# RLE METHODS FOR BINARY-IMAGE ENCODING

**Michal Heczko**
Engineering Informatics, 1st class, full-time study
Supervisor: prof. Ing. Karel Vlček, CSc.

Faculty of Applied Informatics
Tomas Bata University in Zlín
Nad Stráněmi 4511
760 05 Zlín
Czech Republic
heczko@fai.utb.cz

**Abstract:** This work summarizes the basic knowledge about RLE methods for binary-image encoding. These methods are usually applied in the facsimile systems and the ISDN systems. In the first part of this document, there are described the most commonly used one-dimensional and two-dimensional RLE methods. The second part of this work is the demonstration of the possibilities of selected methods implementation by VHDL programming language and FPGA APA075.

**Keywords:** RLE methods, binary image, VHDL, FPGA, APA075

## 1 Introduction

Text and graphical binary images are used nowadays especially in the area of scanners and facsimile systems. Textual information, which is transferred by these systems, usually keeps its information value in monochromatic picture too. This is the reason why the monochromatic data interpretation is used. The main advantage is the smaller size of transferred data.

Generally there is used RLE or arithmetic code for data encoding, which was described. Arithmetic encoding is very effective and it is possible to meet it for example in the JBIG standard (Joint Bi-level Image Experts Group). But RLE has some advantages too. The first advantage is simple implementation and the second advantage is the relatively high speed of RLE coder. Its main disadvantage could be lower compression degree.

It is suitable to realize the implementation of these encoders by the FPGA circuits (Field Programmable Gate Array). These circuits were selected because of the fact, that it is universal solution, which provides low acquisition price advantages and short design-time. Because of these reasons, this document is attended to realization of the encoders with FPGA circuits.

## 2 Run Length Encoding

As it was mentioned before, this work is devoted to the RLE encoding topic and its implementation. This encoding registers the number of the repeated symbols in the monochromatic sections of the line (or two or more lines). The fact, which results from it, is that with the increasing size of the monochromatic areas, the compression degree increases too (the highest compression degree will have the picture, which has only one color). By contrast, the compression degree decreases with the fragmentariness of the image (the lowest compression degree will have the image, which is formed by the “chequer”, in which the short white and black section are periodically varied).

RLE could be divided into two main groups. The one-dimensional RL codes will be in the first group. The second group will be formed by two-dimensional RL codes. Two-dimensional codes include the previous-line values to the calculation.

## 2.1 One-dimensional RL codes

In the first group of RL codes are codes, in which every line is encoded separately. The relationship between actual and previous line is not considered. This coder is the simplest, but they provide very low compression ratio. These four encoding methods could be placed to these groups:

- Linear code
- Logarithmic code
- Shortened Huffman code
- Optimal Huffman code

### 2.1.1 Linear code

In [1] the linear codes are defined: Let $m_k$ is the array of line-section lengths with its output probabilities $p_k$ for $k$ = 1, 2, 3… N and let the number of code-word bits for the length $m_k$ is $b_k$. In the linear codes holds the fact, that if $k$ is increasing linearly, $b_k$ is increasing linearly too.

In the linear codes, there is the output code-word length specified and because of the black and white section changing, the code-word includes only the section length. It is not necessary to include the colour of the section.

Linear codes are often marked as the A-codes and they are used for meteorological maps, plans, and diagrams encoding.

### 2.1.2 Logarithmic code

In [1] the logarithmic codes are defined: Let $m_k$ is the array of line-section lengths with its output probabilities $p_k$ for $k$ = 1, 2, 3… N and let the number of code-word bits for the length $m_k$ is $b_k$. In the logarithmic codes holds the fact, that the relation between $k$ and $b_k$ is approximately logarithmical.

In the following table (table 1.), there is the construction of logarithmical code summarized. In the first column, there are the lengths of the monochrome sector and in the second column, there are the code-words. Contrary of the linear codes, there is necessary to include the colour of the section. The colour is implied by the letter “C” in the table. If the section is white, the number “0” replaces “C” and if the section is black, the number “1” replaces “C”.

As well as the linear code, the logarithmic code has the abbreviation too. In this case it is marked as “B-code”. This encoding type is usually used for text messages encoding.

*Table 1: Logarithmic RL code (first 9 values).*

| Length | Code word | Length | Code word |
|---|---|---|---|
| 1 | `C0` | 6 | `C1C1` |
| 2 | `C1` | 7 | `C0C0C0` |
| 3 | `C0C0` | 8 | `C0C0C1` |
| 4 | `C0C1` | 9 | `C0C1C0` |
| 5 | `C1C0` | … | ... |

### 2.1.3 Shortened Huffman Code

The encoding by shortened Huffman code uses optimal Huffman code for its function (it was described for example in [5] and [6]). This encoding is defined for the white sections with length up to 47 and the black sections with the length up to 15. There are following rules for longer sections:

- In the case of white section, which is longer than 47, 3bit prefix "010" is used. After the prefix it is added the 11bit code word with the section length in linear code.
- In the case of black section, which is longer than 15, 7bit prefix "1101000" is used. After the prefix it is added the 11bit code word with the section length in linear code.

