Institutionen för systemteknik
Department of Electrical Engineering

Examensarbete
Algorithm study and Matlab model for
CCITT Group4 TIFF Image Compression

Master thesis performed in Electronics Systems
By
Azam Khan
LiTH-ISY-EX--10/4451--SE
Linköping December 2010
Algorithm study and Matlab model for
CCITT Group4 TIFF Image Compression

Master thesis in Electronics Systems
At Linköping Institute of Technology

By
Azam Khan
LiTH-ISY-EX--10/4451--SE

Supervisor and Examiner: Kent Palmkvist
External Supervisor: Muhammad Imran (CIIT, Wah)
Linköping December 2010
Publication Title
Algorithm study and Matlab model for CCITT Group4 TIFF Image Compression

Abstract
Smart cameras are part of many automated system for detection and correction in various systems. They can be used for detecting unwanted particles inside a fuel tank or may be placed on an airplane engine to provide protection from any approaching obstacle. They can also be used to detect a finger print or other biometric identification on a document or in some video domain. These systems usually have a very sensitive fast processing nature, to stop some ongoing information or extract some information. Image compression algorithms are used for the captured images to enable fast communication between different nodes i.e. the cameras and the processing units. Nowadays these algorithms are very popular with sensor based smart camera. The challenges associated with these networks are fast communication of these images between different nodes or to a centralized system. Thus a study is provided for an algorithm used for this purpose. In-depth study and Matlab modeling of CCITT group4 TIFF is the target of this thesis. This work provides detail study about the CCITT TIFF algorithms and provides a Matlab model for the compression algorithm for monochrome images.

The compressed images will be of a compression ratio about 1:15 which will help in fast communication and computation of these images. A developed set of 8 test images with different characteristics in size and dimension is compressed by the Matlab model implementing the CCITT group4 TIFF. These compressed images are then compared to same set of images compressed by other algorithms to compare the compression ratio.
Acknowledgment

I dedicate this work to the government of Sweden for providing me an opportunity of free education. Beside this I would dedicate it to my parents and my wife.
Abstract

Smart cameras are part of many automated systems for detection and correction in various systems. They can be used for detecting unwanted particles inside a fuel tank or may be placed on an airplane engine to provide protection from any approaching obstacle. They can also be used to detect a fingerprint or other biometric identification on a document or in some video domain. These systems usually have a very sensitive fast processing nature, to stop some ongoing information or extract some information. Image compression algorithms are used for the captured images to enable fast communication between different nodes i.e. the cameras and the processing units. Nowadays these algorithms are very popular with sensor-based smart cameras. The challenges associated with these networks are fast communication of these images between different nodes or to a centralized system. Thus a study is provided for an algorithm used for this purpose. In-depth study and Matlab modeling of CCITT group4 TIFF is the target of this thesis. This work provides a detailed study about the CCITT TIFF algorithms and provides a Matlab model for the compression algorithm for monochrome images.

The compressed images will be of a compression ratio about 1:15 which will help in fast communication and computation of these images. A developed set of 8 test images with different characteristics in size and dimension is compressed by the Matlab model implementing the CCITT group4 TIFF. These compressed images are then compared to same set of images compressed by other algorithms to compare the compression ratio.
# Table of Contents

1. **Introduction** ............................................................................................................................... 1
   1.1 Image Processing Model ........................................................................................................... 1
   1.2 Image file format ....................................................................................................................... 3
   1.3 Huffman Coding ....................................................................................................................... 5
   1.4 Pixel Packing ............................................................................................................................ 6
   1.5 Run Length Encoding ............................................................................................................... 7
   1.6 Image compression Algorithms ............................................................................................... 8

2. **CCITT and TIFF** ......................................................................................................................... 13
   2.1 Coding Schemes ....................................................................................................................... 13
      2.1.1 Modified Huffman ............................................................................................................ 14
      2.1.2 Modified Read ................................................................................................................. 15
      2.1.3 Modified Modified Read ............................................................................................... 17
   2.2 Tagged Image File Format ...................................................................................................... 18
      2.2.1 Image File Header ............................................................................................................ 18
      2.2.2 Image File Directory ....................................................................................................... 19
   2.3 Group Classification ................................................................................................................. 20
      2.3.1 Group 3 One Dimensional (G31D) .................................................................................. 21
      2.3.2 Group 3 Two Dimensional (G32D) .................................................................................. 22
      2.3.3 Group 4 Two Dimensional (G42D) .................................................................................. 23

3. **CCITT group4 2-D TIFF** ............................................................................................................ 25
   3.1 Group 4 2-dimensional .......................................................................................................... 25
      3.1.1 Coding scheme ................................................................................................................. 25
   3.2 Modes of Coding ..................................................................................................................... 26
      3.2.1 Pass mode ....................................................................................................................... 26
      3.2.2 Vertical mode ............................................................................................................... 27
      3.2.3 Horizontal mode ......................................................................................................... 28
   3.3 Coded Tables ........................................................................................................................... 29
   3.4 Procedure ............................................................................................................................... 32
List of Tables & Figures

Figure 1.1 Image processing
Figure 1.2 Huffman encoding
Figure 1.3 Pixel packing
Figure 1.4 Run-length encoding
Figure 2.1 Coding scheme
Figure 2.2 Modified Huffman structure
Figure 2.3 Modified read structure
Figure 2.4 MR 2-D coding
Figure 2.5 Structure of MR data in a page
Figure 2.6 Scan lines in MMR page
Figure 2.7 File organization of TIFF
Figure 2.8 Different TIFF structures
Figure 2.9 IFH structure
Figure 2.10 IFD structure
Figure 2.11a Makeup and terminating codes for 120
Figure 2.11b Makeup and terminating codes for 8800
Figure 3.1 Changing elements in an image
Figure 3.1 Pass mode
Figure 3.3 Vertical mode
Figure 3.4 Horizontal mode
Figure 3.5 Coding steps for G42D
Figure 3.6 Complete CCITT Group 4 2-D TIFF structure
Figure 4.1 Case study pictures 1
Figure 4.2 Coding of scan line 1
Figure 4.3 Coding lines 2 and 3
Figure 5.1 Matlab coded structure
Figure 7.1 Test images
Figure 6.1 Original vs. compressed
Figure 6.2 Time and compressed size.
Figure 7.2 Comparison of monochrome images compression techniques
Figure 7.1 Generalized vs. G42D
Table 1.1 Image file formats
Table 1.2 Huffman coding methods
Table 1.3 Image compression algorithms.
Table 2.1 Comparisons of MH, MR and MMR
Table 3.1 Terminating codes
Table 3.2 Makeup codes from 64 to 1728
Table 3.3 Makeup codes from 1792 till 2560
Table 3.4 Pass, horizontal and vertical mode codes
Table 6.1 Compressed images vs. original images
Table 7.2 Comparison monochrome image compression techniques.
Table 7.1 Generalized vs. G42D
Chapter 1
Introduction

The aim of this project is to provide a study of an image compression algorithm for monochrome images which has time and compression efficiency. However medium of connectivity provided for these networks are low data rate i.e. ISDN etc. But the demand from these networks is to have fast processing as they are usually used for very sensitive operation. Thus to meet this requirement a study should be provided for an efficient compression algorithm. Several algorithms can be taken in account which includes LZW and all groups of CCITT techniques.

CCITT group 4 is the target of this study because of its efficient compression nature, and it could also be pretty fast when placed on some hardware platform such as FPGA. CCITT group 4 gives a compression ratio of 1:15 by average [10].

Due to the sensitive nature of operations carried out with smart cameras, these compression techniques should hold lossless information which can be better provided with the TIFF file format. TIFF has a wide scope of information about the image in forms of different tags that has the length, width, offset, photometric interpretation and other such information.

This algorithm will work for a smart camera network. An image processing model is also included in the complete design of these smart camera networks which provide the necessary processing in terms of detection and removal. These image processing models can be placed either at the node end or allocated centrally using a network. However image processing models are out of the scope for this study. Chapter 1 provides an overview for the study of basic knowledge needed for this thesis work.

1.1 Image Processing Model

When we say processed images, we mean images captured by camera or other scanned images. Image processing has been in place for a long period of time. These images can be taken in different domain i.e. snap shots, scanned documents etc, which should need some modification for a clearer look at it and this is why image processing is in place. These images can also be taken by a sensor based camera placed on satellites, unmanned aircraft’s or our daily routine pictures. Image processing is used in various fields such as remote sensing, medical imaging, military, movies, documentation, art images and printing. Image processing used to be done by optical, photographic and electronic means but nowadays computers are more popular to do this operation. Over time image processing has advanced and several new dimensions have been added to the process. However the main steps of image processing are presented in figure 1.1.
Image Enhancement

The raw image usually obtained in high sensitive acquisition has some part of the data blurred. Thus image enhancement improves the clarity of the image for good visibility. In this processes noisy parts are removed such as removing blur, adding contrast to differentiate and changing the brightness as required. Such enhancing algorithm monopolizes the pixilated data of the image to perform this function.

Image Restoration

This is the same process as image enhancing but the major difference is that it adds on data elements that are lost in between the pixel. This in return gives more brightness and clarity to the image.

Morphological Processing

Morphological image processing refers to the representation of image for processing. It is very hard to point out some part of an image and make some changes. Morphology is a mathematical way of representing image with simple algorithms and then does the required processing.
Morphology is basically the analysis of the characteristics of image in different sets. These set can be given the form of some transforms. These morphological transforms of an image are erosion, dilation, opening, closing and hit or miss.

- **Segmentation**

  Segmentation as is referred by its name segment pixels in the form of groups. Segmentation of images usually helps changing the image into a form which is easier to process. These segments identifies different characteristic of images that include visual characteristic such as colors. It also defines the boundaries and locates certain areas in an image that will be used for further processing. Different morphological algorithms can be segmented for simplified processing.

- **Object Recognition**

  These are several types of algorithms that are used for recognition of specified pixels in an image. However recognition is the main advantage of digital technology that helps verify certain data elements. Once data has been pixilated it gets very easy to make use of these digital algorithms to process it.

