ISSN (Online) : 2319 - 8753
ISSN (Print) : 2347 - 6710

IJIRSET

International Journal of Innovative Research in Science, Engineering and Technology

Volume 3, Special Issue 3, March 2014

2014 International Conference on Innovations in Engineering and Technology (ICIET'14)

On 21st & 22nd March Organized by

K.L.N. College of Engineering, Madurai, Tamil Nadu, India

Binary Image Processing Implementation Using Micro blaze Processor

R.Ponneela Vignesh[#1], R.Senthil Kumar[*2]

Assistant Professor, [#] Dept of ECE, Tamilnadu College of Engineering, Coimbatore, Tamilnadu, India.

ABSTRACT:-Field Programmable Gate Array (FPGA) technology has become a viable target for the implementation of real time algorithms suited to video image processing applications. The unique architecture of the FPGA has allowed the technology tope used in many applications encompassing all aspects of video image processing. Among those algorithms, linear filtering based on a 2D convolution, and non-linear 2D morphological filters, represent a basic set of image operations for a number of applications. In this work, an implementation of linear and morphological image filtering using a FPGA, Xilinx, Spartan 3E, with educational purposes, is presented. The system is connected to a USB port of a personal computer, which in that way form a powerful and low-cost design station. The FPGA-based system is accessed through a Mat lab graphical user interface, which handles the communication setup. A comparison between results obtained from MATLAB simulations and the described FPGA-based implementation is presented.

KEY WORDS*:* FPGA, Morphological filtering, VLSI architecture.

I. INTRODUCTION

Morphological processing is constructed with operations on sets of pixels. Binary morphology uses only set membership and is indifferent to the value, such as gray level or color, of a pixel. We will examine some basic set operations and their usefulness in image processing. We will deal here only with morphological operations for binary images. This will provide a basic understanding of the techniques. Morphological processing for gray scale images requires more sophisticated mathematical development. Morphological processing is described almost entirely as operations on sets. In this discussion, a set is a collection of pixels in the context of an image. Our sets will be collections of points on an image grid G of size N × M pixels [2].

The basic concepts and analytic tools in mathematical morphology can be found, for binary images, in set theory and integral geometry. In mathematical morphology, a binary image is represented as a subset of the 2D Euclidean space, R2 or its digitized equivalent Z2, and image processing transformations are represented as set mappings between collections of subsets [22]. Erosions and dilations are the two fundamental morphological operators. They are characterized by a subset called the structuring element that is used to probe the image. Mat heron has captured the ubiquity of morphological operators by demonstrating that any increasing (i.e., operators which preserve signal ordering) and translation-invariant operators can be represented as unions (resp., intersections) of erosions (resp., dilations) [22]. In most applications of mathematical morphology, the structuring element remains constant in shape and size as the image is probed. Hence, the focus of mathematical morphology has been mostly devoted to translation-invariant operators [15].

There are numerous applications in military, industrial and robotic vision systems that require ultra high speed image processing on the order of 1,000 frames per second. Binary image morphological operations are well suited to a large class of basic image processing applications. These operations include image analysis tasks such as shape recognition, image segmentation, noise reduction, and feature extraction. Applying morphological operators across an entire image space in an iterative fashion usually implies a computationally expensive algorithm. However, custom hardware can be used for ultra-fast implementations of a number of basic binary morphological operations [12].

The remainder of the paper is organized as follows: Section II reveals the morphological processing. Section III presents the proposed architecture and the hardware implementation. Section IV shows the results and discussion. Section V provides the conclusion and future work.

II. MORPHOLOGICAL PROCESSING

The term morphology refers to the study of shapes and structures from a general scientific perspective [12]. Also, it can be interpreted as shape study using mathematical set theory. In image processing, morphology is the name of a specific methodology for analyzing the geometric structure inherent within an image. The morphological filter, which can be constructed on the basis of the underlying morphological operations, are more suitable for shape analysis than the standard linear filters since the latter sometimes distort the underlying geometric form of the image.

