SEMARAK ILMU PUBLISHING

Semarak International Journal of Electronic System Engineering

Journal homepage:
https://semarakilmu.my/index.php/sijese/index
ISSN: 3030-5519

Development of Digital Image Processing Algorithms via FPGA Implementation

Shamsiah Suhaili[1,*], Joyce Huong Shing Yii[1,♣], Asrani Lit[1], Kuryati Kipli[1], Maimun Huja Husin[1], Mohamad Faizrizwan Mohd Sabri[1], Norhuzaimin Julai[1]

[1] Department of Electrical & Electronics Engineering, Faculty of Engineering, Universiti Malaysia Sarawak, 94300 Kota Samarahan, Sarawak, Malaysia

ARTICLE INFO

Article history:
Received 3 July 2024
Received in revised form 26 August 2024
Accepted 3 September 2024
Available online 15 September 2024

Keywords:
FPGA; image processing; MATLAB; Verilog HDL

ABSTRACT

A real-time image processing is one of the fundamental elements in achieving IR 4.0. The rapid development of digital image processing techniques has enabled various applications in fields such as healthcare, transportation, and manufacturing. People are seeking higher-performance image processing as traditional image processing is no longer fulfilling the demands. FPGA-based digital image processing has become one of the choices for the public due to its parallel pipelining, which enables shorter processing time and better performance. Several digital image processing algorithms have been developed in this project, which are gray level transformation, brightness manipulation, contrast adjustment, thresholding, and inversion. They are the most popular algorithms used in digital image processing. Microsoft Paint is used to convert the format of the color input image to bitmap format, followed by MATLAB to convert it into a hexadecimal file to be read and written in FPGA. Platforms such as ModelSim Altera and Intel Quartus II are used to write Verilog HDL for digital image processing algorithms. As a result, five hexadecimal files are obtained from the simulation. The output hexadecimal files are further processed in MATLAB to generate respective images.

1. Introduction

Technological innovation is taking place and new technologies are introduced every year. Now, the global is focusing on the latest trend, which is the Fourth Industrial Revolution (IR 4.0). It describes how digital technologies are incorporated into industrial and manufacturing processes. It also offers an overview of the complete networking of smart digital systems. Embedded systems can be said to be a technological foundation for realizing IR 4.0. This project is focusing on developing image processing algorithm on FPGA implementation. When it is used on medical images, it aligns with SDG

* *Corresponding author.*
E-mail address: sushamsiah@unimas.my
♣ *Corresponding author.*
E-mail address: 75219@siswa.unimas.my

https://doi.org/10.37934/sijese.3.1.2845b

Goal 3 which is good health and well-being. This is because through FPGA, the efficiency of the image processing is increased, which enhances the diagnosis capabilities and further contributes to improved healthcare outcomes.

FPGA is a semiconductor device that allows users to define its function after it is being manufactured. It can be programmed and reprogrammed to suit the different functionality of the device. Due to its architecture, FPGA allows parallel pipelining, which results in faster execution, higher efficiency and better performance [1]. Digital image processing describes the use of a digital computer to process digital images. Through this process, an image is enhanced and useful information in the image is extracted by means of mathematical models and algorithms [2]. There are numerous techniques used in digital image processing. The fundamental algorithms include grayscale conversion, brightness manipulation, contrast adjustment, thresholding and inversion.

1.1 Literature Review

Study by Vanaparthy *et al.,* [3] presented real time hardware image enhancement techniques using FPGA. It focused on using FPGA to create image enhancement algorithms such as brightness control, contrast stretching, negative transformation, thresholding and filtering techniques. Chiuchisan and Geman [4] proposed a method for implementing image enhancement techniques using FPGA technology to improve the diagnosis of skin cancer through computer-aided diagnosis. In medical imaging applications, digital image processing is essential because it can enhance image quality and extract data that is needed for a more precise diagnosis. There are different aspects of digital image processing, which includes the digitalization of images, compressing images, enhancing image quality, restoring images, matching images, describing images and reconstructing images. Algorithms such as contrast operation, brightness operation, gray level transform, inversion and thresholding are used to enhance the images of skin lesion (tumour). Zhang *et al.,* [5] focused on designing a colour to grayscale converter using a low-cost OV7670 camera. The low-cost camera was used to capture images, which were then processed, and the output was shown on a monitor through a Video Graphics Array (VGA) connector. Verilog HDL was used to write the modules, which are Camera Controller, Image Capture and VGA Master for the FPGA interfaces. Paper by Panappally and Dhanesh [6] discussed the design and implementation of GPU specifically for image processing. The GPU is designed to support 4-stage pipelining, which includes fetching instructions from memory, decoding the instructions, executing the operations on the data pixels and storing the processed data back to memory.

The researcher named Chaithra *et al.,* [7] focused on the use of Verilog HDL and MATLAB to enhance images in real time. It highlights the benefits of using FPGAs as an image processing tool instead of DSPs. The implementation of image enhancement techniques using Verilog HDL on FPGA differs significantly from MATLAB or DSP based image processing due to the parallel nature of HDL. The image enhancement algorithms are implemented on FPGA. It was simulated with Isim from Xilinx ISE Design Suite 14.3 and synthesized with Xilinx XST. The input image of bitmap file is first converted into hexadecimal format using MATLAB. It is then transferred into Xilinx ISE to read the image. The study by Dhanabal *et al.,* [8] focused on using Verilog HDL to apply circle recognition and coin counting through adjustments to a digital image's brightness, threshold and contrast. FPGA based hardware image processing is a good method to enhance the quality of image due to its reconfigurable feature and low manufacturing cost.

Study by Nived *et al.,* [9] focused on the usage of Verilog HDL and FPGA for real time configurable image enhancement. Verilog HDL's parallel nature makes it different from MATLAB or DSP based methods when it comes to implementing image enhancement algorithms on FPGA. The FPGA based

hardware implementation offers faster processing compared to software execution. Azhari *et al.,* [10] focused on implementing image enhancement on the hardware to improve the recognition speed to achieve real-time plate recognition. The input image is converted from bitmap format into a hexadecimal file. MATLAB is used to convert the bitmap image into hexadecimal format. Every RGB pixel value is kept in the Input Image Memory block. Then, the Brightness and Contrast Adjuster block modifies each fetched pixel value from the memory block. The result is later stored into the Output Image Memory block and transformed into bitmap format. Chiuchisan [11] focused on implementing and enhancing the real-time configurable system for image enhancement using Verilog HDL and FPGA. Medical image quality can be improved by performing various image processing operations such as edge detection, sharpening, contrast and brightness manipulation at the hardware level by designing new series of filters.

1.2 Significance of Study

The significance of this study lies in its potential to enhance image quality across various fields such as medical imaging, surveillance and remote sensing. For example, in automotive driver assistant systems, real-time video processing and analytics cannot be effectively performed by the traditional digital signal processing (DSP) processors due to their limited capabilities. Therefore, integrating the entire camera system in a single, low-cost System on a Chip (SoC) FPGA is a robust solution. The SoC FPGA's hard processor system (HPS) can run the software algorithm while the FPGA logic constructs hardware parallel processing engines, optimizing the overall system performance.

FPGAs are unique in their architecture, offering parallel processing capabilities unlike other hardware platforms such as CPUs and GPUs, which perform sequential instructions. For instance, in preparing an image for particle counting (Vision System Design), a convolution filter is applied to sharpen the image. The image is then run through a threshold to produce a binary image. Morphology is used to perform a close function, removing holes in the binary particles. On a CPU, the algorithm must complete each step sequentially across the entire image, taking approximately 166.7 ms. However, on a FPGA, each step is executed in parallel as each pixel completes the previous step, reducing the process to only 8 ms. This demonstrates that FPGAs can execute algorithms almost 20 times faster than CPUs. This is particularly beneficial for real-time applications where processing speed is crucial.

