



首页

博客

学院

下载

图文课

论坛

APP

问答

商城

VIP会员

活动

招聘

ITeye

GitChat

搜博主文章



写博客

赚零钱



消息

登录

注册



CRC 算法的简单说明

2018年05月17日 16:03:04 bangbang170 阅读数：10267



2



2



循环冗余校验（CRC）算法入门引导

2012年08月19日 12:42:34阅读数：167959

写给嵌入式程序员的循环冗余校验（CRC）算法入门引导

前言

CRC校验（循环冗余校验）是数据通讯中最常采用的校验方式。在嵌入式软件开发中，经常要用到CRC 算法对各种数据进行校验。因此，掌握基本的CRC算法应是嵌入式程序员的基本技能。可是，嵌入式程序员中能真正掌握CRC算法的人却很少，平常在项目中见到的CRC的代码多数都是那种效率非常低下的实现方式。

其实，在网上有一篇介绍CRC 算法的非常好的文章，作者是Ross Williams，题目叫：“A PAINLESS GUIDE TO CRC ERROR DETECTION ALGORITHMS”。我常将这篇文章推荐给向我询问C语言的朋友，但不少朋友向我抱怨原文太长了，而且是英文的。希望我能写篇短点的文章，因此就有了本文。不过，我的水平比不了Ross Williams，我的文章肯定也没Ross Williams的写的好。因此，阅读英文没耐心的朋友还是去Ross Williams的原文吧。

本文的读者群设定为软件开发人员，尤其是从事嵌入式软件开发的程序员，而不是专业从事数学或通讯领域研究的学者（我也没有这个水平写的这么高深）。因此，本文的目标是介绍CRC算法的基本原理和实现方法，用到的数学尽量控制在高中生可以理解的深度。

另外，鉴于大多数嵌入式程序员都是半路出家转过来的，不少人只会C语言。因此，文中的示例代码全部采用C语言来实现。作为一篇入门短文，文中给出的代码更注重于示范性，尽可能的保持易读性。因此，文中的代码并不追求最高效的实现，但对于一般的应用却也足够快速了。

从奇偶校验说起

所谓通讯过程的校验是指在通讯数据后加上一些附加信息，通过这些附加信息来判断接收到的数据是否和发送出的数据相同。比如说RS232串行通讯可以设置奇偶校验位，所谓奇偶校验就是在发送的每一个字节后都加上一位，使得每个字节中1的个数为奇数个或偶数个。比如我们要发送的字节是0x1a，二进制表示为0001 1010。

采用奇校验，则在数据后补上个0，数据变为0001 1010 0，数据中1的个数为奇数个（3个）

采用偶校验，则在数据后补上个1，数据变为0001 1010 1，数据中1的个数为偶数个（4个）

接收方通过计算数据中1个数是否满足奇偶性来确定数据是否有错。

奇偶校验的缺点也很明显，首先，它对错误的检测概率大约只有50%。也就是只有一半的错误它能够检测出来。另外，每传输一个字节都要附加一位校验位，对传输效率的影响很大。因此，在高速数据通讯中很少采用奇偶校验。奇偶校验优点也很明显，它很简单，因此可以用硬件来实现，这样可以减少软件的负担。因此，奇偶校验也被广泛的应用着。

奇偶校验就先介绍到这来，之所以从奇偶校验说起，是因为这种校验方式最简单，而且后面将会知道奇偶校验其实就是CRC 校验的一种(CRC-1)。

异或和校验

2019人工智能前景解析

Python小白入门指导

数据库沙龙

春节充电计划

IT运维管理系统

算法专利中译

登录

注册



另一种常见的校验方式是累加和校验。所谓累加和校验实现方式有很多种，最常用的一种是在一次通讯数据包的最后加入一个字节的校验数据。这个字节内容为前面数据包中全部数据的忽略进位的按字节累加和。比如下面的例子：

我们要传输的信息为：6、23、4

加上校验和后的数据包：6、23、4、33

这里 33 为前三个字节的校验和。接收方收到全部数据后对前三个数据进行同样的累加计算，如果累加和与最后一个字节相同的话就认为传输的数据没有错误。

累加和校验由于实现起来非常简单，也被广泛的采用。但是这种校验方式的检错能力也比较一般，对于单字节的校验和大概有1/256 的概率将原本是错误的通讯数据误判为正确数据。之所以这里介绍这种校验，是因为CRC校验在传输数据的形式上与累加和校验是相同的，都可以表示为：通讯数据 校验字节（也可能是多个字节）

初识 CRC 算法

CRC 算法的基本思想是将传输的数据当做一个位数很长的数。将这个数除以另一个数。得到的余数作为校验数据附加到原数据后面。还以上面例子中的数据为例：

6、23、4 可以看做一个2进制数：0000011000010111 00000010

假如被除数选9，二进制表示为：1001

则除法运算可以表示为：



可以看到，最后的余数为1。如果我们将这个余数作为校验和的话，传输的数据则是：6、23、4、1

CRC 算法和这个过程有点类似，不过采用的不是上面例子中的通常的这种除法。在CRC算法中，将二进制数据流作为多项式的系数，然后进行的是多项式的乘法。还是举个例子吧。

比如说我们有两个二进制数，分别为：1101 和1011。

1101 与如下的多项式相联系： $1x^3+1x^2+0x^1+1x^0=x^3+x^2+x^0$

1011与如下的多项式相联系： $1x^3+0x^2+1x^1+1x^0=x^3+x^1+x^0$

两个多项式的乘法： $(x^3+x^2+x^0)(x^3+x^1+x^0)=x^6+x^5+x^4+x^3+x^3+x^2+x^1+x^0$

得到结果后，合并同类项时采用模2运算。也就是说乘法采用正常的多项式乘法，而加减法都采用模2运算。所谓模2运算就是结果除以2后取余数。比如3 mod 2 = 1。因此，上面最终得到的多项式为：

$x^6+x^5+x^4+x^3+x^2+x^1+x^0$ ，对应的二进制数:111111

加减法采用模2运算后其实就成了一种运算了，就是我们通常所说的异或运算：

0+0=0 0+1=1 1+0=1 1+1=0	0-0=0 1-0=1 0-1=1 1-1=0
----------------------------------	----------------------------------



