Visible Image Watermarking

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

Digital High-Level Design

Version 0.1

Classification:	**Template Title:**	Owner	Creation Date	Page
Logic Design Course	General Test Plan	Emmanuel Kofman Ruben Fratty	24, Nov, 2020	1 of 15

Revision Log

Rev	Change	Description	Reason for change	Done By	Date
0.1	Part 1	First part of the project			
0.2					
0.3					

Contents

List of figures

List of tables

Equation Notation

- $I(x,y)$ – Specific pixel of primary image
- $W(x,y)$ – Specific pixel of watermark image
- $i_k - k_{th}$ block of primary image
- $w_k - k_{th}$ block of watermark image
- $i_{w_k} - k_{th}$ block of watermarked image (result)

1. BLOCKS RTL DESCRIPTION

The project consists in the implementation of a watermarking system. It receives as input two images – the “Primary image” and the “Watermark” – and output the watermarked primary image. For practicality, both input images are of the same size $N_p \; X \; N_p$ pixels, where $N_p \in [200, 720]$.

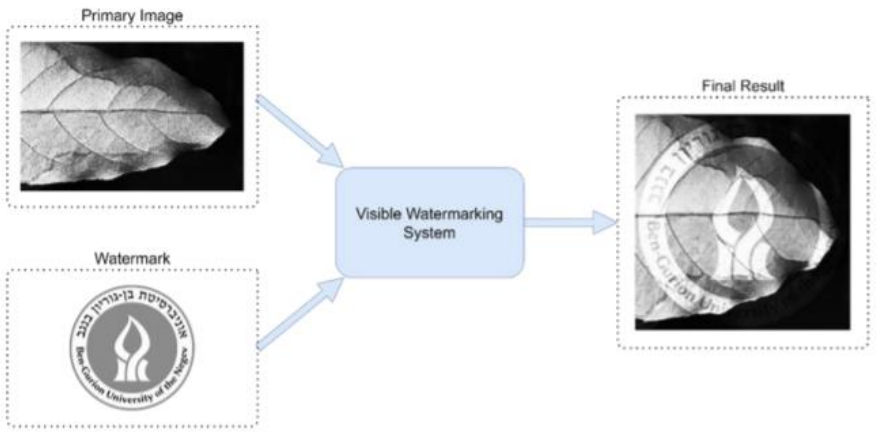

Figure 1: Visibal Watermarking System

In this first part, we implemented the architectural block that calculate the parameters needed for the watermarking and output the result. In addition, we have implemented the APB registers bank, following the AMBA protocol, to enable communication between the CPU/memory and the main block.