Furthermore, FPGAs allow for the creation of custom hardware architectures tailored to specific image processing tasks, optimizing performance and efficiency. This level of customization is difficult to achieve with fixed-function hardware. Partial dynamic reconfiguration is supported by certain FPGA chips. This makes it possible to use them more effectively. Reconfigurable computing benefits greatly from this capability because it allows for easy swapping of the modules into and out of the devices without resetting the entire device for a total reconfiguration [12]. The reconfigurability of the FPGAs makes it possible to update and modify image processing algorithms without changing the hardware, providing a flexible platform for development and experimentation.

Several previous research have been analyzed. Based on these research, three objectives have been developed, which are to develop different types of digital image processing algorithms using Verilog HDL, to verify the functionality of the design through simulation in ModelSim and MATLAB and to validate the performance of the image processing algorithms on an FPGA board. Based on the previous studies, the first objective has been defined by designing digital image processing algorithms such as grayscale conversion, brightness manipulation, thresholding, contrast adjustment and inversion using Verilog HDL. Different platforms are chosen for the project, which are Intel Quartus II and ModelSim Altera. The second objective, which is to verify the functionality of the design

through simulation in ModelSim and MATLAB, is set based on the outcomes in research [6-10,13]. The hexadecimal files outputted from the design can be generated back to images using MATLAB. Furthermore, studies in [5] provides insights for the third objective, which is to validate the performance of the image processing algorithms on an FPGA board.

There are two approaches for handling the image data in digital image processing on FPGA. The first approach inputs the image as a file, processes it in the FPGA and then outputs the processed image back as a file [7]. This approach makes use of FPGA's ability to store and manipulate digital data and results effectively, making it an ideal choice for batch processing and offline applications where real-time limitations are less important. On the other hand, the second approach utilizes a camera to capture live images, processes them on the FPGA in real-time and outputs the processed images straight to a monitor via a VGA interface [20]. This approach offers instantaneous visual feedback and is ideal for applications requiring continuous image processing. It usually demands more complex synchronization and timing considerations to ensure seamless operation.

2. Methodology

The focus of the project is to develop digital image processing algorithms using Verilog HDL for FPGA implementation. The proposed algorithms used in the project are grayscale conversion, brightness manipulation, thresholding, contrast adjustment and inversion. These algorithms are crucial for various image processing applications such as enhancing image quality, extracting important information and adapting images for different display conditions. They are important because they can be used in a variety of fields, including real-time video processing, surveillance systems, satellite imaging and medical imaging [11,14]. In this project, both Intel Quartus II and ModelSim Altera are used. Intel Quartus II is used for the synthesis and implementation of the design into the FPGA whereas ModelSim Altera is used to simulate the Verilog HDL-based design to verify the functionality of the design.

The input image is first converted into hexadecimal format before undergoing image processing by different image processing algorithms in Verilog HDL. Then, the image data undergoes several image processing algorithms such as grayscale conversion, brightness manipulation, thresholding, contrast adjustment and inversion. Later, the processed image data is output to their corresponding hex files. Using these hexadecimal files, their respective output images are generated through MATLAB. The processed data is then output in their respective hex files. By using Matrix Laboratory (MATLAB), the processed output images are generated from the hexadecimal files to Portable Network Graphics (PNG) format.

2.1 Grayscale Conversion

Grayscale conversion is the process of shrinking an image dimension from RGB to the intensity value (I) [15]. The pixel values of a grayscale image range from 0 (black) to 255 (white). This algorithm is used because a grayscale image requires less memory space, and the processing time is faster compared to a colour image. For example, this technique is widely used in medical imaging such as in X-rays, MRIs and CT scans. This is because the grayscale image offers a detailed view of the body's internal components and with the high resolution, even the smallest abnormalities, can be accurately diagnosed [16]. There are three methods of converting a colour image to a grayscale image. The first method is known as the lightness method [15], where it calculates the grayscale values by averaging the minimum and maximum of the RGB data as shown in Eq. (1).

$$Gray = \frac{\min(R,G,B)+\max(R,G,B)}{2} \quad (1)$$

However, the primary flaw of this method is that it does not use the RGB components that are three colours. The second method is known as the average method [15], where it obtains the grayscale values by averaging the values of the RGB data as shown in Eq. (2). This method has a flaw where it gives each of the RGB data the same weight, even though our eyes see colour differently.

$$Gray = \frac{R+G+B}{3} \quad (2)$$

The third method is known as luminosity method [15], where it calculates the grayscale values based on a weighted combination of the RGB data. In this method, the green channel gets the highest weight, followed by red and blue channels. In this project, the second method is chosen. The implementation of grayscale conversion starts by reading the input colour image and separating it into three different channels (red, green and blue). The grayscale data is obtained by summing up the RGB data of the same pixel and then divided by 3 as shown in Eq. (3). The *gray* output image is obtained.

$$Gray = 0.2989 \times R + 0.587 \times G + 0.114 \times B \quad (3)$$

2.2 Brightness Manipulation

Brightness manipulation is a basic operation of digital image processing. When the input is too bright, we would want to increase its brightness to increase its visibility. In contrast, if the image is too dull, brightness addition is crucial to make the image more visible for the users. Basically, brightness manipulation refers to adding or subtracting a constant value to each pixel of the image. Brightness can be increased by adding a value to each pixel of the image and vice versa. The mathematical expression for increasing brightness is shown in Eq. (4) whereas decreasing brightness is in Eq. (5) where *Brightness* is the new pixel value, *Gray* is the original pixel value and *x* is a constant brightness value. The implementation of brightness manipulation starts by reading the data of the grayscale image. The brightness data for both brightness addition and subtraction are obtained based on the respective mathematical expressions illustrated in Eq. (4) and (5). The brightness *Gray* output image is obtained.

$$Brightness = Gray + x \quad (4)$$

$$Brightness = Gray - x \quad (5)$$

2.3 Thresholding

Based on a specified threshold value, thresholding transforms a grayscale image into a binary image. The resulting image becomes black and white only. It is crucial in computer vision applications, such as surveillance and object detection. With a suitable threshold value, it can separate the objects from the background, aiding in automated recognition and tracking. It helps in analysis and decision-making. By referring to Eq. (6), if the initial *Gray* value is more than *x*, the new Threshold value will be set to 255 (white) whereas if it is less than or equal to x, the new Threshold value will be 0 (black). The variable *x* refers to the threshold value that determines the output pixel to be black or white.

