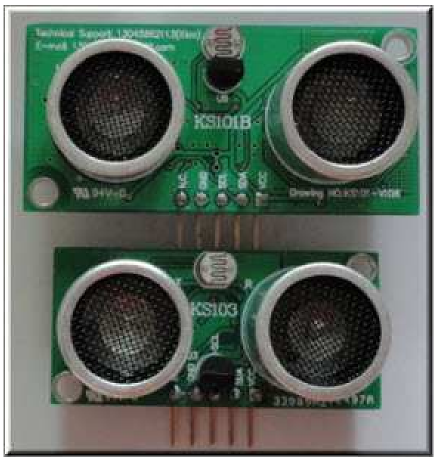
# KS101B/KS103/KS103S Ultra Sonic Range Finder

**Technical Specification**

**Rev 1.10**

**Date：2011.03.10**

**Modify Date: 2011.05.23**



**Function Abstract**

- Ranging with temperature revised, high precision in distance
- Range in 1cm to 650cm, using patent technologies(KS103/KS103S range in 1cm to 550cm)
- Detecting frequency up to 500Hz, which can detect 500 times per second
- Use slave $I^2C$ bus, detect master's command automatically
- 20 $I^2C$ address and can be changed, the address is 0xd0-0xfe (except 0xf0,0xf2,0xf4,0xf6)
- Broadcast address 0x00 allowed(except KS103/KS103S)
- Short and high precision temperature detect of 83ms each time
- Sleep automatically after 5s without commands, wake up by master at any time
- Short range in 10cm, 20cm to 470cm, 47 steps total
- 1ms in light intensity detect. Special command for light intensity detect
- Industrial and Extended Temperature range(-30℃~+85℃)(KS103/KS103S work in 0~70℃)
- Wide operating voltage range (3.0V~5.5V)
- Unique filtering noise reduction technology, can still work under noisy power supply

**Electrical Specification:**

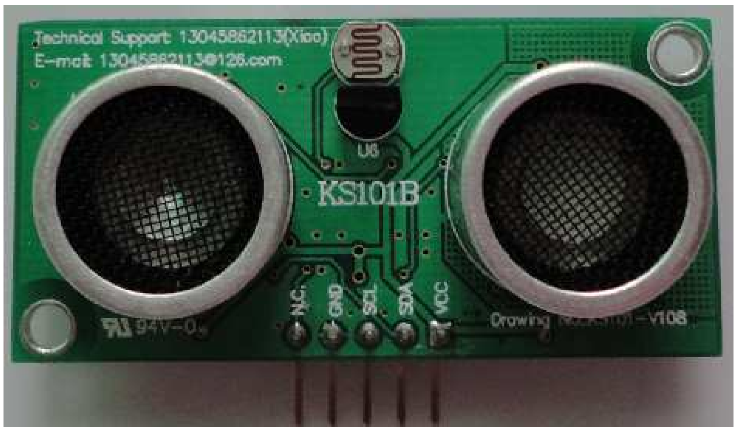
Operating Voltage: ***3.0V~5.5V***

Operating Current: ***1.6-2.7 mA@5.0V, typical    10.6mA@5.0V, max***

Standy Cueernt: ***500uA@5.0V, max.***

Use nano Watt Technology, power saving in sleep mode. Went into sleep mode automatically after 5s without $I^2C$ command.

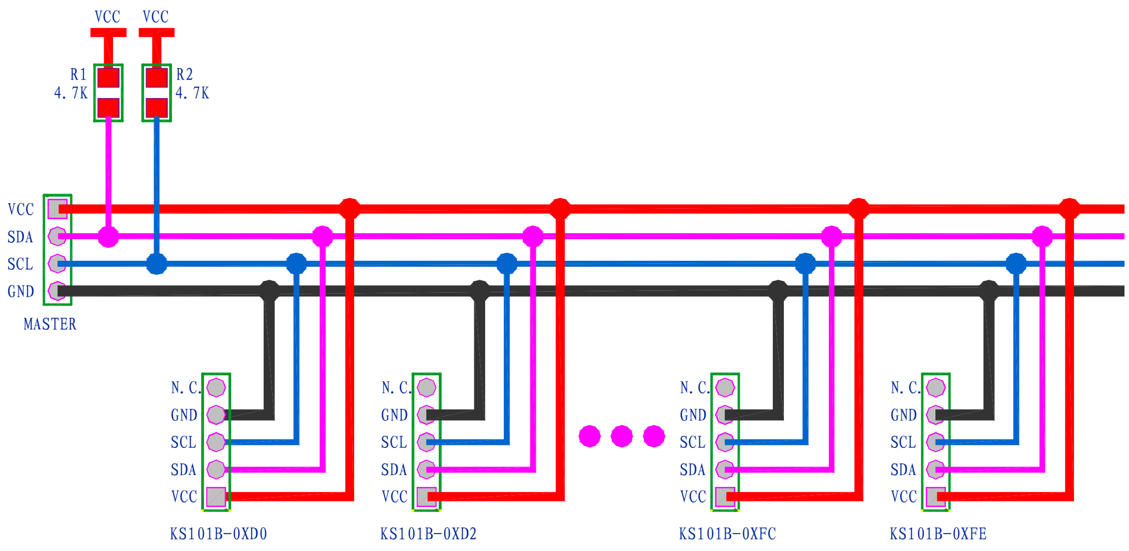
**KS101B/KS103/KS103S's connection:**


Technical Support: 13045862113(Xiao)
E-mail: 13045862113@126.com
U6
KS101B
N.C.
GND
SCL
SDA
VCC
94V-0
Drawing

As shown, there's five pins "VCC, SDA, SCL, GND, N.C." on KS101B/KS103/KS103S's board.The VCC pin should connect to master's VCC, the VCC must be in range of 3.0-5.5V (1). The GND connect to master's ground. The N.C. pin should be left unconnected. It's only be used for program when producing. The SCL is I$^2$C bus's clock line, and SDA is I$^2$C bus's data line. The SCL and SDA lines should each have a pull-up resistor to VCC somewhere on the I$^2$C bus. We suggest one pair of resistors about 1.8k-4.7k on master board. Some master modules already have pull-up resistors to act as I$^2$C bus and you can connect KS101B/KS103/KS103S to I$^2$C bus directly.

Note 1: To achieve KS101B/KS103/KS103S'S best working conditions recommended +5 V power supply. Moreover, VCC and GND is strictly prohibited reverse, it may damage the circuit.

Here's the circuit diagram(20PCS KS101B/KS103/KS103S on I$^2$C bus):



KS101B/KS103/KS103S's default address is 0xe8 and can be change to the one of 0xd0, 0xd2 , 0xd4, case 0xd6, 0xd8, 0xda, 0xdc, 0xde, 0xe0, 0xe2, 0xe4, 0xe6, 0xe8, 0xea, 0xec, 0xee, 0xf8, 0xfa, 0xfc, 0xfe. (2)

Note 2: 0xd0~0xfe except address "0xf0, 0xf2, 0xf4,0xf6"，this 4 address are reserved for 10 bit slave I$^2$C address.

