红外编解码彻底解析

1、编码格式

现有的红外遥控包括两种方式：PWM（脉冲宽度调制）和 PPM（脉冲位置调制）。

两种形式编码的代表分别为 NEC 和 PHILIPS 的 RC-5、RC-6 以及将来的 RC-7。

PWM（脉冲宽度调制）：以发射红外载波的占空比代表“0”和“1”。为了节省能量，一般情况下，发射红外载波的时间固定，通过改变不发射载波的时间来改变占空比。例如常用的电视遥控器，使用 NEC upd6121，其“0”为载波发射 0.56ms,不发射 0.56ms；其“1”为载波发射 0.56ms,不发射 1.68ms；此外，为了解码的方便，还有引导码，upd6121 的引导码为载波发射 9ms，不发射 4.5ms。upd6121 总共的编码长度为 108ms。

但并不是所有的编码器都是如此，比如 TOSHIBA 的 TC9012，其引导码为载波发射 4.5ms，不发射 4.5ms，其“0”为载波发射 0.52ms,不发射 0.52ms,其“1”为载波发射 0.52ms,不发射 1.04ms。

PPM（脉冲位置调制）：以发射载波的位置表示“0”和“1”。从发射载波到不发射载波为“0”，从不发射载波到发射载波为“1”。其发射载波和不发射载波的时间相同，都为 0.68ms，也就是每位的时间是固定的。

通过以上对编码的分析，可以得出以某种固定格式的“0”和“1”去学习红外，是很有可能不成功的。即市面上所宣传的可以学习 64 位、128 位必然是不可靠的。

另外，由于空调的状态远多于电视、音像，并且没有一个标准，所以各厂家都按自己的格式去做一个，造成差异更大。比如：美的的遥控器采用 PWM 编码，码长 120ms 左右；新科的遥控器也采用 PWM 编码，码长 500ms 左右。如此大的差异，如果按“位”的概念来讲，应该是多少位呢？64？128?显然都不可能包含如此长短不一的编码。

2、学习模式

现在用来学习红外的 CPU，无外乎以下几种：

MCS-51 系列、microchip pic16 系列、winbond w741 系列、holtek ht48 系列

以上的 CPU 由于价格便宜、使用量大，被广泛使用在遥控器上。

以上的 CPU 的基本点是：执行速度在 1us 左右，数据存储器一般为 256 个字节。如果按固定格式学习，一般可以学到 128 位（其他程序会占用一些数据存储器）；如果不按固定的格式，需要找出编码的最小公约数作为基本单位，则可以学习到的位数大大降低，达不到实用的效果。但是，即使如此，找到的最小公约数不可能满足所有的红外设备，除非最小单位为 26us(1000000/38k)。如果达到这个速度，以上 CPU 的速度远远不够，并且由于存储量的加大，数据存储器也远远不够用。

对于电视、音响等，一般使用专用的遥控芯片，比 nec,philips,toshiba,sanyo,mitsubish,panasonic 的芯片，其编码格式固定，一个键只有一个编码，学习比较容易。

而空调不一样，各家空调厂商都是按自己的要求用 cpu 做遥控芯片，编码形式就有很多种。比如可能没有引导码（电视音响类都有）、校验方式取累加和（电视音响类一般取反码）等。因为空调的状态多，必须一次发送完毕，有制冷、温度、风速、自动、定时、加湿、制热等，所以编码很长，并且同一个按键，在不同状态下发送的编码不一样，造成学习上的困难。

