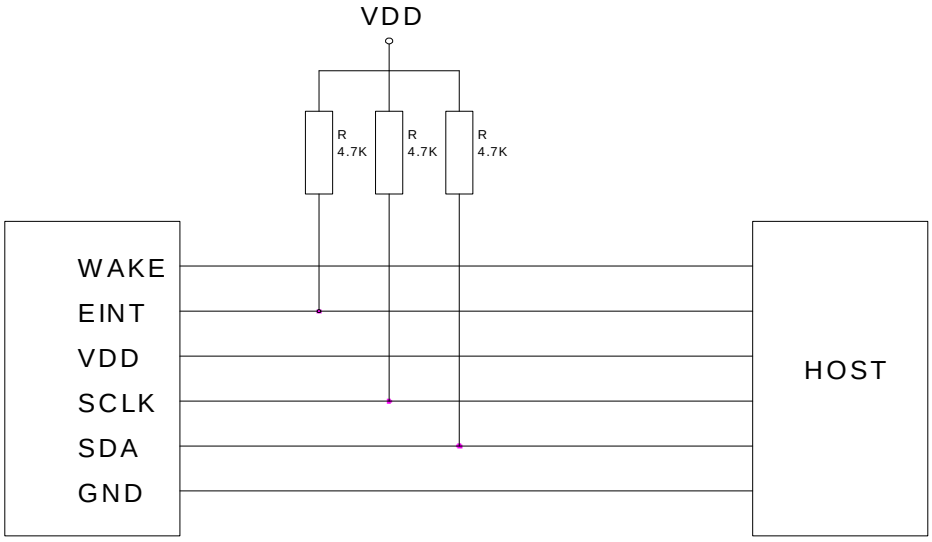
普容 N35007Z-A 项目 I2C 协议说明

Section 1　General Discription

普容公司的触摸屏模组采用的 Fotal 公司的系列芯片，其中：2.8 寸、3.5 寸、4.3 寸使用 FT5202 芯片；7 寸、8.9 寸使用的 FT5406 芯片。系列芯片均采用相同的通讯协议，包括 I2C，SPI，UART 通讯方式，大尺寸支持 USB 口通讯，在 Windows7 系统中自动识别。由于 N35007Z-A 项目使用 Fotal 公司的 FT5202 型号的 IC，此 IC 采用 I2C 协议，所以本篇文档仅限于对 I2C 协议的应用指导。

N35007Z-A 项目通讯连接结构如下：



模组有 6 个引脚连接到 HOST 端分别是 WAKE，EINT，VDD，SCLK，SDA，GND。其中 SCLK，SDA 是 I2C 的通讯引脚；ENIT 是外部中断脚和 I2C 通讯配合，当手触摸到触摸屏时，EINT 脚会被拉低，触发 HOST 端中断后，HOST 开始从触摸屏模组读取数据，读取完数据后，EINT 脚恢复高电平；WAKE 用于休眠模式的唤醒，常态为高电平，当模组进入休眠模式，通过把 WAKE 脚拉低 5ms 后再拉高来唤醒。

需要注意的是：ENIT，SCLK，SDA 在使用时，HOST 需要接 4.7k 的上拉电阻。

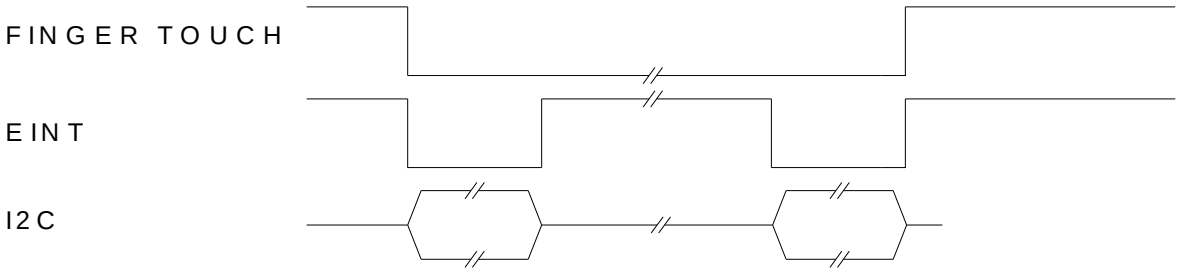
表 1 模组参数表

Sensor 通道	25
接口	I2C Slave 100K,200K,400K@3.3V
I2C 地址	0x70(8bit);0x38(7bit)
分辨率	4096*4096(可以根据 LCD 的分辨率进行调整)
传输数据速率	80packets/sec
反应时间	≦5ms
供电电压	3.3V(min:2.8V;max:3.6V)
IC 功耗	Normal Mode:4.5mA;Monitor Mode 3mA; Sleep Mode:0.1mA
运行温度	-45° to 85°
IC 封装	QFN48
IC 尺寸	6mm×6mm×0.85mm

