International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2014): 5.611

FPGA Implementation of Connected Component Watershed Image Segmentation

Megha Sharma[1], Seema Verma[2]

[1]Research Scholar, Electronics Department, Banasthali University, Rajasthan

[2]Associate Professor, Electronics Department, Banasthali University, Rajasthan

Abstract: ***Image Segmentation is an important process for image processing analysis and for computer vision. A segmentation problem includes the partitioning of an image into a number of homogeneous segments and union of any two segments forms a heterogeneous segment. There are various methods which deal with segmentation and feature extraction like Markova random field based technique, edge based technique, histogram based technique etc. However, the segmentation is a challenging task because of the variety and complexity of images. To reduce the complexity of the hardware architecture, watershed transform using connected component has been used for image segmentation.***

Keywords: image segmentation, watershed transform, connected component, FPGA

1. Image Segmentation

Image segmentation means division of an image into meaningful structures. It is process of extracting and representing information from the image to group pixels together with region of similarity [3]. Sonka et al. define the goal of segmentation as “to divide an image into parts that have a strong correlation with objects or areas of the real world contained in the image" [2].

All the objects of the original image can be identified in segmented image with their boundaries. There are many techniques available for the image segmentation. Examples are, threshold based segmentation, edge based segmentation, region based segmentation, clustering based image segmentation, markov random field based segmentation and hybrid techniques. These segmentation methods differ from their computation complexity and segmentation quality. A segmentation algorithm has been proposed which is feasible to implement in hardware with the minima use of hardware resources (slices/gates), gives best segmentation quality and has possibility to be used for real time image processing applications.

Computation complexity is one of the important criteria for image segmentation which should be considered carefully when real time image segmentation is required. Computational complexity is defined as number of arithmetical operations required for processing single image frame. If the segmentation algorithm is computationally more complex then it needs more computational hardware resources. The best approach should be less computational complexity, less input parameter dependency, minimum segmentation time and provide efficient segmentation output for real time applications.

2. Watershed Image Segmentation

Watershed based image segmentation algorithms are less computational complex and provide very good segmentation results. It is possible to implement in the hardware using pipelined and/or parallel architecture for real time applications because of the independent mathematical computations flow of the algorithms.

Watershed algorithms based on watershed transformation have mainly two classes. The first class contains the flooding based watershed algorithms and it is a traditional approach where as the second class contains rainfalling based watershed algorithms. Many algorithms have been proposed in both classes but connected components based watershed algorithm [1] shows very good performance compared to all others. It comes under the rainfalling based watershed algorithm approach. It gives very good segmentation results, and meets the criteria of less computational complexity for hardware implementation.

There are mainly three stages as indicated by Figure 1 for watershed based image segmentation approach. First stage is defined as pre-processing, second stage as watershed based image segmentation and last stage as post-processing. Input image is first processed by the pre-processing stage, and then given to watershed based segmentation stage. The resulting image is post processed by the final stage to get a segmented image. Pre-processing and post-processing are necessary to overcome the problem of over-segmentation in watershed based image segmentation.

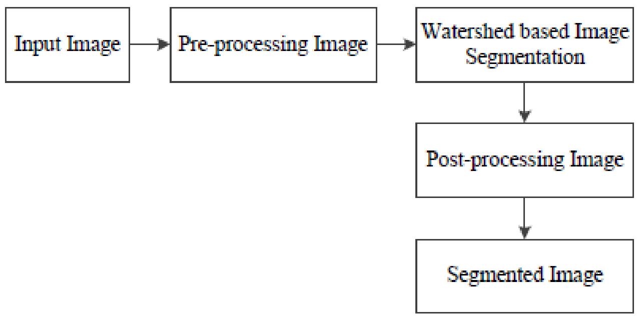


Figure 1: Block diagram of watershed based image segmentation.

The basic concept of watershed algorithm used for the image segmentation is to find the watershed lines. Imagine, holes at each regional minimum, and water is flooded from bottom into these holes with constant rate. Water level will rise in the topographic surface uniformly. When the rising water in different catchment basins is going to merge with nearby catchment basins then a dam is built to prevent all merging of the water. Flooding of water will reach at the point when only top of the dams are visible above water line. These continuous dam boundaries are the watershed lines as shown in figure 2.