The implementation of thresholding starts by reading the data of the grayscale image. The threshold data for thresholding can be obtained using the mathematical expression presented in Eq. (6). The threshold output image is obtained.

$$Threshold = \begin{cases} 255(white) & if\ Gray\ > x \\ 0\ (black) & if\ Gray\ \leq x \end{cases} \quad (6)$$

2.4 Contrast Adjustment

Image enhancement can be done by adjusting contrast to change the difference in intensity between the pixels of the image. By increasing the contrast, the dark pixels will become darker, and the bright pixels will become brighter. This operation can be done by multiplying each image pixel with a contrast factor. When contrast factor > 1, contrast is increased while contrast is decreased when contrast factor < 1. It is widely used in optical character recognition (OCR) systems and digitization projects to ensure the text and details are clear and legible. This enhances the accuracy of text extraction and overall document interpretation. Eq. (7) shows the mathematical expression for contrast adjustment where *x* is the contrast factor. The implementation of contrast adjustment starts by reading data of the grayscale image. The contrast data can be calculated using the mathematical expression displayed in Eq. (7) The contrast output image is obtained.

$$Contrast = x\ \times Gray \quad (7)$$

2.5 Inversion

Inversion involves reversing the intensity values of each pixel of an image. As a result, the dark pixels will become bright whereas the bright pixels will become dark. Inversion is useful in medical imaging for enhancing the visibility of certain structures or features within images. For example, by inverting the image in X-ray imaging, it can highlight certain features that may not be as apparent in the original image. This helps the doctors with a more comprehensive and accurate diagnosis and analysis. The maximum pixel value for a grayscale image is 255. The operation can be done by using mathematical expression in Eq. (8) to obtain the new *Invert* value by subtracting the initial *Gray* value from the maximum pixel value, *Amax*. The implementation of inversion starts by reading the data of the grayscale image. The invert data can be obtained based on the mathematical expression shown in Eq. (8). The invert output image is obtained.

$$Invert = Amax - Gray \quad (8)$$

2.6 Research Design

The first step is to conduct research and study information related to digital image processing algorithms. A lot of related studies and research have been reviewed and studied. The next step is to search for suitable hardware and software for image processing. For example, the model of the FPGA and the FPGA development tools for writing Verilog HDL code. The FPGA board used is the DE-10 Standard and the selected platforms are Intel Quartus II and ModelSim Altera. Next, it is crucial to find a suitable image for digital image processing.

The image “Lenna_colour.png” was chosen as it is a standard test image in image processing. The dimensions of the image need to be known beforehand as they are specified in the Verilog code when

reading and storing the image data. The dimension of the "Lenna_colour.png" is 512 x 512 x 3. Then, Microsoft Paint is used to convert the image from PNG format to bitmap format. Since the FPGA can only read hexadecimal and binary files, MATLAB is used to convert the bitmap image into hexadecimal format. Therefore, a MATLAB script is written to read the bitmap image and convert it into hexadecimal format. The hexadecimal file obtained is later inputted into the FPGA using Intel Quartus II and ModelSim Altera. Various Verilog HDL codes are written to design and perform different digital image processing algorithms on the hexadecimal data of the input image. A testbench is written to verify the functionality of the design. The Verilog HDL code was synthesized, and simulation was performed. The hexadecimal files output by the algorithms are post-processed in MATLAB to generate respective images. Another MATLAB script is written to convert the data in the hexadecimal files into PNG format. The output images generated are checked to see if they meet expectations. If the output images are not as expected, the Verilog HDL code is modified, and the simulation is performed again.

The block diagram for the digital image processing algorithm on FPGA implementation is displayed in Figure 1. A colour image "Lenna_colour.png", with dimensions 512 x 512 x 3, is chosen as the input. By using Microsoft Paint, it is converted to bitmap format, resulting in "Lenna_colour.bmp". It is then converted to hexadecimal format using MATLAB. The output files are "Lenna_colour.hex" and "size_file.hex".

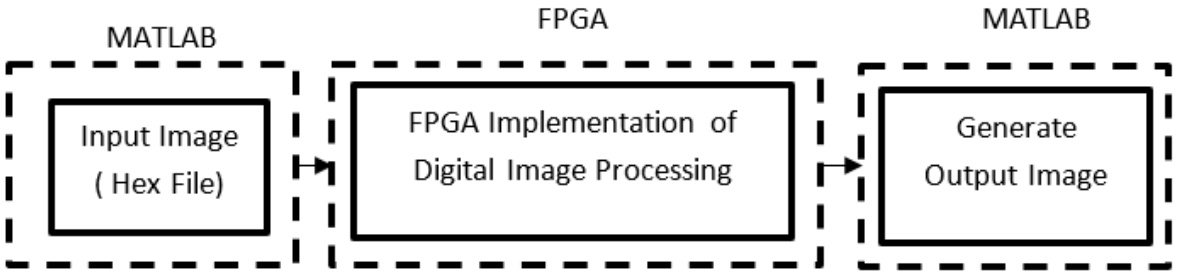


Fig. 1. Block diagram digital image processing algorithm on FPGA implementation

The RGB data from the input image is read by the FPGA and stored in the memory block via Intel Quartus II and ModelSim Altera. Verilog HDL is used to develop various digital image processing algorithms to process the image. The image processing algorithms applied to the input image include grayscale conversion, brightness manipulation, thresholding, contrast adjustment and inversion. The algorithms output the processed data in their respective hexadecimal files. The processed images are then generated and visualized using MATLAB, which converts the image data from hexadecimal format back to PNG format.

3. Results and Discussion

This section discussed the coding used to convert the colour image into hexadecimal format in MATLAB and Verilog coding used to develop various digital image processing algorithms. Then, the output hexadecimal files are converted back to images using MATLAB in post-processing of the image. The details regarding pre-processing of the image are discussed in this section. To effectively utilise the programme, it is necessary to possess prior knowledge of the FPGA board, Intel Quartus II, ModelSim, and MATLAB. This information can be obtained from the following articles [17-20]. Other images from other papers [21] can also be used to observe the results.

3.1 Preprocessing using Microsoft Paint

"Lenna" image is a standardized test image commonly used in digital image processing. The "Lenna" image obtained from the Internet is originally in PNG format and it is saved as "Lenna_colour.png". The dimension of the image is 512x512x3. Later, the image is converted into bitmap format using Microsoft Paint. PNG format is known for its loseless compression characteristics, which efficiently reduces the image file size without sacrificing the image quality. On the other hand, bitmap format is known for its loseless and uncompressed attributes, which stores the image data in pixel-by-pixel manner. This means that the colour of each pixel is stored in a matrix of bytes, allowing bitmap format image to be a preferable format in image processing tasks. Figure 2 shows the "Lenna_colour" image in PNG and bitmap formats respectively.

(a) (b)

Fig. 2. "Lenna_colour" image in (a) PNG format (b) bitmap format

3.2 Pre-processing using MATLAB

Since the FPGA cannot read images directly, conversion of image from bitmap format to hexadecimal format is necessary. FPGA can only read image data in binary and hexadecimal formats. In this project, MATLAB is used to convert the pixel values of the bitmap image into a hexadecimal data file. This is to ensure that the image data is compatible to be transferred and processed within the FPGA.

