Optimization and Implementation of Balise Telegram Encoding Algorithm

ZHUO Peng, LIU Jiangsha, FANG Zhigang

Abstract: The balise is a vital device for onboard-wayside information transmission, and its telegram encoding time-consuming is uncertain. Firstly, the balise telegram format and encoding algorithm are introduced. Then, multiple steps of the algorithm are optimized and improved from two major directions: first, improving computational efficiency, such as improving the calculation efficiency of scrambling, check bits, and Hamming distance through look-up table; by storing two 11-bit words in an integer number to effectively reduce shift operations, making it possible to check and shift operation of the valid word can be executed in parallel; using an integer number to achieve 4 bits under-sampling operation in one execution. Second, avoiding a large number of unnecessary calculations, such as adjusting and optimizing the left-right shift execution order of Off-Synch-Parsing condition test; checking consecutive valid words using a greedy algorithm; executing scrambling and partial Off-Synch-Parsing condition test for valid word in parallel; executing check bit calculations and alphabet condition test in parallel. Finally, testing was conducted on a PC with a CPU clock frequency of 2.93 GHz, and the results showed that the first legal transmission telegram could be

obtained in just 8μs, and the performance on the STM32F207 series chip is comparable to that on the FPGA.

Key words: Balise; Telegram; Encoding; Look-Up Table Method; CRC

Balises are widely used in high-speed railways and urban rail transit in China. To address various threats in onboard-wayside wireless transmission, balise telegrams adopt a unique encoding method. The encoding specification of Chinese balise telegrams is 引自 from "FFFIS for Eurobalise" developed by the European Railway Transport Union [1], and currently, this specification has been updated to Baseline 4 in 2023. The corresponding domestic standard is "Technical Conditions for Balise Transmission System" [2] (TB/T 3485—2017). The characteristics of the balise telegram encoding method determine that its encoding time is uncertain, and in extreme cases, there may be situations where encoding is not possible [1]. An earlier literature [3] on the implementation of the encoding algorithm pointed out that after the scrambling bits are determined, the calculation results of the check bits for the first 928 bits of the long telegram can be determined. This part can be calculated in advance before determining the additional correction bits, thus reducing a large amount of repeated calculation work. At the same time, it also proposed using a binary search method in the inverse transformation process from 10 bits to 11 bits. Literature [4] constructs a reverse-lookup alphabet table and can directly determine whether the current word is a valid

legal word through the look-up table method, which can significantly save time compared to the binary search method. Due to the time-consuming nature of balise telegram encoding, many implementations are based on FPGA [5 - 7], and there are also those based on C language [8 - 10], but no further substantial optimization strategies have been proposed for this algorithm. Given the wide application of balise devices, improving their encoding efficiency helps to reduce various social and economic costs. Based on previous research, this paper uses the C language, the look-up table method, optimizes the storage of 11-bit word data, executes in parallel to end the calculation as early as possible, uses the greedy algorithm to check consecutive valid words, optimizes the check of correction constraint conditions, etc., to significantly improve the performance of the balise telegram encoding algorithm.

1 Balise Telegram Format

The balise transmission telegrams in FFFIS are divided into long telegrams and short telegrams. Both formats of telegrams consist of five parts: shaped data, control bits, scrambling bits, additional correction bits, and check bits [2]. The balise telegram format is shown in Table 1.

Shaped Data	Control Bits	Scrambling Bits	Additional Correction Bits	Check Bits
83×11 =	3	12	10	85

Shaped Data	Control Bits	Scrambling Bits	Additional Correction Bits	Check Bits
913				
21×11 = 231				

The long telegram and the short telegram only differ in the shaped data. The original user data telegram of the long telegram is 830 bits. After scrambling and shaping operations, it becomes 913 bits of shaped data. Adding the data of the following four parts, a total of 1023 bits, is usually represented as bn - 1, bn - 2, ..., b1, b0, where n = 1023. The original user data telegram of the short telegram is 210 bits, and after scrambling and shaping, it becomes 231 bits. The encoding algorithms of short telegrams and long telegrams are not essentially different, and long telegrams are widely used in national railway applications. Therefore, the long telegram is taken as an example for description.

