Visible Image Watermarking	**Digital Design and Logical Synthesis for Electric Computer Engineering** **(36113611)** **Course Project** **Digital High-Level Design** **Version 0.1**

Revision Log

Rev	**Description**	**Done By**	**Date**
0.1	Initial document	Ariel Moshe, Amit Nagar Halevy	30/11/2020
0.2			
0.3			

Table of Content

1. LIST OF FIGURES

2. LIST OF TABLES

3. 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.

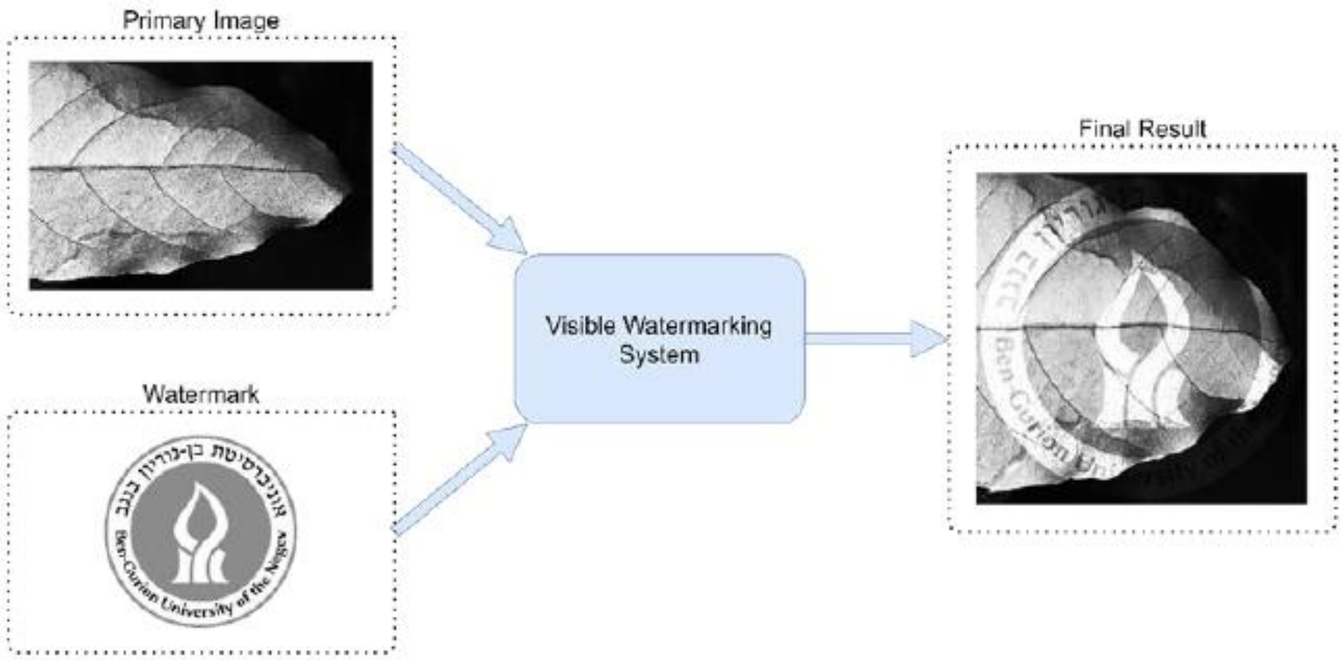


Figure 1: Image Watermarking

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 ik 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 nonedged 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 nonedged block (when G_{μ_k} less than B_{thr}): $i_{W_k} = \alpha_{\mathrm{k}} \cdot i_k + \beta_k \cdot w_k$

The rest of the equations are detailed later.

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

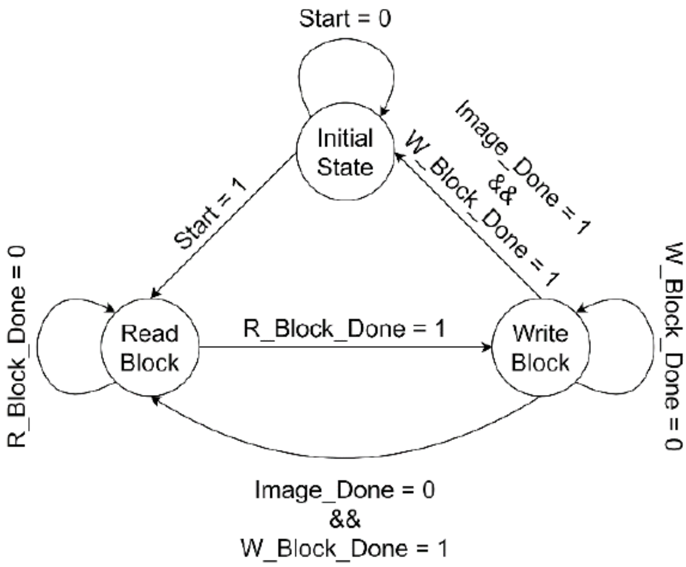


Figure 2: Date Path

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 3). The CPU can read and write a register bank inside the design (marked in red).

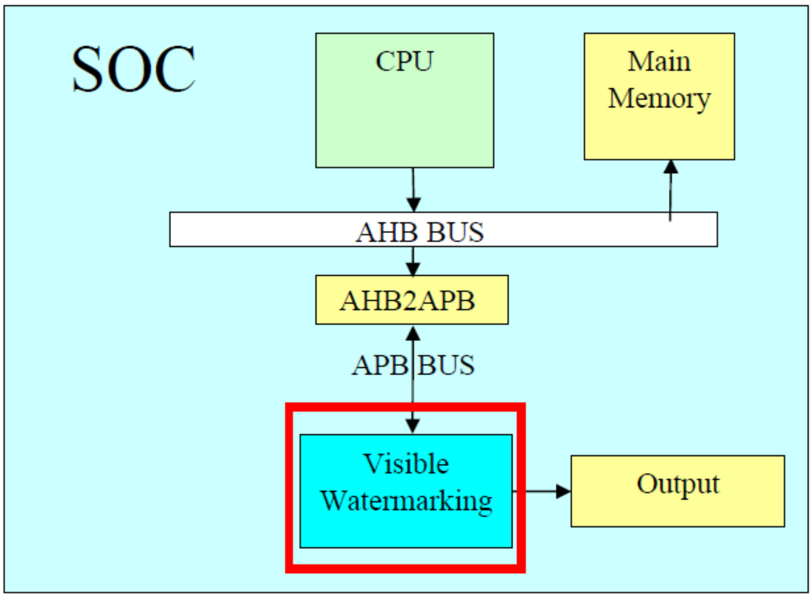


Figure 3: Top view of the SOC environment around the design

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.

