Visible Image Watermarking

Digital Design and Logical Synthesis for Electric Computer Engineering (36113611)
Course Project

Digital High Level Design

Version 0.1

Revision History

Rev	Description	Done By	Date
0.1	Initial document	Yusef Ektilat, Roy Shor	2-Oct-2020
0.2			
0.3			

Contents

LIST OF FIGURES

1. INTRODUCTION

Watermarking is the process that embeds data called a watermark, a tag, or a label into a multimedia object, such as images, video, or paper, for copyright protection. A visible watermark is a secondary transparent image (see-through) overlaid into the primary image and appears slightly visible to the viewer. In this project we will implement a new architecture for visible watermark insertion algorithm explained later.

In general, visible watermarking has three goals:

1) Visible watermark should identify the ownership (copyright owner).

2) The quality of the primary image should be preserved.

3) The watermark should be difficult to remove.

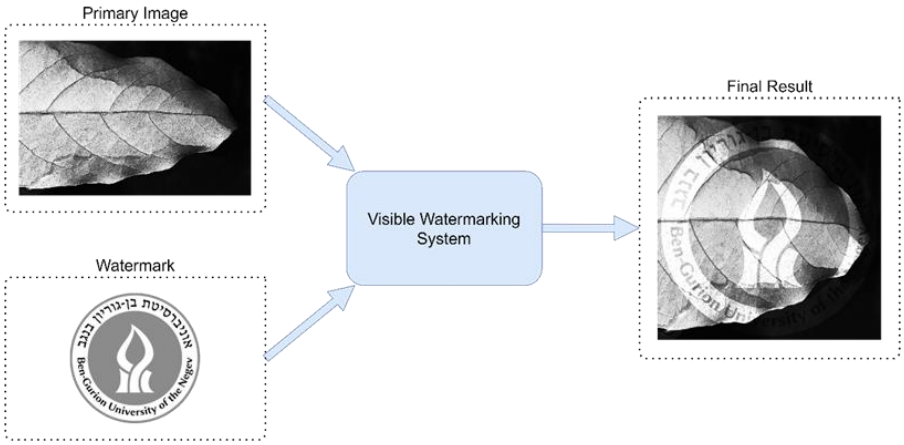


Figure 1: Image Watermarking

This algorithm operates in spatial domain of image data, where the pixel gray values are modified based on local and global statistics. The watermark insertion process is based on the following steps:

1) Primary image and the watermark image are both divided into smaller blocks of equal size 2D square matrix MxM (not necessarily same number of blocks). Assume i_k is the kth block of primary image, w_k is the kth block of watermark image, i_{W_k} is the kth block of watermarked image (result), and $I(x,y)$ denote a specific pixel of primary image, and $W(x,y)$ denote a specific pixel of watermark image.

2) Calculating for each block the scaling (α_k) and embedding (β_k) factors, which determine the extent of watermark insertion (see equations 1-4). At the same time, checking if the current block is edge or nonedge block by calculating the mean amplitude (G_{μ_k}) of current block (see equation 5) and compare it to a pre-defined threshold (B_{thr}).

3) => For edge block (when G_{μ_k} exceeds B_{thr}): $i_{W_k} = \alpha_{max} \cdot i_k + \beta_{min} \cdot w_k$

=> For nonedge block (when G_{μ_k} less than B_{thr}): $i_{W_k} = \alpha_k \cdot i_k + \beta_k \cdot w_k$

The algorithm equations:

$$Eqn.(1): \alpha_k = \alpha_{min} + \frac{(\alpha_{max} - \alpha_{min})}{\sigma_k} \cdot 2^{-(\mu_k - 0.5)^2}$$

$$Eqn.(2): \beta_k = \beta_{min} + \sigma_k \cdot (\beta_{max} - \beta_{min}) \cdot \left(1 - 2^{-(\mu_k - 0.5)^2}\right)$$

$$Eqn.(3): \mu_k = \frac{1}{M^2 \cdot (I_{white} + 1)} \cdot \sum_x \sum_y I(x,y)$$

$$Eqn.(4): \sigma_k = \frac{2}{M^2 \cdot (I_{white}+1)} \cdot \sum_x \sum_y \left| I(x,y) - \frac{I_{white}+1}{2} \right|$$

$$Eqn.(5): G_{\mu_k} = \frac{1}{M^2} \cdot \sum_x \sum_y |I(x,y) - I(x+1,y)| + |I(x,y) - I(x,y+1)|$$

Note: Assume that primary and watermark images are gray-scaled images stored in main memory as 2D square matrix each (N_pxN_p for primary image, N_wxN_w for watermark image), also $I_{white} = 255$ as default.

