应用笔记: AZD062
IQ Switch® - ProxSense® Series
IQS253 通信接口指南

目录

1 引言

这篇应用笔记通过通俗易懂的流程图及源程序代码，来指导读者如何使用 I2C(400K)通信协议实现 MCU 和 IQS253 通信。

下图 1 概述了此文档的主要内容，如果需要可以提供完整的代码。

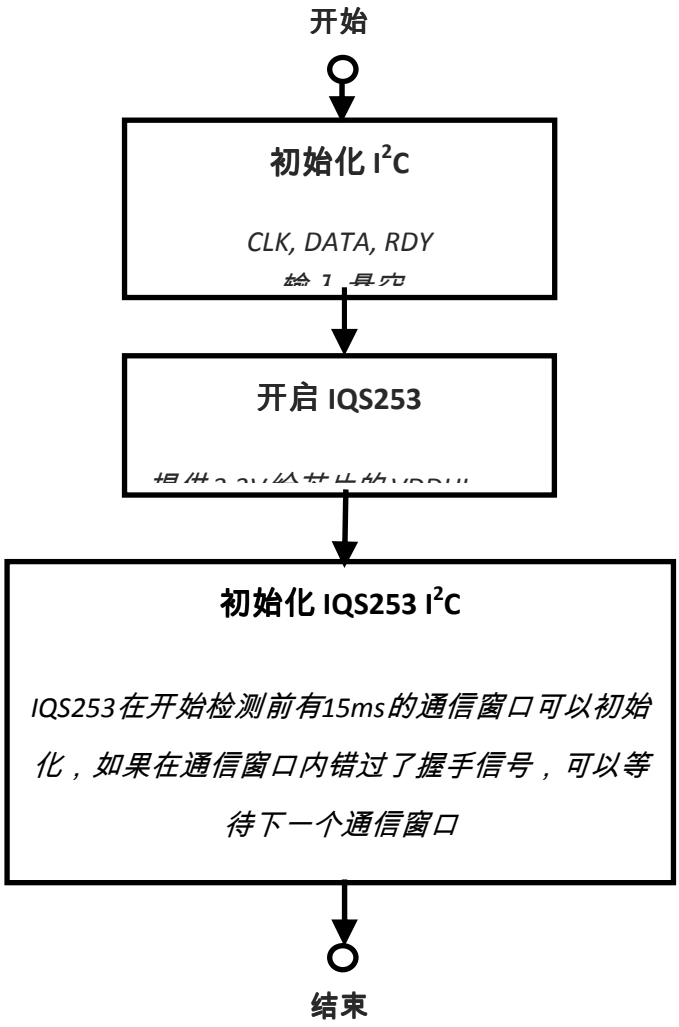

图 1: 初始化 I²C 流程图

2 通信协议

IQS253使用双向2线数据传输协议，兼容 I2C™ 串行协议。IQS253有一个可选的准备脚（RDY）来指示是否进入通信窗口。只能靠监测RDY或者查询ACK来确定是否能和芯片通信。IQS253在总线上是一个从机设备，总线的入口、START和STOP都有主机的SCL控制，SCL和SDA都是开路漏极，需要外部上拉4.7K电阻来得到高电平，RDY脚也是开路漏极功能，需要上拉100K电阻。

在通信窗口，RDY一直保持低电平（工程样品是高电平）2ms。如果主机在这个时间段没有初始化芯片，芯片会跳出通信窗口进入变换模式。在通信窗口地址指针默认为DEFAULT_ADDR寄存器的值。用这个方法用户可以直接读而不需要设置地址指针。在通信时RDY会保持低电平。

下图（图2）描述了本节介绍的通信协议数据传输顺序。

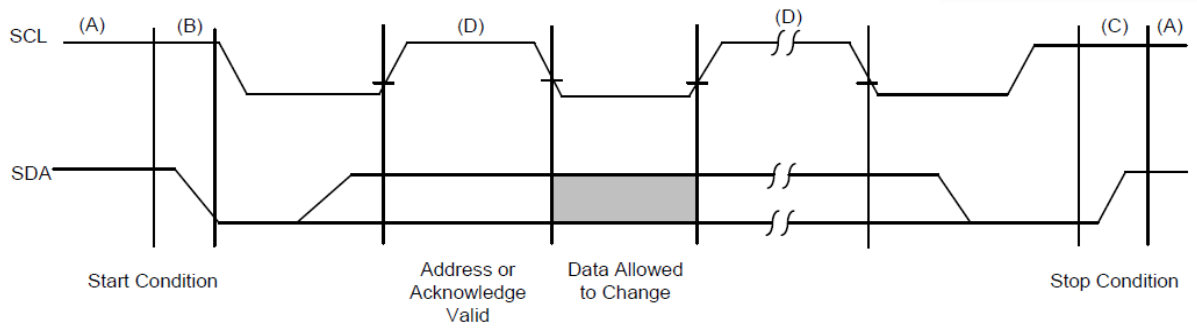

图 2: 串行总线的数据传输顺序

2.1 总线特性

总线协议定义：

- 总线空闲时才能开始数据传输；
- 数据传输期间，当时钟线为高时，数据线必须稳定，如果改变状态，则看作是START或STOP信号。

下面条件总线定义为：（参考图2)

