

**FPGA IMPLEMENTATION OF FAULT TOLLERENT REVERSIBLE CONTRAST  
MAPPING TECHNIQUE**

*by*

**Mahmudul Hasan**

**MASTER OF ENGINEERING  
IN  
INFORMATION AND COMMUNICATION TECHNOLOGY**

**Institute of Information and Communication Technology  
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY  
2017**

The project titled “FPGA Implementation of Fault Tollerent Reversible Contrast Mapping Technique” submitted by Mahmudul Hasan, Roll No. 0411312008, and Session April-2011 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Master of Engineering in Information and Communication Technology (ICT) at 27/09/2017.

#### **BOARD OF EXAMINERS**

A handwritten signature in black ink, appearing to read "Mahmudul Hasan".

Chairman

1. Dr. Md. Liakot Ali  
Professor  
Institute of Information and Communication Technology  
BUET, Dhaka-1000.

A handwritten signature in blue ink, appearing to read "Harun-Ur-Rashid".

Member

2. Dr. A. B. M. Harun-Ur-Rashid  
Professor  
Dept. of Electrical and Electronic Engineering  
BUET, Dhaka-1000.

A handwritten signature in blue ink, appearing to read "Hossen Asif".

Member

3. Dr. Hossen Asiful Mustafa  
Assistant Professor  
Institute of Information and Communication Technology  
BUET, Dhaka-1000.

## **Candidate's Declaration**

It is hereby declared that this project or any part of it has not been submitted elsewhere for the award of any degree or diploma.



---

Mahmudul Hasan

*Dedicated to Almighty Allah*

## Contents

<b>ACKNOWLEDGEMENT .....</b>	<b>VIII</b>
<b>ABSTRACT .....</b>	<b>IX</b>
<b>CHAPTER 1 .....</b>	<b>1</b>
<b>Introduction .....</b>	<b>1</b>
1.1 Introduction .....	1
1.2 Objectives .....	3
1.3 Organization of the Project .....	3
<b>CHAPTER 2 .....</b>	<b>4</b>
<b>Fundamentals of Digital Watermarking.....</b>	<b>4</b>
2.1 Introduction .....	4
2.2 Elements of watermarking system: .....	4
2.3 Image Description .....	5
2.4 Types of Image.....	6
2.5 Uses of Digital Watermarking .....	8
2.5.1 Data Hiding .....	8
2.5.2 Information Integration .....	8
2.5.3 Intellectual Property Rights (IPR) Protection .....	9
2.5.4 Ownership Demonstration .....	10
2.5.5 Tampering Verification.....	11
2.6 Types of digital watermarking techniques .....	11
2.6.1 Visible and Invisible .....	11
2.6.3 Robust, Semi-Fragile, and Fragile .....	13
2.6.4 Blind and Non-blind.....	14
2.6.5 Spatial, Transform, and Quantization .....	14
2.6.6 Reversible and Non-reversible.....	15
2.6.7 Public and Private .....	15
2.6.8 Symmetric and Asymmetric.....	16
2.7 Reversible Contrast Mapping Technique.....	16
2.7.1 Data Embedding Procedure: .....	18
2.7.2 Data Embedding Process: .....	19
2.7.3 Data Retrieving Process: .....	19
2.8 Overview of FPGA .....	24
2.8.1 Advantages of FPGAs.....	25
<b>CHAPTER 3 .....</b>	<b>27</b>
<b>Design and Implementation of Reversible Contrast Mapping Technique .....</b>	<b>27</b>

3.1 Introduction .....	27
3.2 Architecture of the Design .....	27
3.2.1 Controller .....	28
3.2.2 Image Memory Module.....	29
3.2.3 Text Memory.....	29
3.2.4 RCM Converter Module and Text Embedding Module .....	29
3.3 Extraction Unit.....	29
3.3.1 Bit to Text Converter Module .....	29
3.3.2 Image memory module .....	30
3.3.3 Bit Extraction Module.....	30
3.3.4 Bit to text conversion module .....	31
3.4 Verilog HDL (Hardware Definition Language).....	31
3.4.1 Modelsim Simulator.....	32
3.4.2 Simulation Mode.....	33
<b>CHAPTER 4 .....</b>	<b>36</b>
<b>Results and Discussion.....</b>	<b>36</b>
4.1 Introduction .....	36
4.2 Software Simulation.....	36
4.3 Result .....	42
4.4 Text Embedding.....	42
4.5 Image Quality.....	43
4.6 More Implemented Bench Mark Image .....	46
<b>CHAPTER 5 .....</b>	<b>49</b>
<b>Conclusion.....</b>	<b>49</b>
5.1 Conclusion .....	49
5.2 Suggestions for Further Works .....	50
<b>REFERENCES .....</b>	<b>51</b>
<b>APPENDIX .....</b>	<b>54</b>

## LIST OF FIGURES

FIGURE 2.1: ELEMENTS OF WATERMARKING SYSTEM.....	5
FIGURE 2.2: IMAGE COORDINATE SYSTEM FOR ANY IMAGE .....	7
FIGURE 2.3: EXAMPLE OF DATA HIDING .....	8
FIGURE 2.4: EXAMPLE OF INFORMATION INTEGRATION USING READABLE WATERMARKING .....	9
FIGURE 2.5: AN EXAMPLE OF OWNERSHIP DEMONSTRATION USING DIGITAL WATERMARKING..	10
FIGURE 2.6: EXAMPLE OF VISIBLE AND INVISIBLE WATERMARKING TECHNIQUE.....	12
FIGURE 2.7: THE DETECTABLE AND THE READABLE WATERMARKING SYSTEMS.....	13
FIGURE 2.8: TRANSFORM DOMAIN D (A) WITHOUT AND (B) WITH CONTROL DISTORTION ....	19
FIGURE 3.1: FUNCTIONAL BLOCK DIAGRAM OF PROPOSED DESIGN.....	28
FIGURE 3.2: FLOWCHART OF THE PROPOSED DESIGN OF THE RCM ENCODER AND DECODER. ...	31
FIGURE 3.3: EXAMPLE OF A PROGRAM IN MODELSIM. .....	34
FIGURE 4. 1: SOURCE IMAGE (LENA).....	37
FIGURE 4. 2: RUNNING THE WATERMARKING.M FILE IN MATLAB 2015A. ....	38
FIGURE 4. 3: TEXT TO BE EMBEDDED. .....	38
FIGURE 4. 4: TEXT EMBEDDING COMPLETE. .....	39
FIGURE 4. 5: WATERMARKED IMAGE. .....	39
FIGURE 4. 6: RUNNING REVERSE_WATERMARKING.M.....	40
FIGURE 4. 7: REVERSE_BIN2DEC.M FILE RESULTS. ....	41
FIGURE 4. 8: THE ORIGINAL IMAGE RETRIEVED. ....	41
FIGURE 4. 9: (A) ORIGINAL IMAGE (B) TEXT WATERMARKED IMAGE .....	43
FIGURE 4.10: (A) ORIGINAL IMAGE (B) TEXT WATERMARKED IMAGE .....	43
FIGURE 4.11: MORE IMPLEMENTED BENCH MARK IMAGES. ....	47

## LIST OF TABLES

TABLE 2.1: COVERED IMAGE PIXEL .....	20
TABLE 2.2: PIXEL PAIRS.....	20
TABLE 2.3: READY FOR TEXT EMBEDDING .....	21
TABLE 2.4: EMBEDDING THE TEXT .....	22
TABLE 2.5: WATERMARKED IMAGE'S PIXEL VALUES .....	22
TABLE 2.6: WATERMARKED PIXEL PAIRS AND TEXT BIT EXTRACTION.....	23
TABLE 2.7: IMAGE AFTER EXTRACTION.....	23
TABLE 2.8: REVERTED IMAGE .....	24
TABLE 4.1: COMPARISON OF TWO TEST IMAGE FOR TEXT EMBEDDING .....	42
TABLE 4.2: QUALITY RESULTS OF EMBEDDED IMAGES. ....	45
TABLE 4.3: QUALITY RESULTS OF EMBEDDED IMAGES (LENA).....	45
TABLE 4.4: ACCURACY RESULTS OF TEXT IN EMBEDDED IMAGES (LENA). ....	45

## **LIST OF ABBREVIATIONS OF TECHNICAL SYMBOLS AND TERMS**

RCM	Reversible Contrast Mapping
2D	Two Dimensional
DVD	Digital Video Disk
UniSA	University of South Australia
LSB	Least Significant Bits
IPR	Intellectual Property Rights
DWT	Discrete Wavelets Transform
Eq.	Equation
VQ	vector quantization
MSE	Mean Square Error
PSNR	Peak Signal to Noise Ratio
NIQE	Natural Image Quality Evaluator
NR	No Reference
IQA	image quality assessment
FPGA	Field Programmable Gate Array
FPL	Field Programmable Logic
Gbps	Giga bits per second
HDL	Hardware Description Language
LUT	Look-Up Table
NIST	National Institute of Standards and Technology
PDA	Personal Digital Assistant
PLD	Programmable Logic Device
RAM	Random Access Memory
ROM	Read Only memory
VHDL	Very High Speed Integrated Circuit Hardware Description Language

## **Acknowledgement**

During the time I was performing my master's project, I came across many people who have supported and assisted me. First of all, I would like to thank the almighty Allah for giving me the opportunity to conduct this project. I would like to express my heartiest thanks to my supervisor, Professor Dr. Md. Liakot Ali, for giving me the opportunity to do my master's project under his supervision. I am also grateful to him for all his support, advice and encouragement throughout this project.

I gratefully acknowledge the valuable advice and support from Professor and Director Dr. Md Saiful Islam, and all other faculty members of IICT, BUET.

I also express my gratitude to Bangladesh University of Engineering and Technology for allowing me to conduct the research project using its all kinds of facilities.

Finally, I want to thank all my friends and family members for their continuous support and inspirations, making this work a nice experience.

# **Abstract**

In this era of Information and Communication Technology, copyright protection of digital content has become a burning issue due to rapid development in technology. Watermarking is one of the excellent choices for copyright protection problem. It is basically methods and technologies that hide information, for example a number or text, in digital media, such as images, video or audio. It has numerous applications such as copyright protection of digital media, authentication, certification, tamper detection of digital contents, etc. This research project focuses on text watermarking using Reversible Contrast Mapping (RCM) Reversible Watermarking (RW) technique since it offers superiority over all other existing digital watermarking technique in terms of high embedding rate at relatively low visual distortion (embedding distortion), low computation cost and ease of hardware realization. In this project, FPGA implementation of the RCM reversible technique is proposed to make the system fault tolerant. The proposed RCM RW system is designed and simulated in MATLAB environment. Two bench mark images have been watermarked using the proposed system. MATLAB simulation results on benchmark images ensure the accuracy of the functionality (embedding and decoding the original image) of the RCM RW system. One of the highlighting features of our system is that it is capable of embedding character into the digital media which is more user-friendly to the users in comparison with embedding bits. Performance of the proposed system in terms of accuracy, data embedding capability and image quality has been compared with that of other works which proves the superiority of our system over existing RCM RW techniques. Hardware architecture of the proposed system has been identified which can be realized in the future research.

# Chapter 1

## *Introduction*

### **1.1 Introduction**

In this era of information and communication technology, copyright protection of digital content becomes a serious problem due to rapid development in technology. Watermarking can be an alternative solution in this regard. Data hiding system is another issue in this modern era. In watermarking system, data is watermarked in a covered media by storing the data in a convenient manner by which data can be hidden. Storing information and data such as documents, images, video, and audio in digital formats is very much common and necessary. For many people, transferring digital files via the Internet is a daily activity. Owing to the rapid development of digital technology and the widespread use of the Internet, life becomes increasingly more convenient than previously. However, accompanying this convenience, more serious problems are more prevalent. As is well known, due to the nature of digital information, it is easy to make unlimited lossless copies from the original digital source, to modify the content, and to transfer the copies rapidly over the Internet. Therefore, the demands of copyright protection, ownership demonstration, and tampering verification for digital data are becoming more and more demanding. Among the solutions for these problems, digital watermarking [1] is the most popular one. Researchers have given consideration to this in the past decade.

Digital Watermarking [1-5] is a type of technology in which a pattern of bits are inserted into a digital image, audio or video file so that, the pattern of bits which are inserted (called as watermark) contains the file's copyright information like profile of owner, rights of owner and other important information. The name watermark comes from the faint visible watermark which is imprinted on products that identify the manufacturer of the product [6-9]. The purpose of the digital watermarks mainly is to provide copyright protection for intellectual property that is in digital format like the multimedia objects in internet. The multimedia objects can be obtained from desktop PCs, mobile phones, laptops and tablets.

Digital watermarks may be used to verify the authenticity or integrity of the original data. In present days, it is prominently used for tracing copyright infringements and for banknote authentication. Digital watermarking is broadly classified depending on the type of signal like audio watermarking, image watermarking, video watermarking and database watermarking [10-12]. Unlike printed watermarking which are somewhat visible like the TV channel logo, digital watermarks are designed to be completely invisible or in the case of audio clips, inaudible. Generally the actual bits representing the watermark must be scattered throughout the file in a manner that they cannot be identified and manipulated. Also, the digital watermark must be robust enough so that it can withstand normal changes to the file, such as reductions from lossy compression algorithm. To satisfy these requirements we generally make the watermark appear as noise; noise is random data that exists in most digital files normally. To view the watermark, we need a special program or algorithm that knows how to extract the watermark data. Watermarking is also called data embedding and information hiding. Our work is focused on reversible image watermarking and its hardware architecture.

The general framework of digital watermarking consists of three modules: an encoder, a decoder and a comparator. But the main disadvantage in digital watermarking is that it needs the original media to extract watermark during decoding. This problem was overcome by using reversible watermarking technique. The concept of reversible watermarking was first introduced in a U.S. Patent from Barton in 1997 and U.S patent from Honsinger in 1999. Reversible watermarking (RW) is a novel category of watermarking scheme [1]. It is not only can strengthen the ownership of the original media but also can completely recover the original media from the watermarked media. Literature on Reversible watermarking are very rich containing a large number of different concepts, namely difference expansion (DE), least significant bits (LSB) modification, data compression, histogram modification, reversible contrast mapping (RCM) based approach, prediction error adjustment, etc, along with their various modified forms. Among them, DE and RCM is quite popular due to the low cost implementation and high embedding capacity [5]. It is perfectly invertible even if the LSBs of the transformed pixels are lost during embedding process. Spatial domain approaches are better suited for hardware realization due to simplicity and low computational burden. A trend in recent times is seen for hardware implementation of digital watermarking on image and video signals [6, 7]. Hardware implementations offer advantages over software realization in

terms of less area, low execution time, low power, real-time performance, high reliability and also ease of integration with the existing consumer electronic devices [8]. The software designer does not have any direct control over the way with which random access memory (RAM) and processors interact. This poses a limit on operating speed. A software designer must try to limit the total amount of RAM required, while a hardware designer has full control over timing operations into the RAM and direct control over the usage of expensive hardware resources.

A hardware based implementation can be designed on a field programmable gate array (FPGA) board or application specific integrated circuit (ASIC).

## ***1.2 Objectives***

The aim of this project is to develop a digital watermarking system using modified Reversible Contrast Mapping Scheme. To fulfill this aim, we have the following objectives:

- To design RCM based RW system under MATLAB environment
- To simulate the design using suitable tool
- To identify hardware blocks for FPGA implementation

## ***1.3 Organization of the Project***

**Chapter 1** of this report starts with the importance of elementary function followed by a brief background of latest research work in this area.

**Chapter 2** of this report describes the details of Watermarking Techniques and its need.

**Chapter 3** of this report describes main procedure of watermarking system and Complete Example with simulation result, it also describes encoding and decoding unit with example. All procedures are described step by step with proper examples in this chapter.

**Chapter 4** of this report describes results and performance analysis of text watermarking with various aspects.

**Chapter 5** of this report describes conclusion and future work of the research.

# Chapter 2

## *Fundamentals of Digital Watermarking*

### **2.1 Introduction**

This chapter describes fundamental principles of watermarking techniques. Watermarking is a method to insert or embed extra information into the covered media, and also to indicate the method used to obtain the embedded information. A short description of digital image and the classification of watermarking, the concept of modified RCM watermarking [5] systems are introduced in the following sections. The general definitions of some common terms used in the area of watermarking are listed below.

**Watermark:** Watermark is the information to be hidden. The term watermark also contains a hint that the hidden information is transparent like water.

**Cover Media:** Covered media is a media that is used for carrying the watermark. Sometimes the terms original media and host media are used to express it.

**Watermarked Data:** The media which contains the watermark is called watermarked Data.

**Embedding:** The procedure used for inserting the watermark into the cover media is called embedding.

**Extraction:** The procedure used for extracting the embedded watermark from the watermarked data is called extraction.

**Detection:** The procedure used for detecting whether the given media containing a particular watermark is called detection.

**Watermarking:** The method which contains the embedding operator and the extraction/detection operator is called watermarking method.

**Noise:** The natural noise occurred to the watermarked data during transmission.

### **2.2 Elements of watermarking system:**

According to literature [17], a watermarking system is regarded as a communication system consisting of three main parts: a transmitter, a communication channel, and a receiver, as illustrated in Figure 2.1.

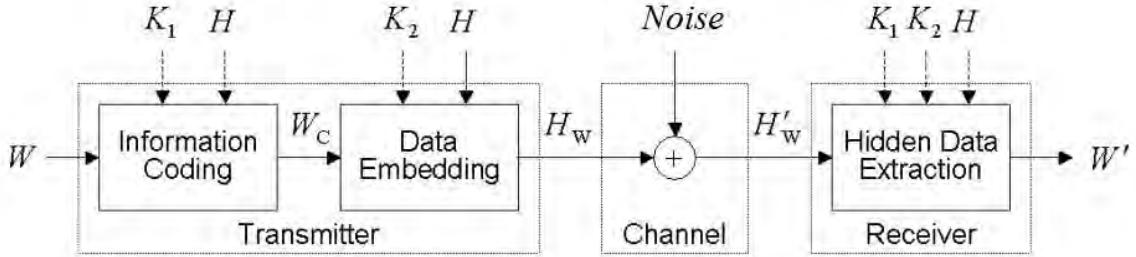


Figure 2.1: Elements of watermarking system

The information-coding procedure encodes, compresses, and/or encrypts the original watermark  $W$  according to a user key  $K_1$ . The data-embedding procedure then embeds the encoded result  $W_C$  into the host data  $H$  according to another user key  $K_2$ . The watermarked data  $H_W$  is then delivered to the receiver via some kind of channel. During the transmission, some natural noise may occur; as a result the received data  $H'_W$  of the receiver may be different from the output data  $H_W$  of the transmitter. To recover the information hidden in  $H'_W$ , the hidden-data-extraction procedure is executed. In the above systems, either  $K_1$ ,  $K_2$ , or  $H$  may or may not have to be published, according to the algorithms used.

### 2.3 Image Description

A digital image is a representation of a two-dimensional image using ones and zeros (binary). Depending on whether or not the image resolution is fixed or not, it may be of vector or raster type. The term ‘digital image’ usually refers to raster images also called bitmap images. Raster images have a finite set of digital values, called picture elements or pixels. The digital image contains a fixed number of rows and columns of pixels. Pixels are the smallest individual element in an image, holding quantized values that represent the brightness of a given color at any specific point. Typically, the pixels are stored in a computer memory as a raster image or raster map, a two-dimensional array of small integers. Raster images can be created by a variety of input devices and techniques, such as digital cameras, scanners, coordinate-measuring machines, seismographic profiling, airborne radar, etc. They can also be synthesized from arbitrary non-image data, such as mathematical functions or three-dimensional geometric models; the latter being a major sub-area of computer graphics.

## **2.4 Types of Image**

There are different image representation techniques in computer graphics, a raster graphics image or bitmap is a data structure representing a generally rectangular grid of pixels, or points of color viewable via monitor, paper or other display medium. Raster images are stored in image files with varying formats. A bitmap is technically characterized by the width and height of the image in pixels and by the number of bits per pixel (a color depth, which determines the number of colors it can represent). Each pixel of raster image is typically associated to a specific position in some 2D region, and has value consisting of one or more quantities (samples) related to that position. Digital images can be classified according to the number and nature of those samples: (i) Binary, (ii) Gray Scale, (iii) Color, (iv) False-color, (v) Multi-spectral, (vi) Thematic, (vii) Picture function. [21]

**A binary image** is a digital image that has only two possible values for each pixel. Typically the two colors used for a binary image are black and white. Here, any two colors can be used. The color used for the object(s) in the image is the foreground color while the rest of the image is the background color. Binary images are also called bi-level or two-level. This means that each pixel is stored as a single bit (0 or 1). The names black-and-white, B&W, monochromatic are often used for this concept.

**A gray-scale digital image** is an image in which the value of each pixel is a single sample, it carries only intensity information. It is composed exclusively of shades of gray, varying from black at the weakest intensity to white at the strongest. Gray scale images are distinct from one-bit black and white images, which in the context of computer imaging are images with only the two colors, black and white (also called bi-level or binary images). Gray-scale images have many shades of gray in between. Gray-scale images are also called monochromatic, denoting the absence of any chromatic variation (i.e. no color). Gray-scale images are often the result of measuring the intensity of light at each pixel in a single band of the electromagnetic spectrum (e.g. infrared, visible light, ultraviolet, etc.), and in such cases they are monochromatic proper when only a given frequency is captured. But also they can be synthesized from a full color image. For our project, our concern is gray-scale image. The test image we had selected is ‘Lena’. Lena is the name given to a standard test image widely used in the field of image processing since 1973.

**A (digital) color image** is a digital image that includes color information for each pixel. For visually acceptable results, it is necessary (and almost sufficient) to provide three samples (color channels) for each pixel, which are interpreted as coordinates in some color space. The RGB color space is commonly used in computer displays, but other spaces such as YCbCr, HSV are often used in other contexts. A color image is usually stored in memory as a raster map, a two-dimensional array of small integer triplets; or (rarely) as three separate raster maps, one for each channel. Eight bits per sample (24 bits per pixel) seem to be adequate for most uses, but particularly demanding application may use 10 bits per sample or more.

**A false color image** is an image that depicts subjects in color that differ from those a full-color photograph would show. A true-color image of a subject is an image that appears to the human eye just like the original subject would: a green tree appears green in the image, a red apple red, a blue sky blue, etc. When applied to black-and-white images, true-color means that the perceived lightness of a subject is preserved in its depiction. Absolute true-color is impossible to achieve due to the differences between the chemistries of the display media and an eye.

**A picture function** is a mathematical representation of a two-dimensional image as a function of two spatial variables. The function  $f(x,y)$  describes the intensity of the point at coordinates  $(x,y)$ . In general any digital image can be represented in two dimension by the spatial coordinates  $x$  and  $y$  and as a function of  $x$  and  $y$ . If we think of a matrix of size  $M \times N$  or an array of dimension 2 of row size  $M$  and column size  $N$ , then each element of the array represents a pixel of the image. The coordinate system is shown in Figure 2.2.

	1	2	3	.....	M
1					
2		$(x-1,y-1)$	$(x-1,y)$	$(x-1, y+1)$	
3		$(x, y-1)$	$(x,y)$	$(x, y+1)$	
4		$(x+1,y-1)$	$(x+1,y)$	$(x+1,y+1)$	
.					
.					
.					
N					

Figure 2.2: Image Coordinate system for any image

## **2.5 Uses of Digital Watermarking**

In the past decade, digital watermarking had been investigated and utilized for a number of different purposes. In this section, we present some common areas served by digital watermarking in order to highlight its usefulness.

### **2.5.1 Data Hiding**

As described previously, digital watermarking can be used for data hiding. For example, a secret message such as “Meet at 13:30, old place” may be regarded as the watermark and hidden in a cover picture, as shown in Figure 2.3. The watermarked picture is then delivered to the receiver without becoming obvious.

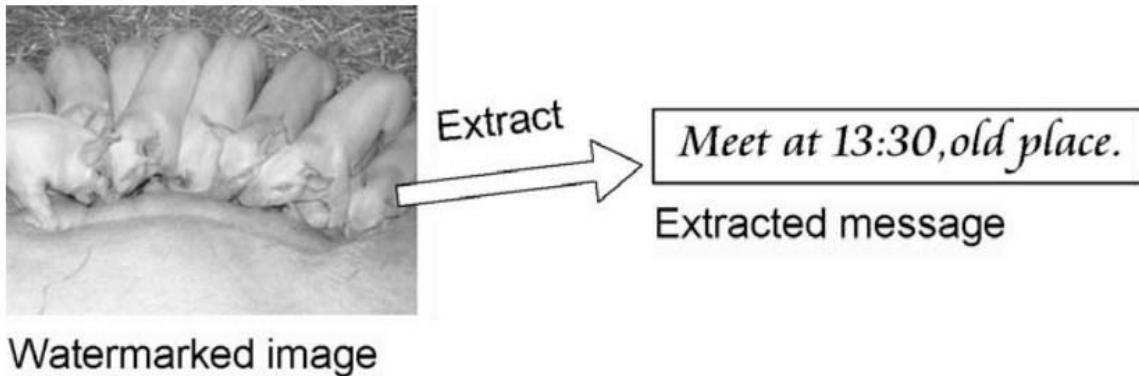


Figure 2.3: Example of data hiding

### **2.5.2 Information Integration**

People sometimes wish to attach information such as the title, the date, and the place to a digital photo for example. This information is sometimes stored in separate digital files attached to the original photo, and is sometimes saved within the photo directly. Figure 2.4 shows the examples of these two methods used on a digital photo. For the first method, the extra separate file sometimes causes problems, such as not easy to maintain. In the second method, the added comments sometimes cause unacceptable degradation. To solve these problems, invisible and readable watermarking techniques can be employed.

Similar to the first application that uses watermarking for the purpose of data hiding, the comments or notes for the photo are regarded as a watermark. It is then embedded into the original photo using the invisible and readable watermarking technique. This achieves the goal of information integration.

Saving the extra information in the header of the picture file (for example, photo saved in JPEG format allows extra information to be stored in its EXIF section of the header), digital watermarking techniques embed the information within the graphic data itself.

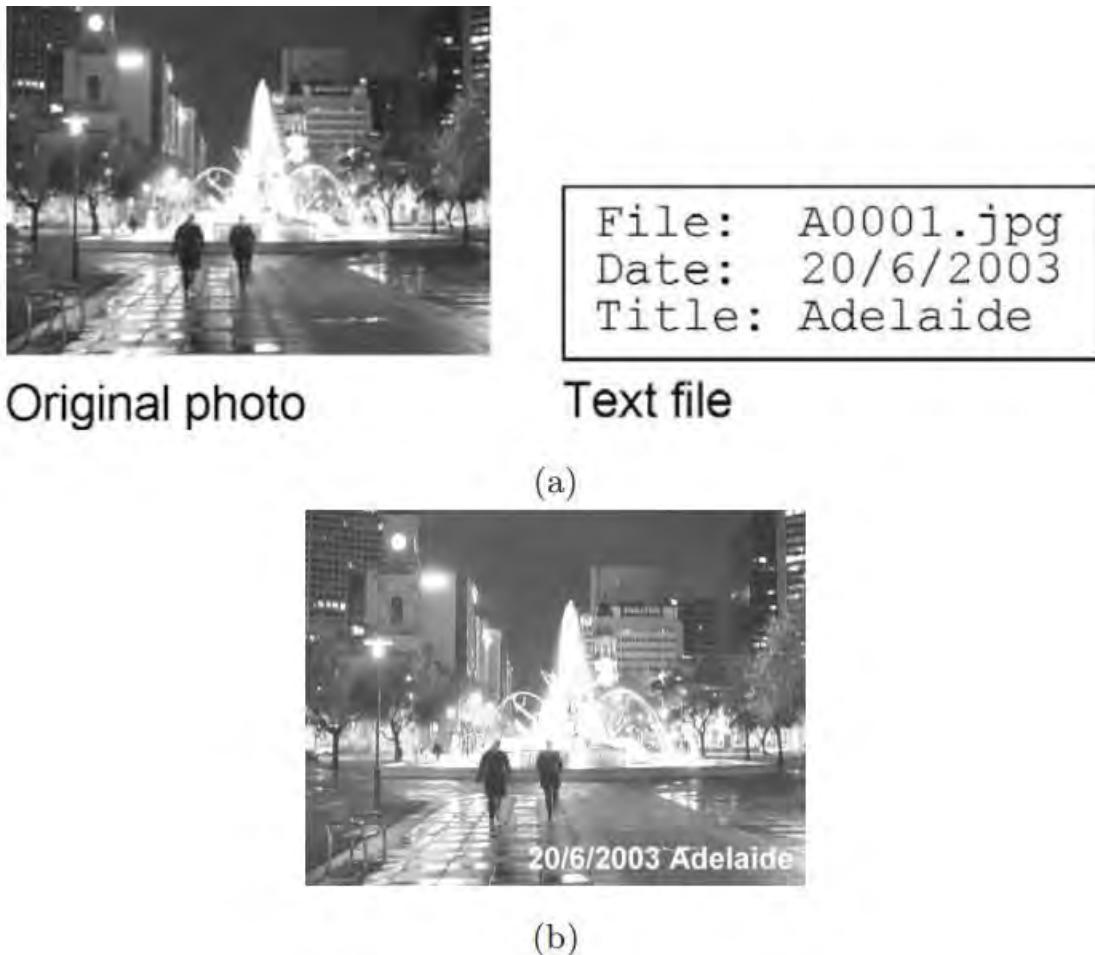


Figure 2.4: Example of information integration using readable watermarking

### 2.5.3 Intellectual Property Rights (IPR) Protection

Due to the popularity of the Internet, the rapid-development of digital techniques, and the easy-distribution of digital assets, digital watermarking is used to protect intellectual property

rights. The creators or legitimate owners of the digital works who want to protect their rights can insert watermarks within their works. Users who plan to make illegal copies of the watermarked works will not be successful because the machines detect the copyright watermarks and abort the copying procedure.