Section2 I2C Bus Protocol

1、 EINT 与 I2C 的配合

当有手指按下时，EINT 引脚会如下图发出脉冲来触发中断：



当 EINT 引脚拉低时，HOST 需要触发中断，后通过 I2C 和模组进行通讯，获取需要的数据。建议 HOST 的中断方式设置为下降沿中断。

注意：

对于采用 MTK6235 芯片的手机方案，由于 6235 不支持下降沿中断，只支持电平中断，建议按照如下方式设置中断：在 EINT 拉低之前，中断脚设置为低电平中断，当 EINT 拉低后触发中断后，在低电平中断的中断函数中首先把中断脚设置成高电平中断；EINIT 低电平持续一段时间后会自动拉高，会触发 6235 的高电平中断，在中断函数中在把中断方式改成低电平中断。在下一次 EINT 拉低时又会触发。

2、 I2C 总线的规范

I2C 总线的电气特性如下图

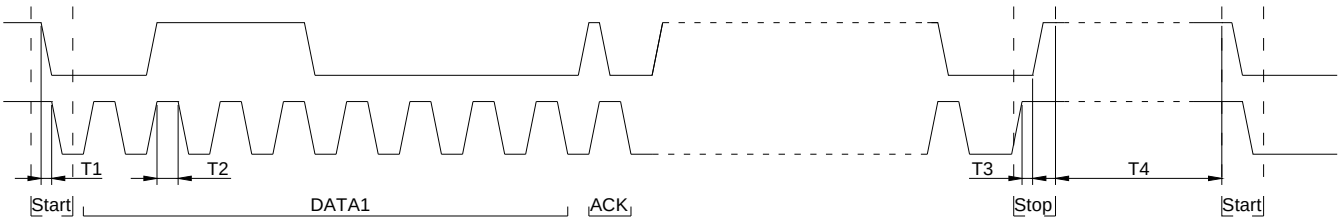
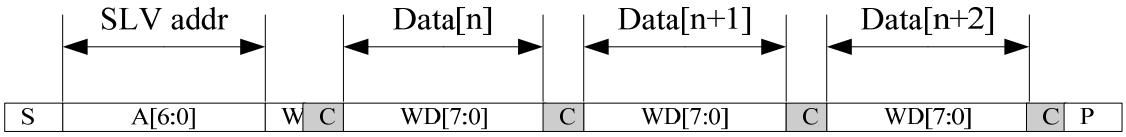


表 2 I2C Interface Timing Characteristic

Parameter	Min	Max	Unit
SCLK frequency		400	KHZ
Bus free time between a STOP and START condition (T4)	4.7		us
Hold time START condition(T1)	4		us
DATA setup time(T2)	250		us
Setup For STOP condition(T3)	4		us

3、 I2C 总线的 write 和 read

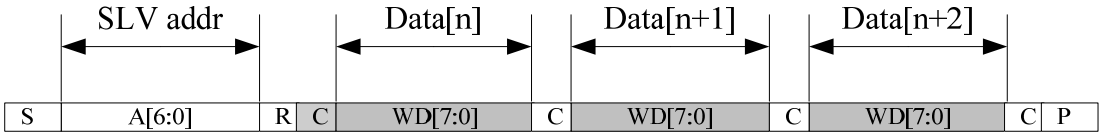
I2C Write



白色区域表示 Master 发出的数据，灰色区域表示 Slave 发出的数据。

字符	说明
S	I2C Start 或者 I2C Restart
A[6:0]	Slave Address，其值为 0x38
W	操作字，bit=0，表示写
C	ACK 信号
P	STOP 信号（也可以没有 STOP，直接由 Restart 开始下一个 Packet）