**Sequence of change I$^2$C address**

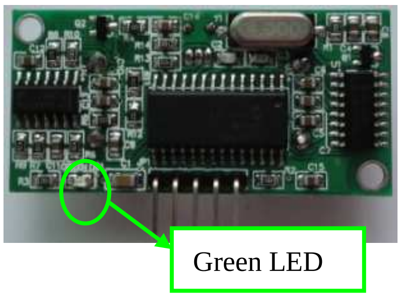
| Address | 2 | 0x9a | Delay 1ms | Address | 2 | 0x92 | Delay 1ms | Address | 2 | 0x9e | Delay 1ms | Address | 2 | New Address | Delay 100ms |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Change I$^2$C address must follow the sequence and the delay time showed on table is the least delay time. It's won't be trouble because there's the right function(seen in attached files No.3) *change_i2c_address(addr_old,addr_new)* to refer to. It's used for 51 MCU as an example.
You can't cut off the power when changing I$^2$C address or the EEPROM may be corrupted .After changing repower KS101B/KS103/KS103S and the LED on the back board will flash the new address in binary system. The change I$^2$C address function should not put in loop for ever such as while(1) loop. We suggest put it in initializtion function, to make sure change I$^2$C address function run just one time when power up.
For KS101B,If you don't want to watch the old address, you can use general call address 0x00 instead.

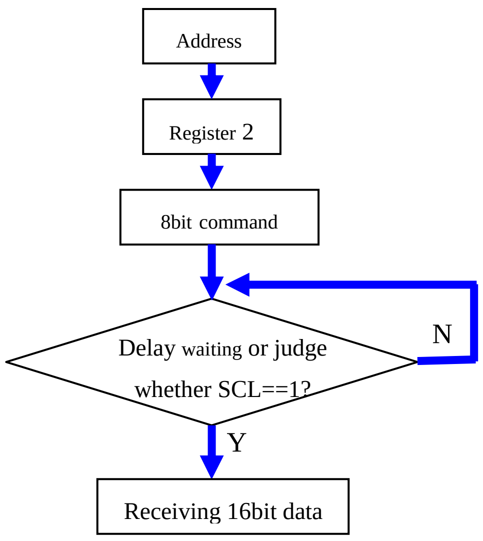
**GENERAL CALL ADDRESS SUPPORT** (KS101B ONLY)
The addressing procedure for the I$^2$C bus is such that,the first byte after the Start condition usually determines which device will be the slave addressed by the master. The exception is the general call address,which can address all devices. When this address is used, all devices should, in theory, respond with an acknowledge.The general call address is one of eight addresses reserved for specific purposes by the I$^2$C protocol. It consists of all 0's with R/W = 0. You also can use address 0x00 to control KS101B, but you cannot get the range result by address "0x00+1", you should use the right address from 0xd0 to 0xfe to get KS101B's range result.

**KS101B/KS103/KS103S's Work Sequence**
After power on KS101B/KS103/KS103S, back green LED will flash the 8-bit address in binary system. Short flash twice will be "1", slow flash once will be "0". Such as address 0xea, it's *0B*11101010 in binary system, the green LED will be *slow light→off→short flash twice→off→short flash twice→off→short flash twice→off→slow flash once→off→short flash twice→off→slow flash once→off→short flash twice→off→slow flash once.* [(3)]

Note 3: LED flashing green light may stimulate the eyes, try not to look at flashing LED directly, you can use the corner of the eye to observe the flashing.



After power up, when KS101B/KS103/KS103S receive valid command, it will stop flashing and begin to detecting at once.
KS101B/KS103/KS103S use the I$^2$C interface to communicate with master, automatic response to master I$^2$C control instructions for the 8-bit data, there's instructions send process:



**Multi-range mode(KS103S doesn't support command 0xb4 and 0xbc and temperature command)**

Detecting command from 0x01 to 0x2f, the greater the value, the greater the signal gain. Command 0x01 corresponding with range about 100mm, command 0x02 corresponding with range about 200mm, ... ..., and so on, command 0x2f corresponding with range about 4700mm. The smaller the range, the faster it detecting. Detection time based on the ultrasonic transmission time plus about 1ms. Note that the 16-bit value returned is a unit of time in μs, and the time spend is from ultrasonic transmit to receive.

And most useful range will be range of 0-5m of command 0xb0/0xb2/0xb4, and range of command 0-11m of 0xb8/0xba/0xbc. By using "address + register 2 + ranging command" to begin a new ranging, and then delay or wait for corresponding time that the table 1 specified, and then using the read function to read the value of register 2 and register 3, then you can get the 16-bit distance data. Command 0xb0 and 0xb8 will return distance in millimeter ,it's calculate by 25 ℃(about 340m/s) according to the actual detecting time; command 0xb2 and 0xba detecting returns a unit μs of time spend, and the time spend is from ultrasonic transmit to receive.

To get accurate distance measurement value, use 0xb4 or 0xbc command, these two commands automatically using high-precision temperature revised technology, more stable and more accurate detection of the value. You can also use 0xb2/0xba (transmission time) + 0xc9/0xca/0xcb/0xcc (ambient temperature) combination, to detect the ultrasonic transmission time in the air and the corresponding temperature, and then translated by the speed of sound to get the exact distance values. Using a temperature revised 0xb4 command, the most accurate cac reach up to 1mm, the error is 0.152mm/17cm. With the change of environment and development of technology, KS101B/KS103/KS103S internal formula may not be in future. For millimeter-level accuracy of the distance, please use the latest possible formula to get the precise distance value when you get ultrasonic transit time and temperature.

Meanwhile, in the long-range detection, if the power supply is noisy, KS101B/KS103/KS103S will be likely not reach 1cm ~ 650cm maximum range, so if you use noisy power supply (for example, take power from the computer USB port), please use the detection range of 0-5m detection commands to get the right value.

**Intelligent recognition of detecting end**

After KS101B/KS103/KS103S finish detect command sending, it take some time to get the right side 16-bit I$^2$C data. The user only know the maximum detecting time, but we do not really know the actual time of each detecting. KS101B/KS103/KS103S use an intelligent recognition technology. Simply say, SCL will remain low when KS101B/KS103/KS103S's detecting, when it finish detecting SCL will become high level at once. User can check whether the SCL line goes high by using while (! SCL) statements to wait, SCL line goes high indicates that the detecting is completed, you can start receiving 16-bit data through the I$^2$C bus. Note that when finish sending the detect command, you need to delay at least 40us and then begin to judge whether the SCL line goes high, for the most fast command 0xa0 will take about 1ms, so we suggest to delay about 1ms, it won't interrupt the ongoing detecting, also won't reduce the detection efficiency. You can also delay a period of time and then starte to receive the 16-bit I$^2$C data. (4)

Note 4: The bus clamp detection methods can provide customers greater speed and efficiency of detection, rather than waiting at least every 65ms. In other words, most of the time users only need to quickly know whether there are obstacles within 1m range. Specific delay time should be greater than the maximum detecting time show in Table 1.

Intelligent recognition of detecting end will be a default configuration, if you do not want to judge the SCL line's level, you can sending a command 0xc3 to close this function, then power off and restart KS101B/KS103/KS103S SCL line won't go low when detecting. If you want to restore SCL clamp down function, you can send 0xc2 command to KS101B/KS103/KS103S.
Intelligent recognition of detecting end is automatically saved after configured, and immediately working under the new configuration.
Follow the attached files 3, the configuration code should be as follows:
*write_byte (0xe8, 2,0 xc2); //SCL clamp*
*delayms (2000);*
KS101B/KS103/KS103S will be run according to the new configuration when restart.

**Command and register list**

| **Register** | **Command** | **Return value range(Decimal)** | **Return value range(Hex)** | **Note** |
|---|---|---|---|---|
| 0 | | 0-255 | 0-0xff | Produce year |
| 1 | | 1-52 | 0x01-0x35 | Produce week |
| 2 | 0x01 | 80-577μs | 0x50-0x241μs | Range of about 100mm, return μs. Maximum time consume = Maximum return value + 1000μs |
| 2 | 0x02 | 66-1154μs | 0x42-0x482μs | Range of about 200mm, return μs. Maximum time consume = Maximum return value + 1000μs |
| 2 | 0x03 | 66-1731μs | 0x42-0x6c3μs | Range of about 300mm, return μs. Maximum time consume = Maximum return value + 1000μs |
| 2 | 0x04 | 80-2308μs | 0x50-0x904μs | Range of about 400mm, return μs. Maximum time consume = Maximum return value + 1000μs |
| 2 | 0x05 | 82-2885μs | 0x52-0xb45μs | Range of about 500mm, return μs. Maximum time consume = Maximum return value + 1000μs |
| 2 | 0x06 | 80-3462μs | 0x50-0xd86μs | Range of about 600mm, return μs. Maximum time consume = Maximum return value + 1000μs |
| 2 | 0x07 | 68-4039μs | 0x44-0xfc7μs | Range of about 700mm, return μs. Maximum time consume = Maximum return value + 1000μs |
| 2 | 0x08 | 80-4616μs | 0x50-0x1208μs | Range of about 800mm, return μs. Maximum time consume = Maximum return value + 1000μs |
| 2 | 0x09 | 80-5193μs | 0x50-0x1449μs | Range of about 900mm, return μs. Maximum time consume = Maximum return value + 1000μs |
| 2 | 0x0a | 66-5770μs | 0x42-0x168aμs | Range of about 1000mm, return μs. Maximum time consume = Maximum return value + 1000μs |
| 2 | 0x0b | 93-6347μs | 0x5d-0x18cbμs | Range of about 1100mm, return μs. Maximum time consume = Maximum return value + 1000μs |
| 2 | 0x0c | 79-6924μs | 0x4f-0x1b0cμs | Range of about 1200mm, return μs. Maximum time |

| | | | | |
|---|---|---|---|---|
| | | | | consume = Maximum return value + 1000µs |
| 2 | 0x0d | 93-7501µs | 0x5d-0x1d4dµs | Range of about 1300mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x0e | 79-8078µs | 0x4f-0x1f8eµs | Range of about 1400mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x0f | 79-8655µs | 0x4f-0x21cfµs | Range of about 1500mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x10 | 79-9232µs | 0x4f-0x2410µs | Range of about 1600mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x11 | 77-9809µs | 0x4d-0x2651µs | Range of about 1700mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x12 | 77-10386µs | 0x4d-0x2892µs | Range of about 1800mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x13 | 77-10963µs | 0x4d-0x2ad3µs | Range of about 1900mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x14 | 79-11540µs | 0x4f-0x2d14µs | Range of about 2000mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x15 | 63-12117µs | 0x3f-0x2f55µs | Range of about 2100mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x16 | 79-12694µs | 0x4f-0x3196µs | Range of about 2200mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x17 | 79-13271µs | 0x4f-0x33d7µs | Range of about 2300mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x18 | 79-13848µs | 0x4f-0x3618µs | Range of about 2400mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x19 | 77-14425µs | 0x4d-0x3859µs | Range of about 2500mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x1a | 93-15002µs | 0x5d-0x3a9aµs | Range of about 2600mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x1b | 63-15579µs | 0x3f-0x3cdbµs | Range of about 2700mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x1c | 79-16156µs | 0x4f-0x3f1cµs | Range of about 2800mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x1d | 79-16733µs | 0x4f-0x415dµs | Range of about 2900mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x1e | 79-17310µs | 0x4f-0x439eµs | Range of about 3000mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x1f | 77-17887µs | 0x4d-0x45dfµs | Range of about 3100mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x20 | 91-18464µs | 0x5b-0x4820µs | Range of about 3200mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x21 | 79-19041µs | 0x4f-0x4a61µs | Range of about 3300mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x22 | 79-19618µs | 0x4f-0x4ca2µs | Range of about 3400mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x23 | 79-20195µs | 0x4f-0x4ee3µs | Range of about 3500mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x24 | 79-20772µs | 0x4f-0x5124µs | Range of about 3600mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x25 | 77-21349µs | 0x4d-0x5365µs | Range of about 3700mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x26 | 79-21926µs | 0x4f-0x55a6µs | Range of about 3800mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x27 | 63-22503µs | 0x3f-0x57e7µs | Range of about 3900mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x28 | 79-23080µs | 0x4f-0x5a28µs | Range of about 4000mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x29 | 63-23657µs | 0x3f-0x5c69µs | Range of about 4100mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x2a | 79-24234µs | 0x4f-0x5eaaµs | Range of about 4200mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x2b | 79-24811µs | 0x4f-0x60ebµs | Range of about 4300mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x2c | 79-25388µs | 0x4f-0x632cµs | Range of about 4400mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x2d | 77-25965µs | 0x4d-0x656dµs | Range of about 4500mm, return µs. Maximum time consume = Maximum return value + 1000µs |