### 2.1.4 Modified Huffman Code

Modified Huffman code is usually used in the multimedia systems and its abbreviation is "MH Code". This code is defined for two groups of lengths of monochrome line-section.

In the first group, there are the sections with lengths from 0 to 63. These sections are encoded by optimal Huffman code, which is defined here by the termination codes table. ([1] or [2]).

The second group is formed by line sections, whose length is equal to the multiple of 64. There are defined supplementary codes ([1] or [2]) for these line-sections. These codes are defined for length up to 2560.

In practise, the line sections are not formed only by the length up to 63 or only the lengths, which are equal to the multiple of 64. In this case it is necessary to resolve the value to the multiple of 64 and the remnant. These two parts are encoded by the supplementary code and terminate code separately. Consequent two code-words are joined to one code-word as it is shown in table 2.

*Table 2: Example of MH Code.*

| | |
|---:|:---:|
| Section length | `1226 (white section)` |
| Dissolution | `19 * 64 + 10 = 1216 + 10` |
| Supplementary code | `1216 -> "011011000"` |
| Terminate code | `10 -> "00111"` |
| Final code for given section length | `01101100000111` |

## 2.2 Two-dimensional RL codes

Two-dimensional RL codes form the second big group. In these codes, there is reflected the relation between every two consecutive lines.

Two encoding-access types of encoding access fall in this category:

- Prediction encoding
- Border picture-elements encoding

### 2.2.1 Prediction encoding

Prediction encoding could be divided into two parts: one-dimension prediction and two-dimension prediction.

In the case of one-dimension encoding, there is implemented the exclusive logical product (XOR) of two correspondent elements in the consequent lines. The results of this operation are encoded (for example by linear RLE).

The same operation will be used for the decoding of the data.

Two-dimensional prediction does not include only two correspondent elements in the consequent lines, but it includes nearest elements in these lines. It is reflected totally *N* nearest elements.

Two-dimensional prediction could be divided into two steps:

- Error-vector generating – in this part, the element X is predicted. This element is compared to the real element (XOR operation). At the output of this operation is error vector.
- Error-vector encoding – error vector from previous step is encoded by one of the one-dimensional methods (for example Linear RLE).

The error-vector values are taken by statistic model. The derivation of these models could be found in [1].

The statistic models are defined by the way of fiction or table. For the written text in English, it is usually used following formula (for N = 3):

$$\hat{X} = X_0 X_1 + X_1 + X_2 X_0$$

The error vector is counted by way of the following formula:

$$E = X \oplus \hat{X}$$

*$X_0$*, *$X_1$*, *$X_2$* and *$X$* are defined as it is shown at picture 1.

*Figure 1: Context of the two dimensional prediction encoding.*

| | | | |
|---|---|---|---|
| *Previous line* | $X_4$ | $X_3$ | $X_2$ |
| *Actual line* | $X_1$ | $X_0$ | **X** |

### 2.2.2 Border picture-elements encoding

The basic idea of all method from this group is the fact that the entire information about binary image is stored in the border picture-elements. In these elements the colour change happens.

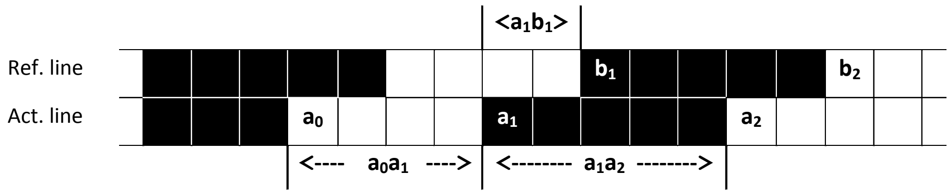
Following encoding methods fall in these groups:

- Relative address Coding (RAC) – border picture-elements are encoded in face of themselves.
- Edge Different Coding (EDIC) – border picture-elements are coupled to pairs. Different types of pairs are encoded.
- Relative Elements Address Designate (READ) – it is using the adaptive selection of picture elements analogous as the RAC method; they are grouped analogous as in the EDIC method.
- Modified Modified READ (MMR) – the most widely used modification of READ method. It is usually used in multimedia system. This method is described in the following text.

