A High-Throughput VLSI Architecture Design of Canonical Huffman Encoder

Zhenyu Shao, Zhixiong Di, *Member, IEEE*, Quanyuan Feng, *Senior Member, IEEE*, Qiang Wu, Yibo Fan, Xulin Yu, and Wenqiang Wang

***Abstract*—In this brief, a high-throughput Huffman encoder VLSI architecture based on the Canonical Huffman method is proposed to improve the encoding throughput and decrease the encoding time required by the Huffman code word table construction process. We proposed parallel computing architectures for frequency-statistical sorting and code-size computational sorting. This architecture results in a process of building a tree and assigning symbols that can be completed by scanning the data only once. This solves the problem of the low efficiency of the traditional algorithm, which needs to scan the data twice. Consequently, in addition to the advantages of the high compression ratio inherited from the Canonical Huffman, the proposed architecture has overridden advantages for a high parallelism processing capacity. The experimental results showed that the proposed architecture decreased the encoding time by 26.30% compared to the available Huffman encoder using the standard algorithm when encoding 256 8-bit symbols. Furthermore, the VLSI architecture could further decrease the encoding time when encoding more 8-bit symbols. In particular, when encoding 212,642 8-bit symbols, the proposed VLSI architecture could reduce the encoding time by 87.40%. Thus, compared with the traditional Huffman encoders, this brief achieved the improvement of coding efficiency.**

***Index Terms*—Huffman encoder, JPEG, Image compression, VLSI.**

I. Introduction

HUFFMAN coding [1] has many applications in the fields of data compression [2], image processing [3], [4], audio compression [5], and data security [6], [7]. As a crucial component of the Huffman coding process, the "code word table" accurately reflects the data compressible space.

To obtain a precise code word table, the input symbols have to be pre-scanned before the compression is started. This mechanism causes the input data to be processed twice, which results in a low coding speed and high hardware cost. Generally, a known code word table is adopted in the available commercial algorithms to remove the pre-scan procedure [8], [9]. However, the table has a high compression ratio only for the input data whose specific frequency distribution is suitable for the recommended code word table. Otherwise, it possesses a lower compression ratio.

An efficient memory allocation scheme for Huffman coding using a known code word table was proposed in [10]. It can considerably reduce the computational burden for memory allocation for the Huffman table with little performance degradation. However, a large number of clock cycles is needed to search the Huffman code from the Huffman table. In [11] a new data structure to improve the efficiency of Huffman coding was proposed. Nevertheless, the parameters of the data structure were generated through a series of complex calculations, leading to a low clock frequency. A PLA solution was proposed to achieve fast Huffman coding, but it typically needs a large amount of hardware to store the code word table [12]. The authors of [13] also proposed a CAM utilization method to store the code word table, in which the code word table is reconstructed with the currently encoded data and updated in real-time. However, the complexity is too high for hardware implementation.

In this brief, a high-throughput Huffman encoder VLSI architecture based on the Canonical Huffman encoding method is proposed to overcome the shortcomings of the available designs. We propose parallel computing architectures for frequency statistical sorting and code-size computational sorting. This architecture allows the process of building a tree and assigning symbols to be completed by scanning the data only once, which greatly improves the coding efficiency. In contrast, the standard Canonical Huffman algorithm needs to scan the input data twice to complete the above process. Consequently, in addition to the advantages of the high compression ratio inherited from the Canonical Huffman algorithm, the proposed architecture has overridden advantages for a high parallelism processing capacity.

This brief is organized as follows. Section II provides an overview of the Canonical Huffman encoder algorithm. In Section III, the proposed series architecture is described in

Manuscript received April 24, 2021; accepted June 18, 2021. Date of publication June 22, 2021; date of current version December 23, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61504110, Grant 61804128, and Grant 61531016; in part by the Sichuan Science and Technology Program under Grant 2019YFG0092; and in part by the Alibaba Group through Alibaba Innovative Research (AIR) Program. This brief was recommended by Associate Editor Y. Pu. *(Corresponding author: Zhixiong Di.)*

Zhenyu Shao and Zhixiong Di are with the School of Information Science and Technology, Southwest Jiaotong University, Chengdu 611756, China, and also with the National Key Discipline Laboratory of Wide Band-Gap Semiconductor, Xidian University, Xi'an 710071, China (e-mail: 857141804@qq.com; dizhixiong2@126.com).

