A HIGH-SPEED VLSI ARCHITECTURE DESIGN OF CANONICAL HUFFMAN ENCODER


*1Assistant Professor, SR Gudlavalleru Engineering College, Seshadri Rao Knowledge Village, Gudlavalleru, India.
*2,3,4,5Student, Department Of Electronics and Communication Engineering SR Gudlavalleru Engineering College, Seshadri Rao Knowledge Village, Gudlavalleru, India.

ABSTRACT
In recent digital network technologies will translate and receive data with greater complexity because of the number of data bits and the number of memory operations which takes more data loss and low throughput. A high-speed Huffman encoder VLSI architecture based on the Canonical Huffman method is proposed to improve the encoding high-throughput and decrease the encoding time required by the Huffman code word table construction process. In Canonical Huffman coding parallel computing architectures for frequency-statistical sorting and code-size computational sorting are proposed.

Keywords: Huffman Encoder, High-Speed, Frequency Sorting, Encoding Time, High-Throughput.

I. INTRODUCTION
HUFFMAN coding[1] has several applications in data compression[2], image processing[3],[4], audio compression[5], and data security [6], [7]. The "code word table" depicts the data compressible space precisely and is a key part of the Huffman encoding process. Before compression begins, the input symbols must be pre-scanned in order to produce an accurate code word table. Due to the double processing of the input data by this technique, the coding speed is low, and the hardware cost is considerable. The pre-scan procedure is often removed in the commercial algorithms that are now available [8], [9]. The table, however, only has a high compression ratio for input data whose unique frequency distribution is appropriate for the suggested code word table. If not, it has a lower compression ratio. In order to increase encoding performance, the code-size-limiting stage of this brief also suggests a useful VLSI design for limiting bit length. Since the final stage is built using conventional methods, this brief efficiently reduces the circuit area and power usage by nesting the standard algorithms.

![Figure 1: Block diagram of Canonical Huffman encoder](image-url)
Frequency-Statistics & Sorting Stage Working Mechanism: The operating mechanism of Frequency Statistics & Sorting stage is nothing but arranging the bits in ascending order. As the sorting process is used to find two nodes with the minimum and sub-minimum frequencies, optimizing the performance of the sorting algorithm can effectively reduce the clock cycle and improve the efficiency. As most of the symbols are within the range of 0 to 255, to realize the parallel characteristics and accelerate the encoding efficiency, we use a total of 257 cells (Cell_0–Cell_256) to store and update each symbol and its frequency. The Cell_256 module stores the symbol with the largest frequency in the final result, Cell_255 stores the symbol with the second largest frequency, and so on. The Cell_0 module stores the symbol with the smallest frequency and the Cell_256 module is used to provide an inverse code point.

Code-size computing and sorting: The main working mechanism of this stage is based on the standard Canonical Huffman encoding algorithms. According to the above algorithm, as shown in Figure 3.1, this stage is composed of three key modules: Sorting-module-1, Code-size_sorting and FSM_code_size. We design a finite state machine as the kernel of the FSM_code_size module to schedule the other two modules operating in parallel.

Code-size-limiting stage: According to the standard algorithms, the key part of this module contains three layers of sequentially nested lookup tables. Which have three large hardware blocks including CODESIZE look-up tables, BITs look-up table, and MUX-1. Firstly, the sequential lookup table in the standard algorithm is optimized for parallel lookup, which ensures there is only one look-up table on the timing paths. Secondly, we conducted extensive simulation tests on the algorithm, and found that there are a large number of redundant contents in the BIT table, which are not used in practice. Hence, we reduced the size of the BIT form 32*257 to 32*16 and used only what was valid in this table. Based on this method, the size of the multiplexer is also reduced. Numerous simulation experiments have proven the functional correctness of the method. This method greatly reduces the logic delay and the area.

II. LITERATURE SURVEY

Lossless data compression is a key ingredient for efficient data storage, and Huffman coding is amongst the most popular algorithm for variable length coding[1]. Given a set of data symbols and their frequencies of occurrence, Huffman coding generates code words in a way that assigns shorter codes to more frequent symbols to minimize the average code length. Since it guarantees optimality, Huffman coding has been widely adopted for various applications. In modern multi-stage compression designs, it often functions as a back-end of the system to boost compression performance after a domain-specific front-end as in GZIP, JPEG, and MP3[3],[4]. Although arithmetic encoding a generalized version of Huffman encoding which translates an entire message into a single number can achieve better compression for most scenarios, Huffman coding is typically the algorithm of choice for production systems since developers do not have to deal with the patent issues surrounding arithmetic encoding. Canonical Huffman coding has two main benefits over traditional Huffman coding[5]. In basic Huffman coding, the encoder passes the complete Huffman tree structure to the decoder. Therefore, the decoder must traverse the tree to decode every encoded symbol[1],[4]. On the other hand, canonical Huffman coding only transfers the number of bits for each symbol to the decoder, and the decoder reconstructs the codeword for each symbol[2],[3],[5]. This makes the decoder more efficient both in memory usage and computation requirements.