红外遥控编码格式

红外遥控器的编码格式通常有两种格式：NEC 和 RC5

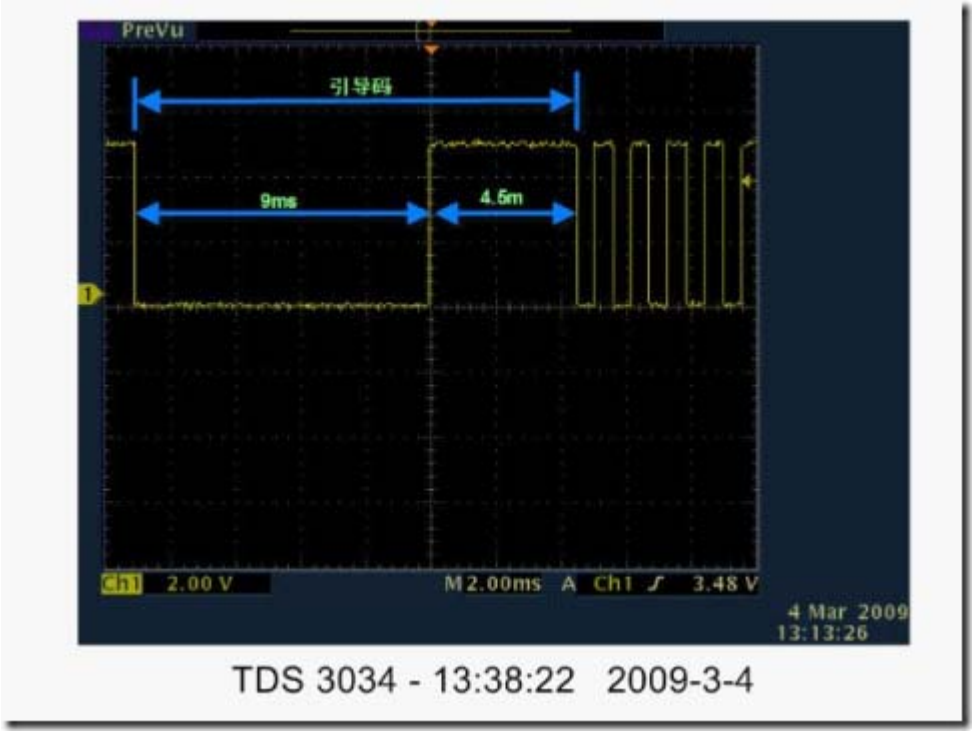
NEC 格式的特征：

1：使用 38 kHz 载波频率

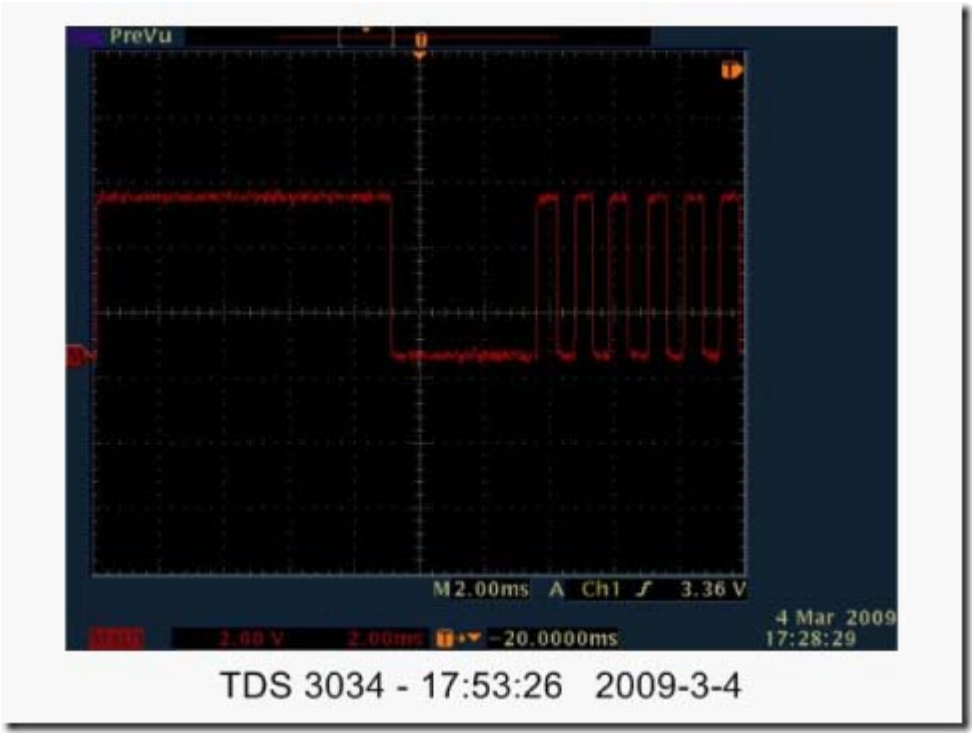
2：引导码间隔是 9 ms + 4.5 ms

3：使用 16 位客户代码

4：使用 8 位数据代码和 8 位取反的数据代码