Some of the salient points regarding the morphological approach are as follows:

1. Morphological operations provide for the systematic alteration of the geometric content of an image while maintaining the stability of the important geometric characteristics.

2. There exists a well-developed morphological algebra that can be employed for representation and optimization.

3. It is possible to express digital algorithms in terms of a very small class of primitive morphological operations.

4. There exist rigorous representations theorems by means of which one can obtain the expression of morphological filters in terms of the primitive morphological operations.

In general, morphological operators transform the original image into another image through the interaction with the other image of a certain shape and size, which is known as the structuring element [17]. Geometric features of the images that are similar in shape and size to the structuring element are preserved, while other features are suppressed. Therefore, morphological operations can simplify the image data, preserving their shape characteristics and eliminate irrelevancies. In view of applications, morphological operations can be employed for many purposes, including edge detection, segmentation, and enhancement of images [11].

A. Binary Dilation

Dilation is found by placing the center of the template over each of the foreground pixels of the original image and then taking the union of all the resulting copies of the structuring element, produced by using the translation. From Figure 2.3, it is clear how dilation modifies the original image with respect to the shape of the structuring element[13].

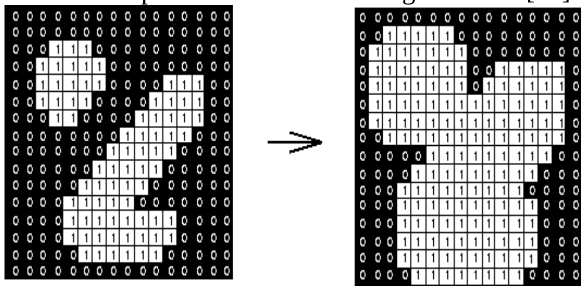


Fig. 1 Dilation: a 3×3 square structuring element

Dilation generally has an effect of expanding an image; so consequently, small holes inside foreground can be filled. In another sense, dilation can be a morphological operation on a binary image defined as:

$$g = f \oplus s$$

The **dilation** of an image f by a structuring element s (denoted $f \oplus s$) produces a new binary image $g = f \oplus s$ with ones in all locations (x,y) of a structuring element's orogin at which that structuring element s hits the the input image f, i.e. $g(x,y) = 1$

if *s* hits *f* and 0 otherwise, repeating for all pixel coordinates (*x*,*y*). Dilation has the opposite effect to erosion -- it adds a layer of pixels to both the inner and outer boundaries of regions [10].

B. Binary Erosion

The **erosion** of the original image by the structuring element can be described intuitively by template translation as seen in the dilation process. Erosion shrinks the original image and eliminates small enough peaks (Note: the terms 'expand' for dilation and 'shrink' for erosion refer to the effects on the foreground). Figure 2 clearly illustrates these effects. The original image is eroded with 7x7 disk-shape structuring element[3].

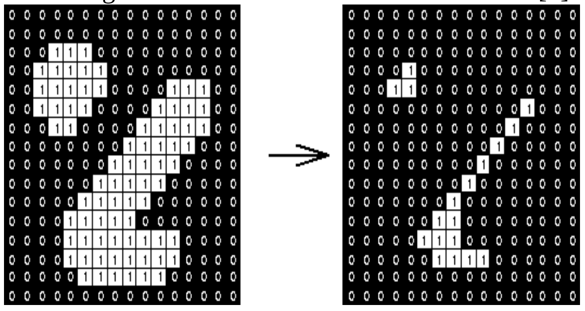


Fig. 2 Erosion: a 3×3 square structuring element

The **erosion** of a binary image *f* by a structuring element *s* (denoted $f \ominus s$) produces a new binary image

$$g = f \ominus s$$

With ones in all locations (*x*,*y*) of a structuring element's origin at which that structuring element *s* fits the input image *f*, i.e. $g(x,y) = 1$ is *s* fits *f* and 0 otherwise, repeating for all pixel coordinates (*x*,*y*). Erosion removes small-scale details from a binary image but simultaneously reduces the size of regions of interest, too. By subtracting the eroded image from the original image, boundaries of each region can be found: $b = f - (f \ominus s)$ where *f* is an image of the regions, *s* is a 3×3 structuring element, and *b* is an image of the region boundaries [19].

