Technium Vol. 2, Issue 6 pp.23-34 (2020)
ISSN: 2668-778X
www.techniumscience.com

Image Steganography Based Sobel Edge Detection Using FPGA

Dhafer J.Ayyed[1]

[1]*AL-Qalam University College, Kirkuk,Iraq*
E-Mail: eng_dhafer88@yahoo.com

Abstract
Now a day's Image processing is widely used in many applications. It is having many advantages. The heart of all these image processing applications is the edge detection. It is used in various applications as security application. As image sizes and bit depths grow larger, software has become less useful in the image processing; the unique architecture of the Field Programmable Gate Array (FPGA) has allowed the technology to be used in many such applications encompassing all aspects of image processing. This work presents hiding information using Sobel edge detection. The alteration in edges cannot be distinguished well so edges can hide more data without losing quality of an image. The algorithms are implemented using MATLAB environment as well as the VHDL (Very High Speed Hardware Description Language) synthesized by using ISE (Integrated Software Environment) version (12.1) software, and implemented on FPGA Spartan 3E starter kit board. And they are implemented at 8 bit grayscale image data of size 256×256 when they are applied the work on Xilinx Spartan 3E FPGA (XC3S1600E) device. The image processing algorithms is produced using the pixel windows (3×3 windows) to calculate their output The results obtained from the implementing of algorithms using VHDL were best objectively than the ones obtained by MATLAB where the comparison between the resultant image using MATLAB and VHDL shows that the two images have a high value of PSNR, that ranges between (72.1292 to 82.8174).

Keywords: Image Processing, Information Hiding, Cover Image, Steganography.

1. Introduction

Hiding information in various media files was a well-known operation, and was used for many purposes ranging from secret messages, to keeping owner rights in a media file he established. The media files may include text, sound, image, or video as a host for the text to be embedded [1]. Host used here is the grayscale image(test and non-test images) with each having one byte of data representing, The embedding is done by writing the text bits in the LSB of each randomly chosen record or byte of the host image. It is applied the work on Xilinx Spartan 3E FPGA (XC3S1600E) device. This is the size that can store it in the block RAM (Bram) for the Xilinx Spartan 3E FPGA (XC3S1600E) device. The effect of embedding on the host image is measured by calculating the PSNR (Peak Signal to Noise Ratio) for the image before and after embedding. Many researchers had worked in the field of embedding text bits in the LSB of a host image pixels, their work included a variety of details with a lot of means [2].

2. The LSB Embedding

The least significant bit (in other words, the 8th bit) of some or all of the bytes in an image is changed to a bit of the secret message. Digital images are mainly of two types (i) 24 bit and (ii) 8 bit. Increasing or decreasing the value by changing the LSB does not change the appearance of the image; hence the resultant stego image looks almost same as the cover image [3]. If the LSB of the pixel value of cover image C(i, j) is equal to the message bit m of secret message to be embedded, C(i, j) remains unchanged; if not, set the LSB of C(i, j) to m. The message embedding procedure is given below [6]:

S (i, j) = C (i, j) - 1, if LSB(C (i, j)) = 1 and m=0
S(i.j)=C(i,j),if LSB(C(i,j))=m (1)
S (i, j) = C (i, j) + 1, if LSB(C (i, j)) = 0 and m=1

Where LSB(C(i, j)) stands for the LSB of cover image C(i, j) and m is the next message bit to be embedded. In this work, primarily an8-bit grayscale images are used. These images are thus m-by-n matrices of integers ranging from 0 to 255, where 0 corresponds to black, 255 to white, and the values in between form a spectrum of varying shades of grey (i.e., darker shades nearer 0 and lighter shades nearer 255). Since these gray scale values form a spectrum ranging in order from dark to light, each grey value varies little from the values on either side of it. For example, the grey value 100 varies little from the grey values 99 or 101. Therefore, changing the LSB creates an imperceptible change in the image.

3. Field-Programmable Gate Array

A Field-Programmable Gate Array (FPGA) is an integrated circuit that can be electrically programmed to become almost any kind of digital circuit or system. The first FPGA was introduced in 1985 by the company Xilinx [4]. FPGAs can be used to implement any logical function that an ASIC (Application Specific Integrated Circuit) could perform. The ability to update the functionality after shipping, partial re-configuration of a portion of the design and the low costs relative to an ASIC design, offer advantages for many applications [5]. One of the most advanced FPGA families in industry and education is the FPGA series

produced by Xilinx. The designed architectures are implemented in this paper using one of the Xilinx FPGA devices, the SPARTAN-3E starter kit board (supported with XC3S1600E device). The FPGA configuration is generally specified using a Hardware Description Language (HDL) that is needed and important for describing the structure and functions of the system to be designed. There are two major hardware description languages, VHDL and Verilog. The HDL that is used in this thesis to implement the designed system is the VHDL, within the use of Xilinx ISE 12.1(Integrated Software Environment) as an Electronic Design Automation (EDA) environment.

