Military Technical College
Kobry Elkobbah,
Cairo, Egypt

5th International Conference
on Electrical Engineering
ICEENG 2006

DESIGN OF AN ITERATIVE IMAGE RESTORATION ALGORITHM USING FPGA

Fawzy Eltohamy Hassan*, Ph.D. Gouda Ismail*, Ph.D. Esam Hassan Hamza*,B.Sc.

Abstract:

Programmable logic is emerging as an attractive solution for many digital signal processing application. This paper presents an FPGA implementation of an iterative image restoration technique. The simulation results show the speedup that can be achieved by implementing this algorithm on reconfigurable hardware as compared to the implementation of the algorithm using software. The process from design entry to files download are introduced.

Keyword: Iterative algorithm, FPGA, Image restoration.

1. Introduction:

Image restoration attempts to recover an image that has been degraded by using a priori knowledge of the degradation phenomena. Thus restoration techniques are oriented toward modeling the degradation and applying the inverse process in order to recover the original image. The degradation can be due to a number of reasons, such as, motion between the camera and the scene, atmospheric turbulence, and defocusing. Noise is typically added to the acquired data, which may originate at the electronics of the system or be due to transmission [1]. A several techniques have appeared in the literature to provide solutions to the restoration problem see, for example, Analysis and FPGA Implementation of Image Restoration under Resource Constraints is presented in [2]. Image Analysis and Partitioning for FPGA Mapping is presented in [3]. Issues in real-time image processing on a custom-computing platform are discussed in [4]. In [5], implementation of a pixel processor for object detection using Xilinx FPGAs is compared with DSP processors. This paper focused on the iterative algorithm due to several reasons, such as, this algorithm possesses a pattern of local computations due to the dependency of a pixel's restored value on its 1-neighbors, those types of computations allow a spatial implementation and also, the same local computation is repeated for a number of iterations and on a considerable amount of data [2]. A very important aspect of the hardware implementation is its performance in terms of running time; it provides a speedup or a faster solution that can be achieved by implementing this algorithm on FPGA. The hardware utilize the parallelism of the algorithm since, at each step, the same restoration operation is performed on each pixel. Segmenting the image and processing each segment independently becomes a necessity because whole image cannot be processed at once. The design steps will be accomplished by using two well-known packages **FPGA advantage for HDL design**, **release 5.2** and **Xilinx ISE, release 5.2i.** In this paper**,** all design steps which includes design entry, functional simulation, synthesis and timing simulation as recommended by Mentor Graphic Vendor of ***FPGA Advantage, Release 5.2*** [6] are introduced.

This paper is arranged into several sections. In the next section, the basic properties of the iterative image restoration algorithm is presented. In section 3 the iterative algorithm design description is explained. In section 4, the simulation results are introduced. In section 5, testing the downloaded design is explained. In section 6, experimental results are presented and finally conclusion is presented in the last section.

---

* Egyptian Armed Forces

2. Iterative image restoration algorithm: An Overview

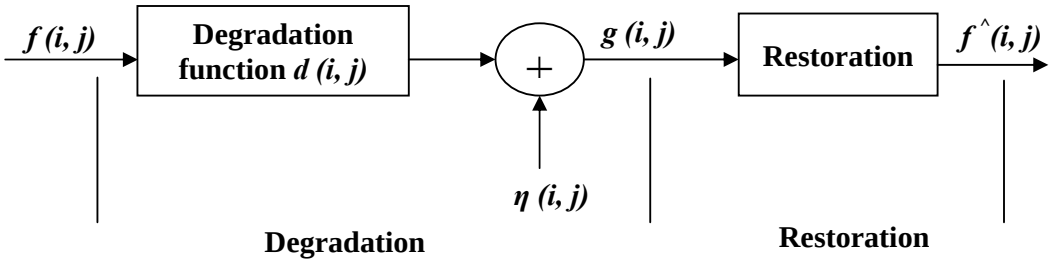


Fig. 1. A model of the image degradation / restoration process

Fig. 1. Shows, the degradation model is given in the spatial domain by [1]:

$$g(i,j) = d(i,j) \otimes\otimes f(i,j) + \eta(i,j) \quad (1)$$

where $f(i,j)$ and $g(i,j)$ denote respectively the original and observed degraded image, $d(i,j)$ is the impulse response of the degradation function, and $\otimes\otimes$ two-dimensional (2D) discrete linear convolution, and $\eta(i,j)$ denotes an additive noise (gaussian noise).

The objective of image restoration process is to obtain an estimate $\hat{f}(i,j)$ as close as possible to the original image $f(i,j)$ given $g(i,j)$ and $d(i,j)$ [7].

