International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 06 | June 2020 www.irjet.net p-ISSN: 2395-0072

VLSI Architecture Implementation for Reversible Watermarking

Shivam Sinha[1], Vishal Gupta[2], Sanjana Gosavi[3], Uttara Bhatt[4]

[1,2,3]*Final-year B.E.-EXTC, Thadomal Shahani Engineering College, Mumbai, India*
[4]*Assistant Professor, Department of EXTC, Thadomal Shahani Engineering College, Mumbai, India*

Abstract - *The paper proposes VLSI architecture of low computational reversible watermarking technique. The architecture uses reversible contrast mapping (RCM)- an algorithm used to embed and extract the watermark bits from the pixels by using forward and reverse transform equations respectively. RCM was chosen because it recovers all LSBs bits even if it has been lost during digital data embedding. The system is designed for 4x4 block of compressed grey scale image. It consists of two major modules, encoder and decoder designed separately using Xilinx ISE design suite 14.7 Spartan 3e FPGA family. Encoder uses 107 4-input LUTs and 67 slices. On the contrary, decoder uses 370 4-input LUTs and 199 slices.*

Key Words: **Reversible watermarking, VLSI architecture, Reversible contrast mapping, VHDL, Encoder, Decoder.**

1. INTRODUCTION

Reversible watermarking techniques are also called as invertible or lossless and were created to be applied mainly in scenarios where the authenticity of a digital image has to be granted and the original content is peremptorily needed at the decoding side. It is important to point out that, initially, a high perceptual quality of the watermarked image was not a need due to the fact that the original one was recoverable and simple problems of overflow and underflow caused due to the watermarking process were not taken into account too. Successively, this aspect has been considered as basic to permit to the end user to operate on the watermarked image and to possibly decide to resort to the uncorrupted version in a second time if needed [1].

A large number of concepts such as data compression, histogram modification [7], reversible contrast mapping (RCM) [3], difference expansion (DE) [6], prediction error judgement etc. along with their modification have been developed to design reversible watermarking. Among these, RCM has gained more popularity because of its low computational feature. RCM was initially proposed by Coltuc [3]. It is based on simple integer transforms applied on a pair of pixels. In this technique, the least significant bits of the transformed pixels are used for data embedding. Several algorithms of RW along with their performance results are reported through software simulations. At the same time, hardware implementation of several conventional watermarking methods on image and video are also reported in the literature [2].

One essential requirement in RW is its real-time implementation that can be met through hardware realization. Although VLSI architecture of several spatial and transform domain conventional watermarking methods are reported, to the best of our knowledge, only very few attempts are made for VLSI architecture design of RW and its prototype design on FPGA platform. To fill up this gap, we have developed VLSI architectures for RW on digital images based on RCM algorithm. As an initial attempt, FPGA architecture on RCM-RW algorithm reported in [4] is developed and reported in [5]. However, there remains scope for further modification on [4] to improve rate-distortion performance. This demands modification in RCM-RW followed by development of hardware design.

2. REVERSIBLE CONTRAST MAPPING

Let (x, y) be the pair of pixels of a compressed grey scale image. The compressed image is a matrix of 4x4 i.e. the image is converted to 4x4 blocks and each block is assigned a pixel value. The pixels can be paired horizontally, vertically or any desired way. After the pairing is done, these pairs are processed or transformed by using reversible contrast mapping. RCM basically consist of two transform equations, namely forward and reverse. These transform equations are applied to the pair of pixels, each pixel intensity values are bounded between [0, L], where L=255 for a grey scale image. The equations are as follows:

Forward transform: $x' = 2x - y$ $y' = 2y - x$ (1)

Reverse transform $x = (2x' + y')/3$ $y = (2y' + x')/3$ (2)

