Volume 5, Issue 6, June 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

VlSI Implementation of Huffman Design Using FPGA With A Comprehensive Analysis of Power Restrictions

Maan Hameed Mohammed, Asem Khmag, Dr. Fakhrul Zaman Rokhani, Assoc. Prof. Abd. Rahman Ramli
Department of Computer & Communication Systems Engineering, (UPM),
Selangor, Malaysia

Abstract— Lossless compression is important in Information hypothesis as well as today's IT field. Lossless design of Huffman is most to a large degree used in the compression arena. However, Huffman coding has some limitations where it depends on the stream of symbols appearing in a file. In fact, Huffman coding creates a code with very few bits for a symbol that has a very high probability of occurrence and an utmost number of bits for a symbol with a low probability of occurrence. In this work Hardware implementation of static Huffman coding for data compression using has been designed, this hardware contains both encoder and decoder-based hardware. The proposed systems Altera DE-2 Board have been used in order to implement the text data compression. The experiments with a simulated environment and the real-time implementation for FPGA with Synopsys power analysis show that constraint has been fulfilled and the target design of the buffer length is appropriate. Power consumption that achieved by the proposed algorithm was 0.0161 mW with frequency 20MHz.and 0.1426 mW with frequency 180MHz within the design limitations.

The proposed design is implemented by using ASIC and FPGA design methodologies. In order to implement the encoder and decoder architectures, 130 nm standard cell libraries was used for ASIC implementation. The simulations are carried out by using Modelsim tool. The architecture of compression and decompression algorithm design has been created using Verilog HDL language. Quartus II 11.1 Web Edition (32-Bit). In addition, simulated using ModelSim-Altera 10.0c (Quartus II 11.1) Starter Edition. And it is implemented using Altera FPGA (DE2) for real time implementation.

Keywords—Data compression, FGPA, ASIC, Binary tree, Clock, synchronous circuit, VLSI.

I. INTRODUCTION

Compression is the art or science of representing data in a compact form. Huffman code is one of the Variable Length Codes (VLC) which uses to compress the information. The advantages of this compression algorithm are efficient usage of channel bandwidth and storage size data [1]. Source coding using Huffman encoding algorithm reduces the number of bits in the data when it compared with the ASCII representation of the string [2]. Data compression is one of the enabling technologies for each of these aspects of the multimedia resolution. Additionally, it reduces the number of bits required to represent a text, image or video. By other words, compression refers to lowering the quantity of information used to represent content without excessively reducing the standard of the original data [3]. Developments information technology forced the designers to work on data compression [2].

Text compression is an application of data compression that encodes the original text with few bits. The objective of data compression is to reduce the redundancy of the text and to store or transmit a specific in an efficient form [4]. The advantage of this method to represent Huffman decoder is that the original information is restoration readily and requires less memory [1]. Figure1, shows Huffman input-output and exhibit length data for each module encoder and decoder. By adding addition signal to display output data form encoder. In this paper, all alphabet characters with different frequencies for each one is used. Then, to implement Huffman techniques to extract the length for all characters by building Huffman tree. After that, writing designing code for the system using Verilog HDL and by using Modelsim in order to design the simulation. Then, to execute the whole design using FPGA for real time implementation. Finally, use Synopsys compiler with library 130 nm technology for analysis power restrictions to point out the limitations of the design with consideration of slack delay for the design in order to get the exact limitations.

This paper is organized as follows. Section2 gives the details on System architecture. Section3, Huffman algorithm. Section4, investigate the quarts, ModelSim and simulation. Section 5, to present the real time implementation of FPGA. Section 6, continuous the Synopsys power analysis. Finally, Section 7, depicts the Conclusions and future work.

