AN EFFICIENT STATE METRIC MEMORY MANAGEMENT METHOD FOR FLEXIBLE VITERBI DECODER

XU BANGJIAN, LIU ZONGLIN, YANG HUI, CHENG LING
1Assoc Prof., 2Assoc.Prof., 3Postgraduate, 4Postgraduate
College of Computer, National University of Defense Technology, People’s Republic of China

E-mail: 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 \( i \mod 4 = 0 \).

![Figure 1: Diagram Of Cascaded ACSU](image-url)
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 $K=5$](image)

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. $s^t$ and $s^t$ 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].

$$\lfloor j/4\rfloor \equiv \lfloor j/4\rfloor \mod 2^{K-3}, \quad i \neq j \mod 2^{K-3}$$

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](image)

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:

$$i = (b_{k-2} b_{k-1} \cdots b_2 b_1 00)_2$$

(1)

$$i + 1 = (b_{k-2} b_{k-1} \cdots b_2 b_1 01)_2$$

(2)

$$i + 2 = (b_{k-2} b_{k-1} \cdots b_2 b_1 10)_2$$

(3)

$$i + 3 = (b_{k-2} b_{k-1} \cdots b_2 b_1 11)_2$$

(4)

where $b_k$ is a binary bit. Then the output state id’s are given as follows:

$$i/4 = (00b_{k-2} \cdots b_3 b_2)_2 = (i) >> 2$$

(5)

$$i/4 + M/4 = (01b_{k-2} \cdots b_3 b_2)_2 = (i+1) >> 2$$

(6)

$$i/4 + M/2 = (10b_{k-2} \cdots b_3 b_2)_2 = (i+2) >> 2$$

(7)

$$i/4 + 3M/4 = (11b_{k-2} \cdots b_3 b_2)_2 = (i+3) >> 2$$

(8)
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):

\[
\begin{align*}
S_{t+2}^1 & \rightarrow S_t^1, \\
S_{t+2}^{4+M/4} & \rightarrow S_t^{t+1}, \\
S_{t+2}^{4+M/2} & \rightarrow S_t^{t+2}, \\
S_{t+2}^{4+3M/4} & \rightarrow S_t^{t+3}
\end{align*}
\]  

(9)

**Theorem 1:** When \(K\) is an odd number, after \(\frac{(K-1)}{2}\) 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 \(t\).

**Proof:** Since after every 2 stages the corresponding output 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 \(S_t^1, S_t^{t+1}, S_t^{t+2}, S_t^{t+3}\), where \(i \mod 4 = 0\)), should be placed in 4 different banks independently.

**Proof:** 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 \([i/4] \times [j/4]\), 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 \(i \neq j \mod 2^{K-3}\), 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 \(\frac{K-1}{2}\) 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:

\[
\begin{bmatrix}
0 & 0 & \ldots & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \ldots & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & \ldots & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & \ldots & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & \ldots & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & \ldots & 0 & 1 & 0 & 1 & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
\]

(10)

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

\[
\begin{bmatrix}
0 & 1 & 2 & 3 & 4 & \ldots & 2^{K-3} - 1
\end{bmatrix}^T
\]

(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:

\[
\begin{bmatrix}
c_{m,0}, & c_{0,0}, & c_{0,1}, & c_{1,0}, & c_{1,1}, & c_{2,0}, & c_{2,1}, & c_{3,0}, & c_{4,1}, & c_{5,1}, & \ldots, & c_{2^{K-3} - 1,0}, & c_{2^{K-3} - 1,1}
\end{bmatrix}
\]

(12)

where \(\{c_{m,0}, c_{m,1}\}\) is a pair of binary bits. It is easy to see that there are \(256 \times 2^{(K-1)}\) 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:

\[
\begin{bmatrix}
c_{0,1}, & c_{0,0}, & 0 & 0 & \ldots & 0 & 0 & 0 & 0 \\
c_{1,1}, & c_{1,0}, & 0 & 0 & \ldots & 0 & 0 & 0 & 1 \\
c_{2,1}, & c_{2,0}, & 0 & 0 & \ldots & 0 & 0 & 1 & 0 \\
c_{3,1}, & c_{3,0}, & 0 & 0 & \ldots & 0 & 0 & 1 & 1 \\
c_{4,1}, & c_{4,0}, & 0 & 0 & \ldots & 0 & 0 & 1 & 0 \\
c_{5,1}, & c_{5,0}, & 0 & 0 & \ldots & 0 & 0 & 1 & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
c_{2^{(K-3)-1},1}, & c_{2^{(K-3)-1},0}, & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
\]

(13)

So, now to obey rule 1, it is easy to see that when \([m/4] = [n/4]\) 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 \(24 \times 2^{(K-1)} = 331776 \times 2^{(K-7)}\). For \(K = 9\), this number is 331776\(^4\). This number is about \(1/10^6\) of \(256^{16}\).

To reduce this number further, rule 2 can also be applied to matrix (12). It is easy to see that when
m = n \mod 2^{K-5} \text{ and } K > 6 \text{ , there are } \{c_{m,1},c_{m,0}\} \neq \{c_{n,1},c_{n,0}\}. \text{ 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 331776^6.

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):

\[
\begin{bmatrix}
00 & 01 & 00 & 11 \\
01 & 01 & 01 & 01 \\
10 & 10 & 10 & 10 \\
11 & 11 & 11 & 11 \\
\end{bmatrix}
\]

(14)

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

\[
\begin{bmatrix}
00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 10 & 00 & 00 & 11 \\
00 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 \\
01 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\
\end{bmatrix}
\]

(15)

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

\[
M = \begin{bmatrix}
M_0 \\
M_1 \\
M_2 \\
M_3 \\
\end{bmatrix}
\]

(16)

Where \(M_0 = \)

\[
\begin{bmatrix}
00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 10 & 00 & 00 & 11 \\
00 & 00 & 01 & 00 & 01 & 00 & 01 & 00 & 01 & 00 & 01 & 00 \\
00 & 00 & 01 & 00 & 01 & 00 & 01 & 00 & 01 & 00 & 01 & 00 \\
00 & 00 & 01 & 00 & 01 & 00 & 01 & 00 & 01 & 00 & 01 & 00 \\
\end{bmatrix}
\]

(17)

\[
M_1 = \begin{bmatrix}
01 & 00 & 00 & 10 & 01 & 00 & 11 & 01 & 00 & 11 \\
01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 \\
01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 \\
01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 \\
\end{bmatrix}
\]

(18)
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) :

$$M_0 = \begin{bmatrix}
0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
\end{bmatrix}$$

(22)

By inverting last bit of each column :

$$M_1 = \begin{bmatrix}
0_1 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_1 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_1 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_1 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_1 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_1 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_1 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_1 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
\end{bmatrix}$$

(23)

By inverting second bit of each column in reverse order :

$$M_2 = \begin{bmatrix}
0_1 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_1 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_1 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_1 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_1 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_1 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_1 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
0_1 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 & 0_0 \\
\end{bmatrix}$$

(24)

By inverting last 2 bits of each column in reverse order :
It is easy to verify that the matrix $M = [M_0 \ M_1 \ M_2 \ M_3]^T$ 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).

REFERENCES:


