颜色传感器

Powered by LaTeX

感为智能科技

2024 年 9 月 5 日

感为科技

开篇语

首先感谢您对感为的支持！

这是一款主动光式颜色传感器，它能有效的解决被动光式颜色传感器的一系列问题，增加了检测距离，提高了准确率。

什么是主动光式颜色传感器呢？主动光式颜色传感器会主动地发射红绿蓝三种颜色的光线，光线经过反射与吸收，被接收器检测到，从而形成对于颜色的认知。

为什么能检测距离更远呢？由于发射管、接收管都自带有高透聚光镜头，相当于给传感器加了个望远镜。相比于被动光颜色传感器的无镜头检测，看的自然更远了。

为什么能检测准确率更高呢？由于透镜因素，光角较小，根据公式推导，其响应曲线更加平滑与线性，所以更加稳定，同时，如果您在读 **IIC** 数据时选择了 **HSL** 色彩格式，稳定性相比于 **RGB** 色彩格式，会有更大的提升。

如果您对上述观点产生了兴趣或者质疑，请您详细阅读《测量原理》部分。

重要事项

- 传感器不可检测发光体的颜色，例如霓虹灯光的颜色，屏幕的颜色、激光投影到地上的颜色。
- 本传感器需要校准，校准的目的是消除三个发射管的差异，同时记录使用间距，这样能让在同样距离下，白色为标准数值（**R:255,G:255,B:255**），黑色也为标准数值（**R:0,G:0,B:0**）。

- 测量的数据会随着您检测距离变远，数值会变小，可以抽象为，距离越远，物体颜色越暗。不过您不必担心距离的问题，因为传感器内部集成了 **HSL** 算法，即 **H:** 色相，**S:** 饱和度，**L:** 亮度。这个算法可以将物体的颜色和亮度分离开，所以不管远近、亮暗，色相基本保持不变，您可以仅凭色相与饱和度判断物体的颜色。

- 传感器内部集成有滤光算法，您不必给传感器额外打光，因为您打的光，在设计之初归类为不稳定光源，会被算法滤掉，这能有效保证您的使用体验。同理，外界环境光有变化也不会对传感器产生影响，所以您也没必要装遮光罩。

- 感光元件具有物理上限，请尽量避免在强烈日照条件下使用。当到达感光上限后，传感器 **ERR** 会亮起，这种情况下传感器数据可能不准确。此时只需对传感器进行简单的遮挡，使之至于阴影下，避免阳光直射即可。

- 为获得最佳效果，建议您避免在阳光直射的室外校准传感器。

- 在强烈日照条件下，且 **ERR** 亮起时，此时不允许校准传感器，请将传感器置于弱光下，断电重启，或用 I^2C 发送重启命令后再行校准。

- 传感器具有记忆功能，重启后不需要重新校准

- 传感器的电压务必准确且稳定。尤其是有电机应用的场景，务必选用优质的电机驱动器，以防电机在启停机时产生电压尖峰冲击传感器。

目录

1 产品参数 **6**

1.1 产品参数 . . . 6
1.2 传感器尺寸 . . . 6
1.3 名称与功能 . . . 7

2 设备的安装 **8**

2.1 传感器接线 . . . 8
2.2 设置校准 . . . 9
2.2.1 校准基础——新手必学 . . . 9
2.2.2 校准详解——为了更好用 . . . 10

3 测量原理 **12**

3.1 光学基础 . . . 12
3.1.1 白光的组成 . . . 12
3.1.2 人眼中的颜色 . . . 12
3.2 被动光式传感器原理 . . . 13
3.3 主动光式传感器原理 . . . 13
3.3.1 初识主动光式原理 . . . 13
3.3.2 如何实现测量 . . . 14
3.4 RGB 与 HSL 色彩格式 . . . 15
3.4.1 RGB 格式 . . . 15
3.4.2 HSL 格式 . . . 15
3.4.3 RGB 转 HSL 公式 . . . 17
3.5 参数对检测效果的影响 . . . 18
3.5.1 数学模型 . . . 18
3.5.2 检测距离对颜色亮度的影响 . . . 19
3.5.3 产品参数对检测距离的影响 . . . 19