4. Implementation Of Sobel Using FPGA

The Sobel Edge Detection algorithm algorithms are implemented as below:

1- *MATLAB Software:* In this thesis, the MATLAB environment is used for preparing image data file so that it becomes suitably handled by the FPGA, simulation performance of the program before designing the Steganography system on the FPGA device, reading image data file and determining the Stego-image. To prepare image data for storing and processing in the FPGA device, first, the test image is input with the help of MATLAB, which is by default a gray scale image. So the image is read (and resized if needed) so as to store it in the image data file (.Coe) file and this step is done by converting each pixel in the image to hexadecimal value then storing it in the (.Coe) file. So, the image data file is a text file that contains the image pixels values in the hexadecimal format to be suitable for storing in the Block RAM memory (Bram).

2- *Storage Units:* There are two types of storage elements (Memory) that may be used in this project, the Block RAM (BRAM) that stores the host image, and the Distributed RAM. BRAM can be generated in two methods: the first method is during writing the program with any synthesis based design using the appropriate "RAMB16" module from the Xilinx design library, the second method is by using the Xilinx CORE generator system. BRAM can be generated in the form of single or dual port by using the Xilinx CORE generator system which supports a variety of modules. The size of each BRAM is 18 Kbit (2Kbit for carry +16Kbit for storage) and the number of BRAMs in Spartan 3E is 36 BRAM (total size = 648 Kbit). In this research, the required BRAM memory size and number of used BRAM depend on the size of used image as in the equations below:

$$\text{required storage memory of used image}_{\text{Kbit}} = \frac{\text{image dimensions} * 8}{1024} \quad (2)$$

$$\text{Number of BRAM used} = \frac{\text{required storage memory of used image}_{\text{Kbit}}}{16\ \text{Kbit}} \quad (3)$$

The size of 256x256 with 8 bit per pixel gray scale for both test and non-test images is used due to the limited storage of the Xilinx Spartan 3E FPGA (XC3S1600E) device , and the required memory size is calculated as follows:= 256×256 ×8 /1024=512kbits and the number of BRAM used= 512 Kbit/16 Kbit=32 BRAM, each BRAM has only 16Kbit used for storing data

3- *Implementation of the Convolution on VHDL:* Convolution algorithm is a part for Sobel edge detection. The design methodology that is done in VHDL (code) is shown in the Figure (1).

.

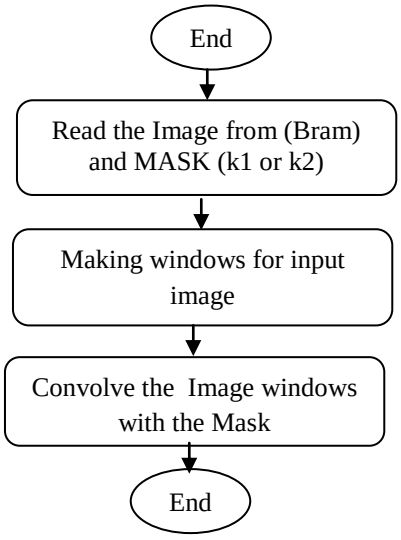


Figure (1): Block diagram for convolution on VHDL

3.1 Read the Image from (Bram) and MASK (k1 or k2):

The image pixels are stored in the Bram and arranged to be as (65536*1) when the size of used image is 256*256 pixels, every pixel needs 1 clock cycle to be stored inside the window (w) with size (3*3) , where each pixel has 8 bits gray scale image, here for every window (w) needs 9 clock cycle to be got as shown in Figure(4.4) . The masks (k1 and k2) with size (3*3) matrix is stored in variable (STD_LOGIC_VECTOR) type, where the horizontal mask (k1) and the vertical mask (k2) are initialized in the beginning of VHDL main program.

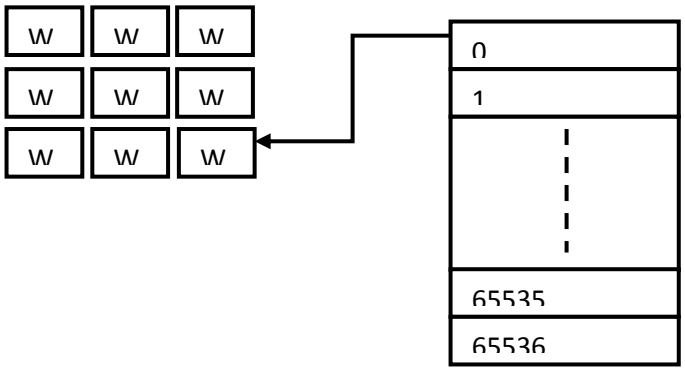


Figure (2): Arranged the pixel value image to windows

3.2 Making windows for input image:

In VHDL program there is no need to use zero matrix in order to decrease the size of used memory and this matrix will be replaced by adding the zeros directly to windows according to the location of each pixel as explained below:

- In the case of first row pixel in original image, the zeros will be added only in first row and first column of window.
- In the case of pixels that are located in first row and between first - last columns, the zeros will be added only in first row of window.
- In the case of pixels that are located in first row and last columns, the zeros will be added only in first row and last column of window.
- In the case of pixels that are located between first-last rows and first columns, the zeros will be added only in first column of window.
- In the case of pixels that are located between first-last rows and between first - last columns, the zeros will be not added in window.
- In the case of pixels that are located between first - last rows and last column, the zeros will be added only in the last column of window.
- In the case of pixels that are located in last row and first column, the zeros will be added only in last row and in first column of window.
- In the case of pixels that are located in last row and between first - last columns, the zeros will be added only in last row of window.
- In the case of pixels that are located in last row and between last column (last pixel in original image, the zeros will be added only in last row and in last column of window.