II. SYSTEM ARCHITECTURE

Figure1, shows the proposed design where it is composed of two modules: encoder and decoder. Encoder circuit is supposed to receive 8 bit data with frequencies range starts from (20, 40, 60, 80,100,120,140,160,180) MHz clocks and encode it into 9 bit code. Basically, encoder module used to encode the input data in form of 9 bits output text data. On the other hand, decoder module used to decode data in form of 8 bits output data (the same form of the input).Then, build Huffman tree and write ASCII code in Verilog HDL language in order to simulate code design using ModelSim jointly with implement the design on FPGA for real time implementation. A real-time system gives the design determinism real-

time does not mean "real fast" (it can be slower) real-time means that the designer can determine (predict) accurately when a section of the program will execute [6]. Finally, by using 130 nm technologies in order to specify power restrictions for the proposed design.

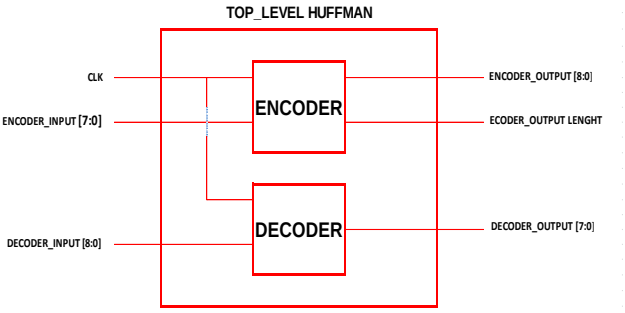

Fig -1: Huffman top_level

III. HUFFMAN CODING

A. Reducing Power Huffman

Reduce Huffman power coding decrease the number of switching activity through information sending. Reducing power Huffman coding involved of two steps: First, reduces the switching activity amidst each symbol by changing the position of leaves node in Huffman tree. Second, rebuild the Huffman tree by completely changing interior nodes to reduce switching activities of any codeword [7]. Huffman coding produced a code with to name few numbers of bits for a symbol that have a very high prospect of occurrence and a larger number of bits for symbols with a low prospect of occurrence [8]. Furthermore, Huffman codes have been exceedingly usage for commencement coding and have shown high eligibility in exploiting the source redundancy [9]. High compressing for test replies is expected because Huffman coding is well-known as a least block coding [3].

B. Huffman Algorithms

Huffman encoding algorithms usage the prospect of the alphabet of the source to generate the code words for symbols. The frequency of all the alphabet of the data input is estimate to find out the prospect the allocation. In the opinion of to the probabilities, the code words are given. Shorter code words are given for upper probabilities and longer code words are given for smaller probabilities. For this task, a binary tree is created using the symbols as leaves according to their probabilities and paths of those leaves are taken as the code words it is codewords one word. Static and Adaptive Huffman Algorithms are considered as the two types of Huffman encoding. Static Huffman algorithms estimate the frequencies first and then generate a ordinary tree for both the compression and decompression operation. The adaptive Huffman algorithms improve the tree while estimating the probabilities and there will be two trees in both processes. In this algorithm, the symbols that have higher frequency are expressed using shorter encodings than symbols which occur less frequently and the two symbols that occur less frequently will have the same length [4]. The proposed design is given a set of symbols and their weights (usually proportional to probabilities), find a prefix free binary code (a set of code words) with minimum expected codeword length (equivalently, a tree with minimum weighted path length from the root [1]. Huffman decoder is implemented for text, and the text compression involves its encoding. Moreover, the text decoder contains Huffman decoder for obtaining the original text. Furthermore, Huffman tree is used by the encoder and the decoder where the alphabets consist of the uppercase letters and the space. All left branches are labeled by 0, and all right branches are labeled by 1. This tree is based on the following assumed frequencies : E 130 ,T 93, N 78, R 77, I 74, O 74, A 73, S 63, D 44, H 35, L 35 ,C 30 ,F 28 ,P 27, U 27 ,M 25 ,Y 19 ,G 16, W 16 ,V 13 ,B 9 ,X 5 ,K 3, Q 3 ,J 2 ,Z 1.
Table-1, shows the input and the output values for all alphabets according to Huffman tree.