4 设备的 I^2C 通讯 **21**

4.1 简介 . . . 21
4.2 设备的功能 . . . 21
4.3 I^2C 协议回顾 . . . 22
4.3.1 I^2C 硬件 . . . 22
4.3.2 I^2C 基础 . . . 22
4.3.3 I^2C 时序 . . . 23
4.4 通讯语法及结构 . . . 24
4.4.1 语法简介 . . . 24
4.4.2 语法 . . . 24
4.4.3 结构 . . . 24
4.5 地址位设置及实例 . . . 25

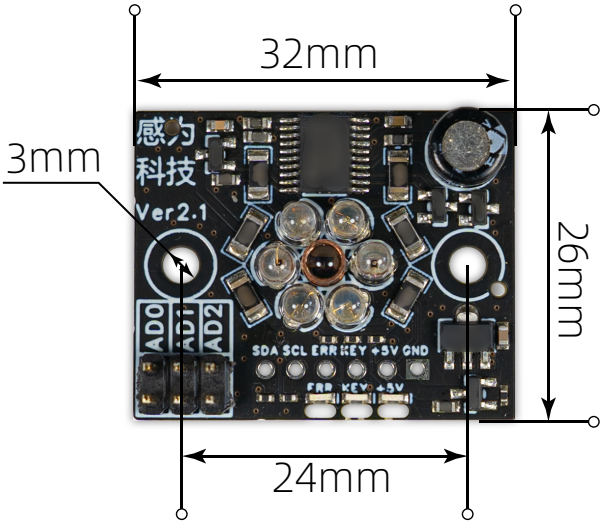
4.5.1 地址位设置 . . . 25
4.5.2 地址程序实例 . . . 25
4.6 读取 RGB 数据 . . . 26
4.6.1 功能描述 . . . 26
4.6.2 通讯方法 . . . 26
4.6.3 代码示例 . . . 27
4.7 读取 HSL 数据 . . . 30
4.7.1 功能描述 . . . 30
4.7.2 通讯方法 . . . 30
4.7.3 代码示例 . . . 31
4.8 软件地址配置 . . . 33
4.8.1 功能描述 . . . 33
4.8.2 用法及例程 . . . 33
4.9 广播重置地址与扫描找回地址 . . . 34
4.10 ping 网络诊断工具 . . . 36
4.10.1 功能描述 . . . 36
4.10.2 命令特性 . . . 36
4.10.3 用法及程序实例 . . . 37
4.11 读取错误信息 . . . 38
4.11.1 功能描述 . . . 38
4.11.2 用法及程序实例 . . . 38
4.12 设备软件重启 . . . 39
4.12.1 功能描述 . . . 39
4.12.2 用法及示例 . . . 39
4.13 固件版本号查询 . . . 40
4.13.1 功能描述 . . . 40
4.13.2 用法及示例 . . . 40
4.14 命令符 . . . 41