Quanyuan Feng and Qiang Wu are with the School of Information Science and Technology, Southwest Jiaotong University, Chengdu 611756, China (e-mail: fengquanyuan@163.com; qwu@home.swjtu.edu.cn).

Yibo Fan is with the State Key Laboratory of ASIC and System, Fudan University, Shanghai 201203, China (e-mail: fanyibo@fudan.edu.cn).

Xulin Yu and Wenqiang Wang are with the Alibaba Cloud Intelligence, Alibaba Group, Hangzhou 311121, China (e-mail: channing.wwq@alibaba-inc.com; xiangwu@alibaba.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSII.2021.3091611.

Digital Object Identifier 10.1109/TCSII.2021.3091611

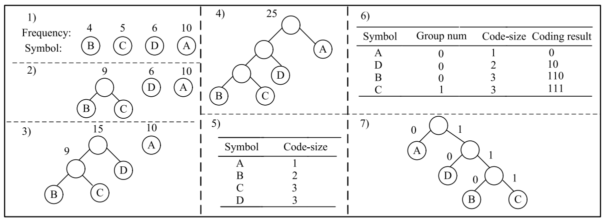


Fig. 1. Process of Canonical Huffman encoding.

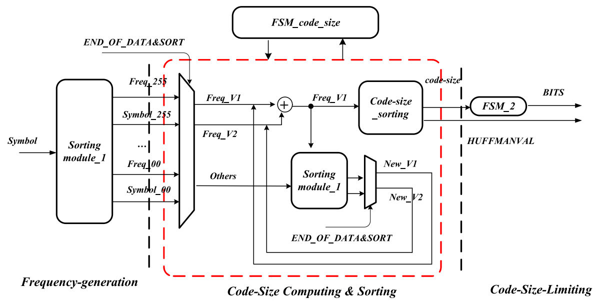


Fig. 2. System circuit block diagram.

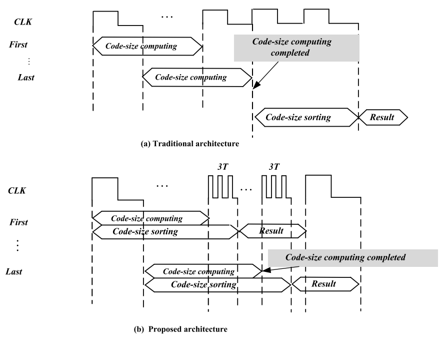

CLK
First
Last
Code-size computing
Code-size computing completed
Code-size sorting
Result
(a) Traditional architecture
3T
Code-size sorting
Code-size computing completed
(b) Proposed architecture

Fig. 3. Timing diagram comparison between the proposed architecture and the traditional designs.

detail. In Section IV, experimental results are provided and compared with other architectures. Finally, brief conclusions are drawn in Section V.

II. Overview of Canonical Huffman Coding

Due to its high compression ratio, Canonical Huffman encoding is widely used in compression algorithms. It mainly consists of two phases: creating a binary tree of nodes and coding based on this tree. This scheme causes the nodes to be scanned twice, which results in a longer encoding time. This means that if we want to achieve a higher compression rate, we must sacrifice coding efficiency.

As is shown in Fig. 1, the tree building procedure is composed of three processes: frequency-statistics process used to determine the frequency of each node, the sorting process used to find two nodes with the minimum and sub-minimum frequencies, and the node creating process. In the node creating process, the two nodes produced by the sorting process will be selected as the children of a newly created node whose frequency is the summary of these two selected nodes. Meanwhile, the code-size of the two selected nodes will be increased by 1.

These three processes are performed iteratively until only one node remains. This remaining node will be the root of the Huffman tree shown in Fig. 1(4) [14]. The code-size of each symbol is shown in Fig. 1(5).

Based on this tree and the corresponding code-size, the Canonical method is executed as follows [14]: (1) The code-size of each symbol is sorted from small to large. Next, the nodes whose code-sizes are equal are divided into the same group. (2) For nodes in the same group, the group number is added in sequential alphabetical order. The coding is allocated according to the code-size and group number. The Canonical Huffman coding result and the corresponding encoding are shown in Fig. 1(6) and 1(7).