The data path (state machine) of this algorithm is shown in Fig. 2.

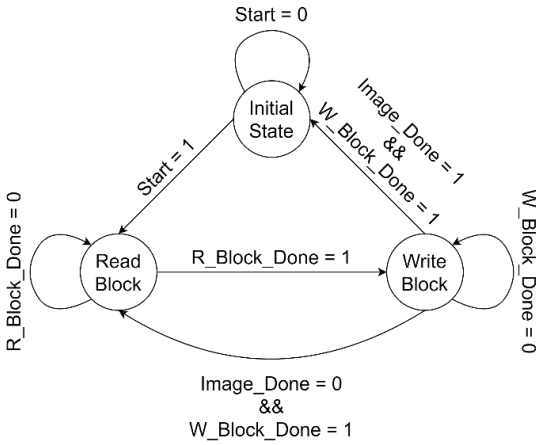


Figure 2: Data Path

In short, this architecture gets an input value of gray-scaled pixels from two images until filling the proper block for each one, process the data according to the given algorithm, write the block result to the output pixel by pixel, and repeats until full images processing, then outputs '1' when done, else the output remains '0'.

Our design is a peripheral part of an ARM Processor and uses APB bus protocols to communicate with it (Figure 2). The CPU can read and write a register bank inside the design (marked in red).

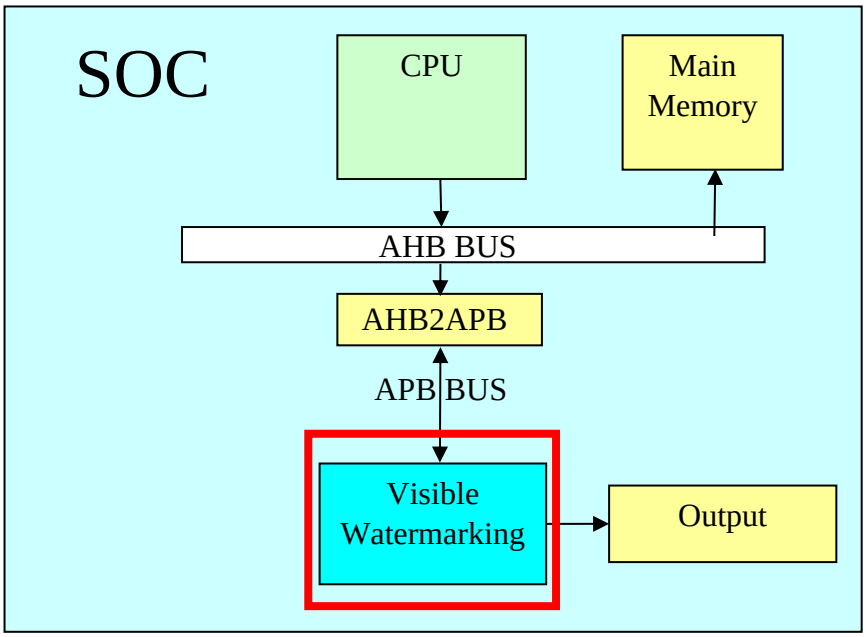


Figure 2: Top view of the SOC environment around the design

The design, named Visible_Watermarking, is controlled by the CPU which configures the design via APB bus and a register bank inside the design. CPU can write to the register bank and read its content.

