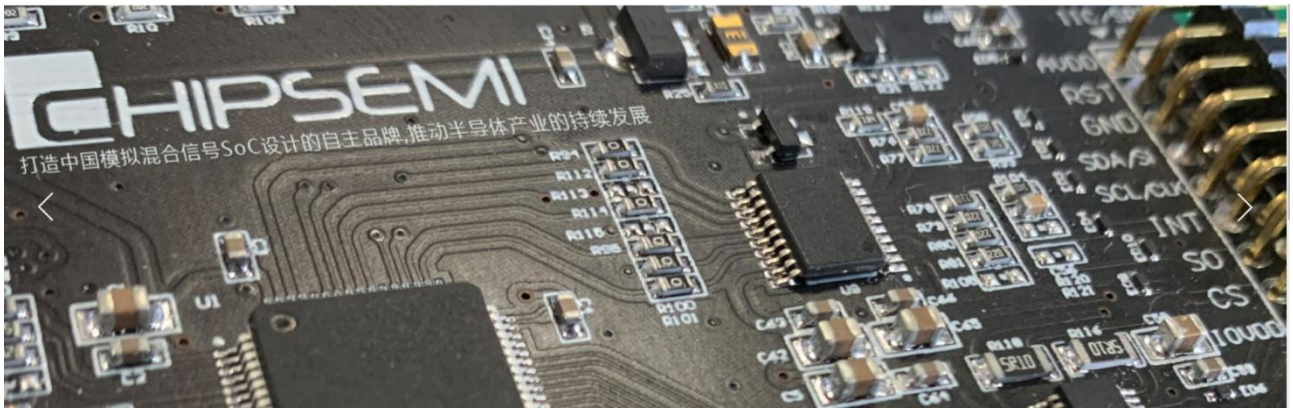
CHSC5816 触控芯片使用说明 V1



CHSC5816 触控芯片使用说明 V1 1

时序：上电 下电 复位 ### 2

IIC 通信读写时序 ### 3

标准命令接口 ### 4

芯片工作模式 ### 6

获取工程信息 ### 7

报点数据格式 ### 7

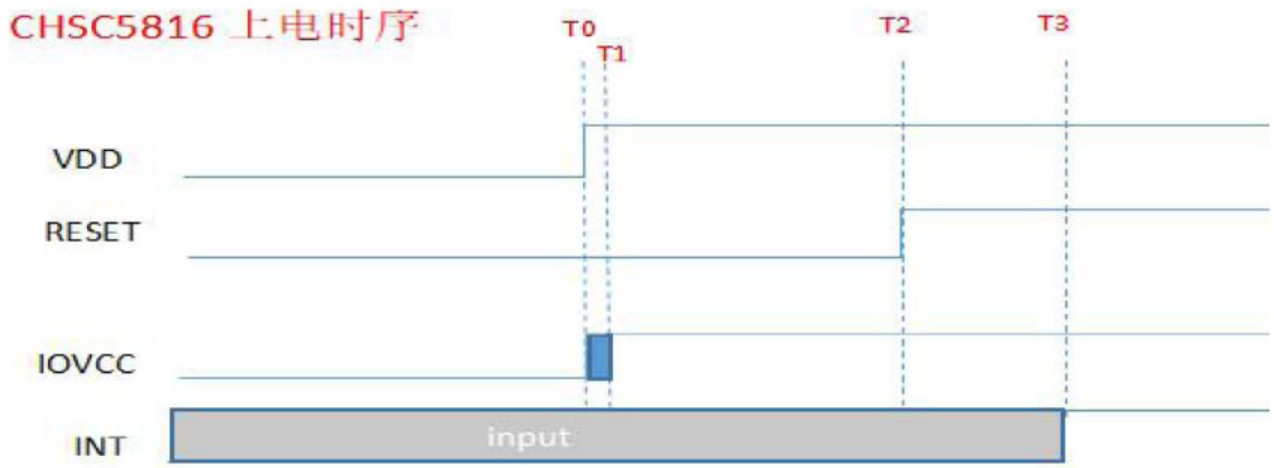
MCU 驱动的移植 ### 9

固件升级 ### 11

常见问题排查 ### 12

时序：上电 下电 复位

CHSC5816 的上电时序。



* CHSC5816 supports dual power supply (the motherboard provides VDD and IOVCC) and single power supply (the motherboard only provides VDD, the IO voltage is provided by the internal 1.8V LDO of CHSC5816, or directly connected VDD)

* VDD > = IOVCC * If IOVCC of motherboard is used, it is required that T1-T0 > = 0, try to minimize the time interval as much as possible.