After that every window (w) is multiplied with the horizontal mask (k1) and the result is put in (Gx) variable, also in same clock this window is multiplied with the vertical mask (k2) and the result is put in (Gy) variable. The above steps were explained is same as the block diagram in the Figure (3), which shows the flow chart diagram of the hardware convolution.

Here (w0-w8) represents the windows value, (k0-k8) represents horizontal mask (k1) or vertical mask (k2), (m0-m8) represents multiplier, (a0-a10) represents adder, Dout represents (Gx) or (Gy). In VHDL code the simple (+) represents Addition process and the simple (*) represents multiplication process.

4- *Implementation of Sobel Edge Detector on VHDL:* At the start step in the design of the Sobel edge detector algorithm on VHDL is to convolve the mask with the image for the two directions x and y. At each pixel location, the results are two numbers Gx (corresponding to the result from the horizontal mask)

and Gy (corresponding to the result from the vertical mask). The graphical representation is similar to that in convolution, the image is read then stores the value of image in register (w), which represents windows value. m represents a multiplier (mx) where the window values are multiplied by the mask(k1, k2). Then the result is added by the adder (ax).This operation is for finding the output of Gx and Gy matrix.

After that the gradient magnitude Gr for the new pixel value as gotten from(Gx, and Gy) is found by applying this equation:

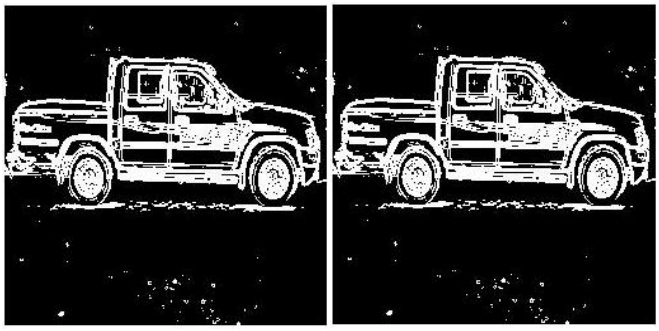
$$Gr = \sqrt{Gx^2 + Gy^2} \quad (4)$$

Here the square root for this magnitude is gotten by using Square Root Generator IP Core, and then the threshold is found by calculating the average of Gr but here in VHDL the division operation is represented by bit wise operation that is right shifting. At last, every value in the gradient magnitude matrix Gr is compared with the threshold to find the edges .

5- *The Thresholding:* The threshold value is calculated by dividing the total summation of pixels values on number of image pixels. The division operation is implemented by using shifting method; the total summations of pixels values (sum) will be shifted to right by 16 times where the total number of pixels is 65536 pixels (256*256) for the edge magnitude result. Finally, the edge magnitude pixel is compared with the threshold value if it is greater, it is identified as an edge. Else it is not identified as an edge as shown in Figure (4). Thresholding process in VHDL is just the same as thresholding process in MATLAB but there is only one difference in the value because the numbers in VHDL are always integer numbers.

(a) (b)

Figure (4): The Lena (256×256) image
(a) the image edge after Sobel edge detection on VHDL
(b) the image edge after Sobel edge detection on MATLAB



(a) (b)

Figure (4): The non-test (256×256) image1
(a) the image edge after Sobel edge detection on VHDL
(b) the image edge after Sobel edge detection on MATLAB

6- *Square Root Core:* When the square root functional configuration is selected a simplified CORDIC algorithm is used to calculate the positive square root of the input [6]. The input, X_IN, and the output, X_OUT, are always positive and are both expressed as either, unsigned fractions or unsigned integers. Number of bits at the output port of this core is equal to half of the number of bits at the input ports as shown in the Figure (5). This core needs two clock cycles to put the result of square root operation on the output port.

Technium Vol. 2, Issue 6 pp.23-34 (2020)
ISSN: 2668-778X
www.techniumscience.com

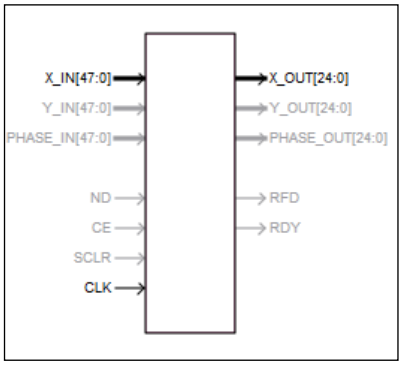


Figure (5): Square root core in VHDL

Figure (6) displays a sample of input and output of the Square root core used in edge magnitude.