I2C Read



白色区域表示 Master 发出的数据，灰色区域表示 Slave 发出的数据。

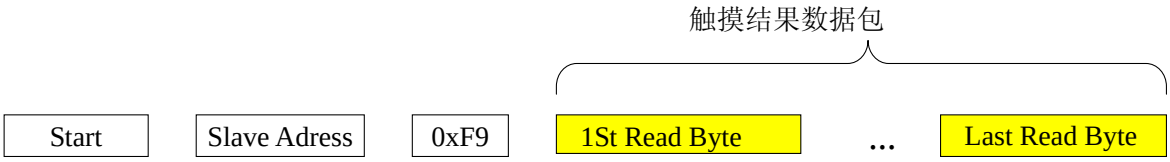
字符	说明
S	I2C Start 或者 I2C Restart
A[6:0]	Slave Address，其值为 0x38
R	操作字，bit=1，表示读
C	ACK 信号
P	STOP 信号（也可以没有 STOP，直接由 Restart 开始下一个 Packet）

4、 I2C 协议层

FT5202 的 I2C 协议层包括 2 种，第 1 种是通过 I2C 读取触摸结果数据包，第 2 种通过 I2C 读取触摸屏寄存器。由于 FT5202 本身的局限性，触摸信息并不是放置在寄存器中，而是通过读取一个触摸结果数据包来实现。触摸屏数据包的内容包括 5 点触摸坐标，手势，触摸手指个数等。触摸屏的一些配置参数则放置在寄存器中。且绝大部分的参数不需要修改，按默认配置即可，仅有个别参数需要用户去控制。其中最常用到的寄存器是 PMODE 寄存器（地址：0x3a）,此寄存器控制触摸屏模组运行在不同的模式下。2 种协议读取方式并不相同，后面会详细说明。

- **通过 I2C 读取触摸数据包**

基本格式如下图



Host 开始后发送设备地址 0x70 后，再发送 0xF9，模组便知道，host 要读取触摸结果数据包，后再 Start 再发送 0x71，连续读取数据，直到需要的最后一个数据。26Bytes 可以不需要读完，例如：只需要 2 点坐标，仅仅需要读取前 13Bytes 即可。

触摸结果数据包以 Head 开头，以 CRC 校验码结束,（16bit 的数据高位在前，低位在后）共有固定的 26 Bytes 数据。

表 3 26Bytes 触摸结果数据包

名称	长　度 (Byte)	意义
Byte0-1(Head)	2	触摸数据数据包的包头
Byte2	1	当前数据包的长度，由低 6bit 表示，代表当前传送的数据包数目，即由 TP Module 发给 Host 的数据的 Byte 数目
Byte3	1	当前的触摸点数，由低 4bit 表示
Byte4	1	手势（具体定义，见表 4 手势定义）
Byte5-6(X1)	2	bit11_0 表示触摸点 1 的横坐标；bit12,bit13 保留；bit15_14 表示触摸状态，11：无事件，00：down，01：up，10：repeat。 样例：0x01AF 表示手指按下，x 坐标 431；0x41AF 表示手指抬起，抬起时的 x 坐标 431；0x81AF 表示手指持续按下，x 坐标 431。
Byte7-8(Y1)	2	bit11_0 表示触摸点 1 的纵坐标；bit15_12 表示触摸点的 ID。
Byte9-10(X2)	2	说明同 X1
Byte11-12(Y2)	2	说明同 Y1
Byte13-14(X3)	2	说明同 X1
Byte15-16(Y3)	2	说明同 Y1
Byte17-18(X4)	2	说明同 X1
Byte19-20(Y4)	2	说明同 Y1
Byte21-22(X5)	2	说明同 X1
Byte23-24(Y5)	2	说明同 Y1
Byte26(CRC)	1	数据包前(N-1)个数据的 CRC 校验码，用于帮助 Host 检验这一个数据包的正确性. CRC 等于触摸结果数据包的前 25 个 Byte 的 XOR 运算结果.