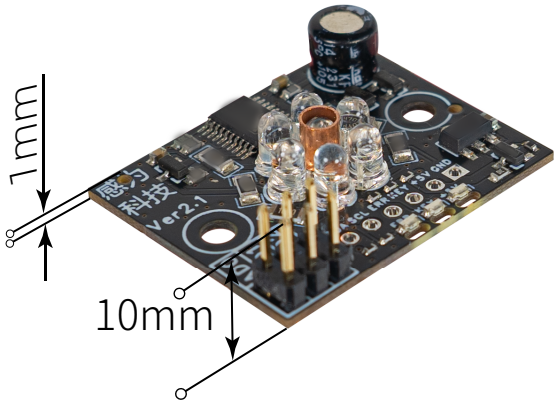
1 产品参数

1.1 产品参数

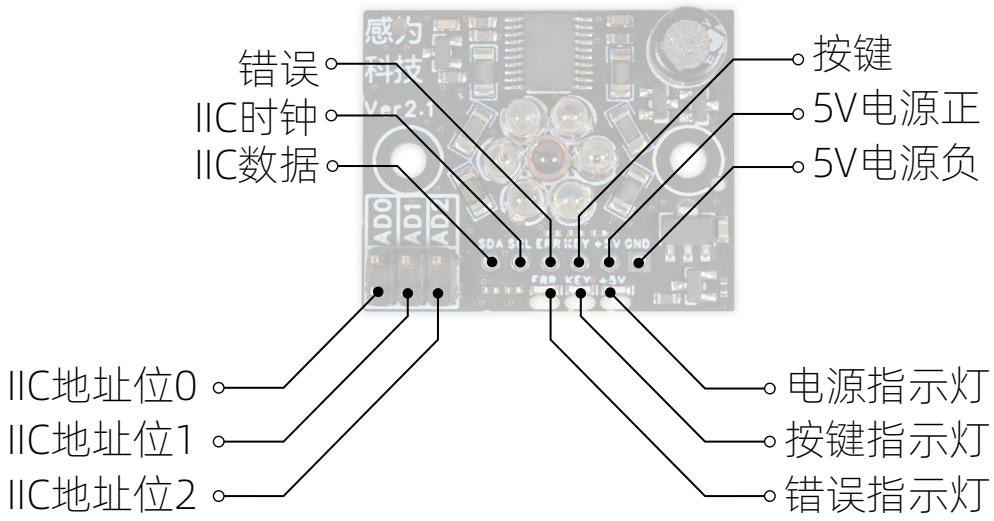
性能参数						
	符号	最小值	典型值	最大值	单位	测试条件
IIC 上拉电压	U_{IIC_max}	4.2	4.4	4.5	V	$U=5V$
工作电压	U	5	5	18	V	——
工作电流	I	31	31.8	32	mA	$U=5V$
检测距离	h	30	40	150	mm	$U=5V$
响应速度	t	0.75	0.78	0.8	ms	$U=5V$
工作温度	T	-25	——	85	℃	——
安装孔尺寸	——	3.3			mm	——
重量	G	3.5			g	——
通讯技术	I^2C 通讯					
I^2C 传输的数据	1、RGB；2、HSL；4、错误信息……					

1.2 传感器尺寸





1.3 名称与功能



指示灯名称及功能：

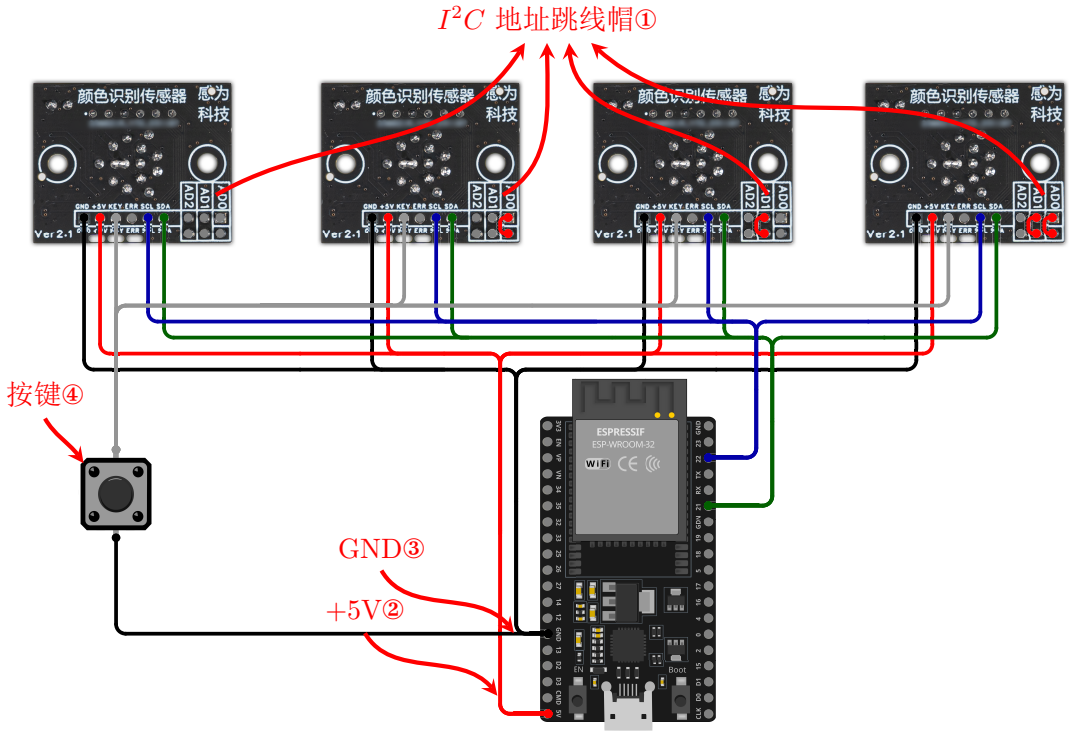
- 电源指示灯：用于指示供电，该指示灯与设备内部的 VCC 相连，上电即亮起。
- 按键指示灯：用于设置与交互。
- 错误指示灯：用于指示故障。