To implement this concept upon digital products such as DVD or computer software, some related issues such as industry establishment of standards, legislation, and cooperation of hardware manufacturers, have to be established. Currently, many issues relating to IPR protection are still proceeding.

#### 2.5.4 Ownership Demonstration

Another classical use is ownership demonstration. The authors of the digital works can embed their own logos or marks within their works to show who the creators or owners are, no matter whether their watermarked works have been modified or not. An example is shown in

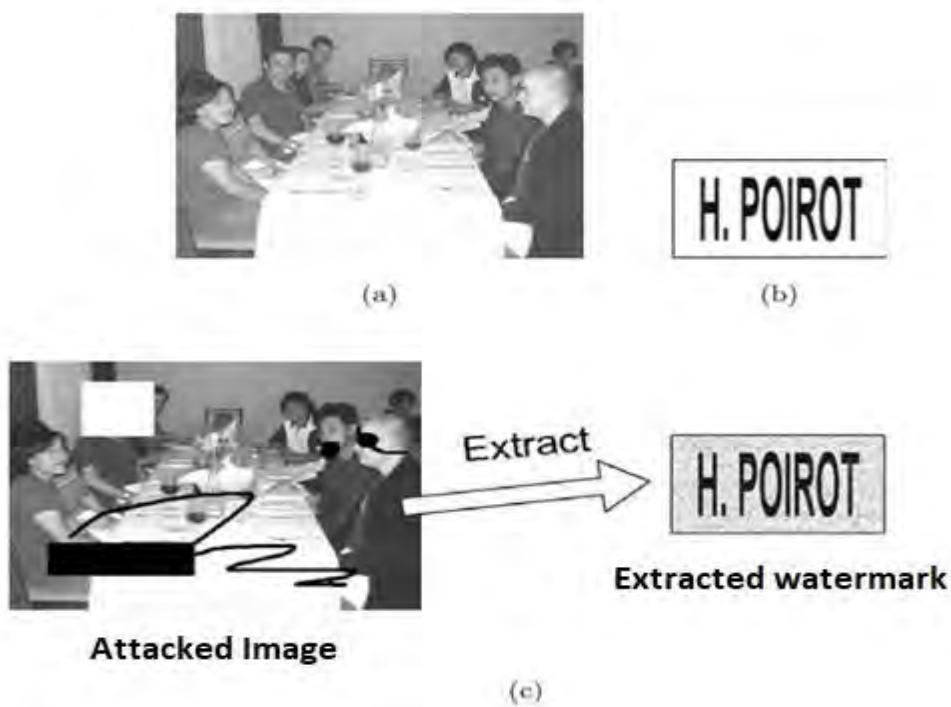


Figure 2.5: An example of ownership demonstration using digital watermarking.

Figure 2.5 which demonstrates such an application where the creator of Figure 2.5(a) embeds his name (Figure. 2.5(b)) within this photo. Afterwards, even someone modifies this watermarked photo; the creator can still extract his name from the attacked photo to show the creator of the photo is H. Poirot (Figure. 2.5(c)). In addition, as the watermarked image has been modified, the extracted watermark would contain distortion.

### **2.5.5 Tampering Verification**

Today modifying or tampering with a digital source using computers is not difficult. Consequently, information technology experts warn people not to trust completely what they see in digital form. For the users who want to know whether the data are trust worthy, fragile watermarking techniques provide a possible solution.

For example, say Bob wants to send a digital file to Alice. He embeds a fragile watermark in the file and delivers it to Alice by a channel which could be the Internet. Before Alice receives the file, John happens to obtain the watermarked file. He modifies the content of the file and sends it to Alice afterwards. When Alice receives the corrupted file, she has no idea as to whether the content is dependable. She therefore verifies if the received file contains the watermark. Due to the fragile nature of the watermark, it has weak resistance against tampering; Alice is unable to find any watermark. She knows immediately that the file she has received had been tampered with.

## ***2.6 Types of digital watermarking techniques***

In this section, all the existing watermarking techniques are classified into several categories according to different points of view [17-19].

### **2.6.1 Visible and Invisible**

From the view point as to whether the embedded watermark can be seen by bare human eyes or not, all watermarking techniques can be classified as visible technique and invisible technique. For example, Figure 2.6(a) is an image, Figure 2.6(b) is the logo of University of South Australia and Figure 2.6(c) shows a picture which contains a visible logo of University of South Australia (in short, UniSA) in its top-left corner and Figure 2.6(d) shows a picture

which contains an invisible watermark.

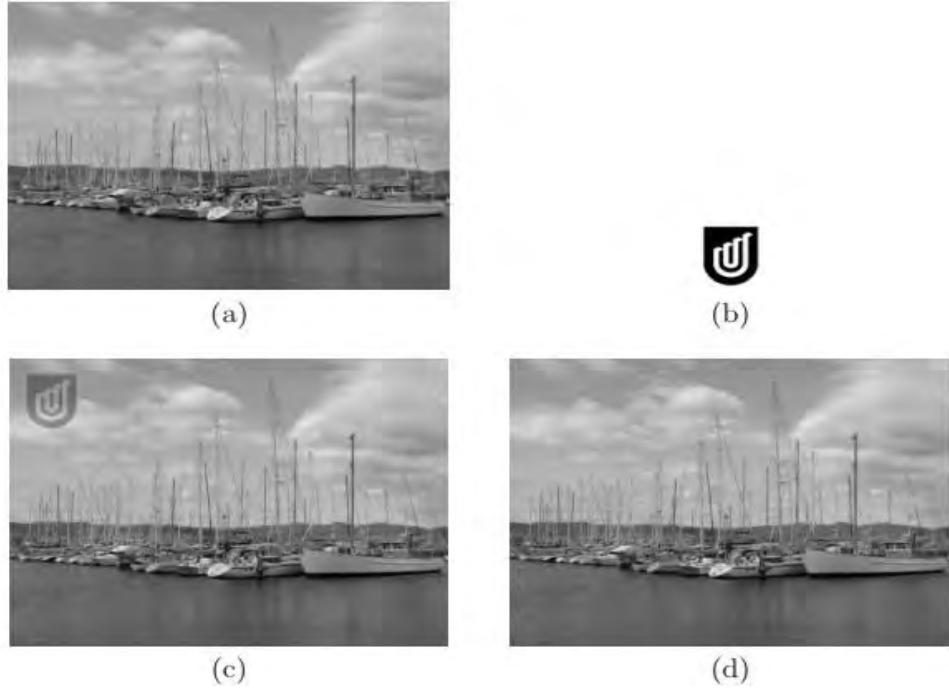


Figure 2.6: Example of visible and invisible watermarking technique.

Obviously, at least two disadvantages exist in visible watermarking techniques:

- (i) The visible watermark is not difficult to be removed. The methods proposed in [22] or the image-cropping schemes can be used for example.
- (ii) The visible watermark degrades the visual quality of the host picture. In the invisible type of watermarking techniques, the embedded watermark is invisible. It is difficult to distinguish between the original image and the watermarked image. Thus, it is not easy to remove or destroy the embedded watermark without degrading the visual quality of the watermarked image significantly.

### 2.6.2 Detectable and Readable

The watermarking techniques can be classified as detectable techniques and readable techniques on the basis of kind of information obtained from the watermarked data. Figure 2.7 illustrates the general distinction between the two types of techniques. In the detectable type of techniques, one can only verify if a specified signal (the watermark) is contained in the cover work. In other words, the detectable type of systems only gives a binary answer: yes or

no. In contrast, the readable watermarking systems extract and reveal the embedded watermark. Researchers use detection to illustrate the process of obtaining a binary answer, and extraction to express the process of revealing the hidden watermark.

For those techniques which belong to the detectable type, the embedded watermark has to be provided during detection. This kind of technique is more private since it is impossible for an attacker to guess the content of the embedded watermark, especially if the embedded watermark is encrypted beforehand.

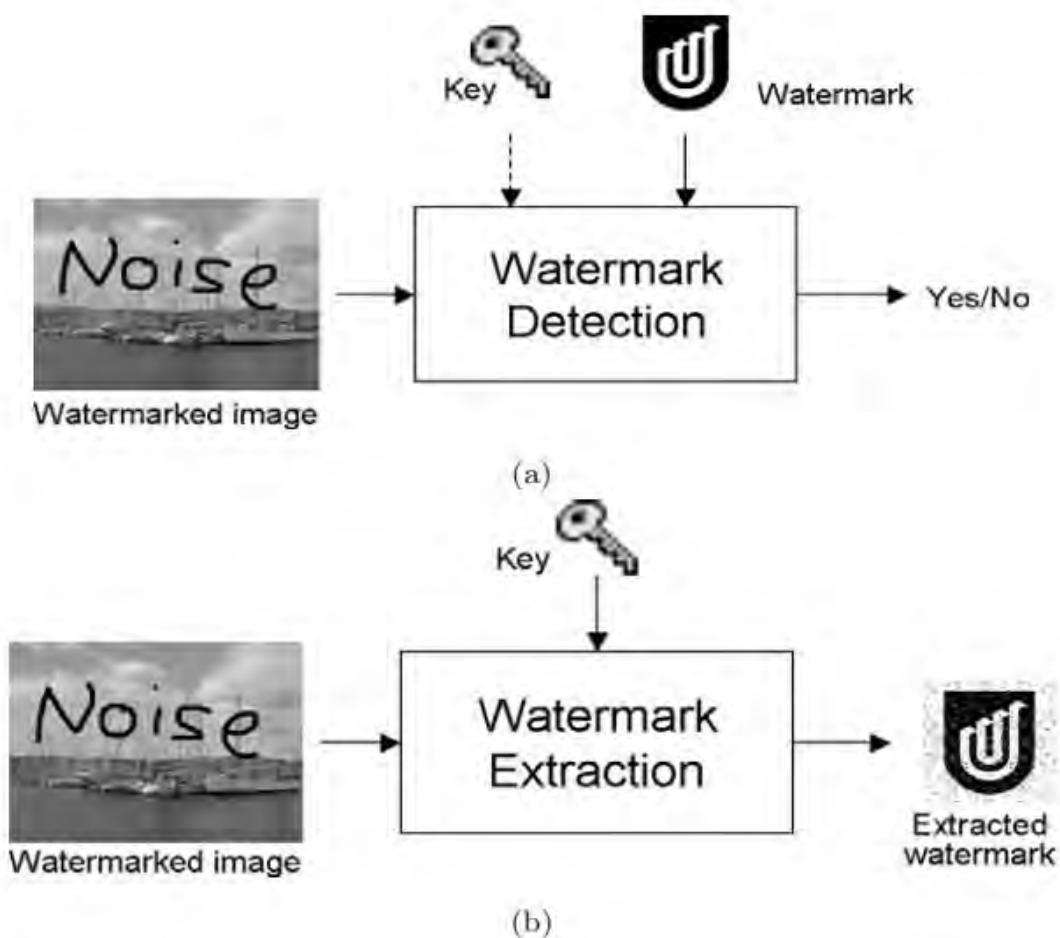


Figure 2.7: The detectable and the readable watermarking systems

### 2.6.3 Robust, Semi-Fragile, and Fragile

Watermarking techniques can also be classified as robust, semi-fragile, and fragile techniques,

according to whether the techniques have strong resistance to natural noise and/or to artificial modification (named attack). If a watermarking technique can detect or extract the hidden watermark successfully from the watermarked data when noise and/or attack occurred, it is called a robust technique. In contrast, a watermarking technique that cannot resist noise or attacks is called a fragile technique.

There are some watermarking techniques which have strong resistance to some kinds of noise or attack but have weak resistance to other kinds of noise or attack. Researchers named these watermarking techniques as semi-fragile techniques.

#### **2.6.4 Blind and Non-blind**

Watermarking can be classified as blind technique and non-blind technique based on comparison between the original non-watermarked data and the watermarked one to recover the watermark. A blind watermarking technique requires no original data for detection or extraction. In contrast, a non-blind watermarking technique requires the original data to be presented during detection or extraction.

In real-world practices, non-blind watermarking algorithms are unsuitable for many practical applications in that they require the non-watermarked data to be presented during extraction or detection. Currently most researchers are focusing on blind watermarking techniques rather than non-blind watermarking techniques.

In addition, definitions of blind and non-blind in nowadays have been extended. Some researchers think that if a watermarking technique requires any information used in embedding for watermark extraction, it then should be classified as non-blind. Based on this definition, one watermarking technique which requires no non-watermarked data but requires the knowledge of embedding position when extraction, is regarded as a non-blind technique.

#### **2.6.5 Spatial, Transform, and Quantization**

Watermarking may be divided into three categories based on where to hide the signal of the watermark: spatial-domain-based techniques, transform-domain-based techniques, and quantization-domain-based techniques. The main concept of spatial-domain-based techniques [23] is to modify the raw data (pixels) of the original host image directly when hiding the

watermark bits. The traditional method is to change the Least Significant Bits (LSB) of certain pixels of the host image according to the watermark bits. For transform-domain-based techniques, the raw data of the host image are first transformed into frequencies using the discrete cosine transform (DCT) [24], the discrete wavelets transform (DWT) [25], or other types of transforms. These frequencies are then modified according to the watermark bits so that the goal of data hiding can be achieved. Then, the inverse transform is executed and a watermarked image is formed. The quantization-domain-based techniques, such as vector quantization (VQ) [27], first quantify the host image using the predefined code-vectors. The indices obtained are then modified according to the watermark bits. The recovery process is finally performed to reconstruct a watermarked image from these modified indices.

Comparing with the three types of techniques, spatial-based techniques possess the advantages including easy implementation, better visual quality, and shorter coding time. However, they also have the disadvantages such as weak robustness for example. The transform-based techniques usually have better robustness and good visual quality in watermarked result. However, they consume more time in the transform and inverse-transform procedures. For the quantization-based techniques, the most significant feature is they enhance the traditional quantization systems the watermarking ability.

### **2.6.6 Reversible and Non-reversible**

A watermarking algorithm is said reversible if the watermarked signal can be converted to a non-watermarked signal after the embedded watermark is extracted. By contrast, watermarking algorithms that cannot convert the watermarked signal to a non-watermarked signal are named non-reversible watermarking algorithms.

Currently, most of the existing watermarking algorithms are non-reversible algorithms, since the selected signals of the cover media have been changed permanently for carrying the watermark bits.

### **2.6.7 Public and Private**

A watermark is named private if only the authorized users can recover it. In other words, it is impossible for unauthorized people to extract the information hidden within the host data. By

contrast, a watermarking technique that allows anyone to read the embedded watermark is referred as a public watermarking technique.

From the view point of information theory, security cannot be based on algorithms but rather on the choice of the user key. Therefore, researchers believe that private watermarking techniques have superior robustness when compared to public watermarking techniques.

#### **2.6.8 Symmetric and Asymmetric**

A watermarking algorithm is called symmetric if the detection/extraction process makes use of the same set of parameters used in the embedding process. Here, the parameters include the secret keys and other information which may be used to define the embedding position and the embedding process. In contrast, a watermarking algorithm is said asymmetric if it uses different keys and parameters for the embedding and the detection/extraction operations.

Researchers believe, for symmetric watermarking technique, knowledge of these parameters is likely to give pirates enough information to remove the watermark from the watermarked data. Therefore, increasing attention has been given to asymmetric watermarking schemes. Generally speaking, asymmetric watermarking algorithms use a private key for watermark embedding and use a public key for watermark detection/extraction.

#### **2.7 Reversible Contrast Mapping Technique**

Reversible watermarking scheme achieves high-capacity data embedding without any additional data compression stage. The scheme is based on the reversible contrast mapping (RCM), a simple integer transform defined on pairs of pixels. In our proposed design, RCM is chosen for text embedding which is also spatial domain based watermarking technique. RCM is perfectly invertible, even if the least significant bits (LSBs) of the transformed pixels are lost. The data space occupied by the LSBs is suitable for data hiding. Reversible contrast mapping (RCM) technique is a low computational watermarking technique because, in the embedding stage, for each pair of pixels, the forward RCM needs two multiplications by two (in fact, simple arithmetical shifts) and two subtractions. This means one multiplication and one subtraction per pixel. No more than two comparisons per pixel are necessary to verify the limit given below in equation (2). There are some low cost logical and bit manipulation



the LSBs of  $m'$  and  $n'$ . It immediately appears that if the LSB of  $m'$  was “1,” the values inside the ceil functions for the computation of and decrease with  $2/3$  and  $1/3$ , respectively. Similarly, if the LSB of  $n'$  was “1,” the corresponding values decrease with  $1/3$  (for the computation of  $m$ ) and  $2/3$  (for the computation of  $n$ ). Except when both LSBs are “1,” the ceil function recovers the correct results. An LSB of “1” means an odd integer number. From (1), it follows that  $(m', n')$  are both odd numbers only if  $(m, n)$  are odd numbers, too. To conclude, on  $D$  without the set of odd pairs, the inverse RCM transform performs exactly; even if the LSBs of the transformed pairs of pixels are lost. The forward transform should not introduce visual artifacts.

By taking the sum and the difference of (1), one gets  $m'+n'=m+n$  and  $m'-n'=3(m-n)$ , respectively. This means that RCM preserves the graylevel averages and increases the difference between the transformed pixels. Consequently, image contrast increases.

### **2.7.1 Data Embedding Procedure:**

Data embedding procedure is very simple. Various types of data embedding procedure is available in [27-28]. We are following the procedure described in [4]. A rhombic region  $D_c$  is also described in [5]. But for convenience we just described here little bit. The watermark substitutes the LSBs of the transformed pairs. At detection, in order to extract the watermark and to restore the original pixels, each transformed pair should be correctly identified. The LSB of the first pixel of each pair is used to indicate if a pair was transformed or not: “1” for transformed pairs and “0” otherwise.

The inverse RCM fails to recover the pairs  $(m,n) \subset D$  composed of odd values. Such pairs can be used as well for data embedding as long as they are correctly identified at detection. This can be easily solved by setting the LSB of the first pixel to “0.” At detection, both LSBs are set to “1” and (2) are checked. If (2) are fulfilled, the pair was composed of odd pixels. In order to avoid decoding ambiguities, some odd pixel pairs should be eliminated, namely, those pairs located on the borders of  $D$ . The pairs subject to ambiguity are found by solving in odd numbers the equations:  $2m-n=1$ ,  $2n-m=1$ ,  $2m-n=255$  and  $2n-m=255$ . For  $L=255$ , there are only 170 such pairs. Let further  $D_c$  be the domain of the transform without the ambiguous odd pixel pairs [4].

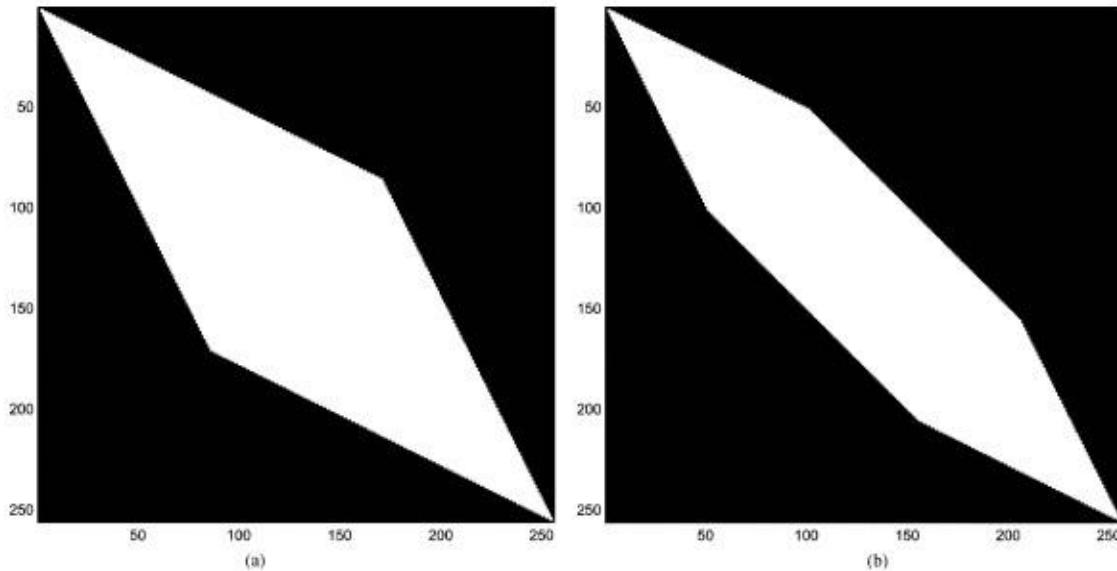


Figure 2.8: Transform domain D (a) without control distortion and (b) with control distortion

### 2.7.2 Data Embedding Process:

Embedding process is as follows.

- a) We have made a partition of image into two columns.
  - i. If  $(m, n) \in D_c$  and if it is not composed of odd pixel do the RCM transform using equation (1). Set LSB of  $m' = 1$  and  $n'$  will be available for data embedding.
  - ii. If  $(m, n) \in D_c$  and if it is composed of odd pixel do the RCM transform using equation (1). Set LSB of  $m= 0$  and  $n$  will be available for data embedding.
  - iii. If  $(m, n) \notin D_c$  set LSB of  $m=0$  and set the original value.
- b) From the available data embedding bit sequence, we embed text's ASCII code bit value to the available data space.
- c) It worked very well.

### 2.7.3 Data Retrieving Process:

This is the opposite of the data embedding process. We had done the following steps.

- a) We had a partition of the embedded image into two columns.
- b) For each pair  $(m', n')$ :

- i. If the LSB of the  $m'$  is 1, take the value of  $n'$  and put it to the detected data sequence. Set LSBs of  $m'$  and  $n'$  to 0 and perform inverse RCM transform.
- ii. If the LSB of the  $m'$  is 0 and the pair of  $(m', n')$  with the LSB of set to 1 belongs to  $D_c$ , put the value of  $n'$  to the detected data sequence and restore the pair by setting LSBs to 1.
- iii. If the LSB of the  $m'$  is 0, set the pair of  $(m', n')$  to 1 which does not belongs to  $D_c$ , the original pair  $(m, n)$  is recovered by replacing the LSB of  $m'$  with the corresponding true value extracted from the watermark sequence.

Finally, we get the data from that image.

From above discussion, we want to provide an example of the embedding and extracting process. Let us consider, we have our cover image pixels  $[0, 255]$  are as follows in Table 2.1.

Table 2. 1: Covered Image Pixel

10	15	55	255
98	60	40	50
80	70	60	53
48	49	90	105

We want to make this image as pixel pair.

So, we had arranged as following in Table 2.2.

Table 2.2: Pixel Pairs

m	n
10	15
98	60
80	70
48	49
55	255
40	50
60	53
90	105

This is pixel pairs  $(m, n)$ . Here, we had taken the odd columns of the image into a column

which is written in m and even columns into another which is written in n. Now, we prepare the image pixels ready for data embedding. For data embedding, we check first pair (m, n) = (10, 15) for data embedding, which satisfies (2.7.2 a) i). So, (m, n) will be = (5, 20) because (10, 15) is not composed of odd pixel. Here, we set LSB of m' is 1. 10's equivalent binary is = 00000101. Here, LSB is 0, so m will here as it is 5. On the other hand, n' is 20. 20's equivalent binary is=00010100. So according to (2.7.2 a) i), n will be 00010100 but LSB of 20 is prepared for data sequence.

For the second pair, (m, n) = (98, 60) satisfies (2.7.2 a) i). So, we have to perform here RCM transformation  $m'=2x98-60=136$ ,  $n'=2x60-98=22$ . Here, binary equivalent of 136 is 10010010. According to (2.7.2 a) i) LSB of 136 will be 1 so the value will be 137. n will be LSB of 22 is available for data sequence.

For the third pixel pair, (m, n) = (80, 70) satisfies (2.7.2 a) i). So RCM transform is  $m'=2x80-70=90$ ,  $n'=2x70-80=60$ . According to (2.7.2 a) i), 90 will become 91 and LSB of 60 will be available for data sequence.

For 4<sup>th</sup> pixel pair, (m, n) = (48, 49) satisfies (2.7.2 a) i). So, RCM transform is  $m'=2x48-49=47$ ,  $n'=2x49-48=50$ . According to (2.7.2 a) i), 47 will become 47 and LSB of 50 will be available for data sequence.

For 5<sup>th</sup> pixel pair ,(m, n) = (55, 255) satisfies (2.7.2 a) iii), so we have to put the LSB of 55 is 0. So, 55 will turn into 54 and keep 255 as it is. 6<sup>th</sup> pixel pair (40, 50) is not odd pixel pair. So, we have RCM transformation here, we get after transform (30, 60).

Table 2.3: Ready for text embedding

m	n	A's ASCII bits
5	20	1
136	22	0
90	60	0
47	50	0
55	255	-
30	60	0
67	46	0
75	120	1

All the values are in Table 2.3.

$7^{\text{th}}$  and  $8^{\text{th}}$  pixel pairs will be same as and  $2^{\text{nd}}$  one of the pairs will be available for data sequence as these pairs satisfy (2.7.2 a) i) after RCM transformation. So we can write the RCM transformed pixels are here. All the values are put into the Table 2.4.

Here,  $n'$  is embedded with text or character A's ASCII bits value.  $n$  is the available data sequence which is already prepared. We put 1 to the 15's LSB value. For the  $2^{\text{nd}}$  pair of pixel we made RCM transformation and we get 137 and 22. But we put here 0 to the 22's LSB. 22's equivalent 8bit binary is 00010110. We put here 0 to the LSB. So decimal 22 remains 22. Nothing is changed.  $3^{\text{rd}}$  pair of pixel is same as  $2^{\text{nd}}$ . We keep (55, 255) as (54, 255) it is because it does not belongs to Dc region. We  $4^{\text{th}}, 5^{\text{th}}, 7^{\text{th}}$  and  $8^{\text{th}}$  pair of pixel are same as the  $1^{\text{st}}$  pixel pair. In the  $8^{\text{th}}$  pixel pair, 105 remain 105 because 1 is inserted to the LSB of 105. So it remains as it is.

Table 2.4: Embedding the text

m	n	A's ASCII bits	m'	n'
5	20	1	5	21
136	22	0	137	22
90	60	0	91	60
47	50	0	47	50
55	255	-	54	255
30	60	0	31	60
67	46	0	67	46
75	120	1	75	121

Now  $(m', n')$  is reverted to its original form is as follows.

Table 2.5: Watermarked image's pixel values

5	21	54	255
137	22	31	60
91	60	67	46
47	50	75	121

This is our watermarked image if we add here the meta data.

Now we have the extraction point. Again the above image's pixels are paired like the embedding process. So we will get following Table 2.6.

Table 2.6: Watermarked pixel pairs and text bit extraction

$m'$	$n'$	Extracted bit	$m$	$n$
5	21	1	10	15
137	22	0	98	60
91	60	0	80	70
47	50	0	48	49
54	255	-	55	255
31	60	0	40	50
67	46	0	60	53
75	121	1	90	105

For the 1<sup>st</sup> pair, LSB of  $m'$  is 0 and belongs to Dc. We extract the LSB of  $n'$  and it is 1. For the second pair of pixel that satisfies (2.7.3 b) i), we extract the bit from  $n'$  and put it to the detected data sequence. After this, we set both the LSB of  $(m', n')$  is 0 and (136, 22) and perform inverse transform by using equation (3) and we get  $(m, n) = (98, 60)$ . For the 3<sup>rd</sup> pixel pair we perform the same thing and we get  $(m, n) = (80, 70)$ .

Table 2.7: Image after extraction

$m$	$n$
10	15
98	60
80	70
48	49
55	255
40	50
60	53
90	105

The fifth pair of pixel does not belong to Dc, so we put 1 to the LSB of 54 and it becomes 55

and we put the same value of  $n'$  as it is. We maintain the rules of detection which is shown in Table 2.6. So, the detected data sequence is 1000001, which is actually A.

The pairs are now transformed to get its original image form and it will be as following Table 2.8.

Table 2.8: Reverted image

98	60	40	50
80	70	60	53
48	49	90	105
98	60	40	50

If we add here the meta data we will get the original image as it is.

## 2.8 Overview of FPGA

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by the customer or designer after manufacturing—hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC). FPGAs can be used to implement any logical function that an ASIC could perform. The ability to update the functionality after shipping, partial re-configuration of the portion of the design and the low non-recurring engineering costs relative to an ASIC offer advantages for many applications.

FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"—somewhat like a one-chip programmable breadboard. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

FPGA is a semiconductor device containing programmable logic components and programmable interconnects. It contains up to thousands of gates. The programmable logic components can be programmed to duplicate the functionality of basic logic gates such as

AND, OR, XOR, NOT or more complex combinational functions such as decoders or simple math functions. In most FPGAs, these programmable logic components (or logic blocks, in FPGA parlance) also include memory elements, which may be simple flip-flops or more complete blocks of memories. These logic blocks and interconnects can be programmed after the manufacturing process by the customer/designer (hence the term "field programmable", i.e. programmable in the field) so that the FPGA can perform whatever logical function is needed.