C. Binary Opening

The effects of the opening process on the original image are smoothing, reducing noise from quantization or the sensor and pruning extraneous structures [6]. These effects result from the fact that the structuring element cannot fit into the regions. Therefore, it can be said that the result of the opening process heavily depends on the shape of structuring elements. Figure 3 presents an example of the opening process.

The whole procedure of opening can be interpreted as "rolling the structuring element about the inside boundary of the image". The **opening** of an image *f* by a structuring element *s* (denoted by $f \circ s$) is erosion followed by dilation:

$$f \circ s = (f \ominus s) \oplus s$$

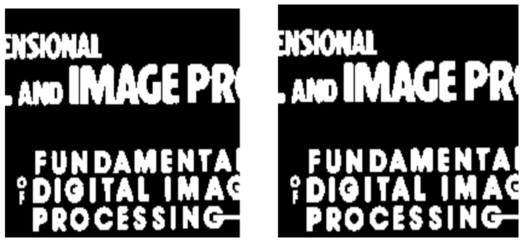


Fig. 3 a)Binary image b)Opening: a 2×2 square structuring element

D. Binary Closing

In the **closing** operation, dilation and erosion are applied successively in that order. Note that this order is reversed for the opening process. In another aspect, the closing process on a binary image can be defined as: The closing operation can be described as in Figure 4 as "rolling the structuring element on the outer boundary of the image."

The closing process has the effect of filling small holes in the original image, smoothing as the opening process does, and filling up the bay in the foreground. Sometimes, it is said that the closing has an effect of clustering each spatial point.

The **closing** of an image *f* by a structuring element *s* (denoted by $f \bullet s$) is a dilation followed by erosion:

$$f \bullet s = (f \oplus s_{\mathrm{rot}}) \ominus s_{\mathrm{rot}}$$

In this case, the dilation and erosion should be performed with a rotated by 180° structuring element. Typically, the latter is symmetrical, so that the rotated and initial versions of it do not differ [16].

Closing is so called because it can fill holes in the regions while keeping the initial region sizes. Like opening, closing is idempotent: $(f \bullet s) \bullet s = f \bullet s$, and it is dual operation of opening (just as opening is the dual operation of closing):

$$f \bullet s = (f^c \circ s)^c; \quad f \circ s = (f^c \bullet s)^c.$$

In other words, closing (opening) of a binary image can be performed by taking the complement of that image, opening (closing) with the structuring element, and taking the complement of the result.

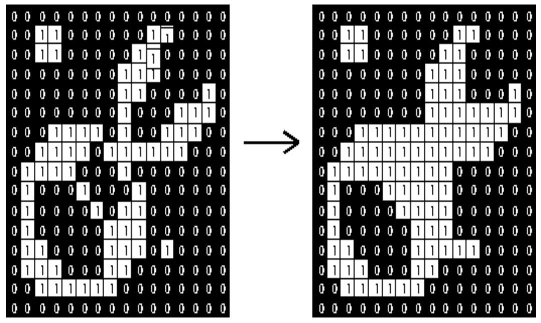


Fig. 4 closing with a 3×3 square structuring element

E. *Binary Hit and Miss Transform*

The **hit and miss transform** allows to derive information on how objects in a binary image are related to their surroundings. The operation requires a matched pair of structuring elements, $\{s_1, s_2\}$, that probe the inside and outside, respectively, of objects in the image:

$$f \circledast \{s_1, s_2\} = (f \ominus s_1) \cap (f^c \ominus s_2)$$

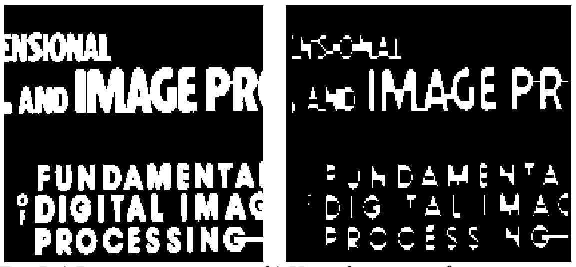


Fig. 5 a) Binary image b) Hit and miss transform

A pixel belonging to an object is preserved by the hit and miss transform if and only if s_1 translated to that pixel fits inside the object AND s_2 translated to that pixel fits outside the object. It is assumed that s_1 and s_2 do not intersect; otherwise it would be impossible for both fits to occur simultaneously.