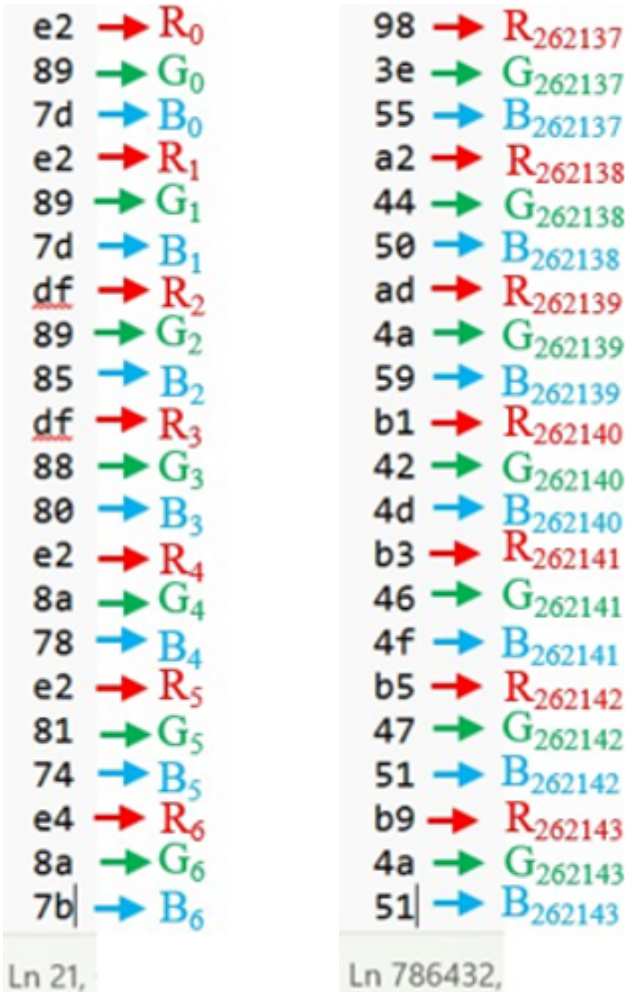
3.2.1 MATLAB coding explanation

The image used in this project is "Lenna_colour.bmp". It is a colour image with an image size of 512 x 512 x 3. This means that both the width and height of the image are 512 and it consists of 3 channels (Red, Green and Blue). The MATLAB coding loads and reads each pixel of the "Lenna_colour.bmp" image. It then converts the RGB values of the image into their corresponding hexadecimal representations. The resulting hexadecimal data is written sequentially to a text file named "Lenna_colour.hex". The image size is also written a text file named "size_file.hex". The image size file will be later used in post-processing to convert the hex data to image. Since the "Lenna_colour" image is a colour image, it consists of three colour channels, known as RGB channels. Therefore, in the hexadecimal file, there are 512 x 512 x 3 = 786432 hexadecimal values. These values

represent the colour information for each pixel in the image. The hexadecimal data file is then read by the Verilog code in Quartus II for FPGA-based digital image processing.

3.2.2 Output of hex files

The hexadecimal data is stored in a text file named “Lenna_colour.hex”. Another text file called “size_file.hex” is created to store the image size data. Figure 3 shows the output of the first and last 21 hexadecimal values that corresponds to the RGB values of a single pixel in “Lenna_colour.hex”. Therefore, the 21 hexadecimal values shown represent 7 sets of RGB values from the beginning and end of the image. Figure 3 illustrates the hexadecimal data stored in “size_file.hex”.



(a) (b)

Fig. 3. First (a) and last (b) for 21 RGB hexadecimal data in “Lenna_colour.hex”

3.3 Image Processing using Intel Quartus II

In Quartus II, the design is compiled. The compilation process involves analysis and synthesis, fitter, assembler, timing analysis and Exploratory Data Analysis (EDA) netlist writer. The Verilog code is checked for any syntax errors and is then converted into a gate-level representation that consists of logic gates and flip-flops. An RTL design is generated, showing the logic circuits and data flow of

the design. In the Pin Assignment, the ports are assigned to respective pins to allow the data to flow into and out of the FPGA.

3.3.1 Verilog coding explanation

The Verilog coding is separated into design part and testbench. The design part consists of six modules, which are the Top Module, Gray Module, Brightness Module, Threshold Module, Contrast Module and Invert Module. The Verilog coding for the design in Quartus II must be synthesizable to be able to generate RTL design and program it into the FPGA hardware. The Verilog coding are designed so that the input RGB data from the input image file "Lenna_colour.hex" are processed every clock cycle. This results in a delay and replication of the hex data in the output hexadecimal files.

The testbench file, "design_module_tb" is used to verify the functionality of the "design_module". It declares all the parameters, wires, registers and integers used for the testbench shown in the blue box. It initializes the 'clk' and 'reset' signals shown in the orange box. It reads the RGB hex data from "Lenna_colour.hex" and stores the data in 'in' shown in the red box. It instantiates the "design_module" and connects with the signals in the testbench, allowing the top-level module to interact with the testbench environment.

3.3.2 Compilation report

Figure 4 shows the hierarchy of the design. The top module is "design_module" and the submodules are "gray_module", "brightness_module", "threshold_module", "contrast_module" and "invert_module". The compilation tasks and time are shown in Figure 5. A summary of the compilation report is presented in Figure 6.

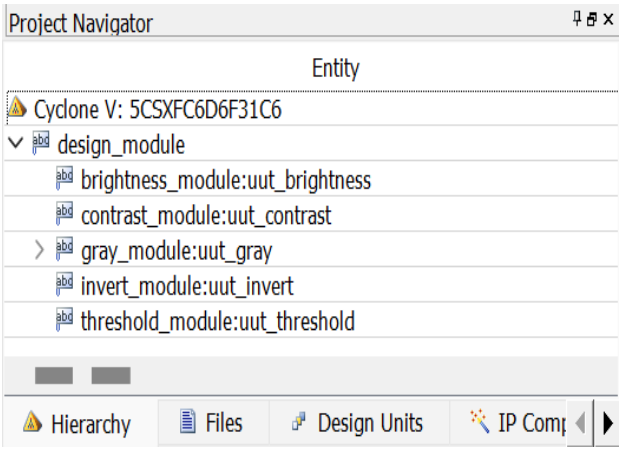


Fig. 4. Project navigator

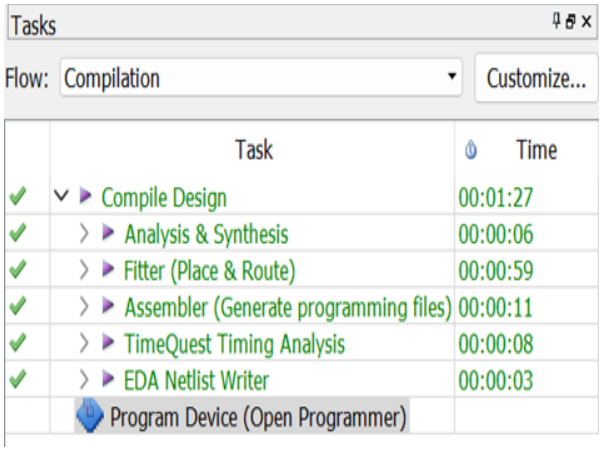


Fig. 5. Compilation tasks and time

Compilation Report - design_module

Flow Summary	
Flow Status	Successful - Sat Jun 22 16:40:13 2024
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Web Edition
Revision Name	design_module
Top-level Entity Name	design_module
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Preliminary
Logic utilization (in ALMs)	94 / 41,910 (< 1 %)
Total registers	77
Total pins	51 / 499 (10 %)
Total virtual pins	0
Total block memory bits	0 / 5,662,720 (0 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI TX Channels	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	0 / 4 (0 %)

Fig. 6. Compilation report

3.3.3 RTL design

The RTL design describes the operation of the digital circuit by outlining the data flow between the registers and the logical operations performed on the data. It is an essential step in the hardware development process, providing a blueprint for the synthesis and implementation phases. RTL designs are generated from the Verilog coding after performing the compilation process in Quartus. Figure 7 represents the RTL schematic of the Top Module, "design_module". It provides a visual representation of the hierarchical structure, showing how the data flows through different components. It also shows the interconnections between the inputs, outputs, internal wires, registers and submodules used in the design. It highlights the integration of different submodules, each contributing to the overall image processing pipeline.