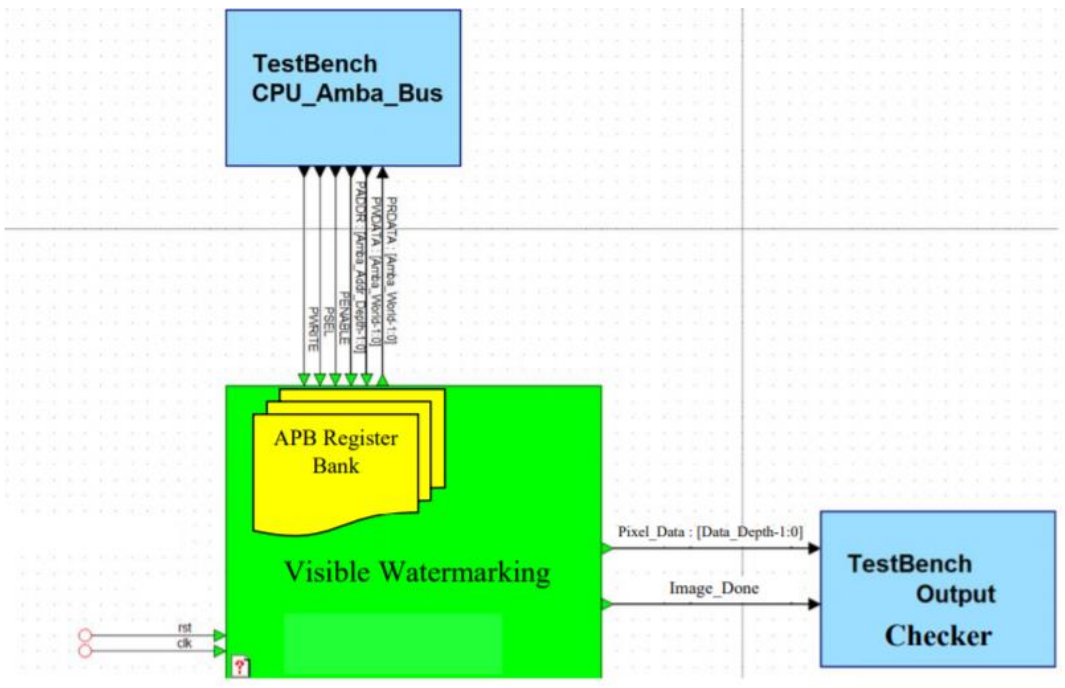

Figure 2: Top view of the design

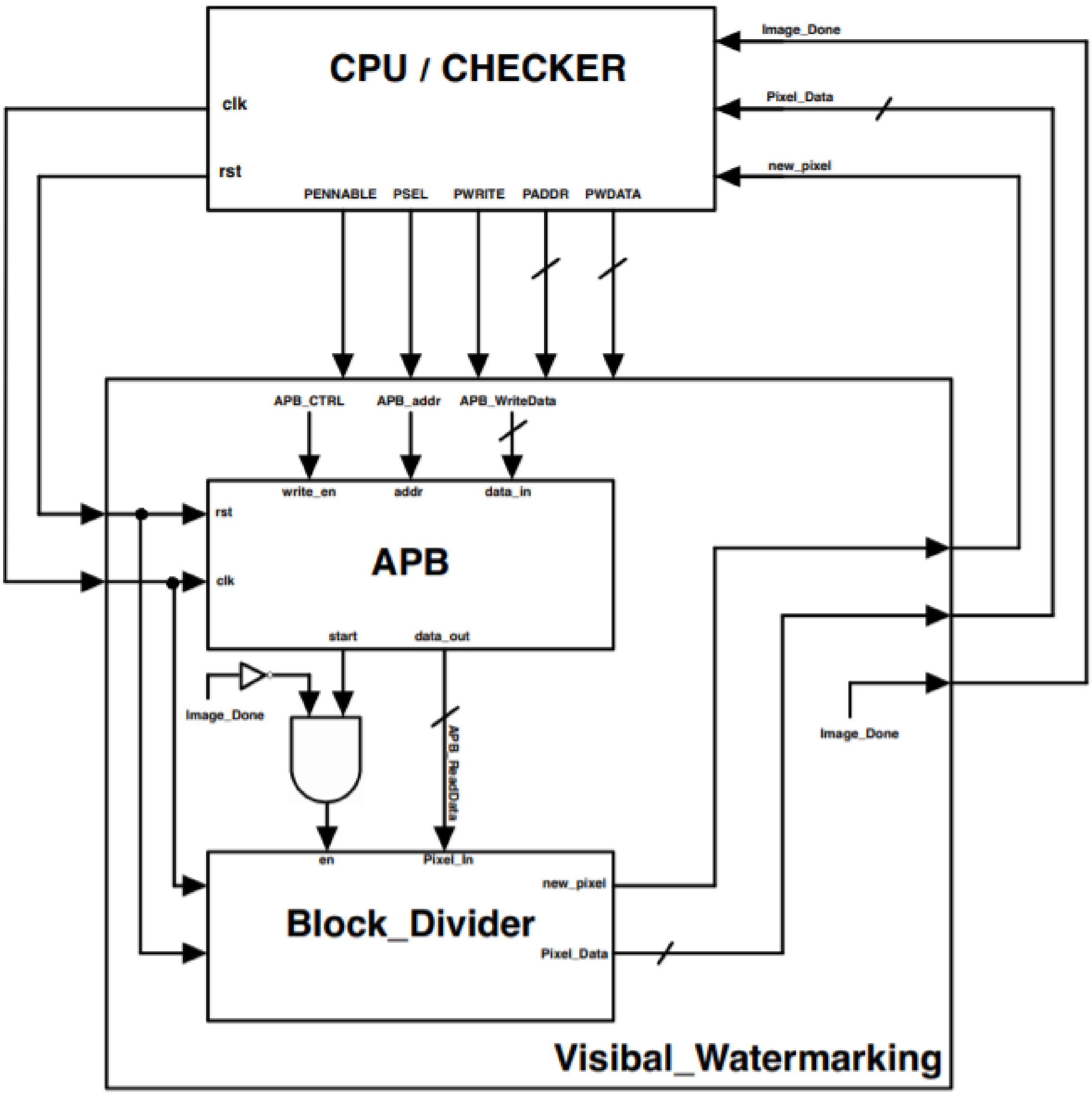

Figure 3: RTL view of the system with inputs and outputs of the blocks

1.1 Visibal_Watermarking.v

The main module of the system. This module is controlled by the CPU through the APB Register Bank from within it. In this stage of the project, the test bench is replacing the CPU.

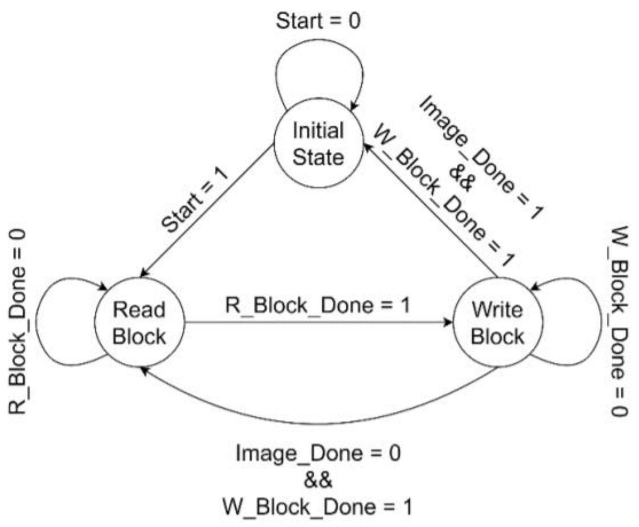

Figure 4: Data path given in the instructing paper

This module is designed to work as a state machine (cf. figure 4). To make the calculation faster, we divided both primary and watermark images into blocks of size M. We then modified the given state diagram to fit our design (cf. figure 5):

Start = ‘0’:

The CPU loads into the ABP register Bank, all the necessary data. When it is done, it sets start = ’1’

Start = ‘1’:

State 0: Idle, following the reset, aim to initialize all the parameters.

State 1: Parameters initialization: the module reads and stores the watermarking parameters from the APB at addresses 0x01-0x09 (cf. table 1).

State 2: Primary Block loading: load a primary image block to the block divider.

State 3: Watermark Block loading: load a watermark image block to the block divider.

State 4: At this stage, both the primary and watermark block have been loaded. The module then waits for the block divider to calculate and output the result watermarked block.

If the all the images have been processed, we return to state 0, waiting for another task from the CPU. Else, we jump back to state 2 to load and process another primary and watermark block.

