TEMIC 系列射频卡开发指南

一. 开发设计简介：

TEMIC 系列射频卡产品包括 **E5550**、**E5560** 卡和 **U2270B** 基站芯片。`在开发过程中由开发商自行设计基站发射、接收电路。由于 **U2270B** 基站芯片只需少量的驱动电路，并且具有多种供电模式。这给用户以极大的简便性和灵活性。用户可以根据不同的应用要求快速、简便的设计出不同特点的基站电路。用户仍然需要绕制基站天线，一般使用铜制漆包线绕制直径 3CM、100 圈的线圈即可。这些特点要求开发射频卡应用软件的同时还要设计基站发射电路。

二. 设计目标：

1. 硬件设计：

根据应用环境（供电条件、功耗要求）的需要设计发射基站电路。绕制合适的天线线圈，要求达到要求的频率特性。设计单片机控制接口电路对基站电路进行控制。

2. 软件设计：

根据设计电路的约定设计射频卡读写程序，要求能够对射频卡进行完备操作（读数据、写数据、加密控制等），并能够提供简便易用的编程接口（函数封装）。设计单片机与 PC 机的通讯程序，设计通讯和命令格式约定等协议。在 PC 端还要设计应用程序接口（动态连接库 .DLL）供上层应用程序调用。

三. 实现：

1. 硬件实现：

E2270B 支持两种供电方式。一种为+5V 直流电源供电，另一种为汽车用+12V 电池供电，并且 E2270B 还具有电压输出功能可以给微处理器或其他外围电路供电。此外，对 E2270B 还有省电模式和 STANDBY 控制可选，所以设计基站电路时应中和以上功能的不同要求，设计基站的外围电路。这里只对几个常用控制功能的实现加以说明。

A．省电模式：

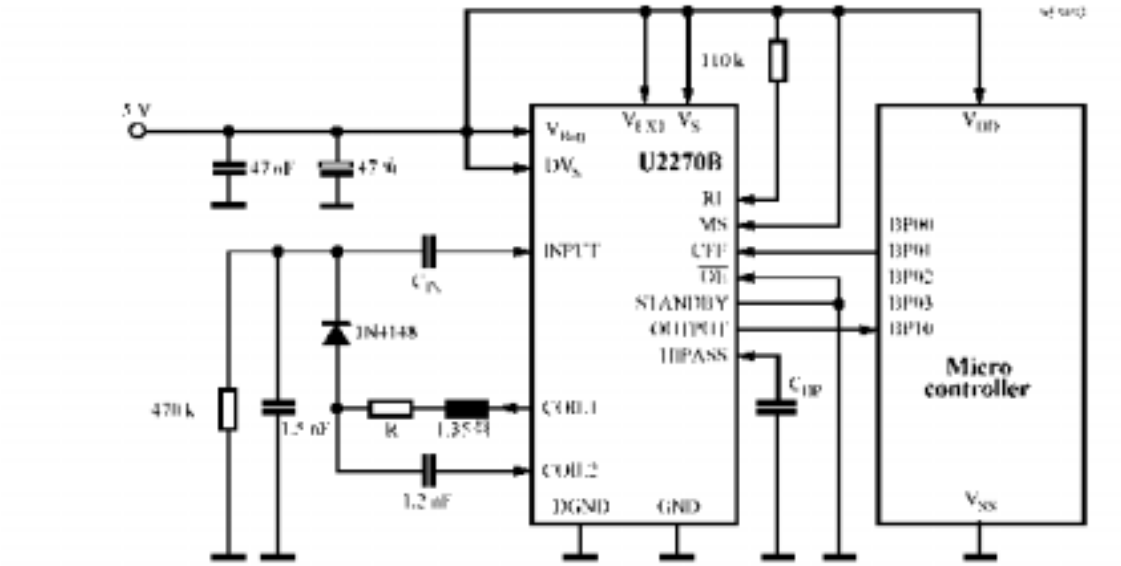
当射频卡系统应用于汽车、门禁等领域如果对电路的耗电量加以控制（射频信号发射的功耗是远远大于接触式 IC 卡的），可以有效的节省能耗延长内置电池的寿命。在需要降低功耗的领域可以在无卡检测的过程中间断的打开和关闭射频输出（E2270B 的 CFE 脚）。也可以使卡处于 STANDBY 模式（控制 U2270B 的 STANDBY 脚）这样可以极大的降低基站的耗电量。此外用户也可以使用外加检测电路的方法如：光电管、红外检测等低耗电手段来控制基站的工作这样可以得到很好的省电要求。

B．串行通讯：

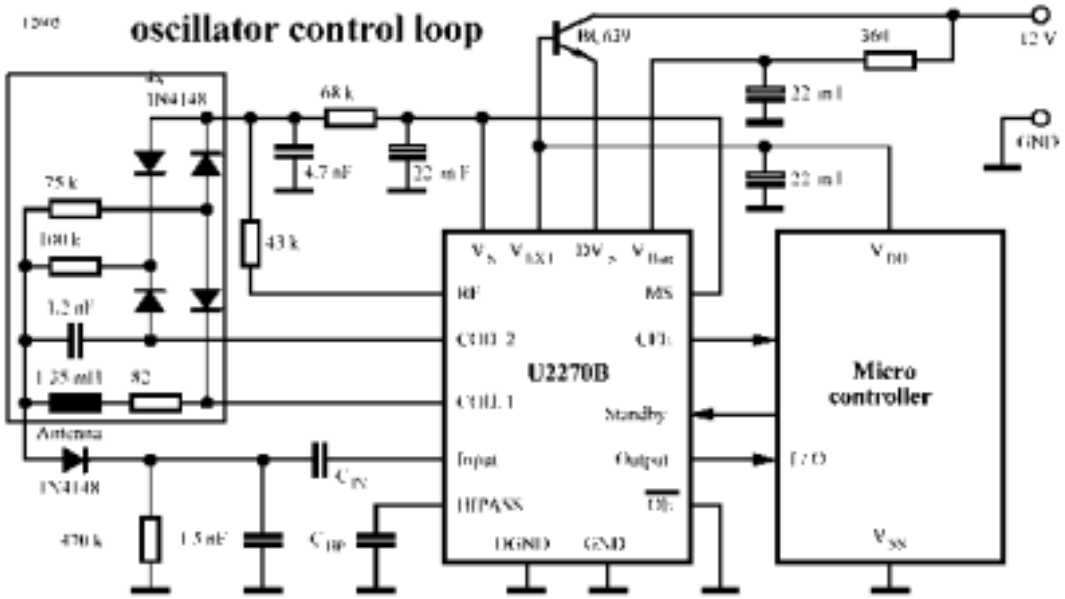
假设射频卡读写器为 PC 机的外围设备，读写器与微机的通讯需要通过串行口进行。根据需要用户可以选择使用各种串行接口电路。这里假设使用 RS232 串行接口电路。

C．电路原理图：

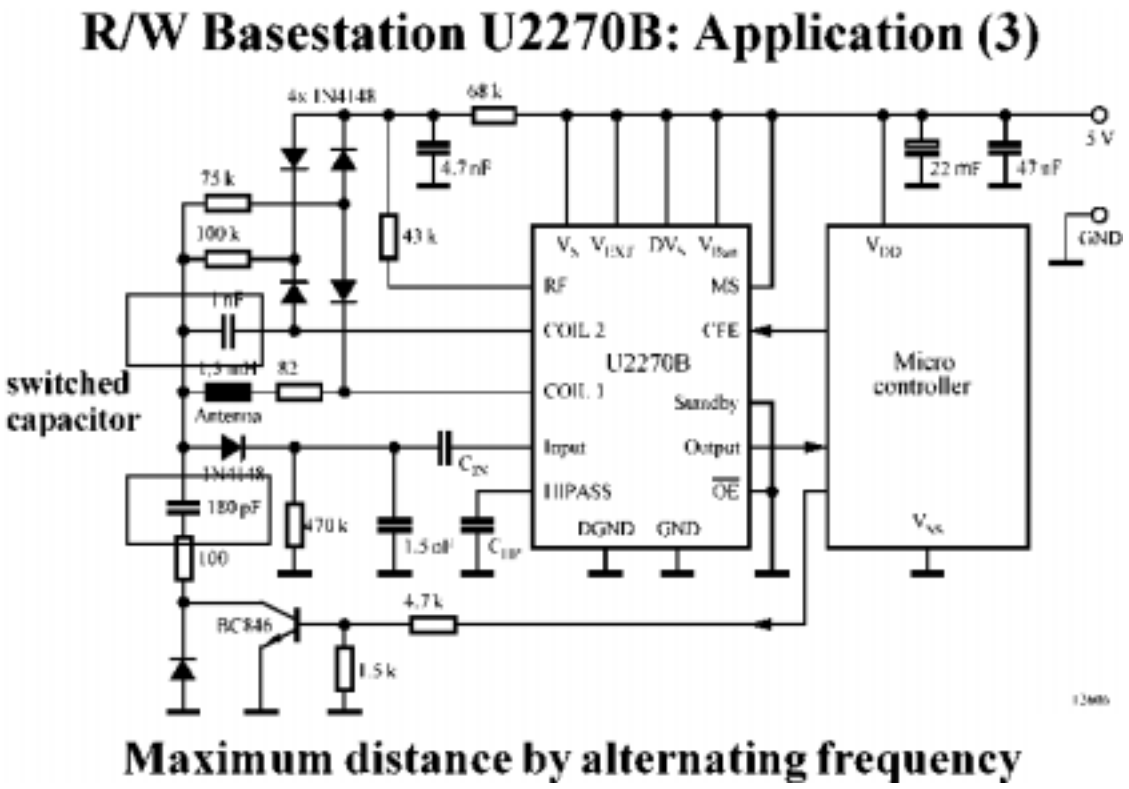
R/W Basestation U2270B: Application (1)



