

JTW6C20

20 键电容式触摸芯片

目录

- 一、概述
- 二、特点
- 三、产品应用
- 四、管脚说明
- 五、电气性能
- 六、功能描述
- 七、特别说明
- 八、示范例程
- 九、应用电路及 **layout** 注意事项
- 十、封装尺寸

一、概述

电容式触摸感应 IC 是为实现人体触摸界面、替代机械式轻触开关而设计的，具有防水防尘、密封隔离、坚固美观的操作界面等优点。JTW6C20 提供 20 个触摸按键并给客户一个简易设定的按键应用方案、外围电路简单、操作方便，客户只要使用 IIC 通讯格式，即可设定并读取独立按键触摸数据。对于防水和抗干扰方面有很优异的表现。

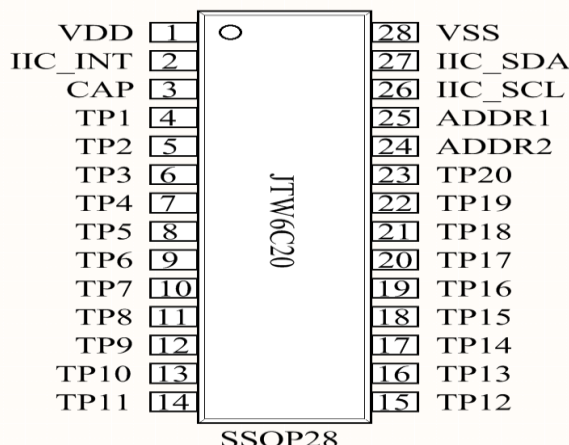
二、特点

- 工作电压范围宽：1.8V – 5.5V，内建稳压电路给触摸使用，提供稳定、可靠的工作环境；
- 工作电流小：可设置为省点模式，适合需长时间待机的应用，如遥控器等。
工作电流 3mA (@VDD=3.3V, 无负载)
待机电流 $\leq 20\mu\text{A}$ (@VDD=3.3V, 无负载, 低功耗模式)；
- 触摸响应快；
- 抗干扰性能强。高抗干扰性，近距离、多角度手机干扰的情况下，触摸响应灵敏度及可靠性不受影响；
- 具有防水及水漫成片水珠覆盖在触摸按键面板，按键仍可有效判别；
- 触摸按键数多。提供 20 个触摸按键供客户依照需求进行规划设计，可同时输出所有触摸按键。
- 触摸灵敏度调整灵活。可以经由 CAP 脚的外接电容来调整灵敏度，电容越大越灵敏度越高。也可通过 IIC 单独设置每一个按键的灵敏度（按键阈值）；
- 面板材料多样化：绝缘材料。如普通玻璃、有机玻璃、钢化玻璃、塑胶等。面板厚度：亚克力材质 0-10mm，不同材质的面板厚度有所差异；
- 封装：SSOP28。

三、产品应用

- 大小家电
- 门禁监控设备
- 消费类电子
- 数码产品

四、封装脚位图及说明



管脚描述

管脚	名称	类型	功能描述
1	VDD	P	电源正端
2	IIC_INT	0	IIC中断脚
3	CAP	AI	电容必须使用NP0或X7R材质电容 使用范围：2200pF-33000pF，电容越大灵敏度越高
4	TP1	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
5	TP2	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
6	TP3	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
7	TP4	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
8	TP5	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
9	TP6	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
10	TP7	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
11	TP8	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
12	TP9	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
13	TP10	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
14	TP11	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
15	TP12	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
16	TP13	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
17	TP14	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
18	TP15	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
19	TP16	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
20	TP17	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
21	TP18	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
22	TP19	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
23	TP20	AI	触摸按键脚，串接100-1000欧姆，能提高抗干扰和提高抗静电能力
24	ADDR2	I	地址脚选项2，默认高电平。
25	ADDR1	I	地址脚选项1，默认高电平。
26	IIC_SCL	I	IIC串行时钟线
27	IIC_SDA	I/O	IIC串行数据线
28	VSS	P	电源负端

* AI---模拟量输入； P---电源

五、电气特性

极限参数

参数	最小值	最大值	单位
直流供电电压	-0.3	6	V
I/O引脚输入电压	-0.3	VDD+0.3	V
工作环境温度	-40	85	°C
储存温度	-45	125	°C

备注：超过“极限参数”范围有可能对芯片造成损坏，无法预期芯片在上述范围的工作状态，若长期在标识范围 外工作，可能会影响芯片的可靠性。

直流电气性能

芯片参数	符号	最小值	典型值	最大值	单位	条件
工作电压	VDD	1.8	3.3	5.5	V	
工作电流		-	-	10	uA	
		4	-	6	mA	
I/O端口输入高电压	V _{hi}	-	0.7*V _{dd}	-	V	VDD=1.8~5.5V
I/O端口输入低电压	V _{lo}	-	0.2*V _{dd}	-	V	VDD=1.8~5.5V
I/O端口推电流	I _{pu}	-	6.05	-	mA	VDD=3.3V
		-	8.46	-		VDD=5V
I/O端口灌电流	I _{ol}	-	19.05	-	mA	VDD=3.3V
		-	70	-		VDD=5V

