Journal of Theoretical and Applied Information Technology
10th December 2015. Vol.82. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AN EFFICIENT STATE METRIC MEMORY MANAGEMENT METHOD FOR FLEXIBLE VITERBI DECODER

[1]XU BANGJIAN, [2]LIU ZONGLIN, [3]YANG HUI, [4]CHENG LING
[1]Assoc Prof., [2]Assoc.,Prof., [3]Postgraduate, [4]Postgraduate
College of Computer, National University of Defense Technology, People's Republic of China
E-mail: [1]chengling2013@yeah.net

ABSTRACT

Flexible Viterbi decoder becomes extremely important as a result of increasingly modern wireless communication standards in SDR (Software Defined Radio) systems. In order to support multi-standard service and area saving, a flexible Viterbi decoder chip with cascaded ACS (Add Compare Select) unit is preferred. In such a decoder chip, there is a big irregular addressing problem for temporary ACS computing results storing. To solve this, a generalized efficient state metric memory management method has been developed. Analysis shows the design is highly flexible and efficient.

Keywords: *Viterbi Decoder, Multi-standard, Reconfigurable Architecture, Software Defined Radio (SDR), Cascaded ACS*

1. INTRODUCTION

In recent years, flexibility is required in modern communication system based on the concept of software defined radio (SDR) . Usually, SDR is composed of reconfigurable hardware which can be reprogrammed to adapt to various wireless protocols[1,2,3,4,5,6].

Convolutional channel decoding is such a significant part in modern SDR systems. It requires to work under different constraint length, code rate and polynomial. The Viterbi algorithm for decoding convolutional code is a maximum likelihood algorithm based on the coding trellis . In the paper [6], a flexible Viterbi decoder chip for convolutional channel coding is presented with the characteristic of efficiency and area saving, as it computes partial states at every stage. It can decode convolutional code with constraint length from 5 to 9, code rate 1/2, 1/3, 1/4 and user defined polynomials.

However, a generalized efficient cascaded ACS addressing algorithm suitable for such flexibility is still lacking. In this paper, such algorithm is presented. Such algorithms are very important during designing process of such decoder chip.

2. FLEXIBLE VITERBI DECODER ARCHITECTURE REVIEW

A flexible Viterbi decoder also consists of three parts [6]:

1)Branch Metric Calculation Unit (BMCU).
2)Add Compare Select Unit (ACSU).
3)Survival Path Management Unit (SPMU).

In the paper [6], a fast cascaded ACSU is used in order to provide the most important flexibility. The cascaded ACSU is shown in figure 1, where $M = 2^{K-1}$ and $(i \bmod 4) = 0$.

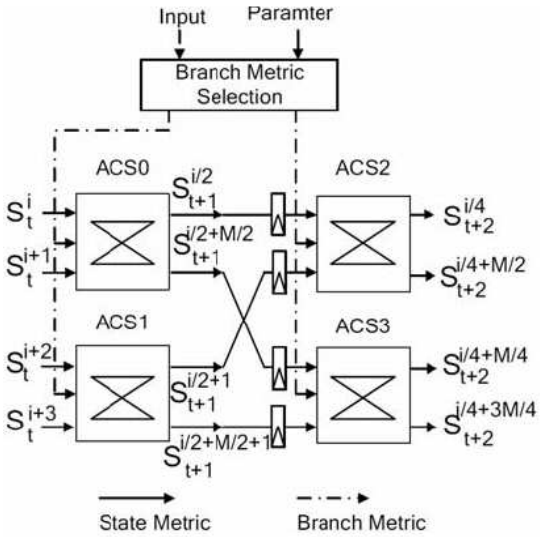

Figure 1: Diagram Of Cascaded ACSU

Journal of Theoretical and Applied Information Technology
10th December 2015. Vol.82. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

This structure updates 4 state metrics in two stages once, similar to Radix-4 Viterbi decoder [2]. The key idea also inherits from [7,8,9,10,11,12] to save chip area. However, the decoder is designed for dealing with convolutional code of constraint length from 5 to 9, code rate 1/2, 1/3, 1/4 and user defined polynomials. By inserting registers between stages, the decoder in [6] can also achieve a higher throughput.