关闭

除法运算与上面给出的乘法概念类似，还是遇到加减的地方都用异或运算来代替。下面是一个例子：

要传输的数据为：1101011011

除数设为：10011

在计算前先将原始数据后面填上4个0：11010110110000，之所以要补0，后面再做解释。

从这个例子可以看出，采用了模2的加减法后，不需要考虑借位的问题，所以除法变简单了。最后得到的余数就是CRC 校验字。为了进行CRC运算，也就是这种特殊的除法运算，必须要指定个被除数，在CRC算法中，这个被除数有一个专有名称叫做“生成多项式”。生成多项式的选取是个很有难度的问题，如果选的不好，那么检出错误的概率就会低很多。好在这个问题已经被专家们研究了很长一段时间了，对于我们这些使用者来说，只要把现成的成果拿来用就行了。

最常用的几种生成多项式如下：

$$\text{CRC8} = X^8 + X^5 + X^4 + X^0$$

$$\text{CRC-CCITT} = X^{16} + X^{12} + X^5 + X^0$$

$$\text{CRC16} = X^{16} + X^{15} + X^2 + X^0$$

$$\text{CRC12} = X^{12} + X^{11} + X^3 + X^2 + X^0$$

$$\text{CRC32} = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X^1 + X^0$$

有一点要特别注意，文献中提到的生成多项式经常会说到多项式的位宽（Width，[简记为W](#)），这个位宽不是多项式对应的二进制数的位数，而是位数减1。比如CRC8中用到的位宽为8的生成多项式，其实对应得二进制数有九位：100110001。另外一点，多项式表示和二进制表示都很繁琐，交流起来不方便，因此，文献中多用16进制简写法来表示，因为生成多项式的最高位肯定为1，最高位的位置由位宽可知，故在简记式中，最高的1统一去掉了，如CRC32的生成多项式简记为04C11DB7实际上表示的是104C11DB7。当然，这样简记除了方便外，在编程计算时也有它的用处。

对于上面的例子，位宽为4（W=4），按照CRC算法的要求，计算前要在原始数据后填上W个0，也就是4个0。

位宽W=1的生成多项式(CRC1)有两种，分别是 X^1 和 $X^1 + X^0$ ，读者可以自己证明10 对应的就是奇偶校验中的奇校验，而11对应则是偶校验。因此，写到这里我们知道了奇偶校验其实就是CRC校验的一种特例，这也我要以奇偶校验作为开篇介绍的原因了。

CRC算法的编程实现

说了这么多总算到了核心部分了。从前面的介绍我们知道CRC校验核心就是实现无借位的除法运算。下面还是通过一个例子来说明如何实现CRC校验。

假设我们的生成多项式为：100110001（简记为0x31），也就是CRC-8

则计算步骤如下：

（1） 将CRC寄存器（8-bits，比生成多项式少1bit）赋初值0

（2） 在待传输信息流后面加入8个0

[2019人工智能前景解析](#)[Python小白入门指导](#)[数据库沙龙](#)[春节充电计划](#)[IT运维管理系统](#)[算法专利申请](#)[登录](#)[注册](#)[×](#)

- (4) Begin
- (5) If (CRC寄存器首位是1)
- (6) reg = reg XOR 0x31
- (7) CRC寄存器左移一位，读入一个新的数据于CRC寄存器的0 bit的位置。
- (8) End
- (9) CRC寄存器就是我们所要求的余数。

实际上，真正的CRC 计算通常与上面描述的还有些出入。这是因为这种最基本的CRC除法有个很明显的缺陷，就是数据流的开头添加一些0并不影响最后校验字的结果。这个问题很让人恼火啊，因此真正应用的CRC 算法基本都在原始的CRC算法的基础上做了些小的改动。

所谓的改动，也就是增加了两个概念，第一个是“余数初始值”，第二个是“结果异或值”。

所谓的“余数初始值”就是在计算CRC值的开始，给CRC寄存器一个初始值。“结果异或值”是在其余计算完成后将CRC寄存器的值在与这个值进行一下异或操作作为最后的校验值。

常见的三种CRC 标准用到个各个参数如下表。

	CCITT	CRC16	CRC32
校验和位宽W	16	16	32
生成多项式	$x^{16}+x^{12}+x^5+1$	$x^{16}+x^{15}+x^2+1$	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+1$
除数（多项式）	0x1021	0x8005	0x04C11DB7
余数初始值	0xFFFF	0x0000	0xFFFFFFFF
结果异或值	0x0000	0x0000	0xFFFFFFFF

加入这些变形后，常见的算法描述形式就成了这个样子了：

- (1) 设置CRC寄存器，并给其赋值为“余数初始值”。
- (2) 将数据的第一个8-bit字符与CRC寄存器进行异或，并把结果存入CRC寄存器。
- (3) CRC寄存器向右移一位，MSB补零，移出并检查LSB。

(4) 如果LSB为1，则CRC寄存器与0x31异或；否则CRC寄存器与0x31左移一位。

2019人工智能前景解析

Python小白入门指导

数据库沙龙

春节充电计划

IT运维管理系统

算法专利申请

登录

注册

×

- (6) 重复第2至第5步直到所有数据全部处理完成。
- (7) 最终CRC寄存器的内容与“结果异或值”进行或非操作后即为CRC值。

示例性的C代码如下所示，因为效率很低，项目中如对计算时间有要求应该避免采用这样的代码。不过这个代码已经比网上常见的计算代码要好了，因为这个代码有一个crc的参数，可以将上次计算的crc结果传入函数中作为这次计算的初始值，这对大数据块的CRC计算是很有用的，不需要一次将所有数据读入内存，而是读一部分算一次，全读完后就计算完了。这对内存受限系统还是很有用的。

```
[cpp]
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. }
```

上面的代码是我从<http://mdfs.net/Info/Comp/Comms/CRC16.htm>找到的，不过原始代码有错误，我做了些小的修改。

下面对这个函数给出个例子片段代码：

```
[cpp]
1. unsigned char data1[] = {'1', '2', '3', '4', '5', '6', '7', '8', '9'};
2. unsigned char data2[] = {'5', '6', '7', '8', '9'};
```

```
5. c2 = crc16(data1, 4, 0xffff);
6. c2 = crc16(data2, 5, c2);
7. printf("%04x\n", c1);
8. printf("%04x\n", c2);
```