There are various vendor manufacturers for different types of FPGA chip such as Altera, Xilinx, Lattice Semiconductor, Actel, Quick Logic, Cypress Semiconductor, Atmel, Achronix Semiconductor, etc. Among them, Altera and Xilinx are the most famous FPGA companies since both of the companies have lot of varieties of FPGA device from small number of gate counts to higher number of gate counts. However, Altera devices offer the general benefits of PLDs as innovative architectures, advanced process technologies, state-of-the-art development tools, and a wide selection of mega function. The common advantages of Altera devices include high performance, high-density logic integration, cost-effectiveness, short development cycles with the Quartus II software, mega Core functions, benefits of in-system programming. In this work the FPGA device used is Altera EP2C35F672C6 from Cyclone II family.

### **2.8.1 Advantages of FPGAs**

The main advantage is everything is programmable in FPGA. Here we collect 20 top advantages from various sources of internet (books and website) including Altera website.

1. It is possible to customize the process fully without wasting the resources as what we do in controllers; also FPGA has ability to carry out very big and complex process.
2. FPGA is a customized IC. It can implement most digital logic. BUT CPU performs an operation by instructions. FPGA is more powerful. For instance, we can implement a PCI bridge by FPGA but not by CPU.
3. FPGA is excellent to implement the glue logic of the system of different chips. It really glues all of them together.
4. FPGAs can be much more powerful than any microcontroller.
5. The real advantage of FPGA's are the ease of prototyping, time to market and No

upfront non-recurring expenses (NRE) and low volume to use.

6. A micro-controller won't have enough processing power in most telecommunication applications, especially the data path. In those applications, one needs direct hardwired logic to process the packets. The only choices are FPGA or ASIC.
7. It is reconfigurable and strongly flexible. The limitation of microcontroller is not existed.
8. It can operate at very high clock speeds than to controllers.
9. It is not necessary to think about layout design; it is enough to repair the code only.
10. It is a matter of time: when the micro is not fast enough, we have to go with hardware i.e., super-fast filtering, sub-microsecond timing, fast counters, video signals. And here comes the FPGA's: hardware that we can configure to do exactly what we need.
11. FPGA can be field programmed. But, if the power is turned off, the logic will be lost. If we need to change some logic in our product, we can easily change it.
12. The project on high speed communication protocol can't be implemented only using microcontroller. In such case, FPGA and ASIC are needed.
13. It is widely used in high-speed and real-time processing field. That means speed can be very fast, and multiple control loops can run on a single FPGA device at different rates.
14. FPGA design tools are increasingly available, allowing embedded control system designers to more quickly create and adapt FPGA hardware.
15. Because the processing paths are parallel, different operations do not have to compete for the same processing resources.
16. The configurability of FPGAs can provide designers with almost limitless flexibility.
17. In manufacturing and automation contexts, FPGAs are well-suited for use in robotics and machine tool applications, as well as for fan, pump, compressor and conveyor control.
18. Unlike processors, FPGAs use dedicated hardware for processing logic and do not have an operating system.

# Chapter 3

## ***Design and Implementation of Reversible Contrast Mapping Technique***

### **3.1 Introduction**

In chapter 2 the brief description of the RCM technique is provided. In this chapter, the procedure of the text embedding technique will be described. Functional block diagram, detail discussion of individual process with a flowchart and complete mathematical example will also be described in this chapter. Finally, complete simulation process using Matlab tools with proper examples and discussion regarding various tools with their usage procedure will also be described at the end of the chapter.

### **3.2 Architecture of the Design**

The method comprises of two main functional units— embedding/encoding unit and extraction unit. Each unit has different functional modules.

Embedding/encoding unit has the following modules-

1. Controller
2. Image Memory Module
3. RCM Converter and Text Embedding Module
4. Text Module for Input

Extraction unit includes

1. Controller
2. Image Memory Module
3. Bit Extraction Module
4. Bit to text converter Module

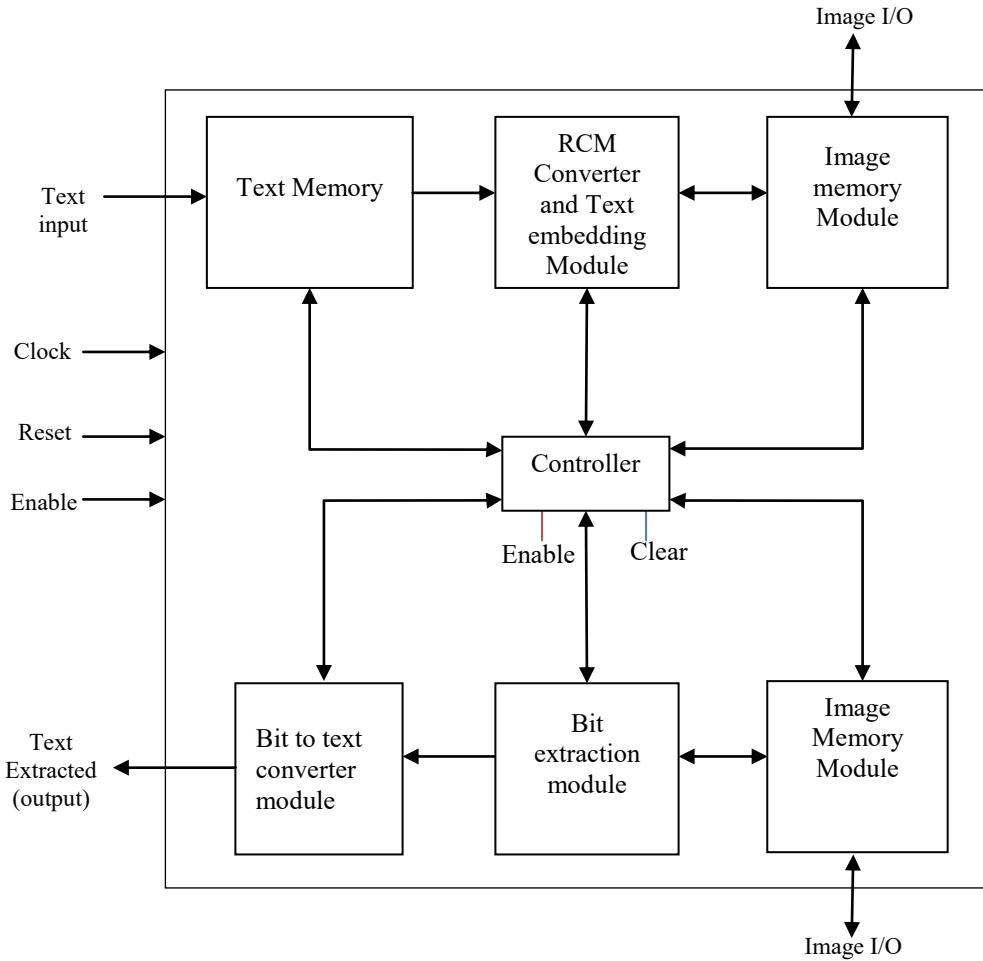


Figure 3.1: Functional block diagram of proposed design.

Figure 3.1 shows the functional block diagram of both embedding unit and extraction unit together. It is a standard practice to partition a complex design into different modules based on their specific functionality and features.

### 3.2.1 Controller

Both embedding and extracting unit has a controller module. The main feature of a RCM encoder is that the pixel pairs of the image is calculated from the image and not read from a look-up table. The purpose of this module is to take input, process data by enabling other modules and store output result into memory. After performing necessary operation, the data is written into the given memory location. The main purpose of this module is to co-ordinate with different modules in transmission or receiver unit to process data and to generate output.

### **3.2.2 Image Memory Module**

This module stores the image into its memory and the memory is a packed array where 512 rows and 512 columns are available. Here, every value of the particular position is the pixel. After storing the image into a variable, the values are the pixel value for each position of the image. Every position here shows the grey level image's pixel value and consists of 8 bits which minimum value is 00000000 and maximum value is 11111111. This module receives the image pixel values and sends to RCM converter and text embedding module so that the pixels are into pairs and RCM can be performed. After RCM, it does the embedding and it sends the pairs to the image memory module so that the watermarking of image can be reformed.

### **3.2.3 Text Memory**

This module takes input of the texts in a register. On that occasion, the texts are then converted into its constituent ASCII value of 7 bits. The bits are made ready for data embedding for the required position after RCM conversion.

### **3.2.4 RCM Converter Module and Text Embedding Module**

This module is the most important module in the embedding unit. Preparation for data embedding is the main task of RCM technique. This module makes the pair of pixels and sets the pair into two columns by assigning a dynamic array in this section. In the first column this module puts the first element of the pair of pixels and in the second column, puts the second element of the pair of pixels. For each column, there is 131072 rows, i.e., 131072 pixels pairs. After pairing the pixels here, this module detects the D region. For ambiguity, it also detects the Dc region so that all the pixel values in this region along with D region will be eliminated for RCM technique. Then, it performs RCM transformation on each element of the pairs of pixels. After transformation, the module embeds the data (i.e. text bits) into the related position. After embedding is complete, it sends back the each pixel to the image module so that the image can be reformed.

## **3.3 Extraction Unit**

This extraction unit has the following modules.

### **3.3.1 Bit to Text Converter Module**

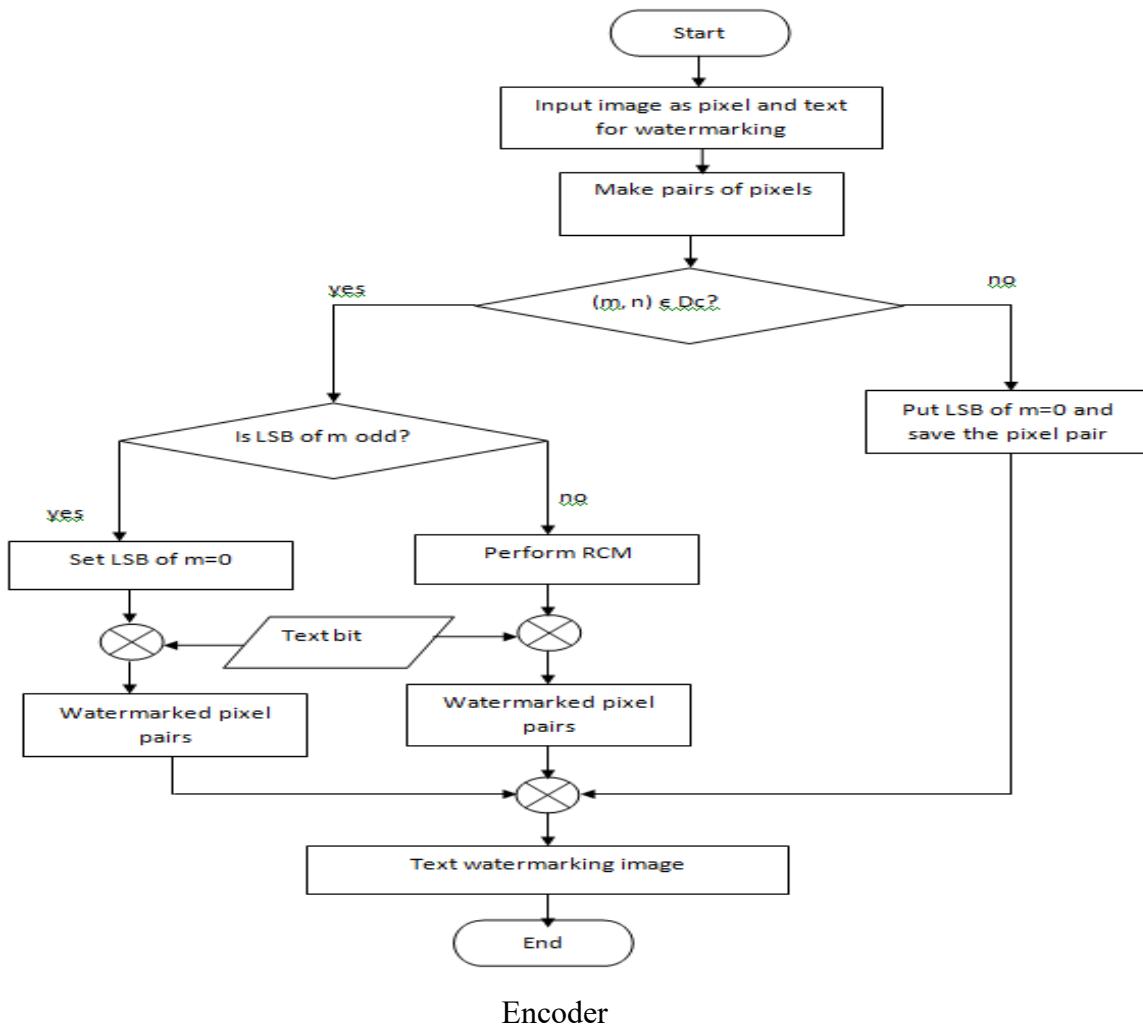
This module is same as in embedding unit. It does the opposite of the text module because here it reverts the bits to text from the extracted data sequence.

### 3.3.2 Image memory module

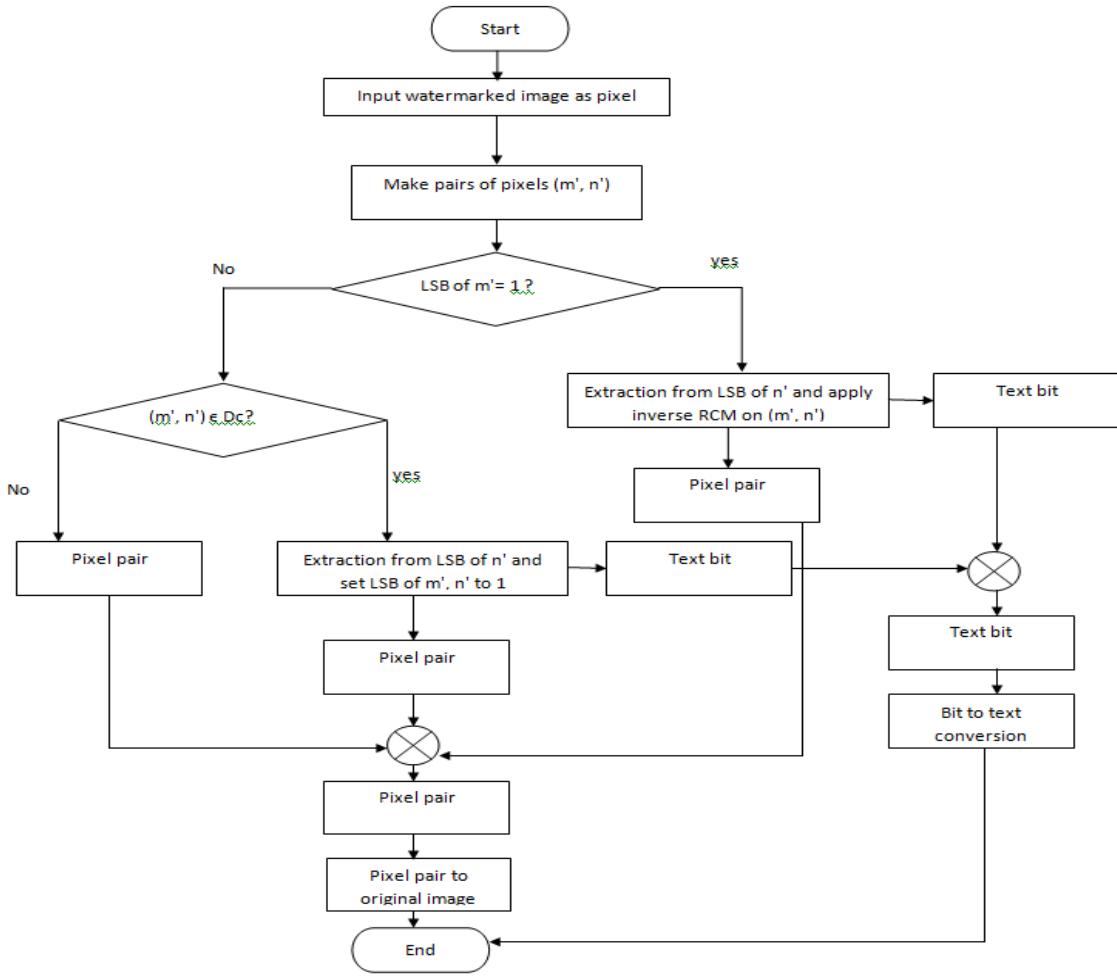
This module is same as the module which is in embedding unit. This module just takes the watermarks unit and passed the image into the bit extraction module. After performing extraction, it also receives the source images pixel values in it.

### 3.3.3 Bit Extraction Module

This module extracts the bits from the image. It pairs the pixels which it gets from the image memory module as the pair is performed in the RCM converter module. After this it performs the inverse transform and gets the text bits which are required. After extraction of the text bits this module sends back the pixels to the image module so that it can retrieve the original image. It sends the text bits to the bit to text converter module to retrieve the texts which were watermarked. Flow chart of text embedding into RCM technique is given below. It will summarize the embedding and extracting technique here.



Encoder



Decoder

Figure 3.2: Flowchart of the proposed design of the RCM encoder and decoder.

### 3.3.4 Bit to text conversion module

After retrieving the text bit, we transform the bits into its constituent ASCII code. We take 7 bits segment of text bits to retrieve the text by the conversion process.

## 3.4 Verilog HDL (Hardware Definition Language)

Brief description of all tools and software are given bellow. In the earlier, the conventional approach such as hand-draw and schematic based design technique was the only choice to the designer to design a digital system. But now millions of transistors are being integrated on a single chip integrated circuit (IC) where the conventional design technique is insufficient to be used. It points towards having a new approach for designing today's complex digital

system and that is hardware description language (HDL). HDL based design technique has been emerged as the most efficient solution. It offers the following advantages over conventional based design approaches.

- It is technology independent. If a particular IC fabrication process becomes outdated, it is possible to synthesize a new level design by only changing the technology file but using the same HDL code.
- HDL shortens the design cycle of a chip by efficiently describing and simulating the behavior of the chip. A complex circuit can be designed using a few lines of HDL code.
- It lowers the cost of design of an IC.
- It improves design quality of a chip. Area and timing of the chip can be optimized and analyzed in different stages of design.

There are different types of HDL available in the market. Some of these are vendor dependent where the HDL code is only useable under the software provided by the specific vendor. For example, Altera hardware description language (AHDL) from Altera company, Lola (Logic Language) from European Silicon Structure (ES2) company, etc. However, Verilog HDL and VHDL (very high speed IC hardware description language) are the two vendor independent HDL which are now widely accepted industry standard electronic design automation (EDA) tool for designing digital system. Verilog HDL is introduced by Cadence Data Systems, Inc. and later its control is transferred to a consortium of companies and universities known as open Verilog international (OVI) whereas VHDL is used primarily by defense contractors. Currently Verilog is widely used by IC designers. Verilog HDL is IEEE standard and easier than VHDL. It is less error prone. It has many pre-defined features very specific to IC design. For this reason Verilog is chosen to design and implement of the proposed system.

### **3.4.1 Modelsim Simulator**

ModelSim simulator is a software made by Mentor Graphics which has multi-language HDL simulation environment. This software is for simulation of hardware description languages such as VHDL, Verilog and SystemC, and includes a built-in C debugger. ModelSim can be used independently, or in conjunction with Altera Quartus or Xilinx ISE. Simulation is

performed using the graphical user interface (GUI), or automatically using scripts. ModelSim is offered in multiple editions, such as ModelSim PE, ModelSim SE, and ModelSimXE.

ModelSim SE offers high-performance and advanced debugging capabilities, while ModelSim PE is an entry-level simulator for hobbyists and students. ModelSim SE is used in large multi-million gate designs, and is supported on Microsoft Windows and Linux, in 32-bit and 64-bit architectures.

ModelSimXE stands for Xilinx Edition, and is specially designed for integration with Xilinx ISE. ModelSimXE enables testing of HDL programs written for Xilinx Virtex/Spartan series FPGA's without needed physical hardware.

ModelSim can also be used with MATLAB/Simulink, using Link for ModelSim. Link for ModelSim is a fast bidirectional co-simulation interface between Simulink and ModelSim. For such designs, MATLAB provides a numerical simulation toolset, while ModelSim provides tools to verify the hardware implementation & timing characteristics of the design.

#### **Language support:**

ModelSim uses a unified kernel for simulation of all supported languages, and the method of debugging embedded C code is the same as VHDL or Verilog.

ModelSim enables simulation, verification and debugging for the following languages:

- VHDL
- Verilog
- Verilog 2001
- SystemVerilog
- PSL

#### **3.4.2 Simulation Mode**

Simulation allows testing a design thoroughly to ensure that it responds correctly in every possible situation before configuring a device. Depending on the type of information need, functional or timing simulation can be performed with the simulator. Functional simulation tests only the logical operation of a design by simulating the behavior of flattened netlist

extracted from the design files, while timing simulation uses a fully compiled netlist containing timing information to test both the logical operation and the worst-case timing for the design in the target device. Before running a simulation, it is necessary to specify input vectors as the stimuli for the Quartus II Simulator. The simulator uses these input vectors to simulate the output signals that a programmed device would produce under the same conditions. The simulator supports input vector stimuli in the form of a vector waveform file (.vWF), vector table output file (.tbl), power input file (.pWF), or a Quartus II generated vector file (.vec) or simulator channel file (.scf).

**Quartus II** is a software tool produced by Altera for analysis and synthesis of HDL designs, which enables the developer to compile their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Quartus II Web Edition is a free version of Quartus II that can be downloaded or delivered by mail for free. This edition provides compilation and programming for a limited number of Altera devices. The low-cost Cyclone family of FPGAs is fully supported by this edition, as well as the MAX family of CPLDs, meaning small developers and educational institutions have no overheads from the cost of development software. License registration is required to use the Web Edition of Quartus II, which is free and can be renewed an unlimited number of times.

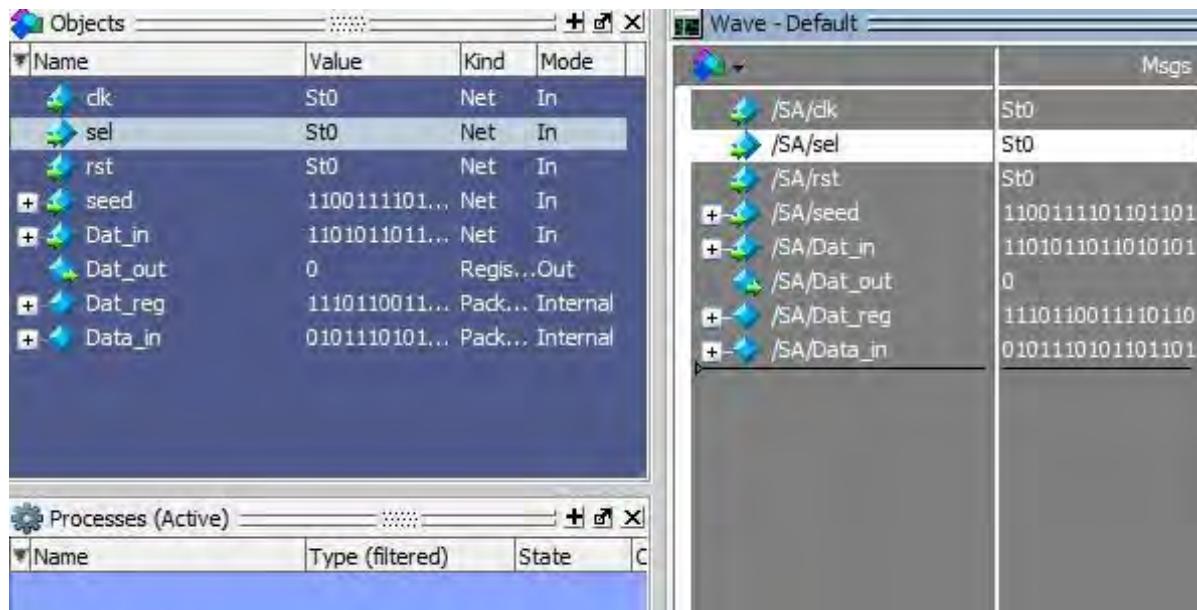


Figure 3.3: Example of a program in modelsim.

Figure 3.3 shows of a program that runs in FPGA.

# Chapter 4

## *Results and Discussion*

### **4.1 Introduction**

This chapter presents the MATLAB simulation of RCM watermarking technique. The following sections describe step by step simulation process in MATLAB environment.

### **4.2 Software Simulation**

Software simulation of this project has been done in MATLAB.

The specification of simulation environment is given below:

Tools: MATLAB 2015a and 2017a, 64-bit

OS: Windows 10 Pro 64-bit

RAM: 4 GB

Processor: Core i5 3.30 GHz

Below sections provide the simulation procedure of the proposed design.

In the our design we had divided our RCM text embedding technique in some several steps and defined all of these steps in matlab m files. We have the following m files which are given below.

- i. Watermarking
- ii. Char2bin
- iii. Data\_embedding
- iv. Chicky512

Watermarking m file is doing the image acquisition, image to pixel calculation and preparation for data embedding. All these three are done into this m file. The source image is Lena as shown in figure 4.1.



Figure 4.1: Source image (Lena)

This image is very much popular in image process area. There are two reasons of choosing Lena as test image according to David C. Munson who is the 10<sup>th</sup> president of Rochester Institute of Technology. First, the Lena image contains a nice mixture of detail, flat regions, shading, and texture that do a good job of testing various image processing algorithms. Second, the Lena image is a picture of an attractive woman. It is not surprising that the (mostly male) image processing research community gravitated toward an image that they found attractive.

We have stored this image into matlab and perform the simulation. Figure 3.14 shows the simulation screenshot.

```

Users > Hasan > Documents > MATLAB >
Editor - C:\Users\Hasan\Documents\MATLAB\char2bin.m
1 %function asciiLogicalArray = char2bin()
2 - clc;
3 - userInput = input('Enter a numerical digit : ', 's')
4 % userInput is the ASCII value of the digit. So for 2, the value is 50.
5 decimalValue = userInput;
6 % Get the binary value of the ascii value of the
7 % characters in a string (a character array).
8 % For example, for 2, strAscii will be 110010 (which = 50 in decimal).
9 strAscii = dec2bin(userInput,7); % Binary of the ascii value.
10 % Convert row-wise to a row vector
11 strAscii = reshape(strAscii', [1, numel(strAscii)]);
12 % Convert the binary string into a logical array.
13 % So for example, 2 = 50 = 110110 will be a logical array [1,1,0,1,1,0]
14 asciiLogicalArray = logical(strAscii - 48);
15 data=asciiLogicalArray;
16 [Row, maxsize]=size(data)

Command Window
Enter a numerical digit : Meet me ballroom at 10 PM
userInput =
Meet me ballroom at 10 PM
Row =
1
maxsize =
175
>>

```

Figure 4.2: Running the watermarking.m file in matlab 2015a.

```

HOME PLOTS APPS EDITOR PUBLISH VIEW
New Open Save Compare Go To Comment Insert Run Breakpoints Run and Advance Run and Time
FILE EDIT BREAKPOINTS RUN
Editor - C:\Users\Hasan\Documents\MATLAB\watermarking.m
1 %grouping the pixel paize
2 img1=imread('C:\Users\Hasan\Desktop\Test image new\lens_grayscale.png');
3 img=rgb2gray (img1);
4 [rows,cols]=size(img);
5 [x,y]=ndgrid(1:rows,1:cols);
6 ind=sub2ind(size(img),x,y);
7 ind_shift=sub2ind(size(img),x,y+1);
8 pixels1=img(ind);
9 pixels2=img(ind_shift);
10 pixels=(pixels1(:,1) pixels2(:,1));
11 [row1,col1]=size(pixels);
12 pixels=uint16 (pixels);
13
14
15
16 %performing Zx-y and Zy-x and to see the values.
17
18 for p=1:131072
19
20 pixels108 (p,1)=2*pixels(p,1)-pixels(p,2);
21

```

Name	Value
cols	\$12
cols1	2
i	1
img	\$12x512 uint8
img1	\$12x512x3 uint8
ind	\$12x512 double
ind_shift	\$12x512 double
m	131072
p	131072
pixels	\$12x512x2 uint16
pixels1	\$12x512 uint8
pixels108	\$12x512x2 uint16
pixels109	\$12x512x2 uint16
pixels2	\$12x512 uint8
row1	131072
rows	\$12
x	\$12x512 double
y	\$12x512 double

Figure 4.3: Text to be embedded.

After RCM transformation, our pixels are ready for data embedding. So, we will run the 2<sup>nd</sup> m file char2bin. Char2bin file is used for the input of text. Figure 4.2 shows it. After this step, we run the watermarked.m file for the text embedding. Figure 4.4 shows this.