| | | | | |
|---|---|---|---|---|
| 2 | 0x2e | 79-26542µs | 0x4f-0x67aeµs | Range of about 4600mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x2f | 63-27119µs | 0x3f-0x69efµs | Range of about 4700mm, return µs. Maximum time consume = Maximum return value + 1000µs |
| 2 | 0x70 | void | void | First level noise reduction, factory default settings, for battery-powered |
| 2 | 0x71 | void | void | Second level noise reduction, for USB-powered |
| 2 | 0x72 | void | void | Third level noise reduction, for long distance USB-powered |
| 2 | 0x73 | void | void | Fourth level noise reduction, for switching power supply |
| 2 | 0x74 | void | void | Fifth level noise reduction, for noisy switching power supply |
| 2 | 0x75 | void | void | Sixth level noise reduction, for high noise power supply |
| 2 | 0x8a | void | void | I2C communications test command, after sending LED will flash display the binary value of the command |
| 2 | 0x8b | void | void | |
| 2 | 0x8c | void | void | |
| 2 | 0x92 | void | void | The second sequence change address |
| 2 | 0x9a | void | void | The first sequence change address |
| 2 | 0x9e | void | void | The third sequence change address |
| 2 | 0xa0 | 0-1023 | 0-0x3ff | Light intensity detect, the light is stronger, the greater the value, the detect takes about 1ms |
| 2 | 0xb0 | 10-5200mm | 0x0a-0x1450mm | 0-5m range, normal range (without temperature revised), return mm, detection took about 33ms maximum |
| 2 | 0xb2 | 79-30000µs | 0x4f-0x7530µs | 0-5m range, normal range (without temperature revised), return µs, detection took about 32ms maximum |
| 2 | 0xb4 | 10-5200mm | 0x0a-0x1450mm | 0-5m range, normal range (with temperature revised), return mm, detection took about 87ms maximum(KS103S doesn't support this command) |
| 2 | 0xb8 | 20-11280mm | 0x14-0x2c10mm | 0-11m range, normal range (without temperature revised), return mm, detection took about 68ms maximum |
| 2 | 0xba | 159-65278µs | 0x9f-0xfefeµs | 0-11m range, normal range (without temperature revised), return µs, detection took about 66ms maximum |
| 2 | 0xbc | 20-11280mm | 0x14-0x2c10mm | 0-11m range, normal range (with temperature revised), return mm, detection took about 87ms maximum(KS103S doesn't support this command) |
| 2 | 0xc0 | void | void | Open LED flash when detecting，default |
| 2 | 0xc1 | void | void | Close LED flash when detecting |
| 2 | 0xc2 | void | void | SCL line force to low when detecting, the default |
| 2 | 0xc3 | void | void | SCL line keep high level when detecting |
| 2 | 0xc4 | void | void | 5 seconds for sleep waiting |
| 2 | 0xc5 | void | void | 1 seconds for sleep waiting |
| 2 | 0xc9 | 0-255 | 0-0xff | Return 9-bit precision temperature data, according to DS18B20 format, range of -40 ℃ - +125 ℃, detection takes about 83ms(KS103S doesn't support this command) |
| 2 | 0xca | 0-255 | 0-0xff | Return 10-bit precision temperature data, according to DS18B20 format, range of -40 ℃ - +125 ℃, detection takes about 168ms(KS103S doesn't support this command) |
| 2 | 0xcb | 0-255 | 0-0xff | Return 11-bit precision temperature data, according to DS18B20 format, range of -40 ℃ - +125 ℃, detection takes about 315ms(KS103S doesn't support this command) |

| | | | | |
|---|---|---|---|---|
| 2 | 0xcc | 0-255 | 0-0xff | Return 12-bit precision temperature data, according to DS18B20 format, range of -40 ℃ - +125 ℃, detection takes about 610ms(KS103S doesn't support this command) |
| 3 | | 0-255 | 0-0xff | Register 3 and register 2 use together, register 2 returns high 8-bit of 16-bit data, and register 3, returns the lower 8-bits of data 16-bit data. |
| 4 | | 0-255 | 0-0xff | High 8-bit data of power on times |
| 5 | | 0-255 | 0-0xff | Low 8-bit data of power on times |
| 6 | | 0-255 | 0-0xff | Program version |
| 7-15 | | 0 | 0 | Reserved |

**Table 1**

**Noise reduction of power supply command**

KS101B/KS103/KS103S recommend using the default battery power. If you use noisy power supply, distance values may be volatile instability. Users can send command 0x70, 0x71, 0x72, 0x73, 0x74, 0x75 to configure KS101B/KS103/KS103S ranging module clutter suppression. Command 0x70 will configure the module on first level noise reduction, suitable for battery-powered occasions, but also the factory default settings. Command 0x71 will enable the module to be second level noise reduction, for USB-powered. Command 0x72 will configure the module to be third level noise reduction, for long distance USB-powered. Command 0x73 will configure the module to be fourth level noise reduction, for switching power supply. Command 0x74 will configure the module to be fifth level noise reduction, for noisy switching power supply. Command 0x75 will configure the module to be sixth level noise reduction, for high noise power supply.

Should choose smaller values such as 0x70 to ensure accuracy. The higher the noise reduction level, the greater the probability that the waveform is eliminated.

Configuration is very simple, send commands to the module as follows: "I$^2$C address + register 2 +0x70/0x71/0x72/0x73/0x74/0x75 " , and at least 2 seconds delay after sending to allow the system to automatically configure . And begin work under the new configuration.

Follow the attached files 3, the configuration code should be as follows:

*write_byte (0xe8, 2,0 x71); //config to the sencond level*

*delayms (2000);*

KS101B/KS103/KS103S will be run according to the new configuration when restart.

**Temperature detecting** (KS101B and KS103 only)

Temperature detecting included 0xc9, 0xca, 0xcb, 0xcc total of four commands, through the "I$^2$C address + register 2 + 0xc9/0xca/0xcb/0xcc" sequence, delay or wait for corresponding time that table 1 specified, and then use register read function to get the 16-bit data by reading the value of register 2 and register 3, the 16-bit data obtained to comply with the rules DS18B20 chip temperature readings, the specific information please refer to the chip DS18B20. To 0xcc instruction, for example, it will get a total of 16-bit data. The high five bit of 16-bit data is the sign bit, if the measured temperature is greater than 0℃, the high five bit will 00000, and then the 16-bit data divide by 16 and we can get the temperature.

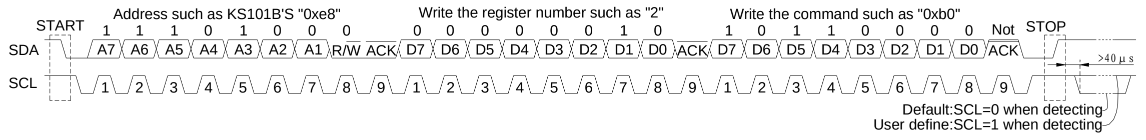
**Light intensity detecting**

Use 0xa0 command, through the "I$^2$C address + register 2 + 0xa0 " sequence, delay or wait for 1ms and then use register read function to read the value of register 2 and register 3, you can quickly get the ambient light intensity. The stronger the light, the greater the return value, and the return value is between 0 to 1023.

