im[1][0];

    // third pixel area is calculated
    interpolated_im[1][1] =
                original_im[1][1];

    ////////////// Embedding sizes //////////////
    emb_sizes_im[0][0] = 0;

    // first pixel area is calculated
    emb_sizes_im[0][1] = 0;

    // second pixel area is calculated
    emb_sizes_im[1][0] = 0;

    // third pixel area is calculated
    emb_sizes_im[1][1] = 0;
    break;
  /////////////// AREA 3 /////////////////
  case 3:
    ////////// Values //////////////
    original_im[0][0] =
                *process_wrt_src_ptr;
    original_im[0][1] =
                *(process_wrt_src_ptr+1);
    original_im[1][0] =
                *(process_wrt_src_ptr+FRAME_W);
    original_im[1][1] =
                *(process_wrt_src_ptr+FRAME_W + 1);
    // known pixel is wrote
    interpolated_im[0][0] =
                original_im[0][0];

    // first pixel area is calculated
    interpolated_im[0][1] =
                original_im[0][1];

    // second pixel area is calculated
    interpolated_im[1][0] =
                original_im[1][0];

    // third pixel area is calculated
```

```
          interpolated_im[1][1] =
                        original_im[1][1];

          ////////////// Embedding sizes //////////////
          emb_sizes_im[0][0] = 0;

          // first pixel area is calculated
          emb_sizes_im[0][1] = 0;

          // second pixel area is calculated
          emb_sizes_im[1][0] = 0;

          // third pixel area is calculated
          emb_sizes_im[1][1] = 0;
          break;
        ////////////////////// ERROR //////////////
        default:
          break;
      }
      // interpolation values are wrote
      *process_wrt_dst_ptr =
                interpolated_im[0][0];
      *(process_wrt_dst_ptr+1) =
                interpolated_im[0][1];
      *(process_wrt_dst_ptr+FRAME_W) =
                interpolated_im[1][0];
      *(process_wrt_dst_ptr+FRAME_W+1) =
                interpolated_im[1][1];

      // embedding mapping is wrote
      *embedding_ptr = emb_sizes_im[0][0];
      *(embedding_ptr+1) = emb_sizes_im[0][1];
      *(embedding_ptr+FRAME_W) = emb_sizes_im[1][0];
      *(embedding_ptr+FRAME_W+1) = emb_sizes_im[1][1];
    }
  }
  // save interpolation image on sd card
  save_interpolation_image_to_sd_card();
  // save bit_number image on sd card
  save_bit_number_image_to_sd_card();
  // apply hiding on interpolation pixels
  add_hide_data_on_interpolation_values();
}

char measure_bit_number( char decimal_number)
{
  char l=0, debug;
  int threshold=0;
  if(decimal_number == 0)
  {
    return 0;
  }
  if(decimal_number == 1|| decimal_number == 2)
  {
    return 1;
  }
  for(l = 1; l < 8; l++)
  {
    threshold = (1<<l )-1;
```

```
    if(threshold > decimal_number)
    {
      return l-1;
    }
  }
  return 7;
}
void add_hide_data_on_interpolation_values()
{
  int i, j, k, l, z, debug;
  int start_pixel_x;
  int start_pixel_y;

  int finish_pixel_x = FRAME_H;
  int finish_pixel_y = FRAME_W-1;

  char bin_data, decimal_value;
  char * embedding_ptr;
  char emb_sizes_im[3];
  int symmetry_point[3][2];
  int binary_read_counter;
  FILE* fd_binary_plaintext;
  fd_binary_plaintext =
        fopen(BINARY_PLAINTEXT_FILE_NAME, "r");
  if (fd_binary_plaintext == NULL)
  {
    printf("Error opening binary plaintext file!\n");
    exit(1);
  }
  binary_read_counter = 0;
  //rewind(fd_binary_plaintext);

  for(i = 0; i < (FRAME_H); i+=2)
  {
    for(j = 0; j < (FRAME_W); j+=2)
    {
      // main pixels are directly written
      process_wrt_src_ptr =
                interpolation_im_addr + i * FRAME_W + j;
      process_wrt_dst_ptr =
                stego_im_addr + i * FRAME_W + j;
      *process_wrt_dst_ptr =
                *process_wrt_src_ptr;
      ////////////////////////////////////////////////////
      //             //              //
      //  known pixel area  //  first pixel area     //
      //    (i, j)       //    (i, j + 1)       //
      //             //              //
      ////////////////////////////////////////////////////
      //             //              //
      //  second pixel area //  third pixel area     //
      //    (i + 1, j)     //    (i + 1, j + 1)     //
      //             //              //
      ////////////////////////////////////////////////////