The 1023 bits of the long telegram can be divided into 93 groups with 11 bits in each group, and each group is called a word. The 913 bits of data after scrambling and shaping the original user data telegram can be divided into 83 groups of words. The control bits are currently fixed as 001b [2]. Together with the high 8 bits of the scrambling bits, they form the 84th group of words. The low 4 bits of the scrambling bits plus the high 7 bits of the additional correction bits form the 85th group of words. The low 3 bits of the additional

correction bits plus the 85-bit check bits form the 86th - 93rd groups of words. The 1023 bits need to meet the correction constraint check conditions [2], which are specifically checked according to the following 4 conditions.

Condition 1: Alphabet Check

Select 1024 specific numbers from integers 0 - (2^11 - 1) to form an alphabet table. It is required that each word in the telegram is in the alphabet table, which is also called a valid word.

Condition 2: Off-Synch-Parsing Check

Perform cyclic shifting on the 1023-bit telegram, and then re-divide it into 93 groups of words. When shifting left or right by 1 bit, the number of consecutive valid words in the newly divided 93 groups of words cannot exceed 2; in other shift cases (since shifting 11 bits returns to itself, shifting left by 10 bits is equivalent to shifting right by 1 bit, so other shifts refer to shifting left by 2 - 9 bits), the number of consecutive valid words in the newly divided 93 groups of words cannot exceed 10. Note that the check for consecutive valid words is a cyclic check of the 93 groups of words.

Condition 3: Check Whether the Aperiodic Condition is Met (Only for Long Telegrams)

Let i be a multiple of 11. The Hamming distance between bi - 1, ..., bi - 22 and bi - 341 - 1, ..., bi - 341 - 22 is at least 3. That is, the total Hamming distance between any two consecutive groups of words in the 93 groups of words and

the two consecutive groups of words 31 groups apart is at least 3. On this basis, for the new telegram after shifting left or right by 1 - 3 bits, it is also required that the Hamming distance corresponding to the above interval with the original telegram is at least 2.

Condition 4: Under-Sampling Check

When the under-sampling coefficient is 2, the telegram sequence becomes bn - 2, bn - 4, ..., b1, bn - 1, bn - 3, ..., b2, b0. It is required that for the new telegram sequence and the new telegrams after cyclic left-shifting this telegram by 1 - 10 bits, the number of consecutive valid words in the cyclic check does not exceed 30. It is also required that when the under-sampling coefficients are 4, 8, and 16, the number of consecutive valid words does not exceed 30.

2 Encoding Algorithm Introduction

The encoding process of balise telegrams in TB/T 3485—2017 is shown in Figure 1.

[Insert Figure 1 here: Balise Telegram Encoding Process]

Step 1: Select 12-bit Scrambling Bits

According to the analysis in Literature [3] and Literature [4], after selecting the scrambling bits, perform the alphabet condition check on the 84th group. In this way, the number of scrambling bits to be tried is reduced from 4096 to

2512. In code implementation, when looping through all the scrambling bits to be tried, only need to first execute the validity check of the 84th group of words in the loop body.

Step 2: Scramble User Data

This can be further divided into 3 steps: ① Replace the first group of 10-bit user bits with a function of all user bits. That is, divide the original user data telegram into 83 groups with 10 bits in each group, and then replace the first group of 10-bit user bits with the low 10 bits of the integer addition of the 83 groups of data; ② Calculate a 32-bit integer S using the 12-bit scrambling bit B, that is, S = (2801775573×B) mod 2^32; ③ Use a 32-bit linear feedback shift register with an initial state of S to perform the scrambling operation. The scrambling polynomial is h(x)=x^32 + x^31 + x^30 + x^29 + x^27 + x^25 + 1. The scrambling operation is not essentially different from the conventional CRC calculation, except that a different polynomial is used, and attention needs to be paid to the problem of high and low bits of the data stream.

Step 3: Conversion from 10 bits to 11 bits

Based on the 1024 words given in Literature [2], convert all 10-bit integers into corresponding 11-bit integers. Use the 10-bit number as the subscript to look up the table given in Literature [2] to obtain the corresponding 11-bit word.

Step 4: Check the Correction Constraint Conditions

Perform the candidate telegram correction constraint condition check as much

as possible on the currently determined first 928 bits. Obviously, the 83 groups of shaped data automatically meet Condition 1; the 84th group has passed the check in Step 1, and the 85th group and subsequent groups need to determine the additional correction bits to be determined. Therefore, there is no need to perform the Condition 1 check here. By trying all scrambling bits for 137 active balise telegrams preset at a certain test station, it is found that the elimination rates of Condition 2, Condition 3, and Condition 4 when executed separately are 86%, 1%, and 0% respectively. Since it is time-consuming to execute Condition 3 and Condition 4 checks, and their elimination rates are very low, only Condition 2 is executed in this step. This is also consistent with the description of the encoding algorithm flow chart in Literature [10]. Finally, after being eliminated by Condition 2, an average of 352 scrambling bits are available for each telegram.