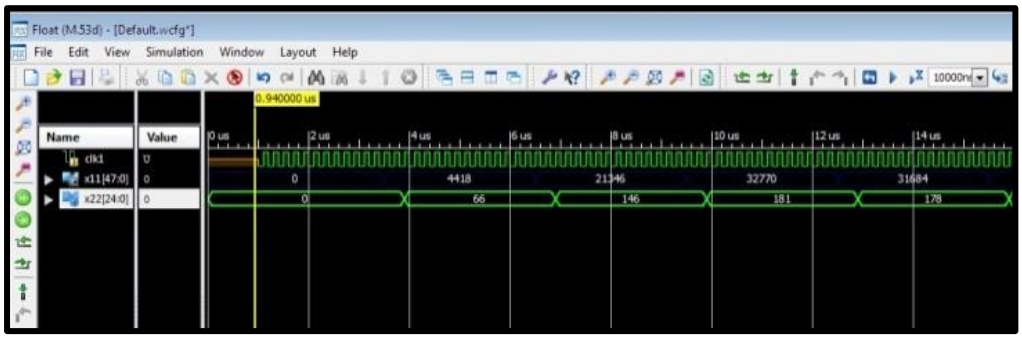


Figure (6): Timing Diagram of the Square root core used in edge Magnitude

5. Implementation of Steganography on VHDL

5.1 Implementation of embedding procedure (hiding) on VHDL

- It is a process that hides a string of characters (code) in an image. This process will be used to hide a string in Gray scale images based on the Sobel edge detection method. Each character has 8 bits, each pixel in the image edge has value '1', the corresponding pixel in original gray image will hide one bit of the character in this pixel using the LSB method. This method is implemented in the following manner:
- The secret message is stored in variable (STD_LOGIC_VECTOR) type, Also the key is input and stored in variable (STD_LOGIC_VECTOR) type , where initialized in the beginning of VHDL main program.
- As in MATLAB, The secret message bits are rearrangement in another variable. Where First bit (MSB) of the first character in the characters string is stored in the first bit (MSB) of a variable (string of bits when a variable type is *STD_LOGIC_VECTOR*) then first bit (MSB) of next character is stored in the next bit (MSB) of the variable. After storing all first bits (MSBs) for all characters in the variable in a sequential manner the second bits from MSBs of all characters are stored in next bits of the variable ;so that these processes continue until storing all bits of all characters in the variable.
- During the process of detecting edges of each pixel has a value greater than (threshold + key). The pixel location for the edges point is stored in a matrix called a matrix of pixels selected by the number of elements is equaled to the number of the secret message bit.
- After computing the edges of an image for all pixels in the image, the pixels that represent edges, are read from their locations in BRAM (original gray image) then storing single bit from the variable (string of bits) in the LSB of each of these pixels, then these pixels are written back in BRAM. This process continues until storing (hiding) all bits in of all characters in the selected pixels.
- Pixels are re-stored after the bit of the secret message in the BRAM using the same pixel location when it is brought from BRAM.

- In order to be able to extract the hidden message from the stego-image at the receiver, the massage length should be embedded in the host image. The last twelve pixels of the image were used for hiding the number of the massage characters rather than the number of their bits. This number is hidden in the LSB of each of the last twelve pixels in the host image. The maximum number allowed of hidden characters by this method is 2^12 or 4096 characters. These last twelve pixels are not included in the process of hiding message bits regardless of their being edges or not, because they are reserved for the message length hiding. Figure (8) shows a flowchart of the hardware embedding procedure.

5.2 Implementation of extracting procedure (Decoding) on VHDL:

This method is implemented in the flowing manner:

- At the start, the last twelve pixels of the image, the pixels which used for hiding the number of the massage characters in the first bits LSBs of these pixels, were read for extracting the number of characters from LSBs of these pixels , then converted this number from binary to integer. Also the key is input and put in variable (*STD_LOGIC_VECTOR*) type, where initialized in the beginning of VHDL main program, when the key value is sent with the stego-image.
- During the process of detecting edges of all pixel which have a value greater than (threshold + key), the pixels locations for the edges points are stored in a matrix called a matrix of selected pixels with the number of elements which are equaled to the number of the secret message bits.
- After extracting the image edges, the first selected pixel in BRAM (stego-image) will be read from BRAM then stores the LSB from it in MSB of a variable (string of bits with variable type *STD_LOGIC_VECTOR*) , then read next selected pixel from BRAM and stores the LSB from it in next MSB of the variable. This process continues until stores all LSBs for all selected pixels in the variable in a sequential manner.
- Finally, the extracted bits that are stored in variable are re-arranged to create the original secret message .Figure (9) shows a flow chart of the hardware Extracting procedure (Decoding).

