永嘉微电
Vinka Microelectronics

VK3804 Datasheet

Touch slider I2C output

Rev.1.1

1 General Description

This software provides customers with a simple and convenient application solution for slider buttons.
Customers can read the data of slider buttons and slider addresses by using the IIC communication format.

2 Key Features

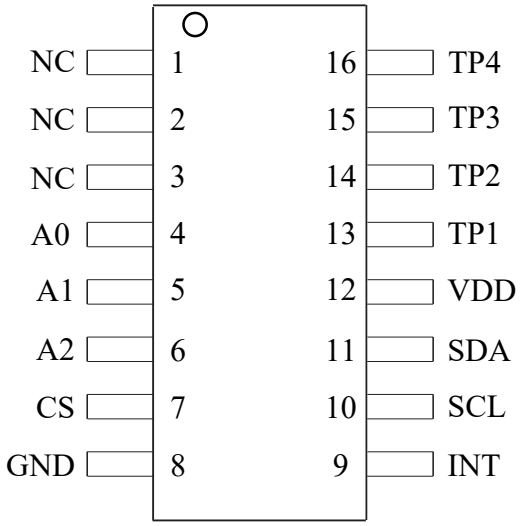
- This application uses four touch keys combined to form a set of sliders.
- It is equipped with an energy-saving mode.

3 Application field

- Slider touch key application
- Access control equipment
- Consumer Electronics

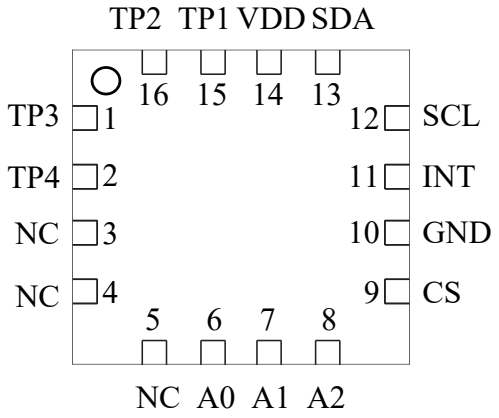
4 Package Pinout Information(SOP16/QFN16L)

TOP VIEW



SOP16

TOP VIEW



QFN16L

For more information: Page 22-23

4.1 VK3804/SOP16/QFN16L Pin Description

QFN16L	SOP16	Name	I/O	Function
14	12	VDD	VDD	Positive power supply
10	8	GND	GND	Negative power supply
9	7	CS	I	Sensitivity adjustment, connection to ground capacitor (1-100 nF)
11	9	INT	I	Touch state output. Open-collector output requires an external pull-up resistor.
12	10	SCL	I	I2C serial clock pin, open drain output needs external pull-up resistor
13	11	SDA	I/O	I2C serial data input/output pins, open-drain output requires an external pull-up resistor
3-5	1-3	NC	—	NC
6-8	4-6	A0-A3	I	I2C slave address selection pin
15-16, 1-2	13-16	TP1-TP4	I	Touch input

Table 1. Pin Description

5 Electrical Characteristics

5.1 Maximum Rated Values

Parameters	Symbol	Conditions	Value	Unit
Operating temperature	Top	——	-40~+85	℃
Storage temperature	T_{STG}	——	-50~+125	℃
Power supply voltage	VDD	Ta=25℃	VSS-0.3~VSS+5.5	V
Input voltage	V_{IN}	Ta=25℃	VSS-0.3~VDD+0.3	V
Chip antistatic strength HBM	ESD	——	>5	KV
Note: VSS represents system grounding.				

5.2 DC/AC Characteristics: (Test conditions: room temperature = 25℃)

Parameters	Symbol	Test conditions	Min.	Typ.	max.	Unit
Operating voltage	VDD		2.5	-	5.5	V
System oscillation frequency	F	VDD=5V	-	8M	-	HZ
Working current	IOP	Work, VDD=3V, output without load	-	1.1	-	mA
	IOFF	Standby mode, VDD=3V, output without load	5.3	6.8	10.0	uA

6 Function Description

Introduction to Touch Buttons：

Touching the buttons is a method that measures the change in capacitance when the human body approaches a conductor, and converts it into a numerical value for judgment. In this application, all the touch buttons have a Threshold setting parameter, which is used to adjust the sensitivity of the touch buttons.

The Threshold is adjusted according to the pressing depth of the button. The smaller the value, the more sensitive it is, but it is also more susceptible to noise interference.

