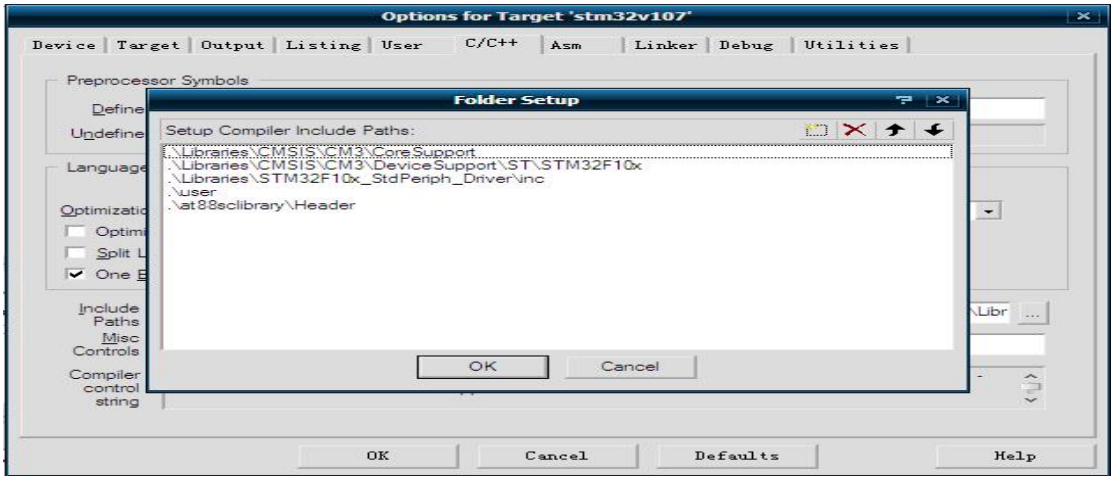
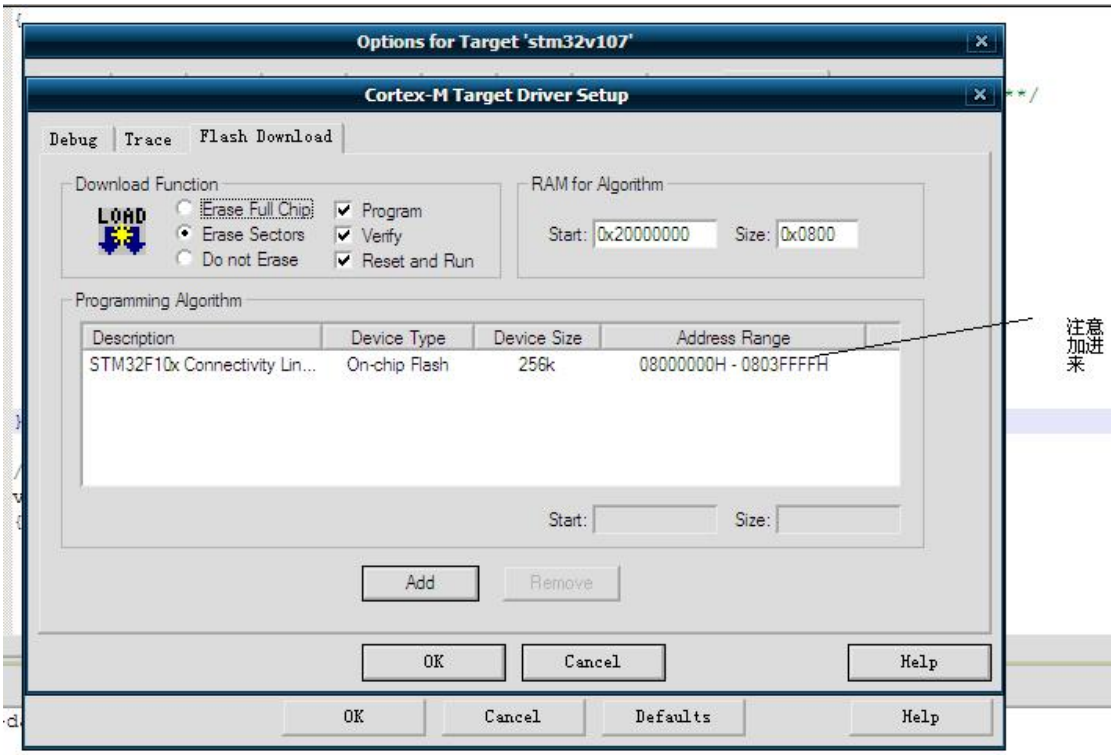
加密调试