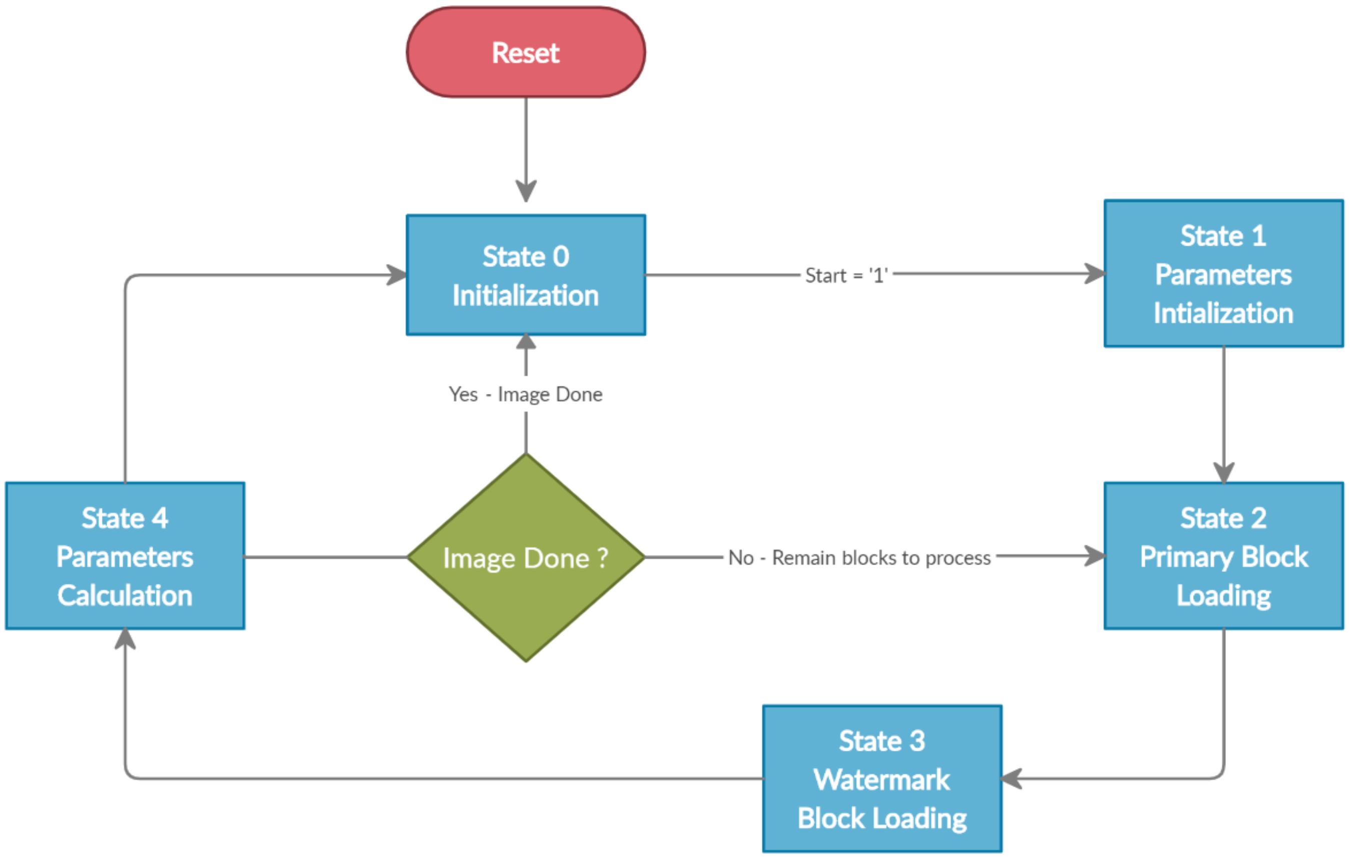

Figure 5: State machine of the main module (Visibal_Watermarking)

It is important to note that:

- From any state go back to "Reset" by setting $\boldsymbol{rst} = \text{'}\mathbf{1}\text{'}$.
- *Reset*- In this state the system the module's registers and wires to the default value.
- *Primary/Watermarking block loading* –To advance inside the memory and assign the matching pixel to its position in the block we have an offset register that tells the system where the starting pixel of the current block.
- *Parameters calculation* – This module is not responsible for any calculations. It only waits for another module to finish the processing of the blocks. When finished, the Block Divider module (responsible for the calculation) sets $\boldsymbol{block\_done} = \text{'}\mathbf{1}\text{'}$**.**

1.2 Block_Divider.v

This is one of the sub-modules of the main module, "Block_Divider.v" is responsible for the calculations of every parameters used during the watermarking process as well as outputting the result watermarked block.

