International Journal of Signal Processing, Image Processing and Pattern Recognition
Vol.8, No.3 (2015), pp.235-250
http://dx.doi.org/10.14257/ijsip.2015.8.3.22

Design of Sliding Window Based Corner Detection Algorithm and Architecture for Image Mosaicing

Jayalaxmi H[1] and S. Ramachandran[2]

Research Scholar
[1]JNTU, Hyderabad, India
[2]SJBIT, Bengaluru, India
jayalaxmi@acharya.ac.in

Abstract

A new Sliding Window Based Corner Detection Algorithm has been developed for image mosaicing. Using this algorithm, a new architecture suitable for VLSI implementation has also been designed. The proposed design incorporates a high degree of pipelining and parallelism and hence offers high throughputs. The color image mosaicing is achieved by first convolving an original image using a 3x3 sliding window followed by smoothening the image using median filters. The design has been coded in Verilog as per RTL coding guidelines. The algorithm has also been coded in Matlab in order to validate the hardware results. The Verilog design of the proposed architecture for image mosaicing has been implemented on Xilinx Spartan 6 xc6slx45-3fgg676 FPGA device. The design utilizes about 144,858 gates and the operating frequency is about 100 MHz. The design is capable of processing high resolution color pictures of sizes of up to 1600× 1200 pixels in real time.

Keywords: *Corner detection, Sliding Window, Verilog, RTL Coding, Pipelining and Parallelism FPGA*

1. Introduction

Image mosaicing algorithms produce high resolution photo-mosaics used to construct today's digital maps and satellite photos. These algorithms get the alignment estimates produced by such registration algorithms and blend the images in a seamless way, taking care to deal with possible problems such as blurring or ghosting caused by alignment error and scene movements as well as unstable image exposures [1]. Many alignment problems are caused in cases where the detected features are not uniform. Feature based approaches have the benefit of being more robust to inter boundary of images being mosaiced and offer faster processing speed. Hitherto, the major area of concern for the corner detection algorithms was that unwanted boundaries were detected along with corners. In order to remove these boundaries, feature based corner detection algorithm was developed by the present authors [2].

Image and Video Processing has been a very active field of research and development for over 20 years and many different systems and algorithms for image Mosaicing, compression, filtering and smoothing have been proposed and developed. Only marginal improvement has been achieved since parallelism and pipelining incorporated in the existing work [3-4] are inadequate.

O. Ranganathan *et al.* have presented the design of a modern camera interface controller that has high processing speed [5]. The focus of the design is on developing the input

ISSN: 2005-4254 IJSIP
Copyright ⓒ 2015 SERSC

interface controller module for interfacing CMOS sensor with FPGA module for converting the raw data into RGB pixel information and the output interface controller. The captured information from the sensor is then converted into RGB pattern and processed for streaming in VGA display. However, the frame rate is limited to 10 fps.

Matthew Zubiel *et al.* have designed an image processing module interface to VmodCAM module [6] for real time. C# program was developed in order to grab the image in a format called RGB565 using the VmodCAM module. Thereafter, the corner detection algorithm was coded in VHDL. Abdul Manan has proposed an algorithm and implementation of morphological image processing on FPGA [7]. To enhance the performance, this work addresses implementation of image processing algorithms like median filter, morphological, convolution and smoothing operations and edge detection on FPGA using VHDL language. Since they [6-7] have used only behavioral type of coding and not RTL coding, the processing speed is low. The speed can be improved by incorporating high parallelism and pipelining in the corner detection as is the case with the present work.

Vimal Singh has designed a feature based image mosaicing using image fusion method [8]. To extract the features from the stitching results, the blending process is done by means of Discrete Wavelet Transform. They have used only planar images as test input images instead of cylindrical and spherical images for mosaicing.

Algorithms are essential for processes related to geometric transformations and distortions for mosaicing applications. In order to resize, rotate, image shape rectification and distortion removal, there are numerous approaches to solve this problem that differ in image quality, processing speed and complexity. Median filtering is very widely used in digital image processing [9, 20]. In our previous work [2, 10], Gaussian based design of an algorithm for feature based corner detection image mosaicing has been presented in order to overcome the limitations of Harris corner and Edge-based corner methods. The Gaussian function used in that work was effective at removing noise in smooth patches or smooth regions of a signal, but unfavorably affect edges or corners. Edges or corners are essential to the visual appearance of images [11-18]. The median filter is obviously better than Gaussian blur at removing noise, preserving edges or corners for any size of the window. In the present work, this limitation has been overcome by convolving *t*he input image using 3×3 sliding window followed by smoothing the image by using median filters. The proposed method uses a new Sliding Window Based Corner Detection Algorithm based on Median arithmetic instead of the time consuming data sorting method.

The rest of the paper is organized as follows. The proposed image mosaicing algorithm is presented in Section 2. Section 3 presents details of architectural design of the proposed algorithm. The penultimate section provides the Simulation and Place and Route results. Conclusion is presented in Section 5.

2. Proposed Image Mosaicing Algorithm

The color image mosaicing algorithm proposed in this paper uses 3×3 Sliding window, where in block of nine pixels of original image are convolved with median filter. The input pixel window may still be chosen as convolution mask for various window sizes such as 3x3, 5x5, and 7x7, without affecting the throughput. In this work, the convolution mask is chosen as 3×3 in order to improve the implementation speed, at the same time minimizing blocking artifacts. Also 3×3 window is small enough to fit into the target FPGA and can be easily compute corner values in the sliding window.

 Copyright ⓒ 2015 SERSC

2.1 Design of ASM Chart

Algorithmic State Machine (ASM) charts developed along with their designed architectures can be converted directly to a hardware using Hardware Design Language (HDL) such as Verilog. The developed ASM chart for Sliding Window Based Corner Detection Algorithm for image mosaicing is shown in Figure 1a and Figure 1b. State "0" initializes all the registers and outputs used in the design. State "1" reads the input gray level image. State "2" is used to select 3x3 sliding window to process the image data. The state "3" is used to convolve 3x3 window. State "4" to state "6" are used to compute gradients. State "7" to state "9" are used to compute image derivatives from convolution window. State "10" computes the corner value and is used to extract features from images in state "11". The above process is repeated for all the blocks of the images. State "12" combines separated components of R, G and B into composite RGB and displays the mosaiced image in the last state.