BYTE0-1	Bit	Description
Head[15:0]	[15:0]	默认值 0xAAAA

BYTE2	Bit	Description
reserve	[7:6]	保留
Length[5:0]	[5:0]	当前传送的数据包的长度

BYTE3	Bit	Description
reserve	[7:4]	保留
Num[3:0]	[3:0]	当前按下的手指数

BYTE4	Bit	Description
Gesture[7:0]	[7:0]	手势信息，详细手势定义见**表 4**，

BYTE5-6（X1）	Bit	Description
Down_up[1:0]	[15:14]	触摸状态标志，0x11 表示无事件，0x00 表示 down，0x01 表示 up，0x10 表示 repeat。
Reserve[1:0]	[13:12]	保留
X1[11:0]	[11:0]	触摸点 1 的横坐标

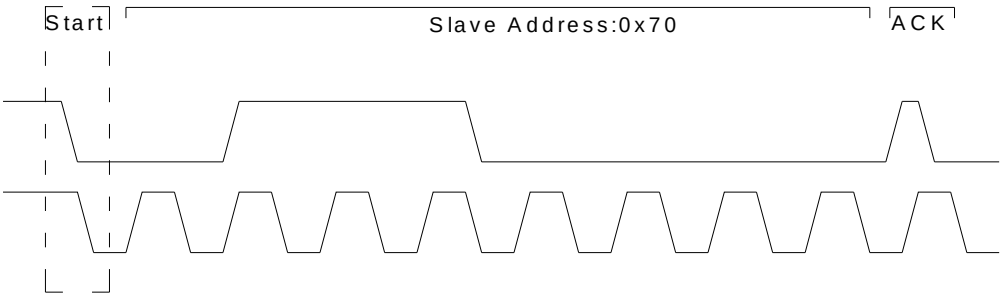
BYTE7-8(Y1)	Bit	Description
ID[3:0]	[15:12]	触摸点 1 的 ID，ID 意义详见后面说明。
Y1[11:0]	[11:0]	触摸点 1 的纵坐标。

BYTE26	Bit	Description
ECC[7:0]	[7:0]	数据包前 25Bytes 的 CRC 校验码，等于前 25Bytes 的 XOR 运算。

表 4 手势定义

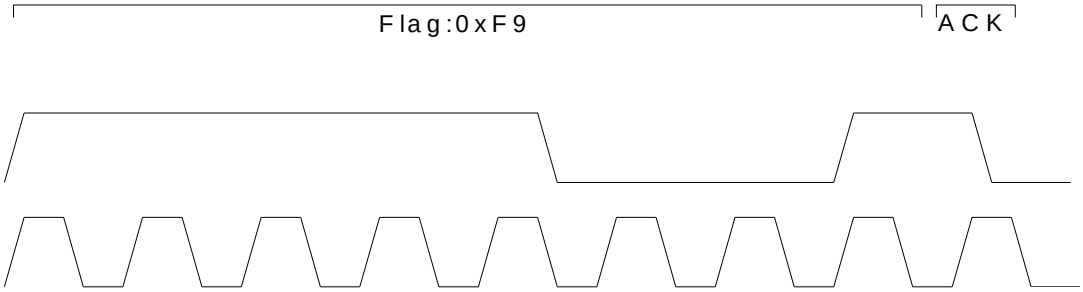
手势名称	定义	备注
MoveUp	0x10	
MoveDown	0x18	
MoveLeft	0x1c	
MoveRight	0x14	
MoveRightRotation	0x28	
MoveLeftRotation	0x29	
MoveZoomIn	0x48	to become big
MoveZoomOut	0x49	to become small
MoveDoubleClick	0x22	
MoveSingleClick	0x20	
MoveNone	0x00	

具体编程调试时，通讯过程可以参考如下波形：
波形 1 Start + Write Slave Address0x70



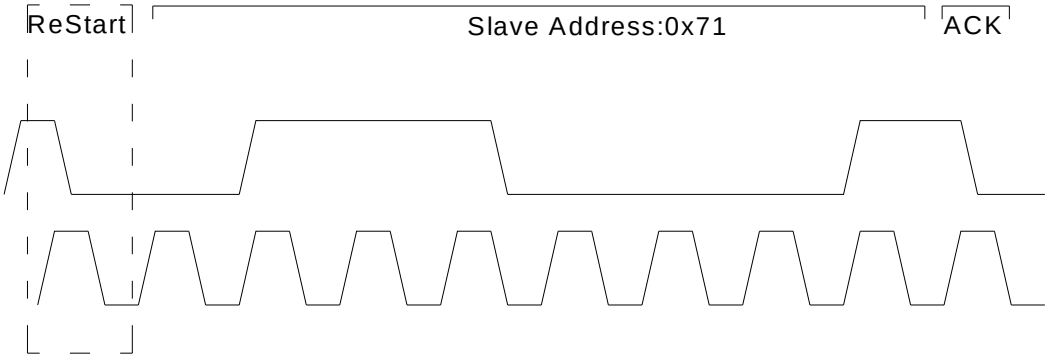
Host 发出开始信号，然后发出设备地址 0x70(写),等待 slave 应答，有 ACK 应答继续下一步；