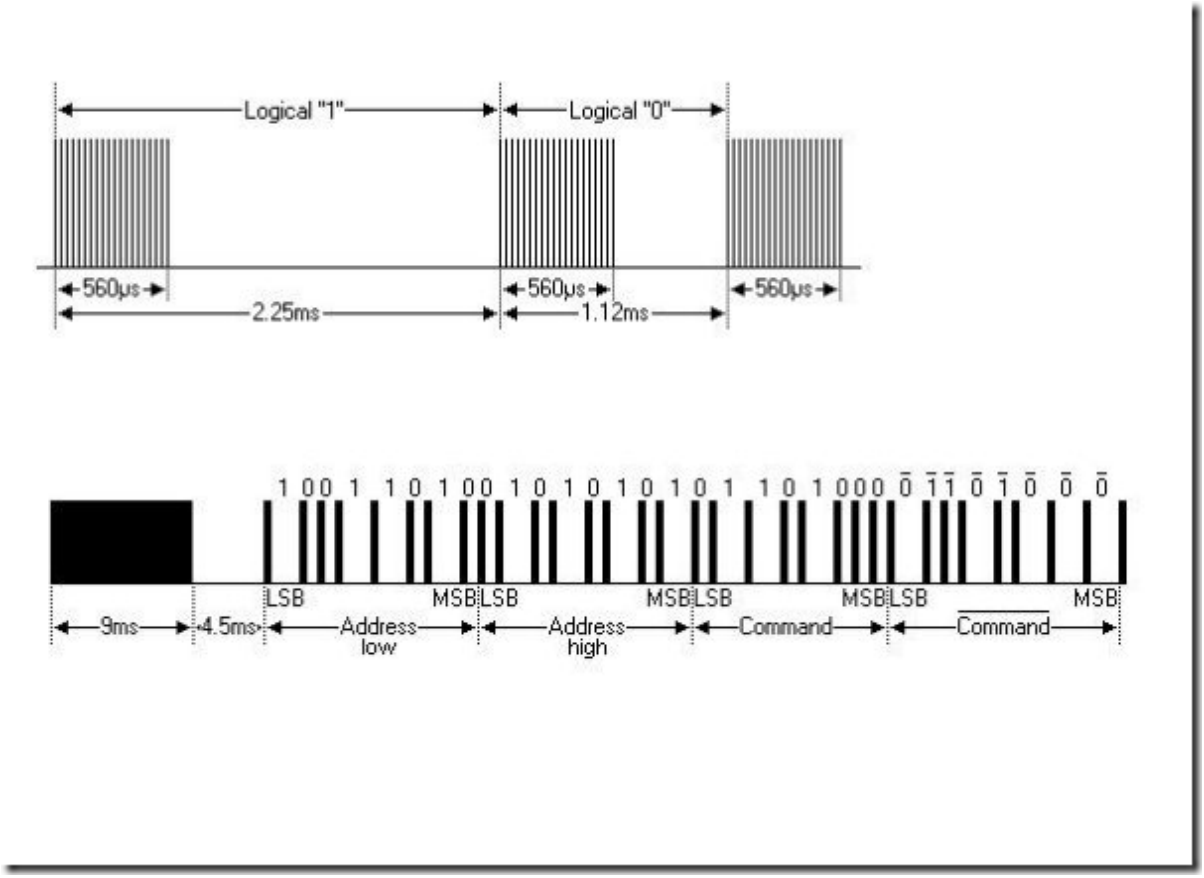


不过需要将波形反转一下才方便分析：



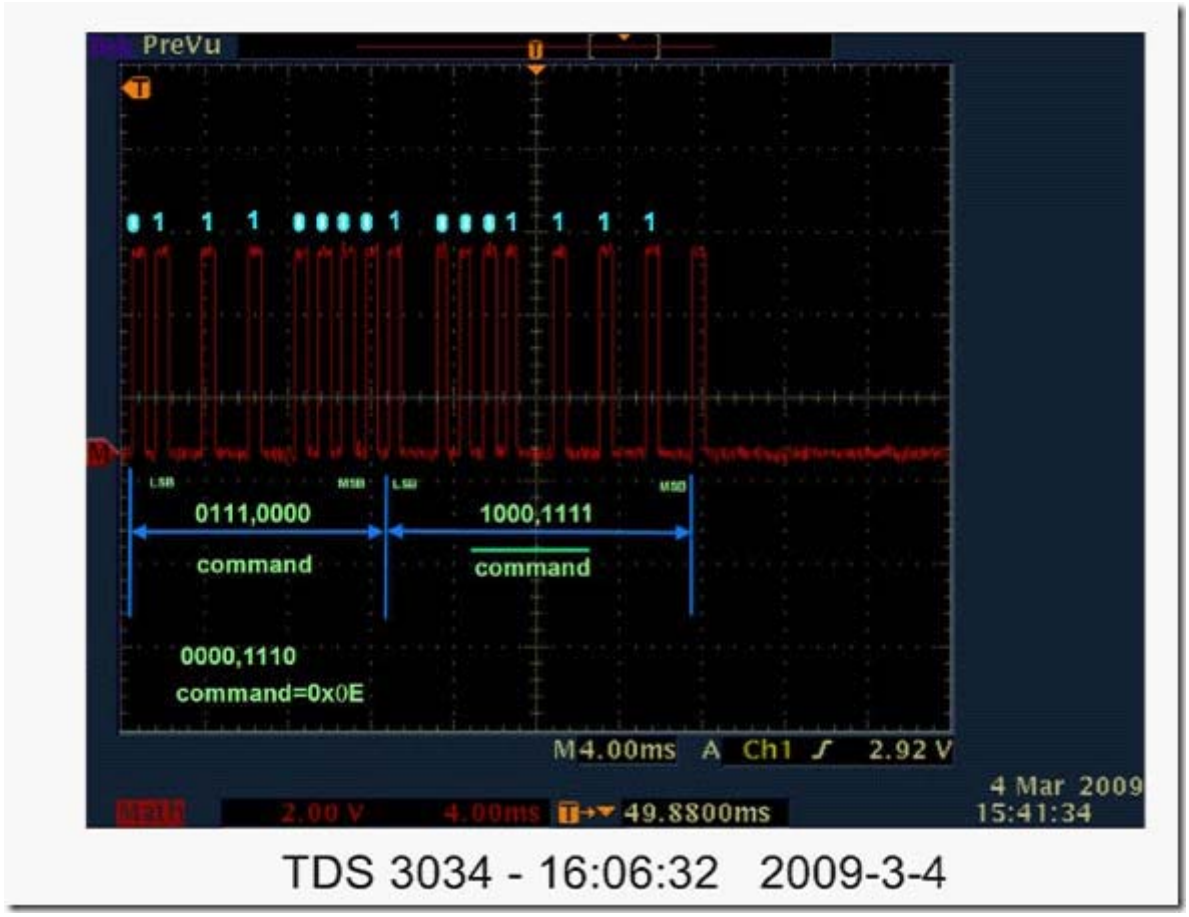
NEC 协议通过脉冲串之间的时间间隔来实现信号的调制（英文简写 PPM）。逻辑“0”是由 0.56ms 的 38KHZ 载波和 0.560ms 的无 载波间隔组成；逻辑“1”是由 0.56ms 的 38KHZ 载波和 1.68ms 的