总线空闲(A) - SCL 和 SDA都是高电平；

开始信号(B) – 当SCL为高时，SDA由高转为低，所有串行通信都在START之后。

停止信号(C) -当SCL为高时，SDA由低转为高，所有串行通信都在STOP之前。注意：当送完STOP后，芯片将跳出通信窗口进入转换模式。

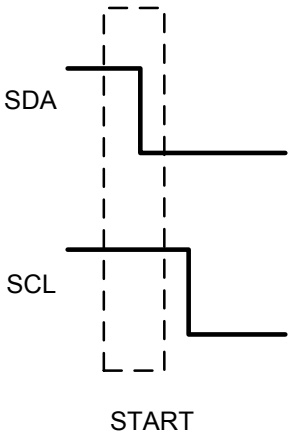

图 3: 开始信号

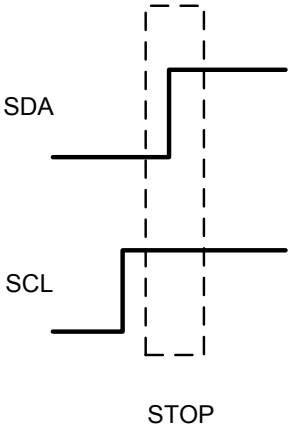

图 4: 停止信号

表 1. 开始信号

```
void i2c_start(void)
{
        I2C_CLK_IN_FLOAT;                       // Set I2C CLK pin as input and floating
        while(!I2C_CLK_IN);                     // Wait for I2C CLK line low (clock stretching)
        wait(2);                                // Wait two I2C clock cycles
        I2C_DATA_IN_FLOAT;                      // Set I2C DATA pin as input and floating
        wait(2);                                // Wait two I2C clock cycles
        while(I2C_RDY_IN);                      // Wait while ready high (This could take long in event mode)
                                                // RDY checks could also be done before generating a start condition
        wait(100);                              // Delay 100 clock cycles
        I2C_DATA_OUT_PP_LOW;                    // Set I2C DATA pin as output (push-pull) and floating
        wait(2);                                // Wait two I2C clock cycles
        I2C_CLK_OUT_PP_LOW;                     // Set I2C CLK pin as output (push-pull) and floating
        wait(2);                                // Wait two I2C clock cycles
}
```

表 2. 停止信号

```
void i2c_stop(void)
{
        I2C_CLK_OUT_PP_LOW;                     // Set I2C CLK pin as output (push-pull) and floating
```

```
        wait(2);                                    // Wait two I2C clock cycles
        I2C_DATA_OUT_PP_LOW;                        // Set I2C DATA pin as output (push-pull) and floating
        wait(2);                                    // Wait two I2C clock cycles
        I2C_CLK_IN_FLOAT;                           // Set I2C CLK pin as input and floating
        while(!I2C_CLK_IN);                         // Wait for I2C CLK line low (clock stretching)
        wait(2);                                    // Wait two I2C clock cycles
        I2C_DATA_IN_FLOAT;                          // Set I2C DATA pin as input and floating
        wait(2);                                    // Wait two I2C clock cycles
}
```

- **有效数据(D)** –当SCL为高时，SDA保持稳定则为有效数据。SDA只有在SCL为低时才能改变电平。每位数据是一个时钟脉冲，从START开始到STOP结束。

- **Ack信号** – 从机在接收完一字节数据后需产生一个握手信号，主机必须发送第9个时钟脉冲来检测握手信号。此时从机会把SDA拉为低电平。注意：不在通信窗口模式IQS253不产生任何握手信号。

表 3. 检测握手信号

```
/*
        Check for acknowledge
*/
unsigned char i2c_ack_check(void)
{
        I2C_DATA_IN_FLOAT;                          // Set I2C DATA pin as input and floating
        wait(2);                                    // Wait two I2C clock cycles
        I2C_CLK_IN_FLOAT;                           // Set I2C CLK pin as input and floating
        wait(2);                                    // Wait two I2C clock cycles
        while(!I2C_CLK_IN);                         // Wait for I2C CLK line low (clock stretching)
        wait(2);                                    // Wait two I2C clock cycles
        if (I2C_DATA_IN)  return 1;                 // Return 1 if no acknowledge received
        else return 0;                              // Return 0 if acknowledge received
}
```