VI. MODELSIM AND SIMULATION

A. Code Design

The length for input/output data after building Huffman tree can be achieved. Then, Verilog code for each module will be designed. Basically, the design involves three main modules (encoder, decoder, top _level Huffman). These three codes for Huffman used as an options for (Quartus II 11.1 Web Edition (32-Bit)) software in order to do more than one process for the design such as (analysis & synthesis , analysis & elaboration) to get the best RTL Schematic and technology map viewer for modules.

B. Modelsim Window

ModelSim is a program window is made up of four sections: the main menu at the top, a set of workspace tabs on the left, a work area on the right, and a command prompt at the down. The main list is used to oncoming functions that available in ModelSim and the workspace accommodates a menu of modules and libraries of the modules. The work area on the right is the extent where the windows it contain waveforms and/or text files which will be displayed after. Finally, the command prompt at the dawn shows reactions from the simulation tool and allows users to enter the commands. In the proposed design to Performing the simulation with ModelSimis targeted by creating project where all design files to be simulated are included. By compiling the whole design in order to get the results from the simulation. Based on the results of the simulation, the design can be altered until it meets the desired specifications [18].

Table 1 Binary Code for Text

No	Characters	Frequency	Code	ASIC
1	E	130	010	0100 0101
2	T	93	111	0101 0100
3	N	78	1011	0110 0111
4	R	77	1010	0101 0010
5	I	74	1000	0100 1001
6	O	74	1001	0100 1111
7	A	73	0111	0100 0001
8	S	63	0010	0101 0011
9	D	44	1100	0100 0100
10	H	35	01100	0100 1000
11	L	35	00111	0100 1100
12	C	30	00011	0100 0011
13	F	28	00010	0100 0110
14	P	27	00001	0101 0000
15	U	27	00000	0101 0101
16	M	25	11111	0100 1101
17	Y	19	011011	0101 1001
18	G	16	011010	0100 0111
19	W	16	001101	0101 0111
20	V	13	110101	0101 0110
21	B	9	110100	0100 0010
22	X	5	00110011	0101 1000
23	K	3	00110010	0100 1011
24	Q	3	00110000	0101 0001
25	J	2	001100011	0100 1010
26	Z	1	001100010	0101 1010

C. Simulation Results

The proposed design of Huffman encoder/decoder using Verilog HDL and simulated using ModelSim-Altera 10.0c (Quartus II 11.1) Starter Edition after writing three testbanch codes for Huffman design (encoder, decoder, top_level). Figure 2, shows the waveform of Huffman encoder using only one signal for data input and two signals for output (encoder output & length) where it has been added length signal to the output of the encoder referred to output length.

In Figure 3, the functional simulations for the Huffman decoder are carried out using Modelsim tool. In case of text file both encoder and decoders are presented. The decoder is used to decode the Huffman encoded data. Before writing any program in Verilog, the input and the output ports should be defined clearly[5]. Figure 4, shows the waveform of Huffman design the input and output signals for encoder and decoder are presented during the simulation top-level Huffman design.

V. FPGA IMPLEMENTATION

Field Programmable Gate Array (FPGA) belongs to a group of reconfigurable digital integrated circuits. Their structure usually contains four units: configurable logic blocks (CLBs), input and output, I/O blocks, configurable routes for connections between blocks and blocks with predefined functions, such as memory and arithmetic circuits [10] .Field Programmable Gate Arrays (FPGAs) can be used to implement any hardware design [11].The data file is converted to text file as FPGA can interpret the digital values. Moreover, this text file is given as an input file for FPGA kit [4]. Basically, FPGA use for simulate the design to get real system behavior [12].

In the proposed design schematic as shown in Figure 5, (8 toggle green,18 toggle red, 18 SW) are used to implement the design on FPGA. In encoder implementation 8 green toggles are utilized to represents the encoder input. And the first nine switches (SW [0] to SW [8]) the first 9 toggles red are used to represents the encoder outputs. Moreover, the last four toggles red is used to represent the output length. While in decoder implementation the first 9 switches (SW) are utilized to represent the decoder input and 8 toggles green to represent decoder output. Figure5, shows all switches and wire connections that used in the design for the implementation on FPGA.