@yedasheng

1 把 at88sc.的源文件放到文件夹 at88sclibrary 下，在 keil 工程里边新建一个文件组 at88sc，把 at88sclibrary 文件夹下的。C 文件加进来（除了 CM_test,c， CM_CARDDET.C,CM_POWER.c），在 keil 里边设置头文件



2 注意用工程模板建立新项目时，要注意下边这行有没有，没有根据芯片加下边的这行，否则下载不了。



3 更改文件 CM_I2C.H，把管脚改为 stm32 对应的控制管脚（硬件上注意加上拉电阻 4.7K）。具体参考 CM_I2C.H 文件。

4 在 CM_I2C.C，CM_NOPOWER.C 增加头文件#include "stm32f10x.h"，注意放在其他头函数之前。

5 在测试 Unlock Config Zone 注意下边 ucData 的数值，atmel 默认值为下边值，若不对，超出 8 次，芯片就被锁住了。代理商给的程序，这些数据不是针对 0104，所以不幸芯片被锁住，注意所用的芯片的初始值，0104 为 0xdd4297

```
    // Unlock Config Zone

    ucData[0] = 0xdd;
    ucData[1] = 0x42;
    ucData[2] = 0x97;
    ucReturn = cm_VerifySecureCode(ucData);
```

Unlock Config Zone 的密码可以改，但一旦熔丝位熔断不可改，所以密码要记住。

DD4297 应该在什么情况下都能过，只要熔丝没被烧断，随时可以配置

累计错误 4 次之前必须断电，否则会被锁死。断电次数就清零，重新计数

对 AT88SC 的来讲，Unlock Config Zone 时，芯片的 SecureCode 是至关重要的，如果在评估期间由于密码校验失败超过限制、或者改写了 SecureCode 而自己又忘记的话，那么这片 IC 的配置区、熔丝等将不可以再次进行修改

6 在测试 // Write Control for user zone 3 and set user zone 这部分时发现，函数 cm_SetUserZone 运行会死机，查看源代码居然是函数最后的 return 有问题，把下边函数的 return 改为 CM_LOW_LEVEL.ReceiveRet(ucCM_InsBuff,0);

```
uchar cm_SetUserZone(uchar ucZoneNumber, uchar ucAntiTearing)
{
uchar ucReturn;

ucCM_InsBuff[0] = 0xb4;
if (ucAntiTearing) ucCM_InsBuff[1] = 0x0b;
else                    ucCM_InsBuff[1] = 0x03;
    ucCM_InsBuff[2] = ucZoneNumber;
ucCM_InsBuff[3] = 0x00;

    // Only zone number is included in the polynomial
cm_GPAGen(ucZoneNumber);

if ((ucReturn = CM_LOW_LEVEL.SendCommand(ucCM_InsBuff))!= SUCCESS) return ucReturn;

// save zone number and anti-tearing state
```

```
    ucCM_UserZone = ucZoneNumber;
    ucCM_AntiTearing = ucAntiTearing;

    // done
return   CM_LOW_LEVEL.ReceiveRet(NU LL,0);
}
```

要把 CM_LOW_LEVEL.ReceiveRet(NULL,0); 改为 return CM_LOW_LEVEL.ReceiveRet(ucCM_InsBuff,0);不然程序会死掉