波形 2 Write 0xF9



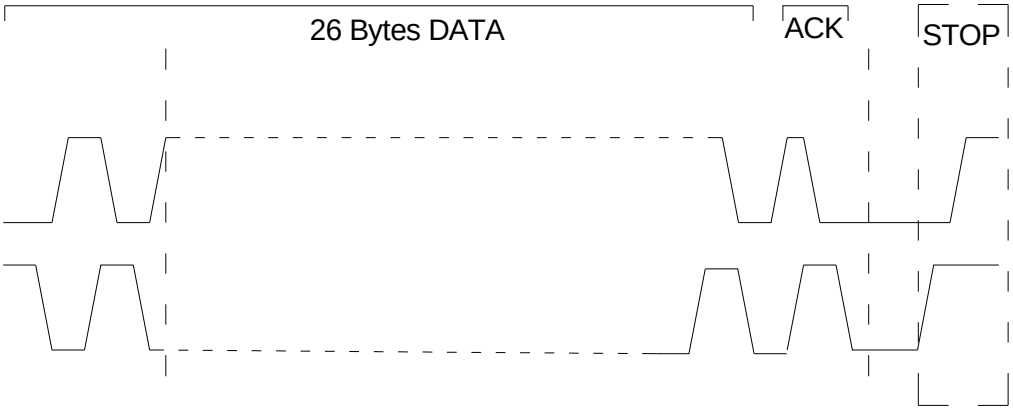
接到 ACK 应答后，Host 发送 0xf9，等待 ACK 应答，有则继续；

波形 3 Restart +Write Slave address 0x71



接到 ACK 应答后，Host 重新发送数据 0x71(读)，等待 ACK 应答，有则继续；

波形 4 26 Bytes DATA +Stop



Host 读取 26 个字节，读完后 Host 发出结束信号。

- **通过 I2C 读写寄存器**

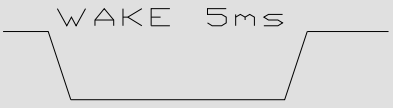
读写寄存器的目的在于控制触摸屏模组。由于模组 IC 中的绝大多数寄存器都是不需要 HOST 端区配置，所以读写寄存器的重要性要次于去读触摸数据包。不过有些寄存器是必须要 HOST 端控制的。其中最有必要的就是 PMODE 寄存器，即运行模式寄存器。此寄存器的作用在于控制模组 IC 运行于不同模式下，不同模式会产生不同的功耗。PMODE 寄存器地址为 0x3a，内容为 0，1，2，3 来表示 4 种运行模式。

如下表

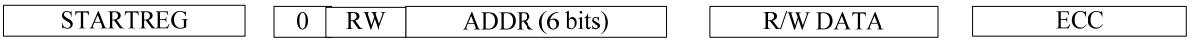
表 5 功耗模式 PMODE

.模式名称	数值	描述	实测功耗
ACTIVE	0x00	全速扫描模式，有手指按下产生触摸进入此模式，产生触摸中断/INT，让 Host 来读取触摸数据.	4mA
MONITOR	0x01	触摸监测模式，低速扫描，不能产生触摸中断/INT. 有触摸发生后自动跳转到 ACTIVE 模式，也可以有 Host 配置进入其它功耗模式.	3mA
STANDBY	0x02	一般低功耗模式.，此模式下停止扫描，但仍可以进行 I2C 通信，写 PMODE 寄存器，恢复到正常模式或进入微功耗的 HIBERNATE 模式	1mA
HIBERNATE	0x03	休眠模式. 系统主要电路停止工作，保留供电电压， 功耗在 100uA 以下。该模式下无法进行 I2C 通信，模组需要由 WAKE 信号唤醒，唤醒后会自动进入 MONITOR 模式。	100uA

注： WAKE 信号如下，低电平需要持续至少大于 5ms



读写模组 IC 的寄存器需要通过特定的协议来访问的，其具体协议如下:

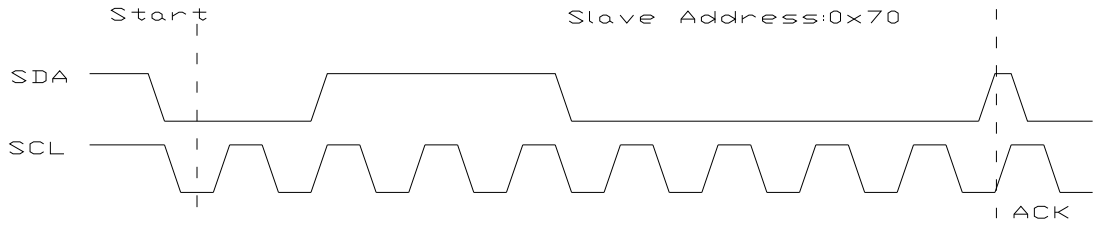


一个寄存器读写数据包包含 4 个数据 Byte. 该 4 个 Byte 以 STARTREG 开始，以 ECC 校验字节结束.

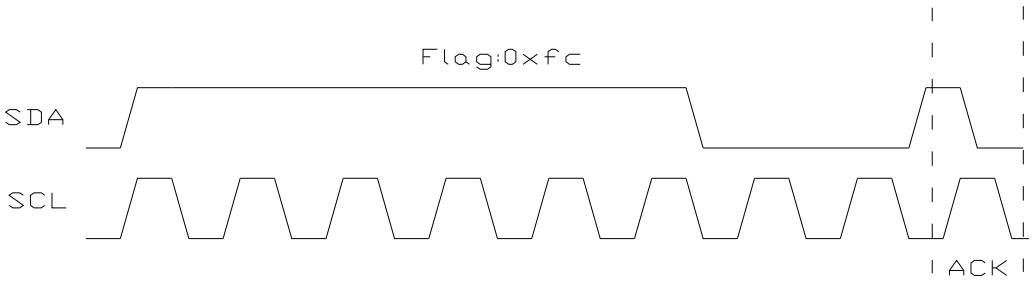
表 6 参数读写协议组成

Symbol	Value	Description
STARTREG	0xFC	参数读写数据包指令，Host 在开始一个参数的访问的时候，必须以 STARTREG 作为起始字节
RW		1 – Read 参数; 0 – Write 参数
ADDR	0 ~ 0x3F	待访问参数对应的地址，容量为 64.
R/W DATA		读/写数据
ECC		校验字节，其值等于前面 3 个 Byte 做 XOR 运算的结果

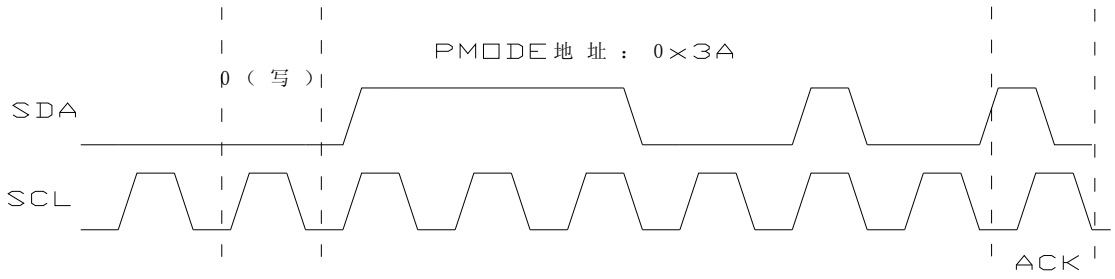
具体编程过程中可以参考如下波形：



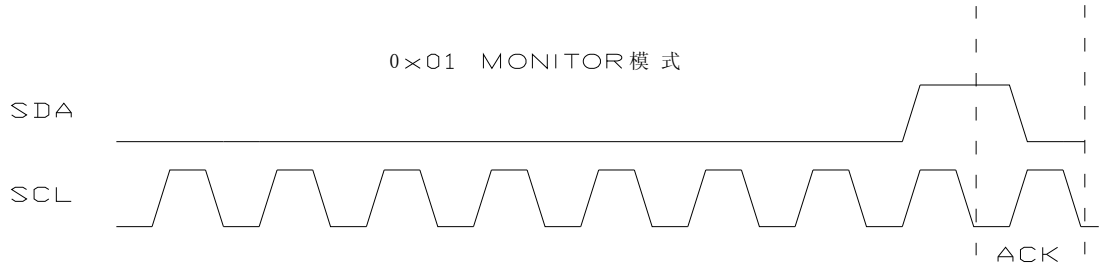
Host 发出开始信号，然后发出设备地址 0x70(写),等待 slave 应答，有 ACK 应答继续下一步；