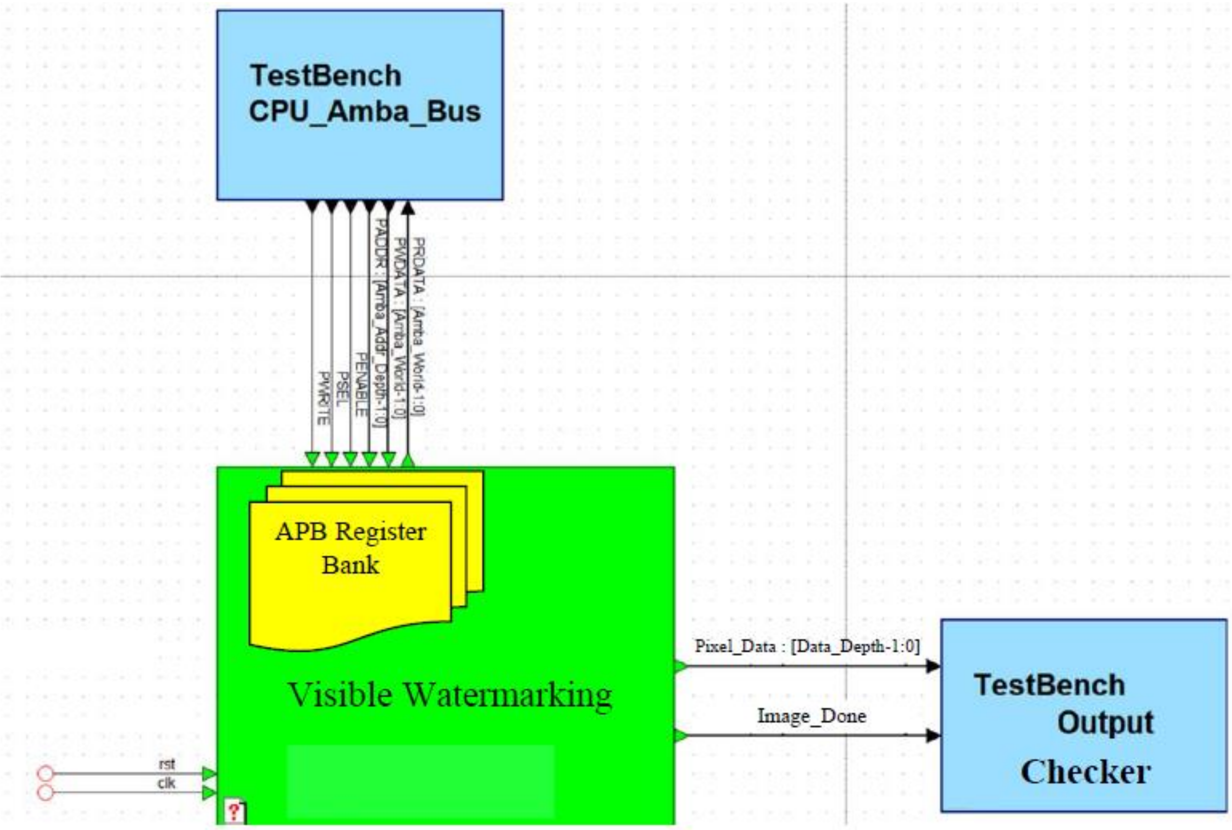


Figure 4: Top view of the environment around the design

3.1 Project Assumptions, Constraints and Work flow

3.1.1 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.

3.1.2 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'.

4. BLOCKS FUNCTIONAL DESCRIPTIONS

4.1 Top Level - Visible Watermarking

4.1.1 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.

The Top-Level Visible Watermarking consist of three main modules as following:

1) Control and Registers – saves information about the images and initial parameters for calculation and send them to the Equation implementation.
2) Equation implementation – calculate the output block using the equations, given the images and initial parameters and output watermarked image and send it to the Block to Pixel block.
3) Block to Pixel – receiving watermarked block and output it pixel by pixel.

All the modules work in parallel if possible, using control signals (e.g. Ready, Block_Done) that synchronize the modules.

4.1.2 Interface

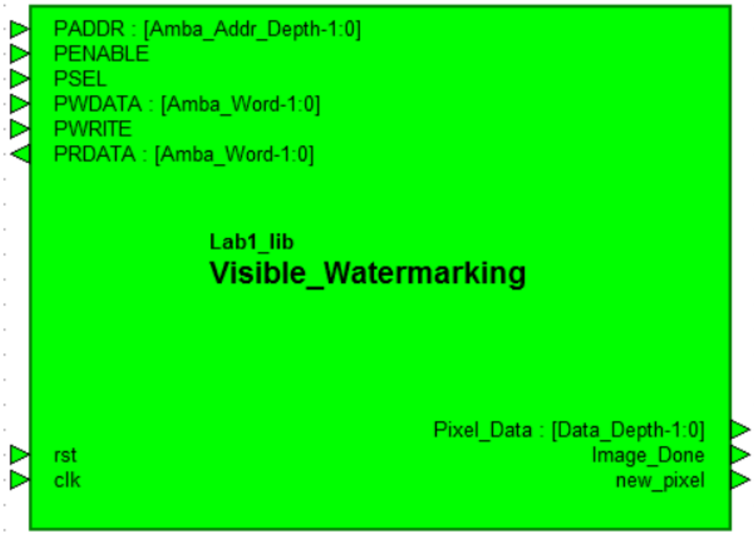


Figure 5: Top Level - Visible Watermarking interface.

	A	B	C	D	E	F	G	H
	Group	Name	Mode	Type	Signed	Bounds	Value	Comment
1		rst	input	wire				// Reset active low
2		clk	input	wire				//system clock
3		PWRITE	input	wire				// APB Bus Write
4		PWDATA	input	wire		[Amba_Word-1:0]		// APB Write Data Bus
5		PSEL	input	wire				// APB Bus Select
6		PENABLE	input	wire				// APB Bus Enable/clk
7		PADDR	input	wire		[Amba_Addr_Depth-1:0]		// APB Address Bus
8		new_pixel	output	reg				//New pixel indicator
9		Pixel_Data	output	reg		[Data_Depth-1:0]		//Modified pixel (Output)
10		PRDATA	output	reg		[Amba_Word-1:0]		//APB Read Data Bus
11		Image_Done	output	reg				//State indicator (Output)

Table 1: Top Level - Visible Watermarking interface.

	A	B	C	D
	Group	Name	Value	Comment
1		Amba_Addr_Depth	20	//Part of the Amba standard at Moodle site; Range - 20,24,32
2		Amba_Word	16	//Part of the Amba standard at Moodle site; Range - 16,24,32
3		Data_Depth	8	//Bit depth of the pixel; Range - 8,16

Table 2: Top Level - Visible Watermarking parameters.