读者可以验算，c1、c2 的结果都为 29b1。上面代码中crc 的初始值之所以为0xffff，是因为CCITT标准要求的除数初始值就是0xffff。

上面的算法对数据流逐位进行计算，效率很低。实际上仔细分析CRC计算的数学性质后我们可以多位多位计算，最常用的是一种按字节查表的快速算法。该算法基于这样一个事实：计算本字节后的CRC码，等于上一字节余式CRC码的低8位左移8位，加上上一字节CRC右移 8位和本字节之和后所求得的CRC码。如果我们把8位二进制序列数的CRC(共256个)全部计算出来，放在一个表里，编码时只要从表中查找对应的值进行处理即可。

按照这个方法，可以有如下的代码（这个代码也不是我写的，是我在Micbael Barr的书“Programming Embedded Systems in C and C++”中找到的，同样，我做了点小小的改动。）：

```
[cpp]
1. /*
2.  crc.h
3.  */
4.
5.  #ifndef CRC_H_INCLUDED
6.  #define CRC_H_INCLUDED
7.
8.  /*
9.   * The CRC parameters. Currently configured for CCITT.
10.  * Simply modify these to switch to another CRC Standard.
11.  */
12. /*
13.  #define POLYNOMIAL      0x8005
14.  #define INITIAL_REMAINDER 0x0000
15.  #define FINAL_XOR_VALUE  0x0000
16.  */
17.  #define POLYNOMIAL      0x1021
18.  #define INITIAL_REMAINDER 0xFFFF
19.  #define FINAL_XOR_VALUE  0x0000
20.
21.  /*
22.  #define POLYNOMIAL      0x1021
23.  #define POLYNOMIAL      0xA001
24.  #define INITIAL_REMAINDER 0xFFFF
25.  #define FINAL_XOR_VALUE  0x0000
26.  */
27.
28.  */
```



关闭

[2019人工智能前景解析](#)[Python小白入门指导](#)[数据库沙龙](#)[春节充电计划](#)[IT运维管理系统](#)[算法专利申请](#)[登录](#)[注册](#)

×

```
31. */
32. typedef unsigned short width_t;
33. #define WIDTH (8 * sizeof(width_t))
34. #define TOPBIT (1 << (WIDTH - 1))
35.
36. /**
37.  * Initialize the CRC lookup table.
38.  * This table is used by crcCompute() to make CRC computation faster.
39.  */
40. void crcInit(void);
41.
42. /**
43.  * Compute the CRC checksum of a binary message block.
44.  * @para message, 用来计算的数据
45.  * @para nBytes, 数据的长度
46.  * @note This function expects that crcInit() has been called
47.  *       first to initialize the CRC lookup table.
48.  */
49. width_t crcCompute(unsigned char * message, unsigned int nBytes);
50.
51. #endif // CRC_H_INCLUDED
```



```
[cpp]
1. /*
2.  *crc.c
3.  */
4.
5. #include "crc.h"
6. /*
7.  * An array containing the pre-computed intermediate result for each
8.  * possible byte of input. This is used to speed up the computation.
9.  */
10. static width_t crcTable[256];
11.
12. /**
13.  * Initialize the CRC lookup table.
14.  * This table is used by crcCompute() to make CRC computation faster.
15.  */
16. void crcInit(void)
17. {
```



关闭

```
20. int bit;
21. /* Perform binary long division, a bit at a time. */
22. for(dividend = 0; dividend < 256; dividend++)
23. {
24.     /* Initialize the remainder. */
25.     remainder = dividend << (WIDTH - 8);
26.     /* Shift and XOR with the polynomial. */
27.     for(bit = 0; bit < 8; bit++)
28.     {
29.         /* Try to divide the current data bit. */
30.         if(remainder & TOPBIT)
31.         {
32.             remainder = (remainder << 1) ^ POLYNOMIAL;
33.         }
34.         else
35.         {
36.             remainder = remainder << 1;
37.         }
38.     }
39.     /* Save the result in the table. */
40.     crcTable[dividend] = remainder;
41. }
42. } /* crcInit() */
43.
44. /**
45.  * Compute the CRC checksum of a binary message block.
46.  * @para message, 用来计算的数据
47.  * @para nBytes, 数据的长度
48.  * @note This function expects that crcInit() has been called
49.  *       first to initialize the CRC lookup table.
50.  */
51. width_t crcCompute(unsigned char * message, unsigned int nBytes)
52. {
53.     unsigned int offset;
54.     unsigned char byte;
55.     width_t remainder = INITIAL_REMAINDER;
56.     /* Divide the message by the polynomial, a byte at a time. */
57.     for( offset = 0; offset < nBytes; offset++)
58.     {
59.         byte = (remainder >> (WIDTH - 8)) ^ message[offset];
60.         remainder = crcTable[byte] ^ (remainder << 8);
61.     }
```



关闭


```
64. } /* crcCompute() */
```

上面代码中crcInit() 函数用来计算crcTable，因此在调用 crcCompute 前必须先调用 crcInit()。不过，对于嵌入式系统，RAM是很紧张的，最好将 crcTable 提前算好，作为常量数据存到程序存储区而不占用RAM空间。CRC 计算实际上还有很多内容可以介绍，不过对于一般的程序员来说，知道这些也就差不多了。余下的部分以后有时间了我再写文章来介绍吧。