III. Proposed Architecture of Huffman Encoder

A. Overview of Proposed Work

As shown in Fig. 2, the proposed design is composed of three stages: Frequency-Generation, Code-Size Computing & Sorting, and Code-Size-Limiting. To improve the encoding efficiency and make up for the shortcomings of the Canonical Huffman encoder, in the first two stages, we propose two types of real-time frequency-sorting architectures that "eat" the input symbol in series schemes. These two architectures are given in detail in Sections III-B and III-C. Based on this hardware architecture, the code-size-sorting module generates a temporary sorted result of the code-size data queue at every clock cycle. Ultimately, once the code-size calculation process is completed, the HUFFMANVAL results will be given simultaneously.

This brief also proposes an efficient VLSI architecture for the Code-Size-Limiting stage that is used to limit the bit length to improve the encoding speed. As the last stage is designed based on the standard algorithms, this biref optimizes the nesting of the standard algorithms to effectively reduce the circuit area and power consumption.

As shown in Fig. 3, compared with the traditional Huffman encoder designs, the proposed architecture can reduce the required clock cycle effectively.

B. Architecture of Frequency-Generation Stage

This stage "eats" one symbol per cycle and then produces 256 sets of ordered frequencies. This stage consists of two parts that operate in parallel: the frequency-statistics process and the frequency-sorting process. Because only two pipeline stages are inserted, the final sorting result can be obtained almost simultaneously when the last symbol is entered, which can improve the throughput significantly and reduce the encoding time. In contrast, in the traditional designs, the sorting module is not started until the frequency statistics of all the input symbols have been completed.

1) Frequency-Statistics & Sorting Stage Working Mechanism: The operating mechanism of Frequency-Statistics & Sorting stage is shown in Fig. 4. As the sorting process is used to find two nodes with the minimum and

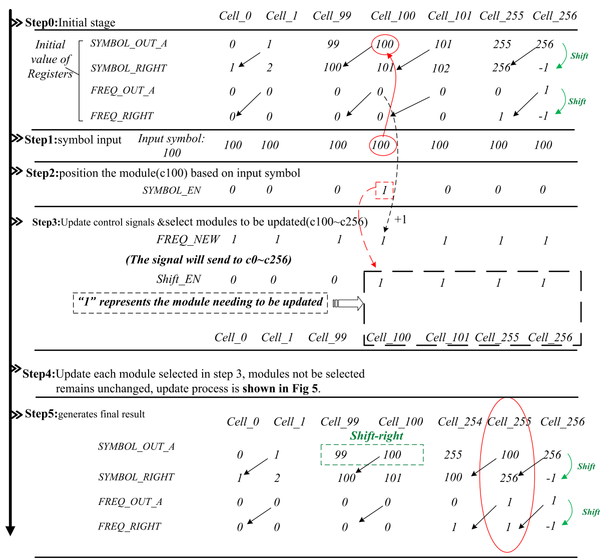
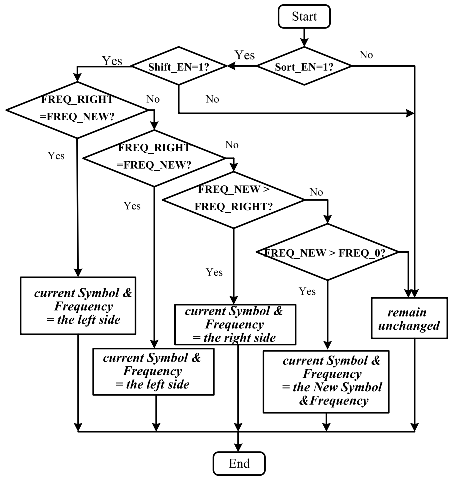


Fig. 4. Working mechanism of Frequency-Statistics & Sorting Stage.



Start
Sort_EN=1?
Shift_EN=1?
FREQ_RIGHT =FREQ_NEW?
FREQ_NEW > FREQ_RIGHT?
FREQ_NEW > FREQ_0?
current Symbol & Frequency = the left side
current Symbol & Frequency = the right side
current Symbol & Frequency = the New Symbol &Frequency
remain unchanged
End

Fig. 5. Updating algorithm flow.

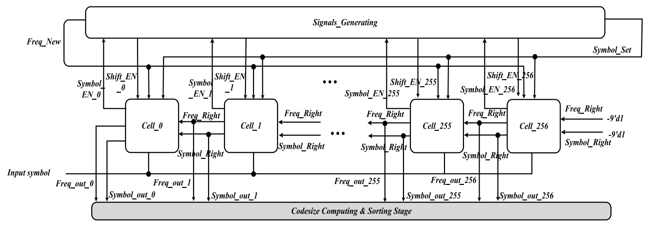


Fig. 6. VLSI architecture diagram of sorting-module-1.

