# An intelligent lossless data compressor implementation using reconfigurable hardware

**4 authors:**

Tribeni Prasad Banerjee
Dr. B.C. Roy Engineering College,
**20** PUBLICATIONS **281** CITATIONS
SEE PROFILE

Amit Konar
Jadavpur University
**611** PUBLICATIONS **9,834** CITATIONS
SEE PROFILE

s. Das
Impulse Dynamics Germany GmbH
**258** PUBLICATIONS **2,190** CITATIONS
SEE PROFILE

Ajith Abraham
FLAME University
**1,791** PUBLICATIONS **41,557** CITATIONS
SEE PROFILE

*Journal of Information Hiding and Multimedia Signal Processing* ©2011 ISSN 2073-4212
**Ubiquitous International** Volume **2**, Number **1**, January **2011**

# An Intelligent Lossless Data Compressor Implementation using Reconfigurable Hardware

Tribeni Prasad Banerjee[1], Amit Konar[2], Swagatam Das[2], and Ajith Abraham[3]
[1]Embedded System Laboratory, Central Mechanical Engineering Research Institute, Durgapur-713209, India. Email: t_p_banerjee@yahoo.com
[2]Electronics and Telecommunication Engineering Department Jadavpur University, Jadavpur-700035, India, Email: swagatamdas19@yahoo.co.in,konaramit@yahoo.co.in
[3]Machine Intelligence Research Labs (MIR Labs), Scientific Network for Innovation and Research Excellence, USA, Email: ajith.abraham@ieee.org

Received April 2010; revised July 2010

Abstract. *Reconfigurable computing is emerging as the new area for satisfying the simultaneous demand for application performance and flexibility. The ability to customize the architecture to match the computation and the data flow of the application has demonstrated significant performance benefits compared to general purpose architectures. In signal processing, multimedia, high speed communication are the major application domains that have significant heterogeneity in their computation and communication structure with various advantages. The reconfigurability of the hardware permits adaptation of the hardware for specific computations in each application to achieve higher performance compared to software. Complex functions can be mapped onto the architecture achieving higher silicon utilization and reducing the instruction fetch and execute bottleneck. In this paper we proposed and implemented a high speed CODEC (for Lossless Compression) which compress the real time image for high speed communication..*
**Keywords:** reconfigurable hardware; CODEC; lossless compressor; FPGA, Huffman coding, CAM, Multimedia Image Compressor.

## 1. Introduction.

The most obvious benefit of data compression is reduction in the volume of data which must be stored .This is important where the storage media itself is costly (such as memory) or the other parameters, such as power consumption, weight or physical volume, are critical to product feasibility. Using the data compression reduce the total storage requirement, thus effecting the cost saving. The use of lossless data compressor can bring about a number of increasingly important benefits to an electronic system. With the audio and video compression systems (such as JPEG and MPEG) which are lossy and hence only recreate an approximation of the original data. The push to roll out high definition video enabled video and imaging equipment is creating numerous challenges for video system architects. The increased image resolution brings with it higher performance requirements for basic video data path processing and next-generation compression standards, outstripping that which standalone digital signal processors (DSPs) can provide. In addition, the system specifications require designers to support a range of standard and custom video interfaces and peripherals usually not supported by off-the-shelf DSPs. While it is possible to go the route of application specific integrated circuits (ASICs) or use application specific standard products (ASSPs), these can be difficult and expensive alternatives that might require a compromised feature set. Furthermore, these choices can hasten a short product life cycle and force yet another system redesign to meet varied and quickly changing market requirements. Field programmable gate arrays (FPGAs) are an option that can bridge the flexibility gap in these types of designs. Additionally, with the increasing number of embedded hard multipliers and high memory bandwidth, the latest generation of FPGAs can enable customized designs for video systems while offering a manifold performance improvement over the fastest available stand-alone DSPs. Designers now have the ability with state-of-the-art FPGA co-processor design flows to implement high-performance DSP video and image processing applications. The major advantages to design a compressor into a FPGA, is first the device can be customized to suit a specific application through post-fabrication, user defined programming , second the system logic functionality and system response and execution of a particular job can be made fast /slow depending upon the requirement. In our proposed system needs that facility of reconfigurable computing that can vary automatically during the execution of the system in real time processing. The impact of reconfigurable hardware on DSP more thorough discussions of FPGAs and reconfigurable computing can be found in [1V3] and [4].