Fly (Slider) Introduction：

The principle of the flywheel (scroll bar) button is a method that uses the measured pressing depth of the touch on the PCBLAYOUT to determine the pressing position. The advantage is that it can use the fewest buttons to resolve the most button addresses. The graphic of the scroll bar mainly includes two applications: circular and straight bar. As shown in the following figure:

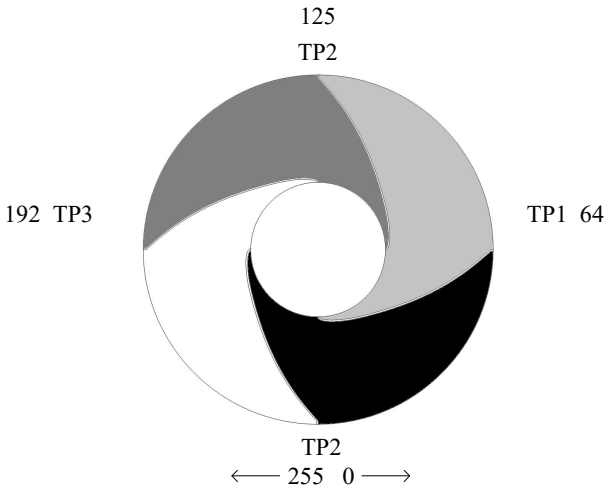


Figure 1. Circular Design

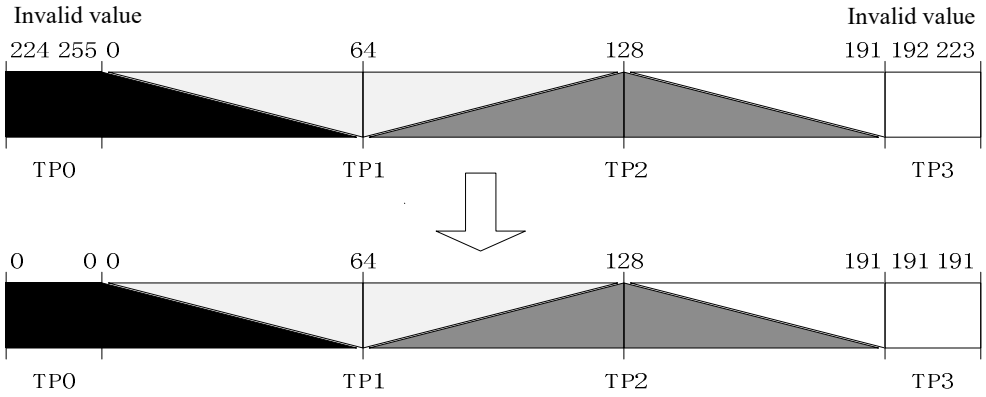


Figure 2. Bar Design

Note: In the straight bar design, since the beginning and the end are not connected (not in a circular shape), the valid address range is 0 to 191. If the value read is 192 to 223 or 224 to 255, it needs to be determined at the master control end whether it is 191 or 0 respectively.

Figure 3. Bar-shaped Design

The principle is to utilize the numerical changes obtained when pressing the Touch Pad, and then use the interpolation method to calculate its relative address. Therefore, at least 3 buttons are required to obtain the difference between the deepest pressed button and the buttons on the left and right sides for calculation.
In terms of design, it is recommended that the distance between the buttons and the center of the buttons should be less than 30mm. The distance between the teeth should be approximately 0.4mm (as shown in the following figure). Generally, a design of 3 to 4 teeth is preferred.

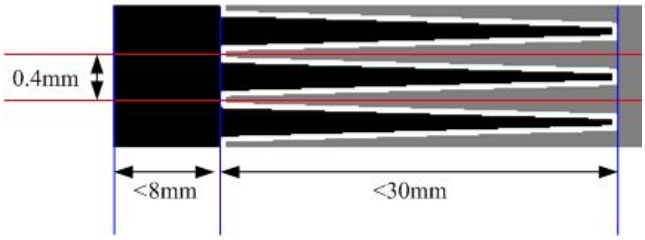


Figure 4. Key Points of Layout Design

Slider button for 4 key (TP4 ~ TP1), the following table:

Slide	4Key
TP1	Slide _1
TP2	Slide _2
TP3	Slide _3
TP4	Slide _4

Table 2. Definition of Sliders and Buttons

Note: The slider keys need to be arranged in the order of their numbers to accurately calculate their positions. It is prohibited to change the arrangement order at will.

7 IIC Protocol

The IC uses the IIC data transmission protocol and employs the two-wire bus SCL and SDA for reading and writing data. The INT pin is used to notify the Master of any changes in the key status.

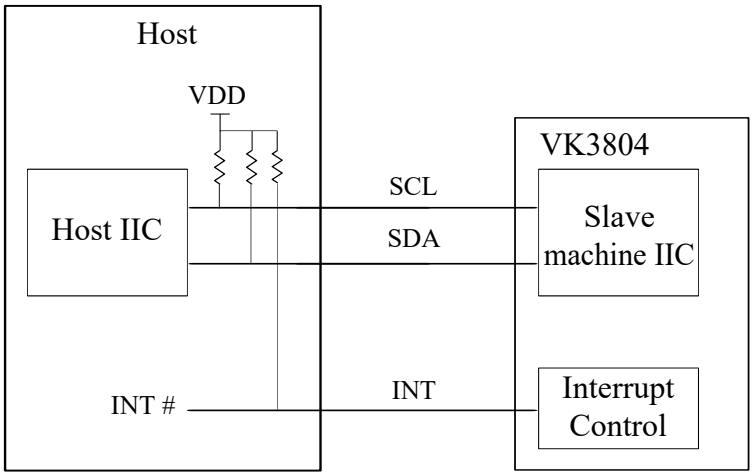


Figure 5. The host and VK3804 are connected via IIC.