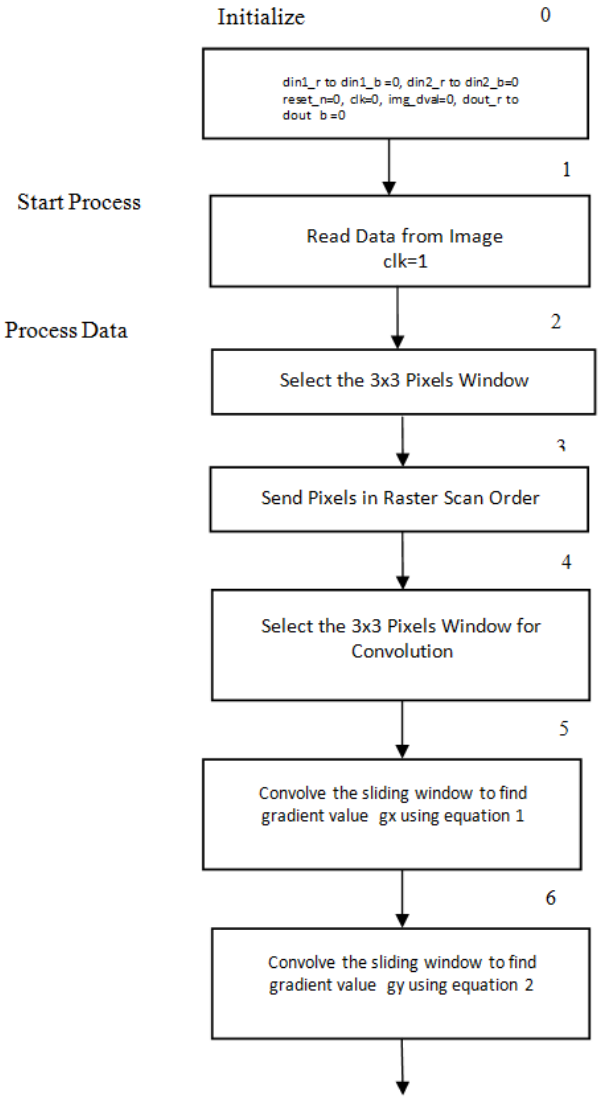


Figure 1a. ASM Chart for Sliding Window Based Algorithm (Continued)

Copyright ⓒ 2015 SERSC

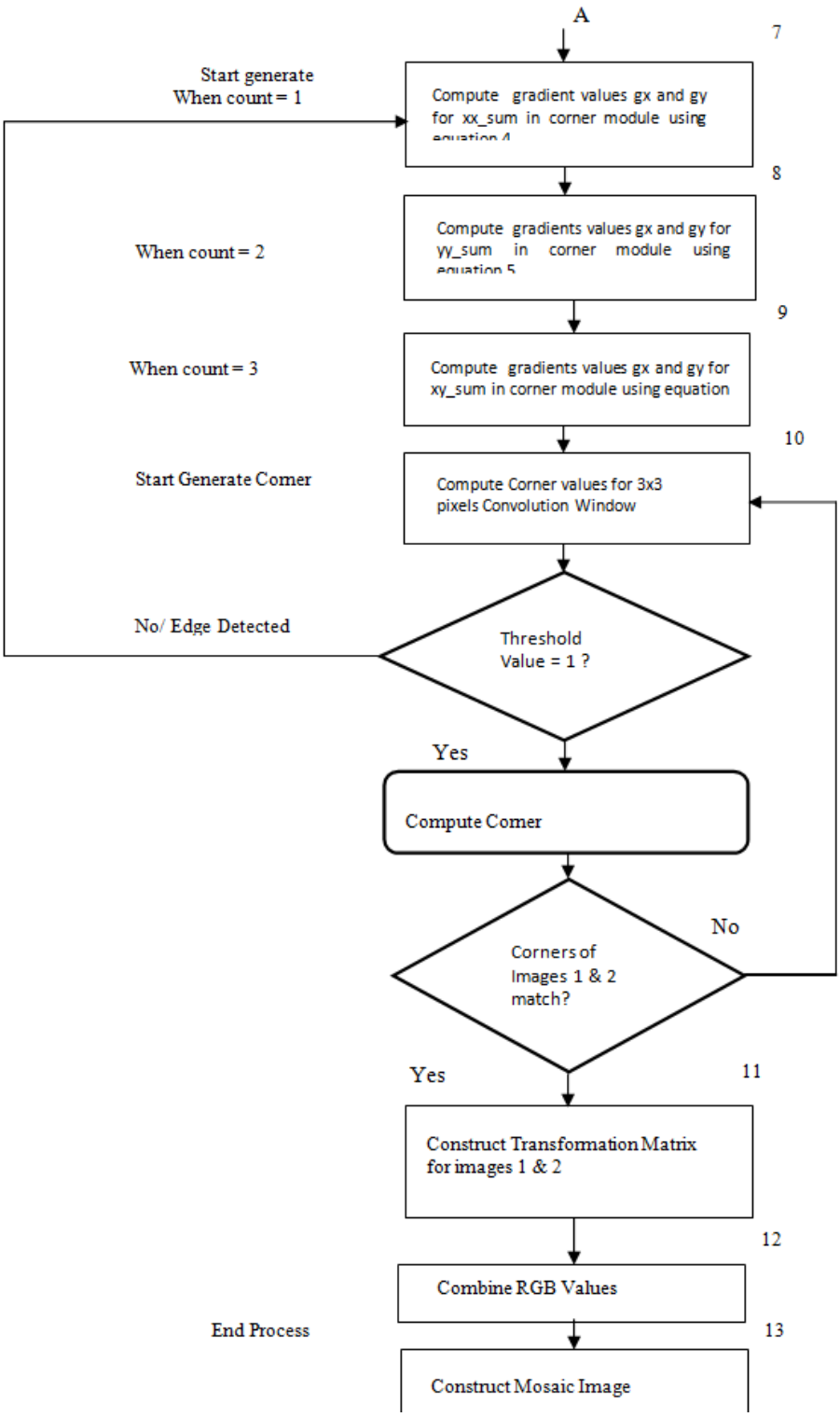


Figure 1b. ASM Chart for Sliding Window Based Algorithm

Copyright ⓒ 2015 SERSC

3. Architecture of the Proposed Sliding Window Based Color Image Mosaicing System

This section presents the architecture of the Sliding Window Based Corner Detection Algorithm presented in the previous section. It is composed of several components such as Color and Gray Image module, 3x3 Sliding Window, 3x3 Convolution module, Corner module, Image Transformation module and Image Blending module.

