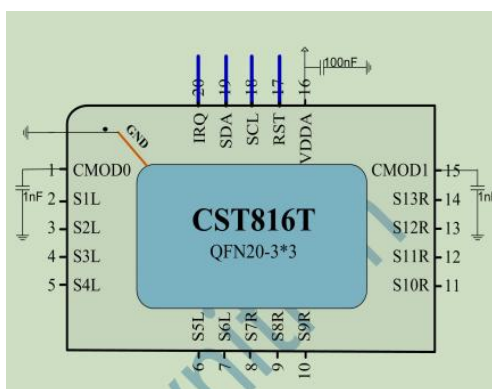
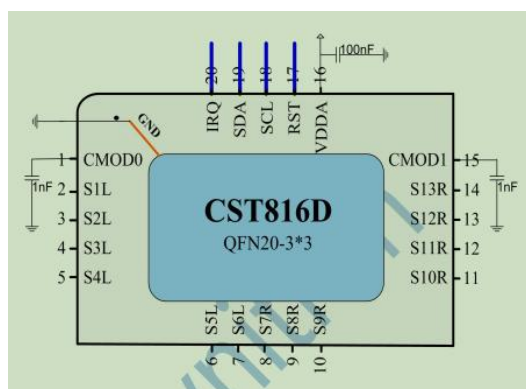
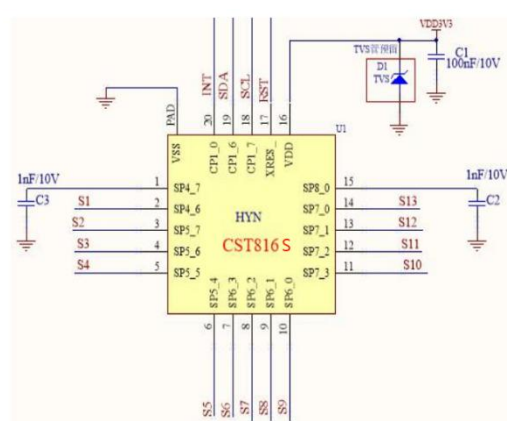
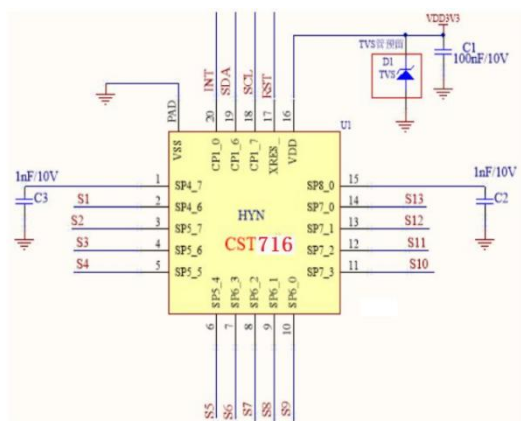


一、封装原理图

- 1、cst716 与 cst816s ,cst816T, cst816D 封装为 pin 对 pin, 可以直接替换。
Cst816S 和 cst816T 有触摸唤醒功能, cst716 与 cst816D 没有触摸唤醒。
详细设计参数, 查看设计要求, 数据手册



二、调试注意事项

- 1、调试项目之前要与代理商业务联系沟通，让业务发送项目调试邮件。
- 2、每个项目都有对应的 TP 固件，通信协议（与主控沟通）。
- 3、项目主板，不同的 TP 厂家，不同的功能片，盖板厚度等与 TP 有关的都要重新调试立项。
- 4、触摸 IC 出厂时为空片没有固件，没有芯片 ID，没有 IIC 地址。只有 BootLoader，需要更据对应项目调试，或者主控更据我们给的升级文件在线升级 TP 固件后才能使用。