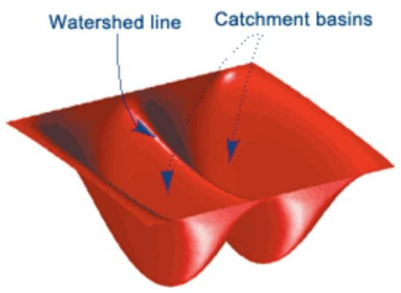
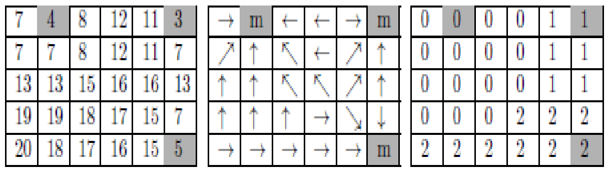


Figure 2: Watershed lines and catchment basin [4]

3. Watershed Algorithm Based on Connected Components

The basic concept of connected components based algorithm is explained by Figure 3. The original 6 x 6 image has three local minimum values indicated by gray boxes. If the component (pixel) is not a local minimum then it is connected to its lowest neighbours as shown by arrows in Figure 3b, where m indicates a local minimum. All components directed towards the same local minimum make a segment and are given a same label value in Figure 3c.



7	4	8	12	11	3
7	7	8	12	11	7
13	13	15	16	16	13
19	19	18	17	15	7
20	18	17	16	15	5

→	m	←	←	→	m
↗	↑	↖	←	↗	↑
↑	↑	↖	↖	↗	↑
↑	↑	↑	→	↘	↓
→	→	→	→	→	m

0	0	0	0	1	1
0	0	0	0	1	1
0	0	0	0	1	1
0	0	0	2	2	2
2	2	2	2	2	2

(a) The original image
(b) Each pixel connect to lowest minimum
(c) The image with labels
Figure 3: Basic concept of connected components approach

4. Hardware Implementation

In this section, hardware architectures for the pre-processing stage and segmentation stage are discussed for a FPGA implementation. Initially, image is processed by the pre-processing module and then the pre-processed image is given to the segmentation module. Block diagram of hardware implementation is show in Figure 4. The original image is loaded in the external memory for processing. Image pixels (top left to bottom right) are stored in the external memory at subsequent addresses (starting address to end address). Stream cache module is used for read and write operations with the external memory and image processing module (pre-processing or segmentation). Stream cache, pre-processing and segmentation module are discussed in detail in following sections. VHDL component descriptions of all modules are given in the appendix. Image size (height and width) is generic parameter for all modules.

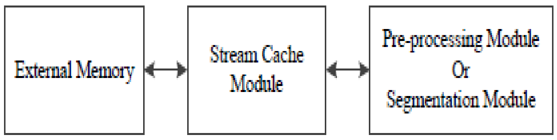


Figure 4: Block Diagram of Hardware implementation

A. Stream Cache Module

Stream cache module is used to make image data communications with the external memory which is designed by Robert Bosch GmbH. The basic concept of the stream cache module is explained in this section [5].

Top level controller module (Pre_processing_Ctrl) makes interface with the stream cache module and slave controller. Pre-processing module is connected to this top level controller module. G_SYSBUS_AWIDTH and G_APPL_DWIDTH are generics for the system address bus and application data bus, both are set to constant value of 32. Pre-processing module gets P_ctrl_start_sl signal from the slave controller which notifies to start reading of the data from external memory using the stream cache.

In read data operations, stream cache takes 64 bytes burst from the external memory during initialization. This packet is stored in the internal memory (block RAMs) of FPGA. When RD_re signal is given then data is read from this internal memory. When the number of bytes in the internal memory are lower than constant value (e.g. 32), then the stream cache reads new packet from the external memory. In write data operations, data is written to internal memory when WR_we signal is activated.

When number of bytes are 64, and if data bus is available then all 64 bytes are written to the external memory. If data bus is busy then output data can be written to the internal memory buffer until buffer is not full. If internal memory buffer gets full and data bus is still busy then it is necessary to deactivate the WR_we signal. All signals of the stream cache module are initialized with the zero. First, starting address of memory read is set in RD_Start_Addr signal. After getting P_ctrl_start_sl signal high for one clock cycle, RD_Init signal is set to high for one clock cycle.

RD_Busy signal defines busy status of the memory data bus or not enough space in the internal memory buffer and it is high in busy mode. RD_Busy signal goes high after the initialization. When RD_Busy signal goes low again which indicate that initialization is finished. When data is required to read, RD_re signal is set to high for given number of clock cycles. If RD_re signal is set to high only for one clock cycle then it reads only one data and load it into RD_Data signal. It is needed to check RD_Busy signal every time before RD_re signal is given and RD_Busy should be low when RD_re signal is asserted high.