Step 5: Select 10-bit Additional Correction Bits

According to the analysis in Literature [3] and Literature [4], after the additional correction bits are determined, the Condition 1 check of the 85th group of words can be executed first. In code implementation, when looping through all the additional correction bits, only need to first execute the check of the 85th group of words in the loop body. Through the test statistics of the preset active balise telegrams of multiple stations and all the passive balise telegrams on the Yantong Line, it is found that this step can eliminate half of the additional correction bits.

Step 6: Generate Check Bits

According to the analysis in Literature [3] and Literature [4], the generation of check bits can be divided into 3 steps: ① First, calculate the check bits for the first 928 bits, and the result is recorded as BCH1 [4]. This step can be advanced to be calculated before Step 5; ② Add the calculation of 10-bit additional correction bits on the basis of BCH1 to obtain the final BCH [4]; ③ Use a linear algorithm plus gL(x) [2] to obtain the final check bits. According to the conclusion of Step 5, in the second and third steps of this step, when trying all the optional additional correction bits for one scrambling bit, an average of 512 calculations are required. Therefore, calculating BCH1 before Step 5 can significantly reduce the calculation time [3 - 4].

Step 7: Check the Correction Constraint Conditions

First, perform the Condition 1 check, and only need to check the last 8 groups of words. Then, execute the checks of Condition 2, Condition 3, and Condition 4 in sequence. Test all the preset active balise telegrams of 3 stations and all the passive balise telegrams on the Yantong Line to obtain all the legal transmission telegrams that can pass the check. The statistical situation of the elimination rates of each condition is shown in Table 2.

Telegram	Condition 1	Condition 2	Condition 3	Condition 4
137 Active	99.61	30.82	1.33	0

Telegram	Condition 1	Condition 2	Condition 3	Condition 4
Telegrams at Tongji Test Station				
86 Active Telegrams at Yongjin Shengzhou Station	99.61	30.96	1.29	0
236 Active Telegrams at Nantong EMU Depot	99.61	30.73	1.35	0
729 Passive Telegrams on Yantong Line	99.61	30.66	1.30	0

It can be seen from Table 2 that Condition 1 has the highest elimination rate. Among all 1024 additional correction bits under one scrambling bit, after the Condition 1 check (including the 85th group), an average of only 2 additional correction bits can pass. The elimination rate of Condition 2 in Table 2 is the result of further elimination by 30% on the basis of the elimination by Condition 1. Similarly, Condition 3 and Condition 4 are also eliminated on the basis of the elimination by the previous conditions. It can be seen from Table 2 that executing Condition 1, Condition 2, Condition 3, and Condition 4 in sequence is the most effective elimination check method.

3 Algorithm Optimization Strategies

3.1 Calculating Scrambling and Check Bits by Look-Up Table Method

From the introduction of the encoding algorithm, it can be seen that the calculations of scrambling and check bits are typical linear feedback shift calculations. This part can be replaced with the mature look-up table method to improve efficiency. Typical look-up table methods usually complete the calculation of 8 bits, that is, one byte, in one look-up.

For the scrambling calculation, since the user data needs to be divided into groups of 10 bits for processing first, a table for calculating 10 bits in one look-up can be created. This avoids converting the 10-bit group data into byte data streams of 8 bits per group, and the calculation result data of the 10-bit look-up table method can be directly used in the subsequent Step 3 to directly convert it into 11-bit words through look-up. The 8-bit look-up table method corresponds to 256 table values, and the 10-bit look-up table method corresponds to 1024 table values. Since descrambling is just the inverse operation of scrambling, the 10-bit look-up table method can also be used to speed up the descrambling operation.

When calculating BCH1 for the check bits, since 928 bits is exactly a multiple of 8, the typical 8-bit look-up table method is adopted; for the remaining 10-bit additional correction bits, the 10-bit look-up table method is used.