INT is high when there is no key change. When there is a key, the INT pin will be pulled low. When there is no key, it is high.

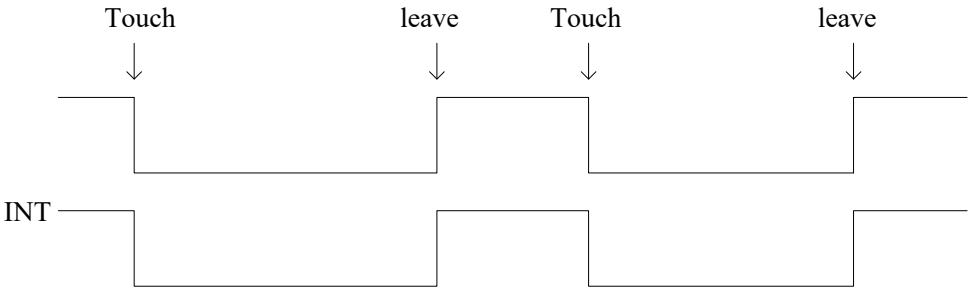
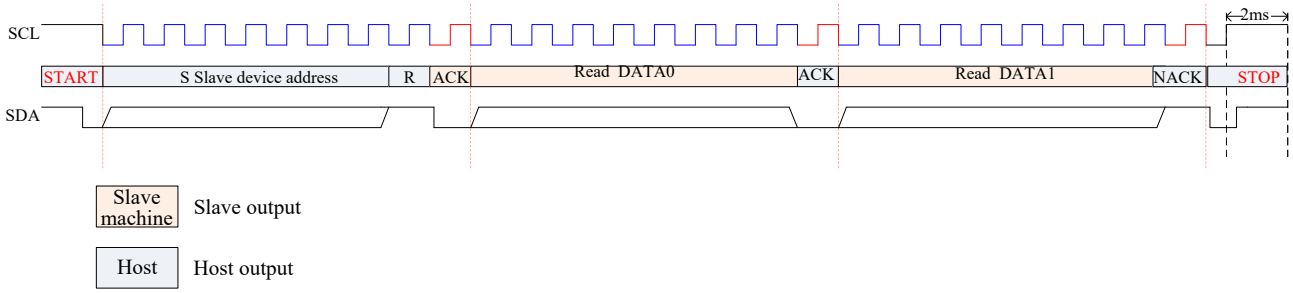


Figure 6. Description of INT pin

7.1 Data Flow



7.2 Slave address

From the machine address by A2 and A1, select A0 foot, in the following table:

A2	A1	A0	Slave address	Read (A6-A0,R)
0	0	0	50H	A1H
0	0	1	51H	A3H
0	1	0	52H	A5H
0	1	1	53H	A7H
1	0	0	54H	A9H
1	0	1	55H	ABH
1	1	0	56H	ADH
1	1	1	57H（Def）	AFH（Def）

Explanation: The default address is that the pins A2/A1/A0 are left floating.

7.3 Read the slider position

Read data	Bit 7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	C	0	0	WT	TP4	TP3	TP2	TP1
1	Slider position(0~255)							

C	Calibration Mark
0	Calibration
1	Calibration completed (def)

WT	Slider button indicator
0	Not touch
1	Touch

TP4-TP1	Touch button indicator
0	No key
1	With key

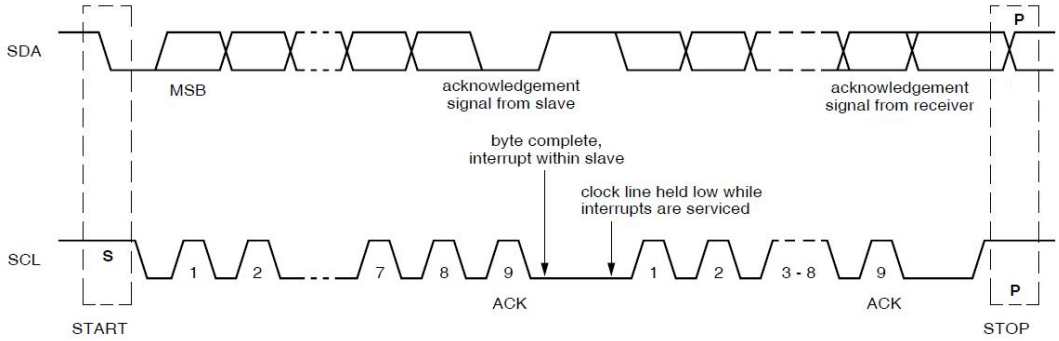
7.4 Slider position

Set to 0. When the touch slider is moved, output the key position. Release it and the last pressed position will be retained.