4.1.3 Block Diagram

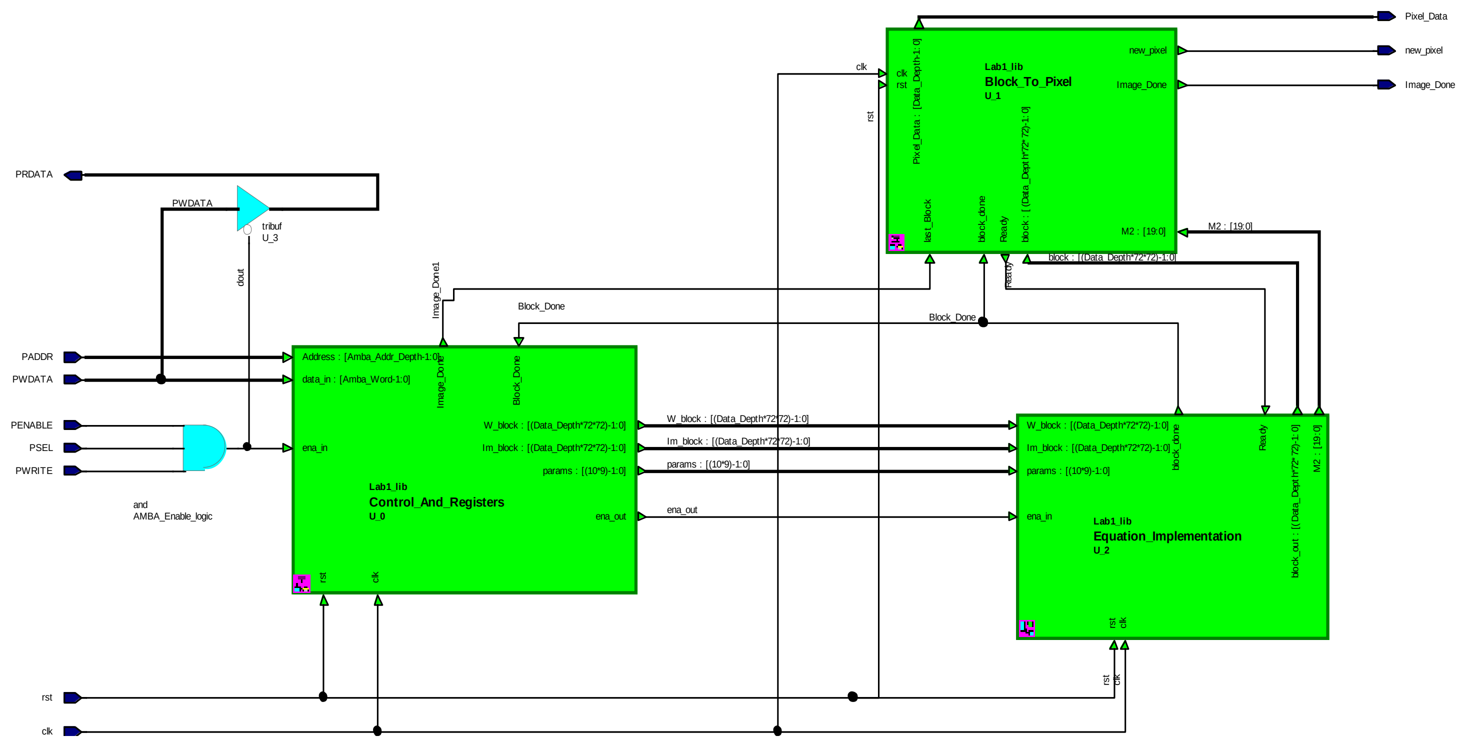


Figure 6: Visible Watermarking block diagram.

4.2 Control and Registers

4.2.1 Functional Description

The Control and Registers has the registers file that contains all the data that accessible to the CPU via the APB interface. Each round, it sends one block of the primary image and another block of the watermark image. In addition, send initial parameters, for calculations in the first round. The module signifies when its finish sending the last blocks.

4.2.2 Interface

Figure 7: Control and Registers interface.

	A	B	C	D	E	F	G	H	I
	Group	Name	Mode	Type	Signed	Bounds	Delay	Value	Comment
1		clk	input	wire					// system clock
2		rst	input	wire					// Reset active low
3		ena_in	input	wire					// Reset active low
4		Address	input	wire		[Amba_Addr_Depth-1:0]			// APB Address Bus
5		data_in	input	wire		[Amba_Word-1:0]			// APB Read Data Bus
6		Block_Done	input	wire					// when block is done
7		ena_out	output	reg					// when a block is ready
8		Image_Done	output	reg					//State indicator (Output)
9		Im_block	output	reg		[(Data_Depth*72*72)-1:0]			//[(72*72)-1:0], // Max size of block is 72*72 pixels and size of pixel is DATA_DEPTH
10		W_block	output	reg		[(Data_Depth*72*72)-1:0]			//[(72*72)-1:0], // Max size of block is 72*72 pixels and size of pixel is DATA_DEPTH
11		params	output	reg		[(10*9)-1:0]			//[9:0]//9 params size of 9 bits
12		registers	local	reg		[Amba_Word-1:0][2**(Amba_Addr_Depth-1):0]			
13		M2	local	reg		[19:0]			
14		N2	local	reg		[19:0]			
15		block_i	local	reg		[18:0]			// max 720*720, number of pixels
16		pixel_i	local	reg		[12:0]			// max pixels in block is 72*72 = 5184
17		param_i	local	reg		[2:0]			// we have 7 params
18		state	local	reg		[1:0]			// 0 - Im_block, 1 - W_block , 2 - params

Table 3: Control and Registers interface.

