

KS114 技术说明书

版本: Ver. 1.05

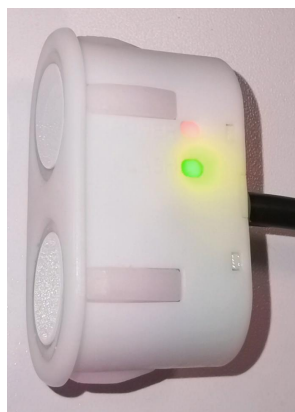
创新技术源自导向技术

深圳市导向机电技术有限公司

Dauxi Technologies Co., Ltd. All rights reserved.

Modify Date	Content	Edit	Revsion	Note
Nov. 22, 2019	Parameter revision	L.Q.B.	1.00	Initial release.
Jun.3,2020	Beam angle added.	X.T.C.	1.01	Page 5.
Nov.23,2020	Increase the content of reference assembly dimensions.	X.Y.C.	1.02	Please refer to Page 23.
Dec.21,2020	Increase the ID and the protocol with verification.	X.Y.C.	1.03	Please refer to Page 17.
Sep.12,2021	The assembly size of the KS114 increases the control tolerance.	Y.M.G.	1.04	Please refer to Page 23.
Sep.6,2022	Add command "0x14" "0x1e"	X.T.C.	1.05	Page 5,8,18

:



目录

KS114 功能摘要:	3
KS114 电性能参数	3
编码规则注解	4
探测范围图	5
I ² C 模式	6
KS114 连线及说明:	6
修改 I2C 地址时序:	7
KS114 工作流程:	7
探测指令	8
距离探测	11
电源降噪指令 (0x70, 0x71, 0x72, 0x73, 0x74, 0x75)、波特率指令 (0x77/0x78/0x79) 波束角配置指令 (0x7a-0x7e/0x80/0x81/0x82)	11
时序图	12
休眠等待时间设置	13
TTL 串口及 485 串口模式	14
TTL 及 485 串口模式接线	14
修改串口地址时序	15
KS114 工作流程	15
探测结束智能识别	17
探测指令	17
电源降噪指令 (0x70, 0x71, 0x72, 0x73, 0x74, 0x75)、波特率指令 (0x77/0x78/0x79) 波束角配置指令 (0x7a-0x7e/0x80/0x81/0x82)	20
时序图	21
完整发货清单	21
KS114 装配尺寸图:	22
安装孔建议:	23
装配注意事项:	23
使用建议:	24
附件-例程:	25

KS114 功能摘要:

- IP65 防水 (IP67 防水型号 KS204/KS214), 收发分体式设计, 针对 KS106/KS106A/KS136/KS136A 的双探头版本设计;
- 主板与探头封装在一起, 输出信号为数字量距离信号, 与 KS103 类似;
- 可软件配置的波束角水平 100° 竖直 50° 或水平 90° 竖直 40° ;
- 探测范围: **1cm-3m (常用 1cm-1.5 米。1cm-5m 范围可以订做但不常用)**; 精度: **2cm**;
- **默认 485 接口 (说明书第 14 页)**, 可订做兼容 KS103 协议的 I²C 接口, 可订做 TTL 接口;
- 可定制 MODBUS RTU 接口, 可定制 CAN 接口;
- 共 20 个可修改的 I²C/TTL/485 地址, 范围为 0xd0 ~ 0xfe (0xf0,0xf2,0xf4,0xf6 除外, 8 位地址);
- 5s 未收到 I²C 控制指令自动进入 uA 级休眠, 并可随时被主机 I²C 控制指令唤醒;
- 使用工业级配置, 工作温度 (-30°C~+85°C);
- 工作电压 **3-5.5V (推荐标准 5V 供电, 另有 9-24V 供电电压可选)** ;
- I²C 模式通信速率 50~100kbit/s; 串口通信速率默认 115200bps; 用户可修改为 9600bps;
- 采用了自体声波识别技术, 提高抗干扰性;
- 环保无铅;
- 可订做单开关量输出、双开关量输出;
- 可靠性: 5 万小时不良率低于百万分之二。

KS114 电性能参数

供电电压: **3-5.5V** 直流电源 (**另有 9-24V 供电电压可选, 现货**)。

启动时瞬间最大电流: **10mA@5.0V, typical**, 持续时间 10 秒。灯闪烁结束后进入待机状态, 电流: **<5mA@5.0V, typical**. 收到有效控制指令后 KS114 进入工作状态, 功耗为: **15mA@5.0V 每 10ms 探测一次; 12mA@5.0V 每 25ms 探测一次; 10mA@5.0V 每 100ms 探测一次; 5mA@5.0V 每 200ms 探测一次;**

另有 9-24V 供电电压可选, 功耗低于 20mA@12.0V.

关于波束角的说明

波束角对于 0xb0 指令恒定水平 100±15° 竖直 50±10° 波束角 (KS114 的宽度 26.5mm 尺寸方向为水平方向; KS114 的长度 50.56mm 尺寸方向为竖直方向), 测试障碍物为 2cm 棒状物。

波束角对于 0x05 指令、0x0a 指令、0x0f 指令出厂默认为水平 90±15° 竖直 40±10° (0x81 配置)。可配置为 0x80 模式的水平 100±15° 竖直 50±10° 波束角, 或配置为 0x82 模式的水平 85±15° 竖直 35±10° 波束角。

接线说明

KS114 引出线 4 根, 颜色依次为: 红色, 黑色, 白色, 黄色。

红色请连接至 **3-5.5V** 电源正极;

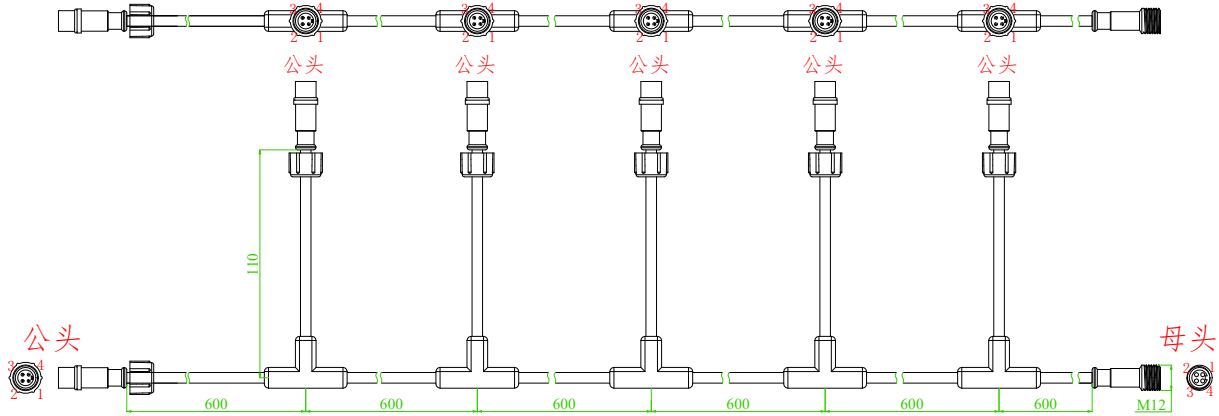
黑色请连接至电源负极或 GND;

白色请连接 I2C 模式的 SCL 或 TTL 模式的 RXD 或 485 模式的 485B。

黄色请连接 I2C 模式的 SDA 或 TTL 模式的 TXD 或 485 模式的 485A；

KS114 工作于何种模式，出厂即固化好，无法修改。因此，客户购买前应注明需要 KS114-I2C 或 KS114-TTL 或 KS114-485 或 KS114-MODBUS RTU。默认出厂为 KS114-485 接口。

当有多台 KS114 时，推荐采用总线式安装，这样无论安装多少台 KS114，只需一条 4 芯总线即可，如下图所示（可接 6 台 KS114，下图尺寸型号为：LINE-3T-6-600-110-CNT）：



编码规则注解

KS114-L15-C702-L2M-B4Y

其中：

KS114：为型号；

L15：代表 KS114 伸出线长度 $15 \pm 3\text{cm}$ ；

C702：代表黑色。另有 C100 代表白色可选，C101 代表银色可选，C107 代表黄色可选；其他颜色可以订制；

L2M：KS114 的小 4 芯通信线，默认 2 米；其他长度例如 0.5 米，则编号为 L0.5M，

B4Y：KS114 的小 4 芯通信线上的端子编码。可以换其他端子或不配端子。

例如仅仅采购黑色 KS114 不需要配通信线，则编码为：**KS114-L15-C702**

例如采购的 KS114 配通信线但不需要配端子，则编码为：**KS114-L15-C702-L2M**

如果需要采购 9~24V 供电的版本，需要在尾部加上“-12V”的尾缀即可。

示例：**KS114-L15-C100-L2M-B4Y-12V**，其代表支持 9-24V 供电的白色 KS114。

探测范围图（探测范围图为圆锥体截面投影。波束角为下图倒三角的角度值）

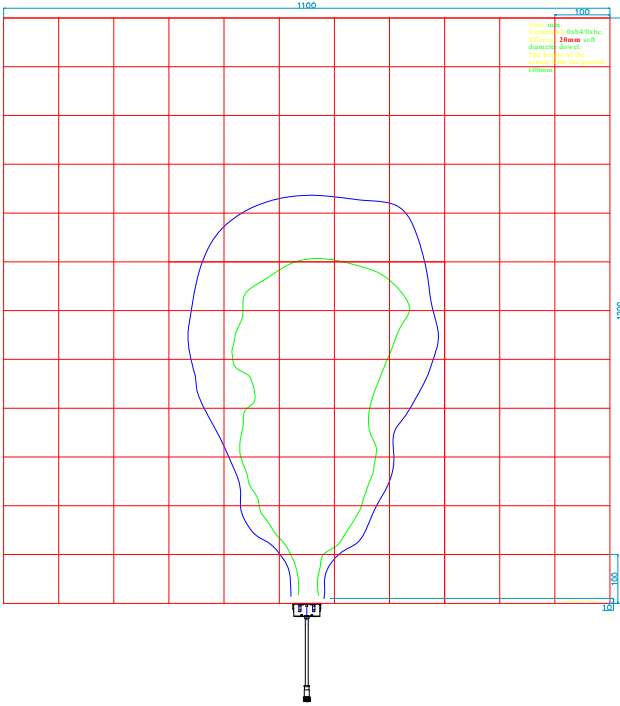


图 B1 (竖直方向波束角, KS114 的长度 50.56mm 尺寸方向)