三、触摸 IC 与主控交互流程

1、流程

开机上电→触摸 TP→触摸 IC 拉中断→主控跟据中断通过 IIC 读取坐标数据。

四、 驱动调试

1、触摸初始化

配置 IO 口，升级固件，配置中断。

读取**芯片 ID**：

```
-----CST716, CST816S, CST816T, CST816D, CST820 读 0xA7 寄存器,
      值:  CST716 : 0x20
           CST816S : 0xB4
           CST816T : 0xB5
           CST816D : 0xB6
-----CST826, CST830, CST836U, 读 0xAA 寄存器,
      值:  CST826: 0x11
           CST830: 0x12
           CST836U: 0x13
```

读取**软件版本号**：

```
-----CST716, CST816S, CST816T, CST816D, CST820 读 0xA9 寄存器,
-----CST826, CST830, CST836U, 读 0xA6 寄存器
```

注意：芯片出厂为空片，IIC 不通，没有芯片 ID，需要烧录升级后才能读值。

触摸 IC 有两个 IIC 器件地址：

进入 BootLoader **升级模式** IIC 地址是：0x6A(7 位)，加上读写为对应是：0xD4 写，0xD5 读
工作模式 IIC 地址是：0x15(7 位)，加上读写为对应是：0x2A 写，0x2B 读

初始化参考代码

```
8  /*
9  * 初始函数要放在开机时调用，每次开机只初始化一次。
10 */
11 bool ctp_cst816s_init(void)
12 {
13     u8 chip_id = 0;
14
15     //配置IO, 复位脚, SDA, SCL
16     GPIO_SetupConfig(GPIO_TOUCH_RESET, GPIO_MODE_OUTPUT | GPIO_MODE_PULLUP_1M, 1);
17
18     //读取芯片ID, 读0xA7寄存器, 值 cst816s:0xB4  cst716: 0x20
19     //注意: 芯片出厂时为空白, 没有芯片ID, 需要升级烧录固件后才有ID值。
20     ctp_cst816s_reset(); //平时为高电平, 复位时拉低10ms, 拉高后延时50ms。
21     drv_Touch_Read(0xA7, &chip_id, 1);
22     LOG_DEBUG("chip_id=%x\n", chip_id);
23
24     #if 1
25         //升级IIC地址为0x6A(7位)
26         ctp_hynitron_update(); //触摸IC升级固件, COB项目, 必须要开升级功能。
27         ctp_cst816s_reset(); //升级后拉一下复位, 确保退出boot模式。
28     #endif
29     //升级后要切换iic地址为0x15(7位)
30
31     //读取TP代码版本号, 通常每调试更新TP固件, 版本号会加1, 初始版本为1。
32     //如果要用到版本号最好要和调FAE说明一下, 确保每次调试固件版本号有更新。
33     drv_Touch_Read(0xA9, &g_tp_version, 1);
34     LOG_DEBUG("FwVersion=%x\n", g_tp_version);
35
36     //中断IO配置, 写中断处理函数。
37     GPIO_SetupConfig(GPIO_TOUCH_INT, GPIO_MODE_OUTPUT | GPIO_MODE_PULLUP_1M, 1);
38
39     return true;
40 }
```

2、升级调试

对于 COB 项目, 触摸 IC 焊在主板上, 不好通过烧录工具更新触摸 IC 固件, 所以需要主控添加升级 TP 固件驱动。我们调试好 TP 固件后通过工具生成升级 .h 或 .Bin 两种升级文件。主控将升级文件合入到驱动中, 通过 IIC 读写, 将数据烧录到触摸 IC 中。

升级流程:

- 1、进入 bootmode (拉复位进入, 有时间限制)
- 2、读取升级文件的 checksum, 与触摸 IC 里的 checksum 对比。
- 3、两种的 checksum 不一样, 调用更新固件函数。
- 4、读取升级后 IC checksum, 如果升级成功, IC 里的 checksum 和升级文件的 checksum 一样。

注意: Cst716 和 cst816S, cst816T, cst816D, cst820 升级驱动有差异, 进入 bootmode 指令不一样, 其他的都一样。Cst716, Cst826, Cst830, Cst836U 升级驱动一样。

- 进入 cst816s_enter_bootmode()函数, IIC 地址是 0x6A(7 位)

进入编程模式指令:

Cst716, cst826, cst830, cst836U: 0xA001 寄存器写 0xAA,
Cst816s, cst816D, cst816T, cst820: 0xA001 寄存器写 0xAB,

读取返回值:

Cst716, cst816S, cst826, cst830, cst836U: 读 0xA003 寄存器, 值 0x55,
Cst816D, cst816T, cst820: 读 0xA003 寄存器, 值 0xC1,

参考代码

```

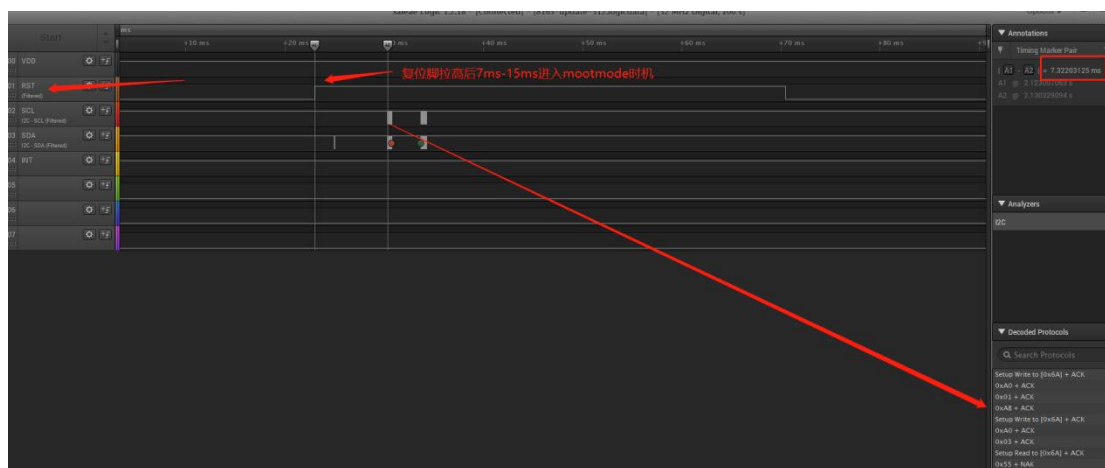
127: static int cst816s_enter_bootmode(void){
128:     char retryCnt = 50;
129:
130:     //进入bootmode需要拉复位，拉低10ms,拉高后延时10ms发指令，拉高后的时间在7~15m内发指令，否则进入不了bootmode
131:     nrf_gpio_pin_clear(CTP_RST_PIN);
132:     nrf_delay_ms(10);
133:     nrf_gpio_pin_set(CTP_RST_PIN);
134:     nrf_delay_ms(10);
135:
136:     while(retryCnt--){
137:
138:         uint8_t cmd[3];
139:         //cmd[0] = 0xAA; //CST716, cst826, cst830, cst836U
140:         cmd[0] = 0xAB; //CST816S, CST816D, CST816T
141:         TP_HRS_WriteBytes_userdata(0x6A<<1,0xA001,cmd,1,REG_LEN_2B); // enter program mode
142:         nrf_delay_ms(2);
143:         TP_HRS_read_userdata(0x6A<<1,0xA003,cmd,1,REG_LEN_2B); // read flag
144:         nrf_delay_ms(2);
145:
146:         //if (cmd[0] != 0xC1){ // CST816D, CST816T 返回值为0xC1
147:         if (cmd[0] != 0x55){ // CST716, CST816S, cst826, cst830, cst836U 返回值为0x55
148:             nrf_delay_ms(2);
149:             continue;
150:         }else{
151:             return 0;
152:         }
153:         nrf_delay_ms(2);
154:     }
155:     return -1;
156: }
157:

