

一.api接口, clock_stretch意见

1.时钟延时机制 (clock_stretch)

从设备在传输信号中, 应答位clk在数据未处理完成时会拉低, 主机需要等待, 从机释放时钟后, 主机才可以继续发送命令, 大部分硬件i2c都是支持这个机制, io模拟的i2c的普遍没有, 最为简单的方式, 是在关键处理数据的时间点, 主机主动延时clk动作, 来解决这一问题。下面是一种修改方式以供参考

```
int write_data(u8 address,u8 *data0,u16 len)          // 向地址address中写入一个字节数据
{
    u8 i = 0;

    IIC_Start();
    Write_Byte(SlaveAddr | 0);
    respons();
++    delay(1000);//delay 200us
    Write_Byte(address);
    respons();
    i++;
    while(len)
    {
        if(len == 0)
        {
            break;
        }
        else
        {
            Write_Byte(*(data0++));
            respons();
        }
        len--;
        i++;
++        if(i == 8)
++        {
++            i = 0;
++            delay(1000);
++        }
    }

    IIC_Stop();
    IIC_Speed();
    return 0;
}
```

```
int Read_data(u8 address,u8 * data1,u16 len)          // 向地址address中读
出一个字节数据
{
    u8 i;
    i = 0;

    IIC_Start();
    Write_Byte(SlaveAddr | 0);
    respons();
```

```

++ delay(1000);
Write_Byte(address);
respons();
IIC_Stop();
++ delay(100);

IIC_Start();
Write_Byte(SlaveAddr | 1);
respons();
++ delay(1000);
while(len)
{
    len--;
    if(len == 0)
    {
        *data1 = Read_Byte(0);
        break;
    }
    else
    {
        *data1 = Read_Byte(1);
    }
    data1++;
    i++;
++    if(i == 8)
++    {
++        i = 0;
++        delay(1000);
++    }

    }
    IIC_Stop();
    IIC_Speed();
}

```

二.坐标解析

1.获取最大手指数（Get Maxtouch）

```

static int get_max_touch(void)
{
    u8 buf[1];
    int ret = 0;
    ret = STX_RD_Reg(0x3F,buf,1); // Read 1 byte maxtouch from Reg.
    maxtouch = (u32)buf[0]; // Return maxtouch.
    return ret;
}

```

2.获取坐标 (Read XY Coordinates)

```
//get_fw_version(&maxtouch); //初始化时获得最大手指数

int STX_thread()
{
    u16 x[10] = {};
    u16 y[10] = {};
    u16 state[10] = {};
    u8 i;
    u8 buf[42];
    if (is_power_down == 0)
    {
#ifdef SITRONIX_GESTURE
        ... ..
#endif
        STX_RD_Reg(0x12, buf, 40);
        for (i = 0; i < max_touches; i++)
        {
            if (buf[i * 4] & 0x80)
            {
                x[i] = (u16)((buf[i * 4] & 0x70) << 4 | buf[i * 4 + 1]);
                y[i] = (u16)((buf[i * 4] & 0x07) << 8 | buf[i * 4 + 2]);
                state[i] = 1;
                printf(" touch down x=%d,Y=%d\r\n", x, y);
            }
            else
            {
                x[i] = 0;
                y[i] = 0;
                state[i] = 0;
                printf(" touch up  x=%d,Y=%d\r\n", x, y);
            }
        }
        STX_report_touch_one_sync(x, y, state, maxtouch);
    }
#ifdef ST_SMART_WAKE_UP
        ... ..
#endif
}
}
```

三.ESD防护

1.守护进程逻辑 (Guardian process logic)

当IC处于异常状态时需要拉resetpin来重置IC,

- 1.该进程进入延时预设为0.3s
- 2.当发生触摸事件时，跳过本次查询

3.发生升级动作事，跳过本次查询

4.休眠模式下。跳过本次查询

程式参考

①从0x01位置读8个byte，发生i2c通讯错误时，计次一次，进入下一次守护进程等待，计数超过三次，拉reset pin重置TP

```
result = 1;
ret = stx_i2c_read(stx_gpts.client, buffer, 8, 0x01);
if (ret < 0)
{
    STX_ERROR("I2C read 0x01 Error ");
    goto exit_i2c_invalid;
}
```

②判断buf[0]低8位，即寄存器0x01的低8位，如果在bootcode模式，记错一次

```
if ((buffer[0] & 0x0F) == 0x06)
{
    result = -1;
    STX_ERROR("%s IC status is buf0 = 0x%x ", __func__, buffer[0]);
    goto exit_i2c_invalid;
}
```