2. **The Requisite behind the Compression Technique.** The following statement (or something similar) has seen made many times over the 20-year history of image and video compression: 'Video compression will become redundant very soon, once transmission and storage capacities have increased to a sufficient level to cope with uncompressed video.' It is truth that both storage and transmission capacities continue to increase. There is a clear gap between the high bit rates demanded by uncompressed video and the available capacity of current networks and storage media. The purpose of video compression (video coding) is to fill this gap. A video compression system aims to reduce the amount of data required to store or transmit video while maintaining an 'acceptable' level of video quality. Most of the practical systems and standards for video compression are 'lossy', i.e. the volume of data is reduced (compressed) at the expense of a loss of visual quality. The quality loss depends on many factors, but in general, higher compression results in a greater loss of quality. A well-designed video compression system gives very significant performance advantages for visual communications at both low and high transmission bandwidths. At low bandwidths, compression enables applications that would not otherwise be possible, such as basic-quality video telephony over a standard telephone connection. At high bandwidths, compression can support a much higher visual quality. For example, a4 .7 Giga byte DVD can store approximately 2 hours of uncompressed QCIF video (at 15 frames per second) or 2 hours of compressed ITU-R 601 video (at 30 frames per second). Most users would prefer to see 'television-quality' video with smooth motion rather than 'postage-stamp' video with jerky motion. Video compression and video CODECs will therefore remain a vital part of the emerging multimedia industry for the foreseeable future, allowing designers to make the most efficient use of available transmission or storage capacity. In this paper we proposed and implemented an intelligent adaptive video encoder (compressor) and decoder (decompress or) using reconfigurable hardware FPGA.

2.1. **Compression Techniques.** When we speak of a compression technique or a compression algorithm we actually refer to two algorithms: the first one takes an input X and generates a representation XC that requires fewer bits; the second one is a reconstruction algorithm that operates on the compressed representation XC to generate the reconstruction Y .In Lossless compression schemes, in which Y is identical to X, and for lossy compression force Y to be different from X but generally provide much higher compression than lossless ones. In fact Shannon showed that the best performance achievable by a lossless compression algorithm is to encode a stream with an average number of bits equal to the I=N value. On the contrary lossy algorithms do not have upper bounds to the compression ratio.

In the Lossless compression techniques it involves no loss of information. If data have been losslessly compressed, the original data can be recovered exactly from the compressed data. Lossless compression is generally used for discrete data, such as text, computer-generated data and some kind of image and video information. There are many situations that require compression where we want the reconstruction to be identical to the original. There are also a number of situations in which it is possible to relax this requirement in order to get more compression: in these cases lossy compression techniques have to be used.

In the case of lossy compression techniques it involves some loss of information and data that have been compressed using lossy techniques generally cannot be recovered or reconstructed exactly. In return for accepting distortion in the reconstruction, we can generally obtain much higher compression ratios than it is possible with lossless compression. Whether the distortion introduced is acceptable or not depends on the specific application: for instance if the input source X contains a physical information plus noise, while the output Y contains only the physical signal, the distortion introduced is completely acceptable.

2.2. **Performance measurement of Compression Techniques.** A compression algorithm can be evaluated in a number of different ways. We could measure the relative complexity of the algorithm, the memory required to implement the algorithm, how fast the algorithm performs on a given machine or on dedicated hardware, the amount of compression and how closely the reconstruction resembles the original.

A very logical way of measuring how well a compression algorithm compresses a given set of data is to look at the ratio of the number of bits required to represent the data before compression to the number of bits required to represent the data after compression. This ratio is called compression ratio. Suppose of storing an image made up of a square array of 256x256 8-bit pixels: it requires 64 K-Bytes. If the compressed image requires only 16 K-Bytes we would then say that the compression ratio is 4. Another way of reporting compression performance is to provide the average number of bits required to represent a single sample. This is generally referred to as the rate. For instance, for the same image described above, the average number of bits per pixel in the compressed representation is 2: thus the rate is 2 bits/pixel. In lossy compression the reconstruction differs from the original data. Therefore, in order to determine the efficiency of a compression algorithm, we have to find some way to quantify the difference. The difference between the original data and the reconstructed ones is often called distortion. This value is usually calculated as a mathematical or perceptual difference among data before and after compression.

In the lossless compression algorithms, we measure the compression effect by the amount of shrinkage of the source file in comparison to the size of the compressed version. Following this idea, several approaches can be easily understood by the definitions below,

**Compression ratio** is simply the ratio of the output to the input file size of a compression algorithm, i.e. the compressed file size after the compression to the source file size before the compression.

**Compression ratio = size after compression / size before compression**

Compression factor is the reverse of the Compression ratio

**Compression factor = size before compression / size after compression**

Saving percentage is the shrinkage as a percentage.