Wheel Position								Position
7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	2	0	2
x	x	x	x	x	x	x	x	...
1	1	1	1	1	1	0	1	253
1	1	1	1	1	1	1	0	254
1	1	1	1	1	1	1	1	255

8 Special Note

1. VK3804 I2C interface with hardware support SCL can support 100 KHZ, but decoding software for processing, processing takes about 20 ~ 100 us depending on the processing situation, so when the Master sends out a set of data packets, should increase the time delay is more than 200 us, to send a set of data packets.



So Master write program, need to pay attention to SCL pull Low action, if controlled by the hardware will mostly support this standard, if controlled by the program IO foot, please add to SCL output High to read back to confirm High, can let the program continue, if Low should wait for SCL High can continue. The C program is as follows:

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

2. If you need to continuously read key values, it is recommended to pause for more than 10ms after reading the previous key value, and then read the next key value. Otherwise, it will affect the response speed of the keys.

3. If the sleep mode is enabled, it is forbidden to read the key value continuously, because each time the key value is read, the sleep timer will be cleared. , causing the system to fail to sleep.

4. When the system enters sleep mode, the IIC function will be turned off. Repeating the IIC command at this time can wake up the system, but a "no ACK" response will be received. You need to wait for the system to wake up before re-executing the read/ write command.

5. Steps for adjusting the key threshold:

Step 1. Select the initial test capacitor for the CS (see the recommended circuit diagram): It is recommended to use 10nF as the initial test capacitor.

Step2. Press test for each key: touch the key at normal speed or use a metal bar for test conditions. If there is key output before touching the key, it means that the sensitivity is too high and the threshold value needs to be increased.

Step 3. Test button response speed:

Button sensitivity in judgment, if feel sensitive enough "keys", need further judgment is the key response fast enough, or key sensitivity is not enough. The method for judgment is to touch for a period of time (about 1 second) and check if there is any button output. Output, if no buttons are pressed key sensitive enough, to Step2 adjustments, if any key output, is the key response speed is not fast, on to the next step. After selecting an appropriate CS capacitor, you need to return to Step 2 to re-adjust the sensitivity. It should be noted that choosing a smaller CS capacitor will also reduce the precision of the slider buttons.