* T2-T0 > = 1ms, Pull RESET low before VDD is pulled up, T2-T0 is the reset time for chip hardware.

* Configure INT PIN to input mode (close pull-up) before VDD is pulled up, and set to interrupt input mode after CHSC5816 reset.

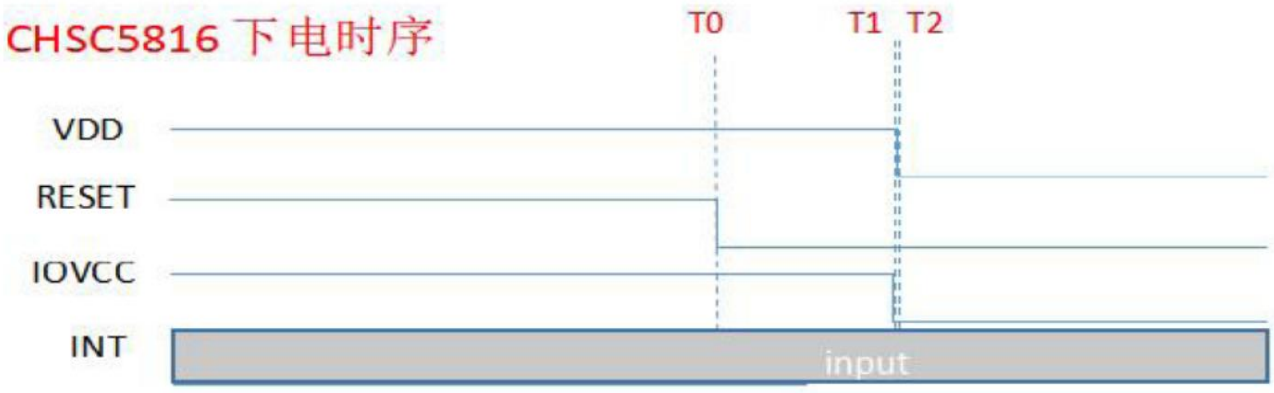
中文：CHSC5816 支持双电源（主板提供 VDD 和 IOVCC）和单电源（主板仅提供 VDD，IO 电压由 CHSC586 的内部 1.8V LDO 提供，或直接连接 VDD）。

VDD>=IOVCC，如果使用主板的 IOVCC，则要求 T1-T0>=0，尽量缩短时间间隔。

T2-T0>=1ms，在 VDD 上拉之前将 RESET 拉低，T2-T0 是芯片硬件的复位时间。

在 VDD 抬升之前将 INT PIN 配置为输入模式（关闭上拉），并在 CHSC5816 复位之后将其设置为中断输入模式。

CHSC5816 的下电时序。

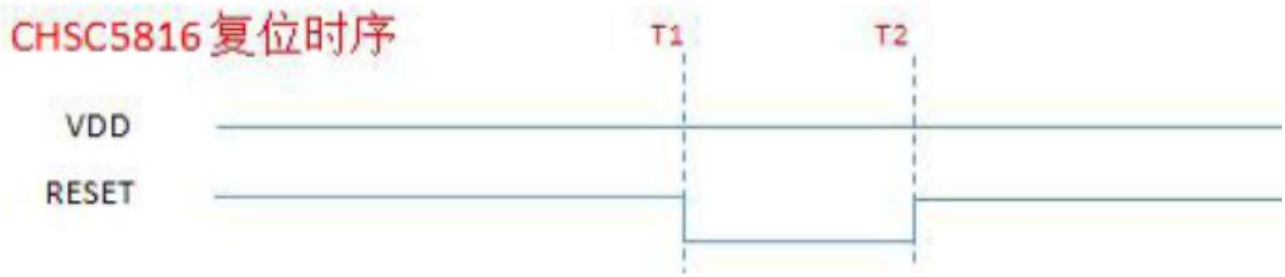


* If IOVCC of motherboard is used, it is required that T2-T1 > = 0, try to minimize the time interval as much as possible.

* Pull RESET low before VDD is turned off

* Keep INT in input mode(turn off pull-up) and can be changed to output low after T2

中文：如果使用主板 IOVCC，则要求 T2-T1>=0，尽量缩短时间间隔。VDD 关闭前将 RESET 拉低。将 INT 保持在输入模式（关闭上拉），并可在 T2 后变为输出低。



T2-T1 > = 3ms, Interrupt of INT can be turned off before RESET, and re-enabled after RESET is completed.