B. Pre-processing Module

The input gray scale image is given to pre-processing module from the external memory. The pre-processing module gets one new pixel on every clock cycle from the external memory, processes it and writes back a processed pixel to the external memory. Pre-processing module is designed using a pipeline concept in such a way that it processes a new pixel on each clock cycle. This module has components like serializer, 3 x 3 moving window module, median filter, morphological gradient, thresholding and Serial-In-Parallel-Out (SIPO) shift register module. Flow of the input data and controller signals from one module to another is shown in below figure 5.

The pixel size is 8 bit in the gray scale image but only 32 bit data read or write operations are performed with the external memory on every clock cycle. There are total four pixels available on each clock cycle. The reason behind 32 bit data reading

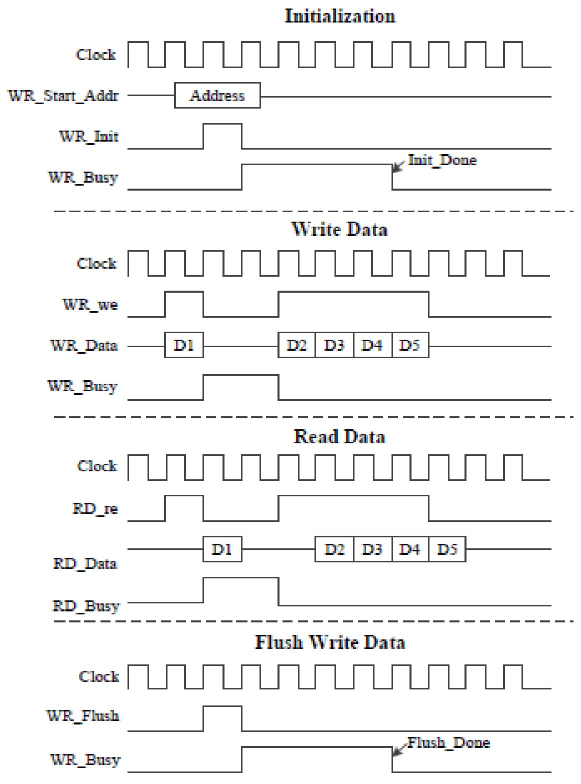


Figure 5: Stream cache - initialization, read, write and flush operations

or writing on each clock cycle is to make read and write operations compatible with the segmentation module because segmentation module is designed to process 32 bit data on each clock cycle. Stream cache module can not switch to different data width format for read and write operations during the run time, so either it needs to set for 8 bits, 16 bits or 32 bits data width.

C. Serializer

As the pre-processing module is designed to process only single gray image pixel (8 bit) per clock cycle, serializer is needed to serialize the data. It loads 32 bit data in one clock cycle and provides same data as a stream of four 8 bit data in four clock cycles (one clock cycle for one 8 bit data). In 32 bit data, eight most significant bits (MSB) are part of the first pixel. Eight bits are shifted left on each clock cycle, and a new pixel is shifted to the MSB position and taken as an output data. The schematic diagram is shown in Figure 6. Control and data inputs are captured and new output data is formed on rising-edge of the clock signal. Reset signal is active low synchronous reset which reset the serializer. When enable is deasserted (Low), all the synchronous inputs are ignored and internal data of serializer are not changed.

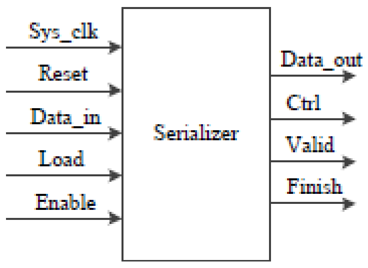


Figure 6: Serializer module schematic

Input data (32 bit width) is given to data in signal and 8 bit output data is provided by data out signal. Valid signal is asserted high when first 8 bit output data of the given 32 bit input data is ready. Finish signal goes high when total number of output data from the serializer are equal to the image size.

D. Segmentation Module

Segmentation module performs computation for the image segmentation. RD_Busy signal and WR_Busy signal indicate whether memory data bus is free or busy for data processing with external memory. When segmentation module cannot get the input data or cannot write the output data because of the memory data bus is busy with data processing of other modules, then this module needs to be halt. When data bus is available again for the segmentation module, then it can resume computation from the last halted state.