备注：最小值、典型值和最大值的数据测量条件：VDD=1.8V，TA=25℃，除非另有说明。

六、功能描述

触摸按键介绍：

触摸按键是利用测量人体接近导体时产生的电容变化，转换为数值判断的一种方式。此应用中所有的触摸按键都有 Threshold 设定参数，用来调整触摸按键的灵敏度。

Threshold 依照按键的按压深度来做调整，数值越小越灵敏，但也越容易受到噪声干扰，并依照实际按压读取的数据来调整。

Switching Characteristics

Symbol	Description	Min	Max	Units
FSCL	SCL clock frequency.	0	100	KHz
THDSTA	Hold time(repeated) star condition.	4	-	us
	After this period, the first clock pulse is generated.			
TLOW	Low period of the SCL clock.	4.7	-	us
THIGH	High period of the SCL clock.	4	-	us
TSUSTA	Set-up time for a repeated start condition.	4.7	-	us
THDDAT	Data hold time.	0	-	us
TSUDAT	Data set-up time.	250	-	ns
TSUSTO	Set-up time for stop condition.	4	-	us
TBUF	Bus free time between a stop and start condition.	4.7	-	us
TSPI	Pulse width of spikes are suppressed by the input filter.	0	50	ns
TSPT	Slave processor time	10	75	us