9 Demonstration program

```
/*    Project Name: Example Program for Controlling VK3804 via IIC from the
      Master Controller
      Project Objective:
      1. Simulate the IIC master controller writing setting parameters to VK3804
      through software
      2. Through software simulation, the IIC master control unit reads the key
      status of VK3804.

      MaincontrolMCU: C51
      Date & Version:V1.0
*/ //-----------------------------------------------------------------------------------------
----#include<reg51.h>
#include<intrins.h>
#define uint unsigned int
#define uchar unsigned char
#define address_W 0xa6 //Slave's address and write flag
#define address_R 0xa7 //The slave's address and read flag

sbit SINT=P0^0; //IIC interface between master and slave
sbit SDA=P0^1; //IIC interface between master and slave
sbit SCL=P0^2; //IIC interface between master and slave
uchar Write_Buffer[3]; //Master the written data cache
```

```
uchar Read_Buffer[3]; //Master the read data cache
 //---------------------------------------------------------------------------------------------
-//Function name: void delay(uint x)
//Function: Program Delay
//Function input: x
//Function output: None
//Intermediate variable: i,
j //---------------------------------------------------------------------------------------------
--void delay(uint x)
{
      uint i,j;
      for(i=x;i>0;i--)
            for(j=0x40;j>0;j--);
} //---------------------------------------------------------------------------------------------
-// Function Name: void sendStart()
// Function Purpose: Initiating bit of IIC
// Function Input: None
// Function Output: None
// Intermediate Variables: None
//---------------------------------------------------------------------------------------------
void sendStart() //Starting position
{

      SDA=1; /* Send the initial condition data signal */
      SCL=1;
      while(SCL! } {}
      SDA=0; /* Send the starting signal */
      _nop_();
      SCL=0; /* At this position, all that is needed is to set the SCL
      output to 0 and then wait for 4US. */
```

```
} //-----------------------------------------------------------------------------------------------
// Function Name: void sendStop()
// Function Purpose: IIC stop bit
// Function Input: None
// Function Output: None
// Intermediate Variables: None
//-----------------------------------------------------------------------------------------------
void sendStop() //Stop bit
{
    SCL=0;
    SDA=0; /* Sending the data signal indicating the end condition */
    _nop_();
    SCL=1;
    while(SCL!=1) { };
    _nop_();
    SDA=1;
} //-----------------------------------------------------------------------------------------------
//Function name: bit readACK()
//Function function: Read the acknowledge flag bit of IIC
//Function input: None
//Function output: The ACK signal of IIC returns 1, indicating no acknowledge, and
0, indicating acknowledge
// Intermediate variable: None
//-----------------------------------------------------------------------------------------------
bit readACK() //Read the response signal
{
SCL=0;
SDA=1; /*Here, the SDA bus is released, and a low-level response is issued by the
slave*/
_nop_();
SCL=1;
_nop_();
if(SDA)
```

```
            return 1; //no ACK
        else
            return 0; //ACK
}
//-------------------------------------------------------------------------------------------
// Function Name: void sendACK()
// Function function: The master sends a reply signal
// Function Input: None
// Function Output: None
// Intermediate Variables: None
//-------------------------------------------------------------------------------------------
void sendACK() //Output response signal
{
    SCL=0;
    SDA=0;
    _nop_();
    SCL=1;
}
//-------------------------------------------------------------------------------------------
// Function Name: void sendNOACK()
// Function function: The master sends no reply signal
// Function input: None
// Function output: None
// Intermediate variables: None
//-------------------------------------------------------------------------------------------
void sendNOACK() //Output an unresponsive signal
{
    SCL=0;
    SDA=1;
```

```
        _nop_();
SCL=1;
} //-----------------------------------------------------------------------------------------------
---
// Function Name: void sendByte(uchardat)
// Function: Master writes a byte to slave
// Function Input: dat = The byte to be sent
// Function Output: None
// Intermediate Variables: i
//-----------------------------------------------------------------------------------------------
-void sendByte(uchar dat) //Write a byte
{
        uchar i;
        for(i=0;i<8;i++)
        {
                SCL=0; /*Lock the I2C bus and prepare to send data*/
                if(dat&0x80)
                        SDA=1;
                else
       SDA=0;
_nop_(); /*If resistors need to be connected in series on SDA, SCL, and INT,
depending on the size of the resistors, it is recommended to appropriately extend
the time for larger resistors. Within 100 KHz, this adjustment is sufficient.；  */
        nop_();
SCL=1; /*Here due to the nature of 51 single-chip microcomputer don't need to do
input and output Settings, but if it is other SCM need to pull on the IO mouth to
enter the Settings, read high, SCL to output is high. The first clock falling edge
after reading and writing the ACK will clamp the SCL foot for data processing, so
set the SCL foot as input pull up and wait for SCL to be released.。  */
```

```
        while(SCL!=1) { };
        dat<<=1;
      }
}
//-----------------------------------------------------------------------------------------
-// Function Name: ucharreadByte()
// Function: Master reads a byte from slave
// Function Input: None
// Function Output: The byte that has been read
// Intermediate Variables: i, dat
//-----------------------------------------------------------------------------------------
-uchar readByte() //Read a byte
{
      uchar i, dat=0;
      for(i=0;i<8;i++)
      {
         SCL=0;
         SDA=1;
_nop_(); /*If resistors need to be connected in series on SDA, SCL, and INT,
depending on the size of the resistors, it is recommended to appropriately extend
the time for larger resistors. Within 100 KHz, this adjustment is sufficient.；*/
            dat<<=1;
SCL=1; /*Here, due to the characteristics of the 51 microcontroller, no input/output
settings are required. However, if it is an other microcontroller, its IO ports need to
be set to input with pull-up first. Once the data is read as high, SCL will be changed
to output high. After the first clock edge of ACK is received, the SCL pin will be
held for data processing by the opportunity latch. Therefore, the SCL pin is set as
input with pull-up, and then wait for SCL to be released.。*/
      while(SCL!=1) { };
      if(SDA==1)
      dat|=0x01;
```

```
    }
return dat;
}
//-----------------------------------------------------------------------------------------------/
/Function name: bit writeIIC(uchar addrW, uchar *writeData, uchar length)
// Function function: Master end writes data to slave device
// Function input: addrW = Slave device address and write flag
//                 *writeData = Prepare the first address of the write data
//                 length =  The length (in bytes) of the written data
Function output: Returns the acknowledge status of the IIC communication. If it is
1, stop and return. If it is 0, return after the communication is completed. //
Intermediate variable: i, ACK
//-----------------------------------------------------------------------------------------------
bit writeIIC(uchar addrW, uchar *writeData, uchar length)
{
uchar i;
bit ACK;
sendStart();
sendByte(addrW); //Transmit address and write mark
ACK = readACK();
if (ACK)
{
 sendStop(); //Address is incorrect or the device is not connected, send a stop signal
 return ACK;
}

for(i = 0; i<length; i++)
{
    sendByte(writeData[i]);
    ACK = readACK();
    if (ACK)
    {
```

```
                sendStop(); //No ACK was received, so the stop signal was sent.
                return ACK;
            }
        }
        sendStop(); //Data writing is complete. Sending stop signal.
        return ACK;
} //-----------------------------------------------------------------------------------------------
// Function name: bit readIIC(uchar addrR, uchar *readData, uchar length)
// Function function: Master end reads data from slave device
// Function input: addrR = Slave device address and read flag
//                 *readData =  The first address to hold data before it is read
//                 length =  The length (in bytes) of the data being read
//Function output: Returns the acknowledge status of the IIC communication. If it is
1, stop and return. If it is 0, return after the communication is completed. //
Intermediate variable: i, ACK
//-----------------------------------------------------------------------------------------------
bit readIIC(uchar addrR, uchar *readData, uchar length)
{
        uchar i;
        bit ACK;
        sendStart();
        sendByte(addrR); //Transmitting address and reading mark
        ACK = readACK();
if (ACK)
{
sendStop(); //Address is incorrect or the device is not connected, send a stop signal
return ACK;
}
 for(i = 0; i<length; i++)
 {
```