Another important feature to provide flexibility is the state metric memory management subsystem without conflict as shown for $K=5$ in figure 2. With this feature it can read from / write to these banks in parallel [6].

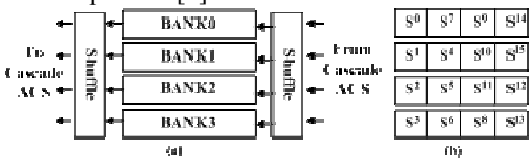

Figure 2: (A) 4 Bank State Metric Memory (B) Data Arrangement At K=5

3. FLEXIBLE VITERBI DECODER ARCHITECTURE REVIEW

The ACSU contains a cascaded ACS for butterfly computation with state metric memory and address generation subsystem to provide flexibility [6]. To simplify designing, 4 states are used at the same time in figure 1. In normal cases, this design is enough [3]. For different code parameters, the state metric computation is controlled by the address of read/write operation. It defines the sequence of states that are to be computed [6]. This is the state metric memory management problem in such flexible Viterbi decoder. The specific management method must ensure that memory access conflict does not exist.

3.1 State Metric Memory Management Method Review

Four banks of RAM are designed, and the in-place replace schedule is applied [4,6]. The kernel problem of state metric memory management is to find a suitable initial data arrangement map for 4 banks at stage 0. S^i and S^j are put into the same bank if the following inequations are both satisfied for each 2 recursive stages as in Figure 2 (2^{K-3} indicates the element number in each bank) [6].

$$\lfloor i/4 \rfloor \neq \lfloor j/4 \rfloor \text{ , and } \quad i \neq j \bmod 2^{K-3}$$

A method based on computer search for finding the proper final arrangement is claimed in [6]. To place state 0 to 255 in initial state metric memory properly, a feasible scheme satisfying above inequations is needed at each 2 recursive stages. The address generation part is implemented by circular shift of state id. It is shown [6] by Figure 3.

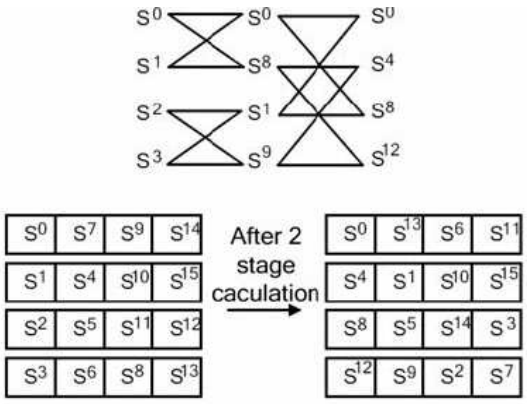

Figure 3: State Metric Address Generation Example

Detailed final memory management arrangement for all flexible cases is not given in [6]. But by analysis of next segment in this paper, the computer search method used in [6] is not satisfactory, as the number of search times is too huge. So it can't be assured to find a suitable result. This will lead to confusion in practical designing process.

3.2 Rules and Difficulties of Computer Search Method

To give the efficient method proposed by this paper, detailed computer search procedure should be discussed firstly. The input state id's in Figure 1 are given as follows:

$$i = (B_{K-2}B_{K-1}\cdots B_3B_2 00)_2 \quad (1)$$

$$i+1 = (B_{K-2}B_{K-1}\cdots B_3B_2 01)_2 \quad (2)$$

$$i+2 = (B_{K-2}B_{K-1}\cdots B_3B_2 10)_2 \quad (3)$$

$$i+3 = (B_{K-2}B_{K-1}\cdots B_3B_2 11)_2 \quad (4)$$

where B_k is a binary bit. Then the output state id's are given as follows:

$$i/4 = (00B_{K-2}\cdots B_3B_2)_2 = (i) >> 2 \quad (5)$$

$$i/4 + M/4 = (01B_{K-2}\cdots B_3B_2)_2 = (i+1) >> 2 \quad (6)$$

$$i/4 + M/2 = (10B_{K-2}\cdots B_3B_2)_2 = (i+2) >> 2 \quad (7)$$

$$i/4 + 3M/4 = (11B_{K-2}\cdots B_3B_2)_2 = (i+3) >> 2 \quad (8)$$

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

By applying in-place replace schedule, the result of stage $(t+2)$ will be written back to bank location of stage t given by Equation (9):

$$S_{t+2}^{i/4} \to S_t^i \quad , \quad S_{t+2}^{i/4+M/4} \to S_t^{i+1} \quad , \quad S_{t+2}^{i/4+M/2} \to S_t^{i+2} \quad , \quad S_{t+2}^{i/4+3M/4} \to S_t^{i+3} \qquad (9)$$

