IMPLEMENTATION OF THE HARDWARE MODULE FOR IMAGE SKELETONIZATION SYSTEM

Jana Janković
Faculty of Technical Sciences
University of Novi Sad
e-mail: jana.jankovic99@yahoo.com

Abstract

This paper presents realization and implementation of a system for skeletonization of binary images. The skeletonization algorithm is based on iterative parallel thinning approach. The purpose of this work was to exploit the parallelism of algorithm by implementing it in hardware block that will process more image pixels in parallel. The hardware block for algorithm acceleration is implemented and embedded in the system.

Keywords: Skeletonization, FPGA, Hardware accelerator.

INTRODUCTION

Skeletonization is an image pre-processing technique that is important in a number of applications like pattern recognition, data compression and data storage. All algorithms can be classified as either iterative or non-iterative. In iterative methods, algorithms produce a skeleton by examining and deleting contour pixels through an iterative process in either sequential or parallel way.[1] The Zhang-Suen algorithm is probably the most used parallel algorithm for skeletonization. Instead of implementing algorithms by programs executed in a general purpose computer, to obtain better performance and efficiency, it is sometimes beneficial to realize an algorithm in custom hardware. The register transfer methodology provides a systematic way to convert an algorithm into hardware. [2]

EXPOSITION

In a parallel algorithm, the deletion of pixels in the *nth* iteration would depend only on the result that remains after the $(n-1)th$ iteration; therefore, all pixels can be examined independently in a parallel manner in each iteration.[3]

DISCLAIMER:
This work resulted from the bachelor thesis whose mentor was Vuk Vranjković, PhD.

In hardware module every pixel will be examined by functional unit called worker. Module has parametrized number of workers (NUM_WORKERS) and parametrized BRAM dimensions (WIDTH, DEPTH) which requires three different architectures of hardware module. Workers will process a pixel in one cycle, so the maximum number of workers is limited to a number for two smaller than the width of the BRAM.

Interface

- Input interface

  height_i – type STD_LOGIC_ VECTOR ld(DEPTH)-1 downto 0) – represents the number of image rows.

  last_px – type STD_LOGIC_ VECTOR (ld(PARAMETAR) downto 0) – represents the number of worker who will receive the right border pixel at the entrance.

  num_px – type STD_LOGIC_ VECTOR (ld(WIDTH-2)/ PARAMETAR+1) downto 0) – represents the number of pixels that the worker who does not process the last pixel, should process in one row of the image.

- AXI-*Stream* slave interface

  sdata – type STD_LOGIC_ VECTOR (63 downto 0) – the input data bus, represents the 64 pixels of the image that needs to be processed.

  svalid – type STD_LOGIC– indicates whether the data sent by the external component is valid.

  sready – type STD_LOGIC– indicates whether the module is ready to receive the data sent by the external component.

- AXI-*Stream* master interface

  mdata – type STD_LOGIC_ VECTOR (63 downto 0) – output data bus, represents 64 pixels of the processed image.

  mvalid – type STD_LOGIC– indicates whether the data sent by the module is valid.

  mready – type STD_LOGIC– indicates whether the external component is ready to receive the data sent by the module.

- Command interface

  start – type STD_LOGIC – module starts working when the start signal is activated.

- Status interface

  ready – type STD_LOGIC - asserted when the module is ready to accept new inputs.

In addition to these ports, the digital system must also have standard ports for clock and reset signals, clk and rst. Figure 1 shows the complete interface of the image skeletonization module.