R/W Basestation U2270B: Application (2)



Application for extended R/W distance



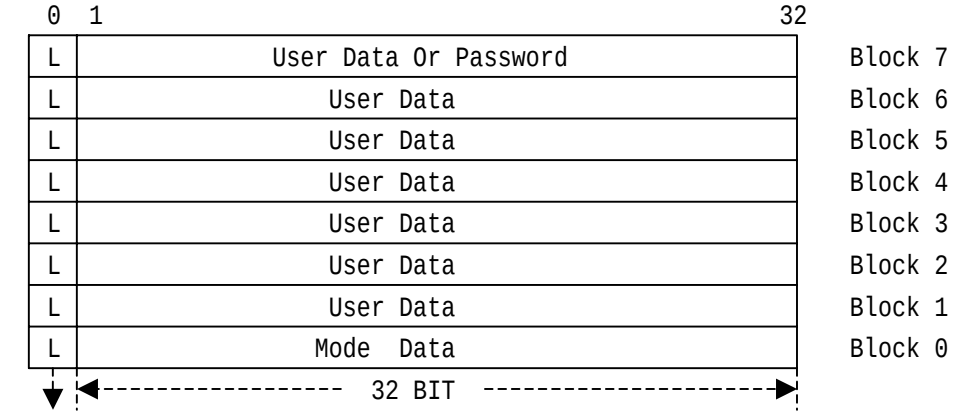
2. 软件实现：

软件设计要求设计汇编程序完成对射频卡的完备操作（包括读操作、写操作以及命令发送等）。读卡程序要求**用软件模拟信号时序**，自动检测同步信号同步后要根据选择的编码方式进行软件解码，最后将解码得到的数据流按合理顺序存人指定存储区。写卡程序使用开关天线负载的方法对数据进行编码，要求能向 IC 卡发射各种组合的数据流，完成对 IC 卡的各种控制功能（注：写程序只完成向 IC 卡发送数据的功能，对命令或写操作的执行情况不做检测，而是由读卡程序获取 IC 卡的反馈信息后再进行比较判断）。由于 E5550 卡与 E5560 卡的特性区别所以对两种卡的读写程序不尽相同，但两种卡读写程序的设计思想是基本相同的。

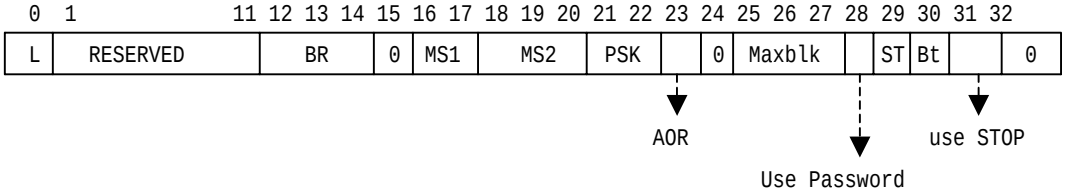
A. E5550 卡读写程序：

在介绍 E5550 卡的读写卡程序之前，先要了解该类型卡的读写特性。这些特性包括：EEPROM 的存储分配、卡的同步信号、发射频率、BITRATE、卡控制块的结构、写卡以及其他卡操作的命令格式等。

E5550 卡内置 264 位的 EEPROM 这些存储区分为 8 个 BLOCK 每块 33 位，其中第 **0** 位为块锁定位，**一旦该位被置‘1’则该块儿数据将不能再做任何修改，而且 LOCK 位是无法恢复的**。LOCK 位不随其他位一起发射到基站（即：LOCK 位是不可读的用户实际使用的数据区位每块的后 32 位共 256 位）。8 个 BLOCK 中的 BLOCK 0 是作为卡片的控制块存储卡的控制信息，BLOCK 7 是 PASSWORD 区在口令加密功能启动时这里存放卡的读写控制密码，当加密功能没有使用时该区也可以存放用户数据。其他六个存储块用户可以用来存放各种数据。EEPROM 结构如图：



E5550 卡控制块用于控制卡的各种操作的特性，如：同步信号、数据流格式、数据流长度、加密、口令唤醒和停止发射等功能的启用关闭等。控制块位于 EEPROM 的第 0 块数据区可由用户进行编程控制（用户向卡发送写命令给该区写入一定格式的数据即可）。一般一个应用系统的卡的模式块的值是统一的，在发卡时建议写入数据后将该块的 LOCK 位置‘1’这样可以仿制对控制块的误修改引起卡的操作不正常。E5550 卡的控制块的结构和功能说明如图：（**模式设置将影响读写程序的设计**）



下面结合对控制块的说明，简单介绍 **E5550** 卡的各种工作模式和操作特性：

在 **E5550** 卡中控制块的第 1 位至第 11 位之间的 11 位和第 32 位为保留位，现在没有使用，用户可以写入任何值，建议写入‘0’用来和其他功能位区别。控制块中的第 15 位和第 24 位必须写入‘0’否则卡将不能正常工作。从第 12 位至第 14 位为 Bitrate 设置位。用户通过设置这三位的值可以决定卡发射数据时的 Bitrate。用户可按下表中的值进行设置。（**一般使用 RF/32 的 BITRATE**）

第 12 位	第 13 位	第 14 位	Bit Rate
0	0	0	RF/8
0	0	1	RF/16
0	1	0	RF/32
0	1	1	RF/40
1	0	0	RF/50
1	0	1	RF/64
1	1	0	RF/100
1	1	1	RF/128

第 16~17 位、18~20 位以及 21~22 位结合在一起设定卡发射数据的调制方法，具体配合方式如下表所示。用户设置 16、17 位为‘00’时 18~20 位的设置有效，如果 18~19 位设置为‘001’、‘010’、‘011’时可继续使用第 21~22 位设置在 PSK 调制方法下的频率变化。第 23 位用来控制是否启动 AOR（Answer-On-Request）功能。该位设置为‘1’时启动 AOR 功能，这时 IC 卡进入射频区域后不主动发射数据，而要由基站给 IC 卡发射唤醒命令后再发射数据。该功能要求首先启动口令加密功能，也就是说基站要

唤醒一个 IC 卡时必须在唤醒命令序列中向 IC 卡发射口令密码，IC 卡检测到包含合法口令的唤醒命令时才恢复发送数据。要启动口令加密功能就要求将控制块的第 28 位

16	17	Modu Mode
0	0	direct
0	1	Mancheester
1	0	Biphase
1	1	Reserved

21	22	PSKCF
0	0	RF/2
0	1	RF/4
1	0	RF/8
1	1	reserved

18	19	20	Modu Mode	More	Data1	Data0
0	0	0	direct			
0	0	1	PSK1			
0	1	0	PSK2			
0	1	1	PSK3			
1	0	0	FSK1		RF/8	RF/5
1	0	1	FSK2		RF/8	RF/10
1	1	0	FSK1a		RF/5	RF/8
1	1	1	FSK2a		RF/10	RF/8

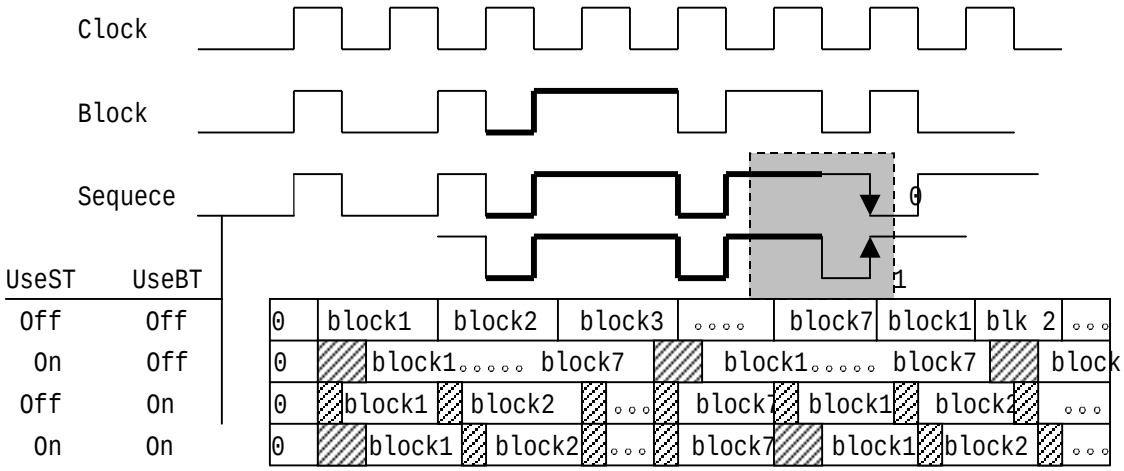
设置为‘1’。启动口令加密功能后第 7 块数据区将保存 IC 卡的口令密码，所以**启动加密功能之前应该事先写入密码**。如果允许修改密码则不用锁定 BLOCK 7 如果密码永久有效则要在写入密码的同时锁定 BLOCK 7 这样用户将不能修改密码。在加密模式下用户对卡中数据进行任何修改均要求提供密码验证。密码正确时修改操作有效，密码不正确则修改无效。后面将讲到加密模式和非加密模式下的写命令格式是不同的。

为了保护密码不被未知用户截获，用户在启动加密功能后还应该对控制块的第 25~27 位进行设置。这三位设置的为 IC 卡发射数据时发射的最大数据块数（Max Block）这三位的设置和发射数据流的关系如下表：

