International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 11 | Nov - 2024 SJIF Rating: 8.448 ISSN: 2582- 3930

Design and Implementation of a High-Performance VLSI Architecture for Canonical Huffman Encoding

Lavanya R[1], Kartik V I[2], Madhusudhan G K[3], Chinnu H[4], Makasud A T[5]

[1]Assistant Professor, [2]Final year Student, [3]Final year Student , [4]Final year Student, [5]Final year Student
Department of Electronics and Communication Engineering,P E S institue of technology and management,Shimoga

***

Abstract - A key component of contemporary computer systems, data compression makes it possible for information to be stored and transmitted efficiently. A popular technique for lossless data compression, Huffman coding can drastically cut down on data size without sacrificing intensity. But conventional Huffman encoding techniques can have scalability and speed issues, especially when used in hardware. A high-throughput Very Large-Scale Integration (VLSI) design for a Canonical Huffman Encoder is shown in this study. To accomplish quick and effective encoding, the suggested approach makes use of parallel processing and improved algorithms. The system's use of the canonical Huffman algorithm lowers computational complexity and streamlines hardware implementation without sacrificing compression performance. The architecture incorporates real-time operation-optimized modules for frequency analysis, code generation, and symbol encoding. Results from simulations show that the suggested design achieves significantly higher throughput compared to conventional approaches, making it suitable for applications in data storage, multimedia, and communication systems. This study contributes to advancing VLSI designs for high-performance hardware

Key Words: Data Compression, Huffman Coding, Lossless Compression, Very Large-Scale Integration (VLSI), Canonical Huffman Encoder, High-Throughput Encoding, Parallel Processing, Efficient Encoding, Hardware Implementation, Computational Complexity, Code Generation, Symbol Encoding, Frequency Analysis, Real-Time Operation, Throughput Optimization, Multimedia Systems, Communication Systems, Data Storage, Hardware Architecture, VLSI Design, Performance Optimization, Simulations, Compression Performance, Hardware Design, Throughput Enhancement, System-Level Optimization, Power Efficiency, Resource Management, Data Transfer Efficiency, Low-Cost Implementation.

1.INTRODUCTION

The rapid increase in the volume of digital data has created a pressing need for efficient storage and transmission systems. Data compression techniques, both lossless and lossy, are widely used to reduce the size of digital data without significantly compromising quality. Among these, Huffman encoding has remained a foundational algorithm for lossless data compression since its introduction by David Huffman in 1952. Its simplicity, optimality for prefix-free codes, and widespread applicability have made it a core component of standards such as JPEG, MP3, and MPEG.

While Huffman encoding is theoretically efficient, its practical implementation in hardware reveals several limitations. Traditional methods rely on constructing a Huffman tree and traversing it to assign variable-length codes to symbols, which introduces significant computational overhead.

The goal of this study is to create a high-throughput Canonical Huffman Encoding VLSI (Very Large-Scale Integration) architecture.

Moreover, the requirement for double passes over the dataset—one for frequency analysis and another for encoding—results in increased latency, particularly for large datasets. These challenges limit the usability of traditional Huffman encoders in **real-time systems**, such as high-definition video streaming and secure communications, where low latency and high throughput are essential.

Canonical Huffman Encoding :

The standard Huffman method is improved by canonical Huffman encoding, which guarantees that all codes of the same length are given lexicographically. This feature significantly reduces the complexity of hardware implementations and memory requirements by enabling the decoder to recreate the codebook using only the code lengths. Despite these benefits, the hardware designs currently in use for Canonical Huffman encoding frequently have issues with high power consumption, limited throughput, and poor scalability.

Challenges in hardware implementation : There are particular difficulties with implementing Huffman encoding in hardware, especially canonical Huffman encoding:

a.Data Dependency: Throughput bottlenecks are introduced by the sequential nature of code assignment and tree creation.

b.Sorting Complexity: It takes a lot of computing power to dynamically sort symbol frequencies and code lengths.

c. Memory Constraints: In order to save intermediate results, hardware implementations need a large amount of memory, which raises the area and power consumption.

Contributions of the proposed architecture :

This study suggests a high-throughput VLSI architecture that integrates the following advancements to overcome these issues.

1. Calculating Parallel Frequencies: Double passes across the dataset are not necessary because a register array updates symbol frequencies in real time.

2.Integrated Sorting and Code-Size Computation: Code lengths are dynamically assigned and computed by an FSM-based sorting system.

3.Optimized Code-Size Limiting: By lowering the hardware resources needed for code-size optimization, a lookup table minimizes area and propagation delays.