6. Experimental Comparison of VHDL and MATLAB Algorithm's Results

Experimental results using the algorithms described in proposed work section have been applied on standard and non-standard images of size 256x256 pixels are presented in this section. Two parameters measurements are applied in the presented work that is:

- **Mean square error (MSE)** of an estimator is to quantify the difference between an estimator and the true value of the quantity being estimated [7].

$$MSE=\frac{1}{MN}\sum_{I=0}^{M-1}\sum_{J=0}^{N-1}\left(x(i,j)-y(i,j)\right)^2 \qquad (4)$$

Where:

i, j: refer to the pixels positions in the image.

M,N: refer to the number of rows and columns in the input image, respectively.

- **Peak Signal to Noise Ratio (PSNR):** The PSNR ratio is often used as a quality measurement between the original and reconstructed image. The higher the PSNR, the better is the quality of the compressed or reconstructed image [16]. The PSNR is Defined as:

$$PSNR = 10 \log \frac{(R^2)}{MSE} \qquad (5)$$

Where: R is the maximum pixel value in the input image data type.

Tables (1) and (2) show the secret message length effects on the (PSNR) and (MSE) values for different test and non-test images

Table (1): Shows the PSNR and MSE values vs. messages length for different test images

Name of Used Image	Number of Character in Text	MSE	PSNR in dB
Lena.bmp	100	0.0058	70.5092
	200	0.0121	67.3193
	300	0.0177	65.6473
	500	0.0303	63.3136
	600	0.0361	62.5555
	800	0.0486	61.2686
	1000	0.0613	60.2555
Barbara.png	100	0.0062	70.2318
	200	0.0121	67.2865
	300	0.0182	65.5256
	500	0.0310	63.2185
	600	0.0365	62.5098
	800	0.0489	61.2360
	1000	0.0605	60.3121
boat.png	100	0.0063	70.1572
	200	0.0120	67.3524
	300	0.0177	65.6473
	500	0.0299	63.3730
	600	0.0364	62.5225
	800	0.0490	61.2251
	1000	0.0614	60.2523

Table (2): Shows the PSNR and MSE values vs. messages length for different non- test images

Name of Used Image	Number of Character in Text	MSE	PSNR in dB
non-test1.png	100	0.0062	70.2103
	200	0.0127	67.0892

	300	**0.0189**	**65.3684**
	500	**0.0315**	**63.1527**
	600	**0.0368**	**62.4772**
	800	**0.0495**	**61.1808**
	1000	**0.0607**	**60.2968**
non-test2.png	**100**	**0.0062**	**70.2211**
	200	**0.0121**	**67.2974**
	300	**0.0183**	**65.5074**
	500	**0.0310**	**63.2164**
	600	**0.0365**	**62.5025**
	800	**0.0484**	**61.2795**
	1000	**0.0601**	**60.3428**

7. CONCLUSION AND DISCUSS RESULT

In this paper, a hardware and software implementation of image processing system for Steganography algorithm is proposed; using MATLAB language for software and VHDL hardware description language for hardware design. The functionality of the algorithms was tested and verified successfully on a Xilinx Spartan 3E (XC3S1600E) FPGA is simulated by ISE12.1. The conclusions are summarized by the following points:

- The increase of message length affects the image quality. It also indicates that the longer the message the higher MSE value gets and the longer messages the lower PSNR value becomes.
- When Steganography process is applied, the results of PSNR and MSE in both VHDL and MATLAB are similar because the selected pixels in VHDL and MATLAB to hide the secret message bits have values more than threshold + key, so the difference in the in the gradient magnitude between VHDL and MATLAB does not affect on the pixels that have these values (threshold + key).
- For high speed system FPGA used windowing algorithms as well as series of image processing techniques which can be synthesized for high-speed applications.
- Spartan 3E (XC3S1600E) FPGA are easy to use and flexible when working for different sizes of images.

REFERNCES

[1] Sagar S.Pawar and. Vinit Kakde," Review On Steganography For Hiding Data",International Journal of Computer Science and Mobile Computing, Vol. 3, pp(225 – 229), April 2014.
[2] Afroja Akter, Nur-E-Tajnina, and Muhammad Ahsan Ullah," Digital Image Watermarking Based on DWT-DCT: Evaluate for a New Embedding Algorithm", 3rd International Conference On Informatics, Electronics & Vision, May 2014.
[3] Nitin Jain, Sachin Meshram and Shikha Dubey ," Image Steganography Using LSB and Edge Detection Technique ", International Journal of Soft Computing and Engineering (IJSCE), Vol. 2, pp(218-222) July 2012.
[4] Christian Baumann, "Field Programmable Gate Array(FPGA)", Summary paper for the seminar "Embedded System Architecture", University of Innsbruck January 13, 2010.
[5] C. Bobda, (2007), "Introduction to Reconfigurable Computing Architectures, Algorithms, and Applications", Springer, ISBN: 978-1-4020-6100-4.
[6] Xilinx, " Spartan 3E Starter Kit Exercise 1" Xilinx, Inc.
[7] Dhanabal R,Bharathi V , S.Kartika, " Digital Image Processing Using Sobel Edge Detection Algorithm InFPGA Journal of Theoretical and Applied Information Technology, Volume 58, PP.130 -139, 2013.