      ////////////////////// SYMMETRY POINTS /////////////////
      // first pixel area
      // calculate symmetry point
      // according to key[0][0] and key[0][1]
```

```
// HEIGHT


symmetry_point[0][0] = 2 * key[0][0] - (i);
if(symmetry_point[0][0] < 0)
{
  symmetry_point[0][0] =
              symmetry_point[0][0] + FRAME_H;
  if(symmetry_point[0][0] < 0)
  {
    symmetry_point[0][0] =
                symmetry_point[0][0] + FRAME_H;
  }
}
if(symmetry_point[0][0] > (FRAME_H-1))
{
  symmetry_point[0][0] =
              symmetry_point[0][0] - FRAME_H;
  if(symmetry_point[0][0] > (FRAME_H-1))
  {
    symmetry_point[0][0] =
                symmetry_point[0][0] - FRAME_H;
  }
}


// WIDTH
symmetry_point[0][1] = 2 * key[0][1] - (j + 1);
if(symmetry_point[0][1] < 0)
{
  symmetry_point[0][1] =
              symmetry_point[0][1] + FRAME_W;
  if(symmetry_point[0][1] < 0)
  {
    symmetry_point[0][1] =
                symmetry_point[0][1] + FRAME_W;
  }
}
if(symmetry_point[0][1] > (FRAME_W-1))
{
  symmetry_point[0][1] =
              symmetry_point[0][1] - FRAME_W;
  if(symmetry_point[0][1] > (FRAME_W-1))
  {
    symmetry_point[0][1] =
                symmetry_point[0][1] - FRAME_W;
  }
}

symmetry_point[1][0] = 2 * key[1][0] - (i + 1);
if(symmetry_point[1][0] < 0)
{
  symmetry_point[1][0] =
              symmetry_point[1][0] + FRAME_H;
  if(symmetry_point[1][0] < 0)
  {
    symmetry_point[1][0] =
                symmetry_point[1][0] + FRAME_H;
  }
```

```
    }
    if(symmetry_point[1][0] > (FRAME_H-1))
    {
      symmetry_point[1][0] =
                  symmetry_point[1][0] - FRAME_H;
    }


    // WIDTH
    symmetry_point[1][1] = 2 * key[1][1] - (j);
    if(symmetry_point[1][1] < 0)
    {
      symmetry_point[1][1] =
                  symmetry_point[1][1] + FRAME_W;
      if(symmetry_point[1][1] < 0)
      {
        symmetry_point[1][1] =
                      symmetry_point[1][1] + FRAME_W;
      }
    }
    if(symmetry_point[1][1] > (FRAME_W-1))
    {
      symmetry_point[1][1] =
                  symmetry_point[1][1] - FRAME_W;
      if(symmetry_point[1][1] > (FRAME_W-1))
      {
        symmetry_point[1][1] =
                      symmetry_point[1][1] - FRAME_W;
      }
    }


    symmetry_point[2][0] = 2 * key[2][0] - (i + 1);
    if(symmetry_point[2][0] < 0)
    {
      symmetry_point[2][0] =
                  symmetry_point[2][0] + FRAME_H;
      if(symmetry_point[2][0] < 0)
      {
        symmetry_point[2][0] =
                      symmetry_point[2][0] + FRAME_H;
      }
    }
    if(symmetry_point[2][0] > (FRAME_H-1))
    {
      symmetry_point[2][0] =
                  symmetry_point[2][0] - FRAME_H;
      if(symmetry_point[2][0] > (FRAME_H-1))
      {
        symmetry_point[2][0] =
                      symmetry_point[2][0] - FRAME_H;
      }
    }

    // WIDTH
    symmetry_point[2][1] = 2 * key[2][1] - (j + 1);
    if(symmetry_point[2][1] < 0)
    {
      symmetry_point[2][1] =
```

```
                symmetry_point[2][1] + FRAME_W;
    if(symmetry_point[2][1] < 0)
    {
      symmetry_point[2][1] =
                    symmetry_point[2][1] + FRAME_W;
    }
  }
  if(symmetry_point[2][1] > (FRAME_W-1))
  {
    symmetry_point[2][1] =
                symmetry_point[2][1] - FRAME_W;
    if(symmetry_point[2][1] > (FRAME_W-1))
    {
      symmetry_point[2][1] =
                    symmetry_point[2][1] - FRAME_W;
    }
  }

  ////////////////////// EMBEDDING PROCESS ////////////////
  // read embedding size according to symmetry point
  // first area
  embedding_ptr =
            embedding_map_addr +
            symmetry_point[0][0] * FRAME_W +
            symmetry_point[0][1];
  emb_sizes_im[0] = *embedding_ptr;