4.2.3 Block Diagram

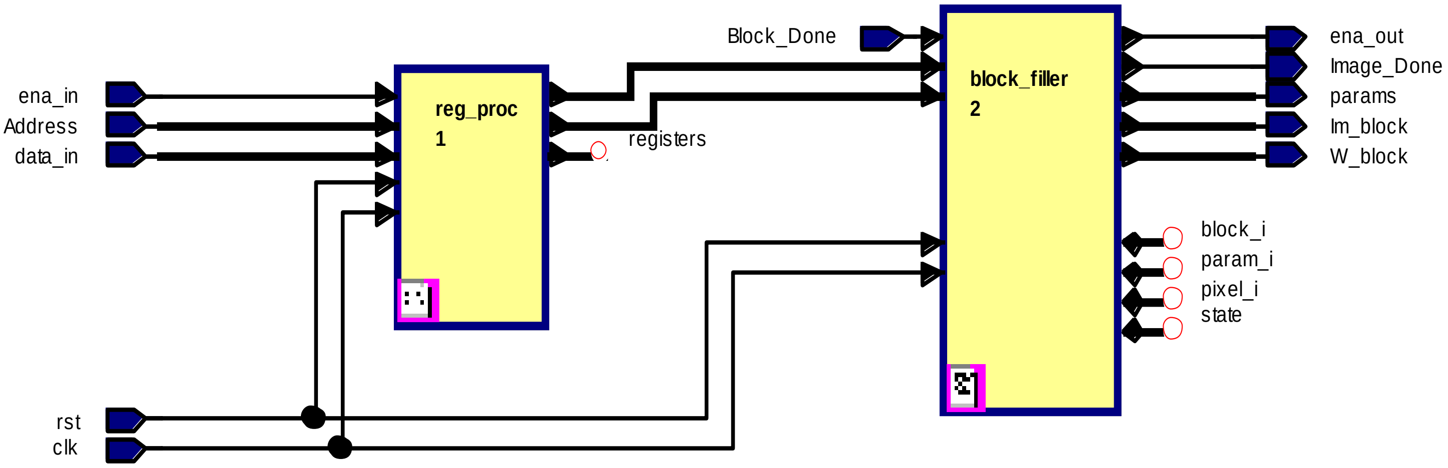


Figure 8: Control and Registers block diagram.

4.2.1 Flow chart\State machine diagrams

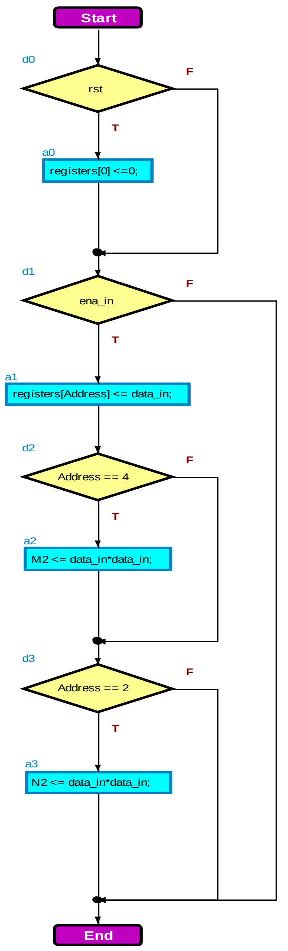


Figure 10: Registers Process

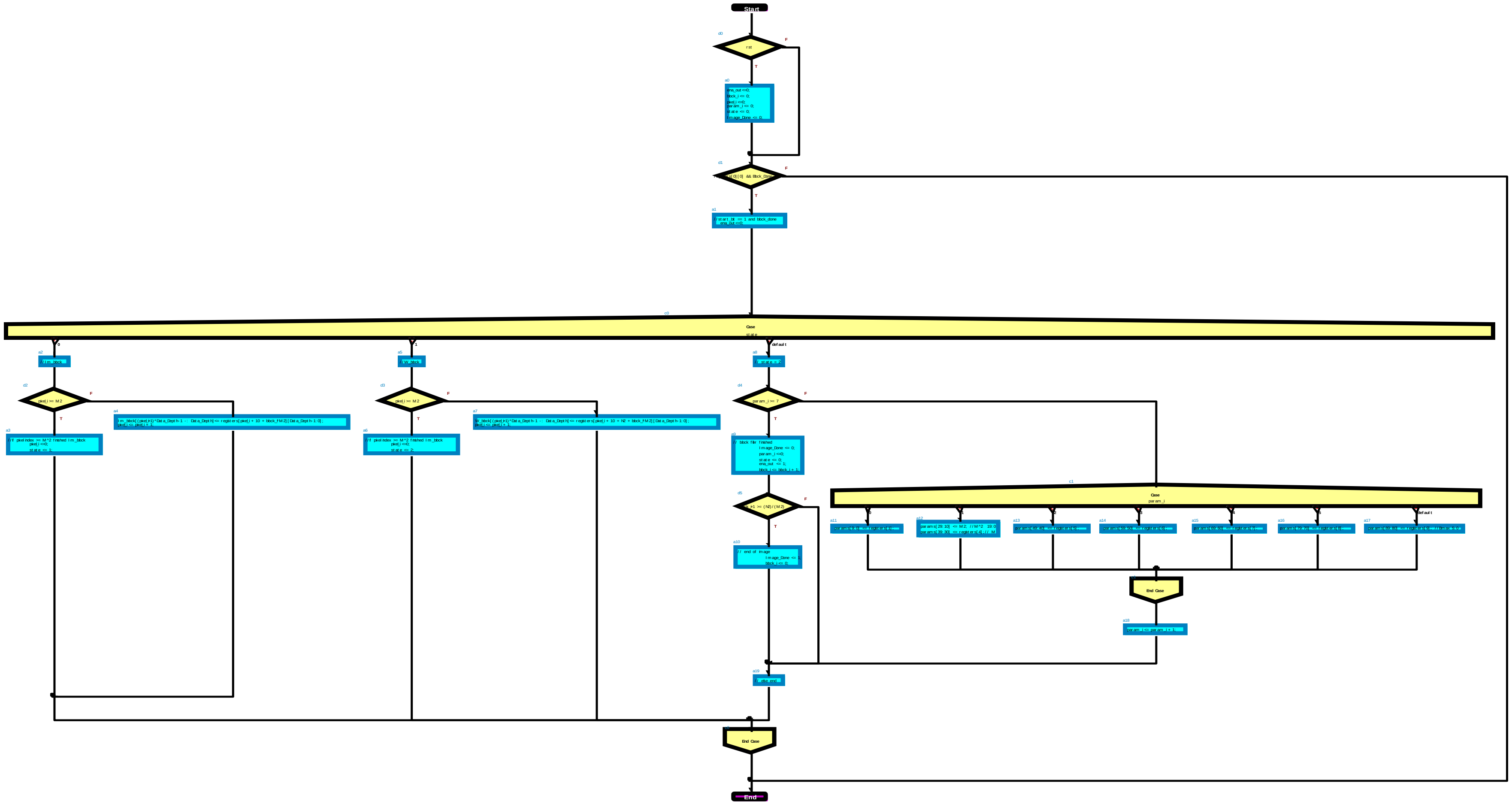


Figure 9: Block Filler Process

4.3 Equation Implementation

4.3.1 Functional Description

This module calculates the output block using the equations, given the images, initial parameters and output watermarked image and send it to the Block_to_Pixel module.

All the measurements are calculated with unsigned registers in Milli units, such that the precession is 1:1000.