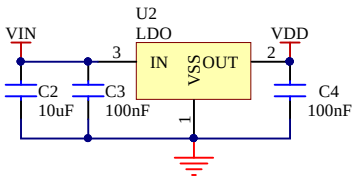
```
            readData[i] = readByte();
            if(i<length-1)
                sendACK();
            else
        sendNOACK(); //Read the last piece of data and send out NoACK
         }
        sendStop(); //Data reading is complete. Sending stop signal.
        return ACK;
}
//-------------------------------------------------------------------------------------------
//Function name: void setWrite_Buffer_3(uchar byte1, uchar byte2, uchar byte3)
//Function  function:  Write 3 bytes into the write cache register
//Function input: byte1
//                byte2
//                byte3
// Function output: None
/ / intermediate variable: no
//-------------------------------------------------------------------------------------------
void setWrite_Buffer_3(uchar byte1, uchar byte2, uchar byte3)
{
        Write_Buffer[0] = byte1;
        Write_Buffer[1] = byte2;
        Write_Buffer[2] = byte3;
}
void main()
{
bit ACK;
SINT = 1;
setWrite_Buffer_3(0xB1, 0xB9, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //MCU Setting
delay(1); //delay 1ms
setWrite_Buffer_3(0xC0, 0x10, 0x00);
```

```
ACK = writeIIC(address_W, &Write_Buffer, 3); //TP0 Threshold
delay(1); //delay 1ms
setWrite_Buffer_3(0xC1, 0x10, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //TP1 Threshold
delay(1); //delay 1ms
setWrite_Buffer_3(0xC2, 0x10, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //TP2 Threshold
delay(1); //delay 1ms
setWrite_Buffer_3(0xC3, 0x10, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //TP3 Threshold
delay(1); //delay 1ms
setWrite_Buffer_3(0xC4, 0x10, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //TP4 Threshold
delay(1); //delay 1ms
setWrite_Buffer_3(0xC5, 0x10, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //TP5 Threshold
delay(1); //delay 1ms
setWrite_Buffer_3(0xC6, 0x10, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //TP6 Threshold
delay(1); //delay 1ms
setWrite_Buffer_3(0xC7, 0x10, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //TP7 Threshold
delay(1); //delay 1ms
setWrite_Buffer_3(0xC8, 0x10, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //TP8 Threshold
delay(1); //delay 1ms
setWrite_Buffer_3(0xC9, 0x10, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //TP9 Threshold
delay(1); //delay 1ms
setWrite_Buffer_3(0xCA, 0x10, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //TP10 Threshold
delay(1); //delay 1ms
```

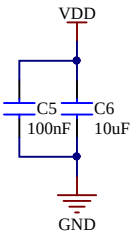
```
setWrite_Buffer_3(0xCB, 0x10, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //TP11 Threshold
delay(1); //delay 1ms
setWrite_Buffer_3(0xD0, 0x02, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //Sleep Group 1 Threshold
delay(1); //delay 1ms
setWrite_Buffer_3(0xD1, 0x0F, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //Sleep Group 1
delay(1); //delay 1ms
setWrite_Buffer_3(0xD2, 0x02, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //Sleep Group 2 Threshold
delay(1); //delay 1ms
setWrite_Buffer_3(0xD3, 0xF0, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //Sleep Group 2
delay(1); //delay 1ms
setWrite_Buffer_3(0xD4, 0x02, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //Sleep Group 3 Threshold
delay(1); //delay 1ms
setWrite_Buffer_3(0xD5, 0x00, 0x0F);
ACK = writeIIC(address_W, &Write_Buffer, 3); //Sleep Group 3
delay(1); //delay 1ms
setWrite_Buffer_3(0xD6, 0x02, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //Sleep Group 4 Threshold
delay(1); //delay 1ms
setWrite_Buffer_3(0xD7, 0x00, 0x00);
ACK = writeIIC(address_W, &Write_Buffer, 3); //Sleep Group 4
delay(50);

while(1)
{
if(!SINT) /*Waiting for the read request. If the power-saving mode is
turned off, there is no need to read the SINT. However, it is recommended
to wait for a duration of 30ms between each read of the key.*/
      ACK = readIIC(address_R, &Read_Buffer, 3); //Read the status of the buttons
   }
}
```