**Sequence**

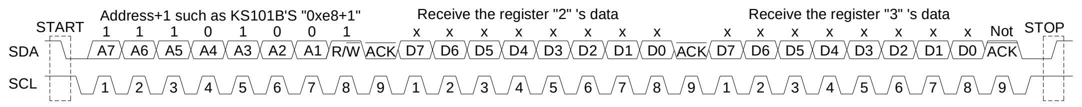
Send detect command to KS101B/KS103/KS103S:

| Address | Register 2 | 8-bit command |
|---|---|---|


START
Address such as KS101B'S "0xe8"
1 1 1 0 1 0 0 0
Write the register number such as "2"
0 0 0 0 0 0 1 0
Write the command such as "0xb0"
1 0 1 1 0 0 0 0
Not
STOP
SDA
A7 A6 A5 A4 A3 A2 A1 R/W ACK D7 D6 D5 D4 D3 D2 D1 D0 ACK D7 D6 D5 D4 D3 D2 D1 D0 ACK ACK
SCL
1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9
>40 μ s
Default:SCL=0 when detecting
User define:SCL=1 when detecting

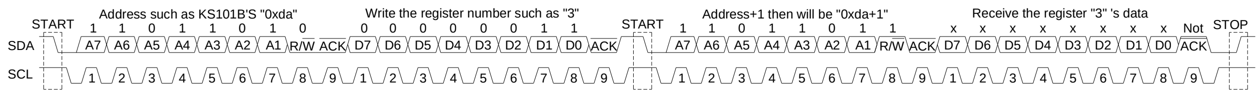
Follow the command sending, then begin to receive data:

| Address+1 | Read Register 2 | Read Register 3 |
|---|---|---|


START
Address+1 such as KS101B'S "0xe8+1"
1 1 1 0 1 0 0 1
Receive the register "2" 's data
x x x x x x x x
Receive the register "3" 's data
x x x x x x x x
Not
STOP
SDA
A7 A6 A5 A4 A3 A2 A1 R/W ACK D7 D6 D5 D4 D3 D2 D1 D0 ACK D7 D6 D5 D4 D3 D2 D1 D0 ACK
SCL
1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9

Read any register(Such as register 3): **(5)**

| Address | Register 3 | Address+1 | Read Register 3 |
|---|---|---|---|


START
Address such as KS101B'S "0xda"
1 1 0 1 1 0 1 0
Write the register number such as "3"
0 0 0 0 0 0 1 1
START
Address+1 then will be "0xda+1"
1 1 0 1 1 0 1 1
Receive the register "3" 's data
x x x x x x x x
Not
STOP
SDA
A7 A6 A5 A4 A3 A2 A1 R/W ACK D7 D6 D5 D4 D3 D2 D1 D0 ACK A7 A6 A5 A4 A3 A2 A1 R/W ACK D7 D6 D5 D4 D3 D2 D1 D0 ACK
SCL
1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9

Note 5: To read any register will be OK except register 2 and register 3. If you want to read register 2 and register 3, you must send detecting command to register 2 first. Note that all detecting command are stored in the register 2.

**Further power saving measures**

If you want to save power, you can send command 0xc1 to turn off LED flash to reduce current consumption. Send command 0xc0 will turn on LED flash when detecting.

LED flash when decting will be a default configuration.

The configuration of LED flash mode is automatically saved after configured, and immediately working under the new configuration. KS101B/KS103/KS103S will be run according to the new configuration when restart.

Follow the attached files 3, the configuration code should be as follows:

*write_byte (0xe8, 2,0 xc1); //close LED flash*

*delayms (2000);*

KS101B/KS103/KS103S will be run according to the new configuration when restart.

**Sleep waiting time settings**

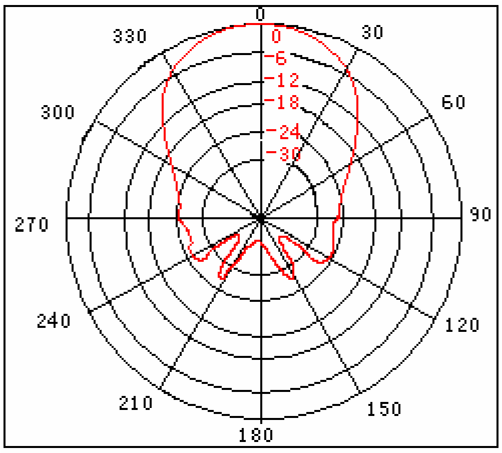
The default setting of sleep mode is to wait five seconds, KS101B/KS103/KS103S will enter sleep mode automatically if there’s no detecting command in five seconds. Another sleep mode is to wait one seconds. Send command 0xc5 through the $I^2C$ bus will change KS101B/KS103/KS103S to one seconds waiting sleep mode; and send command 0xc4 the five seconds waiting sleep mode can be restored.

The configuration of sleep mode is automatically saved after configured, and immediately working under the new configuration. KS101B/KS103/KS103S will be run according to the new configuration when restart.

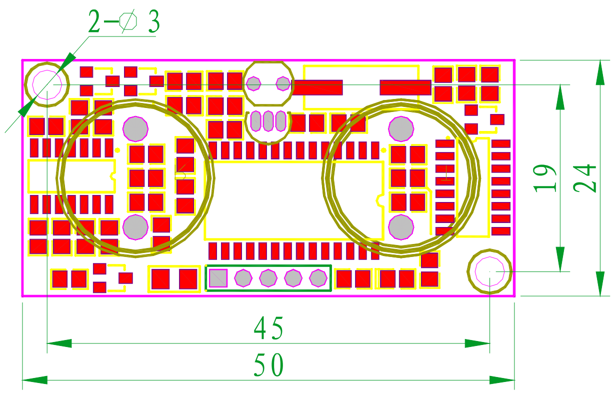
Follow the attached files 3, the configuration code should be as follows:

*write_byte (0xe8, 2,0 xc5); //one seconds waiting*
*delayms (2000);*
KS101B/KS103/KS103S will be run according to the new configuration when restart.

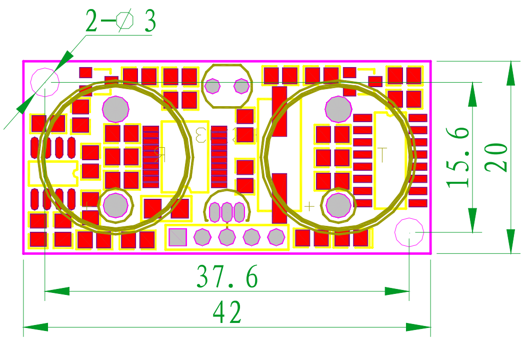
**Beam Angle:**
Tested at 40.0Khz frequency



**Fixing Size(Unit:mm)**
**KS101B:**



**KS103/KS103S:**



Suggest using M3 screw and M3 or Φ3 boss.

