Sitronix

ST1232/ST1332

Touch Screen Controller

Note: Sitronix Technology Corp. reserves the right to change the contents in this document without prior notice. This is not a final specification. Some parameters are subject to change.

I2C Interface Protocol

Version 1.6

2011/01/17

Sitronix Technology Corporation

CONTENT

1. REVISION HISTORY

Version	Data	Author	Description
0.1	2009/11/23	CT Chen	First release.
0.2	2009/12/21	CT Chen	Move Z0 from address 0x18 to 0x15.
0.3	2010/2/2	CT Chen	Move Z0 from address 0x15 to 0x18.
0.4	2010/3/3	CT Chen	Add Raw Data Enable bit to Device Control Register.
0.5	2010/3/10	CT Chen	1. Change Raw Data Registers to Data Registers add Raw Mode and Delta Mode to Device Control Register for Data Register. 2. Add Page Register.
0.6	2010/3/11	CT Chen	1. Rename XY Max Coord. to XY Resolution and support changing XY Resolution by host. 2. Add Keys to register 0x10 and move Proximity to 0x11. 3. Add Firmware Update section.
0.7	2010/4/19	CT Chen	1. Rename Sleep mode to Power Down mode. 2. Rename Device Control Register to Device Control Register 1. 3. Rename Sensitivity Control Register to Device Control Register 2. 4. Add Idle mode and Idle Timeout. 5. Add Device Status: Init. 6. Support 8 Keys and change Keys register to address 0x11. 7. Remove Proximity. 8. Change Gesture register to address 0x10(bit 7:3).
0.8	2010/5/12	CT Chen	1. Add Auto Tune control bit to Device Control Register. 2. Add gestures for Rotate CW and CCW.
0.9	2010/5/17	CT Chen	Add Max Drift Threshold.
1.0	2010/6/11	CT Chen	Add Sample Codes.
1.1	2010/7/12	CT Chen	Add ISP Firmware Update Protocol.
1.2	2010/8/10	CT Chen	Add Firmware Rev.
1.3	2010/10/25	MengChih Chen	Add Flash Update Disable Option.
1.4	2010/11/17	CT Chen	1. Add Default I2C Address section. 2. Add Object Approaching and Leaving gestures for proximity.
1.5	2010/12/21	CT Chen	Add Touch Threshold, Noise Threshold and Key Threshold to Threshold Setting section.

1.6	2011/1/17	CT Chen	1. Add Invalid Platform Error Code for valid firmware checking between ST1232 and ST1332. 2. Remove Idle bit and add Proximity Enable bit.

2. INTRODUCTION

ST1232/ST1332 is a low-cost single chip solution for capacitive touch screen. It is a RISC architecture microcontroller device with SPI, I2C, UART and I/O host interfaces. For application, ST1232/ST1332 supports raw data, coordinate, gestures and device control information for host side application. For communication interface, ST1232/ST1332 supports register mapped interface protocol for host device to retrieve information through ST1232/ST1332 host interfaces. Developer can get information about raw data, coordinates, gestures or device control and develop their system very easily through the register interface protocol.

The capacitive touch sensor is covered with a plastic or glass cover lens. It provides auto-calibrate parameters for a wide range of capacitance on the touch sensor (5pF~35pF). The touch sensor controller converts touch sensor data into X and Y coordinates for each finger, or into gestures according to the motions of fingers detected by controller. The ST1232/ST1332 supports multi-package for various application. Package types are QFN and TSSOP.

3. I2C HOST INTERFACE PROTOCOL

3.1. Default I2C Address

I2C address is default to **0x55** (7-bits address) for ST1232/ST1332. If the I2C address is conflict with another I2C device's address on same bus, user can change ST1232/ST1332 I2C address by TTK PC Utility.

3.2. Register Read

For reading register value from I2C device, host has to tell I2C device the *Start Register Address* before reading corresponding register value.

I2C Start	I2C Header (W)	Start Reg. Addr. (a)	I2C Stop	I2C Start	I2C Header (R)	Value of Reg(a)	Value of Reg(a+1)	…	Value of Reg(a+n)	I2C Stop

Figure 1 - Register Read Format.