```

注意复位脚拉高后的延时时间

注意不同IC发的指令

注意不同IC读到的返回值



● 升级函数 ctp_hynitron_update ()

```

1: /*
2:
3:
4:
5: kal_bool ctp_hynitron_update(void)
6: {
7:     kal_uint8 lvalue;
8:     kal_uint8 write_data[2];
9:     kal_bool temp_result = CTP_TRUE;
10:
11:     if (cst816s_enter_bootmode() == 0){
12:         if(sizeof(app_bin) > 10){
13:             kal_uint16 startAddr = app_bin[1];
14:             kal_uint16 length = app_bin[3];
15:             kal_uint16 checksum = app_bin[5];
16:             startAddr <= 8; startAddr |= app_bin[0];
17:             length <= 8; length |= app_bin[2];
18:             checksum <= 8; checksum |= app_bin[4];
19:
20:             if(cst816s_read_checksum() != checksum){
21:                 cst816s_update(startAddr, length, &app_bin[6]);
22:                 cst816s_read_checksum();
23:             }
24:         }
25:         return CTP_TRUE;
26:     }
27:     return CTP_FALSE;
28: }
29:

```

进入boot模式，iic地址是0x6A

获取升级文件的checksum

读取IC的checksum和升级文件checksum对比，不一样就升级固件

具体升级函数，通过IIC给IC写数据

升级后再读一次芯片里的checksum,如果后升级文件checksum一样说明升级成功。可将checksum打log查看

- 读取 cst816s_read_checksum()函数

```

120:  *
121:  */
122: static uint32_t cst816s_read_checksum()
123: {
124:     union{
125:         uint32_t sum;
126:         uint8_t buf[4];
127:     }checksum;
128:     uint8_t cmd[3];
129:     char readback[4] = {0};
130:
131:     if (cst816s_enter_bootmode() == -1){
132:         return -1;
133:     }
134:
135:     cmd[0] = 0;
136:     TP_HRS_WriteBytes_updata(0x6A<<1,0xA003,cmd,1,REG_LEN_2B);
137:     nrf_delay_ms(500); ← 这个要延时500ms
138:
139:     checksum.sum = 0;
140:     TP_HRS_read_updata(0x6A<<1,0xA008,checksum.buf,2,REG_LEN_2B);
141:     LOG(LEVEL_INFO,"checksum.sum=%x \r\n",checksum.sum);
142:
143:     return checksum.sum;
144: }
145: } « end cst816s_read_checksum »
146:
147:

```

- 更新数据函数: cst816s_update(startAddr, length, &app_bin[6])

```

static int cst816s_update(uint16_t startAddr,uint16_t len,const unsigned char *src){
    uint32_t sum_len;
    uint8_t cmd[10];
    sum_len = 0;
    uint32_t k_data=0;
    k_data=len/512; //触摸IC缓存BUF大小512字节, 写满buf后刷到Flash。

    if (cst816s_enter_bootmode() == -1){//进编程模式, 不能省
        return -1;
    }

    for(uint32_t i=0;i<k_data;i++){
        if (sum_len >= len){
            return -1;
        }

        cmd[0] = startAddr&0xFF;
        cmd[1] = startAddr>>8;
        //参数: 器件iic地址, 寄存器地址, 数据, 数据长度, 寄存器地址长度
        TP_HRS_WriteBytes_updata(0x6A<<1,0xA014,cmd,2,2);
        delay1ms(2);
        #if 0
            TP_HRS_WriteBytes_updata(0x6A<<1,0xA018,src,512,2); //将数据写到buf区, 要写够512字节。
        #else
            {
                uint8_t i=0;
                for(i=0; i<4; i++){ //一次写128字节数据, 循环4次, 写够512字节
                    TP_HRS_WriteBytes_updata(0x6A<<1,0xA018+(i*128),src+(i*128),128,2);
                    delay1ms(2);
                }
            }
        #endif
        delay1ms(4);

        cmd[0] = 0xEE; // 一定要往寄存器地址0xA018写够512个字节数据后才发这条指令
        TP_HRS_WriteBytes_updata(0x6A<<1,0xA004,cmd,1,2);
        delay1ms(100); //0xA004写0xEE是将buf数据写到flash, 需要时间 >100ms

        {
            uint8_t retrycnt = 50;
            while(retrycnt--){
                cmd[0] = 0;
                TP_HRS_read_updata(0x6A<<1,0xA005,cmd,1,2);
                if (cmd[0] == 0x55){
                    cmd[0] = 0;
                    break;
                }
                delay1ms(10);
            }
        }

        startAddr += 512; //地址偏移
        src += 512;
        sum_len += 512;
    }
}

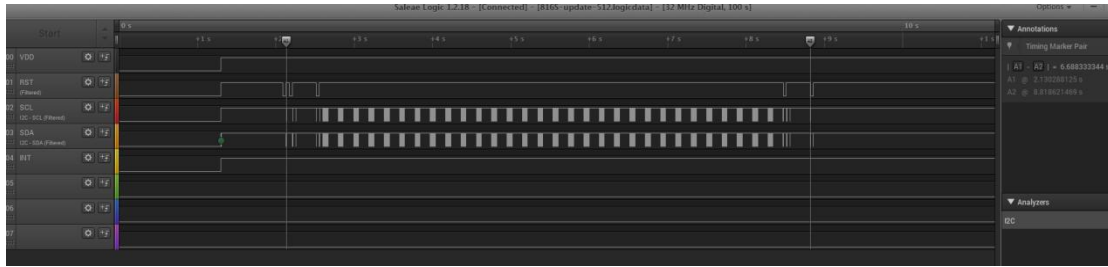
```