中文：T2-T1>=3ms，INT 的中断可以在 RESET 之前关闭，并在 RESET 完成后重新启用。

IIC 通信读写时序

设备地址：7 位地址为 0x2E，左移 1 位后，写通信为 0x5C，读通信为 0x5D。

IIC 通信写流程：

Start + 0x5C + ACK + ADDR[31:24] + ACK + ... + ADDR[7:0] + ACK + DATA + ACK + ... + DATA + ACK + STOP

Start	id	w	ack	Addr[31:24]	ack	Addr[23:16]	ack	Addr[15:8]	ack	Addr[7:0]	ack	-------连续多个 byte 数据（大端）	Stop
												ack	

IIC 通信读流程：

Step1：Start + 0x5C + ACK + ADDR[31:24] + ACK + ... + ADDR[7:0] + ACK + STOP

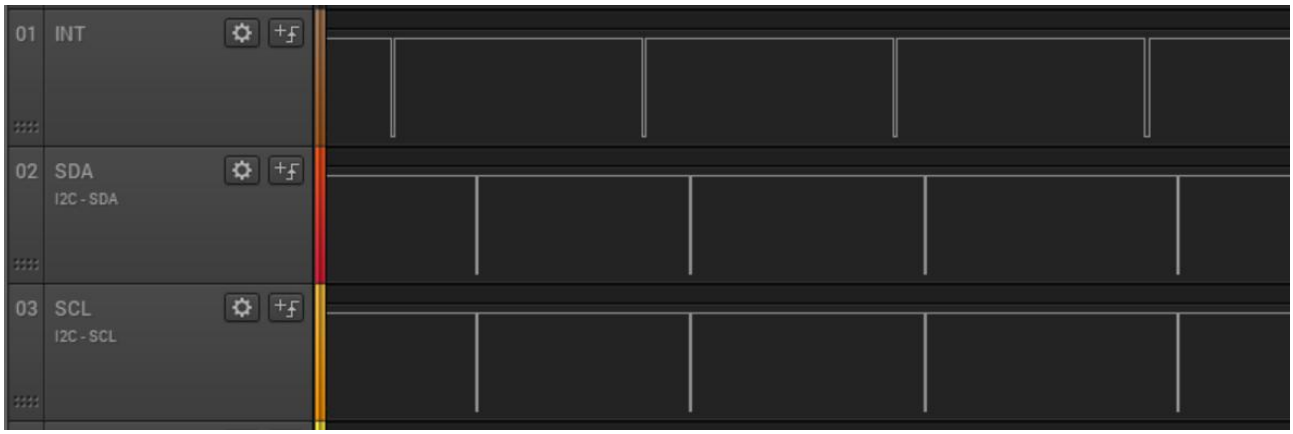
Step2：Start + 0x5D + ACK + DATA + ACK + ... + DATA + NACK + STOP

Start	id	w	ack	Addr[31:24]	ack	Addr[23:16]	ack	Addr[15:8]	ack	Addr[7:0]	ack	Stop 可能没有

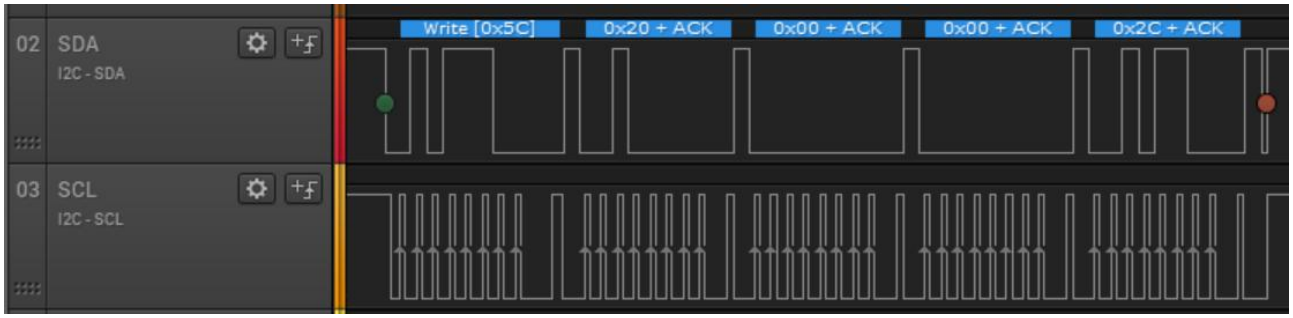
Start	id	r	ack	data[7:0]	ack	data[7:0]	ack	data[7:0]	ack	data[7:0]	ack	-------连续多个 byte 数据（大端）	stop
												NAK	