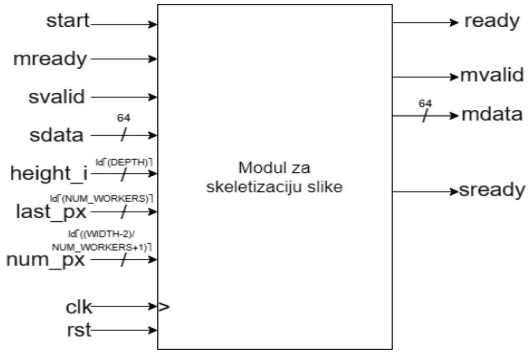

Fig. 1. *Hardware module interface*

Controlpath

In the ASM diagram, for all three architectures, the first six and the last three states are identical. In the first, idle state, it waits for an instruction to start processing. In the second, shift state, the content of the register is shifted to the left, while the data arriving from the DMA controller is written to the beginning. In the third state, the contents of the register is written to BRAM. The next three states load the first three rows of the image from BRAM into the three registers, so that the workers can process them. In state l1, it is checked whether the skeletonization process has been completed. In the penultimate state, content is loaded from the appropriate location from BRAMA into the register. In the last state (shift1), via the AXI-Stream interface, the lower 64 bits are sent and then the contents of the register are shifted to the right.

For the first architecture, the BRAM width minus two is equal to the number of workers, it is specific that all the pixels of one row of the image are processed in one clock. This happens in the worker state. At the same time, the outputs of the register red2_reg are fed to the input of the register red1_reg, the outputs red3_reg are output to the inputs red2_reg, and the next line of the image from BRAM is fed to the inputs red3_reg. In this way, a new row of the image will be processed in the next cycle. The results of workers are also placed in the rez_reg register and its contents are written into BRAM in the next cycle. The outputs of the worker representing the variable flag are fed to the inputs of the register flags_reg. Figures 2 and 3 show the ASM diagram of the first architecture.

The second architecture, the BRAM width minus two is divisible by the number of workers. Due to the smaller number of workers, it is necessary to move the contents of the registers every cycle, in order to allow all pixels to appear at the worker inputs. This happens in the states workeri2 and workeri3. The content of the rez_reg register is also moved to the left, while the results of the worker processing are written at the end. In the worker state, as with the previous architecture, the content of the register red2_reg is written

into the register red1_reg, the content of red3_reg into red2_reg and into red3_reg the new image row. The difference is that now it is necessary to properly connect the outputs and inputs of the registers, because, due to the previous shift, the pixels located at the end of the row were at the beginning.

The difference between the second and third architectures is in the writing the results of workers in the register rez_reg. In the third architecture, the number of pixels processed is not divisible by the number of workers, which means that when the last group of pixels is processed, pixels that have already been processed will appear at the inputs of some workers, and their processing results are not written into rez_reg.