Theorem 1: When K is an odd number, after $\frac{(K-1)}{2}\times 2$ stages, the data arrangement map in memory banks at stage $t+(K-1)$ will be the same as the map at stage t .When K is an even number, after $(K-1)\times 2$ stages, the data arrangement map in memory banks at stage $t+2(K-1)$ will be the same as the map at stage t .

Prove: Since after every 2 stages the corresponding ouput state id is circular shifted from input id by 2 bits as (5)~(8), it is easy to obtain this conclusion.

Theorem 2: Per 4 consecutive states at arbitrary stage t (such as S_t^i , S_t^{i+1} , S_t^{i+2} , S_t^{i+3} , where $i \bmod 4 = 0$), should be placed in 4 different banks independently.

Prove: Since it is desired to parallel reading these 4 consecutive states from 4 banks , this conclusion is obvious.

Hence, the following rules can be concluded:

Rule 1: The initial data arrangement map for every bank at stage 0 should satisfy $\lfloor i/4 \rfloor \neq \lfloor j/4 \rfloor$, to assure that 4 consecutive data do not exist in one bank.

Rule 2: The initial data arrangement map for every bank at stage 0 should satisfy $i \neq j \bmod 2^{K-3}$, to assure that 4 consecutive data do not exist in one bank when parallel reading data from these 4 banks at stage 2.

Rule 3: When K is an odd number, the initial data arrangement map should be circular shifted right by 2 bits for $\left(\frac{K-1}{2}-1\right)$ times. Every time the new state id in each bank should also satisfy rule 1 and rule 2.

Rule 4: When K is an even number, the initial data arrangement map should be circular shifted right by 2 bits for $(K-2)$ times. Every time the new state id in each bank should also satisfy rule 1 and rule 2.

According to rule 1, in the initial data arrangement map, for all state id that can be put into the same bank, the previous $(K-3)$ bits of state id's binary code form a $2^{K-3}\times(K-3)$ dimension matrix as following:

$$\begin{bmatrix} 0&0&\cdots&\cdots&0&0&0&0 \\ 0&0&\cdots&\cdots&0&0&0&1 \\ 0&0&\cdots&\cdots&0&0&1&0 \\ 0&0&\cdots&\cdots&0&0&1&1 \\ 0&0&\cdots&\cdots&0&1&0&0 \\ 0&0&\cdots&\cdots&0&1&0&1 \\ \vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots \\ 1&1&1&1&1&1&1&1 \end{bmatrix} \qquad (10)$$

which equals to a matrix with decimal number given by equation (11):

$$[\,0\;\;1\;\;2\;\;3\;\;4\;\;5\;\;\cdots\;\;(2^{K-3}-1)\,]' \qquad (11)$$

For all state id that can be put into the same bank, the whole $(K-1)$ bits of state id's binary code form a $2^{(K-3)}\times(K-1)$ dimension matrix as following:

$$\begin{bmatrix} 0&0&\cdots&\cdots&0&0&0&0&c_{0,1}&c_{0,0} \\ 0&0&\cdots&\cdots&0&0&0&1&c_{1,1}&c_{1,0} \\ 0&0&\cdots&\cdots&0&0&1&0&c_{2,1}&c_{2,0} \\ 0&0&\cdots&\cdots&0&0&1&1&c_{3,1}&c_{3,0} \\ 0&0&\cdots&\cdots&0&1&0&0&c_{4,1}&c_{4,0} \\ 0&0&\cdots&\cdots&0&1&0&1&c_{5,1}&c_{5,0} \\ \vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots \\ 1&1&1&1&1&1&1&1&c_{(2^{K-3}-1),1}&c_{(2^{K-3}-1),0} \end{bmatrix} \qquad (12)$$