2.2 控制字节和设备地址

控制字节包含7位设备地址和1位读写位，结构参考图5。

I2C设备在控制字节中有7位是从机地址，如图5，软件用接收到的地址和设备地址来确认，预配的设备地址请参考规格书，为

避免地址冲突，IQS253有两位地址可以更改，故可同时在一条总线上接4个IQS253。

如果多个IQS253在总线上，各个芯片需要预设从机地址。

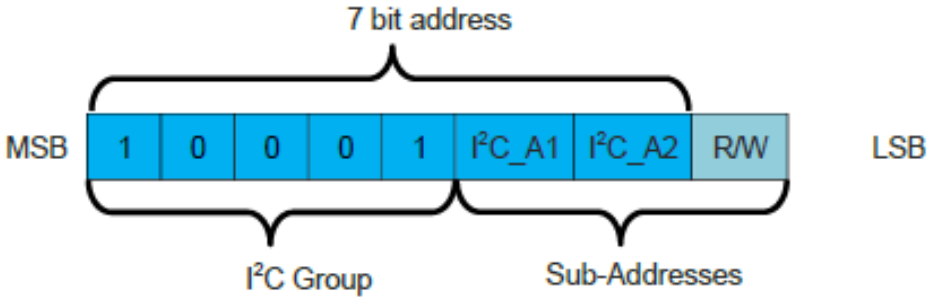

图 5: 控制字节格式

3 IQS253 通信窗口

只有两个方法进入通信窗口，用RDY或者检测ACK信号，但需要注意，如果用检测ACK信号的方法，错过了第一个通信窗口则无法再进入通信窗口。

IQS253上电后的首次通信比较特别，只有在这个通信窗口才能自动校准。

3.1 RDY线的使用

当未用查询ACK的方式时，可用简单的等待RDY拉低检测是否进入通信窗口的握手方式，把RDY设为输出低电平，10ms后再设为输入模式，芯片如果握手成功，会把RDY拉低，知道获得握手信号。

3.2 查询ACK信号

如果主机不够脚位接RDY脚，就可以用查询ACK的方式来确定设备是否进入通信窗口。在转换的时候设备不发出ACK信号，因此可以用来判断是否转换完成进入通信窗口了，一旦主机发送了STOP命令，则设备又进入下一个转换周期。如果设备进入通信窗口，都可以执行查询ACK命令，RDY脚在通信的时候接没接主机功能都一样。

主机发送一个START信号及控制字节来执行查询ACK信号，如果设备忙则无ACK信号返回，如果设备完成了转换，则会产生ACK信号，主机可以进行读或写操作。概括步骤如下：

1. MCU送START 信号
2. MCU 送控制字节
3. MCU 检测是否接收到ACK信号.
4. 如果没有重复1步骤
5. MCU 与IQS253进行读写操作

注意设置查询次数以免无信号时等待时间

图 6: 启动握手模式流程图

表 4. 启动握手模式

```
/*
        Invokes Communication Window
*/
void i2c_event_mode_handshake(void)
{
        unsigned int i;                 // Counter for stuck check

        do
        {
            I2C_RDY_OUT_PP_LOW;
            delay_ms(10);
            I2C_RDY_IN_FLOAT;
            i2c_wait();
            delay_ms(1);
            i++;
        }while(!I2C_RDY_IN || i == 15) ;// Test for Comms. Window
}
```

过长，特别是在检测模式下，需要时间给主机通信，记住，即使没用RDY脚，也要加上拉电阻。

3.3 初始化窗口

初始化通信窗口或叫上电设置窗口给用户一个在转换之前输入设置的机会。

除了在自容和互容技术之间切换只能在设置窗口设置外，大部分设置可以更新，这可以让用户控制设备在特定时间应用特定设置，VDDHI也可以控制。图7为上电通信窗口的发生时序图。

在VDDHI为高 (3.3V)后T_{START_UP}（大约15ms)RDY线会拉低进入设置窗口，然后IC寻址，初始化设置（章节5），再送STOP命令，如果设置窗口没有在t_{COMMS}(22ms)完成，RDY线会再拉高，IC开始转换及留在这个模式。