ST1232/ST1332 I2C host interface protocol supports *Repeated Register Read*. That is, once the *Start Register Address* has been set by host, consequent I2C Read(R) transactions will directly read register values starting from the *Start Register Address* without setting address first, as shown in Figure 2.

I2C Start	I2C Header (R)	Value of Reg(a)	Value of Reg(a+1)	…	Value of Reg(a+n)	I2C Stop	I2C Start	I2C Header (R)	Value of Reg(a)	Value of Reg(a+1)	…	Value of Reg(a+n)	I2C Stop

Figure 2 - Repeated Register Read.

3.3. Register Write

For writing register to I2C device, host has to tell I2C device the Start Register Address in each I2C Register Write transaction. Register values to the I2C device will be written to the address starting from the Start Register Address described in Register Write I2C transaction as shown in Figure 3.

I2C Start	I2C Header (W)	Start Reg. Addr. (a)	Value to Reg(a)	Value to Reg(a+1)	…	Value to Reg(a+n)	I2C Stop

Figure 3 - Register Write Format.

4. REPORT PAGE REGISTERS

ST1232/ST1332 provides a register set for host to configure device attributes and retrieve information about fingers, proximity, gestures or raw data through device host interface. Host interface registers are listed below.

Host Interface Registers (Report Page)									
Reg. Addr.	**Name**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0x00	**Firmware Version**	**Version (RO)**							
0x01	**Status Reg.**	**Error Code (RO)**				**Device Status (RO)**			
0x02	**Device Control Reg.**	**Auto Tune (RW)**	***Flash Update Disable (RW)***	**Data Mode (RW)**		**Gest. Enable (RW)**	**Proximity Enable (RW)**	**Power Down (RW)**	**Reset (RW)**
0x03	**Timeout to Idle Reg.**	**Timeout to Idle (sec.) (RW)**							
0x04	**XY Resolution (High Byte)**	**X_Res_H (RW)**				**Y_Res_H (RW)**			
0x05	**X Resolution (Low Byte)**	**X_Res_L (RW)**							
0x06	**Y Resolution (Low Byte)**	**Y_Res_L (RW)**							
0x07	**Max Drift Threshold**	**Max Drift Threshold (RW)**							
0x08	**Touch Threshold (High Byte)**	**Touch_TH_H (RW)**							
0x09	**Touch Threshold (Low Byte)**	**Touch_TH_L (RW)**							
0x0A	**Noise Threshold**	**Noise_TH (RW)**							
0x0B	**Key Threshold**	**Key_TH (RW)**							
0x0C	**Firmware Revision 3**	**FW_Rev_3**							
0x0D	**Firmware Revision 2**	**FW_Rev_2**							
0x0E	**Firmware Revision 1**	**FW_Rev_1**							
0x0F	**Firmware Revision 0**	**FW_Rev_0**							
0x10	**Fingers / Gesture**	**Gesture Code (RO)**					**Fingers (RO)**		
0x11	**Keys Reg.**	**Keys (RO)**							
0x12	**XY0 Coord. (High Byte)**	**Valid 0 (RO)**	**X0_H (RO)**			***Reserved***	**Y0_H (RO)**		

Host Interface Registers (Report Page)									
Reg. Addr.	**Name**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0x13	**X0 Coord. (Low Byte)**	**X0_L (RO)**							
0x14	**Y0 Coord. (Low Byte)**	**Y0_L (RO)**							
0x15	**XY1 Coord. (High Byte)**	**Valid 1 (RO)**	**X1_H (RO)**			***Reserved***	**Y1_H (RO)**		
0x16	**X1 Coord. (Low Byte)**	**X1_L (RO)**							
0x17	**Y1 Coord. (Low Byte)**	**Y1_L (RO)**							
0x18	**Z0 Coord.**	**Z0 (RO)**							
0x19	**Z1 Coord.**	**Z1 (RO)**							
0x1A … 0x3F	**…**	***Reserved***							
0x40	**Raw Data [0] (High Byte)**	**Data_H [0]**							
0x41	**Raw Data [0] (Low Byte)**	**Data_L [0]**							
0x42	**Raw Data [1] (High Byte)**	**Data_H [1]**							
0x43	**Raw Data [1] (Low Byte)**	**Data_L [1]**							
0x44	**Raw Data [2] (High Byte)**	**Data_H [2]**							
0x45	**Raw Data [2] (Low Byte)**	**Data_L [2]**							
…	**…**	**…**							
0x7E	**Raw Data[31] (High Byte)**	**Data_H [31]**							
0x7F	**Raw Data [31] (Low Byte)**	**Data_L [31]**							
0x80 … 0xFE	**…**	***Reserved***							
0xFF	**Page Reg.**	**Page Number (RW)**							