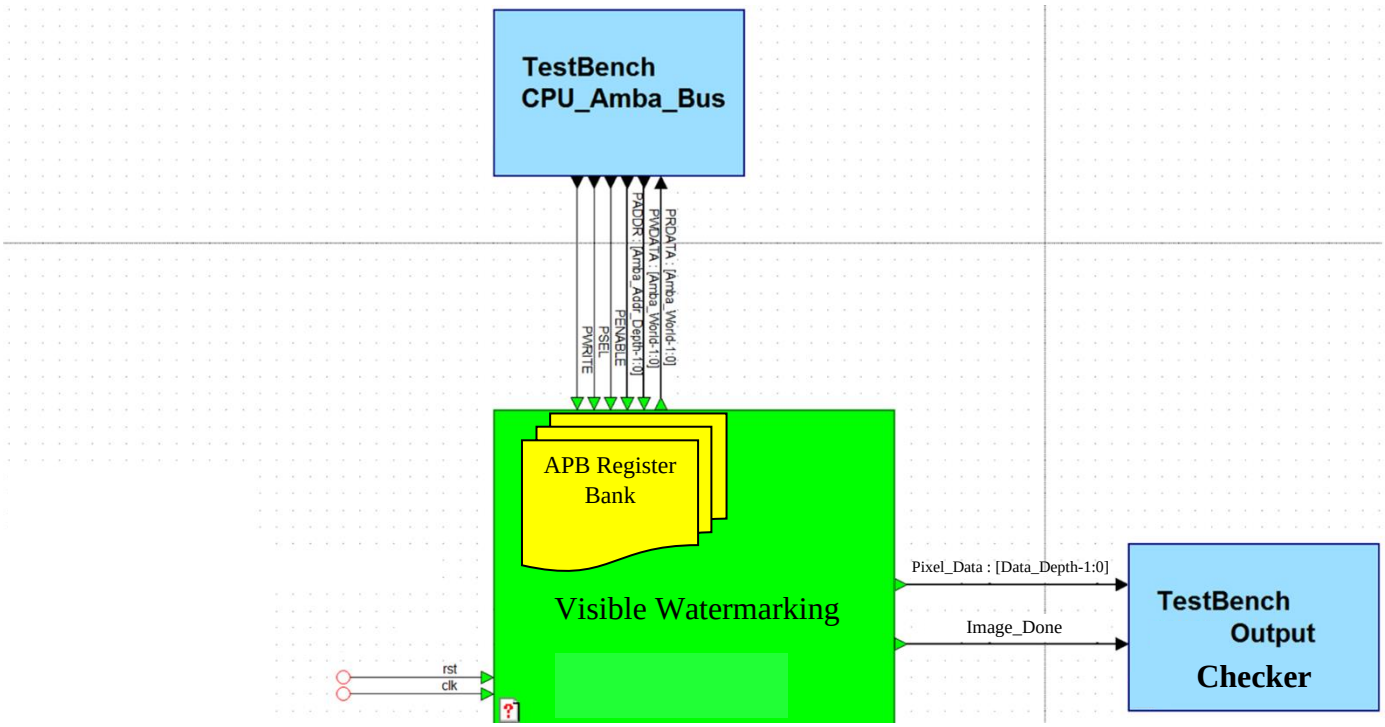


Figure 3: Top view of the environment around the design

2. SPECIFICATION

2.1 Functional Description

The design is a slave to the CPU master, which controls it via APB bridge and register files. The CPU sends the image data to the visible watermark insertion block and it gives back the image data of the result, and an output of ‘1’ when watermark insertion is completed and ‘0’ while it still in process.

2.2 Interface

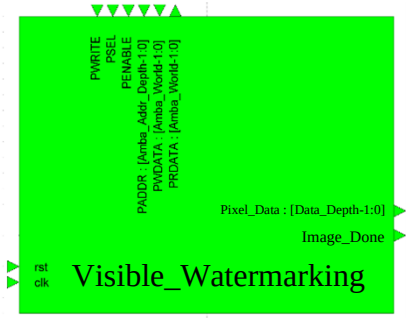


Figure 5: Top view of the block.

Name	Mode	Type	Bound	Comment
PADDR	input	wire	[Amba_Addr_Depth-1:0]	// APB Address Bus
PENABLE	input	wire		// APB Bus Enable/clk
PSEL	input	wire		// APB Bus Select
PWDATA	input	wire	[Amba_Word-1:0]	// APB Write Data Bus
PWRITE	input	wire		// APB Bus Write
clk	input	wire		system clock
rst	input	wire		// Reset active low
PRDATA	output	reg	[Amba_Word-1:0]	// APB Read Data Bus
Image_Done	output	reg		// State indicator (Output)
Pixel_Data	output	reg	[Data_Depth-1:0]	//Modified pixel (Output)
new_pixel	output	reg		// New Pixel Indicator

Table 1 : Block interface.