示例：MCU 响应中断，读取触摸数据的波形。

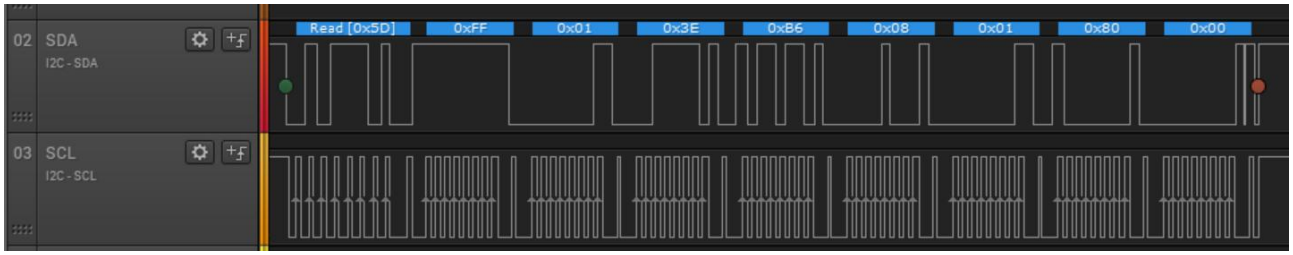
CHSC5816 的 INT 引脚，触发一次中断，MCU 就读一次数据。



写设备地址 0x5C，写四个寄存器地址。



写设备地址 0x5D，读取所需长度触摸数据。



标准命令接口

CHSC5816 软件具有 CMD（命令）和 RSP（响应）接口，所有 CMD 和 RSP 均为 16 字节。命令包括：暂停 CDSP 扫描，重新启动 CDSP 扫描、更新 CFG、切换电源模式、更改报告率等。CMD 和 RSP 的目标地址为 0x2000_0000。

CMD（命令）和 RSP（响应）数据格式如下：

```
typedef struct _ctp_cmd_std_t{
    u16 chk; // 16 bit checksum
    u16 d0;  //data 0
    u16 d1;  //data 1
    u16 d2;  //data 2
    u16 d3;  //data 3
    u16 d4;  //data 4
    u16 d5;  //data 5

    u8  id;   //offset 15
    u8  tag;  //offset 16
}ctp_cmd_std_t;
```

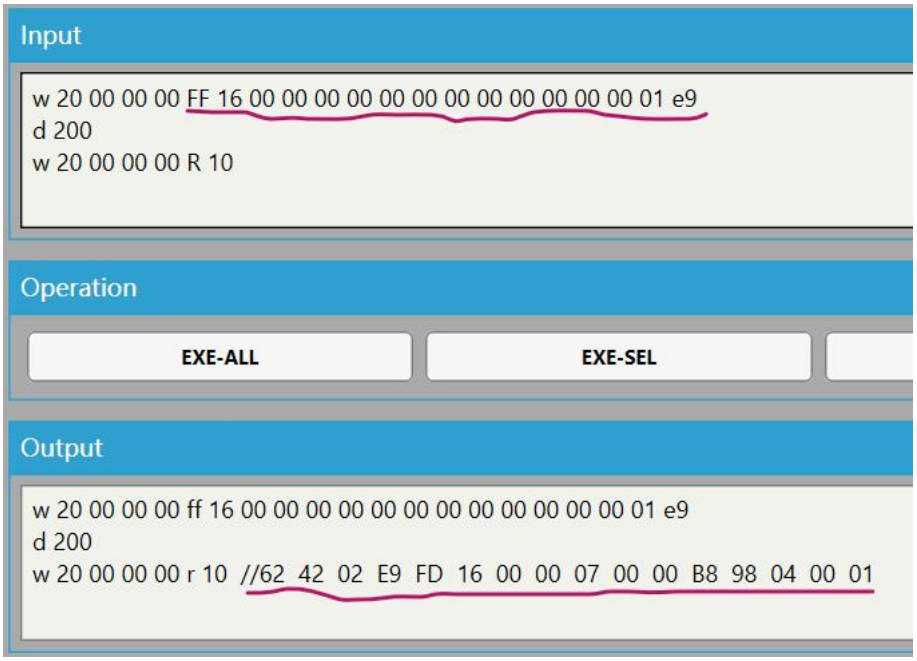
```
typedef struct _ctp_rsp_std_t{
    u16 chk; // 16 bit checksum
    u16 d0;  //data 0
    u16 d1;  //data 1
    u16 d2;  //data 2
    u16 d3;  //data 3
    u16 d4;  //data 4
    u16 d5;  //data 5

    u8  cc;  //offset 15
    u8  id;  //offset 16
}ctp_rsp_std_t;
```