③判断TP进入Idle状态==0xFE跳过本次

```
else if ((buffer[0] != 0xFF) && (buffer[2] == 0xFE))
{
    // kthread_stop(SitronixMonitorThread);
    STX_ERROR("%s:", "Sitronix_ts_monitoring Idle time 0xFE");
    SitronixMonitorThread = NULL;
    result = 0;
    return 0;
}
```

④buff[6] buff[7],即寄存器0x07 作为高8位，0x08作为低八位，两次ESD功能中没有变化，记错一次

```
else
{
    count = buffer[6] * 0x100 + buffer[7];
    /*如果数值一样，记录错误一次*/
    if (count != pre_count)
    {
        pre_count = count;
        result = 0;
    }else{
        STX_ERROR("sensing ERROR ");
        goto exit_i2c_invalid;
    }
    result = get_dist_value();
}
```

⑤记错次数超过3次，或者dist值获取(可不加此内容)失败，拉reset pin重启TP

```
exit_i2c_invalid:
    if ((result == 1) || (result == -1))
    {
        i2cErrorCount++;
        if ((2 <= i2cErrorCount) || (result == -1))
        {
            STX_ERROR("I2C abnormal, reset it!");
            st_reset_ic();
            i2cErrorCount = 0;
        }
    }
    else
    {
        i2cErrorCount = 0;
    }
}
```

四.手势信息

1.普通模式下读手势

手势值宏定义

```
//坐标中断产生的同时，上报手势产生。通过寄存器读对应值
#define SITRONIX_GESTURE

#ifdef SITRONIX_GESTURE
#define G_NO 0x0
#define G_ZOOM_IN 0x2
#define G_ZOOM_OUT 0x3
#define G_L_2_R 0x4
#define G_R_2_L 0x5
#define G_T_2_D 0x6
#define G_D_2_T 0x7
#define G_PALM 0x8
#define G_SINGLE_TAB 0x9
#endif
```

参考逻辑

下例为正常报点时，读到大手掌覆盖手势，进入休眠模式

```
int STX_thread()
{
    u16 x[10] = {};
    u16 y[10] = {};
    u16 state[10] = {};
    u8 i;
    u8 buf[42];
    if (is_power_down == 0)
    {
#ifdef SITRONIX_GESTURE
```

```

    STX_RD_Reg(0x10, buf, 1);
    switch(buf[0])
    {
        case G_PALM:    // 符合大手掌手势
            system_suspend();    // 调用STX_suspend, 关闭背光, 等一系列休眠操作
            return 0;
            //...
        default:
            break;
    }
#endif

    ... ..

}

}

```

2.智能唤醒模式下读手势

手势值宏定义

//进入一种低功耗模式，这个模式下不再上报坐标，当符合某种手势时，拉中断，上报对应数值，驱动可以以此作为唤醒信号

```

#define ST_SMART_WAKE_UP

#ifdef ST_SMART_WAKE_UP
#define SWK_NO 0x0
#define GESTURE_LEFT 0xB4
#define GESTURE_RIGHT 0xB0
#define GESTURE_UP 0xBC
#define GESTURE_DOWN 0xB8
#define GESTURE_DOUBLECLICK 0xC0
#define GESTURE_SINGLECLICK 0xC8
#define GESTURE_O 0x6F
#define GESTURE_W 0x77
#define GESTURE_M 0x6D
#define GESTURE_E 0x65
#define GESTURE_S 0x73
#define GESTURE_V 0x76
#define GESTURE_Z 0x7A
#define GESTURE_C 0x63
#endif

```

参考逻辑

```

int STX_thread()
{
    u16 x[10] = {};
    u16 y[10] = {};
    u16 state[10] = {};
}

```

```

u8 i;
u8 buf[42];
if (is_power_down == 0)
{
    ... ..

#ifdef ST_SMART_WAKE_UP
    else
    {
        STX_RD_Reg(0xF2, buf, 1);
        switch(buf[0])
        {
            case GESTURE_DOUBLECLICK: // 符合双击手势
                system_resume(); // 调用STX_resume，点亮背光，等一系列唤醒操作
                break;
            case GESTURE_SINGLECLICK: // 符合单击手势
                system_resume();
                break;
            //...
            default:
                break;
        }
    }
}
#endif
}

```

进入智能唤醒模式需要添加的指令

```

/*触摸IC进入休眠模式
void STX_suspend()
{
    u8 ret = 0;
    u8 buffer[2] = {0};
#ifdef ST_SMART_WAKE_UP
    ret = STX_RD_Reg(0xF1, buffer, 1);
    if (ret == 1)
    {
        buffer[1] = buffer[0] | 0x80;
        buffer[0] = 0xF1;
        STX_WR_Reg_direct(buffer, 2);

        delay_ms(50);
    }
#endif
    buffer[0] = 0x2;
    STX_WR_Reg(0x2, buffer, 1);
    is_power_down = 1;
}