端口名称及功能：

- 电源正、电源负端口：用于接 5V 电源。
- 按键端口：用于外接按键，按键的两个脚需连接至按键端口与电源负极。按键仅是个接地动作，无需按键也可以进行校准，具体请看 B 站“感为”视频，看 UP 是如何操作的。
- 错误端口：用于向外输出报错信号，该端口与红色 LED 灯相连，可以悬空不用。
- IIC 数据、IIC 时钟端口：用于连接 IIC 主设备，是 IIC 的标准接口。
- IIC 地址位 0、1、2：插入跳线帽相应的地址位置一，如需了解更多点击4.5。

2 设备的安装

2.1 传感器接线



①： 示例中有 4 台设备，他们用跳线帽区分了他们的地址，从左至右地址依次为 0b1001 000X、0b1001 001X、0b1001 010X、0b1001 011X;

②： 请勿低于 5V 供电，产品自带稳压芯片，+5V 端口可以承受 18V 电压，但官方推荐 5V 供电，目的是防止接错线时击穿设备。

③： 灰度传感器要与 ESP32 共地，设备间需要共地才能完成通讯。

⑤： 请将按键的一个引脚接入灰度传感器的按键端口，另一个引脚请接入电源负端口。此外，当有多个灰度传感器共同使用时，您可以共用按键。按键只是一个接地动作，实际上您也可以不用按键，拿一根公对母的杜邦线，母头插 KEY 上，公头怼到 GND 杜邦线的开窗上，这样可以模仿按键按下，这种方式不需要按键。

2.2 设置校准

在拿到产品后，应该先进行校准，校准是为了适应您的应用场景而设立的，以便给您更加准确的检测，这一步很重要，校准不好的话，有可能检测准确度会出现问题。

2.2.1 校准基础——新手必学

准备工作

按照上一章节，只需要接按键的线，还有供上电。请再准备一张白纸，一个黑纸（或者随手找到的，例如黑色鼠标垫。如果实在没有黑色，也可以将传感器对准超量程的任意位置，最好是远超量程，例如对准距离探头正前方 2 米的墙，2 米远超了量程，此时根本不用管墙什么颜色，传感器的滤光算法，会认为他是黑色，因为传感器仅接收自己探头发出的 RGB 光线，这么远根本不会传到探头里，所以是一种很黑的黑色。我一般称这种空间为虚空，传感器感受不到任何信号即为黑，像茫茫宇宙一样黑暗）

第一步

长按校准按键，按键指示灯 key 会快闪，此时松手，您会发现指示灯开始慢闪了，这样就进入了校准模式。假如您长按时间超过了 15 秒还不松手，系统判定为 key 与 GND 短路，则会自动退出校准模式，且锁死按键端口，不在相应，您应该在排除短路后，断电重启设备。

