

# Pixel-based Character Image Compression for Data Transfer from ARM Controller to FPGA System

Thanh-Hai Nguyen<sup>1,\*</sup>, Ba-Viet Ngo<sup>1</sup>, Thanh-Tam Nguyen<sup>2</sup>, Duc-Dung Vo<sup>1</sup>, Truong-Duy Nguyen<sup>1</sup>

<sup>1</sup>Department of Industrial Electronic - Biomedical Engineering, HCMC University of Technology and Education, HCM City, Vietnam

<sup>2</sup>Department of Biomedical Engineering, International University-Vietnam National University, HCM City, Vietnam

## Email address:

nthai@hcmute.edu.vn (Thanh-Hai N.)

\*Corresponding author

## To cite this article:

Thanh-Hai Nguyen, Ba-Viet Ngo, Thanh-Tam Nguyen, Duc-Dung Vo, Truong-Duy Nguyen. Pixel-based Character Image Compression for Data Transfer from ARM Controller to FPGA System. *American Journal of Electrical and Computer Engineering*. Vol. 3, No. 2, 2019, pp. 58-66. doi: 10.11648/j.ajece.20190302.12

**Received:** February 4, 2020; **Accepted:** February 20, 2020; **Published:** March 2, 2020

---

**Abstract:** This paper proposes a pixel-based compression algorithm for character digital image in improving the storage of characters in memory during system operation. In particular, in this algorithm, each character binary image in text is grouped by binary numbers and then encoded to reduce the character image capacity of the character compared to the original character. In addition, a novel point in this algorithm is that one character image type is differently grouped binary numbers for compressing. Therefore, the compressed character image is stored in a memory using an ARM microcontroller system and transferred to an FPGA module for decoding before printing. Moreover, the compression ratio of each character is high or low depending on the font type of image characters. Therefore, the high compression ratio using this compression algorithm will allow saving memory space in the memory system. Simulation results show to illustrate the effectiveness of the proposed algorithm and also this compression algorithm was implemented to texts with characters for encoder data transfer from an ARM microcontroller into an FPGA system for effectively printing the text/logo/barcode/QR code/expired date on products with high speed after decoding. Moreover, this compression algorithm can be developed to apply to many different font types and sizes, as well as be utilized different microcontrollers/Microprocessors connected to FPGA systems for processing with high speed. It means that one industrial system using this algorithm can obtain very high performance related to processing digital image characters.

**Keywords:** Character Encoder and Decoder, Pixel Groups in One Character, Character Compression Rate, ARM Microcontroller and FPGA

---

## 1. Introduction

Character compression always attracts researchers due to its effectiveness during storing and transferring data. In many control systems, processing with high speed in real time is very important due to increasing product performance [1-4]. For example, products such as drinking water bottles, smartphones, shampoo bottles, computers and others are recorded information (company logo, brand, date of manufacture, expiry date and product code and product parameters). To be able to print a lot of information at a high speed in a system connected between microcontroller and Field-Programmable Gate Array (FPGA), compressing image containing product information to store in memory with the

microcontroller and to use FPGA for printing high speed is a suitable solution [5-7].

In practice, embedded systems often include Advanced RISC Machine (ARM) microcontroller, Random Access Memory (RAM), Read-Only Memory (ROM), NAND flash and peripherals suitable for use [8]. In the field of industrial printing (printing models on the packaging, outer cover of the product), the input data is usually an image file, so processing the image file on the microcontroller takes a lot of time. However, in order to meet the requirements in real-time control, the combination of FPGA (Field-programmable gate array) and microcontroller is suitable.

The computer has a PCI-E (Peripheral Component Interconnect Express) standard that allows high-speed communication between Video Graphics Array (VGA) card

and Central Processing Unit (CPU), while the microcontroller only has data transfer standards such as SPI, I2C, UART and CAN for transferring an image file from one microcontroller to an FPGA and then the image must be downsized before transfer [9, 10]. Therefore, for transfer with high speed, it is necessary to study data compression and decompression algorithms for applying hardware systems to meet the set targets.

The compression algorithm using LZSS method is capable of processing up to 50 MB per second on a Virtex-5 FPGA chip [11]. The authors exploited dual-port RAM blocks that could be independently addressed within the FPGA chip to achieve an average performance of 2 bytes of speed. To make the compressed stream compatible with the ZLib library, the output encoding of the LZSS algorithm was applied for a fixed Huffman table defined by the Deflate specification [12]. In this study, Fowers recommended a method of changing the amount of memory allocated to different internal tables affected performance and compression ratio.