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.

As shown in Fig. 4, the steps are as follows:

- *Step-0:* Initial stage.
- *Step-1:* The proposed architecture receives the input symbol. In Fig. 4, we assume the current input symbol is "100." The symbols stored in each cell will be compared with "100."
- *Step-2:* The cell that stores "100" will be positioned, and the corresponding signal SYMBOL_EN[100] will be set to "1".The signal Shift_EN_100~Shift_EN_256 will be set to "1."
- *Step-3:* The signal Shift_EN that is set to 1'b1 indicates that the corresponding Cell should be updated. At the same time, the new frequency, represented by FREQ_NEW is added by 1 based on the original frequency. The FREQ_OUT_A in cell that stores the symbol "100" will also add 1.
- *Step-4:* Each cell selected in Step-3 is updated based on the scheme shown in Fig. 5.
- *Step-5:* The sorted frequency and its symbol are output. As shown in Fig. 4, Cell_255 has the frequency "1," and the corresponding symbol is "100."

2) Cell Updating Mechanism: If the signal Shift_EN = 1'b1, the frequency and symbol of the corresponding modules will update according to the relationship between signals FREQ_NEW, FREQ_RIGHT and FREQ_0. The updating scheme is shown in Fig. 5.

For each cell, FREQ_NEW will be compared with the original frequency to determine whether the cell needs to be updated. In this way, the values stored in Cell_0~Cell_256 can be updated simultaneously in the same clock cycle.

3) Sorting-Module-1 VLSI Architecture: Fig. 6 shows the VLSI architecture of sorting-module-1. The Signal_Generating block is designed to count the frequencies of the input symbols according to Step-3 in Fig. 4, the value of Symbol_Set is consistent with the input symbol. Cell_0~Cell_256 are designed to generate sorted results based on the methods shown in Step-4 and Step-5 of Fig. 4.

The Signals_Generating block and the 257 cells are triggered by the clock synchronously and operate in parallel. Thus, a temporary frequency statistic and its sorted results are obtained at every clock cycle. Consequently, when the last symbol is imported, the precision frequency statistics of all these input symbols and the final correct sorted result will be produced simultaneously. Compared with the traditional sequential structure of the statistics before sorting, this proposed architecture can save a lot of clock cycles and improve the encoding efficiency.

C. Code-Size Computing & Sorting Stage

The main working mechanism of this stage is based on the standard Canonical Huffman encoding algorithms as follows:

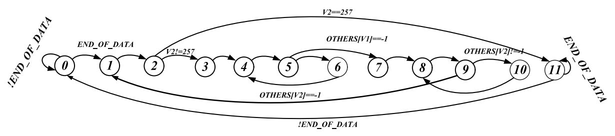


Fig. 7. State transition diagram.

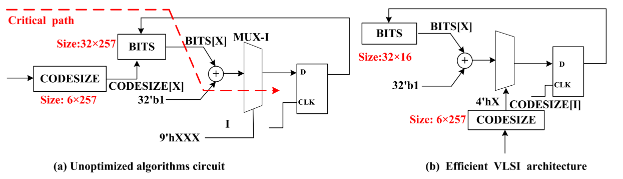


Fig. 8. Code-Size-Limiting look-up table circuit optimization.

- *Step0:* Symbol V1 and V2, which represent the minimum and minor non-zero frequency, respectively, are selected to create a new symbol node.
- *Step1:* The frequencies of V1 and V2 are summed as the frequency of the newly created node in the Step0. The code-sizes of V1 and V2 are increased by one.
- *Step2:* The new symbol node is sorted again with other symbols to select the new V1 and V2 symbols, and Step0–Step2 are repeated. After several iterations, the code-sizes of all the symbols will be generated.
- *Step3:* When the computation of all the symbols' code-sizes has been completed, the code-size will be sorted from small to large to group all the symbols.

According to the above algorithm, as shown in Fig. 2, 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.

If the signal END_OF_DATA = 1'b1, which means all the 256 frequency sets have been ordered by the first stage, the FSM_code_size module starts to calculate the code-size process, re-order the updated frequency queue using Sorting module-1 and order the code-size queue executed by the Code-size_sorting module.

The state transition diagram of the FSM_code_size is presented in Fig. 7:

- State "0" represents the initial state.
- State "1" is to find the symbols V1 and V2.
- States"2" and "3" are to sum the frequencies of V1 and V2.
- States"4"–"10" are used to calculate the code-sizes of V1 and V2 and update the new V1 and V2 simultaneously, and then the code-size is sent to the Code-size_sorting module.