10 Application Circuits

LDO is recommended for power supply



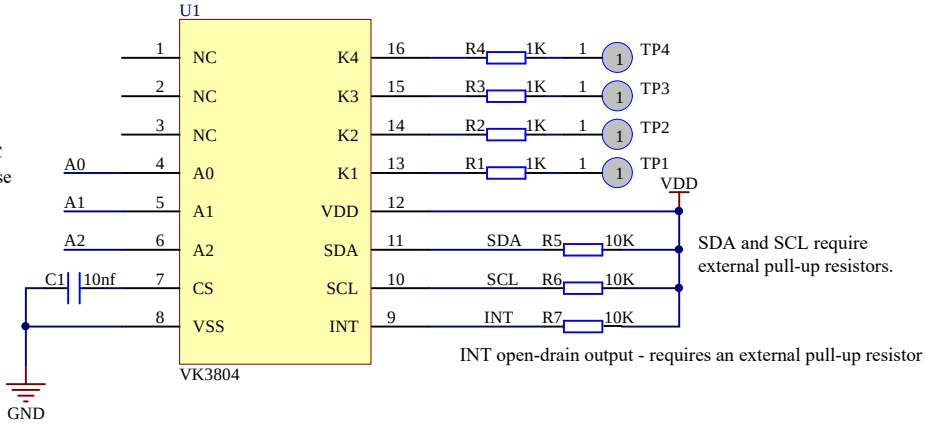
Add a filter capacitor to the power pin to stabilize the power supply



Feet touch the resistance of the concatenated general selection can enhance anti-jamming ability 0 to 10 k, interference significantly increase added value customer demand

A0-A2 Change chip I2C address for details, please refer to the data sheet

C1 is used to adjust the overall sensitivity. Generally, 1nF~47nF is larger, the more sensitive it is

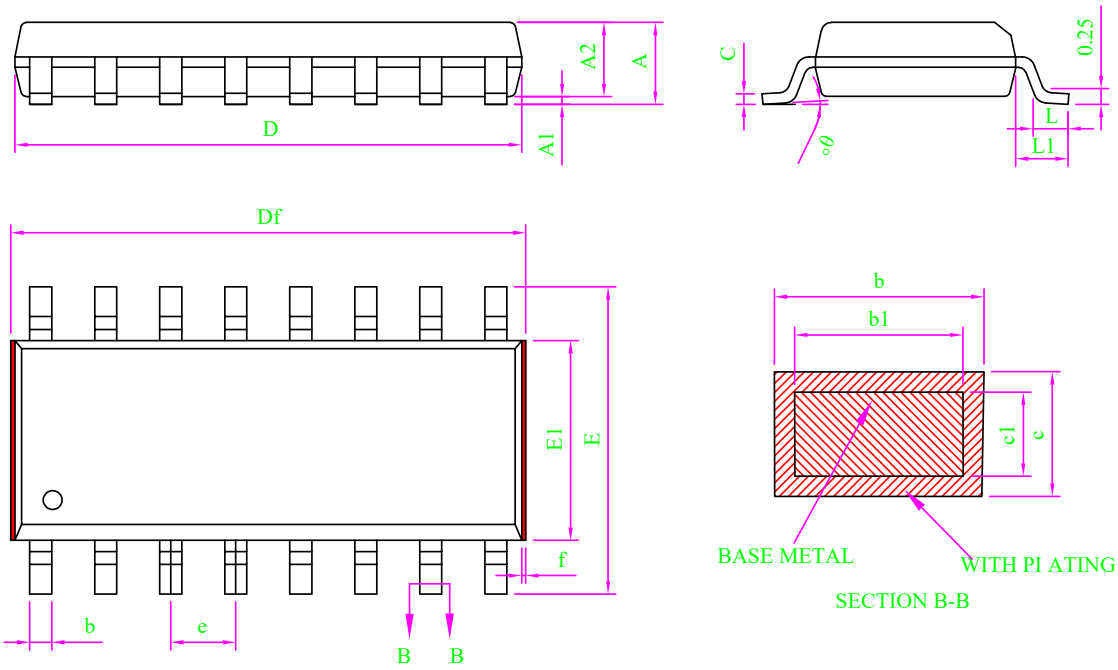


SDA and SCL require external pull-up resistors.