The LZW decompression algorithm has been studied in recent years and the authors applied it for an FPGA system [13]. The experimental results showed that a proposed module for the Virtex-7 FPGA XC7VX485T-2 family run 2.16 times faster than when decompressing using the sequential LZW method on a single CPU with the 301.02 MHz frequency of FPGA. Because this proposed module was designed to be compact and used in some FPGA resources. Therefore, the study has been successful in implementing 150 identical modules on the FPGA, where the frequency of the FPGA is 245.4MHz.

In another study, the authors designed one system based on FPGA and implemented a set of 3D triangular grids [14]. Triangular mesh is the main advantage in 3D geometry. Therefore, the prototype extraction process was performed based on a simple and highly effective triangle mesh compression algorithm, called BFT coding. Moreover, this is the first hardware made for triangular decompression. Therefore, the decompression procedure could be added at the top of the interface of a 3D graphics card on PCI/AGP. This reduced the bus bandwidth required between the host and the graphics card by up to 80% compared to standard triangular mesh representations.

Research [15] described a data compression method for FPGA systems called the Golomb encryption algorithm. This method was widely used for data compression with lower complexity in encryption and decoding methods. The main goal of this data compression was to find redundancy and to eliminate it through the Golomb algorithm. Therefore, the data are required less memory and the small size of the data, so the transmission cost was lower reduction. This study showed the low-complexity data compression and accurate reproduction of the original data from the compressed data, while the data compression could be lost and unable to reconstruct the original data completely from the data compressed.

In other compression techniques, the authors put out 3 steps: 1-intelligent arrangement of compression bits which

can significantly reduce the cost of the decompression engine; 2-combination of bitmask-based compression and long running code and repeating patterns; 3-parameter selection is beneficial for bit stream compression [16, 17]. Moreover, exploring the idea of configuration compression, developing algorithms for identification systems and these algorithms aimed to apply Xilinx Virtex FPGA with minimal hardware modification for significantly reducing the amount of data. In this study, the authors used compression techniques including Huffman coding, arithmetic coding and LZ (Lempel-Ziv) encryption, different algorithms developed to target different hardware structures. In the "Read back" algorithm, certain frames were reused as a dictionary and it was fully utilized in the configuration bit stream [18].

GNU Zip (GZIP) is a popular compression utility that provides a reasonable compression ratio without exploiting the patented compression algorithms [11-13]. The authors introduced the compression algorithm in GZIP using variants of LZ77 encoding, static Huffman coding and dynamic Huffman coding. Given that web traffic accounts for 42% of all internet traffic, the acceleration of algorithms like GZIP could be beneficial for reducing internet traffic. The hardware implementation of the GZIP algorithm could be used to allow CPUs to perform other tasks, thus it increased the performance of the system [19]. Besides many compression and encryption algorithms such as Jbit Encryption (JBE), this algorithm manipulated every bit of data inside the file to minimize the size without losing data after decoding and then it was classified into lossless data [20]. Moreover, a word lookup algorithm was the part of the operating system and the reduction was worked out by the operating system [21].

To solve the reduction of storage capacity in a microcontroller system, the authors used one JPEG lossy compression method with DCT encryption algorithm combined with Huffman encryption [22-25]. In particular, the image used in this research is a grayscale image with a code word created by scanning a quantized matrix. The results achieved used on Keil C software platform for ARM7 microcontrollers with good compression ratio of 90% to 95% [26]. In the article [27], the authors utilized one dictionary encryption method in combination with a Huffman encryption one to compress data running on different hardware platforms. With the instructions used for PowerPC, i386, and ARM microcontrollers, it was summarized that the important factors affected compression results. The first important factor is the size of the dictionary and this is the most important parameter to achieve a good compression ratio. The second factor is that the size of the code word below the size of the script is reduced. The results of this paper were achieved by an average reduction of 39%, 34% and 26% for PowerPC, ARM and i386, respectively.

In this article, to be able to store large amounts of character images, as well as to reduce the processing time of text, the character image compression algorithm is recommended so that the system using the ARM microcontroller can save many characters and other information in memory. Besides, this

character compression method is performed by encoding and decoding the characters and this allows reducing the amount of storage in memory. In addition, it can increase processing speed and design more processing functions with meeting real-time requirements during the operation of a system. Moreover, this paper shows an application of connecting between one ARM microcontroller (encoder block) and one FPGA (decoder block) for the input character images, in which the decoding block of the FPGA for decompressing the data and then recreate the character image sent by the encryption block for printing information as shown in Figure 1.

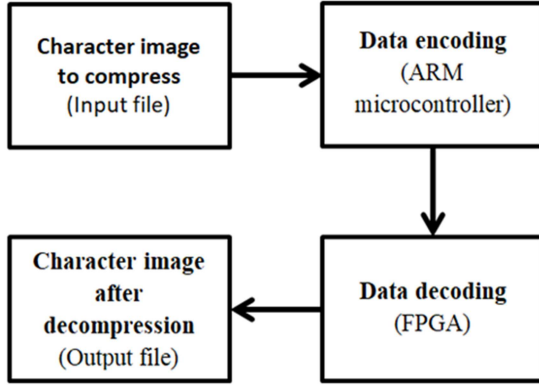


Figure 1. Block diagram depicting the process of compression and decompression.

