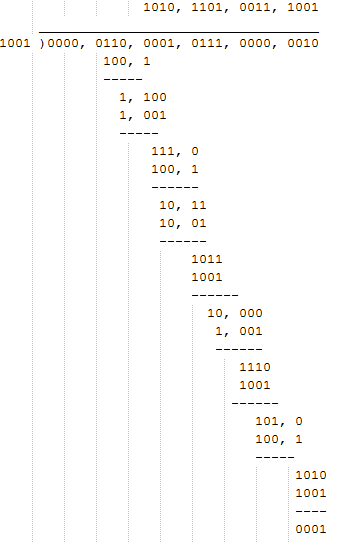
**Cyclical Redundancy Check（CRC）**

The basic idea of the CRC algorithm is to use the data transmitted as a very long number of digits. Divide this number by another number. The remainder is added to the original data as the check data. Take the data in the above example, for example:

6, 23, and 4 can be seen as a number of 2: 0000011000010111 00000010

If the divisor is selected 9, the binary is represented as: 1001

The division operation can be expressed as:



As you can see, the final remainder is 1. If we use the remainder as a checksum, the data is 6, 23, 4, 1.

The CRC algorithm is a bit similar to the process, but it is not the usual division of the above example. In the CRC algorithm, the binary data stream as the coefficients of the polynomial, then the polynomial multiplication and division. Let's take an example.

For example, we have two binary numbers, 1101 and 1011, respectively.

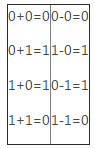
1101 is associated with the following Polynomials: 1x3+1x2+0x1+1x0=x3+x2+x0

1011 is associated with the following Polynomials: 1x3+0x2+1x1+1x0=x3+x1+x0

The multiplication of two polynomials: (x3+x2+x0) (x3+x1+x0) =x6+x5+x4+x3+x3+x3+x2+x1+x0

Results after the merger of similar items by mode 2 operation. That is to say the polynomial multiplication and division multiplication and subtraction are normal, using modulo 2 arithmetic. The operation of the model 2 is the result of dividing the result by 2 and taking the remainder. For example, 3 mod 2 = 1. So, the final polynomial is: x6+x5+x4+x3+x2+x1+x0, the corresponding binary number: 111111

The addition and subtraction method, after the operation of the model 2, is actually an operation, which is what we usually call a foreign or an operation:



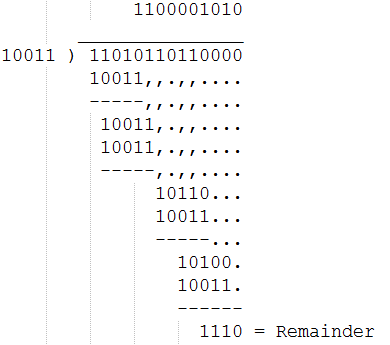
The above said a long time polynomial, in fact it is not the concept of polynomial multiply and divide can show the special features of these operations. But almost all of the documents that explain the CRC algorithm refer to polynomials, so here are some basic concepts. However, it is always very wordy with this polynomial expression, and the following explanation will try to use more concise writing.

The division of the division is similar to the concept of multiplication given above, or in places where the addition and subtraction are replaced by foreign or foreign operations. The following is an example:

The data to be transmitted is: 1101011011

The divisor is set to: 10011

Before the calculation, the original data is filled with 4 0:11010110110000, the reason is to fill 0, then then explain.



From this example we can see that the mode of addition and subtraction 2, does not need to consider the borrow problem, so simple division. The final remainder is the CRC verifying word. In order to perform CRC operation, that is, this special division operation, we must specify a divider. In CRC algorithm, this divider has a proper name called "generating polynomial". The selection of generating polynomials is a very difficult problem. If the selection is not good, then the probability of detecting errors will be much lower. Fortunately, this problem has been studied by experts for a long time. For those of us, we only need to use the available results.

The most commonly used generation polynomials are as follows:

CRC8=X8+X5+X4+X0

CRC-CCITT=X16+X12+X5+X0

CRC16=X16+X15+X2+X0

CRC12=X12+X11+X3+X2+X0

CRC32=X32+X26+X23+X22+X16+X12+X11+X10+X8+X7+X5+X4+X2+X1+X0

It is important to note that the width of the literature mentioned often when it comes to polynomial polynomial (Width, abbreviated as W), the number of binary bits is not polynomial corresponding to the digits, but the number of minus 1. For example, bit used in CRC8 to generate polynomial 8, actually the corresponding binary number nine: 100110001. In addition, polynomial representation and binary representation are cumbersome, the exchange is not convenient, therefore, with more than 16 hexadecimal shorthand to represent the literature, because of the high generation polynomial certainly is 1, the highest position by the width of that in short form, will be the highest of the 1 unified removed, such as the polynomial is 04C11DB7 called CRC32 actually said is 104C11DB7. Of course, this short, in addition to the convenience, the programming also has its use.

For the example above, width 4 (W=4), CRC in accordance with the requirements of the algorithm, to fill in the W 0 in the original data after the calculation, which is 4 to 0.

Polynomial width W=1 (CRC1) two, respectively is X1 and X1+X0, the reader can prove that parity is 10 corresponding parity check, and the 11 is the corresponding parity check. Therefore, when we write here, we know that parity is actually a special case of CRC check, which is why I want to introduce parity as an introduction.