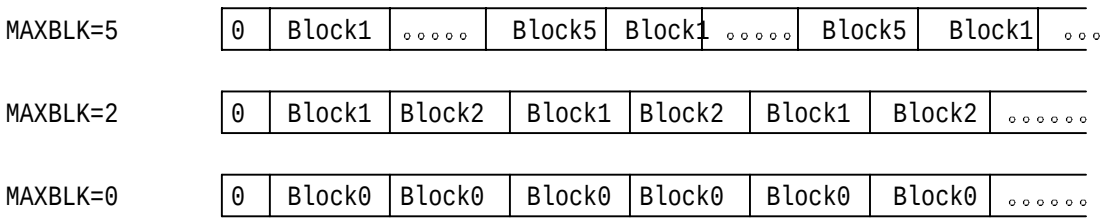
第 25 位	第 26 位	第 27 位	Send Blocks
0	0	0	Only block 0
0	0	1	block 1
0	1	0	block 1~2
0	1	1	block 1~3
1	0	0	block 1~4
1	0	1	block 1~5
1	1	0	block 1~6
1	1	1	block 1~7

当 MAXBLK 设置为‘0’时 IC 卡只发射 BLOCK 0 的数据给基站；当设置为‘1’时 IC 卡只发射 BLOCK 1 的数据给基站；当设置为‘2’时 IC 卡发射 BLOCK 1 和 BLOCK 2 的数据给基站；设置为‘3’时 IC 卡发射 BLOCK1 至 BLOCK 3 的数据该基站其他的依次类推当设置为‘7’时 IC 卡发射 BLOCK1 至 BLOCK 7 的数据给基站。在启动口令模式后 MAXBLK 的值应小于‘7’这样 IC 卡将不发射存放在第 7 块中的数据。

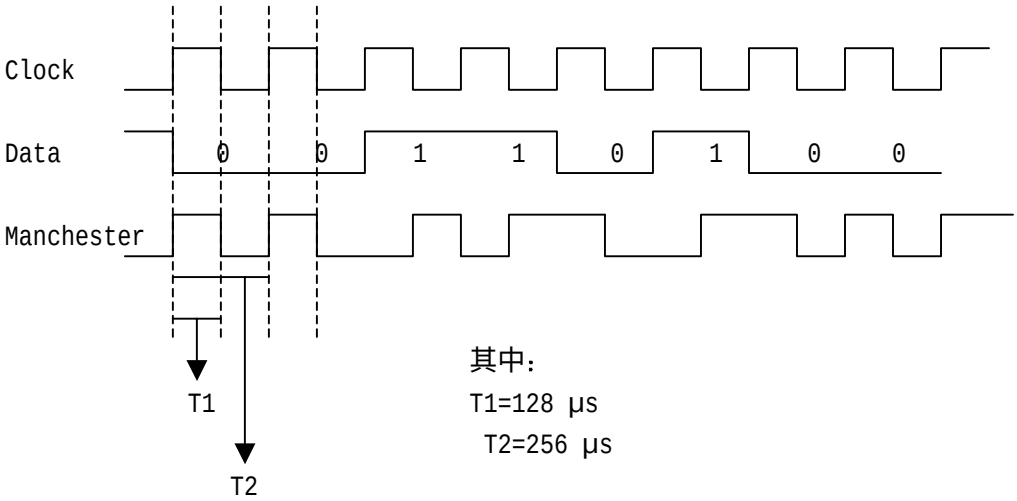
用户除了设置以上各项设置项以外，还可以设置 IC 卡发射数据时的同步信号类型。IC 卡可以使用两种不同的同步信号，它们是 Sequence Terminator 和 Block Terminator。Sequence Terminator 在每个数据循环开始时出现。Block Terminator 在每个 BLOCK 的数据的开始时出现。两种同步信号可以独立使用也可以结合使用。同步信号的波形和其与数据流的结合情况如下图所示：（假设 MAXBLK=7，使用 Manchester 编码。）



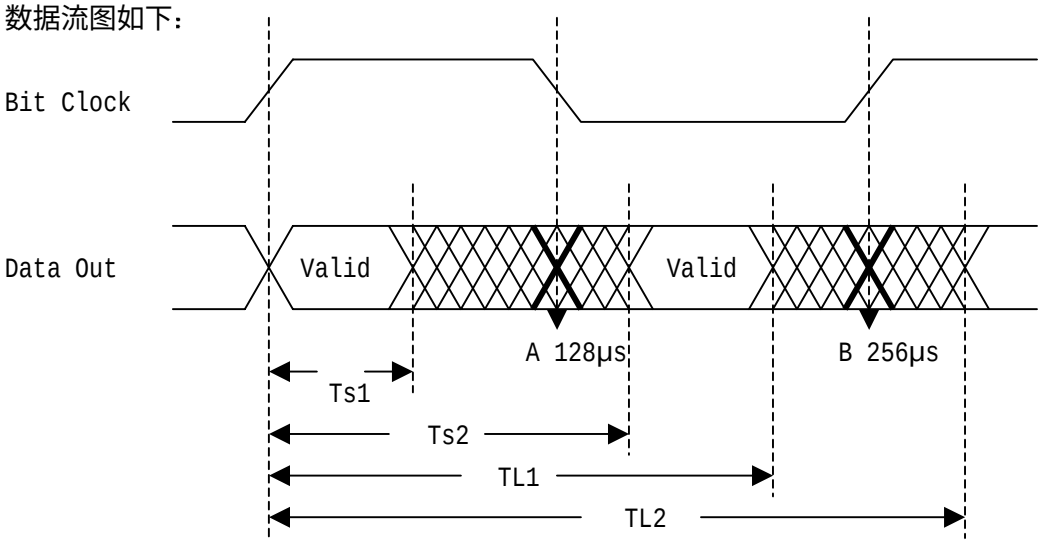
MAXBLK 值与数据流的关系图如下：（假设 UseBT=off AND UseST=off）



IC 卡发射数据由基站天线接收后，由基站处理后经基站的 Output 脚把得到的数据流发给微处理器的输入口。这里基站只完成信号的接收和整流的工作，而信号的解调解码的工作要由微处理器来完成。微处理器要根据输入信号在高电平、低电平的持续时间来模拟时序进行解码操作。下面以 Manchester 编码、125 kHz 频和 RF/32 的 Bitrate 的条件下时序为例给出基站读取数据流的时序图。



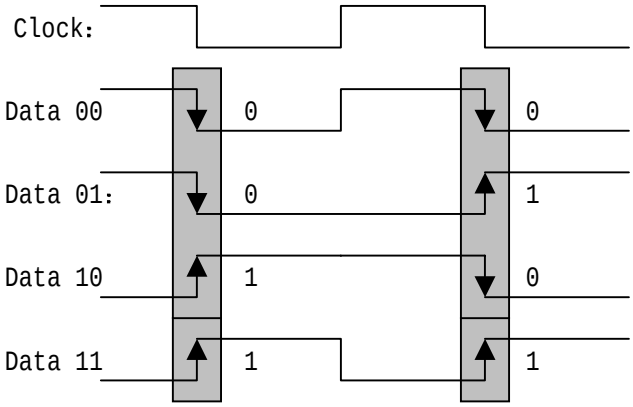
数据流图如下：



上图所示的是程序检测跳变的时间基准。图中阴影部分为跳变的不稳定区间 ，Valid 区域是稳定区。程序检测电平跳变是在一个时间区间以内，如：半个周期的跳变理想状态应为 128μs 如图中 A 点所示但实际检测区域为 Ts1~Ts2（即：凡是时间在 Ts1 和 Ts2 之间的跳变信号均视为半个周期的跳变信号）。同样，在 TL1~TL2 之间的跳变都可以视为一个周期的跳变。E5550 在上图假设条件下时这四个时间检测标准点的值为：

TS1=70μs ， TS2=190μs ， TL1=210μs ， TL2=300μs

现在介绍 E5550 卡在使用 Manchester 编码时的解码方法。下图表示 Manchester 编码的电平状态变化情况。



由上图所示可以看出：当数据位为‘1’时跳变总是由低向高，而数据位为‘0’时跳变总是由高向低。结合 Manchester 编码的特点我们可以这样进行解码：在位时钟周期的半周期处检测电平的变化情况，如果检测到电平变化发生则继续判断变化后的电平情况，是高电平则该位解码为‘1’，低电平则解码为‘0’，没有跳变发生则可视为信号异常进行出错处理。

IC 卡与基站的数据交换是双向的，基站要向 IC 卡发送命令和数据，完成对 IC 卡各种控制操作。对 E5550 卡而言，基站可以向卡片发送的命令有四种格式分别完成四种控制功能。

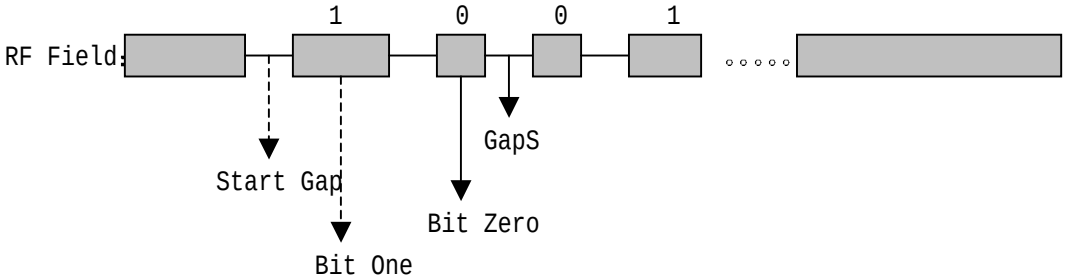
OP

10	L	1	Data Bits	32	2	ADR	0		Standard Write
10	1	Password	32	L	1	Data Bits	32	2 ADR 0	Password Mode
10	1	Password	32						AOR Wake Up
11	Stop Modulation								