## 2. Proposed Methodology

### 2.1. Compression Algorithm Based on Pixels

For character compression, all character images in text are converted into binary images and each binary character will be analyzed to divide binary pixels into pixel groups for encoding.

#### 2.1.1. Pixel Analysis of Character image

In applications, bitmap fonts often have high performance rates because they are already pre-decoded. However, these fonts are very memory intensive to store in memory spaces.

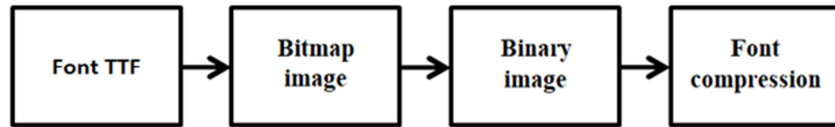


Figure 2. Block diagram of TTF font compression solution.

According to Figure 2, to convert TTF fonts to bitmap formats, a proposed algorithm allows displaying the characters of the font converted and then capturing the displayed image to save as a bitmap file. To convert the bitmaps to binary images, the image segmentation method is applied to retain the lines and the font compression algorithm with the binary font is performed by the following steps:

Step 1: Put the character to be compressed into a coordinate system to address each pixel in the character.

Step 2: Analyze each column of the character to be compressed one by one and mark the coordinates of the

displayed areas.

For example, to store an Arial font with unsigned for the print head, the font size can be determined by the following formula:

$$Size = 3 \times h \times w \times n_c \times n_s \quad (1)$$

where  $Size$  is the font package size;  $h$  denotes the height of the character;  $w$  is the width of the character;  $n_c$  describes the number of characters to create in the font table;  $n_s$  is the number of font sizes of fonts

For characters with  $h=300$ ,  $w=300$ ,  $n_c=128$ ,  $n_s=27$ , the font size is calculated as follows:

$$Size = 3 \times 300 \times 300 \times 128 \times 27 = 933120000 \text{ (bytes)} \quad (2)$$

According to Eq. (2), storing the Arial font set with 27 different sizes is required about 1GB of memory. Because this capacity is too large for a microcontroller-based system, so the use of bitmap fonts is not feasible and the design for this system type should be consider by other options.

In the case of using binary fonts, the memory capacity is also significantly reduced. However, when decoding, the use of bit splitting operations will take more time and this can causes a delay in the printer system. Therefore, the formula to calculate the binary font size is described as follows:

$$Size = \frac{3 \times h \times w \times n_c \times n_s}{24} \quad (3)$$

For characters with  $h=300$ ,  $w=300$ ,  $n_c=128$ ,  $n_s=27$ , the font size is calculated as follows:

$$Size = \frac{3 \times 300 \times 300 \times 128 \times 27}{24} = 38880000 \text{ (bytes)} \quad (4)$$

To further reduce the font size and the decoding time, the font compression solution can be chosen as described in Figure 2.

displayed areas.

Step 3: Remove adjacent columns with the same parameters.

Figure 3 depicts the result of compressing the letter E. After compressing memory, it takes 13 bytes for the letter E: 17, 1, 0, 27, 4, 0, 2, 4, 14, 16, 4, 25, 27. Specifically:

17: the word includes 17 columns.

1, 0, 27: column 1<sup>st</sup> displays from row 0<sup>th</sup> to row 27<sup>th</sup>.

4, 0, 2: column 4<sup>th</sup> displayed from row 0<sup>th</sup> to row 2<sup>nd</sup>.

4, 14, 16: column 4<sup>th</sup> displays from row 14<sup>th</sup> to row 16<sup>th</sup>.

4, 25, 27: column 4<sup>th</sup> displays from row 25<sup>th</sup> to row 27<sup>th</sup>.

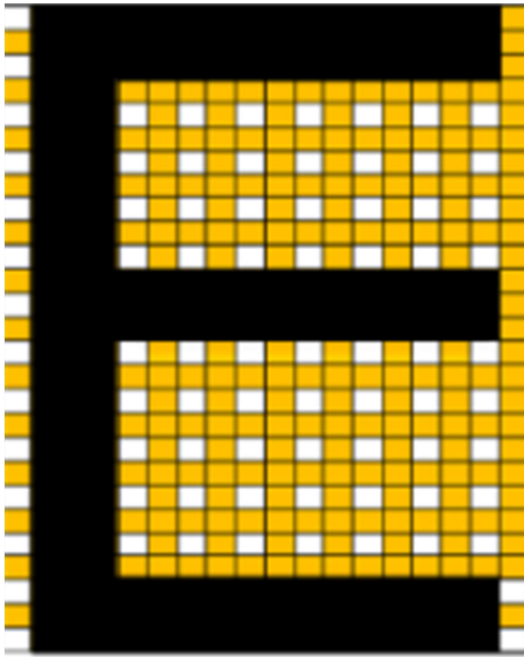


Figure 3. Description of the character E for compressing fonts.