The screenshot shows the MATLAB environment. In the Editor tab, the file 'Watermarked.m' is open, displaying MATLAB code for text embedding. The workspace browser on the right lists various variables and their values, including 'data' containing the text 'Meet me ballroom a...', 'img' (512x512 uint8), and 'pixels' (512x256 double). The Command Window at the bottom shows the command 'Watermarked' being run.

```

Editor - C:\Users\Hasan\Documents\MATLAB\Watermarked.m
+2 watermarking.m Watermarked.m char2bin.m Reverse_from_image.m Reverse_watermarking.m chiky512.m
1 -     embed_pixels= uint8(pixels10800);
2 -     position1;
3 -
4 -     for m=1:131072
5 -         for i=1:1
6 -             if (embed_pixels(m,i)==1 || embed_pixels(m,i)==255) && ((embed_pixels(m,i+1))==1 || embed_pixels(m,i+1)==255)
7 -                 embed_pixels(m,i)=bitset(embed_pixels(m,i),1,0);
8 -                 embed_pixels(m,i+1)=embed_pixels(m,i+1);
9 -
10 -             elseif (embed_pixels(m,i)==1 || embed_pixels(m,i)==255) || ((embed_pixels(m,i+1))==1 || embed_pixels(m,i+1)==255)
11 -                 embed_pixels(m,i)=bitset(embed_pixels(m,i),1,0);
12 -                 embed_pixels(m,i+1)=embed_pixels(m,i+1);
13 -
14 -             elseif (mod((embed_pixels(m,i)),2)==0) && (mod((embed_pixels(m,i+1)),2)==0)
15 -                 if position<maxsize
16 -                     if position<maxsize
17 -                         embed_pixels(m,i)=bitset(embed_pixels(m,i),1,0);
18 -                         embed_pixels(m,i+1)=embed_pixels(m,i+1);
19 -
20 -                     elseif (mod((embed_pixels(m,i)),2)==0) && (mod((embed_pixels(m,i+1)),2)==0)
21 -                         if position<maxsize
22 -                             if position<maxsize
23 -                                 if position<maxsize
24 -                                     if position<maxsize

```

Figure 4.4: Text embedding complete.

In this step, data i.e., text is embedded. After embedding, the pixels are taken back to its related original position to form watermarked image. Now we can see the next watermarked image in figure 4.5.

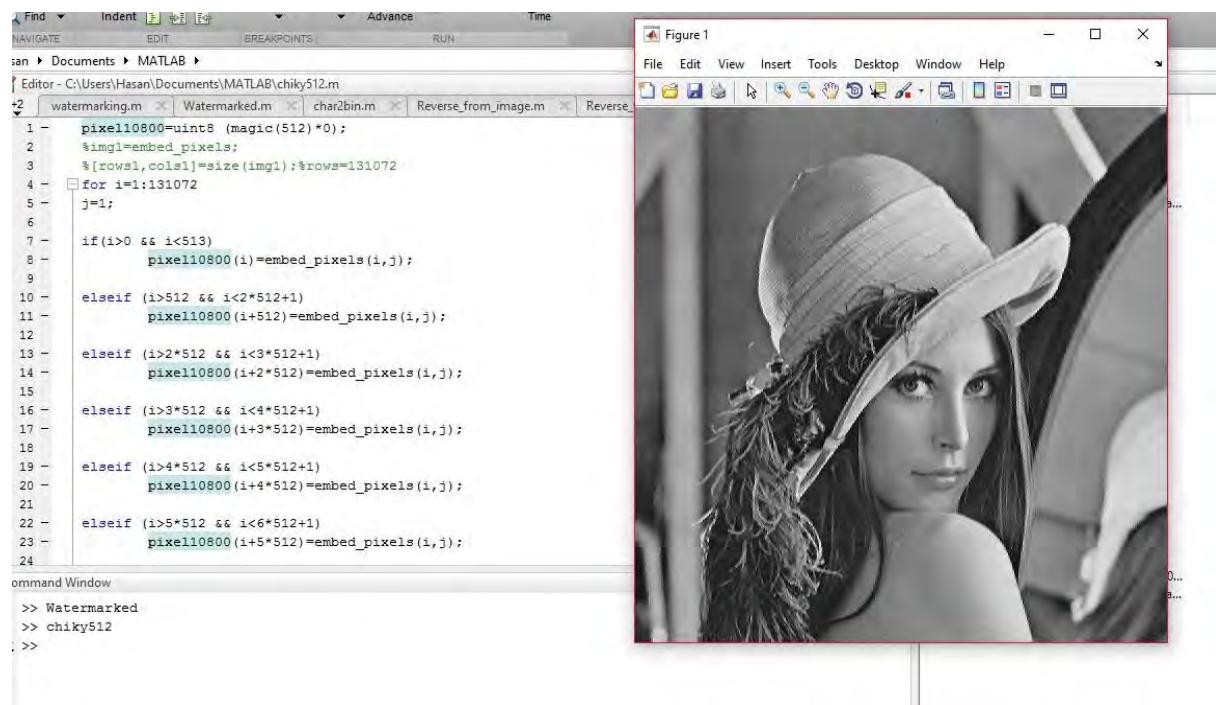
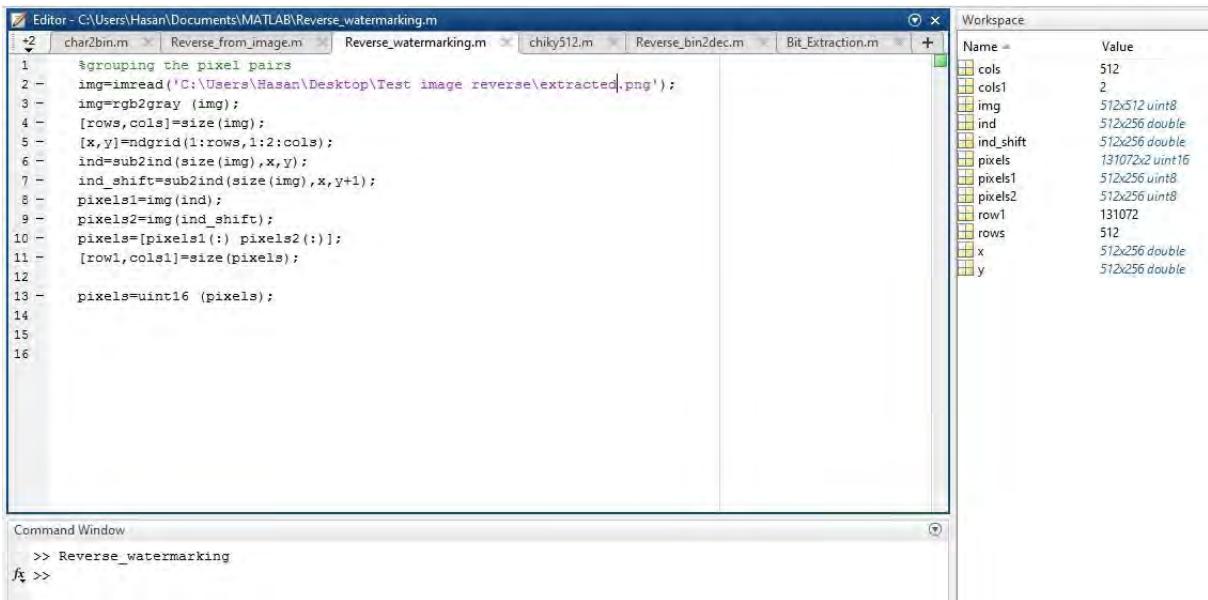


Figure 4.5: Watermarked image.

Our encoding simulation is completed. After encoding, we decode the text from this watermarked image. We had simulated this in four matlab .m files. These are.

- i. Reverse\_watermarking.m
- ii. Bit extraction.m
- iii. Reverse\_bin2dec.m
- iv. Reverse\_from\_image

Reverse\_watermarking.m file receives the image and does the pixel pairs. After that bit\_extraction.m extracts the text data from the pixel pair.



The screenshot shows the MATLAB environment with the following details:

- Editor:** The current file is `Reverse_watermarking.m`. The code reads an image, groups pixels into pairs, and then extracts text from these pairs.
- Workspace:** A table showing variables and their values:
 

Name	Value
cols	512
cols1	2
img	512x512 uint8
ind	512x256 double
ind_shift	512x256 double
pixels	131072x2 uint16
pixels1	512x256 uint8
pixels2	512x256 uint8
row1	131072
rows	512
x	512x256 double
y	512x256 double
- Command Window:** Shows the command `>> Reverse_watermarking`.

Figure 4.6: Running Reverse\_watermarking.m

Bit extraction extracts the text and performs inverse transformation for the image Lena's integrity. We had extracted the text but it is in bit form; so we need to reverse the bits back to its respective text letters as it is. The Reverse\_bin2dec.m matlab file does the text bits back to its respective text letter. Reverse\_bin2dec.m file is shown in Figure 4.7.

The screenshot shows the MATLAB Editor window with the file `Reverse_bin2dec.m` open. The code deembeds a watermark from a binary image. The workspace on the right contains variables like `A`, `bit_segment_7`, `cols`, `data1`, `data_bit`, `data_obtained`, `decimalvalue_of...`, `i`, `img`, `ind`, `ind_shift`, `m`, `maindata`, `mainpixel`, `pixels`, `pixels1`, `pixels2`, `pos`, `row1`, `rows`, `Text`, `x`, and `y`. The Command Window shows the command `>> Reverse_watermarking` and the output text "Meet me ballroom at 10 PM".

```

Editor - C:\Users\Hasan\Documents\MATLAB\Reverse_bin2dec.m
1 %A=logical(databit<0);
2 %databit = char (A+48);
3
4 for i=1:7:numel(databit)
5
6 if (databit (i)==1) && (databit (i+1)==1) && (databit (i+2)==1) && (databit (i+3)==1) && (databit
7 break
8
9 else
10
11 if (i==numel(databit))
12 break
13 else
14 data_obtained (i)= databit (i);
15 data_obtained (i+1)= databit (i+1);
16 data_obtained (i+2)= databit (i+2);
17 data_obtained (i+3)=databit (i+3);
18 data_obtained (i+4)=databit (i+4);
19 data_obtained (i+5)=databit (i+5);
20 data_obtained (i+6)=databit (i+6);
21 end
22 end
23

```

Figure 4.7: Reverse\_bin2dec.m file results.

Reverse\_bin2dec.m file results are also shown in the Figure 4.7. The text which we watermarked the original text is gained. After this, we got our original image by running reverse\_from\_image.m file which is shown in Figure 4.8.

The screenshot shows the MATLAB Editor window with the file `reverse_from_image.m` open. The code embeds a watermark into a grayscale image. The figure window displays a grayscale portrait of a woman wearing a hat. The Command Window shows errors related to `imshow` and the command `>> imshow (pixel10800)`.

```

Editor - C:\Users\Hasan\Documents\MATLAB\reverse_from_image.m
1534 elseif (i>249*512 && i<250*512+1)
1535 pixel10800(i+250*512)=mainpixel(i,j);
1536
1537 elseif (i>250*512 && i<251*512+1)
1538 pixel10800(i+251*512)=mainpixel(i,j);
1539
1540 elseif (i>251*512 && i<252*512+1)
1541 pixel10800(i+252*512)=mainpixel(i,j);
1542
1543 elseif (i>252*512 && i<253*512+1)
1544 pixel10800(i+253*512)=mainpixel(i,j);
1545
1546 elseif (i>253*512 && i<254*512+1)
1547 pixel10800(i+254*512)=mainpixel(i,j);
1548
1549 elseif (i>254*512 && i<255*512+1)
1550 pixel10800(i+255*512)=mainpixel(i,j);
1551
1552 elseif (i>255*512 && i<256*512+1)
1553 pixel10800(i+256*512)=mainpixel(i,j);
1554
1555 end
1556
1557 end

```

Figure 4.8: The original image retrieved.

This we successfully retrieved the image and completed the simulation successfully.

### **4.3 Result**

The proposed design is implemented by Matlab2015a which is a very useful tool for image processing. This proposed system requires bit to text converter and text to bit converter with the existing RCM technique. A comparative evaluation of text embedding and image quality is shown into the next sections.

### **4.4 Text Embedding**

Our aim is to develop a robust text watermarking system with the existing RCM bit watermarking system. We have successfully done the text embedding and retrieve the texts as it is in Lena. In [5] Coltuc and Chassery had implemented the RCM technique for bits but we have implemented here for characters. In Lena we had embedded up to 70 characters and retrieved the characters 70 characters because, Lena has maximum 70 characters hiding capacity. If the region of Dc is increased the data hiding capacity is also increased. We also implemented this technique in another image that is sailed boat image. In this image we got better result than Lena. In this image we successfully embedded 80 characters. Here, D region is not so big but Dc region is greater so data capacity has increased. We summarize these two images in the following table.

Table 4.1: Comparison of two test image for text embedding

Image Name	Size	Embedded characters
Lena	512X512	70
Sailed Boat	512X512	80

Figure 4.9 (a) shows the original image Lena and after embedding, the embedded image is shown in 4.9 (b). Again we also run our codes to the other image namely sailed boat. Figure 4.10 (a) shows the original image of sailed boat and after embedding, the embedded image is shown in 4.10 (b).



Figure 4. 9: (a) Original image



(b) Text watermarked image



Figure 4.10: (a) Original Image



(b) Text watermarked image

#### 4.5 Image Quality

Image quality is one of the great issues here. We have determined the image quality in terms of Mean Square Error (MSE), Peak Signal to Noise Ration (PSNR) and Naturalness Image Quality Evaluator (NIQE).

Two of the error metrics that were used to compare the various image compression techniques are the Mean Square Error (MSE) and the Peak Signal to Noise Ratio (PSNR). The MSE is the cumulative squared error between the compressed and the original image, whereas PSNR is a measure of the peak error. The mathematical formulae for the two are

$$\text{MSE} = \frac{1}{MN} \sum_{y=1}^M \sum_{x=1}^N [I(x,y) - I'(x,y)]^2 \dots \quad (4.1)$$

$$\text{PSNR} = 20 * \log_{10} (255 / \text{sqrt}(\text{MSE})) \dots \quad (4.2)$$

where  $I(x,y)$  is the original image,  $I'(x,y)$  is the approximated version (which is actually the decompressed image) and  $M,N$  are the dimensions of the images. A lower value for MSE means lesser error, and as seen from the inverse relation between the MSE and PSNR, this translates to a high value of PSNR. Logically, a higher value of PSNR is good because it means that the ratio of Signal to Noise is higher. Here, the 'signal' is the original image, and the 'noise' is the error in reconstruction. So, if we find a compression scheme having a lower MSE (and a high PSNR), we can recognize that it is a better one.

No reference opinion unaware distortion unaware image quality assessment (OU-DU IQA) is called NIQE. [30] Natural Image Quality Evaluator (NIQE) blind image quality assessment (IQA) is a completely blind image quality analyzer that only makes use of measurable deviations from statistical regularities observed in natural images, without training on human-rated distorted images, and indeed without any exposure to distorted images. However, all current state-of-the-art general purpose no reference (NR) IQA algorithms require knowledge about anticipated distortions in the form of training examples and corresponding human opinion scores. NIQE is defined as

$$D(\nu_1, \nu_2, \Sigma_1, \Sigma_2) = \sqrt{\left( (\nu_1 - \nu_2)^T \left( \frac{\Sigma_1 + \Sigma_2}{2} \right)^{-1} (\nu_1 - \nu_2) \right)} \dots \quad (4.3)$$

Where  $\nu_1, \nu_2$  and  $\Sigma_1, \Sigma_2$  are the mean vectors and covariance matrices of the Natural MVG (Multivariate Gaussian) model and the distorted image's MVG model. It is based on the construction of a quality aware collection of statistical features based on a simple and

successful space domain natural scene statistic (NSS) model. These features are derived from a corpus of natural, undistorted images. Experimental results show that the new index delivers performance comparable to top performing NR IQA models that require training on large databases of human opinions of distorted images.

Table 4.2: Quality results of embedded images.

Image	MSE	PSNR	NIQE
Lena	39.39	32.18	13.6276
Sailed boat	67.5240	29.8362	11.4690

Table 4.3: Quality results of embedded images (Lena).

Research	MSE	PSNR	NIQE
Literature [29]	28.013	33.38	-
Our	39.39	32.18	13.6276

From Table 4.2 we can see that for text watermarked images quality does not degraded too much. In literature [29] for Lena they got Mean Square Error (MSE) is 28.013 but we get 39.39 and PSNR is 33.38 but we get here 32.18. Naturalness Image Quality Evaluator (NIQE) no-reference image quality score of Lena is 13.6276. For sailed boat image MSE is also greater than the Lena and PSNR is also reduced and NIQE is also smaller than the Lena. Sailed boat has greater embedding capacity.

Table 4.4: Accuracy results of text in embedded images (Lena).

Research	Accuracy
Literature [31 ]	91.18 % (Maximum)
Ours	100.00 %

Table 4.3 shows the comparison of literature [29] with ours work regarding image quality. Literature [29] has lower MSE but we had better PSNR than their work. They did not show the naturalness of the image, but we show it. Table 4.4 shows a comparison of literature [31] with ours work regarding the accuracy of text in embedded images Lena. Literature [31] have

their accuracy of text extraction is 91.18% maximum with various aspects but we have 100% accurate results.

This MATLAB task presents a solution of the issue which is simple, robust and low hardware overhead. The proposed design has been simulated in the MATLAB environment. Simulation results ensure the proper functionality of the design. Architecture for design and implementing the proposed scheme in the FPGA platform is figured out.

#### ***4.6 More Implemented Bench Mark Image***





Covered Images

Watermarked Images

Figure 4.11: More implemented bench mark images.

Figure 4.11 shows some implemented bench mark images where we had successfully run our MATLAB code and get good results. For every image we had more than 60 words capacity except the baby image (Only 10 characters).

# **Chapter 5**

## ***Conclusion***

### **5.1 Conclusion**

Protecting digital content such as copyright protection, copy control, integrity verification, content authentication, etc, is a demand of this day in this era of ICT. This project proposed RCM based reversible watermarking as a solution of the problem. RCM RW is lossless watermarking technique where the embedded watermark is not only extracted but also perfect reconstruction of the host signal is possible from the watermarked one and so it has enormous applications such as military, e-healthcare, telemedicine, tele-surgery, legal domains, etc. The objective of this project was to design a RCM based RW system and further to implement into suitable FPGA based device to make it fault tolerant. The proposed system has been designed under MATLAB environment. One of the attractive features of the design is to embed characters rather bits makes it user friendly. Simulation of the design using benchmark images have been performed in the MATLAB environment. Results obtained from the simulation verify correctness and effectiveness of overall system. The simulation results show very impressive results in terms of text embedding and image quality of the cover image after watermarked and also after extracted. The simulation results show that the text embedding encoder is working properly with respect to RCM requirement. It has been compared with the other researches and proved its superiority over other researches. As we know RCM is fault tolerant on bits watermarking, so this text watermarking system will be fault tolerant as well. The text embedding and watermarked image quality of this design have been compared with that of other researchers which show its superiority over all the existing design in the said area. In terms of text embedding 70 characters are embedded in Lena and 80 characters embedded in sailed boat. After extracting the characters, we saw 100% accuracy of the texts which is our one of the main objectives. In terms of image quality, this is also very good comparing with PSNR. In this watermarking technique, the data embedding capacity also depends on the iteration of this process. Due to time constraint, it was not possible to

implement the design into FPGA hardware however, the hardware architecture of the design has been identified which can be implemented in the future research.

## **5.2 Suggestions for Further Works**

This project was focused on achieving text embedding in the images. However, various aspects such as area attack, tempering, benchmark, etc., were not addressed. In the future, these issues will be addressed. Again the system will be designed using HDL and will be implemented into suitable FPGA device.

## References

1. Cox, I., Miller, M. & Jeffrey, B. (2002) "Digital Watermarking: Principles and Practice" Morgan Kaufmann.
2. Fridrich, J. Goljan, M., & Du, R. (2001), "Invertible authentication watermark for JPEG images". Proceeding of the IEEE International Conference on Information Technology, 223-227
3. Van Leest, A., Van der Veen, M., & Bruckers, F. (2003). Reversible image Watermarking". Proceedings of the IEEE International Conference on Image Processing, II, 731-734
4. J. Tian. "Wavelet-based reversible watermarking for authentication". In E.J. Delp III and P.W. Wong, editors, Security and Watermarking of Multimedia Contents volume 4675 of Proc. Of SPIE, pages 679-690, Jan.2002
5. Coltuc, D. Chassery, J.M.: "Very Fast Watermarking by Reversible Contrast Mapping". IEEE Signal Processing Letters 14, 255-258 (2007).
6. Maity, S.P., Banerjee, A., Abhijit, A., Kundu, M.K., 2007. VLSI design of spread spectrum image watermarking. In: Proc. of 13th National Conference on communications (NCC-2007), Indian Institute of Technology, Kanpur, India, January 26–28, pp. 251–257.
7. Mohanty, S.P., Nayak, R.K.C.S., 2004. FPGA based implementation of an invisible-robust image watermarking encoder. Lect. Notes Comput. Sci. 3356, 344–353.
8. Maity, S.P., Kundu, M.K., 2013. Distortion free image-in-image communication with implementation in FPGA. Int. J. Electron. Commun. 67 (May (5)), 438–447.
9. Juergen Seitz, "Digital Watermarking for digital media", University of Cooperative Education Heidenheim, Germany, 2005
10. M.U. Ceilk, G. Sharma, A. M. Tekalp and E. Saber. "Reversible data hiding". In proc. Of International Conference on Image Processing, volume II, pages 157-160, Sept 2002.
11. Mohanty SP, Ranganathan N, Namballa RK. "VLSI Implementation of invisible digital watermarking algorithms towards the development of a secure JPEG encoder" In: Proceddings of the IEEE workshop o signal processing systems; 2003. P. 183-188.

12. Mohanty SP, Nayak S. "FPGA based implementation of an invisible-robust image watermarking encoder." In: Lecture notes in computer science, vol. 3356; 2004.p. 344-353.
13. Mohanty SP, Kougianos E, Ranganathan N. "VLSI architecture and chip for combined invisible robust and fragile watermarking." IET Comput Digital Tech (CDT) 2007; 1(5):600-611.
14. A. Garimella, M. V. V. Satyanarayan, R. S. Kumar, P. S. Murugesh, and U. C. Niranjan, "VLSI Implementation of Online Digital Watermarking Techniques with Difference Encoding for the 8-bit Gray Scale Images," in Proceedings of the International Conference on VLSI Design, 2003, pp. 283-288.
15. S. P. Mohanty, N. Ranganathan and R. K. Namballa, "A VLSI Architecture for Visible Watermarking in a Secure Still Digital Camera (S<sup>2</sup>DC) Design," IEEE Transactions on Very Large Scale Integration System, vol. 13, no. 8, pp. 1002-1012, August 2005.
16. Roberto Caldelli, Francesco Filippini, and Rudy Becarelli, "Reversible watermarking techniques: An overview and a classification," EURASIP Journal on Information Security, vol. 2010, Article ID 13454.
17. Barni, M., Bartolini, F.: Watermarking Systems Engineering: Enabling Digital Assets Security and Other Applications. Marcel Dekker, Inc., New York (2004)
18. Katzenbeisser, S., Petitcolas, F. (eds.): Information Hiding Techniques for Steganography and Digital Watermarking. Artech House, Norwood (2000)
19. Nikolaidis, N., Pitas, I.: Digital image watermarking: an overview. In: IEEE Int. Conf. on Multimedia Computing and Systems, vol. 1, pp. 1–6 (1999)
20. De Vleeschouwer, C., Delaigle, J. F., Macq, B.: Invisibility and application functionalities in perceptual watermarking An overview. Proc. of the IEEE 90(1), 64–77 (2002)
21. Jewel F. A. "Automated Vehicle License Plate Detection System Using FRIT Algorithim" Thesis Paper for M Sc in ICT, BUET (2013)
22. Huang, C.H., Wu, J.L.: Attacking visible watermarking schemes. IEEE Trans. on Multimedia 6(1), 16–30 (2004)
23. Cheung, W.N.: Digital image watermarking in spatial and transform domains.

In: Proc. IEEE TENCON 2000, vol. 3, pp. 374–378 (2000)

24. Ahmed, N., Natarajan, T., Rao, K.R.: Discrete cosine transforms. IEEE Trans. on Comp. C-23, 90–93 (1974)
25. Vetterli, M., Kovacevic, J.: Wavelets and Subband Coding. Prentice-Hall Inc., Englewood Cliffs (1995)
26. Gersho, A., Gray, R.M.: Vector Quantization and Signal Compression. Kluwer Academic Publisher, London (1992)
27. S.Ghosh, B. Kundu, D. Datta, S.P. Maity, H. Rahman “Design and implementation of fast FPGA based architecture for reversible watermarking,” Electrical Information and Communication Technology (EICT), 2013 International Conference on 13-15 Feb. 2014.
28. R. Patel, M. Turuk, "VLSI Implementation of Reversible Watermarking using RCM", International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169, Volume: 3 Issue: 5 2724 – 2727
29. S. Ghosh, , N. Das, S. Das, S.P. Maity and H. Rahman “An adaptive feedback based reversible watermarking algorith using difference expansion” 2015 IEEE 2nd International Conference on Recent Trends in Information Systems (ReTIS).
30. A. Mittal, R. Soundararajan, and A.C. Bovik, “Making a Completely Blind Image Quality Analyzer”, IEEE Signal Processing Letters, Volume: 20, Issue: 3, March 2013.
31. K.U. Jaseena, J. Anita, “An Invisible Zero Watermarking Algorithm using Combined Image and Text for Protecting Text Documents”, International Journal on Computer Science and Engineering (IJCSE), Volume: 3, No: 6, June 2011.