Figure 4 - Host Interface Registers

4.1. Firmware Version Register

Reg. Addr.	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x00	Firmware Version	Version (RO)							

Firmware Version Register provides version information about current firmware. Host application can support version control in firmware upgrade function by reading *Firmware Version Register* and comparing with the version of new firmware binary.

4.2. Status Register

Reg. Addr.	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x01	Status Reg.	Error Code (RO)				Device Status (RO)			

Status Register shows current status of the device to host, including *Device Status* and *Error Code*. *Init* status represents that the device is in *Init* state and not ready for host access. Host has to wait for the device to change into *Normal* state before accessing registers other than *Status Register*. If *Device Status* shows *Error*, the *Error Code* field in the *Status Register* gives reason of the error.

Device Status	
0x0	Normal
0x1	Init
0x2	Error
0x3	Auto Tuning
0x4	Idle
0x5	Power Down
0x6 … 0xF	*Reserved*

Error Code	
0x0	No Error
0x1	Invalid Address
0x2	Invalid Value
0x3	Invalid Platform
0x4 … 0xF	*Reserved*

4.3. Device Control Register

Reg. Addr.	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x02	Device Control Reg.	Auto Tune (RW)	*Flash Update Disable (RW)*	Data Mode (RW)		Gest. Enable (RW)	Proximity Enable (RW)	Power Down (RW)	Reset (RW)

Device Control Register provides device control bits for host to reset the device, power down the device, enable/disable proximity detection, enable/disable gestures or data mode. Power Down state will be updated to Device Status field of Status Register, 0x01, after setting/clearing Power Down bit. Set Data Mode to 0x1 for Raw Data mode. Set Data Mode to 0x02 for Delta mode. Set Auto Tune to 0x1 will enable Auto Tune. Set Flash Update Disable to 0x00 will write the Auto Tune's result to flash.

4.4. Proximity Enable and Timeout to Idle Register

Reg. Addr.	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x03	Timeout to Idle Reg.	Timeout to Idle (sec.) (RW)							

Timeout to Idle Register provides timeout control to enter Idle Mode for host. The touch controller will enter Idle Mode after the number of seconds specified in Timeout to Idle Register if there is no touch detected in this period. Set this field to 0xFF will disable Idle Mode. Set this field to 0 will entering Idle Mode immediately. Idle state will be updated to Device Status field of Status Register, 0x01, after entering Idle Mode automatically. The default value of Timeout to Idle Register is set to 0x08 for 8 seconds to Idle Mode.

4.5. XY Resolution Registers

Reg. Addr.	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x04	XY Resolution (High Byte)	X_Res_H (RW)				Y_Res_H (RW)			
0x05	X Resolution (Low Byte)	X_Res_L (RW)							
0x06	Y Resolution (Low Byte)	Y_Res_L (RW)							

XY Resolution Registers represents resolution of X and Y coordinates of the touch screen. Host can change XY Resolution at run time by updating new resolution to these registers.

4.6. Threshold Setting

Reg. Addr.	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x07	Max Drift Threshold	Max Drift Threshold (RW)							
0x08	Touch Threshold (High Byte)	Touch_TH_H (RW)							
0x09	Touch Threshold (Low Byte)	Touch_TH_L (RW)							
0x0A	Noise Threshold	Noise_TH (RW)							
0x0B	Key Threshold	Key_TH (RW)							

Max Drift Threshold field defines the largest allowable drift in reported coordinates before issuing a new interrupt. Setting this field to 0 will disable Max Drift Threshold function. The default setting is 0.

Touch Threshold, *Noise Threshold* and *Key Threshold* define threshold of touch event detecting for Touch, Noise and Key sensors.