  // second area
  embedding_ptr =
            embedding_map_addr +
            (symmetry_point[1][0]) * FRAME_W +
            symmetry_point[1][1];
  emb_sizes_im[1] = *embedding_ptr;

  // third area
  embedding_ptr =
            embedding_map_addr +
            (symmetry_point[2][0]) * FRAME_W +
            symmetry_point[2][1];
  emb_sizes_im[2] = *embedding_ptr;
  // add embedding size amount of
  // plaintext part to symmetry point
  for(k = 0; k < 3; k++)
  {
    if(binary_read_counter <= binary_plaintext_len)
    {
      decimal_value = 0;
      // calculate decimal value according to plaintext part
      for(l = (emb_sizes_im[k] - 1); l > -1; l--)
      {
        fscanf(fd_binary_plaintext, "%d ", &bin_data);
        decimal_value += bin_data << l;
      }
      // adding process
      process_wrt_src_ptr =
                    interpolation_im_addr +
                    symmetry_point[k][0] * FRAME_W +
                    symmetry_point[k][1];
      process_wrt_dst_ptr =
```

```
                        stego_im_addr +
                        symmetry_point[k][0] * FRAME_W +
                        symmetry_point[k][1];
          *process_wrt_dst_ptr =
                        *process_wrt_src_ptr +
                        decimal_value;

          // bin read counting
          binary_read_counter += emb_sizes_im[k];

        }else
        {
          // after all bits added from plaintext
          process_wrt_src_ptr =
                        interpolation_im_addr +
                        symmetry_point[k][0] * FRAME_W +
                        symmetry_point[k][1];
          process_wrt_dst_ptr =
                        stego_im_addr +
                        symmetry_point[k][0] * FRAME_W +
                        symmetry_point[k][1];
          *process_wrt_dst_ptr = *process_wrt_src_ptr;
        }

      }
    }
  }
  fclose(fd_binary_plaintext);

}
void show_buffer_on_VGA(void)
{
  unsigned int i, j;
  // after first frame is copied,
  // we should copy other vga buffers from first vga_buffer
  // because last frame wrote on ram from camera
  // can be change during the writing process
  // our destinations are from vga_buffer_1 to vga_buffer_7
  virt_addr_dst = map_base_reader;
  // this is showing vga_buffer_1

    ////////////////////////// STEP 4 ////////////////////////////
  // copy all vga_buffer from vga_buffer_0
  for(i = 0; i < (FRAME_BUFFER_NUM_READER); i++)
  {
    // our source is  stego_im_ptr
      virt_addr_src = buffer_addr;
    //printf ("\n virt_addr_dst = %d,
    // virt_addr_src = %d \n", virt_addr_dst, virt_addr_src);
    for(j = 0; j < (FRAME_BUFFER_DIM_READER); j++)
    {
      *((unsigned char*)virt_addr_dst) =
                *((unsigned char*)virt_addr_src);
      virt_addr_dst++;
      virt_addr_src++;
    }
  }
  printf ("\n buffer vga done \n");
```

```
}

void show_result_on_VGA(void)
{
  unsigned int i, j;
  // after first frame is copied,
  // we should copy other vga buffers from first vga_buffer
  // because last frame wrote on ram from camera
  // can be change during the writing process

  // our destinations are from vga_buffer_1 to vga_buffer_7
  virt_addr_dst = map_base_reader;
  // this is showing vga_buffer_1

    ////////////////////////// STEP 4 ////////////////////////////
  // copy all vga_buffer from vga_buffer_0
  for(i = 0; i < (FRAME_BUFFER_NUM_READER); i++)
  {
    // our source is  stego_im_ptr
      virt_addr_src = stego_im_addr;
    for(j = 0; j < (FRAME_BUFFER_DIM_READER); j++)
    {
      *((unsigned char*)virt_addr_dst) =
                *((unsigned char*)virt_addr_src);
      virt_addr_dst++;
      virt_addr_src++;
    }
  }
  printf ("\n vga done \n");
  fflush(stdout);
  printf ("\n continue to video stream press a key \n");
  scanf("%d", &key[2][1]);
  fflush(stdout);
  ov7670_device.image_writer_base_address =
        FRAME_BUFFER_BASE_ADDR_WRITER;
  ov7670_device.image_reader_base_address =
        FRAME_BUFFER_BASE_ADDR_WRITER;
  write(fd_camera, &ov7670_device, sizeof(ov7670_device));
    // Closing camera config file
    close(fd_camera);
}

void save_interpolation_image_to_sd_card(void)
{
    unsigned int i, j;
    FILE* fd_interpolation_image_save;
  // create or open existing original image data file
  fd_interpolation_image_save =
        fopen(INTERPOLATION_IMAGE_FILE_NAME, "w+");
  if (fd_interpolation_image_save == NULL)
  {
    printf("Error opening or creating file!\n");
    exit(1);
  }
  // our src is buffer
  file_wrt_src_ptr = interpolation_im_addr;
  printf("file_wrt_src_ptr =%p.\n", file_wrt_src_ptr);