一次写512字节

主控不支持写512字节, 可以循环4次写128字节

注意延时时间

升级完整波形，循环 30 次每次写 512 字节数据，总共 512Byte*30=15KB



注意：IC 升级需要封装两个 IIC 读写函数。

```
//升级用的IIC写函数，iic地址0x6A(7位)，寄存器地址2两个字节，len: 写数据长度，lenth: 读寄存器长度(值为: 1或2)
kal_bool TP_HRS_WriteBytes_update(uint8_t device_addr,uint16_t RegAddr, uint8_t *Data,uint16_t len,uint8_t lenth)
{
    uint8_t data[550];
    if(lenth == 2 ){
        data[0]=RegAddr>>8;
        data[1]=RegAddr;
    }else{
        data[0]=RegAddr;
    }
    memcpy(&data[lenth],Data,len);
    twi_master_transfer(device_addr, data, len+lenth,true);
}

//升级用的IIC读函数，iic地址0x6A(7位)，寄存器地址2两个字节，len: 读数据长度，lenth: 读寄存器长度(值为: 1或2)
kal_bool TP_HRS_read_update(uint8_t device_addr,uint16_t RegAddr, uint8_t *Data,uint8_t len,uint8_t lenth)
{
    uint8_t data[3];
    if(lenth == 2 ){
        data[0]=RegAddr>>8;
        data[1]=RegAddr;
    }else{
        data[0]=RegAddr;
    }
    twi_master_transfer(device_addr, data, lenth,false);
    twi_master_transfer(device_addr+1, Data, len,true);
}
```

3、睡眠唤醒

● 睡眠

cst716, cst826, cst830, cst836U: 息屏前主控给触摸 IC 发睡眠指令 (0xA5,0x03)
cst816D: 息屏前主控给触摸 IC 发睡眠指令 (0xE5,0x03)
cst816S, cst816T, cst820: 指令模式: 息屏前主控给触摸 IC 发睡眠指令 (0xE5,0x03)
自动进入低功耗模式: 无触摸自动进入睡眠模式

● 唤醒

cst716, cst816D, cst826, cst830, cst836U: 主控在按键亮屏或翻腕亮屏前给触摸 IC 拉复唤醒。
cst816S, cst816T, cst820: 指令模式: 主控在按键亮屏, 翻腕亮屏前给触摸 IC 拉复唤醒。
自动进入低功耗模式: 触摸 TP 自动唤醒。

cst816s, cst816T 发指令时注意: cst816s, cst816T, cst820 有自动睡眠功能, 无触摸时就会进入低功耗睡眠模式, 在低功耗睡眠模式下, 触摸 IC 的 iic 将停止工作, 此时发指令 IC 会无应答, 导致发送指令失败。为确保指令发送成功, 在给触摸 IC 发送指令前, 需要主控先拉复位唤醒触摸 IC。