1.2 **Image file format**

- **Graphics Interchange format**

  This is a bitmap image file format. It was introduced in 1987 by CompuServe before JPEG (Joint Photographic Expert Group) and 24 bit were introduced. This is commonly used on the World Wide Web. It is an 8 bit image format which means that one pixel in a GIF (Graphics Interchange Format) represents 256 different colors which are chosen from a true 24 bit RGB (red green blue) color. GIF is used for line art and logo images where as it is unpopular in real life pictures i.e. photographic images. In case of true colors GIF is still very popular for web images. GIF images with 16 colors are much better in quality than a JPEG file. In logos and other type of graphics images where shades are not involved, GIF provides these images with much smaller size at the same resolution. GIF uses lossless image compression for images. As GIF is limited to 256 colors so it is important to use lossless compression techniques so that no colors are lost due to compression. Thus images can make use of GIF compression to provide a smaller file size and better quality instead of JPEG.

- **Joint Photographic Experts Group**

  JPEG is 16 bit color format which is twice as that of the GIF image format. JPEG holds the capability to have millions of color at a time. It allows better quality for images with varying properties for a better quality such as brightness, contrast hue etc.

  JPEG is most commonly used for real time images as photographic images. It is also popular for web images. The compression algorithm used by JPEG is the JPEG compression algorithm which is
very complex and reliable. It provides up to 80 percent compression of the real image. However to achieve a more optimum and better quality of the image, it can be compressed to 60 percent. This compression algorithm is applied on a sliding scale of quality and size. This compression algorithm removes many details while compressing the image and encodes these details while displaying the image.

- **Portable Network Graphics**

PNG (Portable Network Graphics) was basically designed for World Wide Web and image editing. It replaced GIF because of its portable nature. It can represent images up to 48 bits per pixel. Main types of images encoded using PNG are true color, gray scale and palette. This wide support factor of images gives PNG an edge over the JPEG image format. PNG compression is a lossless technique. The compression algorithms used by PNG are LZW and ZIP.

- **Tagged Image File format**

Tagged Image File Format (TIFF) is a common file format that is found in most imaging programs. TIFF is as known by its name a tagged file that holds additional information about the image. The main compression algorithm used by TIFF is LZW, JPEG compression and CCITT. TIFF is the aim of this Thesis work which is explained in detail in the following chapters. However the main CCITT compression algorithms for TIFF are

- CCITT Group 3 1-D
- CCITT Group 3 2-D
- CCITT Group 4 2-D

These CCITT compression algorithms are defined in two TIFF standards. These two standards are T.4 and T.6. T.4 holds the specification for CCITT group 3 where as the T.6 has the extended quality of 2 dimensional CCITT group 4 compression algorithms.

<table>
<thead>
<tr>
<th>File formats</th>
<th>Bits per sample</th>
<th>Compression algorithms</th>
<th>BMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIF</td>
<td>1 to 8 bits</td>
<td>LZW</td>
<td>Indexed colors</td>
</tr>
<tr>
<td>JPEG</td>
<td>24 bits</td>
<td>JPEG compression</td>
<td>RGB</td>
</tr>
<tr>
<td></td>
<td>8 bits</td>
<td></td>
<td>Gray scale</td>
</tr>
<tr>
<td>PNG</td>
<td>24 or 48 bits</td>
<td>ZIP</td>
<td>RGB</td>
</tr>
<tr>
<td></td>
<td>8 or 16 bits</td>
<td>LZW</td>
<td>Gray scale</td>
</tr>
<tr>
<td></td>
<td>1 to 8 bits</td>
<td></td>
<td>Indexed color Line Art</td>
</tr>
<tr>
<td></td>
<td>1 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIFF</td>
<td>24 or 48 bits</td>
<td>JPEG compression</td>
<td>RGB</td>
</tr>
<tr>
<td></td>
<td>8 or 16 bits</td>
<td>LZW</td>
<td>Gray scale</td>
</tr>
<tr>
<td></td>
<td>1 to 8 bits</td>
<td>CCITT</td>
<td>Indexed color Line Art</td>
</tr>
<tr>
<td></td>
<td>1 bit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 1.1 Image file format.*
1.3 Huffman Coding

Huffman encoding is a compression algorithm where define data elements with higher frequency are coded using special technique. Huffman encoding is based on a minimum coding technique. There are two types of prefix coding methods used for this purpose,

- Fixed length codes
- Variable length codes

As the name suggests fixed length coding refers to codes having the same length whereas the variable length code words are of varying length. Consider four elements a, b, c and d. The frequencies are 20, 10, 30 and 40. Hence fixed length and variable length codes can be given as follow.

<table>
<thead>
<tr>
<th>Coding</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>20</td>
<td>10</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>Fixed length</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
</tr>
<tr>
<td>Variable length</td>
<td>1000</td>
<td>101</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1.2 Huffman coding methods.

The fixed length coding takes 20x3 + 10x3 + 30x3 + 40x3 so the total is 300 bits whereas variable length coding takes 1x40 + 2x30 + 3x20 + 4x10 = 200 bits to represent the same data. Thus high number of bits can be reduced with Huffman coding as given in the following figure. The following figure is the implementation of the Huffman coding. The steps followed above are.

- Compute the complete frequency of all the data N.
- Start with the data element with the highest frequency going down to the lowest. As in the above figure starts with d and ends with a.
- Hence a compressed code will be obtained with variable-length codes for the data.
1.4 Pixel Packing

Pixel packing is a technique used to store data sequentially in memory rather than being a data compression technique. Due to its advantage in preserving all important memory space is the mostly utilized technique for storage of bitmap images. Suppose an image with 4 bits per pixel store each pixel in one byte of the memory. This configuration will then not utilize half of the byte, and half of the precious memory will be wasted. To Utilize the whole of the memory we will store two 4-bit pixels in a byte of memory thus making use of whole the memory. This can be illustrated by an image as in figure 1.3.
Pixel packing provides a straightforward solution but it comes with a drawback when this solution is applied to a data arranged in the form of an array where each byte contains a pixel. It would be faster to access and attain this 4-bit pixel from a byte rather than 2-pixels stored in a byte which would require further masking and shifting of each byte of data for proper retrieval. So, an optimum solution brings a balance between read/writes and size of the image can be adopted.

1.5 Run Length Encoding

RLE is a data compression technique widely accepted by the bitmap file formats that includes TIFF, BMP and PCX. An important characteristic of RLE is that it compresses any kind of information independent of its type, only difference will be the achieved compression ratio which depends on the content. The output in terms of achieving high compression ratios is very rare as compared to complex compression techniques. RLE can both be implemented and executed in a very simple manner thus providing it with an edge against using much more complicated techniques any compression algorithm at all.

RLE uses a technique which looks for strings of repeated characters and thus bringing down the physical size by encoding these repeated strings. The encoding is applied on the repeated string (run), folding it in two bytes, amongst the two bytes the first one gives the number of characters called the run count containing 1-256 Characters. Taking as an example 16 '0' characters would require 8 bytes to store i.e. 0000000000000000. After compressing RLE requires 2 bytes: 08 00 here 08 is the count and 00 is the character string. The block diagram for RLE algorithm is given as follow.
1.6 Image compression Algorithms

Image compression algorithms are divided into two groups

- lossless compression
- lossy compression
Image compression ratio is the ratio between the real image data and the compressed data. Consider an image compression that reduces the image to half the size of the original image or 2:1. The lossless data in this case refers to the compressed data which hold all the information of the image such that when extracted back it gives the original copy of the image. The data files that need to use lossless compression algorithms are text file, lossless video and audio data etc. If the text file is considered for compression then it can be seen that if a single text is lost, it will make the text meaningless and misleading.

While on the other side lossy image compression refers to the phenomena that some part of the image data can be lost. This type of compression does not provide a complete copy of the original image. There is always such information in images that can be thrown away and it helps to reduce the image size by discarding unwanted data.

Thus it depends on the Image type what kind of technique is required. However their might be cases where both lossy and lossless compression are used for better results. Different types of compression algorithms are given as follow.

- **LZ compressors**
  LZ compressors technique developed by Abraham Lempel and Jakob Zivin, has many variation like LZ1 (LZ1977), LZ2 (LZ1978). LZ is a group of lossless compression schemes used as a base for many different algorithm, like GZIP, PKZIP, WINZIP, ALDC, LZS and PNG and others.

- **Deflate**
  Deflate was developed by Phil Katz in 1996. It is lossless and use LZ77 and Huffman techniques as a base. Deflate algorithm technique allows us to extract the original data from the already compressed data.

- **Lempel-Ziff-Welch (LZW)**
  Lempel-Ziff-Welch is an enhanced version of LZ78 and was developed by Terry Welch in 1984 and is owned by Unisys Corporation. It can be applied to any data type but is mostly used for images. In LZW the string of characters are replaced by single codes. Instead of analysis of incoming text, it adds new string to a table of strings. The outputs are of any arbitrary length, but it contains more bits than a single character.

- **CCITT Groups**
  Group 3, also known as CCITT T.4, was developed by the International Telegraph and Telephone Consultative Committee in 1985. This was developed for encoding and compressing of image data. Its main use is in fax transmission but it is also optimized for handwritten document and scanned images. There are two forms of Group 3, one dimensional and two dimensional. The two
dimensional has great compression rates. As it was developed as a transmission protocol, its encoding scheme contains error detection codes. Group 4, also known as CCITT T.6, is an enhanced version of two dimensional Group 3. It is faster and the compression rate is double that of Group 3 as it is designed for storage therefore it does not contain error detection.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Loss</th>
<th>Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLE</td>
<td>Lossless</td>
<td>Monochrome</td>
</tr>
<tr>
<td>LZ compressor</td>
<td>Lossless</td>
<td>All Images</td>
</tr>
<tr>
<td>Huffman coding</td>
<td>Lossless</td>
<td>All Images</td>
</tr>
<tr>
<td>Deflate</td>
<td>Lossless</td>
<td>All Images</td>
</tr>
<tr>
<td>CCITT groups</td>
<td>Lossless</td>
<td>Monochrome</td>
</tr>
<tr>
<td>LZW</td>
<td>Lossless</td>
<td>All Images</td>
</tr>
<tr>
<td>JPEG</td>
<td>Lossy</td>
<td>True colors and complex</td>
</tr>
<tr>
<td>JPEG 2000</td>
<td>Lossy</td>
<td>True colors and complex</td>
</tr>
<tr>
<td>PNG</td>
<td>Lossless</td>
<td>All Images</td>
</tr>
<tr>
<td>Fractal</td>
<td>Lossy</td>
<td>True colors and complex</td>
</tr>
</tbody>
</table>

*Table 1.3 Image compression algorithms.*

**JPEG**

It was developed by the Joint Photographic Experts Group of the International Standards Organization (ISO) and CCITT. It is the most common compression technique used in images with complex 24-bit (True color). It uses Discrete Cosine Transform (DCT) technique to discard image data that is imperceptible to human eye, and Huffman compression technique for more compression. First is the image divided into 8*8 blocks and then DCT is used for the transformation of each block. Quantization followed by variable length coding (VLC) is used for compression. The quantization step size is constant for all blocks.