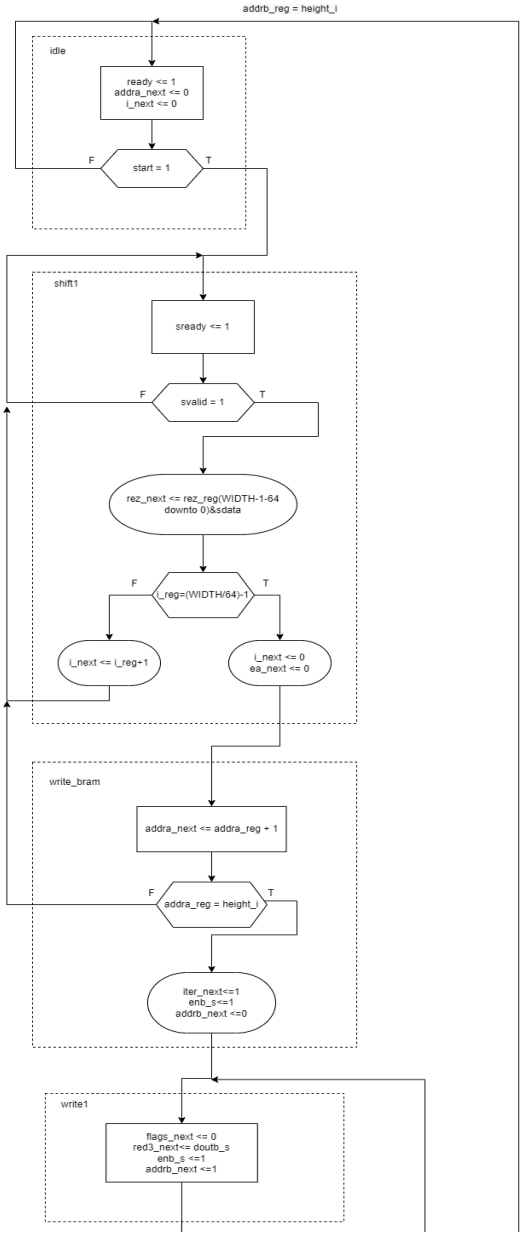

Fig. 2. *ASMD of first architecture- first part*

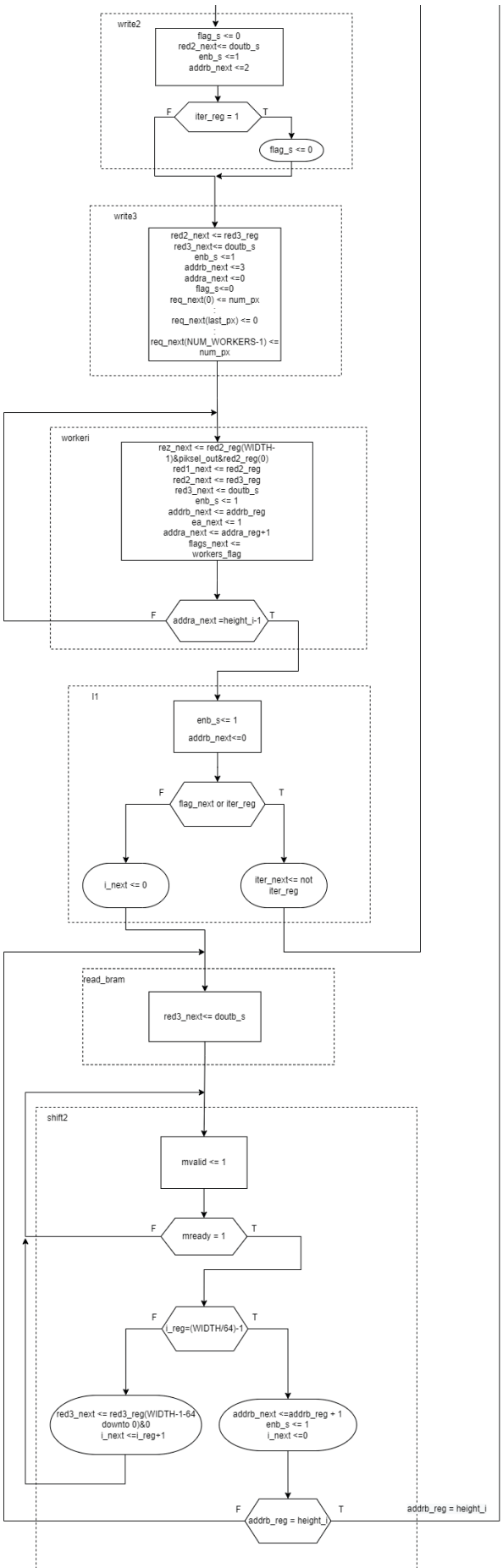

Fig. 3. *ASMD of first architecture- second part*

Datapath

Hardware module for skeletonization consists of workers that examine pixels. Structure of the worker is shown in figure 4.