where $\{c_{m,1}, c_{m,0}\}$ is a pair of binary bits. It is easy to see that there are $256^{(2^{K-5})}$ possible combinations. For $K=9$, this number is 256^{16}. This number is huge. To reduce this number further, rule 3 and rule 4 can be applied to equation (12). After circular shift right by 2 bits, equation (13) is given :

$$\begin{bmatrix} c_{0,1}&c_{0,0}&0&0&\cdots&\cdots&0&0&0&0 \\ c_{1,1}&c_{1,0}&0&0&\cdots&\cdots&0&0&0&1 \\ c_{2,1}&c_{2,0}&0&0&\cdots&\cdots&0&0&1&0 \\ c_{3,1}&c_{3,0}&0&0&\cdots&\cdots&0&0&1&1 \\ c_{4,1}&c_{4,0}&0&0&\cdots&\cdots&0&1&0&0 \\ c_{5,1}&c_{5,0}&0&0&\cdots&\cdots&0&1&0&1 \\ \vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots \\ c_{(2^{K-3}-1),1}&c_{(2^{K-3}-1),0}&1&1&1&1&1&1&1&1 \end{bmatrix} \qquad (13)$$

So, now to obey rule 1, it is easy to see that when $\lfloor m/4 \rfloor = \lfloor n/4 \rfloor$ and $K>6$, there are $\{c_{m,1}, c_{m,0}\} \neq \{c_{n,1}, c_{n,0}\}$. This result will reduce the combination number to $24^{(2^{K-5})} = 331776^{(2^{K-7})}$. For $K=9$, this number is 331776^4. This number is about $1/10^{16}$ of 256^{16}.

To reduce this number further, rule 2 can also be applied to matrix (12). It is easy to see that when

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$m = n \bmod 2^{K-5}$ and $K > 6$, there are $\{c_{m,1}, c_{m,0}\} \neq \{c_{n,1}, c_{n,0}\}$. This result will reduce the combination number to

$$24^{(2^{K-7})} \times 18^{(2^{K-7})} \times 8^{(2^{K-7})} = 3456^{(2^{K-7})}$$

For $K = 9$, this number is 3456^4. This number is still too large for computer searching, but is about $1/10^8$ of 331776^4.

3.3 A New State Metric Memory Management Method

To solve the problem, methods are developed such as in [12,13]. But these methods are not well suitable for the problem in this paper, as the flexible decoder deals with different constraint lengths, code rates and polynomials. In this paper, a new method based on recursion construction is put forward, and it is independently of the idea in [6].

Firstly, initial data arrangement map of 4 banks for $K = 5$ at stage 0 is given by Eq. (14) (each column represents a bank, and each element is a binary number):

$$\left[\begin{array}{c|c|c|c} 00\_00 & 00\_01 & 00\_10 & 00\_11 \\ 01\_01 & 01\_10 & 01\_11 & 01\_00 \\ 10\_10 & 10\_11 & 10\_00 & 10\_01 \\ 11\_11 & 11\_00 & 11\_01 & 11\_10 \end{array}\right] \quad \begin{matrix} \Uparrow \\ Bank0 \end{matrix} \quad \begin{matrix} \Uparrow \\ Bank1 \end{matrix} \quad \begin{matrix} \Uparrow \\ Bank2 \end{matrix} \quad \begin{matrix} \Uparrow \\ Bank3 \end{matrix} \qquad (14)$$

Then the initial data arrangement map for $K = 7$ at stage 0 is constructed as following:

$$\left[\begin{array}{cccc} 00\_00\_00 & 00\_00\_01 & 00\_00\_10 & 00\_00\_11 \\ 00\_01\_01 & 00\_01\_10 & 00\_01\_11 & 00\_01\_00 \\ 00\_10\_10 & 00\_10\_11 & 00\_10\_00 & 00\_10\_01 \\ 00\_11\_11 & 00\_11\_00 & 00\_11\_01 & 00\_11\_10 \\ \hline 01\_00\_01 & 01\_00\_10 & 01\_00\_11 & 01\_00\_00 \\ 01\_01\_10 & 01\_01\_11 & 01\_01\_00 & 01\_01\_01 \\ 01\_10\_11 & 01\_10\_00 & 01\_10\_01 & 01\_10\_10 \\ 01\_11\_00 & 01\_11\_01 & 01\_11\_10 & 01\_11\_11 \\ \hline 10\_00\_10 & 10\_00\_11 & 10\_00\_00 & 10\_00\_01 \\ 10\_01\_11 & 10\_01\_00 & 10\_01\_01 & 10\_01\_10 \\ 10\_10\_00 & 10\_10\_01 & 10\_10\_10 & 10\_10\_11 \\ 10\_11\_01 & 10\_11\_10 & 10\_11\_11 & 10\_11\_00 \\ \hline 11\_00\_11 & 11\_00\_00 & 11\_00\_01 & 11\_00\_10 \\ 11\_01\_00 & 11\_01\_01 & 11\_01\_10 & 11\_01\_11 \\ 11\_10\_01 & 11\_10\_10 & 11\_10\_11 & 11\_10\_00 \\ 11\_11\_10 & 11\_11\_11 & 11\_11\_00 & 11\_11\_01 \end{array}\right] \quad \begin{matrix} \Uparrow \\ Bank0 \end{matrix} \quad \begin{matrix} \Uparrow \\ Bank1 \end{matrix} \quad \begin{matrix} \Uparrow \\ Bank2 \end{matrix} \quad \begin{matrix} \Uparrow \\ Bank3 \end{matrix}$$

(15)

Then the initial data arrangement map for $K = 9$ at stage 0 is constructed as following:

$$M = \begin{bmatrix} M_0 \\ M_1 \\ M_2 \\ M_3 \end{bmatrix} \qquad (16)$$

Where $M_0 =$

$$\left[\begin{array}{cccc} 00\_00\_00\_00 & 00\_00\_00\_01 & 00\_00\_00\_10 & 00\_00\_00\_11 \\ 00\_00\_01\_01 & 00\_00\_01\_10 & 00\_00\_01\_11 & 00\_00\_01\_00 \\ 00\_00\_10\_10 & 00\_00\_10\_11 & 00\_00\_10\_00 & 00\_00\_10\_01 \\ 00\_00\_11\_11 & 00\_00\_11\_00 & 00\_00\_11\_01 & 00\_00\_11\_10 \\ \hline 00\_01\_00\_01 & 00\_01\_00\_10 & 00\_01\_00\_11 & 00\_01\_00\_00 \\ 00\_01\_01\_10 & 00\_01\_01\_11 & 00\_01\_01\_00 & 00\_01\_01\_01 \\ 00\_01\_10\_11 & 00\_01\_10\_00 & 00\_01\_10\_01 & 00\_01\_10\_10 \\ 00\_01\_11\_00 & 00\_01\_11\_01 & 00\_01\_11\_10 & 00\_01\_11\_11 \\ \hline 00\_10\_00\_10 & 00\_10\_00\_11 & 00\_10\_00\_00 & 00\_10\_00\_01 \\ 00\_10\_01\_11 & 00\_10\_01\_00 & 00\_10\_01\_01 & 00\_10\_01\_10 \\ 00\_10\_10\_00 & 00\_10\_10\_01 & 00\_10\_10\_10 & 00\_10\_10\_11 \\ 00\_10\_11\_01 & 00\_10\_11\_10 & 00\_10\_11\_11 & 00\_10\_11\_00 \\ \hline 00\_11\_00\_11 & 00\_11\_00\_00 & 00\_11\_00\_01 & 00\_11\_00\_10 \\ 00\_11\_01\_00 & 00\_11\_01\_01 & 00\_11\_01\_10 & 00\_11\_01\_11 \\ 00\_11\_10\_01 & 00\_11\_10\_10 & 00\_11\_10\_11 & 00\_11\_10\_00 \\ 00\_11\_11\_10 & 00\_11\_11\_11 & 00\_11\_11\_00 & 00\_11\_11\_01 \end{array}\right] \qquad (17)$$