7 C0-C3（对应程序里边的 ucCi ，Ci1）可以在编程器里面写个随机数，S0-S3（对应程序里边的 ucSk ， Sk1）规格书说可以不写。
8 可以随时读取配置区的 NC，哪怕芯片被锁死。
9 认证或者加密最主要的参数 ucG，务必要对，否则几次芯片被锁死。这时哪怕熔丝位没熔断也知道 securecode 也无法配置了。不过这时可以对配置成标准模式的用户区访问，当成普通的 eeprom 用。

配置芯片思路：
1 用默认的 SecureCode（0xdd4297）进行 Unlock Config Zone。
// Unlock Config Zone

```
    ucData[0] = 0xdd;
            ucData[1] = 0x42;
            ucData[2] = 0x97;
    ucReturn = cm_VerifySecureCode(ucData);
```

2 写 NC，可以随机也可以按顺序，若是用 stm32，就用 stm32 本身自带的 ID（96 位，不是简单累加，所以要随机 NC，可以对 id 进行 crc32，取 4 个字节，前边三个字节用 ff），或者通过上位机来设置。

接着进行这步：
（用这步操作是为了提高加密等级，也可以不用）在配置前先往用户区写入一个密码 key，也可以符合某种算法格式的数据 securedata，例如读取 NC，把这些 NC 处理一下（加上一定规律的数据）之后进行一些例如 crc 等处理，之后把这串数据保存到用户区。

3Ci Sk 等寄存器可以用默认值，DCR 寄存器要用它的默认值 0xFF。
4 用自定义的算法以 NC 作为输入，计算 G，把四个 G0-G4 设成一样。
5 配置四个用户区，把所有的种子都设成一个值，AR0-3 PR0-3 也设成一样，这样就可以用一个种子访问整个 128 字节空间.

AR PR 分别设成密码模式,没有密码，对时序加密见下面的代码

```
        data[0]=0xD7;//ARnormal authentication, encrypted required
```

```
data[1]=0x23;//PR

cm_WriteConfigZone(0x20, data, 2, 0);

data[1]=0x63;//

cm_WriteConfigZone(0x22, data, 2, 0);

data[1]=0xa3;//

cm_WriteConfigZone(0x24, data, 2, 0);

data[1]=0xe3;//

cm_WriteConfigZone(0x26, data, 2, 0);
```

7 熔断熔丝。

之后该芯片就可以焊接到板上，或者给控制器下载程序前，iap 检查是否对芯片进行了配置，已配置加密芯片就可以下载用户代码，否则下载配置程序。

注意配置是一次写入，不能断电后，想对其他用户区再设置好像不行（有待确认）。

加密模式下的运用。

1 采用多个地方不同时刻，随机进行加密认证。最好不要在开机地方，免得破解者容易跟踪到。

2 在某个模块里边启动认证思路如下：

读取 NC，用自定义的算法以 NC 作为输入，计算 ucG

调用下边函数进行加密认证

ucReturn = cm_ActiveSecurity(1, ucG, NULL, TRUE);若返回值为 0 则认证加密通过（否则程序运行不正常的程序），访问用户区，读取 key 或者 securedata，看是否满足预订的格式，若是则程序正常执行，否则程序运行不正常的程序。

