IMAGE CHARACTER COMPRESSION ALGORITHM FOR SYSTEMS USING MICROCONTROLLER

THUẬT TOÁN NÉN KÝ TỰ ẢNH CHO CÁC HỆ THỐNG DÙNG VI ĐIỀU KHIỂN

Nguyen Thanh Hai, Phan Van Hoan, Nguyen Dinh Phu, Ta Xuan Vu
Ho Chi Minh City University of Technology and Education, Vietnam

Received 11/4/2018, Peer reviewed 16/4/2018, Accepted for publication 25/4/2018

ABSTRACT

This paper proposes with the method of image character compression for improving the storage of text in memory during system operation using microcontrollers interfaced with FPGA. In particular, to apply this method, each image character is encoded to reduce the image size of the character compared to the original character and when using it, one needs to decode the image character. Moreover, the compression ratio of each image character is higher compared to the that of the original one depending on the type of image characters. In addition, the high compression ratio allows to saving memory spaces in micro control systems. This compression method can be employed to edit texts for printers with digital systems in the industry, including ARM microcontrollers interfaced with FPGA systems to perform printing control with faster speed and also display results and programming instructions on the screen. In addition, the obtained result can be implemented for other applications and developed in the future.

Keywords: *Character encoder; Character decoder; Pixel group in one character; Character compression rate; ASCII font.*

TÓM TẮT

Bài báo này kiến nghị một phương pháp nén ký tự ảnh để cải thiện việc lưu trữ văn bản trong bộ nhớ suốt quá trình hoạt động của hệ thống sử dụng vi điều khiển giao tiếp với hệ thống FPGA. Cụ thể, để thực hiện phương pháp này, mỗi ký tự ảnh được mã hóa để làm giảm dung lượng ảnh ký tự so với ký tự ban đầu và khi sử dụng nó thì giải mã để cho ra lại ảnh ký tự như ban đầu. Hơn nữa, tỷ lệ sau khi nén của mỗi ký tự ảnh so với ban đầu là vài chục phần trăm tùy thuộc vào mỗi loại ký tự ảnh. Hơn nữa, việc tỷ số nén cao cho phép tiết kiệm được không gian nhớ trong những hệ thống vi điều khiển rất nhiều. Phương pháp nén có thể được ứng dụng để soạn thảo các văn bản cho hệ thống thiết bị phụ trợ in kỹ thuật số trong công nghiệp, trong đó vi điều khiển ARM giao tiếp với FPGA để thực hiện việc điều khiển in với tốc độ cao và còn hiển thị những kết quả và những hướng dẫn lập trình trên màn hình. Hơn nữa, những kết quả thu được có thể được áp dụng cho những ứng dụng khác và phát triển trong tương lai.

Từ khóa: *Mã hóa ký tự; Giải mã ký tự; Chia các điểm ảnh của ký tự; Tỷ lệ nén ký tự; dạng chữ mã ASCII.*

1. INTRODUCTION

In recent years, many algorithms for compressing, encoding and decoding characters and data on microcontroller applications related to communication and interfacing to Field-Programmable Gate Array (FPGA) have been used. A Lempel-Ziv-Storer-Szymanski (LZSS) compression algorithm can be applied to handle up to 50 MB per second on a Virtex-5 FPGA chip. Moreover, the authors of this

paper [1] exploited blocks of Random Access Memory (RAM) with dual-gates that could address independent addresses within an FPGA chip to achieve an average performance of 2 bytes. In order to make the compression flows compatible with the ZLib library [1], the encoding of the output of the LZSS algorithm using a fixed Huffman table was defined by the Deflate specification [2]. In this research, the authors proposed this LZSS method to be able to change the number of memories allocated to different internal tables that affect the performance and compression rate.

The Lempel-Ziv-Welch (LZW) decompression algorithm was developed for applying in the FPGA. Experimental results suggest that a proposed module for the Virtex-7 FPGA family, XC7VX485T-2, runs 2.16 times faster than the decompressed one with the sequential LZW method on a single CPU, where the frequency of FPGA is 301.02 MHz [3]. In addition, the proposed module used a number of FPGA resources, so the research has succeeded in implementing 150 identical FPGA modules, where the FPGA frequency is 245.4 MHz.