To prevent the overflow and underflow problems a domain is defined, such that the transformed pairs are restricted within a sub domain of Dc [0, L] x [0, L], defined as: $0 <= 2x - y <= 255$ and $0 <= 2y - x <= 255$. This is done to make sure that, after transformation the pixel values should be between 0 and 255. We are giving a grey scale image as input, and we expect to get the output as grey scale image as well, with watermarks. Before the watermarked bits from the pixels

are extracted in the decoder side, the LSBs of these pixels are cleared and the equation (2) is modified as follows:

$$x=2x^*+y^*/3 \qquad y=2y^*+x^*/3 \qquad (3)$$

where $x^*=x'$ and 00000001 $y^*=y'$ and 00000001 (4)

Furthermore, the algorithm is basically used so that, whatever data is embedded in the image it can easily be extracted at the decoder side to maintain the robustness of the image. Since, we are pairing the pixel values in a random way, each pair is an input to the system. If we observe the pairs, there can be four possibilities i.e. pixel pairs are both even, one even one odd and both odd. Based on these variations RCM works in a following way.

- LSB of x is '0', LSB of y is '0': LSB of both x' and y' is '0' (as difference of even numbers, since x and y are even), therefore $x^*=x'$ and $y^*=y'$ => x and y are exactly recovered;
- LSB of x is '0', LSB of y is '1': LSB of x' is '1' and LSB of y' is '0' => $x^*=x'-1$ and $y^*=y'$. Inserting these in equation (3) we get $x= (2(2x-y-1) +2y-x)/3=(x-2/3)$ and $y= (2(2y-x)+2x-y-1)/3=(y-1/3)$ therefore, x and y are easily recovered.
- LSB of x is '1', LSB of y is '0': LSB of x' is '0' and LSB of y' is '1' => $x^*=x'$ and $y^*=y'-1$. Inserting these in equation (3) we get $x= (2(2x-y) +2y-x-1)/3=(x-1/3)$ and $y= (2(2y-x-1)+2x-y)/3=(y-2/3)$ therefore, x and y are easily recovered.
- LSB of x is '1', LSB of y is '1': LSB of x' is '1' and LSB of y' is '1' => $x^*=x'-1$ and $y^*=y'-1$. Inserting these in equation (3) we get $x= (2(2x-y-1) +2y-x-1)/3=(x-1)$ and $y= (2(2y-x-1)+2x-y-1)/3=(y-1)$ therefore, neither x nor y are recovered.

From the above analysis, it is clear that for the first three cases the pixels are easily recovered and for odd pairs case, pixels cannot be recovered. Hence, the above equations (1) and (2) are only used for the first three cases defined above.

3. WATERMARK EMBEDDING

To realize RCM-RW, we consider an 8-bit input. The goal is to develop hardware architecture of the given input. Watermark bit is inserted into the LSBs of the transformed pixel pair values (x', y'). The LSBs of the first transformed pixel value (x') is used to indicate whether a pair is transformed or not, and the second pixel value (y') is used for watermark embedding. Data flow diagram of watermark embedding is summarized in Fig-1. Next, for each pair of pixel values, two conditions (x, y) ∈ DC are checked. If the pixel pair (x, y) do not belongs to Dc, Step 3 in Fig-1 is followed, else whether the pixel pairs are odd numbers or not is verified. If both of them are odd in nature, Step 2 is followed; else Step 1 is performed for watermark embedding. The whole process is repeated until all the pixel pairs are covered. Finally, the watermarked data are collected through multiplexer.