四种命令分别完成以下功能：

- Standard Write：对卡数据的普通读写，其中‘10’为操作码，‘L’位为指定数据块的锁定位，紧接着‘L’位是 32 位数据。数据后面是命令要写入的数据块的块地址，这里块地址用 3 位二进制码表示。
- Password Mode：该操作和 Standard Write 操作完成类似功能，只是在 Password Mode 启动后对卡中数据的修改就要求提供口令。使用该命令就是要完成 Password Mode 下卡中数据的修改。命令数据流中其他部分和 Standard Write 的含义一样，只是在操作码和‘L’位之间加入了长度为 32 位的口令数据。卡接收到命令后在对数据区进行修改之前要检验命令提供的口令与卡中密码区保存的数据是否一致，只有两者一致时 IC 卡才真正的修改数据区的数据。这样可以防止不知道密码的非法用户对卡中数据的修改。
- AOR Wake Up：该命令是卡的 AOR 功能启动后，基站发给卡片的唤醒命令。命令由操作字‘10’和 32 位的口令字组成。使用该命令可以唤醒密码和命令字中提供的密码一致的卡片。卡片唤醒后即可向基站发送数据。
- Stop Modulation：该命令用来关闭 IC 卡使接收到命令的 IC 卡进入睡眠状态。进入睡眠状态的 IC 卡不再向外发送数据，而在接收到 AOR 命令后再开始发射数据。利用这种机制可以完成一定的防冲撞功能（通常情况下当多个卡片同时进入射频区域时，基站是无法读取数据的。这时可以由基站发射 Stop 命令，使所有卡片进入睡眠状态，然后再由基站使用不同的密码发射唤醒命令来唤醒密码相同的卡片。读写操作完成后再关闭该卡片，依次可以处理各个卡片。

基站给卡片发送数据时也要对数据进行编码，使数据信号加载到天线的发射信号中。TEMIC 公司的系列产品使用一种改变发射天线负载的方式对信号进行编码。这种方法使用短暂的 RF 信号间隔（GAP）来把 RF 信号分割成不同长短的区间的方法对数据进行编码。起始 GAP 一般比其他 GAPS 略长，用来与卡片同步。在发送数据时一个长度为 16~32 field clocks 时间长度的 RF 区间表示数据为‘0’，一个长度为 48~64 field clocks 时间长度的 RF 区间表示数据为‘1’。在编制程序时可以使用延时中断 RF 区域的方法进行发送数据。发送数据时的 RF 区域状态如图：



对 E5550 卡各段区间的时间长度为：

Tgaps=300μs ， T1=350μs ， T0=100μs

最后介绍以下在编制程序时应该注意的一些细节问题：

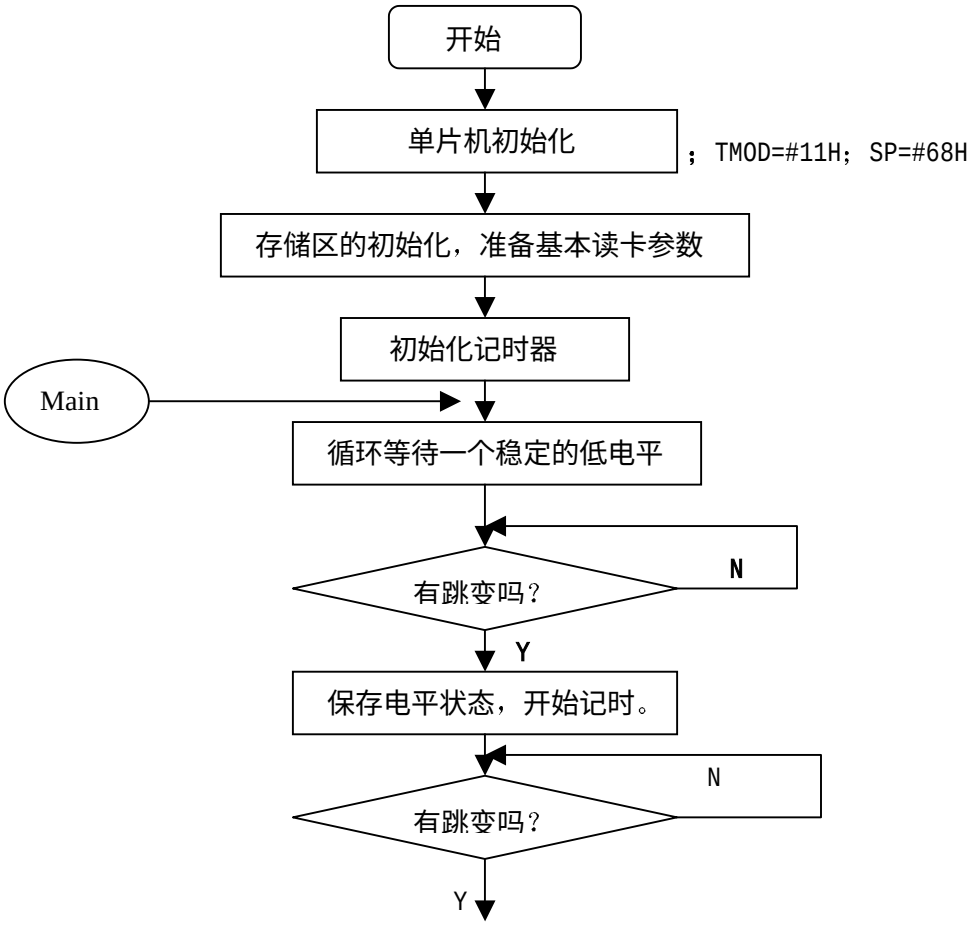
- IC 卡发射数据高低位顺序：IC 卡向基站发射数据时是根据 BLOCK 1 的设置从第一区到第 MAXBLK 区循环发射的。数据以选择的同步信号开始按照块的顺序发送的。每块数据的发送是低位在前，高位在后，即先发送第 1 位数据然后发送第 2 位依次类推到第 32 位（第 0 位是数据块的锁定位是不随数据一起发送的）。
- 对数据存储时应注意字节地址的选择：由上面的介绍我们可以知道，E5550 卡读写的单位为 32 bit ，所以要用 4 个字节的空间存储一个数据区的数据。程序中使用移位的方法取输入口检测到数据位。这就要求字节内移位方向和字节间的地址变化有机的结合在一起，否则将出现读数据高低位或字节间顺序与实际顺序不一致的情况。
- 写数据时同样要注意发送数据的高低位顺序，特别是发送数据区地址信息时。如果不注意发送数据的顺序则极有可能错误的将数据写入其他的数据区中。如：向第 1 数据区写数据时，地址应为‘001’使用移位操作时应使用循环左移的方法依次发送地址信息，如果这里错误的使用了循环右移的方法则卡获得的实际地址为‘100’写入的为第 4 区的数据。