在校准模式下，传感器停止检测，停止传数据。此时，如果不再动按键，同时没有断电重启，该模式则无法主动退出，所以有充足时间给您操作，请您放心。

第二步

将传感器对准待测物固定（传感器与待测物的距离应在检测范围内），例如您检测贴纸的颜色，则贴纸是您的待测物。

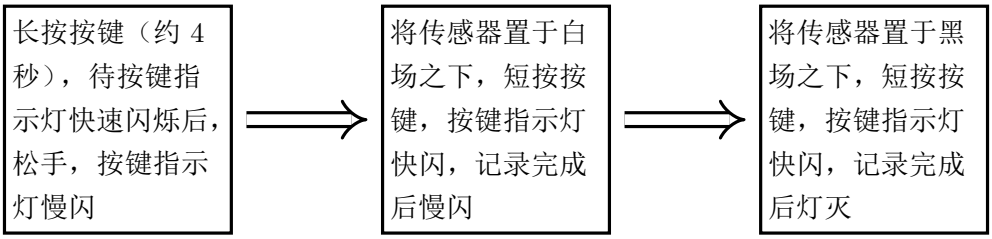
之后取来一张白纸（白场），贴近待测物前方平铺，例如取白纸盖住贴纸。短按按键，按键指示灯快闪，证明传感器在检测白色，同时记录数值，记录完成后，灯会改为慢闪；慢闪状态同样的，除非断电重启，不会自动超时退出。

第三步

取来一张黑纸（黑场），贴近待测物前方平铺，例如取黑纸盖住贴纸。短按按键，之后按键指示灯再次快闪，证明传感器在检测黑色，同时记录数值，记录完成后，灯会熄灭，证明已经完成了校准，退出了校准模式。这样校准就完成了

校准流程

注意：黑白校准没有顺序，可以先黑后白。



2.2.2 校准详解——为了更好用

接下来讲述一下校准的目的，只有了解设计者的初衷，才能更好的校准传感器。

校准的意义

校准的意义是告诉设备什么是白色，什么是黑色。从而在白色的情况下输出 RGB:(255,255,255)，黑色的情况下输出 RGB:(0,0,0)，听到这里，有很多同学会想到相机的白平衡[1]，但并非如此，校准没有这么简单。

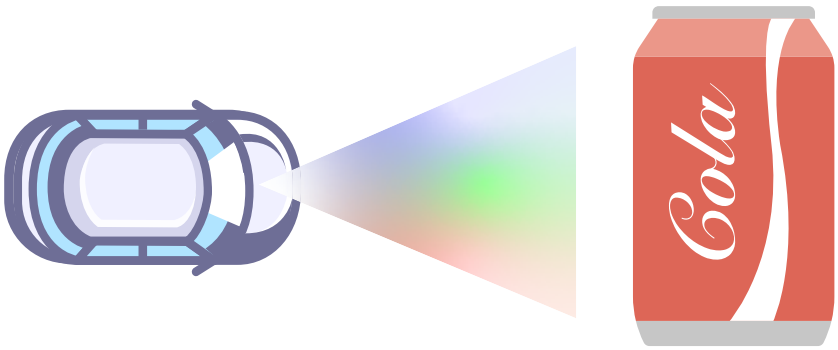
我们知道随着检测距离的增加，颜色是变黑的，可以想象在伸手不见五指的夜晚用手机录像，同时开着闪光灯补光。物体离你越远，颜色越暗，越黑，越看不清。想看清的话，相机必须重新设置曝光度，提高 COMS 的灵敏度，让他更亮一些。于是远处的物体显色正常了，但是同时你会发现，你近处的物体将会白茫茫的一片，因为过曝了，也就是失去了颜色细节，RGB 三个颜色均饱和，呈现为白色。所以这是一个相悖的事情，想让远处的物体正常，近处就会失常，想让近处的物体正常，远处的物体就会变暗变黑，失去本身的颜色细节。当然您要知道这一切发生在仅有闪光灯补光的夜晚，这与传感器检测过程非常相似，传感器的校准是相当于在手动调整曝光度。可是如今手机摄像头不会这么麻烦，因为他们是相机有海量的数据可以分析，可以自动化完成曝光调整，但是颜色传感器仅有一个像素点，没那么多可以分析的数据，只能手动完成。这对于工业应用，或者比赛场景，不算什么，因为他们的检测距离相对固定，且有 HSL 算法加持，距离的影响变得就不那么重要了。