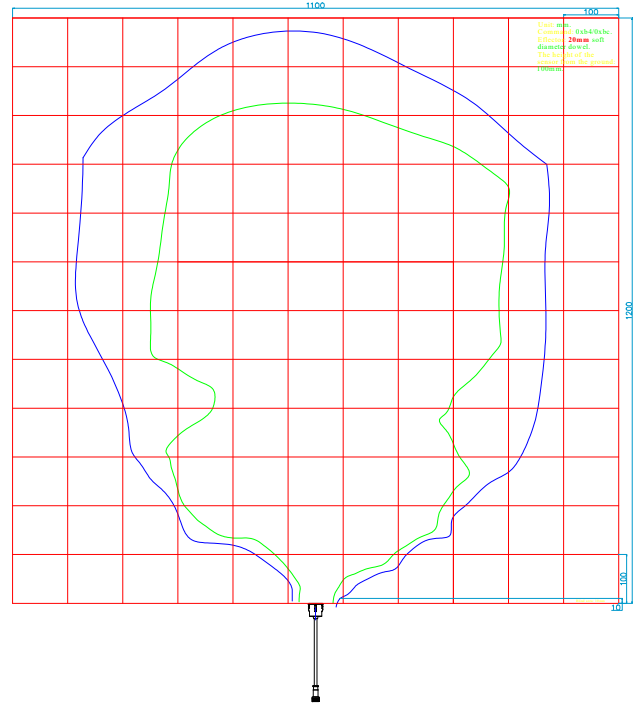


图 B2 (水平方向波束角, KS114 的宽度 26.5mm 尺寸方向)

图号	条件	反射物大小	材料	探头离地高度 (地面不能有障碍物如瓷砖缝隙、凸起等)	电压	波束角方向	波束角级别	波束角大小	备注
图 B1		直径 20mm	PVC	100mm	5V	竖直方向波束角为 KS114 的长度 50.56mm 尺寸方向	0x71/0x7b 出厂默认	蓝色 50±10° ; 绿色 40±10°	蓝色线为 0xb0/0xb2/0xb4/0x14/0x1e 指令的探测范围; 绿色线为其他指令 (0x05,0x0a,0x0f,0xb8,0xbc) 配置为 0x81 时的探测范围, 如果其他指令需要波束角由绿色放大到蓝色, 则配置为 0x80 即可。
图 B2		圆棒				水平方向波束角为 KS114 的宽度 26.5mm 尺寸方向		蓝色 100±15° ; 绿色 90±15°	

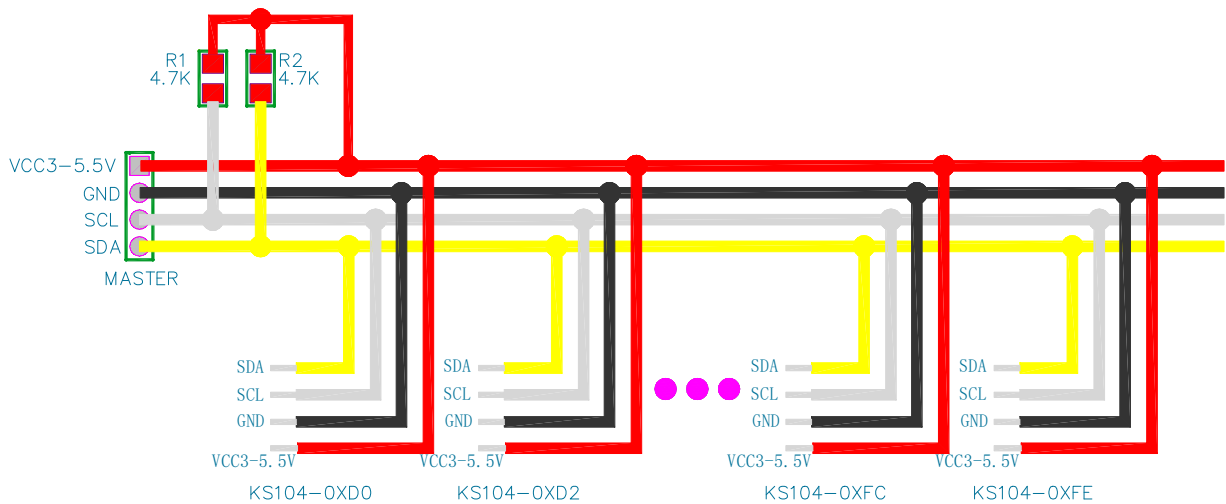
I²C 模式



KS114 连线及说明:

KS114 的 I2C 通信线 SCL,SDA 内部无上拉电阻,客户的主机需要 SCL 及 SDA 线均需要由主机接一个 4.7K(阻值 1~10K 均可)电阻到 VCC (必须为 5V)。

接线图如下所示:红色线接电源正极 3~5.5V,黑色线接电源负极 GND,白色线接 SCL,黄色线接 SDA,。**注意:请勿带电操作,接好线再上电!如果需要带电操作,请先连接上电源负极 GND 后再接其他线。**



KS114 的 I²C 通信速率建议不要高于 100kbit/s。

KS114 默认地址为 0xe8,用户可以将地址修改为 20 种地址中的任何一个:0xd0, 0xd2, 0xd4, 0xd6, 0xd8, 0xda, 0xdc, 0xde, 0xe0, 0xe2, 0xe4, 0xe6, 0xe8, 0xea, 0xec, 0xee, 0xf8, 0xfa, 0xfc, 0xfe. ⁽¹⁾

Note 1: 请注意，以上地址并不包括 0xf0, 0xf2, 0xf4, 0xf6，这 4 个地址保留用于 I²C 从机的 10 位地址。控制本模块的主机设备可能只支持 7 位的 I²C 从机地址，此时需要将 8 位地址右移 1 位作为地址来使用。例如，本模块默认地址 0xc8，对应 7 位的地址 0x74。

修改 I2C 地址时序：

地址	2	0x9a	延时 1ms	地址	2	0x92	延时 1ms	地址	2	0x9e	延时 1ms	地址	2	新地址	延时 100ms
----	---	------	-----------	----	---	------	-----------	----	---	------	-----------	----	---	-----	-------------

修改 I²C 地址须严格按照时序来进行，时序中的延时时间为最小时间。对于 51 单片机主机，其可调用附件 3 所示的 `change_i2c_address(addr_old, addr_new)` 函数来实现。

修改完毕后请给 KS114 重新上电，可观察到 LED 显示新地址。在修改 KS114 的 I²C 地址过程中，严禁突然给 KS114 断电。修改地址函数请不要放在 `while(1)` 循环中，保证在程序中上电后只运行一次。

在 I²C 地址设置为不同之后，在主机的两根 I²C 总线上可以同时连接 20 个 KS114。主机在对其中一个 KS114 模块进行控制时，其他模块自动进入微瓦级功耗休眠模式，因此不必担心电流供应不足问题。

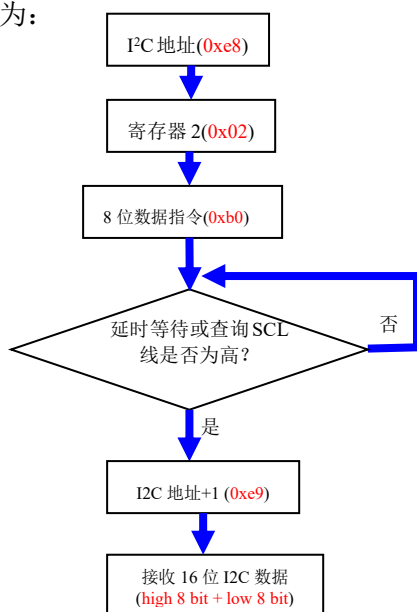
KS114 工作流程：

在 KS114 上电启动时，系统会开始自检，自检正常后，KS114 尾部引出线附件的红色 LED 会以二进制方式闪烁显示其 8 位 I²C 地址，快闪两下代表“1”，慢闪一下代表“0”。例如显示 0xea 地址，其二进制数为 0B11101010，绿色 LED 飞闪两下→快闪两下→灭→快闪两下→灭→快闪两下→灭→慢闪一下→灭→快闪两下→灭→慢闪一下→灭。⁽³⁾

Note 3: LED 闪烁时的绿色亮光可能会刺激到眼睛，请尽量不要近距离直视工作中的 LED，可以使用眼睛的余光来观察其闪烁。

KS114 启动后如果收到主机的有效数据指令，LED 将立即停止闪烁显示。进入指令探测模式。

KS114 使用 I²C 接口与主机通信，自动响应主机的 I²C 控制指令。指令为 8 位数据，指令发送流程为：



探测结束智能识别

KS114 屏蔽了此功能。

探测指令

探测指令发送完成后，KS114 将依据探测指令进入相应探测模式，主机此时须等待一段时间方可开始通过 I²C 总线查询探测结果，过早查询 I²C 总线将获得 0xff 值。注意，每一帧探测指令格式均为：