It receives block after block of the Primary image and the Watermark until all the image has been processed. After loading both blocks, it can start calculating the parameters that will be used for calculating the watermarked image – $\boldsymbol{G_{u_k}, s_k, u_k, a_k \text{ and } b_k}$ calculated as followed:

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

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

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

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

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

When every parameter is calculated, it can finally process the watermarked block:

$$Eqn.(6): i_{w_k} = \begin{cases} \alpha_{max} \cdot i_k + \beta_{min} \cdot w_k & , G_{\mu_k} \geq B_{thr} \\ \alpha_k \cdot i_k + \beta_k \cdot w_k & , G_{\mu_k} < B_{thr} \end{cases}$$

To do so, as in the main module, this design follows a specific states diagram (cf. figure 6).

Start = '1':

State 0: Idle, following the reset, aim to initialize all the parameters. If the this is the first block, jump to state 1, otherwise go to state 2.

State 1: Parameters initialization: the module reads and stores the watermarking parameters sent from the main module at addresses 0x01-0x09 (cf. table 1).

State 2: Primary Block loading: load saves a full block (MxM pixels) of the primary image. In addition, to facilitate the parameters calculation (state 4), we performs every reading the sub-sum necessary for $\boldsymbol{G_{u_k}, s_k, u_k}$:

$$Eqn.(7): sigma_M = \sum_x \sum_y I(x,y)$$

$$Eqn.(8): sigma_S = \sum_x \sum_y \left| I(x,y) - \frac{I_{white}+1}{2} \right|$$

$$Eqn.(9): sigma_G = \sum_x \sum_y |I(x,y) - I(x+1,y)| + |I(x,y) - I(x,y+1)|$$

State 3: Watermark Block loading: load and saves a full block (MxM pixels) of the watermark image. When every pixel of the block has been saved, the module performs the $\boldsymbol{G_{u_k}, s_k, u_k}$ calculation:

$$\mu_k = \frac{sigma_M}{M^2 \cdot (I_{white}+1)}, \qquad \sigma_k = \frac{2 \cdot sigma_S}{M^2 \cdot (I_{white}+1)}, \qquad G_{\mu_k} = \frac{sigma_G}{M^2}$$

State 4: At this stage, we can compute the last two parameters $\boldsymbol{\alpha_k \ and \ \beta_k}$ according to Eqn. (1) and (2).

State 5: When entering stage 5, everything is now ready for calculating and outputting the watermarked block according to Eqn.(6). During M^2 clocks, each computed pixel will be load on ***Pixel_Data*** which will be latter read by the CPU.

If the all the blocks have been processed, we return to state 0, waiting for another block in a next image. Moreover, like in the main module, setting $\boldsymbol{rst = '1',}$ brings the system to the "Reset" state whichever state it is operates in the current moment.

Reset – In this state the system sets all parameters' values needed for the module to zero and then it procedures to the next state.