最后，给出个 C++ 代码，实现了 CRC8、CRC16 和 CRC32 的计算。收集了常见的各种 CRC 系数。代码可以从这里下载：https://code.csdn.net/liyuanbhu/crc_compute/tree/master

```
[cpp]
1. #ifndef CRCCOMPUTE_H
2. #define CRCCOMPUTE_H
3.
4. #include <stdint.h>
5.
6. template <typename TYPE> class CRC
7. {
8. public:
9.     CRC();
10.    CRC(TYPE polynomial, TYPE init_remainder, TYPE final_xor_value);
11.    void build(TYPE polynomial, TYPE init_remainder, TYPE final_xor_value);
12.    /**
13.     * Compute the CRC checksum of a binary message block.
14.     * @para message, 用来计算的数据
15.     * @para nBytes, 数据的长度
16.     */
17.    TYPE crcCompute(char * message, unsigned int nBytes);
18.    TYPE crcCompute(char * message, unsigned int nBytes, bool reinit);
19. protected:
20.    TYPE m_polynomial;
21.    TYPE m_initial_remainder;
22.    TYPE m_final_xor_value;
23.    TYPE m_remainder;
24.    TYPE crcTable[256];
25.    int m_width;
26.    int m_topbit;
27.    /**
28.     * Initialize the CRC lookup table.
29.     * This table is used by crcCompute() to make CRC computation faster.
30.     */
31.    void crcInit(void);
```

```
34. template <typename TYPE>
35. CRC<TYPE>::CRC()
36. {
37.     m_width = 8 * sizeof(TYPE);
38.     m_topbit = 1 << (m_width - 1);
39. }
40.
41. template <typename TYPE>
42. CRC<TYPE>::CRC(TYPE polynomial, TYPE init_remainder, TYPE final_xor_value)
43. {
44.     m_width = 8 * sizeof(TYPE);
45.     m_topbit = 1 << (m_width - 1);
46.     m_polynomial = polynomial;
47.     m_initial_remainder = init_remainder;
48.     m_final_xor_value = final_xor_value;
49.
50.     crcInit();
51. }
52.
53. template <typename TYPE>
54. void CRC<TYPE>::build(TYPE polynomial, TYPE init_remainder, TYPE final_xor_value)
55. {
56.     m_polynomial = polynomial;
57.     m_initial_remainder = init_remainder;
58.     m_final_xor_value = final_xor_value;
59.
60.     crcInit();
61. }
62.
63. template <typename TYPE>
64. TYPE CRC<TYPE>::crcCompute(char * message, unsigned int nBytes)
65. {
66.     unsigned int offset;
67.     unsigned char byte;
68.     TYPE remainder = m_initial_remainder;
69.     /* Divide the message by the polynomial, a byte at a time. */
70.     for( offset = 0; offset < nBytes; offset++)
71.     {
72.         byte = (remainder >> (m_width - 8)) ^ message[offset];
73.         remainder = crcTable[byte] ^ (remainder << 8);
74.     }
75.     /* The final remainder is the CRC result. */
```



关闭

[2019人工智能前景解析](#)[Python小白入门指导](#)[数据库沙龙](#)[春节充电计划](#)[IT运维管理系统](#)[算法专利解读](#)[登录](#)[注册](#)

×

```
78.
79. template <typename TYPE>
80. TYPE CRC<TYPE>::crcCompute(char * message, unsigned int nBytes, bool reinit)
81. {
82.     unsigned int offset;
83.     unsigned char byte;
84.     if(reinit)
85.     {
86.         m_remainder = m_initial_remainder;
87.     }
88.     /* Divide the message by the polynomial, a byte at a time. */
89.     for( offset = 0; offset < nBytes; offset++)
90.     {
91.         byte = (m_remainder >> (m_width - 8)) ^ message[offset];
92.         m_remainder = crcTable[byte] ^ (m_remainder << 8);
93.     }
94.     /* The final remainder is the CRC result. */
95.     return (m_remainder ^ m_final_xor_value);
96. }
97.
98. class CRC8 : public CRC<uint8_t>
99. {
100. public:
101.     enum CRC8_TYPE {eCRC8, eAUTOSAR, eCDMA2000, eDARC, eDVB_S2, eEBU, eAES, eGSM_A, eGSM_B, eI_CODE,
102.                     eITU, eLTE, eMAXIM, eOPENSAFETY, eROHC, eSAE_J1850, eWCDMA};
103.     CRC8(CRC8_TYPE type);
104.     CRC8(uint8_t polynomial, uint8_t init_remainder, uint8_t final_xor_value)
105.         :CRC<uint8_t>(polynomial, init_remainder, final_xor_value){}
106. };
107.
108. class CRC16 : public CRC<uint16_t>
109. {
110. public:
111.     enum CRC16_TYPE {eCCITT, eKERMIT, eCCITT_FALSE, eIBM, eARC, eLHA, eSPI_FUJITSU,
112.                     eBUYPASS, eVERIFONE, eUMTS, eCDMA2000, eCMS, eDDS_110, eDECT_R,
113.                     eDECT_X, eDNP, eEN_13757, eGENIBUS, eEPC, eDARC, eI_CODE, eGSM,
114.                     eLJ1200, eMAXIM, eMCRF4XX, eOPENSAFETY_A, eOPENSAFETY_B, ePROFIBUS,
115.                     eIEC_61158_2, eRIELLO, eT10_DIF, eTELEDISK, eTMS37157, eUSB,
116.                     eCRC_A, eMODBUS, eX_25, eCRC_B, eISO_HDLC, eIBM_SDL_C, eXMODEM,
117.                     eZMODEM, eACORN, eLTE};
118.     CRC16(CRC16_TYPE type);
119.     CRC16(uint16_t polynomial, uint16_t init_remainder, uint16_t final_xor_value)
```



关闭

```
122.
123. class CRC32 : public CRC<uint32_t>
124. {
125. public:
126.     enum CRC32_TYPE {eADCCP, ePKZIP, eCRC32, eAAL5, eDECT_B, eB_CRC32, eBZIP2, eAUTOSAR,
127.                     eCRC32C, eCRC32D, eMPEG2, ePOSIX, eCKSUM, eCRC32Q, eJAMCRC, eXFER};
128.     CRC32(CRC32_TYPE type);
129. };
130.
131.
132. #endif // CRCCOMPUTE_H
```

```
[cpp]
1. #include "crcCompute.h"
2.
3. template <typename TYPE>
4. void CRC<TYPE>::crclnit(void)
5. {
6.     TYPE remainder;
7.     TYPE dividend;
8.     int bit;
9.     /* Perform binary long division, a bit at a time. */
10.    for(dividend = 0; dividend < 256; dividend++)
11.    {
12.        /* Initialize the remainder. */
13.        remainder = dividend << (m_width - 8);
14.        /* Shift and XOR with the polynomial. */
15.        for(bit = 0; bit < 8; bit++)
16.        {
17.            /* Try to divide the current data bit. */
18.            if(remainder & m_topbit)
19.            {
20.                remainder = (remainder << 1) ^ m_polynomial;
21.            }
22.            else
23.            {
24.                remainder = remainder << 1;
25.            }
26.        }
27.        /* Save the result in the table. */
28.    }
```

```
31.
32. CRC8::CRC8(CRC8_TYPE type)
33. {
34.     switch (type)
35.     {
36.     case eCRC8:
37.         m_polynomial = 0x07; //http://reveng.sourceforge.net/crc-catalogue/all.htm
38.         m_initial_remainder = 0x00;
39.         m_final_xor_value = 0x00;
40.         break;
41.     case eAUTOSAR:
42.         m_polynomial = 0x2f;
43.         m_initial_remainder = 0xff;
44.         m_final_xor_value = 0xff;
45.         break;
46.     case eCDMA2000:
47.         m_polynomial = 0x9b;
48.         m_initial_remainder = 0xFF;
49.         m_final_xor_value = 0x00;
50.         break;
51.     case eDARC:
52.         m_polynomial = 0x39;
53.         m_initial_remainder = 0x00;
54.         m_final_xor_value = 0x00;
55.         break;
56.     case eDVB_S2:
57.         m_polynomial = 0xd5;
58.         m_initial_remainder = 0x00;
59.         m_final_xor_value = 0x00;
60.         break;
61.     case eEBU:
62.     case eAES:
63.         m_polynomial = 0x1d;
64.         m_initial_remainder = 0xFF;
65.         m_final_xor_value = 0x00;
66.         break;
67.     case eGSM_A:
68.         m_polynomial = 0x1d;
69.         m_initial_remainder = 0x00;
70.         m_final_xor_value = 0x00;
71.         break;
72.     case eGSM_B:
```