I ² C 地址	寄存器 2	8 位数据
---------------------	-------	-------

所有 I²C 控制指令汇总如下：

寄存器	命令	返回值范围 (10 进制)	返回值范围 (16 进制)	备注
0		1~254	0x01~0xff	程序版本标识及厂家标识。 可参考附件 3 示例函数，返回值=read_byte(0xe8,0);
1		1~252	0x01~0xfc	制造日期标识。16 位数据的高 8 位为制造年份，低 8 位为制造月份。11 年开始制造标识为 1；12 年开始制造标识为 2；……；25 年开始制造标识为 F；26 年开始制造标识为 0；27 年开始制造标识为 1。月份：1 月份标识为 1；以此类推，10 月份标识为 A；12 月份对应 C。 可参考附件 3 示例函数，返回值=read_byte(0xe8,1);
2	0x05	1-2817us	0x01-0xb01us	默认有效探测范围 1cm-50cm。返回 us 值，除以 58 可获得 cm 距离值。本指令为高速指令，执行时间不超过 9ms。
2	0x0a	1-3840us	0x01-0xf00us	默认有效探测范围 1cm-70cm。返回 us 值，除以 58 可获得 cm 距离值。本指令为高速指令，执行时间不超过 9ms。
2	0x0f	1-6273us	0x01-0x1881us	默认有效探测范围 1cm-110cm。返回 us 值，除以 58 可获得 cm 距离值。本指令为高速指令，执行时间不超过 9ms。
2	0xb0	30-5657mm	0x1e-0x1619mm	默认有效探测范围 1cm-5m。返回 mm 值。本指令执行时间不超过 33ms。
2	0x14	30-2016mm	0x1e-0x7e0mm	默认有效探测范围 1cm-2m。返回 mm 值。本指令执行时间不超过 15ms。
2	0x1e	30-3009mm	0x1e-0xbc1mm	默认有效探测范围 1cm-3m。返回 mm 值。本指令执行时间不超过 21ms。
2	0xb1	0-255	0-0xff	只发射一束波，无其他功能。返回值是寄存器 2 和寄存器 3 的上一次的储存值
2	0xb2	1-32639us	0x01-0x7f7f μs	有效探测范围 1cm-5m。返回 us 值
2	0xb3	1-32639us	0x01-0x7f7f μs	只接收回波，有效探测范围 1cm-5.6 米，与 0xb1 指令配合使用，用于 2 台 KS104 之间的对射测距。
2	0xb4	30-5657mm	0x1e-0x1619mm	默认有效探测范围 1cm-5m。返回 mm 值。
2	0xb8	30-5657mm	0x1e-0x1619mm	默认有效探测范围 1cm-5m。返回 mm 值。
2	0xba	1-32639us	0x01-0x7f7f μs	有效探测范围 1cm-5m。返回 us 值

2	0xbc	30-5657mm	0x1e-0x1619mm	有效探测范围 1cm-5m。返回 mm 值
2	0x70	无	无	第一级降噪。 所有指令指令将工作于第一级降噪。适用于电池供电
2	0x71	无	无	第二级降噪。 所有指令指令将工作于第一级降噪。出厂默认设置，适用于电池供电
2	0x72	无	无	第三级降噪。 所有指令指令将工作于第一级降噪。适用于 USB 供电。
2	0x73	无	无	第四级降噪。 所有指令指令将工作于第一级降噪。适用于较长距离 USB 供电。
2	0x74	无	无	第五级降噪。 所有指令指令将工作于第一级降噪。适用于开关电源供电
2	0x75	无	无	第六级降噪。 所有指令指令将工作于第一级降噪。适用于开关电源供电
2	0x77	无	无	将串口通信波特率配置为 9600bps
2	0x78	无	无	将串口通信波特率配置为 57600bps
2	0x79	无	无	将串口通信波特率配置为 115200bps，出厂默认设置
2	0x7a	无	无	波束角配置指令
2	0x7b	无	无	出厂默认设置。用于配置盲区。
2	0x7c	无	无	波束角配置指令
2	0x7d	无	无	波束角配置指令
2	0x7e	无	无	波束角配置指令
2	0x80	无	无	配置 0x05 指令 0x0a 指令 0x0f 指令的波束角为水平 100° 竖直 50°
2	0x81	无	无	配置 0x05 指令 0x0a 指令 0x0f 指令的波束角为水平 90° 竖直 40°，为出厂默认设置
2	0x82	无	无	配置 0x05 指令 0x0a 指令 0x0f 指令的波束角为水平 80° 竖直 35°
2	0x8A	无	无	配置串口协议，参考串口部分

2	0x8B	无	无	配置串口协议，参考串口部分
2	0x8F	无	无	配置串口协议，参考串口部分
2	0x95	无	无	0x70~0x8f 参数配置第二时序
2	0x98	无	无	0x70~0x8f 参数配置第三时序
2	0x9c	无	无	0x70~0x8f 参数配置第一时序
2	0x92	无	无	修改地址第二时序
2	0x9a	无	无	修改地址第一时序
2	0x9e	无	无	修改地址第三时序
2	0xc4	无	无	5 秒休眠等待
2	0xc5	无	无	1 秒休眠等待
2		0~255	0~0xff	读数据时寄存器 3 与寄存器 2 联合使用，寄存器 2 返回 16 位数据探测结果的高 8 位，寄存器 3 返回 16 位数据的低 8 位。 必须在发送完 地址+寄存器 2+探测指令 后方可查询本寄存器返回值。 可参考附件 3 示例函数，返回值=read_byte(0xe8,2);
3		0~255	0~0xff	读数据时寄存器 3 与寄存器 2 联合使用，寄存器 2 返回 16 位数据探测结果的高 8 位，寄存器 3 返回 16 位数据的低 8 位。 必须在发送完 地址+寄存器 2+探测指令 后方可查询本寄存器返回值。 可参考附件 3 示例函数，返回值=read_byte(0xe8,3);
4			0x77~0x79	本寄存器存储的是串口通信波特率 0x77~0x79，供查询用。0x77 对应波特率 9600bps；0x78 对应波特率 57600bps；0x79 对应波特率 115200bps。 可参考附件 3 示例函数，返回值=read_byte(0xe8,4);
5			0xd0~0xfe	本寄存器存储的是 20 个 I ² C 或串口地址，不包括 0xf0, 0xf2, 0xf4, 0xf6，供查询用。 可参考附件 3 示例函数，返回值=read_byte(0xe8,5);
6			0x70~0x75	本寄存器存储的是降噪级别 0x70~0x75，供查询用。默认 0x71。 可参考附件 3 示例函数，返回值=read_byte(0xe8,6);
7			0x7a~0x7e	本寄存器存储的是盲区配置，默认 0x7b。 可参考附件 3 示例函数，返回值=read_byte(0xe8,7);
8			0xe0	保留供升级用
			0xe1	保留供升级用

			0xe2	保留供升级用
			0xe3	保留供升级用
9			0x6a	初始化进行中
			0x69	初始化结束标志。 可参考附件 3 示例函数，返回值=read_byte(0xe8,9);
10		0~255	0~0xff	初始化温度高 8 位，未开放 可参考附件 3 示例函数，返回值=read_byte(0xe8,10);
11		0~255	0~0xff	初始化温度低 8 位，未开放 可参考附件 3 示例函数，返回值=read_byte(0xe8,11);
12		0~255	0~0xff	当前环境声速高 8 位，未开放 可参考附件 3 示例函数，返回值=read_byte(0xe8,12);
13		0~255	0~0xff	当前环境声速低 8 位，单位：mm/100ms，未开放 可参考附件 3 示例函数，返回值=read_byte(0xe8,13);
15~36				保留供升级用

表 1

距离探测

具体参数及控制指令请参见上表 1。

通过“I²C 地址 + 寄存器 2 + 距离探测指令”时序，延时或等待上表中规定的相应时间后，再使用读取函数读寄存器 2 及寄存器 3 的值，即可取得 16 位的距离数据。返回 mm 距离值是按照当前环境温度值换算而来的距离值；返回 us 值代表超声波从发出到遇到障碍物反射收回所经历的时间。

电源降噪指令 (0x70, 0x71, 0x72, 0x73, 0x74, 0x75)、波特率指令 (0x77/0x78/0x79)

波束角配置指令 (0x7a-0x7e/0x80/0x81/0x82)

KS114 默认电源推荐使用电池供电。如果使用噪音较大的电源，测距值可能会出现不稳定的波动。用户可以通过发送 0x70, 0x71, 0x72, 0x73, 0x74, 0x75 命令来配置 KS114 测距模块的杂波抑制功能。0x70 为测试用级别，**0x71** 指令将使本模块配置为第一级降噪，适用于电池供电的场合，同时也是**出厂默认设置**。0x72 指令将使本模块配置为第二级降噪，适用于 USB 供电等有一定高频噪音的场合。0x73 指令将使本模块配置为第三级降噪，适用于较长距离 USB 供电的场合。0x74 指令将使本模块配置为第四级降噪，适用于开关电源供电的场合。0x75 指令将使本模块配置为第五级降噪，如无特别要求，不推荐配置为本级。

用户可以通过发送 0x7a, 0x7b, 0x7c, 0x7d, 0x7e 来配置波束角，值越大波束角越小。**出厂默认**为 0x7b。参见表 1。

配置方法非常简单，向本模块发送指令时序：“I²C 地址 + 寄存器 2 + 0x9c; I²C 地址 + 寄存器 2 + 0x95; I²C 地址 + 寄存器 2 + 0x98; I²C 地址 + 寄存器 2 + 0x70/0x71/0x72/0x73/0x74/0x75/0x7a/0x7b/0x7c/0x7d/0x7e/0x80/0x81/0x82”即可，发送完成后请延时至少 2 秒，以让系统自动完成配置。并开始按照新配置工作。

以附件 3 所示程序为例，将本模块配置为二级降噪，配置代码如下：

```
config_0x71_0x7d(0xe8,0x72); //如果 I2C 地址为 0xe8
```

delays(2000);