**Package details:**

1) Size of KS101B: 50mm×24mm×17mm
2) Size of KS103/KS103S: 42mm╳20mm╳17mm;
2) Weight：KS101B:11g; KS103/KS103S:9g.
3) Package size：KS101B:85mm×80mm×32mm(1PCS/each box)
4) Package weight：KS101B:75g 。

**Attached files:**

1) Use PIC16F877A to control KS101B/KS103/KS103S(Hardware $I^2C$)
2) Use PIC16F877A to control KS101B/KS103/KS103S(simulate $I^2C$)
3) Use 51 MCU to control KS101B/KS103/KS103S(simulate $I^2C$)
4) KS101B vedio show:
http://v.youku.com/v_show/id_XMjYwMjUwNTg4.html
Twenty KS101B working on I2C bus:
http://v.youku.com/v_show/id_XMjYxMzMxNDE2.html

1) Use PIC16F877A to control KS101B/KS103/KS103S(Hardware $I^2C$)
/*connection：PIC16F877A's IO PORT SCL、SDA conect to KS101B/KS103/KS103S's SCL、SDApin. PIC16F877A's SCL、SDA need a 4.7K resistance pull-up*/

```
#include <pic.h>                              //4MHz
__CONFIG(0x3d76);                             //WDT open
#define DELAY() delay(10)
#difine SCL RC3                   // a 4.7K resistance pull-up
#difine SDA RC4                   // a 4.7K resistance pull-up
void setup(void);
unsigned int detect_KS101B/KS103/KS103S(unsigned char ADDRESS, unsigned char command);
void delay(unsigned int ms);
void change_address(unsigned addr_old,unsigned char addr_new);
void send_command(unsigned char cmd);
void display(unsigned int distance,unsigned int delay);      //display function,you should apply it to the master
unsigned int distance;
void main(void)
{
        setup();
        //change_address(0xe8,0xe0);    //change default address 0xe8 to 0xe0
        while(1)
        {
                CLRWDT();
                distance = detect_KS101B/KS103/KS103S(0xe8,0xb4);    //Address:0xe8; command:0xb4.
                                                          //Get detect result from KS101B/KS103, 16 bit data.
                display(distance,100);                    //display function,you should apply it to the master
                delayms(200);
        }
}
void display(unsigned int distance,unsigned int delay);      //display function,you should apply it to the master
{
        CLRWDT();
}
void change_address(unsigned addr_old,unsigned char addr_new)
{
        SEN = 1;                                                  // send start bit to KS101B/KS103/KS103S
        while(SEN);                                               // wait for it to clear
        while(!SSPIF);                                  // wait for interrupt
        SSPIF = 0;                                      // then clear it.

        SSPBUF = addr_old;                          // KS101B/KS103/KS103S's I2C address
        while(!SSPIF);                              // wait for interrupt
        SSPIF = 0;                                  // then clear it.

        SSPBUF = 2;                                 // write the register number
        while(!SSPIF);                              // wait for interrupt
        SSPIF = 0;                                  // then clear it.

        SSPBUF = 0x9a;                  //command=0x9a, change I2C address, first sequence
        while(!SSPIF);
        SSPIF = 0;

        PEN = 1;                                        // send stop bit
        while(PEN);
    DELAY();                                // let KS101B/KS103/KS103S to break to do something

        SEN = 1;                                        // send start bit
        while(SEN);                                     // and wait for it to clear
        while(!SSPIF);
        SSPIF = 0;

        SSPBUF = addr_old;                          // KS101B/KS103/KS103S's I2C address
        while(!SSPIF);                              // wait for interrupt
        SSPIF = 0;                                  // then clear it.

        SSPBUF = 2;                                     // address of register to write to
        while(!SSPIF);                              //
        SSPIF = 0;

        SSPBUF = 0x92;                  //command=0x92, change I2C address, second sequence
        while(!SSPIF);                              //
        SSPIF = 0;

        PEN = 1;                                        // send stop bit
        while(PEN);                                     //
    DELAY();                                            // let KS101B/KS103/KS103S to break to do
    something
        SEN = 1;                                        // send start bit
        while(SEN);                                     // and wait for it to clear
        while(!SSPIF);
        SSPIF = 0;

        SSPBUF = addr_old;                      // KS101B/KS103/KS103S's I2C address
        while(!SSPIF);                              // wait for interrupt
        SSPIF = 0;                                  // then clear it.

        SSPBUF = 2;                                     // address of register to write to
        while(!SSPIF);                              //
        SSPIF = 0;

        SSPBUF = 0x9e;                  //command=0x9e,, change I2C address,third sequence
        while(!SSPIF);                              // wait for interrupt
        SSPIF = 0;                                  // then clear it.

        PEN = 1;                                        // send stop bit
        while(PEN);                                     //
    DELAY();                                            // let KS101B/KS103/KS103S to break to do
    something
        SEN = 1;                                        // send start bit
        while(SEN);                                     // and wait for it to clear
        while(!SSPIF);
        SSPIF = 0;

        SSPBUF = addr_old;                          // KS101B/KS103/KS103S's I2C address
        while(!SSPIF);                                  // wait for interrupt
        SSPIF = 0;                                      // then clear it.

        SSPBUF = 2;                                         // address of register to write to
        while(!SSPIF);                                  //
        SSPIF = 0;

        SSPBUF = addr_new;          //new address, it will be 0xd0~0xfe(without 0xf0,0xf2,0xf4,0xf6)
        while(!SSPIF);                                  //
        SSPIF = 0;

        PEN = 1;                                            // send stop bit
        while(PEN);                                         //
DELAY();                                                    // let KS101B/KS103/KS103S to break to do
something
}

unsigned int detect_KS101B/KS103/KS103S(unsigned char ADDRESS, unsigned char command)
{       // ADDRESS will be KS101B/KS103/KS103S's address such as 0xb0, command will be the detect command
such as 0xb4
unsigned int range=0;
        SEN = 1;                                            // send start bit
        while(SEN);                                         // and wait for it to clear
        while(!SSPIF);
        SSPIF = 0;

        SSPBUF = ADDRESS;                               // KS101B/KS103/KS103S's I2C address
        while(!SSPIF);                                  // wait for interrupt
        SSPIF = 0;                                      // then clear it.

        SSPBUF = 2;                                         // address of register to write to
        while(!SSPIF);                                  //
        SSPIF = 0;
        SSPBUF = command;

        while(!SSPIF);                                  //
        SSPIF = 0;
                                        //

        PEN = 1;                                            // send stop bit
        while(PEN);                                         //

        TMR1H = 0;                                      // delay while the KS101B/KS103/KS103S is
ranging
        TMR1L = 0;
        T1CON = 0x31;                                   //configuration of TIME1
        TMR1IF = 0;                                     //clean TIME1 interrupt flag
        while((!SCL) || (!TMR1IF))display(distance,100);    //you can delete the display function
        TMR1ON = 0;                                         // stop timer
        // finally get the range result from KS101B/KS103/KS103S
        SEN = 1;                                            // send start bit
        while(SEN);                                         // and wait for it to clear
        ACKDT = 0;                                          // acknowledge bit
        SSPIF = 0;

        SSPBUF = ADDRESS;                               // KS101B/KS103/KS103S I2C address
        while(!SSPIF);                                  // wait for interrupt
        SSPIF = 0;                                      // then clear it.
        SSPBUF = 2;                                     // address of register to read from - high byte of result
        while(!SSPIF);                                          //
        SSPIF = 0;                                              //

        RSEN = 1;                                               // send repeated start bit
        while(RSEN);                                            // and wait for it to clear
        SSPIF = 0;                                              //
        SSPBUF = ADDRESS+1;                     // KS101B/KS103/KS103S I2C address - the read bit is set this time
        while(!SSPIF);                                          // wait for interrupt
        SSPIF = 0;                                              // then clear it.
        RCEN = 1;                                               // start receiving
        while(!BF);                                     // wait for high byte of range
        range = SSPBUF<<8;                                      // and get it
        ACKEN = 1;                                                      // start acknowledge sequence
        while(ACKEN);                                           // wait for ack. sequence to end
        RCEN = 1;                                               // start receiving
        while(!BF);                                     // wait for low byte of range
        range += SSPBUF;                                        // and get it
        ACKDT = 1;                                                      // not acknowledge for last byte
        ACKEN = 1;                                                      // start acknowledge sequence
        while(ACKEN);                                           // wait for ack. sequence to end
        PEN = 1;                                                        // send stop bit
        while(PEN);                                                     //
        return range;
}

void send_command(unsigned char command)      //send a 8-bit command to KS101B/KS103/KS103S
{
        SEN = 1;                                                        // send start bit
        while(SEN);                                                     // and wait for it to clear
        while(!SSPIF);
        SSPIF = 0;
        SSPBUF = ADDRESS;                                       // KS101B/KS103/KS103S I2C address
        while(!SSPIF);                                          // wait for interrupt
        SSPIF = 0;                                              // then clear it.
        SSPBUF = 2;                                                     // address of register to write to
        while(!SSPIF);                                          //
        SSPIF = 0;
        SSPBUF = command;
        while(!SSPIF);                                          //
        SSPIF = 0;
        PEN = 1;                                                        // send stop bit
        while(PEN);                                                     //
}

void setup(void)                                //PIC16F877A's hardware init
{
        SSPSTAT = 0x80;
        SSPCON = 0x38;
        SSPCON2 = 0x00;
        SSPADD = 50;
        OPTION=0B10001111;                  //PSA = 1;    1:128    to WDT, WDT must clear in 32.64ms
        TRISC=0B00011000;
        PORTC=0x01;
        RBIE=0;
}

void delay(unsigned int ms)
{
  unsigned char i;
  unsigned int j;
  for(i=0;i<70;i++)
        for(j=0;j<ms;j++)CLRWDT();
```

```
}
```