INT open-drain output - requires an external pull-up resistor

11 Package Information

11.1 SOP16(9.9mm x 3.9mm PP=1.27mm)

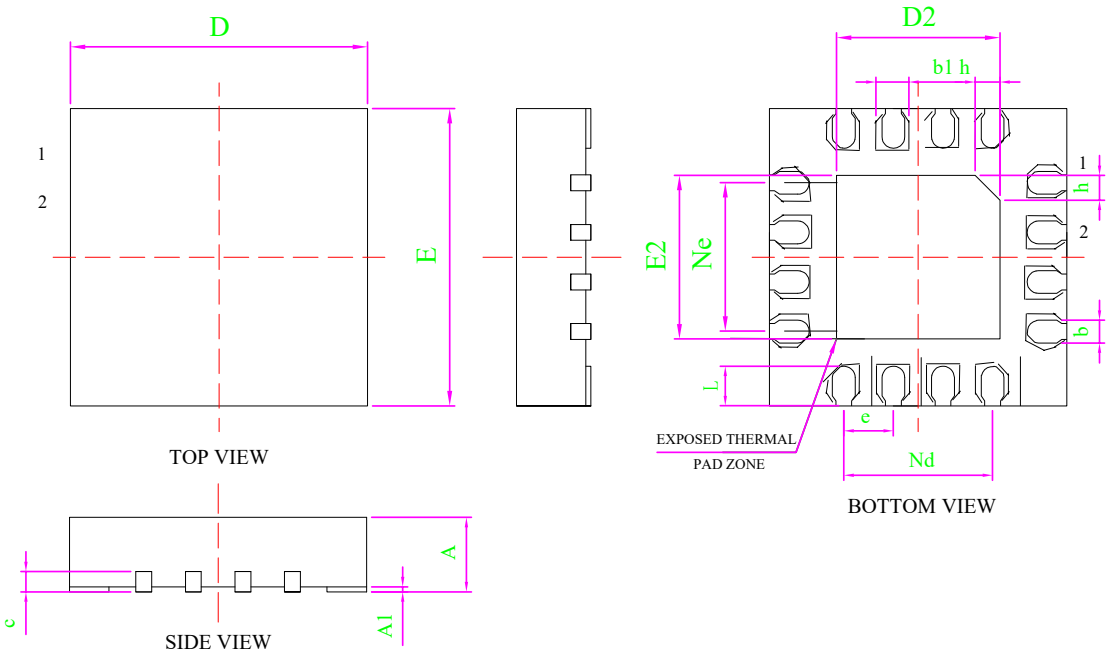


Note:

1. All dimension are in mm.
   Dim D&El does not include plastic flash; Df includes plastic flash(f);
   Flash: Plastic residual around body edge after de junk/singulation.
2. Dim b does not include dambar protrusion/intrusion.
3. Plating thickness 0.007mm-0.020mm

MILLIMETER			
SYMBOL	MIN	NOM	MAX
A	-	-	1.75
A1	0.10	0.15	0.20
A2	1.35	1.45	1.55
b	0.39	-	0.47
b1	0.38	0.41	0.43
c	0.20	-	0.25
c1	0.19	0.20	0.21
D	9.80	9.90	10.00
Df	9.90	-	10.40
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e	1.27BSC		
L	0.51	0.66	0.81
L1	0.95	1.05	1.15
θ	0	-	8°
f	0.05	-	0.20

11.2 QFN16L(3.0mm × 3.0mm PP=0.5mm)



Dimensions			
SYMBOL	MIN	NOMINAL	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
b	0.18	0.25	0.30
b1	0.30	0.35	0.40
c	0.18	0.20	0.25
D	2.90	3.00	3.10
D2	1.55	1.65	1.75
e	0.50BSC		
Ne	1.50BSC		
Nd	1.50BSC		
E	2.90	3.00	3.10
E2	1.55	1.65	1.75
L	0.35	0.40	0.45
h	0.20	0.25	0.30
L/F 载体尺寸（miL）	75*75		

12 Disclaimer

Warranty and liability —— The information provided in this document is believed to be accurate and reliable. However, Shenzhen Vinka Microelectronics Co., Ltd. (hereinafter "the Company") makes no warranties, express or implied, as to the completeness or suitability of this information for any specific purpose.

In no event shall the Company be liable for any indirect, incidental, or consequential damages, including but not limited to loss of profits, equipment damage, or system failure, arising out of the use of this product or documentation, regardless of the legal theory under which such liability is asserted.

Right to change —— The Company reserves the right to modify any information contained herein without prior notice. The latest version of this document is available at: https://www.szvinka.com/

Applicability —— This product is not designed or intended for use in life-critical, medical, or safety systems where failure could result in injury or death. The customer shall assume full responsibility for any such use.

Application —— All product application descriptions provided herein are intended for illustrative purposes only.
The Company makes no representations or warranties, express or implied, regarding the suitability of any specific application without further testing or modification.

The customer is solely responsible for determining whether the Company's products are appropriate for their intended applications or end customers.

The customer shall ensure proper design practices, implementation safeguards, and operational validation to minimize risks associated with product use.

The Company shall not be held liable for any defects, losses, costs, or damages arising from weaknesses or failures in the customer's own products or applications, or from the integration or use of third-party products.

Furthermore, the customer shall conduct all necessary testing and validation for any third-party deployment of the Company's products to avoid potential misuse or associated damages. The Company assumes no liability in this regard.

Commercial terms of sale —— Unless otherwise agreed in writing, sales of this product are subject to the Company's standard terms and conditions of sale. The Company expressly rejects the applicability of the customer's general terms and conditions.

Export control—— This product may be subject to applicable export control regulations. The customer is solely responsible for compliance with such regulations, including obtaining any necessary export licenses.

13 Revision History

No.	Version	Date	Modify the content	Check
1	1.0	2024-03-22	Initial release	YES
2	1.1	2026-02-27	Change Description	YES

[1] Please refer to the latest version of this document before starting or finalizing any design.

[2] Since the release of this document, the status or availability of this product may have changed. For the most up-to-date information, please visit: https://www.szvinka.com/