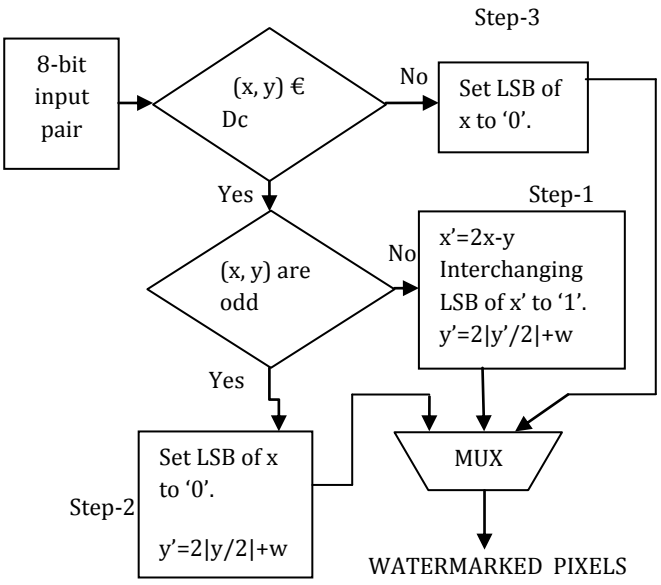

Fig -1: Encoder Flowchart

STEP-1: This step is designed for those pixel pairs that belong to domain that has been previously explained i.e. Dc and are not odd values. This step is used to demonstrate the RCM algorithm forward transform equations. For example, x'=2x-y (where x, y are the pair of input pixels), here we see that pixel (x) is first multiplied by 2, which can be demonstrated in VHDL by designing an 8-bit left shift register by using shift operations. Further, the multiplied pixel is subtracted with y pixel value. Here, we use a 9-bit subtractor because the multiplied value coming from an 8-bit left shift register is not necessarily an 8-bit value. After subtraction, output is first right shifted to discard the LSB and then left shifted, and finally 1 is padded in the LSB. These two blocks explain that after the forward transform, pixels are processed in such a way that will help the decoder on other side to differentiate between pixel values. Padding 1 in last block indicates that the pixel value (x) is transformed using RCM. Similarly, same explanation is valid for pixel value y(equation is y'=2y-x) , only difference being-last block i.e. instead of padding 1, we insert digital watermark in the LSB of y'.

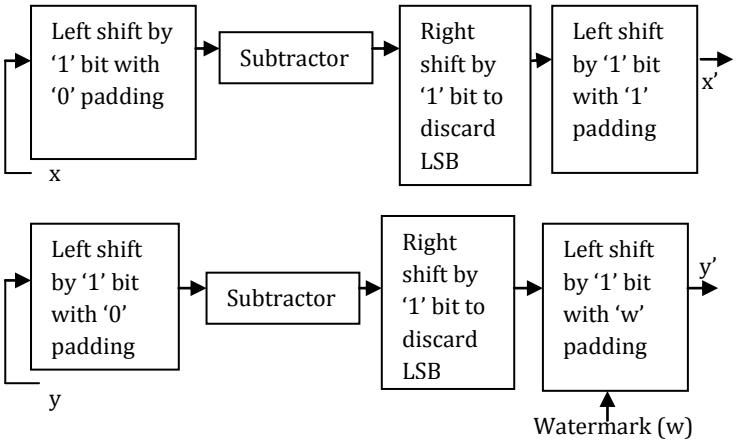

Fig -2: Data path for Step-1.

STEP-2: This step is designed for those pair of pixels that belong to Dc domain and are odd valued. Considering pixel (x) first, since it is an odd valued pixel LSB should be 1. The main working of this step is to remove 1 from the LSB of pixel (x) because RCM equations do not give the original pixel back for odd valued pair of pixels. If output of an encoder has 1 in its LSB (for x pixel), then according to step-1 it is denoted as transformed pixel. Therefore, to clear this contradiction the odd valued pixel is first right shifted and then left shifted using logical shift operators in VHDL. Similarly for y pixel, first it is right shifted and the left shifted to insert the digital watermark bit in its LSB.

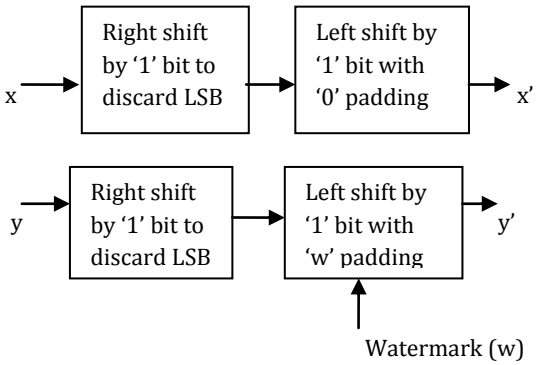

Fig -3: Data path for Step-2.