Authors showed the design of the FPGA-based with the implementation of a triangular 3D grid. Triangle grids are the main advantages of the 3D geometry model of this paper [4]. The decompression process is based on a simple triangular grid compression algorithm with the high performance, called the BFT encoding. Moreover, this is the first hardware which is performed for triangular decompression. Therefore, the decompression process can be added at the interface first part of a 3D graphics card on the PCI/AGP. This can reduce the required bandwidth on the bus between the host machine for storing data and the graphics card up to 80% compared to the standard triangular grid kinds.

In data compression, authors proposed the Golomb compression algorithm for FPGA systems and this method has been widely used for data compression with the lower complexity in encode and decode systems [5]. The main goal of this Golomb compression algorithm is to find the redundancy and then eliminate it. Therefore, the data is required less memory, as well as the size of the data reduction and this, will reduce the transmission cost. In addition, this method allows to compress low complex data and to precisely decompress into the original data from compressed data.

In other compression techniques, authors proposed three steps: 1) smart arrangement of compression bits to be able to significantly reduce the cost of the decompression; 2) combination of bitmask-based compression and long code run and repeat patterns; 3) parameter selection is beneficial for bitstream compression [6-7]. Further, in discovering the concept of configuration compression, developing algorithms for identification, algorithms are aimed for applying Xilinx Virtex FPGAs with minimal hardware modifications. Therefore, this can significantly reduce the amount of necessary data to transmit during configuration. In research, the authors used compression techniques including Huffman coding, arithmetic coding and Lempel-Ziv (LZ) coding, in which various algorithms were developed to target different hardware structures [8]. The readback algorithm allows certain frames to be reused as a dictionary and fully utilized in the usual way for the configuration bitstream.

GNU Zip (GZIP) is a popular compression method that provides reasonable compression rates [1-3]. Compression algorithms in GZIP using variants of LZ77 encoding, static Huffman coding and dynamic Huffman coding. The fact is that web access accounts is about 42% of all internet traffic accounts. Therefore, GZIP algorithms can be beneficial for reducing internet traffic. Moreover, the hardware implementation with the GZIP algorithm can allow CPU to perform other tasks and then it can increase the system’s performance [9]. In practice, there are many compression algorithms with encoding and decoding such

as jbit encoding (JBE). This algorithm allows to manipulate each bit of data inside the file to minimize its size without losing data after decoding and it is classified to be the lossless data algorithm [10]. Another algorithm is that a word lookup table is the part of the operating system and the data reduction is made by the operating system [11].

In order to work out the reduction of storage capacity in the system using microcontrollers, a method with the process for character compression through ASCII character encoding is suggested in this paper. This is a lossless compression method, meaning that the data after decompressing is the same to data before compression. Moreover, this method not only compresses fonts to reduce the size, but also helps the microcontrollers decode faster than regular bitmap fonts. The advantage of this method is that it is possible to change the font size easily without having to create additional fonts of any size.

2. COMPRESSION ALGORITHM

In order to save memory capacity for storing a large number of fonts, as well as reduce font processing time, the new font compression algorithm is proposed for microcontroller systems that can write multiple letters and process multiple types of fonts in memory. In addition, the compression method is applied for the image font by encoding and decoding characters, it will reduce the amount of memory capacity for saving. Therefore, this can result in higher processing speed and design more processing functions so that the system can meet the real-time requirements throughout the process.

2.1 Character Encoding

In this paper, the font used is very popular ASCII code. In order to apply the proposed compression solution for the font, the analysis of ASCII code is very necessary. In particular, based on the ASCII code table, Latin characters with layouts are continuous lines or and less discrete ones. Thus, the proposed compression method is the statistics of adjacent pixels to perform the compression for each character.

In order to perform compression for each character, the pixels of an image character is divided into column and row groups and then they are worked out using the process of encoding characters as shown in Figure 1.