3.1 Architecture for Sliding Window Based Color Mosaicing System

The Signal Diagram of the Top Level Color Image Mosaicing System is shown in Figure 2 and the signals used in this module are presented in Table 1. The architecture for Sliding Window Based Corner Detection uses

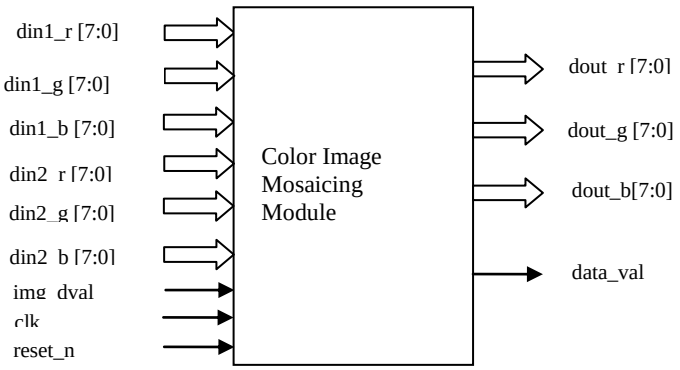


Figure 2. Block Diagram of Top Level Color Image Mosaicing Module

Table 1. Signal Description for Color Image Mosaicing Module

Signals	Input/Output	Description
clk	Input	Clock Signal
reset_n	Input	Active low system reset
img_dval	Input	Input image Data Signal
din1_r [7:0]	Input	First Image data for ‘r’ color
din1_g [7:0]	Input	First Image data for ‘g’ color
din1_b [7:0]	Input	First Image data for ‘b’ color
din2_r [7:0]	Input	Second Image data for ‘r’ color
din2_g [7:0]	Input	Second Image data for ‘g’ color
din2_b [7:0]	Input	Second Image data for ‘b’ color
dout_r [7:0]	Output	Mosaiced Image for ‘r’ color
dout_g [7:0]	Output	Mosaiced Image for ‘g’ color
dout_b [7:0]	Output	Mosaiced Image for ‘b’ color
data_val	Output	Data Valid Signal

Copyright ⓒ 2015 SERSC

Pipelining stages and parallel processing in the design in order to increase the processing speed. This architecture mosaics two images whose inputs "din1" and "din2" are fed pixel by pixel. The input pixels are valid at the positive edge of the clock. When the pixel is valid, the "img_dval" signal is asserted. Each of the color components from the two images being mosaiced is integrated and output as "dout_r", "dout_g"and "dout_b" of size 8 bits. Basic Architecture for the Sliding Window Image Mosaicing System is shown Figure 3. This architecture mosaics two images whose inputs "din1" and "din2" are fed pixel by pixel. The input color images "image1" and "image2" from a camera arrive as separated color components Red (R), Green (G) and Blue (B).

The "din1_r", "din1_g", "din1_b" and "din2_r", "din2_g", "din2_b" of size 8 bits are the individual color components are fed to the 3x3 pixels sliding window in a raster scan manner. The input pixels are valid at the positive edge of the clock. When the pixel is valid, the "data_val" signal is asserted. The output values from 3x3 pixels Sliding Window are sent to the 3x3 Convolution Module for convolving the gradients of each pixel using a pipelined architecture with FIFO module [19, 20]. The convolved derivatives are input to the next module in order to compute the corner values for each of the two images being mosaiced. The choice of corner for feature detection is stable when corner is greater than 1.

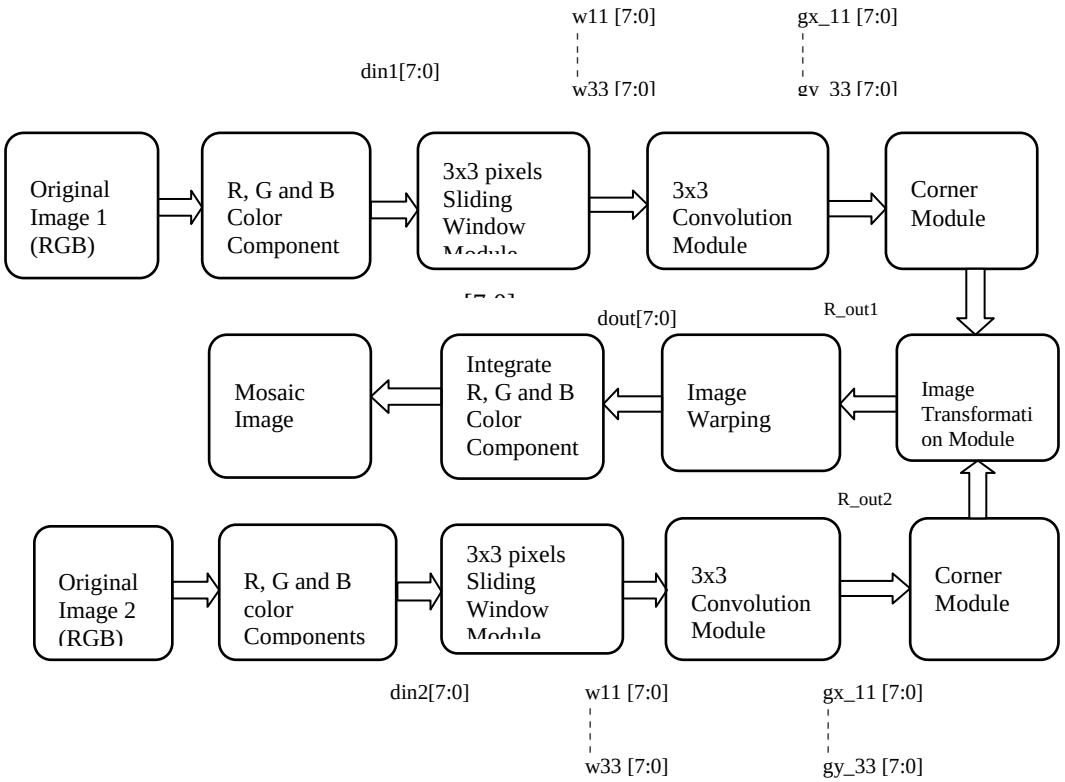


Figure 3. Basic Architecture for the Sliding Window Based Image Mosaicing

Once the two sets of corner values in the images have been detected, the aim is to match the corresponding features to align the images. These corner values of each block are aligned pixel by pixel using Image Warping in order to correct image distortion, if any. The final step is to blend the pixel in the overlapped region to avoid the seams or artifacts. Image blending is the technique, which modifies the image in the overlapped area to obtain a smooth