It is easier to describe it by considering s_1 and s_2 as a single structuring element with 1s for pixels of s_1 and 0s for pixels of s_2; in this case the hit-and-miss transform assigns 1 to an output pixel only if the object (with the value of 1) and background (with the value of 0) pixels in the structuring element exactly match object (1) and background (0) pixels in the input image. Otherwise that pixel is set to the background value (0).

The hit and miss transform can be used for detecting specific shapes (spatial arrangements of object and background pixel values) if the two structuring elements present the desired shape, as well as for thinning or thickening of object linear elements.

III. PROPOSED SYSTEM

In this proposed system VLSI architecture is designed and implemented, which is to perform the binary image processing with high speed and reduced complexity. For that, the coprocessor Micro blaze is converted into morphological processing architecture using Xilinx platform studio in system C language and then tested in Spartan 3EDK FPGA kit. RS232 cable is used for interfacing the test circuit with PC. This hardware implementation can overcome the shortages of previous works not only that it can achieve accuracy, noise and also the speed in computation and low power consumption also.

A. *Processor Design Technique*

The Micro Blaze embedded soft core is a reduced instruction set computer (RISC) optimized for implementation in Xilinx field programmable gate arrays (FPGAs). See Fig.6. for a block diagram depicting the Micro Blaze core. Field-programmable gate arrays (FPGA'S) are flexible and reusable high-density circuits that can be easily re-configured by the designer, enabling the VLSI design / validation /simulation cycle to be performed more quickly and less expensive. Increasing device densities have prompted FPGA manufacturers, such as Xilinx and Altera, to incorporate larger embedded components, including multipliers, DSP blocks and even embedded processors. One of the recent architectural enhancements in the Xilinx Spartan, Virtex family architectures is the introduction of the Micro Blaze (Soft IP) and PowerPC405 hard-core embedded processor. The Micro blaze processor is a 32-bit Harvard Reduced

Instruction Set Computer (RISC) architecture optimized for implementation in Xilinx FPGAs with separate 32-bit instruction and data buses running at full speed to execute programs and access data from both on-chip and external memory at the same time. An interrupt controller is available for use with the Xilinx Embedded Development Kit (EDK) software tools.

The processor will only react to interrupts if the Interrupt Enable (IE) bit in the Machine Status Register (MSR) is set to 1. On an interrupt the instruction in the execution stage will complete, while the instruction in the decode stage is replaced by a branch to the interrupt vector (address Ox 10). The interrupt return address (the PC associated with the instruction in the decode stage at the time of the interrupt) is automatically loaded into general purpose register. In addition, the processor also disables future interrupts by clearing the IE bit in the MSR. The IE bit is automatically set again when executing the RTID instruction.

Due to the advancement in the fabrication technology and the increase in the density of logic blocks on FPGA, the use of FPGA is not limited to anymore to debugging and prototyping digital circuits. Due to enormous parallelism achievable on FPGA and the increasing density of logic blocks, it is being used now as a replacement to ASIC solutions in a few applications. Soft cores are technology independent and require only simulation and timing verification after synthesized to a target technology.