$M_1 =$

$$\left[\begin{array}{cccc} 01\_00\_00\_01 & 01\_00\_00\_10 & 01\_00\_00\_11 & 01\_00\_00\_00 \\ 01\_00\_01\_10 & 01\_00\_01\_11 & 01\_00\_01\_00 & 01\_00\_01\_01 \\ 01\_00\_10\_11 & 01\_00\_10\_00 & 01\_00\_10\_01 & 01\_00\_10\_10 \\ 01\_00\_11\_00 & 01\_00\_11\_01 & 01\_00\_11\_10 & 01\_00\_11\_11 \\ \hline 01\_01\_00\_10 & 01\_01\_00\_11 & 01\_01\_00\_00 & 01\_01\_00\_01 \\ 01\_01\_01\_11 & 01\_01\_01\_00 & 01\_01\_01\_01 & 01\_01\_01\_10 \\ 01\_01\_10\_00 & 01\_01\_10\_01 & 01\_01\_10\_10 & 01\_01\_10\_11 \\ 01\_01\_11\_01 & 01\_01\_11\_10 & 01\_01\_11\_11 & 01\_01\_11\_00 \\ \hline 01\_10\_00\_11 & 01\_10\_00\_00 & 01\_10\_00\_01 & 01\_10\_00\_10 \\ 01\_10\_01\_00 & 01\_10\_01\_01 & 01\_10\_01\_10 & 01\_10\_01\_11 \\ 01\_10\_10\_01 & 01\_10\_10\_10 & 01\_10\_10\_11 & 01\_10\_10\_00 \\ 01\_10\_11\_10 & 01\_10\_11\_11 & 01\_10\_11\_00 & 01\_10\_11\_01 \\ \hline 01\_11\_00\_00 & 01\_11\_00\_01 & 01\_11\_00\_10 & 01\_11\_00\_11 \\ 01\_11\_01\_01 & 01\_11\_01\_10 & 01\_11\_01\_11 & 01\_11\_01\_00 \\ 01\_11\_10\_10 & 01\_11\_10\_11 & 01\_11\_10\_00 & 01\_11\_10\_01 \\ 01\_11\_11\_11 & 01\_11\_11\_00 & 01\_11\_11\_01 & 01\_11\_11\_10 \end{array}\right] \qquad (18)$$

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$M_2 =$

10_00_00_10	10_00_00_11	10_00_00_00	10_00_00_01
10_00_01_11	10_00_01_00	10_00_01_01	10_00_01_10
10_00_10_00	10_00_10_01	10_00_10_10	10_00_10_11
10_00_11_01	10_00_11_10	10_00_11_11	10_00_11_00
10_01_00_11	10_01_00_00	10_01_00_01	10_01_00_10
10_01_01_00	10_01_01_01	10_01_01_10	10_01_01_11
10_01_10_01	10_01_10_10	10_01_10_11	10_01_10_00
10_01_11_10	10_01_11_11	10_01_11_00	10_01_11_01
10_10_00_00	10_10_00_01	10_10_00_10	10_10_00_11
10_10_01_01	10_10_01_10	10_10_01_11	10_10_01_00
10_10_10_10	10_10_10_11	10_10_10_00	10_10_10_01
10_10_11_11	10_10_11_00	10_10_11_01	10_10_11_10
10_11_00_01	10_11_00_10	10_11_00_11	10_11_00_00
10_11_01_10	10_11_01_11	10_11_01_00	10_11_01_01
10_11_10_11	10_11_10_00	10_11_10_01	10_11_10_10
10_11_11_00	10_11_11_01	10_11_11_10	10_11_11_11

(19)

$M_3 =$

11_00_00_11	11_00_00_00	11_00_00_01	11_00_00_10
11_00_01_00	11_00_01_01	11_00_01_10	11_00_01_11
11_00_10_01	11_00_10_10	11_00_10_11	11_00_10_00
11_00_11_10	11_00_11_11	11_00_11_00	11_00_11_01
11_01_00_00	11_01_00_01	11_01_00_10	11_01_00_11
11_01_01_01	11_01_01_10	11_01_01_11	11_01_01_00
11_01_10_10	11_01_10_11	11_01_10_00	11_01_10_01
11_01_11_11	11_01_11_00	11_01_11_01	11_01_11_10
11_10_00_01	11_10_00_10	11_10_00_11	11_10_00_00
11_10_01_10	11_10_01_11	11_10_01_00	11_10_01_01
11_10_10_11	11_10_10_00	11_10_10_01	11_10_10_10
11_10_11_00	11_10_11_01	11_10_11_10	11_10_11_11
11_11_00_10	11_11_00_11	11_11_00_00	11_11_00_01
11_11_01_11	11_11_01_00	11_11_01_01	11_11_01_10
11_11_10_00	11_11_10_01	11_11_10_10	11_11_10_11
11_11_11_01	11_11_11_10	11_11_11_11	11_11_11_00