These are the used equations:

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

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

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

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

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

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

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

The calculations are done by first sum up all the sums from eq.3-5, and then multiply and divide by the given parameters. The exponentiation in eq.1-2 is calculated using the Power2 module.

4.3.2 Interface

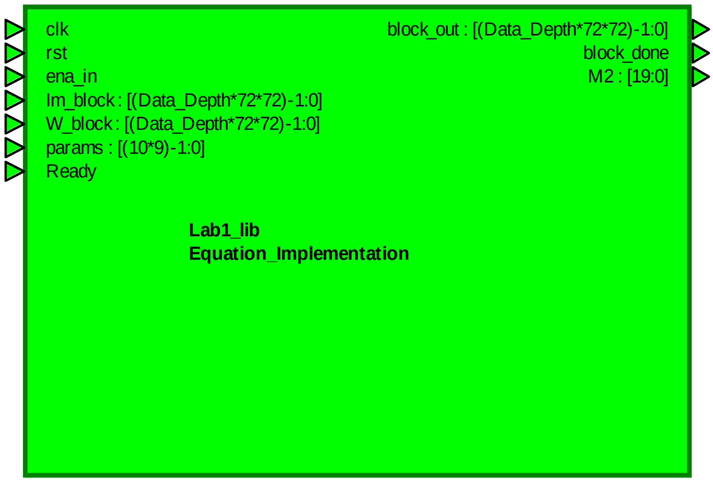


Figure 11: Equation Implementation interface.

	A	B	C	D	E	F	G	H	I
	Group	Name	Mode	Type	Signed	Bounds	Delay	Value	Comment
1		clk	input	wire					//system clock
2		rst	input	wire					//Reset active low
3		ena_in	input	wire					//Enable
4		Im_block	input	wire		[(Data_Depth*72*72)-1:0]			//[(72*72)-1:0], // Max size of block is 72*72 pixels and size of pixel is DATA_DEPTH
5		W_block	input	wire		[(Data_Depth*72*72)-1:0]			//[(72*72)-1:0], // Max size of block is 72*72 pixels and size of pixel is DATA_DEPTH
6		params	input	wire		[(10*9)-1:0]			//[9:0]//9 params size of 9 bits
7		Ready	input	wire					//Block_To_Pixel is ready to recive new block
8		M2	output	reg		[19:0]			//M*M number of pixel in a block
9		block_done	output	reg					//Finished watermark insertion of block
10		block_out	output	reg		[(Data_Depth*72*72)-1:0]			//[(72*72)-1:0], // Max size of block is 72*72 pixels and size of pixel is DATA_DEPTH
11		sum3	local	reg		[31:0]			// each sumX is the double Sigma from each (X) equation // 510*72*72*data_depth
12		sum4	local	reg		[31:0]			// each sumX is the double Sigma from each (X) equation // 510*72*72*data_depth
13		sum5	local	reg		[31:0]			// each sumX is the double Sigma from each (X) equation // 510*72*72*data_depth
14		sums_done	local	reg					//sums_calc block is finished
15		i	local	reg		[19:0]			//index of block's pixels
16		endOfRow	local	reg		[19:0]			// indicator for end of row in the block, for eq.5
17		eq345_done	local	reg					//equation 3,4,5 are finished
18		Sigma	local	reg		[9:0]			// sigma between 0-1000 milli (0-255/256)
19		Mu	local	reg		[9:0]			// mu between 0-1000 milli (0-1)
20		G	local	reg		[8:0]			// G between 0-510 milli
21		alpha	local	reg		[9:0]			// 0-1000 in milli
22		beta	local	reg		[19:0]			// 0-1000 in milli
23		eq12_done	local	reg					//equation 1,2 are finished
24		pow	local	wire		[9:0]			// return answer for 2^(-((Mu-0.5)^2)), in milli!!
25		i_res	local	reg		[19:0]			//index of block's pixels for result

Table 4: Equation Implementation interface.

4.3.3 Block Diagram

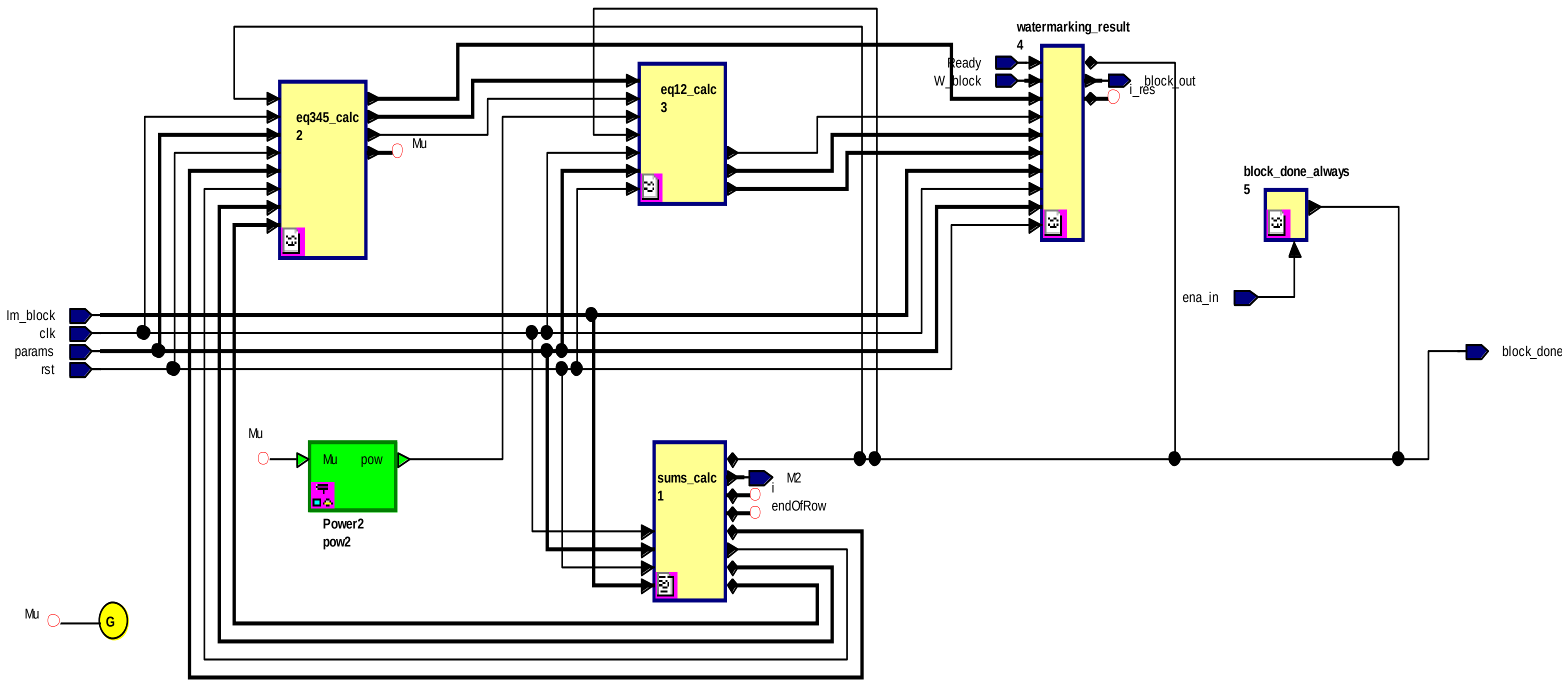


Figure 12: Equation Implementation block diagram.