Literature [11] mentions using the byte-type algorithm look-up table method in balise decoding. For the check bit calculation in encoding, the corresponding table value is 85 bits, which can be stored in 9 byte arrays or 3 integer arrays. Obviously, the efficiency of 3 integer arrays is better than that of 9 byte arrays. The generation method of the table values is the same as that of the look-up table values for scrambling calculation. The generation of 85-bit table values and 32-bit table values is also the same in logic and form, which has been introduced in many literatures and will not be repeated here.

3.2 Optimizing the Storage Method of 11-bit Words

In the checking of correction constraint conditions, Conditions 2, 3, and 4 all require a large number of shift operations on 11-bit words. A global integer array variable can be defined, with each integer storing 2 11-bit words, which can effectively reduce shift operations. That is, the conventional 11-bit word array is like the first group $b_{1022},b_{1021},\cdots,b_{1012}$, the second group $b_{1011},b_{1010},\cdots,b_{1001}$, ..., and the last group $b_{10},b_{9},\cdots,b_{0}$. It can be adjusted to the first group $b_{1022},b_{1021},\cdots,b_{1012},b_{1011},b_{1010},\cdots,b_{1001}$, the second group $b_{1011},b_{1010},\cdots,b_{1001},b_{1000},b_{999},\cdots,b_{990}$, ..., and the last group $b_{10},b_{9},\cdots,b_{0},b_{1022},b_{1021},\cdots,b_{1012}$. In this way, for the shift operations of left - shifting 1 - 10 bits on 11-bit words, only two steps are needed, that is, shifting first and then taking the lower 11 bits.

Therefore, when performing the checks of Conditions 2, 3, and 4, there is no need to complete the shift operations of all words in advance. Instead, the shift operations are executed only when they are needed. In this way, if the check fails during the execution process, unnecessary subsequent shift operations can be avoided. This can save a significant amount of time compared to the conventional approach.

3.3 Optimizing the Shift Execution Order of Condition 2

When performing the Condition 2 check in the correction constraint condition check of Step 4, according to the conventional implementation idea, it is to shift left cyclically by 1 bit, perform a valid word check, then shift left cyclically by 1 bit again, and perform another valid word check, with a total of 10 rounds. However, the condition that the number of consecutive valid words when shifting right or left by 1 bit cannot exceed 2 is obviously easier to meet than the condition that the number of consecutive valid words in other shifts cannot exceed 10. Therefore, this check should be executed first. Through the statistics of the elimination of all possible scrambling bits for 137 telegrams at a certain test station, it is found that whether shifting left by 1 bit or shifting right by 1 bit first can eliminate more than 170,000 candidates. This shows that the order of shifting left by 1 bit and shifting right by 1 bit does not affect the efficiency. Due to the randomness of the data, it can also be inferred that the order of shifting left by n bits and shifting right by n bits does not affect the

efficiency. The elimination amount of shifting left by 1 bit plus shifting right by 1 bit accounts for 88% of the total elimination amount.

For left - shifting 2 - 9 bits, the results of three execution orders are recorded and compared (as shown in Table 3). In Order 1, shift left successively by 2 - 9 bits. The elimination amount of shifting left by 9 bits (which is equivalent to shifting right by 2 bits) is large, but it is executed last, so the efficiency is greatly affected. Order 2 is to shift left and right successively by 2 bits, 3 bits, 4 bits, and 5 bits. Order 3 is to shift left and right successively by 5 bits, 4 bits, 3 bits, and 2 bits. By comparing Order 2 and Order 3, it can be seen that shifting left and right by 2 bits always eliminates more candidates than shifting left and right by 3 - 5 bits. Combining with the elimination situation of Order 1, it can be shown that Order 2 is the best execution order, which can save time more effectively than other execution orders.

This optimization strategy is applicable to both the Condition 2 checks in Step 7 and Step 4.