A basic form of iterative restoration algorithm is

$$\hat{f}_o(i,j) = g(i,j) \quad (2)$$

$$\hat{f}_{k+1}(i,j) = \hat{f}_k(i,j) + \beta\,(g(i,j) - \hat{f}_k(i,j) ** d(i,j)) \quad (3)$$

where $\hat{\mathbf{fo}}$ represents the initial image, $\hat{f}_{k+1}$ represents the estimated restored image at **k-th** iteration, and β a parameter which controls convergence. The sufficient condition for convergence is given by [8]:

$$|1 - \beta D(u,v)| < 1 \quad (4)$$

Where $D(u,v)$ is the 2D discrete Fourier transform **(DFT)** of $d(i,j)$ and $|z|$ denotes the magnitude of a complex number z. Using the fact that $|D(u,v)| \le 1$, this condition is simplified to [8]:

$$0 < \beta < 2 \quad , \quad D(u,v) > 0 \quad (5)$$

In this work, $\beta = 1$, $\eta(i,j)$ is ignored, and an impulse response $d(i,j)$ of support 3 x 3 samples is used, with $d(0,0) = \mathbf{r_0} = 1/2$ and $d(i,j) = \mathbf{r_1} = 1/16$ for i = -1, 0, 1 , j = -1, 0, 1 , $(i, j) \neq (0,0)$. Since the degradation system is presented as the following equation [2]:

$$\sum_{i=-1}^{1}\sum_{j=-1}^{1} d(i,j) = 1 \qquad or \qquad r_o + 8r_1 = 1 \quad (6)$$

3. Iterative image restoration algorithm design description

The general basic setup of the hardware implementation consists of some medium to store the image data and a processor to perform the restoration. The memory is external but the processor is configured on the FPGA. Fig. (2) shows this setup.

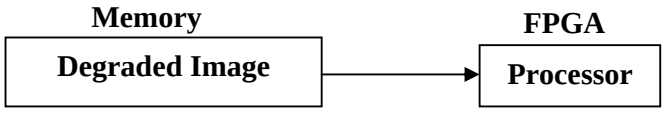


Fig. 2. The basic hardware model

The algorithm could be implemented in hardware just as in the software model. The sequence is sending 9 pixel values to the processor at a time, letting the processor computes the new value for the center pixel and writing the new value back. This sequence have to be repeated for several iteration steps. Since the communication between the memory and the processor is relatively slow that would introduce a large delay, an improved way to perform this task is to utilize the parallelism of the algorithm [9,10]. Since at each step the same restoration operation is performed on each pixel, by assigning a center pixel value and its eight neighboring pixel values to each processor, all pixels can be processed in parallel, which increases the performance in terms of speed substantially. Also, loading the pixel values onto FPGA once, performing several iterations and writing the result back to the memory, decreases the overhead of communication between the FPGA board and the memory unit. This model is not realistic because of the limited hardware resources. The number of processors that can fit into an FPGA chip is less than the number of pixels contained in the image we usually deal with. Therefore one whole image cannot be processed at once, and the need to segment the image arises. The image is segmented into regions of size m x n such that there is enough number of processors on the FPGA chip to process all of them in parallel as desired and each time one segment is loaded onto the FPGA. After restoration of one segment is completed the data is written back and the next segment is loaded. As mentioned before, the algorithm uses the value of the pixel itself and its eight neighbors. At the image boundaries, data of all eight neighbors are not available. This introduces loss of image quality. At each iteration the effect of the boundary will penetrate one pixel into the image, since the new value of a pixel depends on the neighbors. One solution to lessen this effect is to allow the segments to overlap with 1 pixel. Fig. 3. Shows the main block of the iterative image restoration algorithm.

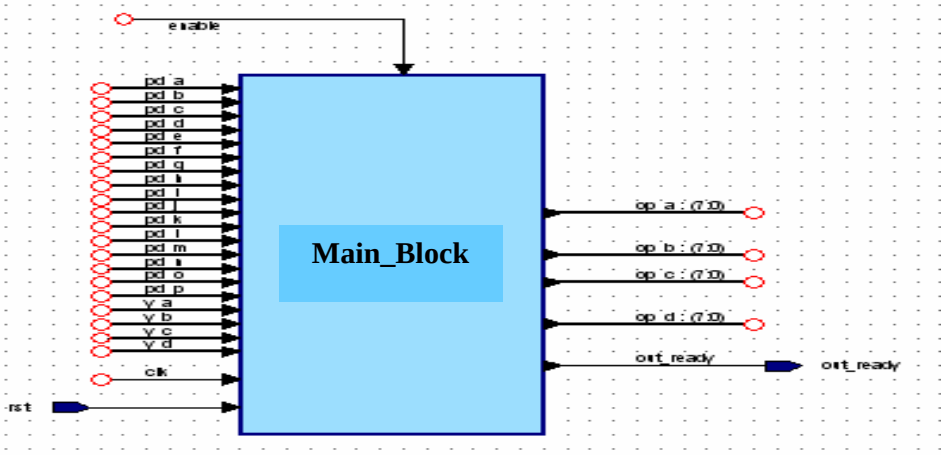


Fig. 3. The main block of iterative image restoration algorithm

The segment 4x4 (pd_a, pd_b, pd_c,….pd_p) plus the fixed degraded center pixels (y_a,…. y_d) is loaded via 20 input ports. By assigning the center pixel value and its eight neighboring pixel values as input to each processor. There are 4 processors on the FPGA, which computes the next iteration value, (new 4 outputs (op_a, op_b, op_c, op_d)); in parallel, (at the same time), that increases the performance in terms of speed substantially. Table 1. shows the function of the input / output ports of the design.