In the MMR method (as well as the READ method), there are defined three encoding modes:

- Pass mode – $b_1$ and $b_2$ elements are between the $a_0$ and $a_1$ elements in the reference line ($b_2$ is not lie over $a_1$).
- Horizontal mode – mode, for which holds the fact that $|a_0a_1| \leq |a_1b_1|$. Distances $a_0a_1$ and $a_1a_2$ are transmitted as the encoding intervals.
- Vertical mode – mode, for which holds the fact that $|a_0a_1| > |a_1b_1|$. Distance $a_1b_1$ is transmitted as the encoding interval. It is defined for $| a_1b_1| \leq 3$.

*Figure 2: Horizontal and vertical mode in READ and in MMR encoding methods.*



Border picture-elements in the READ and in the MMR (Figure 2.) methods have the following meaning:

- $a_0$ … Starting picture element in actual line. It is the reference element.
- $a_1$ … Border picture-element follows the $a_0$. It lies on the right from $a_0$.
- $a_2$ … Border picture-element follows the $a_1$. It lies on the right from $a_1$.
- $b_1$ … The first border picture-element in the reference line, whose colour is different from the colour of $a_0$.
- $b_2$ … Border picture-element following the $b_1$. It lies on the right from $b_1$.

*Table 3: MMR encoding method.*

| **Mode** | **Encoded picture elements** | | **Write in** | **Code** |
|---|---|---|---|---|
| Pass | $b_1b_2$ | | P | 0001 |
| Horizontal | $a_0a_1$ | | $H(a_0a_1, a_1a_2)$ | 001 + $M(a_0a_1)$ + $M(a_1a_2)$ |
| Vertical | $a_1$ pod $b_1$ | $a_1b_1 = 0$ | V(0) | 1 |
| | $a_1$ on the left of $b_1$ | $a_1b_1 = 1$ | $V_R(a_1b_1) = 1$ | 011 |
| | | $a_1b_1 = 2$ | $V_R(a_1b_1) = 2$ | 000011 |
| | | $a_1b_1 = 3$ | $V_R(a_1b_1) = 3$ | 0000011 |
| | $a_1$ on the right of $b_1$ | $a_1b_1 = 1$ | $V_L(a_1b_1) = 1$ | 010 |
| | | $a_1b_1 = 2$ | $V_L(a_1b_1) = 2$ | 000010 |
| | | $a_1b_1 = 3$ | $V_L(a_1b_1) = 3$ | 0000010 |

The MMR method is usually used in facsimile systems (where is marked as the G4 group) and In telematics services (ISDN).

# 3 Implementation of RLE at the FPGA

For shortness and clearness this chapter contains only the examples of some encoding methods, which was described in previous chapters. Every example consists of the VHDL code (the code is not complete, because the full code is too long).

The application works by following process:

- Data from the input are saved to the vector in memory.
- Data from the memory are processed into the output vector in the memory by optimal method (for example linear RLE, logarithmic RLE …).
- Data from the output vector are sent to the output.

## 3.1 Linear RLE

In the first step, the data from the vector *y* (in this vector data from the input are stored) are processed into the vector with information about the number of repeating:

```
for i in 0 to ROWS-1 loop
  prev := y(i*COLS);
  count := "00000001";
  for j in 1 to COLS-1 loop
    act := y((i*COLS)+j);
    if (j < COLS-1) then
      if (act = prev) then
        if (count = "11111111") then
          count := "00000000";
        else
          count := count + 1;
        end if;
        prev := act;
      else
        data_cnt(ptr) := count;
        data_col(ptr) := prev;
        count := "00000001";
        ptr := ptr + 1;
        prev := act;
      end if;
    else
      if (count = "11111111") then
        count := "00000000";
      else
```

```
        count := count + 1;
      end if;
      data_cnt(ptr) := count;
      data_col(ptr) := prev;
      count := "00000000";
      ptr := ptr + 1;
    end if;
  end loop;
end loop;
```

In the second part of this program, the numbers of repeating are sent to the output in binary code:

```
if (clk'event and (clk='1')) then
  if (x2 >= 0) then
    v_out := data_cnt(x);
    Q <= v_out(x2);
    x2 := x2 - 1;
  else
    x2 := 7;
    x := x + 1;
  end if;
end if;
```