为什么有了 HSL 还需要校准

起初传感器设计上是没有校准这一步骤的，开发者的核心思路是让大家尽可能的用 HSL（色相 H、饱和度 S、亮度 L）色彩格式（具体请访问章节：3.4.2），因为他将距离的影响，转化为亮度 L 的变化，把 L 删掉，我们就不再需要关注距离的影响。但是 HSL 有个缺点，无法靠 HS 来判断黑白，他只能依靠 L 识别黑白色，白色就是亮度 L 接近 240，黑色就是亮度 L 接近 0，与 HS 数据相关性不大，而且数据会发生大幅跳跃。如果没有校准这一步骤，传感器将无法区分您要检测的东西是因为距离远而变黑，还是本身就是黑色，亦或是无法区分是因为过曝表现为白色还是本身就是个白色。

所以如果您没有黑白检测场景，那么校准过程，您可以直接选在 30–40mm 处校准。但是如果您有黑白的使用场景，那么就需要正式校准了。

但是举个例子我想检测 100mm 处的物体，校准基础章节告诉我们，就在 100mm 处校准，这样校准是最标准的，这个没有错误。但是是有代价的，假如传感器在一辆小车上，物体固定在地上，小车需要走过去检测，这种情况下，必然距离会有波动，如果按照标准校准在 100mm，但是某一次他过冲了，冲到了 50mm 的位置，是不是有可能会过曝，导致所有颜色都是白色，这种情况下，我们就需要脑子灵活了。

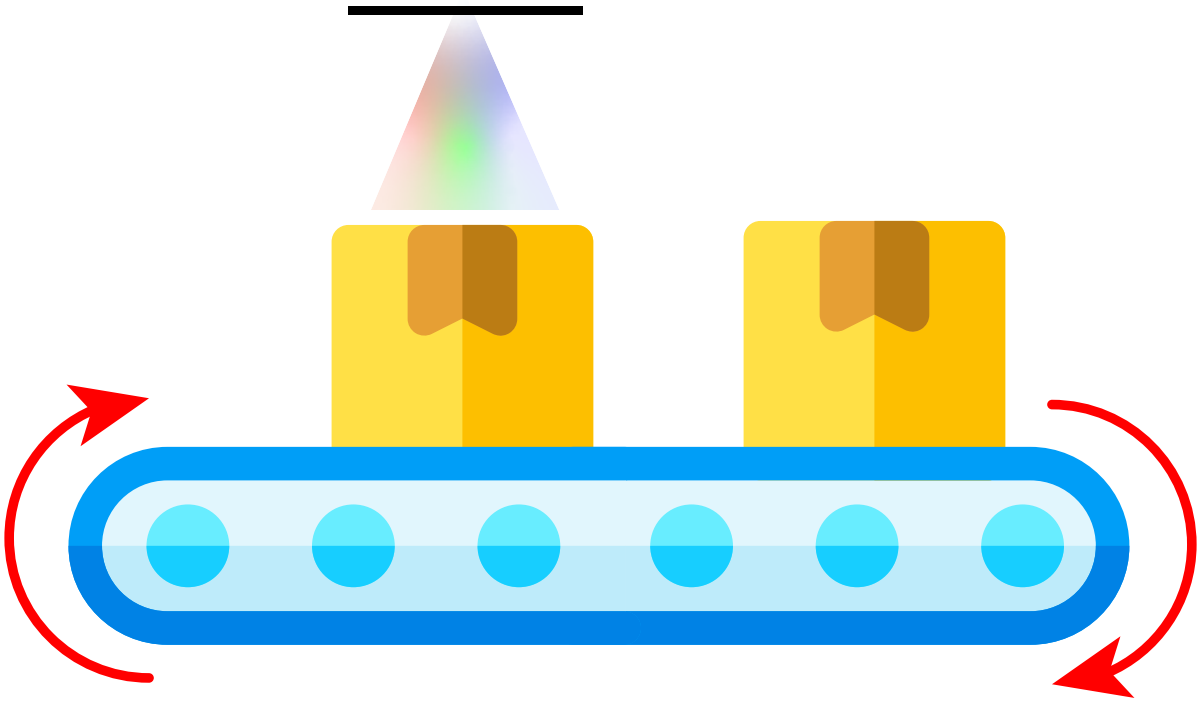


[1]白平衡：字面上的理解是白色的平衡。白平衡是描述显示器中红、绿、蓝三基色混合生成后白色精确度的一项指标。白平衡是电视摄像领域一个非常重要的概念，通过它可以解决色彩还原和色调处理的一系列问题。