IQS253可以预设置为自容或互容模式，这个功能不能通过I^2C命令更改（除非应用需要）。

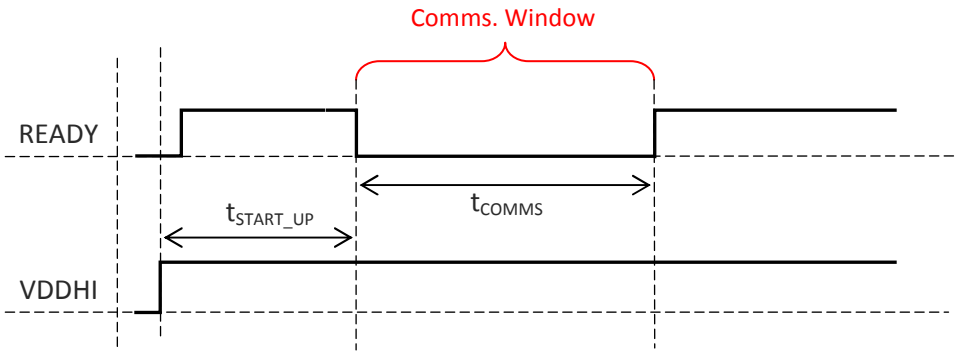

图 7: 初始化窗口时序图

4 IQS253读写

进入通信窗口后，即可进行读写操作，读写操作格式参考表5-8，当主机完成读写操作，可以再开始或者结束。再开始可以继续读写操作，停止跳出通信窗口进入转换模式。

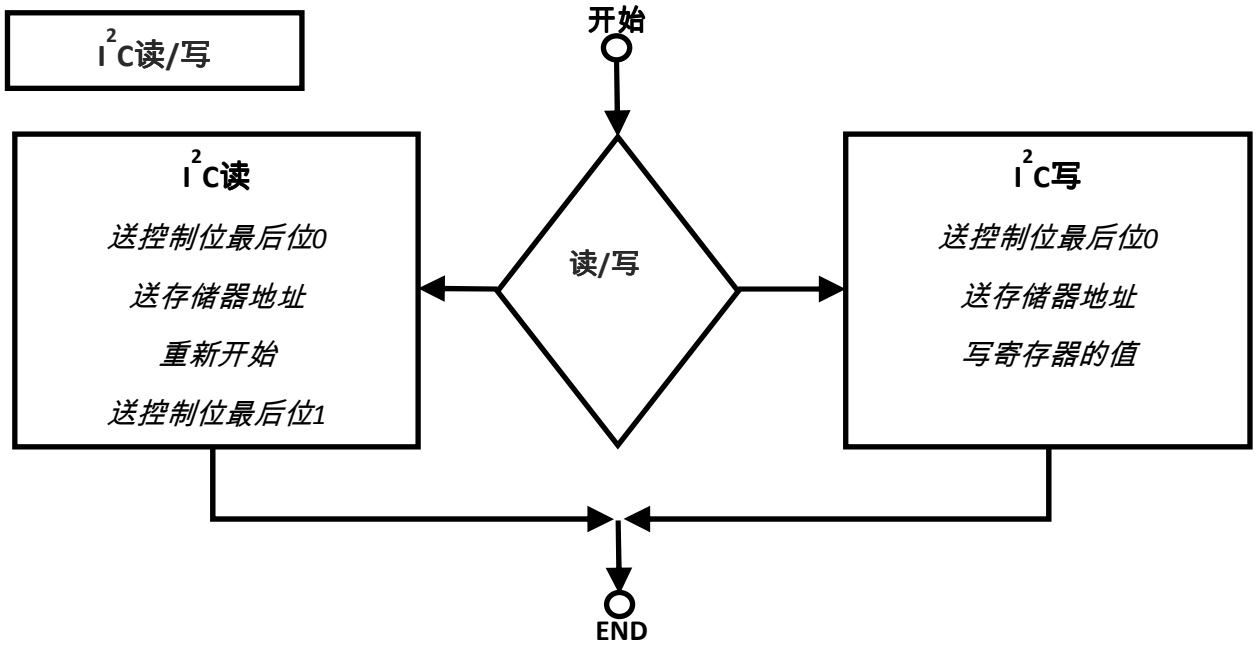

图 8: 读/写操作流程图

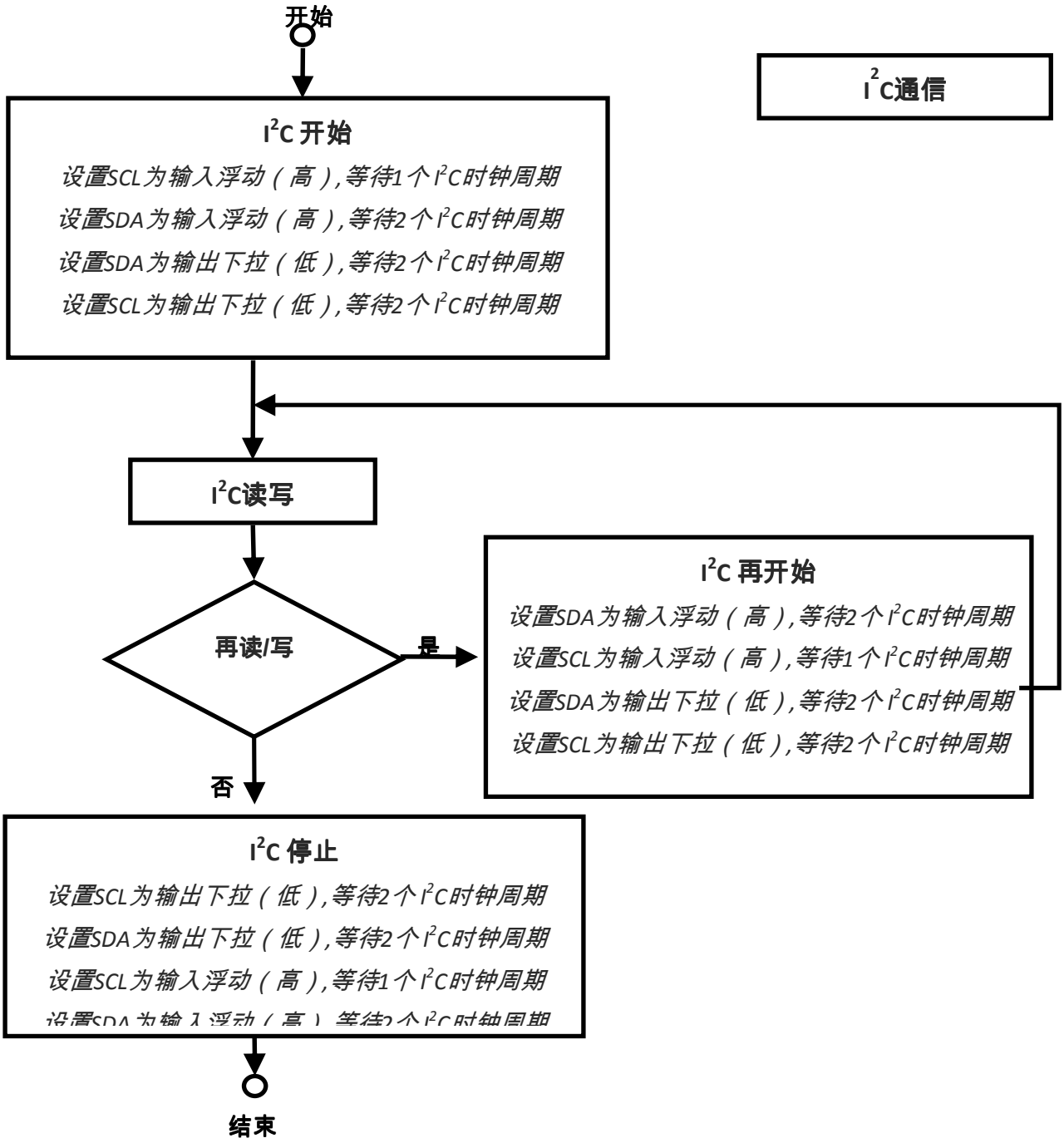

图 8: 通信流程图

4.1 写操作

R/W位写0为写操作，写完控制字节，接着是地址，再是数据，地址只需要写一次，后面都是数据，写一次地址后，可以写多块数据，连续写时，地址指针会自动增加，如果增加到最大值时，就不能再写了。

注意，指针不会自动从LTA块跳到设置块。图10描述了写数据的过程。

DATA WRITE

Start	Control Byte		Word Address(n)		Data n		Data n+1		Stop
S		ACK		ACK		ACK		ACK	S

图 9: I²C写数据

表 5. 写操作

```
/*
        Send a given byte via I2C

        @param send_byte      – byte that has to be send via I2C
        @return unsigned char – Boolean value that signifies a acknowledge returned or not
*/
unsigned char i2c_send_byte (unsigned char send_byte)
{
        unsigned char ack;                              // Variable to store acknowledge boolean in
        unsigned char i;                                // Counter variable to count off bits send

        for ( i = 0; i < 8; i++ )                       //Send 8 bits to I2C Bus
        {
                wait(1);                                // Wait one I2C clock cycle
                if ( send_byte & 0x08 )                 // If most significant bit equal to 1
                {
                        I2C_DATA_IN_FLOAT;              // Set DATA pin HIGH to clock out a 1
                }
                else
                {
                        I2C_DATA_OUT_PP_LOW;            // Set DATA pin LOW to clock out a 0
                }

                send_byte <<= 1;                        // Shift 'send_byte' left with one bit in order to send the next bit

                wait(1);                                // Wait one I2C clock cycle
                I2C_CLK_IN_FLOAT;                       // Set CLK pin HIGH
                wait(1);                                // Wait one I2C clock cycle
                while(!I2C_CLK_IN);                     // Wait for I2C CLK pin low (clock stretching)
                wait(1);                                // Wait one I2C clock cycle
                I2C_CLK_OUT_PP_LOW;                     // Set CLK pin LOW
        }
```

```
        ack  = i2c_ack_check ();                    // Check for an acknowledge bit
        I2C_CLK_OUT_PP_LOW;                         // Set CLK pin LOW
        wait(1);                                    // Wait one I2C clock cycle

        return ack;                                 // Return acknowledge boolean
}
```