This font compression process only takes 13 bytes to save the E font instead of 11250 bytes for binary fonts (300 x 300/8) and 270000 bytes for bitmap fonts. Besides, for bitmap and binary fonts, it is necessary to recalculate the height of each column to pass to the vertical reconstruction function while compressed fonts only take calculated parameters. The built-in transmission to this function leads to a significantly increased decoding speed.

### 2.1.2. Character Encoding

Currently, the popular fonts applied in the system with embedded between ARM microcontroller and FPGA are ASCII codes. In order to implement the recommended compression solution for fonts, it is necessary to analyze ASCII code. Specifically, based on ASCII code, we find that Latin characters have solid lines and some discrete lines. Therefore, we can choose the compression method with statistics adjacent pixels to perform the compression for each character in text.

In order to perform compression for each character, pixels of an image character are divided into column and row groups and then they are worked out using the process of encoding characters. This method 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 and the procedure for this process is described in Algorithm 1. In particular, the compression method will be performed by creating a compressed font table with the largest font size. Assume that a much smaller font ( $\beta$  times) is used in this case and just divided into all the values of the encrypted font by  $\beta$  times for decoding. In particular, to compress any character, the character image is divided into pixels in length and width to equal to half the font size of an original character by following steps:

Step 1: Take all elements of the original data divided by 2

Step 2: Decode the font according to the new data table obtained in step 1, and then get a reduced-size character.

Algorithm 1: The character encoding algorithm

Input:

- 1: Initial variables k, m, n=0, sobyte=0
- 2: Read bitmap file of character for information of high, long, width, byte on one column and content
- 3: if n < width then
- 4: k=0
- 5: if m < sobyte on 1 column then
- 6: if m<sup>th</sup> byte of column n=m<sup>th</sup> byte of column n+1 then
- 7: k=k+1
- 8: m=m+1
- 9: else m=m+1
- 10: else if k < sobyte on 1 column then
- 11: Store value n (column 1)
- 12: Store the begin location, bit 1 of column n
- 13: Store the end location, bit 1 of column n Add 3 for sobyte
- 14: n=n+1
- 15: else n=n+1
- 16: end if
- 17: Output:
- 18: The data string of the encrypted character

### 2.2. Character Decoding

After the character image has been encoded, the decoding of the character is performed by returning the encoding sequence and then rebuilding the character's frame. In particular, one 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 and it is similar to the previous column until obtaining a new value. This algorithm, called Algorithm 2, is described as follows:

Algorithm 2: The character decoding algorithm

Input:

- 1: Initial and assign variables "size", n=0
- 2: Divide bytes of compressed character codes to decode for "size" to change font size
- 3: if n < total byte of character compressed data then
- 4: if Data byte n+3 subtract n<sup>th</sup> data byte >1 then
- 5: Plot rectangular with 4 peaks: (n, n+1), (n, n+2), (n+3, n+1), (n+3, n+2)
- 6: n=n+3
- 7: Else
- 8: Plot straight line across 2 points: (n, n+1) and (n, n+2)
- 9: n=n+3
- 10: end if
- 11: Output:
- 12: Decoded characters

Algorithm 2 shows that a character decoding is processed based on dividing the pixels of that character after decoding. It means that this algorithm allows decoding the characters and performing texts for display or transfer of data to other modules. After decoding, the character capacity is reduced for storing and processing faster. Moreover, character

capacity reducing after the compression algorithm depends on the character type and font.

### 2.3. Compression Rate

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

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

$$CR = \frac{S}{CS} \quad (5)$$

in which  $S$  is the original character image size and  $CS$  denoting the character capacity after the compression in byte.

$$S = \frac{h \times w \times b}{8} \quad (6)$$

where  $h$  is the number of vertical pixels and  $w$  denotes the number of horizontal pixels,  $b$  describes the gray bit in the image. Therefore, the compression ratio  $SCR$  is calculated using the following formula:

$$SCR = \frac{S - CS}{S} \times 100\% \quad (7)$$

With the proposed compression algorithm, compression ratio of different characters can be different depending on the complexity of the image character. In addition, it can relate to the suitability of each design in a microcontroller-FPGA system when performing interface between different components or devices.

## 3. Results and Discussion

In this study, the pixels of any image character are divided into columns and rows by groups of pixels and then the compression process is performed by coding. The compression performance will depend on the different font type and give different size after compression.