Table 1. Main Block inputs / outputs

Port	Function
pd_a, pd_b, pd_c, pd_d, pd_e, pd_f, pd_g, pd_h, pd_i, pd_j, pd_k, pd_l, pd_m, pd_n, pd_o, pd_p,	To input the values of the degraded pixels of the segment into the processors, the type of this data is unsigned.
y_a, y_b, y_c, y_d,	Represents the original fixed values of the center degraded pixels of the segment, where the iterations are performed on them. In the first iteration (initial values) y_a = pd_f , y_b = pd_g, y_c = pd_j , y_d = pd_k,
Rst	**Reset**, when rst='**0**' the circuit is in reset state and no processing occurs, when rst='**1**' the circuit is ready for processing procedures.
Clk	**Clock**, in every clock rising edge a new segment is entered to the processor
op_a,op_b, op_c, op_d	The output data (pixel value or the gray level), the type of this data is unsigned
Out_ready	Out_ready='1' means the circuit output is shown Out_ready='0' means no output
Enable	**Enable** of the circuit (enable='**1**' the circuit is 'ON' and ready for processing steps, enable='**0**' the circuit is 'OFF' and no processing sequence occurs).

The block diagram of the main block consists of 4 processors, as shown in Fig. 4.

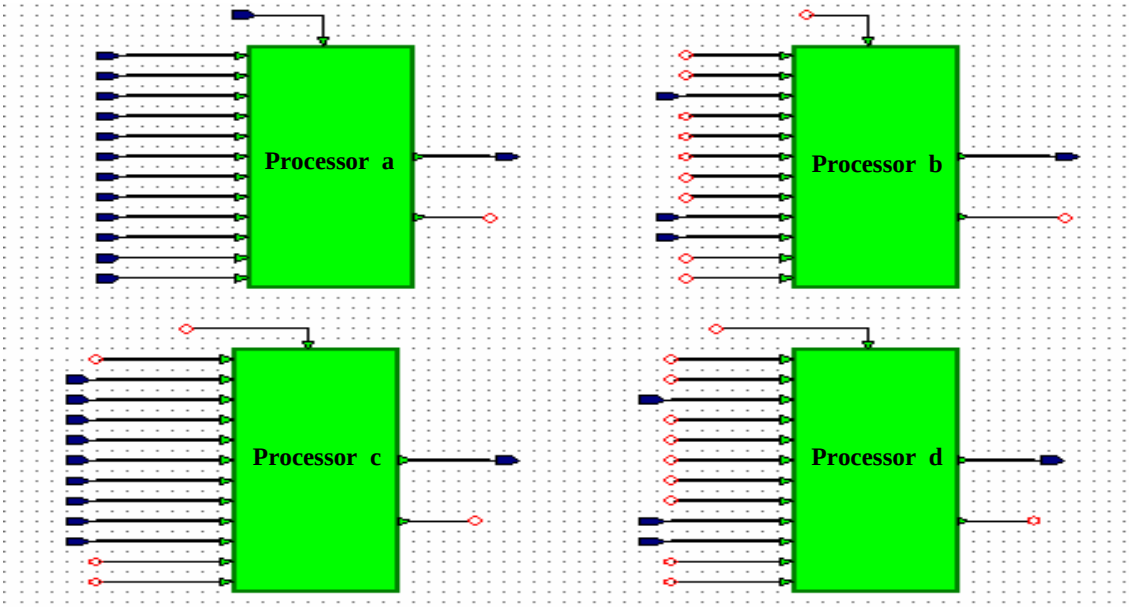


Fig. 4. The block diagram of the main block

The structure of one processor is shown in Fig. 5.

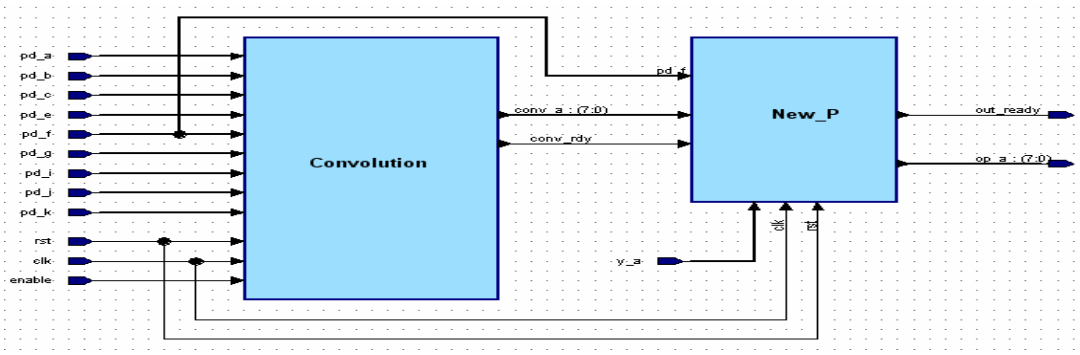


Fig. 5. The block diagram of the processor

The main blocks of the processor are:

A. Convolution

B. New_P

A. Convolution:

The block diagram of convolution is shown in Fig. 6. This block computes the convolution that is one of the most common spatial domain operations performed on an image. According to it a kernel (mask) of numbers is multiplied by each pixel in the neighborhood; the results are summed, and used to calculate the new value of the current pixel [11]. Fig. 7. shows the convolution operation.