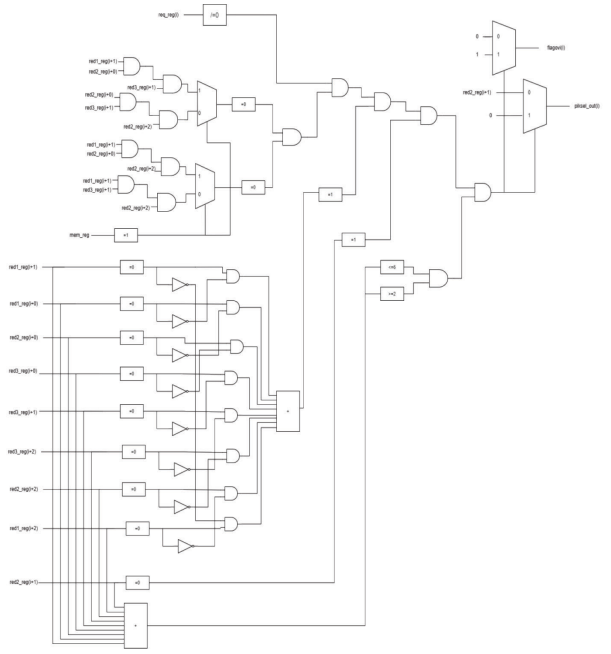

Fig. 4. *Structure of the worker*

The inputs of the worker are connected to the outputs of the registers in which the rows of the image are located. Additional worker inputs are register outputs that indicate whether a pixel needs to be processed and whether it is the first or second iteration of the algorithm. The worker outputs are connected to the register in which the processed image row is placed. The additional output of the worker is connected to the register flags_reg in which the values of the flag variable are placed.

The output of that register goes to the inputs or gate to determine if at least one worker has changed the pixel value. Since the number of workers can be large, in order to reduce the path, instead of a cascaded chain of two-input or gates, a parameterized tree structure was implemented (Figure 5). The output of the circuit is fed to the input of the register whose output is fed back to the input of the circuit. This needs to be done so that the value of the flag variable, from the processing of the previous group of pixels, will be taken into account during the next calculation, and in this way it will be possible to know at the end of passing through the entire image whether any pixel of the image has been changed.

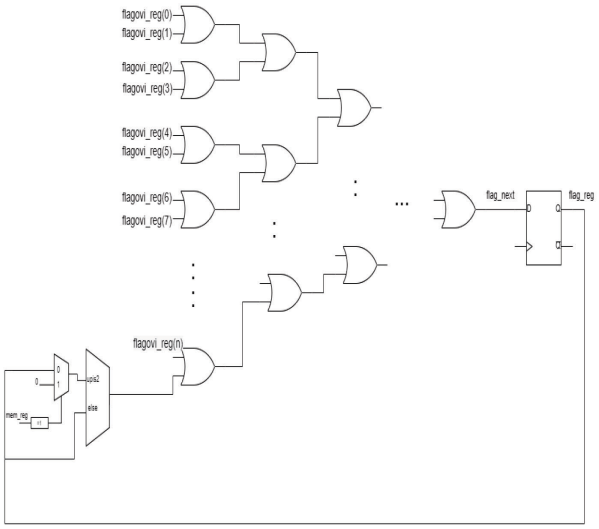

Fig. 5. *Parameterized tree structure of or gate*

For each of the module architectures, there is a data routing network and memory elements. In case of the first architecture, there are eleven registers in the module (Figure 6): three registers are used to save the rows of the image that are being processed, one register is used to store the processing result, two registers are used to store address values, and one register is used to store write permissions. in memory, one register helps in calculating positions, one register that stores information about whether it is the first or second iteration, one register based on the value of which the worker decides whether to process a pixel and one register that stores the values of the flag variable.

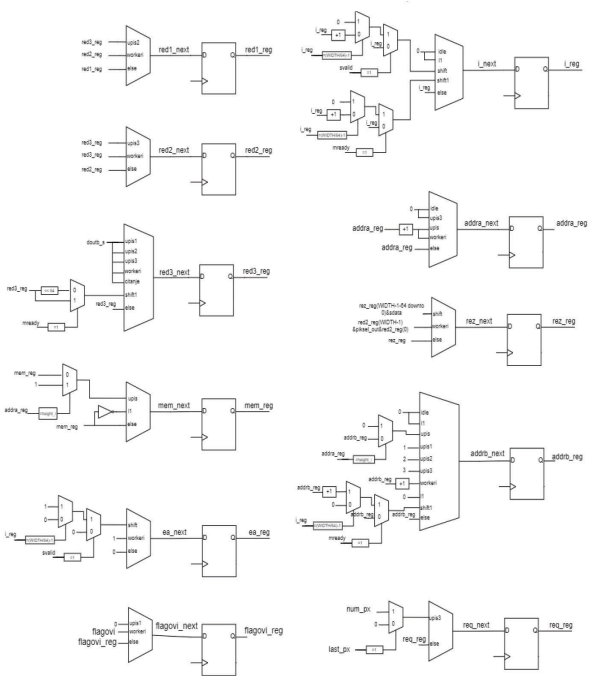

Fig. 6. *Data routing network and memory elements for first architecture*

In case of the second and third architectures, a register was added to track which group of pixels is being processed. The other registers are the same, the difference is in the values that are written into them.