**JPEG 2000**

JPEG 2000 is an enhanced version of JPEG. It was developed by the Joint Photographic Experts Group of International Standards Organization in 2000. It is based on DWT. DWT is applied on image tiles to decompose it into different decomposition levels which contain a number of sub bands. Two types of wavelet filters (Debaucheries 9-tap/7-tap and 5-tap/3-tap filters) are implemented. The default is 9-tap/7-tap filter.
• **PNG Compression**

PNG was developed in 1996 and is free to use but can be implement only in the PNG file format. It uses the Deflate compression. The PNG compression use the deflate method with a sliding window of a maximum 32768 bytes.

• **Fractal Compression**

Fractal Compression was developed by Michael Barnsley in 1991. It uses a mathematical process to identify redundant repeating patterns within images. These matching patterns are identified by geometrical transformations. These repeating patterns are stored one time along with its location information. It is a lossy technique which is fast.
CHAPTER 2
CCITT and TIFF

2.1 Coding Schemes

Before compressing the image data, information about the data to be compressed should be known. Consider a CCITT algorithms dealing with a page of size 8.5 x 11 inch. The page is divided into horizontal and vertical lines. These horizontal lines are known as scan lines. Dots per inch and pixels per inch are two standards for image resolution. An 8.5 x 11 inch page is 1728 x 2200 pixels. One scan line is 1728 pixel long, the normal resolution is 200 x 100 dpi and a fine resolution is 200 x 200 dpi.

Each pixel is represented by 1 bit; the number of pixels that will form the above page is 3,801,600. Sending this data through an ISDN line will take approximately 7 min. If the resolution of the page is increased, the time taken by the transmission will increase. Image compression does not require transferring every exact bit of the binary page information thus reducing the transfer time.

The most common encoding used for CCITT compression is Modified Huffman which is supported by all the fax compression techniques. Other options used are Modified Read and Modified Modified Read. The following table gives an overview of these encoding/decoding techniques.
Table 2.1: Comparisons of MH, MR and MMR[11].

### 2.1.1 Modified Huffman

The fax pages contain many runs of white and black pixels which make RLE efficient for minimizing these run lengths. The efficiently compressed run lengths are then combined with Huffman coding. Thus an efficient and simple algorithm is achieved by combining RLE with Huffman coding and this is known as Modified Huffman. RLE consists of terminating and makeup codes.

MH coding uses specified tables for terminating and makeup codes. Terminating codes represent shorter runs while the makeup codes represent the longer runs. The white and black pixel runs from 0 to 63 are represented by terminating codes while greater than 63 are represented with makeup codes which mean that greater than 63 bit runs are defined in multiples of 64 bits which are formed by the terminating codes. These tables are given in chapter 4. A scan line represented with long runs gives a make code which is less than or equal to the pixel run and then the difference is given by the terminating code. The following example will help in understanding how it works. There are three different types of bit pattern in MH coding:

- Pixel information (data)
- Fill
- EOL

The term Fill refers to the extra 0 bits that are added to a small data line which fills the left space in the data. The Fill patterns bring highly compressed scan line to a preferred minimum scan line time (MSLT), which makes it complete and transmittable. Consider a transmission rate of 4800 bps with an MSLT 10ms so the minimum bits per scan line is 48 bits. 1728 pixels scan line is compressed to 43 bit. 31 data bit + 12 EOL bits which in total is 43 bits. The left space is filled by 5 Fill bits given as follow.
In addition to this another special bit pattern used in the MH coding is EOL. EOL are special bit patterns which have several different identification function i.e.

- EOL at the start of the scan line indicate the start of the scan of line
- EOL at the end of the scan line consist of 11 0's followed by a 1. It helps in stopping the error from one scan line penetrating into other scan lines and each line is independently coded.
- At the end of each page an RTC signal is given which holds six EOL patterns which identify the end of page.

2.1.2 Modified Read

MR is also known as Modified Relative Element address designated (READ). MR exploits the correlation between successive lines. It is known that two consecutive lines have a very high percentage of single pixel transition due to a very high resolution of the images. Using these phenomena, instead of scanning each scan line as done in MH, MR takes in account a reference line and then encodes each scan line that follows. In fact it is more appropriate to say that MR is more complex than the MH algorithm.

MR encoding encounters both MH and MR coding technique. The reference line is encoded using MH and the subsequent line is encoded using MR encoding until the next reference line appears. The decision on how to encounter the next reference line is taken by a parameter K. The value of K defines the resolution of the compression.

MR is a 2-dimensional algorithm. The value of K defines the number of lines that uses 2-dimensional phenomena, which is K-1 lines. However the reference line using the MH algorithm is using 1-dimension. For a normal resolution of an image the value of K is set to 2 the reference line is encoded every second scan line. Whereas the value of K set to 4 will give a higher compression ratio because the reference line is MH encoded every 4 line, making it more complex and compressed. The following figure shows scan lines for both resolution of K set to 2 and 4.
The advantage of having low resolution over high resolution is that the error propagation into the subsequent line is reduced with lower number of dependent scan lines. However in MR encoding the value of K can be set as high as 24. The change between two subsequent lines i.e. the reference line and the next scan line given by MR can be given as follow:

- `a0` is start of changing element in the coding line which is also the reference for the next changing elements.
- `a1` first transition on the coding line.
- `a2` second transition on the coding line.
- `b1` first transition on the reference line on the right of the `a0`, first opposite color transition
- `b2` first transition on the reference line.

In the above figure the reference line is coded with the MH coding while the next scan line is coded with MR. Hence it can be seen that there are very minor changes between both the scan lines. MR takes advantage of the minor changes and encodes only the changing elements `a0`, `a1` and `a2` instead of the complete scan line. There are three functional encoding modes of MR, which decide on how to code these changing elements of the scan line with respect to the reference line. These modes are:

- Pass mode
- Vertical mode
- Horizontal mode
It is due to these different modes of MR which makes it more complex. These MR functional modes are discussed in detail in chapter 3. And then one can refer back to this part to completely understand it. The structure of MR is given as follow

<table>
<thead>
<tr>
<th>EOL</th>
<th>Data</th>
<th>fill</th>
<th>EOL</th>
<th>Data</th>
<th>EOL</th>
<th>Data</th>
<th>fill</th>
<th>EOL</th>
<th>Data</th>
<th>EOL</th>
<th>Data</th>
<th>EOL</th>
<th>Data</th>
<th>EOL</th>
<th>Data</th>
<th>EOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>1-D</td>
<td></td>
<td>+0</td>
<td>1-D</td>
<td>+1</td>
<td>1-D</td>
<td></td>
<td>+0</td>
<td>1-D</td>
<td>+1</td>
<td>1-D</td>
<td>+1</td>
<td>1-D</td>
<td>+1</td>
<td>1-D</td>
<td>+1</td>
</tr>
</tbody>
</table>

\[ K = 2 \]

- **EOL+1** MH coding of next line
- **EOL+0** MR coding of next line
- **FILL** Extra 0 bits
- **RTC** End of page with 6 EOLs

Figure 2.5 Structure of MR data in a page.

### 2.1.3 Modified Modified Read

ITU-T Recommendation T.6 gives the Modified Modified Read or MMR encoding algorithm. MMR is an upgraded version of the MR. They are both 2-dimensional algorithms but MMR is an extended version of the 2-dimension. The fundamentals of MMR are same as MR except a few minor changes to the algorithm however the modes of MR i.e. pass mode, vertical mode and horizontal mode are the same for MMR encoding.

The major change in the MMR with respect to MR is the K parameter. The MMR algorithm does not use the K parameter and recurring reference line. Instead of these the MMR algorithm uses an imaginary scan line which consist of all white pixels which is the first line at the start of each page and a 2-Dimension line follows till the end of the page. This introduced scan line of all whites is the reference lines alike the MR.

The error propagation in MMR has a very high predictability because of the connected coding method of all the scan lines. Thus ECM (error correction mode) is required for MMR to be enabled. ECM guaranties error free MMR algorithm. Thus MMR does not require any EOL however a EOFB (end of facsimile block) is required at the end of page which is the same as RTC in MH. The organization of data in MMR and the EOFB block bit sequence is given as follow.

<table>
<thead>
<tr>
<th>Data 2-D</th>
<th>Data 2-D</th>
<th>Data 2-D</th>
<th>Data 2-D</th>
<th>Data 2-D</th>
<th>Data 2-D</th>
<th>Data 2-D</th>
<th>Data 2-D</th>
<th>Data 2-D</th>
<th>Data 2-D</th>
<th>Data 2-D</th>
<th>Data 2-D</th>
<th>Data 2-D</th>
<th>Data 2-D</th>
<th>Data 2-D</th>
<th>Data 2-D</th>
<th>EOFB</th>
</tr>
</thead>
</table>

---Scan lines of page---

**EOFB bit sequence**

00000000000001 000000000001

Figure 2.6 Scan lines in MMR page.
2.2 Tagged Image File Format

Tagged Image File Format (TIFF) is purely a graphical format i.e. pixilated, bitmap or rasterized. TIFF is a common file format that is found in most imaging programs. The discussion here covers mostly the TIFF standard of ITU-T.6 which is the latest. T.6 includes all the specification of the earlier versions with little addition. TIFF is flexible and has good power rating but at the same time it is more complex. Extensibility of TIFF makes it more difficult to design and understand. TIFF is as known by its name a tagged file that holds the information about the image. TIFF structure is organized into three parts

- Image file header (IFH)
- Bit map data (black and white pixels)
- Image File Directory (IFD)

Figure 2.7 File organization of TIFF.