2) Use PIC16F877A to control KS101B/KS103/KS103S(simulate $I^2C$)

```
#include <pic.h>                  //4MHz
__CONFIG(XT&WDTEN); //WDT open
#define    SDA    RD6          // a 4.7K resistance pull-up
#define    SCL    RD5          // a 4.7K resistance pull-up
#define    SDAPORT    TRISD6 //
#define    SCLPORT    TRISD5     // RD6, RD5 can change to any other I/O port
void delay(void)                          //short delay
{
        unsigned char k;
        for(k=0;k<180;k++)
                asm("CLRWDT");
}
void delayms(unsigned char ms) //delay some ms
  {
        unsigned int i,j;
        for (i=0;i<ms;i++)
                for(j=0;j<110;j++)
                asm("CLRWDT");
}
void i2cstart(void)    // start the i2c bus
{
        SCLPORT=0;
        SDAPORT=0;
        SCL=1;
        asm("NOP");      asm("NOP");      asm("NOP");      asm("NOP");      asm("NOP");
        SDA=1;
        delay();
        SDA=0;
        delay();
        SCL=0;
        delay();
}
void i2cstop(void)    // stop the i2c bus
{
        SDA=0;
        SCLPORT=0;
        SDAPORT=0;
        SDA=0;
        asm("NOP");      asm("NOP");      asm("NOP");      asm("NOP");      asm("NOP");
        SCL=1;
        delay();
        SDA=1;
        delay();
}
void bitin(void)      //read a bit from i2c bus
{
        eepromdi=1;
        SCLPORT=0;
        SDAPORT=1;
        SCL=1;
        asm("NOP");      asm("NOP");      asm("NOP");      asm("NOP");      asm("NOP");
        eepromdi=SDA;
        asm("NOP");      asm("NOP");      asm("NOP");      asm("NOP");      asm("NOP");
        SCL=0;
        asm("NOP");      asm("NOP");      asm("NOP");      asm("NOP");      asm("NOP");
}
void bitout(void)    //write a bit to i2c bus
{
        SCLPORT=0;
        SDAPORT=0;
        SDA=eepromdo;
        asm("NOP");     asm("NOP");     asm("NOP");     asm("NOP");     asm("NOP");
        SCL=1;
        asm("NOP");     asm("NOP");     asm("NOP");     asm("NOP");     asm("NOP");
        SCL=0;
        asm("NOP");     asm("NOP");     asm("NOP");     asm("NOP");     asm("NOP");
}
void i2cwrite(unsigned char sedata)    //write a byte to i2c bus
{
        unsigned char k;
        for(k=0;k<8;k++)
        {
        if(sedata&0x80)
        {
                eepromdo=1;
        }
        else
        {
                eepromdo=0;
        }
        sedata=sedata<<1;
        bitout();
        }
        bitin();
}
unsigned char i2cread(void)          //read a byte from i2c bus
{
        unsigned char redata;
        unsigned char m;
        for(m=0;m<8;m++)
        {
        redata=redata<<1;
        bitin();
        if(eepromdi==1)
        {
                redata|=0x01;
        }
        else
        {
                redata&=0xfe;
        }
        asm("NOP");
        }
        eepromdo=1;
        bitout();
        return redata;
}

unsigned char KS101B/KS103/KS103S_read(unsigned char address,unsigned char buffer)
//////////////read register: address + register ,there will be 0xe8 + 0x02/0x03
{
        unsigned char eebuf3;
//      unsigned int range;
        i2cstart();
        i2cwrite(address);
        i2cwrite(buffer);
        i2cstart();
        i2cwrite(address+1);
        i2cstart();
        eebuf3=i2cread();
        i2cstop();
        return eebuf3;
}
```

```
void KS101B/KS103/KS103S_write(unsigned char address,unsigned char buffer,unsigned char command)
  /////////////////write a command: address + register + command,there will be 0xe8 + 0x02 + 0xb0
{
        i2cstart();
        i2cwrite(address);
        i2cwrite(buffer);
        i2cwrite(command);
        i2cstop();
}
void change_i2c_address(addr_old,addr_new) // addr_old is the address now, addr_new will be the new address
{                                                              //that you want change to
    delayms(200);                                         //Protect the eeprom,you can delete this
    KS101B/KS103/KS103S_write( (addr_old,2,0x9a);
    delayms(1);
    KS101B/KS103/KS103S_write(addr_old,2,0x92);
    delayms(1);
    KS101B/KS103/KS103S_write(addr_old,2,0x9e);
    delayms(1);
    KS101B/KS103/KS103S_write(addr_old,2, addr_new);
    delayms(100);                                         //Protect the eeprom,you can delete this
}

unsigned int detect_KS101B/KS103/KS103S(unsigned char address, unsigned char command)
{
unsigned int range1;
        KS101B/KS103/KS103S_write(address,2,command);
        delayms(1);
        delayms(80);      //this delat should be longer if detecting the temperature，and can be delete
        //SCLPORT=1;while(!SCL);
        // delayms(80) can change to " SCLPORT=1;while(!SCL);" to improve the detection efficiency
        range1 = KS101B/KS103/KS103S_read(address,2);
        range1 =(range1<<8) + KS101B/KS103/KS103S_read(address,3);
        delayms(5);
        return range1;
}

void main(void)
{
   unsigned int range;
//change_i2c_address(0xe8,0xfe);     //change default address 0xe8 to 0xfe
        delayms(200);
        while(1)
        {
                asm("CLRWDT");
                range = detect_KS101B/KS103/KS103S(0xe8,0xb4);    //you just need the only one sentence to get the
range.
                delayms(200);
        }
}
```