例如:

```
gpio_pin_write(TP_RESET_PIN,0); //拉低复位脚
delay_ms(10); //延时 10ms
gpio_pin_write(TP_RESET_PIN,1); //拉高复位脚
delay_ms(50); //IC 复位后需要 30ms 以上的初始化时间后才能正常工作
drv_Touch_Write(0xE5, 0x03, 1); //CST816S, CST816T 进入睡眠, 往 0xE5 寄存器写 0x03
```

```

//触摸 IC 进入睡眠模式
static void ctp_cst816s_deep_sleep(void)
{
    u8 data[1];
    u8 regAddr;

    regAddr = 0xE5;    //CST816s, CST816D, CST816T,CST820
    //regAddr = 0xA5;    //CST716, cst826, cst830, cst836U
    data[0] = 0x03;
    drv_Touch_Write(0xE5, data, 1); //往 0xE5 寄存器写 0x03
}

//cst816S 和 cst716 , cst816D, cst816T,cst820, cst826, cst830, cst836U 拉复位唤醒
static void ctp_cst816s_wakeup(void)
{
    ctp_cst816s_reset(); //复位脚拉低 10ms,拉高后延时 50ms
}

```

4、中断处理函数

4.1、触摸 IC 上报数据有两种模式

1、报点模式

只要有触摸，触摸 IC 就会每隔 10ms 左右拉一次中断，主控更据中断读取实时坐标。上下左右滑动由主控更据读到连续坐标值进行判断处理。有按下抬起状态码。

2、手势模式

触摸 IC 更据用户触摸轨迹识别成指定手势码（单击，双击，长按，上下左右滑动），当手势码生成时，会拉一次中断，主控更据中断读取生成的手势码。

● 注意点

报点模式：要求主控芯片有高速的处理能力，要做界面跟随效果，必须触摸 ic 设置为报点模式。

手势模式：不能做界面跟随效果。没有按下抬起状态码，做不了 UI 图标按下抬起变化真实效果（单击手势是抬起后生成的，没有按下状态码）。

Cst816S, cst816D, cst816T,cst820 支持报点模式，手势模式。支持寄存器配置切换两种模式。Cst716, cst826, cst830, cst836U 默认支持报点模式，手势模式需要与 FAE 沟通修改。不支持寄存器配置切换两种模式。

4.2、报点寄存器

触摸 IC 拉中断后，主控在中断处理函数中读取数据，建议从 0 寄存器连续读取 7 个字节。

```
u8 tp_temp[10];
drv_Touch_Read(0x00, &tp_temp[0], 7);
```

GestureID		@0x01							
	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	
Desc.	GestureID								
	位	名称	描述						
	[7:0]	GestureID	手势码						
			0x00: 无手势						
			0x01: 上滑						
			0x02: 下滑						
			0x03: 左滑						
			0x04: 右滑						
			0x05: 单击						
			0x0B: 双击						
			0x0C: 长按						

FingerNum		@0x02							
	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	
Desc.	FingerNum								
	位	名称	描述						
	[7:0]	FingerNum	手指个数。0: 无手指 1: 1个手指						

XposH		@0x03							
	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	
Desc.	Xpos[11:8]								
	位	名称	描述						
	[3:0]	XPos	X坐标高4位						

XposL		@0x04							
	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	
Desc.	Xpos[7:0]								
	位	名称	描述						
	[7:0]	XPos	X坐标低8位						

YposH		@0x05							
	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	
Desc.	Ypos[11:8]								
	位	名称	描述						
	[3:0]	YPos	Y坐标高4位						

YposL		@0x06							
	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	
Desc.	Ypos[7:0]								
	位	名称	描述						
	[7:0]	YPos	Y坐标低8位						