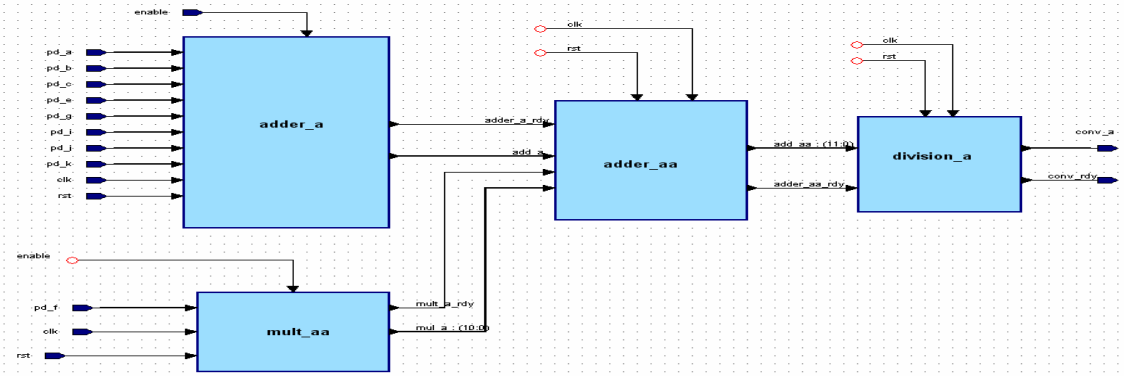


Fig. 6. The block diagram of the convolution

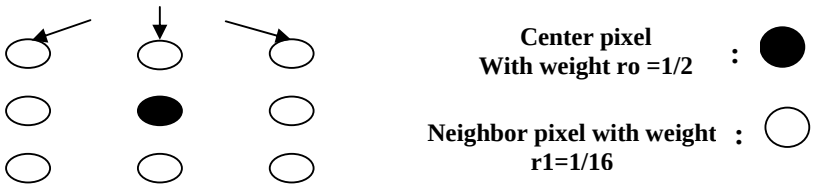


Fig. 7. Set of pixels necessary to restore one pixel

The main blocks of convolution are:

(a) Adder_a , (b) Multi_a, (c) Adder_aa and (d) Division_a

a) Adder_a:

In this block, the inputs are 8-neighbor of the center pixel, and the output (add_a) is the summation of these pixels.

add_a = pd_a + pd_b +pd_c + pd_d + pd_f + pd_g + pd_h + pd_i

b) Multi_a:

In this block, the input is the center pixel (pd_e), and the output (mul_a) is the value of center pixel multiplied by 8.

mul_a = pd_e * 8

c) Adder_aa:

In this block, the inputs is (add_a) and (mul_a), the output (add_aa) is the summation of these two values.

add_aa = add_a + mul_a

d) Division_a

In this block, the input is (add_aa); the output (conv_a) is the division of (add_aa) over 16.

Conv_a = add_aa / 16

B. New_p:

The block diagram of new_p is shown in Fig. 8. This block is used to calculate the next iteration (new value of the center pixel (op_a)).

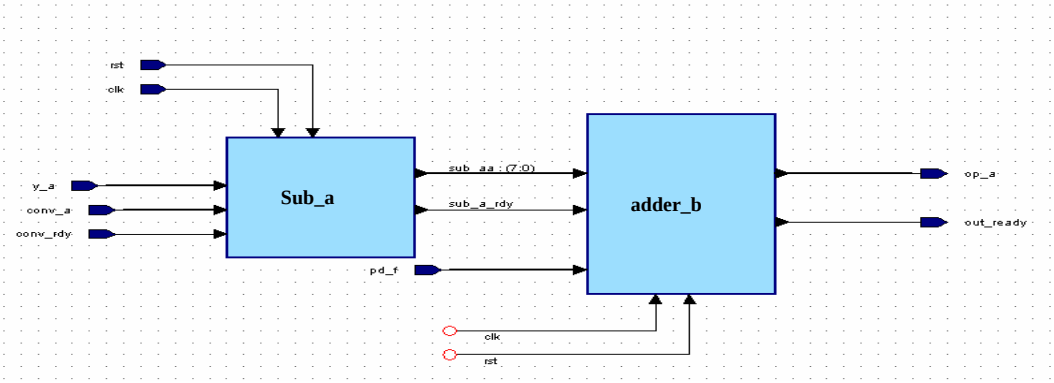


Fig. 8. The block diagram of new_p

The main blocks of new_pt are:

(a) Sub_a , (b) Adder_b

a) Sub_a

In this block, the inputs are the original center pixel (y_a) and (conv_a); the output (add_b) is the subtraction of them.

add_b = y_a – conv_a

b) Adder_b

In this block, the inputs are the new center pixel (newpd_e), (from previous iteration), and (add_b); the output (op_a), (next iteration), is the summation of them.

Op_a = newpd_e + add_b

4. Simulation Results

The functional simulation is done by using **ModelSim 5.5e.** The simulation example is performed on one segment of the degraded image with size 4x4 and the number of iteration is 40.

107	**128**	**127**	**126**
129	**155**	**157**	**157**
127	**158**	**157**	**156**
128	**155**	**156**	**157**

The iterative restoration algorithm computes the restored values of the degraded center pixels (155, 157, 158, 157) after 40 iterations. The restored pixel values are (172, 166, 168, 155) respectively. The functional simulation results are shown in the waveform as illustrated in Fig. 9.