(20)

It is easy to verify that the above equation (16) satisfy all the requiring rules for $K = 5,7,9$. For $K = 6,8$, the initial data arrangement map of 4 banks for $K = 6$ at stage 0 is constructed as following by inverting last 2 bits of each column:

0_00_00	0_00_01	0_00_10	0_00_11
0_01_01	0_01_10	0_01_11	0_01_00
0_10_10	0_10_11	0_10_00	0_10_01
0_11_11	0_11_00	0_11_01	0_11_10
1_00_11	1_00_10	1_00_01	1_00_00
1_01_10	1_01_01	1_01_00	1_01_11
1_10_01	1_10_00	1_10_11	1_10_10
1_11_00	1_11_11	1_11_10	1_11_01

(21)

It is easy to verify that the above equation (21) satisfy all the requiring rules for $K = 6$. Then the initial data arrangement map for $K = 8$ is given by equation (22) :

$M_0 =$

00_0_00_00	00_0_00_01	00_0_00_10	00_0_00_11
00_0_01_01	00_0_01_10	00_0_01_11	00_0_01_00
00_0_10_10	00_0_10_11	00_0_10_00	00_0_10_01
00_0_11_11	00_0_11_00	00_0_11_01	00_0_11_10
00_1_00_11	00_1_00_10	00_1_00_01	00_1_00_00
00_1_01_10	00_1_01_01	00_1_01_00	00_1_01_11
00_1_10_01	00_1_10_00	00_1_10_11	00_1_10_10
00_1_11_00	00_1_11_11	00_1_11_10	00_1_11_01

(22)

By inverting last bit of each column :

$M_1 =$

01_0_00_01	01_0_00_00	01_0_00_11	01_0_00_10
01_0_01_00	01_0_01_11	01_0_01_10	01_0_01_01
01_0_10_11	01_0_10_10	01_0_10_01	01_0_10_00
01_0_11_10	01_0_11_01	01_0_11_00	01_0_11_11
01_1_00_10	01_1_00_11	01_1_00_00	01_1_00_01
01_1_01_11	01_1_01_00	01_1_01_01	01_1_01_10
01_1_10_00	01_1_10_01	01_1_10_10	01_1_10_11
01_1_11_01	01_1_11_10	01_1_11_11	01_1_11_00

(23)

By inverting second bit of each column in reverse order :

$M_2 =$

10_0_00_10	10_0_00_11	10_0_00_00	10_0_00_01
10_0_01_11	10_0_01_00	10_0_01_01	10_0_01_10
10_0_10_00	10_0_10_01	10_0_10_10	10_0_10_11
10_0_11_01	10_0_11_10	10_0_11_11	10_0_11_00
10_1_00_01	10_1_00_00	10_1_00_11	10_1_00_10
10_1_01_00	10_1_01_11	10_1_01_10	10_1_01_01
10_1_10_11	10_1_10_10	10_1_10_01	10_1_10_00
10_1_11_10	10_1_11_01	10_1_11_00	10_1_11_11

(24)

By inverting last 2 bits of each column in reverse order :