Consider an example of three TIFF images file structures. These three structures hold the same data in possible three different formats. The IFH or the header of TIFF is the first in all the three arrangements. However in the first arrangement IFD's have been written first and then followed by the image data which is efficient if IFD data is needed to be read quickly. In the second structure the IFD is followed by its particular image which is the most common internal structure of the TIFF. In the last example the image data is followed by its IFD's. This structure is applicable if the image data is available before the IFD's.

Figure 2.8 Different TIFF structures.

2.2.1 Image File Header

A TIFF file header consists of 8-bytes which is the start of a TIFF file. The bytes are organized in the following order

- The first two bytes defines the byte order which is either Little Endian (II) or Big Endian (MM). The Little Endian byte order is that it starts from least significant bit and ends on the most significant and Big Endian is vice versa.
II = 4949H  
MM = 4D4DH

- The third and fourth bytes hold the value 42H which is the definition for the TIFF file
- The next four bytes hold the offset value for the IFD. The IFD may be at any location after the header but must begin after a word boundary.

| Byte order | 42 | Byte offset for IFD |

*Figure 2.9 IFH structure.*

### 2.2.2 Image File Directory

Image file directory (IFD) is a 12 byte field that holds information about the image including the color, type of compression, length, width, physical dimension, location of the data, and other such information of the image. Before the IFD there is a 2 byte tag counter. This tag counter holds the number of IFD used. This is followed by a 12 byte IFD and four 0 bytes at the end of the last byte. Each IFD entry has the following format:

- The first two bytes of the IFD hold the identification field. This field gives information what characteristic of the image it is pointing to. This is also known as the tag.
- The next two bytes gives the type of the IFD i.e. short, long etc
- The next four bytes hold the count for the defined tag type
- The last two bytes hold the offset value for the next IFD which is always an even number. However the next IFD starts by a word difference. This value offset can point anywhere in the Image even after the image data.

The IFD are sorted in ascending order according to the Tag number. Thus a TIFF field is a logical entity which consist of a tag number and its value.

| Tag entry count 2-bytes |
| Tag 0 12 bytes |
| Tag 1 12 bytes |
| Tag n 12 bytes |
| Next IFD offset or null bytes 4 bytes |

*Figure 2.10 IFD structure.*
The IFD is the basic tag field that holds information about the image data in a complete TIFF file. The data is either found in the IFD or retrieved from an offset location pointed in the IFD. Due to offset value to other location instead of having a fixed value makes TIFF more complex. The offset values in TIFF are in three places.

- Last four bytes of the header which indicates the position of the first IFD
- Last four bytes of the IFD entry which offsets the next IFD.
- The last four bytes in the tag may contain an offset value to the data it represents or possibly the data itself

2.3 Group Classification

This type of compression is used for facsimile and document imaging files. It is a lossless type of image compression. The CCITT (International telegraph and telephone consultative committee) is an organization which provides standards for communication protocol for black and white images or telephone or other low data rate data lines. The standards given by ITU are T.4 and T.6. These standards are the CCITT group 3 and group 4 compression methods respectively. CCITT group compression algorithms are designed specifically for encoding 1 bit image. CCITT is a non adaptive compression algorithm. There are fixed tables that are used by CCITT algorithms. The coded values in these tables were taken from a reference of set of documents containing both text and graphics.

The compression ratio obtained with CCITT algorithms is much better than 4:1. The compression ratio for a 200 x 200 dpi image achieved with group 3 is 5:1 to 8:1 which is much increased with group 4 that is up to 15:1 with the same image resolution. However the complexity of the algorithms increases with the ratio of its comparisons. Thus group 4 is much more complex than group 3.

The CCITT algorithms are specifically designed for typed or handwritten scanned images, other images with composition different than that of target for CCITT will have different runs of black and white pixels. Thus such bi-level images compressed will not give the required results. The compression will be either to a minimum or even the compressed image will be greater in size than the original image. Such images at maximum can achieve a ratio of 3:1 which is very low if the time taken by the comparisons algorithms is very high. The CCITT has three algorithms for compressing bi level images,

- Group 3 one dimensional
- Group 3 two dimensional
- Group 4 two dimensional

Earlier when group 3 one dimensional was designed it was targeted for bi level, black and white data that was processed by the fax machines. Group 3 encoding and decoding has the tendency of being fast and has a reputation of having a very high compression ratio. The error correction inside a group 3 algorithm is done with the algorithm itself and no extra hardware is required. This is done with special data inside the group3 decoder. Group 3 makes use of MH algorithm to encode.
The MMR encoding has the tendency to be much more efficient. Hence group 4 has a very high percentage of compression as compared to group 3, which is almost half the size of group 3 data but it is much more time consuming algorithm. The complexity of such an algorithm is much higher than that of group 3 but they do not have any error detection which propagates the error however special hardware configuration will be required for this purpose. Thus it makes it a poor choice for image transfer protocols.

Document imaging system that stores these images have adopted CCITT compression algorithms to save disk spaces. However in age of good processing speeds and handful of memory CCITT encoded algorithms are still needed printing and viewing of data as done with adobe files. However the transmission of data through modems with lower data rates still require these algorithms.

2.3.1 Group 3 One Dimensional (G31D)

The main features of G31D are given as follow

- G31D is a variation of the Huffman type encoding known as Modified Huffman encoding.
- The G31D encodes a bi-level image of black and white pixels with black pixels given by 1 and white with 0's in the bitmap.
- The G31D encodes the length of a same pixel run in a scan line with variable length binary codes.
- The variable length binary codes are taken from pre defined tables separate for black and white pixels.

The variable code tables are defined in T.4 and T.6 specification for ITU-T. These tables are determined by taking a number of typed and handwritten documents. Which were statistically analyzed to the show the average frequency of these bi level pixels. It was decided that run length occurring more frequently were assigned small code will other were given bigger codes.

As G31D is a MH coding scheme which is explained earlier in the chapter so we will give some example of how the coding is carried out for longer run of same pixels. The coded tables have continuous values from 0 to 63 which are single terminating codes while the greater are coded with addition of makeup codes for the same pixels, only for the values that are not in the tables for a particular pixel. The code from 64 to 2623 will have one makeup code and one terminating code while greater than 2623 will have multiple makeup codes. Hence we have two types of tables one is from 0 to 63 and other from 64 till 2560. The later table is selected by statistical analysis as explained above.

Consider a pixel run for 20 black. Hence it is less than the 63 coded marks in the table. We will look for the value of 20 in the black pixel table which is 00001101000. Hence this will be the terminating code for the 20 black pixel run which is half the size of the original. Thus a ratio 2:1 is achieved.

Let us take the value 120 which is greater than 63 and is not present in the statistically selected pixel run. Here we will need a makeup code and a terminating code. The pixel run can be broken into 64 which is the highest in the tables for this pixel run and 57 which will give 120 pixels run.
120 = 64 + 57
64 coded values is 11011
57 coded values is 01011010
Hence, for 120 is 11011 the makeup code and 01011010 terminating code as given in the figure 2.11a.

Now consider a bigger run of black pixel which is 8800. This can be given a sum of 4 make up and one terminating code
8800 = 2560 + 2560 + 2560 + 1088 + 32
This is 000000011111, 000000011111, 000000011111, 000001110101 and 000001101010.
So it can be given as shown in figure 2.11b

<table>
<thead>
<tr>
<th>Makeup Code</th>
<th>Terminating Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>11011</td>
<td>1011010</td>
</tr>
</tbody>
</table>

*Figure 2.11a Makeup and terminating codes for 120*

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000000011111</td>
<td>000000011111</td>
<td>000000011111</td>
<td>000001110101</td>
<td>000001101010</td>
</tr>
</tbody>
</table>

*Figure 2.11b Makeup and terminating codes for 8800*

The structure of G31D is as MH encoding. The beginning EOL code of each page is a 12 bit code with eleven 0's and a 1 at the end of the code. This unique code which gives the start of one scan line where as the end of another scan line. If a noise corrupts a signal during G31D transmission the decoder discards the data and stop decoding data until the next EOL, start of line is detected and continues with normal transmission of data.

This G31D page transmission is aborted with the RTC code which consists of 6 EOL codes. These EOL and RTC are not part of the data but they only comprise the protocol of transmission. However the FILL bits which are variable number of 0's add to the data is part of the data. These FILL bits are used to increase the transfer time to a certain level which is the requirement to be meet.

### 2.3.2 Group 3 Two Dimensional (G32D)

As the name of G32D shows, it is a 2-dimensional. It means that it compresses two lines at a time. The bit packing is done with MR encoding as explained above. G32D takes two scan lines at a time where one is a reference line for the other. The reference line itself is coded with MH and the scan line to follow is done with MR coding. There is a K factor that describes the resolution of the algorithm. If you go down to pixel level for these written or typed documents, there are a very high percentage of same scan lines. Thus 2-D compression is more efficient for these reasons.
These algorithms can be hit with errors twice the effect then that of 1-D algorithms. But if played correctly with the K factor this problem can be handled. If the K is 2 then K-1 will be lost which is the only line encoded with 2-D. Usually the K factor has a value of 2 or 4 which identifies the data in a block having one 1-D encoded lines in each block.

The decoder for G32D does not use these K factor to identify the correct data. Instead the EOL code is used for this purpose. As shown in the structure of MR encoding an EOL with +1 is added to 1-D coding line and +0 to a 2-D line which is encounter by the decoder to read out the correct data.

2.3.3 Group 4 Two Dimensional (G42D)

G42D is the extension of G32D which has almost taken over the G32D. This algorithm does not use the K factor; instead it uses an imaginary white line to start the page encoding and is followed by the scan lines. All lines in the G42D are 2-D coded. Group 4 will usually have ratio twice as that of 1-D group 3. However the main disadvantage of G42D is that it takes more execution time than that of the Group 3 algorithms but this significance can be competed with the hardware implementation of group 4. G42D TIFF is main aim of this thesis work. However this algorithm is described in detail in chapter 3 of this document.
Chapter 3
CCITT group4 2-D TIFF

The most efficient of all the CCITT encoding methods is the CCITT group4 2-D TIFF. This type of compression has a very high ratio of compression to the original image that is about 15:1. However CCITT group4 2-D TIFF is very complex to implement. There are two algorithms which comprises CCITT group4 2-D TIFF.

- TIFF algorithm
- G42D algorithm

G42D is compression algorithm while the TIFF is a file format that tags the information about the compressed image.