例如 0x01 为读取固件 ID 命令：

发送命令：FF 16 00 00 00 00 00 00 00 00 00 00 00 00 01 E9

得到结果：62 42 02 E9 FD 16 00 00 07 00 00 B8 98 04 00 01

其中，d0 为 0xE902 表示固件 ID，d1 为 0x16FD 为 ID 的反码。



例如 0x04 为读取固件信息命令：

您可以通过发送命令（CMD=0x04）来获取触摸屏的物理信息，如果该命令可以正确响应，响应数据包含触摸屏的一些信息，即：

1、X 分辨率：“D0”是触摸屏的 X 轴分辨率

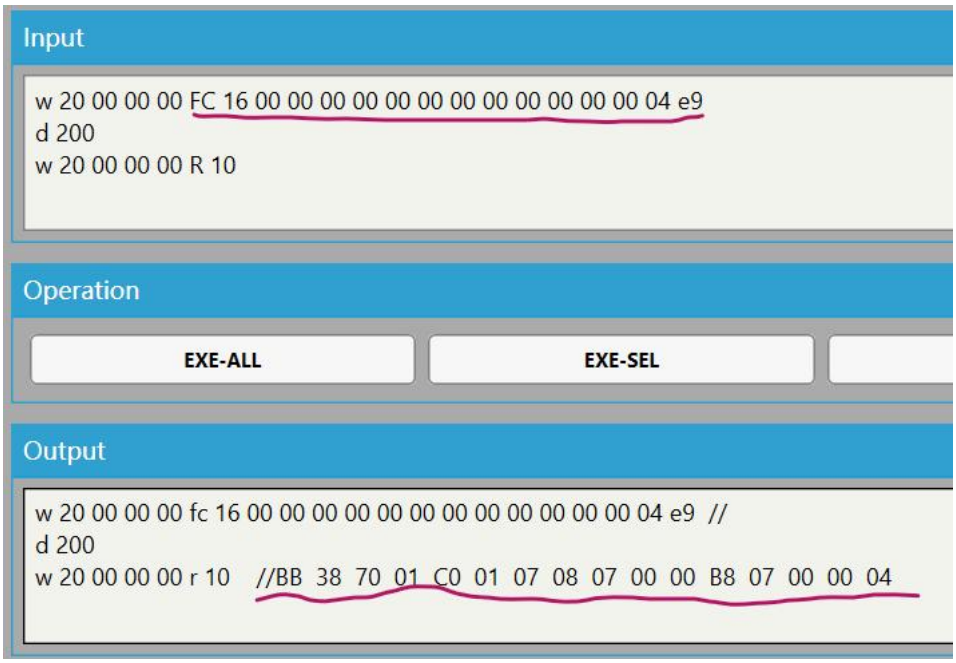
2、Y 分辨率：“D1”是触摸屏的 Y 轴分辨率

3、通道数：“D2”的低字节是 RX 数目，“D2“的高字节是 TX 数目。

发送命令：FC 16 00 00 00 00 00 00 00 00 00 00 00 00 04 e9

得到结果：BB 38 70 01 C0 01 07 08 07 00 00 B8 07 00 00 04

其中，d0 为 0x0170 表示 368，d1 为 0x01C0 表示 448，d2 为 0x0807 表示 TX 通道数目为 8，RX 通道数目为 7。



芯片工作模式

5816 在各种模式下，消耗的电流。

（1）Active 模式电流 1.45mA 左右。

（2）DeepSleep 电流 1uA 左右。

（3）IDLE 模式电流 24.6uA 左右。

Active 模式

亮屏状态是用户最有可能操作 TP 的阶段，需要快速响应并准确地反映客户想要执行的操作。这种状态下的工作模式称为 Active 模式。Active 模式可以大致分为两种情况：触摸和无触摸。

Deep sleep 低功耗模式

进入该模式：调用以下函数，发送 16 字节命令。

```
static void semi_touch_enter_deep(void)
{
    uint8  CmdBuffer[16] = {0xF8, 0x16, 0x05, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xE9};

    prompt_trace(MOD_WAP,"semi_touch_enter_deep\n");

    semi_touch_write_bytes(0x20000000, CmdBuffer, 16);
}
```