B. *Xilinx Platform Studio*

The Xilinx Platform Studio (XPS) is the development environment or GUI used for designing the hardware portion of your embedded processor system. B. Embedded Development Kit Xilinx Embedded Development Kit (EDK) is an integrated software tool suite for developing embedded systems with Xilinx Micro Blaze and PowerPC CPUs. EDK includes a variety of tools and applications to assist the designer to develop an embedded system right from the hardware creation to final implementation of the system on an FPGA. System design consists of the creation of the hardware and software components of the embedded processor system and the creation of a verification component is optional. A typical embedded system design project involves: hardware platform creation, hardware platform verification (simulation), software platform creation, software application creation, and software verification. Base System Builder is the wizard that is used to automatically generate a hardware platform according to the user specifications that is defined by the MHS (Microprocessor Hardware Specification) file. The MHS file defines the system architecture, peripherals and embedded processors]. The Platform Generation tool creates the hardware platform using the MHS file as input. The creation of the verification platform is optional and is based on the hardware platform. The MHS file is taken as an input by the Simgen tool to create simulation files for a specific simulator.

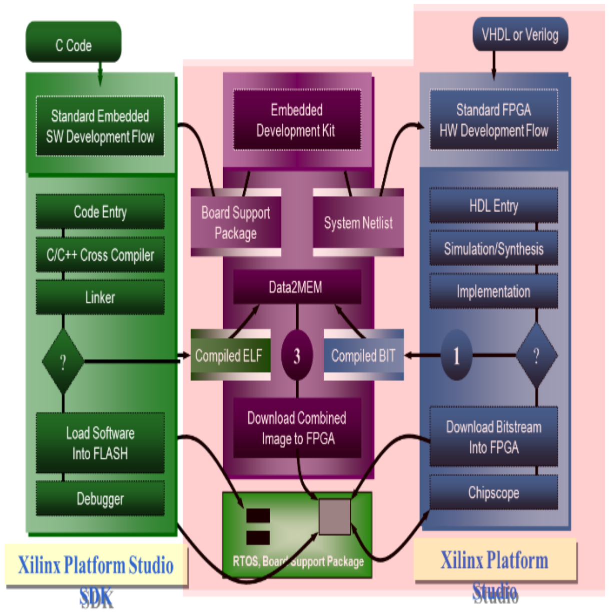


Fig. 6 Embedded Development Kit Design

Three types of simulation models can be generated by the Simgen tool: behavioral, structural and timing models. Some other useful tools available in EDK are Platform Studio which provides the GUI for creating the MHS and MSS files. Create Import IP Wizard which allows the creation of the designer's own peripheral and import them into EDK projects. Bit stream Initializer tool initializes the instruction memory of processors on the FPGA. GNU Compiler tools are used for compiling and linking application executables for each processor in the system.

There are two options available for debugging the application created using EDK namely: Xilinx Microprocessor Debug (XMD) for debugging the application software using a Microprocessor Debug Module (MDM) in the embedded processor

system, and Software Debugger that invokes the software debugger corresponding to the compiler being used for the processor. C. Software Development Kit Xilinx Platform Studio Software Development Kit (SDK) is an integrated development environment, complimentary to XPS, that is used for C/C++ embedded software application creation and verification. The software application can be written in a "C or C++" then the complete embedded processor system for user application will be completed, else debug & download the bit file into FPGA.

Then FPGA behaves like processor implemented on it in a Xilinx Field Programmable Gate Array (FPGA) device.

IV. RESULTS AND DISCUSSION

Experiments are performed on gray level images to verify the proposed method. These images are represented by 8 bits/pixel and size is 128 x 128. Image used for experiments are shown in below figure. The architectures were implemented in system c and placed and routed on Xilinx spartan3 XC3S200 FPGA, using Xilinx platform studio v.10.1

TABLE I

Comparison of Processor

Processor	[22]	[20]	This paper
Accuracy(bit)	7-bit	8-bit	16-bit
Speed(MHz)	10	28	50
Image processing	Binary	Binary and gray scale	Binary
Voltage (V)	4.5	3.3	2.5
Power consumption (mW)	148.5	120.43	50.35
Image pixels	16*16	64*64	128*128
Architecture	1-D SIMD	1-D SIMD	2-D MIMD

After convert the header file in the Mat lab the XPS could be processed. In XPS the impulse C language is used. The processing of the input image and the watermarked image could be processed in the SPARTAN 3EDK. These could be processed by converting these codings into the Bit streams. Then the net list will be created for that bit streams to get the proper output.