**Space of Memory saving (%) ≈ (size before compression - size after compression) / size before compression**

2.3. **Shannon's theorem for Information Coding.** Without going into details we just want to recall Shannon's theorem [13]. He defines the information contents of a message in the following way: given a message which is made up of N characters in total containing n different symbols, the information contents measured in bits of the message is the following:

$$I = N \sum_{i=1}^{n} \left(-p_i \log\left(p_i\right)\right) \tag{1}$$

Where $p_i$ is the occurrence probability of symbol i. A symbol depends on the application: it might be an ASCII code, 16 or 32 bit words, words in a text and so on. A practical illustration of the Shannon theorem is the following: let us assume to measure a charge or any other physical quantity using an 8-bit digitizer. Very often measured quantities will be distributed approximately exponentially. Let us assume that the mean value of the statistical distribution is one tenth of the dynamic range, i.e. 25.6. Each value between 0 and 255 is regarded as a symbol. Applying the Shannon's formula with n = 256 and with the equation 2.we obtain a mean information content I=N of 6.11 bits per measured value which is almost 25% less than the 8 bits we need saving the data as a sequence of bytes. Even if we had increased the dynamic range by a factor of 4 using a 10-bit ADC, it turns out that the mean information contents expressed as the number of bits per measurement would have been virtually the same and hence the possible compression gain even higher (39%). This might be surprising but considering that an exponential distribution delivers a value beyond ten times the mean only every e10= 22026 samples.

$$P_i = \frac{e^{\frac{-(i+0.5)}{25.6}}}{25.6} \tag{2}$$

Based upon the requirements of reconstruction, data compression schemes can be divided into two broad classes on is lossy Compression and another one is Loss less Compression.

3. **A General Compressor CODEC Architecture.** A video signal consists of a sequence of individual frames. Each frame may be compressed individually using an image CODEC as described above: this is described as intra Vframe coding, where each frame is 'intra' coded without any reference to other frames. However, better compression performance may be achieved by exploiting the temporal redundancy in a video sequence (the similarities between successive video frames). This may be achieved by adding a 'front end' to the image CODEC, with two main functions:

1. Prediction: create a prediction of the current frame based on one or more previously transmitted frames.
2. Compensation: subtract the prediction from the current frame to produce a 'residual frame'

The residual frame is then processed using an 'image CODEC'. The key to this approach is the prediction function: if the prediction is accurate, the residual frame will contain little data and will hence be compressed to a very small size by the image CODEC. In order to decode the frame, the decoder must 'reverse' the compensation process, adding the prediction to the decoded residual frame (reconstruction). This is inter-frame coding: frames are coded based on some relationship with other video frames, i.e. coding exploits the interdependencies of video frames.

A motion-compensated decoder is usually simpler than the corresponding encoder. The decoder does not need a motion estimation function (since the motion information is transmitted in the coded bit stream) and it contains only a decoding path (compared with the encoding and decoding paths in the encoder).

In this paper we also concentrate on a specific block of a proposed CODEC system .Here we mainly discussed with a specific block implementation that is the Huffman coding with a novel adaptive CAM based compression technique[], and we also simulated and tested into the reconfigurable hardware.

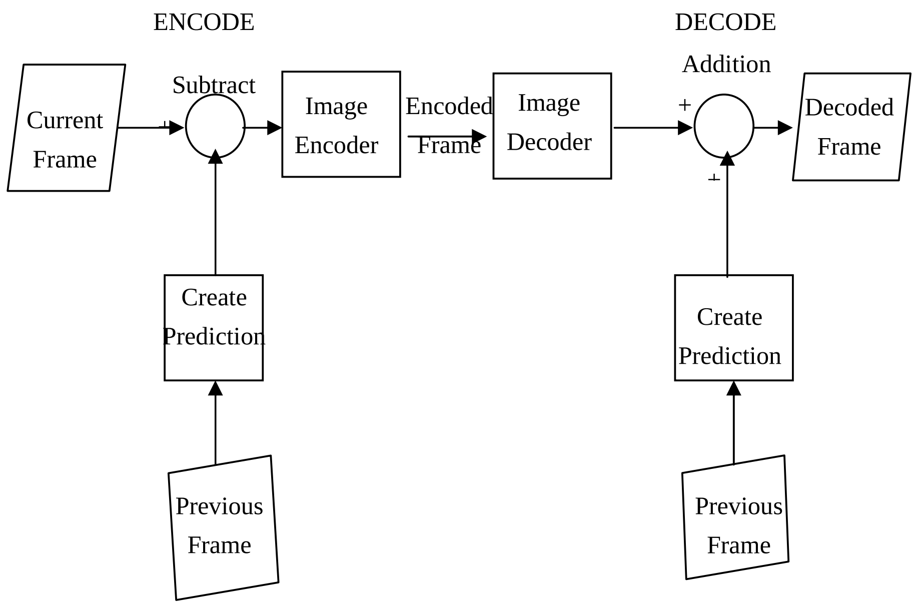


Figure 1. The general architecture of CODEC System.