将本模块配置为最大波束角，配置代码如下：

config_0x71_0x7d(0xe8,0x7a); //如果PC地址为0xe8

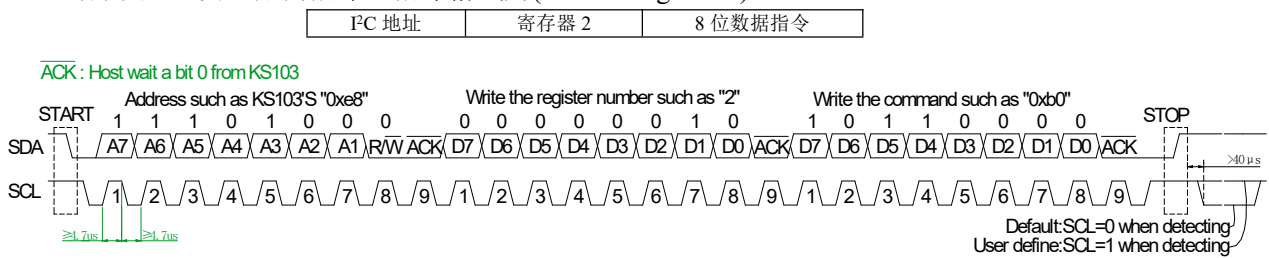
delays(2000);

配置代码请放在程序的初始化函数中，即 while(1)循环之前，以保护模块。KS114 收到有效配置指令之后，LED 灯将长亮 5s，表明配置成功。

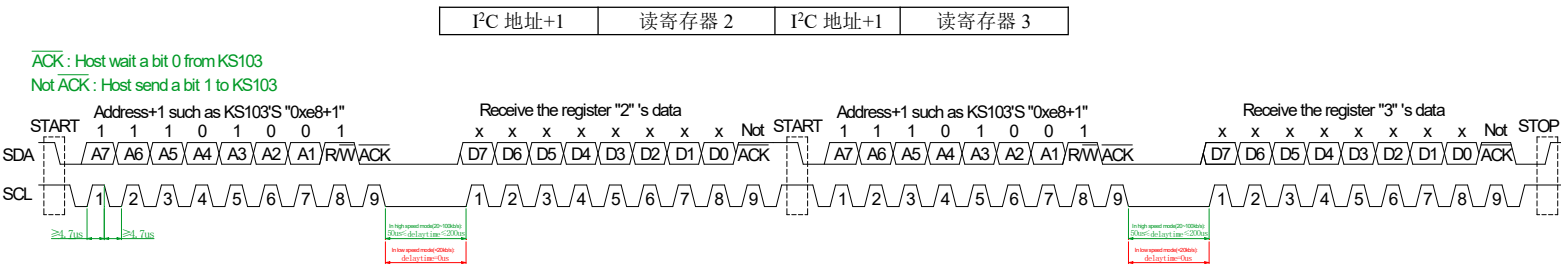
KS114 在重新上电后将永久性按新配置运行。无须再次配置。

时序图

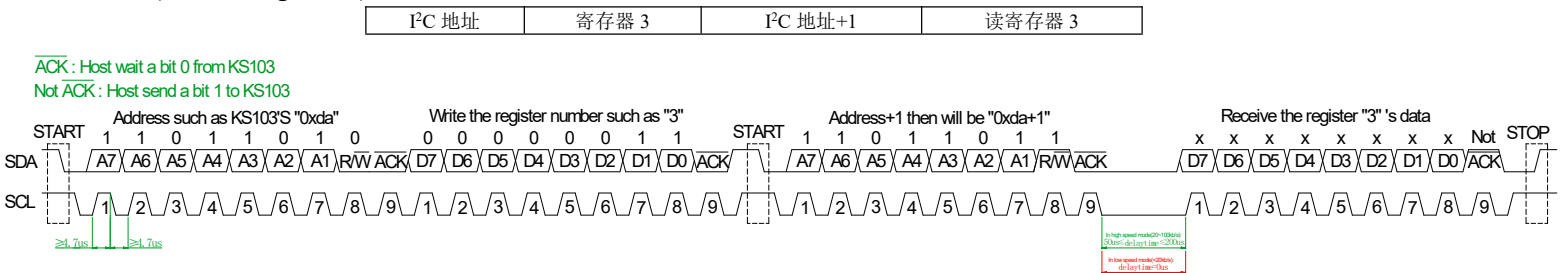
时序图 1：发送探测指令，指令格式为(Such as register 2):



时序图 2：执行完时序图 1 后，等待 SCL 变高或延时 100ms 后接收 16 位数据，先高位后低位，指令格式为：



时序图 3：执行完时序图 1 后，等待 SCL 变高或延时 100ms 后接收寄存器 x 的数据(本例为寄存器 3)，读任意寄存器指令格式(Such as register 3): ⁽⁵⁾



Note 5: 采用读任意寄存器指令时，如果读寄存器 2 及寄存器 3，必须先发送针对寄存器 2 的探测指令。注意，所有探测指令都储存在寄存器 2 中。例程中采用了先 发送探测指令 再 读任意寄存器指令时序(读寄存器 2 + 读寄存器 3)。向 KS114 写入“PC 地址+1”后，在 20~100kb/s 的 I²C 通信速率时，不能立即去接收 8bit 的数据，要等待 ACK 低电平的有效回应，或再延时至少 50us(delaytime)，才可以接收到寄存器的数据。在写“PC 地址+1”与“读寄存器 2/3”之间加一个至少 50us 延时(delaytime)的话，I²C 通信速率可以调大仍可以与 KS114 可靠通信。小于 20kb/s 的 I²C 通信速率时，可以不用前面所述至少 50us(delaytime)的延时。另外，小于 10cm 的距离探测，相隔时间建议大于 1ms，否则可能存在上次的超声波被下一次探测所接收到的问题。总之，**确保成功建立 I²C 通信的关键有两点**：第一，高低电平延时均应不小于 4.7us；第二，KS114 收到主机的有效探测数据绿色 LED 快闪但返回值不正确时，主机需要加

上 delaytime 不小于 50us 的延时，即可获取正确数据。请遵从时序图 1~3 之规定。

休眠等待时间设置

休眠模式默认为 5s 等待，5s 内未收到探测指令则自动进入休眠模式。另有 1s 模式可供用户选择。通过 I²C 总线发送数据指令 0xc5 进入 1s 休眠模式；发送 0xc4 可以恢复 5s 休眠模式。

配置方法非常简单，向本模块发送指令时序：“I²C 地址 + 寄存器 2 +0xc4/0xc5”即可，发送完成后请延时至少 2 秒，以让系统自动完成配置。并开始按照新配置工作。

以附件 3 所示程序为例，配置代码如下：

```
write_byte(0xe8,2,0xc4);  
delayms(2000);
```

休眠等待时间设置好之后 KS114 会自动保存，并立即按照新配置工作。KS114 在重新上电后将按新配置运行。

TTL 串口及 485 串口模式

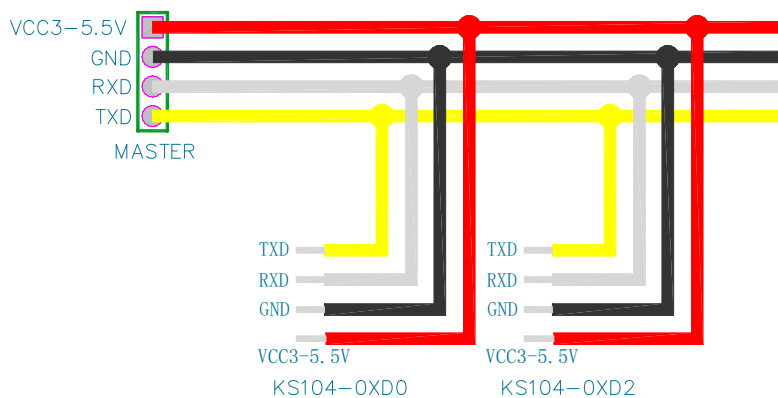


KS114 的串口模式波特率为 115200bps，1 启动位，8 数据位，1 停止位，无校验位，TTL 电平。波特率 115200bps 可修改为 9600 等其他波特率。

TTL 及 485 串口模式接线

TTL 串口模式连线依次为：红色线接电源正极 3~5.5V，黑色线接电源负极 GND，白色线接 RXD，黄色线接 TXD。此处的 TTL 串口不是 232 串口，TTL 电平可以与单片机的 TXD/RXD 直接相连，但不能与 232 串口直接相连(直接连将烧坏本模块)，需要一个 MAX232 电平转换将 TTL 电平转换为 232 电平才可以。

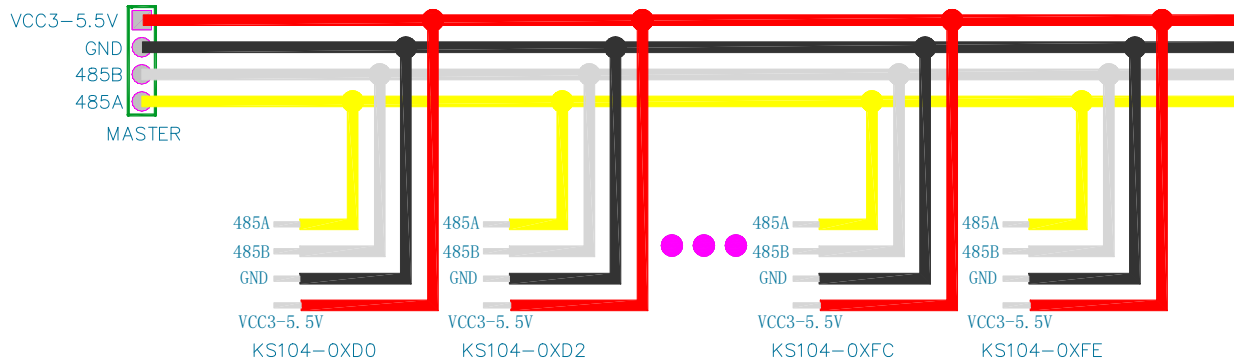
TTL 串口模式具体连线如下所示（最多接 2 个）：



485 串口模式时信号线接法为：红色线接电源正极 3~5.5V，黑色线接电源负极 GND，白色

线接 485B，黄色线接 485A。

485 串口模式具体连线如下所示（最多接 20 个）：



KS114 默认串口地址为 0xe8，用户可以将地址修改为 20 种地址中的任何一个：0xd0, 0xd2, 0xd4, 0xd6, 0xd8, 0xda, 0xdc, 0xde, 0xe0, 0xe2, 0xe4, 0xe6, 0xe8, 0xea, 0xec, 0xee, 0xf8, 0xfa, 0xfc, 0xfe。⁽⁸⁾