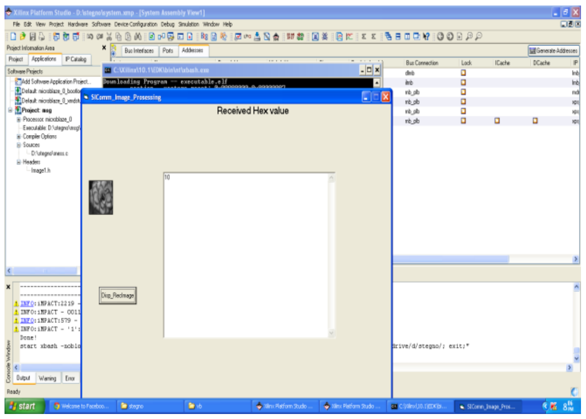

Xilinx Platform Studio - D:\stegno\system.xmp - [System Assembly View1]
C:\Xilinx\10.1\EDK\bin\nt\xbash.exe
Downloading Program -- executable.elf
SIComm_Image_Prosessing
Received Hex value
10
Disp_RecImage

Fig. 7 Input image

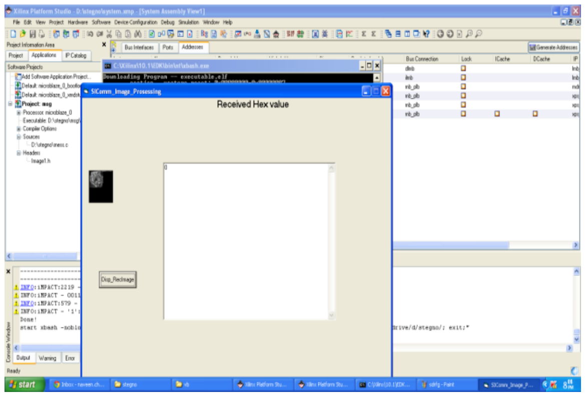
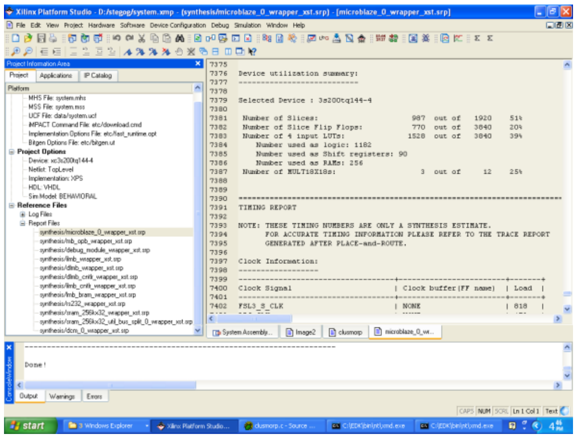

Xilinx Platform Studio - D:\stegno\system.xmp - [System Assembly View1]
C:\Xilinx\10.1\EDK\bin\nt\xbash.exe
Downloading Program -- executable.elf
SIComm_Image_Prosessing
Received Hex value
Disp_RecImage