## Appendix

### Watermarking.m

```
%grouping the pixel pairs
img=imread('C:\Users\Hasan\Desktop\Test image new\flower1.png');
%img=rgb2gray (img1);
[rows,cols]=size(img);
[x,y]=ndgrid(1:rows,1:2:cols);
ind=sub2ind(size(img),x,y);
ind_shift=sub2ind(size(img),x,y+1);
pixels1=img(ind);
pixels2=img(ind_shift);
pixels=[pixels1(:) pixels2(:)];
[row1,cols1]=size(pixels);

pixels=uint16 (pixels);

%performing 2x-y and 2y-x and to see the values.

for p=1:131072

pixels108 (p,1)=2*pixels(p,1)-pixels(p,2);
pixels108 (p,2)=2*pixels(p,2)-pixels(p,1);

end

% ready for data embedding position

for m=1:131072

    for i=1:1

        if (pixels108(m,i)==1 || pixels108(m,i)==255 ) &&
((pixels108(m,i+1))==1 || pixels108(m,i)==255)

            pixels10800(m,i)=pixels108(m,i);
            pixels10800(m,i+1)=pixels108(m,i+1);

        elseif (pixels108(m,i)==1 || pixels108(m,i)==255 ) ||
((pixels108(m,i+1))==1 || pixels108(m,i+1)==255)

            pixels10800(m,i)=pixels108(m,i);
            pixels10800(m,i+1)=pixels108(m,i+1);

        elseif (mod((pixels108(m,i)),2)==0) && (mod((pixels108(m,i+1)),2)==0)

            pixels10800(m,i)=pixels108(m,i);
            pixels10800(m,i+1)=pixels108(m,i+1);

    end
end
```

```

elseif mod((pixels108(m,i)),2)==0 && mod((pixels108(m,i+1)),2)==1
    pixels10800(m,i)=pixels(m,i);
    pixels10800(m,i+1)=pixels(m,i+1);

elseif mod((pixels108(m,i)),2)==1 && mod((pixels108(m,i+1)),2)==0
    pixels10800(m,i)=pixels(m,i);
    pixels10800(m,i+1)=pixels(m,i+1);

else
    pixels10800(m,i)=pixels(m,i);
    pixels10800(m,i+1)=pixels(m,i+1);

end
end
end

```

### char2bin.m

```

%function asciiLogicalArray = char2bin()
clc;
userInput = input('Enter a numerical digit : ', 's')
% userInput is the ASCII value of the digit. So for 2, the value is 50.
decimalValue = userInput;
% Get the binary value of the ascii value of the
% characters in a string (a character array).
% For example, for 2, strAscii will be 110010 (which = 50 in decimal).
strAscii = dec2bin(userInput,7); % Binary of the ascii value.
% Convert row-wise to a row vector
strAscii = reshape(strAscii', [1, numel(strAscii)]);
% Convert the binary string into a logical array.
% So for example, 2 = 50 = 110110 will be a logical array [1,1,0,1,1,0]
asciiLogicalArray = logical(strAscii - 48);
data=asciiLogicalArray;
[Row, maxsize]=size(data)
%Reversal Commands.
%strAscii = reshape(strAscii', [12,7])
%decimalvalue=bin2dec(strAscii)
%s=char(decimalValue)

```

### Watermarked.m

```

embed_pixels= uint8(pixels10800);
position=1;

for m=1:131072
    for i=1:1

```

```

    if (embed_pixels(m,i)==1 || embed_pixels(m,i)==255 ) &&
((embed_pixels(m,i+1))==1 || embed_pixels(m,i)==255)

    embed_pixels(m,i)=bitset(embed_pixels(m,i),1,0);
    embed_pixels(m,i+1)=embed_pixels(m,i+1);

    elseif (embed_pixels(m,i)==1 || embed_pixels(m,i)==255 ) ||
((embed_pixels(m,i+1))==1 || embed_pixels(m,i+1)==255)

    embed_pixels(m,i)=bitset(embed_pixels(m,i),1,0);
    embed_pixels(m,i+1)=embed_pixels(m,i+1);

    elseif (mod((embed_pixels(m,i)),2)==0) &&
(mod((embed_pixels(m,i+1)),2)==0)

    if position<=maxsize

        embed_pixels(m,i)=bitset(embed_pixels(m,i),1);
        embed_pixels(m,i+1)=bitset(embed_pixels(m,i+1),1,data(position));
        position=position+1;

    else
        embed_pixels(m,i)=bitset(embed_pixels(m,i),1);
        embed_pixels(m,i+1)=bitset(embed_pixels(m,i+1),1);

    end

elseif mod((embed_pixels(m,i)),2)==0 && mod((embed_pixels(m,i+1)),2)==1

    if position<=maxsize

        embed_pixels(m,i)=bitset(embed_pixels(m,i),1,1);
        embed_pixels(m,i+1)=bitset(embed_pixels(m,i+1),1,data(position));
        position=position+1;
    else

        embed_pixels(m,i)=bitset(embed_pixels(m,i),1,1);
        embed_pixels(m,i+1)=bitset(embed_pixels(m,i+1),1);

    end

elseif mod((embed_pixels(m,i)),2)==1 && mod((embed_pixels(m,i+1)),2)==0

    if position<=maxsize

        embed_pixels(m,i)=bitset(embed_pixels(m,i),1,1);
        embed_pixels(m,i+1)=bitset(embed_pixels(m,i+1),1,data(position));
        position=position+1;

    else

```

```

    embed_pixels(m,i)=bitset(embed_pixels(m,i),1,1);
    embed_pixels(m,i+1)=bitset(embed_pixels(m,i+1),1);

    end
elseif mod((embed_pixels(m,i)),2)==1 && mod((embed_pixels(m,i+1)),2)==1

    if position<=maxsize

        embed_pixels(m,i)=bitset(embed_pixels(m,i),1,0);
        embed_pixels(m,i+1)=bitset(embed_pixels(m,i+1),1,data(position));
        position=position+1;

    else

        embed_pixels(m,i)=bitset(embed_pixels(m,i),1,0);
        embed_pixels(m,i+1)=bitset(embed_pixels(m,i+1),1);

    end

else

    embed_pixels(m,i)=bitset(embed_pixels(m,i),1,0);
    embed_pixels(m,i+1)=embed_pixels(m,i+1);

end

end

```

### chiky512.m

```

pixel10800=uint8 (magic(512)*0);
%img1=embed_pixels;
%[rows1,cols1]=size(img1);%rows=131072
for i=1:131072
j=1;

if(i>0 && i<513)
    pixel10800(i)=embed_pixels(i,j);

elseif (i>512 && i<2*512+1)
    pixel10800(i+512)=embed_pixels(i,j);

elseif (i>2*512 && i<3*512+1)
    pixel10800(i+2*512)=embed_pixels(i,j);

elseif (i>3*512 && i<4*512+1)
    pixel10800(i+3*512)=embed_pixels(i,j);


```

```

elseif (i>4*512 && i<5*512+1)
pixel10800(i+4*512)=embed_pixels(i,j);

elseif (i>5*512 && i<6*512+1)
pixel10800(i+5*512)=embed_pixels(i,j);

elseif (i>6*512 && i<7*512+1)
pixel10800(i+6*512)=embed_pixels(i,j);

elseif (i>7*512 && i<8*512+1)
pixel10800(i+7*512)=embed_pixels(i,j);

elseif (i>8*512 && i<9*512+1)
pixel10800(i+8*512)=embed_pixels(i,j);

elseif (i>9*512 && i<10*512+1)
pixel10800(i+9*512)=embed_pixels(i,j);

elseif (i>10*512 && i<11*512+1)
pixel10800(i+10*512)=embed_pixels(i,j);

elseif (i>11*512 && i<12*512+1)
pixel10800(i+11*512)=embed_pixels(i,j);

elseif (i>12*512 && i<13*512+1)
pixel10800(i+12*512)=embed_pixels(i,j);

elseif (i>13*512 && i<14*512+1)
pixel10800(i+13*512)=embed_pixels(i,j);

elseif (i>14*512 && i<15*512+1)
pixel10800(i+14*512)=embed_pixels(i,j);

elseif (i>15*512 && i<16*512+1)
pixel10800(i+15*512)=embed_pixels(i,j);

elseif (i>16*512 && i<17*512+1)
pixel10800(i+16*512)=embed_pixels(i,j);

elseif (i>17*512 && i<18*512+1)
pixel10800(i+17*512)=embed_pixels(i,j);

elseif (i>18*512 && i<19*512+1)
pixel10800(i+18*512)=embed_pixels(i,j);

elseif (i>19*512 && i<20*512+1)
pixel10800(i+19*512)=embed_pixels(i,j);

elseif (i>20*512 && i<21*512+1)
pixel10800(i+20*512)=embed_pixels(i,j);

elseif (i>21*512 && i<22*512+1)
pixel10800(i+21*512)=embed_pixels(i,j);

```

```

elseif (i>22*512 && i<23*512+1)
pixel10800(i+22*512)=embed_pixels(i,j);

elseif (i>23*512 && i<24*512+1)
pixel10800(i+23*512)=embed_pixels(i,j);

elseif (i>24*512 && i<25*512+1)
pixel10800(i+24*512)=embed_pixels(i,j);

elseif (i>25*512 && i<26*512+1)
pixel10800(i+25*512)=embed_pixels(i,j);

elseif (i>26*512 && i<27*512+1)
pixel10800(i+26*512)=embed_pixels(i,j);

elseif (i>27*512 && i<28*512+1)
pixel10800(i+27*512)=embed_pixels(i,j);

elseif (i>28*512 && i<29*512+1)
pixel10800(i+28*512)=embed_pixels(i,j);

elseif (i>29*512 && i<30*512+1)
pixel10800(i+29*512)=embed_pixels(i,j);

elseif (i>30*512 && i<31*512+1)
pixel10800(i+30*512)=embed_pixels(i,j);

elseif (i>31*512 && i<32*512+1)
pixel10800(i+31*512)=embed_pixels(i,j);

elseif (i>32*512 && i<33*512+1)
pixel10800(i+32*512)=embed_pixels(i,j);

elseif (i>33*512 && i<34*512+1)
pixel10800(i+33*512)=embed_pixels(i,j);

elseif (i>34*512 && i<35*512+1)
pixel10800(i+34*512)=embed_pixels(i,j);

elseif (i>35*512 && i<36*512+1)
pixel10800(i+35*512)=embed_pixels(i,j);

elseif (i>36*512 && i<37*512+1)
pixel10800(i+36*512)=embed_pixels(i,j);

elseif (i>37*512 && i<38*512+1)
pixel10800(i+37*512)=embed_pixels(i,j);

elseif (i>38*512 && i<39*512+1)
pixel10800(i+38*512)=embed_pixels(i,j);

elseif (i>39*512 && i<40*512+1)

```

```

pixel10800(i+39*512)=embed_pixels(i,j);

elseif (i>40*512 && i<41*512+1)
pixel10800(i+40*512)=embed_pixels(i,j);

elseif (i>41*512 && i<42*512+1)
pixel10800(i+41*512)=embed_pixels(i,j);

elseif (i>42*512 && i<43*512+1)
pixel10800(i+42*512)=embed_pixels(i,j);

elseif (i>43*512 && i<44*512+1)
pixel10800(i+43*512)=embed_pixels(i,j);

elseif (i>44*512 && i<45*512+1)
pixel10800(i+44*512)=embed_pixels(i,j);

elseif (i>45*512 && i<46*512+1)
pixel10800(i+45*512)=embed_pixels(i,j);

elseif (i>46*512 && i<47*512+1)
pixel10800(i+46*512)=embed_pixels(i,j);

elseif (i>47*512 && i<48*512+1)
pixel10800(i+47*512)=embed_pixels(i,j);

elseif (i>48*512 && i<49*512+1)
pixel10800(i+48*512)=embed_pixels(i,j);

elseif (i>49*512 && i<50*512+1)
pixel10800(i+49*512)=embed_pixels(i,j);

elseif (i>50*512 && i<51*512+1)
pixel10800(i+50*512)=embed_pixels(i,j);

elseif (i>51*512 && i<52*512+1)
pixel10800(i+51*512)=embed_pixels(i,j);

elseif (i>52*512 && i<53*512+1)
pixel10800(i+52*512)=embed_pixels(i,j);

elseif (i>53*512 && i<54*512+1)
pixel10800(i+53*512)=embed_pixels(i,j);

elseif (i>54*512 && i<55*512+1)
pixel10800(i+54*512)=embed_pixels(i,j);

elseif (i>55*512 && i<56*512+1)
pixel10800(i+55*512)=embed_pixels(i,j);

elseif (i>56*512 && i<57*512+1)
pixel10800(i+56*512)=embed_pixels(i,j);

```

```

elseif (i>57*512 && i<58*512+1)
pixel10800(i+57*512)=embed_pixels(i,j);

elseif (i>58*512 && i<59*512+1)
pixel10800(i+58*512)=embed_pixels(i,j);

elseif (i>59*512 && i<60*512+1)
pixel10800(i+59*512)=embed_pixels(i,j);

elseif (i>60*512 && i<61*512+1)
pixel10800(i+60*512)=embed_pixels(i,j);

elseif (i>61*512 && i<62*512+1)
pixel10800(i+61*512)=embed_pixels(i,j);

elseif (i>62*512 && i<63*512+1)
pixel10800(i+62*512)=embed_pixels(i,j);

elseif (i>63*512 && i<64*512+1)
pixel10800(i+63*512)=embed_pixels(i,j);

elseif (i>64*512 && i<65*512+1)
pixel10800(i+64*512)=embed_pixels(i,j);

elseif (i>65*512 && i<66*512+1)
pixel10800(i+65*512)=embed_pixels(i,j);

elseif (i>66*512 && i<67*512+1)
pixel10800(i+66*512)=embed_pixels(i,j);

elseif (i>67*512 && i<68*512+1)
pixel10800(i+67*512)=embed_pixels(i,j);

elseif (i>68*512 && i<69*512+1)
pixel10800(i+68*512)=embed_pixels(i,j);

elseif (i>69*512 && i<70*512+1)
pixel10800(i+69*512)=embed_pixels(i,j);

elseif (i>70*512 && i<71*512+1)
pixel10800(i+70*512)=embed_pixels(i,j);

elseif (i>71*512 && i<72*512+1)
pixel10800(i+71*512)=embed_pixels(i,j);

elseif (i>72*512 && i<73*512+1)
pixel10800(i+72*512)=embed_pixels(i,j);

elseif (i>73*512 && i<74*512+1)
pixel10800(i+73*512)=embed_pixels(i,j);

elseif (i>74*512 && i<75*512+1)
pixel10800(i+74*512)=embed_pixels(i,j);

```

```

elseif (i>75*512 && i<76*512+1)
pixel10800(i+75*512)=embed_pixels(i,j);

elseif (i>76*512 && i<77*512+1)
pixel10800(i+76*512)=embed_pixels(i,j);

elseif (i>77*512 && i<78*512+1)
pixel10800(i+77*512)=embed_pixels(i,j);

elseif (i>78*512 && i<79*512+1)
pixel10800(i+78*512)=embed_pixels(i,j);

elseif (i>79*512 && i<80*512+1)
pixel10800(i+79*512)=embed_pixels(i,j);

elseif (i>80*512 && i<81*512+1)
pixel10800(i+80*512)=embed_pixels(i,j);

elseif (i>81*512 && i<82*512+1)
pixel10800(i+81*512)=embed_pixels(i,j);

elseif (i>82*512 && i<83*512+1)
pixel10800(i+82*512)=embed_pixels(i,j);

elseif (i>83*512 && i<84*512+1)
pixel10800(i+83*512)=embed_pixels(i,j);

elseif (i>84*512 && i<85*512+1)
pixel10800(i+84*512)=embed_pixels(i,j);

elseif (i>85*512 && i<86*512+1)
pixel10800(i+85*512)=embed_pixels(i,j);

elseif (i>86*512 && i<87*512+1)
pixel10800(i+86*512)=embed_pixels(i,j);

elseif (i>87*512 && i<88*512+1)
pixel10800(i+87*512)=embed_pixels(i,j);

elseif (i>88*512 && i<89*512+1)
pixel10800(i+88*512)=embed_pixels(i,j);

elseif (i>89*512 && i<90*512+1)
pixel10800(i+89*512)=embed_pixels(i,j);

elseif (i>90*512 && i<91*512+1)
pixel10800(i+90*512)=embed_pixels(i,j);

elseif (i>91*512 && i<92*512+1)
pixel10800(i+91*512)=embed_pixels(i,j);

elseif (i>92*512 && i<93*512+1)

```

```

pixel10800(i+92*512)=embed_pixels(i,j);

elseif (i>93*512 && i<94*512+1)
pixel10800(i+93*512)=embed_pixels(i,j);

elseif (i>94*512 && i<95*512+1)
pixel10800(i+94*512)=embed_pixels(i,j);

elseif (i>95*512 && i<96*512+1)
pixel10800(i+95*512)=embed_pixels(i,j);

elseif (i>96*512 && i<97*512+1)
pixel10800(i+96*512)=embed_pixels(i,j);

elseif (i>97*512 && i<98*512+1)
pixel10800(i+97*512)=embed_pixels(i,j);

elseif (i>98*512 && i<99*512+1)
pixel10800(i+98*512)=embed_pixels(i,j);

elseif (i>99*512 && i<100*512+1)
pixel10800(i+99*512)=embed_pixels(i,j);

elseif (i>100*512 && i<101*512+1)
pixel10800(i+100*512)=embed_pixels(i,j);

elseif (i>101*512 && i<102*512+1)
pixel10800(i+101*512)=embed_pixels(i,j);

elseif (i>102*512 && i<103*512+1)
pixel10800(i+102*512)=embed_pixels(i,j);

elseif (i>103*512 && i<104*512+1)
pixel10800(i+103*512)=embed_pixels(i,j);

elseif (i>104*512 && i<105*512+1)
pixel10800(i+104*512)=embed_pixels(i,j);

elseif (i>105*512 && i<106*512+1)
pixel10800(i+105*512)=embed_pixels(i,j);

elseif (i>106*512 && i<107*512+1)
pixel10800(i+106*512)=embed_pixels(i,j);

elseif (i>107*512 && i<108*512+1)
pixel10800(i+107*512)=embed_pixels(i,j);

elseif (i>108*512 && i<109*512+1)
pixel10800(i+108*512)=embed_pixels(i,j);

elseif (i>109*512 && i<110*512+1)
pixel10800(i+109*512)=embed_pixels(i,j);

```

```

elseif (i>110*512 && i<111*512+1)
pixel10800(i+110*512)=embed_pixels(i,j);

elseif (i>111*512 && i<112*512+1)
pixel10800(i+111*512)=embed_pixels(i,j);

elseif (i>112*512 && i<113*512+1)
pixel10800(i+112*512)=embed_pixels(i,j);

elseif (i>113*512 && i<114*512+1)
pixel10800(i+113*512)=embed_pixels(i,j);

elseif (i>114*512 && i<115*512+1)
pixel10800(i+114*512)=embed_pixels(i,j);

elseif (i>115*512 && i<116*512+1)
pixel10800(i+115*512)=embed_pixels(i,j);

elseif (i>116*512 && i<117*512+1)
pixel10800(i+116*512)=embed_pixels(i,j);

elseif (i>117*512 && i<118*512+1)
pixel10800(i+117*512)=embed_pixels(i,j);

elseif (i>118*512 && i<119*512+1)
pixel10800(i+118*512)=embed_pixels(i,j);

elseif (i>119*512 && i<120*512+1)
pixel10800(i+119*512)=embed_pixels(i,j);

elseif (i>120*512 && i<121*512+1)
pixel10800(i+120*512)=embed_pixels(i,j);

elseif (i>121*512 && i<122*512+1)
pixel10800(i+121*512)=embed_pixels(i,j);

elseif (i>122*512 && i<123*512+1)
pixel10800(i+122*512)=embed_pixels(i,j);

elseif (i>123*512 && i<124*512+1)
pixel10800(i+123*512)=embed_pixels(i,j);

elseif (i>124*512 && i<125*512+1)
pixel10800(i+124*512)=embed_pixels(i,j);

elseif (i>125*512 && i<126*512+1)
pixel10800(i+125*512)=embed_pixels(i,j);

elseif (i>126*512 && i<127*512+1)
pixel10800(i+126*512)=embed_pixels(i,j);

elseif (i>127*512 && i<128*512+1)
pixel10800(i+127*512)=embed_pixels(i,j);

```

```

elseif (i>128*512 && i<129*512+1)
pixel10800(i+128*512)=embed_pixels(i,j);

elseif (i>129*512 && i<130*512+1)
pixel10800(i+129*512)=embed_pixels(i,j);

elseif (i>130*512 && i<131*512+1)
pixel10800(i+130*512)=embed_pixels(i,j);

elseif (i>131*512 && i<132*512+1)
pixel10800(i+131*512)=embed_pixels(i,j);

elseif (i>132*512 && i<133*512+1)
pixel10800(i+132*512)=embed_pixels(i,j);

elseif (i>133*512 && i<134*512+1)
pixel10800(i+133*512)=embed_pixels(i,j);

elseif (i>134*512 && i<135*512+1)
pixel10800(i+134*512)=embed_pixels(i,j);

elseif (i>135*512 && i<136*512+1)
pixel10800(i+135*512)=embed_pixels(i,j);

elseif (i>136*512 && i<137*512+1)
pixel10800(i+136*512)=embed_pixels(i,j);

elseif (i>137*512 && i<138*512+1)
pixel10800(i+137*512)=embed_pixels(i,j);

elseif (i>138*512 && i<139*512+1)
pixel10800(i+138*512)=embed_pixels(i,j);

elseif (i>139*512 && i<140*512+1)
pixel10800(i+139*512)=embed_pixels(i,j);

elseif (i>140*512 && i<141*512+1)
pixel10800(i+140*512)=embed_pixels(i,j);

elseif (i>141*512 && i<142*512+1)
pixel10800(i+141*512)=embed_pixels(i,j);

elseif (i>142*512 && i<143*512+1)
pixel10800(i+142*512)=embed_pixels(i,j);

elseif (i>143*512 && i<144*512+1)
pixel10800(i+143*512)=embed_pixels(i,j);

elseif (i>144*512 && i<145*512+1)
pixel10800(i+144*512)=embed_pixels(i,j);

elseif (i>145*512 && i<146*512+1)

```

```

pixel10800(i+145*512)=embed_pixels(i,j);

elseif (i>146*512 && i<147*512+1)
pixel10800(i+146*512)=embed_pixels(i,j);

elseif (i>147*512 && i<148*512+1)
pixel10800(i+147*512)=embed_pixels(i,j);

elseif (i>148*512 && i<149*512+1)
pixel10800(i+148*512)=embed_pixels(i,j);

elseif (i>149*512 && i<150*512+1)
pixel10800(i+149*512)=embed_pixels(i,j);

elseif (i>150*512 && i<151*512+1)
pixel10800(i+150*512)=embed_pixels(i,j);

elseif (i>151*512 && i<152*512+1)
pixel10800(i+151*512)=embed_pixels(i,j);

elseif (i>152*512 && i<153*512+1)
pixel10800(i+152*512)=embed_pixels(i,j);

elseif (i>153*512 && i<154*512+1)
pixel10800(i+153*512)=embed_pixels(i,j);

elseif (i>154*512 && i<155*512+1)
pixel10800(i+154*512)=embed_pixels(i,j);

elseif (i>155*512 && i<156*512+1)
pixel10800(i+155*512)=embed_pixels(i,j);

elseif (i>156*512 && i<157*512+1)
pixel10800(i+156*512)=embed_pixels(i,j);

elseif (i>157*512 && i<158*512+1)
pixel10800(i+157*512)=embed_pixels(i,j);

elseif (i>158*512 && i<159*512+1)
pixel10800(i+158*512)=embed_pixels(i,j);

elseif (i>159*512 && i<160*512+1)
pixel10800(i+159*512)=embed_pixels(i,j);

elseif (i>160*512 && i<161*512+1)
pixel10800(i+160*512)=embed_pixels(i,j);

elseif (i>161*512 && i<162*512+1)
pixel10800(i+161*512)=embed_pixels(i,j);

elseif (i>162*512 && i<163*512+1)
pixel10800(i+162*512)=embed_pixels(i,j);

```

```

elseif (i>163*512 && i<164*512+1)
pixel10800(i+163*512)=embed_pixels(i,j);

elseif (i>164*512 && i<165*512+1)
pixel10800(i+164*512)=embed_pixels(i,j);

elseif (i>165*512 && i<166*512+1)
pixel10800(i+165*512)=embed_pixels(i,j);

elseif (i>166*512 && i<167*512+1)
pixel10800(i+166*512)=embed_pixels(i,j);

elseif (i>167*512 && i<168*512+1)
pixel10800(i+167*512)=embed_pixels(i,j);

elseif (i>168*512 && i<169*512+1)
pixel10800(i+168*512)=embed_pixels(i,j);

elseif (i>169*512 && i<170*512+1)
pixel10800(i+169*512)=embed_pixels(i,j);

elseif (i>170*512 && i<171*512+1)
pixel10800(i+170*512)=embed_pixels(i,j);

elseif (i>171*512 && i<172*512+1)
pixel10800(i+171*512)=embed_pixels(i,j);

elseif (i>172*512 && i<173*512+1)
pixel10800(i+172*512)=embed_pixels(i,j);

elseif (i>173*512 && i<174*512+1)
pixel10800(i+173*512)=embed_pixels(i,j);

elseif (i>174*512 && i<175*512+1)
pixel10800(i+174*512)=embed_pixels(i,j);

elseif (i>175*512 && i<176*512+1)
pixel10800(i+175*512)=embed_pixels(i,j);

elseif (i>176*512 && i<177*512+1)
pixel10800(i+176*512)=embed_pixels(i,j);

elseif (i>177*512 && i<178*512+1)
pixel10800(i+177*512)=embed_pixels(i,j);

elseif (i>178*512 && i<179*512+1)
pixel10800(i+178*512)=embed_pixels(i,j);

elseif (i>179*512 && i<180*512+1)
pixel10800(i+179*512)=embed_pixels(i,j);

elseif (i>180*512 && i<181*512+1)
pixel10800(i+180*512)=embed_pixels(i,j);

```

```

elseif (i>181*512 && i<182*512+1)
pixel10800(i+181*512)=embed_pixels(i,j);

elseif (i>182*512 && i<183*512+1)
pixel10800(i+182*512)=embed_pixels(i,j);

elseif (i>183*512 && i<184*512+1)
pixel10800(i+183*512)=embed_pixels(i,j);

elseif (i>184*512 && i<185*512+1)
pixel10800(i+184*512)=embed_pixels(i,j);

elseif (i>185*512 && i<186*512+1)
pixel10800(i+185*512)=embed_pixels(i,j);

elseif (i>186*512 && i<187*512+1)
pixel10800(i+186*512)=embed_pixels(i,j);

elseif (i>187*512 && i<188*512+1)
pixel10800(i+187*512)=embed_pixels(i,j);

elseif (i>188*512 && i<189*512+1)
pixel10800(i+188*512)=embed_pixels(i,j);

elseif (i>189*512 && i<190*512+1)
pixel10800(i+189*512)=embed_pixels(i,j);

elseif (i>190*512 && i<191*512+1)
pixel10800(i+190*512)=embed_pixels(i,j);

elseif (i>191*512 && i<192*512+1)
pixel10800(i+191*512)=embed_pixels(i,j);

elseif (i>192*512 && i<193*512+1)
pixel10800(i+192*512)=embed_pixels(i,j);

elseif (i>193*512 && i<194*512+1)
pixel10800(i+193*512)=embed_pixels(i,j);

elseif (i>194*512 && i<195*512+1)
pixel10800(i+194*512)=embed_pixels(i,j);

elseif (i>195*512 && i<196*512+1)
pixel10800(i+195*512)=embed_pixels(i,j);

elseif (i>196*512 && i<197*512+1)
pixel10800(i+196*512)=embed_pixels(i,j);

elseif (i>197*512 && i<198*512+1)
pixel10800(i+197*512)=embed_pixels(i,j);

elseif (i>198*512 && i<199*512+1)

```

```

pixel10800(i+198*512)=embed_pixels(i,j);

elseif (i>199*512 && i<200*512+1)
pixel10800(i+199*512)=embed_pixels(i,j);

elseif (i>200*512 && i<201*512+1)
pixel10800(i+200*512)=embed_pixels(i,j);

elseif (i>201*512 && i<202*512+1)
pixel10800(i+201*512)=embed_pixels(i,j);

elseif (i>202*512 && i<203*512+1)
pixel10800(i+202*512)=embed_pixels(i,j);

elseif (i>203*512 && i<204*512+1)
pixel10800(i+203*512)=embed_pixels(i,j);

elseif (i>204*512 && i<205*512+1)
pixel10800(i+204*512)=embed_pixels(i,j);

elseif (i>205*512 && i<206*512+1)
pixel10800(i+205*512)=embed_pixels(i,j);

elseif (i>206*512 && i<207*512+1)
pixel10800(i+206*512)=embed_pixels(i,j);

elseif (i>207*512 && i<208*512+1)
pixel10800(i+207*512)=embed_pixels(i,j);

elseif (i>208*512 && i<209*512+1)
pixel10800(i+208*512)=embed_pixels(i,j);

elseif (i>209*512 && i<210*512+1)
pixel10800(i+209*512)=embed_pixels(i,j);

elseif (i>210*512 && i<211*512+1)
pixel10800(i+210*512)=embed_pixels(i,j);

elseif (i>211*512 && i<212*512+1)
pixel10800(i+211*512)=embed_pixels(i,j);

elseif (i>212*512 && i<213*512+1)
pixel10800(i+212*512)=embed_pixels(i,j);

elseif (i>213*512 && i<214*512+1)
pixel10800(i+213*512)=embed_pixels(i,j);

elseif (i>214*512 && i<215*512+1)
pixel10800(i+214*512)=embed_pixels(i,j);

elseif (i>215*512 && i<216*512+1)
pixel10800(i+215*512)=embed_pixels(i,j);

```

```

elseif (i>216*512 && i<217*512+1)
pixel10800(i+216*512)=embed_pixels(i,j);

elseif (i>217*512 && i<218*512+1)
pixel10800(i+217*512)=embed_pixels(i,j);

elseif (i>218*512 && i<219*512+1)
pixel10800(i+218*512)=embed_pixels(i,j);

elseif (i>219*512 && i<220*512+1)
pixel10800(i+219*512)=embed_pixels(i,j);

elseif (i>220*512 && i<221*512+1)
pixel10800(i+220*512)=embed_pixels(i,j);

elseif (i>221*512 && i<222*512+1)
pixel10800(i+221*512)=embed_pixels(i,j);

elseif (i>222*512 && i<223*512+1)
pixel10800(i+222*512)=embed_pixels(i,j);

elseif (i>223*512 && i<224*512+1)
pixel10800(i+223*512)=embed_pixels(i,j);

elseif (i>224*512 && i<225*512+1)
pixel10800(i+224*512)=embed_pixels(i,j);

elseif (i>225*512 && i<226*512+1)
pixel10800(i+225*512)=embed_pixels(i,j);

elseif (i>226*512 && i<227*512+1)
pixel10800(i+226*512)=embed_pixels(i,j);

elseif (i>227*512 && i<228*512+1)
pixel10800(i+227*512)=embed_pixels(i,j);

elseif (i>228*512 && i<229*512+1)
pixel10800(i+228*512)=embed_pixels(i,j);

elseif (i>229*512 && i<230*512+1)
pixel10800(i+229*512)=embed_pixels(i,j);

elseif (i>230*512 && i<231*512+1)
pixel10800(i+230*512)=embed_pixels(i,j);

elseif (i>231*512 && i<232*512+1)
pixel10800(i+231*512)=embed_pixels(i,j);

elseif (i>232*512 && i<233*512+1)
pixel10800(i+232*512)=embed_pixels(i,j);

elseif (i>233*512 && i<234*512+1)
pixel10800(i+233*512)=embed_pixels(i,j);

```

```

elseif (i>234*512 && i<235*512+1)
pixel10800(i+234*512)=embed_pixels(i,j);

elseif (i>235*512 && i<236*512+1)
pixel10800(i+235*512)=embed_pixels(i,j);

elseif (i>236*512 && i<237*512+1)
pixel10800(i+236*512)=embed_pixels(i,j);

elseif (i>237*512 && i<238*512+1)
pixel10800(i+237*512)=embed_pixels(i,j);

elseif (i>238*512 && i<239*512+1)
pixel10800(i+238*512)=embed_pixels(i,j);

elseif (i>239*512 && i<240*512+1)
pixel10800(i+239*512)=embed_pixels(i,j);

elseif (i>240*512 && i<241*512+1)
pixel10800(i+240*512)=embed_pixels(i,j);

elseif (i>241*512 && i<242*512+1)
pixel10800(i+241*512)=embed_pixels(i,j);

elseif (i>242*512 && i<243*512+1)
pixel10800(i+242*512)=embed_pixels(i,j);

elseif (i>243*512 && i<244*512+1)
pixel10800(i+243*512)=embed_pixels(i,j);

elseif (i>244*512 && i<245*512+1)
pixel10800(i+244*512)=embed_pixels(i,j);

elseif (i>245*512 && i<246*512+1)
pixel10800(i+245*512)=embed_pixels(i,j);

elseif (i>246*512 && i<247*512+1)
pixel10800(i+246*512)=embed_pixels(i,j);

elseif (i>247*512 && i<248*512+1)
pixel10800(i+247*512)=embed_pixels(i,j);

elseif (i>248*512 && i<249*512+1)
pixel10800(i+248*512)=embed_pixels(i,j);

elseif (i>249*512 && i<250*512+1)
pixel10800(i+249*512)=embed_pixels(i,j);

elseif (i>250*512 && i<251*512+1)
pixel10800(i+250*512)=embed_pixels(i,j);

elseif (i>251*512 && i<252*512+1)