4.4 Power 2

4.4.1 Functional Description

This module estimates the result of $2^{-(\mu_k-0.5)^2}, 0 \le \mu_k \le \frac{255}{256}$ for eq.1-2 calculations.

The boundaries of μ_k can be deduce from e.q.3.

The function graph in the specific boundaries:

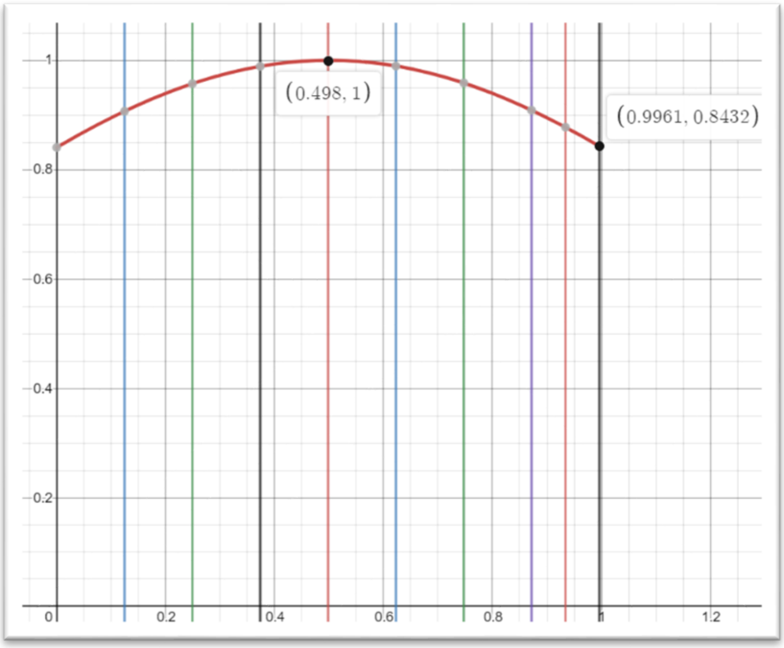


Figure 13: Exponential function.

We can see that the function values change between $843[m]$ to $1000[m]$, hence a good way for implementing it is by quantization. The module divides the range of the result to 8 equal ranges, and each range, outputs a constant result. Since the function is symmetric around 498[m], 4 constants are enough for the implementation.

$$2^{-(\mu_k-0.5)^2}[m] = \begin{cases} 875[m], 0 \le \mu_k < 125 \ \cup 875 < \mu_k \le 996 \\ 934[m], 125 \le \mu_k < 250 \ \cup 750 \le \mu_k < 875 \\ 975[m], 250 \le \mu_k < 375 \ \cup 625 \le \mu_k < 750 \\ 997[m] \qquad\qquad , 375 \le \mu_k < 625 \end{cases}$$

Using quantization, we achieve fast calculations and hardware efficiency, with less than 4% absolute error from the desired result. This error is reasonable and does not result in significant changes.

4.4.2 Interface

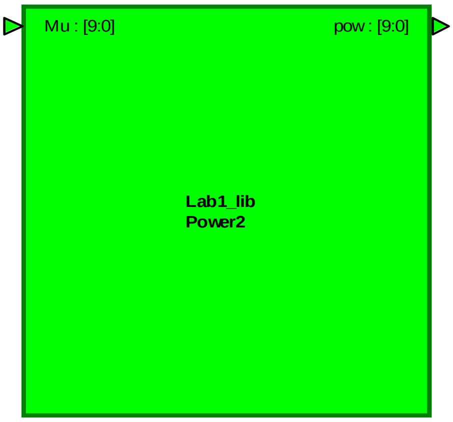


Figure 14: Power 2 interface.

	A	B	C	D	E	F	G	H
	Group	Name	Mode	Type	Signed	Bounds	Value	Comment
1		Mu	input	wire		[9:0]		// mu between 0-1000 milli (0-1)
2		pow	output	reg		[9:0]		// return answer for 2^(-((Mu-0.5)^2)), in milli!!

Table 5: Power 2 interface.