STEP-3: This step is designed for those pixels that do not belong to domain Dc. Since it only processes those pixels that do not belong to domain Dc, it does not care about the nature of pixel value. Pixel (x) from a pair is first right shifted to discard the LSB and this discarded LSB is also taken as the output of this step along with pair of pixel and then left shifted to get the output. For pixel y, it is passed through the encoder as it is with no watermarks because, y pixel does not belong to Dc domain.

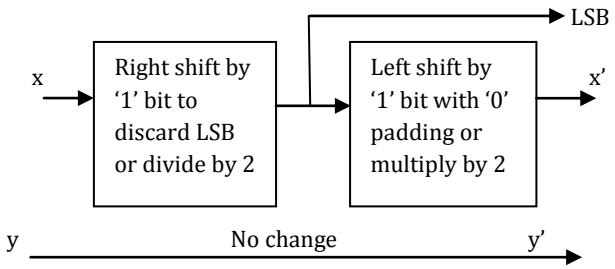

Fig -4: Data path for Step-3.

4. WATERMARK EXTRACTION

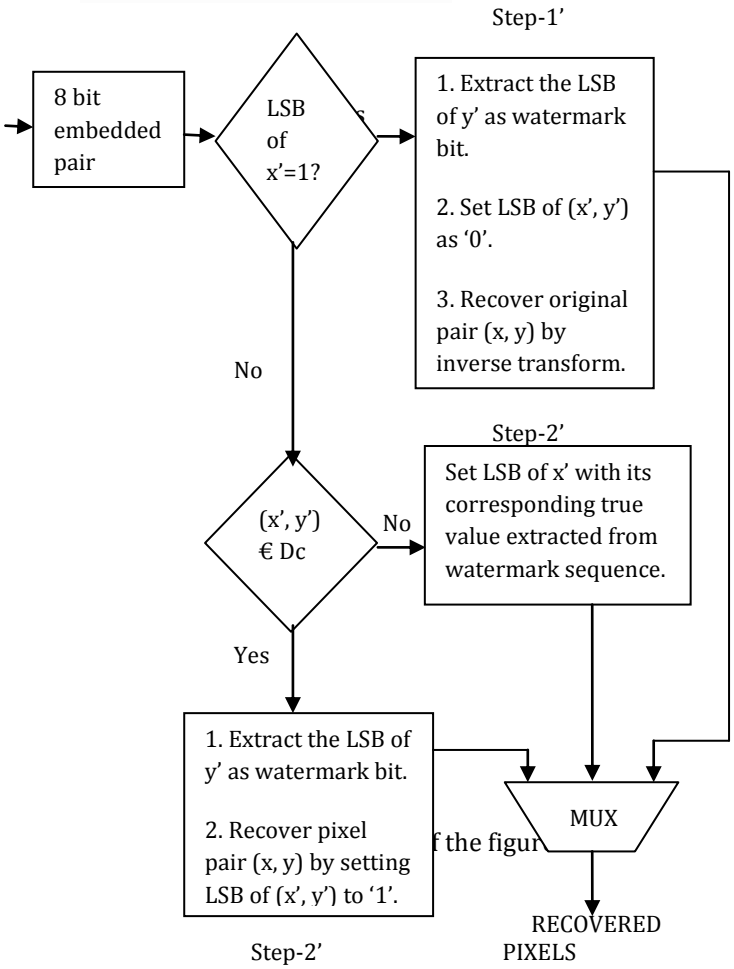

Fig -5: Decoder Flowchart

The input to the watermark extraction block is an 8- bit that consists of Step-1', Step-2', Step-3' and condition check block. If the LSB of x' is 1, then Step-1' is performed regardless of inclusion of the pair in Dc domain. If x' is not equal to 1, then the condition of inclusion of pairs in Dc domain is checked, if the pairs belong to Dc domain than step-2' is performed else step-3'. All of these different conditions checking and individual data path multiplexing for watermark extraction

unit are shown in Fig-5. The hardware realization of individual step is given below.