代理商给的源码为模拟的 iic，读数据采用下边的时序（参考 datasheet 的 28 页）

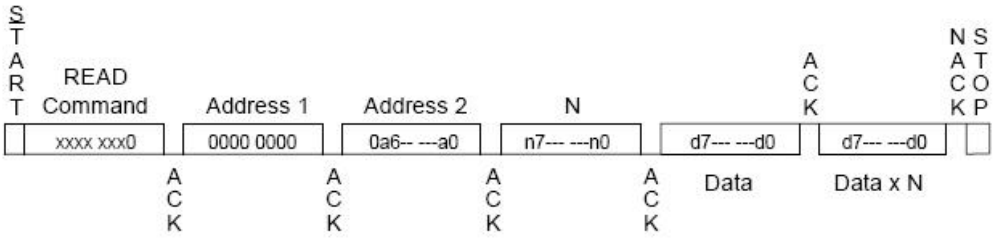
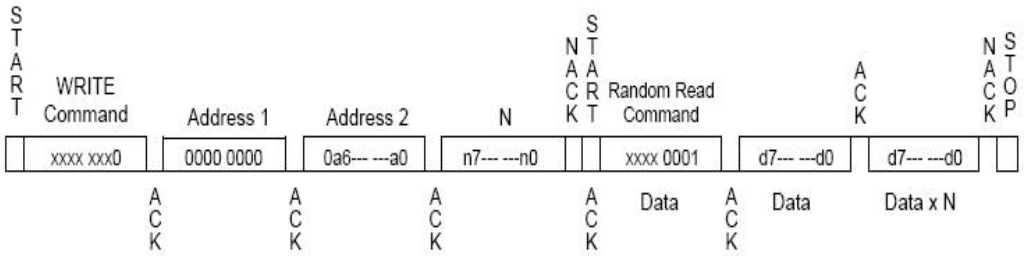


图 1

Stm32 的硬件 iic 不支持上边这种格式，要采用下边的时序

Figure 8-4. Random Read Command



改为硬件 iic(还没调通读，读出来数据不对)

1 读数据的 cm_ReadConfigZone 函数里边的 ucCM_InsBuff[0] = 0xb0;见 datasheet 的 29 页，采用图 1，ucCM_InsBuff[0] = 0xb6。

2 初始化 iic 先用 io 口发 5 个时钟。

3 更改 Cm_i2c.c 的相关函数，具体见源程序，里边的 nacktest 是为了实现上边图的第一个 NACK 而增加的全局变量。

问题： datasheet 根本没讲清楚上边时序的 Write command 虽然说 xxxxxxx0，其中高四位为芯片地址 b，即 0xbxxx0，但是 0xb2,0xb4 又是有其他用途，火大了。

对测试区 mtz 操作：

```
ucData[0] = 0x78;
ucData[1] = 0x34;
      ucReturn = cm_WriteConfigZone(0x0A, ucData, 2, FALSE);
      ucData[0] = 0x00;
      ucData[1] = 0x00;
      ucReturn = cm_ReadConfigZone(0x0A, ucData, 2);
      USART_SendByte(USART1,ucData[0] );
      USART_SendByte(USART1,ucData[1] );
```

用户区 3 设置成认证加密模式，其他三个区为标准模式。

对用户区 0 进行读写操作如下：

```
  ucReturn = cm_SetUserZone(0, FALSE);   //返回 0 说明操作成功,对 0 区操作
    // ucReturn = cm_SetUserZone(1, FALSE); ,对 1 区操作
   USART_SendByte(USART1,ucReturn    );

    for (i = 0; i < 32; ++i) ucData[i] = 0x78-i;
    ucReturn = cm_WriteSmallZone(0, ucData, 16);//对 0 区第一页写，一页为 16 字节
```

```
ucReturn = cm_WriteSmallZone(1, &ucData[16], 16);//对 0 区第二页写

    USART_SendByte(USART1,ucReturn    );


    for (i = 0; i < 16; ++i) ucData[i] = 0x00;
    ucReturn = cm_ReadSmallZone(0, ucData, 16);//对 0 区读 1 页

    for (i = 0; i < 16; ++i) USART_SendByte(USART1,ucData[i] );
   ucReturn = cm_ReadSmallZone(1, ucData, 16);//对 0 区读 2 页
```