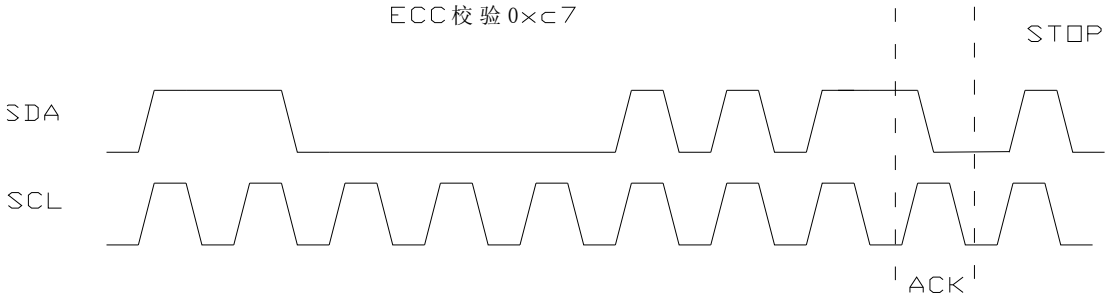
收到 ACK 应答后，Host 发送 0xfc（TPM 参数读写数据包指令, Host 在开始一个参数的访问的时候，必须以 **STARTREG**（即 0xfc）作为起始字节）,等待 ACK 应答；



接到 ACK 应答后，Host 发送 0x3A(第一位为 0，第二位为读写位 0 为写 1 为读，低六位为 PMODE 的地址)，等待 ACK 应答；



收到 ACK 应答后，设置功耗模式 0x01 为 MONITOR 模式（功耗模式见上表），等待 ACK；



收到 ACK 应答后，进行 ECC 校验（ECC 为前 3 个 byte 相异或（xor）即 0XFC xor 0x3A xor 0x01），收到 ACK 后，Host 发出结束信号。

Section 3 Q&A

1、 INT 引脚一直为低或为高，手按下也没有反应
答：请检查 INT 引脚是否加了上拉电阻
2、 读取不到数据检查波形，写 Slave Address 的时候就没有 ACK
答：和 host 端的 I2C 读写程序有关。有些平台采用的硬件 I2C，地址参数应该是 7bit 设备地址，进行通讯时会自动转成 8bit 地址，此时程序中把地址参数写成 8bit 就会出错。例如：FT5202 的设备地址是 0x70（8bit），0x38(7bit)。如果 HOST 端 I2C 地址参数为 7bit 地址，就应该用 0x38，如果用 0x70，I2C 地址就会出错，没有 ACK。
3、 读取触摸结果数据包时，写 Slave Address 有 ACK 就是读取不到数据
答：在写完 Slave Address，0xF9 后，没有 restart。