Fig. 8 Output image


Xilinx Platform Studio - D:/stegog/system.xmp - (synthesis/microblaze_0_wrapper_xst.srp) - [microblaze_0_wrapper_xst.srp]
File Edit View Project Hardware Software Device Configuration Debug Simulation Window Help
Project Information Area
Project Applications IP Catalog
System Assembly... Image2 clusmorp microblaze_0_w...
Output Warnings Errors
CAPS NUM SCRL Ln 1 Col 1 Text
start 3 Windows Explorer Xilinx Platform Studio... clusmorp.c - Source ... C:\EDK\bin\nt\xmd.exe C:\EDK\bin\nt\xmd.exe

```
Platform
   MHS File: system.mhs
   MSS File: system.mss
   UCF File: data/system.ucf
   iMPACT Command File: etc/download.cmd
   Implementation Options File: etc/fast_runtime.opt
   Bitgen Options File: etc/bitgen.ut
Project Options
   Device: xc3s200tq144-4
   Netlist: TopLevel
   Implementation: XPS
   HDL: VHDL
   Sim Model: BEHAVIORAL
Reference Files
   Log Files
   Report Files
      synthesis/microblaze_0_wrapper_xst.srp
      synthesis/mb_opb_wrapper_xst.srp
      synthesis/debug_module_wrapper_xst.srp
      synthesis/ilmb_wrapper_xst.srp
      synthesis/dlmb_wrapper_xst.srp
      synthesis/dlmb_cntlr_wrapper_xst.srp
      synthesis/ilmb_cntlr_wrapper_xst.srp
      synthesis/lmb_bram_wrapper_xst.srp
      synthesis/rs232_wrapper_xst.srp
      synthesis/sram_256kx32_wrapper_xst.srp
      synthesis/sram_256kx32_util_bus_split_0_wrapper_xst.srp
      synthesis/dcm_0_wrapper_xst.srp
```

```

Device utilization summary:
---------------------------

Selected Device : 3s200tq144-4

 Number of Slices:                     987  out of   1920    51%
 Number of Slice Flip Flops:           770  out of   3840    20%
 Number of 4 input LUTs:              1528  out of   3840    39%
    Number used as logic: 1182
    Number used as Shift registers: 90
    Number used as RAMs: 256
 Number of MULT18X18s:                   3  out of     12    25%


================================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
FSL3_S_CLK                         | NONE                   | 818   |
```

```
------------------------------------------------------------

Done !
```

Fig. 9 Synthesis result

V. CONCLUSION

This paper presented an approach towards VLSI implementation of the morphological image filtering) for binary image processing. Binary image processing having several advantages, and it could be recently proposed for the JPEG2000 standard. Consequently, this has become an area of active research and several architectures have been proposed in recent years. In this paper, we provide architecture for morphological filtering for binary image processing. The architectures are representative of many design styles and range from highly parallel architectures. Here a binary image-based reconfigurable system is designed using the EDK tool. Hardware architectures of morphological binary images have been implemented as a coprocessor in an embedded system. In addition, the hardware cost of this architecture is compared for benchmark images. This type of work using EDK can be extended to other applications of embedded system. The comparison showed that our processor is more suitable for binary image processing and vision systems. In Future we are going to implement this architecture in ALTERA or VIRTEX to achieve the high quality and noise free images.

ACKNOWLEDGMENT:

We wish to acknowledge the efforts of Pantech ProEd Pvt ltd.,for guidance which helped us work hard towards producing this research work.

REFERENCES

[1] Y. Liu and C. Pomalaza-Raez, "A low-complexity algorithm for the on-chip moment computation of binary images," in *Proc. Int. Conf. Mechatron. Autom.*, 2009, pp. 1871–1876.

[2] E. C. Pedrino, O. Morandin, Jr., and V. O. Roda, "Intelligent FPGA based system for shape recognition," in *Proc. 7th Southern Conf. Programmable Logic*, 2011, pp. 197–202.

[3] M. F. Talu and I. Turkoglu, "A novel object recognition method based on improved edge tracing for binary images," in *Proc. Int. Conf. Appl. Inform. Commun. Technol.*, 2009, pp. 1–5.

[4] A. J. Lipton, H. Fujiyoshi, and R. S. Patil, "Moving target classification and tracking from real-time video," in *Proc. Workshop Appl. Comput. Vision*, 1998, pp. 8–14.

[5] J. Kim, J. Park, K. Lee *et al.*, "A portable surveillance camera architecture using one bit motion detection," *IEEE Trans. Consumer Electron.*, vol. 53, no. 4, pp. 1254–1259, Nov. 2007.

[6] D. J. Dailey, F. W. Cathey, and S. Pumrin, "An algorithm to estimate mean traffic speed using uncalibrated cameras," *IEEE Trans. Intell. Transportation Syst.*, vol. 1, no. 2, pp. 98–107, Jun. 2000.