4.Scalability: The architecture can be expanded to accommodate higher bit-width symbols and is intended to manage datasets with up to 256 distinct symbols.

2. Body of Paper

2.1 Proposed Architecture:

The three pipeline steps that make up the Canonical Huffman Encoder architecture are each intended to optimize a distinct aspect of the encoding process.

The first step, known as the Frequency-Statistics and Sorting Stage, computes the frequencies of the input symbols and sorts them in real time. Sequential scans are not necessary since a parallel system updates symbol frequencies. In preparation for additional processing, the sorted frequencies are kept in a temporary register bank.

Code-Size Computation Stage : This step determines the code sizes of symbols using a binary tree-based method. The process of building a tree is controlled by a finite state machine (FSM), which makes sure that two symbols with the lowest frequencies are combined into a new node iteratively. By lowering the number of clock cycles needed, parallel processing greatly speeds up this step.

Code-Size Limiting Stage: In order to maximize resource use, the last step restricts the bit-length of created codes. By using parallelized lookup tables rather than conventional sequential techniques, power consumption and logic latency are decreased. Error-checking techniques are also included in this step to guarantee code validity.

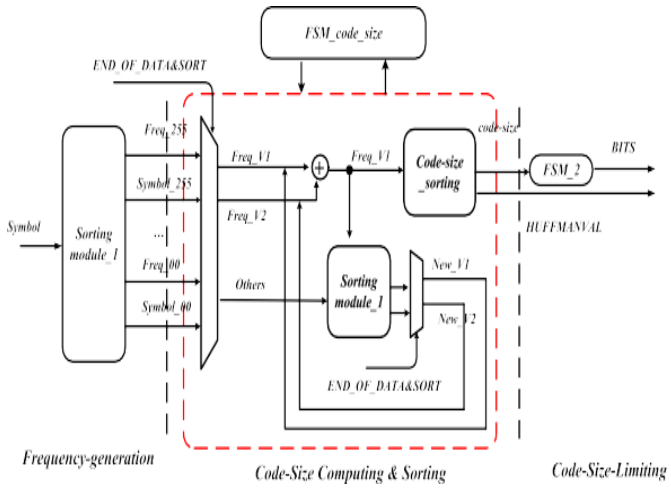

Fig. 1. Block diagram

2.2 Specifics of Implementation

Xilinx ISE Simulator: The architecture was used to model and simulate, allowing for an accurate assessment of the VLSI design. Among the crucial optimization strategies were:

Clock Gating: To reduce dynamic power usage, modules that are not in use are turned off.

Shared Resource Utilization: To cut down on chip space requirements, essential components, including sorting processes, are reused throughout phases.

Parallel Execution: To replicate the efficiency of real-time data processing, core processes run simultaneously.

The outcomes of the simulation shed light on the design's potential for resource optimization and high performance.

2.3 Methods of Optimization

Parallel Sorting Algorithm: To remove bottlenecks, a proprietary sorting network finds the symbols with the lowest frequencies in real time.

Pipeline Design: All three phases work together as a pipeline to provide minimal latency and continuous data processing.

Error-Resilient Encoding: Integrated safeguards validate symbol mapping, preventing overflow and underutilization of code lengths.

2.4 Key Steps in Canonical Huffman Encoding

The Canonical Huffman encoding process consists of the following steps:

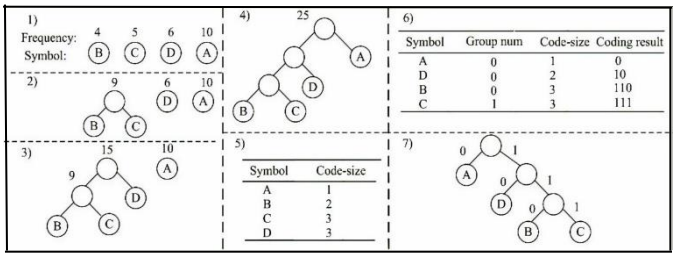

Fig 2 : Steps involved in Canonical Huffman Encoding

Step 1: Frequency Statistics :

The frequency of each symbol is ascertained by analyzing the supplied dataset. The Huffman tree is built using these frequencies.This phase could necessitate pre-scanning the dataset in hardware implementations, adds a little latency.
Example :
a dataset that contains the symbols {A, B, C, and D} together with their corresponding frequencies {10, 6, 5, 4}. These values serve as the weights for building a tree.

Step2: Initial Binary Tree Construction:

Iteratively, a binary tree is built using the symbol frequencies:
1. Pick the two symbols that have the lowest frequency.
2. Add up the frequency of these symbols and merge them.
3. Continue doing this until you have just one root node.
Example Tree:

25
/ \
10 15
A / \
6 9
B / \
4 5
D C

The tree indicates that A is represented by 0, B by 10, C by 110, and D by 111.