Section4 Sample Code

本样例程序采用的 GPIO 口模拟 I2C 来和模组通讯，硬件 I2C 的参考此代码进行移植。

```
#define SCK_TIME   2
#define START_TIME 4
#define ACK_TIME   3
#define BEFOR_RS_DELAY    5

void IiC_Init(void)
{

}


void SCK_LOW(void)
{

}

void SCK_HIGH(void)
{

}

void SDA_LOW(void)
{

}

void SDA_HIGH(void)
{

}

int ACK_GET(void)
{
```

```
    int data;
    rPCONF=( rPCONF&(~0x0c));
    LCD_DelayUs(ACK_TIME);
    SCK_HIGH();
    LCD_DelayUs(1);
    data=rPDATF&0x02;
    LCD_DelayUs(1);
    SCK_LOW();
    return data;
}

void ACK_SET(int ack)
{
    LCD_DelayUs(SCK_TIME);
    if(ack)
    {
        rPDATF= rPDATF|0x02;
        rPCONF=(rPCONF&(~0x0c))|0x04;
    }
    else
    {
        rPDATF= rPDATF&(~0x02);
        rPCONF=(rPCONF&(~0x0c))|0x04;
        LCD_DelayUs(2);
    }
    SCK_HIGH();
    LCD_DelayUs(3);     //12us
    SCK_LOW();
}

void Iic_Start(void)
{
    SDA_HIGH();
    SCK_HIGH();
    LCD_DelayUs(SCK_TIME);
    SDA_LOW();
    LCD_DelayUs(START_TIME);
    SCK_LOW();
}

void Iic_Stop(void)
{
    LCD_DelayUs(SCK_TIME);
    SDA_LOW();
```

```
	SCK_HIGH();
	LCD_DelayUs(SCK_TIME);
	SDA_HIGH();
}

void I2C_write_byte(U8 data)
{
	U8 mask;
	for(mask=0X80;mask!=0;mask=mask>>1)
	{
		LCD_DelayUs(SCK_TIME);
		if(data&mask)
			SDA_HIGH();
		else
			SDA_LOW();
		LCD_DelayUs(SCK_TIME - 2);
		SCK_HIGH();
		LCD_DelayUs(SCK_TIME);
		SCK_LOW();
	}
}
U8 I2C_read_byte(void)
{
	U8 mask;
	U8	data=0,val;

	rPCONF=( rPCONF&(~0x0c));
	for(mask=0X80;mask!=0;mask=mask>>1)
	{
		LCD_DelayUs(SCK_TIME*5);
		SCK_HIGH();
		val = rPDATF&0x02;
		if(val )
		{
			data = data|mask;
		}
		else
		{
			data = data&(~mask);
		}
		LCD_DelayUs(SCK_TIME*5);
		SCK_LOW();
	}
	return	data;
```

```
}

//读取 26BYTES 数据包
void Iic_ReadBytes(U32 slvAddr,U8 *data)
{
	U8 i;

	Iic_Start();

	I2C_write_byte(slvAddr);
	if(ACK_GET())
	{
		Iic_Stop();
		return;
	}

	I2C_write_byte(0xF9);
	if(ACK_GET())
	{
		Iic_Stop();
		return;
	}

	LCD_DelayUs(BEFOR_RS_DELAY);
	Iic_Start();

	I2C_write_byte(slvAddr|0x01);
	LCD_DelayUs(2);
	if(ACK_GET())
	{
		Iic_Stop();
		return;
	}

	for(i=0;i<26;i++)
	{
		data[i] = I2C_read_byte();
		if((26 -i)> 1)
			ACK_SET(0);
	}

	ACK_SET(1);
	Iic_Stop();
```

```
}

//读取寄存器
void Iic_WriteBytetoFT5202Reg(U8 regadr,U8    data )
{
        U8 i;

        Iic_Start();

        I2C_write_byte(0x70);
        if(ACK_GET())
        {
                Iic_Stop();
                return;
        }

        I2C_write_byte(0xFC);
        if(ACK_GET())
        {
                Iic_Stop();
                return;
        }

        I2C_write_byte(regadr&0x3f);
        if(ACK_GET())
        {
                Iic_Stop();
                return;
        }

        I2C_write_byte(data);
        if(ACK_GET())
        {
                Iic_Stop();
                return;
        }

        I2C_write_byte(0xfc^(regadr&0x3f)^data);
        if(ACK_GET())
        {
                Iic_Stop();
                return;
        }
```

```
}


void Iic_ReadByteFromF5202Reg(U8 regadr,U8 *data)
{

	U8 i;

	Iic_Start();

	I2C_write_byte(0x70);
	if(ACK_GET())
	{
		Iic_Stop();
		return;
	}

	I2C_write_byte(0xFC);
	if(ACK_GET())
	{
		Iic_Stop();
		return;
	}

	I2C_write_byte((regadr&0x7f)|0x40);
	if(ACK_GET())
	{
		Iic_Stop();
		return;
	}

	LCD_DelayUs(BEFOR_RS_DELAY);
	Iic_Start();
	I2C_write_byte(0x71);
	LCD_DelayUs(2);
	//读取寄存器内容
	data[0]= I2C_read_byte();
	ACK_SET(0);

	//读取 ECC 内容
	data[1]= I2C_read_byte();
	ACK_SET(0);
```

```
}

void FT5202_IIC_TEST(void)
{
	U8 PackageData[26];
	U8 RegData[2];
	//读取 26bytes 数据包
	Iic_ReadBytes(0x70,PackageData);

	//向 0x3A 寄存器写 1
	Iic_WriteBytetoFT5202Reg(0x3a,1);

	//读取 0x3A 寄存器的内容

	Iic_ReadByteFromF5202Reg(0x3a,RegData);
}
```