4.7. Firmware Revision Registers

Reg. Addr.	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x0C	Firmware Revision 3	FW_Rev_3							
0x0D	Firmware Revision 2	FW_Rev_2							
0x0E	Firmware Revision 1	FW_Rev_1							
0x0F	Firmware Revision 0	FW_Rev_0							

Firmware Revision Registers provide revision information about current firmware.

4.8. Fingers and Gesture Register

Reg. Addr.	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x10	Fingers / Gesture	Gesture Code (RO)					Fingers (RO)		

Fingers field represents number of fingers detected by touch controller. The coordinates of each finger detected are represents in X Coordinate and Y Coordinate fields. *Gesture Register* tells host which gesture is detected by the controller. Gesture Codes for each gesture are listed below.

Gesture Code	
0x00	No Detected
0x01	Single Touch Tap
0x02	Single Touch Double Tap
0x03	Single Touch Slide Up
0x04	Single Touch Slide Down
0x05	Single Touch Slide Left
0x06	Single Touch Slide Right
0x07	Two Finger Slide Up
0x08	Two Finger Slide Down
0x09	Two Finger Slide Left
0x0A	Two Finger Slide Right
0x0B	Pinch In (Zoom In)
0x0C	Pinch Out (Zoom Out)
0x0D	Rotate CW (CCW, for Top Down Mapping)
0x0E	Rotate CCW (CW, for Top Down Mapping)
0x0F	Object Approaching
0x10	Object Leaving
0x11 … 0x1F	*Reserved*

4.9. Keys Register

Reg. Addr.	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x11	Keys	Keys (RO)							

Key field represents which key is pressed or released. Each bit in the *Key* field represents the pressed or released state of one key. If the bit is set, it means that the corresponding key is pressed. Otherwise, the key is released.

4.10. XY Coordinate Registers

Reg. Addr.	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x12	XY0 Coord. (High Byte)	Valid 0 (RO)	X0_H (RO)			*Reserved*	Y0_H (RO)		
0x13	X0 Coord. (Low Byte)	X0_L (RO)							
0x14	Y0 Coord. (Low Byte)	Y0_L (RO)							

XY Coordinate Registers represent the XY coordinates for each touch point ID. Valid bit field tells that this point ID is valid and the XY information represents a real touch point on touch sensor.

4.11. Z Coordinate Register

Reg. Addr.	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x18	Z0 Coord.	Z0 (RO)							
0x19	Z1 Coord.	Z1 (RO)							

Z Coordinate Register indicates the touch strength of corresponding touch point ID. *Z0* represents touch strength of point ID 0 and *Z1* represents touch strength of point ID 1.

4.12. Data Registers

Reg. Addr.	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x40	Data [0] (High Byte)	Data_H [0]							
0x41	Data [0] (Low Byte)	Data_L [0]							
0x42	Data [1] (High Byte)	Data_H [1]							
0x43	Data [1] (Low Byte)	Data_L [1]							
0x44	Data [2] (High Byte)	Data_H [2]							
0x45	Data [2] (Low Byte)	Data_L [2]							
…	…	…							
0x7E	Data[31] (High Byte)	Data_H [31]							
0x7F	Data [31] (Low Byte)	Data_L [31]							

Data Registers provide raw or delta data detected by touch sensor controller. If Data Mode of Device Control Register (0x02) is set to Raw Mode, Data Registers represent raw data. If Data Mode is set to Delta Mode, Data Registers represent delta data. Data Registers will be updated for each scan frame when in raw or delta mode. Otherwise, Raw Data Registers will not be updated.

4.13. Page Register

Reg. Addr.	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0xFF	Page Reg.	Page Number (RW)							

Page Register provides changing page of Host Interface Register. Default page is Report Page.

Page Number	Description
0x00	Report Page
0x01	AutoTune Page

5. FIRMWARE UPDATE

Firmware update is enabled by writing an 8 bytes sequence of pattern to the register address, 0x00, of Report Page. The firmware update pattern is 0x53, 0x54, 0x58, 0x5F, 0x46, 0x57, 0x55 and 0x50 (ASCII: STX_FWUP). After writing the firmware update pattern to touch screen controller, the controller will switch to In System Programming (ISP) mode.