退出该模式：Reset 引脚给予一个复位，即可恢复到 active 模式。

IDLE 模式，也叫手势唤醒模式

进入该模式：调用以下函数，发送 16 字节命令 20 16 02 00 db 00 00 00 00 00 00 00 00 00 03 e9。

```
static void semi_touch_enter_idle(void)
{
    uint8  CmdBuffer[16] = {0x20, 0x16, 0x02, 0x00, 0xDB, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xE9};

    prompt_trace(MOD_WAP,"semi_touch_enter_idle\n");

    semi_touch_write_bytes(0x20000000, CmdBuffer, 16);
}
```

退出该模式：用手势唤醒，如点击、上滑、下滑、左滑、右滑。唤醒后，5816 会发送“FE + 唤醒手势”给 MCU。

获取工程信息

5816 的工程信息位于 0x20000014 地址处。

Little-Endian				
Addr.	byte 0	byte 1	byte 2	byte 3
0x20000014	fw-version		reserved	
0x20000018	firmware startup status			
0x2000001C	fw-version	projec ID		vendor ID
0x20000020	offset address of RAW data		offset address of DIF data	

代码中定义了名为 img_header_t 的结构体，用来描述这些信息。驱动读取的 image_header 信息，其中 sig 的值应该为 0x43534843，即"CHSC"。如果不是该值表示有异常，需检查驱动，甚至需要进一步检查 5816 芯片。

```
typedef struct _img_header_t
{
    uint16_t fw_ver;
    uint16_t resv;
    uint32_t sig;
    uint32_t vid_pid;
    uint16_t raw_offet;
    uint16_t dif_offet;
}img_header_t;
```

报点数据格式

CHSC5816 上报给 MCU 的触摸数据格式。

地址	BIT0	BIT1	BIT2	BIT3	BIT4	BIT5	BIT6	BIT7
0x2000002C	EVENT type(0xFF:normal touch event, 0xFE:gesture event)							
0x2000002D	Finger number or gesture code							
0x2000002E	point 1: X coordinate - L8							
0x2000002F	point 1: Y coordinate - L8							
0x20000030	pressure value							
0x20000031	point 1: X coordinate - H4				point 1: Y coordinate - H4			
0x20000032	point 1: finger ID				point 1: touch event			
0x20000033	point 2: X coordinate - L8							
0x20000034	point 2: Y coordinate - L8							
0x20000035	pressure value							
0x20000036	point 2: X coordinate - H4				point 2: Y coordinate - H4			
0x20000037	point 2: finger ID				point 2: touch event			
0x20000038	CheckSum 校验和（前 12 个字节的累加和，低 8 位，取反）							

代码中定义了 2 个结构体用于处理数据。结构体 rpt_data_t 为报点的帧格式，结构体 rpt_point_t 为单个触摸点的数据格式。其中，手指 id 为 0 或 1；Event 事件以及编码为 Down=0x0、Continue=0x8、Up=0x4；

```
typedef struct _rpt_point_t {
    union {
        struct {
            u8 x_l8;
            u8 y_l8;
            u8 z;
            u8 x_h4:4;
            u8 y_h4:4;
            u8 id:4;
            u8 event:4;
        };
        u8 data[5];
    };
}rpt_point_t;

typedef struct _rpt_data_t {
    u8 act;
    u8 num;
    rpt_point_t points[MAX_FINGER_NUM];
    u8 pad[4];
}rpt_data_t;
```

报一个点的数据格式。

如果支持手势，0x20000030 处的 pressure value 值的高 4 位为手势编码，左滑 0x1、右滑 0x2、上滑 0x3、下滑 0x4，低 4 位固定为 0x8。当 Finger number 为 0x00 时，表示手指抬起，Event 事件为 Up=0x4，X Y 为手指抬起位置的坐标值。

地址	BIT0	BIT1	BIT2	BIT3	BIT4	BIT5	BIT6	BIT7
0x2000002C	0xFF：normal touch event							
0x2000002D	Finger number 0x01 或 0x00							
0x2000002E	point 1: X coordinate - L8							
0x2000002F	point 1: Y coordinate - L8							
0x20000030	pressure value							
0x20000031	point 1: X coordinate - H4				point 1: Y coordinate - H4			
0x20000032	point 1: finger ID				point 1: touch event			
0x20000033	CheckSum 校验和（前 7 个字节的累加和，低 8 位，取反）							