```

```

pixel10800(i+251*512)=embed_pixels(i,j);

elseif (i>252*512 && i<253*512+1)
pixel10800(i+252*512)=embed_pixels(i,j);

elseif (i>253*512 && i<254*512+1)
pixel10800(i+253*512)=embed_pixels(i,j);

elseif (i>254*512 && i<255*512+1)
pixel10800(i+254*512)=embed_pixels(i,j);

elseif (i>255*512 && i<256*512+1)
pixel10800(i+255*512)=embed_pixels(i,j);

end
end

for i=1:131072
j=2;

if(i>0 && i<1*512+1)
pixel10800(i+1*512)=embed_pixels(i,j);

elseif (i>1*512 && i<2*512+1)
pixel10800(i+2*512)=embed_pixels(i,j);

elseif (i>2*512 && i<3*512+1)
pixel10800(i+3*512)=embed_pixels(i,j);

elseif (i>3*512 && i<4*512+1)
pixel10800(i+4*512)=embed_pixels(i,j);

elseif (i>4*512 && i<5*512+1)
pixel10800(i+5*512)=embed_pixels(i,j);

elseif (i>5*512 && i<6*512+1)
pixel10800(i+6*512)=embed_pixels(i,j);

elseif (i>6*512 && i<7*512+1)
pixel10800(i+7*512)=embed_pixels(i,j);

elseif (i>7*512 && i<8*512+1)
pixel10800(i+8*512)=embed_pixels(i,j);

elseif (i>8*512 && i<9*512+1)
pixel10800(i+9*512)=embed_pixels(i,j);

elseif (i>9*512 && i<10*512+1)
pixel10800(i+10*512)=embed_pixels(i,j);

elseif (i>10*512 && i<11*512+1)

```

```

pixel10800(i+11*512)=embed_pixels(i,j);

elseif (i>11*512 && i<12*512+1)
pixel10800(i+12*512)=embed_pixels(i,j);

elseif (i>12*512 && i<13*512+1)
pixel10800(i+13*512)=embed_pixels(i,j);

elseif (i>13*512 && i<14*512+1)
pixel10800(i+14*512)=embed_pixels(i,j);

elseif (i>14*512 && i<15*512+1)
pixel10800(i+15*512)=embed_pixels(i,j);

elseif (i>15*512 && i<16*512+1)
pixel10800(i+16*512)=embed_pixels(i,j);

elseif (i>16*512 && i<17*512+1)
pixel10800(i+17*512)=embed_pixels(i,j);

elseif (i>17*512 && i<18*512+1)
pixel10800(i+18*512)=embed_pixels(i,j);

elseif (i>18*512 && i<19*512+1)
pixel10800(i+19*512)=embed_pixels(i,j);

elseif (i>19*512 && i<20*512+1)
pixel10800(i+20*512)=embed_pixels(i,j);

elseif (i>20*512 && i<21*512+1)
pixel10800(i+21*512)=embed_pixels(i,j);

elseif (i>21*512 && i<22*512+1)
pixel10800(i+22*512)=embed_pixels(i,j);

elseif (i>22*512 && i<23*512+1)
pixel10800(i+23*512)=embed_pixels(i,j);

elseif (i>23*512 && i<24*512+1)
pixel10800(i+24*512)=embed_pixels(i,j);

elseif (i>24*512 && i<25*512+1)
pixel10800(i+25*512)=embed_pixels(i,j);

elseif (i>25*512 && i<26*512+1)
pixel10800(i+26*512)=embed_pixels(i,j);

elseif (i>26*512 && i<27*512+1)
pixel10800(i+27*512)=embed_pixels(i,j);

elseif (i>27*512 && i<28*512+1)
pixel10800(i+28*512)=embed_pixels(i,j);

```

```

elseif (i>28*512 && i<29*512+1)
pixel10800(i+29*512)=embed_pixels(i,j);

elseif (i>29*512 && i<30*512+1)
pixel10800(i+30*512)=embed_pixels(i,j);

elseif (i>30*512 && i<31*512+1)
pixel10800(i+31*512)=embed_pixels(i,j);

elseif (i>31*512 && i<32*512+1)
pixel10800(i+32*512)=embed_pixels(i,j);

elseif (i>32*512 && i<33*512+1)
pixel10800(i+33*512)=embed_pixels(i,j);

elseif (i>33*512 && i<34*512+1)
pixel10800(i+34*512)=embed_pixels(i,j);

elseif (i>34*512 && i<35*512+1)
pixel10800(i+35*512)=embed_pixels(i,j);

elseif (i>35*512 && i<36*512+1)
pixel10800(i+36*512)=embed_pixels(i,j);

elseif (i>36*512 && i<37*512+1)
pixel10800(i+37*512)=embed_pixels(i,j);

elseif (i>37*512 && i<38*512+1)
pixel10800(i+38*512)=embed_pixels(i,j);

elseif (i>38*512 && i<39*512+1)
pixel10800(i+39*512)=embed_pixels(i,j);

elseif (i>39*512 && i<40*512+1)
pixel10800(i+40*512)=embed_pixels(i,j);

elseif (i>40*512 && i<41*512+1)
pixel10800(i+41*512)=embed_pixels(i,j);

elseif (i>41*512 && i<42*512+1)
pixel10800(i+42*512)=embed_pixels(i,j);

elseif (i>42*512 && i<43*512+1)
pixel10800(i+43*512)=embed_pixels(i,j);

elseif (i>43*512 && i<44*512+1)
pixel10800(i+44*512)=embed_pixels(i,j);

elseif (i>44*512 && i<45*512+1)
pixel10800(i+45*512)=embed_pixels(i,j);

elseif (i>45*512 && i<46*512+1)
pixel10800(i+46*512)=embed_pixels(i,j);

```

```

elseif (i>46*512 && i<47*512+1)
pixel10800(i+47*512)=embed_pixels(i,j);

elseif (i>47*512 && i<48*512+1)
pixel10800(i+48*512)=embed_pixels(i,j);

elseif (i>48*512 && i<49*512+1)
pixel10800(i+49*512)=embed_pixels(i,j);

elseif (i>49*512 && i<50*512+1)
pixel10800(i+50*512)=embed_pixels(i,j);

elseif (i>50*512 && i<51*512+1)
pixel10800(i+51*512)=embed_pixels(i,j);

elseif (i>51*512 && i<52*512+1)
pixel10800(i+52*512)=embed_pixels(i,j);

elseif (i>52*512 && i<53*512+1)
pixel10800(i+53*512)=embed_pixels(i,j);

elseif (i>53*512 && i<54*512+1)
pixel10800(i+54*512)=embed_pixels(i,j);

elseif (i>54*512 && i<55*512+1)
pixel10800(i+55*512)=embed_pixels(i,j);

elseif (i>55*512 && i<56*512+1)
pixel10800(i+56*512)=embed_pixels(i,j);

elseif (i>56*512 && i<57*512+1)
pixel10800(i+57*512)=embed_pixels(i,j);

elseif (i>57*512 && i<58*512+1)
pixel10800(i+58*512)=embed_pixels(i,j);

elseif (i>58*512 && i<59*512+1)
pixel10800(i+59*512)=embed_pixels(i,j);

elseif (i>59*512 && i<60*512+1)
pixel10800(i+60*512)=embed_pixels(i,j);

elseif (i>60*512 && i<61*512+1)
pixel10800(i+61*512)=embed_pixels(i,j);

elseif (i>61*512 && i<62*512+1)
pixel10800(i+62*512)=embed_pixels(i,j);

elseif (i>62*512 && i<63*512+1)
pixel10800(i+63*512)=embed_pixels(i,j);

elseif (i>63*512 && i<64*512+1)

```

```

pixel10800(i+64*512)=embed_pixels(i,j);

elseif (i>64*512 && i<65*512+1)
pixel10800(i+65*512)=embed_pixels(i,j);

elseif (i>65*512 && i<66*512+1)
pixel10800(i+66*512)=embed_pixels(i,j);

elseif (i>66*512 && i<67*512+1)
pixel10800(i+67*512)=embed_pixels(i,j);

elseif (i>67*512 && i<68*512+1)
pixel10800(i+68*512)=embed_pixels(i,j);

elseif (i>68*512 && i<69*512+1)
pixel10800(i+69*512)=embed_pixels(i,j);

elseif (i>69*512 && i<70*512+1)
pixel10800(i+70*512)=embed_pixels(i,j);

elseif (i>70*512 && i<71*512+1)
pixel10800(i+71*512)=embed_pixels(i,j);

elseif (i>71*512 && i<72*512+1)
pixel10800(i+72*512)=embed_pixels(i,j);

elseif (i>72*512 && i<73*512+1)
pixel10800(i+73*512)=embed_pixels(i,j);

elseif (i>73*512 && i<74*512+1)
pixel10800(i+74*512)=embed_pixels(i,j);

elseif (i>74*512 && i<75*512+1)
pixel10800(i+75*512)=embed_pixels(i,j);

elseif (i>75*512 && i<76*512+1)
pixel10800(i+76*512)=embed_pixels(i,j);

elseif (i>76*512 && i<77*512+1)
pixel10800(i+77*512)=embed_pixels(i,j);

elseif (i>77*512 && i<78*512+1)
pixel10800(i+78*512)=embed_pixels(i,j);

elseif (i>78*512 && i<79*512+1)
pixel10800(i+79*512)=embed_pixels(i,j);

elseif (i>79*512 && i<80*512+1)
pixel10800(i+80*512)=embed_pixels(i,j);

elseif (i>80*512 && i<81*512+1)
pixel10800(i+81*512)=embed_pixels(i,j);

```

```

elseif (i>81*512 && i<82*512+1)
pixel10800(i+82*512)=embed_pixels(i,j);

elseif (i>82*512 && i<83*512+1)
pixel10800(i+83*512)=embed_pixels(i,j);

elseif (i>83*512 && i<84*512+1)
pixel10800(i+84*512)=embed_pixels(i,j);

elseif (i>84*512 && i<85*512+1)
pixel10800(i+85*512)=embed_pixels(i,j);

elseif (i>85*512 && i<86*512+1)
pixel10800(i+86*512)=embed_pixels(i,j);

elseif (i>86*512 && i<87*512+1)
pixel10800(i+87*512)=embed_pixels(i,j);

elseif (i>87*512 && i<88*512+1)
pixel10800(i+88*512)=embed_pixels(i,j);

elseif (i>88*512 && i<89*512+1)
pixel10800(i+89*512)=embed_pixels(i,j);

elseif (i>89*512 && i<90*512+1)
pixel10800(i+90*512)=embed_pixels(i,j);

elseif (i>90*512 && i<91*512+1)
pixel10800(i+91*512)=embed_pixels(i,j);

elseif (i>91*512 && i<92*512+1)
pixel10800(i+92*512)=embed_pixels(i,j);

elseif (i>92*512 && i<93*512+1)
pixel10800(i+93*512)=embed_pixels(i,j);

elseif (i>93*512 && i<94*512+1)
pixel10800(i+94*512)=embed_pixels(i,j);

elseif (i>94*512 && i<95*512+1)
pixel10800(i+95*512)=embed_pixels(i,j);

elseif (i>95*512 && i<96*512+1)
pixel10800(i+96*512)=embed_pixels(i,j);

elseif (i>96*512 && i<97*512+1)
pixel10800(i+97*512)=embed_pixels(i,j);

elseif (i>97*512 && i<98*512+1)
pixel10800(i+98*512)=embed_pixels(i,j);

elseif (i>98*512 && i<99*512+1)
pixel10800(i+99*512)=embed_pixels(i,j);

```

```

elseif (i>99*512 && i<100*512+1)
pixel10800(i+100*512)=embed_pixels(i,j);

elseif (i>100*512 && i<101*512+1)
pixel10800(i+101*512)=embed_pixels(i,j);

elseif (i>101*512 && i<102*512+1)
pixel10800(i+102*512)=embed_pixels(i,j);

elseif (i>102*512 && i<103*512+1)
pixel10800(i+103*512)=embed_pixels(i,j);

elseif (i>103*512 && i<104*512+1)
pixel10800(i+104*512)=embed_pixels(i,j);

elseif (i>104*512 && i<105*512+1)
pixel10800(i+105*512)=embed_pixels(i,j);

elseif (i>105*512 && i<106*512+1)
pixel10800(i+106*512)=embed_pixels(i,j);

elseif (i>106*512 && i<107*512+1)
pixel10800(i+107*512)=embed_pixels(i,j);

elseif (i>107*512 && i<108*512+1)
pixel10800(i+108*512)=embed_pixels(i,j);

elseif (i>108*512 && i<109*512+1)
pixel10800(i+109*512)=embed_pixels(i,j);

elseif (i>109*512 && i<110*512+1)
pixel10800(i+110*512)=embed_pixels(i,j);

elseif (i>110*512 && i<111*512+1)
pixel10800(i+111*512)=embed_pixels(i,j);

elseif (i>111*512 && i<112*512+1)
pixel10800(i+112*512)=embed_pixels(i,j);

elseif (i>112*512 && i<113*512+1)
pixel10800(i+113*512)=embed_pixels(i,j);

elseif (i>113*512 && i<114*512+1)
pixel10800(i+114*512)=embed_pixels(i,j);

elseif (i>114*512 && i<115*512+1)
pixel10800(i+115*512)=embed_pixels(i,j);

elseif (i>115*512 && i<116*512+1)
pixel10800(i+116*512)=embed_pixels(i,j);

```

```

elseif (i>116*512 && i<117*512+1)
pixel10800(i+117*512)=embed_pixels(i,j);

elseif (i>117*512 && i<118*512+1)
pixel10800(i+118*512)=embed_pixels(i,j);

elseif (i>118*512 && i<119*512+1)
pixel10800(i+119*512)=embed_pixels(i,j);

elseif (i>119*512 && i<120*512+1)
pixel10800(i+120*512)=embed_pixels(i,j);

elseif (i>120*512 && i<121*512+1)
pixel10800(i+121*512)=embed_pixels(i,j);

elseif (i>121*512 && i<122*512+1)
pixel10800(i+122*512)=embed_pixels(i,j);

elseif (i>122*512 && i<123*512+1)
pixel10800(i+123*512)=embed_pixels(i,j);

elseif (i>123*512 && i<124*512+1)
pixel10800(i+124*512)=embed_pixels(i,j);

elseif (i>124*512 && i<125*512+1)
pixel10800(i+125*512)=embed_pixels(i,j);

elseif (i>125*512 && i<126*512+1)
pixel10800(i+126*512)=embed_pixels(i,j);

elseif (i>126*512 && i<127*512+1)
pixel10800(i+127*512)=embed_pixels(i,j);

elseif (i>127*512 && i<128*512+1)
pixel10800(i+128*512)=embed_pixels(i,j);

elseif (i>128*512 && i<129*512+1)
pixel10800(i+129*512)=embed_pixels(i,j);

elseif (i>129*512 && i<130*512+1)
pixel10800(i+130*512)=embed_pixels(i,j);

elseif (i>130*512 && i<131*512+1)
pixel10800(i+131*512)=embed_pixels(i,j);

elseif (i>131*512 && i<132*512+1)
pixel10800(i+132*512)=embed_pixels(i,j);

elseif (i>132*512 && i<133*512+1)
pixel10800(i+133*512)=embed_pixels(i,j);

elseif (i>133*512 && i<134*512+1)
pixel10800(i+134*512)=embed_pixels(i,j);

```

```

elseif (i>134*512 && i<135*512+1)
pixel10800(i+135*512)=embed_pixels(i,j);

elseif (i>135*512 && i<136*512+1)
pixel10800(i+136*512)=embed_pixels(i,j);

elseif (i>136*512 && i<137*512+1)
pixel10800(i+137*512)=embed_pixels(i,j);

elseif (i>137*512 && i<138*512+1)
pixel10800(i+138*512)=embed_pixels(i,j);

elseif (i>138*512 && i<139*512+1)
pixel10800(i+139*512)=embed_pixels(i,j);

elseif (i>139*512 && i<140*512+1)
pixel10800(i+140*512)=embed_pixels(i,j);

elseif (i>140*512 && i<141*512+1)
pixel10800(i+141*512)=embed_pixels(i,j);

elseif (i>141*512 && i<142*512+1)
pixel10800(i+142*512)=embed_pixels(i,j);

elseif (i>142*512 && i<143*512+1)
pixel10800(i+143*512)=embed_pixels(i,j);

elseif (i>143*512 && i<144*512+1)
pixel10800(i+144*512)=embed_pixels(i,j);

elseif (i>144*512 && i<145*512+1)
pixel10800(i+145*512)=embed_pixels(i,j);

elseif (i>145*512 && i<146*512+1)
pixel10800(i+146*512)=embed_pixels(i,j);

elseif (i>146*512 && i<147*512+1)
pixel10800(i+147*512)=embed_pixels(i,j);

elseif (i>147*512 && i<148*512+1)
pixel10800(i+148*512)=embed_pixels(i,j);

elseif (i>148*512 && i<149*512+1)
pixel10800(i+149*512)=embed_pixels(i,j);

elseif (i>149*512 && i<150*512+1)
pixel10800(i+150*512)=embed_pixels(i,j);

elseif (i>150*512 && i<151*512+1)
pixel10800(i+151*512)=embed_pixels(i,j);

elseif (i>151*512 && i<152*512+1)

```

```

pixel10800(i+152*512)=embed_pixels(i,j);

elseif (i>152*512 && i<153*512+1)
pixel10800(i+153*512)=embed_pixels(i,j);

elseif (i>153*512 && i<154*512+1)
pixel10800(i+154*512)=embed_pixels(i,j);

elseif (i>154*512 && i<155*512+1)
pixel10800(i+155*512)=embed_pixels(i,j);

elseif (i>155*512 && i<156*512+1)
pixel10800(i+156*512)=embed_pixels(i,j);

elseif (i>156*512 && i<157*512+1)
pixel10800(i+157*512)=embed_pixels(i,j);

elseif (i>157*512 && i<158*512+1)
pixel10800(i+158*512)=embed_pixels(i,j);

elseif (i>158*512 && i<159*512+1)
pixel10800(i+159*512)=embed_pixels(i,j);

elseif (i>159*512 && i<160*512+1)
pixel10800(i+160*512)=embed_pixels(i,j);

elseif (i>160*512 && i<161*512+1)
pixel10800(i+161*512)=embed_pixels(i,j);

elseif (i>161*512 && i<162*512+1)
pixel10800(i+162*512)=embed_pixels(i,j);

elseif (i>162*512 && i<163*512+1)
pixel10800(i+163*512)=embed_pixels(i,j);

elseif (i>163*512 && i<164*512+1)
pixel10800(i+164*512)=embed_pixels(i,j);

elseif (i>164*512 && i<165*512+1)
pixel10800(i+165*512)=embed_pixels(i,j);

elseif (i>165*512 && i<166*512+1)
pixel10800(i+166*512)=embed_pixels(i,j);

elseif (i>166*512 && i<167*512+1)
pixel10800(i+167*512)=embed_pixels(i,j);

elseif (i>167*512 && i<168*512+1)
pixel10800(i+168*512)=embed_pixels(i,j);

elseif (i>168*512 && i<169*512+1)
pixel10800(i+169*512)=embed_pixels(i,j);

```

```

elseif (i>169*512 && i<170*512+1)
pixel10800(i+170*512)=embed_pixels(i,j);

elseif (i>170*512 && i<171*512+1)
pixel10800(i+171*512)=embed_pixels(i,j);

elseif (i>171*512 && i<172*512+1)
pixel10800(i+172*512)=embed_pixels(i,j);

elseif (i>172*512 && i<173*512+1)
pixel10800(i+173*512)=embed_pixels(i,j);

elseif (i>173*512 && i<174*512+1)
pixel10800(i+174*512)=embed_pixels(i,j);

elseif (i>174*512 && i<175*512+1)
pixel10800(i+175*512)=embed_pixels(i,j);

elseif (i>175*512 && i<176*512+1)
pixel10800(i+176*512)=embed_pixels(i,j);

elseif (i>176*512 && i<177*512+1)
pixel10800(i+177*512)=embed_pixels(i,j);

elseif (i>177*512 && i<178*512+1)
pixel10800(i+178*512)=embed_pixels(i,j);

elseif (i>178*512 && i<179*512+1)
pixel10800(i+179*512)=embed_pixels(i,j);

elseif (i>179*512 && i<180*512+1)
pixel10800(i+180*512)=embed_pixels(i,j);

elseif (i>180*512 && i<181*512+1)
pixel10800(i+181*512)=embed_pixels(i,j);

elseif (i>181*512 && i<182*512+1)
pixel10800(i+182*512)=embed_pixels(i,j);

elseif (i>182*512 && i<183*512+1)
pixel10800(i+183*512)=embed_pixels(i,j);

elseif (i>183*512 && i<184*512+1)
pixel10800(i+184*512)=embed_pixels(i,j);

elseif (i>184*512 && i<185*512+1)
pixel10800(i+185*512)=embed_pixels(i,j);

elseif (i>185*512 && i<186*512+1)
pixel10800(i+186*512)=embed_pixels(i,j);

elseif (i>186*512 && i<187*512+1)
pixel10800(i+187*512)=embed_pixels(i,j);

```

```

elseif (i>187*512 && i<188*512+1)
pixel10800(i+188*512)=embed_pixels(i,j);

elseif (i>188*512 && i<189*512+1)
pixel10800(i+189*512)=embed_pixels(i,j);

elseif (i>189*512 && i<190*512+1)
pixel10800(i+190*512)=embed_pixels(i,j);

elseif (i>190*512 && i<191*512+1)
pixel10800(i+191*512)=embed_pixels(i,j);

elseif (i>191*512 && i<192*512+1)
pixel10800(i+192*512)=embed_pixels(i,j);

elseif (i>192*512 && i<193*512+1)
pixel10800(i+193*512)=embed_pixels(i,j);

elseif (i>193*512 && i<194*512+1)
pixel10800(i+194*512)=embed_pixels(i,j);

elseif (i>194*512 && i<195*512+1)
pixel10800(i+195*512)=embed_pixels(i,j);

elseif (i>195*512 && i<196*512+1)
pixel10800(i+196*512)=embed_pixels(i,j);

elseif (i>196*512 && i<197*512+1)
pixel10800(i+197*512)=embed_pixels(i,j);

elseif (i>197*512 && i<198*512+1)
pixel10800(i+198*512)=embed_pixels(i,j);

elseif (i>198*512 && i<199*512+1)
pixel10800(i+199*512)=embed_pixels(i,j);

elseif (i>199*512 && i<200*512+1)
pixel10800(i+200*512)=embed_pixels(i,j);

elseif (i>200*512 && i<201*512+1)
pixel10800(i+201*512)=embed_pixels(i,j);

elseif (i>201*512 && i<202*512+1)
pixel10800(i+202*512)=embed_pixels(i,j);

elseif (i>202*512 && i<203*512+1)
pixel10800(i+203*512)=embed_pixels(i,j);

elseif (i>203*512 && i<204*512+1)
pixel10800(i+204*512)=embed_pixels(i,j);

```

```

elseif (i>204*512 && i<205*512+1)
pixel10800(i+205*512)=embed_pixels(i,j);

elseif (i>205*512 && i<206*512+1)
pixel10800(i+206*512)=embed_pixels(i,j);

elseif (i>206*512 && i<207*512+1)
pixel10800(i+207*512)=embed_pixels(i,j);

elseif (i>207*512 && i<208*512+1)
pixel10800(i+208*512)=embed_pixels(i,j);

elseif (i>208*512 && i<209*512+1)
pixel10800(i+209*512)=embed_pixels(i,j);

elseif (i>209*512 && i<210*512+1)
pixel10800(i+210*512)=embed_pixels(i,j);

elseif (i>210*512 && i<211*512+1)
pixel10800(i+211*512)=embed_pixels(i,j);

elseif (i>211*512 && i<212*512+1)
pixel10800(i+212*512)=embed_pixels(i,j);

elseif (i>212*512 && i<213*512+1)
pixel10800(i+213*512)=embed_pixels(i,j);

elseif (i>213*512 && i<214*512+1)
pixel10800(i+214*512)=embed_pixels(i,j);

elseif (i>214*512 && i<215*512+1)
pixel10800(i+215*512)=embed_pixels(i,j);

elseif (i>215*512 && i<216*512+1)
pixel10800(i+216*512)=embed_pixels(i,j);

elseif (i>216*512 && i<217*512+1)
pixel10800(i+217*512)=embed_pixels(i,j);

elseif (i>217*512 && i<218*512+1)
pixel10800(i+218*512)=embed_pixels(i,j);

elseif (i>218*512 && i<219*512+1)
pixel10800(i+219*512)=embed_pixels(i,j);

elseif (i>219*512 && i<220*512+1)
pixel10800(i+220*512)=embed_pixels(i,j);

elseif (i>220*512 && i<221*512+1)
pixel10800(i+221*512)=embed_pixels(i,j);

```

```

elseif (i>221*512 && i<222*512+1)
pixel10800(i+222*512)=embed_pixels(i,j);

elseif (i>222*512 && i<223*512+1)
pixel10800(i+223*512)=embed_pixels(i,j);

elseif (i>223*512 && i<224*512+1)
pixel10800(i+224*512)=embed_pixels(i,j);

elseif (i>224*512 && i<225*512+1)
pixel10800(i+225*512)=embed_pixels(i,j);

elseif (i>225*512 && i<226*512+1)
pixel10800(i+226*512)=embed_pixels(i,j);

elseif (i>226*512 && i<227*512+1)
pixel10800(i+227*512)=embed_pixels(i,j);

elseif (i>227*512 && i<228*512+1)
pixel10800(i+228*512)=embed_pixels(i,j);

elseif (i>228*512 && i<229*512+1)
pixel10800(i+229*512)=embed_pixels(i,j);

elseif (i>229*512 && i<230*512+1)
pixel10800(i+230*512)=embed_pixels(i,j);

elseif (i>230*512 && i<231*512+1)
pixel10800(i+231*512)=embed_pixels(i,j);

elseif (i>231*512 && i<232*512+1)
pixel10800(i+232*512)=embed_pixels(i,j);

elseif (i>232*512 && i<233*512+1)
pixel10800(i+233*512)=embed_pixels(i,j);

elseif (i>233*512 && i<234*512+1)
pixel10800(i+234*512)=embed_pixels(i,j);

elseif (i>234*512 && i<235*512+1)
pixel10800(i+235*512)=embed_pixels(i,j);

elseif (i>235*512 && i<236*512+1)
pixel10800(i+236*512)=embed_pixels(i,j);

elseif (i>236*512 && i<237*512+1)
pixel10800(i+237*512)=embed_pixels(i,j);

elseif (i>237*512 && i<238*512+1)
pixel10800(i+238*512)=embed_pixels(i,j);

elseif (i>238*512 && i<239*512+1)
pixel10800(i+239*512)=embed_pixels(i,j);

```

```

elseif (i>239*512 && i<240*512+1)
pixel10800(i+240*512)=embed_pixels(i,j);

elseif (i>240*512 && i<241*512+1)
pixel10800(i+241*512)=embed_pixels(i,j);

elseif (i>241*512 && i<242*512+1)
pixel10800(i+242*512)=embed_pixels(i,j);

elseif (i>242*512 && i<243*512+1)
pixel10800(i+243*512)=embed_pixels(i,j);

elseif (i>243*512 && i<244*512+1)
pixel10800(i+244*512)=embed_pixels(i,j);

elseif (i>244*512 && i<245*512+1)
pixel10800(i+245*512)=embed_pixels(i,j);

elseif (i>245*512 && i<246*512+1)
pixel10800(i+246*512)=embed_pixels(i,j);

elseif (i>246*512 && i<247*512+1)
pixel10800(i+247*512)=embed_pixels(i,j);

elseif (i>247*512 && i<248*512+1)
pixel10800(i+248*512)=embed_pixels(i,j);

elseif (i>248*512 && i<249*512+1)
pixel10800(i+249*512)=embed_pixels(i,j);

elseif (i>249*512 && i<250*512+1)
pixel10800(i+250*512)=embed_pixels(i,j);

elseif (i>250*512 && i<251*512+1)
pixel10800(i+251*512)=embed_pixels(i,j);

elseif (i>251*512 && i<252*512+1)
pixel10800(i+252*512)=embed_pixels(i,j);

elseif (i>252*512 && i<253*512+1)
pixel10800(i+253*512)=embed_pixels(i,j);

elseif (i>253*512 && i<254*512+1)
pixel10800(i+254*512)=embed_pixels(i,j);

elseif (i>254*512 && i<255*512+1)
pixel10800(i+255*512)=embed_pixels(i,j);

elseif (i>255*512 && i<256*512+1)
pixel10800(i+256*512)=embed_pixels(i,j);

end