无载波间隔组成；结束位是 0.56ms 的 38K 载波。



下面实例是已知 NEC 类型遥控器所截获的波形：
遥控器的识别码是 Address=0xDD20;其中一个键值是 Command=0x0E；

注意波形先是发低位地址再发高位地址。所以 0000,0100,1011,1011 反转过来就是 1101,1101,0010,000 十六进制的 DD20；
键值波形如下：

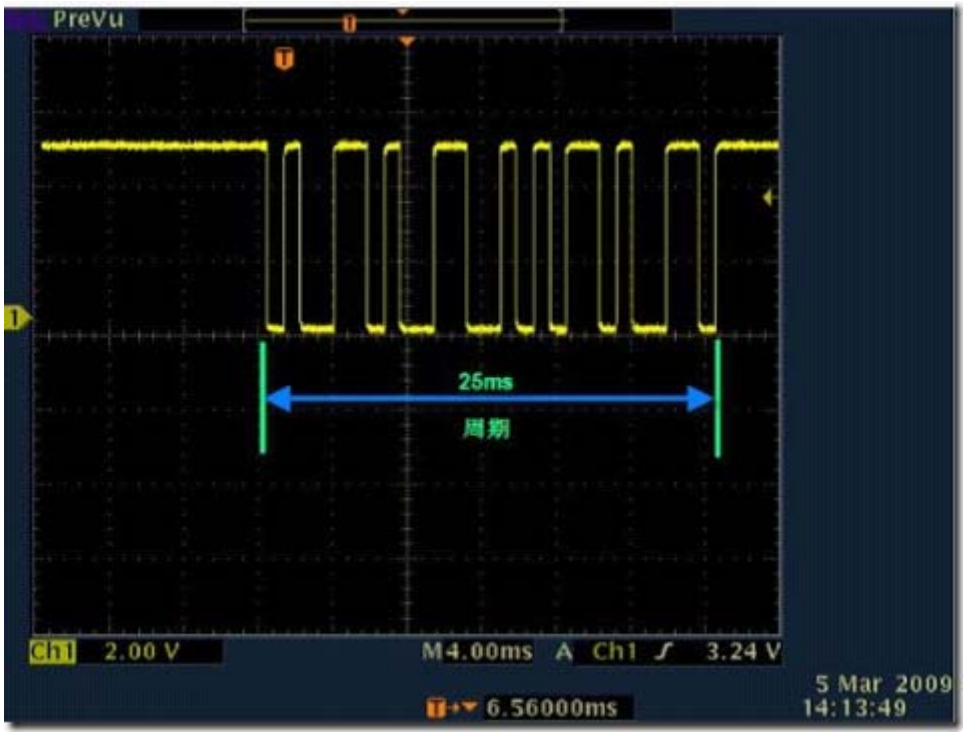


也是要将 0111,0000 反转成 0000,1110 得到十六进制的 0E；另外注意 8 位的键值代码是取反后再发一次的,如图 0111,0000 取反后为 1000,1111。