Figure 1 describes the flowchart of encoding one image character by dividing the pixels of that character. This flowchart allows encoding to reduce the necessary memory capacity for storing characters. This means that the memory capacity used to store texts throughout the editing process in the microcontroller system will reduce. This algorithm is described as follows:

- Block 1: Declare and assign a default value of 0 for the variables used.
- Block 2: Read binary image bitmap file for image information (width, height, file size and image content).
- Block 3: Increase the value of n to 1 for the next column of the image.
- Block 4: if all the columns of the image are compressed, then stop, else continue compression.
- Block 5: Reset k to 0 so that the next step counts the same bytes between the n^{th} and (n^{th} +1) columns.
- Block 6: Increase the m variable to 1 unit for access to the next byte in the n^{th} column.
- Block 7: all the bytes in the n^{th} column are compared with all the bytes in the $(n + 1)^{th}$ column for performing the next.
- Block 8: Compare the m^{th} byte of the n^{th} column to the m^{th} byte of the $(m +1)^{th}$ column. If the same, increase k to 1 to count the same number of bytes between two columns and else, then check the next byte.
- Block 9: Check the n^{th} and $(n + 1)^{th}$ columns for the same. If it is the same, it is not necessary to record;

- Block 10: Increase the *k* variable to count the same bytes between n^{th} and $(n+1)^{th}$ columns.
- Block 11: If Block 9 is used to compare the n^{th} and $(n+1)^{th}$ columns, the range of '1' appears in the n^{th} column. Variable of *sobyte* is to count the number of data bytes after compression and serves the purpose of statistical compression performance.