[7] T. Ikenaga and T. Ogura, "A fully parallel 1-Mb CAM LSI for real-time pixel-parallel image processing," *IEEE J. Solid State Circuits*, vol. 35, no. 4, pp. 536–544, Apr. 2000.

[8] E. C. Pedrino, J. H. Saito, and V. O. Roda, "Architecture for binary mathematical morphology reconfigurable by genetic programming," in *Proc. 6th Southern Programmable Logic Conf.*, 2010, pp. 9398.

[9] M. R. Lyu, J. Song, and M. Cai, "A comprehensive method for multilingual video text detection, localization, and extraction," *IEEE Trans. Circuit Syst. Video Technol.*, vol. 15, no. 2, pp. 243–255, Feb. 2005.

[10] W. Miao, Q. Lin, W. Zhang *et al.*, "A programmable SIMD vision chip for real-time vision applications," *IEEE J. Solid State Circuits*, vol. 43, no. 6, pp. 1470–1479, Jun. 2008.

[11] A. Lopich and P. Dudek, "A SIMD cellular processor array vision chip with asynchronous processing capabilities," *IEEE Trans. Circuits Syst. I*, vol. 58, no. 10, pp. 2420–2431, Oct. 2011.

[12] H. Yang and A. C. Kot, "Binary image authentication with tampering localization by embedding cryptographic signature and block identifier," *IEEE Signal Process. Lett.*, vol. 13, no. 12, pp. 741–744, Dec. 2006.

[13] M. Wu and B. Liu, "Data hiding in binary image for authentication and annotation," *IEEE Trans. Multimedia*, vol. 6, no. 4, pp. 528–538, Aug.2004.

[14] H. Yang, A. C. Kot, and S. Rahardja, "Orthogonal data embedding for binary images in morphological transform domain: A high-capacity approach," *IEEE Trans. Multimedia*, vol. 10, no. 3, pp. 339–351, Apr. 2008.

[15] H. Yang and A. C. Kot, "Pattern-based data hiding for binary image authentication by connectivity-preserving," *IEEE Trans. Multimedia*, vol. 9, no. 3, pp. 475–486, Apr. 2007.

[16] K. M. Shaaban, S. A. Ali, and Y. B. Mahdy, "A chip design for binary and binary morphological operations," in *Proc. Int. Conf. Inform. Intell. Syst.*, 1999, pp. 554–559.

[17] K. Fujii, M. Nakanishi, S. Shigematsu *et al.*, "A 500-dpi cellular-logic

processing array for fingerprint-image enhancement and verification," in *Proc. IEEE Custom Integr. Circuits Conf.*, May 2002, pp. 261–264.

[18] H. J. Park, K. B. Kim, J. H. Kim *et al.*, "A novel motion detection pointing device using a binary CMOS image sensor," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2007, pp. 837–840.

[19] M. Laiho, J. Poikonen, and A. Paasio, "Space-dependent binary image processing within a 64×64 mixed-mode array processor," in *Proc. Eur. Conf. Circuit Theory Design*, 2009, pp. 189–192.

[20] E. N. Malamas, A. G. Malamos, and T. A. Varvarigou, "Fast implementation of binary morphological operations on hardware-efficient systolic architectures," *J. VLSI Signal Process.*, vol. 25, no. 1, pp. 79–93, 2000.

[21] J. Velten and A. Kummert, "Implementation of a high-performance

hardware architecture for binary morphological image processing operations," in *Proc. 47th IEEE Int. Midwest Symp. Circuits Syst.*, Jul.2004, pp. 241–244.

[22] R. Dominguez-Castro, S. Espejo, A. Rodriguez-Vazquez *et al.*, "A 0.8-μm CMOS 2-D programmable mixed-signal focal-plane array processor with on-chip binary imaging and instructions storage," *IEEE J. Solid-State Circuits*, vol. 32, no. 7, pp. 1013–1026, Jul. 1997.