4. **Huffman Coding.** Huffman based compression algorithm [5-7] encodes data samples in this way: symbols that occur more frequently (i.e. symbols having a higher probability of occurrence) will have shorter code words than symbols that occur less frequently. This leads to a variable-length coding scheme, in which each symbol can be encoded with a different number of bits. The choice of the code to assign to each symbol or, in other words, the design of the Huffman look-up table is carried out with standard criteria. An example can better explain this sentence. Suppose to have 5 data, a1, a2, a3, a4 and a5, each one with a probability of occurrence, P(a1) = 0:2, P(a2) = 0:4, P(a3) = 0:2, P(a4) = 0:1, P(a5) = 0:1; at first, in order to write down the encoding c(ai) of each data ai, it is necessary to order data from the higher probable to the lower probable one, as shown in Table. 1. The Huffman coding blocks are shown into the figure 2, and the simulated result shown in the figure. 4.

Table 1. Final Huffman Coded Table.

| Data | Probability | Code |
|---|---|---|
| $a_2$ | 0.4 | 1 |
| $a_1$ | 0.2 | 01 |
| $a_3$ | 0.2 | 000 |
| $a_4$ | 0.1 | 0010 |
| $a_5$ | 0.1 | 0011 |

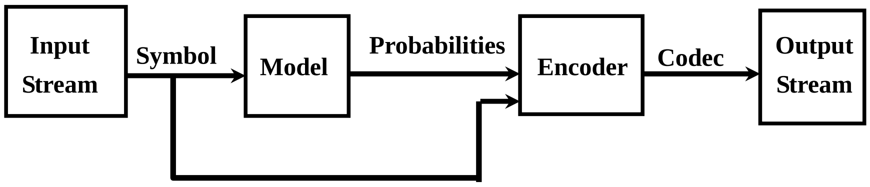


Figure 2. Basic Flow Structure of Huffman Coding.

5. **Proposed Compressor Block with CAM and Huffman coder.** There are lots of research on CAM has been done. Classified as statistical-based or dictionary-based algorithms [8]. Research on statistical-based compression has focused on pushing compression levels to the theoretical limit via highly complex algorithms that, unfortunately, translate to low compression processing speeds such as the prediction by partial matching (PPM) class of algorithms [9V11]. In addition, the algorithmic complexity itself has resulted in only a few relatively simple hardware implementations of statistical-based algorithms [12, 13]. Conversely, dictionary-based compression has concentrated on achieving high-throughput and good compression ratios and is based primarily around the two very popular LZ1 and LZ2 algorithms proposed in [9] and [14].

But in our architecture we used a different dictionary based algorithm. The algorithm uses dictionary of previously seen data and attempt to match the current data element with an entry in the dictionary. This CAM based dictionary has 16, 32 or 64 tuples. The n-tuple dictionary is formed by a total no of n*32 CAM cells. Each cell stores one bit of data tuple and it can maintains its current data, or load the data present in the cell above. The no of tuple increases the more and more possible to finding the match into the dictionary but when it take care of the small data to compress then its gives degraded compression because that time it use only fraction of dictionary length available. So we have think a optimum size, that we take the width of the CAM is 4B/word. The architecture has been shown in the below figure 4. The architecture compares the search data with the data present in the dictionary using one XOR gate to do the comparison of each input bit plus (log2(dictionary width)) 2 input and gates tree to obtain a single comparison bit per dictionary position. The delay of the search operation although in principle is independent of dictionary length, in fan outs and long wires of large dictionaries its speed considerably. An adaptation vector named move in fig and whose length equal to dictionary length defines which cells keep its current data and which cells load data from its north-neighboring cell. We implemented the CAM [15, 18] in hardware by using Xilinx ISE 8.1i [17].The proposed architecture of the Huffman based coder shown into the figure 3. In this paper we introduce a new methodology which is CAM and the Huffman coding techniques simultaneously implemented into the CODEC structure. The CAM implementation for data compression [18] has been introduces in many article, but in our case the CAM (Content Addressable Memory) and the Huffman Coder with this both compression techniques gives the better performance as well as cost effective.