Table3. AC characteristics of the IIC SDA and SCL pins for VDD

Timing Waveform

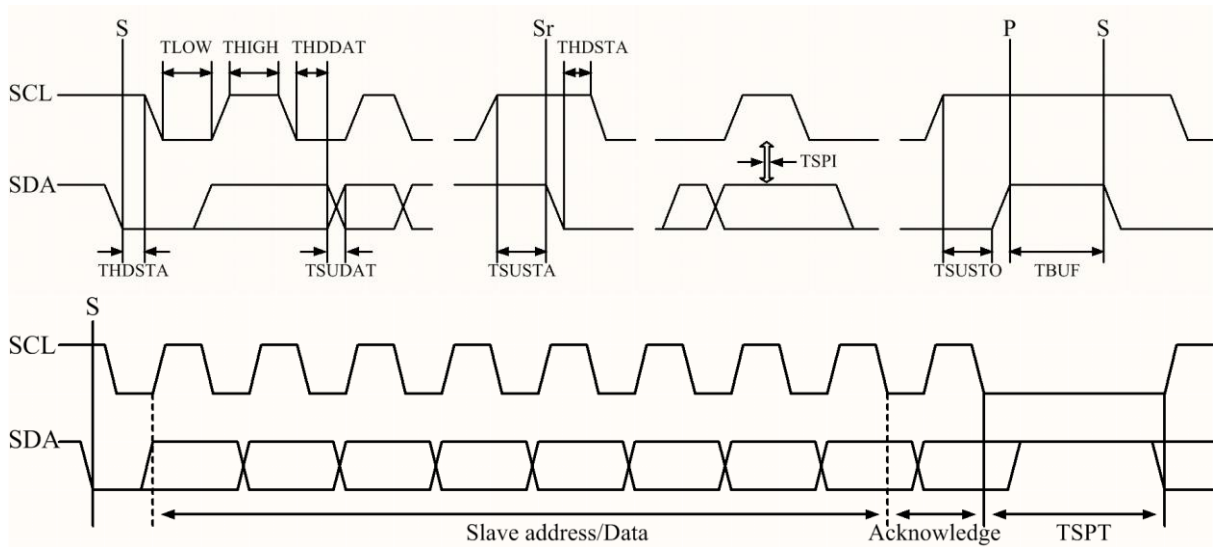


Figure8. Definition for timing for fast/standard mode on the I2C

Packet Stream

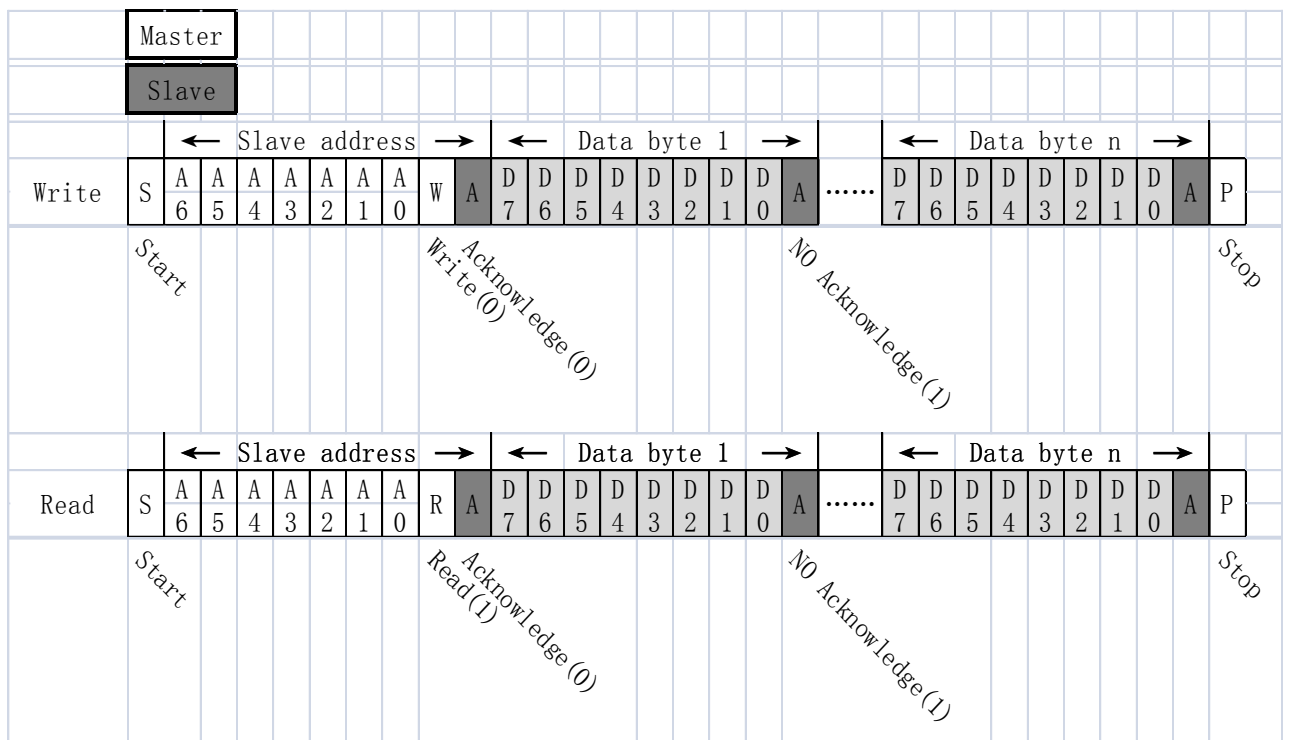


Figure9. Write / Read byte form I2C

Slave address

Slave address(A6-A0): 53H; Write(A6-A0,R): A6H; Read(A6-A0,R): A7H

ADDR2	ADDR1	Slave address (A6-A0)	Write (A6-A0, W)	Read (A6-A0, R)
0	0	50H	A0H	A1H
0	1	51H	A2H	A3H
1	0	52H	A4H	A5H
1	1	53H(默认)	A6H	A7H

Application mode

Write Data

1. Setting commands

Data byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	1	CT=0			PSM			
2								
3								

CT

写入数据区分成应用设定以及阈值设定，当 CT 为 0 时是写入应用设定，当 CT=1 时是写入阈值设定。

CT	Custom Threshold
0	Setting commands
1	Custom threshold commands

PSM

省电模式，无按键 5 秒后进入睡眠模式。

PSM	Power Save Mode
0	Disable
1	Enable (Define)

2. Custom threshold commands

阈值则是设定按键承认的门坎。分为按键阈值，睡眠唤醒阈值两种。

Item

选择切换不同的写入参数的设定。

Item	Item
0	TPx setting
1	Sleep setting

● TPx Setting

Data byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	1	CT=1	Item	TP Num				
2	TPx Threshold L							
3	TPx Threshold H							

TPx Threshold：按键承认阈值。(Define：0x3C)

按键承认阈值越小灵敏度越高，越大灵敏度越低。预设的阈值为 0x3C，建议的最小值为 0x1E，若调整到 0x1E 按键灵敏度仍然不够，则建议加大 CS 电容，CS 电容的值则建议小于 39nF。

Read Data

Data byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	1	WSET						
2	Key 8	Key 7	Key 6	Key 5	Key 4	Key 3	Key 2	Key 1
3	Key 16	Key 15	Key 14	Key 13	Key 12	Key 11	Key 10	Key 9
4					Key 20	Key 19	Key 18	Key 17