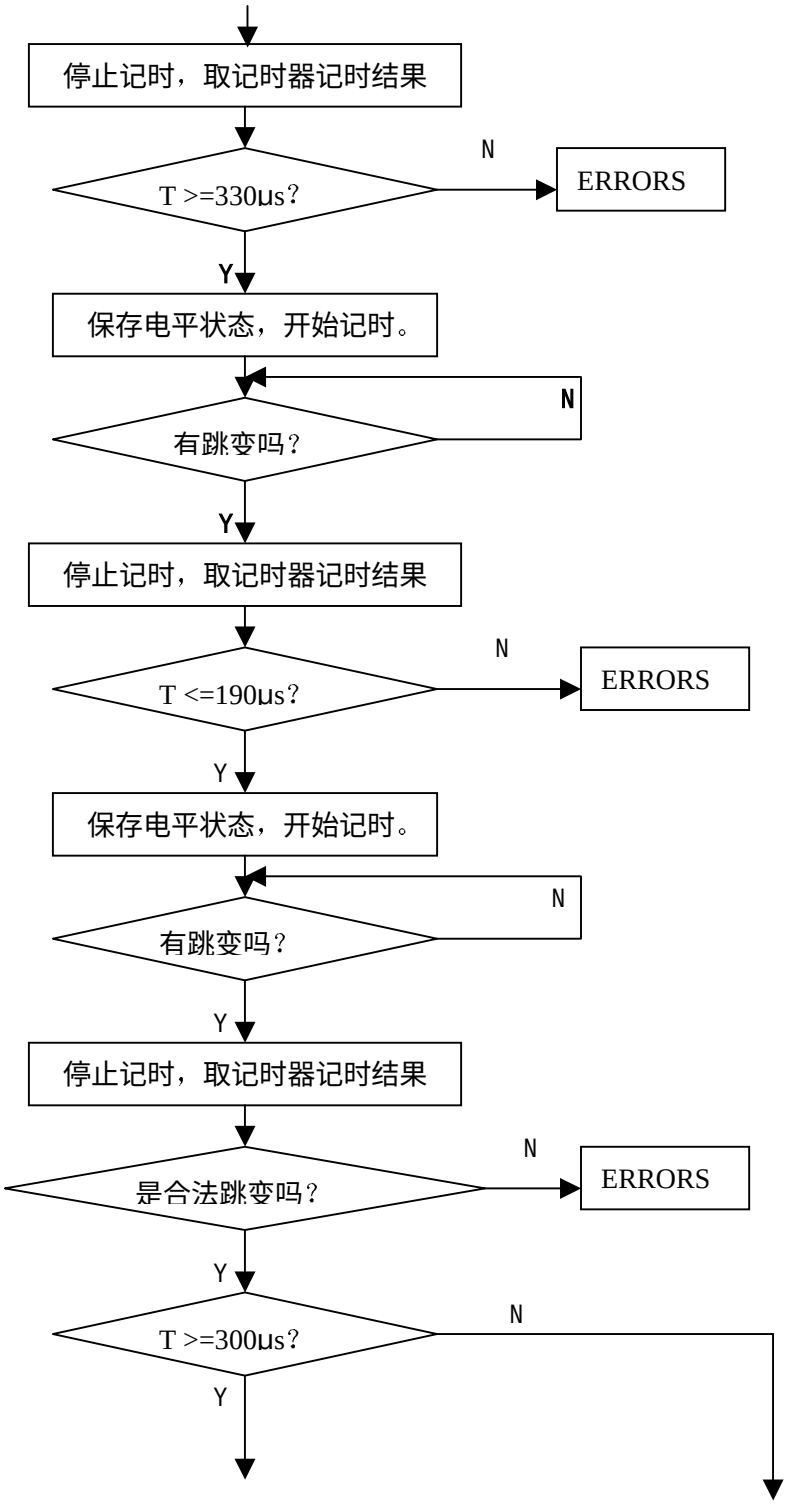
最后给出一个 E5550 卡的读写程序以供参考：

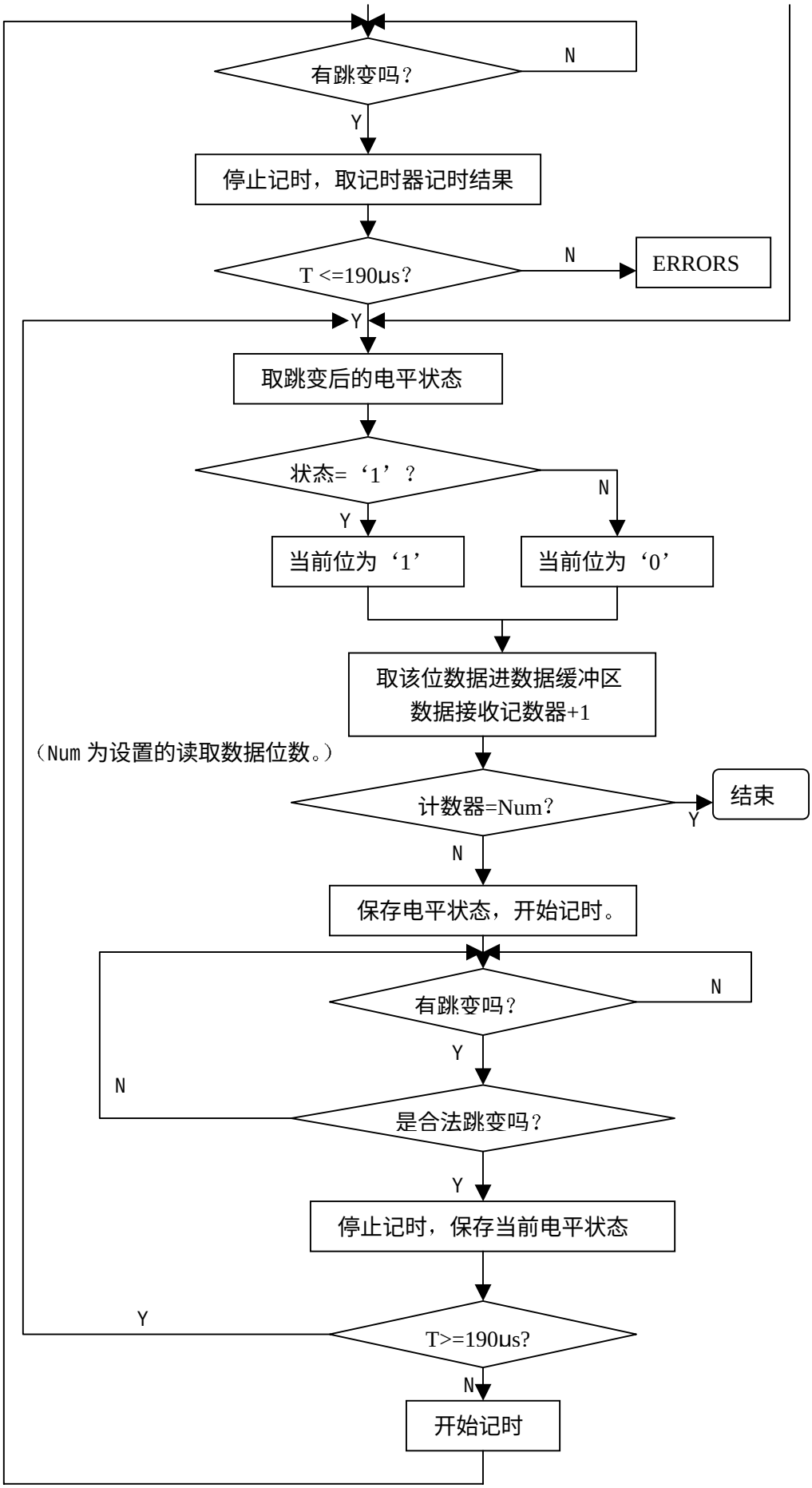
- **设计的环境条件**：

  假设工作在 125 kHz 的射频频率下，采用 RF/32 的 Bit Rate，Manchester 编码，使用 Sequences Terminator 同步信号，MAXBLK=7，使用 ATMEL 89C51 单片机。

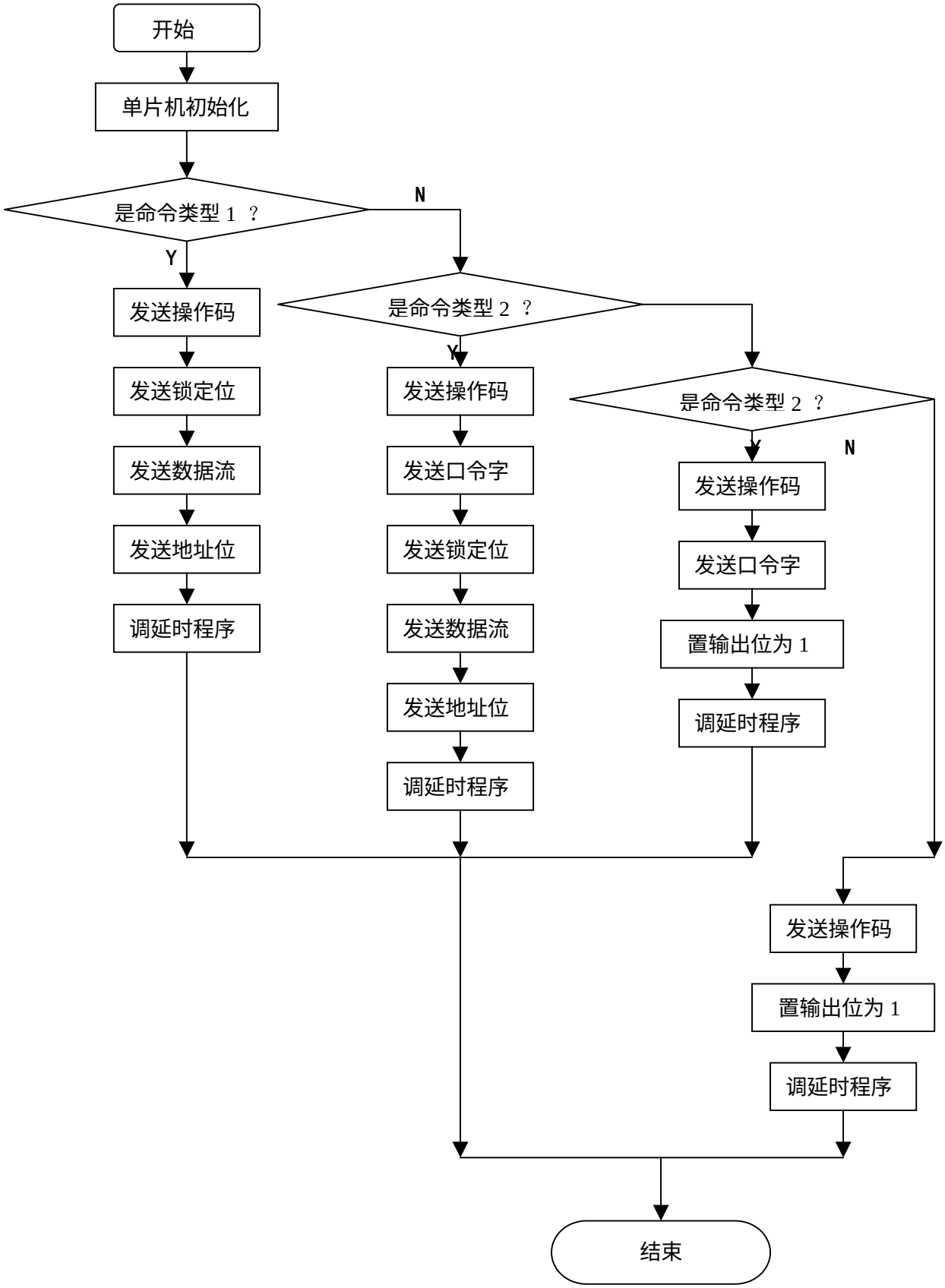
- **读卡程序框图：**




停止记时，取记时器记时结果
T >=330μs?
N
ERRORS
Y
保存电平状态，开始记时。
有跳变吗？
N
Y
停止记时，取记时器记时结果
T <=190μs?
N
ERRORS
Y
保存电平状态，开始记时。
有跳变吗？
N
Y
停止记时，取记时器记时结果
是合法跳变吗？
N
ERRORS
Y
T >=300μs?
N
Y