2.3 Parameters

Parameter Name	Range	Default values	Comments
Amba_Word	16,24,32	16	Part of the Amba standard at moodle site
Amba_Addr_Depth	20,24,32	20	Part of the Amba standard at moodle site
Data_Depth	8,16	8	Bit depth of the pixel

Table 2 : Parameters.

2.4 Register block and programming model:

The controller has a set of several registers at size Amba_Word bit registers. The registers are accessible via the APB interface. You should implement this interface according to the AMBA standard (APB section).

The registers are:

REGISTER NAME	ADDRESS OFFSET	COMMENTS	ACCESS TYPE
Control (CTRL)	0x00	Controls the design	CPU Read/Write, Visible_Watermarking Read only
WhitePixel	0x01	White pixel value	CPU Read/Write, Visible_Watermarking Read only
PrimarySize	0x02	Primary Image matrix rows/columns number	CPU Read/Write, Visible_Watermarking Read only
WatermarkSize	0x03	Watermark Image matrix rows/columns number	CPU Read/Write, Visible_Watermarking Read only
BlockSize	0x04	The small blocks matrix rows/columns number (M)	CPU Read/Write, Visible_Watermarking Read only
EdgeThreshold	0x05	Predefined Edge detection threshold	CPU Read/Write, Visible_Watermarking Read only
Amin	0x06	Scaling factor minimum percentage value	CPU Read/Write, Visible_Watermarking Read only
Amax	0x07	Scaling factor maximum percentage value	CPU Read/Write, Visible_Watermarking Read only
Bmin	0x08	Embedding factor minimum percentage value	CPU Read/Write, Visible_Watermarking Read only
Bmax	0x09	Embedding factor maximum percentage value	CPU Read/Write, Visible_Watermarking Read only
PrimaryPixel00	0x0A	Primary image (0,0) pixel value	CPU Read/Write, Visible_Watermarking Read only
⋮	⋮	⋮	⋮
PrimaryPixelNN	0x09+(N_p^2)h	Primary image (N_p,N_p) pixel value	CPU Read/Write, Visible_Watermarking Read only
WatermarkPixel00	0x0A+(N_p^2)h	Watermark image (0,0) pixel value	CPU Read/Write, Visible_Watermarking Read only
⋮	⋮	⋮	⋮
WatermarkPixelNN	0x09+($N_p^2 + N_w^2$)h	Watermark image (N_w,N_w) pixel value	CPU Read/Write, Visible_Watermarking Read only

REGISTER NAME	ADDRESS OFFSET	COMMENTS	ACCESS TYPE
⋮	⋮	Size of the register bank is set by Amba_Addr_Depth	⋮

Table 3 : Registers in the register bank.

2.4.1 CTRL register:

Bit 0 asserts the design to work. When start is '0' the design is turned off. Registers Amba_word-1:7, are unused. Address: 0x00; default 0.

bit number	Amba_word-1:1	0
Name	unused	start

2.4.2 WhitePixel register:

This register holds the default value of white pixel for defining the image pixel value scale (gray-scale). Address offset 0x01; default 255, minimum 1, maximum 255.

bit number	Amba_word-1:8	7:0
Name	unused	Iwhite

2.4.3 PrimarySize register:

This register holds the value of primary image's matrix rows/columns for defining the image matrix size and limits. Address offset 0x02; minimum 200, maximum 720.

bit number	Amba_word-1:10	9:0
Name	unused	Np

2.4.4 WatermarkSize register:

This register holds the value of watermark image's matrix rows/columns for defining the image matrix size and limits. Address offset 0x03; minimum 200, maximum 720.

bit number	Amba_word-1:10	9:0
Name	unused	Nw

2.4.5 BlockSize register:

This register holds the value of the divided block's matrix rows/columns for defining the image matrix size and limits. Address offset 0x04; minimum 1, maximum Np/10.

bit number	Amba_word-1:10	9:0
Name	unused	M

2.4.6 EdgeThreshold register:

This register holds the value of the predefined threshold for detecting edge blocks. Address offset 0x05; minimum 1, maximum 20.

bit number	Amba_word-1:8	7:0
Name	unused	Bthr

2.4.7 Amin register:

This register holds the minimum value of the Scaling factor. Address offset 0x06; minimum 80, maximum Amax. The real factor should be between 0.8-Amax, therefore, it needs to be divided by 100 in the design.