It should be noted that we reuse the sorting-module-1 of the Frequency-Statistics & Sorting stage in this stage to ensure that we can find the new V1 and V2 at every clock cycle, which can also decrease the area of the whole design significantly. Moreover, in the traditional methods, the sorting of the code-size is not started until the computation of all symbols' code-sizes have been completed, leading to considerable time waste. However, in this stage, the computing and sorting of the code-size are performed in parallel.

D. Code-Size-Limiting Stage

According to the standard algorithms, the key part of this module contains three layers of sequentially nested lookup tables. Its corresponding VLSI design is shown in Fig. 8(a) which have three large hardware blocks including CODESIZE look-up tables, BITS look-up table, and MUX-I. These large blocks cause a long propagation delay, large area and power consumption. To overcome these shortcomings, this brief designs the optimized architecture as shown in Fig. 8(b). Firstly, the sequential lookup table in the standard algorithm is optimized for parallel lookup, which ensure 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.

IV. Experimental Result and Comparison

The proposed VLSI architecture was described by the Verilog HDL and synthesized using the Synopsys Design Compiler with the SMIC 0.18μm micron standard CMOS cell library.

Synthesis Result: The synthesis result shows that the operating frequency of the proposed architecture was able to run at 400 MHz. The area of the designed VLSI architecture was 2,008,766μm^2, and the power consumption was 850.84mW. The design presented in [11] ran at 50 Mhz and the area of the design was 340,114μm^2.

Throughput: Ref. [11] used a total of 256 8-bit symbols to test the throughput of the proposed Huffman encoder. The result showed that the encoding time was 840 $\times$ 20 ns $=$ 16800 ns. To demonstrate the performance of the proposed encoder, 200 groups of 256 8-bit symbols were tested, and the VLSI architecture designed in this brief required $4,952 \times 2.5$ ns $= 12,380$ ns for encoding. Compared with the design in [11], the proposed VLSI architecture reduced the encoding time by 26.30%.

As presented in Fig. 9, to further test the designed high throughput of the VLSI architecture, the test began with 256 8-bit symbols and increased the number by 256 8-bit symbols at a time. The VLSI architecture needed 17,726 $\times$2.5 ns $=$ 44,315 ns and the design in [11] needed 10,824 $\times$20 ns $=$ 216,480 ns. When encoding 10,240 8-bit symbols, the proposed VLSI architecture could reduce the encoding time by 79.52%.

Performance: This brief used Kodak24 data set to test the performance. Here only the AC coefficients of Y channel are shown to test the encodeing time. There are 212,642 8-bit symbols on average. The proposed VLSI architecture needs 214,935 cycles on average and the average encoding time

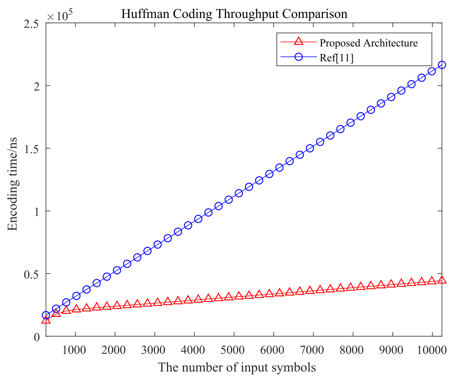


Fig. 9. Throughput comparison of the VLSI architecture.

is 537,338.4 ns. Ref. [11] needs 213,226 cycles on average and the average encoding time is 4,264,517.5 ns. The proposed VLSI architecture could reduce the encoding time by 87.40%. By replacing the traditional Huffman encoder with our proposed VLSI architecture to a JPEG compression system, the compression rate is improved by 12.24% on average, when applied on 100 testing pictures downloaded from Taobao.com with Q value of 100. Compared with the traditional Huffman encoder, the Canonical Huffman encoder can reduce the image size by 12.24%, which means the compressed image size ratio is 87.76%.