### 3.1. Experiments of Character Encoding

In this article, from pixel groups of an image character in columns and rows, the character is compressed by encoding as described in Algorithm 1. Moreover, the compression of the character depends on the type of fonts. In particular, different fonts after compression will produce the different capacity. It means that the storage space of the compressed character is different.

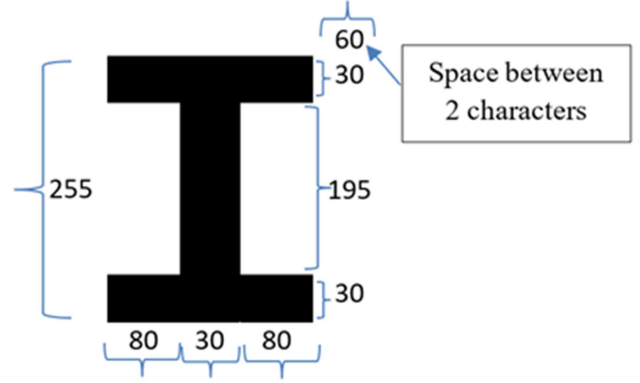


Figure 4. Division of the pixel groups of the character image “I” for encoding.

I-character compression: Assume that “I” character is compressed using the proposed compression algorithm, the first step is that pixels of the binary image need to be rearranged in row and column as shown in Figure 4.

Figure 4 shows the “I” character shape for encoding using the character compression algorithm. In particular, the character “I” is divided into groups of pixels in row and column such as 255, 30, 80, 195, 60. After dividing, the pixel groups are arranged in sequence for each group for encoding and then the compression process of the “I” character is performed using Algorithm 1.

The character compression algorithm of “I” is worked out by compressing the sequence of data according to the process as shown in Algorithm 1, in which pixel groups of columns and rows are allocated such as 250, 0, 0, 29, 0, 225, 254, 80, 0, 254, 110, 0, 29, 110, 225, 254, 190, 0, 0. Therefore, when encoding the character “I”, the capacity of the compressed character is 18 bytes and 1 byte is obtained for applying the total number of columns. In addition, after compressing the “I” character, the capacity of this character is 19 bytes using the proposed algorithm. In particular, the “I” character is encoded as follows:

250: the column number of the character “I” is  $(80+30+80+60=250)$ .

0, 0, 29: it means that the 0<sup>th</sup> column is displayed from pixel 0 to 29.

0, 225, 254: the 0<sup>th</sup> column is displayed from pixel 225 to 254.

80, 0, 254: the 80<sup>th</sup> column is displayed from pixel 0 to 254.

110, 0, 29: it means that the 110<sup>th</sup> column is displayed from pixel 0 to 29.

110, 225, 254: similarly the 110<sup>th</sup> column is displayed from pixel 225 to 254.

190, 0, 0: the 190<sup>th</sup> column is nothing to display.

The result is that the storage space rate of the character “I” after compressing is saved about 99.76%.

L-character compression: When performing “L” character compression, pixels are arranged as shown in Figure 5, in which the character “L” is divided into pixel groups of 255, 30, 160, 225, 60 for encoding.



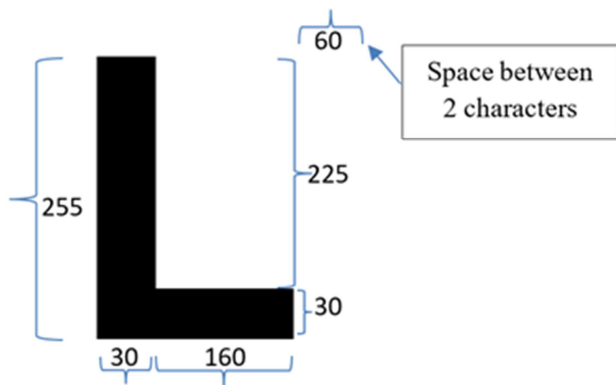


Figure 5. Division of the pixel groups of the character image "L" for encoding.

Encoding for the "L" character compression is similarly performed by dividing the image pixels into groups in rows and columns as follows: 250, 0, 0, 254, 30, 225, 254, 190, 0, 0. This encoding allows the "L" character to be compressed into 10 bytes, in which 9 bytes of data and 1 byte for determining the total number of columns. The encoding process is described as follows:

250, 0, 0: the total number of columns of the "L" character ( $30+160+60=250$ ).

0, 0, 254: the 0<sup>th</sup> column is displayed from pixel 0 to 254.

30, 225, 254: the 30<sup>th</sup> column is displayed from pixel 225 to 254.

160, 0, 0: the 160<sup>th</sup> column is nothing to display.

The result of the storage space rate of the character "L" after compressing is about 99.87%.