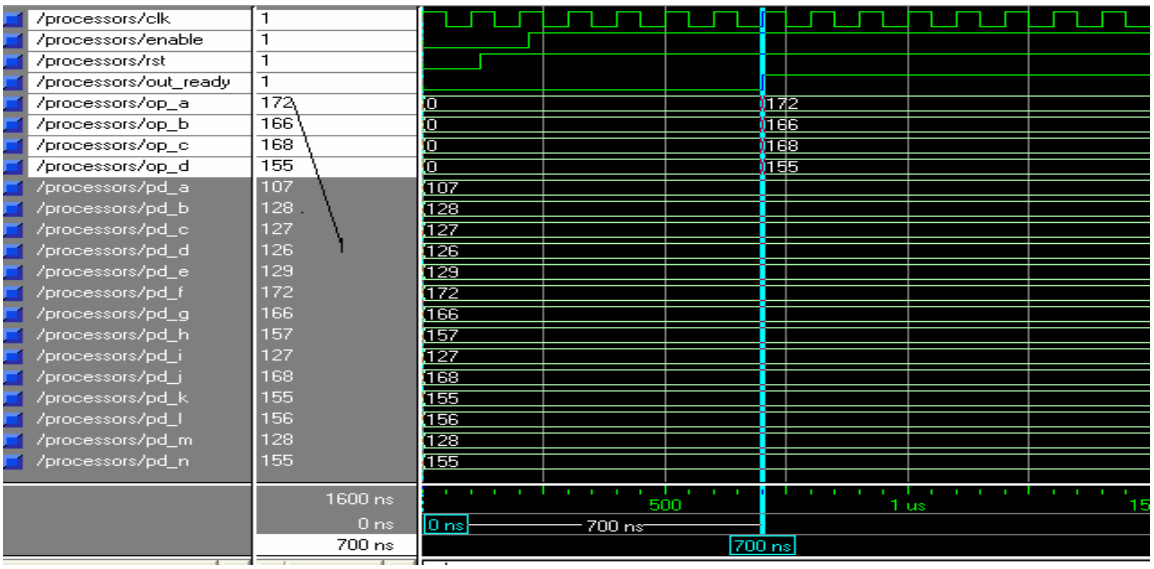


Fig. 9. Functional simulation waveform

From Fig. 9. It is noticed that the output is exiting exactly with the clock (clk) rising edge i.e., there is no delays in output values since the simulation is a functional (ideal) simulation.

The design Synthesis is done using **Leonardo Spectrum**. Xilinx-SpartanII-xc2s100pq208 was selected as a download target device to download the iterative algorithm design. A 12 MHz clock rate was selected as a time constraint for the design where, the available clock source, which is located on the SpartanII-xc2s100pq208 development board, is 25 MHz [12]. Table 2. Shows the area report (Hardware utilization for FPGA) and Table 3. Shows the time report.

Table 2. FPGA area report for iterative algorithm design

	Used	Available	Utilization
CLB_s	255	1200	21.25 %
IO_s	7	140	5.00 %
Function Generators	509	2400	21.21 %
Dffs or Latches	483	3120	15.48 %

Table 3. FPGA time report for iterative algorithm design

Data required Time (n sec.)	82.48
Data arrival time (n sec.)	13.06
Slack (n sec.)	69.42

The time report indicates that the data arrival time is **13.06 n sec;** this means that the processing time of one segment after one iteration takes **13.06 n sec** (ideal simulation). The data arrival time is not a real arrival time since, there is no time delays are taken into consideration. The design is then converted into its gates level to be ready for timing simulation (real simulation).

The *place and route* step is done using **Xilinx ISE5.2i** in order to make the design suitable for the timing simulation where time delays between different gates are taken into consideration. After timing simulation of the previous functionally simulated data, it is noticed that the output is not exiting exactly with the clock (*clk*) rising edge as illustrated in Fig. 10. But delayed by **14 n sec** this means that the real output data arrival occurred after **27.06 n sec** not in **13.06 n sec** as in functional simulation.

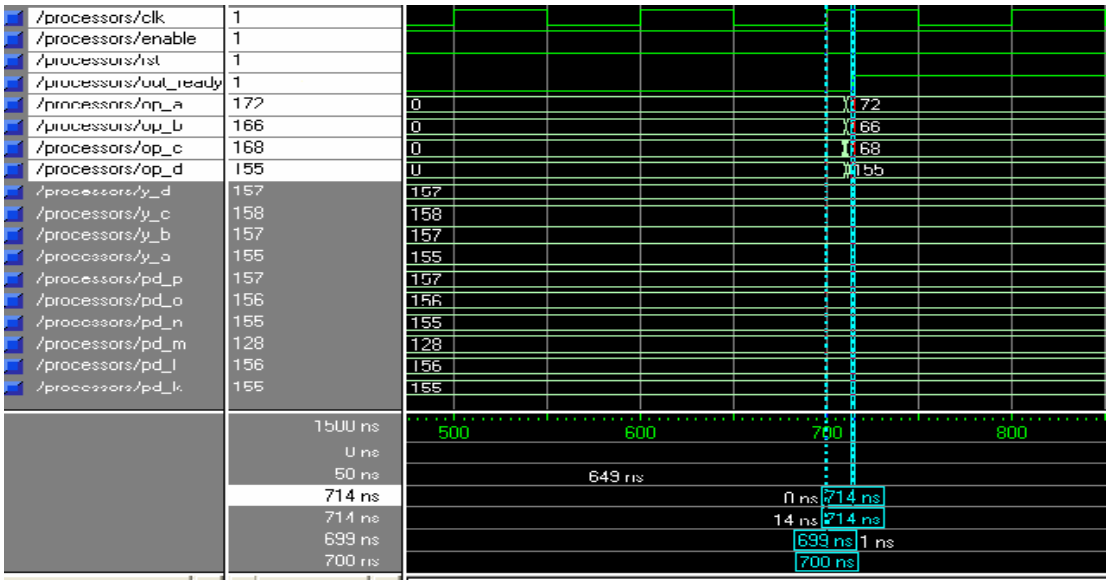


Fig.10. Timing simulation waveform

After timing Simulation the total processing time (T) of the whole image can be calculated using the following formula.