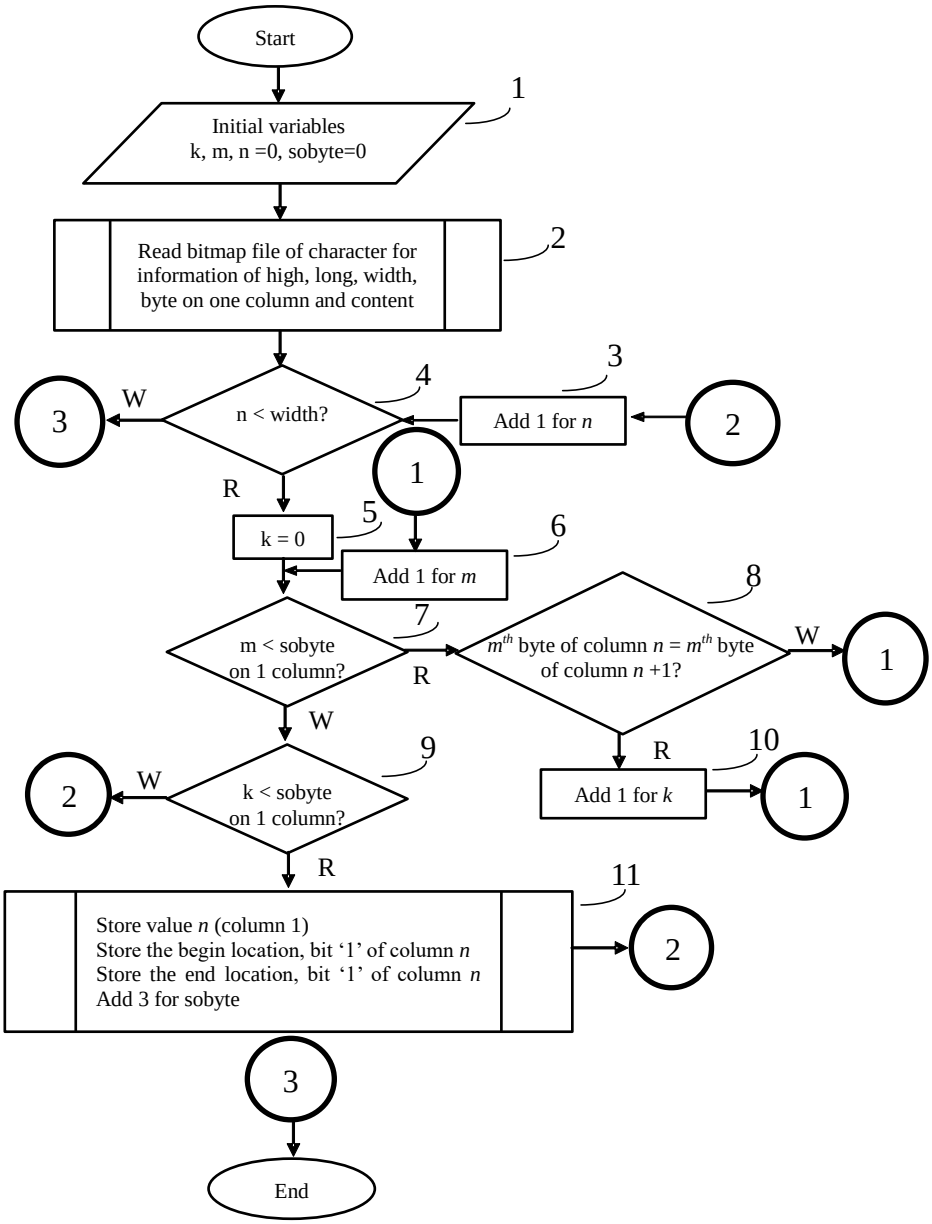

Figure 1. *Flowchart for compressing one character*

For this compression method, the compressed font table with the largest size will be created. However, fonts can be performed smaller than many times, particularly β times, by dividing all the values of the encoded font for β times. To compress an arbitrary character, one can do it by dividing the character with a font length and width equal to half the font of the original character using the following steps:

- Step 1: Get all the elements of the original data and split into 2 original data, so one will get the data after dividing 2.
- Step 2: Decode font according to new data sheet obtained in step 1, one will get the character with reduced capacity.

2.2 Character Decoding

After the character has been encoded, the decoding is performed by returning the encoding sequence above and rebuilding the character's frame. In particular, one will set level 1 (dividing the pixels into level 1s or level 0s) of all empty columns according to the procedure of the following empty column will be similar to the previous column until obtaining a new value.