表 6. 寄存器写操作

```
/*
        Send a given byte via I2C to specific register

        Note: This function is called once already in comms window, thus start() or repeat_start() called prior calling i2c_write_register
                After the function call the slave will still be in a comms window, waiting for either a stop or a repeat start

        @param control_byte   – I2C control byte
        @param mem_address – Address or register that has to be written to
        @param mem_value     – Byte that has to written to register
        @return unsigned char – Boolean value that signifies a acknowledge returned or not
*/
unsigned char i2c_write_register(unsigned char device_address, unsigned char mem_address, unsigned char mem_value)
{
        unsigned char ack;                                      // Variable to store acknowledge boolean in
        unsigned char polling_attempt = 0;                      // Counter for polling attempts

        ack = i2c_send_byte(device_address);                    // Send device address to I2C Bus

        #ifdef POLLING                                          // Include code segment if polling enabled
        while ( ack && (polling_attempt < POLLING_ATTEMPTS))
        {
                wait(2);
                i2c_start();
                ack = i2c_send_byte (device_address);           // Send control byte to I2C Bus
                polling_attempt++;                              // Increase polling attempts counter
        }
        #endif

        if (!ack)
        {
                ack = i2c_send_byte (mem_address);
                ack = i2c_send_byte (mem_value);
                i2c_wait();
        }
        return ack;
}
```

4.2 读操作

控制字节的R/W位为1时，为读操作，从当前地址指针读数据（图12），读数据时地址指针可以自动增加（到下一个可用地址），如果随机写入个地址则可以随机读数据（图11），步骤为写地址指针，再开始，读数据。读模式下，主机发出ACK信号。如果读完一个数据主机发送了ACK信号从机再送下一个数据（CLK要延时），然后从机等待主机的再开始或结束命令。