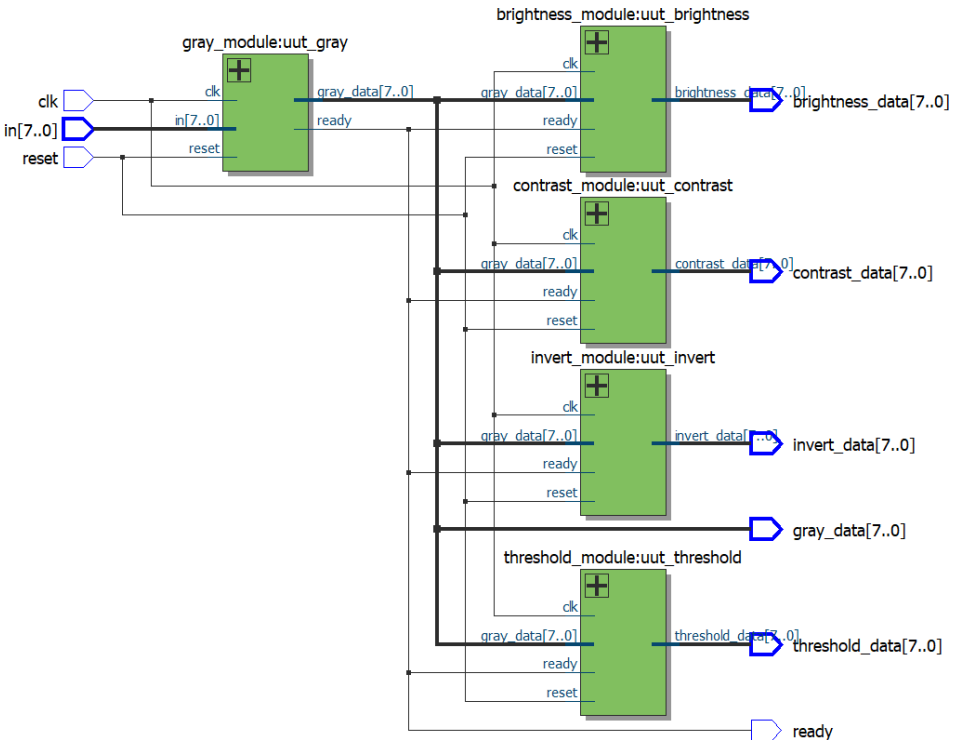


Fig. 7. RTL design for "design_module"

3.3.4 FPGA implementation

Several steps are required before implementing the design in FPGA to ensure the functionality and integration with FPGA. The input and output ports are assigned to the designated pins in Pin Planner. Programmer is used to detect the FPGA hardware and upload the Verilog coding into the FPGA.Pin Planner is used to map the input and output ports of the design to the physical pins on the FPGA device. This is to ensure that the FPGA can interact correctly with the external components and signals. The pin assignment can be found in DE10-Standard User Manual. The 'clk' is assigned to CLOCK_50, with 50 MHz clock input.

In the Programmer, the FPGA is configured in JTAG mode. In the 'Hardware Setup', 'DE-SoC [USB-1]' is chosen. Next, the button 'Auto Detect' is selected and the device '5CSXFC6D6'. Both the FPGA and HPS are shown in the Programmer interface. The FPGA is right clicked to change the device to 5CSXFC6D6F31. The FPGA is right clicked again to change the file and the output files "design_module.sof" is chosen. In Figure 8, the box 'Program/Configure' in the .sof file is clicked and the button 'Start' is clicked to download the .sof file into the FPGA. The '100% (Successful)' indicates that the Verilog coding has been successfully uploaded into the FPGA board.

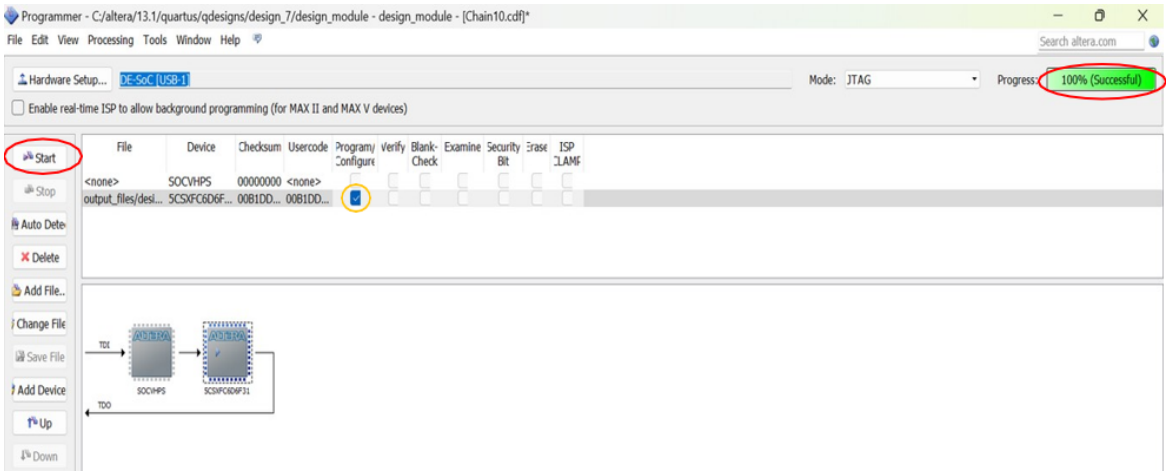


Fig. 8. Programmer interface for programming the .sof file into the FPGA

Figure 9 shows the data captured by the FPGA when the Verilog coding is implemented on the FPGA. The 6 7- segment displays capture the 'red', 'green' and 'blue' data of 'e2', '98' and '7d' respectively when a stream of input image data flows into the FPGA. The LEDs output the binary number 10000010 of the 'gray_data' from the averaging of the 'red' 'green' and 'blue' data captured. Figure 10 displays the connection between the FPGA board and the laptop and the monitor of the laptop showing the progress of the program uploadinto the FPGA.

