**PROJECT TASK**

The task of this project was to implement a program that, when executed on an FPGA board, would perform image scaling. Image data would be loaded into the board's memory, which would then perform the scaling operation. I was unable to complete the part of the project that runs on the FPGA (Spartan3E Starter Kit), so only image scaling was performed in simulation. The code to be run on the FPGA would be written in the Verilog programming language using the ISE Design Suite 14.1 development environment.

**READING DATA FROM THE IMAGE**

FPGAs and Verilog do not support formats like .bmp or .jpg; they use only .bin and .hex formats. In this project, we would use the .hex format. The initial image would be in .bmp format and needed to be converted to .hex format. This conversion would be done using the MATLAB programming language.

To perform this conversion, the MATLAB code first specifies the path and name of the image to be converted. MATLAB then carries out the conversion, and the result is an output file in .hex format. The values within this output file represent individual pixels of the image based on the following principle: the first three values represent the first pixel, the next three values represent the second pixel, and so on. Each pixel consists of red, green, and blue components (RGB), which is why three values are used to describe each pixel. Each of these values can range from 0 to 255.

After the input image is converted into .hex format, it needs to be read in Verilog using the readmemh or readmemb commands. The difference between these commands is that readmemh is used for reading hexadecimal values, while readmemb is used for reading binary values.

**PROCESSING RECEIVED DATA ACCORDING TO PROJECT REQUIREMENTS**

To process the received data, an output\_master module is used. The following are some parts of this module:

In the first part of the block, a loop is used to initialize the temp\_memory array with values from the memory array. This way, pixels from the memory array are stored in the temp\_memory array for further use.

In the second part of the block, a new for loop is used to copy pixel values from temp\_memory into three different arrays to separate the red, green, and blue components of the pixels.

In the second block, the values r, g, and b represent the red, green, and blue components of the pixel at the coordinates specified by row and column from the red, blue, and green arrays.

**CONVERTING NEW DATA INTO AN IMAGE AND DISPLAYING THE NEW IMAGE**

To display the processed image, a new module needs to be written that takes the output from the image processing module as input. The fwrite command is used for output. Before actually outputting the data, it is necessary to add a BMP header that informs the operating system about the file format.

In the infile parameter, the path to the output BMP file is specified.

The most crucial part, which must not be wrong, is the BMP headers. If the headers are incorrect, the image will not be loaded correctly, resulting in an incorrect output when the code is executed.

In summary, this project involves converting an input image into a suitable format for an FPGA, processing the image, and then converting the processed data back into an image format for display. Proper header values are essential for correct image loading.