5.1. **Pseudo Code of Huffman Coding.**

A VHDL [18] pseudo code of the Huffman code is given below.

```
Define Library
entity name is
port
(
    MATCH_TYPE: in bit_vector(3 down to 0);
    CODE: out bit_vector(5 down to 0);
    LENGTH: out bit_vector(2 down to 0)
);
end entity ;
architecture HUFFMAN of entity is
begin
MATCH : process(MATCH_TYPE)
begin
case MATCH_TYPE is
    when "0000" => CODE <= "100000"; LENGTH <= "001";
    when "0001" => CODE <= "010000"; LENGTH <= "011";
    when "0010" => CODE <= "001111"; LENGTH <= "110";
    when "0011" => CODE <= "001000"; LENGTH <= "100";
    when "0100" => CODE <= "001110"; LENGTH <= "110";
    when "1000" => CODE <= "000000"; LENGTH <= "011";
    when "1001" => CODE <= "001101"; LENGTH <= "110";
    when "1100" => CODE <= "001100"; LENGTH <= "110";
    when others => CODE <= "000000"; LENGTH <= "000";
end case;
end process ;
end architecture;
```

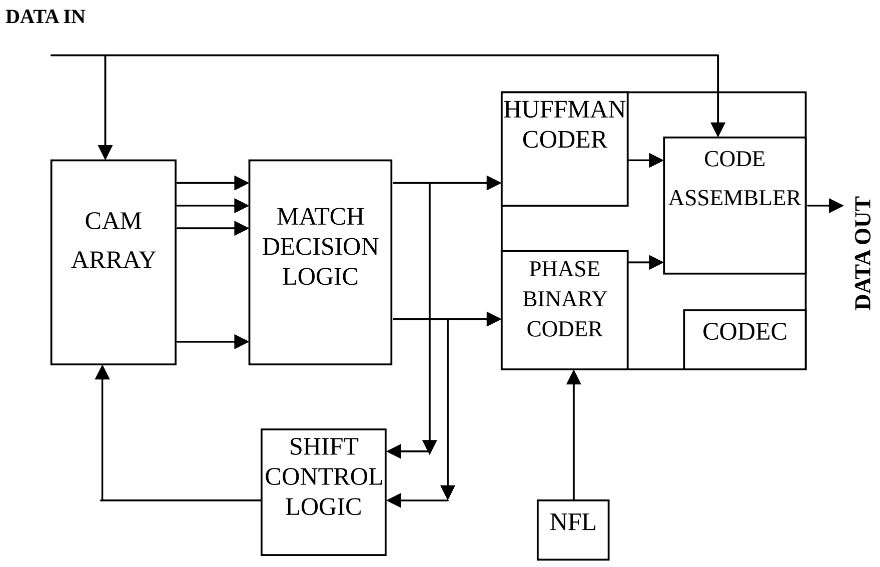
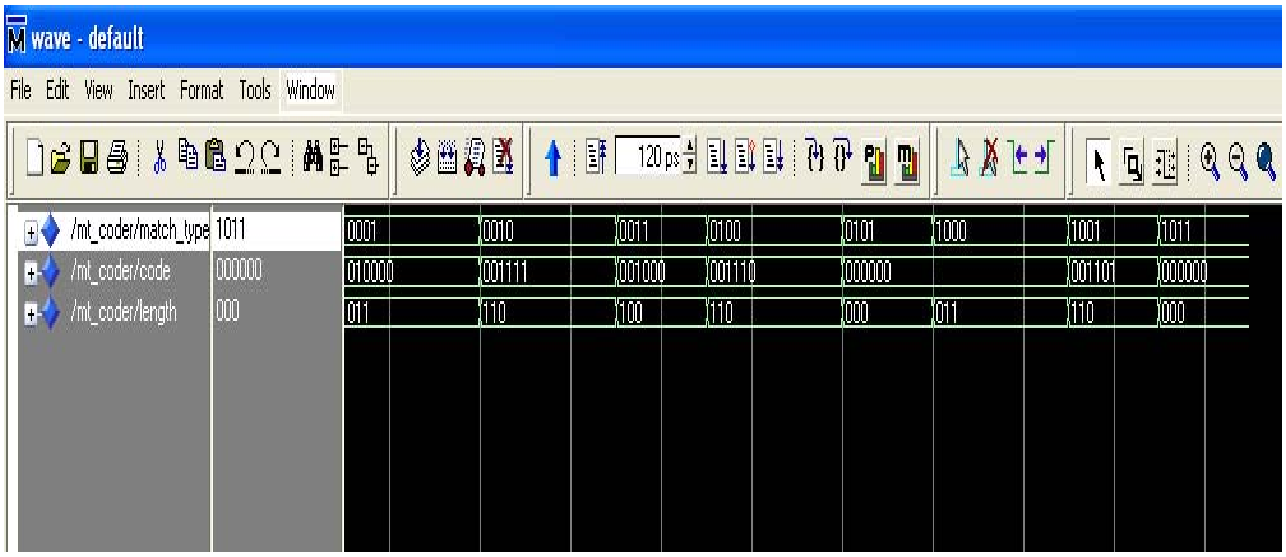


FIGURE 3. Compressor Coder Internal Block Diagram.