Step-1': The step is designed for those output pairs of encoder whose x' valued pixel has 1 in its LSB; whether the pair lies in Dc domain or not it doesn't matter. In this step, initially the LSBs of pixel (x') and pixel (y') are removed so that before applying the reverse transformation these pixels are converted to original transformed values. This is done by first right shifting the pixel value to discard the LSB (which is 1 in pixel (x') and watermark bit in pixel (y')) and the left shifting the pixel value. The right shifting and left shifting is done by shift operators present in VHDL. After this, reverse transform equation is applied to these pixels to get original pixel pair back. Let's say the equation is for example, x=2x'+y'/3. To process the given equation initially, a 9-bit shift register is designed to perform multiplication by 2 and the result is given to a 9-bit adder, where the addition with y' pixel take place and finally, result of the adder is given divider, where division by 3 occurs. Divider is designed such that it can take 12-bit input and give 8-bit final output so that we get the grey scale pixel back. Similarly, for equation y=2y'+x'/3 same modules are designed and working is also the same. The only difference being, along with original pixel we also get back the embedded watermark bit.

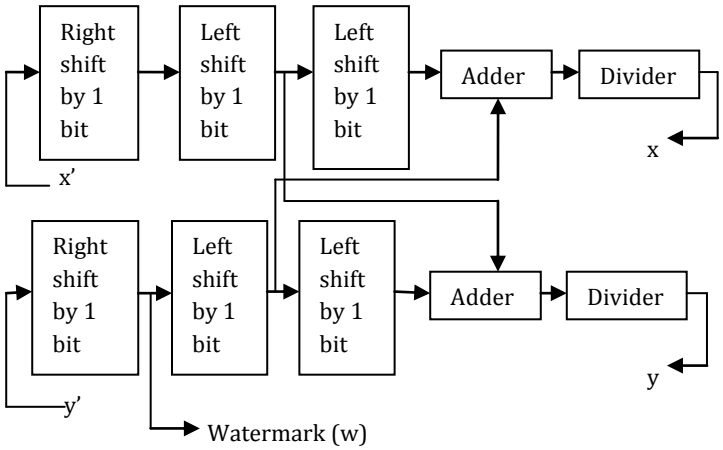

Fig -6: Data path for step-1 of decoder.

Step-2': This step is designed for those pixel pairs, which belongs to Dc domain and LSB of pixel (x') is not 1. Since LSB of pixel (x') is not one, no reverse transform equations are applied to these pairs. To get the original pixel pair back, first an 8-bit pixel is right shifted to remove the LSB (for pixel (y') watermark bit is removed) and then left shifted with 1 padding. Padding 1 in LSB is done to get the odd valued pair back since, we saw in encoder, for odd valued pair no transformation was done. Hence, these pixel are obtained back without applying any reverse transformation.

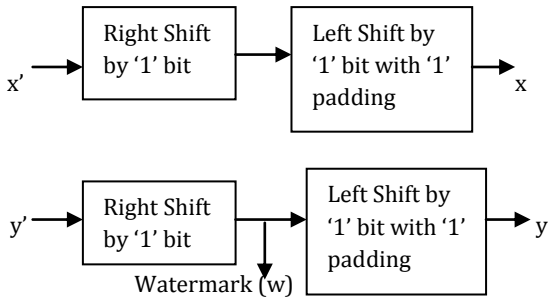

Fig -7: Data path for step-2 of decoder.

Step-3': This step is designed for those output pairs of encoder which do not belong to Dc and LSB of pixel (x') is not 1. Input to this step is the resultant output pair of encoder step-3. In this step, pixel (x') is first right shifted to remove the LSB and then left shifted with padding it with LSB out that we got from encoder step-3 output. Pixel (y') is passed as it is through the decoder.