Note (8): 请注意，以上地址不包括 0xf0, 0xf2, 0xf4, 0xf6，其与 I²C 版地址完全一致。此外，TTL 串口协议规定一对一，因此建议使用 TTL 串口模式时，TTL 串口总线上最好只备有 1 台 KS114，**最多请不要超过 2 台**。使用 485 串口模式时，**485 串口总线上则可以接最多 20 台 KS114**。

485 串口与 TTL 串口除了接线上有所区别，控制代码上是完全一致的，以下关于“串口”的描述等同于“485 串口或 TTL 串口”。

修改串口地址时序：

地址	2	0x9a	延时	地址	2	0x92	延时	地址	2	0x9e	延时	地址	2	新地址	延时
			1ms				1ms				1ms				100ms

修改串口地址须严格按照时序来进行，时序中的延时时间为最小时间。

修改完毕后 LED 灯将长亮，给 KS114 重新上电，可观察到 LED 显示新地址。在修改 KS114 的串口地址过程中，严禁突然给 KS114 断电。修改地址函数请不要放在 while(1) 循环中，保证在程序中上电后只运行一次。

在串口地址设置为不同之后，在主机的两根串口线上可以同时连接 20 个 KS114(485 模式)；或同时连接 2 个 KS114 (TTL 模式)。主机在对其中一个 KS114 模块进行控制时，其他模块不会受到影响。

KS114 工作流程：

在 KS114 上电启动时，系统会首先开始自检，自检需要约 1200ms。在此自检过程中，KS114 将会检测各路探头是否有正常插上，检测各配置是否正常。有异常会自动将探头故障位置上报。初始化完毕 KS114 将通过串口自动向上位机发送如下十六进制代码：

69 a9 79 e8 71 7b e3 69 00 00 87 55 81 00 00 00 00 0a 4b 53 31 30 34 5f 64 61 75
78 69 2e 63 6f 6d 0a 43 6f 70 79 20 52 69 67 68 74 3a 44 61 75 78 69 2e 49 6e 63 20 32 30

31 39 0a 56 65 72 2e 31 2e 30 2e 31 30 0a 53 65 74 20 62 79 20 58 2e 51 2e 4f 6e 20 4e 6f
76 2e 36 2c 32 30 31 39 0a 00 01

其中依次地，

0x69: 程序版本，存储在寄存器 0 中；

0xa9: 制造日期标识存储在寄存器 1 中；

0x79: 串口通信波特率，存储在寄存器 4 中；

0xe8: I2C 或串口地址，存储在寄存器 5 中；

0x71: 为降噪级别，存储在寄存器 6 中；

0x7b: 出厂默认设置，用于配置盲区。存储在寄存器 7 中；

0xe3: 错误代码，存储在寄存器 8 中；

0x68: 初始化结束标志，存储在寄存器 9 中，初始化开始时其值为 0x69；

各十六进制值说明请参考如下表 2。再往后返回的是 0x0A，此为换行标识。后面的返回值请转为字符格式观察，其返回的是制造母公司网站等信息。

各寄存器列表如下，串口无法查询。

寄存器	命令	返回值范围 (10 进制)	返回值范围 (16 进制)	备注
0		1~254	0x01~0xff	程序版本标识及厂家标识。
1		1~252	0x01~0xfc	制造日期标识。16 位数据的高 8 位为制造年份，低 8 位为制造月份。11 年开始制造标识为 1；12 年开始制造标识为 2；……；25 年开始制造标识为 F；26 年开始制造标识为 0；27 年开始制造标识为 1。月份：1 月份标识为 1；以此类推，10 月份标识为 A；12 月份对应 C。
4			0x77~0x79	本寄存器存储的是串口通信波特率 0x77~0x79，供查询用。0x77 对应波特率 9600bps；0x78 对应波特率 57600bps；0x79 对应波特率 115200bps。
5			0xd0~0xfe	本寄存器存储的是 20 个 I ² C 或串口地址，不包括 0xf0, 0xf2, 0xf4, 0xf6，供查询用。
6			0x70~0x75	本寄存器存储的是降噪级别 0x70~0x75，供查询用。默认 0x71。
7			0x7a~0x7e	本寄存器存储的是盲区配置，默认 0x7b。
8			0xe0	保留供升级用
			0xe1	保留供升级用
			0xe2	保留供升级用
			0xe3	保留供升级用
9			0x6a	初始化进行中
			0x69	初始化结束标志。
10		0~255	0~0xff	初始化温度高 8 位
11		0~255	0~0xff	初始化温度低 8 位
12		0~255	0~0xff	当前环境声速高 8 位
13		0~255	0~0xff	当前环境声速低 8 位，单位：mm/100ms

表 2

自检初始化完毕后图 13 所示 LED 会以二进制方式闪烁显示其 8 位串口地址，快闪两下代表“1”，慢闪一下代表“0”。例如显示 0xea 地址，其二进制数为 0B11101010，绿色 LED 飞闪两下→灭→快闪两下→灭→快闪两下→灭→快闪两下→灭→慢闪一下→灭→快闪两下→灭→慢闪一下→灭→快闪两下→灭→慢闪一下→灭。⁽⁹⁾

Note (9): LED 闪烁时的绿色亮光可能会刺激到眼睛，请尽量不要近距离直视工作中的 LED，可以使用眼睛的余光来观察其闪烁。

KS114 启动后如果收到主机的有效数据指令，LED 将立即停止闪烁显示。进入指令探测模式。

KS114 使用串口接口与主机通信时，自动响应主机的控制指令。指令为 8 位数据，指令发送及接收探测结果流程为：

串口地址(向 KS114 发送 0xe8) → 寄存器(向 KS114 发送 0x02) → 探测指令(向 KS114 发送 0xb0) → 通过串口接收 KS114 的探测数据高 8 位 → 接收 KS114 的探测数据低 8 位

以上通信协议是可以修改的，其中，配置为 0x8F 为默认协议，返回“高 8 位+低 8 位”；配置为 0x8A 返回“0xA5+高 8 位+低 8 位+以上三个字节异或校验值”；配置为 0x8B 返回“当前地址+高 8 位+低 8 位+以上三个字节和的低 8 位”。

配置方法非常简单，向本模块发送指令时序：“串口地址 + 寄存器 2 + 0x9c; 串口地址 + 寄存器 2 + 0x95; 串口地址+寄存器 2 + 0x98; 串口地址 + 寄存器 2 + 0x8A/0x8B/0x8F”即可，发送完成后请延时至少 2 秒，以让系统自动完成配置。并开始按照新配置工作。

KS114 工作于串口模式时，只能写寄存器 0x02，写其他值将不响应。单片机接收 KS114 的探测结果时，可启用串口中断来接收 16 位探测结果，探测结果将先发高 8 位，再发低 8 位。接收到返回的 16 位探测结果之后才可以再发探测指令进行下一轮探测，否则串口将返回不正确值。

探测结束智能识别

由于探测指令发出后 KS114 会自动通过串口返回 16 位探测结果，因此串口模式无此功能。

探测指令

探测指令发送完成后，KS114 将依据探测指令进入相应探测模式，主机此时开启串口中断，未接收到返回的探测结果不能又重新发探测指令。注意，每一帧探测指令格式均为：

TTL 串口地址	寄存器 2	8 位数据
----------	-------	-------

所有串口控制指令汇总如下：

寄存器	命令	返回值范围 (10 进制)	返回值范围 (16 进制)	备注
2	0x05	1-2817us	0x01-0xb01us	默认有效探测范围 1cm-50cm。返回 us 值，除以 58 可获得 cm 距离值。本指令为高速指令，执行时间不超过 9ms。
2	0x0a	1-3840us	0x01-0xf00us	默认有效探测范围 1cm-70cm。返回 us 值，除以 58 可获得 cm 距离值。本指令为高速指令，执行时间不超过 9ms。
2	0x0f	1-6273us	0x01-0x1881us	默认有效探测范围 1cm-110cm。返回 us 值，除以 58 可获得 cm 距离值。本指令为高速指令，执行

				时间不超过 9ms。
2	0xb0	30-5657mm	0x1e-0x1619mm	默认有效探测范围 1cm-5m。返回 mm 值。本指令执行时间不超过 33ms。
2	0x14	30-2016mm	0x1e-0x7e0mm	默认有效探测范围 1cm-2m。返回 mm 值。本指令执行时间不超过 17ms。
2	0x1e	30-3009mm	0x1e-0xbc1mm	默认有效探测范围 1cm-3m。返回 mm 值。本指令执行时间不超过 25ms。
2	0xb1	0-255	0-0xff	只发射一束波，无其他功能。返回值是寄存器 2 和寄存器 3 的上一次的储存值
2	0xb2	1-32639us	0x01-0x7f7f μs	有效探测范围 1cm-5m。返回 us 值
2	0xb3	1-32639us	0x01-0x7f7f μs	只接收回波，有效探测范围 1cm-5.6 米，与 0xb1 指令配合使用，用于 2 台 KS104 之间的对射测距。
2	0xb4	30-5657mm	0x1e-0x1619mm	默认有效探测范围 1cm-5m。返回 mm 值。
2	0xb8	30-5657mm	0x1e-0x1619mm	默认有效探测范围 1cm-5m。返回 mm 值。
2	0xba	1-32639us	0x01-0x7f7f μs	有效探测范围 1cm-5m。返回 us 值
2	0xbc	30-5657mm	0x1e-0x1619mm	有效探测范围 1cm-5m。返回 mm 值
2	0x70	无	无	第一级降噪。 所有指令指令将工作于第一级降噪。适用于电池供电
2	0x71	无	无	第二级降噪。 所有指令指令将工作于第一级降噪。出厂默认设置。适用于电池供电
2	0x72	无	无	第三级降噪。 所有指令指令将工作于第一级降噪。适用于 USB 供电。
2	0x73	无	无	第四级降噪。 所有指令指令将工作于第一级降噪。适用于较长距离 USB 供电。
2	0x74	无	无	第五级降噪。 所有指令指令将工作于第一级降噪。适用于开关电源供电
2	0x75	无	无	第六级降噪。 所有指令指令将工作于第一级降噪。适用于开关电源供电
2	0x77	无	无	将串口通信波特率配置为 9600bps
2	0x78	无	无	将串口通信波特率配置为 57600bps
2	0x79	无	无	将串口通信波特率配置为 115200bps，出厂默认设置