关闭

2019人工智能前景解析

Python小白入门指导

数据库沙龙

春节充电计划

IT运维管理系统

算法专利中译

登录

注册

×

```
75.     m_final_xor_value = 0xFF;
76.     break;
77. case eI_CODE:
78.     m_polynomial = 0x1d;
79.     m_initial_remainder = 0xFD;
80.     m_final_xor_value = 0x00;
81.     break;
82. case eITU:
83.     m_polynomial = 0x07;
84.     m_initial_remainder = 0x00;
85.     m_final_xor_value = 0x55;
86.     break;
87. case eLTE:
88.     m_polynomial = 0x9b;
89.     m_initial_remainder = 0x00;
90.     m_final_xor_value = 0x00;
91.     break;
92. case eMAXIM:
93.     m_polynomial = 0x31;
94.     m_initial_remainder = 0x00;
95.     m_final_xor_value = 0x00;
96.     break;
97. case eOPENSAFETY:
98.     m_polynomial = 0x2f;
99.     m_initial_remainder = 0x00;
100.    m_final_xor_value = 0x00;
101.    break;
102. case eROHC:
103.    m_polynomial = 0x07;
104.    m_initial_remainder = 0xff;
105.    m_final_xor_value = 0x00;
106.    break;
107. case eSAE_J1850:
108.    m_polynomial = 0x1d;
109.    m_initial_remainder = 0xff;
110.    m_final_xor_value = 0xff;
111.    break;
112. case eWCDMA:
113.    m_polynomial = 0x9b;
114.    m_initial_remainder = 0x00;
115.    m_final_xor_value = 0x00;
116.    break;
```



关闭

[2019人工智能前景解析](#)[Python小白入门指导](#)[数据库沙龙](#)[春节充电计划](#)[IT运维管理系统](#)[算法专利中译](#)[登录](#)[注册](#)

×

```
119.     m_initial_remainder = 0x00;
120.     m_final_xor_value = 0x00;
121.     break;
122. }
123. crcInit();
124.
125. }
126.
127. CRC16::CRC16(CRC16_TYPE type)
128. {
129.     switch (type)
130.     {
131.     case eCCITT_FALSE:
132.     case eMCRF4XX:
133.         m_polynomial = 0x1021;
134.         m_initial_remainder = 0xFFFF;
135.         m_final_xor_value = 0x0000;
136.         break;
137.     case eIBM:
138.     case eARC:
139.     case eLHA:
140.     case eBUYPASS:
141.     case eVERIFONE:
142.     case eUMTS:
143.         m_polynomial = 0x8005;
144.         m_initial_remainder = 0x0000;
145.         m_final_xor_value = 0x0000;
146.         break;
147.     case eSPI_FUJITSU:
148.         m_polynomial = 0x1021;
149.         m_initial_remainder = 0x1d0f;
150.         m_final_xor_value = 0x0000;
151.         break;
152.     case eCCITT:
153.     case eKERMIT:
154.     case eXMODEM:
155.     case eZMODEM:
156.     case eACORN:
157.     case eLTE:
158.         m_polynomial = 0x1021;
159.         m_initial_remainder = 0x0000;
160.         m_final_xor_value = 0x0000;
```



关闭

[2019人工智能前景解析](#)[Python小白入门指导](#)[数据库沙龙](#)[春节充电计划](#)[IT运维管理系统](#)[算法专利](#)[登录](#)[注册](#)

×

```
163.     m_polynomial = 0xc867;
164.     m_initial_remainder = 0xffff;
165.     m_final_xor_value = 0x0000;
166.     break;
167. case eCMS:
168. case eMODBUS:
169.     m_polynomial = 0x8005;
170.     m_initial_remainder = 0xffff;
171.     m_final_xor_value = 0x0000;
172.     break;
173. case eDDS_110:
174.     m_polynomial = 0x8005;
175.     m_initial_remainder = 0x800d;
176.     m_final_xor_value = 0x0000;
177.     break;
178. case eDECT_R:
179.     m_polynomial = 0x0589;
180.     m_initial_remainder = 0x0000;
181.     m_final_xor_value = 0x0001;
182.     break;
183. case eDECT_X:
184.     m_polynomial = 0x0589;
185.     m_initial_remainder = 0x0000;
186.     m_final_xor_value = 0x0000;
187.     break;
188. case eDNP:
189. case eEN_13757:
190.     m_polynomial = 0x3d65;
191.     m_initial_remainder = 0x0000;
192.     m_final_xor_value = 0xffff;
193.     break;
194. case eGENIBUS:
195. case eEPC:
196. case eDARC:
197. case eI_CODE:
198. case eX_25:
199. case eCRC_B:
200. case eISO_HDLC:
201. case eIBM_SDLC:
202.     m_polynomial = 0x1021;
203.     m_initial_remainder = 0xffff;
204.     m_final_xor_value = 0xffff;
```