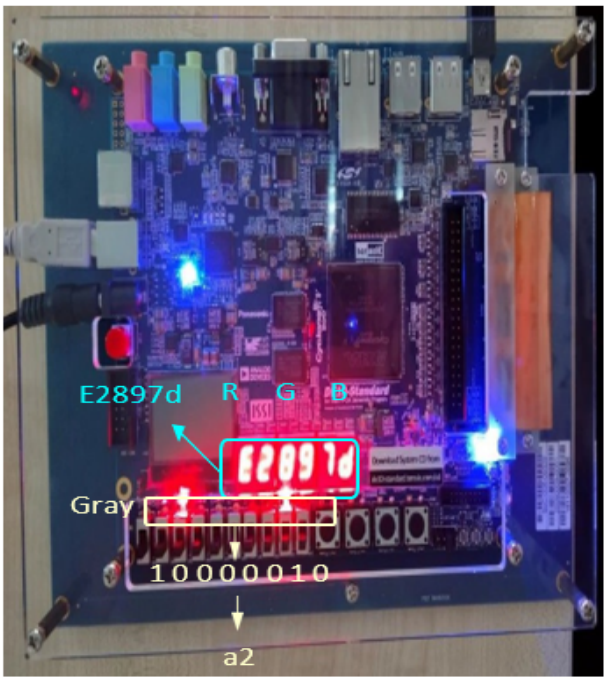


Fig. 9. The data captured by the FPGA

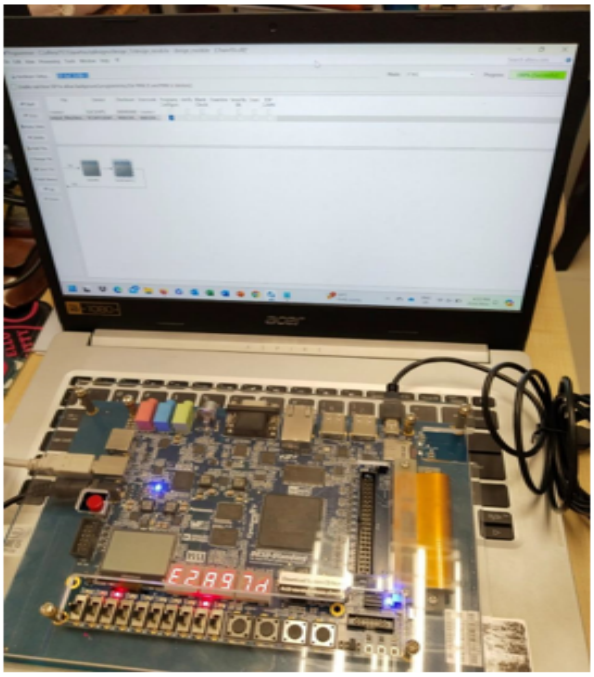


Fig. 10. The connection between the FPGA board and laptop

3.3 Image Processing using ModelSim Altera

In ModelSim, functional simulation is performed to verify the functionality of the Verilog coding used for digital image processing. Figure 11 displays the output waveforms for the design and testbench for the simulation from 0 ps to 251000 ps. Initially, the 'clk' and 'reset' signals are set to low. The 'reset' signal is set to high after a clock cycle and set to low again after another clock cycle. The "gray_module" starts to read the hexadecimal data from the image file "Lenna_colour.hex" at the next rising edge of the clock. Then, the 'ready' signal is high when the 'gray_data' for each pixel is available.

In Figure 11 the yellow box highlights the output data for each submodule in the beginning of the simulation. The 'gray_data' is passed to the "brightness_module", "threshold_module", "contrast_module" and "invert_module" for further image processing. Since the always blocks in the submodules are triggered by the change in 'gray_data', there is no delay between the data transfer for the "gray_module" with other submodules. This ensures that the 'brightness_data', 'threshold_data', 'contrast_data' and 'invert_data' are updated in the same clock cycle as the 'gray_data'.

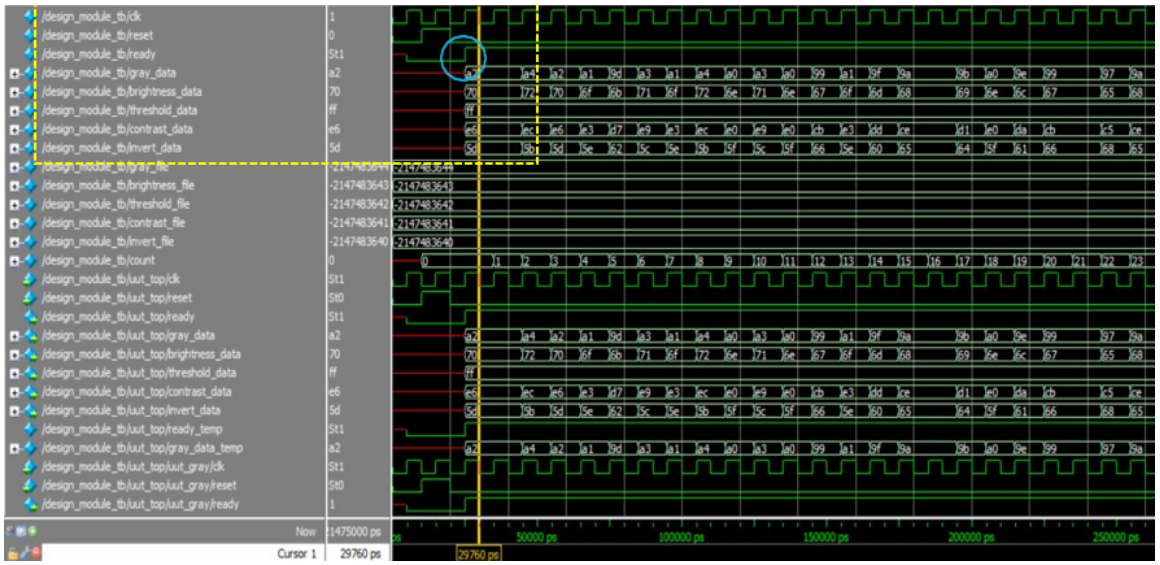


Fig. 11. Output simulation waveforms in the beginning

Figure 12 displays the output simulation waveforms for the design and testbench from 2621225000 ps to 2621475000 ps. The 'ready' signal goes low when all the 'gray_data' has been successfully obtained from the averaging of the RGB data input. The integer 'count' tracks the pixel number of the grayscale image and stops the simulation when the file write operations for each submodule is complete. The simulation automatically stops when the 'count' reaches 262144 and with a clock cycle delay. This is because the always block in the testbench is triggered by the rising edge of the 'clk' and 'reset' signals.

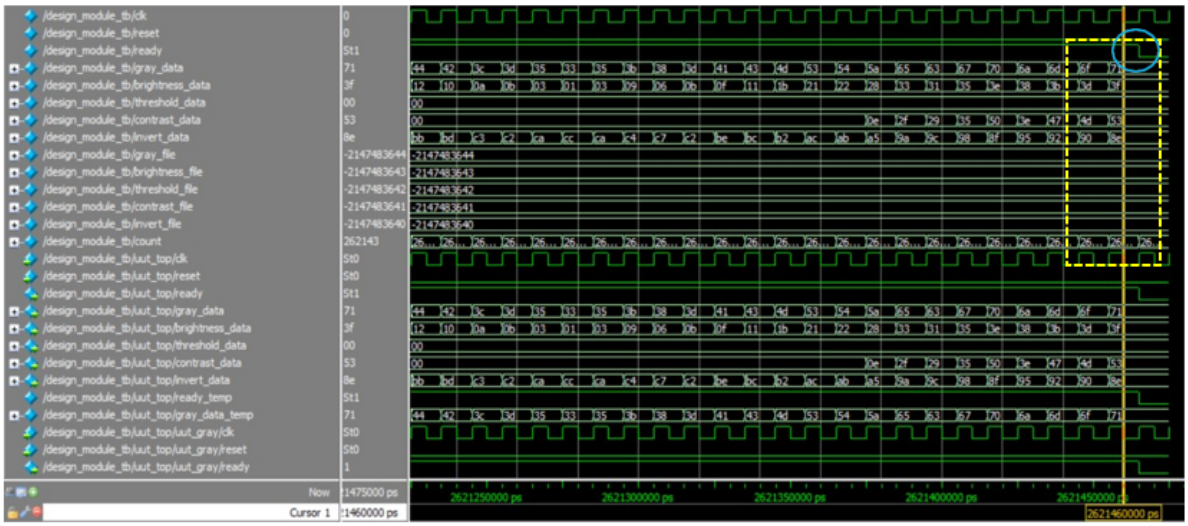


Fig. 12. Output simulation waveforms at the end

Figure 13 highlights the output waveform of the "contrast_module". The 'contrast_data_temp' is used to tackle the overflow that might happen in the "contrast_module". When the 'contrast_data_temp' is a negative value, the 'contrast_data' is 00.