3.1 Group 4 2-dimensional

G42D is an extension of the Modified Modified Read algorithm discussed in chapter 2. This algorithm is defined in the ITU-T recommendation T.6. This algorithm is the same as MR encoding with the removal of the K factor and instead an addition of an imaginary line which serves as the reference line. This reference line is a complete scan line of white pixels. The efficient nature of G42D has made the algorithm replace the use of G32D in many applications. The encoding speed is as fast as the earlier algorithms and it also sometime encodes much faster than other algorithms.

The main advantage of the G42D over the G32D is the removal of the K factor. Due to this K factor an EOL bit was added to the end of every line, which increases the size of the compression file. By the removal of the K factor and addition of the imaginary line removes these EOL bits which in addition reduce the file size by half the ratio to G32D. The RTC code at the end of the G32D is replaced by a EOFB code at the end of the page. This EOFB has two EOL codes which are padded with some extra fill bits to end on a byte boundary. G42D is designed for data networks and data on disk drives.

3.1.1 Coding scheme

G42D uses MMR compression algorithm. This is a two dimensional coding scheme which takes two lines at a time. One is the reference line and the other is the coding line. It can be seen that when going down to 200 x 200 dpi image resolution. The difference between two consecutive line is very small, hence by making a two dimensional coding exploits this phenomena. Thus the coded line is coded with respect to the reference line. The reference line lies above the coded line. The reference line for the first coding line for the image is an imaginary white line.

MMR coding scheme is an extension of the MR coding scheme which has the same mode of coding i.e. pass mode, horizontal mode and vertical mode. These coding modes are described on the basis of the changing element in the scan line and the reference line these changing elements are also described in The MR coding however a summery is given in Figure 3.1.
3.2 Modes of Coding

This coding scheme has three modes. These modes are defined on the basis of the difference between the changing elements of the scan line to the reference line.

- Pass mode
- Vertical mode
- Horizontal mode

3.2.1 Pass mode

Pass mode is addressed in MR and MMR coding schemes. Pass mode of pixel coding occurs when b1 and b2, the first two changing elements of the reference line occurs in between the a0 and a1, the first two changing elements of the scan line. In binary encoding the pass mode is addressed with a 0001 code. The binary value of the reference in this case is known by the encoder, if it lies between the same pixel run of the scan line dose not needs to be included. Thus due to no change in the pixel run in scan line it is not referred to the code. Pass mode is very efficient in regard to encode larger runs of pixels in the scan line with just four bits.

When once the pass mode in encountered and coded the notions for the scan line and reference line shifts to the right in the lines respectively. That is the position of a0 is shifted to the position of b2.
Parameter for the reference and scan line are then recalculated. Taking in account that b2 does not reside above a1.

![Pass mode](image)

*Figure 3.2 Pass mode.*

This is a pass mode as is evident that b1 and b2 lies between a0 and a1 which is a run of 12 white pixel and can be given with a pass code of 0001 the changing elements are shifted.

### 3.2.2 Vertical mode

If the second changing element of the reference line b1 occurs within a distance of 3 pixels of the second changing element of the scan line a1 either to its left or to the right, this is known as Vertical mode. According to the distance between the two changing elements and the position of b1 with respect to a1 defines the code for that setup. These codes are given in table 3.1.

![Vertical mode](image)

*Figure 3.3 Vertical mode.*
The vertical mode gives seven different codes which gives the distance of a1b1 are given as \( V(l/r)n \), where \( v \) stands for vertical mode \( l/r \) stands for left or right of the changing element a1 and n is the distance. The vertical mode can only handle difference of distance from 0 to 3. Once the vertical encoding is done shift the position of a0 to a1 and refer the other changing elements with respect to a0. The following figure shows how a vertical mode is identified and then coded.

Hence b1 is at a 2 pixel distance to the right hence it can be denoted with the vertical modes \( V(R)2 \) and the code from table 000010.

### 3.2.3 Horizontal mode

Horizontal mode is the most complex and least efficient of all the coding modes. Horizontal mode makes use of the modified Huffman codes that are given in table 4.2, 4.3 and 4.4 separately for black and white pixels. Horizontal mode occurs when vertical mode and pass mode does not exist i.e. the distance between the a1b1 grouping of pixels is greater than 3 to either side of right left. However horizontal mode appears lesser than the vertical and pass mode which is the main essence of CCITT group4. The code formation of a scan line with horizontal mode is given as the following:

- The code starts with a 001 which is a standard for all horizontal mode codes
- Next follows the code for the pixel run between the first two changing elements, that are if the run is a 5 white pixel than the code for 5 white pixel is taken from table 4.2.
- The last bits of the code are formed by the pixel run from a1 till a2 and the horizontal code ends. This code is given as 001+MH (a0a1) +MH (a1a2) if horizontal mode appears.

![Figure 3.4 Horizontal mode](image)

*Figure 3.4 Horizontal mode.*
This is horizontal mode because the distance between a1 and b1 is more than 3 pixels. Thus horizontal code are given as a combination of various MH codes. Here we have to use two MH codes one is 4 white pixels and the other is for 8 black pixels so it can be given as 001+MH4W+MH8B =001+1011+0011 = 00110110011

3.3 Coded Tables

In this document there are four coded tables in the following order

- Pixel runs of white and black pixels from 0 to 63.
- Statistically selected longer run from 64 till 1728.
- Same table from 1792 till 2560pixel run for black and white pixels.
- Coded tables for coding modes.

<table>
<thead>
<tr>
<th>Terminating codes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>White pixel run</strong></td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td>25</td>
</tr>
<tr>
<td>26</td>
</tr>
<tr>
<td>27</td>
</tr>
<tr>
<td>28</td>
</tr>
<tr>
<td>29</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>31</td>
</tr>
<tr>
<td>32</td>
</tr>
</tbody>
</table>
Table 3.1 Terminating codes [11].

The following two tables hold the codes for statistically selected makeup codes. Table 4.2 holds separate values for black and white pixels while Table 4.3 has the same makeup code for both types of pixel runs.

**Makeup codes**

<table>
<thead>
<tr>
<th>White pixel run</th>
<th>code</th>
<th>Black pixel run</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>11011</td>
<td>64</td>
<td>00000110111</td>
</tr>
<tr>
<td>128</td>
<td>10010</td>
<td>128</td>
<td>000011001000</td>
</tr>
<tr>
<td>192</td>
<td>01011</td>
<td>192</td>
<td>000011001001</td>
</tr>
<tr>
<td>256</td>
<td>0110111</td>
<td>256</td>
<td>00001101111</td>
</tr>
<tr>
<td>320</td>
<td>00110110</td>
<td>320</td>
<td>000001100111</td>
</tr>
<tr>
<td>384</td>
<td>00110111</td>
<td>384</td>
<td>000001101000</td>
</tr>
<tr>
<td>448</td>
<td>0100100</td>
<td>448</td>
<td>000001101011</td>
</tr>
<tr>
<td>512</td>
<td>01100101</td>
<td>512</td>
<td>000001101100</td>
</tr>
<tr>
<td>576</td>
<td>01101000</td>
<td>576</td>
<td>000001101101</td>
</tr>
<tr>
<td>640</td>
<td>01100111</td>
<td>640</td>
<td>000001101110</td>
</tr>
<tr>
<td>704</td>
<td>011001100</td>
<td>704</td>
<td>000001101111</td>
</tr>
<tr>
<td>768</td>
<td>011001101</td>
<td>768</td>
<td>000001101100</td>
</tr>
<tr>
<td>Code</td>
<td>Make up code</td>
<td>Code</td>
<td>Make up code</td>
</tr>
<tr>
<td>--------</td>
<td>--------------</td>
<td>--------</td>
<td>--------------</td>
</tr>
<tr>
<td>832</td>
<td>011010010</td>
<td>832</td>
<td>000000100110</td>
</tr>
<tr>
<td>896</td>
<td>011010011</td>
<td>896</td>
<td>000000110010</td>
</tr>
<tr>
<td>960</td>
<td>011010100</td>
<td>960</td>
<td>0000001110011</td>
</tr>
<tr>
<td>1024</td>
<td>011010101</td>
<td>1024</td>
<td>0000001110100</td>
</tr>
<tr>
<td>1088</td>
<td>011010110</td>
<td>1088</td>
<td>0000001110101</td>
</tr>
<tr>
<td>1152</td>
<td>011010111</td>
<td>1152</td>
<td>0000001111010</td>
</tr>
<tr>
<td>1216</td>
<td>011011000</td>
<td>1216</td>
<td>0000001111011</td>
</tr>
<tr>
<td>1280</td>
<td>011011001</td>
<td>1280</td>
<td>0000001110100</td>
</tr>
<tr>
<td>1344</td>
<td>011011010</td>
<td>1344</td>
<td>0000001110101</td>
</tr>
<tr>
<td>1408</td>
<td>011011011</td>
<td>1408</td>
<td>0000001111010</td>
</tr>
<tr>
<td>1472</td>
<td>011011100</td>
<td>1472</td>
<td>0000001110101</td>
</tr>
<tr>
<td>1536</td>
<td>011011101</td>
<td>1536</td>
<td>0000001111011</td>
</tr>
<tr>
<td>1600</td>
<td>011011110</td>
<td>1600</td>
<td>0000001110010</td>
</tr>
<tr>
<td>1664</td>
<td>011011111</td>
<td>1664</td>
<td>0000001110011</td>
</tr>
<tr>
<td>1728</td>
<td>011011111</td>
<td>1728</td>
<td>0000001111100</td>
</tr>
</tbody>
</table>

Table 3.2 Makeup codes from 64 to 1728[11].