对用户区 3 操作：

```
ucReturn = cm_ActivateSecurity(1, ucG, NULL, TRUE);   //发起认证加密，通过才能对 3 区操作
USART_SendByte(USART1,ucReturn );
ucReturn = cm_SetUserZone(3, FALSE); //指定对 3 区操作
    for (i = 0; i < 16; ++i) ucData[i] = 0x00;
    ucReturn = cm_ReadSmallZone(0, ucData, 16);//读出数据

    for (i = 0; i < 16; ++i) USART_SendByte(USART1,ucData[i] );//显示数据


    for (i = 0; i < 16; ++i) ucData[i] = 0x89-i;
    ucReturn = cm_WriteSmallZone(0, ucData, 16);//重新写数据进去
    USART_SendByte(USART1,ucReturn    );
    ucReturn = cm_SendChecksum(NULL);     //发送校验，不能省略，否则写不进去

    for (i = 0; i < 16; ++i) ucData[i] = 0x00;
    ucReturn = cm_ReadSmallZone(0, ucData, 16); //读出数据

    for (i = 0; i < 16; ++i) USART_SendByte(USART1,ucData[i] ); //显示数据
```

读取熔丝位，未熔断前 0x07

```
                ucReturn =cm_ReadFuse(ucData);
                    USART_SendByte(USART1,ucReturn);
                    USART_SendByte(USART1,ucData[0]);
```

在系统调试成功以后，才将 AT88SC0104C 中的熔丝熔断。熔断之后，配置不再更改，这一点需特别注意。熔断的过程如下：

1）向熔丝标志寄存器写入 06H；

2）向熔丝标志寄存器写入 04H；

3）向熔丝标志寄存器写入 00H。

```
ucReturn =cm_BurnFuse(0x6);
                                    USART_SendByte(USART1,ucReturn);
                                  ucReturn =cm_BurnFuse(0x4);
                                    USART_SendByte(USART1,ucReturn);
                                  ucReturn = cm_BurnFuse(0x0);
```

20140825 测试被修改 nc

1：新的芯片是 ff, 7f, 7f, 7f, 7f, 7f, 7f

2：写进0x2, 0x54, 0x1, 0x1, 0x0, 0x0, 0x20　后三位是变化的id

出去产品反馈，退回来的两台的加密芯片的现象是：1 号机居然可以重新配置加密，之后再也不行。四个区都可以操作。原来陈总是用了 1、2 区且熔丝位断了，按理是无法重新配置呀。2 号机 nc 读出来正常，12 区无法操作，需要密码，3，4 区可以正常操作。符合实际使用，就是验证密码通不过？

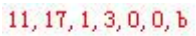
1 号机四个区都可以操作正常，nc 是我写进去。

```
 start
[2014:08:26:13:40:26][接收]channel 4 config fail:
 id =36
[2014:08:26:13:40:27][接收]channel 4 user zone0 ok:
[2014:08:26:13:40:27][接收]channel 4 user zone1 ok:
[2014:08:26:13:40:28][接收]channel 4 user zone2 ok:
[2014:08:26:13:40:28][接收]channel 4 user zone3 ok:
[2014:08:26:13:40:30][接收]VerifySecureCode fail
 CH 4:2,54,1,1,0,0,5
```

2 号机

```
 CH 4:11,17,1,3,0,0,b
[2014:08:26:13:36:12][接收]channel 4 config fail:
 id =36
[2014:08:26:13:36:12][接收]channel 4 user zone0 activesecurity fa
[2014:08:26:13:36:12][接收]channel 4 user zone0 fail:
[2014:08:26:13:36:13][接收]channel 4 user zone1 fail:
[2014:08:26:13:36:13][接收]channel 4 user zone2 ok:
[2014:08:26:13:36:13][接收]channel 4 user zone3 ok:
[2014:08:26:13:36:16][接收]write nc fail
 CH 4:11,17,1,3,0,0,b
[2014:08:26:13:36:16][接收]write nc fail
 CH 4:11,17,1,3,0,0,b
```

nc 可以读，2，3 用户区操作都正常



读到的 nc。熔丝位已断，无法再写 nc.