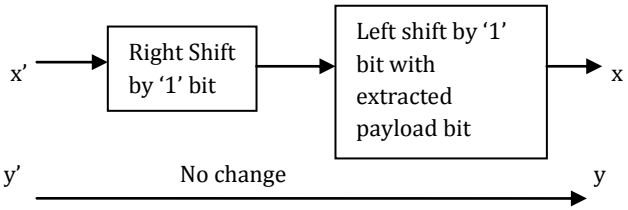

Fig -8: Data path for step-3 of decoder.

Outputs of all these steps are given to the multiplexer input to choose single input at a time. The selection is based on two conditions: input pixel pair belongs to Dc domain or not and pixel pairs are odd or not. Two 4:1 multiplexers are designed separately with same select lines, one for x' pixel and one for y' pixel. Finally, depending on the select lines we get the final output of decoder i.e. original pixel pair along with watermark bits.

6. RESULTS

6.1 Encoder

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	67	960	6%
Number of Slice Flip Flops	90	1920	4%
Number of 4 input LUTs	107	1920	5%
Number of bonded IOBs	40	83	48%
Number of GCLKs	1	24	4%

Fig -9: Device Utilization Summary for encoder.

Above figure summarizes the hardware realization of the design that was developed. RTL schematic of the encoder was developed by coding in VHDL. All the steps that have been explained in detail in the above sections are designed using Xilinx ISE design suite and the simulations are also done in the same software by creating a VHDL test-bench code for the whole design. The simulation result describes the final output of the encoder when they are provided with 8 pairs(x, y) of input pixels simultaneously. VHDL test-bench code is created to give input to the encoder and output result is presented in the form of waveforms as shown below.

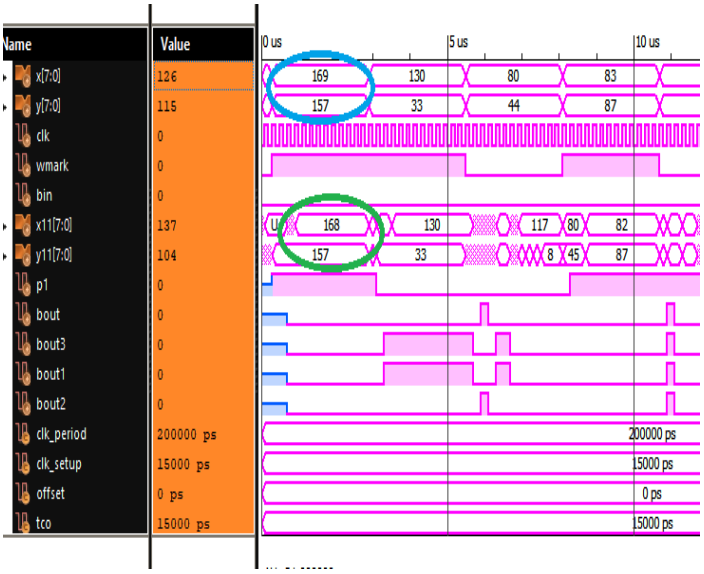

Fig -10: Simulation results of encoder.

Given waveform summarizes the working of encoder. Blue circle indicates input pixel pair (169,157) given to the encoder along with watermark bit (in this case it is 1, designated by w-mark in the waveform). Green circle indicates output of encoder i.e. the final watermarked pixel pair. Same explanation can be given to various input pairs given to the encoder.

6.2 Decoder

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	199	960	20%
Number of Slice Flip Flops	188	1920	9%
Number of 4 input LUTs	370	1920	19%
Number of bonded IOBs	72	83	86%
Number of GCLKs	1	24	4%

Fig -11: Device Utilization Summary for decoder

Above figure gives summarizes the hardware realization of the design that was developed. RTL schematic of the decoder was developed by coding in VHDL. All the steps that have been explained in detail in the above sections are designed using Xilinx ISE design suite and the simulations are also done in the same software by creating a VHDL test-bench code for the whole design. The simulation result describes the final output of the decoder when they are provided with 8 pairs (x', y') of input pixels simultaneously. VHDL test-bench code is created to give input to the decoder and output result is presented in the form of waveforms as shown below.