```

五.升级固件

1.升级注意事项 (Upgrading precautions)

1.客户需帮忙实现注释标记"HFST Platform specific"划定的函数接口,主要是一些平台相关的函数, 需要全部实现。

如: stx_i2c_read_direct、stx_i2c_write_bytes、stx_i2c_read_bytes等I2C通讯接口及st_msleep延时函数(单位: ms)

```
/******HFST Platform specific******/
#define STX_ERROR //printf
#define STX_INFO //printf
#define STX_DEBUG //printf

int st_i2c_upg_read_direct(unsigned char *rxbuf, int len)
{
    // IF I2C addr = 0x55
    // I2C start -> 0xAB -> read with ACK -> ...read with NACK -> I2C stop
    return STX_RD_Reg_direct(rxbuf, len); //if success, return len
}

int st_i2c_upg_write_bytes(unsigned char *txbuf, int len)
{
    // IF I2C addr = 0x55
    // I2C start -> 0xAA -> RegAddr ->value0 ->value1 ->value2 ...I2C stop
    return STX_WR_Reg_direct(addr, rxbuf, len); //if success, return len
}

int st_i2c_upg_read_bytes(unsigned char addr, unsigned char *rxbuf, int len)
{
    // IF I2C addr = 0x55
    // I2C start -> 0xAA -> RegAddr ->I2C stop -> delay -> I2C start -> 0xAB ->
    read with ACK -> ...read with NACK -> I2C stop
    return STX_RD_Reg(addr, rxbuf, len); //if success, return len
}

unsigned char get_client_i2c_addr(void)
{
    //Obtain the i2c address for normal communication of the entire machine
    return 0x55;
}

void stx_i2c_addr_set(unsigned char addr)
{
    // Host I2C addr = addr //Reconfigure the i2c address of the host driver
    return;
}

int st_irq_off(void)
{
    // Turn off interrupt
    return 0;
}
```



```

int st_irq_on(void)
{
    // Open interrupt
    return 0;
}

void st_msleep(int time)
{
    msleep(time);
}

void st_usleep(int time)
{
    // usleep(time);
}

/*****

```

以上接口(1)和(2)就是客户的通用的I2C读写接口，客户在这里直接移植即可，(3)接口是直接读，是直接去读取寄存器，不需要下dummy write的一种方式，客户可参照通用读(1)功能，去除掉dummy write即可。

2.将固件文件由我们FAE帮忙转成h数组格式，按照数组名填入 SITRONIX_DUMP 和BTLD_CONTENT_1K 内

3.完成后通过调用 st_upgrade_fw接口完成升级。

```
st_upgrade_fw();
```

4.升级完成可通过获取0x0版本寄存器以及0x01状态寄存器，来确认升级成功。

六.获取原始值

1.获取原始值注意事项（Precautions for obtaining raw values）

当分析IC问题时，需要获取触摸容值的原始数据，建议合入此逻辑以供分析，可以直接使用下代码，除了特定api，其他逻辑均为C逻辑，理论上实现特定api（HFST Platform specific）后，调用 stx_get_mutualRaw_value，在log中打印原始值，如果没有办法串口打印信息，请帮忙保存文件以供查看。

```

/*****HFST Platform specific*****/
int stx_i2c_read_bytes(unsigned char addr, unsigned char *rxbuf, int len)
{
#ifdef SIX_I2C_SIMULATE
    return STX_RD_Reg(addr, rxbuf, len);
#else
    // IF I2C addr = 0x55
    // I2C start -> 0xAA -> RegAddr -> I2C stop -> delay -> I2C start -> 0xAB ->
    read with ACK -> ...read with NACK -> I2C stop
    return len; // if success, return len
#endif
}

```

```

int stx_i2c_write_bytes(unsigned char *txbuf, int len)
{
#ifdef SIX_I2C_SIMULATE
    return STX_WR_Reg_direct(addr, rxbuf, len);
#else
    // IF I2C addr = 0x55
    // I2C start -> 0xAA -> RegAddr ->value0 ->value1 ->value2 ...I2C stop
    return len; //if success, return len
#endif
}
/*****

```

简要概述流程:

1. 先获取TP通道数 (sitronix_get_xy_chs)
2. 设置相应的获取容值的模式 (Dist 则不需要设置)
3. 从 0x40 循环获取固定 Byte 的通道数据,每次获取完的 buf[2] 为相应的index.
(stx_i2c_read_bytes(0x40,disbuf,(tx_chs * 2 + 8));)
4. 数据均通过 disv打印输出。stx_get_mutualRaw_value接口, disv值会存放在stx_getraw.c 136行的rawl中

```

static int stx_get_mutualRaw_value(void)

```