T = (number of segments)*(number of iterations)*(27.06 n sec)

After finishing synthesis and timing simulation steps an EDIF (Electronic Digital Interchange Format) file is obtained, this file is used for downloading the design on the selected chip by the aid of **Xilinx ISE5.2i.**

5. Testing the iterative algorithm downloaded design

This step is necessary for testing the functionality of the FPGA after downloading the design on the FPGA chip. Input data should be entered to the FPGA pins by the designer and then observing the output predicted data in order to prove the correct functionality of the downloaded design on the selected FPGA chip. The testing procedures are performed by the aid of a serial port. A design of a UART (Universal Asynchronous Receiver/Transmitter) circuit should be added to the iterative algorithm processing circuit in order to test the correct functionality of that downloaded design. The UART design should be added in order to make the serial port suitable to transmit and receive data to and from FPGA.

The transmitted (Tx) and received (Rx) data by the PC to and from another destination is done through its internal UART so a design of a UART circuit should be included and downloaded with the iterative algorithm processing circuit in order to make the design adapted with the transmitting and receiving protocol. Fig. 11. illustrates the sequence of data flow from PC to FPGA chip via serial port and vice-versa.

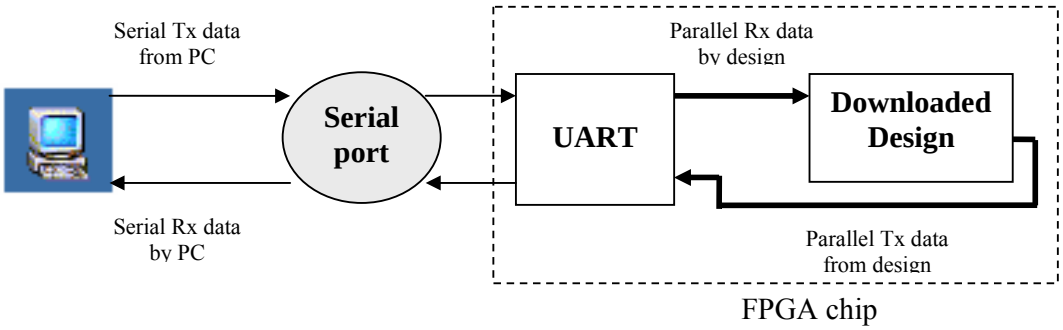


Fig. 11. Data flow from PC to FPGA chip and vice versa through serial port.

To test a whole image, it is sent to the FPGA chip through the serial port by using interfacing software (visual basic studio). The function of the visual basic software program is to read the whole degraded image as pixels and send these pixels serially to the FPGA chip, and received the restored output pixels from the FPGA chip. So The interfacing blocks are added to the iterative algorithm design as an interfacing design with the serial port to able us to send and receive the data to and from the serial port and the FPGA, as shown in Fig. 12.

The interfacing blocks are:

1- **Freq_ divider block and Clk_divider block**
2- **Controller block**
3- **Demux block**
4- **UART block**
5- **Selector block**

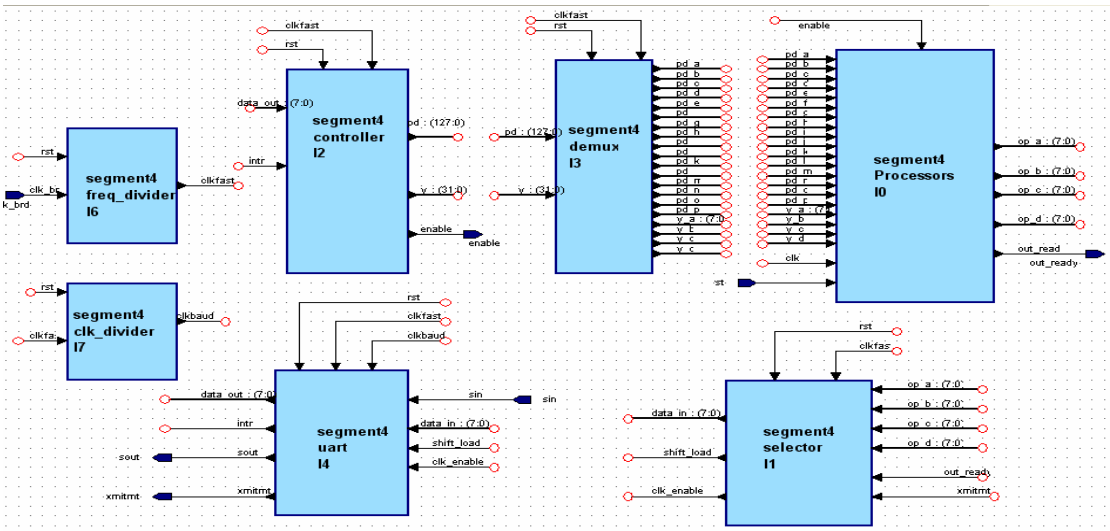


Fig. 12. The iterative algorithm with the interfacing blocks

The flow of the data in the iterative algorithm design with the interfacing blocks passes through two main modes: as shown in Fig. 12.