WSET

系统写入标志，上电为 0，写入设定后该标志设置为 1。

WSET	Have write setting
1	Have write setting
0	No write setting

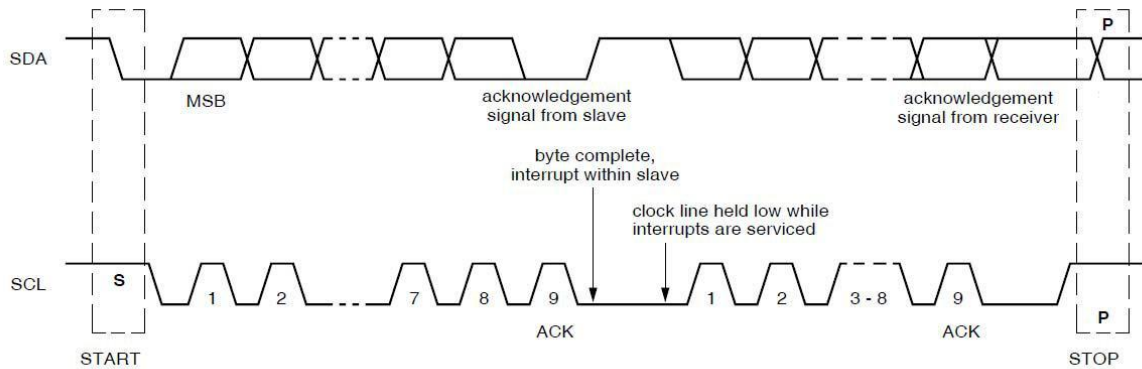
Key1...Key12

触摸按键标志，无按键为 0，有按键为 1。

Key	Key x Touch
0	No touch
1	Touch

七、特别说明:

1. JTW6C20 的 I2C 界面有硬件的支持 SCL 可支持 100KHz, 但是译码为软件处理, 所以当 Master 的第 9 个 SCL 为 Low 时, JTW6C20 会马上将 SCL 的 bus 拉 Low, 表示 JTW6C20 进入 busy 的状态, 同时 JTW6C20 内部会产生中断处理 I2C 的解码, 处理约需 20~100us 视处理的情况而定, 等处理完就会释放 SCL, 一般主控的 SCL 控制脚为 Nmos 的输出, 需外加上拉电阻, 以免主控无法将 SCL 拉 High。



所以 Master 写程序时, 需注意 SCL 拉 Low 的动作, 若由硬件控制大多会支持此标准, 若由程序控制 IO 脚, 请增加对 SCL 输出 High 时要读回确认为 High, 才让程序继续进行, 若为 Low 应等待 SCL 为 High 后才可继续进行。C 的程序如下:

```
SCL=1;
While(SCL!=1) { };
```

2. 若需要连续读取键值, 建议读取完后暂停 10ms 以上, 再读取下一次键值。否则会影影响按键 的反应速度。
3. 若开启睡眠模式, 则禁止连续读取键值, 因为每次读取键值时, 都会清除进入睡眠的计时, 会导致系统无法睡眠。
4. 在系统进入睡眠模式时, 会将 IIC 功能关闭。系统唤醒后才能进行 IIC 读写操作。
5. 按键阈值调整的步骤:

- Step1. 选择初始测试用的 CS 电容(见建议线路图): 建议使用 10nF 作为初始测试电容。
- Step2. 每个按键做按压测试: 以正常速度轻触按键或使用金属棒做测试条件, 若在触摸到按键之前有按键输出, 表示灵敏度太高, 需要调高阈值, 若触摸按键没有按键输出, 或是要重压才有按键输出, 表示灵敏度太低, 需要降低阈值。
- Step3. 测试按键反应速度: 在判断按键灵敏度的时候, 若觉得按键”不够灵敏”, 需要进一步判断是按键响应速度不够快, 还是按键灵敏度不够。判断方法是触摸停留一段时间(约 1 秒), 并检查是否有按键输出。若没有按键输出, 则是按键不够灵敏, 重新进行 Step2 调整, 若有按键输出, 则是按键响应速度不够快, 则进行下一步。

八、示范程序:

```
/*
 项目名称: STM32F103C8T6 软件模拟 I2C 参考程序
 项目目的: 1. 透过软件模拟 IIC 主控端对 JTW6C20 写入设定参数
           2. 透过软件模拟 IIC 主控端对 JTW6C20 读取按键状态
 主控 MCU: STM32F103C8T6

  Date & Version: 2018/09/03_V1.0
*/
#define      pTTP_SCK      PAout(5)
#define      pTTP_SDA      PAout(6)
#define      pTTP_INT_IN   PAin(7)
#define      pTTP_SCK_IN   PAin(5)
#define      pTTP_SDA_IN   PAin(6)
#define      W_ADDR        0XA6
#define      R_ADDR        0XA7
u8 gBuF[4];
//-----
//读取键值
//第一字节: 0x80 未配置过参数  0xC0 已配置过参数
//第二字节: bit7-bit0: 对应触摸按键 TP8-TP1
//第三字节: bit7-bit0: 对应触摸按键 TP16-TP9
//第四字节: bit3-bit0: 对应触摸按键 TP20-TP17
void TTP_Server(void)
{
    if(pTTP_INT_IN == 0)
    {
        TTP_ReadData(R_ADDR, gBuF, 4);
    }
}
//-----
void TTP_I2C_Init(void)
{
    GPIO_InitTypeDef  GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

    //32 pa7-int
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