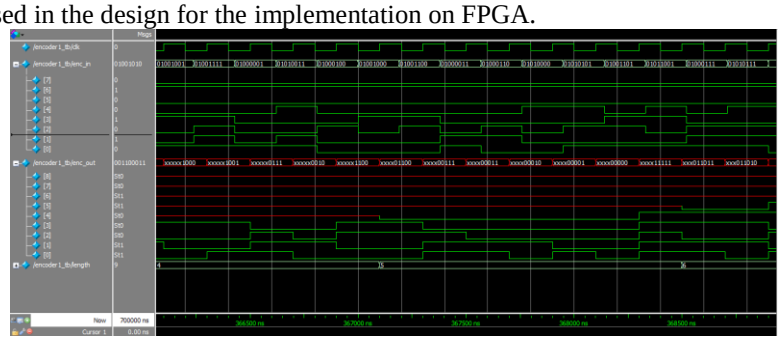

Fig -2: Simulation result of encoder

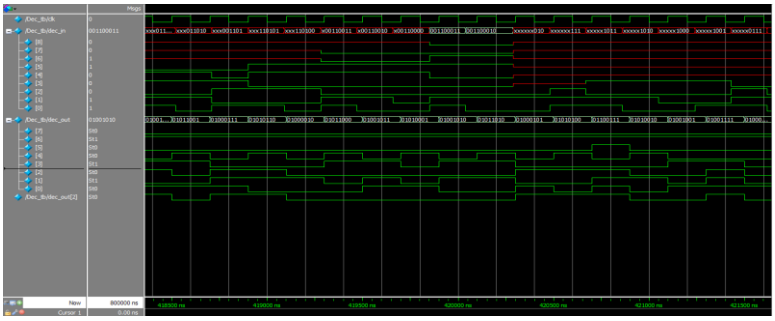

Fig -3: Simulation result of decoder

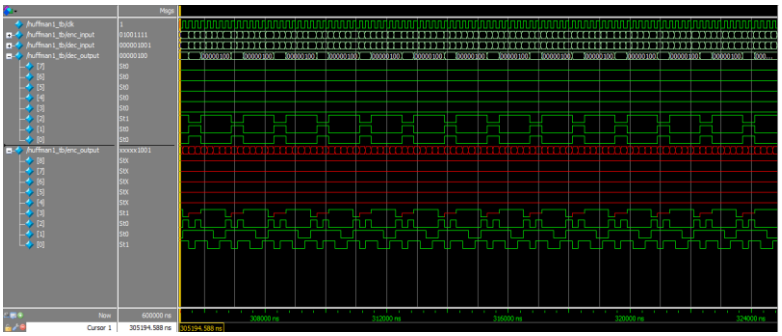

Fig -4: Huffman simulation

VI. SYNOPSYS AND POWER RESTRICTRIONS

Synopsys Verilog compiler (SVC) simulator is a Synopsys tool from it designed especially to simulate and debug designs. There are three main steps in debugging the design, which are follows [12].

1- Compiling the Verilog source code.
2- Running the simulation.
3- Viewing and debugging the generated waveform

A. Synthesis and Verification

1) RTL Synthesis: The translation is to map the register transfer level (RTL) source code to generic technology. Then, based on the constraints, the synthesis tool optimizes the logic and maps it into the target process technology [13]. After synthesis, a netlist file and a standard delay format (SDF) file are generated. The netlist file describes logic by using the components in the target process technology. The SDF file contains path delays, time constraints, interconnect delays, high level technology parameters and etc. Besides the netlist file and SDF file, simulation model files and testbench should be prepared. During this work, the 130 nm process technology is used and the time constraint is set to 50 ns initially to 5.5555 ns.