Left Shift 1 bit	Right Shift 1 bit	Left Shift 2 bits	Right Shift 2 bits	Left Shift 3 bits	Right Shift 3 bits	Left Shift 4 bits	Right Shift 4 bits	Left Shift 5 bits	Right Shift 5 bits

	Left Shift 1 bit	Right Shift 1 bit	Left Shift 2 bits	Right Shift 2 bits	Left Shift 3 bits	Right Shift 3 bits	Left Shift 4 bits	Right Shift 4 bits	Left Shift 5 bits	Right Shift 5 bits
Order 1	1736 98	8767 7	1548 2	1088 6	454	362	1357	1206	2454	2277
Order 2	1736 98	8767 7	1548 2	1236 0	359	375	1101	1005	1943	1853
Order 3	1736 98	8767 7	1358 2	1088 6	471	449	1642	1529	3033	2886

3.4 Checking Consecutive Valid Words with the Greedy Algorithm

Condition 2 requires judging that the number of consecutive valid words does not exceed 2 or 10, and Condition 4 requires judging that the number of consecutive valid words does not exceed 30. Since most of these judgments are non - satisfied cases, there is no need to check whether they are continuously satisfied in an incremental manner one by one. Instead, it can be assumed that n consecutive words starting from the current position meet the

requirements, and then check backwards to see if they actually meet the requirements one by one. If not, continue to assume that the next n consecutive words starting from the current position meet the requirements and check backwards again. When it is necessary to jump forward continuously but the remaining length is insufficient, that is, there are no consecutive words that meet the requirements, then the check passes. When the nth word is checked backwards and it is found that there are consecutive words that meet the requirements, the check can be ended immediately. In this way, when there are a large number of illegal words, the check can be completed quickly almost by jumping forward n words at a time. Ideally, the efficiency can be theoretically increased by n times. The greedy algorithm for checking consecutive valid words is shown in Figure 2.

[Insert Figure 2 here: Greedy Algorithm for Checking Consecutive Valid Words]

3.5 Executing Scrambling and the Valid Word Check of Condition 2 in Parallel

Since the number of consecutive valid words when shifting left or right by 1 bit in the Condition 2 check cannot exceed 2, even if the greedy algorithm is used, the maximum improvement is only 3 times. By adjusting the processing ideas of Step 2, Step 3, and Step 4, it is not necessary to wait until all scrambling calculations are completed before performing the conversion in Step 3, nor is it necessary to wait until all conversions in Step 3 are completed

before entering the correction constraint condition check in Step 4. That is, based on the previous 10 - bit look - up table method, once a 10 - bit scrambling result is calculated, the conversion from 10 bits to 11 bits can be carried out immediately, and then the corresponding left - shift 1 bit and right - shift 1 bit operations are performed. Without using the greedy algorithm, simply count in sequence. Once the number exceeds 2, the attempt of the current scrambling bit can be stopped immediately, which can effectively reduce the subsequent scrambling calculations and the conversion calculations from 10 bits to 11 bits. Through the tests on extreme telegrams, the strategy of ending the 83 - group scrambling calculation in advance performs better than the greedy algorithm of shifting left and right by 1 bit.

3.6 Executing Check Bit Calculation and the Valid Word Check of Condition 1 in Parallel

As mentioned in the previous algorithm, the calculation of check bits is carried out in 3 steps. Steps 2 and 3 need to be calculated 512 times on average for all possible candidate additional correction bits. The conventional idea is to combine the final 85 - bit check bits with the lower 3 - bit additional correction bits to form 8 groups of 11 - bit words after obtaining them, and then check the valid words one by one. However, the calculation in Step 2 can be completed in one step using the 10 - bit look - up table method, and Step 3 is to perform a linear addition operation with the fixed $g_L(x)$. Essentially,

these two steps are both linear addition operations on 85 - bit data. Since the linear addition operation satisfies the associative law, the result of Step 1 can be first linearly added to $g_L(x)$ in Step 3. In fact, the calculation method of Step 2 can be split into 8 groups of 11 - bit words (the first group is 8 bits, and the rest are 11 bits) for separate calculations. The corresponding 10 - bit look - up table values can also be stored in 8 groups of short integer arrays. In this way, a large amount of calculations can be carried out before the additional correction bits are determined. After the additional correction bits are determined, only need to simply calculate each 11 - bit word, and as soon as each 11 - bit word is calculated, the alphabet check can be executed immediately, instead of waiting until all check bits (8 groups of words) are calculated before performing the alphabet check. This can significantly improve the calculation efficiency.

Through the test statistics of 729 passive balise telegrams on the Yantong Line, it is found that the elimination situation of checking valid words one by one in sequence for groups 86 - 93 is the same as that of checking in reverse order. That is, the first valid word check always eliminates 1/2, the second valid word check eliminates 1/4, and the subsequent ones are all close to eliminating half of the remaining words. Based on this, combined with the fact that the 86th group of words still needs to be spliced with the lower 3 - bit of the additional correction bits, in the specific implementation, the method of checking in reverse order can be adopted to further reduce the calculations.