 Copyright ⓒ 2015 SERSC

transition between images by removing these seams. Finally the separated R, G and B components are combined into composite RGB in order to obtain the mosaic image.

3.2 Sliding Window Architecture

The proposed method uses 3×3 Sliding Window architecture as shown in Figure 4 and the signals used in this module are presented in Table 2. Inside a 3×3 Sliding Window, three numbers of First-In-First-Out (FIFO) buffers are used to reduce the memory access to one pixel per clock cycle. In order to access all values of the window for every clock cycle, these FIFO must be full [5]. The FIFOs are cascaded as shown in Figure 5. The contents of the window are shifted right when the signal "start_win" is asserted and also to read the input data "d" of width 8-bits. Thereafter, during positive edge of clock "clk", the pixel values of the input "din[7:0]" are sent pixel by pixel in a raster scan order into the sliding window module. Figure 5 shows the schematic diagram of 3×3 sliding window used for row buffers for 8-bit pixels. The output from sliding window module "w11"to "w33" of size 8 bits sent to the next module for computing gradients when data valid for sliding window "dvwin" is asserted.

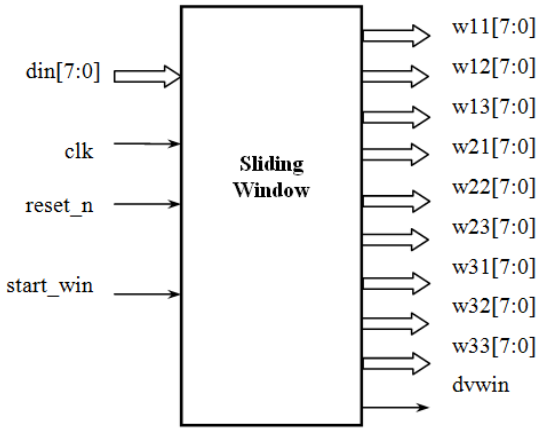


Figure 4 Architecture of 3x3 Sliding Window Module

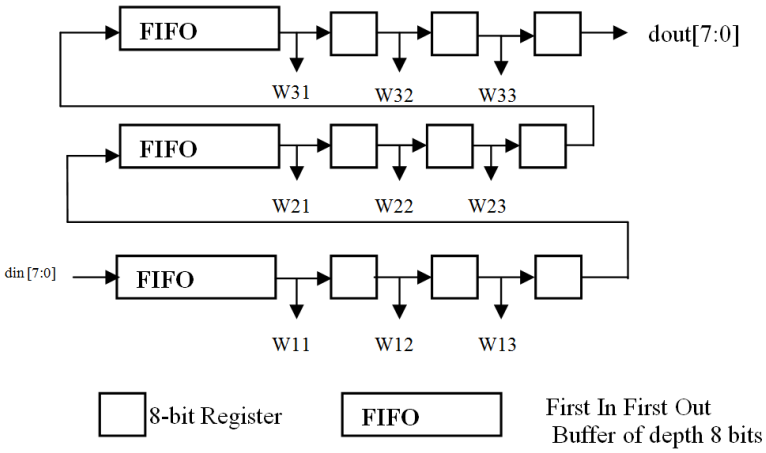


Figure 5. Schematic Diagram of 3x3 Sliding Window

Copyright ⓒ 2015 SERSC

Table 2. Signal Description for the Sliding Window Module

Signals	Input/Output	Description
clk	Input	This is the global clock signal
reset_n	Input	Active low system reset
din[7:0]	Input	Input pixel component
start_win	Input	Valid signal for input data
w11 [7:0] to w13 [7:0]	Output	First row pixel values for the 3×3 sliding window
w21 [7:0] to w23 [7:0]	Output	Second row pixel values for the 3×3 sliding window
w31 [7:0] to w33 [7:0]	Output	Third row pixel values for the 3×3 sliding window
dvwin	Output	Data valid for sliding window

3.3 Convolution Module

The next module after Sliding window module is the Convolution Module, which is used for convolving the gradients of each pixel.

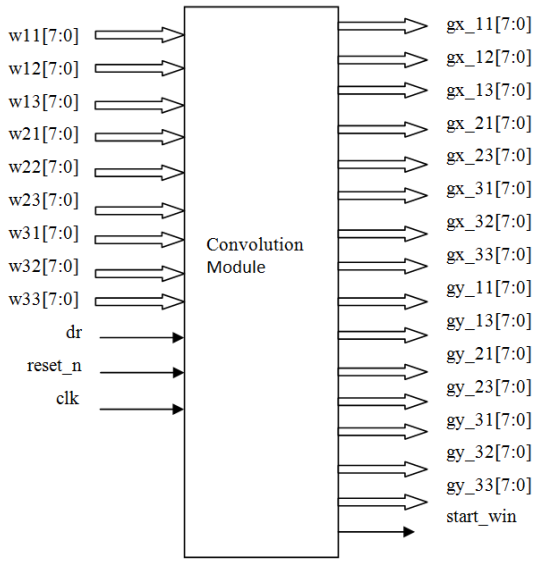


Figure 6. Architecture of Convolution Module

Table 3. Signal Description for the Convolution Module

Signals	Input/Output	Description
clk	Input	System clock
w11[7:0]	Input	Pixel p0 value
w12[7:0]	Input	Pixel p1 value
w13[7:0]	Input	Pixel p2 value
w21[7:0]	Input	Pixel p3 value
w22[7:0]	Input	Pixel p4 value
w23[7:0]	Input	Pixel p5 value