<table>
<thead>
<tr>
<th>Pixel run (white and black)</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1792</td>
<td>00000001000</td>
</tr>
<tr>
<td>1856</td>
<td>00000001100</td>
</tr>
<tr>
<td>1920</td>
<td>00000001101</td>
</tr>
<tr>
<td>1984</td>
<td>000000010010</td>
</tr>
<tr>
<td>2048</td>
<td>000000010111</td>
</tr>
<tr>
<td>2112</td>
<td>000000010100</td>
</tr>
<tr>
<td>2176</td>
<td>000000010101</td>
</tr>
<tr>
<td>2240</td>
<td>000000010110</td>
</tr>
<tr>
<td>2304</td>
<td>000000010111</td>
</tr>
<tr>
<td>2368</td>
<td>000000011100</td>
</tr>
<tr>
<td>2432</td>
<td>000000011101</td>
</tr>
<tr>
<td>2496</td>
<td>000000011110</td>
</tr>
<tr>
<td>2560</td>
<td>000000011111</td>
</tr>
</tbody>
</table>

Table 3.3 Makeup codes from 1792 till 2560[11].
Codes for modes of coding

<table>
<thead>
<tr>
<th>Mode</th>
<th>Elements to be coded</th>
<th>Notion</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass.</td>
<td>b1,b2</td>
<td>P</td>
<td>001</td>
</tr>
<tr>
<td>Horizontal</td>
<td>a0a1, a1a2</td>
<td>H</td>
<td>001+MH(a0a1)+MH(a1a2)</td>
</tr>
<tr>
<td>Vertical</td>
<td>a1b1 = 0</td>
<td>V(0)</td>
<td>1</td>
</tr>
<tr>
<td>Vertical</td>
<td>a1b1 = 1</td>
<td>VR(1)</td>
<td>011</td>
</tr>
<tr>
<td>Vertical</td>
<td>a1b1 = 2</td>
<td>VR(2)</td>
<td>00011</td>
</tr>
<tr>
<td>Vertical</td>
<td>a1b1 = 3</td>
<td>VR(3)</td>
<td>0000011</td>
</tr>
<tr>
<td>Vertical</td>
<td>a1b1 = -1</td>
<td>VL(1)</td>
<td>010</td>
</tr>
<tr>
<td>Vertical</td>
<td>a1b1 = -2</td>
<td>VL(2)</td>
<td>000010</td>
</tr>
<tr>
<td>Vertical</td>
<td>a1b1 = -3</td>
<td>VL(3)</td>
<td>0000010</td>
</tr>
<tr>
<td>Extension</td>
<td></td>
<td></td>
<td>0000001xxx</td>
</tr>
</tbody>
</table>

The values of a1b1 in vertical mode i.e. 1, 2 and three gives a1 to be at the right of b1 while -1, -2, and -3 shows that a1 is to the left of b1.

Table 3.4 Pass, horizontal and vertical mode codes [11].

When a page is encoded with G42D it will have pass, vertical and horizontal modes of bit patterns. These pixel runs represents the changing elements between the reference and the scan line. Horizontal mode is used to represent the pixel between a0a1 and a1a2. Thus it is inefficient in comparison to pass and vertical mode because they take one pixel run while horizontal mode takes two pixel runs and gives a longer run of pixels. However it will be ideal to say that horizontal mode that have large pixel runs should have lesser encoding.

This type of encoding can be used to compress any kind of image but they are specially designed for text images in any form. They have high efficiency of pass and vertical images. The coded bit patterns for pass and vertical mode have fewer bits as compared to their original bit runs.

3.4 Procedure

The complete procedure in according to all the above discussion is very easy to define. This coding procedure will give the steps followed to decide between pass, vertical and horizontal modes. These steps are given in detailed followed by the state flow diagram given in figure 3.5.
Figure 3.5 Coding steps for G42D [13].
• At the start of the page encoding a reference line is given to encode the first scan line.
• Then we have to give a scan line.
• Place a0 at the first changing element on the scan line.
• Detect a1, a2, b1 and b2.
• Take a decision to choose between the pass vertical and horizontal mode.
• After processing one of the modes replace a0 and detect the rest changing elements accordingly.
• Once the line is ended then go for taking the next scan line and encode the entire page accordingly.
• If the page has ended put the EOFB at the end of the page coupled with some pad bits.

3.5 TIFF structure

The TIFF structure for G42D has one IFD per image. The TIFF starts with a header followed by the image data and then the IFD. In this design we have used six tag entries given as follows

• Image length: Gives the length of image pixel. Represented by 0100H
• Image width: Gives the image pixel width. The hex value is 0101H
• Bits per sample: each sample has 8 bits and the tag is 0102H
• compression method: The tag is 0103H
• photometric interpretation: this is represent the type of image which is black and white (0106)
• Strip offsets: gives the offset to the next IFD with a tag 0111H.

Each tag entry has four different fields,

• The first two bytes identifies the tag for each of the IFD above.
• The next two bytes gives the type field for each IFD. Each IFD is a short type given with a 0003H value. The strip offset has a type field with long type and the value 0004H, which is 4 byte.
• Byte 4 to 11 inside an IFD entry represents the value or count of the described type in the first two bytes.
• This is the offset of the tag with an even number anywhere in the file.

The TIFF file is ended with an EOB. This EOB is given with eight 0 bytes. However this TIFF structure is given as follow in figure 3.6.
3.6 Tips for designing

There are some general guidelines for CCITT coding that needs to be followed,

- The model should be aware of the number of pixels in a scan line before decoding. Any scan line decoding to greater or lesser than the expected scan line are discarded.
- The encoding should not always be an 8 bit sample. The encoder should not have FILL bit padding always.
- The encoder should not always assign a 0 to white and 1 to black thus an opposite decoding is required.
- RTC or EOFB may occur before the end of the page encoding which means that the remaining scan lines are all white.
- If the scan line count completes and end of line does not occur then a warning should be issued by the decoder.
- Ignore any number of bits after the EOFB or RTC.
Chapter 4
Case study

4.1 CCITT group 4 2-D encoding

Consider the following image with black and white pixels.

The pixel data of this 16 x 5 image will be compressed using CCITT group4 2-dimensional TIFF. This process will be carried out in the following steps.

The first reference line for the first scan line is an imaginary line with all white pixels. The first two lines can be given as follow

\[
\begin{array}{cccc}
\text{Imaginary line} & & & \text{b1b2} \\
\hline
\text{Scan line 1} & a0 & a1 & a2 \\
\hline
\end{array}
\]

\[
\begin{array}{cccc}
\text{Imaginary line} & & & \text{b1b2} \\
\hline
\text{Scan line 1} & a0 & a1 & a2 \\
\hline
\end{array}
\]

Figure 4.2 Coding of scan line 1.

Hence this can be given as H4b4w. For the next shift of a0 all the three changing elements occur at the end of the scan line so a1 and b1 are at a distance of 0 pixels so it is vertical mode with V(1). So the complete code for this scan line can be given as
Coding the next line, the first scan line act as a reference line.

As both the reference and the scan line are identical so it means that b1 will always be at a 0 pixel distance from a1 hence a vertical mode of coding with V(0) hence the a0 will be shifted five times in the scan lines and other changing elements will change accordingly hence the line is coded as

\[
\text{scan line 2} = V(0)V(0)V(0)V(0)V(0)
\]

\[
\text{Scan line 2} = 11111
\]

Similar is the case with scan line 3 and it is coded as

\[
\text{scan line 3} = V(0)V(0)V(0)V(0)V(0)
\]

\[
\text{Scan line 3} = 11111
\]
In the first figure above a₁ is to the left of b₁ with a one pixel distance. Hence it will be coded as VL (1) a₀ will be shifted to a₁ as given below. It’s given as V(0) as a₁ appear at a one pixel distance b₁. Then after shifting a₀ it is vertical mode with V(0) and the next shift also gives a V(0) in similar fashion.

Hence the line is coded as

\[
\text{scan line } 4 = VL(1) V(0) V(0) V(0)
\]

\[
\text{Scan line } 4 = 011111
\]

Figure 4.5 Scan line 5.

The first is a pass mode followed with a one pixel a₁ to the left and then with a one pixel to the right of b₁ giving a vertical mode with VL (1) and then with V (0). Hence the line is coded as

\[
\text{Scan line } 5 = 0111
\]

Hence the complete image can be given as

Scan line 1 = 001100011001101110111
Scan line 2 = 11111
Scan line 3 = 11111
Scan line 4 = 011111
Scan line 5 = 0111
The complete image can be given as

\[
001100011001101111111111110111110111
\]

The image data is followed with an EOBF code hence the complete image can be given as hexadecimal

\[
31 9B BF FE FB 80 01 01
\]
4.2 TIFF Structure

The structure is in the following order

- Header
- Image data (CCITT encoding)
- IFD
- EOB

Header

The header has the following bytes.

- Byte 0-1: 4D4D H for big Endian.
- Byte 2-3: 002A H for CCITT encoding.
- Byte 4 to 7: 0008 offset to the IFD.

\[\text{Header} \quad 4D\ 4D\ 00\ 2A\ 00\ 08\]

IFD

There is only one IFD having the following tags with their particular 12 bytes.

- Image width:
  01 00 (tag for identifying the width IFD entry)
  00 03 (short type of bytes)
  00 00 00 01 (count of the tag in IFD)
  00 0F (width of the image in pixels)
  00 00 (offset)
- Image length: 01 01 00 03 00 00 00 01 00 05 00 00.
  The ninth and tenth byte gives the length of the image.
- Bits per sample: 01 02 00 03 00 00 00 01 00 01 00 00.
  The ninth and tenth byte shows that bits per sample are one.
- Compression method: 0103 00 00 03 00 00 00 01 00 04 00 00.
  A 04 on the tenth byte identify CCITT group4 2-D encoding.
- Photometric Interpretation: 01 06 00 03 00 00 00 01 00 01 00 00.
  The tenth byte is 01 which is for compressed.
- Strip offset: 01 11 00 04 00 00 00 01 00 00 00 08
- EOB: 00 00 00 00
4.3 Complete CCITT TIFF image

A complete CCITT group 4 2-D TIFF is given as

- The first 8 bytes is the header.
- The next 8 bytes is the image.
- Followed with an offset of the IFD.
- The next 12 x 6 are the IFD entries.
- The last 4 bytes is the EOB.

```
4D 4D 00 2A 00 00 00 08
31 9B BF FE FB 80 01 01
 00 06
 01 01 00 03 00 00 00 01 00 05 00 00
 01 02 00 03 00 00 00 01 00 01 00 00
 01 03 00 03 00 00 00 01 00 04 00 00
 01 06 00 03 00 00 00 01 00 01 00 00
 01 11 00 04 00 00 00 01 00 00 00 00
 00 00 00 00
```

This how the image file will be after compression. In this case as it was a simplified image to make the overall process run, the ratio between the image and the compressed image file should not be as small as required. However this compression ratio can be well explained in the chapters to follow.
Chapter 5  
Modeling

5.1 Matlab behavior model

CCITT G4 2D TIFF algorithm for simulation purpose is modeled in matlab. This model is the behavior model consisting of the following matlab files

- main.m
- function_fax4.m
- tables.m
- make_up_white.m
- make_up_black.m
- make_up.m

**main.m**

As the name suggests it is main block of the design. It takes an input image and gives out the TIFF image as a compressed image. It is only connected to function_fax4.m. It gives the image raw data to the function_fax4.m and takes the CCITT group 4 compressed data. The main.m then combines the TIFF file to this image and gives the output of the algorithm.