bit number	Amba_word-1:7	6:0
Name	unused	Amin

2.4.8 Amax register:

This register holds the maximum value of the Scaling factor. Address offset 0x07; minimum 90, maximum 99. The real factor value should be between 0.9-0.99, therefore, it needs to be divided by 100 in the design.

bit number	Amba_word-1:7	6:0
Name	unused	Amax

2.4.9 Bmin register:

This register holds the minimum value of the Embedding factor. Address offset 0x08; minimum 20, maximum Bmax. The real factor value should be between 0.2-Bmax, therefore, it needs to be divided by 100 in the design.

bit number	Amba_word-1:5	5:0
Name	unused	Bmin

2.4.10 Bmax register:

This register holds the maximum value of the Embedding factor. Address offset 0x09; minimum 30, maximum 40. The real factor value should be between 0.3-0.4, therefore, it needs to be divided by 100 in the design.

bit number	Amba_word-1:5	5:0
Name	unused	Bmax

2.4.11 PrimaryPixelXY registers:

These registers hold the value of the Primary image pixel at position (x,y). Address offset of first pixel is 0x0A and for the last pixel is 0x0A+$\left(N_p^2\right)_h$; minimum 1, maximum 255.

bit number	Amba_word-1:8	7:0
Name	unused	Pixel Value

2.4.12 WatermarkPixelXY registers:

These registers hold the value of the Watermark image pixel at position (x,y). Address offset of first pixel is 0x0B+$\left(N_p^2\right)_h$ and for the last pixel is 0x0B+$\left(N_p^2\right)_h$+$(N_w^2)_h$; minimum 1, maximum 255.

bit number	Amba_word-1:8	7:0
Name	unused	Pixel Value

2.5 Design Objectives

Set **a priority and** plan your design accordingly:

- Smallest power consumption and smallest design area.
- Fastest performance.
- Best Precision.

2.6 Project Assumptions, Constraints and Work flow

2.6.1 Constraints

1) Write/read process only when APB protocol is correct, otherwise unexpected results may happen. Visible_Watermarking works only when start is set to '1' by the CPU.
2) CPU and design can read the contents of the register bank simultaneously.
3) For each register, in the bank, only CPU has write permission.

2.6.2 Workflow

1) CPU writes pixel data values and other required registers values to the register bank via APB bus.
2) CPU writes start set to '1'.
3) Apply given algorithm for watermark insertion to primary image and outputs each pixel result.
4) Visible_Watermarking design outputs ‘1’ when full image has been processed.
5) Go to stage 3 unless CPU writes start set to '0'.

Note: Check Figure 2 for clearer workflow and data path.

3. SUBMISSION REQUIREMENTS

For the first (design) part of the project you have to submit the following:

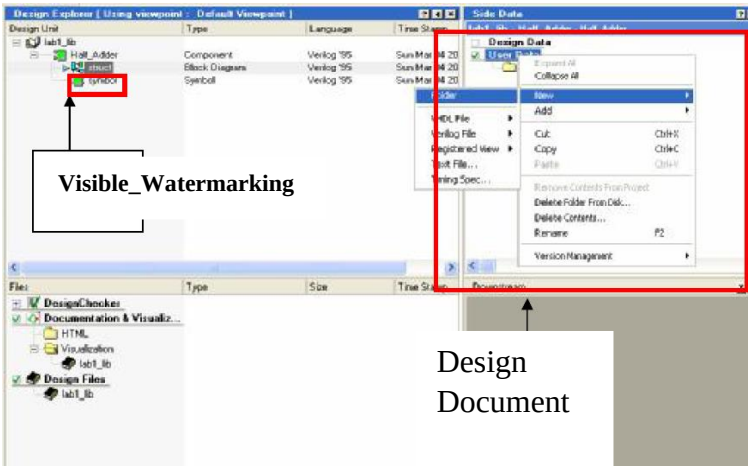
1) HDL Designer library including project file, hdl and hds libraries. Top level module should be named **Visible_Watermarking**, and the file should be named **Visible_Watermarking.v**. Verilog 2005 syntax

2) **Short Design Guide** (about 5-10 pages) containing the following (in pdf format):

   - Top level Block Diagram
   - Functionality description of your design.
   - Flow chart\State machine diagrams.
   - Explain which rules you waved in design checker.