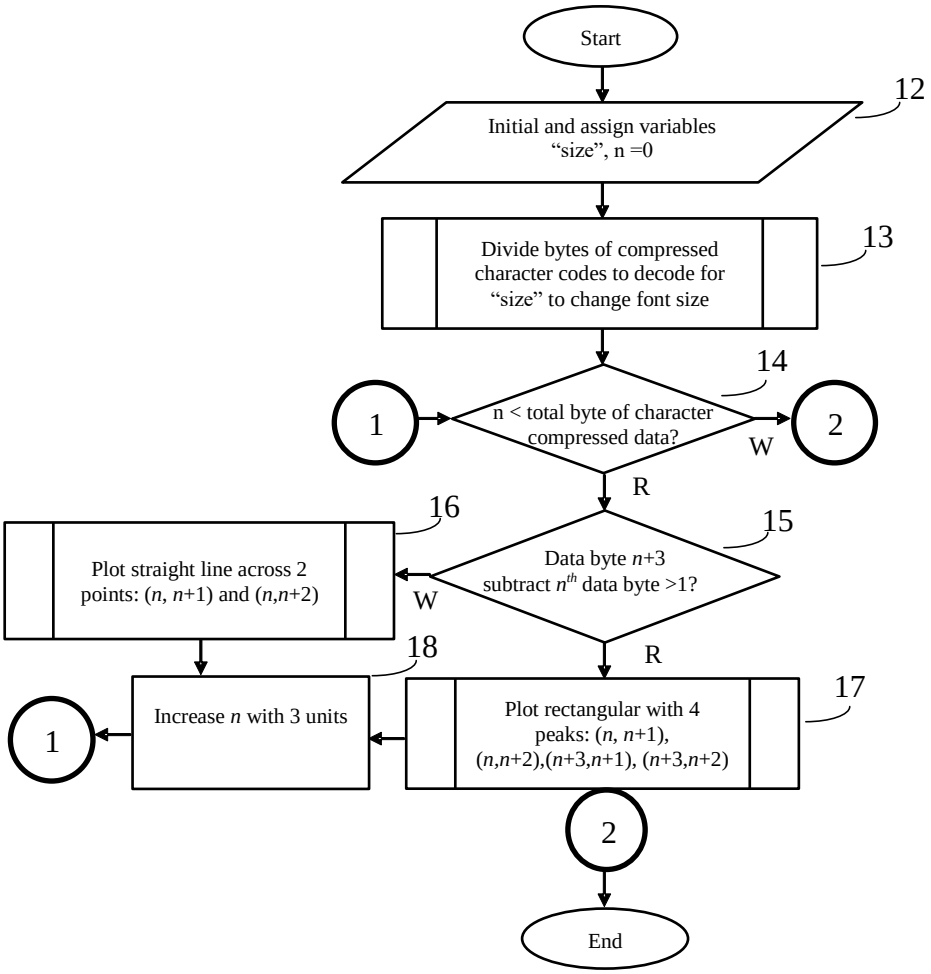

Figure 2. *Flowchart of change of character size and decoding*

Figure 2 shows the flowchart of decoding an image character based on dividing the pixels of the image character. This flowchart allows to decode for use of characters with reduced memory capacity, as well as reducing the text capacity during the editing process using the microcontroller system. This algorithm is described as follows:

- Block 12: Declare the variable *n* to mark the location of the data being processed, variable *size* is to change the size of the character to be decoded.
- Block 13: To change the size of the character, one divides all bytes of the compressed data equal to the size of the character.
- Block 14: Check if decoding until completing an end.
- Block 15: Check if the column being decoded is close to the next column? If correct, just re-draw the current column, else draw a rectangle from the current column to the next one.
- Block 16: If the current column and the next column are adjacents, draw the current column.
- Block 17: If the next column and the next column are far away, draw the rectangles

to connect the current column to the next one.

- Block 18: Increase variable of *n* to decode the next data.

Figure 2 is the description of the flowchart for decoding a character based on dividing the pixels of that character after decoding. This algorithm allows decoding characters and performing texts for display or communication with other modules. After decoding, this compression method allows characters to reduce the amount of memory capacity for storing. Depending on the type and font of the user, this compression solution reduces the size of the file.

2.3 Compression Rate

After encoding, decoding the font character is performed by applying the encoding sequence and rebuilding the frame of the character as the original character. Therefore, one will perform to assign level 1 of all empty columns according to the procedure of the next empty column similar to the following column until getting a new value. Thus, the compression of the font by the encoding and decoding of any character will result in the byte capacity of the character much smaller than that of the original character.

Thus, the calculation of the compression ratio CR of a character using bitmap font is made according to the following formula:

$$CR = Size/ComSi\ ze \quad (1)$$

in which Size is the original character image size and ComSize is the byte number after the compression of one character.

$$Size = (h * w) * b/8 \quad (2)$$

where h is the number of vertical pixels and w is the number of horizontal pixels in the image, b is the gray bit. Therefore, the compression ratio SaveCR for storing is calculated using the following formula:

$$SaveCR = [(Size - ComSize)/S\ ize] * 100 \quad (3)$$

For this compression method, each character. In order to produce a different compression ratio, the compression algorithm depends on the complexity of the image character, as well as the suitability of each design in the microcontroller system when performing interface between different components or devices.

3. RESULTS AND DISCUSSIONS

With the proposed method, the compression of any character to reduce the capacity is performed before decoding to save the storage space into memory. In this article, some characters are designed for compression through encoding and decoding of characters when they need to implement for the typical system based on dividing the pixels into rows and columns.

3.1 Experiments of Character Encoding

In this research, pixels of any image character are divided into columns and rows of pixel groups, and then they are compressed by encoding as described in Figure 1. Moreover, compression of the character depends on the type of fonts. Particularly different fonts after compression will produce the capacity as well as storage space different.