5.1. ISP Protocol Introduction

ISP mode provides several commands about flash erase/read/write and state checking. ISP command and data packets between host and touch screen controller are composed of 8 bytes. When host writes information to touch screen controller, the 1st byte of packet is command type, and the other bytes are parameters or data.

ISP operations on flash are page programming which is based on 512-bytes page size. The touch screen controller totally has 32 pages (0 to 31).

Command Type	Name	Description
0x80	**ISP_ERASE (H)**	**Flash page erase.**
0x81	**ISP_SEND_DATA (H)**	**Send data to touch screen controller for ISP_WRITE_FLASH command.**
0x82	**ISP_WRITE_FLASH (H)**	**Write one page to flash.**
0x83	**ISP_READ_FLASH (H)**	**Read one page from touch screen controller.**
0x84	**ISP_RESET (H)**	**Reset touch screen controller.**
0x8F	**ISP_READY (D)**	**Indicates touch screen controller is ready.**

Figure 5-1 - Command Summary.

Protocol Flow describes ISP command flow for Page Erase, Page Program and Page Read of flash data and Reset of touch screen controller.

Command	Description
Flash Page Erase	**Start -> ISP_ERASE (H) -> ISP_READY (D) -> End.**
Flash Page Program	**Start -> ISP_WRITE_FLASH (H) -> ISP_SEND_DATA (H) x 74(include 512-bytes valid data) -> ISP_READY (D) -> End.**
Flash Page Read	**Start -> ISP_READ_FLASH (H) -> 8-bytes package x 64 (totally 512-bytes data) (D) -> End.**
Reset Controller	**Start -> ISP_RESET (H) -> End.**

Figure 5-2 - Protocol Flow.

Command/data direction:

H: Host -> Device.

D: Device -> Host.

5.2. ISP Protocol Format

5.2.1. ISP Erase

ISP_ERASE (H)								
Byte	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0	Command Type (0x80)							
1	*Reserved*							
2	PAGE_NUMBER (0~31)							
3	*Reserved*							
…								
7								

Touch screen controller will send ISP_READY command as below in response to this command.

ISP_READY (D)								
Byte	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0	Command Type (0x8F)							
1	*Reserved*							
…								
7								

5.2.2. ISP Send Data

ISP_SEND_DATA (H)								
Byte	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0	Command Type (0x81)							
1	Data							
…								
7								

5.2.3. ISP Write Flash

ISP_WRITE_FLASH (H)								
Byte	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0	Command Type (0x82)							
1	*Reserved*							
2	PAGE_NUMBER (0~31)							
3	*Reserved*							
4	Low Byte Checksum of 512-bytes Page							
5	High Byte Checksum of 512-bytes Page							
6	Data							
7								

After sending ISP_WRITE_FLASH command to touch screen controller, host must send ISP_SEND_DATA (W) x 74 (include 512-bytes valid data) to the controller. ISP will write 512-bytes sent from host to flash automatically once the ISP_SEND_DATA (W) x 74 command is completed. Finally, touch screen controller will return ISP_READ (R) of the format described below.

ISP_READY (D)								
Byte	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0	**Command Type (0x8F)**							
1	***Reserved***							
2	***Reserved***			**NML/ERR**	***Reserved***		**ERR_CODE**	
3	***Reserved***							
4	**Low Byte Checksum of 512-bytes Page Received From Host**							
5	**High Byte Checksum of 512-bytes Page Received From Host**							
6	***Reserved***							
7								

NML/ERR:

0: Normal

1: Error

ERR_CODE:

BIT 0: Flash error flag

0: Flash correct

1: Flash error

BIT 1: Interface transfer flag, check with 16-bit checksum

0: Interface transfer correct

1: Interface transfer error

5.2.4. ISP Read Flash

ISP_READ_FLASH (H)								
Byte	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0	**Command Type (0x83)**							
1	***Reserved***							
2	**PAGE_NUMBER (0~31)**							
3	***Reserved***							
…								
7								

Touch screen controller will send 512 bytes (8-bytes packet x 64) to host in response to ISP_READ_FLASH command.