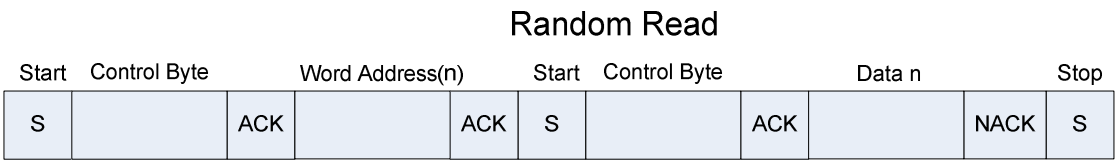

图 10: I²C随机读

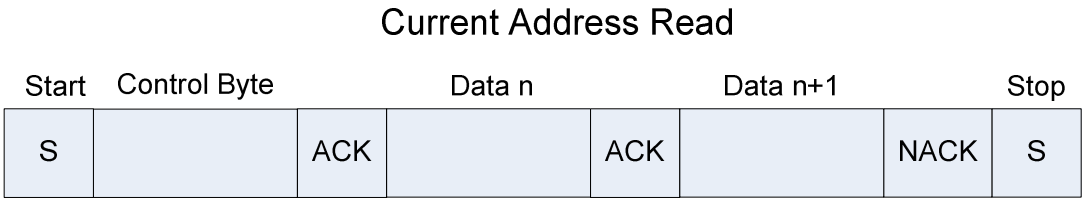

图 11: I²C 当前地址读

表 7. 读操作

```
/*
        Read byte via I²C
        @param  ack if 1 send acknowledge bit else don't send acknowledge bit.
        @return byte received
*/
unsigned char i2c_read_byte(unsigned char ack)
{
        unsigned char i, receive_byte = 0;

         I2C_DATA_IN_FLOAT;                                  // Set I2C DATA pin as input and floating
        wait(2);                                             // Wait two I2C clock cycles

        for (i = 8; i > 0; i--)                              // Loop and read 8 bits from I2C DATA pin
        {
                wait(2);                                     // Wait two I2C clock cycles
                 I2C_CLK_IN_FLOAT;                           // Set I2C CLK pin as input and floating
                while(!I2C_CLK_IN);                          // Wait for I2C CLK line low (clock stretching)

                if (I2C_DATA_IN)  receive_byte  |= (1 << (i - 1));   //Read data from I2C DATA pin

                wait(1);                                     // Wait one I2C clock cycle
                I2C_CLK_OUT_PP_LOW;                          // Set I2C CLK pin as output (push-pull) and floating
        }

        wait(1);                                             // Wait one I2C clock cycle

        if (ack == 0)  I2C_DATA_IN_FLOAT;                    //
        else  I2C_DATA_OUT_PP_LOW;                           //Send acknowledge if required

        wait(2);                                             // Wait two I2C clock cycles
        I2C_CLK_IN_FLOAT;                                    // Set I2C CLK pin as input and floating
        while(!I2C_CLK_IN);                                  // Wait for I2C CLK pin low (clock stretching)
        wait(2);                                             // Wait two I2C clock cycles
        I2C_CLK_OUT_PP_LOW;                                  // Set I2C CLK pin as output (push-pull) and floating
        wait(2);                                             // Wait two I2C clock cycles
```

```
	I2C_DATA_IN_FLOAT;                                          // Set I2C DATA pin as input and floating

	return  receive_byte;
}
```

表 8. 读I2C 数据

```
/*
	Read byte from specified address via I2C

	Note: This function is called once already in comms window, thus start() or repeat_start() called prior calling i2c_write_register
		After the function call the slave will still be in a comms window, waiting for either a stop or a repeat start

	@param mem_address
	@param
	@return acknowledge status
*/
unsigned char i2c_read_register(unsigned char device_address, unsigned char mem_address, unsigned char *data_read)
{
	unsigned char temp = 0;
	unsigned char ack = 0;
	unsigned char control_byte = (device_address << 1);
	unsigned char polling_attempt = 0;                          //Counter for polling attempts

	ack = i2c_send_byte (control_byte);                         // Send device address

	#ifdef POLLING                                              //If polling enabled
	while ( ack && (polling_attempt  <  POLLING_ATTEMPTS) )
	{
		wait(2);
		I2CStart();
		ack = 2c_send_byte (control_byte);
		polling_attempt++;                                  //Increase polling attempts counter
	}
	#endif

	if (ack == 0)
	{
		i2c_send_byte (mem_address);                        // Write mem_address to internal pointer
		i2c_repeat_start();
		control_byte = (device_address <<  1)  |  0x01;
		ack = i2c_send_byte (control_byte);                 // Send controlbyte with r/w = 1
		temp =  i2c_read_byte (1);              //Read byte and don't acknowledge to indicate a repeat start or stop will follow
		(*data_read) = temp;
	}

	return ack;
}
```