代理商的技术过来说 iic 时序有问题，用逻辑分析仪检测，对着 datasheet 更改时序。
1：更改文件中 cm_low.c 蓝色部分为更改
手册虽然说最大可以 1M，但实际测试只要大于 71k 就无法过。

```
void cm_Delay(uchar ucDelay)
{
uint ucTimer;
 //ucTimer = 7200/AT88SC_CLOCK ; //最初的，5K 左右
     ucTimer = 720/AT88SC_CLOCK;  //34k
    // ucTimer =20;//46k
    //ucTimer =10;//71k    就无法通过 VerifySecureCode 无论好坏芯片

     while(ucDelay)
     {

        while(ucTimer) ucTimer--;
       ucDelay--;
     }



}
```

补充说明：上边这个 clk 不能太高，在 20140904 这天，把下边几个问题解决，重新来验证 clk 的速率，发现这时 clk 可以调高，不会出错，所以应该是下边问题引起的。把上边函数里边全部注释掉，clk 达到 222K 都没问题。

```
void cm_Delay(uchar ucDelay)
{
uint ucTimer;
 //ucTimer = 7200/AT88SC_CLOCK ; //最初的，5K 左右
 /*    ucTimer = 720/AT88SC_CLOCK;  //34k
    // ucTimer =20;//46k
    //ucTimer =10;//71k
    //ucTimer =1;//147k
     while(ucDelay)
```

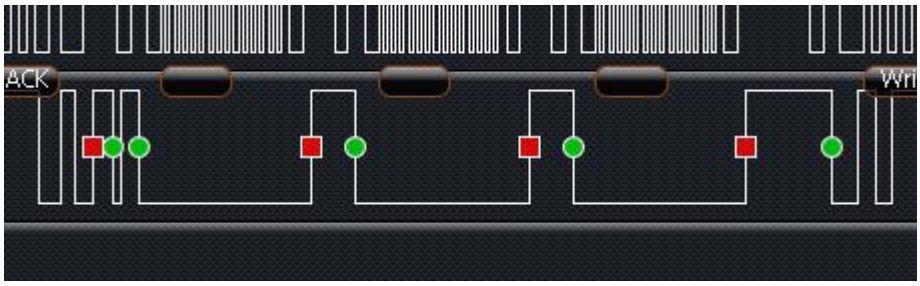
```
    {

      while(ucTimer) ucTimer--;
      ucDelay--;
    }

    */

}
```

2：下图为 Cm_Password 文件的函数 cm_VerifyPassword 里边的 CM_LOW_LEVEL.WaitClock(3,channel); 时序，3 个开始、结束时序。即从第三个绿色圆点开始。
下图的第二绿色圆点代理技术说不对。