**function_fax4.m**

This file is the matlab description for the CCITT group 4 algorithm. The actions taken in this code are as follow.

- Adds a reference line to the image.
- Decides for the changing elements of the reference and the scan line.
- Decides between the three modes of coding i.e. pass vertical and horizontal modes.
- Add FILL bits to the compressed image data if required and sends the data with the size of the compressed data to the main file

**tables.m**

This file implements table 3.1 that is accessed by the function_fax4.m for coded values of the pixel runs.

**make_up_white.m & make_up_black.m**

Here Table 3.2 is implemented which provides codes values to function_fax4.m.
**make_up.m**

This is table 3.3 that hold the combined coded values for black and white and that is referred for their particular codes by function_fax4.m.

**5.2 Matlab structure**

The main structure of the matlab code is given the following figure 5.1. However the matlab codes are given in appendix D for main.m and function_fax4.m.

*Figure 5.1 Matlab coded structure.*
Chapter 6
Results

6.1 Test Images

I have developed these images with different dimension and sizes to test the depth of the matlab model. These different characteristics of these images are given in table 6.1.

Figure 6.1 Test images.
6.2 Compressed vs. BMP Images

The following table 6.1 gives the compressed data and the processing time of the algorithms.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Image 1</th>
<th>Image 2</th>
<th>Image 3</th>
<th>Image 4</th>
<th>Image 5</th>
<th>Image 6</th>
<th>Image 7</th>
<th>Image 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>50</td>
<td>312</td>
<td>140</td>
<td>291</td>
<td>312</td>
<td>364</td>
<td>385</td>
<td>300</td>
</tr>
<tr>
<td>Length</td>
<td>50</td>
<td>212</td>
<td>70</td>
<td>166</td>
<td>212</td>
<td>264</td>
<td>293</td>
<td>800</td>
</tr>
<tr>
<td>Original size (bytes)</td>
<td>3.59K</td>
<td>8.34K</td>
<td>10.6K</td>
<td>48.3K</td>
<td>65.6K</td>
<td>94.8K</td>
<td>112K</td>
<td>235K</td>
</tr>
<tr>
<td>Compressed size (byte)</td>
<td>112</td>
<td>314</td>
<td>220</td>
<td>252</td>
<td>306</td>
<td>154</td>
<td>186</td>
<td>320</td>
</tr>
<tr>
<td>Processing Time (milliseconds)</td>
<td>15.9</td>
<td>78</td>
<td>46.8</td>
<td>62.4</td>
<td>93.6</td>
<td>46.8</td>
<td>50</td>
<td>93.6</td>
</tr>
</tbody>
</table>

Table 6.1 Compressed images vs. original images.

![Original Versus Compressed size](image)

Figure 6.1 Original vs. compressed.
Here the processing time refers to the time taken by each image to compress on a machine with processing speed of 1.8 GHz. However this processing time has no resemblance to the time this algorithm will take on a hardware platform. But this is very useful in terms of comparing how this Matlab model will function in time domain. If you look at the results image 5 takes longer than image 6 which lesser in size. This happens because the changing elements in image 5 occur more than that of Image 6 which takes more time.

![Time and Compressed size](image)

*Figure 6.2 Time and compressed size.*
Chapter 7
Comparisons

The CCITT G4 TIFF is being compared with two groups of algorithm. One are the general algorithm that can compress monochrome, grayscale and RGB images. Whereas the second group are specified for black and white images compression. All these compression techniques uses TIFF as their file format. Table 7.1 gives the data for general algorithm. The first column with NONE refers to BMP images stored as a TIFF.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Image 1</th>
<th>Image 2</th>
<th>Image 3</th>
<th>Image 4</th>
<th>Image 5</th>
<th>Image 6</th>
<th>Image 7</th>
<th>Image 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>4.26K</td>
<td>8.40K</td>
<td>11.3K</td>
<td>48.9K</td>
<td>66.3K</td>
<td>96.6K</td>
<td>113K</td>
<td>236K</td>
</tr>
<tr>
<td>LZW</td>
<td>1.85K</td>
<td>1.16K</td>
<td>2.34K</td>
<td>2.28K</td>
<td>3.51K</td>
<td>3.66K</td>
<td>4.24K</td>
<td>6.89K</td>
</tr>
<tr>
<td>Pack Bits</td>
<td>1.92K</td>
<td>3.15K</td>
<td>2.69K</td>
<td>3.54K</td>
<td>3.87K</td>
<td>3.58K</td>
<td>4.30K</td>
<td>7.29K</td>
</tr>
<tr>
<td>JPEG</td>
<td>1.59K</td>
<td>15.9K</td>
<td>3.79K</td>
<td>8.86K</td>
<td>12.3K</td>
<td>7.01K</td>
<td>10.1K</td>
<td>19.8K</td>
</tr>
<tr>
<td>ZIP</td>
<td>1.77K</td>
<td>550</td>
<td>2.06K</td>
<td>3.03K</td>
<td>2.53K</td>
<td>2.30K</td>
<td>2.56K</td>
<td>3.47K</td>
</tr>
<tr>
<td>G42D</td>
<td>112</td>
<td>314</td>
<td>220</td>
<td>252</td>
<td>306</td>
<td>154</td>
<td>186</td>
<td>320</td>
</tr>
</tbody>
</table>

Table 7.1 generalized vs. G42D image size.

Figure 7.1 Generalized vs. G42D image size.
Other than G42D the rest of the compressed images are taken from Irfan View, which is an Image compression software using wide range of compression algorithms. The second table that is table 7.2 gives comparison between the algorithms for monochrome images.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Image 1</th>
<th>Image 2</th>
<th>Image 3</th>
<th>Image 4</th>
<th>Image 5</th>
<th>Image 6</th>
<th>Image 7</th>
<th>Image 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>G21D</td>
<td>398</td>
<td>2.12K</td>
<td>668</td>
<td>1.29K</td>
<td>1.58K</td>
<td>1.40K</td>
<td>1.67K</td>
<td>4.12</td>
</tr>
<tr>
<td>G32D</td>
<td>470</td>
<td>2.35K</td>
<td>768</td>
<td>1.50K</td>
<td>1.83K</td>
<td>1.76K</td>
<td>2.01K</td>
<td>5.17</td>
</tr>
<tr>
<td>G42D</td>
<td>112</td>
<td>314</td>
<td>220</td>
<td>252</td>
<td>306</td>
<td>154</td>
<td>186</td>
<td>320</td>
</tr>
</tbody>
</table>

Table 7.2 Comparison of monochrome image compression techniques in terms of size of the image.

![Monochrome Compression Algorithm](image.jpg)

Figure 7.2 Comparison of monochrome images compression technique in terms of size of the images.
Chapter 8

Conclusion

This thesis work is a comprehensive study of monochrome or black and white images. It almost describes the algorithmic growth from a very basic level. These algorithms are in place since the time when fax machine were evolved. These algorithms are also known as fax compression algorithms. The latest of all these algorithms is the CCITT group 4 2 dimensional TIFF, which is the focus of this study work.

The study was given a simulated form in the form of a matlab model. This matlab model was an encoder using the CCITT group 4 2 dimensional algorithm for compressing the raw image. This compressed image was then tagged with all the image dimensional information using the TIFF structure. The main purpose of using TIFF with such type of image compression techniques is to recover all the compressed data. The case study given in chapter 4 is complete manual compression of an image with all the TIFF tags.

The results taken from the Matlab model suggests that this algorithm has the ability to compress images of any size to much lesser than the original size. However the time taken by each image to compress in figure 6.2 is an approximation of the CPU time taken by the system for each algorithm. This CPU time is the smallest possible that this Matlab algorithm can take. But these result shows that the time taken to compress these images is quite small. Hence, targeting this algorithm onto hardware can make it more efficient and faster.

The comparison with various forms of image compression techniques suggests that CCITT has the best compressed size with an efficient time taken by this compression process. It has been compared to various generalized algorithm that are being used for different images type including RGB gray scale etc. They were also compared to other CCITT group algorithms i.e. group 2 and two dimensional group 3. But still the result suggests that group 4 is more reliable in terms of compression. However the complexity can be traded when Group 4 algorithms are implemented on hardware.

Nowadays these algorithms are very popular with sensor based smart camera. These smart cameras are usually working in a network with a wireless node or connected in some remote location with a smaller data rate line. Hence the compression ratios of these algorithms makes them a popular choice and have re evolved a technology that has been sidelined by so many newer and better algorithms that focus on more improved colored images.
This study provides a base for implementation of this algorithm on any flexible hardware platform such as FPGA, micro controller etc. which will provide faster compression of image received through any medium such as cameras and scanners.
References