4.4.3 Flow chart\State machine diagrams

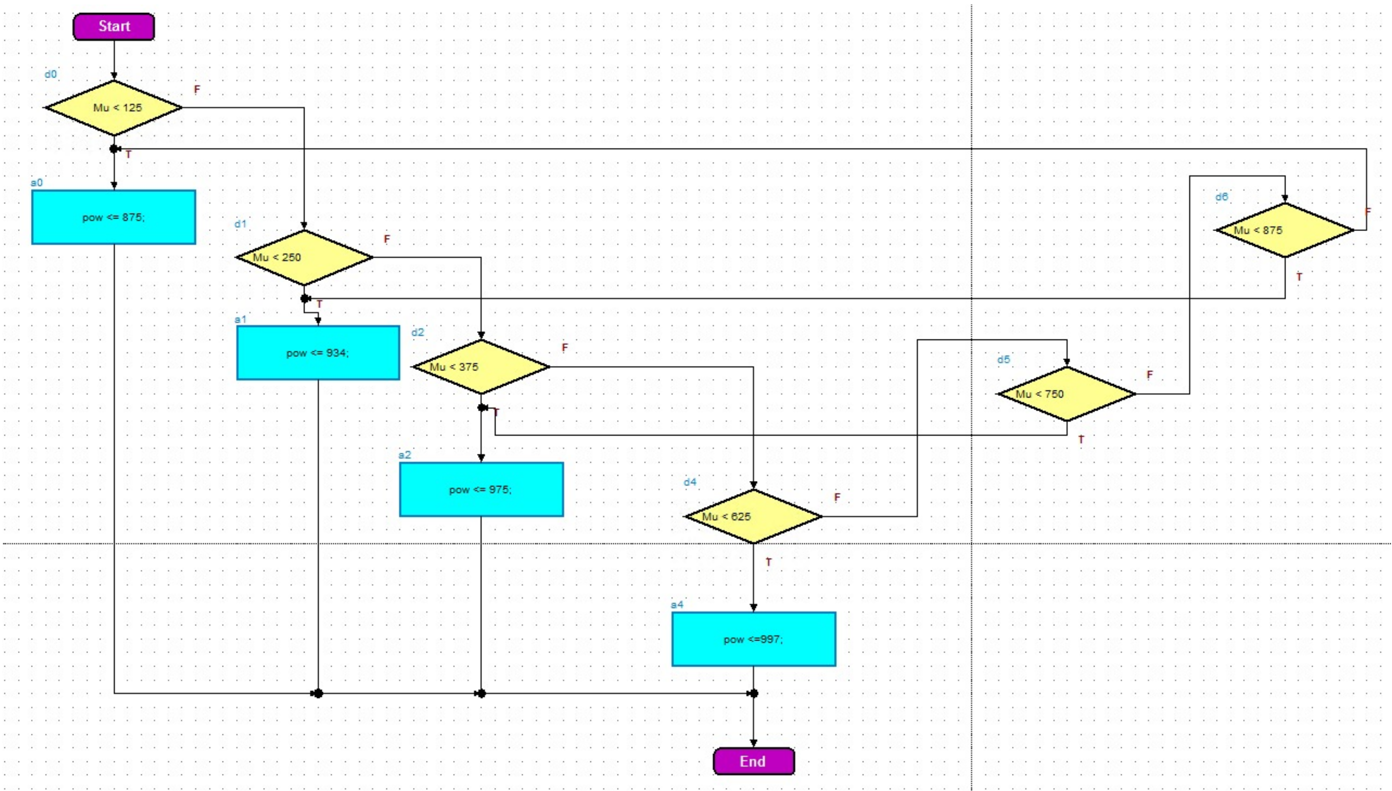


Figure 15: Power2 flow chart.

4.5 Block to Pixel

4.5.1 Functional Description

Receiving watermarked block and output it pixel by pixel.

When it receives the last_block bit from the Control_and_Registers module it's marks out the Image_Done after finish sending the next block pixels.

Each time, it sends a new pixel to Pixel_Data, it set new_pixel to '1'.

4.5.2 Interface

Figure 16: Block to Pixel interface.

	A	B	C	D	E	F	G	H	I
M	Group	Name	Mode	Type	Signed	Bounds	Delay	Value	Comment
1		block_done	input	wire					//Finished watermark insertion of block
2		block	input	wire		[(Data_Depth*72*72)-1:0]			//Block (after watermarking) that sent to Block to Pixel
3		M2	input	wire		[19:0]			//M*M number of pixel in a block
4		clk	input	wire					//system clock
5		rst	input	wire					// Reset active low
6		last_Block	input	wire					//State indicator - Control and Registers Image done
7		Ready	output	reg					//Block_To_Pixel is ready to recive new block
8		Pixel_Data	output	reg		[Data_Depth-1:0]			//Modified pixel (Output)
9		new_pixel	output	reg					//New pixel indicator
10		Image_Done	output	reg					//State indicator (Output)
11									

Table 6: Block to Pixel interface.

4.5.3 Block Diagram

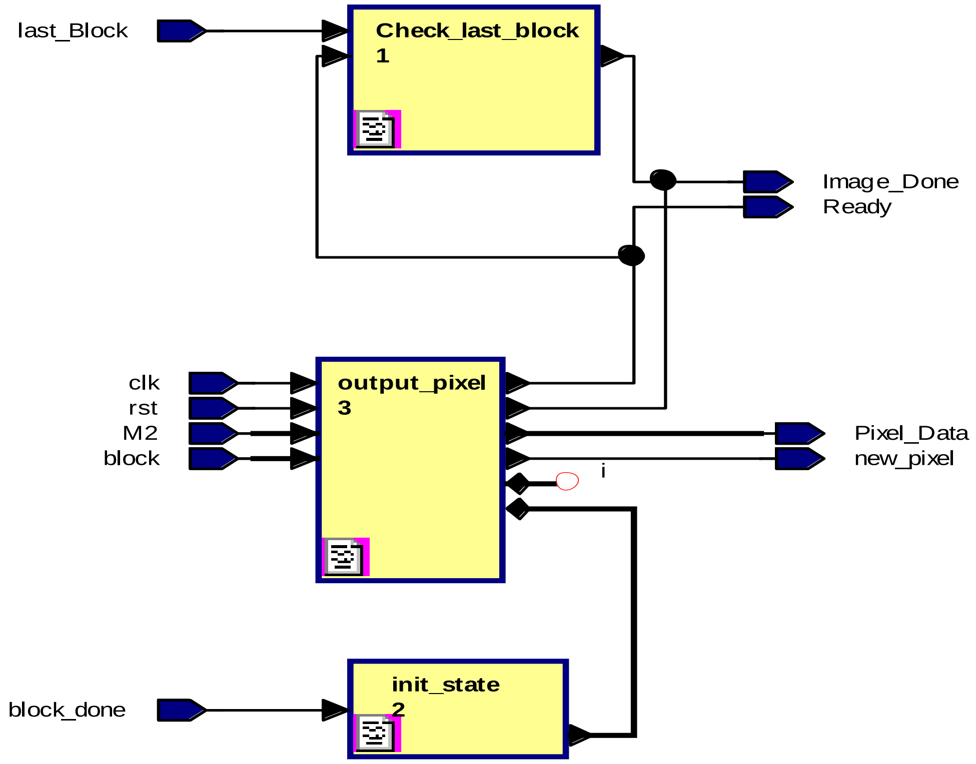
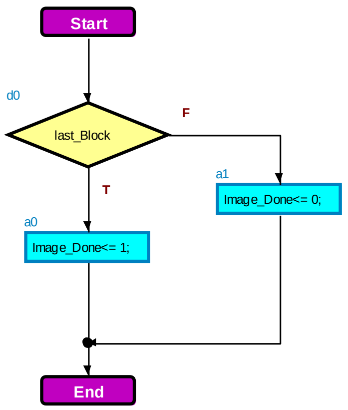


Figure 17: Block to Pixel Block Diagram.