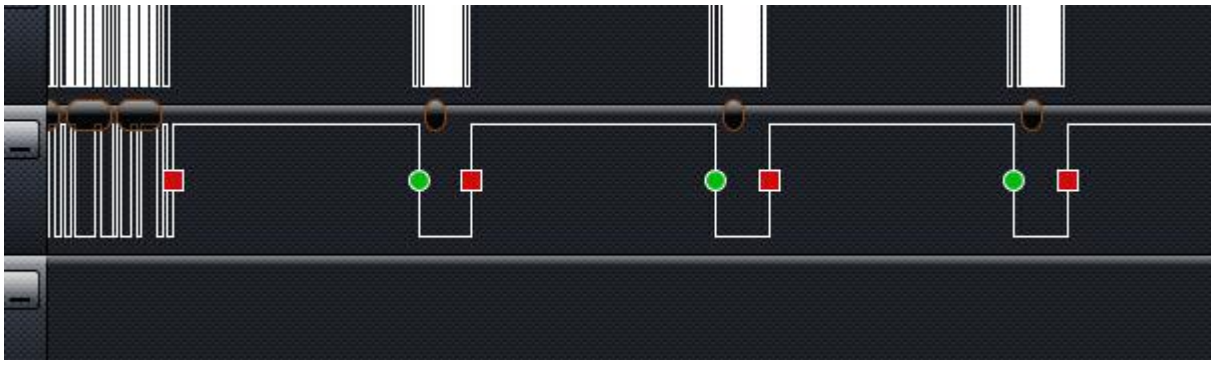


更改下边函数把 CM_DATA_LO(channel);注释掉就可以。

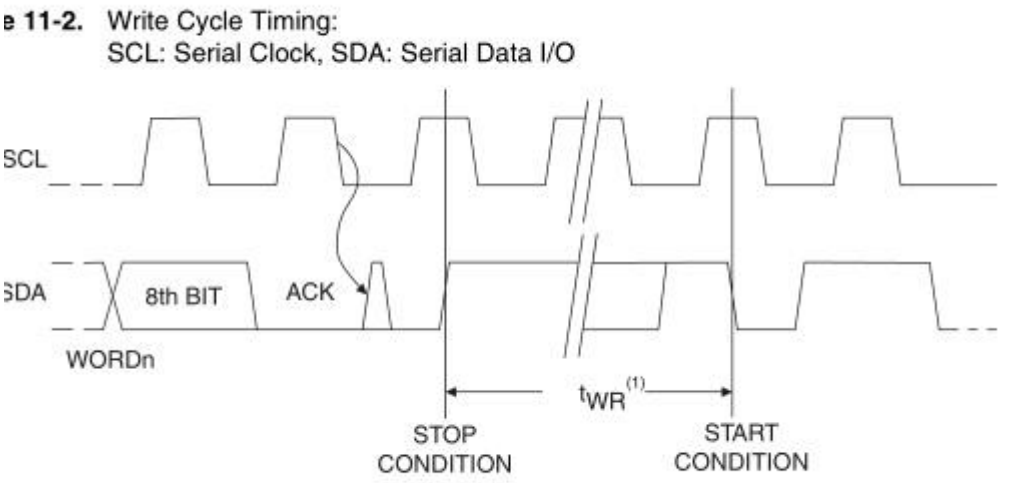
```
void cm_WaitClock(uchar loop,uchar channel)
{
    uchar i, j;

    //CM_DATA_LO(channel);  //20140902
    for(j=0; j<loop; j++) {
        cm_Start(channel);
        for(i = 0; i<15; i++) cm_ClockCycle(channel);
        cm_Stop(channel);
    }
```

}



3：把 stop 到开始的时间延长，即下图的红色到绿色的时间加长到 ms 级
手册的 75 页有说 twr 最大要 7ms 左右。所以起码要 8ms 比较保险。所以要更改文件 cm_i2c 的函数 cm_Stop，增加蓝色部分。
为了跟原来的延时函数区分增加了 delay_nms（）函数。




```
void cm_Stop(uchar channel)
{
    //CM_DATA_OUT;                          // Data line must be an
output to send a stop sequence
    CM_DATA_OUT(channel);
    cm_Clocklow(channel);
    //CM_DATA_LO;
    CM_DATA_LO(channel);
    cm_Delay(4);
    cm_Clockhigh(channel);
    cm_Delay(8);
```