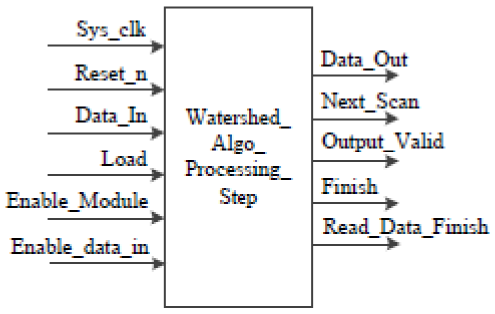


Figure 7: Segmentation module schematic

Figure 7 represents schematic diagram of the segmentation module. It has pixel width, image row width, image height, VMAX and LMAX as generic parameters. Enable module signal is used to enable all ports and internal digital logics. Input data is given to Data In signal and output data is provided by Data_Out signal. Enable data in signal indicates that when it is asserted, input data is given to internal register from the Data In signal. Output Valid signal is activated when first valid data is available on the Data_Out signal. When one scan of the image read, segmentation computation and write back to the external memory are completed, then finish signal is asserted for one clock cycle. Next Scan represents that subsequent image scans are required or not for the image segmentation. When Next Scan is active high after the finish signal, then subsequent image scans are necessary for the segmentation. When it is low, then the subsequent image scans are not required for the segmentation and segmentation of the given image is completed.

State Machine for Segmentation Module

Three state machines are used for data and control signals flow. First state machine is needed for the memory read operations only during the first image scan, second one is needed for the memory read operations for all subsequent image scans except the first image scan, and third one is needed for the memory write operations for all image scans. S First read signal (type_std_logic) is used to notify whether it is a first image scan or not. It initializes with high and after the first image scan is finished,

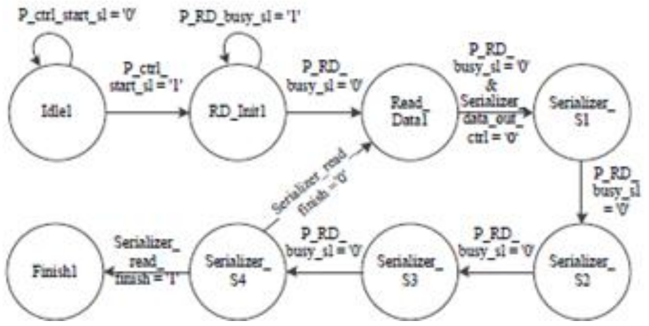


Figure 8: State diagram of memory read operation (only for first scan using serializer) for segmentation module

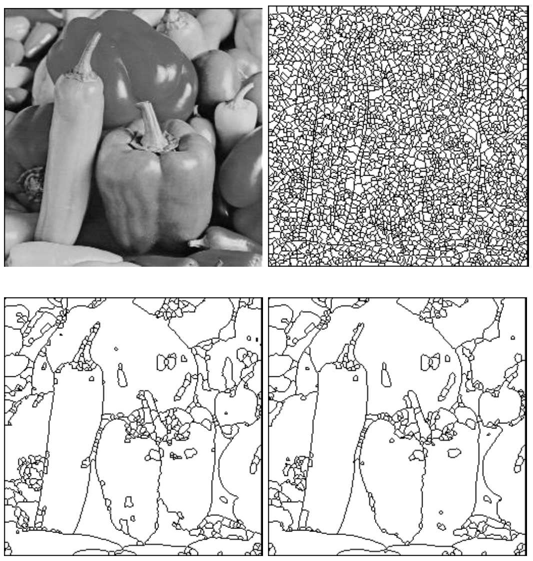
it is set to low. Segmentation module gets valid input data in read data state and writes back valid output data in write data state.

Figure 8 shows state machine diagram for the memory read operations during the first image scan. Serializer module is used in the first image scan. Initially, state is in Idle1 state. Starting address location of the memory for read operations and memory read in forward direction signal are set in the Idle1 state. When slave controller asserts P_ctrl_start_sl signal, state is changed to RD_Init1 state. RD_Init1 state waits until P_RD_busy signal goes low, and after that state moves to the Read_Data state. Read request is given to the external memory in the Read_Data state. Serializer module asserts Serializer data_out_ctrl signal which indicates that new input data (32 bit) is possible to read from the external memory and load into the serializer module. Serializer module gives one output data (8 bit) on each subsequent states, Serializer S1, Serializer S2, Serializer S3 and Serializer S4. Serializer data out ctrl signal is asserted high after each last 8 bit output data for the given 32 bit input data. Serializer module activates the valid data signal with each output data which notifies the segmentation module to load new 32 bit input data. Serializer module counts number of output data and when this number is equal to the image size, it asserts Serializer read_finish signal. After the Serializer read_finish signal is activated for a one clock cycle, data read from the memory is stopped and state moves to Finish1 state. S _First _read signal is set to low in Finish1 state, so data is given directly to the segmentation module in next subsequent scans without using the serializer module.