Copyright ⓒ 2015 SERSC

w31[7:0]	Input	Pixel p6 value
w32[7:0]	Input	Pixel p7 value
w33[7:0]	Input	Pixel p8 value
dr	Input	Data ready Input signal
reset_n	Input	Active low system reset
gx_11[7:0]to gx_13[7:0]	Output	Output gradient gx for 1st row
gx_21[7:0]& gx_23[7:0]	Output	Output gradient gx for 2nd row
gx_31[7:0]to gx_33[7:0]	Output	Output gradient gx for 3rd row
gy_11[7:0]to gy_13[7:0]	Output	Output gradient gy for 1st row
gy_21[7:0]& gy_23[7:0]	Output	Output gradient gy for 2nd row
gy_31[7:0]to gy_33[7:0]	Output	Output gradient gy for 3rd row
start_win	Output	Output start window signal

The architecture for Convolution Module is shown in Figure 6. Signal descriptions are presented in Table 3. The pixel values were sent to convolution module in order to compute image derivatives [2]. The outputs from 3x3 Sliding Window "w11" to "w33", each of 8 bit size, are used for computing gradients for each window when read signal "dr" goes high. For every clock cycle, this module convolves input gradients from sliding window to calculate image gradient values "gx_11" to "gx_13", "gx_21" to "gx_23" and "gx_31" to "gx_33" along row and "gy_11" to "gy_13", "gy_21" to "gy_23" and "gy_31" to "gy_33" along column. Based on these two derivatives, the rest of the derivatives necessary for the algorithms (Ix2, Iy2, and IxIy) were computed by simple multiplication. It works by convolving a 3x3 window of pixels with two horizontal and vertical filters in order to find the corner value of the center pixel in that 3x3 window. The calculation to obtain the gradients is shown in equation 1 and equation 2.

$$gx = \big((p2 - p0) + (p5 - p3) + (p8 - p6)\big) \quad (1)$$

$$gy = \big((p0 - p6) + (p1 - p7) + (p2 - p8)\big) \quad (2)$$

3.4 Corner Module

The architecture of the Corner Module is shown in Figure 7 and signal descriptions are presented in Table 4. This module computes corner values for the images in a pipelined manner using 3 stages [2]. The convolved gradients were computed from previous module are the inputs to the corner module. The corner module takes 9 pixel values in order to calculate 18 convolved derivatives per 3x3 pixels window, 9 in the x-direction ("gx_11" to "gx_33") and 9 in the y-direction ("gy_11" to "gy_33"). After calculating gx and gy, the module computes the magnitude of these gradients. The output is truncated to 8 bits. In order to truncate it to 8 bits, the bits 10, 9 and 8 are ORed together. If the result is 1, then it outputs 8'hFF, otherwise it outputs the actual value. After calculating gx and gy from equation 1 and equation 2, the module takes the magnitude of these gradients. The Harris corner detector uses both x and y gradients in order to find the value of pixel are a corner or edge. To be corner, there has to be significant changes in both directions. However, the corner algorithm calculates the likelihood that a pixel is a corner from summing all the surrounding gradients around a pixel and performing some calculations to determine its likelihood of being a corner. In order to extract the corners, the corner detector needs a 3x3 window to compute a score for the pixel at location (2, 2). The values of each of these derivatives were computed for every point in the image as shown in equation 3.

Copyright ⓒ 2015 SERSC

$$R\_out = \sum_{i=1}^{m} Ix^2 \times \sum_{j=1}^{n} Iy^2 - (\sum_{i=1}^{m} Ix \sum_{j=1}^{n} Iy)^2 - k(\sum_{i=1}^{m} Ix^2 \times \sum_{j=1}^{n} Iy^2)^2 \quad (3)$$

The sum of gradient values "xx_sum", "yy_sum" and "xy_sum", each of size 12 bits, are computed using convolved gradients as shown in equation 4 to equation 6. These values are fed synchronously to the corner module to calculate the corner value "R_out" every clock cycle. The gradients are used to measure the change in the intensity of the pixel in the window. This corner value is compared with a threshold to determine the good corner response. This corner module computes the determinant and trace of the corner matrix and outputs the determinant – k * trace^2. k is a constant value of 1/8 needed for this calculation. Also, because this is the most computationally intensive calculation owing to the width of each bus, the module carries out the calculation in a pipelined manner using 3 stages. The calculation of R_out is shown in equation 3.

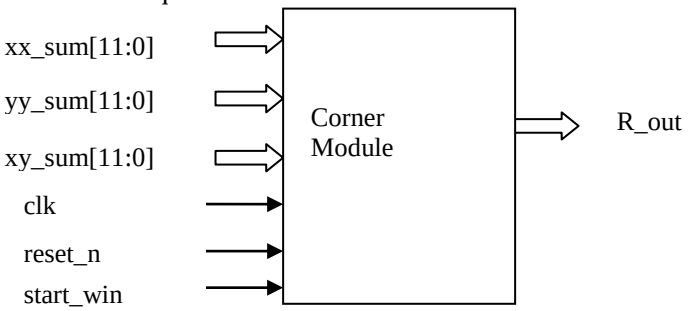


Figure 7. Architecture of Corner Module

xx_sum=gx_11*gx_11+gx_12*gx_12+gx_13*gx_13+gx_21*gx_21+gx_23*gx_23+ gx_31*gx_31+gx_32*gx_32+gx_33*gx_33 (4)
yy_sum=gy_11*gy_11+gy_12*gy_12+gy_13*gy_13+gy_21*gy_21+gy_23*gy_23+ gy_31*gy_31+gy_32*gy_32+gy_33*gy_33 (5)
xy_sum=gx_11*gy_11+gx_12*gy_12+gx_13*gy_13+gx_21*gy_21+gx_23*gy_23+ gx_31*gy_31+gx_32*gy_32+gx_33*gy_33 (6)

Table 4. Signal Description for the Corner Module

Signals	Input/Output	Description
clk	Input	System clock
xx_sum [11:0]	Input	Sum of gradient value xx
yy_sum [11:0]	Input	Sum of gradient value yy
xy_sum [11:0]	Input	Sum of gradient value xy
reset_n	Input	Active low system reset
R_out	Output	Corner Response

3.5 Image Transformation Module

The next module after Corner module is the Image Transformation Module, which is used for mosaicing the images. The architecture for Convolution Module is shown in Figure 8. Signal descriptions are presented in Table 5. The corner value for each window was computed in the previous section. The number of corner value in the 3x3 window is calculated for all the windows in an image.The extracted corners from each window are fed to the Image transformation module. In this module, the "din_1" and "din_2" are the input image data values for image 1 and image 2 respectively. Two sets of corner values in the images have been detected using corner module.