关闭

2019人工智能前景解析

Python小白入门指导

数据库沙龙

春节充电计划

IT运维管理系统

算法专利中法

登录

注册

×


```
207.     m_polynomial = 0x1021;
208.     m_initial_remainder = 0x0000;
209.     m_final_xor_value = 0xffff;
210.     break;
211. case eLJ1200:
212.     m_polynomial = 0x6f63;
213.     m_initial_remainder = 0x0000;
214.     m_final_xor_value = 0x0000;
215.     break;
216. case eMAXIM:
217.     m_polynomial = 0x8005;
218.     m_initial_remainder = 0x0000;
219.     m_final_xor_value = 0xffff;
220.     break;
221. case eOPENSAFETY_A:
222.     m_polynomial = 0x5935;
223.     m_initial_remainder = 0x0000;
224.     m_final_xor_value = 0x0000;
225.     break;
226. case eOPENSAFETY_B:
227.     m_polynomial = 0x755b;
228.     m_initial_remainder = 0x0000;
229.     m_final_xor_value = 0x0000;
230.     break;
231. case ePROFIBUS:
232. case eIEC_61158_2:
233.     m_polynomial = 0x1dcf;
234.     m_initial_remainder = 0xffff;
235.     m_final_xor_value = 0xffff;
236.     break;
237. case eRIELLO:
238.     m_polynomial = 0x1021;
239.     m_initial_remainder = 0xb2aa;
240.     m_final_xor_value = 0x0000;
241.     break;
242. case eT10_DIF:
243.     m_polynomial = 0x8bb7;
244.     m_initial_remainder = 0x0000;
245.     m_final_xor_value = 0x0000;
246.     break;
247. case eTELEDISK:
248.     m_polynomial = 0xa097;
```



关闭

2019人工智能前景解析

Python小白入门指导

数据库沙龙

春节充电计划

IT运维管理系统

算法专利中法

登录

注册

×

```
251.     break;
252. case eTMS37157:
253.     m_polynomial = 0x1021;
254.     m_initial_remainder = 0x89ec;
255.     m_final_xor_value = 0x0000;
256.     break;
257. case eUSB:
258.     m_polynomial = 0x8005;
259.     m_initial_remainder = 0xffff;
260.     m_final_xor_value = 0xffff;
261.     break;
262. case eCRC_A:
263.     m_polynomial = 0x1021;
264.     m_initial_remainder = 0xc6c6;
265.     m_final_xor_value = 0x0000;
266.     break;
267. default:
268.     m_polynomial = 0x8005;
269.     m_initial_remainder = 0x0000;
270.     m_final_xor_value = 0x0000;
271.     break;
272. }
273. crcInit();
274. }
275.
276.
277. CRC32::CRC32(CRC32_TYPE type)
278. {
279.     switch (type)
280.     {
281.     case eADCCP:
282.     case ePKZIP:
283.     case eCRC32:
284.     case eBZIP2:
285.     case eAAL5:
286.     case eDECT_B:
287.     case eB_CRC32:
288.         m_polynomial = 0x04c11db7;
289.         m_initial_remainder = 0xFFFFFFFF;
290.         m_final_xor_value = 0xFFFFFFFF;
291.         break;
292.     case eAUTOSAR:
```



关闭

[2019人工智能前景解析](#)[Python小白入门指导](#)[数据库沙龙](#)[春节充电计划](#)[IT运维管理系统](#)[算法专利申请](#)[登录](#)[注册](#)

×

```
295.     m_final_xor_value = 0xFFFFFFFF;
296.     break;
297. case eCRC32C:
298.     m_polynomial = 0x1edc6f41;
299.     m_initial_remainder = 0xFFFFFFFF;
300.     m_final_xor_value = 0xFFFFFFFF;
301.     break;
302. case eCRC32D:
303.     m_polynomial = 0xa833982b;
304.     m_initial_remainder = 0xFFFFFFFF;
305.     m_final_xor_value = 0xFFFFFFFF;
306.     break;
307. case eMPEG2:
308. case eJAMCRC:
309.     m_polynomial = 0x04c11db7;
310.     m_initial_remainder = 0xFFFFFFFF;
311.     m_final_xor_value = 0x00000000;
312.     break;
313. case ePOSIX:
314. case eCKSUM:
315.     m_polynomial = 0x04c11db7;
316.     m_initial_remainder = 0x00000000;
317.     m_final_xor_value = 0xFFFFFFFF;
318.     break;
319. case eCRC32Q:
320.     m_polynomial = 0x814141ab;
321.     m_initial_remainder = 0x00000000;
322.     m_final_xor_value = 0x00000000;
323.     break;
324. case eXFER:
325.     m_polynomial = 0x000000af;
326.     m_initial_remainder = 0x00000000;
327.     m_final_xor_value = 0x00000000;
328.     break;
329. default:
330.     m_polynomial = 0x04C11DB7;
331.     m_initial_remainder = 0xFFFFFFFF;
332.     m_final_xor_value = 0xFFFFFFFF;
333.     break;
334. }
335. crcInit();
336. }
```



关闭

```
[cpp]
1. #include <iostream>
2. #include <stdio.h>
3. #include "crcCompute.h"
4.
5. using namespace std;
6.
7. int main(int argc, char *argv[])
8. {
9.
10.     CRC16 crc16(CRC16::eCCITT_FALSE);
11.     char data1[] = {'1', '2', '3', '4', '5', '6', '7', '8', '9'};
12.     char data2[] = {'5', '6', '7', '8', '9'};
13.     unsigned short c1, c2;
14.     c1 = crc16.crcCompute(data1, 9);
15.     c2 = crc16.crcCompute(data1, 4, true);
16.     c2 = crc16.crcCompute(data2, 5, false);
17.
18.
19.     printf("%04x\n", c1);
20.     printf("%04x\n", c2);
21.
22.     return 0;
23. }
```



文章标签：算法 嵌入式 通讯 byte c

上班族都说好！为什么学琴要先租钢琴？马上点此咨询更多！

尼卡钢琴·顶新



想对作者说点什么

CRC校验原理及步骤

阅读数 7.8万

什么是CRC校验？CRC即循环冗余校验码：是数据通信领域中最常用的一种查错校验码，其特征是信息字段和校验字...