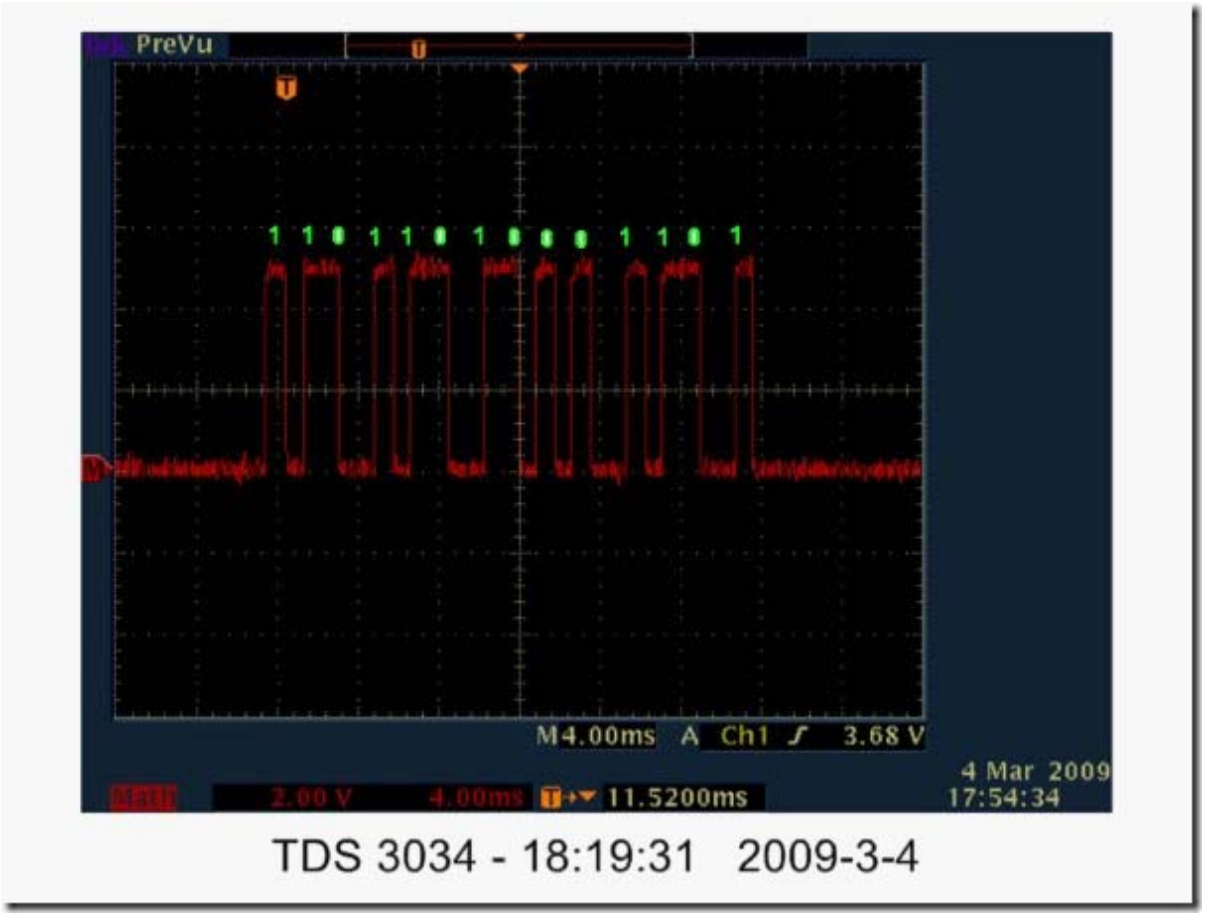
最后一位是一个逻辑“1”。

RC5 编码相对简单一些：

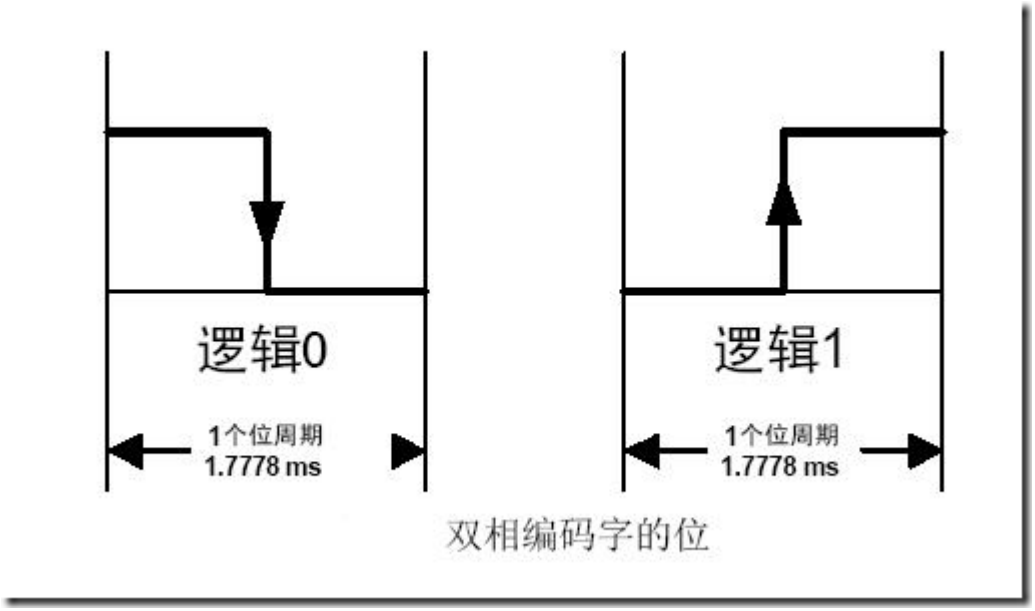
同样由于取自红外接收头的波形需要反相一下波形以便于分析：



反相后的波形：

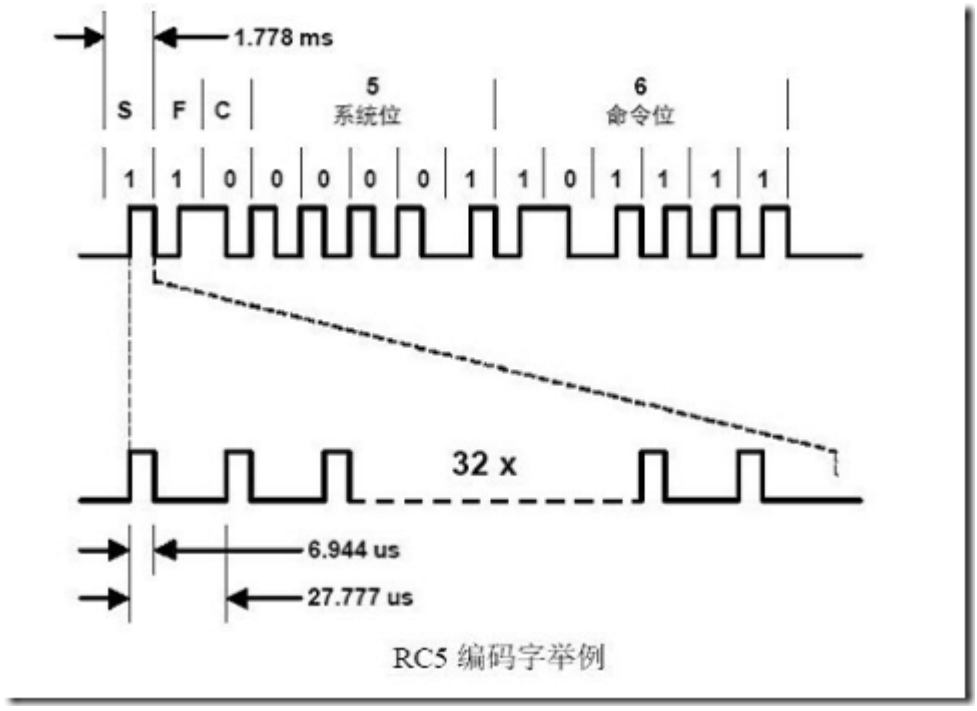