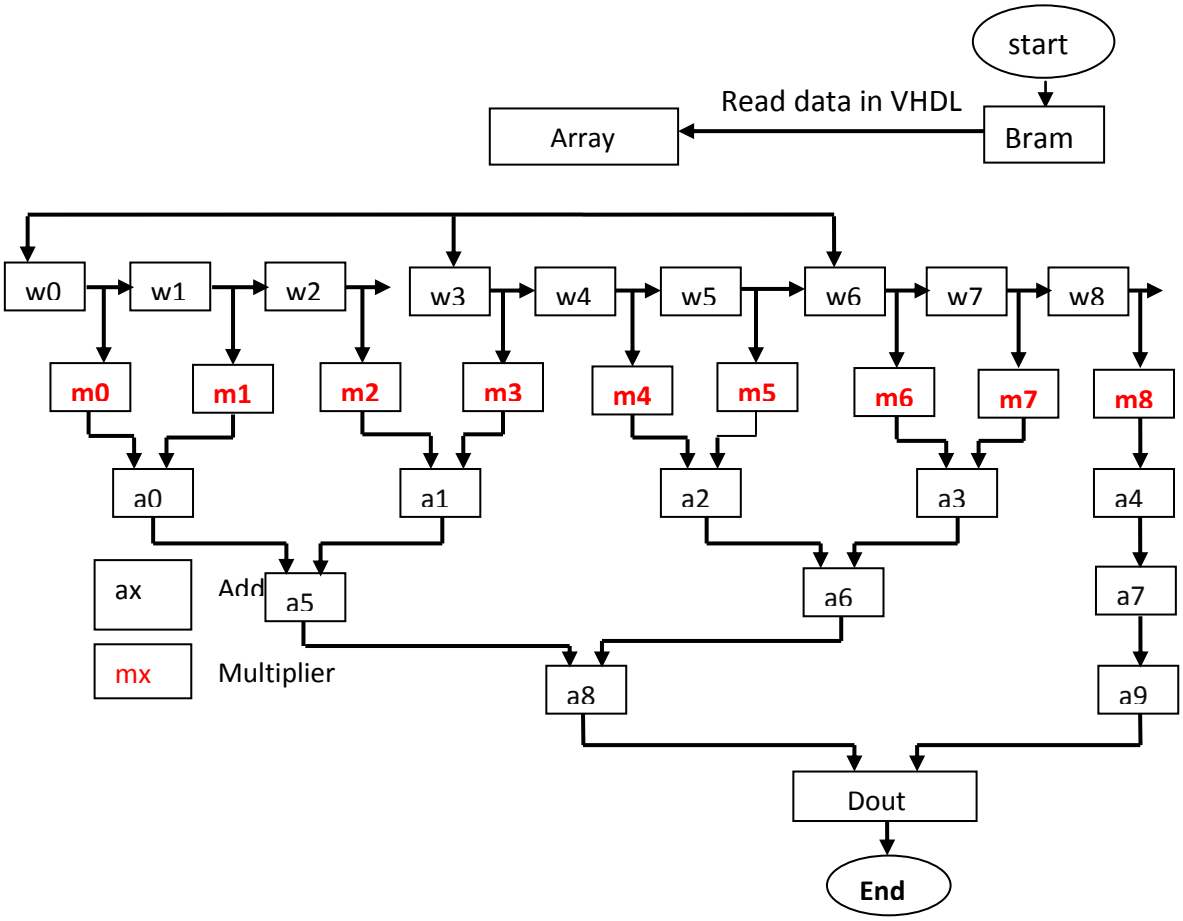


Figure (3): Flow chart diagram of the Convolution algorithm on VHDL

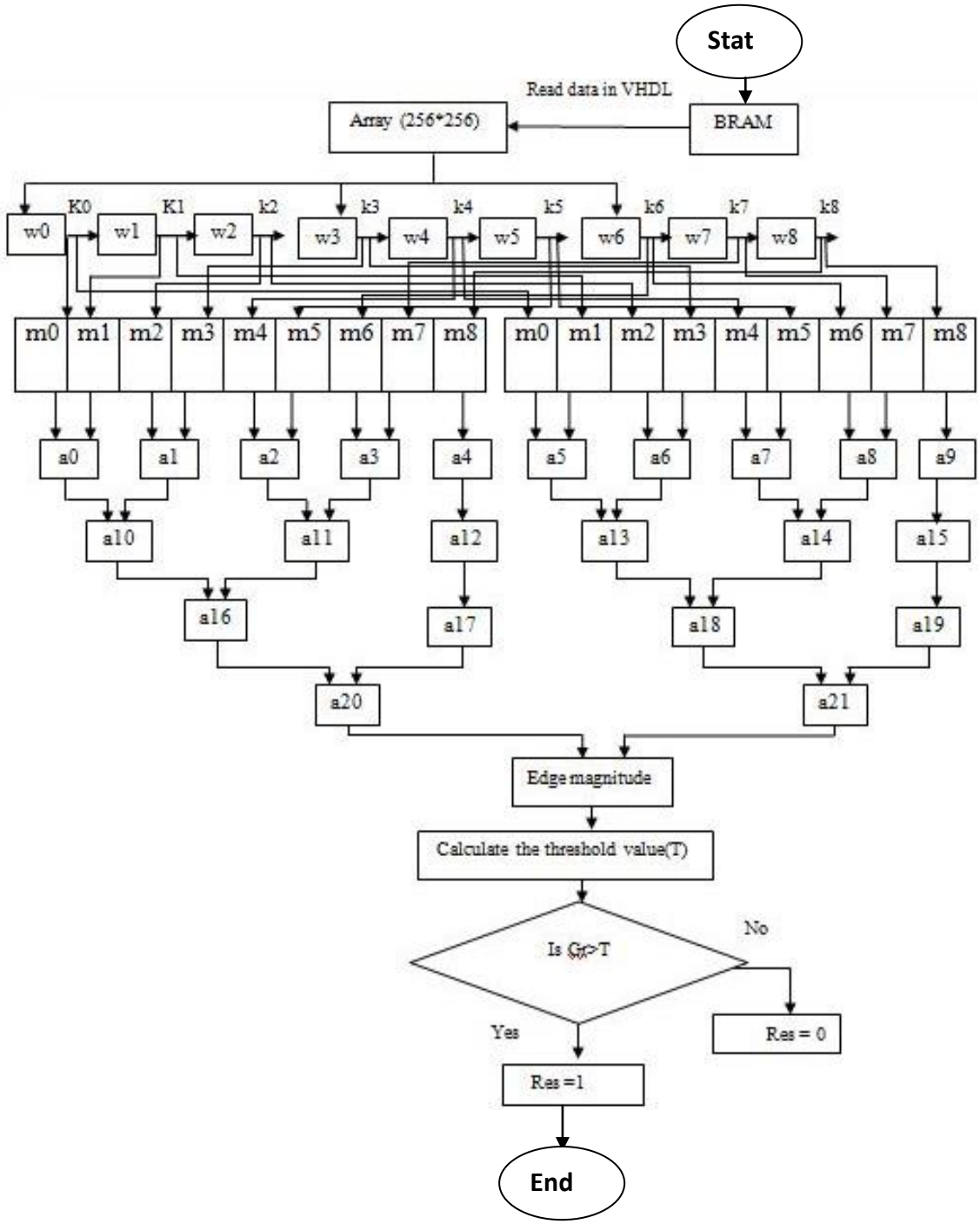