博文 来自： [D_leo的博客](#)

2019人工智能前景解析

Python小白入门指导

数据库沙龙

春节充电计划

IT运维管理系统

算法专利中法

登录

注册



关闭

CRC原理详解(附crc16校验代码)

参考链接：https://www.cnblogs.com/esestt/archive/2007/08/09/848856.htmlCyclic Redundancy Check循环...

博文

来自：追逐火焰的飞蛾

CRC的基本原理详解

CRC(CyclicRedundancyCheck)被广泛用于数据通过程中的差错检测，具有很强的检错能力。本文详细介绍了CRC...

博文

来自：TerryZjl的博客

北京本科名校面向上班族招生,学期1年,毕业就是本科!

威都·顶新

CRC算法详解

CRC(CyclicRedundancyCheck)：循环冗余检验，在链路层被广泛使用的检错技术。CRC原理介绍（通俗讲）1、发...

博文

来自：xinyuan510214的

CRC码的计算

简介CRC码详解计算简介 循环冗余码是最常用的差错控制编码方法之一，又称为CRC码，它是利用除法及余数的...

博文

来自：郭满亮 廊坊师范学

CRC校验

一、CRC原理。 CRC校验的原理非常简单，如下图所示。其中，生成多项式是利用抽象代数的一些规则推导出来...

博文

来自：一个人要像一支队伍

CRC查找表法推导及代码实现比较

2018/02/08再次更新—————...

博文

来自：huang_shiyang的

循环冗余校验（CRC）算法入门引导

写给嵌入式程序员的循环冗余校验（CRC）算法入门引导前言CRC校验（循环冗余校验）是数据通讯中最常采用的校...

博文

来自：Ivan 的专栏

50万码农评论：英语对于程序员有多重要！

不背单词和语法，老司机教你一个数学公式秒懂天下英语

CRC8讲解

转载：http://blog.csdn.net/zjli321/article/details/529984681、CRC8标准生成多项式CRC-8x8+x5+x4+10x31（...

博文

来自：jinqg的博客

卷积码（Convolutional Code）

卷积码（ConvolutionalCode）本文主要简单介绍了卷积码。关键词：卷积码生成多项式网格图状态图=====...

博文

来自：途次客的专栏

legendox

62篇文章

关注

排名:千里之外

搬砖公司董事长

46篇文章

关注

排名:千里之外

KevinBetterQ

93篇文章

关注

排名:千里之外

dabang_007

59篇文章

关注

排名:千里之外

CRC不可逆的“真谛”

博文

来自：1063

2019人工智能前景解析

Python小白入门指导

数据库沙龙

春节充电计划

IT运维管理系统

算法专利申请

登录

注册

×

广告

关闭

https://blog.csdn.net/u012923751/article/details/80352325

21/26

阅读数 159

博文 来自: [fly夏天的博客](#)

阅读数 8.3万

博文 来自： [开发随笔](#)

厚泽金融·顶新

阅读数 722

博文 来自: [上发条的博客](#)

阅读数 7880

博文 来自: [xinm1001的博客](#)

阅读数 3914

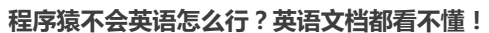
博文 来自: [z69183787的专栏](#)

阅读数 4968

博文 来自: [小侯的开发专栏](#)

阅读数 111

博文 来自: [fhqlongteng的博客](#)



不背单词和语法，一个公式教你读懂天下英文→

阅读数 705

博文 来自: wangweidong

阅读数 1026

博文 来自: [xiaozi0221的博客](#)



阅读数 523

博文 来自: [legendox的专栏](#)

阅读数 389

博文 来自：搬砖公司

算法专利审查

CRC（循环冗余）算法		阅读数 2486
CRC（循环冗余）算法简单理解1、什么是CRC码CyclicalRedundancyCheck，简称CRC。利用多项式除法及余数的...		博文 来自： 码路，心路，知行
<div></div> <div>程序猿不会英语怎么行？英语文档都看不懂！</div> <div>不背单词和语法，一个公式教你读懂天下英文→</div>		
【转】CRC原理及其逆向破解方法		阅读数 4051
转自： http://blog.163.com/j_drew/blog/static/11601844200692681123935/ 介绍：这篇短文包含CRC原理介绍...		博文 来自： dabang_007的专栏
CRC16按位计算简单代码		阅读数 4791
1．设置CRC寄存器，并给其赋值FFFF(hex)。2．将数据的第一个8-bit字符与16位CRC寄存器的低8位进行异或，并把...		博文 来自： fenglifeng1987的
Qt5.9 写的一个crc校验例子		阅读数 635
界面如下图：界面很简单从网上随便找了个crc8 crc16crc32算法直接加进去了，想实现其他算法的自己添加就可以了...		博文 来自： smaller
循环冗余检验CRC原理		阅读数 1.7万
为什么引入CRC现实的通信链路都不会是理想的。这就是说，比特在传输的过程中可能会产生差错：1可能会变成0，...		博文 来自： 菜鸟成长记
CRC 循环冗余校验码 的计算方法		阅读数 6383
循环冗余校验CRC（CyclicalRedundancyCheck）字段位于尾部，有32位，有时称为IEEE/ANSI标准的CRC32.要使...		博文 来自： My Blog
<div></div> <div>程序猿不会英语怎么行？英语文档都看不懂！</div> <div>不背单词和语法，一个公式教你读懂天下英文→</div>		
CRC8和CRC16的计算方法		阅读数 5132
CRC8转载地址： http://blog.csdn.net/d_leo/article/details/73572373 什么是CRC校验？CRC即循环冗余校验码：...		博文 来自： Chuck_lin的博客
汉明码的原理、生成和检验		阅读数 1.2万
在计算机运行过程中，由于种种原因导致数据在存储过程中可能出现差错，为了能够及时发现错误并且将错误纠正，...		博文 来自： ynd_sg的博客
简单实用的单片机CRC快速算法		阅读数 1839
1引言 CRC（循环冗余码）检验技术广泛应用于测控及通信领域。在很多情况下，CRC计算是靠专用的硬件来实现的...		博文 来自： ncdawen的专栏
简单实用的单片机CRC快速算法.rar		02-01
简单实用的单片机CRC快速算法.rar 简单实用的单片机CRC快速算法.rar		下载
CRC7校验的Verilog实现		阅读数 5442
CRC校验代码：moduleCRC_7(BITVAL,Enable,CLK,RST,CRC);inputBITVAL;//NextinputbitinputEnable;inputCLK;...		博文 来自： 小墨



关闭

金领教育 · 顶新

关于CRC校验算法及其C代码实现

阅读量 644

以CRC16作为参考：CRC16常见的标准有以下几种，被用在各个规范中，其算法原理基本一致，就是在数据的输入和...