## Appendix

### Appendix A Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIF</td>
<td>Graphic interface format</td>
</tr>
<tr>
<td>JPEG</td>
<td>Joint Photographic Experts Group</td>
</tr>
<tr>
<td>LZW</td>
<td>Lempel–Ziv–Welch</td>
</tr>
<tr>
<td>JPEG</td>
<td>Joint Photographic Experts Group</td>
</tr>
<tr>
<td>PNG</td>
<td>Portable Network Graphic</td>
</tr>
<tr>
<td>TIFF</td>
<td>Tagged image file format</td>
</tr>
<tr>
<td>CCITT</td>
<td>Comite Consultatif International Telegraphique et Telephonique</td>
</tr>
<tr>
<td>G42D</td>
<td>Group 4 two dimensional</td>
</tr>
<tr>
<td>G32D</td>
<td>Group 3 two dimensional</td>
</tr>
<tr>
<td>G31D</td>
<td>Group 3 one dimensional</td>
</tr>
<tr>
<td>G21D</td>
<td>Group 2 one dimensional</td>
</tr>
<tr>
<td>RLE</td>
<td>Run length encoding</td>
</tr>
<tr>
<td>BMP</td>
<td>Bit map</td>
</tr>
<tr>
<td>PCX</td>
<td>Paintbrush image format</td>
</tr>
<tr>
<td>LZ</td>
<td>Lempel Ziv</td>
</tr>
<tr>
<td>DPI</td>
<td>Dots per inch</td>
</tr>
<tr>
<td>PPI</td>
<td>Pixels per inch</td>
</tr>
<tr>
<td>ISDN</td>
<td>Integrated Services Digital Network</td>
</tr>
<tr>
<td>MH</td>
<td>Modified Huffman</td>
</tr>
<tr>
<td>MR</td>
<td>Modified read</td>
</tr>
<tr>
<td>MMR</td>
<td>Modified Modified read</td>
</tr>
<tr>
<td>EOL</td>
<td>End of line</td>
</tr>
<tr>
<td>MSLT</td>
<td>Minimum scan line test</td>
</tr>
<tr>
<td>IFH</td>
<td>Image file header</td>
</tr>
<tr>
<td>IFD</td>
<td>Image file directory</td>
</tr>
<tr>
<td>RTC</td>
<td>Return to control</td>
</tr>
<tr>
<td>EOFB</td>
<td>End of facsimile block</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunication Union</td>
</tr>
</tbody>
</table>
Appendix B Matlab Code

**Code for main.m**

```matlab
clear

time = CPUtime;
image = imread('8.bmp');
[buffer1 k] = function_fax4(image);
[l w] = size(image);
lhex = dec2hex(l);
if l > 256
    l1 = hex2dec(lhex(1));
    l2 = hex2dec([lhex(2) lhex(3)]);
end
whex = dec2hex(w);
if w > 256
    w1 = hex2dec(whex(1));
    w2 = hex2dec([whex(2) whex(3)]);
end
k = k+8;
buffer(1) = 77;buffer(2) = 77;buffer(3) = 00;buffer(4) =42;buffer(5) = 00;
buffer(6) = 00;
k_f = 0;
if (mod(k,2)~=0)
    k = k+1;
    k_f = 1;
end
if k<256
    buffer(7) = 0;
    buffer(8) = k;
else
    khex = dec2hex(k);
    k1 = hex2dec(khex(1));
    k2 = hex2dec([khex(2) khex(3)]);
end
fid = fopen('tes6666.tif','w');fwrite(fid,buffer);clear buffer
fwrite(fid,buffer1);clear buffer
```
if (k_f == 1)
    buffer(1) = 00;
    fwrite(fid,buffer);
    clear buffer
end

% Storing info about no. of directories
buffer(1) = 00; buffer(2) = 06;

% Storing info about Image width
buffer(3) = 01; buffer(4) = 00; buffer(5) = 00; buffer(6) = 03;
buffer(7) = 00; buffer(8) = 00; buffer(9) = 00; buffer(10) = 01;
if w > 256
    buffer(11) = w1;
    buffer(12) = w2;
else
    buffer(11) = 00;
    buffer(12) = w;
end
buffer(13) = 00; buffer(14) = 00; fwrite(fid,buffer);
clear buffer

% Storing info about Image Length
buffer(1) = 01; buffer(2) = 01; buffer(3) = 00; buffer(4) = 03;
buffer(5) = 00; buffer(6) = 00; buffer(7) = 00; buffer(8) = 01;
if l > 256
    buffer(9) = l1; buffer(10) = l2; else
    buffer(9) = 00; buffer(10) = l;
end
buffer(11) = 00; buffer(12) = 00; fwrite(fid,buffer); clear buffer

% Storing info about Bits per sample
buffer(1) = 01; buffer(2) = 02; buffer(3) = 00; buffer(4) = 03;
buffer(5) = 00; buffer(6) = 00; buffer(7) = 00; buffer(8) = 01;
buffer(9) = 00; buffer(10) = 01; buffer(11) = 00; buffer(12) = 00;
fwrite(fid,buffer); clear buffer

% Storing info about Compression Method
buffer(1) = 01; buffer(2) = 03; buffer(3) = 00; buffer(4) = 03;
buffer(5) = 00; buffer(6) = 00; buffer(7) = 00; buffer(8) = 01;
buffer(9) = 00; buffer(10) = 04; buffer(11) = 00; buffer(12) = 00;
fwrite(fid,buffer);clear buffer

% Storing info about Photometric Interpretation
buffer(1) = 01;buffer(2) = 06;buffer(3) = 00;buffer(4) = 03;
buffer(5) = 00;buffer(6) = 00;buffer(7) = 00;buffer(8) = 01;
buffer(9) = 00;buffer(10) = 01;buffer(11) = 00;buffer(12) = 00;
fwrite(fid,buffer);clear buffer

hex1 = ('011100040000000100000008');
k = 1;[tem2 max1] = size(hex1);
for j = 1:2:max1
    buffer(k) = hex2dec(hex1(j:j+1));
k = k+1;
end
fwrite(fid,buffer);
clear buffer

% Storing info Next IFD address all 0 means only one IFD above
buffer(1) = 00;buffer(2) = 00;buffer(3) = 00;buffer(4) = 00;
fwrite(fid,buffer);clear buffer

status = fclose(fid);

CPUtime - time

**Code for function_fax4.m**

function [decimal_code k] = function_fax4(image)
[Rows Cols] = size(image);run_length_black = 1;run_length_white = 1;
compressed = [];
k = 1;
for i=1:Rows
    for j = 1:Cols
        if image(i,j)== 0:image(i,j) = 0;else image(i,j) = 1;end
    end
end image;
row_scan = ones(1,Cols+2);
Cols = Cols + 2;
for i=1:Rows
    a0_f = 0; % To have a0a1-1 for the first time
    a2_f = 1; a0 = 1; a1 = Cols;a2 = Cols;b1 = Cols;
b2 = Cols;  row_reference = row_scan;
row_scan = [1 image(i,:) 1];
j = 2;
while (j <= Cols)
    ja = j;jb = j;aa_f = 1;
    bb_f = 1;while (ja <= Cols & & aa_f == 1)
        if (ja >= Cols)
            a1 = Cols;
            a2 = Cols;
            a2_f = 1;
        end
        if (ja <= Cols & & row_scan(1,ja) ~= row_scan(1,a0))
            a1 = ja;k = ja;
            while (k <= Cols & & row_scan(1,k) ~= row_scan(1,a0))
                k = k+1;ja = k;
            end
            a2 = k;
            aa_f = 0;
            a2_f = 0;
        else
            ja = ja +1;
        end
    end % end of while
    jb=jb;
    tes1_row_refer=row_reference(1,jb);
    tes1_row_scan=row_scan(1,a0);
    while (jb <= Cols & & bb_f == 1)
        if (jb >= Cols)
            b1 = Cols;b2 = Cols;end
        if (jb <= Cols & & row_reference(1,jb) ~= row_scan(1,a0) & & row_reference(1,jb) ~= row_reference(1,jb-1))
            k = jb;b1 = jb;
            while (k <= Cols & & row_reference(1,k) ~= row_scan(1,a0))
                k = k+1;
                jb = k;end
            b2 = k;bb_f = 0;
        else
            break; end
end
```
jb = jb + 1;
end
end
a1b1 = abs(a1-b1); a1a2 = abs(a1-a2); a0a1 = abs(a0-a1);
if (b2 < a1) % Pass Mode
    compressed = [compressed, 0 0 0 1];
    a0 = b2; j = a0 + 1; a0_f = 0;
elseif (a1b1 <= 3) % Vertical Mode
    % Vertical Mode
    compressed; if(a1b1 == 0)
        compressed = [compressed, 1];
        a0 = a1; j = a0 + 1; a0_f = 0;
    elseif (a1b1 == 1 && a1 < b1)
        compressed = [compressed, 0 1 0];
        a0 = a1; j = a0 + 1; a0_f = 0;
    elseif (a1b1 == 2 && a1 < b1)
        compressed = [compressed, 0 0 0 1 0];
        a0 = a1; j = a0 + 1; a0_f = 0;
    elseif (a1b1 == 3 && a1 < b1)
        compressed = [compressed, 0 0 0 0 1 0];
        a0 = a1; j = a0 + 1; a0_f = 0;
    elseif (a1b1 == 1 && a1 > b1)
        compressed = [compressed, 0 1 1];
        a0 = a1; j = a0 + 1; a0_f = 0;
    elseif (a1b1 == 2 && a1 > b1)
        compressed = [compressed, 0 0 0 0 1 1];
        a0 = a1; j = a0 + 1; a0_f = 0;
    elseif (a1b1 == 3 && a1 > b1)
        compressed = [compressed, 0 0 0 0 1 1];
        a0 = a1; j = a0 + 1; a0_f = 0;
    endelseif (a1b1 > 3) % Horizontal Mode
    if (row_scan(1,a0) == 1) run_length_black = a1a2 - a2_f;
        run_length_white = a0a1 - a0_f;
    else run_length_black = a0a1 - a0_f;
        run_length_white = a1a2 - a2_f;
    end
    a0_value = row_scan(1,a0);
```
a0 = a2;j = a0 +1;
horizontal_code = horizontal_coding(a0_value,run_length_black,run_length_white);
horizontal_code;compressed = [compressed, horizontal_code];
compressed; a0_f = 0; end ja = j;jb= j;compressed;end % End of while j % pause
end % end of for i
compressed = [compressed 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1];[R L] = size(compressed);
mod8 = mod(L,8);
% Filling 0 bits
if (mod8 == 1)
    compressed = [compressed 0 0 0 0 0 0 0];
elseif (mod8 == 2)
    compressed = [compressed 0 0 0 0 0 0 0];
elseif (mod8 == 3)
    compressed = [compressed 0 0 0 0 0 0 0];
elseif (mod8 == 4)
    compressed = [compressed 0 0 0 0 0 0 0];
elseif (mod8 == 5)
    compressed = [compressed 0 0 0 0 0];
elseif (mod8 == 6)
    compressed = [compressed 0 0 0 0 0];
elseif (mod8 == 7)
    compressed = [compressed 0 0 0 0 0];
end
compressed_binary = dec2bin(compressed);
[size1 l] = size
    (compressed_binary);k = 1;
for i = 1 :8:size1;
    a = compressed_binary(i:i + 7);
    a = a'; decimal_code(k) = bin2dec(a);
    k = k+1; end k = k-1;
hex = dec2hex(decimal_code);