```

```
end  
  
imshow (pixel10800)
```

#### Reverse\_watermarking.m

```
%grouping the pixel pairs  
img=imread('C:\Users\Hasan\Desktop\Test image reverse\flower1.png');  
img=rgb2gray (img);  
[rows,cols]=size(img);  
[x,y]=ndgrid(1:rows,1:2:cols);  
ind=sub2ind(size(img),x,y);  
ind_shift=sub2ind(size(img),x,y+1);  
pixels1=img(ind);  
pixels2=img(ind_shift);  
pixels=[pixels1(:) pixels2(:)];  
[row1,cols1]=size(pixels);  
  
pixels=uint16 (pixels);
```

#### Bit\_extraction.m

```
%Data=logical(A);  
%char databit;  
pos=1;  
for m=1:131072  
    for i=1:1  
  
        if bitget(pixels(m,i),1)==1  
            databit (pos) =bitget (pixels (m,i+1),1);  
            mainpixel (m,i)=ceil ((2*pixels(m,i)+1*pixels(m,i+1))/3);  
            mainpixel (m,i+1)=ceil ((1*pixels(m,i)+2*pixels(m,i+1))/3);  
            pos=pos+1;  
  
        elseif (bitget(pixels(m,i),1)==0) %&& ((pixels(m,i)==1 ||  
pixels(m,i)==255 ) || (pixels(m,i+1))==1 || pixels(m,i+1)==255)  
            databit (pos)=bitget (pixels (m,i+1),1);  
  
            mainpixel (m,i)= bitset (pixels (m,i),1);  
            mainpixel (m,i+1)= bitset (pixels (m,i+1),1);  
  
            pos=pos+1;  
  
        % elseif ((pixels(m,i)==1 || pixels(m,i)==255 ) ||  
        % (pixels(m,i+1))==1 || pixels(m,i+1)==255)  
        % databit (pos)=bitget (pixels (m,i+1),1);  
        % mainpixel (m,i)= bitset (pixels (m,i),1);  
        % mainpixel (m,i+1)= pixels (m,i+1);
```

```

%           pos=pos+1;
else

    mainpixel (m,i)= bitset (pixels (m,i),1);
    mainpixel (m,i+1)= pixels (m,i+1);

end

end

Reverse_bin2dec.m

%A=logical(databit-0);
%databit = char (A+48);

for i=1:7:numel(databit)

if (databit (i)==1) && (databit (i+1)==1) && (databit (i+2)==1) && (databit (i+3)==1) && (databit (i+4)==1) && (databit (i+5)==1) && (databit (i+6)==1)
    break
else

    if (i==numel(databit))
        break
    else
        data_obtained (i)= databit (i);
        data_obtained (i+1)= databit (i+1);
        data_obtained (i+2)= databit (i+2);
        data_obtained (i+3)=databit (i+3);
        data_obtained (i+4)=databit (i+4);
        data_obtained (i+5)=databit (i+5);
        data_obtained (i+6)=databit (i+6);
    end
end
end
end
A=logical(data_obtained-0);
maindata=char (A+48);

bit_segment_7 = reshape(maindata,7,[]);%bit_segment_7 =
reshape(maindata,[12,7]);
decimalvalue_of_bitsegments=bin2dec(bit_segment_7);
Text=char(decimalvalue_of_bitsegments')

Reverse_from_image.m

pixel10800=uint8 (magic(512)*0);
%img1=mainpixel;
%[rows1,cols1]=size(img1);%rows=131072
for i=1:131072
j=1;

```

```

if(i>0 && i<513)
    pixel10800(i)=mainpixel(i,j);

elseif (i>512 && i<2*512+1)
    pixel10800(i+512)=mainpixel(i,j);

elseif (i>2*512 && i<3*512+1)
    pixel10800(i+2*512)=mainpixel(i,j);

elseif (i>3*512 && i<4*512+1)
    pixel10800(i+3*512)=mainpixel(i,j);

elseif (i>4*512 && i<5*512+1)
    pixel10800(i+4*512)=mainpixel(i,j);

elseif (i>5*512 && i<6*512+1)
    pixel10800(i+5*512)=mainpixel(i,j);

elseif (i>6*512 && i<7*512+1)
    pixel10800(i+6*512)=mainpixel(i,j);

elseif (i>7*512 && i<8*512+1)
    pixel10800(i+7*512)=mainpixel(i,j);

elseif (i>8*512 && i<9*512+1)
    pixel10800(i+8*512)=mainpixel(i,j);

elseif (i>9*512 && i<10*512+1)
    pixel10800(i+9*512)=mainpixel(i,j);

elseif (i>10*512 && i<11*512+1)
    pixel10800(i+10*512)=mainpixel(i,j);

elseif (i>11*512 && i<12*512+1)
    pixel10800(i+11*512)=mainpixel(i,j);

elseif (i>12*512 && i<13*512+1)
    pixel10800(i+12*512)=mainpixel(i,j);

elseif (i>13*512 && i<14*512+1)
    pixel10800(i+13*512)=mainpixel(i,j);

elseif (i>14*512 && i<15*512+1)
    pixel10800(i+14*512)=mainpixel(i,j);

elseif (i>15*512 && i<16*512+1)
    pixel10800(i+15*512)=mainpixel(i,j);

elseif (i>16*512 && i<17*512+1)
    pixel10800(i+16*512)=mainpixel(i,j);

elseif (i>17*512 && i<18*512+1)
    pixel10800(i+17*512)=mainpixel(i,j);

```

```

elseif (i>18*512 && i<19*512+1)
pixel10800(i+18*512)=mainpixel(i,j);

elseif (i>19*512 && i<20*512+1)
pixel10800(i+19*512)=mainpixel(i,j);

elseif (i>20*512 && i<21*512+1)
pixel10800(i+20*512)=mainpixel(i,j);

elseif (i>21*512 && i<22*512+1)
pixel10800(i+21*512)=mainpixel(i,j);

elseif (i>22*512 && i<23*512+1)
pixel10800(i+22*512)=mainpixel(i,j);

elseif (i>23*512 && i<24*512+1)
pixel10800(i+23*512)=mainpixel(i,j);

elseif (i>24*512 && i<25*512+1)
pixel10800(i+24*512)=mainpixel(i,j);

elseif (i>25*512 && i<26*512+1)
pixel10800(i+25*512)=mainpixel(i,j);

elseif (i>26*512 && i<27*512+1)
pixel10800(i+26*512)=mainpixel(i,j);

elseif (i>27*512 && i<28*512+1)
pixel10800(i+27*512)=mainpixel(i,j);

elseif (i>28*512 && i<29*512+1)
pixel10800(i+28*512)=mainpixel(i,j);

elseif (i>29*512 && i<30*512+1)
pixel10800(i+29*512)=mainpixel(i,j);

elseif (i>30*512 && i<31*512+1)
pixel10800(i+30*512)=mainpixel(i,j);

elseif (i>31*512 && i<32*512+1)
pixel10800(i+31*512)=mainpixel(i,j);

elseif (i>32*512 && i<33*512+1)
pixel10800(i+32*512)=mainpixel(i,j);

elseif (i>33*512 && i<34*512+1)
pixel10800(i+33*512)=mainpixel(i,j);

elseif (i>34*512 && i<35*512+1)
pixel10800(i+34*512)=mainpixel(i,j);

elseif (i>35*512 && i<36*512+1)

```

```

pixel10800(i+35*512)=mainpixel(i,j);

elseif (i>36*512 && i<37*512+1)
pixel10800(i+36*512)=mainpixel(i,j);

elseif (i>37*512 && i<38*512+1)
pixel10800(i+37*512)=mainpixel(i,j);

elseif (i>38*512 && i<39*512+1)
pixel10800(i+38*512)=mainpixel(i,j);

elseif (i>39*512 && i<40*512+1)
pixel10800(i+39*512)=mainpixel(i,j);

elseif (i>40*512 && i<41*512+1)
pixel10800(i+40*512)=mainpixel(i,j);

elseif (i>41*512 && i<42*512+1)
pixel10800(i+41*512)=mainpixel(i,j);

elseif (i>42*512 && i<43*512+1)
pixel10800(i+42*512)=mainpixel(i,j);

elseif (i>43*512 && i<44*512+1)
pixel10800(i+43*512)=mainpixel(i,j);

elseif (i>44*512 && i<45*512+1)
pixel10800(i+44*512)=mainpixel(i,j);

elseif (i>45*512 && i<46*512+1)
pixel10800(i+45*512)=mainpixel(i,j);

elseif (i>46*512 && i<47*512+1)
pixel10800(i+46*512)=mainpixel(i,j);

elseif (i>47*512 && i<48*512+1)
pixel10800(i+47*512)=mainpixel(i,j);

elseif (i>48*512 && i<49*512+1)
pixel10800(i+48*512)=mainpixel(i,j);

elseif (i>49*512 && i<50*512+1)
pixel10800(i+49*512)=mainpixel(i,j);

elseif (i>50*512 && i<51*512+1)
pixel10800(i+50*512)=mainpixel(i,j);

elseif (i>51*512 && i<52*512+1)
pixel10800(i+51*512)=mainpixel(i,j);

elseif (i>52*512 && i<53*512+1)
pixel10800(i+52*512)=mainpixel(i,j);

```

```

elseif (i>53*512 && i<54*512+1)
pixel10800(i+53*512)=mainpixel(i,j);

elseif (i>54*512 && i<55*512+1)
pixel10800(i+54*512)=mainpixel(i,j);

elseif (i>55*512 && i<56*512+1)
pixel10800(i+55*512)=mainpixel(i,j);

elseif (i>56*512 && i<57*512+1)
pixel10800(i+56*512)=mainpixel(i,j);

elseif (i>57*512 && i<58*512+1)
pixel10800(i+57*512)=mainpixel(i,j);

elseif (i>58*512 && i<59*512+1)
pixel10800(i+58*512)=mainpixel(i,j);

elseif (i>59*512 && i<60*512+1)
pixel10800(i+59*512)=mainpixel(i,j);

elseif (i>60*512 && i<61*512+1)
pixel10800(i+60*512)=mainpixel(i,j);

elseif (i>61*512 && i<62*512+1)
pixel10800(i+61*512)=mainpixel(i,j);

elseif (i>62*512 && i<63*512+1)
pixel10800(i+62*512)=mainpixel(i,j);

elseif (i>63*512 && i<64*512+1)
pixel10800(i+63*512)=mainpixel(i,j);

elseif (i>64*512 && i<65*512+1)
pixel10800(i+64*512)=mainpixel(i,j);

elseif (i>65*512 && i<66*512+1)
pixel10800(i+65*512)=mainpixel(i,j);

elseif (i>66*512 && i<67*512+1)
pixel10800(i+66*512)=mainpixel(i,j);

elseif (i>67*512 && i<68*512+1)
pixel10800(i+67*512)=mainpixel(i,j);

elseif (i>68*512 && i<69*512+1)
pixel10800(i+68*512)=mainpixel(i,j);

elseif (i>69*512 && i<70*512+1)
pixel10800(i+69*512)=mainpixel(i,j);

elseif (i>70*512 && i<71*512+1)
pixel10800(i+70*512)=mainpixel(i,j);

```

```

elseif (i>71*512 && i<72*512+1)
pixel10800(i+71*512)=mainpixel(i,j);

elseif (i>72*512 && i<73*512+1)
pixel10800(i+72*512)=mainpixel(i,j);

elseif (i>73*512 && i<74*512+1)
pixel10800(i+73*512)=mainpixel(i,j);

elseif (i>74*512 && i<75*512+1)
pixel10800(i+74*512)=mainpixel(i,j);

elseif (i>75*512 && i<76*512+1)
pixel10800(i+75*512)=mainpixel(i,j);

elseif (i>76*512 && i<77*512+1)
pixel10800(i+76*512)=mainpixel(i,j);

elseif (i>77*512 && i<78*512+1)
pixel10800(i+77*512)=mainpixel(i,j);

elseif (i>78*512 && i<79*512+1)
pixel10800(i+78*512)=mainpixel(i,j);

elseif (i>79*512 && i<80*512+1)
pixel10800(i+79*512)=mainpixel(i,j);

elseif (i>80*512 && i<81*512+1)
pixel10800(i+80*512)=mainpixel(i,j);

elseif (i>81*512 && i<82*512+1)
pixel10800(i+81*512)=mainpixel(i,j);

elseif (i>82*512 && i<83*512+1)
pixel10800(i+82*512)=mainpixel(i,j);

elseif (i>83*512 && i<84*512+1)
pixel10800(i+83*512)=mainpixel(i,j);

elseif (i>84*512 && i<85*512+1)
pixel10800(i+84*512)=mainpixel(i,j);

elseif (i>85*512 && i<86*512+1)
pixel10800(i+85*512)=mainpixel(i,j);

elseif (i>86*512 && i<87*512+1)
pixel10800(i+86*512)=mainpixel(i,j);

elseif (i>87*512 && i<88*512+1)
pixel10800(i+87*512)=mainpixel(i,j);

elseif (i>88*512 && i<89*512+1)

```

```

pixel10800(i+88*512)=mainpixel(i,j);

elseif (i>89*512 && i<90*512+1)
pixel10800(i+89*512)=mainpixel(i,j);

elseif (i>90*512 && i<91*512+1)
pixel10800(i+90*512)=mainpixel(i,j);

elseif (i>91*512 && i<92*512+1)
pixel10800(i+91*512)=mainpixel(i,j);

elseif (i>92*512 && i<93*512+1)
pixel10800(i+92*512)=mainpixel(i,j);

elseif (i>93*512 && i<94*512+1)
pixel10800(i+93*512)=mainpixel(i,j);

elseif (i>94*512 && i<95*512+1)
pixel10800(i+94*512)=mainpixel(i,j);

elseif (i>95*512 && i<96*512+1)
pixel10800(i+95*512)=mainpixel(i,j);

elseif (i>96*512 && i<97*512+1)
pixel10800(i+96*512)=mainpixel(i,j);

elseif (i>97*512 && i<98*512+1)
pixel10800(i+97*512)=mainpixel(i,j);

elseif (i>98*512 && i<99*512+1)
pixel10800(i+98*512)=mainpixel(i,j);

elseif (i>99*512 && i<100*512+1)
pixel10800(i+99*512)=mainpixel(i,j);

elseif (i>100*512 && i<101*512+1)
pixel10800(i+100*512)=mainpixel(i,j);

elseif (i>101*512 && i<102*512+1)
pixel10800(i+101*512)=mainpixel(i,j);

elseif (i>102*512 && i<103*512+1)
pixel10800(i+102*512)=mainpixel(i,j);

elseif (i>103*512 && i<104*512+1)
pixel10800(i+103*512)=mainpixel(i,j);

elseif (i>104*512 && i<105*512+1)
pixel10800(i+104*512)=mainpixel(i,j);

elseif (i>105*512 && i<106*512+1)
pixel10800(i+105*512)=mainpixel(i,j);

```

```

elseif (i>106*512 && i<107*512+1)
pixel10800(i+106*512)=mainpixel(i,j);

elseif (i>107*512 && i<108*512+1)
pixel10800(i+107*512)=mainpixel(i,j);

elseif (i>108*512 && i<109*512+1)
pixel10800(i+108*512)=mainpixel(i,j);

elseif (i>109*512 && i<110*512+1)
pixel10800(i+109*512)=mainpixel(i,j);

elseif (i>110*512 && i<111*512+1)
pixel10800(i+110*512)=mainpixel(i,j);

elseif (i>111*512 && i<112*512+1)
pixel10800(i+111*512)=mainpixel(i,j);

elseif (i>112*512 && i<113*512+1)
pixel10800(i+112*512)=mainpixel(i,j);

elseif (i>113*512 && i<114*512+1)
pixel10800(i+113*512)=mainpixel(i,j);

elseif (i>114*512 && i<115*512+1)
pixel10800(i+114*512)=mainpixel(i,j);

elseif (i>115*512 && i<116*512+1)
pixel10800(i+115*512)=mainpixel(i,j);

elseif (i>116*512 && i<117*512+1)
pixel10800(i+116*512)=mainpixel(i,j);

elseif (i>117*512 && i<118*512+1)
pixel10800(i+117*512)=mainpixel(i,j);

elseif (i>118*512 && i<119*512+1)
pixel10800(i+118*512)=mainpixel(i,j);

elseif (i>119*512 && i<120*512+1)
pixel10800(i+119*512)=mainpixel(i,j);

elseif (i>120*512 && i<121*512+1)
pixel10800(i+120*512)=mainpixel(i,j);

elseif (i>121*512 && i<122*512+1)
pixel10800(i+121*512)=mainpixel(i,j);

elseif (i>122*512 && i<123*512+1)
pixel10800(i+122*512)=mainpixel(i,j);

elseif (i>123*512 && i<124*512+1)
pixel10800(i+123*512)=mainpixel(i,j);

```

```

elseif (i>124*512 && i<125*512+1)
pixel10800(i+124*512)=mainpixel(i,j);

elseif (i>125*512 && i<126*512+1)
pixel10800(i+125*512)=mainpixel(i,j);

elseif (i>126*512 && i<127*512+1)
pixel10800(i+126*512)=mainpixel(i,j);

elseif (i>127*512 && i<128*512+1)
pixel10800(i+127*512)=mainpixel(i,j);

elseif (i>128*512 && i<129*512+1)
pixel10800(i+128*512)=mainpixel(i,j);

elseif (i>129*512 && i<130*512+1)
pixel10800(i+129*512)=mainpixel(i,j);

elseif (i>130*512 && i<131*512+1)
pixel10800(i+130*512)=mainpixel(i,j);

elseif (i>131*512 && i<132*512+1)
pixel10800(i+131*512)=mainpixel(i,j);

elseif (i>132*512 && i<133*512+1)
pixel10800(i+132*512)=mainpixel(i,j);

elseif (i>133*512 && i<134*512+1)
pixel10800(i+133*512)=mainpixel(i,j);

elseif (i>134*512 && i<135*512+1)
pixel10800(i+134*512)=mainpixel(i,j);

elseif (i>135*512 && i<136*512+1)
pixel10800(i+135*512)=mainpixel(i,j);

elseif (i>136*512 && i<137*512+1)
pixel10800(i+136*512)=mainpixel(i,j);

elseif (i>137*512 && i<138*512+1)
pixel10800(i+137*512)=mainpixel(i,j);

elseif (i>138*512 && i<139*512+1)
pixel10800(i+138*512)=mainpixel(i,j);

elseif (i>139*512 && i<140*512+1)
pixel10800(i+139*512)=mainpixel(i,j);

elseif (i>140*512 && i<141*512+1)
pixel10800(i+140*512)=mainpixel(i,j);

elseif (i>141*512 && i<142*512+1)

```

```

pixel10800(i+141*512)=mainpixel(i,j);

elseif (i>142*512 && i<143*512+1)
pixel10800(i+142*512)=mainpixel(i,j);

elseif (i>143*512 && i<144*512+1)
pixel10800(i+143*512)=mainpixel(i,j);

elseif (i>144*512 && i<145*512+1)
pixel10800(i+144*512)=mainpixel(i,j);

elseif (i>145*512 && i<146*512+1)
pixel10800(i+145*512)=mainpixel(i,j);

elseif (i>146*512 && i<147*512+1)
pixel10800(i+146*512)=mainpixel(i,j);

elseif (i>147*512 && i<148*512+1)
pixel10800(i+147*512)=mainpixel(i,j);

elseif (i>148*512 && i<149*512+1)
pixel10800(i+148*512)=mainpixel(i,j);

elseif (i>149*512 && i<150*512+1)
pixel10800(i+149*512)=mainpixel(i,j);

elseif (i>150*512 && i<151*512+1)
pixel10800(i+150*512)=mainpixel(i,j);

elseif (i>151*512 && i<152*512+1)
pixel10800(i+151*512)=mainpixel(i,j);

elseif (i>152*512 && i<153*512+1)
pixel10800(i+152*512)=mainpixel(i,j);

elseif (i>153*512 && i<154*512+1)
pixel10800(i+153*512)=mainpixel(i,j);

elseif (i>154*512 && i<155*512+1)
pixel10800(i+154*512)=mainpixel(i,j);

elseif (i>155*512 && i<156*512+1)
pixel10800(i+155*512)=mainpixel(i,j);

elseif (i>156*512 && i<157*512+1)
pixel10800(i+156*512)=mainpixel(i,j);

elseif (i>157*512 && i<158*512+1)
pixel10800(i+157*512)=mainpixel(i,j);

elseif (i>158*512 && i<159*512+1)
pixel10800(i+158*512)=mainpixel(i,j);

```

```

elseif (i>159*512 && i<160*512+1)
pixel10800(i+159*512)=mainpixel(i,j);

elseif (i>160*512 && i<161*512+1)
pixel10800(i+160*512)=mainpixel(i,j);

elseif (i>161*512 && i<162*512+1)
pixel10800(i+161*512)=mainpixel(i,j);

elseif (i>162*512 && i<163*512+1)
pixel10800(i+162*512)=mainpixel(i,j);

elseif (i>163*512 && i<164*512+1)
pixel10800(i+163*512)=mainpixel(i,j);

elseif (i>164*512 && i<165*512+1)
pixel10800(i+164*512)=mainpixel(i,j);

elseif (i>165*512 && i<166*512+1)
pixel10800(i+165*512)=mainpixel(i,j);

elseif (i>166*512 && i<167*512+1)
pixel10800(i+166*512)=mainpixel(i,j);

elseif (i>167*512 && i<168*512+1)
pixel10800(i+167*512)=mainpixel(i,j);

elseif (i>168*512 && i<169*512+1)
pixel10800(i+168*512)=mainpixel(i,j);

elseif (i>169*512 && i<170*512+1)
pixel10800(i+169*512)=mainpixel(i,j);

elseif (i>170*512 && i<171*512+1)
pixel10800(i+170*512)=mainpixel(i,j);

elseif (i>171*512 && i<172*512+1)
pixel10800(i+171*512)=mainpixel(i,j);

elseif (i>172*512 && i<173*512+1)
pixel10800(i+172*512)=mainpixel(i,j);

elseif (i>173*512 && i<174*512+1)
pixel10800(i+173*512)=mainpixel(i,j);

elseif (i>174*512 && i<175*512+1)
pixel10800(i+174*512)=mainpixel(i,j);

elseif (i>175*512 && i<176*512+1)
pixel10800(i+175*512)=mainpixel(i,j);

elseif (i>176*512 && i<177*512+1)
pixel10800(i+176*512)=mainpixel(i,j);

```

```

elseif (i>177*512 && i<178*512+1)
pixel10800(i+177*512)=mainpixel(i,j);

elseif (i>178*512 && i<179*512+1)
pixel10800(i+178*512)=mainpixel(i,j);

elseif (i>179*512 && i<180*512+1)
pixel10800(i+179*512)=mainpixel(i,j);

elseif (i>180*512 && i<181*512+1)
pixel10800(i+180*512)=mainpixel(i,j);

elseif (i>181*512 && i<182*512+1)
pixel10800(i+181*512)=mainpixel(i,j);

elseif (i>182*512 && i<183*512+1)
pixel10800(i+182*512)=mainpixel(i,j);

elseif (i>183*512 && i<184*512+1)
pixel10800(i+183*512)=mainpixel(i,j);

elseif (i>184*512 && i<185*512+1)
pixel10800(i+184*512)=mainpixel(i,j);

elseif (i>185*512 && i<186*512+1)
pixel10800(i+185*512)=mainpixel(i,j);

elseif (i>186*512 && i<187*512+1)
pixel10800(i+186*512)=mainpixel(i,j);

elseif (i>187*512 && i<188*512+1)
pixel10800(i+187*512)=mainpixel(i,j);

elseif (i>188*512 && i<189*512+1)
pixel10800(i+188*512)=mainpixel(i,j);

elseif (i>189*512 && i<190*512+1)
pixel10800(i+189*512)=mainpixel(i,j);

elseif (i>190*512 && i<191*512+1)
pixel10800(i+190*512)=mainpixel(i,j);

elseif (i>191*512 && i<192*512+1)
pixel10800(i+191*512)=mainpixel(i,j);

elseif (i>192*512 && i<193*512+1)
pixel10800(i+192*512)=mainpixel(i,j);

elseif (i>193*512 && i<194*512+1)
pixel10800(i+193*512)=mainpixel(i,j);

elseif (i>194*512 && i<195*512+1)

```

```

pixel10800(i+194*512)=mainpixel(i,j);

elseif (i>195*512 && i<196*512+1)
pixel10800(i+195*512)=mainpixel(i,j);

elseif (i>196*512 && i<197*512+1)
pixel10800(i+196*512)=mainpixel(i,j);

elseif (i>197*512 && i<198*512+1)
pixel10800(i+197*512)=mainpixel(i,j);

elseif (i>198*512 && i<199*512+1)
pixel10800(i+198*512)=mainpixel(i,j);

elseif (i>199*512 && i<200*512+1)
pixel10800(i+199*512)=mainpixel(i,j);

elseif (i>200*512 && i<201*512+1)
pixel10800(i+200*512)=mainpixel(i,j);

elseif (i>201*512 && i<202*512+1)
pixel10800(i+201*512)=mainpixel(i,j);

elseif (i>202*512 && i<203*512+1)
pixel10800(i+202*512)=mainpixel(i,j);

elseif (i>203*512 && i<204*512+1)
pixel10800(i+203*512)=mainpixel(i,j);

elseif (i>204*512 && i<205*512+1)
pixel10800(i+204*512)=mainpixel(i,j);

elseif (i>205*512 && i<206*512+1)
pixel10800(i+205*512)=mainpixel(i,j);

elseif (i>206*512 && i<207*512+1)
pixel10800(i+206*512)=mainpixel(i,j);

elseif (i>207*512 && i<208*512+1)
pixel10800(i+207*512)=mainpixel(i,j);

elseif (i>208*512 && i<209*512+1)
pixel10800(i+208*512)=mainpixel(i,j);

elseif (i>209*512 && i<210*512+1)
pixel10800(i+209*512)=mainpixel(i,j);

elseif (i>210*512 && i<211*512+1)
pixel10800(i+210*512)=mainpixel(i,j);

elseif (i>211*512 && i<212*512+1)
pixel10800(i+211*512)=mainpixel(i,j);

```

```

elseif (i>212*512 && i<213*512+1)
pixel10800(i+212*512)=mainpixel(i,j);

elseif (i>213*512 && i<214*512+1)
pixel10800(i+213*512)=mainpixel(i,j);

elseif (i>214*512 && i<215*512+1)
pixel10800(i+214*512)=mainpixel(i,j);

elseif (i>215*512 && i<216*512+1)
pixel10800(i+215*512)=mainpixel(i,j);

elseif (i>216*512 && i<217*512+1)
pixel10800(i+216*512)=mainpixel(i,j);

elseif (i>217*512 && i<218*512+1)
pixel10800(i+217*512)=mainpixel(i,j);

elseif (i>218*512 && i<219*512+1)
pixel10800(i+218*512)=mainpixel(i,j);

elseif (i>219*512 && i<220*512+1)
pixel10800(i+219*512)=mainpixel(i,j);

elseif (i>220*512 && i<221*512+1)
pixel10800(i+220*512)=mainpixel(i,j);

elseif (i>221*512 && i<222*512+1)
pixel10800(i+221*512)=mainpixel(i,j);

elseif (i>222*512 && i<223*512+1)
pixel10800(i+222*512)=mainpixel(i,j);

elseif (i>223*512 && i<224*512+1)
pixel10800(i+223*512)=mainpixel(i,j);

elseif (i>224*512 && i<225*512+1)
pixel10800(i+224*512)=mainpixel(i,j);

elseif (i>225*512 && i<226*512+1)
pixel10800(i+225*512)=mainpixel(i,j);

elseif (i>226*512 && i<227*512+1)
pixel10800(i+226*512)=mainpixel(i,j);

elseif (i>227*512 && i<228*512+1)
pixel10800(i+227*512)=mainpixel(i,j);

elseif (i>228*512 && i<229*512+1)
pixel10800(i+228*512)=mainpixel(i,j);

elseif (i>229*512 && i<230*512+1)
pixel10800(i+229*512)=mainpixel(i,j);

```

```

elseif (i>230*512 && i<231*512+1)
pixel10800(i+230*512)=mainpixel(i,j);

elseif (i>231*512 && i<232*512+1)
pixel10800(i+231*512)=mainpixel(i,j);

elseif (i>232*512 && i<233*512+1)
pixel10800(i+232*512)=mainpixel(i,j);

elseif (i>233*512 && i<234*512+1)
pixel10800(i+233*512)=mainpixel(i,j);

elseif (i>234*512 && i<235*512+1)
pixel10800(i+234*512)=mainpixel(i,j);

elseif (i>235*512 && i<236*512+1)
pixel10800(i+235*512)=mainpixel(i,j);

elseif (i>236*512 && i<237*512+1)
pixel10800(i+236*512)=mainpixel(i,j);

elseif (i>237*512 && i<238*512+1)
pixel10800(i+237*512)=mainpixel(i,j);

elseif (i>238*512 && i<239*512+1)
pixel10800(i+238*512)=mainpixel(i,j);

elseif (i>239*512 && i<240*512+1)
pixel10800(i+239*512)=mainpixel(i,j);

elseif (i>240*512 && i<241*512+1)
pixel10800(i+240*512)=mainpixel(i,j);

elseif (i>241*512 && i<242*512+1)
pixel10800(i+241*512)=mainpixel(i,j);

elseif (i>242*512 && i<243*512+1)
pixel10800(i+242*512)=mainpixel(i,j);

elseif (i>243*512 && i<244*512+1)
pixel10800(i+243*512)=mainpixel(i,j);

elseif (i>244*512 && i<245*512+1)
pixel10800(i+244*512)=mainpixel(i,j);

elseif (i>245*512 && i<246*512+1)
pixel10800(i+245*512)=mainpixel(i,j);

elseif (i>246*512 && i<247*512+1)
pixel10800(i+246*512)=mainpixel(i,j);

elseif (i>247*512 && i<248*512+1)