  // write all pixel values on file
```

```
  for(j = 0; j < (FRAME_W); j++)
  {
    for(i = 0; i < (FRAME_H); i++)
    {
      file_wrt_src_ptr =
                interpolation_im_addr +
                i * FRAME_W + j;
      fprintf(
                fd_interpolation_image_save,
                "%c",
                *file_wrt_src_ptr);
    }
  }
  // close interpolation image file
  fclose(fd_interpolation_image_save);
  printf("interpolation_image_save done.\n");
  fflush(stdout);
}

void save_bit_number_image_to_sd_card(void)
{
    unsigned int i, j;
    FILE* fd_bit_number_image_save;
  // create or open existing original image data file
  fd_bit_number_image_save =
        fopen(BIT_NUMBER_IMAGE_FILE_NAME, "w+");
  if (fd_bit_number_image_save == NULL)
  {
    printf("Error opening or creating file bit_number!\n");
    exit(1);
  }
  // our src is buffer
  file_wrt_src_ptr = embedding_map_addr;
  printf("file_wrt_src_ptr =%p.\n", file_wrt_src_ptr);
  // write all pixel values on file
  for(j = 0; j < (FRAME_W); j++)
  {
    for(i = 0; i < (FRAME_H); i++)
    {
      file_wrt_src_ptr =
                embedding_map_addr + i * FRAME_W + j;
      fprintf(
                fd_bit_number_image_save,
                "%c",
                *file_wrt_src_ptr);
    }
  }

  // close bit_number image file
  fclose(fd_bit_number_image_save);
  printf("bit_number_image_save done.\n");
  fflush(stdout);
}

void save_stego_image_to_sd_card(void)
{
    unsigned int i, j;
    FILE* fd_stego_image_save;
  // create or open existing original image data file
```

```
  fd_stego_image_save =
       fopen(STEGO_IMAGE_FILE_NAME, "w+");
  if (fd_stego_image_save == NULL)
  {
    printf("Error opening or creating file!\n");
    exit(1);
  }

  // our src is buffer
  file_wrt_src_ptr = stego_im_addr;
  printf("file_wrt_src_ptr =%p.\n", file_wrt_src_ptr);

  // write all pixel values on file
  for(j = 0; j < (FRAME_W); j++)
  {
    for(i = 0; i < (FRAME_H); i++)
    {
      file_wrt_src_ptr =
                stego_im_addr + i * FRAME_W + j;
      fprintf(
                fd_stego_image_save,
                "%c",
                    *file_wrt_src_ptr);
    }
  }
  printf("file_wrt_src_ptr 2 =%p.\n", file_wrt_src_ptr);
  // close original image file
  fclose(fd_stego_image_save);
  printf("fd_stego_image_save done.\n");
  fflush(stdout);
}

void release_mems(void)
{
  // Release embedding_map
    free(symmetry_map_addr_rows);
  // Release embedding_map
  free(symmetry_map_addr_columns);
  // releasing virtual memory for reader
  if(munmap(map_base_reader, MAP_SIZE_READER) == -1)
  {
    FATAL;
  }
  // Release result buffer
  free(stego_im_addr);
  // Release embedding_map
  free(embedding_map_addr);
  // Closing memory file
  close(fd_mem);
}
```

**CURRICULUM VITAE**

**Name Surname: Utku ESEN**

**Place and Date of Birth: Gölcük/Kocaeli - Turkey 04.05.1991**

**E-Mail: esenu@itu.edu.tr**

**EDUCATION:**

- **B.Sc.:** 2013, Kocaeli University, Engineering Faculty, Electronics and Communication Engineering Department
- **M.Sc.:** 2018, Istanbul Technical University, Electronics and Telecommunication Engineering Faculty, Electronics Engineering Department

**PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:**

- Utku Esen, S. Berna Örs Yalçın, Data Hiding Method Using Image Interpolation And Pixel Symmetry, *9th International Conference on Electrical and Electronics Engineering(ELECO 2015)*, November 27, 2015 Bursa, Turkey.
- Utku Esen, S. Berna Örs Yalçın, Stefano Mattoccia, Information Hiding Technics For Digital Images And Example Embedded System Application, *Hacktrick 2017-National Conference On Cyber Security*, April 28, 2017 Ankara, Turkey.