状态码：按下第一拍：0x0_，按下：0x8_，抬起：0x4_

4.3、解析报点数据

解析一

```
746
747 //中断处理函数
748 int CaptouchInterruptHandle(void)
749 {
750     //u8 TP_type;
751     u16 pdwSampleX, pdwSampleY;
752     u8 tp_temp[10];
753     u8 finger_num;
754
755     drv_Touch_Read(0x00, &tp_temp[0], 7); //从0寄存器连续读取7个字节数据
756
757     finger_num = tp_temp[2]; //手指个数
758     pdwSampleX = ((u16)(tp_temp[3] & 0x0F) << 8) + (u16)tp_temp[4];
759     pdwSampleY = ((u16)(tp_temp[5] & 0x0F) << 8) + (u16)tp_temp[6];
760     //LOG_DEBUG("x = %d, y = %d\r\n", pdwSampleX, pdwSampleY);
761
762     if (tp_temp[1] == 0x00) //报点模式，要求FAE将手势码清零，很多时候手势码没有被清除
763     {
764         //报点模式可根据finger_num判断
765         u8 touch_type = tp_temp[3] >> 4; //状态码
766         LOG_DEBUG("touch_type = %x\r\n", touch_type);
767
768         if (touch_type == 0x00) { //第一拍按下
769             LOG_DEBUG("TP_EVENT_TYPE_PRESS_DOWN x = %d, y = %d\r\n", pdwSampleX, pdwSampleY);
770             tp_onEventProcess(TP_EVENT_TYPE_PRESS_DOWN, pdwSampleX, pdwSampleY);
771         } else if (touch_type == 0x04) { //抬起
772             LOG_DEBUG("TP_EVENT_TYPE_PRESS_UP x = %d, y = %d\r\n", pdwSampleX, pdwSampleY);
773             tp_onEventProcess(TP_EVENT_TYPE_PRESS_UP, pdwSampleX, pdwSampleY);
774         }
775     }
776
777     //以下是手势模式，通过读取手势码做响应动作
778     else if (tp_temp[1] == 0x05)
779     {
780         LOG_DEBUG("TP_EVENT_TYPE_CLICK x = %d, y = %d\r\n", pdwSampleX, pdwSampleY);
781         tp_onEventProcess(TP_EVENT_TYPE_CLICK, pdwSampleX, pdwSampleY);
782     }
783     else if (tp_temp[1] == 0x0C)
784     {
785     }
```

解析二

4.4、CST816S, CST816T 常用寄存器配置

读 shipID

```
drv_Touch_Read(0xA7, &chip_id, 1); //值 cst816s: 0xB4    cst716: 0x20
```

读 TP 固件版本号

```
drv_Touch_Read(0xA9, &g_tp_version, 1); //每更新一版固件，版本号递增 1
```

设置报点率，拉中断时间间隔

```
data[0] = 0x02;           //单位 10ms, 设置时间= data[0]*10ms  
drv_Touch_Write(0xEE, data, 1); //默认报点率为 100HZ, 10ms 拉一次中断,
```

设置为报点模式

```
data[0] = 0x60;  
drv_Touch_Write(0xFA, data, 1); // 触摸 TP 时每隔 10ms 左右拉一次中断。时时上报坐标
```

设置为手势模式

```
data[0] = 0x11;  
drv_Touch_Write(0xFA, data, 1); // 生成手势后拉一次中断。
```

设置为报点+手势模式

```
data[0] = 0x71;  
drv_Touch_Write(0xFA, data, 1);
```

设置自动复位时间

```
data[0] = 0x05;           //单位 1S, 为 0 时不启用此功能。默认为 5  
drv_Touch_Write(0xFC, data, 1); // x 秒内有触摸但无有效手势时，自动复位
```

data[0] = 0x10; //单位 1S, 为 0 时不启用此功能。默认为 10

```
drv_Touch_Write(0xFD, data, 1); // 长按 x 秒后自动复位
```

禁止自动进入低功耗模式

```
data[0] = 0x01;           //为非 0 值时，禁止自动进入低功耗模式。  
drv_Touch_Write(0xFE, data, 1); // 默认为 0，使能自动进入低功耗模式
```