```

//31 pa5-sck  33 pa6-sda
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5+GPIO_Pin_6;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_OD;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &GPIO_InitStructure);
pTTP_SDA = 1;
pTTP_SCK = 1;
TTP_I2C_Delay();
/*
  上电初始化写入 3 类参数，可设置每个按键的灵敏度，唤醒灵敏度，休眠设置
  按键灵敏度命令：0xC0-0xCB，第 2, 3 个参数,16BIT 灵敏度，低字节在前，高
  字节在后。
  唤醒灵敏度命令：0xE0, 第 2, 3 个参数,16BIT 灵敏度，低字节在前，高字节在
  后。
  休眠开启命令：    (0x88, 0x00, 0x00)
  休眠关闭命令：    (0x80, 0x00, 0x00)
  上电不写入参数：默认按键灵敏度为 60，唤醒灵敏度为 10，休眠开启
*/
setBuF(0xc0, 60, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP1  Threshold

setBuF(0xc1, 61, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP2  Threshold

setBuF(0xc2, 62, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP3  Threshold

setBuF(0xc3, 63, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP4  Threshold

setBuF(0xc4, 64, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP5  Threshold

setBuF(0xc5, 65, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP6  Threshold

setBuF(0xc6, 66, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP7  Threshold

setBuF(0xc7, 67, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP8  Threshold

setBuF(0xc8, 68, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP9  Threshold

```

```
setBuF(0xc9, 69, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP10 Threshold

setBuF(0xca, 70, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP11 Threshold

setBuF(0xcb, 71, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP12 Threshold

setBuF(0xcc, 71, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP13 Threshold

setBuF(0xcd, 71, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP14 Threshold

setBuF(0xce, 71, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP15 Threshold

setBuF(0xcf, 71, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP16 Threshold

setBuF(0xd0, 71, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP17 Threshold

setBuF(0xd1, 71, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP18 Threshold

setBuF(0xd2, 71, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP19 Threshold

setBuF(0xd3, 71, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write TP20 Threshold

setBuF(0xe0, 11, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //Write WakeUp Threshold

//setBuF(0x80, 0x00, 0x00);
//TTP_WriteData(W_ADDR, gBuF, 3);           //SLEEP Disable

setBuF(0x88, 0x00, 0x00);
TTP_WriteData(W_ADDR, gBuF, 3);           //SLEEP Enable
```

```
/*
  读命令是为了验证写入参数是否 OK
  读完后，写入 0x5a, 0x00, 0x00, 结束读取参数。
  setBuF(0x5a, 0x00, 0x00);           //Close   Read CMD
  TTP_WriteData(W_ADDR, gBuF, 3);
*/
/*
  setBuF(0x5a, 0xa5, 0xc0);           //Read TP1  Threshold
  TTP_WriteData(W_ADDR, gBuF, 3);
  TTP_ReadData(R_ADDR, gBuF, 3);

  setBuF(0x5a, 0xa5, 0xc1);           //Read TP2  Threshold
  TTP_WriteData(W_ADDR, gBuF, 3);
  TTP_ReadData(R_ADDR, gBuF, 3);

  setBuF(0x5a, 0xa5, 0xc2);           //Read TP3  Threshold
  TTP_WriteData(W_ADDR, gBuF, 3);
  TTP_ReadData(R_ADDR, gBuF, 3);

  setBuF(0x5a, 0xa5, 0xc3);           //Read TP4  Threshold
  TTP_WriteData(W_ADDR, gBuF, 3);
  TTP_ReadData(R_ADDR, gBuF, 3);

  setBuF(0x5a, 0xa5, 0xc4);           //Read TP5  Threshold
  TTP_WriteData(W_ADDR, gBuF, 3);
  TTP_ReadData(R_ADDR, gBuF, 3);

  setBuF(0x5a, 0xa5, 0xc5);           //Read TP6  Threshold
  TTP_WriteData(W_ADDR, gBuF, 3);
  TTP_ReadData(R_ADDR, gBuF, 3);

  setBuF(0x5a, 0xa5, 0xc6);           //Read TP7  Threshold
  TTP_WriteData(W_ADDR, gBuF, 3);
  TTP_ReadData(R_ADDR, gBuF, 3);

  setBuF(0x5a, 0xa5, 0xc7);           //Read TP8  Threshold
  TTP_WriteData(W_ADDR, gBuF, 3);
  TTP_ReadData(R_ADDR, gBuF, 3);

  setBuF(0x5a, 0xa5, 0xc8);           //Read TP9  Threshold
  TTP_WriteData(W_ADDR, gBuF, 3);
  TTP_ReadData(R_ADDR, gBuF, 3);
*/
```

```
setBuF(0x5a, 0xa5, 0xc9); //Read TP10 Threshold
TTP_WriteData(W_ADDR, gBuF, 3);
TTP_ReadData(R_ADDR, gBuF, 3);

setBuF(0x5a, 0xa5, 0xca); //Read TP11 Threshold
TTP_WriteData(W_ADDR, gBuF, 3);
TTP_ReadData(R_ADDR, gBuF, 3);

setBuF(0x5a, 0xa5, 0xcb); //Read TP12 Threshold
TTP_WriteData(W_ADDR, gBuF, 3);
TTP_ReadData(R_ADDR, gBuF, 3);

setBuF(0x5a, 0xa5, 0xcc); //Read TP13 Threshold
TTP_WriteData(W_ADDR, gBuF, 3);
TTP_ReadData(R_ADDR, gBuF, 3);