报两个点的数据格式。

最末尾 0x20000038 地址处，增加一个字节作为校验和。

地址	BIT0	BIT1	BIT2	BIT3	BIT4	BIT5	BIT6	BIT7
0x2000002C	0xFF：normal touch event							
0x2000002D	Finger number 0x02							
0x2000002E	point 1: X coordinate - L8							
0x2000002F	point 1: Y coordinate - L8							
0x20000030	pressure value							

0x20000031	point 1: X coordinate - H4	point 1: Y coordinate - H4
0x20000032	point 1: finger ID	point 1: touch event
0x20000033	point 2: X coordinate - L8	
0x20000034	point 2: Y coordinate - L8	
0x20000035	pressure value	
0x20000036	point 2: X coordinate - H4	point 2: Y coordinate - H4
0x20000037	point 2: finger ID	point 2: touch event
0x20000038	CheckSum 校验和（前 12 个字节的累加和，低 8 位，取反）	

睡眠手势唤醒。

如果设置了手势唤醒，且有唤醒动作时，上报数据为 0xFE + 手势编码。

0x2000002D 处的手势编码为左滑 0x20、右滑 0x21、上滑 0x22、下滑 0x23、单击 0x24、双击 0x25。

地址	BIT0	BIT1	BIT2	BIT3	BIT4	BIT5	BIT6	BIT7
0x2000002C	EVENT type(0xFE gesture event)							
0x2000002D	gesture code							
0x2000002E	CheckSum 校验和（前 2 个字节的累加和，低 8 位，取反）							
0x2000002F								
0x20000030								
0x20000031								
0x20000032								
0x20000033								

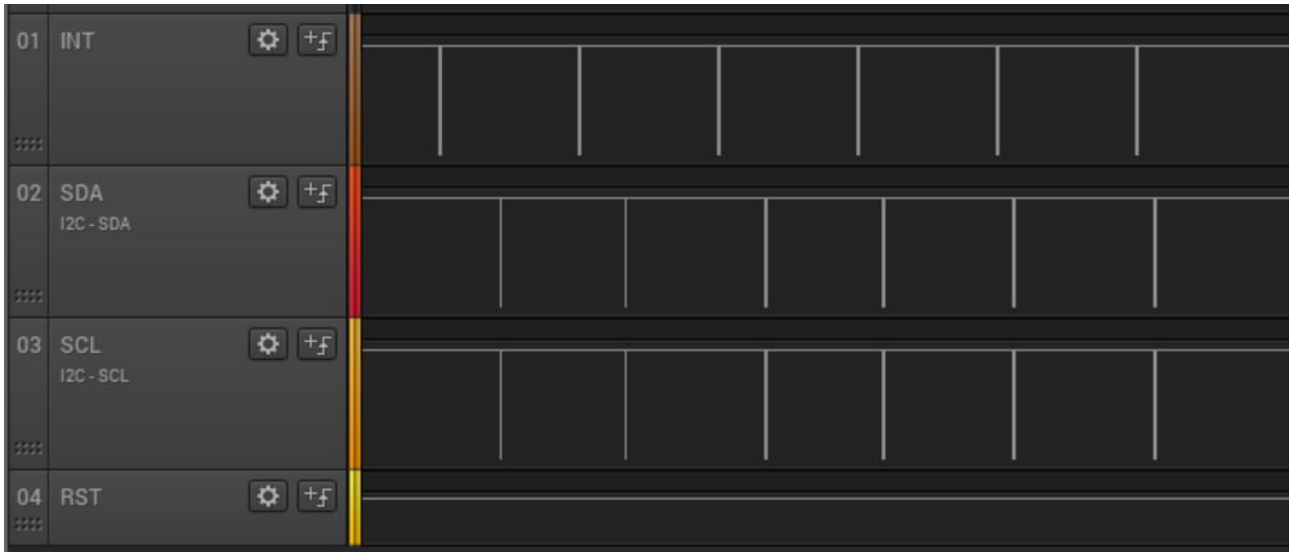
覆盖屏幕检测。

如果配置了打开覆盖检测功能，上报数据为 0xFE 0xDE。

地址	BIT0	BIT1	BIT2	BIT3	BIT4	BIT5	BIT6	BIT7
0x2000002C	EVENT type(0xFE gesture event)							
0x2000002D	gesture code 0xDE							
0x2000002E	CheckSum 校验和（前 2 个字节的累加和，低 8 位，取反）							
0x2000002F								
0x20000030								
0x20000031								
0x20000032								
0x20000033								