 Copyright ⓒ 2015 SERSC

For every clock cycle, these corner pixel values of each block are aligned pixel by pixel using Image Warping method [2]. There may be mismatch occurring between the pixels during image mosiacing. In order to overcome this, image blending is used to obtain a smooth transition between images by removing the intensity seam in the neighbourhood of the boundary.

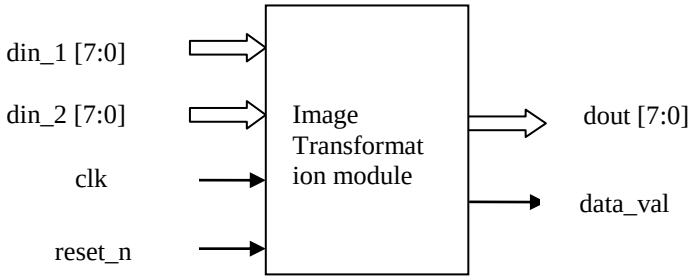


Figure 8. Architecture of Image Transformation Module

The final step is to blend the pixel in the overlapped region to avoid seams. Image blending is the technique, which modifies the image gray levels in the overlapped area to obtain a smooth transition between images by removing these seams. Finally the separated R, G and B components are combined into composite RGB in order to obtain the mosaic image.

Table 5. Signal Description for the Image Transformation Module

Signals	Input/Output	Description
clk	Input	System clock
din_1 [7:0]	Input	Output data from Image1
din_2 [7:0]	Input	Output data from Image2
reset_n	Input	Active low system reset
dout [7:0]	Output	Output Image data
data_val	Output	Data valid signal

3.6 Architecture of Multiplier

The multiplier design presented here incorporates a high degree of parallel circuits and pipelining of eight levels. The Verilog design of this multiplier has been presented in the book [9] by one of the present authors. The architectures for adders and subtractors for various bit sizes which are used in image mosaicing module has been modeled based on Verilog code presented in one of the authors' book [9]. The detailed architectures and simulation results for these modules are discussed in the paper [10]. It uses lots of parallelism and pipelining in its design as can be easily inferred.

4. Results and Discussions

The proposed Sliding Window Based Corner Detection Algorithm, whose architecture was presented in the previous section, has been coded and tested in Matlab first in order to ensure the accurate functioning of the algorithm. Subsequently, the complete system has been coded in RTL compliant Verilog so that it may be implemented on a Xilinx FPGA. The simulation

Copyright ⓒ 2015 SERSC

of the proposed method has been done using ModelSim and synthesized using Xilinx ISE 13.2. The algorithm has been targeted on Xilinx Spartan6 xc6slx45-3fgg676 FPGA device.

Matlab program was written in order to convert the images to be mosaiced into text format since Modelsim simulation tool accepts only text inputs. The Verilog design for 3x3 pixels sliding window whose architecture was presented in above section was run in Modelsim to get the reconstructed picture in "txt" format. These reconstructed "txt" files were converted back to image format using another Matlab program. This program displays both the original picture as well as the mosaiced picture. The Matlab program also computes the quality of the mosaiced image referred to as PSNR expressed in dB.

The "img_dval" process commences at 36448 ps as presented in Figure 9. As presented in Figure 10, the image data are fed pixel by pixel to the sliding window module starts at 39664 ps.The mosaicing process starts at 39664 ps and ends at 819204712 ps as shown in Figure 11. The sliding window based process, thus, takes 2.221 clock cycles per pixel data.

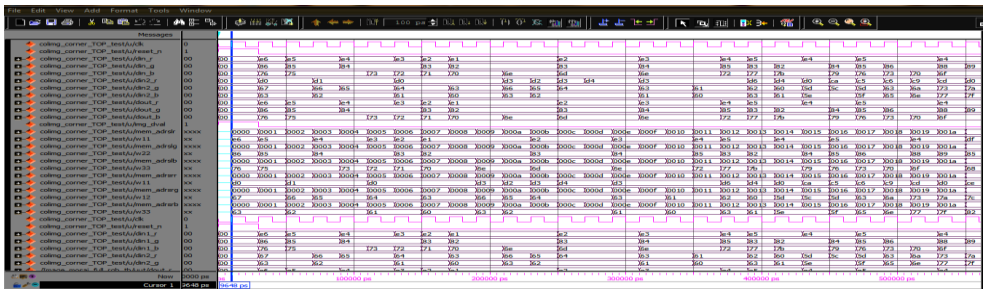


Figure 11. Waveforms for Sliding Based Color Image Mosaicing System: Validity of Input Data

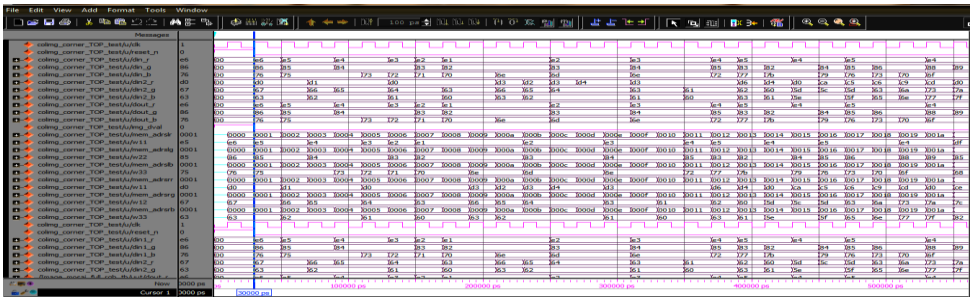


Figure 12. Waveforms for Sliding Based Color Image Mosaicing System: Start of Image Data