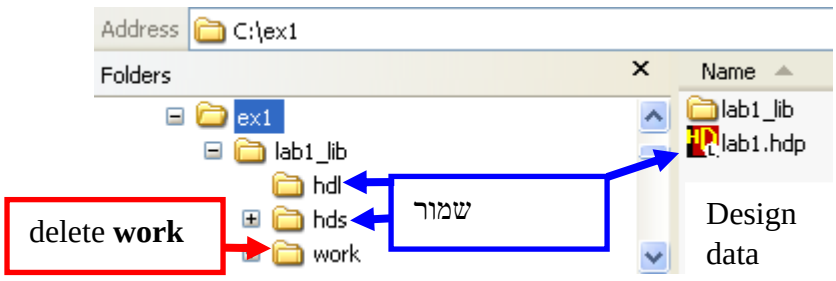
Use the template provided in the moodle→מעבדות→part1 template:
Digital Block_02_Definition_Template.docx

1) Document should be attached in the **Visible_Watermarking** design in HDL designer using side data library



2) **Delete** the **work** library

3) Compress **Project file**, **hds** and **hdl** libraries into a ZIP file named ***<ID1>_<ID2>.zip***, where ID1 and ID2 are the ID (*teudat zeut*) numbers of the submitters.



4) All the library files should be zipped to a file named **id1_id2.zip** and submitted to moodle.

5) Both Students Should Submit Their work to moodle.

SUBMISSION DATE: 28/11/2020 AT 23:55

4. PROJECT GRADE EVALUATION

This part is **30 points** **from the project grade.** Preliminary checks must pass before your work is checked. **Failure in the preliminary checks means grade of 0 for this part of the project.**

Task	Description
Code compiles	Code compiles without compilation error in QuestaSim and Design Compiler

Table 4 : Perliminary tests

After passing the preliminary tests the system's performance will be checked with additional tests.

grade	Task	Description
40%	RTL code quality	**Grade given by Design Checker on the customized course policy (DO-254 +RMM).** All rules considered except the ones use chose to exclude and explained them well in your document.
30%	Document	Block diagrams, flow diagram, FSM machine diagrams **easy to read and understand your design** (use **Digital Block_02_Definition_Template**).
30%	Reuse	Design works and compiles under all the ranges of the parameters from Table 2.