MCU 驱动的移植

驱动框架非常简单。触摸功能，需要实现两处主要的地方：IIC 底层读写接口、中断响应函数读取触摸数据。



要将驱动移植到当前平台，需要初始化 IIC SCL SDA 通信引脚、INT 中断低脉冲输入引脚、RESET 复位输出引脚，同时修改驱动的以下几个地方。

（1）延时函数：ms 为单位的延时。

```
static void semi_touch_msdelay(uint32_t millisecs)
{
    HAL_Delay(millisecs);
}
```

（2）复位函数：将 Reset 引脚拉低再拉高，复位芯片。

```
static void semi_touch_reset(void)
{
    HAL_GPIO_WritePin(RST_GPIO_Port, RST_Pin, GPIO_PIN_RESET);
    HAL_Delay(5);
    HAL_GPIO_WritePin(RST_GPIO_Port, RST_Pin, GPIO_PIN_SET);
    HAL_Delay(3);
}
```

（3）IIC 通信写数据函数：将指定长度的数据，写到 reg 开始的地址。

```
int32_t semi_touch_iic_write(uint32_t reg, uint8_t* pdata, uint16_t len)
```

（4）IIC 通信读数据函数：从 reg 开始的地址，读取指定长度的数据，。

```
int32_t semi_i2c_read_bytes(uint32_t reg, uint8_t* pdata, uint16_t len)
```

（5）中断响应函数：当 5816 有数据上报时，TP 芯片会在 INT 引脚输出低脉冲，MCU 需要及时响应中断，调用 void semi_touch_irq_handler_imp(void) 中断处理函数，并读取触摸数据。

```
676
677  void semi_touch_irq_handler_imp(void)
678  {
679      int  pointed = 0;
680      union rpt_point_t* ppt;
681      unsigned char gestCode;
682      unsigned char data[8];
683      int       x, y;
684      PRINT_DBG("semi_touch_irq_handler_imp\r\n");
685
686      if(semi_touch_read_bytes(0x2000002c, data, 8)){
687          PRINT_DBG("chsc:read pixel data fail\n" );
688          return;
689      }
690
691      PRINT_DBG("imp = %x %x\r\n",data[0], data[1]);
692
693      pointed = 0;
694      ppt = (union rpt_point_t*)&data[2];
695      if((data[0] == 0xff) && (data[1] <= 2)){
696          if(data[1] > 0){
697              pointed = 1;
698              x = (unsigned int)(ppt->rp.x_h4 << 8) | ppt->rp.x_l8;
699              y = (unsigned int)(ppt->rp.y_h4 << 8) | ppt->rp.y_l8;
700          }
701      }else{
702          return;
703      }
```

固件升级

CHSC5816 支持固件升级，驱动代码大部分都是固件升级相关的代码。

在头文件“chsc5816_ctp.h”中，有两个重要的数组。

数组 const __align(4) uint8_t chsc_upd_data[]放置的是 CHSC5816 的固件数据。如果固件需要升级到新版本，这个新固件数据由基合这边提供，用户只需要用新固件的数据，替换这个数组的旧数据。

数组 const uint8_t fw_5816_burn[]放置的是 Bridge_Sram-Code 小程序，它起到“接收、烧录”的中间作用。升级时，驱动需要先将这个小程序下载到 5816 内部的 SRAM，并运行起来。然后驱动再继续将 5816 的固件，传递给这个小程序，由它烧录到 5816 内部的 FLASH 存储器。

驱动在启动过程，会检查 5816 的固件版本，如果版本是最新，则跳过升级。如果版本低于 uint8_t chsc_upd_data[]数组存储的固件版本，则启动升级流程。

如果想跳过版本检查，进行强制升级，可以在 int semi_touch_init()初始化函数里面，将“st_dev.needUpd”强制赋值为 1，如下图。升级结束后，需删除“st_dev.needUpd = 1”这行代码。

```
727
728 int semi_touch_init()
729 {
730     semi_touch_power_int();
731
732     if(semi_touch_dect() != SEMI_DRV_ERR_OK){
733         PRINT_DBG("chsc:no chsc5816\r\n" );
734         return -1;
735     }
736
737     semi_touch_setup_check();
738
739     semi_touch_update_check();
740
741     st_dev.needUpd = 1;
742
743     if(st_dev.needUpd){
744         semi_touch_update(st_dev.updPdata, st_dev.newBootLen);
745         semi_touch_setup_check();
746     }
747
748     semi_touch_reset();
749     st_dev.ctp_status = CTP_POINTING_WORK;
750
751     return 0;
752 }
753
```

常见问题排查

Todo