setBuF(0x5a, 0xa5, 0xcd); //Read TP14 Threshold
TTP_WriteData(W_ADDR, gBuF, 3);
TTP_ReadData(R_ADDR, gBuF, 3);

setBuF(0x5a, 0xa5, 0xce); //Read TP15 Threshold
TTP_WriteData(W_ADDR, gBuF, 3);
TTP_ReadData(R_ADDR, gBuF, 3);

setBuF(0x5a, 0xa5, 0xcf); //Read TP16 Threshold
TTP_WriteData(W_ADDR, gBuF, 3);
TTP_ReadData(R_ADDR, gBuF, 3);

setBuF(0x5a, 0xa5, 0xd0); //Read TP17 Threshold
TTP_WriteData(W_ADDR, gBuF, 3);
TTP_ReadData(R_ADDR, gBuF, 3);

setBuF(0x5a, 0xa5, 0xd1); //Read TP18 Threshold
TTP_WriteData(W_ADDR, gBuF, 3);
TTP_ReadData(R_ADDR, gBuF, 3);

setBuF(0x5a, 0xa5, 0xd2); //Read TP19 Threshold
TTP_WriteData(W_ADDR, gBuF, 3);
TTP_ReadData(R_ADDR, gBuF, 3);

setBuF(0x5a, 0xa5, 0xd3); //Read TP20 Threshold
TTP_WriteData(W_ADDR, gBuF, 3);
TTP_ReadData(R_ADDR, gBuF, 3);
```

```
    setBuF(0x5a, 0xa5, 0xe0);           //Read WakeUp  Threshold
    TTP_WriteData(W_ADDR, gBuF, 3);
    TTP_ReadData(R_ADDR, gBuF, 3);

    setBuF(0x5a, 0xa5, 0x80);           //Read SLEEP EN
    TTP_WriteData(W_ADDR, gBuF, 3);
    TTP_ReadData(R_ADDR, gBuF, 3);

    setBuF(0x5a, 0x00, 0x00);           //Close  Read CMD
    TTP_WriteData(W_ADDR, gBuF, 3);
*/
}
//-----
void TTP_I2C_Delay(void)
{
    delay_us(1);
}
//-----
void TTP_I2C_Start(void)
{
    pTTP_SDA = 1;
    pTTP_SCK = 1;
    while(pTTP_SCK_IN != 1) { };
    TTP_I2C_Delay();
    pTTP_SDA = 0;
    TTP_I2C_Delay();
    pTTP_SCK = 0;
}
//-----
void TTP_I2C_Stop(void)
{
    pTTP_SCK = 0;
    pTTP_SDA = 0;
    TTP_I2C_Delay();
    pTTP_SCK = 1;
    while(pTTP_SCK_IN != 1) { };
    TTP_I2C_Delay();
    pTTP_SDA = 1;
}
//-----
u8 TTP_I2C_ReadAck(void)
{
    pTTP_SCK = 0;
    pTTP_SDA = 1;
```

```
TTP_I2C_Delay();
pTTP_SCK = 1;
while(pTTP_SCK_IN != 1) { };
TTP_I2C_Delay();
if(pTTP_SDA_IN)
    return 1;
else
    return 0;
}
//-----
void TTP_I2C_SendAck(void)
{
    pTTP_SCK = 0;
    pTTP_SDA = 0;
    TTP_I2C_Delay();
    pTTP_SCK = 1;
}
//-----
void TTP_I2C_SendNoAck(void)
{
    pTTP_SCK = 0;
    pTTP_SDA = 1;
    TTP_I2C_Delay();
    pTTP_SCK = 1;
}
//-----
void TTP_I2C_SendByte(u8 sdata)
{
    u8 i;
    for(i=0; i<8; i++)
    {
        pTTP_SCK = 0;
        if(sdata & 0x80)
            pTTP_SDA = 1;
        else
            pTTP_SDA = 0;
        TTP_I2C_Delay();
        pTTP_SCK = 1;
        while(pTTP_SCK_IN != 1) { };
        TTP_I2C_Delay();
        sdata <<= 1;
    }
}
//-----
```

```
u8 TTP_I2C_ReadByte(void)
{
    u8 i, sdata;
    sdata = 0;
    for(i=0; i<8; i++)
    {
        pTTP_SCK = 0;
        pTTP_SDA = 1;
        TTP_I2C_Delay();
        sdata <<= 1;
        pTTP_SCK = 1;
        while(pTTP_SCK_IN != 1) { };
        TTP_I2C_Delay();
        if(pTTP_SDA_IN)
            sdata |= 0x01;
    }
    return sdata;
}
//-----
u8 TTP_WriteData(u8 addr, u8 *buf, u8 length)
{
    u8 i;
    TTP_I2C_Start();
    TTP_I2C_SendByte(addr);
    if(TTP_I2C_ReadAck())
    {
        TTP_I2C_Stop();
        return 1;
    }
    for(i=0; i<length; i++)
    {
        TTP_I2C_SendByte(buf[i]);
        if(TTP_I2C_ReadAck())
        {
            TTP_I2C_Stop();
            return 1;
        }
    }
    TTP_I2C_Stop();
    return 0;
}
//-----
```