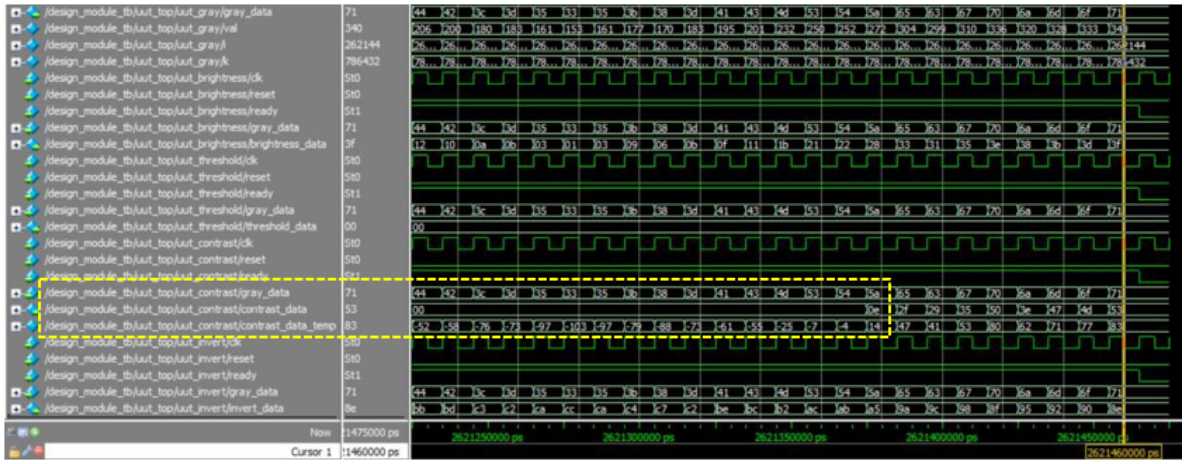


Fig. 13. Output waveforms of "contrast_module"

3.5 Post-processing using MATLAB

After running the simulation in ModelSim, five hexadecimal files are outputted. The 'gray_data', 'brightness_data', 'threshold_data', 'contrast_data' and 'invert_data' are written to their respective files. Figure 14 shows the block diagram of Design Module with input hex file and generation of output hex file from Verilog HDL code after simulation process.

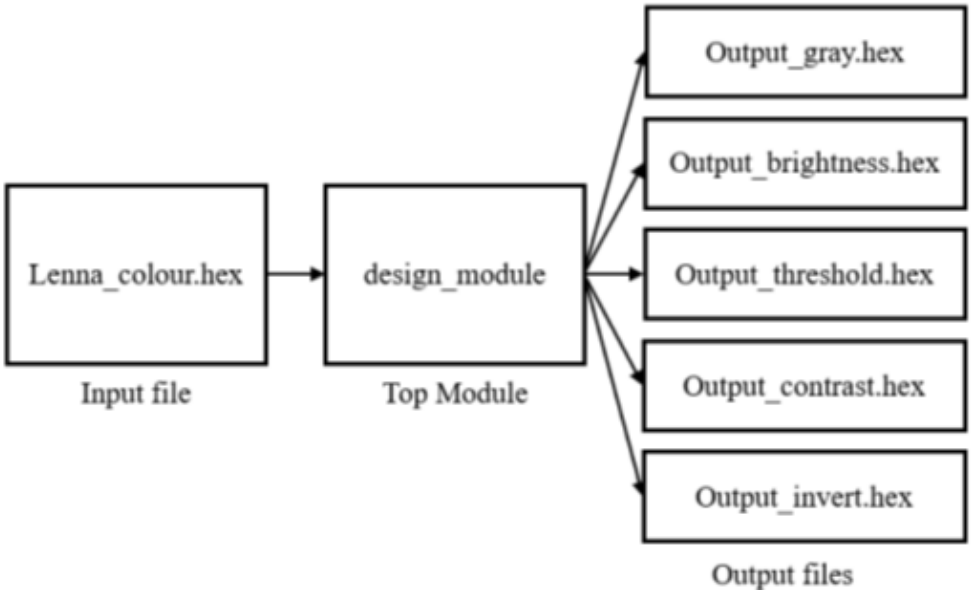
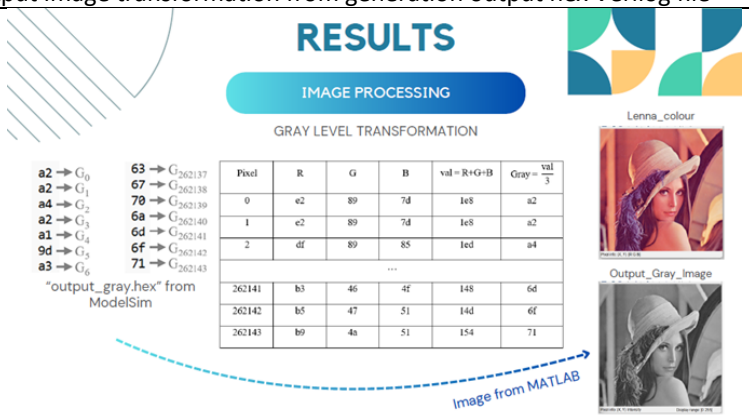
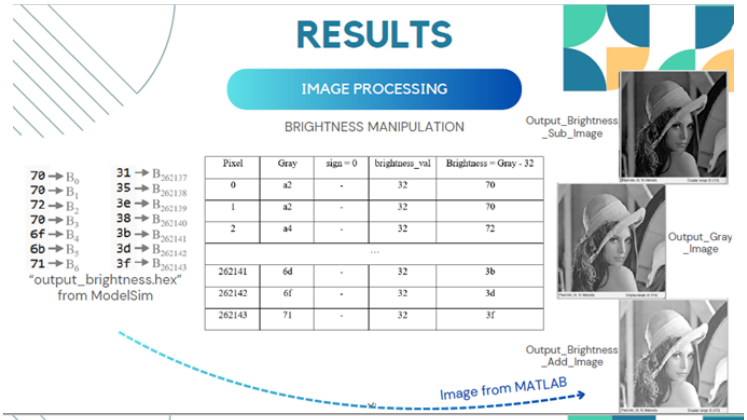
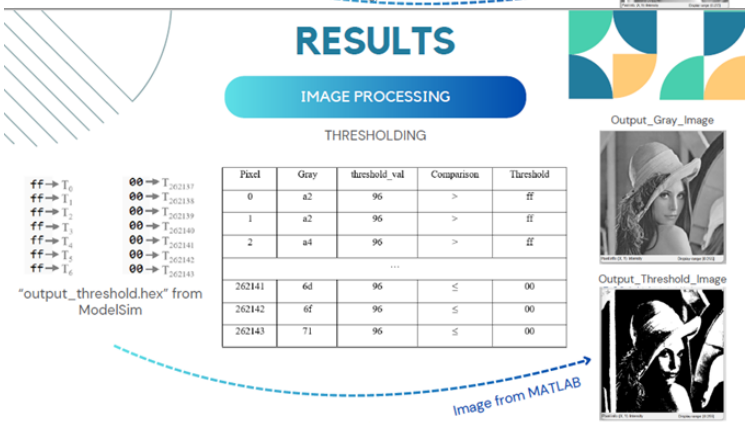
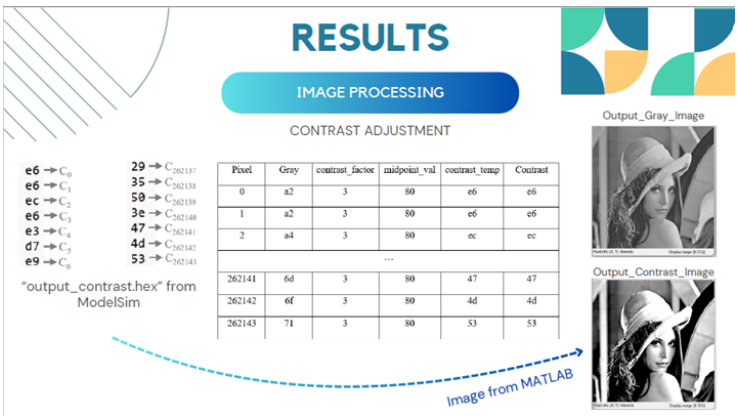


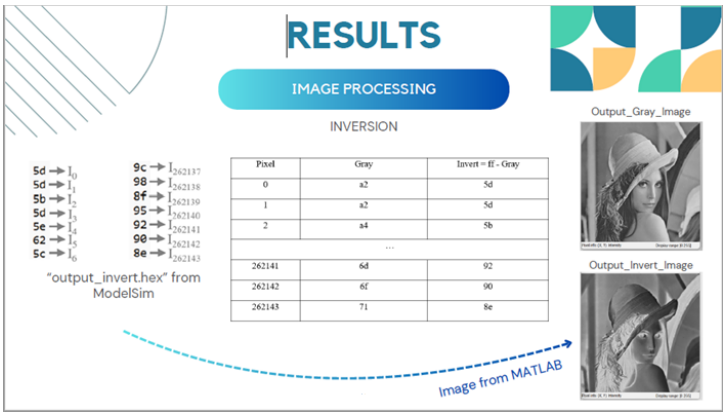
Fig. 14. Block diagram with input and output file of design module

In this stage, MATLAB is utilized to convert the hexadecimal data outputted by ModelSim into visible images. This step is crucial for visual inspection and verification of the results of the simulation performed in ModelSim. A total of five images are obtained from five different hexadecimal files. Table 1 shows the transformation of five different types of hexadecimal output result.