In summary, the mosaiced image pixels start issuing at 39664 ps and ends at 819204712 ps, thus taking 819165048 ps for processing a complete image data. Since each "clk" cycle is of duration 2 ps during simulation, it takes 409582524 "clk" cycles to mosaic both image data. Therefore, for a picture of size 256x256 pixels such as Lena is splitted into two images as Lena1(256x160) and Lena2 (256x160) are used in the present simulation, it takes 2.2 clock cycles to process each pixel. Assuming an operating frequency of 100 MHz for "clk", this works out to 10.1 milli second per image ignoring latency, which is small. Extrapolating this processing time for a picture of resolution 512x512 pixels, we get the processing time of 20.01 milli second per image or in other words, we have achieved a mosaiced image rate of 20 pictures per second.The simulation waveforms for the start and the end of 3x3 pixels sliding window module are presented in Figure 14 and Figure 15 respectively. The input

 Copyright ⓒ 2015 SERSC

pixels are valid at the positive edge of the clock. When the pixel is valid, the "img_dval" signal is asserted. The Sliding Window module produces nine pixels (w11 [7:0] to w13 [7:0] through w31 [7:0] to w33 [7:0]) in parallel once the FIFOs are full.

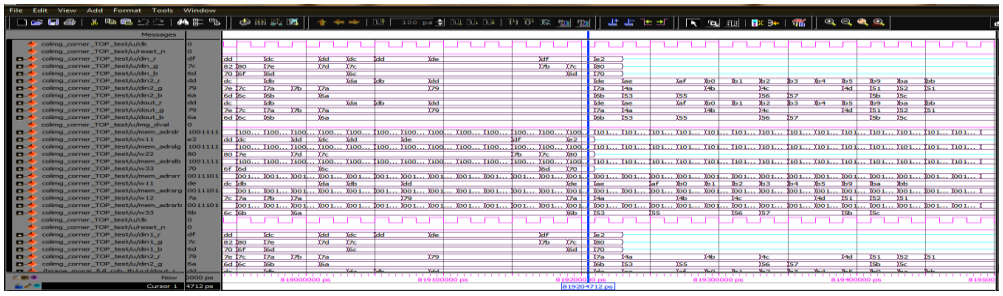


Figure 13. Waveforms for Sliding Based Color Image Mosaicing System: End of Image Data

These pixels from the sliding window module are fed as inputs to the Convolution module in order to carry out the convolution operation. The Simulation Waveforms for 3x3 Convolution Module for start and end process time are shown. The outputs from 3x3 Sliding Window "w11" to "w33", each of 8 bit size, are used for computing gradients for each window. The validity of these pixels is indicated by asserting "dr" signal. For every clock cycle, this module convolves input gradients from sliding window to calculate image gradient values "gx_11" to "gx_13", "gx_21"to "gx_23" and "gx_31" to "gx_33" along row and "gy_11" to "gy_13", "gy_21" to "gy_23" and "gy_31" to "gy_33" along column. The convolved derivatives are input to the next module in order to compute the corner values for each of the two images being mosaiced. The Simulation Waveforms for Corner Module is shown in Figure 15. These values are fed synchronously to the corner module to calculate corner value "R_out" every clock signal.

a b c d

Figure 14. Simulation Results for Sliding Window Based Color Image Mosaicing

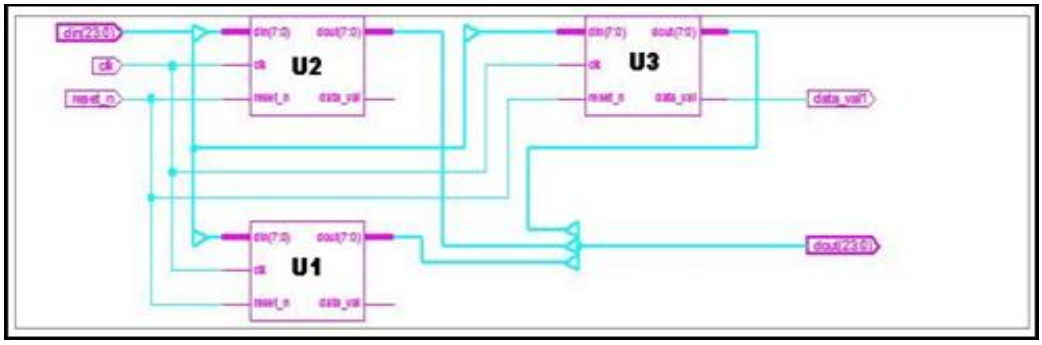
Original Lena (left half) image (225×225 pixels) (b) Original Lena (right half) image (225×225 pixels) (c) Mosaiced Lena Image using Matlab, PSNR: 39.3dB (d) Mosaiced Image using Verilog, PSNR: 39.1 dB.

Copyright ⓒ 2015 SERSC

The RTL Verilog simulation results are compared with Matlab results described earlier in order to validate the hardware design. Elaborate experiments were conducted on various images and consistently good results have been obtained. Lena image has been mosaiced as shown in Figure 14, wherein (a) and (b) are the two images that needs to be mosaiced. The final mosaiced Lena image (c) is generated by Matlab and the mosaiced image (d) by simulation of Verilog RTL code.

4.1 Place and Route Results

The various modules described in previous sections were coded in Verilog, simulated using ModelSim, synthesized and place and routed using Xilinx ISE 13.2. The simulation results of the Image mosaicing system was presented in the previous section. The target device chosen was Xilinx Spartan6 xc6slx45-3fgg676 FPGA. The core modules of the image mosaicing design described in earlier sections utilizes 144,858 gates and 12 numbers of block RAMs with 25158 numbers of occupied slices. The maximum frequency of operation is 100 MHz for "clk". This works out to a frame rate of 20 per second for a picture size of 512x512 pixels as explained earlier. With higher speed FPGA, the frame rate can be increased to 30. The Verilog codes developed for this project is fully RTL compliant and technology independent. As ASIC, it is likely to work for higher resolutions up to 1600x1200 pixels at 30 frames/sec. The ISE generated RTL view of the Single Window Color Image Mosaicing architecture is shown in Figure 15. The device utilization summary for Sliding window based image mosaicing system are presented in Table 6.