F-character compression: In similarity, in Figure 6, F-character compression with dividing pixel groups is performed as follows:

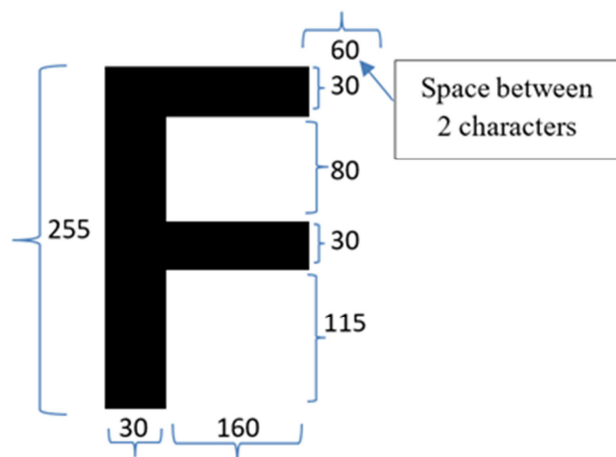


Figure 6. Division of the pixel groups of the character image "F" for encoding.

The process of the "F" character compression is consists of 13 bytes, in which:

250: the total number of columns of the F character ( $30+160+ 60=250$ ).

0, 0, 254: the 0<sup>th</sup> column is displayed from pixel 0 to 254.

30, 0, 29: the 30<sup>th</sup> column is displayed from pixel 0 to 29.  
30, 110, 139: the 30<sup>th</sup> column is displayed from pixel 110 to 139.

160, 0, 0: the 160<sup>th</sup> column is nothing to display.

The result is that the storage space rate of the character "F" after compressing is about 99.84%.

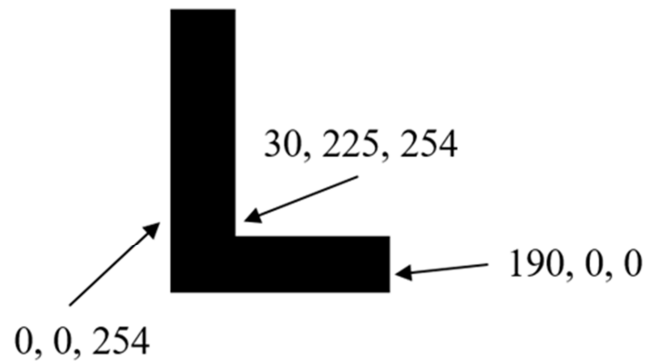


Figure 7. Representation of the "L" character after compression.

Figure 7 shows the "L" character is compressed and its capacity is reduced haft in size using the proposed encoding algorithm based on the pixel groups of rows and columns re-arranged.

Table 1 describes the compression results for different characters in detail. For this compression algorithm, 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 3.11, while the space 'character has the highest compression ratio of 2656.33.

Table 1. Results of character images after compression.

Character	S (bytes)	CS (bytes)	CR	SCR (%)
B	9563	1069	8.95	88.82
E	9244	17	543.76	99.82
F	7969	13	613	99.84
H	7969	13	613	99.83
I	7969	19	419.42	99.76
L	7969	10	796.9	99.87
@	7969	2566	3.11	67.8
Space	7969	3	2656.33	99.96

### 3.2. Experiments of Character Decoding

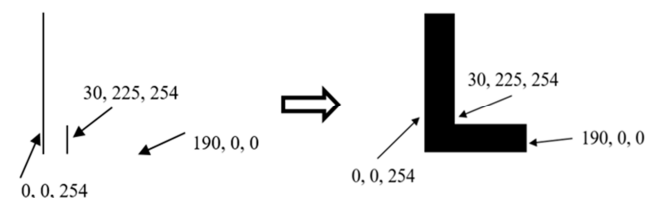


Figure 8. Representation of decompressing L character.

Figure 8 shows the process of decoding the "L" character. In particular, the "L" character after encoding was decoded with the sorted pixels in sequences such as 125, 0, 0, 127, 113,

127, 95, 0, 0. In similarity, characters of “I”, “F”, and other ones when decoded can produce different pixel sequences. The compression of image characters is to reduce the storage capacity of memory for quickly editing texts and printing on products.

### 3.3. Application of Character Compression

In our application, one printing system embedded between one ARM microcontroller and one FPGA for printing information on products, processing character compression and transferring from the microcontroller to the FPGA are described as follows:

Data transfer is duplex because both devices must always transfer data to each other at the same time (the microcontroller transfers data to the FPGA for printing. In addition, the FPGA transfers to the microcontroller operating system parameters such as: temperature, time, number of products and others)

High speed data transfer to be able to respond well and do not create time delay.