根据编码规则：



得到一组数字：110，11010，001101

根据编码定义：



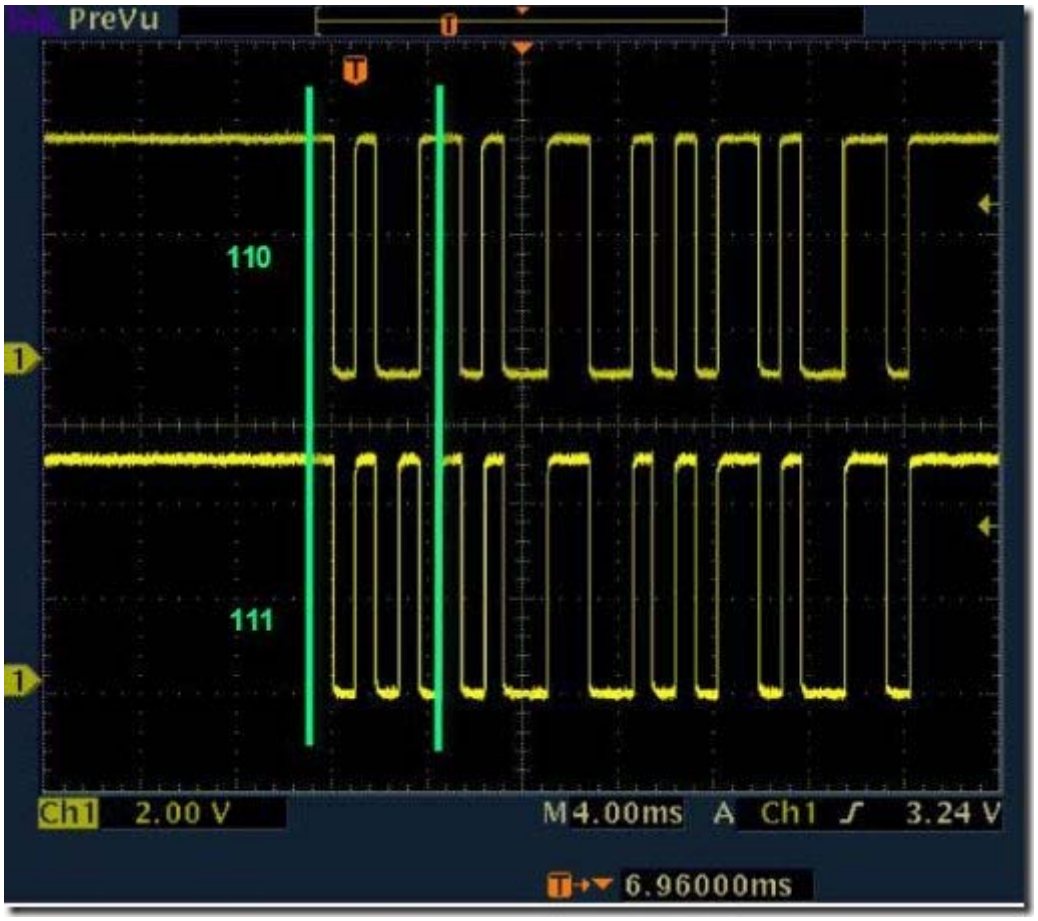
RC5 编码字举例

第一位是起始位 S 通常是逻辑 1

第二位是场位 F 通常为逻辑 1，在 RC5 扩展模式下它将最后 6 位命令代码扩充到 7 位代码（高位 MSB），这样可以从 64 个键值扩充到 128 个键值。