## 3) Use 51 MCU to control KS101B/KS103/KS103S(simulate I$^2$C)

```
#include <reg51.h>                          //12.0MHz
#include <intrins.h>
sbit SDA=P3^6;                              // a resistance 4.7k pull-up
sbit SCL=P3^7;                              // a resistance 4.7k pull-up
unsigned int range;
void display(unsigned int range)
{
        //input your display function,please.
}
void delay(void)                       //short delay
```

```
 {
        _nop_(); _nop_(); _nop_(); _nop_();
        _nop_(); _nop_(); _nop_(); _nop_();
        _nop_(); _nop_(); _nop_(); _nop_();
        _nop_(); _nop_(); _nop_(); _nop_();
 }

 void start(void)                    //I2C start
 {
         SDA = 1;
        delay();
        SCL = 1;
        delay();
        SDA = 0;
        delay();
 }

 void stop(void)                           //I2C stop
 {
        SDA = 0;
        delay();
        SCL = 1;
        delay();
        SDA = 1;
        delay();
 }

 void ack(void)                           //ack
 {
        unsigned char i;
        SCL = 1;
        delay();
        while(SDA == 1 && i < 200)
        {
               i++;
        }
        SCL = 0;
        delay();
 }

 void no_ack()                            //not ack
{
        SDA = 1;
        delay();
        SCL = 1;
        delay();
        SCL = 0;
        delay();
}

void i2c_write_byte(unsigned char dat)          //write a byte
 {
        unsigned char i;
        SCL = 0;
        for(i = 0; i < 8; i++)
        {
               if(dat & 0x80)
               {
                      SDA = 1;
               }
               else
               {
                      SDA = 0;
        }
        dat = dat << 1;
        delay();
        SCL = 1;
        delay();
        SCL = 0;
        delay();
    }
    SDA = 1;
    delay();
}

unsigned char i2c_read_byte(void)       //read a byte
{
    unsigned char i,dat;
    SCL = 0;
    delay();
    SDA = 1;
    delay();
    for(i = 0; i < 8; i++)
    {
        SCL = 1;
        delay();
        dat = dat << 1;
        if(SDA == 1)
        {
            dat++;
        }
        SCL = 0;
        delay();
    }
    return dat;
}

void init_i2c(void)                     //i2c init
{
    SDA = 1;
    SCL = 1;
}

void write_byte(unsigned char address,unsigned char reg,unsigned char command) //address+register+command
{
    init_i2c();
    start();
    i2c_write_byte(address);
    ack();
    i2c_write_byte(reg);
    ack();
    i2c_write_byte(command);
    ack();
    stop();
}

unsigned char read_byte(unsigned char address,unsigned char reg)    //address(with bit 0 set) + register
{
   unsigned char dat;
   init_i2c();
   start();
   i2c_write_byte(address);
   ack();
   i2c_write_byte(reg);
   ack();
   start();
```

```
       i2c_write_byte(address+1);
       ack();
       dat = i2c_read_byte();
       no_ack();
       stop();
       return dat;
  }

void delayms(unsigned int ms)        //delay ms
{
        unsigned char i;
        unsigned int j;
        for(i=0;i<110;i++)
                for(j=0;j<ms;j++);
}

void change_i2c_address(unsigned char addr_old,unsigned char addr_new)
// addr_old is the address now, addr_new will be the new address
{                                                        //that you want change to
    delayms(2000);                                        // Protect the eeprom ,you can delete this sentence
    write_byte(addr_old,2,0x9a);
    delayms(1);
    write_byte(addr_old,2,0x92);
    delayms(1);
    write_byte(addr_old,2,0x9e);
    delayms(1);
    write_byte(addr_old,2, addr_new);
    delayms(500);                                       //Protect the eeprom, you can delete this sentence
}

unsigned int detect(unsigned char address,unsigned char command)    //0xe8(address) + 0xb0(command)
{
        unsigned int distance,count;
        write_byte(address,2,command);              //use command "0xb0" to detect the distance
        delayms(1);                                                 //delay
        //delayms(80);                             //the delay should follow the time show in table 1
        count=800;while(--count || !SCL)display(range);              //wait for detecting end
//      while(!SCL)display(range);      //you can delete "display(range)"
//also can use while(!SCL)
        distance=read_byte(address,2);
        distance <<= 8;
        distance += read_byte(address,3);
        return distance;                           //return 16 bit distance in millimeter
}

void main(void)
{
        //change_i2c_address(0xe8,0xfe);    //change default address 0xe8 to 0xfe
        while(1)
        {
                range = detect(0xe8,0xb0);
                //0xe8 is the address; 0xb0 is the command.you just need the only one sentence to get the range.
                //display(range);
                delayms(200);
        }
}
```