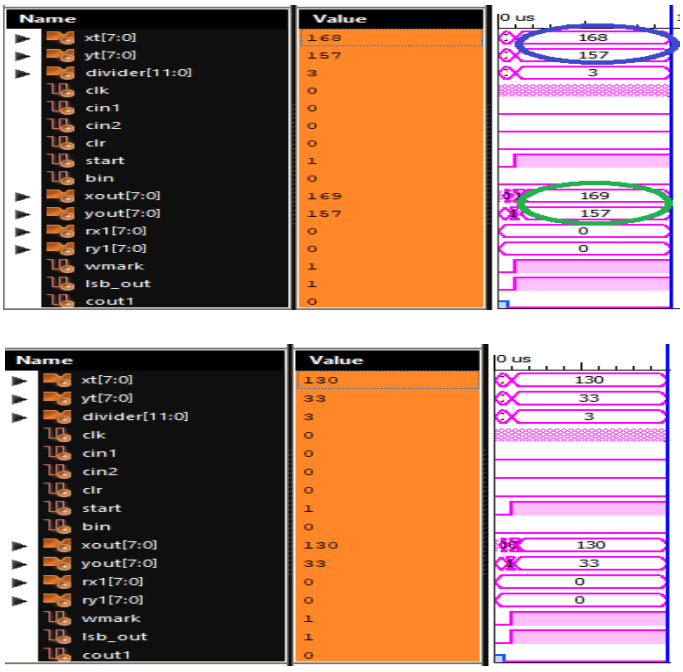

Fig -12: Simulation result of decoder.

Given waveform summarizes working of decoder. Blue circle indicates input pixel pair given to the decoder (168,157) and green circle indicates output pixel pair of the decoder along with watermark bit (in this case it is 1, designated by w-mark in the waveform). Therefore, we get the original pixel pair back from the decoder along with extracted watermark bit.

7. CONCLUSION

This paper highlights low computational reversible digital watermarking technique and uses reversible contrast mapping as an algorithm to design FPGA based architecture for watermark embedding and extraction. Encoder and decoder are designed separately using VHDL and simulation results are also obtained separately. From the simulation results, it is clear that RCM maintains the robustness of image pixel pair because, after the consecutive process of data embedding and extracting, we were able to get the original pixel pair back with very less distortion.

ACKNOWLEDGEMENT

We would like to thank our project guide Ms. Uttara bhatt for helping us to understand the topic theoretically and practically and guiding us throughout the project.

REFERENCES

[1] Roberto Caldelli, Francesco Filippini & Rudy Becarelli, "Reversible Watermarking Techniques: An Overview and a Classification", *EURASIP Journal on Information Security* volume 2010

[2] Hirak Kumar Maity, Santi P. Maity, "FPGA Implementation for Modified RCM-RW on Digital Images"

[3] Dinu Coltuc and Alain rfremeaub, "Simple Reversible Watermarking Schemes"

[4] D. Coltuc and J.-M. Chassery, "Very fast watermarking by reversible contrast mapping", IEEE Signal Process. Lett. 14 (2007) 255–258

[5] H. K. Maity and S. P. Maity, "FPGA implementation of reversible watermarking in digital images using reversible contrast mapping", J. Syst. Softw. 96 (2014) 93–104.

[6] Sudip Ghosh, Nachiketa Das, Subhajit Das, Santi P, Maity, Hafizur Rahaman, "FPGA and SoC based VLSI architecture of reversible watermarking using rhombus interpolation by difference expansion", 2014 Annual IEEE India Conference (INDICON)

[7] Sambaran Hazra, Sudip Ghosh, Sayandip De, Hafizur Rahaman, "FPGA implementation of semi-fragile reversible watermarking by histogram bin shifting in real time", Received: 14 October 2016 / Accepted: 3 February 2017