第三位是控制位 C 它在每按下了一个键后翻转,这样就可以区分一个键到底是一直按着没松手还是松手后重复按。

如图所示是同一按键重复按两次所得波形，只有第三位是相反的逻辑，其它的位逻辑都一样。



其后是五个系统地址位:11010=1A

最后是六个命令位:001101=0D
参考：http://www.sbprojects.com/knowledge/ir/nec.htm
红外线遥控器软件解码程序(能解大部分遥控器的编码)
红外线一开始发送一段 13.5ms 的引导码，引导码由 9ms 的高电平和 4.5ms 的低电平组成，跟着引导码是系统码，系统反码，按键码，按键反码，如果按着键不放，则遥控器则发送一段重复码，重复码由 9ms 的高电平，2.25ms 的低电平，跟着是一个短脉冲，本程序是免费给大家，版权所有，不得用于商业目的，如需用到本程序到商业上请与本人联系经本人同意后方可用于商业目的，本程序经过试用，能解大部分遥控器的编码！

```
#include    “at89x52.h”
#define     NULL       0×00//数据无效
#define     RESET      0X01//程序复位
#define     REQUEST    0X02//请求信号
#define     ACK        0×03//应答信号，在接收数据后发送 ACK 信号表示数据接收正确，
也位请求信号的应答信号
#define     NACK       0×04//应答信号，表示接收数据错误
#define     BUSY       0×05//忙信号，表示正在忙
#define     FREE       0×06//空闲信号，表示处于空闲状态
#define     READ_IR    0x0b//读取红外
#define     STORE_IR   0x0c//保存数据
#define     READ_KEY   0x0d//读取键值
#define     RECEIVE    0Xf400//接收缓冲开始地址
#define     SEND       0xfa00//发送缓冲开始地址
#define     IR         0×50//红外接收缓冲开始地址
#define     HEAD       0xaa//数据帧头
#define     TAIL       0×55//数据帧尾
#define     SDA        P1_7
#define     SCL        P1_6
unsigned  char  xdata  *buf1;   //接受数据缓冲
unsigned  int   buf1_length;   //接收到的数据实际长度
unsigned  char  xdata  *buf2;   //发送数据缓冲
unsigned  int   buf2_length;   //要发送的数据实际长度
bit  buf1_flag;    //接收标志，1 表示接受到一个数据帧，0 表示没有接受到数据帧或数据
帧为空
bit  buf2_flag;    //发送标志，1 表示需要发送或没发送完毕，0 表示没有要发送的数据或
发送完毕
unsigned  char  state1,state2;        //用来标志接收字符的状态，state1 用来表示接
收状态，state2 用来表示发送状态
unsigned  char  data  *ir;
union{
    unsigned  char  a[2];
    unsigned  int  b;
    unsigned  char  data  *p1[2];
    unsigned  int  data  *p2[2];
    unsigned  char  xdata  *p3;    //红外缓冲的指针
    unsigned  int  xdata  *p4;
}p;
//union{                    //
  //   unsigned  char  a[2];          //
```

```
  //  unsigned int b;
  //  unsigned char data *p1[2];
//  unsigned int data *p2[2];
  //  unsigned char xdata *p3;
  //  unsigned int xdata *p4;      //地址指针
//}q;                    //
union{
   unsigned char a[2];
   unsigned int b;
}count;
union{
   unsigned char a[2];
   unsigned int b;
}temp;
union{
   unsigned char a[4];
   unsigned int b[2];
   unsigned long c;
}ir_code;
union{
   unsigned char a[4];
   unsigned int b[2];
   unsigned long c;
   unsigned char data *p1[4];
   unsigned int data *p2[4];
   unsigned char xdata *p3[2];
   unsigned int xdata *p4[2];
}I;
unsigned char ir_key;
bit ir_flag;       //红外接收标志，0 为缓冲区空，1 为接收成功，2 为缓冲溢出
void sub(void);
void delay(void);
void ie_0(void);
void tf_0(void);
void ie_1(void);
void tf_1(void);
void tf_2(void);
void read_ir(void);
void ir_jiema(void);
void ir_init(void);
void ir_exit(void);
void store_ir(void);
void read_key(void);
void reset_iic(void);
unsigned char read_byte_ack_iic(void);
unsigned char read_byte_nack_iic(void);
bit write_byte_iic(unsigned char a);
void send_ack_iic(void);
```

```
void  send_nack_iic(void);
bit  receive_ack_iic(void);
void  start_iic(void);
void  stop_iic(void);
void  write_key_data(unsigned  char  a);
unsigned  int  read_key_data(unsigned  char  a);
void  ie0(void)    interrupt  0{ie_0();}
void  tf0(void)    interrupt  1{tf_0();}
void  ie1(void)    interrupt  2{ie_1();}
void  tf1(void)    interrupt  3{tf_1();tf_2();}
void  tf2(void)    interrupt  5{          //采用中断方式跟查询方式相结合的办法解码
   EA="0";                          //禁止中断
   if(TF2){                        //判断是否是溢出还是电平变化产生的中断
       TF2=0;                       //如果是溢出产生的中断则清除溢出位，重新开放中断退出
       EA="1";
       goto  end;
    }
   EXF2=0;                          //清除电平变化产生的中断位
   *ir=RCAP2H;                        //把捕捉的数保存起来
   ir++;
   *ir=RCAP2L;
   *ir++;
   F0=1;
   TR0=1;                           //开启计数器 0
loop:
   TL0=0;  //将计数器 0 重新置为零
   TH0=0;
   while(!EXF2){                     //查询等待 EXF2 变为 1
       if(TF0)goto  exit;             //检查有没超时，如果超时则退出
   };
   EXF2=0;                          //将 EXF2 清零
   if(!TH0)                        //判断是否是长低电平脉冲过来了
   {                               //不是长低电平脉冲而是短低电平
     if(F0)count.b++;                   //短脉冲数加一
     temp.a[0]=RCAP2H;                  //将捕捉数临时存放起来
     temp.a[1]=RCAP2L;
     goto  loop;                    //返回继续查询
   }
   else{                           //是低电平脉冲，则进行处理
       F0=0;
       *ir=temp.a[0];       //把连续的短脉冲总时间记录下来
       ir++;
       *ir=temp.a[1];
       ir++;
       *ir=RCAP2H;          //把长电平脉冲时间记录下来
       ir++;
       *ir=RCAP2L;
       ir++;
```

```
        if(ir>=0xda) {
                goto exit;    //判断是否溢出缓冲，如果溢出则失败退出
        }
        goto loop;         //返回继续查询
        }
exit:
        ir_flag=1;       //置 ir_flag 为 1 表示接收成功
end:
        ;
}
void rs232(void)   interrupt   4{