2	0x7a	无	无	波束角配置指令
2	0x7b	无	无	出厂默认设置。用于配置盲区。
2	0x7c	无	无	波束角配置指令
2	0x7d	无	无	波束角配置指令
2	0x7e	无	无	波束角配置指令
2	0x80	无	无	配置 0x05 指令 0x0a 指令 0x0f 指令的波束角为水平 100° 竖直 50°
2	0x81	无	无	配置 0x05 指令 0x0a 指令 0x0f 指令的波束角为水平 90° 竖直 40°，为出厂默认设置
2	0x82	无	无	配置 0x05 指令 0x0a 指令 0x0f 指令的波束角为水平 80° 竖直 35°
2	0x8A	无	无	配置串口协议返回为， 0xA5+高 8 位+低 8 位+以上三个字节异或校验
2	0x8B	无	无	配置串口协议返回为， 当前地址+高 8 位+低 8 位+以上三个字节和的低 8 位
2	0x8F	无	无	恢复串口协议至默认，返回高 8 位+低 8 位
2	0x95	无	无	0x70-0x8f 参数配置第二时序
2	0x98	无	无	0x70-0x8f 参数配置第三时序
2	0x9c	无	无	0x70-0x8f 参数配置第一时序
2	0x92	无	无	修改地址第二时序
2	0x9a	无	无	修改地址第一时序

2	0x9e	无	无	修改地址第三时序
2	0xc4	无	无	5 秒休眠等待
2	0xc5	无	无	1 秒休眠等待
2	0x99	69 a9 04 02 79 e8 71 7b e3 68 00 00 87 55 81 00 00 00 00 98 04 02		<p>通过串口查询配置指令，通常用于初始化或参数查询，可以看到各配置情况。</p> <p>其中依次地，</p> <p>0x69: 程序版本，存储在寄存器 0 中；</p> <p>0xa9: 制造日期标识存储在寄存器 1 中；</p> <p>0x0402: 此值不确定，由初始化环境决定，可以忽略具体值，但与帧尾的 0x0402 是保持相等的 16 位数据；</p> <p>0x79: 串口通信波特率，存储在寄存器 4 中；</p> <p>0xe8: I2C 或串口地址，存储在寄存器 5 中；</p> <p>0x71: 为降噪级别，存储在寄存器 6 中；</p> <p>0x7b: 出厂默认设置，用于配置盲区。存储在寄存器 7 中；</p> <p>0xe3: 错误代码，存储在寄存器 8 中；</p> <p>0x68: 初始化结束标志，存储在寄存器 9 中，初始化开始时其值为 0x69；</p>

表 3

电源降噪指令 (0x70, 0x71, 0x72, 0x73, 0x74, 0x75)、波特率指令 (0x77/0x78/0x79) 波束角配置指令 (0x7a-0x7e/0x80/0x81/0x82)

KS114 默认电源推荐使用电池供电。如果使用噪音较大的电源，测距值可能会出现不稳定的波动。用户可以通过发送 0x70, 0x71, 0x72, 0x73, 0x74, 0x75 命令来配置 KS114 测距模块的杂波抑制功能。0x70 为测试用级别，**0x71** 指令将使本模块配置为第一级降噪，适用于电池供电的场合，同时也是 **出厂默认设置**。0x72 指令将使本模块配置为第二级降噪，适用于 USB 供电等有一定高频噪音的场合。0x73 指令将使本模块配置为第三级降噪，适用于较长距离 USB 供电的场合。0x74 指令将使本模块配置为第四级降噪，适用于开关电源供电的场合。0x75 指令将使本模

块配置为第五级降噪，如无特别要求，不推荐配置为本级。

用户可以通过发送 0x7a, 0x7b, 0x7c, 0x7d, 0x7e 来配置波束角，值越大波束角越小。**出厂默认为 0x7b。**

配置方法非常简单，向本模块发送指令时序：“TTL 串口地址 + 寄存器 2 + 0x9c; TTL 串口地址 + 寄存器 2 + 0x95; TTL 串口地址+寄存器 2 + 0x98; TTL 串口地址 + 寄存器 2 + 0x71/0x72/0x73/0x74/0x75/0x7a/0x7b/0x7c/0x7d/0x7e/0x80/0x81/0x82”即可，发送完成后请延时至少 2 秒，以让系统自动完成配置。并开始按照新配置工作。

配置代码请放在程序的初始化函数中，即 while(1)循环之前，以保护模块。KS114 收到有效配置指令之后，LED 灯将长亮，表明配置成功。

KS114 在重新上电后将按新配置运行。

时序图

发送探测指令，指令格式为(Only register 2):

串口地址	延时 20~100us	寄存器 2	延时 20~100us	8 位数据指令
------	-------------	-------	-------------	---------

接收数据建议采用串口中断，这样单片机可以抽出时间做其他的事情。单片机采用模拟串口时请根据串口协议判断 SDA/TX 引脚的电平变化来接收数据，数据依次为：

探测结果高 8 位	探测结果低 8 位
-----------	-----------

接收完数据后方可进行下一轮探测指令(例如：0xe8+0x02+0xbc)的发送。

休眠等待时间设置

串口模式不进入休眠。

完整发货清单



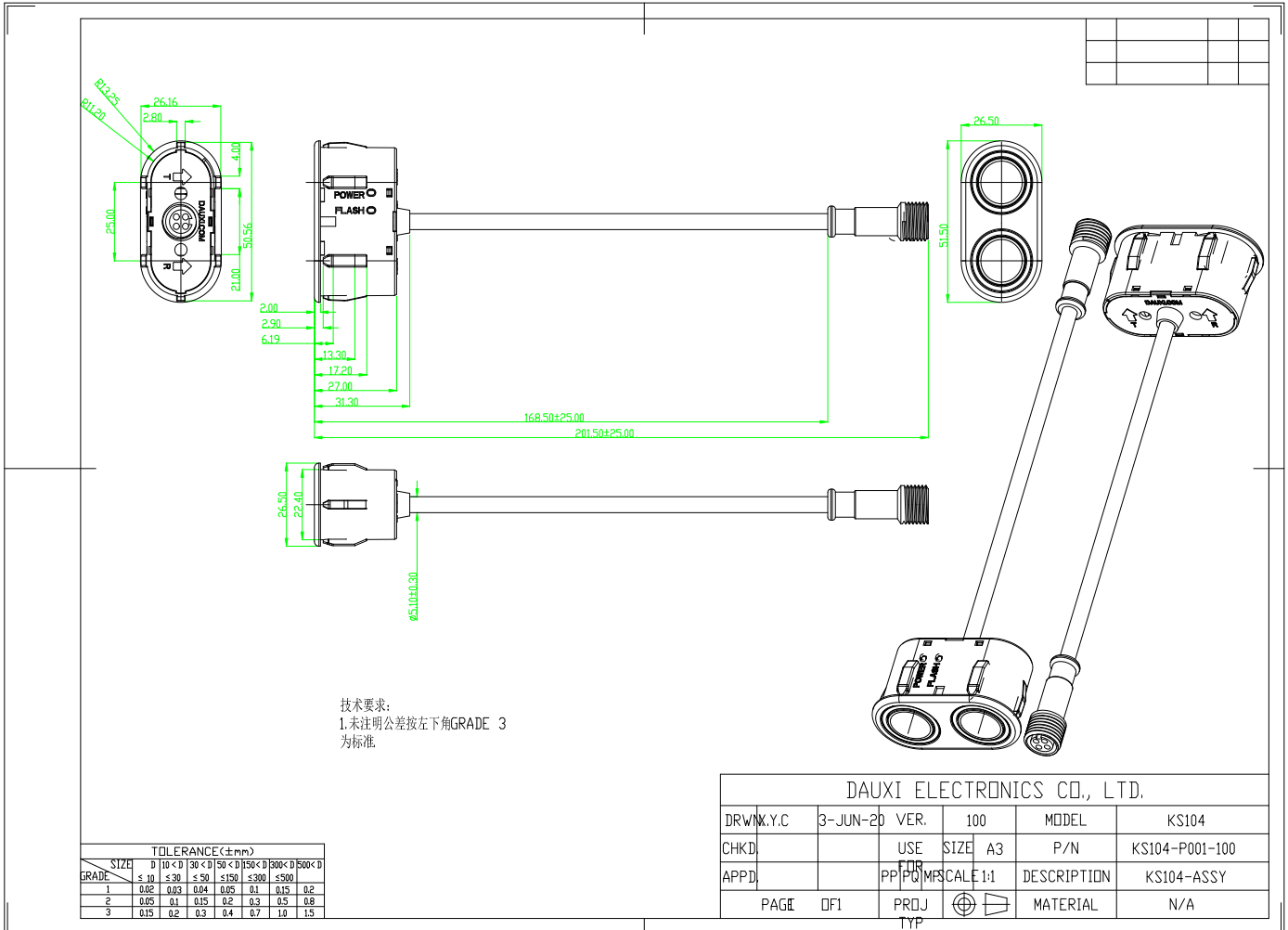
默认发黑色或白色，其他颜色可订制。

包含 KS114 主体 1 套，2 米延长线 1 根。如果需要接多路 KS114，可使用本说明书第 2 页所述之三通总线 *LINE-3T-6-600-110-CNT*。