Table 5 : Grading policy

Late submission will degrade the project score by 5*days

5. APPENDIX

5.1 Terminology

5.2 References

[1] Amba standard moodle→Amba Specifications

5.3 MATLAB Code for Golden Model (also usable for creating input files)

```
clc;
clear;

for k=1:1:10

    filename = sprintf('primary_image_%d.jpg',k);
    I = imread(filename);
    Nx = randi([200 720]);
    I = imresize(I,[Nx Nx]);
    W = imread('watermark_image.png');

    I = rgb2gray(I);
    W = rgb2gray(W);

    Amax_min = 0.90;
    Amax_max = 0.99;
    Amax = Amax_min+rand(1)*(Amax_max-Amax_min);
    Amax = round(Amax,2);

    Amin_min = 0.80;
    Amin_max = Amax;
    Amin = Amin_min+rand(1)*(Amin_max-Amin_min);
    Amin = round(Amin,2);

    Bmax_min = 0.30;
    Bmax_max = 0.40;
    Bmax = Bmax_min+rand(1)*(Bmax_max-Bmax_min);
    Bmax = round(Bmax,2);

    Bmin_min = 0.20;
    Bmin_max = Bmax;
    Bmin = Bmin_min+rand(1)*(Bmin_max-Bmin_min);
    Bmin = round(Bmin,2);

    Bthr = randi([0 20]);

    Np = (size(I));
    Np = Np(1);
    Nw = (size(W));
    Nw = Nw(1);
    N  = Np;

    M = randi([1 round(N/10)]);
    while ((mod(N,M)))
        M = randi([1 round(N/10)]);
    end

    Iw = uint8(zeros(N,N));
    Iwk = uint8(zeros(M,M));

    I = double(imresize(I,[N N]));
    I_new = I(:);
    Ik = double(zeros(M,M));
    W = double(imresize(W,[N N]));
    W_new = W(:);
    Wk = double(zeros(M,M));
```

```
    sigma = 0.0;
    sigmau = 0.0;
    sigmas = 0.0;
    Guk = 0.0;
    uk = 0.0;
    sk = 0.0;
    ak = 0.0;
    bk = 0.0;

    for r=1:1:(N/M)
        for c=1:1:(N/M)
            %%%% 1 - Fill Block
            for rows=1:1:M
               for cols=1:1:M
                   Ik(rows,cols) = I_new(rows + (cols-1)*N + (r-1)*M + (c-1)*M*N);
                   Wk(rows,cols) = W_new(rows + (cols-1)*N + (r-1)*M + (c-1)*M*N);
               end
            end
            %%%% 2 - Edge Detector
            for rows=1:1:M
               for cols=1:1:M
                   if ((rows == M) && (cols == M))
                       sigma = sigma + 2*Ik(rows,cols);
                   elseif (rows == M)
                       sigma = sigma + Ik(rows,cols) + abs(Ik(rows,cols) - Ik(rows,cols+1));
                   elseif (cols == M)
                       sigma = sigma + abs(Ik(rows,cols) - Ik(rows+1,cols)) + Ik(rows,cols);
                   else
                       sigma = sigma + abs(Ik(rows,cols) - Ik(rows+1,cols)) + abs(Ik(rows,cols) -
Ik(rows,cols+1));
                   end
               end
            end
            Guk = round(sigma / (M*M));
            %%%% 3 - Parameters Calculator
            if (Guk >= Bthr)
                for rows=1:1:M
                    for cols=1:1:M
                        Iw((r-1)*M + rows,(c-1)*M + cols) = round(Amax*Ik(rows,cols) + Bmin*Wk(rows,cols));
                    end
                end
            else % (guk < bthr)
                for rows=1:1:M
                    for cols=1:1:M
                        sigmau = sigmau + Ik(rows,cols);
                        sigmas = sigmas + abs(Ik(rows,cols) - 128);
                    end
                end
                uk = round(sigmau / (M*M*256));
                sk = round((2*sigmas) / (M*M*256));
                ak = round(Amin + ((Amax - Amin)*(2^(-(uk-0.5)^2)))/(sk));
                bk = round(Bmin + (sk)*((Bmax - Bmin)*(1-2^(-(uk-0.5)^2))));
                for rows=1:1:M
                    for cols=1:1:M
                        Iw((r-1)*M + rows,(c-1)*M + cols) = round(ak*Ik(rows,cols) + bk*Wk(rows,cols));
                    end
                end
            end
        end
    end

    figure;
    imshow(Iw);
    imagname = sprintf('watermarked_image(result)_%d.jpg',k);
    imwrite(Iw,imagname);

    % Primary Image
    outname = sprintf('primary_image_%d.txt',k);
    fid = fopen(outname, 'wt');
    fprintf(fid, '%d\n', N); % decimal writing of image row/cols size
    fprintf(fid, '%d\n', I); % decimal writing of pixels value
    disp('Text file write done');disp(' ');
    fclose(fid);

    % Watermark Image
    outname = sprintf('watermark_image_%d.txt',k);
    fid = fopen(outname, 'wt');
    fprintf(fid, '%d\n', N); % decimal writing of image row/cols size
    fprintf(fid, '%d\n', W); % decimal writing of pixels value
```

```
    disp('Text file write done');disp(' ');
    fclose(fid);

    % Watermarked Image (Result)
    outname = sprintf('watermarked_image(result)_%d.txt',k);
    fid = fopen(outname, 'wt');
    fprintf(fid, '%d\n', N); % decimal writing of image row/cols size
    fprintf(fid, '%d\n', Iw); % decimal writing of pixels value
    disp('Text file write done');disp(' ');
    fclose(fid);

    % Paramters Random Values
    outname = sprintf('parameters_random_value_%d.txt',k);
    fid_ai = fopen(outname, 'wt');
    fprintf(fid_ai, '%d %d %d %d %d %d\n', M, Bthr, round(Amin*100), round(Amax*100), round(Bmin*100),
round(Bmax*100)); % decimal writing of parameters value
    disp('Text file write done');disp(' ');
    fclose(fid_ai);

end
```