    static unsigned char sbuf1,sbuf2,rsbuf1,rsbuf2;       //sbuf1,sbuf2 用来接收
发送临时用，rsbuf1,rsbuf2 用来分别用来存放接收发送的半字节
    EA="0";                                 //禁止中断
    if(RI){
      RI="0";                               //清除接收中断标志位
      sbuf1=SBUF;                             //将接收缓冲的字符复制到 sbuf1
      if(sbuf1==HEAD){                              //判断是否帧开头
                state1=10;               //是则把 state 赋值为 10
                buf1=RECEIVE;              //初始化接收地址
      }
      else{
      switch(state1){
      case  10:sbuf2=sbuf1>>4;                  //把高半字节右移到的半字节
            sbuf2=~sbuf2;                   //把低半字节取反
            if((sbuf2&0x0f)!=(sbuf1&0x0f))     //判断接收是否正确
                {                       //接收错误，有可能接收的是数据帧尾，也有可能是接收错误
                 if(sbuf1==TAIL)            //判断是否接收到数据帧尾
                    {                   //是接收到数据帧尾
                      buf1=RECEIVE;      //初始化接收的地址
                      if(*buf1==RESET)    //判断是否为复位命令
                        {
                          ES="0";
                          sbuf2=SP+1;
                          for(p.p1[0]=SP-0×10;p.p1[0]<=sbuf2;p.p1
[0]++)*p.p1[0]=0;
                        }
                      state1=0;         //将接收状态标志置为零，接收下一个数据帧
                      buf1_flag=1;       //置接收标志为 1，表示已经接收到一个数据帧
                      REN="0";           //禁止接收
                    }
                 else
                   {                    //不是接受到数据帧尾，表明接收错误
                     state1=0;          //  将接收状态标志置为零，重新接收
                     buf1=RECEIVE;       //初始化发送的地址
                     *buf1=NACK;         //把 NACK 信号存入接收缓冲里
                     buf1_flag=1;       //置标志位为 1，使主程序能对接收错误进行处理
                     REN="0";           //禁止接收
```

```
                }
              }
            else
            {                          //接收正确
               rsbuf1=~sbuf1;             //按位取反，使高半字节变原码
               rsbuf1&=0xf0;              //仅保留高半字节，低半字节去掉
               state1=20;                 //将状态标志置为 20，准备接收低半字节
            }
            break;
      case  20:sbuf2=sbuf1>>4;                //把高半字节右移到的半字节
            sbuf2=~sbuf2;                 //将低半字节取反
            if((sbuf2&0x0f)!=(sbuf1&0x0f))    //判断接收是否正确
              {                        //接受错误
                 state1=0;               // 将接收状态标志置为零，重新接收
                 buf1=RECEIVE;             //初始化接收的地址
                 *buf1=NACK;               //把 NACK 信号存入发送缓冲里
                 buf1_flag=1;              //置标志位为 1，使主程序能对接收错误进行处理
                 REN="0";                  //禁止接收
              }
            else
              {
              sbuf1&=0x0f;                //仅保留低半字节，去掉高半字节
              rsbuf1|=sbuf1;              //高低半字节合并
              *buf1++=rsbuf1;             //将接收的数据保存至接收缓冲里,并且数据指针加一
              buf1_length++;              //接收数据长度加一
              state1=10;                  //将 state1 置为 10，准备接收下个字节的高半字节
              }
            break;
      }
      }
  }
else{
    TI="0";                           //清除发送中断标志
    if(buf2_length)                     //判断发送长度是否为零
          {                           //发送长度不为零
          if(state2==0)                 //判断是否发送高半字节
            {                         //发送高半字节
               sbuf2=*buf2;             //将要发送的字节送到 sbuf2
               rsbuf2=~sbuf2;           //取反，使高半字节变为反码
               sbuf2>>=4;               //将高半字节右移到低半字节
               rsbuf2&=0xf0;            //保留高半字节，去掉低半字节
               sbuf2&=0x0f;             //保留低半字节，去掉高半字节
               rsbuf2|=sbuf2;           //合并高低半字节
               SBUF="rsbuf2";             //发送出去
               state2=10;               //将 state2 置为 10 准备发送下半字节
            }
          else
            {                         //发送低半字节
```

```
                sbuf2=*buf2;              //将要发送的字节送到 sbuf2
                buf2++;                   //指针加一
                buf2_length–;             //发送数据长度减一
                rsbuf2=~sbuf2;             //取反，使低半字节变为反码
                rsbuf2<<=4;               //将低半字节反码左移到高半字节
                rsbuf2&=0xf0;              //保留高半字节，去掉低半字节
                sbuf2&=0x0f;               //保留低半字节，去掉高半字节
                rsbuf2|=sbuf2;             //合并高低半字节
                SBUF="rsbuf2";               //发送出
                state2=0;
              }
          }
      else
          {                               //如果发送数据长度为零则发送数据帧尾
            if(buf2_flag){                //判断是否发过数据帧尾
            SBUF="TAIL";                    //将数据帧尾发送出去
            while(TI==0);
            TI="0";
            buf2_flag=0;                  //置发送标志为零，表示发送完毕
            }
          }
  }
  EA="1";                                 //开放中断
}
```