2) Verification Process: After RTL synthesis and static timing analysis, the produced netlist should be verified to accomplish that task successfully, that is the design meets the specifications [14]. There are three levels for the verification:

- Block level: To test each block to make sure each of them works according to the specification separately.
- Interface level: To test the neighboring blocks with the interface to look into the interaction between blocks.
- System level: To test the entire system.

3) Design Environment Definition: In order to apply the optimization for the proposed design the environment in which the design is expected to operate should be defined. Then, define the environment by specifying operating conditions, wire load models, and system interface characteristics. Operating conditions include temperature, voltage, and process variations are considered as well. Wire load models estimate the effect of wire length on design performance. System interface characteristics include input drives, input and output loads, and fanout loads. The environment model directly affects design synthesis results [15]**.**

*4) Static Timing Analysis***:** The concept of timing path, which is from one flip-flop to another flip-flop. There are many time checks for a flip-flop, such as setup time, hold time, clock skew, and recovery time etc. [17]. In Figure 6, the setup time is the time interval in which input data must be stable before the clock edge. While the hold time is the time interval in which input data must be stable after the clock edge. In the simulation model files, the simulations are expressed in the following way. The setup and hold time constraints are the fundamental ones. And it can affect the stimulus generation [17]**.**

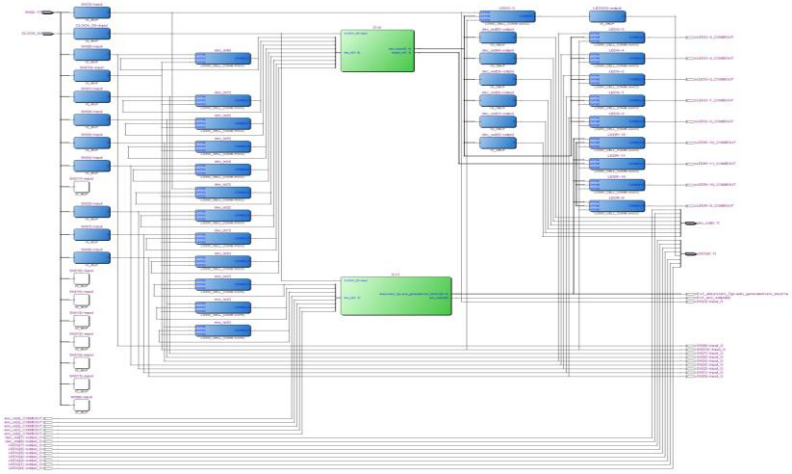

Fig -5: FPGA Implementation of Huffman design

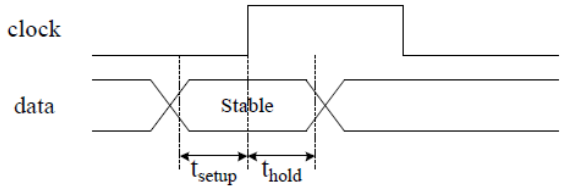

Fig -6: Setup and hold time

B. Power Definition

The total power dissipation in a circuit conventionally consists of two components, namely, the static and dynamic power dissipation [11]. These two elements are the source of power consumption in CMOS circuit's components, dynamic and static consumption [16]. P = P dynamic + P static ……. (1)

1. *6.2.2 Dynamic Power: Dynamic* power dissipation consists of two components one is switching power due to charging and discharging of load capacitance and the other is the short circuit power due to the nonzero rise and fall time of input waveforms. The switching power of a single gate can be expressed as [16]. $P_D = \alpha\ C_L\ V_{DD}^2\ f$…… (2) Where α is the switching activity, f is the operation frequency, C_L is the load capacitance, V_{DD} is the supply voltage. The short circuit power of an unloaded inverter can be approximately given by: $P_{SC} = \beta\ (V_{DD} - V_{th})^3\ \tau / 12T$ ……(3) Where β is the transistor coefficient, τ is the rise/fall time, T (1/f) is the delay.
2. *Leakage Power: Static* power is caused when a transistor is not switching. It comes from subthreshold, gate, and junction leakage currents [11]. In nanometer processes with low threshold voltages and thin gate oxides, leakage power is comparable to dynamic power. In some cases, it may even dominate the overall power consumption [13]. The leakage power (static) and dynamic power analysis of the design can be done by using Synopsys design compiler logic synthesizer. The design has been mapped 130 nm technology and ASIC design methodology adopted for Huffman design, yields low power dissipation and relatively higher performance [1].