Table 1
Image transformation from generation output hex Verilog file

Transformation	Output image transformation from generation output hex Verilog file
Gray Level Transformation	
Brightness Manipulation	
Thresholding	
Contrast Adjustment	

Inversion



4. Conclusions

The digital image processing algorithms employed in this project are grayscale conversion, brightness manipulation, thresholding, contrast adjustment and inversion. The implementation of these algorithms in digital image processing relies on both hardware and software. FPGA is the only hardware used for implementing the algorithms. For software, Microsoft Paint, MATLAB, Intel Quartus II and ModelSim Altera are utilized. Microsoft Paint is used for converting the input image from PNG format to bitmap format. MATLAB is used for converting the bitmap image to hexadecimal format. Intel Quartus II and ModelSim Altera are used to design the algorithms in Verilog HDL. They are also used to simulate and verify image processing operations. MATLAB is used again to convert the output hexadecimal files to PNG format, where the images are visualized for comparison between the input colour image and the processed images.

The success of the project is assessed by the successful implementation of the digital image algorithms in Verilog HDL and integration into the FPGA. The developed algorithms are evaluated through simulation and verification using Intel Quartus II and ModelSim Altera to ensure the functionality of the design. Comparisons between the grayscale image with the input image and with the other algorithms are done to validate the correctness of the algorithms. Validation on the FPGA hardware is carried out to confirm the practical applicability and performance of the algorithms.

Acknowledgement

This research was funded by Universiti Malaysia Sarawak (Osaka Gas Foundation in Cultural Exchange (OGFICE) research Grant Scheme INT/FO2/IG-OSAKA/85047/2022).

References

[1] Moore, Andrew, and Ron Wilson. "FPGAs For Dummies®, 2nd Intel® Special Edition." (2017).

[2] Gonzalez, Rafael C, and Woods, Richard E. "Digital Image Processing, 2nd Ed. Prentice Hall." (2002).

[3] Vanaparthy, Praveen, G. Sahitya, Krishna Sree, and C. D. Naidu. "FPGA implementation of image enhancement algorithms for biomedical image processing." *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering* 2, no. 11 (2013): 5747-5753.

[4] Chiuchisan, Iuliana, and Oana Geman. "An approach of fpga technology in skin lesion detection."In *2018 International Conference and Exposition on Electrical And Power Engineering (EPE)*, pp. 0175-0178. IEEE, 2018. https://doi.org/10.1109/ICEPE.2018.8559866

[5] Zhang, Yunxiang, Xiaokun Yang, Lei Wu, and Jean H. Andrian. "A case study on approximate FPGA design with an open-source image processing platform."In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 372-377. IEEE, 2019.https://doi.org/10.1109/ISVLSI.2019.00074

[6] Panappally, J. George Cherian, and M. S. Dhanesh. "Design of graphics processing unit for image processing." In *2014 First International Conference on Computational Systems and Communications (ICCSC)*, pp. 299-302. IEEE, 2014. https://doi.org/10.1109/COMPSC.2014.7032666

[7] Chaithra, S., Nithya Priya, H.L., Pragna, V., and Spoorthy, M. "Enhancement of Image using Verilog & MATLAB." *International Research Journal of Modernization in Engineering Technology and Science (IRJMETS)*, vol. 5, no. 7, pp. 870–872, 2023, https://doi.org/10.56726/IRJMETS43063

[8] Dhanabal, R., Sarat Kumar Sahoo, V. Bharathi, Kalyan Dowluri, Bh SR Phanindra Varma, and V. Sasiraju. "FPGA based image processing unit usage in coin detection and counting." In *2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]*, pp. 1-5. IEEE, 2015. https://doi.org/10.1109/ICCPCT.2015.7159440

[9] Nived, Ch Sai, A. Rohith Kumar, G. Sai Dheeraj, and P. Jithendra. "Image Enhancement based on Verilog Hardware Description Language." *International Journal of Engineering Research and Technology*. Volume 14, Number 7 (2021), pp. 647-651, 2021.

[10] Azhari, Zul Imran, Samsul Setumin, Emilia Noorsal, and Mohd Hanapiah Abdullah. "Digital image enhancement by brightness and contrast manipulation using Verilog hardware description language." *International Journal of Electrical and Computer Engineering* 13, no. 2 (2023): 1346. https://doi.org/10.11591/ijece.v13i2.pp1346-1357

[11] Chiuchisan, Iuliana. "An approach to the Verilog-based system for medical image enhancement." In *2015 E-Health and Bioengineering Conference (EHB)*, pp. 1-4. IEEE, 2015. https://doi.org/10.1109/EHB.2015.7391461

[12] "CPU or FPGA for image processing: Which is best?, " Vision Systems Design. Accessed: Jul. 04, 2024. [Online].

[13] Raikovich, Tamás, and Béla Fehér. "Application of partial reconfiguration of FPGAs in image processing." In *6th Conference on Ph. D. Research in Microelectronics & Electronics*, pp. 1-4. IEEE, 2010.

[14] Keerthi, A., Nikitha, A., Nikhil, A., and Kumar, A. P. "Implementation of Image Enhancement using Verilog HDL". *ZKG International*, vol. 4, no. 1, pp. 1243– 1263, 2024.

[15] Khudhair, Zaid Nidhal, Ahmed Nidhal Khdiar, Nidhal K. El Abbadi, Farhan Mohamed, Tanzila Saba, Faten S. Alamri, and Amjad Rehman. "Color to grayscale image conversion based on singular value decomposition." *Ieee Access* 11 (2023): 54629-54638. https://doi.org/10.1109/ACCESS.2023.3279734

[16] "What are the Benefits of using Grayscale Scanning for Specific Applications, such as Medical Imaging?," Electronic Office Systems. Accessed: Jun. 28, 2024. [Online].

[17] "DE10-Standard: Layout, " Terasic Technologies. Accessed: Jan. 14, 2024. [Online].

[18] "Altera Quartus II," University of Nevada, Las Vegas (UNLV). Accessed: Jan. 09, 2024. [Online].

[19] Mentor Graphics Corporation, "ModelSim Tutorial, " 2015. Accessed: Jul. 04, 2024. [Online].

[29] "What is MATLAB?, " MathWorks. Accessed: Jan. 09, 2024. [Online].

[21] Yusefi, Mostafa, Ong Su Yee, and Kamyar Shameli. "Bio-mediated production and characterisation of magnetic nanoparticles using fruit peel extract." *Journal of Research in Nanoscience and Nanotechnology* 1, no. 1 (2021): 53-61. https://doi.org/10.37934/jrnn.1.1.5361