Inside the module there is a BRAM memory with parameterized dimensions in which the image is stored. Figure 7 shows which datapath signals the BRAM memory is connected to.

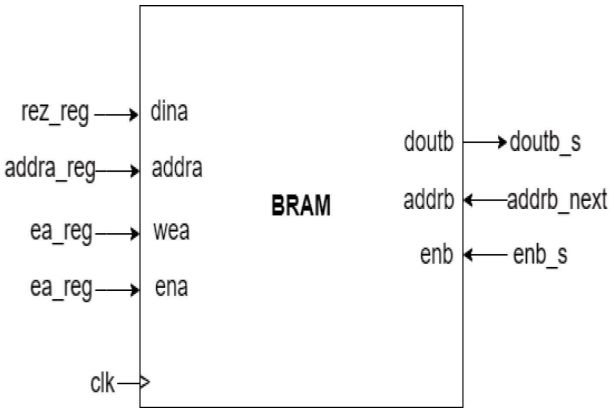

Fig. 7. *Signals connected to BRAM interface*

DMA reads 64-bit data from memory and sends it to the module, which then stores it in BRAM, one location of which can be more than 64 bits wide. Due to the difference in width of the data being exchanged, it is necessary to perform a width conversion. This is done using the shift register. After the data is placed in the lower 64 bits of the register, the content of the register is shifted, so that the next data can be received. When the register is full, the contents of the register are written to BRAM. When data is sent to the DMA component, the contents of BRAM are first written to a register. The upper 64 bits of the register are connected to the m_axis_s2mm _tdata bus of the DMA component. After the first 64 bits are read, the contents of the register are shifted to read the next 64 bits. The process is repeated until the entire image is sent.

Integration in Zynq platform

After the IP core is designed, the system can be implemented. Using the Vivado integrator, the IP core is connected to existing components into a single system (Figure 8). The system consists of a Zynq7 processor, a DMA controller, an IP core that performs image skeletonization, and two interconnect components. The processor is connected to the DMA controller and the IP core via AXI-Lite interface using an AXI interconnect. The IP core is connected via AXI-Stream slave and master interface to the DMA controller. The DMA controller is connected via the AXI-Full interface to the memory.

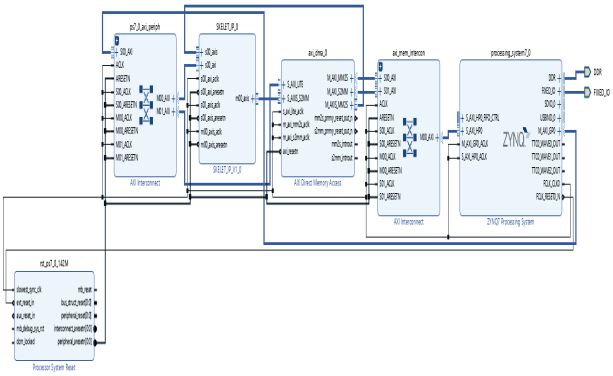

Fig. 8. *Block diagram of the system*

Resource utilization and operating frequency

After the implementation of the system, where the values of 510 worker and 512 for BRAM dimensions were selected for the IP core parameters, a time analysis was performed. The maximum operating frequency of the system is 142MHz.

Using the Vitis tool, an application was created that tests the system, and using the timing functions, it was measured that the delay of the system, is 241us, which means that its throughput is 4165 images per second.

The consumption of hardware resources is shown in Tables 1, 2. LUTs were used the most, followed by BRAM memory, while DSPs were not used at all in the design. Figure 9 shows the spatial distribution of used resources.