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$$M_3 = \begin{bmatrix} 11\_0\_00\_11 & 11\_0\_00\_10 & 11\_0\_00\_01 & 11\_0\_00\_00 \\ 11\_0\_01\_10 & 11\_0\_01\_01 & 11\_0\_01\_00 & 11\_0\_01\_11 \\ 11\_0\_10\_01 & 11\_0\_10\_00 & 11\_0\_10\_11 & 11\_0\_10\_10 \\ 11\_0\_11\_00 & 11\_0\_11\_11 & 11\_0\_11\_10 & 11\_0\_11\_01 \\ \hline 11\_1\_00\_00 & 11\_1\_00\_01 & 11\_1\_00\_10 & 11\_1\_00\_11 \\ 11\_1\_01\_01 & 11\_1\_01\_10 & 11\_1\_01\_11 & 11\_1\_01\_00 \\ 11\_1\_10\_10 & 11\_1\_10\_11 & 11\_1\_10\_00 & 11\_1\_10\_01 \\ 11\_1\_11\_11 & 11\_1\_11\_00 & 11\_1\_11\_01 & 11\_1\_11\_10 \end{bmatrix} \quad (25)$$

It is easy to verify that the matrix $M = [M_0 \quad M_1 \quad M_2 \quad M_3]'$ composed of above results as equation (16) satisfy all the requiring rules for $K = 6,8$.

4. CONCLUSIONS

In this paper, a simple recursive efficient state metric memory management method for multi-standard wireless communication Viterbi decoder is presented. This method is very easy to extend to other K value, and it is very important to solve such irregular addressing problem in this kind of viterbi decoder for efficiency and chip area saving. Although not formulated through strictly theoretical formulating, this method is easier to use for engineers than existing methods.

ACKNOWLEDGEMENTS

This work was partly supported by National NSFC of China (60903045).

REFRENCES:

[1] D.A.F. El-Dib, M.I. Elmasry, "Low-power register-exchange Viterbi decoder for high-speed wireless communications," Proceedings of the 2002 IEEE International Symposium on Circuits and Systems(ISCAS 2002), May 2002, pp. 737–740.

[2] P. J. Black and T. H. Meng, "A 140-Mb/s, 32-state, radix-4 Viterbi decoder," Solid-State Circuits, IEEE Journal of, vol. 27, pp. 1877-1885, 1992.

[3] D. Yeh, et al., "RACER: a reconfigurable constraint-length 14 Viterbi decoder," FPGAs for Custom Computing Machines, 1996. Proceedings. IEEE Symposium on, 1996, pp. 60-69.

[4] W. Ching-Wen and C. Yun-Nan, "Design of Viterbi decoders with in-place state metric update and hybrid traceback processing, " Signal Processing Systems, 2001 IEEE Workshop on, 2001, pp. 5-15.

[5] B. Pandita and S. K. Roy, "Design and implementation of a Viterbi decoder using FPGAs," VLSI Design, 1999. Proceedings. Twelfth International Conference On, 1999, pp. 611-614.

[6] Li Zhou, et al., "A flexible viterbi decoder for software defined radio", Journal of Theoretical and Applied Information Technology, 20th January 2013. Vol. 47 No.2, pp. 702-706.

[7] C. M. Rader, "Memory Management in a Viterbi Decoder," IEEE Trans. Commun., vol.COM-29, no.9, pp.1399-1401, 1981.

[8] P. G. Gulak and T. Kailath, "Locally Connected VLSI Architectures for the Viterbi Algorithm," IEEE J. Select. Areas Commun., vol.6, no.3, pp.527-537, 1988.

[9] K.A. Wen, T.S. Wen, and J.F. Wang, "A New Transform Algorithm for Viterbi Decoding," IEEE Trans. Commun., vol.38, no.6, pp.764-772, 1990.

[10] G. Fettweis and H. Meyr, "High-Speed Parallel Viterbi Decoding: Algorithm and VLSI-Architecture," IEEE Commun. Magazine, pp.46-55, May 1991.

[11] H.-D. Lin and D. G. Messerschmitt, "Parallel Viterbi Decoding Methods for Uncontrollable and Controllable Sources," IEEE Trans. Commun., vol.41, no.1, pp.62-69, 1993.

[12] M.D.Shieh, M.H.Sheu and W.S.Ju, "Efficient Management of In-Place Path Metric Update and Its Implementation for Viterbi Decoders", IEEE International Symposium on Circuits and Systems, vol.4,pp.449-452, 1998

[13] C.M.Wu,M.D.Shieh,C.H.Wu and M.H.Sheu, "An Efficient Approach for In-Place Scheduling of Path Metric Update in Viterbi Decoders", IEEE International Symposium on Circuits and Systems, May 28-31, 2000