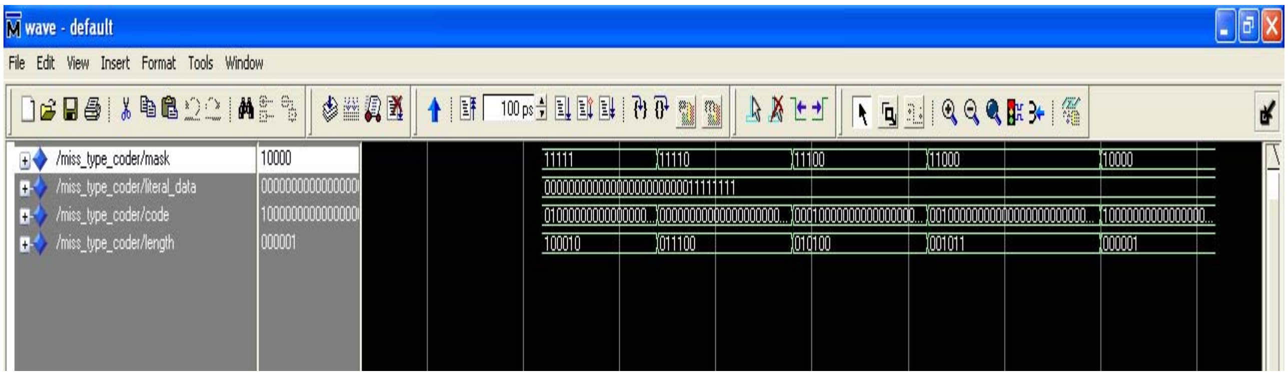
5.2. **Implementation and Simulated Output of the Huffman Compressor Block.**

5.3. **Design Flow of Implementation.** The Xilinx hardware description language (HDL) design flow, illustrated in Figure 5, begins with design entry using an HDL language such as VHDL or Verilog. Then a logic synthesizer such as Synplify Pro [Synp2000] and FPGA Express [Syno2000] will synthesize the design into a netlist of logic gates in the EDIF [SM89] format. Timing constraints are also generated to drive the subsequent stages of the design flow. Then a Xilinx tool called NGD Build converts the netlist into its native circuit description database format (NGD). Then the technology mapper (MAP) will pack the logic gates into Configurable Logic Blocks (CLBs) and I/O Blocks (IOBs) compatible with the target chip technology. The resulting file (NCD) can then be placed and routed using the Xilinx Place and Route (PAR) tool. It can also be Floor planned using the



```
/mt_coder/match_type  1011    0001    0010    0011    0100    0101    1000    1001    1011
/mt_coder/code        000000  010000  001111  001000  001110  000000          001101  000000
/mt_coder/length      000     011     110     100     110     000     011     110     000
```

(a) A Huffman Coding In Put



```
/miss_type_coder/mask          10000              11111  11110  11100  11000  10000
/miss_type_coder/literal_data  0000000000000000   0000000000000000000000000011111111
/miss_type_coder/code          1000000000000000   0100000000000000... 00000000000000000000... 00010000000000000000... 001000000000000000000000... 1000000000000000...
/miss_type_coder/length        000001             100010  011100  010100  001011  000001
```

(b) Huffman Coding Out Put

Figure 4

Xilinx Floor planner, or manually edited using the Xilinx FPGA Editor. TRACE is a timing analyzer that takes an NCD file as input. The Xilinx Description Language (XDL) program can convert between the Xilinx proprietary NCD formats to a text based XDL circuit description file. Synplify- Pro is an advanced synthesis tool that provides useful features such as resource sharing, automatic pipelining and register balancing of the circuit. It has extensive knowledge of the target Xilinx chip architectures so it can produce a highly optimized netlist for the Xilinx backend tools. The Xilinx hardware design flow is timing-driven. With timing constraints set, each stage of the design flow will attempt to optimize the circuit to meet the constraints. The Xilinx Floor-planner is a graphical placement tool that controls where circuit elements get placed in the FPGA. It allows the user to describe coarse constraints on where portions of the circuit are positioned by the automated placement and routing software (PAR). The Floor-planner has two windows, one showing the current placement information of the circuit on the FPGA, and the other showing a Floor-plan, which the user can modify. Each window displays the FPGA as a grid of CLBs, with each CLB showing its associated circuit elements including flip-flops, look-up tables, carry elements, and tri-state buffers. The user employs a drag-and-drop paradigm to move logical grouping of elements onto the Floor-plan (initial Floor-plan) or modify the Floor-plan of a previously placed and routed design by dragging circuit elements around the Floor-plan. The Floor-planner makes use of logical information of the circuit to present all circuit elements in a hierarchical grouping. For example, circuit elements that belong to the same carry chain will be grouped together, so the user can Floor-plan the whole carry chain as a group. Ungrouping and regrouping of circuit elements are also allowed. Figure shows a screen capture of the Xilinx Floor-plan of our design.