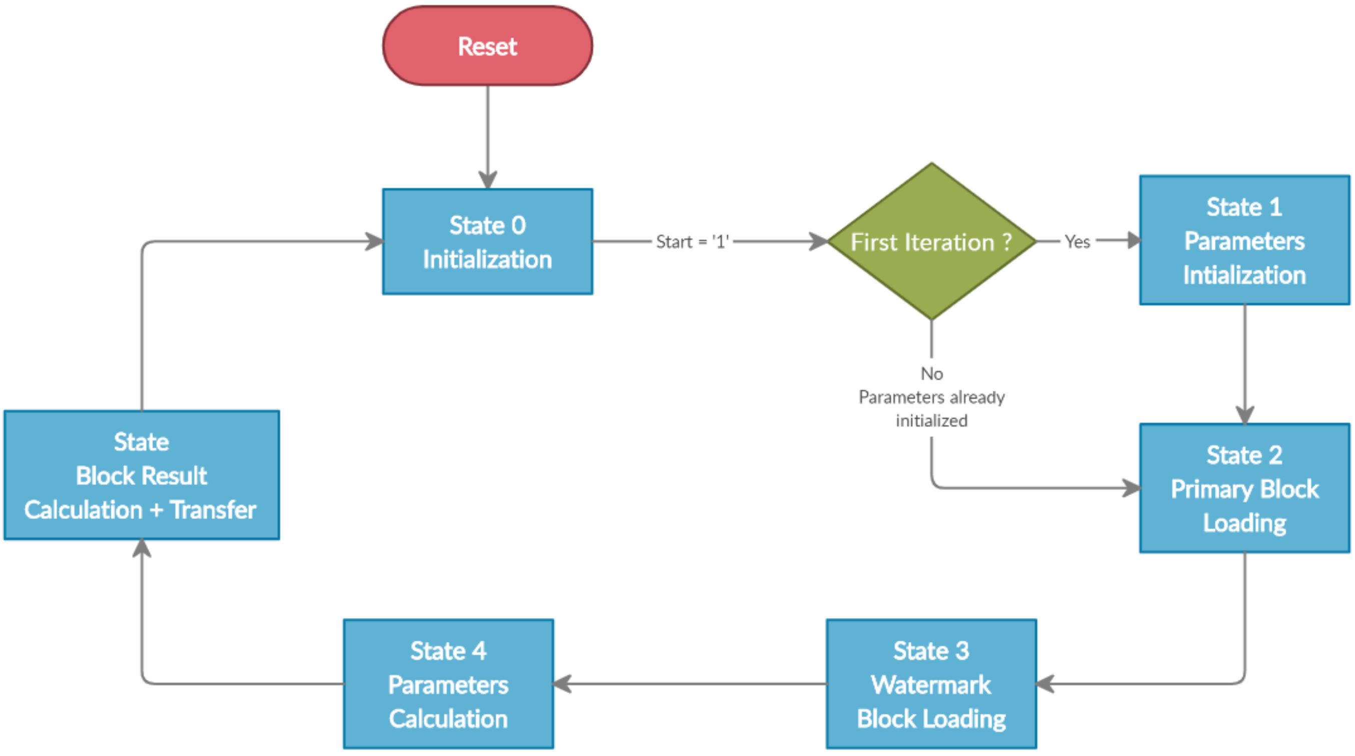

Figure 6: State machine of the "BlockDivider" sub-module

1.3 APB.v

The CPU controls the design through the APB Register Bank. The ABP is composed of numerous registers (each of them of size Amba_Word), in particular both the primary and watermark images:

Table 1: The APB registers

Register Name	Address Offset	Comments	Access Type
Control (CTRL)	0x00	Control the design: 0 – Wait, 1 – Start	CPU Read/Write, Visible_Watermarking Read Only
White_Pixel	0x01	White pixel value Default = 255	CPU Read/Write, Visible_Watermarking Read Only
Np	0x02	Primary Image matrix rows/colums number Min = 200, Max = 720	CPU Read/Write, Visible_Watermarking Read Only
Nw	0x03	Primary Watermark matrix rows/colums number Min = 200, Max = 720	CPU Read/Write, Visible_Watermarking Read Only
M	0x04	The small blocks matrix rows/colomns number Min = 1, Max = Np/10	CPU Read/Write, Visible_Watermarking Read Only
Bthr	0x05	Predefined Edge detection threshold Min = 1, Max = 20	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
I(0,0)	0x0A	First Primary pixel value	CPU Read/Write, Visible_Watermarking Read Only
...	...	...	...
I(Np,Np)	0x09+(Np)^2	Last Primary pixel value	CPU Read/Write, Visible_Watermarking Read Only
W(0,0)	0x0A+(Np)^2	First Watermark pixel value	CPU Read/Write, Visible_Watermarking Read Only
….	...	...	...
W(Nw,Nw)	0x09+(Np^2+ Nw^2)	Last Watermark pixel value	CPU Read/Write, Visible_Watermarking Read Only