1. The receiving mode (The degraded image pixel values from the PC to the design).

2. The transmitting mode (The restored image pixel values from the design to the PC).

The receiving mode:

The degraded image is transmitted from the PC serially bit by bit to the UART receiver, which transfers the data (bit by bit) from serial mode to parallel mode (each 8 bit = 1pixel). The gathered pixels are sent to the controller pixel by pixel. When the controller receives the 20 pixels that are needed for the 4 processors to begin its function, the controller sends these 20 pixels as one line of data (20 pixel x 8 bits) to the demux. The demux separate them into 20 separated pixels ready to be processed by the 4 processors. The processors compute the new pixel values of the entered 20 pixels. When the 4 processors finish its function, the receiving mode finished and the transmitting mode begins.

The transmitting mode:

This mode begins after the processors finish restoring the 20 pixels and it needs to transfer the output of the processors to the PC. These output pixels are sent to the selector, which transfers them one by one to the UART transmitter and controlling the UART transmitter to transmit them serially without conflict in sending any signal. The function of UART transmitter is to transfer the data from parallel mode (each 8 bit =1 pixel) to serial mode (bit by bit) to be transmitted through the serial port to the PC.

5.1 The function of each interfacing blocks

To test the data by using the serial port we must add these interfacing blocks to the iterative algorithm design to able us to send and received the data to and from the serial port to the FPGA.

5.1.1 The function of the freq_divider and clk_divider

The ***freq_divider*** receives the clock of FPGA board **(*clk_brd*)** which is 12 MHz as illustrated in Fig. 12. and converts it to a clock of 4800 Hz **(*clkfast*)** which is sixteen times faster than the baud rate (300 Hz). The function of ***clkfast*** is to check the correctness of the input received data (*sin*) by the ***UART_Rx***. As shown in Fig. 13. With each clock rising edge of ***clkfast*** (at least eight rising edge), if the corresponding input data value is '**1**' then the input data **(*sin*)** certainly is '**1**' and vice versa. The ***clkfast*** can be named as data verification clock.

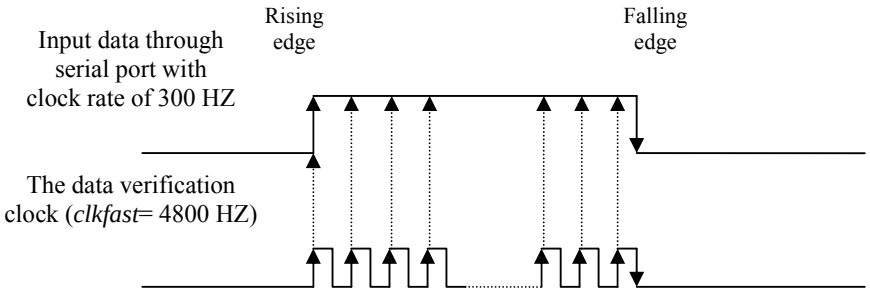


Fig. 13. Input data verification through *clkfast*

The ***clk_divider*** divides the ***clkfast*** which is 4800 Hz by 16 to produce 300 Hz **(*clkbaud*)** which is used for transmitting the output restored data from the iterative algorithm design circuit through the UART transmitter ***(UART_Tx)*** to the PC through serial port. The design procedures of the clock dividers **(*freq_divider* and *clk_divider*)** are written by VHDL code.

5.1.2 The function of the UART

The **UART** consists of receiver **(*UART_Rx*)** and transmitter **(*UART_TX*)** as shown in Fig. 14. The ***UART_Rx*** receives the serial data (*sin*) coming from PC via the serial port and converts them into parallel form **(*data_out*).** The ***data_out*** represents the input data to the iterative algorithm processing circuit. A star bit (**0**) and a stop bit (**1**) should be added to each eight bit of the input serial data (Pixel value). This means that the frame of input data consists of ten bits (a start bit, a stop bit and in between 8-bits which represent the input pixel value). The ***intr*** signal equals '**0**' during entering the bits of the pixel value (8-bit) and then converts to '**1**' at the end of entering the pixel value in order to declare that the pixel is ready to send to the controller block and this process is repeated until the 20 pixels is totally received pixel by pixel to the controller block.

The **UART** transmitter serially transmits the output data from iterative algorithm design via serial port to the PC, i.e., ***sout*** represents the output-restored data in a serial form. The ***data_in*** is entered to the ***UART_Tx*** and loaded in a register when ***shift_load*** signal equals '**0**' and ***clk_enable*** equals '**1**' (***clk_enable*** is considered the enable signal of the UART transmitter). The ***shift_load*** signal converts to '**1**' during data transmission process via a serial port. The ***clk_enable*** and ***shift_load*** signals become '**0**' at the end of each data transmission. These control signals **(*clk_enable, shift_load*)** are controlled by the selector block. The ***xmitmt*** signal works as a flag where, it equals '**0**' through data transmission from ***UART_Tx*** to PC and equals '**1**' at the end of data transmission which enables the selector that the UART transmitter finished sending the data (the first output pixel) and ready to send another pixel. The design procedures of the receiver and transmitter of the UART are written in VHDL code.

Fig. 14. Block diagram of UART Rx / Tx

5.1.3 The function of the controller block

It receives the **(data_out)** from the **(*UART_Rx*)** with ***(intr)*** signal to declare that the data is ready. The controller gathers the segment into two main ports (pd, y) and passes them to the demux block.

5.1.4 The function of the demux block

It receives the 2 ports (pd, y) and transfer them into 20 separated pixels each pixel is 8 bit and passes them to the processors to begin processing the segment.