We used a ratio to evaluate the efficiency of the circuit. The numerator is the product of the proposed architecture encoding cycles, working frequency, area and image size compression ratio, while the denominator is the product encoding cycles, working frequency, and area corresponding to [11]. A smaller value of the ratio corresponds to a better performance of the circuit designed in this brief. The ratio is defined as follows:

$$\begin{aligned} Ratio &= \frac{The\ proposed}{Ref.\ [11]} \times Compressed\ image\ size\ ratio \\ &= \frac{214,935 cycles \times 2.5 ns \times 2,008,766 \mu m^2}{213,226 cycles \times 20 ns \times 340,114 \mu m^2} \times 87.76\% \\ &= 0.653. \end{aligned} \tag{1}$$

V. Conclusion

In this brief, a high throughput Huffman encoder VLSI architecture based on the Canonical Huffman method is proposed to improve the encoding efficiency. Based on this proposed architecture, the Frequency-Statistics process and the sorting process are operated almost in parallel in the Frequency- Statistics & Sorting Stage, which can reduce the time consumption of the pre-scan. The code-size computing and sorting are also operated almost in parallel in the Code-Size Computing & Sorting Stage, which can reduce the required clock cycle effectively. Compared with the traditional Huffman encoder, the proposed Canonical Huffman encoder circuit in this brief achieved improvements of the coding efficiency.

References

[1] D. A. Huffman, “A method for the construction of minimum-redundancy codes,” *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.

[2] S. J. Sarkar, N. K. Sarkar, and A. Banerjee, “A novel Huffman coding based approach to reduce the size of large data array,” in *Proc. Int. Conf. Circuit Power Comput. Technol. (ICCPCT)*, Nagercoil, India, 2016, pp. 1–5.

[3] Y. Liu and L. Luo, “Lossless compression of full-surface solar magnetic field image based on Huffman coding,” in *Proc. IEEE 2nd Inf. Technol. Netw. Electron. Autom. Control Conf. (ITNEC)*, Chengdu, China, 2017, pp. 899–903.

[4] N. Markandeya and S. Patil, “Improve information rate in Thien and Lin’s image secret sharing scheme using Huffman coding technique,” in *Proc. Int. Conf. Comput. Commun. Control Autom. (ICCUBEA)*, Pune, India, 2017, pp. 1–5.

[5] R. B. Patil and K. D. Kulat, “Audio compression using dynamic Huffman and RLE coding,” in *Proc. 2nd Int. Conf. Commun. Electron. Syst. (ICCES)*, Coimbatore, India, 2017, pp. 160–162.

[6] N. H. Kumar, R. M. Patil, G. Deepak, and B. M. Murthy, “A novel approach for securing data in IoTcloud using DNA cryptography and Huffman coding algorithm,” in *Proc. Int. Conf. Innovat. Inf. Embedded Commun. Syst. (ICIIECS)*, Coimbatore, India, 2017, pp. 1–4.

[7] S. V. Keerthy, T. K. C. R. Kishore, B. Karthikeyan, V. Vaithiyanathan, and M. M. A. Raj, “A hybrid technique for quadrant based data hiding using Huffman coding,” in *Proc. Int. Conf. Innovat. Inf. Embedded Commun. Systems (ICIIECS)*, Coimbatore, India, 2015, pp. 1–6.

[8] *Information Technology—Digital Compression and Coding of Continuous-Tone Still Images: Requirements and Guidelines*, Standard ISO/IEC 10918-1:1994, 2017.

[9] *Information Technology—Generic Coding of Moving Pictures and Associated Audio Information—Part 2: Video*, Standard ISO/IEC 13818-2:2013, 2019.

[10] S. J. Lee, K. H. Yang, J. S. Song, and C. W. Lee, “An efficient memory allocation scheme for Huffman coding of multiple sources,” *Signal Process. Image Commun.*, vol. 14, pp. 311–323, Jan. 1999.

[11] R. Weia and X. Zhang, “Efficient VLSI Huffman encoder implementation and its application in high rate serial data encoding,” *IEICE Electron. Exp.*, vol. 14, no. 21, pp. 1–11, Oct. 2017.

[12] S.-M. Lei and M.-T. Sun, “An entropy coding system for digital HDTV applications,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 1, no. 1, pp. 147–155, Mar. 1991.

[13] T. Kumaki *et al.*, “CAM-based VLSI architecture for Huffman coding with real-time optimization of the code word table,” in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 5. Kobe, Japan, 2005, pp. 5202–5205.

[14] J. Matai, J.-Y. Kim, and R. Kastner, “Energy efficient canonical Huffman encoding,” in *Proc. IEEE 25th Int. Conf. Appl. Spec. Syst. Archit. Process.*, Zurich, Switzerland, 2014, pp. 202–209.