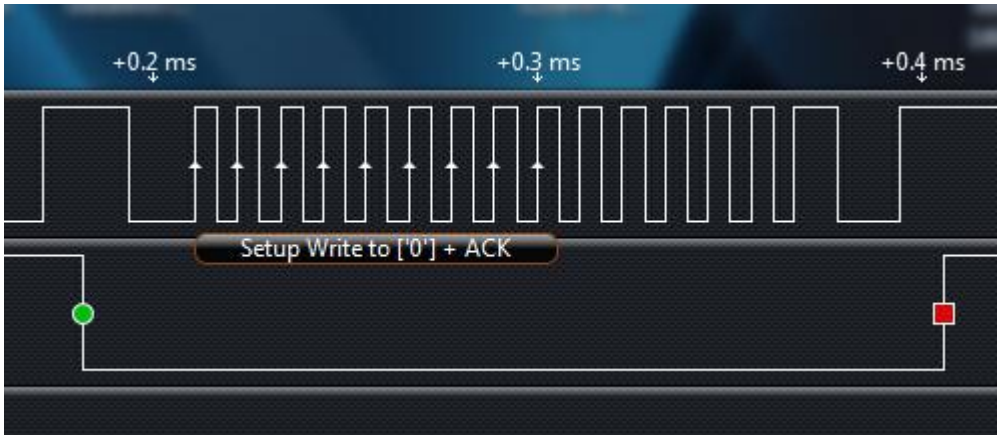
```
    CM_DATA_HI(channel);
    cm_Delay(4);
   //cm_Delay(200);//test
    delay_nms(8);//20140902
}
```

增加的部分。放在文件 cm_i2c 中。

```
void delay_nms(uint n)
{
  uint i,j;
    for (i=n;i>0;i--)
    {
        for (j=6400;j>0;j--);
    }
}
```

20140903　代理技术提出下边两个问题

1：



这个图，前面 9 个 SCL 脉冲发送了一个字节，后面多了 6 个脉冲，这个要确认一下，如果是多余了 6 个脉冲，最好去掉；如果是后一个字节没发完，要修改一下。

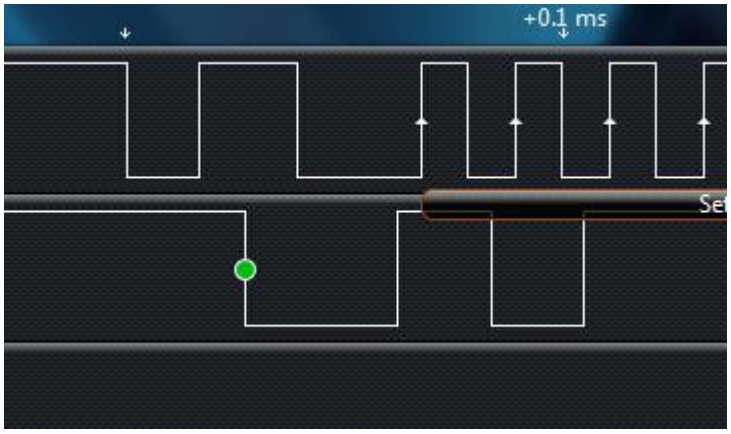
这个不是写一个字节是在文件 cm_i2c 中的函数 cm_WaitClock 里边产生，在手册上没找到相应的时序，改为 9 也正常。

```
void  cm_WaitClock(uchar  loop,uchar  channel)
{
uchar  i,  j;

//CM_DATA_LO(channel);     //20140902
```

```
for(j=0;  j<loop;  j++)  {
cm_Start(channel);
for(i  =  0;  i<15;  i++)  cm_ClockCycle(channel);

cm_Stop(channel);
}
}
```



在每个开始条件之前，SCL 都有一个没用的负脉冲，原理上不会影响，能去掉最好去掉/
文件 cm_i2c 的函数 cm_Start（）注释掉下边蓝色部分

```
void cm_Start(uchar channel)
{//iic start,clk=1then da from 1 to 0
                              // Data line must be an output to send a
start sequence
      CM_DATA_OUT(channel);
      //cm_Clocklow(channel);
      CM_DATA_HI(channel);
      cm_Delay(4);
      cm_Clockhigh(channel);
      cm_Delay(4);
      CM_DATA_LO(channel);
      cm_Delay(8);
      cm_Clocklow(channel);
      cm_Delay(8);
}
```