5 IQS253调整设置

参考IQS253芯片规格书中的特殊寄存器的地址。

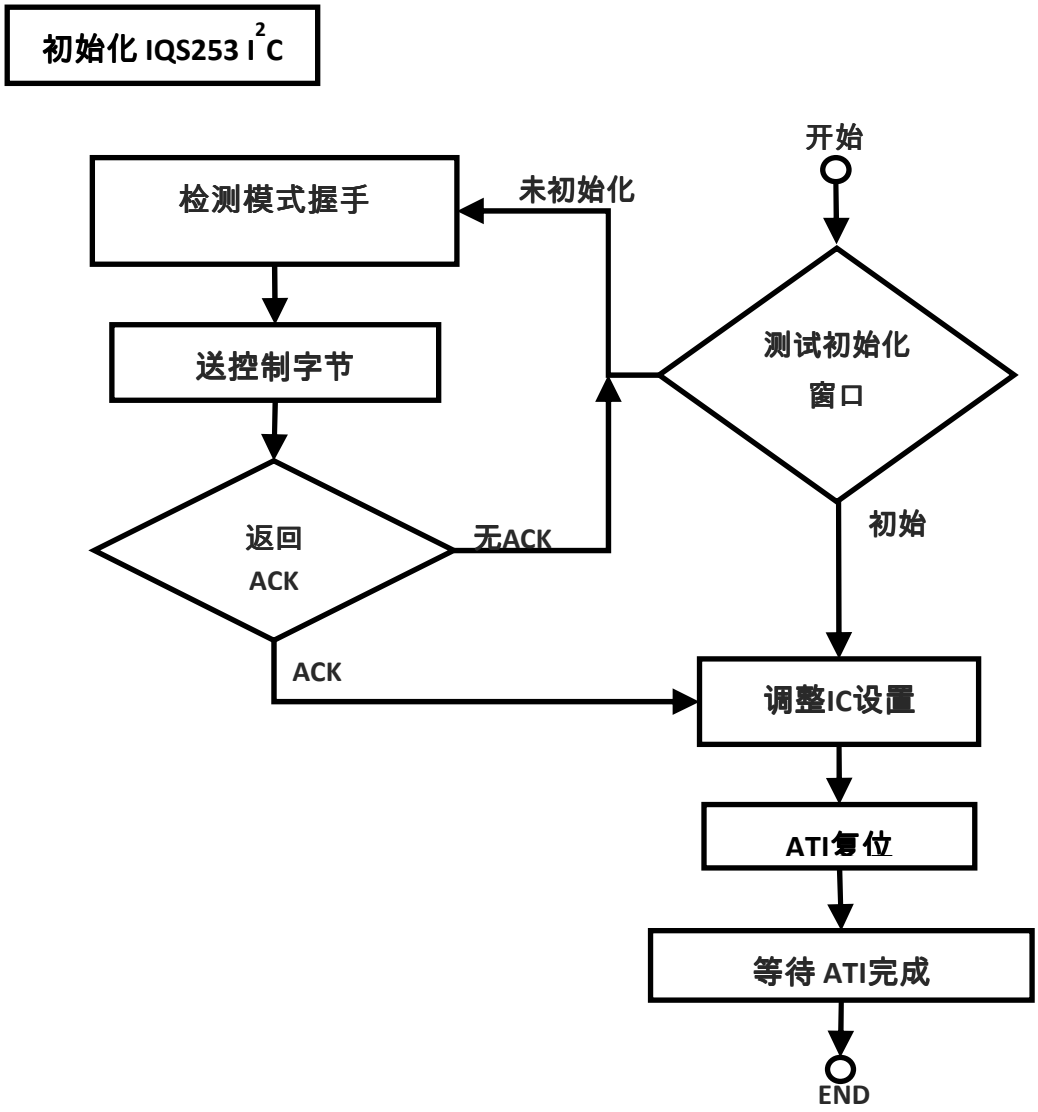

图 12: 初始化 I2C流程图

Listing 9. Adjusting Setting for IQS253

```
/*
        Initialize IQS253
*/
void Init_IQS253(void)
{
        unsigned char result;

        do {
                i2c_event_mode_handshake();
                i2c_start();
                result = i2c_write_register(PROX_SETTINGS2, 0x14);              //set to streaming mode for setup
                i2c_stop();
        } while(result);

        do {
                i2c_start();
                i2c_write_register(CHAN_ACTIVE, 0x07) ;                         // Enable Channels (0-2)
                i2c_repeat_start();
                i2c_write_register(PROX_SETTINGS0, 0x40) ;                      // ATI OFF, ATI partial OFF
                i2c_repeat_start();
                i2c_write_register(PROX_SETTINGS2, 0x40) ;                      // WDT Off
                i2c_stop();

                /*Set Touch Thresholds*/
                i2c_start();
                result |= i2c_write_register(CH0_TTH, 0x04);
                i2c_repeat_start();
                result |= i2c_write_register(CH1_TTH, 0x20);
                i2c_repeat_start();
                result |= i2c_write_register(CH2_TTH, 0x20);
                /* Set Proximity Thresholds */
                i2c_repeat_start();
                result |= i2c_write_register(CH0_PTH, 0x04);
                i2c_repeat_start();
                result |= i2c_write_register(CH1_PTH, 0x04);
                i2c_repeat_start();
                result |= i2c_write_register(CH2_PTH, 0x04);
                i2c_repeat_start();
                i2c_write_register(TARGET, 0x40);                               //Set Target Current Count = 512
                i2c_repeat_start();
                result |= i2c_write_register(CHAN_ENABLE, 0x07);                //Disable distributed PROX CH0
                i2c_stop();
        }while (result);

        delay_ms(200);

        do {                                                                    //read ATI busy bit
                delay_ms(10);
                i2c_start();
                result = i2c_read_register(STATUS, 1);
                i2c_stop();
        } while ( (result & 0x04) != 0 );

}
```

The following patents relate to the device or usage of the device: US 6,249,089 B1, US 6,621,225 B2, US 6,650,066 B2, US 6,952,084 B2, US 6,984,900 B1, US 7,084,526 B2, US 7,084,531 B2, US 7,119,459 B2, US 7,265,494 B2, US 7,291,940 B2, US 7,329,970 B2, US 7,336,037 B2, US 7,443,101 B2, US 7,466,040 B2, US 7,498,749 B2, US 7,528,508 B2, US 7,755,219 B2, US 7,772,781, US 7,781,980 B2, US 7,915,765 B2, EP 1 120 018 B1, EP 1 206 168 B1, EP 1 308 913 B1, EP 1 530 178 B1, ZL 99 8 14357.X, AUS 761094

WWW.AZOTEQ.COM

ProxSenseSupport@azoteq.com