```

```

pixel10800(i+247*512)=mainpixel(i,j);

elseif (i>248*512 && i<249*512+1)
pixel10800(i+248*512)=mainpixel(i,j);

elseif (i>249*512 && i<250*512+1)
pixel10800(i+249*512)=mainpixel(i,j);

elseif (i>250*512 && i<251*512+1)
pixel10800(i+250*512)=mainpixel(i,j);

elseif (i>251*512 && i<252*512+1)
pixel10800(i+251*512)=mainpixel(i,j);

elseif (i>252*512 && i<253*512+1)
pixel10800(i+252*512)=mainpixel(i,j);

elseif (i>253*512 && i<254*512+1)
pixel10800(i+253*512)=mainpixel(i,j);

elseif (i>254*512 && i<255*512+1)
pixel10800(i+254*512)=mainpixel(i,j);

elseif (i>255*512 && i<256*512+1)
pixel10800(i+255*512)=mainpixel(i,j);

end
end

```

```

for i=1:131072
j=2;

if(i>0 && i<1*512+1)
pixel10800(i+1*512)=mainpixel(i,j);

elseif (i>1*512 && i<2*512+1)
pixel10800(i+2*512)=mainpixel(i,j);

elseif (i>2*512 && i<3*512+1)
pixel10800(i+3*512)=mainpixel(i,j);

elseif (i>3*512 && i<4*512+1)
pixel10800(i+4*512)=mainpixel(i,j);

elseif (i>4*512 && i<5*512+1)
pixel10800(i+5*512)=mainpixel(i,j);

elseif (i>5*512 && i<6*512+1)
pixel10800(i+6*512)=mainpixel(i,j);

elseif (i>6*512 && i<7*512+1)

```

```

pixel10800(i+7*512)=mainpixel(i,j);

elseif (i>7*512 && i<8*512+1)
pixel10800(i+8*512)=mainpixel(i,j);

elseif (i>8*512 && i<9*512+1)
pixel10800(i+9*512)=mainpixel(i,j);

elseif (i>9*512 && i<10*512+1)
pixel10800(i+10*512)=mainpixel(i,j);

elseif (i>10*512 && i<11*512+1)
pixel10800(i+11*512)=mainpixel(i,j);

elseif (i>11*512 && i<12*512+1)
pixel10800(i+12*512)=mainpixel(i,j);

elseif (i>12*512 && i<13*512+1)
pixel10800(i+13*512)=mainpixel(i,j);

elseif (i>13*512 && i<14*512+1)
pixel10800(i+14*512)=mainpixel(i,j);

elseif (i>14*512 && i<15*512+1)
pixel10800(i+15*512)=mainpixel(i,j);

elseif (i>15*512 && i<16*512+1)
pixel10800(i+16*512)=mainpixel(i,j);

elseif (i>16*512 && i<17*512+1)
pixel10800(i+17*512)=mainpixel(i,j);

elseif (i>17*512 && i<18*512+1)
pixel10800(i+18*512)=mainpixel(i,j);

elseif (i>18*512 && i<19*512+1)
pixel10800(i+19*512)=mainpixel(i,j);

elseif (i>19*512 && i<20*512+1)
pixel10800(i+20*512)=mainpixel(i,j);

elseif (i>20*512 && i<21*512+1)
pixel10800(i+21*512)=mainpixel(i,j);

elseif (i>21*512 && i<22*512+1)
pixel10800(i+22*512)=mainpixel(i,j);

elseif (i>22*512 && i<23*512+1)
pixel10800(i+23*512)=mainpixel(i,j);

elseif (i>23*512 && i<24*512+1)
pixel10800(i+24*512)=mainpixel(i,j);

```

```

elseif (i>24*512 && i<25*512+1)
pixel10800(i+25*512)=mainpixel(i,j);

elseif (i>25*512 && i<26*512+1)
pixel10800(i+26*512)=mainpixel(i,j);

elseif (i>26*512 && i<27*512+1)
pixel10800(i+27*512)=mainpixel(i,j);

elseif (i>27*512 && i<28*512+1)
pixel10800(i+28*512)=mainpixel(i,j);

elseif (i>28*512 && i<29*512+1)
pixel10800(i+29*512)=mainpixel(i,j);

elseif (i>29*512 && i<30*512+1)
pixel10800(i+30*512)=mainpixel(i,j);

elseif (i>30*512 && i<31*512+1)
pixel10800(i+31*512)=mainpixel(i,j);

elseif (i>31*512 && i<32*512+1)
pixel10800(i+32*512)=mainpixel(i,j);

elseif (i>32*512 && i<33*512+1)
pixel10800(i+33*512)=mainpixel(i,j);

elseif (i>33*512 && i<34*512+1)
pixel10800(i+34*512)=mainpixel(i,j);

elseif (i>34*512 && i<35*512+1)
pixel10800(i+35*512)=mainpixel(i,j);

elseif (i>35*512 && i<36*512+1)
pixel10800(i+36*512)=mainpixel(i,j);

elseif (i>36*512 && i<37*512+1)
pixel10800(i+37*512)=mainpixel(i,j);

elseif (i>37*512 && i<38*512+1)
pixel10800(i+38*512)=mainpixel(i,j);

elseif (i>38*512 && i<39*512+1)
pixel10800(i+39*512)=mainpixel(i,j);

elseif (i>39*512 && i<40*512+1)
pixel10800(i+40*512)=mainpixel(i,j);

elseif (i>40*512 && i<41*512+1)
pixel10800(i+41*512)=mainpixel(i,j);

elseif (i>41*512 && i<42*512+1)
pixel10800(i+42*512)=mainpixel(i,j);

```

```

elseif (i>42*512 && i<43*512+1)
pixel10800(i+43*512)=mainpixel(i,j);

elseif (i>43*512 && i<44*512+1)
pixel10800(i+44*512)=mainpixel(i,j);

elseif (i>44*512 && i<45*512+1)
pixel10800(i+45*512)=mainpixel(i,j);

elseif (i>45*512 && i<46*512+1)
pixel10800(i+46*512)=mainpixel(i,j);

elseif (i>46*512 && i<47*512+1)
pixel10800(i+47*512)=mainpixel(i,j);

elseif (i>47*512 && i<48*512+1)
pixel10800(i+48*512)=mainpixel(i,j);

elseif (i>48*512 && i<49*512+1)
pixel10800(i+49*512)=mainpixel(i,j);

elseif (i>49*512 && i<50*512+1)
pixel10800(i+50*512)=mainpixel(i,j);

elseif (i>50*512 && i<51*512+1)
pixel10800(i+51*512)=mainpixel(i,j);

elseif (i>51*512 && i<52*512+1)
pixel10800(i+52*512)=mainpixel(i,j);

elseif (i>52*512 && i<53*512+1)
pixel10800(i+53*512)=mainpixel(i,j);

elseif (i>53*512 && i<54*512+1)
pixel10800(i+54*512)=mainpixel(i,j);

elseif (i>54*512 && i<55*512+1)
pixel10800(i+55*512)=mainpixel(i,j);

elseif (i>55*512 && i<56*512+1)
pixel10800(i+56*512)=mainpixel(i,j);

elseif (i>56*512 && i<57*512+1)
pixel10800(i+57*512)=mainpixel(i,j);

elseif (i>57*512 && i<58*512+1)
pixel10800(i+58*512)=mainpixel(i,j);

elseif (i>58*512 && i<59*512+1)
pixel10800(i+59*512)=mainpixel(i,j);

elseif (i>59*512 && i<60*512+1)

```

```

pixel10800(i+60*512)=mainpixel(i,j);

elseif (i>60*512 && i<61*512+1)
pixel10800(i+61*512)=mainpixel(i,j);

elseif (i>61*512 && i<62*512+1)
pixel10800(i+62*512)=mainpixel(i,j);

elseif (i>62*512 && i<63*512+1)
pixel10800(i+63*512)=mainpixel(i,j);

elseif (i>63*512 && i<64*512+1)
pixel10800(i+64*512)=mainpixel(i,j);

elseif (i>64*512 && i<65*512+1)
pixel10800(i+65*512)=mainpixel(i,j);

elseif (i>65*512 && i<66*512+1)
pixel10800(i+66*512)=mainpixel(i,j);

elseif (i>66*512 && i<67*512+1)
pixel10800(i+67*512)=mainpixel(i,j);

elseif (i>67*512 && i<68*512+1)
pixel10800(i+68*512)=mainpixel(i,j);

elseif (i>68*512 && i<69*512+1)
pixel10800(i+69*512)=mainpixel(i,j);

elseif (i>69*512 && i<70*512+1)
pixel10800(i+70*512)=mainpixel(i,j);

elseif (i>70*512 && i<71*512+1)
pixel10800(i+71*512)=mainpixel(i,j);

elseif (i>71*512 && i<72*512+1)
pixel10800(i+72*512)=mainpixel(i,j);

elseif (i>72*512 && i<73*512+1)
pixel10800(i+73*512)=mainpixel(i,j);

elseif (i>73*512 && i<74*512+1)
pixel10800(i+74*512)=mainpixel(i,j);

elseif (i>74*512 && i<75*512+1)
pixel10800(i+75*512)=mainpixel(i,j);

elseif (i>75*512 && i<76*512+1)
pixel10800(i+76*512)=mainpixel(i,j);

elseif (i>76*512 && i<77*512+1)
pixel10800(i+77*512)=mainpixel(i,j);

```

```

elseif (i>77*512 && i<78*512+1)
pixel10800(i+78*512)=mainpixel(i,j);

elseif (i>78*512 && i<79*512+1)
pixel10800(i+79*512)=mainpixel(i,j);

elseif (i>79*512 && i<80*512+1)
pixel10800(i+80*512)=mainpixel(i,j);

elseif (i>80*512 && i<81*512+1)
pixel10800(i+81*512)=mainpixel(i,j);

elseif (i>81*512 && i<82*512+1)
pixel10800(i+82*512)=mainpixel(i,j);

elseif (i>82*512 && i<83*512+1)
pixel10800(i+83*512)=mainpixel(i,j);

elseif (i>83*512 && i<84*512+1)
pixel10800(i+84*512)=mainpixel(i,j);

elseif (i>84*512 && i<85*512+1)
pixel10800(i+85*512)=mainpixel(i,j);

elseif (i>85*512 && i<86*512+1)
pixel10800(i+86*512)=mainpixel(i,j);

elseif (i>86*512 && i<87*512+1)
pixel10800(i+87*512)=mainpixel(i,j);

elseif (i>87*512 && i<88*512+1)
pixel10800(i+88*512)=mainpixel(i,j);

elseif (i>88*512 && i<89*512+1)
pixel10800(i+89*512)=mainpixel(i,j);

elseif (i>89*512 && i<90*512+1)
pixel10800(i+90*512)=mainpixel(i,j);

elseif (i>90*512 && i<91*512+1)
pixel10800(i+91*512)=mainpixel(i,j);

elseif (i>91*512 && i<92*512+1)
pixel10800(i+92*512)=mainpixel(i,j);

elseif (i>92*512 && i<93*512+1)
pixel10800(i+93*512)=mainpixel(i,j);

elseif (i>93*512 && i<94*512+1)
pixel10800(i+94*512)=mainpixel(i,j);

elseif (i>94*512 && i<95*512+1)
pixel10800(i+95*512)=mainpixel(i,j);

```

```

elseif (i>95*512 && i<96*512+1)
pixel10800(i+96*512)=mainpixel(i,j);

elseif (i>96*512 && i<97*512+1)
pixel10800(i+97*512)=mainpixel(i,j);

elseif (i>97*512 && i<98*512+1)
pixel10800(i+98*512)=mainpixel(i,j);

elseif (i>98*512 && i<99*512+1)
pixel10800(i+99*512)=mainpixel(i,j);

elseif (i>99*512 && i<100*512+1)
pixel10800(i+100*512)=mainpixel(i,j);

elseif (i>100*512 && i<101*512+1)
pixel10800(i+101*512)=mainpixel(i,j);

elseif (i>101*512 && i<102*512+1)
pixel10800(i+102*512)=mainpixel(i,j);

elseif (i>102*512 && i<103*512+1)
pixel10800(i+103*512)=mainpixel(i,j);

elseif (i>103*512 && i<104*512+1)
pixel10800(i+104*512)=mainpixel(i,j);

elseif (i>104*512 && i<105*512+1)
pixel10800(i+105*512)=mainpixel(i,j);

elseif (i>105*512 && i<106*512+1)
pixel10800(i+106*512)=mainpixel(i,j);

elseif (i>106*512 && i<107*512+1)
pixel10800(i+107*512)=mainpixel(i,j);

elseif (i>107*512 && i<108*512+1)
pixel10800(i+108*512)=mainpixel(i,j);

elseif (i>108*512 && i<109*512+1)
pixel10800(i+109*512)=mainpixel(i,j);

elseif (i>109*512 && i<110*512+1)
pixel10800(i+110*512)=mainpixel(i,j);

elseif (i>110*512 && i<111*512+1)
pixel10800(i+111*512)=mainpixel(i,j);

elseif (i>111*512 && i<112*512+1)
pixel10800(i+112*512)=mainpixel(i,j);

```

```

elseif (i>112*512 && i<113*512+1)
pixel10800(i+113*512)=mainpixel(i,j);

elseif (i>113*512 && i<114*512+1)
pixel10800(i+114*512)=mainpixel(i,j);

elseif (i>114*512 && i<115*512+1)
pixel10800(i+115*512)=mainpixel(i,j);

elseif (i>115*512 && i<116*512+1)
pixel10800(i+116*512)=mainpixel(i,j);

elseif (i>116*512 && i<117*512+1)
pixel10800(i+117*512)=mainpixel(i,j);

elseif (i>117*512 && i<118*512+1)
pixel10800(i+118*512)=mainpixel(i,j);

elseif (i>118*512 && i<119*512+1)
pixel10800(i+119*512)=mainpixel(i,j);

elseif (i>119*512 && i<120*512+1)
pixel10800(i+120*512)=mainpixel(i,j);

elseif (i>120*512 && i<121*512+1)
pixel10800(i+121*512)=mainpixel(i,j);

elseif (i>121*512 && i<122*512+1)
pixel10800(i+122*512)=mainpixel(i,j);

elseif (i>122*512 && i<123*512+1)
pixel10800(i+123*512)=mainpixel(i,j);

elseif (i>123*512 && i<124*512+1)
pixel10800(i+124*512)=mainpixel(i,j);

elseif (i>124*512 && i<125*512+1)
pixel10800(i+125*512)=mainpixel(i,j);

elseif (i>125*512 && i<126*512+1)
pixel10800(i+126*512)=mainpixel(i,j);

elseif (i>126*512 && i<127*512+1)
pixel10800(i+127*512)=mainpixel(i,j);

elseif (i>127*512 && i<128*512+1)
pixel10800(i+128*512)=mainpixel(i,j);

elseif (i>128*512 && i<129*512+1)
pixel10800(i+129*512)=mainpixel(i,j);

elseif (i>129*512 && i<130*512+1)
pixel10800(i+130*512)=mainpixel(i,j);

```

```

elseif (i>130*512 && i<131*512+1)
pixel10800(i+131*512)=mainpixel(i,j);

elseif (i>131*512 && i<132*512+1)
pixel10800(i+132*512)=mainpixel(i,j);

elseif (i>132*512 && i<133*512+1)
pixel10800(i+133*512)=mainpixel(i,j);

elseif (i>133*512 && i<134*512+1)
pixel10800(i+134*512)=mainpixel(i,j);

elseif (i>134*512 && i<135*512+1)
pixel10800(i+135*512)=mainpixel(i,j);

elseif (i>135*512 && i<136*512+1)
pixel10800(i+136*512)=mainpixel(i,j);

elseif (i>136*512 && i<137*512+1)
pixel10800(i+137*512)=mainpixel(i,j);

elseif (i>137*512 && i<138*512+1)
pixel10800(i+138*512)=mainpixel(i,j);

elseif (i>138*512 && i<139*512+1)
pixel10800(i+139*512)=mainpixel(i,j);

elseif (i>139*512 && i<140*512+1)
pixel10800(i+140*512)=mainpixel(i,j);

elseif (i>140*512 && i<141*512+1)
pixel10800(i+141*512)=mainpixel(i,j);

elseif (i>141*512 && i<142*512+1)
pixel10800(i+142*512)=mainpixel(i,j);

elseif (i>142*512 && i<143*512+1)
pixel10800(i+143*512)=mainpixel(i,j);

elseif (i>143*512 && i<144*512+1)
pixel10800(i+144*512)=mainpixel(i,j);

elseif (i>144*512 && i<145*512+1)
pixel10800(i+145*512)=mainpixel(i,j);

elseif (i>145*512 && i<146*512+1)
pixel10800(i+146*512)=mainpixel(i,j);

elseif (i>146*512 && i<147*512+1)
pixel10800(i+147*512)=mainpixel(i,j);

elseif (i>147*512 && i<148*512+1)

```

```

pixel10800(i+148*512)=mainpixel(i,j);

elseif (i>148*512 && i<149*512+1)
pixel10800(i+149*512)=mainpixel(i,j);

elseif (i>149*512 && i<150*512+1)
pixel10800(i+150*512)=mainpixel(i,j);

elseif (i>150*512 && i<151*512+1)
pixel10800(i+151*512)=mainpixel(i,j);

elseif (i>151*512 && i<152*512+1)
pixel10800(i+152*512)=mainpixel(i,j);

elseif (i>152*512 && i<153*512+1)
pixel10800(i+153*512)=mainpixel(i,j);

elseif (i>153*512 && i<154*512+1)
pixel10800(i+154*512)=mainpixel(i,j);

elseif (i>154*512 && i<155*512+1)
pixel10800(i+155*512)=mainpixel(i,j);

elseif (i>155*512 && i<156*512+1)
pixel10800(i+156*512)=mainpixel(i,j);

elseif (i>156*512 && i<157*512+1)
pixel10800(i+157*512)=mainpixel(i,j);

elseif (i>157*512 && i<158*512+1)
pixel10800(i+158*512)=mainpixel(i,j);

elseif (i>158*512 && i<159*512+1)
pixel10800(i+159*512)=mainpixel(i,j);

elseif (i>159*512 && i<160*512+1)
pixel10800(i+160*512)=mainpixel(i,j);

elseif (i>160*512 && i<161*512+1)
pixel10800(i+161*512)=mainpixel(i,j);

elseif (i>161*512 && i<162*512+1)
pixel10800(i+162*512)=mainpixel(i,j);

elseif (i>162*512 && i<163*512+1)
pixel10800(i+163*512)=mainpixel(i,j);

elseif (i>163*512 && i<164*512+1)
pixel10800(i+164*512)=mainpixel(i,j);

elseif (i>164*512 && i<165*512+1)
pixel10800(i+165*512)=mainpixel(i,j);

```

```

elseif (i>165*512 && i<166*512+1)
pixel10800(i+166*512)=mainpixel(i,j);

elseif (i>166*512 && i<167*512+1)
pixel10800(i+167*512)=mainpixel(i,j);

elseif (i>167*512 && i<168*512+1)
pixel10800(i+168*512)=mainpixel(i,j);

elseif (i>168*512 && i<169*512+1)
pixel10800(i+169*512)=mainpixel(i,j);

elseif (i>169*512 && i<170*512+1)
pixel10800(i+170*512)=mainpixel(i,j);

elseif (i>170*512 && i<171*512+1)
pixel10800(i+171*512)=mainpixel(i,j);

elseif (i>171*512 && i<172*512+1)
pixel10800(i+172*512)=mainpixel(i,j);

elseif (i>172*512 && i<173*512+1)
pixel10800(i+173*512)=mainpixel(i,j);

elseif (i>173*512 && i<174*512+1)
pixel10800(i+174*512)=mainpixel(i,j);

elseif (i>174*512 && i<175*512+1)
pixel10800(i+175*512)=mainpixel(i,j);

elseif (i>175*512 && i<176*512+1)
pixel10800(i+176*512)=mainpixel(i,j);

elseif (i>176*512 && i<177*512+1)
pixel10800(i+177*512)=mainpixel(i,j);

elseif (i>177*512 && i<178*512+1)
pixel10800(i+178*512)=mainpixel(i,j);

elseif (i>178*512 && i<179*512+1)
pixel10800(i+179*512)=mainpixel(i,j);

elseif (i>179*512 && i<180*512+1)
pixel10800(i+180*512)=mainpixel(i,j);

elseif (i>180*512 && i<181*512+1)
pixel10800(i+181*512)=mainpixel(i,j);

elseif (i>181*512 && i<182*512+1)
pixel10800(i+182*512)=mainpixel(i,j);

elseif (i>182*512 && i<183*512+1)
pixel10800(i+183*512)=mainpixel(i,j);

```

```

elseif (i>183*512 && i<184*512+1)
pixel10800(i+184*512)=mainpixel(i,j);

elseif (i>184*512 && i<185*512+1)
pixel10800(i+185*512)=mainpixel(i,j);

elseif (i>185*512 && i<186*512+1)
pixel10800(i+186*512)=mainpixel(i,j);

elseif (i>186*512 && i<187*512+1)
pixel10800(i+187*512)=mainpixel(i,j);

elseif (i>187*512 && i<188*512+1)
pixel10800(i+188*512)=mainpixel(i,j);

elseif (i>188*512 && i<189*512+1)
pixel10800(i+189*512)=mainpixel(i,j);

elseif (i>189*512 && i<190*512+1)
pixel10800(i+190*512)=mainpixel(i,j);

elseif (i>190*512 && i<191*512+1)
pixel10800(i+191*512)=mainpixel(i,j);

elseif (i>191*512 && i<192*512+1)
pixel10800(i+192*512)=mainpixel(i,j);

elseif (i>192*512 && i<193*512+1)
pixel10800(i+193*512)=mainpixel(i,j);

elseif (i>193*512 && i<194*512+1)
pixel10800(i+194*512)=mainpixel(i,j);

elseif (i>194*512 && i<195*512+1)
pixel10800(i+195*512)=mainpixel(i,j);

elseif (i>195*512 && i<196*512+1)
pixel10800(i+196*512)=mainpixel(i,j);

elseif (i>196*512 && i<197*512+1)
pixel10800(i+197*512)=mainpixel(i,j);

elseif (i>197*512 && i<198*512+1)
pixel10800(i+198*512)=mainpixel(i,j);

elseif (i>198*512 && i<199*512+1)
pixel10800(i+199*512)=mainpixel(i,j);

elseif (i>199*512 && i<200*512+1)
pixel10800(i+200*512)=mainpixel(i,j);

elseif (i>200*512 && i<201*512+1)

```

```

pixel10800(i+201*512)=mainpixel(i,j);

elseif (i>201*512 && i<202*512+1)
pixel10800(i+202*512)=mainpixel(i,j);

elseif (i>202*512 && i<203*512+1)
pixel10800(i+203*512)=mainpixel(i,j);

elseif (i>203*512 && i<204*512+1)
pixel10800(i+204*512)=mainpixel(i,j);

elseif (i>204*512 && i<205*512+1)
pixel10800(i+205*512)=mainpixel(i,j);

elseif (i>205*512 && i<206*512+1)
pixel10800(i+206*512)=mainpixel(i,j);

elseif (i>206*512 && i<207*512+1)
pixel10800(i+207*512)=mainpixel(i,j);

elseif (i>207*512 && i<208*512+1)
pixel10800(i+208*512)=mainpixel(i,j);

elseif (i>208*512 && i<209*512+1)
pixel10800(i+209*512)=mainpixel(i,j);

elseif (i>209*512 && i<210*512+1)
pixel10800(i+210*512)=mainpixel(i,j);

elseif (i>210*512 && i<211*512+1)
pixel10800(i+211*512)=mainpixel(i,j);

elseif (i>211*512 && i<212*512+1)
pixel10800(i+212*512)=mainpixel(i,j);

elseif (i>212*512 && i<213*512+1)
pixel10800(i+213*512)=mainpixel(i,j);

elseif (i>213*512 && i<214*512+1)
pixel10800(i+214*512)=mainpixel(i,j);

elseif (i>214*512 && i<215*512+1)
pixel10800(i+215*512)=mainpixel(i,j);

elseif (i>215*512 && i<216*512+1)
pixel10800(i+216*512)=mainpixel(i,j);

elseif (i>216*512 && i<217*512+1)
pixel10800(i+217*512)=mainpixel(i,j);

```

```

elseif (i>217*512 && i<218*512+1)
pixel10800(i+218*512)=mainpixel(i,j);

elseif (i>218*512 && i<219*512+1)
pixel10800(i+219*512)=mainpixel(i,j);

elseif (i>219*512 && i<220*512+1)
pixel10800(i+220*512)=mainpixel(i,j);

elseif (i>220*512 && i<221*512+1)
pixel10800(i+221*512)=mainpixel(i,j);

elseif (i>221*512 && i<222*512+1)
pixel10800(i+222*512)=mainpixel(i,j);

elseif (i>222*512 && i<223*512+1)
pixel10800(i+223*512)=mainpixel(i,j);

elseif (i>223*512 && i<224*512+1)
pixel10800(i+224*512)=mainpixel(i,j);

elseif (i>224*512 && i<225*512+1)
pixel10800(i+225*512)=mainpixel(i,j);

elseif (i>225*512 && i<226*512+1)
pixel10800(i+226*512)=mainpixel(i,j);

elseif (i>226*512 && i<227*512+1)
pixel10800(i+227*512)=mainpixel(i,j);

elseif (i>227*512 && i<228*512+1)
pixel10800(i+228*512)=mainpixel(i,j);

elseif (i>228*512 && i<229*512+1)
pixel10800(i+229*512)=mainpixel(i,j);

elseif (i>229*512 && i<230*512+1)
pixel10800(i+230*512)=mainpixel(i,j);

elseif (i>230*512 && i<231*512+1)
pixel10800(i+231*512)=mainpixel(i,j);

elseif (i>231*512 && i<232*512+1)
pixel10800(i+232*512)=mainpixel(i,j);

elseif (i>232*512 && i<233*512+1)
pixel10800(i+233*512)=mainpixel(i,j);

elseif (i>233*512 && i<234*512+1)
pixel10800(i+234*512)=mainpixel(i,j);

elseif (i>234*512 && i<235*512+1)
pixel10800(i+235*512)=mainpixel(i,j);

```

```

elseif (i>235*512 && i<236*512+1)
pixel10800(i+236*512)=mainpixel(i,j);

elseif (i>236*512 && i<237*512+1)
pixel10800(i+237*512)=mainpixel(i,j);

elseif (i>237*512 && i<238*512+1)
pixel10800(i+238*512)=mainpixel(i,j);

elseif (i>238*512 && i<239*512+1)
pixel10800(i+239*512)=mainpixel(i,j);

elseif (i>239*512 && i<240*512+1)
pixel10800(i+240*512)=mainpixel(i,j);

elseif (i>240*512 && i<241*512+1)
pixel10800(i+241*512)=mainpixel(i,j);

elseif (i>241*512 && i<242*512+1)
pixel10800(i+242*512)=mainpixel(i,j);

elseif (i>242*512 && i<243*512+1)
pixel10800(i+243*512)=mainpixel(i,j);

elseif (i>243*512 && i<244*512+1)
pixel10800(i+244*512)=mainpixel(i,j);

elseif (i>244*512 && i<245*512+1)
pixel10800(i+245*512)=mainpixel(i,j);

elseif (i>245*512 && i<246*512+1)
pixel10800(i+246*512)=mainpixel(i,j);

elseif (i>246*512 && i<247*512+1)
pixel10800(i+247*512)=mainpixel(i,j);

elseif (i>247*512 && i<248*512+1)
pixel10800(i+248*512)=mainpixel(i,j);

elseif (i>248*512 && i<249*512+1)
pixel10800(i+249*512)=mainpixel(i,j);

elseif (i>249*512 && i<250*512+1)
pixel10800(i+250*512)=mainpixel(i,j);

elseif (i>250*512 && i<251*512+1)
pixel10800(i+251*512)=mainpixel(i,j);

elseif (i>251*512 && i<252*512+1)
pixel10800(i+252*512)=mainpixel(i,j);

elseif (i>252*512 && i<253*512+1)

```

```
pixel10800(i+253*512)=mainpixel(i,j);

elseif (i>253*512 && i<254*512+1)
pixel10800(i+254*512)=mainpixel(i,j);

elseif (i>254*512 && i<255*512+1)
pixel10800(i+255*512)=mainpixel(i,j);

elseif (i>255*512 && i<256*512+1)
pixel10800(i+256*512)=mainpixel(i,j);

end

end
```