The proposed design uses scales of frequencies within the design limitation. Each frequency reads for dynamic power in two types (switching & internal) and leakage power as shown in Table 1.In each process of reading the frequency the value of slack time and critical point for the design should be considered.

Table -1: Power analysis report for Huffman design

Frequency (MH)	Time (NS)	Internal Power(mw)	Switching Power(mw)	Dynamic Power(mw)	Leakage Power(mw)	Total power (mw)	Delay Time (NS)	Area (mm2)
20	50	0.0146	0.0012	0.0159	0.0002	0.0161	45.2900	0.0019
40	25	0.0293	0.0025	0.0318	0.0002	0.0320	20.2900	0.0019
60	16.6666	0.0437	0.0036	0.0473	0.0002	0.0476	11.9500	0.0019
80	12.5	0.0583	0.0049	0.0632	0.0002	0.0635	7.7900	0.0019
100	10	0.0731	0.0062	0.0793	0.0002	0.0796	5.2900	0.0019
120	8.3333	0.0877	0.0074	0.0952	0.0002	0.0954	3.6200	0.0019
140	7.1428	01028	0.0088	0.1115	0.0002	0.1118	2.4300	0.0019
160	6.25	0.1174	0.0100	0.1273	0.0002	0.1276	1.5400	0.0019
180	5.5555	0.1314	0.0110	0.1423	0.0002	0.1426	0.8400	0.0019

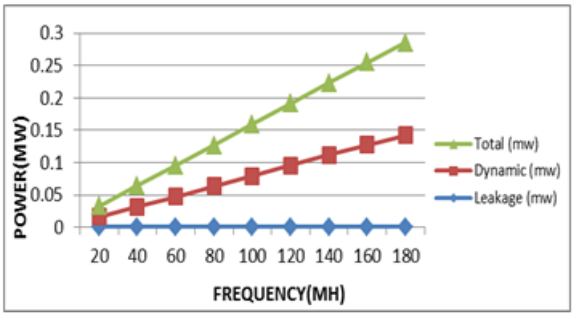

Chart – 1: Power with frequency

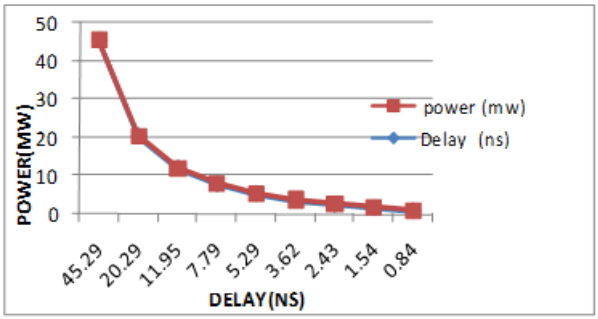

Chart – 2: Power with delay

IX. CONCLUSION

Huffman design using binary tree algorithm was implemented on ASIC and FPGA platforms. Table -1 show Huffman power analysis results. It gives the power, area, and delay time for different scales of frequencies. We used Synopsys design compiler logic synthesizer. The design has been mapped with library 130 nm technology for analysis power

constrains. ASIC design methodology adopted for Huffman encoder/decoder design, yields low power dissipation and relatively higher performance.