Programming implementation of CRC algorithm：

We know from the front of the introduction of CRC check core is to realize the division of borrow. Here is an example of how to implement CRC validation.

Our hypothesis generation: 100110001 polynomial (0x31), which is CRC-8

The calculation steps are as follows:

(1) assign the initial value 0 to the CRC register (8-bits, 1bit less than the generation polynomial)

(2) add 8 0 to the stream of information to be transferred.

(3) While (unprocessed data)

(4) Begin

(5) If (the first of the CRC registers is 1)

(6) reg = reg XOR 0x31

(7) the CRC register moves one bit left and reads a new data in the 0 bit position of the CRC register.

(8) End

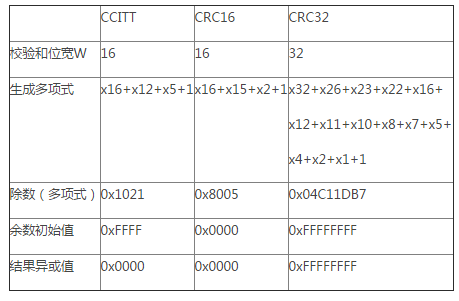
(9) the CRC register is the remainder we require.

In fact, real CRC calculations are usually somewhat different from those described above. This is because the most basic CRC Division has an obvious defect, which is that adding some 0 to the beginning of the data stream does not affect the result of the final verifying word. This problem is very annoying, so the real application of the CRC algorithm basically has made minor changes on the basis of the original CRC algorithm.

The so-called changes, that is, add two concepts, the first is the "remainder initial value", the second is the "result or value".

The so-called "remainder initial value" is the beginning of the calculation of the CRC value, which gives the CRC register an initial value. "Result or value" is the final check value of the value of the CRC register after the rest of the calculation and the value of the register with this value.

The three common CRC standards are used for each of the following tables.



After adding these deformations, the common algorithmic description forms the same way:

(1) set the CRC register and assign it "the initial value of the remainder".

(2) the first 8-bit character of the data is different from the CRC register, and the result is stored in the CRC register.

(3) the CRC register moves to one bit to the right, MSB makes up the zero, moves out and checks the LSB.

(4) if LSB is 0, repeat the third step; if LSB is 1, the CRC register is different from the 0x31.

(5) repeat the third and fourth steps until the 8 shift is complete. At this point, a 8-bit data processing is completed.

(6) repeat second to fifth steps until all data are processed.

(7) the content of the final CRC register is CRC value after or without the "result or value".

The example C code is shown below, because the efficiency is very low and the code should be avoided in the project, such as the request for computing time. However, this code is a common computing code is better than online, because this code is a CRC parameter, the CRC can be the last calculation results to the initial calculation as the function value in the calculation is very useful for large data blocks CRC, do not need once all data will be read into memory, but read part one, read after the calculation. This is very useful for a memory constrained system.

1. #define POLY        0x1021
2. /\*\*
3. \* Calculating CRC-16 in 'C'
4. \* @para addr, start of data
5. \* @para num, length of data
6. \* @para crc, incoming CRC
7. \*/
8. uint16\_t crc16(unsigned **char** \*addr, **int** num, uint16\_t crc)
9. {
10. **int** i;
11. **for** (; num > 0; num--)              /\* Step through bytes in memory \*/
12. {
13. crc = crc ^ (\*addr++ << 8);     /\* Fetch byte from memory, XOR into CRC top byte\*/
14. **for** (i = 0; i < 8; i++)             /\* Prepare to rotate 8 bits \*/
15. {
16. **if** (crc & 0x8000)            /\* b15 is set... \*/
17. crc = (crc << 1) ^ POLY;    /\* rotate and XOR with polynomic \*/
18. **else**                          /\* b15 is clear... \*/
19. crc <<= 1;                  /\* just rotate \*/
20. }                             /\* Loop for 8 bits \*/
21. crc &= 0xFFFF;                  /\* Ensure CRC remains 16-bit value \*/
22. }                               /\* Loop until num=0 \*/
23. **return**(crc);                    /\* Return updated CRC \*/
24. }

The above code is what I found from <http://mdfs.net/Info/Comp/Comms/CRC16.htm> , but the original code was wrong, and I made some minor modifications.

Here is an example fragment code for this function:

1. unsigned **char** data1[] = {'1', '2', '3', '4', '5', '6', '7', '8', '9'};
2. unsigned **char** data2[] = {'5', '6', '7', '8', '9'};
3. unsigned **short** c1, c2;
4. c1 = crc16(data1, 9, 0xffff);
5. c2 = crc16(data1, 4, 0xffff);
6. c2 = crc16(data2, 5, c2);
7. printf("%04x\n", c1);
8. printf("%04x\n", c2);

The reader can check that the results of C1 and C2 are all 29b1. The initial value of CRC in the above code is 0xFFFF because the initial value of the divisor required by the CCITT standard is 0xFFFF.

The above algorithm calculates bit by bit data stream with low efficiency. In fact, after careful analysis of the mathematical properties of CRC computing, we can have multiple bits and multiple computing, and the most commonly used is a fast algorithm for checking tables by bytes. The algorithm is based on the fact that calculating the CRC code after the byte is equal to the 8 bit left shift 8 bits of the last byte CRC code, plus the last byte CRC moves to the CRC code after the sum of 8 bits and the sum of this byte. If we compute all the CRC numbers of 8 binary sequence numbers (a total of 256) and put them in a table, we only need to find the corresponding values from the table for processing.