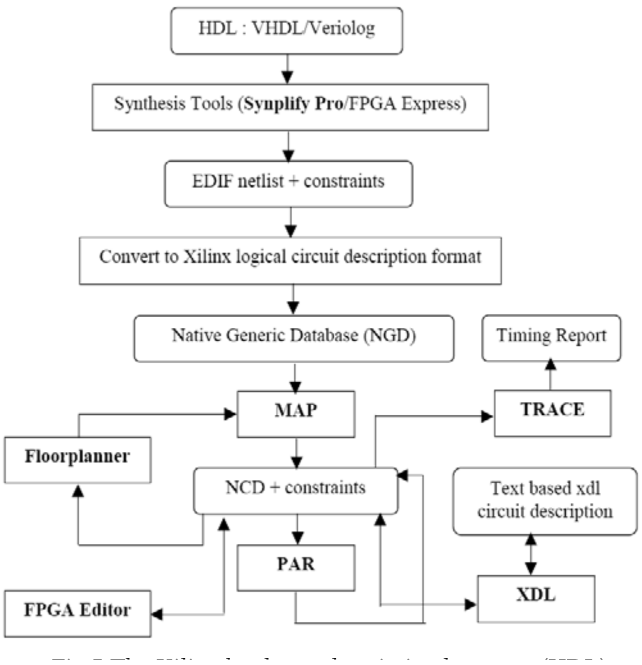
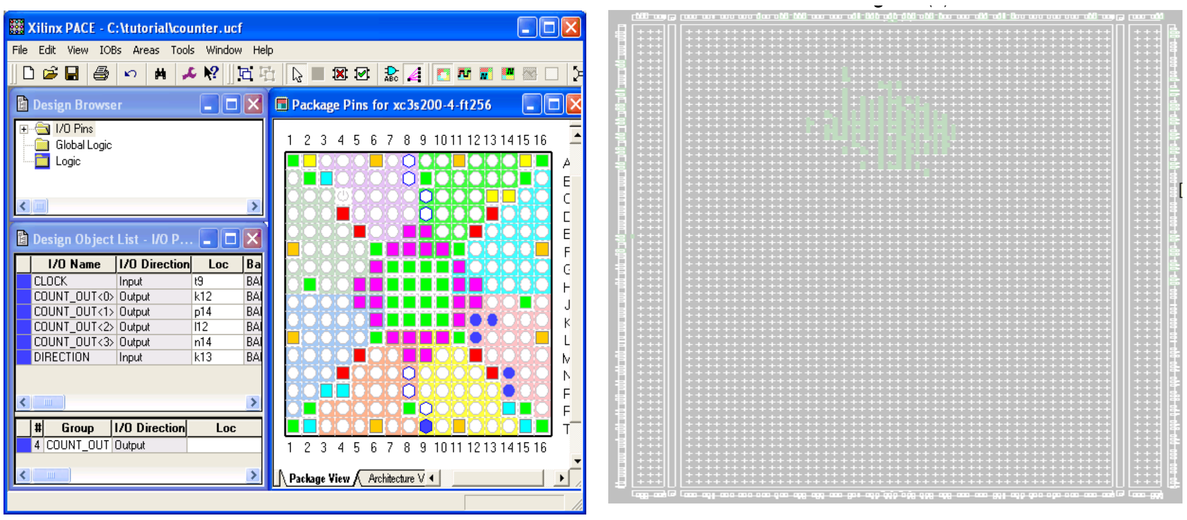

HDL : VHDL/Veriolog
Synthesis Tools (Synplify Pro/FPGA Express)
EDIF netlist + constraints
Convert to Xilinx logical circuit description format
Native Generic Database (NGD)
Timing Report
MAP
TRACE
Floorplanner
NCD + constraints
Text based xdl circuit description
PAR
XDL
FPGA Editor

Figure 5. The Xilinx hardware description language (HDL).