Figure (7): Flowchart of the Sobel edge detector on VHDL

Technium Vol. 2, Issue 6 pp.23-34 (2020)
ISSN: 2668-778X
www.techniumscience.com

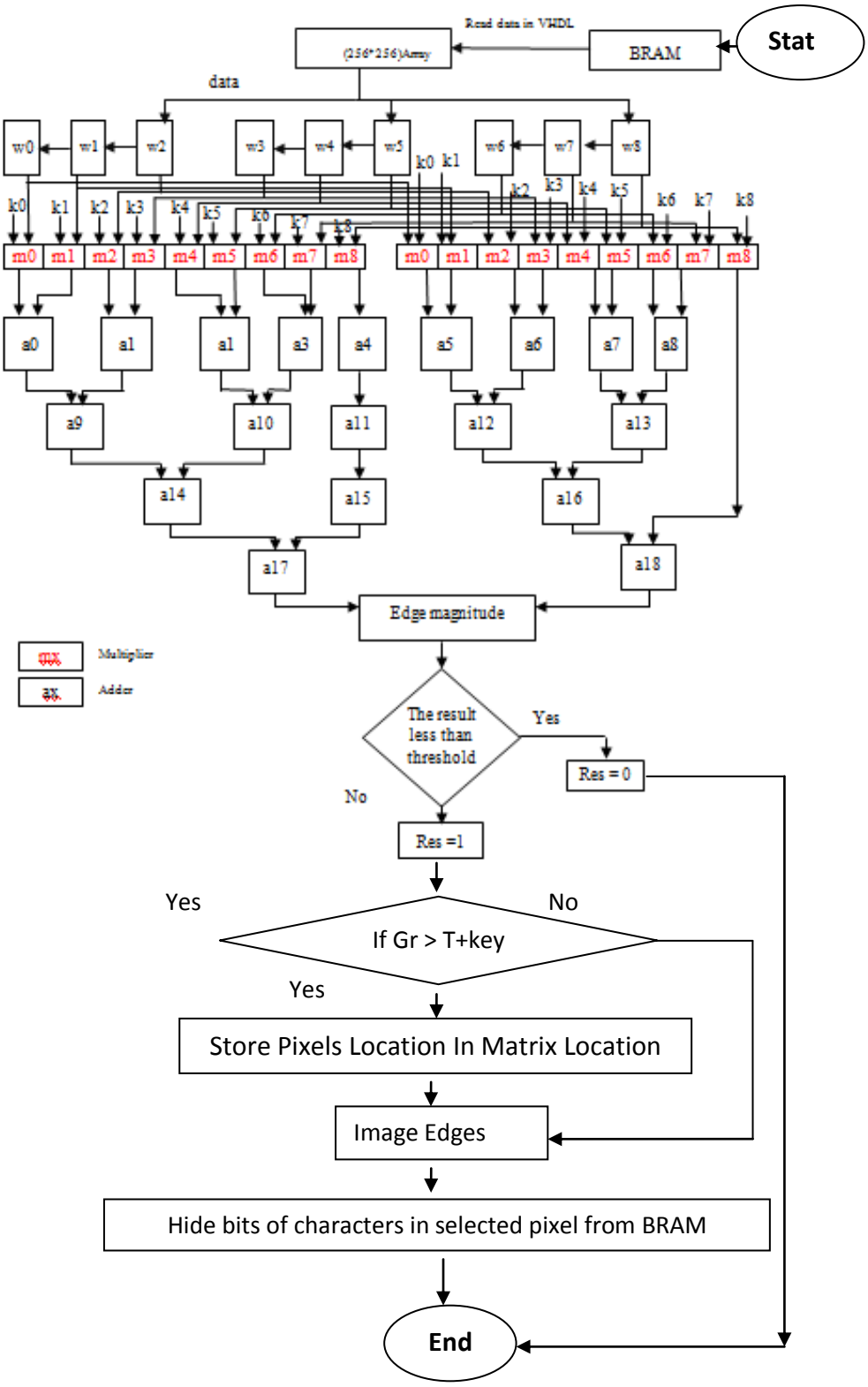