With the above requirements, the project chooses SPI transmission standard as a method of communication between the microcontroller and FPGA due to relation in a high-speed synchronous duplex transmission standard that can be up to 30Mb. In addition, this application selected the UART standard as an auxiliary communication gateway for the two boards to make speed-up of transmission through two independent channels. Figure 9 and Figure 10 describe two circuit boards containing the two independent channels for data transmission.

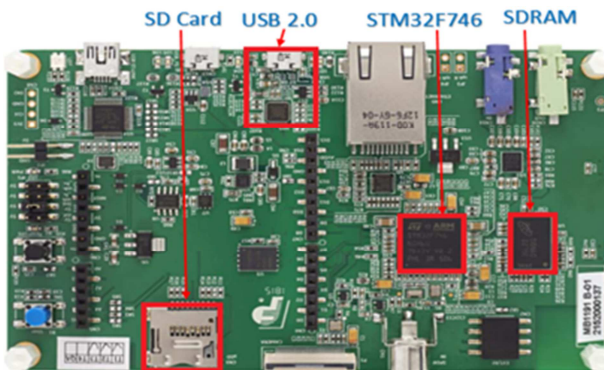


Figure 9. ARM microcontroller.

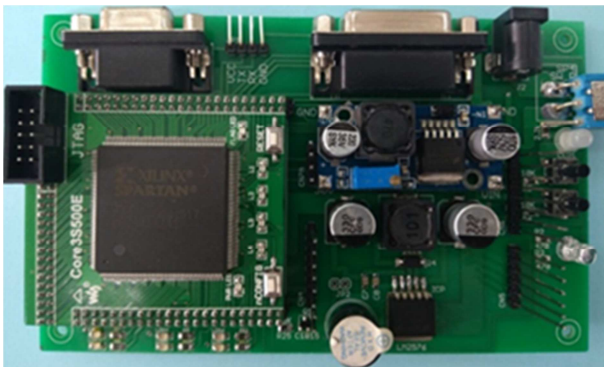


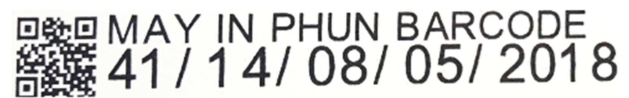
Figure 10. FPGA system.



(a) Printing installation at 300 dpi resolution.



(b) Printing logo and text.



(c) Printing QR code, expiry date and text.

Figure 11. Representation of information printed on products.

In this application, the microcontroller and FPGA play a main role in the digital printer system for printing product information in the industry. Moreover, the interface design and character image compression processing are optimized for transferring faster and controlling in real-time. In practice, The printing system using the microcontroller and FPGA was completely designed and printed on couch paper as shown in Figure 11, in which product information were printed such as logo, text with over 20 characters, QR code and others.

In practice, text fonts with bitmap format are applied in almost all microcontroller systems for editing and displaying contents. Using the bitmap fonts has advantages and disadvantages as follows:

Advantages: Simple, easy to use and fast decoding.

Disadvantages: Large memory capacity is required 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 be large or small in each font size, so users have to create the own font code table. This will consume memory resources of the system and also spend 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 compression algorithm and the practical results for compressing the characters “F”, “I” and “L” to be

able to print on the packaging products, the usefulness of this algorithm through the compression ratio is very high. In addition, this character image compression algorithm illustrates that the ARM microcontroller systems connected to FPGA for fast editing and printing product information such as logos, barcodes, QR and texts.

## 4. Conclusion

The proposed compression algorithm in this article was applied for ASCII font to reduce the size of image character for saving memory capacity during the control system operation. In particular, the compression process of one binary image character was performed by encoding, in which the character was calculated to divide the binary image character into pixel groups in columns and rows. Thus, the decoding process of the character was based on the encoding sequence to re-build the character frame as the original character. Therefore, the character compression algorithm reduced the character capacity for storing memory spaces with the high compression ratio. This compression algorithm was applied for the microcontroller-FPGA system for printing information on products with the high speed in real time.

## Acknowledgements

This work is supported by Ho Chi Minh City University of Technology and Education (HCMUTE) under Grant No. T2019-48TD.