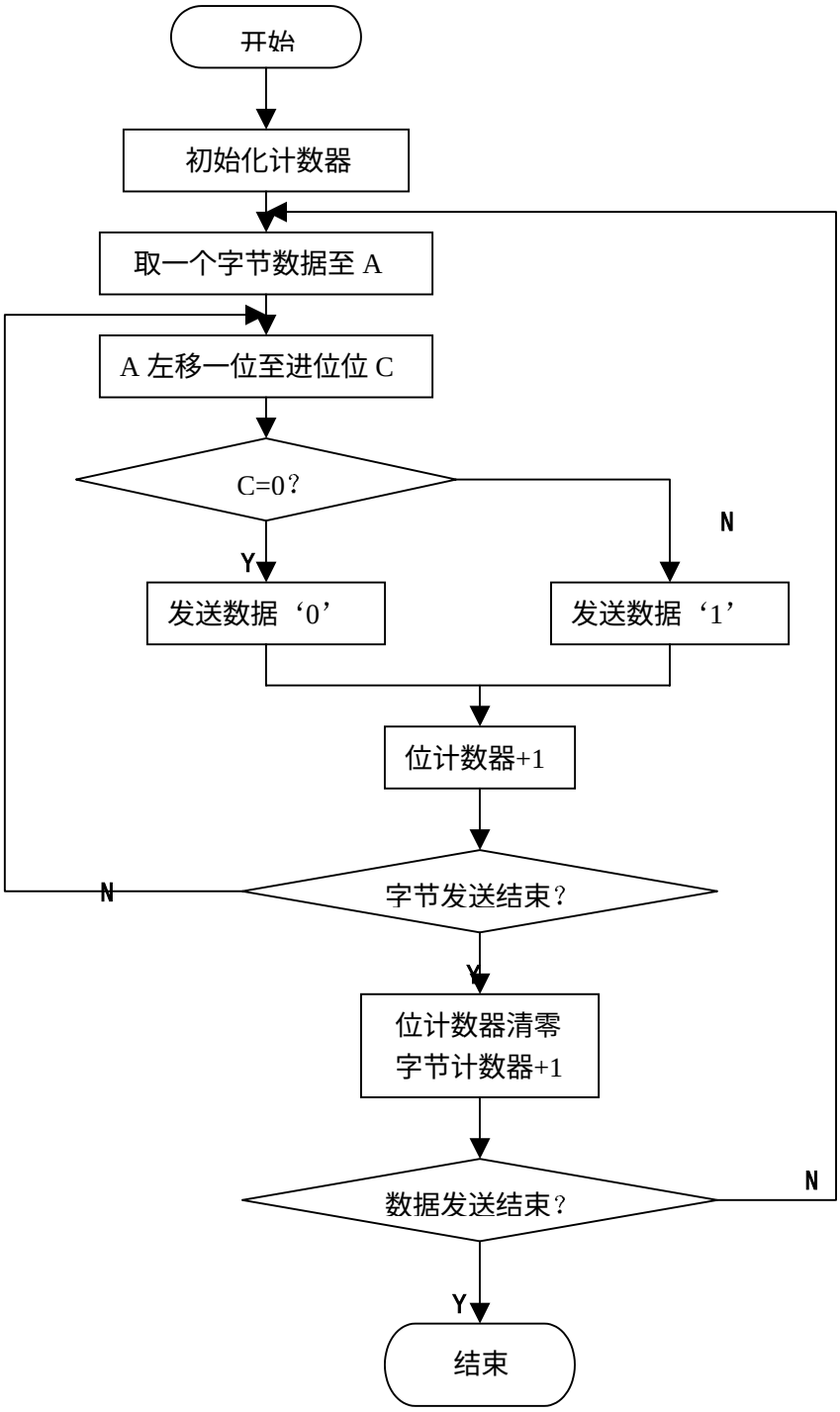

有跳变吗？
N
Y
停止记时，取记时器记时结果
T <=190μs?
N
ERRORS
Y
取跳变后的电平状态
状态=‘1’？
N
Y
当前位为‘1’
当前位为‘0’
取该位数据进数据缓冲区
数据接收记数器+1
（Num 为设置的读取数据位数。）
计数器=Num?
结束
Y
N
保存电平状态，开始记时。
有跳变吗？
N
Y
N
是合法跳变吗？
Y
停止记时，保存当前电平状态
Y
T>=190μs?
N
开始记时

- **写卡程序框图：**

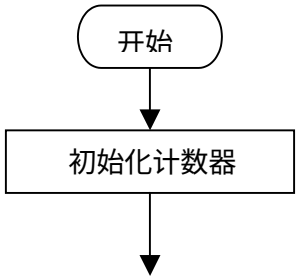


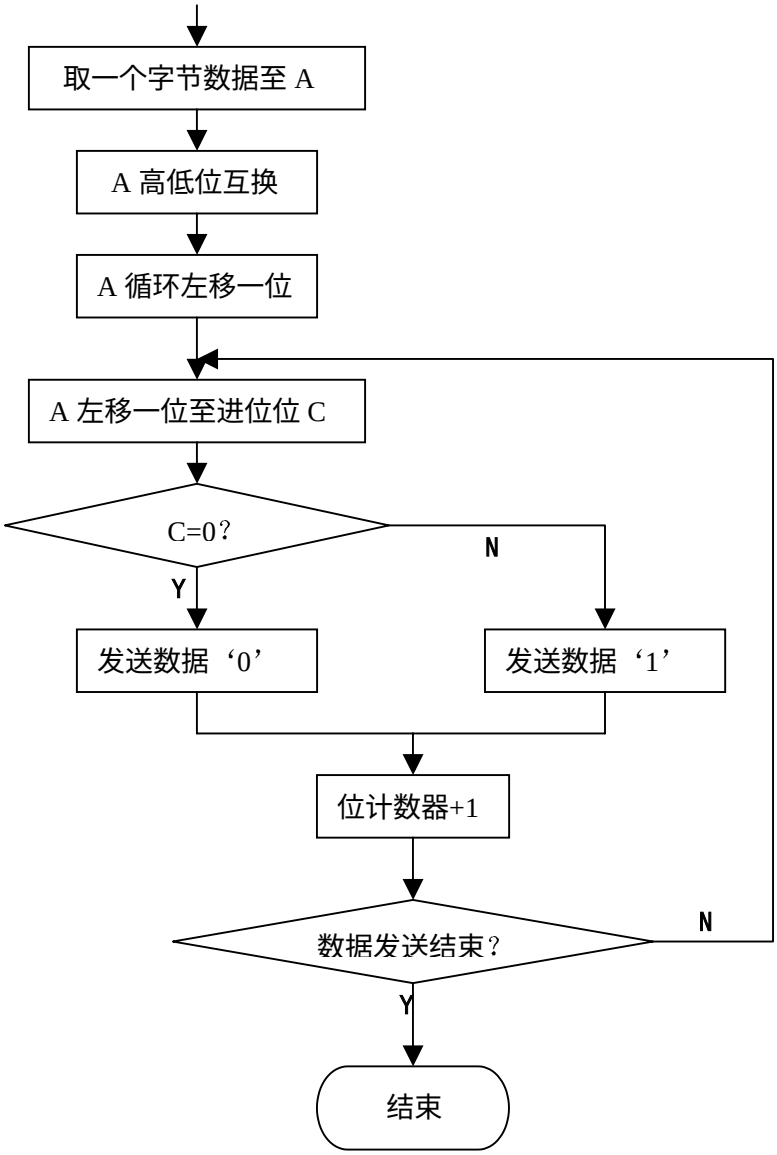
该程序分为多个子程序各子程序的程序框图如下：

- **数据流发送子程序：**


开始
初始化计数器
取一个字节数据至 A
A 左移一位至进位位 C
C=0?
Y
N
发送数据‘0’
发送数据‘1’
位计数器+1
字节发送结束?
N
Y
位计数器清零
字节计数器+1
数据发送结束?
N
Y
结束

- 发送地址子程序：


开始
初始化计数器



- 其他子程序和以上两个类似，分别完成发送指定数据流的作用。此外还要编制四个延时子程序，分别代表 LGAP_DELAY、SGAP_DELAY、ONE_DELAY、ZERO_DELAY。

- **读写程序调用接口：**

读卡程序入口参数：

参数名	功能
R0	存放读取数据的存储缓冲区的首地址指针
R1	保存输入口电平状态的
R2	读取数据的位计数器
R3	字节内数据位记数
R4	缓冲区字节计数器
R5	定时器时间的暂存器
R6	读数据块计数器
R7	输入口电平状态暂存

写卡程序入口参数：

参数名	功能
R0	欲写入数据的暂存区的地址指针
R1	卡密码的暂存区的地址指针
R2	操作的命令代码
R3	欲写入数据的卡中 EEPROM 块的块地址
00 位	指令执行结束标志
01 位	命令中的 LOCK 位值

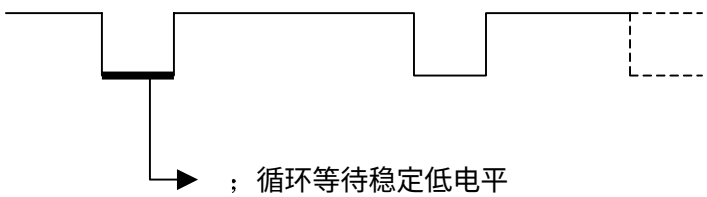
- **程序源代码：**

读卡程序代码：

该程序完成功能为，读取 E5550 卡中的 7 个 Block 中的数据，存入片内地址 30H 开始的 28 个字节中。并比较卡中第 4 块中的数据是否为 14H 14H 14H 14H，如果是则提示绿灯亮，否则提示红灯亮。程序在 ATMEL89C51/6MHz 条件下运行。卡片的模式设置如前所示。

```
        ORG     0000H
        AJMP    TEST
;-------------------------------------------------------
;------------------ MAIN FUNCTION ----------------------
;-------------------------------------------------------
TEST:   MOV     SP,#68H
        MOV     TMOD,#11H           ；单片机初始化
        MOV     4CH,#14H            ；准备比较数据
        MOV     4DH,#14H
        MOV     4EH,#14H
        MOV     4FH,#14H
        ACALL   GREEN
        ACALL   RED
        ACALL   ALL                 ；单片机自检提示
TEST0:  ACALL   READ                ；读卡子程序
        ACALL   COMP                ；比较子程序
        JNB     00H,TEST1
        ACALL   GREEN               ；比较正确提示
        AJMP    TEST2
TEST1:  ACALL   RED                 ；比较错误提示
TEST2:  ACALL   DELAY
        AJMP    TEST0
        RET                         ；结束
;-------------------------------------------------------
;----------------- READ FUNCTION -----------------------
;-------------------------------------------------------
```