Table 1. *Logic gates utilization*

Site Type	Used	Fixed	Available	Util%
Slice LUTs	14122	0	17600	80.24
LUT as Logic	13926	0	17600	79.13
LUT as Memory	196	0	6000	3.27
LUT as Distributed RAM	22	0		
LUT as Shift Register	174	0		
Slice Registers	7107	0	35200	20.19
Register as Flip Flop	7107	0	35200	20.19
Register as Latch	0	0	35200	0.00
F7 Muxes	4085	0	8800	46.42
F8 Muxes	2040	0	4400	46.36

Table 2. *BRAM utilization*

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	16.5	0	60	27.50
RAMB36/FIFO*	15	0	60	25.00
RAMB36E1 only	15			
RAMB18	3	0	120	2.50
RAMB18E1 only	3			

For systems with a smaller number of workers, the difference in frequency is not significant, while the differences in resource consumption can be seen in Table 3.

Table 3. *Resource utilization for different parameter values*

Number of workers	BRAM dimensions	LUT utilization	Flip-flop utilization	BRAM utilization
1	256x256	4249 (24%)	5042 (14%)	13 (21%)
127	256x256	7092 (40%)	5405 (15%)	13 (21%)
254	256x256	8693 (49%)	5540 (15%)	13 (21%)
1	1024x1024	8904 (50%)	8266 (23%)	37.5 (62%)
340	1024x1024	16247 (92%)	9631 (27%)	37.5 (62%)

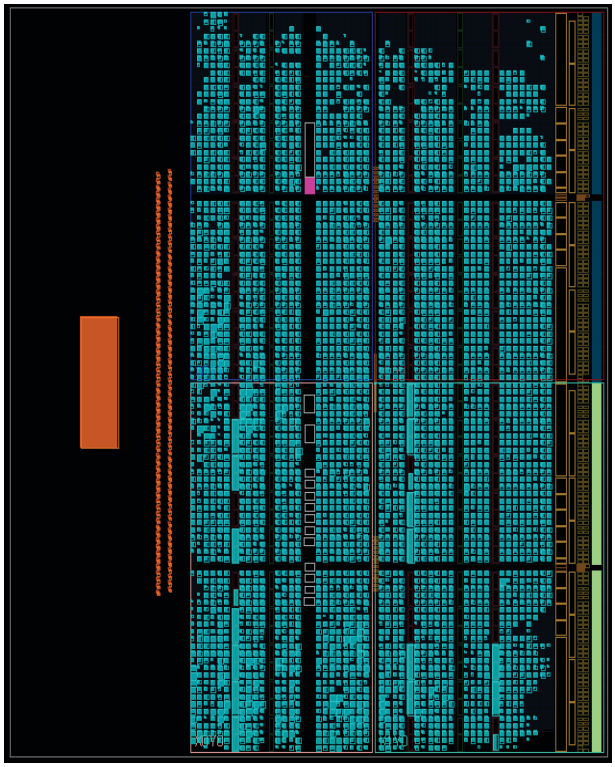

Fig. 9. *Spatial distribution of used resources*

CONCLUSION

The goal of this paper was to implement a system for image skeletonization. For implementation of the skeletonization process, the Zhang-Suen algorithm was chosen. The hardware block was successfully implemented in VHDL, using RTL methodology. The realized IP core has a parameterized number of functional units - workers that process pixels, which can be changed according to available resources and desired image processing speed. The IP core, using the AXI-Stream interface, through the DMA controller receives and sends the image before and after processing.

Implementing a system that would potentially utilize less resources than implemented system, would require, the implementation of an AXI controller inside the core, instead of DMA controller, which would instead of the AXI-Stream interface, use AXI-Full interface for memory access.

REFERENCE

[1] Khalid Saeed, Marek Tabedzki, Mariusz Rybnik, Martin Adamski, „K3M: A universal algorithm for image skeletonization and a review of thinning techniques”, Int. J. Appl. Math. Comput. Sci., 2010, Vol. 20, No. 2, 317–335

[2] Pong P. Chu, „RTL Hardware Design Using VHDL”, Wiley-Interscience, 2006

[3] Lynda Ben Boudaoud, Abderrahmane Sider, Abdelkamel Tari, „A new thinning algorithm for binary images”, 3rd International Conference on Control, Engineering & Information Technology (CEIT) 2015