3.7 Optimizing the Correction Constraint Condition Check in Step 7

Based on the parallel calculation of the valid word check of Condition 1 and the check bits in Step 7, continue to analyze and optimize the implementation of Conditions 2, 3, and 4.

1. In Step 4, the Condition 2 check has been performed on the first 84 groups of data. Therefore, in the Condition 2 check of Step 7, this result can be utilized. When shifting left or right by 1 bit, it is only necessary to check whether there are more than 2 consecutive valid words in groups 82 - 93 and groups 1 - 3. When shifting in other ways, check whether there are more than 10 consecutive valid words in groups 72 - 93 and groups 1 - 11. This can save two - thirds of the time compared to the conventional method of re - performing the Condition 2 check on all 93 groups of words.
2. Condition 3 mainly involves a large number of Hamming distance calculations, that is, the Hamming distance calculation between two 11 - bit words. The look - up table method in Solution 5 of Literature [12] can be used for reference. Only need to construct the Hamming distance table values of 11 - bit words in advance. Since the maximum

distance is 11, it can be stored in a byte array, and a total of only 2048 bytes are required.

3. Analyze the under - sampling operation of Condition 4. When the under - sampling coefficient is 2, the telegram sequence becomes b_{n-2}, b_{n-4}, $\square$, b_1, b_{n-1}, b_{n-3}, $\square$, b_2, b_0. Since the bit positions are checked cyclically, b_{n-1}, b_{n-3}, $\square$, b_2, b_0 can be placed in front of b_{n-2}, b_{n-4}, $\square$, b_1. In this way, the first half is exactly 64 bytes, and the last bit position of the second half is left blank. This can greatly facilitate code implementation. The sequence with an under - sampling of 4 can be obtained by performing an under - sampling of 2 on the original sequence with an under - sampling of 2. In this way, all under - sampling operations can be easily implemented with a loop.

In the specific implementation of the under - sampling code, the conventional idea is to pick out the bits one by one to form a new sequence. By analyzing the characteristics of the under - sampling operation and combining the fact that an integer consists of 4 bytes, a fast under - sampling operation strategy of picking out 4 bits at a time is designed. Theoretically, this can save 75% of the time compared to the conventional idea. For the convenience of example, taking a big - endian CPU as an example, the operation process is shown in Figure 3.

[Insert Figure 3 here: Schematic Diagram of the Fast Under - Sampling Operation Process]

4 Conclusion

This paper analyzes each step of the balise telegram encoding algorithm and optimizes the encoding algorithm through seven strategies. The optimized program was tested on a PC with the same CPU clock frequency (2.93GHz) as that in Literature [4]. For all 729 passive balise telegrams on the Yantong Line, to obtain their corresponding first legal transmission telegram, 11 telegrams only took 8μs, and only 3 telegrams took more than 56μs, which were 58μs, 61μs, and 81μs respectively. Compared with the 2 - 6ms required to obtain the first legal transmission telegram given in Literature [4], the efficiency has been improved by two orders of magnitude. When obtaining all possible legal transmission telegrams for the 729 passive balise telegrams on the Yantong Line, the fastest one could obtain all 409 legal telegrams in only 6.4ms, and the slowest one could obtain all 524 legal telegrams in only 8.4ms.

The optimized program was also tested on the STM32F207 series chip. The encoding time for obtaining the first legal telegram was 1 - 4ms, which is comparable to the encoding time using FPGA in Literature [5] and also comparable to the encoding time using FPGA in our company. Therefore, from the cost - effectiveness perspective, the devices previously using FPGA

dedicated chips can consider being replaced with general - purpose CPU chips.

Due to the significant improvement in computational efficiency, for passive telegrams used in the field, the optimal telegram selection strategy proposed in Literature [4] can be adopted to select the telegram with the best evaluation index as the final transmission telegram from all the legal telegrams that pass the correction constraint condition check. For active balise telegrams, currently, the train control centers of various manufacturers use dedicated encoding units to complete real - time encoding work. For train control center equipment with strong safety - master computing capabilities, it can be attempted to cancel the dedicated encoding unit and directly implement encoding in the safety - master computing unit using the encoding algorithm in this paper. This can not only reduce equipment costs, simplify equipment composition, and reduce fault points, but also improve equipment reliability to a certain extent.