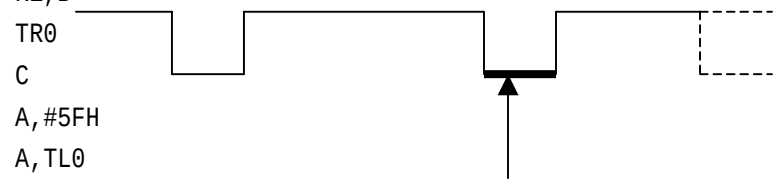
```
READ:   MOV     A,#00H
READD:  CLR     00H
        MOV     R5,#00H              ；缓存寄存器清零 ‘R5’
        MOV     R6,#07H              ；读 7 个数据块‘R6’
        MOV     B,R6
        MOV     A,#20H
        MUL     AB
        MOV     R2,A                 ；计算读数据位数总和‘R2’
        MOV     B,R6
        MOV     A,#04H
        MUL     AB
        MOV     R4,A                 ；计算读出数据的字节数‘R4’
        MOV     R3,#00H              ；缓存寄存器清零‘R3’
        MOV     R0,#30H
LOOPR:  MOV     @R0,#00H
        INC     R0
        DJNZ    R4,LOOPR             ；清数据存储区数据
        MOV     R0,#30H              ；置数据存储区首地址‘ R0’
        MOV     R1,#00H              ；状态寄存器清零‘R1’
MAIN:   CLR     TR0
        MOV     TH0,#00H
        MOV     TL0,#00H             ；记时器初始化
MAINL:  JB      P1.1,MAINL
        JB      P1.1,MAINL
        JB      P1.1,MAINL
        JB      P1.1,MAINL
        JB      P1.1,MAINL
        MOV     R1,#00H
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP                          ；循环等待稳定低电平
MAIN0:  MOV     A,#00H
```

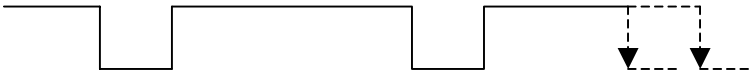


```
        MOV     C,P1.1
        RLC     A
        MOV     R7,A
        XRL     A,R1
        JNZ     MAINM           ；跳变检测
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        AJMP    MAIN0
MAINM:  MOV     B,R7
        MOV     R1,B
        MOV     TH0,#00H
        MOV     TL0,#07H
        SETB    TR0             ；开始记时
        NOP
        NOP
        NOP
        NOP
        NOP
MAIN1:  MOV     A,#00H
        MOV     C,P1.1
        RLC     A
        MOV     R7,A
        XRL     A,R1
        JNZ     MAIN2           ；跳变检测
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        AJMP    MAIN1
MAIN2:  MOV     B,R7
        MOV     R1,B
        CLR     TR0
        CLR     C
        MOV     A,#0AFH
```

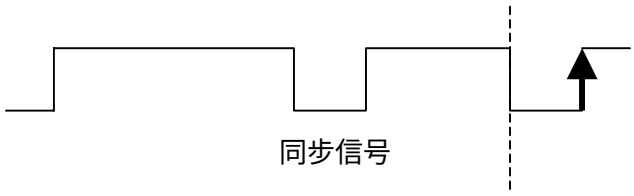
```
        SUBB    A,TL0
        JC      RIGHT              ；是否为一个半位周期
```

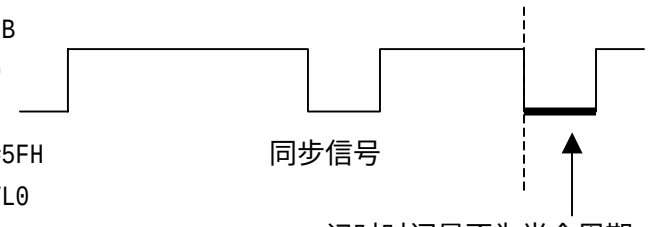
```
        AJMP    MAIN
RIGHT:  MOV     TH0,#00H
        MOV     TL0,#07H
        SETB    TR0                ；从新开始记时
MAIN5:  MOV     A,#00H
        MOV     C,P1.1
        RLC     A
        MOV     R7,A
        XRL     A,R1
        JNZ     MAIN6              ；跳变检测
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        AJMP    MAIN5
MAIN6:  MOV     B,R7
        MOV     R1,B
        CLR     TR0
        CLR     C
        MOV     A,#5FH
        SUBB    A,TL0
        JNC     RIGHT2             ；是否为半个位周期的跳变
        AJMP    MAIN
RIGHT2: MOV     TH0,#00H
        MOV     TL0,#07H
        SETB    TR0
MAIN7:  MOV     A,#00H
        MOV     C,P1.1
        RLC     A
        MOV     R7,A
        XRL     A,R1
        JNZ     MAIN8              ；检测跳变
        NOP
        NOP
        NOP
        NOP
        NOP
```



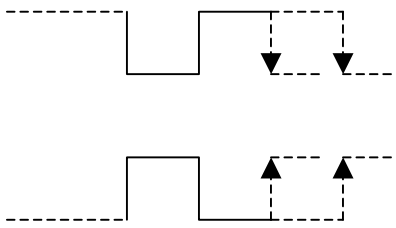


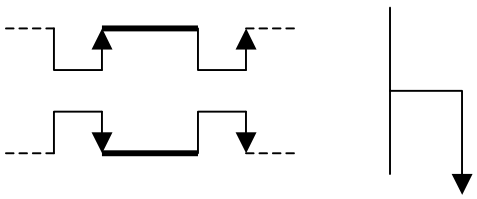
```
        NOP
        NOP
        AJMP    MAIN7
MAIN8:  CLR     TR0
        MOV     R5,TL0
        MOV     B,R7
        MOV     R1,B
        CLR     C
        MOV     A,#5FH
        SUBB    A,TL0
        JC      MAIN9                   ; 是否为合法跳变
        AJMP    MAIN
MAIN9:  MOV     TH0,#00H
        MOV     TL0,#10H
        SETB    TR0
        MOV     A,#96H                  ; 跳变类型
        SUBB    A,R5
        JC      READ2                   ; 跳变是数据位跳变‘0’
                                        ; 数据流以‘0’开头
------------------------------------------------------
;-----------------------------------------------------
READ1:  MOV     A,#00H                  ; 是同步结束跳变
        MOV     C,P1.1                  ; 数据流以‘1’开头
        RLC     A
        MOV     R7,A
        XRL     A,R1
        JNZ     READ11                  ; 检测跳变
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        AJMP    READ1
READ11: MOV     B,R7
        MOV     R1,B
        CLR     TR0
        CLR     C
        MOV     A,#5FH
        SUBB    A,TL0
        JNC     READ2                   ; 记时时间是否为半个周期
        AJMP    ERRORR
READ2:  MOV     A,@R0                   ; 数据位解码，存放数据进
        CJNE    R1,#01H,READ5           ; 数据缓冲区
```





```
        CLR     C                   ；数据位为‘0’
        AJMP    READ6
READ5:  SETB    C                   ；数据位为‘1’
READ6:  RLC     A                   ；取数据位进寄存器 A
        MOV     @R0,A
        INC     R3                  ；字节内位记数器+1
        CJNE    R3,#08H,READ7       ；是否满一个字节
        MOV     R3,#00H             ；字节满时字节内记数器清零
        INC     R0                  ；字节计数器+1
READ7:  DJNZ    R2,READ8            ；数据读取是否结束？
        RET                         ；结束操作返回调用程序
READ8:  MOV     TH0,#00H
        MOV     TL0,#10H
        SETB    TR0
READ3:  MOV     A,#00H
        MOV     C,P1.1
        RLC     A
        MOV     R7,A
        XRL     A,R1
        JNZ     READ4               ；检测跳变
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        AJMP    READ3
READ4:  CLR     TR0
        MOV     B,R7
        MOV     R1,B    ；11
        CLR     C
        MOV     A,#5FH  ；00
        SUBB    A,TL0
        JC      READ2   <-----      ；跳变时间如果是一个周期，则认为
        MOV     TH0,#00H            ；相邻两位数据相反。跳转区数据处
        MOV     TL0,#07H
        SETB    TR0
        AJMP    READ1   <-----      ；跳变时间是半个周期，则认为相邻
ERRORR: AJMP    READD               ；两位数据相同，需要继续检测跳变。
```





```
;-------------------------------
;-------- DELAY FUNCTION -------
;-------------------------------; 10
DELAY:  MOV     R0,#30H
DELAY0: MOV     R1,#30H          ; 01
DELAY1: NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        DJNZ    R1,DELAY1
        DJNZ    R0,DELAY0
        RET
;-------------------------------------------------
;-------------- GREEN   FUNCTION -----------------
;-------------------------------------------------
GREEN:  MOV     R2,#07H
GREEN1: CLR     P1.2
        ACALL   DELAY
        SETB    P1.2
        ACALL   DELAY
        DJNZ    R2,GREEN1
        RET
;-------------------------------------------------
;-------------- RED FUNCTION ---------------------
;-------------------------------------------------
RED:    MOV     R2,#07H
RED1:   CLR     P1.3
        ACALL   DELAY
        SETB    P1.3
        ACALL   DELAY
        DJNZ    R2,RED1
        RET
;-------------------------------------------------
;-------------- ALL FUNCTION ---------------------
;-------------------------------------------------
ALL:    MOV     R2,#07H
ALL1:   CLR     P1.3
        CLR     P1.2
        ACALL   DELAY
        SETB    P1.3
```

```
        SETB    P1.2
        ACALL   DELAY
        DJNZ    R2,ALL1
        RET
;-------------------------------------------------
;--------------- COMPARA FUNCTION ----------------
;-------------------------------------------------
COMP:   MOV     R2,#04H
        MOV     R0,#40H
        MOV     R1,#4CH
COMP1:  MOV     A,@R0
        CLR     C
        SUBB    A,@R1
        JNZ     CERROR
        INC     R1
        INC     R0
        DJNZ    R2,COMP1
        SETB    00H
        RET
CERROR: CLR     00H
        RET
```