The architecture implemented by ASIC design, using library 130 nm technologies, yields constant values for leakage power dissipation of 0.0002 mW during scales of frequencies as shown in Table -1. Dynamic power increases with increasing frequency until it reaches to the near of negative slack. In conclusion, the timing and power analyses of the implementation of the design have shown that the design can be clocked at 20 GHz and dissipates power 0.0161 mW with maximum delay (slack) with 45.29 ns, while when the frequency is increased by taking in our consideration the maximum delay (slack) when it reaches to the critical point for the design we should stop increasing the frequency to avoid design violation. Therefore, the maximum frequency that adequate with the design is 180 MHz with maximum delay 0.84 ns that generate total power dissipation of 0.1423 mW

REFERENCES

[1] Suvvari, V., & Murthy, M. B. (2013). VLSI Implementation of Huffman Decoder using Binary Tree Algorithm. *International Journal of Electronics and Communication Engineering &Technology (IJECET), 4*(6), 85-92.

[2] Anjana, P. (2014). *FPGA Based Iterative JSC Decoding of Huffman Encoded Data for a Communication System.* Paper presented at the International Journal of Engineering Research and Technology.

[3] Kate, D. M. (2012). Hardware Implementation of the Huffman Encoder for Data Compression Using Altera DE2 Board. International Journal of Advances in Engineering Sciences, 2(2), 11-15.

[4] Almelkar, M., & Gandhe, S. Implementation of Lossless Image Compression Using FPGA. Anjana, P. (2014). FPGA Based Iterative JSC Decoding of Huffman Encoded Data for a Communication System. Paper presented at the International Journal of Engineering Research and Technology.

[5] Maadi, M. (2015). An 8b/10b Encoding Serializer/ Deserializer (SerDes) Circuit for High Speed Communication Applications Using a DC Balanced, Partitioned-Block, 8b/10b Transmission Code.

[6] Rinzler, D. G. (2009). Design and Implementation of an FPGA-Based Image Processor: Exploring a Distributed Data Multi-Core Co-Processor Architecture.

[7] Chen, C.-Y., Pai, Y.-T., & Ruan, S.-J. (2006). *Low power Huffman coding for high performance data transmission.* Paper presented at the Hybrid Information Technology, 2006. ICHIT'06. International Conference on.

[8] Beak, S., Van Hieu, B., Park, G., Lee, K., & Jeong, T. (2010). A new binary tree algorithm implementation with Huffman decoder on FPGA. Paper presented at the Consumer Electronics (ICCE), 2010 Digest of Technical Papers International Conference on.

[9] Asha Latha, P., & Rambabu, B. (2012). A New Binary Tree approach of Huffman Code. IJSCE, 1(4), 4.

[10] Kovacevic, J., Stojanovic, R., Karadaglic, D., Asanin, B., Kovacevic, Z., Bundalo, Z., & Softic, F. (2014). FPGA low-power implementation of QRS detectors. Paper presented at the Embedded Computing (MECO), 2014 3rd Mediterranean Conference on.

[11] Aanandam, S. K. (2007). *Deterministic clock gating for low power VLSI design.* National Institute of Technology, Rourkela.

[12] Kommuru, H. B., & Mahmoodi, H. (2009). ASIC Design Flow Tutorial Using Synopsys Tools. *Nano-Electronics & Computing Research Lab, School of Engineering, San Francisco State University San Francisco, CA, Spring*.

[13] KANG, L. (2014). Design and implementation of a decompression engine for a Huffman-based compressed data cache.

[14] Spear, C. (2008). *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*: Springer Science & Business Media.

[15] Mailloux, R. J. (2005). *Phased array antenna handbook*: Artech House Boston.

[16] Nejat, M., Abdevand, M. M., & Farahani, A. M. (2013). *A novel circuit topology for clock-gating-cell suitable for sub/near-threshold designs.* Paper presented at the Computer Architecture and Digital Systems (CADS), 2013 17th CSI International Symposium on.

[17] "Synopsys Timing Constraints and Optimization User Guide" version D-2010.03, March 2010.

[18] Altera "Using ModelSim to Simulate Logic Circuits for Altera FPGA Devices", Altera corporation 2011