H-character compression: When H character is compressed, the first one is that pixels of the image need to be rearranged in row and column as shown in Figure 3.

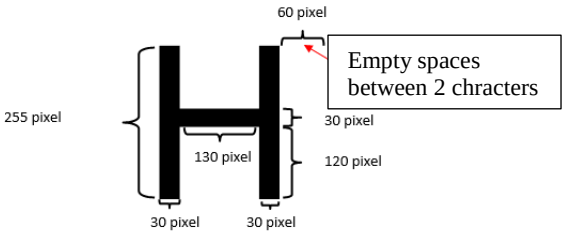

Figure 3. *Arranging the character image H to perform encoding*

Figure 3 shows the H-shape for encoding in the character compression. In particular, the character H is divided into groups of pixels in rows and columns such as 255, 30, 130, 120, 60. After dividing, the pixel groups are arranged in sequence for each group for encoding and performing compression process as described in Figure 1.

The method of character compression H is worked out by compressing the sequence of data according to the process as shown in Figure 1, in which groups of columns and rows of pixels are allocated such as 250, 0, 0, 255, 30, 120, 150, 160, 0, 255, 190, 0, 0. Therefore, when encoding the character H, the compressed character is 12 bytes of data and 1 byte is obtained for applying the total number of columns. Thus, after H compression, there are a total of 13 bytes using the algorithm as described in Figure 1. In particular, the H character is encoded as follows:

- 250: the total columns of the character H is (30+130+30+60 =250)
- 0, 0, 255: meaning that the 0^{th} column is displayed from pixel 0 to 255
- 30, 120, 150: meaning that the 30^{th} column is displayed from pixel 120 to 150
- 160, 0, 255: the 160^{th} column is displayed from pixel 0 to 225
- 190, 0, 0: meaning that the 190^{th} column is nothing to display

Therefore, the storage space rate of the character H after compressing is saved about 99.83%.

E-character compression: When performing E-character compression, pixels are arranged as shown in Figure 4. In particular, the character E is divided into pixel groups of 255, 30, 200, 85, 80, 60 and then perform encoding.

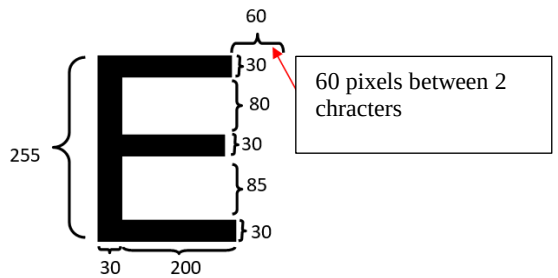

Figure 4. *Arranging the character image E to perform encoding*

The encoding for E character compression is worked out by dividing the image pixels in rows and columns as follows: 290, 0, 0, 255, 30, 0, 30, 30, 115, 145, 30 225, 255, 230, 0, 0. This encoding allows the E character to be compressed to 17 bytes, where 15 bytes of data and 2 bytes for determining the total number of columns. The encoding process is described as follows:

- 290: the total number of columns of the E character (30+200+60 =290), because the pixel number of 290 is more than that of 255, 2 bytes are used to store data
- 0, 0, 255: meaning that the 0^{th} column is displayed from pixel 0 to 255
- 30, 0, 30: meaning that the 30^{th} column is displayed from pixel 0 to 30
- 30, 115, 145: meaning that the 30^{th} column is displayed from pixel 115 to 145
- 30, 225, 255: meaning that the 30^{th} column is displayed from pixel 225 to 255
- 230, 0, 0: meaning that the 230^{th} column is nothing to display

in which, the 30^{th} column is selected with 3 segments connected on the same vertical edge. The ruslt is that the storage space rate of the character E after compressing is saved about 99.82%.

E-character compression: When performing B-character compression, pixels are arranged as shown in Figure 5.