4.5.4 Flow chart\State machine diagrams



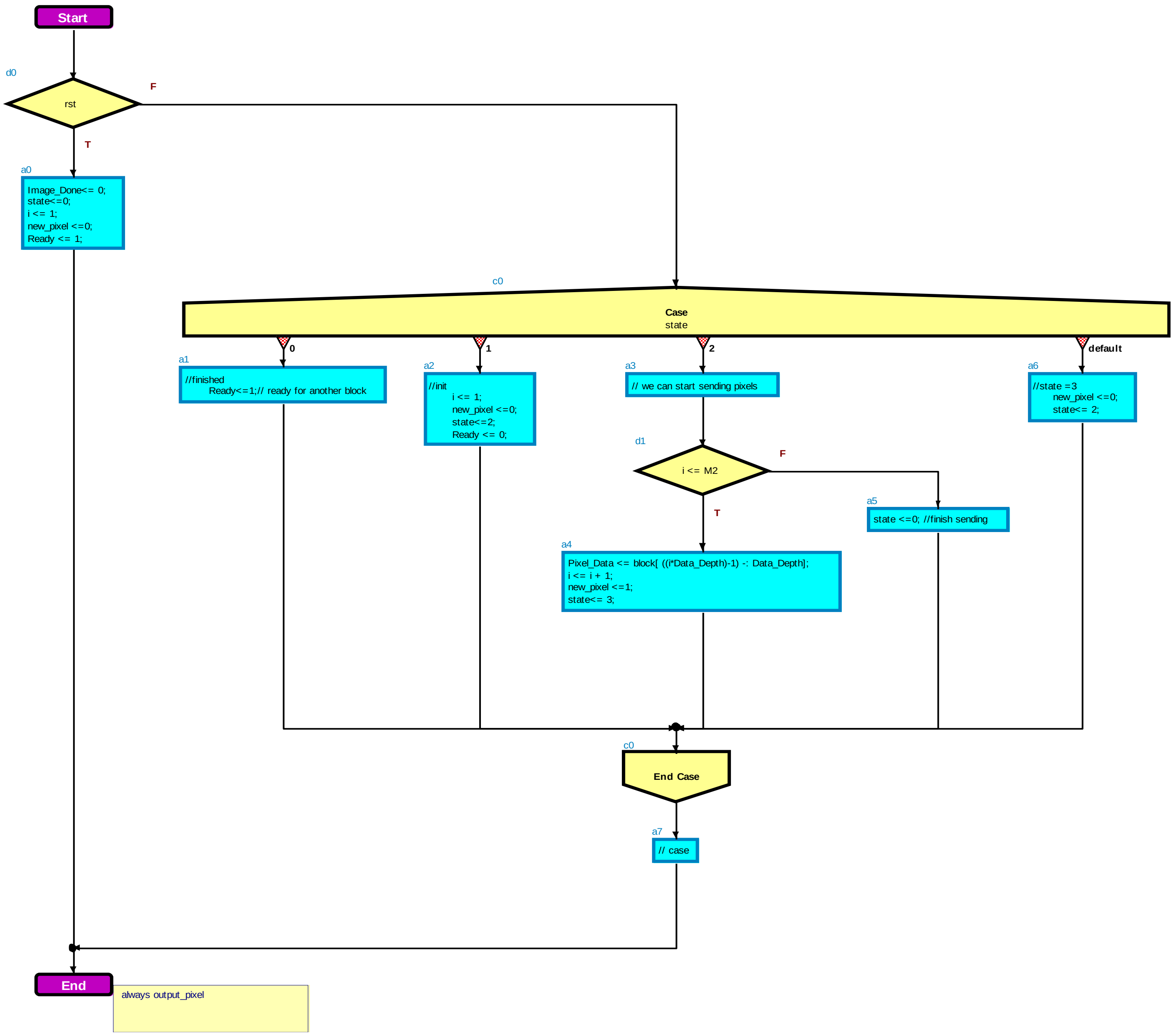
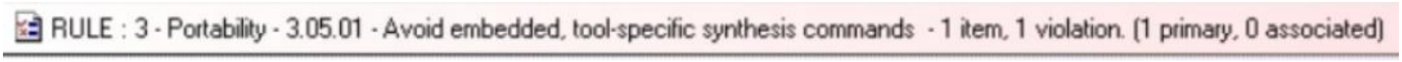


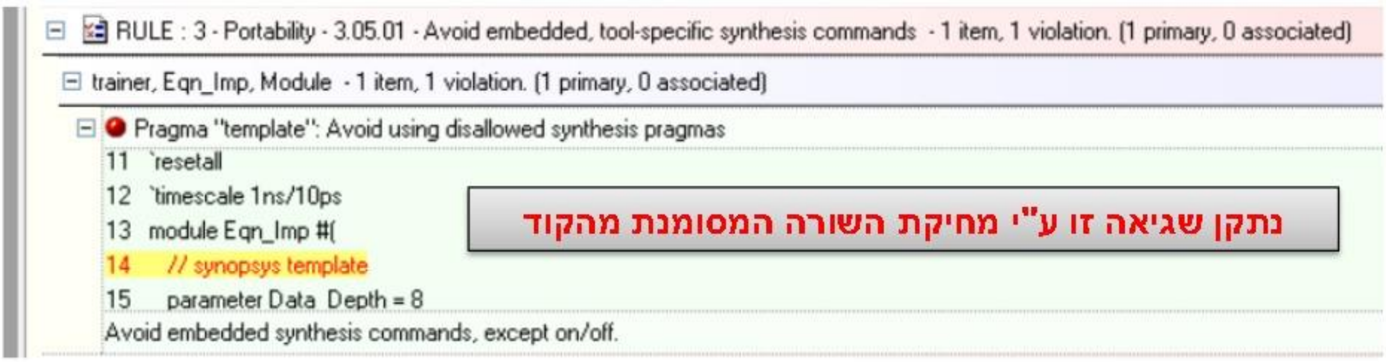
Figure 18: Block to Pixel flow chart.

5. RULES IN DESIGN CHECKER

In our design we used the RMM set of rules and disable the following rule –



For the reason of repeated compilation error –



6. APPENDIX

6.1 Registers File Description

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 + (Np^2)h	Primary image (Np,Np) pixel value	CPU Read/Write, Visible_Watermarking Read only
WatermarkPixel00	0x0A + (Np^2)h	Watermark image (0,0) pixel value	CPU Read/Write, Visible_Watermarking Read only
⁝	⁝	⁝	⁝
WatermarkPixelNN	0x09 + (Np^2+ Nw^2)h	Watermark image (Nw,Nw) pixel value	CPU Read/Write, Visible_Watermarking Read only
⁝	⁝	Size of the register bank is set by Amba_Addr_Depth	⁝

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.

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.

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.

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.

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.

EdgeThreshold register:

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

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.

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.

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.

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.

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 0x09 + (Np^2)h; minimum 1, maximum 255.

WatermarkPixelXY registers:

These registers hold the value of the Watermark image pixel at position (x,y). Address offset of first pixel is 0x0A + (Np^2)h and for the last pixel is 0x09 + (Np^2+ Nw^2)h; minimum 1, maximum 255.

6.2 References

[1] Amba standard Moodle→Amba Specifications