## References

- [1] Aurelle Tchagna Kouanou, Daniel Tchiotsop, Theophile Fonzin Fozin, Bayangmbe Mounmo, René Tchinda, "Real-Time Image Compression System Using an Embedded Board," *Science Journal of Circuits, Systems and Signal Processing*, vol. 7, no. 4, pp. 81-86, 2018.
- [2] Ikerionwu Charles, Isonkobong Christopher Udousoro, "The Application of Selective Image Compression Techniques," *Software Engineering*, vol. 6, no. 4, pp. 116-120, 2018.
- [3] Ruchita K. Ingole, "Embedded Image Compression: A Review," *International Journal of Data Science and Analysis*, vol. 3, no. 1, pp. 1-4, 2017.
- [4] Syed Muhammad Arsalan Bashir, "Font Acknowledgment and Character Extraction of Digital and Scanned Images," *International Journal of Computer Applications*, vol. 70, no. 8, pp. 1-10, 2013.
- [5] Jahanzeb Ahmad, Mansoor Ebrahim, "FPGA based implementation of Baseline JPEG decoder," *International Journal of Electrical & Computer Sciences IJECS*, vol. 9, no. 9, pp. 371-377, 2009.
- [6] Yeli Li, Likun Lu, Binbin Yan, "The design and implementation of high-speed data interface based on Ink-jet printing system," in *Proceedings of the 2015 International Symposium on Computers & Informatics*, pp. 1725- 1732, 2015.
- [7] M. Akil, L. Perroton, T. Grandpierre, "FPGA-based architecture for hardware compression/decompression of wide format images," *Journal of Real-Time Image Processing*, vol. 1, pp. 163-170, 2006.
- [8] S. Banerjee and A. Kuchibhotla, "Real-time optimal-memory image rotation for embedded systems," *16th IEEE International Conference on Image Processing (ICIP)*, pp. 3277-3280, 2009.
- [9] Samrin Shaikh, Shashank Pujari, "Migration from microcontroller to FPGA based SoPC design: Case study: LMS adaptive filter design on Xilinx Zynq FPGA with embedded ARM controller," *International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT)*, pp. 129-134, 2016.
- [10] Paulo Garcia, Deepayan Bhowmik, Robert Stewart, Greg Michaelson and Andrew Wallace, "Optimized Memory Allocation and Power Minimization for FPGA-Based Image Processing," *Journal of Imaging*, vol. 5, no. 7, pp. 27-49, 2019.
- [11] 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)*, IEEE 26<sup>th</sup> International, pp. 449-453, 2012.
- [12] 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)*, IEEE 23<sup>rd</sup> Annual International Symposium on, pp. 52-59, 2015.
- [13] X. Zhou, Y. Ito, and K. Nakano, "An efficient implementation of LZW decompression in the FPGA," in *Parallel and Distributed Processing Symposium Workshops*, IEEE International, pp. 599-607, 2016.
- [14] T. Mitra and T.-c. Chiueh, "An FPGA implementation of triangle mesh decompression," *Proceedings 10<sup>th</sup> Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, CA, USA, pp. 22-31, 2002.
- [15] 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.
- [16] P. M. Sandeep and C. S. Manikandababu, "Compression and decompression of FPGA bitstreams," *International Conference on Computer Communication and Informatics*, Coimbatore, pp. 1-4, 2013.
- [17] P. Hemnath and V. Prabhu, "Compression of FPGA bitstreams using improved RLE algorithm," in *Information Communication and Embedded Systems (ICICES)*, International Conference on, pp. 834-839, 2013.
- [18] Z. Li and S. Hauck, "Configuration compression for virtex FPGAs," in *Field-Programmable Custom Computing Machines*, The 9<sup>th</sup> Annual IEEE Symposium on, pp. 147-159, 2001.
- [19] S. Rigler, "FPGA-Based Lossless Data Compression Using GNU Zip," *University of Waterloo*, 2007.
- [20] I. Suarjaya, "A new algorithm for data compression optimization," *International Journal of Advanced Computer Science and Applications*, vol. 3, no. 8, pp. 14-17, 2012.
- [21] M. Azad, A. Kalam, R. Sharmeen, S. Ahmad, and S. Kamruzzaman, "An efficient technique for text compression," *The 1<sup>st</sup> International Conference on Information Management and Business (IMB2005)*, pp. 467-473, 2010.



- [22] R. Gallager, "Variations on a theme by Huffman," IEEE Transactions on Information Theory, vol. 24, no. 6, pp. 668-674, 1978.
- [23] N. Faller, "An adaptive system for data compression," in Record of the 7<sup>th</sup> Asilomar Conference on Circuits, Systems and Computers, pp. 593-597, 1973.
- [24] D. E. Knuth, "Dynamic huffman coding," Journal of algorithms, vol. 6, no. 2, pp. 163-180, 1985.
- [25] J. S. Vitter, "Design and analysis of dynamic Huffman codes," Journal of the ACM (JACM), vol. 34, no. 4, pp. 825-845, 1987.
- [26] N. Ganvir, A. Jadhav, and P. Scoe, "Explore the Performance of the ARM Processor Using JPEG," International Journal on Computer Science and Engineering, vol. 2, no. 1, pp. 12-17, 2010.
- [27] C. Lefurgy, P. Bird, I.-C. Chen, and T. Mudge, "Improving code density using compression techniques," in Proceedings of the 30<sup>th</sup> Annual international symposium on Microarchitecture ACM/IEEE, pp. 194-203, 1997.