5.1.5 The function of the selector block

The main function of selector is to receive the outputs from the processors and passes them one by one to the ***(UART_Tx)*** and controlling the timing of sending the data by 2 controlling signals ***(shift_load, clk_enable)*** to be sure that no conflict between the transmitted data.

6- Experimental Results

The analysis is performed on 10 satellite sub-images. The images format is TIF, gray scale of 8-bit, and the images size is 256X256 pixels. The analysis programs are performed using C++ program with a PC whose configuration is (Pentium IV, Intel 2.6 GHz processor, 512 Mbytes cash memory, 256 Mbytes RAM).

The processing time (T) to restore the whole image using FPGA can be calculated using the following formula.

T = (number of segments)*(number of iterations)*(27.06 n sec)

T= 4096 * 40 * 27.06 n sec ~ 4.4 m sec

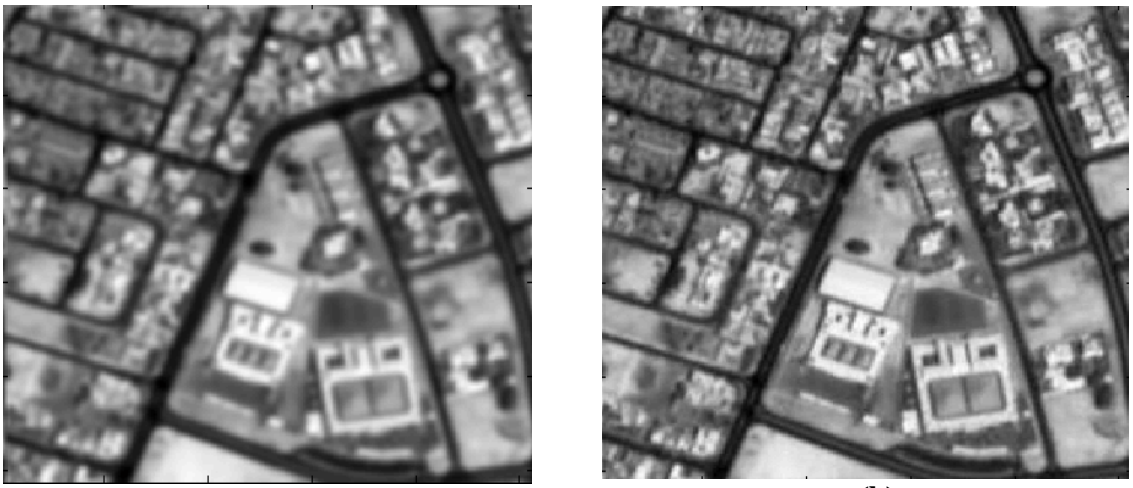
Restoration of the same image using the C + + program software takes 100 m sec. Therefore , the restoration algorithm can be executed 22 times faster on hardware than software.

Fig. 15. shows an example of the output-restored image from FPGA with size 128x128 pixels after 40 iteration by using visual basic interfacing program between the PC (the whole image) and the FPGA board (the design).

(a) **(b)**

Fig. 15. (a) Degraded image (b) Restored image from FPGA

Fig. 16. shows an example of restored image from the software after 40 iteration.



(a) **(b)**

Fig. 16. (a) Degraded image (b) Restored image from C++ program

7- Conclusion

- The use of a larger capacity chip and larger segment size lead to the CLB utilization approaches 100 percent and the highest clock speed at which the design can run will decrease. In this manner, for every available programmable hardware platform, the optimal implementation can be found.
- This work shows that certain digital image processing algorithms needed to go through modification in order comply with FPGA.

- The hardware implementation on FPGA provides a speedup or a faster solution than that can be achieved by implementing this algorithm on software.

- For each m x n segment, an overlap of one pixel needed to be performed to obtain high image quality.

References

[1] Rafel C.Gonzalez, Richard E.Woods, "*Digital image processing*", Second edition, university of Tennesse, MedData Interactive, (2000).

[2] F.S.Ogrenci, K.Bazargan, and M.sarrafzadeh,"Analysis and FPGA Implementation of Image Restoration Under Resource Constraints". IEEE (2003).

[3] F.S.Ogrenci, K.Bazargan, and M.sarrafzadeh,"Image Analysis and Partitioning for FPGA Mapping ". IEEE (2002).

[4] P.M. Athanas and A.L. Abbot, "Real-Time Image Processing on a Custom Computing Platform," Computer, vol. 28, no. 2, pp. 16-24, Feb. (1995).

[5] P. McCurry, F. Morgan, and L. Kilmartin, "Xilinx FPGA Implementation of a Pixel Processor for Object Detection Applications," Proc. Irish Signals and Systems Conf., (2000).

[6] Designing with FPGA Advantage, Mentor Graphic, student workbook, software V5.2, January (2002).

[7] A.K.Katsaggelos."Iterative Image Restoration Algorithms". Optical Engineering, Vol.28, pp.735-748 July (1989).

[8] Reginald L. Lagendijk and Jan Biemond,"Handbook of Image and Video Processing". (2000).

[9] B. Wilkinson, M. Allen. Parallel Programming. Prentice Hall, pp.335-338, (1999).

[10] L. Wanhammar. DSP Integrated Circuits. Academic Press, pp.371-379, (1999).

[11] J.Russ.The Image Processing Handbook.CRC press, IEEE press, pp.353-357, (1999).

[12] Memec Spartan II "LC Users Guide V1.0", July 21,(2003).