5.2.5. ISP Reset

ISP_RESET								
Byte	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0	**Command Type (0x84)**							
1	**Reserved**							

ISP_RESET								
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
…								
7								

6. SAMPLE CODES

6.1. Data Structures and APIs

```
typedef struct {
        u8    y_h: 3,
              reserved: 1,
              x_h: 3,
              valid: 1;
        u8    x_l;
        u8    y_l;
} xy_data_t;

typedef struct {
        u8          fingers: 3,
                    gesture: 5;
        u8          keys;
        xy_data_t xy_data[2];
} stx_report_data_t;

// I2C Master sends count bytes data stored in buf to I2C Slave.
// I2C package: | S | I2C Addr | W | Data (buf) | P |
extern int i2c_master_send(const char *buf, int count);

// I2C Master reads count bytes data to buf from I2C Slave.
// I2C package: | S | I2C Addr | R | Data (buf) | Nak | P |
extern int i2c_master_recv(char *buf, int count);
```

6.2. Get Version

```
static int get_fw_version(u32 *ver)
{
        u8 buf[1];
        int ret = 0;

        buf[0] = 0x0;                       // Set Reg. address to 0x0 for reading FW Version.
        if (ret = i2c_master_send(buf, 1))
                goto err;
        if (ret = I2c_master_recv(buf, 1))  // Read 1 byte FW Version from Reg. 0x0 set previously.
                goto err;

        *ver = (u32) buf[0];                // Return FW Version.

        buf[0] = 0x10;                      // Set Reg. address back to 0x10 for Coordinates.
        if (i2c_master_send(buf, 1))
                goto err;

err:
        return ret;
}
```

6.3. Set Power Down (PD)

```
static int set_power_down()
{
        u8 buf[2];
        int ret = 0;

        buf[0] = 0x2;                       // Set Reg. address to 0x2 for Device Control Reg.
        buf[1] = 0xA;                       // Keep Gesture bit and set PD bit to enter Power Down.
        if (ret = i2c_master_send(buf, 2))
                goto err;
        buf[0] = 0x10;                      // Set Reg. address back to 0x10 for Coordinates.
        if (i2c_master_send(buf, 1))
```

```
                goto err;
err:
        return ret;
}
```

6.4. Set XY Resolution

```
static int set_resolution(u16 x_res, u16 y_res)
{
        u8 buf[4];
        int ret = 0;

        buf[0] = 0x4;                           // Set Reg. address to 0x4 for XY Resolution Registers.
        buf[1] = ((x_res & 0xF00) >> 4) | ((y_res & 0xF00) >> 8);  // Set X_Res_H and Y_Res_H.
        buf[2] = x_res & 0xFF;                  // Set X_Res_L.
        buf[3] = y_res & 0xFF;                  // Set Y_Res_L.
        if (ret = i2c_master_send(buf, 4))
                goto err;
        buf[0] = 0x10;                          // Set Reg. address back to 0x10 for Coordinates.
        if (i2c_master_send(buf, 1))
                goto err;
err:
        return ret;
}
```

6.5. Read XY Coordinates

The function, get_coordinates(), reads XY Coordinate registers from I2C Slave, extracts XY information from data buffer and returns to upper layer. This function shall be called from ISR each time when host receives and INT from device.

```
static int get_coordinates(u8 *count, u32 *x0, u32 *y0, u32 *x1, u32 *y1)
{
        u8 buf[8];
        stx_report_data_t *pdata;
        int ret = 0;
        *count = 0;                             // Set point detected count to 0.
        if (i2c_master_recv(buf, 8))            // Read Coordinates from default Reg. address 0x10.
                goto err;

        pdata = (stx_report_data_t *) buf;
        if (pdata->fingers) {
                if (pdata->xy_data[0].valid) {
                        *x0 = pdata->xy_data[0].x_h << 8 | pdata->xy_data[0].x_l;
                        *y0 = pdata->xy_data[0].y_h << 8 | pdata->xy_data[0].y_l;
                        (*count)++;
                }
                if (pdata->xy_data[1].valid) {
                        *x1 = pdata->xy_data[1].x_h << 8 | pdata->xy_data[1].x_l;
                        *y1 = pdata->xy_data[1].y_h << 8 | pdata->xy_data[1].y_l;
                        (*count)++;
                }
        }
err:
        return ret;
}
```