写卡程序代码：

该写卡程序完成功能为：根据用户提供参数，向 E5550 卡发射指定命令序列。该程序可以完成发射所有四种命令格式命令的功能。测试条件和上面给出的读卡程序相同。

```
        ORG     0000H
        AJMP    MAIN
;-----------------------------------------
;------------ main function --------------
;-----------------------------------------
MAIN:   MOV     SP,#68H
        MOV     R0,#30H
        MOV     R1,#4CH
        MOV     30H,#00H
        MOV     31H,#08H
        MOV     32H,#80H
        MOV     33H,#0E8H
        MOV     4CH,#00H
        MOV     4DH,#01H
        MOV     4EH,#02H
        MOV     4FH,#03H
        MOV     R2,#02H
```

```
        MOV     R3,#00H
        CLR     00H
        CLR     01H
        CLR     02H
COMP1:  CJNE    R2,#01H,COMP2
RDA:    CLR     P1.0
        NOP
        ACALL   SENDC
        ACALL   SENDL
        ACALL   SENDD
        ACALL   SENDA
        ACALL   DELAYG
        ACALL   DELAYG
        ACALL   DELAYG
        SETB    00H
        RET
COMP2:  CJNE    R2,#02H,COMP3
RDP:    CLR     P1.0
        NOP
        ACALL   SENDC
        ACALL   SENDP
        ACALL   SENDL
        ACALL   SENDD
        ACALL   SENDA
        ACALL   DELAYG
        ACALL   DELAYG
        ACALL   DELAYG
        SETB    00H
        RET
COMP3:  CJNE    R2,#03H,COMP4
RWP:    CLR     P1.0
        NOP
        ACALL   SENDC
        ACALL   SENDP
        SETB    P1.0
        ACALL   DELAYG
        ACALL   DELAYG
        ACALL   DELAYG
        ACALL   DELAYG
        SETB    00H
        RET
COMP4:  CJNE    R2,#04H,WRITER
RST:    CLR     P1.0
        NOP
```

```
        ACALL   SENDC
        SETB    P1.0
        ACALL   DELAYG
        ACALL   DELAYG
        ACALL   DELAYG
        ACALL   DELAYG
        SETB    00H
        RET
WRITER: CLR     00H
        RET
;----------------------------------------
;----------- send data function -----------
;----------------------------------------
SENDD:  NOP
        NOP
        NOP
        NOP
        NOP
        CLR     C
        MOV     R4,#00H
        MOV     R5,#00H
LOOPD:  MOV     A,@R0
LOOPS:  RLC     A
        SETB    P1.0
        JC      SENDO
        ACALL   DELAYZ
        NOP
        AJMP    SENDN
SENDO:  ACALL   DELAYO
        NOP
        NOP
        NOP
SENDN:  CLR     P1.0
        ACALL   DELAYL
        INC     R4
        CJNE    R4,#08H,SENDD0
        MOV     R4,#00H
        INC     R5
        INC     R0
        CJNE    R5,#04H,SENDD1
        RET
SENDD0: NOP
        NOP
        NOP
```

```
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        AJMP    LOOPS
SENDD1: NOP
        NOP
        NOP
        NOP
        AJMP    LOOPD
;----------------------------------------
;---------- send addrss function ----------
;----------------------------------------
SENDA:  CLR     C
        MOV     R4,#00H
        MOV     A,R3
        SWAP    A
        RLC     A
LOOPA:  RLC     A
        SETB    P1.0
        JC      SENDAO
        ACALL   DELAYZ
        NOP
        AJMP    SENDAN
SENDAO: ACALL   DELAYO
        NOP
        NOP
        NOP
SENDAN: CLR     P1.0
        ACALL   DELAYG
        INC     R4
        CJNE    R4,#03H,LOOPA
        SETB    P1.0
        ACALL   DELAYG
        RET
;----------------------------------------
;---------- send code function ----------
;----------------------------------------
SENDC:  CJNE    R2,#04H,SENDC0
        CLR       P1.0
        ACALL   DELAYG
```

```
        NOP
        NOP
        NOP
        NOP
        NOP
        SETB    P1.0
        ACALL   DELAYO
        NOP
        NOP
        NOP
        NOP
        NOP
        CLR     P1.0
        ACALL   DELAYG
        NOP
        NOP
        NOP
        NOP
        NOP
        SETB    P1.0
        ACALL   DELAYO
        NOP
        NOP
        NOP
        NOP
        NOP
        CLR     P1.0
        ACALL   DELAYG
        NOP
        RET
SENDC0: CLR     P1.0
        ACALL   DELAYG
        NOP
        NOP
        NOP
        NOP
        NOP
        SETB    P1.0
        ACALL   DELAYO
        NOP
        NOP
        NOP
        NOP
        NOP
```

```
        CLR       P1.0
        ACALL   DELAYG
        NOP
        NOP
        NOP
        NOP
        NOP
        SETB      P1.0
        ACALL   DELAYZ
        NOP
        NOP
        NOP
        NOP
        NOP
        CLR       P1.0
        ACALL   DELAYG
        RET
;---------------------------------------
;-------- SEND LOCK BIT ----------------
;---------------------------------------
SENDL:  NOP
        NOP
        SETB    P1.0
        JB      01H,SENDL0
        ACALL   DELAYZ
        NOP
        AJMP    SENDL1
SENDL0: ACALL   DELAYO
        NOP
        NOP
        NOP
SENDL1: CLR     P1.0
        ACALL   DELAYL
        RET
;-----------------------------------------
;------------ SEND PASSWORD ---------------
;-----------------------------------------
SENDP:  CLR     C
        MOV     R4,#00H
        MOV     R5,#00H
LOOPP:  MOV     A,@R1
LOOPPS: RLC     A
        SETB    P1.0
        JC      SENDPO
```

```
        ACALL   DELAYZ
        NOP
        AJMP    SENDPN
SENDPO: ACALL   DELAYO
        NOP
        NOP
        NOP
SENDPN: CLR     P1.0
        ACALL   DELAYL
        INC     R4
        CJNE    R4,#08H,SENDP0
        MOV     R4,#00H
        INC     R5
        INC     R1
        CJNE    R5,#04H,SENDP1
        RET
SENDP0: NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        AJMP    LOOPPS
SENDP1: NOP
        NOP
        NOP
        NOP
        AJMP    LOOPP
;---------------------------------------
;-------------- gap delay --------------
;---------------------------------------
DELAYG: MOV     R7,#0EH
LOOPG:  NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
```

```
        DJNZ    R7,LOOPG
        RET
;-------------------------------------
;------------ '1' delay --------------
;-------------------------------------
DELAYO: MOV     R7,#11H
LOOPO:  NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        DJNZ    R7,LOOPO
        RET
;-------------------------------------
;------------ '0' delay --------------
;-------------------------------------
DELAYZ: MOV     R7,#04H
LOOPZ:  NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        DJNZ    R7,LOOPZ
        RET
;--------------------------------------
;------------ GAP DELAY1----------------
;--------------------------------------
DELAYL: MOV     R7,#0DH
LOOPN:  NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        DJNZ    R7,LOOPN
        RET
```

说明：

以上两个程序只是示范程序，所以将其完成的功能参数的设置和单片机的模式设置放在一起。用户实际使用时可以只选择其中的读写程序部分，而对单片机模式的设置可以和其他应用结合在一起完成。对读写参数可以在使用时设置。另外，如果用户选择使用其他型号的单片机，引起命令和控制的变化可以参照硬件说明书进行修改。由于程序使用软件模拟硬件时序，所以程序使用的振荡器的振荡频率对程序的影响很大。如果要变换振荡器则要同时对程序中的部分常数和 NOP 语句数量进行调整，否则程序将不能正常运行。（ **注：**程序是假设 IC 卡设置为：Manchester 编码、RF/32、125kHz、MaxBlock=7 的条件下调试的。这些设置同样对程序读写时序的产生有影响，改变这些设置时同样要对程序进行较大的改动）

A．E5560 卡读写程序：

E5560 卡和 E5550 卡读写程序的基本原理是相同的，只是 E5560 卡和 E5550 卡在功能结构上存在一些差异。E5560 卡在卡片中封装了一个加密逻辑模块，可以提供比 E5550 卡高的多的安全特性。下面对 E5560 卡的读写特性进行简单介绍（此处只介绍与 E5560 有关的读写特性，而和 E5550 相同的部分将不再详述）。

E5560 卡内置 320 位的 EEPROM。这些存储区分为 10 个 Block 每块 32 位。和 E5550 卡一样，10 个 Block 中的 Block 0 也是作为卡片的控制块用于存储卡的控制信息。E5560 卡的在后一个 Block 9 中存放的仍然是口令密码，所不同的是 Block 9 的第四位存放的不是密码信息而是 4 位控制信息，有关控制信息将在以后加以说明。E5560 卡的 Block 中不包括单独的 LOCK 位，这些块锁定位是作为控制信息的一部分存放在卡片的 EEPROM 中。E5560 卡的控制信息也不局限与 Block 0 一个块中而是分为四部分存放在不同的数据块中。E5560 卡中的其他部分内存放用户数据，其中 Block 1~Block 4 存放用户数据，如果启动加密验证功能则 Block 5~Block 8 存放 Crypto Key 数据。E5560 内部 EEPROM 的结构如图：