```
u8 TTP_ReadData(u8 addr, u8 *buf, u8 length)
{
    u8 i;

    TTP_I2C_Start();
    TTP_I2C_SendByte(addr);

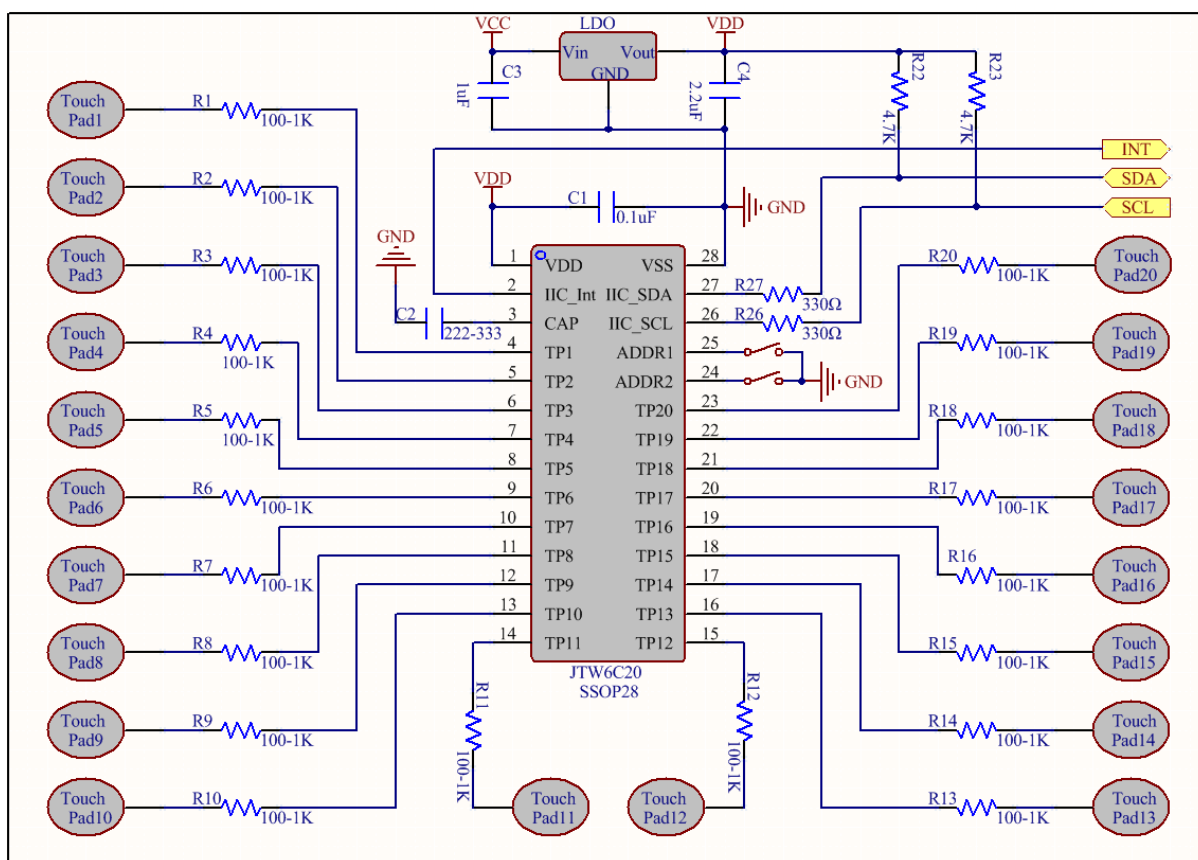
    if(TTP_I2C_ReadAck())
    {
        TTP_I2C_Stop();
        return 1;
    }

    for(i=0; i<length; i++)
    {
        buf[i] = TTP_I2C_ReadByte();

        if(i < length - 1)
            TTP_I2C_SendAck();
        else
            TTP_I2C_SendNoAck();
    }