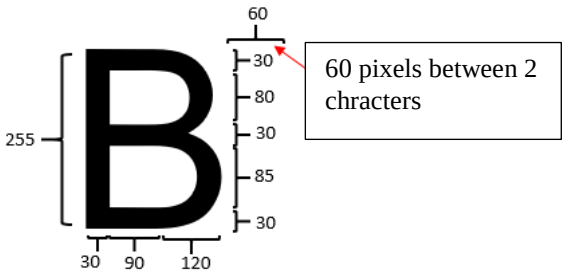

Figure 5. *Arranging the character image B to perform encoding*

Similar to the encoding for H and E compression, B is performed by dividing the pixel group into 255, 30, 90, 120, 85, 80, 60. Therefore, the pixels are divided into the rows and columns of the B character image

for encoding as follows: 300, 0, 0, 255, 30, 0, 30, 30, 115, 145, 30, 225, 255, 120, 1, 31, 120, 114, 146, 120, 224, 245, 122, 2, 32, 122, 113, 147, 122, 123, 253.

The result of the experimental compression of B character using the algorithm as described in Figure 1 is consists of 1069 bytes, in which

- 300: the total number of columns of the B character (30+90+120 + 60 =300), because the pixel number of 300 is more than that of 255, 2 bytes are used to store data
- 0, 0, 255: meaning that the 0^{th} column is displayed from pixel 0 to 255
- 30, 0, 30: meaning that the 30^{th} column is displayed from pixel 0 to 30
- 30, 115, 145: meaning that the 00^{th} column is displayed from pixel 115 to 145
- 30,225,255: meaning that the 30^{th} column is displayed from pixel 225 to 255
- 120, 1, 31: meaning that the 120^{th} column is displayed from pixel 1 to 31
- 120, 114, 146: meaning that the 120^{th} column is displayed from pixel 114 to 246
- 120, 224, 254: meaning that the 120^{th} column is displayed from pixel 224 to 254
- 122, 2, 32: meaning that the 122^{th} column is displayed from pixel 2 to 32
- 122, 113, 144: meaning that the 122^{th} column is displayed from pixel 113 to 144
- 122, 223, 253: meaning that the 122^{th} column is displayed from pixel 223 to 253

The result is that the storage space rate of the character B after compressing is saved about 88.9%.

For this compression method, each character produces a different compression ratio and it depends on the complexity of each character. In addition, experiments with the special characters will produce the low compression ratio, particularly the '@' character can produce the lowest compression ratio with the compressed size of $322*10^{-3}$, while the space-' ' character can produce the highest compression ratio of $376*10^{-6}$.

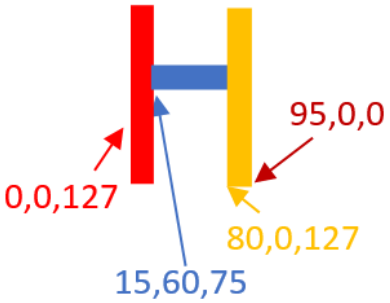

Figure 6. *Representation of H character with haft size reduced*

Figure 6 shows the H character that compressed to reduce haft in size using the proposed encoding algorithm with the pixel groups of rows and columns re-arranged as described in Figure 6.

3.2 Experiments of Character Decoding

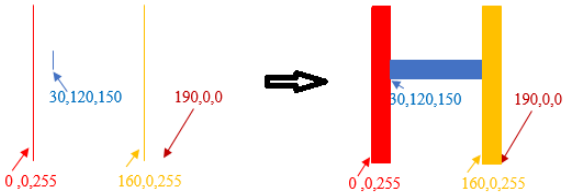

Figure 7. *Representation of decompressing H character*

Figure 7 shows the process of decoding the H character. In particular, the H character after encoding will be decoded with the sorted pixels in sequences such as 125, 0, 0, 127, 15, 60, 75, 80, 0, 127, 95, 0, 0. In similarity, characters of E, B, and other ones when decoded will produce different pixel sequences. Compression of image characters is to reduce the storage capacity of the memory for quickly editing texts and printing on products.

In practice, text fonts (bitmap format) are applied in almost all microcontroller systems for editing and displaying contents. Using bitmap fonts has advantages and disadvantages can be described as follows:

- Advantages: Simple, easy to use and fast decoding.