Xilinx PACE - C:\tutorial\counter.ucf
File Edit View IOBs Areas Tools Window Help
Design Browser
I/O Pins
Global Logic
Logic
Design Object List - I/O P...
Package Pins for xc3s200-4-ft256
Package View
Architecture V

| | I/O Name | I/O Direction | Loc | Ba |
|---|---|---|---|---|
| 🟦 | CLOCK | Input | t9 | BA |
| 🟦 | COUNT_OUT<0> | Output | k12 | BA |
| 🟦 | COUNT_OUT<1> | Output | p14 | BA |
| 🟦 | COUNT_OUT<2> | Output | l12 | BA |
| 🟦 | COUNT_OUT<3> | Output | n14 | BA |
| 🟦 | DIRECTION | Input | k13 | BA |

| | # | Group | I/O Direction | Loc |
|---|---|---|---|---|
| 🟦 | 4 | COUNT_OUT | Output | |

(a) (b)

Figure 6. (a),(b) The Xilinx Screen Shot of Floor Plan of the Hoffman Codec Block.

6. **Conclusion.** The proposed Video Design and the Verification of the many modules has been done but many of them has not verified and implemented in to the FPGA.In

this paper we try to complete (Design, Test, and Verification and Implementation) major blocks of the lossless compressor .The design flow has been shown in figure 5. The test and the verification of some block have been done. Few of the design have already tested, verified and implemented as shown in the figure 6 (a), & (b). In our design there are many directions of future work to explore based on this paper. Like in design aspect we only consider the Huffman coding, it can also consider the adaptive Huffman coding, or Wavelet based Method, it can give better result for the compression so it can proposed as future work .The subsequent work are also been published in latter issues.

**Acknowledgement.** The author is thankful to Embedded System Lab of CMERI, Durgapur for giving support to continue his research.

## REFERENCES

[1] W. Mangione et. al, Seeking solutions in configurable computing, *Proc. of IEEE Computer*, vol. 30, no. 12, pp. 38-43, 1997.
[2] J. Villasenor and W. Mangione-Smith, Configurable computing, *Scientific American*, vol. 276, no. 6, pp. 66-71, 1997.
[3] S. Hauck, The role of FPGAs in reprogrammable systems, *Proceeding of IEEE*, vol. 86, no. 4, pp. 615-638, 1997.
[4] J. Villasenor, and B. Hutchings, The flexibility of configurable computing, *Proc. of IEEE Signal Processing Magazine*, pp. 67-84, 1998.
[5] K. Sayood et. al, *Introduction to Data Compression*, 1996.
[6] E. S. Ventsel, *Teoria Delle Probabilita*, Mir Publisher, Mascow, 1983.
[7] D. A. Huffman, A method for the construction of minimum-redundancy codes, *Proc. of IRE*, vol. 40, no. 9, pp. 1098-1101, 1952.
[8] M. Nelson, *The Data Compression Book*, Prentice-Hall, 1991.
[9] J. Cleary, and I. Witten, Data compression using adaptive coding and partial string matching, *IEEE Trans. Communication*, vol. 32, no. 4, pp. 396-402, 1984.
[10] A. Moffat, Implementing the PPM data compression scheme, *IEEE Trans. Communication*, vol. 38, no. 11, pp. 1917-1921, 1998.
[11] G. V. Cormack, and R. N. S. Horspool, Data compression using dynamic Markov modeling, *Computer. Journal*, vol. 30, no. 6, pp. 541-549, 1987.
[12] M. Boo, J. D. Bruguera, and T. Lang, A VLSI architecture for arithmetic coding of multilevel images, *IEEE Trans. Circuits System. II, Analog Digital Signal Processing*, vol. 45, no. 1, pp. 163-168, 1998.
[13] S. R. Kuang, J. M. Joung, R. D. Chen, and Y. H. Shiau, Dynamic pipeline design of an adaptive binary arithmetic coder, *IEEE Trans. Circuits System II, Analog Digital Signal Processing*, vol. 48, no. 6, pp. 813-825, 2001.
[14] J. Ziv, and A. Lempel, A universal algorithm for sequential data compression, *IEEE Trans. Information Theory*, IT-23, pp. 337-343, 1977.
[15] T. P. Banerjee, A. Konar, and J. R. Chowdhury, High-speed communication system development using FPGA based CAM implementation, *Emerging Trends in Engineering and Technology (ICETET),Proc. of the 2nd International Conference*, no. 16-18, pp. 352-357, 2009.
[16] Xilinx Corporation, *The Programmable Logic Data Book*, 2009.
[17] Peter J. Ashenden, *The VHDL Cook book*, First Edition July, 1990.
[18] T. P. Banerjee, A. Konar, and A. Abraham, CAM based high-speed compressed data communication system development using FPGA, *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, no. 9-11, pp. 959-964, 2009.