5. Performance Measurements

Connected components based watershed image segmentation algorithm gives over-segmentation without using the pre-processing stage. Watershed based image segmentation is only applied to the gradient image. Original pepper image is converted to the gradient image which is segmented without applying median filter and thresholding, and output image is over-segmented as shown in Figure 9b.

Total number of labels decrease as the threshold value is increased. If threshold value is selected very high, then some boundaries of segments are not preserved. Figure 9 shows segmented images with different threshold value for the pepper image (256x 256). Total numbers of labels in over-segmented image of Figure 9b are 2987 whereas after using median filter and the threshold value of 12, number of labels are reduced to 660.



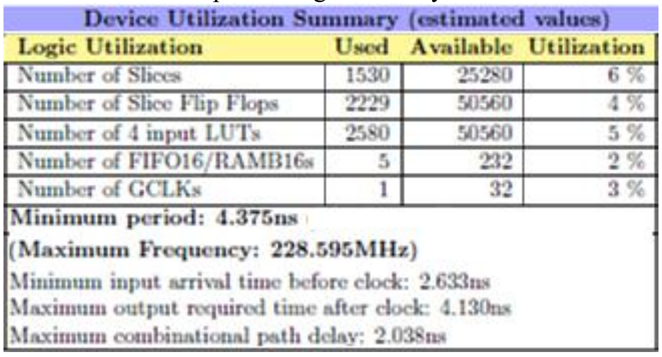
(a) Original image (256 x 256)
(b) Segmentation of gradient image without median filter and thresholding
(c) Segmented image (Threshold=12)
(d) Segmented image (Threshold=14)
Figure 9: Segmentation results with different threshold values for pepper image (256 x 256)

Synthesis results of the pre-processing and segmentation modules are discussed under this section. The target FPGA device for synthesis is Xilinx “Virtex-4" FPGA family with device id “xc4vfx60", package id “12ff672 " and speed grade “-12". The size of input image is 512 x 512 pixels. The frequency unit is MHz (Mega Hertz) and time unit is ms (mili seconds) or ns (nano seconds).

Table 1 describes synthesis results for the pre-processing stage. Maximum operating frequency of the pre-processing design is 228.59 MHz. In pre-processing module, one pixel is processed in single clock cycle. An image (512 x 512) takes 1.15 msec, if the system runs on the maximum operational frequency.

Total image processing time = Number of pixels ×Minimum period= 512 × 512 × 4:375 ns
= 1:15 ms

Table 1: Pre-processing module synthesis result



Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	1530	25280	6 %
Number of Slice Flip Flops	2229	50560	4 %
Number of 4 input LUTs	2580	50560	5 %
Number of FIFO16/RAMB16s	5	232	2 %
Number of GCLKs	1	32	3 %

Minimum period: 4.375ns
(Maximum Frequency: 228.595MHz)
Minimum input arrival time before clock: 2.633ns
Maximum output required time after clock: 4.130ns
Maximum combinational path delay: 2.038ns

6. Conclusion

Different images are tested for segmentation quality and execution time. Segmentation results are visually acceptable and almost identical segmentation results can always be obtained using the given implementation. The computation time for a 512 x 512 image is about 35 to 45 milliseconds with the implemented segmentation architecture. The use of external memory can also be avoided by using the FPGA device with the larger internal memory. Single pipelined segmentation unit uses very few hardware resources, so it is possible to use multiple segmentation units to achieve higher performance.

References

[1] A. Bieniek, A. Moga, An e_cient watershed algorithm based on connected components, Pattern Recognition 33, pp.907-916, 2000.
[2] M. Sonka, V. Hlavac, and R. Boyle, Image Processing, Analysis, and Machine Vision, PWS Publishing, 1999.
[3] Dr. Sukhendu Das, Lecture notes, IIT Madras,India, http://vplab.iitm.ac.in/courses/CV_DIP/PDF/lect-Segmen.pdf
[4] MATLAB Notes, http://www.mathworks.de/company/newsletters/news_notes/win02/watershed.html
[5] *Dang Ba Khac Trieu and Tsutomu Maruyama"* Implementation of a parallel and pipelined watershed algorithm on fpga" ieee transaction of computer vision, 2006