The ABP data bank is the only way for the design to communicate with the CPU. Hence, to do so, it has several inputs/outputs:

Inputs:

- Clk – System clock
- Rst – Reset active low
- Write_en – Controls the read/write state: 0 – Read, 1 – Write
- Addr (Amba_Addr_Depth bits) - The address to read or write the data
- Data_in (Amba_Word bits) - Data received from the CPU

Outputs:

- Data_out (Amba_Word bits) - Data read from the data bank to the CPU or design
- Start – Last bit written to the data base by the CPU: 0 – System off, 1 – Start processing

The ABP interface works according to the AMBA standard:

Write: When the CPU wishes to write data to the APB, it needs to follow the AMBA protocol:

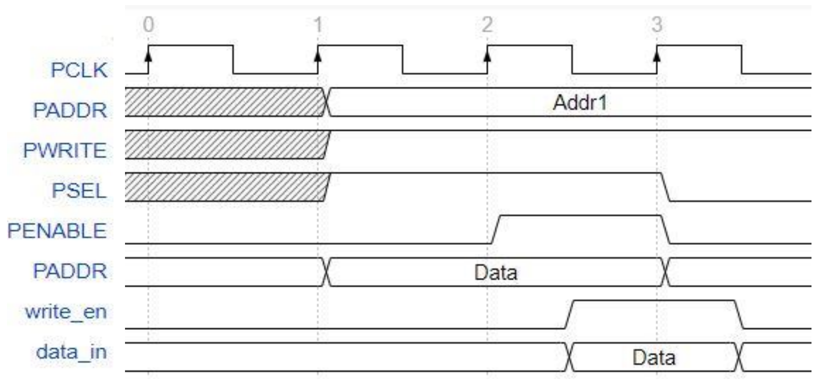

Figure 7: Writing data to the APB register bank

Read: When the design wishes to write data from the APB, it needs to set the address of the requested data and on the next cycle, the data will be ready on the bus:

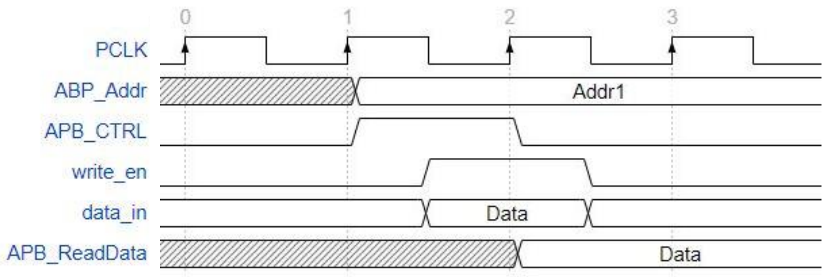

Figure 8: Reading data from the APB registers

1.4 tb_Visibal_Watermarking.v

The test bench of the system. It is supposed to replace the CPU in the original design. It generates the clock, extracts the pixels from the images and stores them in the memory.

At the beginning of the run, it loads both the primary and the watermark images as well as all the parameters of the system (Np, Nw, M, Bthr...). When all the data has been sent, it set the CTRL bit (address 0x00 in the APB) to ‘1’ which starts the watermarking.

During the process, it receives and stores blocks after blocks of the watermarked (result) image. The process ends when Image_Done is set to ‘1’ by the design, which means the result is ready.