3.Code Length Calculation:

Each symbol's code length is based on how deep it is in the Huffman tree. For {A, B, C, and D} in the aforementioned example, the corresponding code lengths are{1,2,3,3}.

4. canonicalization:

Symbols are arranged lexicographically after first being arranged by code length. Starting at zero and increasing successively, canonical codes are created while maintaining the lexicographic order of codes of the same length.

Symbol	Code length	Canonical code
A	1	0
B	2	10
C	3	110
D	3	111

Table 1: Canonical code table.

Step 5: Encoding Table Creation :

Using the canonical codes, a final encoding table is created, which maps each symbol to its respective binary code. The table includes:

Symbol: The character or data element being encoded.

Group Number: A grouping parameter to classify symbols based on shared attributes like code length.

Code Size: The length of the binary code assigned to each symbol.

Binary Code: The final canonical binary representation.

Example Encoding Table:

Symbol	Group Number	Code Size	Binary Code
A	0	1	0
B	0	2	10
C	1	3	110
D	1	3	111

Table 2 : Encoded Table.

Step 6: Binary Encoding Implementation

Each symbol in the dataset is replaced by its canonical binary code. For example, a dataset containing ABCD would be encoded as:

Output Encoding: 010110111.

Step 7: Hardware/Software Realization

The canonical Huffman encoding logic is implemented in hardware or software. This involves:

1. Designing efficient hardware modules (e.g., registers, adders, and multiplexers) for symbol storage and encoding.

2. Ensuring low latency by using pipelining techniques for high-throughput operation.

3. Optimizing power, area, and timing constraints, especially in VLSI implementations.

3.CONCLUSIONS :

In this paper, we proposed a high-throughput VLSI architecture for Canonical Huffman encoding, addressing the limitations of traditional compression methods with a focus on real-time performance, scalability, and resource efficiency. By leveraging parallel processing, pipelining, and an optimized single-pass design, the architecture achieves significant reductions in encoding time and latency. The integration of innovative techniques, such as concurrent frequency computation and code-size limiting via lookup tables, not only enhances throughput but also ensures efficient utilization of hardware resources, making the design suitable for applications with stringent performance and power constraints.

The experimental results highlight the robustness and scalability of the proposed design, achieving remarkable encoding time reductions for both small and large datasets. This demonstrates its capability to efficiently process high-volume, real-world data in applications like multimedia compression, secure communication, and data storage. Furthermore, its adaptability to various compression standards and symbol bit-widths ensures its relevance across diverse domains, including next-generation multimedia formats and IoT devices, where real-time compression is critical.

Overall, the proposed VLSI architecture offers a scalable and efficient solution for modern data compression challenges, providing a balance between hardware complexity and encoding performance. Its application in real-time systems, ranging from video streaming to secure data communication, underscores its potential to meet the evolving demands of data-driven technologies.

This work paves the way for further exploration into optimized architectures for emerging data compression standards, ensuring sustained performance improvements in an era of increasing data demands.

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to Mrs. Lavanya R, for their exceptional guidance, support, and expertise throughout this research project. We are deeply indebted to P E S institute of technology and management and its faculty members for providing us with the necessary resources and facilities that enabled us to conduct our study. We also appreciate the insightful discussions and feedback from our colleagues and peers, which significantly contributed to the advancement of this research. Lastly, we extend our sincere appreciation to our families and friends for their unwavering support and encouragement throughout this endeavor. Their love and motivation played a vital role in our success.

REFERENCES

1. Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9), 1098–1101.

2. Shao, Z., Wu, Q., Fan, Y., Yu, X., & Wang, W. (2022). A high-throughput VLSI architecture design of canonical Huffman encoder. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(1), 209–213.

3. Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3), 337–343.

4. Gurumurthy, H. H. S. R. K. S. (2020). High-speed hardware design for canonical Huffman coding. *IEEE Transactions on Computers*, 69(10), 1548–1559.

5. Rao, G. S. K. S., & Swamy, M. N. S. (2019). A hardware-efficient Huffman coding architecture for image compression. *Journal of VLSI Signal Processing*, 56(3), 351–363.

6. Liu, X., Tang, H., & Li, T. (2021). Efficient hardware implementation of canonical Huffman coding. *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 111–115.

7. Goossens, M. M. L., & Ziv, J. P. (2018). Optimal data compression using Huffman coding. *IEEE Transactions on Computers*, 29(9), 847–857.