    TTP_I2C_Stop();
    return 0;
}
//-----
void setBuF(u8 dat1, u8 dat2, u8 dat3)
{
    gBuF[0] = dat1;
    gBuF[1] = dat2;
    gBuF[2] = dat3;
}
```

九、应用电路及 layout 注意事项



- R1---R20: 串一个 100R-1K 的电阻是可以增强抗手机和对讲机的干扰, 一般情况是可以省略。
- R26、R27: 建议在靠近触摸 IC 串一个 330R-1K 的电阻, 保证信号的完整性, 从而保证系统工作稳定。
- R22、R23: 建议预留一个上拉电阻, 增强通信的抗干扰性。
- C2: 灵敏度调节电容, 根据具体产品去实际调试电容大小, 使之达到最佳的触摸效果。

Layout 注意事项:

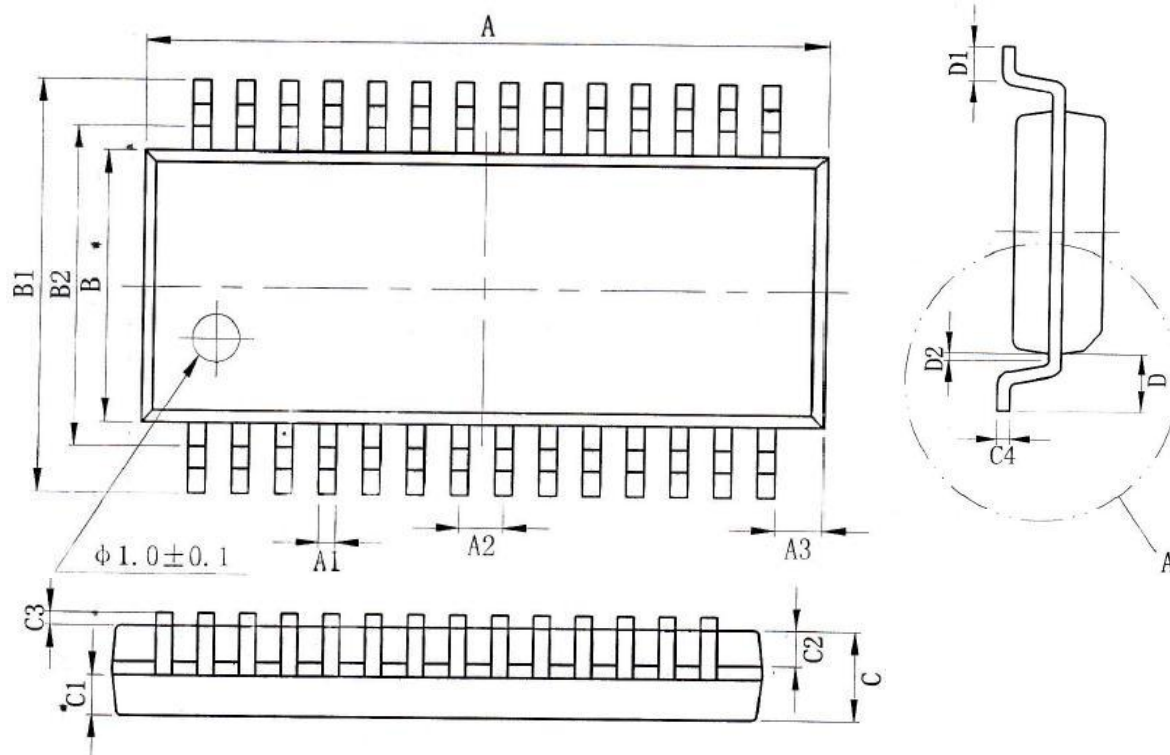
PCB Layout 最关键的两点是减少 PCB 的基准电容和避免干扰。

为了增大抗干扰性, 避免被其他干扰:

- 1、触摸 PAD 可以是任何形状, 建议使用规则形状, 大小在 8*8-15*15mm 之间;
- 2、触摸 PAD 周围敷铜接地。敷铜到触摸 PAD 的距离要大于等于 1mm, 建议在 2mm 间距最佳, 距离太近将会降低触摸的灵敏度。
- 3、触摸 PAD 底部不要走线。如果干扰环境较大, 可以在触摸 PAD 下面铺网格铜;
- 5、触摸 PAD 到芯片间的连线尽量短和细。连线宽度 0.15-0.2mm 为最佳, 连线周围敷铜, 连线到敷铜的距离保持在 0.5-1mm 之间。
- 6、触摸 PAD 到芯片间的连线底部尽量不要过线, 不能避免的情况下不要与触摸走线平齐, 过线宽度不要大于触摸连线的宽度、不能过高频和电源线。
- 7、电源的布线, 首先要以电路区块分割, 触摸芯片能有独立的走线到电源正端, 若无法独立的分支走线, 则尽量优先提供触摸电路后再接到其他电路。接地部分也相同, 希望能有独立的分支走线到电源的接地点, 也就是星型接地, 避免与其他电路的干扰。

十、封装尺寸

SSOP28



序号	最小值 (mm)	标准值 (mm)	最大值 (mm)
A	9.80	9.90	10.00
A1	---	0.254 TYP	---
A2	---	0.635 TYP	---
A3	---	0.695 TYP	---
B	3.85	3.90	3.95
B1	5.85	6.00	6.24
B2	---	5.00 TYP	---
C	1.40	1.50	1.60
C1	0.61	0.66	0.71
C2	0.54	0.59	0.64
C3	0.05	0.15	0.25
C4	0.203	0.215	0.233
D	---	1.05 TYP	---
D1	0.40	0.55	0.70
D2	0.15	0.20	0.25

修改记录

版本	更新日期	更新内容	修改人	确认人
V1.0	2017/9/3	初始版本	KarYung	William
V1.1	2018/9/5	优化灵敏度	KarYung	William