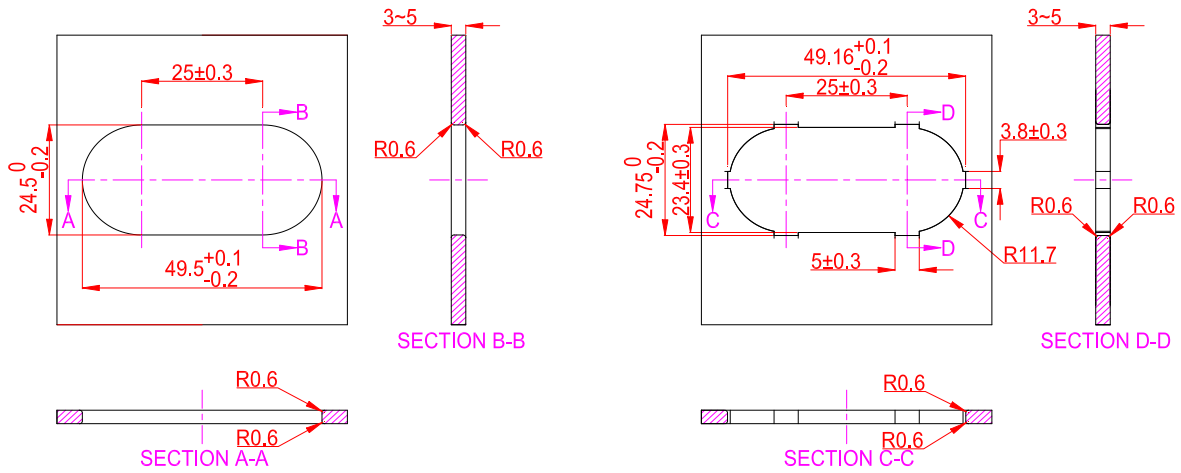
因产品改进需要，可能会对本资料进行修改，客户不能及时获得修改通知时，请在本公司网站

www.dauxi.com 获取最新产品资料。

KS114 装配尺寸图:

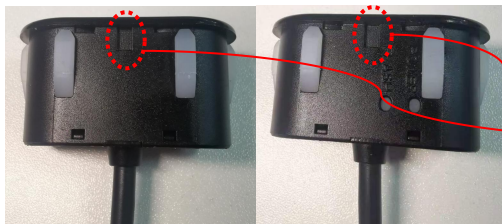


安装孔建议:



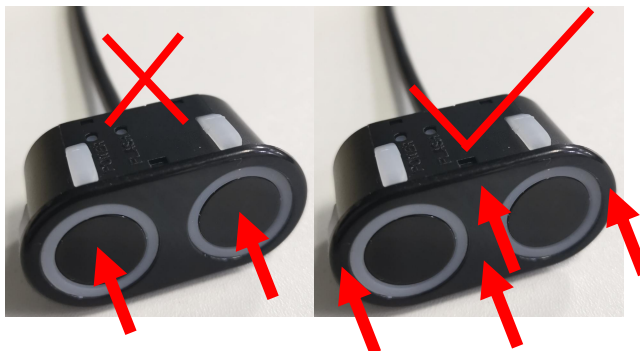
以上两种安装孔都是可以的，注意与 KS114 硅胶接触边正反面都要倒圆角 R0.5~1mm，安装面板厚度建议 $\geq 3\text{mm}$ 。请留意 KS114 宽度方向才是大波束角方向，因此 KS114 实际应用时经常采用竖直安装的方式，即安装孔竖直方向设计，而不是本图的水平方向排布，要旋转 90° 。

也可以采用如下扣位安装方案，在壳体上设计好倒扣，将 KS104 接入壳体，壳体上的卡扣扣入下图方槽中，用于防爆设计时的安装。



图示红色虚线框内的正反面两个方槽可用于设计扣位安装。

装配注意事项:



以上左图画“×”为错误操作受力点，装配时不可以直接按压中间圆形的探芯振动部位。右图箭头指向为可受力点，画勾为正确操作，按压 KS114 外沿的塑胶壳体部分将 KS114 装配到位。

使用建议:

1)快速上手

第一步: 下载软件及说明书: http://www.dauxi.com/Inc/driver_control.zip

第二步: 打开压缩包 driver_control.zip 中的“KS1XX 测试软件-免安装版-USART.exe”及“KS10R-V210.pdf”, 参考 KS10R-V210.pdf 第五页开始的部分使用软件, 可以快速上手控制 KS114.

KS10R 购买可以在以下链接找到: <http://www.dauxi.com/productn.asp?ArticleID=1764>

其他型号的 USB 转 485 也可以。

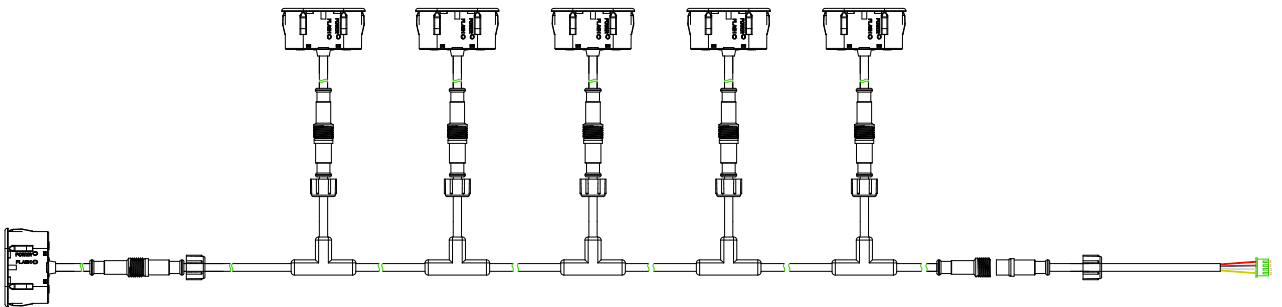
2)控制算法

轮询算法上参考此文件第 45~50 页;

<http://www.dauxi.com/%E5%AF%BC%E5%90%91%E8%B6%85%E5%A3%B0%E6%B3%A2%E9%80%89%E5%9E%8B%E5%8F%8A%E4%BD%BF%E7%94%A8%E5%BB%BA%E8%AE%AE.pdf>

即建议定时, 例如 15ms 一台 KS114, 每个模块分配固定的时间。

接线参考并联式总线, 一线到底, 机器人内部只有一根线。如图所示:



由于 KS114 的功耗极低, 低于 50 毫瓦。因此在总线上并联 20 台也才 1w 的功耗。

附件-例程:

- 1) PIC16F877A 主机采用硬件 I²C 通讯与 KS114 连接控制 C 代码
- 2) PIC16F877A 主机采用模拟 I²C 通讯与 KS114 连接控制 C 代码
- 3) 51 单片机主机模拟 I²C 通讯与 KS114 连接控制 C 代码
- 4) STM32 CORTEX-3 ARM 主机模拟 I²C 通讯与 KS114 连接控制 C 代码

```

1) PIC16F877A 主机采用硬件 I2C 通讯与 KS114 连接控制 C 代码
/*电路连接方式: PIC16F877A 的 IO 口 SCL、SDA 与 KS114 的 SCL、SDA 连接, PIC16F877A
的 SCL、SDA 线均需个上拉一个 4.7K 的电阻到电源正极 VCC。*/
#include <pic.h>                //4MHz 晶振
__CONFIG(0x3d76);             //开看门狗
#define DELAY() delay(10)
#define SCL RC3                // 此引脚须上拉 4.7K 电阻至 VCC
#define SDA RC4                // 此引脚须上拉 4.7K 电阻至 VCC
void setup(void);
unsigned int detect_KS101B(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); //显示函数请根据主机的实际接线编写
unsigned int distance;
void main(void)
{
    setup();
    //change_address(0xe8,0xe0); //将默认地址 0xe8 改为 0xe0
    while(1)
    {
        CLRWDT();
        distance = detect_KS101B(0xe8,0x30); //Address:0xe8; command:0x30.
                                                //Get detect result from KS114, 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); //显示函数请根据主机的实际接线编写
{
    CLRWDT();
}
void change_address(unsigned addr_old,unsigned char addr_new)
{
    SEN = 1;                                // send start bit to KS114
    while(SEN);                              // wait for it to clear
    while(!SSPIF);                          // wait for interrupt
    SSPIF = 0;                               // then clear it.

    SSPBUF = addr_old;                       // KS114'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 KS114 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; // KS114'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 KS114 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; // KS114'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 KS114 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; // KS114'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 KS114 to break to do something
}

unsigned int detect_KS101B(unsigned char ADDRESS, unsigned char command)
{
//ADDRESS will be KS114's address such as 0x30, command will be the detect command such as 0x30
unsigned int range=0;
    SEN = 1;                   // send start bit
    while(SEN);                // and wait for it to clear
    while(!SSPIF);
    SSPIF = 0;
    SSPBUF = ADDRESS;          // KS114'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 KS114 is ranging
    TMR1L = 0;
    T1CON = 0x31;              //configuration of TIME1
    TMR1IF = 0;                //clean TIME1 interrupt flag
    while(!SCL || (!TMR1IF))display(distance,100); //要获得连续显示，这儿要加上显示函数
    TMR1ON = 0;                // stop timer
    // finally get the range result from KS114
    SEN = 1;                   // send start bit
    while(SEN);                // and wait for it to clear
    ACKDT = 0;                 // acknowledge bit
    SSPIF = 0;

    SSPBUF = ADDRESS;          // KS114 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;        // KS114 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) //向 KS114 发送一个 8 位数据指令
{
    SEN = 1;                                         // send start bit
    while(SEN);                                     // and wait for it to clear
    while(!SSPIF);
    SSPIF = 0;
    SSPBUF = ADDRESS;                               // KS114 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 硬件 I2C 初始化配置
{
    SSPSTAT = 0x80;
    SSPCON = 0x38;
    SSPCON2 = 0x00;
    SSPADD = 50;
    OPTION=0B10001111;//PSA = 1;切换到 1:128 分频给 WDT,即 32.64ms 之内必须清一次看门狗
    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++)CLRWDTO);
}