III. PROPOSED MODEL

A. Overview of Proposed Work

The suggested architecture is divided into three steps, as illustrated in Fig.1: frequency generation, computing and sorting of code sizes, and code size limiting. We provide two different real-time frequency-sorting designs that “eat” the input symbol in series methods to increase encoding efficiency and make up for the drawbacks of the canonical Huffman encoder in the first two stages. Based on this hardware design, the code-size-sorting module creates a temporary sorted result of the code-size data queue at each clock cycle. Finally, the HUFFMANVAL results will be shown concurrently with the code-size calculation process. In order to increase encoding performance, the code-size-limiting stage of this brief also suggests a useful VLSI design for limiting bit length. Since the final stage is built using conventional methods, this brief efficiently reduces the circuit area and power usage by nestng the standard algorithms.
B. Architecture of Frequency-Generation Stage

In each cycle, this step "eats" one symbol, resulting in 256 sets of ordered frequencies. The frequency-statistics procedure and the frequency-sorting process are the two processes that makeup this step and run concurrently. The final sorting result maybe acquired practically instantaneously when the last symbol is input since just two pipeline steps are implemented. This can considerably increase throughput and decrease encoding time. In contrast, the sorting module in conventional designs is not activated until all of the input symbols frequency statistics have been completed.

Figure 2: Algorithm flow diagram

Mechanism: Figure 2 represents the Frequency, Statistics, and Sorting stage's operational mechanism. The sorting process is used to identify the two nodes with the least and sub-minimum frequencies; therefore, improving the sorting algorithm's effectiveness may significantly cut down on the clock cycle and increase efficiency. We use a total of 257 cells(Cell_0, Cell_256) to store and update each symbol and its frequency because the majority of the symbols fall within the range of 0 to 255. This allows us to utilize the parallel features and accelerate the encoding efficiency.

If the signal Shift_EN1'b1 is received, the frequency and symbol of the corresponding modules will be updated based on the relationship between the signals FREQ_NEW, FREQ_RIGHT, and FREQ_0. State table represents the updating scheme to decide if a cell has to be updated, FREQ_NEW will be compared to the original frequency. As a result, the values stored in Cell_0 and Cell_256 can be changed sequentially.

IV. RESULTS

The proposed VLSI design was created using the Synopsys Design Compiler and the SMIC0.18-micron standard CMOS cell library. It was specified by the Verilog HDL.
The simulation window will appear pass the input values by making force constant and if it is clock by making force clock. Mentioned the simulation period and run for certain time and results will appear as shown in Figure 3.

**Figure 3:** Simulation result

<table>
<thead>
<tr>
<th>System</th>
<th>Area</th>
<th>Power Consumption</th>
<th>Encoding Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existed system</td>
<td>2,008,766μm²</td>
<td>0.850w</td>
<td>26.695ns</td>
</tr>
<tr>
<td>Proposed system</td>
<td>532 μm²</td>
<td>0.082w</td>
<td>46.096ns</td>
</tr>
</tbody>
</table>

The above table 4 represents the system analysis of existed and proposed work in which almost 92% of power is saved during device utilization. In proposed system encoding time and area are reduced by 90 to 95% with existed system.

### V. CONCLUSION

In order to increase encoding efficiency, a high-speed Huffman encoder VLSI design based on the Canonical Huffman algorithm is suggested. The frequency-statistics process and the sorting process are run practically simultaneously in the frequency-statistics and sorting stage according to the suggested design, which can cut down on the pre-scan's processing time. The Code-Size Computing & Sorting Stage also operates code-size computing and sorting virtually simultaneously, effectively lowering the necessary clock cycle. The suggested Canonical Huffman encoder circuit in this short improved the coding efficiency when compared to the traditional Huffman encoder.

### VI. FUTURE SCOPE

The canonical Huffman encoding process can be extended by introducing clock gating technique to reduce
static power consumption. If the clock gating technique is implemented at top module it reduces power. By reducing unwanted clock cycles toggling reduces which in turn reduces power.

VII. REFERENCES