Figure (8): Flow chart of the embedding procedure (hiding) on VHDL

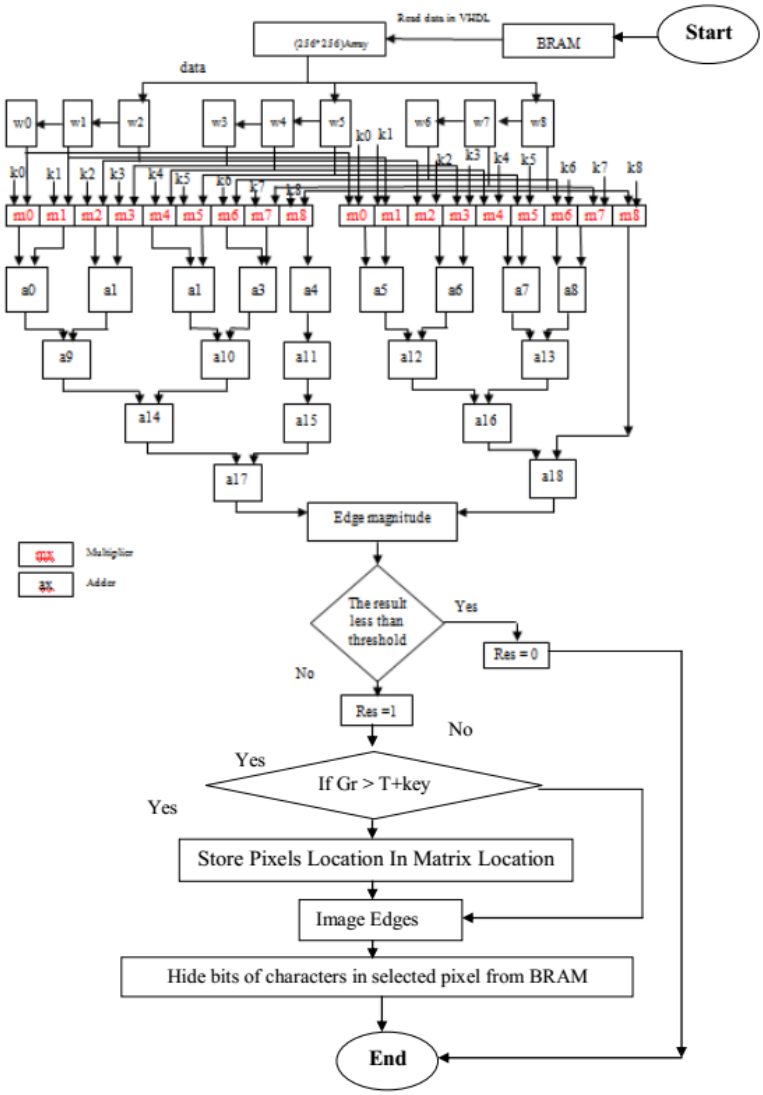


Figure (9): Flowchart of the Extracting procedure (Decoding)on VHDL