## 3.2 Logarithmic RLE

First part is the same as in the linear RLE. In the next step, there is necessary to make a codeword, as it is shown in the table 1. These data are stored in the vector *tmp_ram* (lines 2 – 21) and in the next step they are sent to the output vector (lines 22 – 28):

```
1.  for i in 0 to ptr-1 loop
2.    for j in 1 to data_cnt(ptr) loop
3.      pp := 0;
4.      px := (2 ** i);
5.      for  k in x-2 to data_cnt(ptr) loop
6.        if (pp mod x) < ((2 ** i) / 2) then
7.          if j = data_cnt(ptr) then
8.            tmp_ram(pt) := data_col(ptr);
9.            tmp_ram(pt+1) := '0';
10.           pt := pt+2;
11.         end if;
12.       else
13.         if j = data_cnt(ptr) then
14.           tmp_ram(pt) := data_col(ptr);
15.           tmp_ram(pt+1) := '1';
16.           pt := pt+2;
17.         end if;
18.       end if;
19.       pp := pp + 1;
20.     end loop;
21.   end loop;
22.   while pt > 0 loop
23.   v_out(ptr2) := tmp_ram(pt-1);
24.   v_out(ptr2+1) := tmp_ram(pt);
25.   pt := pt-2;
26.   ptr2 := ptr2 + 2;
27.   end loop;
28. end loop;
```

## 3.3 Modified Huffman code

Data are stored in the array like in the previous methods. Every item includes a color and a number of repeating. In the following step, each item is tested by this code (The code is simplified, because of its length. Following example describes the most important part of the program; other parts are similar to the previous methods.

```
for i in 0 to ptr-1 loop
  -- dissolution
  ter :=  data_cnt(i) mod 64;
  sup :=  (data_cnt(i) - ter);
  if (data_col(i) = '0') then
    -- white
    -- supplementary code
    if sup = 64 then
        d_out(ptr2) := '1';
        d_out(ptr2+1) := '1';
        d_out(ptr2+2) := '0';
        d_out(ptr2+3) := '1';
        d_out(ptr2+4) := '1';
        ptr2 := ptr2 + 8;
    elsif sup = 128 then
        … … …
    end if;
    -- terminating code
    if ter = 0 then
        d_out(ptr2) := '0';
        d_out(ptr2+1) := '0';
        d_out(ptr2+2) := '1';
        d_out(ptr2+3) := '1';
        d_out(ptr2+4) := '0';
        d_out(ptr2+5) := '1';
        d_out(ptr2+6) := '0';
        d_out(ptr2+7) := '1';
        ptr2 := ptr2 + 8;
    elsif ter = 1 then
        … … …
    end if;
  else
    -- black
    … … …
  end if;
end loop;
```

Data from the vector *d_out* are sent to the output.

### 3.4 Prediction encoding

The procedure is similar to the previous method. But there is one change. At the beginning the error vector is generating. This error vector is consequently encoded by the linear RLE.

```
-- first line;
for j in 0 to COLS-1 loop
   e(j) := y(j);
end loop;
-- next lines
for i in 1 to ROWS-1 loop
   for j in 0 to COLS-1 loop
     e((i*COLS)+j) :=
y((i*(COLS-1))+j) xor
y((i*COLS)+j);
   end loop;
end loop;
```

## 4 Conclusion

The aim of this work was summarising of basic facts about RLE methods for binary-image encoding. This encoding is usually used in the area of scanners, facsimile systems and the ISDN.

The FPGA ProASIC APA075 was chosen for the implementation of coders. This FPGA was made by the ACTEL. These arrays provides universal solution and fast and easy implementation. Other advantages are for example high speed and relatively low acquisition price.

## Acknowledgment

This work was originally made as a preparation for the exam from the “Multimedial data processing”, which is a part of my PHD study curriculum in the “Engineering informatics” study program at the Faculty of Applied Informatics of the Thomas Bata University in Zlin.

I would like to say thanks to prof. Karel Vlček for the inspiration to processing this topic.

## References

[1] LEVICKÝ Dušan. Multimediálne telekomunikácie. 1. vydání. Košice: Elfa, s.r.o. 2002. 241 s. ISBN 80-89066-58-5

[2] MAI Chi Luong. Introduction to computer vision and image processing [online]. Department of Pattern Recognition and Knowledge Engineering, Institute of Information Technology, Hanoi, Vietnam. [cit. 2009-01-25]. Dostupný z WWW: <http://web.archive.org/web/20021031000324/www.netnam.vn/unescocourse/computervision/computer.htm>

[3] PINKER Jiří, POUPA Martin. Číslicové systémy a jazyk VHDL. 1. Vydání. Praha: BEN – Technická literatury 2006. 352 s. ISBN 80-7300-198-5

[4] ŠŤASTNÝ Jakub. Programovatelná hradlová pole. Automatizace, 2008, roč. 51. č. 1, s. 9 - 14.

[5] VLČEK Karel. Komprese a kódování dat v multimediálních komunikacích. 2. vydání. Praha: BEN – Technická literatura. 2004. 260 s. ISBN 80-7300-134-9

[6] ZELINKA Ivan. Základy informatiky. 1. Vydání. Zlín: Univerzita Tomáše Bati. 2005. 112 s. ISBN 80-214-1423-5