博文 来自： 潜心钻研

CRC校验算法原理分析

阅读量 2239

CRC校验码的基本思想是利用线性编码理论，在发送端根据要传送的k位二进制码序列，以一定的规则产生一个校验...

博文 来自： 一颗偏执的心

一个比较轻松的CRC错误校验算法指南（全文翻译）

阅读量 2741

以下是我翻译的一篇关于CRC算法原理的英语文档，如果读者有什么疑问，欢迎随时在留言区咨询~~另外由于编辑器...

博文 来自： 大熊现在很幸福的

CRC32加密算法原理

阅读量 8617

一、基本原理CRC检验原理实际上就是在一个p位二进制数据序列之后附加一个r位二进制检验码(序列)，从而构成一...


博文 来自： 暴脾气的琨哥的博客

什么是CRC以及如何生成检验


阅读量 2757

在网络的信息的传输中，现实的通信链路都不会是理想的。这就是说，比特在传输过程中可能会产生差错：1可能变...

博文 来自： Number_0_0的博客

码农不会英语怎么办？英语文档都看不懂！

不背单词和语法，一个公式教你读懂天下英文→



RIP协议距离向量算法——如何更新路由表

阅读量 5104

题目：假定网络中路由器B的路由表有以下项目：目的网络距离下一跳路由N17AN22CN68FN84EN94F现在B收到C...

博文 来自： hml666888的博客

细说循环冗余校验码

阅读量 8200

初识循环冗余校验码：为了保证数据传输的可靠性，在计算机网络传输数据时，必须采用各种差错检验措施，目前广...

博文 来自： weizhengbo的博客

【计算机网络】循环冗余校验CRC算法原理&计算过程

阅读量 9854

前言我们知道，一台主机向另外一台主机发送报文的时候，需要一层层经过自己的协议栈进行数据封装，到达最后一...

博文 来自： pointer_y的博客

循环冗余校验码CRC原理和实例

阅读量 2994

今天同事问了一个CRC(循环冗余校验码)的问题，好奇心之下学习了一下。首先说它的原理，百度百科上也有，我就...


博文 来自： France_man的专栏

32位CRC FPGA Verilog并行算法

12-18

32位CRC FPGA Verilog并行算法，本人亲测，用于网络报文CRC校验项目。

下载

对于程序员来说，英语到底多重要？

不背单词和语法，一个公式秒懂英语！

Java CRC校验和算法Demo

阅读量 1111

2019人工智能前景解析

Python小白入门指导

数据库沙龙

春节充电计划

IT运维管理系统

算法专利申请

登录

注册

×



关闭

delphi编程CRC算法的实现

阅读数 2712

http://blog.csdn.net/aroc_lo/article/details/6097393 usesWindows,SysUtils,Classes;const//Crc32表Table...

博文 来自： l799623787的专栏

JAVA下两种方法实现CRC算法

08-27

JAVA下使用两种方法（ 计算法、 查表法 ）实现CRC（ XMODEM ）算法，以及验证代码

下载

IPv4编址；A类、B类、C类、D类、E类IP地址（IP地址；网络地址和主机地址；子网掩码；网关；广...

阅读数 4896

IP地址，点分十进制记法，与接口相关联，每台主机和路由器上的每个接口，必须拥有全球唯一的IP地址。点击打开...

博文 来自： Unique-You的博客

CRC32校验原理及实现

阅读数 5839

CRC即循环冗余校验(CyclicRedundancyCheck)：是数据通信领域中最常用的一种差错校验码，其特征是信息字段和...

博文 来自： 建建的博客



码农不会英语怎么办？英语文档都看不懂！

不背单词和语法，一个公式教你读懂天下英文→

STM32的CRC检验函数

阅读数 528

#include "crc_cfg.h"//---CRC32表const UINT32_T crc32Tab[256] = { 0x00000000,0x04C11DB7,0x...

博文 来自： Haiguozhe的博客

关于在STM32F103上执行CRC16出错问题

阅读数 356

在STM32上执行CRC16时计算结果与原来在407上计算的不一樣。代码如下图计算结果如下：此结果为错误结果，但...

博文 来自： august_edward的

jquery/js实现一个网页同时调用多个倒计时(最新的)

阅读数 62952

jquery/js实现一个网页同时调用多个倒计时(最新的) 最近需要网页添加多个倒计时. 查阅网络,基本上都是千遍一律的...

博文 来自： websites

人脸检测工具face_recognition的安装与应用

阅读数 20086

人脸检测工具face_recognition的安装与应用

博文 来自： roguesir的博客

R语言逻辑回归、ROC曲线和十折交叉验证

阅读数 20060

自己整理编写的逻辑回归模板，作为学习笔记记录分享。数据集用的是14个自变量Xi，一个因变量Y的australian数据...

博文 来自： Tiaaaaa的博客

强连通分量及缩点tarjan算法解析

阅读数 184795

强连通分量：简言之 就是找环（每条边只走一次，两两可达）孤立的一个点也是一个连通分量 使用tarjan算法 在...

博文 来自： 九野的博客

算法类型 算法面试 随机森林算法 stacking算法 CAVLC算法

c# crc算法 简单的说明c#委托的作用 crc校验算法c++程序 crc vc++ crc的的c++ 最简单的python教程 python最简单的教程



关闭



2019人工智能前景解析

Python小白入门指导

数据库沙龙

春节充电计划

IT运维管理系统

算法专利中译

登录

注册



原创

15

粉丝

48

喜欢

13

评论

8

等级：

博客 3

积分：

762

勋章：

恒

访问：

5万+

排名：

8万+



最新文章

【转载】如何解决fpga high fanout问题

TCP/IP协议数据包文件PCAP分析器

速度换算Gb/s

转载：verilog 可综合和不可综合语句

vivado和modesim对应的版本

个人分类

zongjie

3篇

fpga

37篇

xilinx

7篇

altera

4篇

power

2篇

展开

归档

2019年1月

3篇