```

2) PIC16F877A 主机采用模拟 I²C 通讯与 KS114 连接控制 C 代码

```

#include <pic.h> //4MHz 晶振
__CONFIG(XT&WDTEN); //开看门狗
#define SDA RD6 // 此引脚须上拉 4.7K 电阻至 VCC
#define SCL RD5 // 此引脚须上拉 4.7K 电阻至 VCC
#define SDAPORT TRISD6 //
#define SCLPORT TRISD5 //引脚 RD6, RD5 可换为其他任何 I/O 脚
bit eepromdi;
bit eepromdo;

void delay(void)
{

```

```
    unsigned char k;
    for(k=0;k<180;k++)
        asm("CLRWDI");
}

void delays(unsigned char ms)//ms 延时函数
{
    unsigned int i,j;
    for (i=0;i<ms;i++)
        for(j=0;j<110;j++)
            asm("CLRWDI");
}

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_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_write(unsigned char address,unsigned char buffer,unsigned char command)
//////////write a command: address + register + command,there will be 0xe8 + 0x02 + 0x30
{
```

```

    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
    delaysms(200); //Protect the eeprom,you can delete this
    KS101B_write(addr_old,2,0x9a);
    delaysms(1);
    KS101B_write(addr_old,2,0x92);
    delaysms(1);
    KS101B_write(addr_old,2,0x9e);
    delaysms(1);
    KS101B_write(addr_old,2, addr_new);
    delaysms(100); //Protect the eeprom,you can delete this
}

unsigned int detect_KS101B(unsigned char address, unsigned char command)
{
    unsigned int range1;
    KS101B_write(address,2,command);
    delaysms(1); //安全延时,如果显示不清晰可以将延时调大一些
    delaysms(80); //如果是探测温度此处延时需延长, 使用 while(!SCL)此处可删除
    //SCLPORT=1;while(!SCL);
    // delaysms(80)也可换为 SCLPORT=1;while(!SCL);直接查询 SCL 线的等待时间将最短, 探测速度最快
    range1 = KS101B_read(address,2);
    range1 =(range1<<8) + KS101B_read(address,3);
    delaysms(5);
    return range1;
}

void main(void)
{
    unsigned int range;
    //change_i2c_address(0xe8,0xfe); ///将默认地址 0xe8 改为 0xfe
    delaysms(200);
    while(1)
    {
        asm("CLRWDI");
        range = detect_KS101B(0xe8,0x30); //you just need the only one sentence to get the range.
        delaysms(200);
    }
}

```

3) 51 单片机主机模拟 I²C 通讯与 KS114 连接控制 C 代码

```

#include <reg51.h>
#include <intrins.h>
sbit SDA=P3^6; // 此引脚须上拉 4.7K 电阻至 VCC
sbit SCL=P3^7; // 此引脚须上拉 4.7K 电阻至 VCC
unsigned int range;

void display(unsigned int range)
{
    //input your display function, please.
}

```

```
void delay(void)          //short delay 使用速度较快的单片机时，I2C 通讯可能不正常，在此函数中多加 4~8 个_nop_();即可
{
    _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();
    delay();delay();delay();delay();delay(); //此处延时对于 STC89C 系列单片机，可以删除，如果对于快速单
//片机，需要加至少 50us 的延时，才可以可靠读到数据
    dat = i2c_read_byte();
    no_ack();
    stop();
    return dat;
}

void delaysms(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
    delaysms(2000); // Protect the eeprom ,you can delete this sentence
    write_byte(addr_old,2,0x9a);
    delaysms(1);
    write_byte(addr_old,2,0x92);
    delaysms(1);
    write_byte(addr_old,2,0x9e);
    delaysms(1);
    write_byte(addr_old,2, addr_new);
    delaysms(500); //Protect the eeprom, you can delete this sentence
}

void config_0x71_0x7d(unsigned char addr_old, unsigned char flag)
//flag will be 0x71,0x72,0x73,0x74,0x7a,0x7b,0x7c,0x7d
{
    //that you want change to
    delaysms(2000); // Protect the eeprom ,you can delete this sentence
    write_byte(addr_old,2,0x9c);
    delaysms(1);
    write_byte(addr_old,2,0x95);
    delaysms(1);
    write_byte(addr_old,2,0x98);
    delaysms(1);
    write_byte(addr_old,2, flag);
    delaysms(500); //Protect the eeprom, you can delete this sentence
}

unsigned int detect(unsigned char address,unsigned char command) //0xe8(address) + 0x30(command)
{
    unsigned int distance,count;
    write_byte(address,2,command); //use command "0x30" to detect the distance
    delaysms(1); //安全延时,如果显示不清晰可以将延时调大一些
    //delaysms(80); //如果是探测温度此处延时需根据表 1 所列时间相应延长
    count=800;
    while(--count || !SCL) //等待探测结束，count 值调小将减小探测等待时间
    {
        ; // 空语句
        display(range); //显示语句，可根据需要保留或删除
    }
    // while(!SCL)display(range); //you can delete "display(range)"
//通过查询 SCL 线来智能识别探测是否结束，使用本语句可删除上条语句(count=800;while...)以节省探测时间
    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(0xe8,0x30);
        //0xe8 is the address; 0x30 is the command.you just need the only one sentence to get the range.
        //display(range);
        delayms(200);
    }
}
```

4) STM32 CORTEX-3 ARM 主机模拟 I²C 通讯与 KS114 连接控制 C 代码

//单片机型号：STM32F103RBT //本程序未示出所有系统配置函数

```
#include <stm32f10x_lib.h>
#include "sys.h"
#include "usart.h"
#include "delay.h"

u8 KS114_ReadOneByte(u8 address, u8 reg)
{
    u8 temp=0;

    IIC_Start();
    IIC_Send_Byte(address); //发送低地址
    IIC_Wait_Ack();
    IIC_Send_Byte(reg); //发送低地址
    IIC_Wait_Ack();
    IIC_Start();
    IIC_Send_Byte(address + 1); //进入接收模式
    IIC_Wait_Ack();

    delay_us(50); //增加此代码通信成功!!!
    temp=IIC_Read_Byte(0); //读寄存器 3
    IIC_Stop();//产生一个停止条件
    return temp;
}

void KS114_WriteOneByte(u8 address,u8 reg,u8 command)
{
    IIC_Start();
    IIC_Send_Byte(address); //发送写命令
    IIC_Wait_Ack();
    IIC_Send_Byte(reg);//发送高地址
    IIC_Wait_Ack();
    IIC_Send_Byte(command); //发送低地址
    IIC_Wait_Ack();
    IIC_Stop();//产生一个停止条件
}
```

```

void IIC_Init(void)
{
    RCC->APB2ENR|=1<<4;//先使能外设 IO PORTC 时钟
    GPIOC->CRH&=0XFFF00FFF;//PC11/12 推挽输出
    GPIOC->CRH|=0X00033000;
    GPIOC->ODR|=3<<11;    //PC11,12 输出高
}
//产生 IIC 起始信号
void IIC_Start(void)
{
    SDA_OUT();    //sda 线输出
    IIC_SDA=1;
    IIC_SCL=1;
    delay_us(10);
    IIC_SDA=0;//START:when CLK is high,DATA change form high to low
    delay_us(10);
    IIC_SCL=0;//钳住 I2C 总线，准备发送或接收数据
}
//产生 IIC 停止信号
void IIC_Stop(void)
{
    SDA_OUT();//sda 线输出
    IIC_SCL=0;
    IIC_SDA=0;//STOP:when CLK is high DATA change form low to high
    delay_us(10);
    IIC_SCL=1;
    IIC_SDA=1;//发送 I2C 总线结束信号
    delay_us(10);
}
//等待应答信号到来
//返回值： 1，接收应答失败
//      0，接收应答成功
u8 IIC_Wait_Ack(void)
{
    u8 ucErrTime=0;
    SDA_IN();    //SDA 设置为输入
    IIC_SDA=1;delay_us(6);
    IIC_SCL=1;delay_us(6);
    while(READ_SDA)
    {
        ucErrTime++;
        if(ucErrTime>250)
        {
            IIC_Stop();
            return 1;
        }
    }
    IIC_SCL=0;//时钟输出 0
    return 0;
}
//产生 ACK 应答
void IIC_Ack(void)
{
    IIC_SCL=0;
    SDA_OUT();
    IIC_SDA=0;
    delay_us(10);
    IIC_SCL=1;
    delay_us(10);
    IIC_SCL=0;
}
//不产生 ACK 应答
void IIC_NAck(void)

```

```
{
    IIC_SCL=0;
    SDA_OUT();
    IIC_SDA=1;
    delay_us(10);
    IIC_SCL=1;
    delay_us(10);
    IIC_SCL=0;
}
//IIC 发送一个字节
//返回从机有无应答
//1, 有应答
//0, 无应答
void IIC_Send_Byte(u8 txd)
{
    u8 t;
    SDA_OUT();
    IIC_SCL=0;//拉低时钟开始数据传输
    for(t=0;t<8;t++)
    {
        IIC_SDA=(txd&0x80)>>7;
        txd<<=1;
        delay_us(10);
        IIC_SCL=1;
        delay_us(10);
        IIC_SCL=0;
        delay_us(10);
    }
}
//读 1 个字节, ack=1 时, 发送 ACK, ack=0, 发送 nACK
u8 IIC_Read_Byte(unsigned char ack)
{
    unsigned char i, receive=0;
    SDA_IN();//SDA 设置为输入
    for(i=0;i<8;i++)
    {
        IIC_SCL=0;
        delay_us(10);
        IIC_SCL=1;
        receive<<=1;
        if(READ_SDA)receive++;
        delay_us(5);
    }
    if(!ack)
        IIC_NAck();//发送 nACK
    else
        IIC_Ack();//发送 ACK
    return receive;
}

int main(void)
{
    u16 range;
    Stm32_Clock_Init(9);//系统时钟设置
    delay_init(72); //延时初始化
    uart_init(72,9600);//串口 1 初始化
    while(1)
    {
        KS114_WriteOneByte(0XE8,0X02,0x30);
        delay_ms(80);
        range = KS114_ReadOneByte(0xe8, 0x02);
        range <<= 8;
        range += KS114_ReadOneByte(0xe8, 0x03);
    }
}
```

}
}