Note: U1: Red Color, U2: Green Color, U3: Blue Color Component Processors

Figure 15 . RTL View of Single Window Color Image Mosaicing System

Table 6. FPGA Resource Consumption of the RTL Verilog Design

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
No. of Slices	25158	54576	46%
No. of Slice Flip Flops	24867	49486	55%
No. of 4 input LUTs	13134	62620	20%
No. of Bounded IOB	78	358	21%
No. of Block RAM	20	104	20%

The generated timing report with maximum delay is given below:

The synthesis as well as place and route results for the various modules of the top image mosaicing module are presented below.

 Copyright ⓒ 2015 SERSC

Timing Summary: Speed Grade: -3
Minimum period: 10.061ns (Maximum Frequency: 99.397MHz)
Minimum input arrival time before clock: 9.584ns
Maximum output required time after clock: 6.854ns
Maximum combinational path delay: 6.746ns
Timing constraint: Default period analysis for Clock 'clk'

Device utilization summary:
Selected Device : 6slx45fgg676-3
Slice Logic Utilization:
Number of Slice Registers: 25158 out of 54576 46%
Number of Slice LUTs: 24867 out of 49486 55%
Number of LUT Flip Flop pairs used: 62620
Number with an unused Flip Flop: 37462 out of 62620 59%
Number with an unused LUT: 13134 out of 62620 20%
Number of fully used LUT-FF pairs: 12024 out of 62620 19%
Number of unique control sets: 19
IO Utilization:
Number of IOs: 78
Number of bonded IOBs: 78 out of 358 21%
Specific Feature Utilization:
Number of BUFG/BUFGCTRLs: 4 out of 16 25%

5. Conclusion

A new Sliding Window Based Corner Detection Algorithm has been developed for image mosaicing. Using this algorithm, a new architecture suitable for VLSI implementation has also been designed. The proposed design incorporates a high degree of pipelining and parallelism and hence offers high throughputs. The Verilog design of the proposed architecture for image mosaicing has been targeted on Xilinx Spartan 6 xc6slx45-3fgg676 FPGA device. The design utilizes about 25158 slices and the operating frequency is about 100 MHz. The design is capable of processing high resolution color pictures of sizes of up to 1600×1200 pixels in real time.

References

[1] B. Zitova and J. Flusser, "Image registration methods: a survey", Image andvision Computing, vol. 21, (**2003**), pp. 977-1000.
[2] H. Jayalaxmi and S. Ramachandran, "Novel Algorithm for Image Mosaicing using Feature based Corner Detection", DOI: 03.AETS.2014.5.366© Association of Computer Electronics and Electrical Engineers, 2014.
[3] N. Shirazi, M. Athansa and A. L. Abbott, "Implementation of a 2-D Fast Fourier Trasform on FPGA based Custom Computing Machine", Proceedings of 12th Reconfigurable Architecture Workshop, Denver (**2005**).
[4] D. G. Bailey, "Design for Embedded Image Processing on FPGAs", John Wiley and Sons Inc., (**2011**).
[5] O. Ranganathan and A. I. Rasheed, "Design And Development Of Camera Interface Controller With Video Pre- Processing Modules On FPGA for Mavs", SASTECH Journal vol. 12, no. 1, (**2013)** April**.**
[6] M. Zubiel, "Firefighter Indoor Navigation using Distributed SLAM", Worcester Polytechnic Institute, 2012.
[7] A. Manan, "Implementation of Image Processing Algorithm on FPGA", AKGEC JOURNAL OF TECHNOLOGY, vol. 2, no. 1, (**2006**).
[8] V. S. Bind " Robust Techniques for Feature-based Image Mosaicing", National Institute of Technology, Rourkela, (**2013**).
[9] G. R. Arce, "Nonlinear Signal Processing: A Statistical Approach", Wiley:New Jersey, USA, (**2007**).

Copyright ⓒ 2015 SERSC

[10] H. Jayalaxmi and S. Ramachandran, "Design of Novel Algorithm and Architecture for Feature Based Corner Detection for Image Mosaicing", Journal of VLSI and Signal Processing, DOI: 10.9790/4200-04631224, Volume 4, Issue 6, Ver. III (Nov - Dec. 2014), (**2014**), pp. 12-24
[11] S. Ramachandran, "Digital VLSI Systems design: A Design Manual for Implementation of Projects on FPGAs and ASICs using Verilog", Springer Verlag, (**2008**).
[12] M. Heikkila and M. Pietikainen, "An Image Mosaicing module for Wide–Area Surveillance", VSSN'05, November 11, 2005, Singapore, ACM 1-59593-242-9/05/0011.
[13] B. Zitova and J. Flusser, "Image registration methods: a survey", Image and vision Computing, vol. 21, (**2003**), pp. 977-1000, doi:10.1016/S0262-8856(03)00137-9.
[14] P. Azzari, L. D. Stefano and M. Stefano, "An Evaluation Methodology For Image Mosaicing Algorithms", Advanced Concept for Intelligent Vision Systems, Springer, vol. 52-59, (**2008**), pp. 89-100.
[15] D. K. Jain, G. Saxena and V. K. Singh, "Image Mosaicing Using Corner Technique", International Conference on Communication System and Network Technologies, (**2012**), pp. 79-84.
[16] Y. Li, "FPGA Implementation for Image Processing Algorithms", EEL 6562 Course Project Report, (**2006**) December.
[17] H. Joshi and Khomlalsinha, " Novel techniques Image Mosaicing based on Image Fusion Using Harris and Surf", International conference on computer science and information Technology, (**2013**) March, Hyderabad, ISBN:978- 93-82208-70-9.
[18] J. Lia nd J. Du, "Study on Panoramic Image Stitching Algorithm", Second Pacific Asia Conference on Circuits, Communications and System, vol. 1, (**2010**), pp. 417-420.
[19] M. C. H. Raju, "Design of Novel Algorithm and Architecture for Gaussian Based Color Image Enhancement System for Real Time Applications", International Journal of Computer Vision and Pattern Recognition, 10.1007/978-3-642-36321-456, (**2014**).
[20] A. Bovik, "Handbook of Image and Video Processing", 2nd edition, Elsevier Academic Press, (**2005**).

 Copyright ⓒ 2015 SERSC