- Disadvantages: Memory capacity is large to be able to store texts in microcontroller systems. In particular, the microcontrollers with flash memory of 1 Mega Byte (MB) can only store a bitmap font (bmp) with height and width. In addition, it is difficult to change the character size to large or small and in each font size, users have to create the own font code table. This will consume memory resources of the system and also waste a lot of processing time. Therefore, it is not suitable for applications required real-time responses, such as printing industrial products or other applications in industry.

From the proposed method and experimental results for the characters H, E and B to be able to print on the packaging products, the usefulness of this method through the compression ratio is very high. In addition, this compression algorithm is proven on the ARM microcontroller systems for fast editing and also it allows data transfer connected to the FPGA board for printing samples of logos, barcodes and texts on the couche cards.

In the comparison with Huffman's compression method, Huffman's compression ratio was lower. For example, when compressing the character H with 20 rows and 30 columns of pixels, the compression ratio was 42%, while the character H compressed using the proposed method has a compression ratio of 99.83%. This means that the proposed compression method is very effective.

4 CONCLUSION

The proposed font compression algorithm in this article is applied for ASCII font to reduce the size and save memory capacity using encoding and decoding. In particular, the compression process one character is performed by encoding, in which the character is calculated to divide the character image pixels into columns and rows. Thus, the decoding process of the character is based on the encoding sequence to re-build the character frame so that it can be the same to the character before compression. In addition, the character compression algorithm reduces the character capacity for storing as well as the high compression ratio, which proves that the proposed method is suitable and effective in applying for the microcontroller systems, especially when editing texts and transferring to the FPGA systems for fast processing in real time.

ACKNOWLEDGEMENT

This work is supported by HoChiMinh Department of Science and Technology and Sang Tao LMT. company under Grant 44/2017/HĐ-SKHCN. We would also like to thank HCMUTE, students and colleagues for support on this project.

REFERENCES

[1] I. Shcherbakov, C. Weis, and N. Wehn, "A high-performance fpga-based implementation of the lzss compression algorithm," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, 2012, pp. 449-453.

[2] J. Fowers, J.-Y. Kim, D. Burger, and S. Hauck, "A scalable high-bandwidth architecture for lossless compression on fpgas," in *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, 2015, pp. 52-59.

[3] X. Zhou, Y. Ito, and K. Nakano, "An Efficient Implementation of LZW Decompression in the FPGA," in *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*, 2016, pp. 599-607.

[4] T. Mitra and T.-c. Chiueh, "An FPGA implementation of triangle mesh decompression," in *Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on*, 2002, pp. 22-31.
[5] M. P. Sarkar, P. Indurkar, and R. Kadam, "An optimum algorithm for data compression using VHDL," *Int. Res. J. Eng. Technol*, vol. 2, pp. 572-576, 2015.
[6] K. K. Gouse, N. Chitra, and K. Maheshwari, "Compression and Decompression of FPGA Bit Stream Using Bitmask."
[7] P. Hemnath and V. Prabhu, "Compression of FPGA bitstreams using improved RLE algorithm," in *Information Communication and Embedded Systems (ICICES), 2013 International Conference on*, 2013, pp. 834-839.
[8] Z. Li and S. Hauck, "Configuration compression for virtex FPGAs," in *Field-Programmable Custom Computing Machines, 2001. FCCM'01. The 9th Annual IEEE Symposium on*, 2001, pp. 147-159.
[9] S. Rigler, "FPGA-Based Lossless Data Compression Using GNU Zip," University of Waterloo, 2007.
[10] I. Suarjaya, "A new algorithm for data compression optimization," *arXiv preprint arXiv:1209.1045*, 2012.
[11] M. Azad, A. Kalam, R. Sharmeen, S. Ahmad, and S. Kamruzzaman, "An efficient technique for text compression," *arXiv preprint arXiv:1009.4981*, 2010.

Corresponding author:
Nguyen Thanh Hai
Ho Chi Minh City University of Technology and Education
Email: nthai@hcmute.edu.vn