怎么校准才更好

为了解决这个问题，我们其实应该很清楚了，方法就是，我们校准的位置，要比预定的位置更近一些才行，这样能保证充足的动态裕度。我们举例，标准距离为 100mm 的物体，那我们校准时应该在比 100mm 稍微近些的距离进行校准，可以是 70、80、90。这个需要出错以后慢慢尝试。如果程序上从未出错，则不必管这章节的知识。

如果像这种检测场景，那么我们仅需要按照标准校准来做了。因为他的检测距离十分固定，即使有波动，也没能力产生检测干扰。

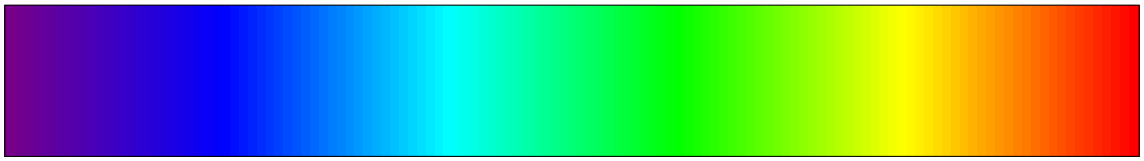


3 测量原理

3.1 光学基础

3.1.1 白光的组成

想了解主动式测量原理，需要先从被动式测量[2] 说起。我们在物理中曾经学过，白光他包含了赤、橙、黄、绿、青、蓝、紫所有颜色。这些颜色组合在一起构成了白光，所以说白光是一种合成光。



那么白卡纸为什么是白色的呢？原因是白卡纸反射了几乎所有颜色的光，所以我们能看到白色。

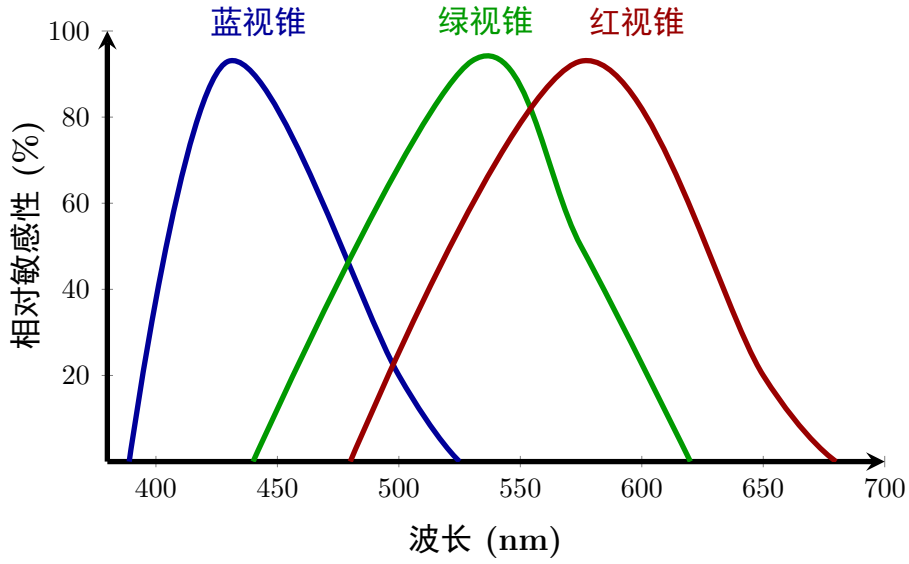
那么红卡纸为什么是红色的呢？因为它吸收了所有的光，只有红光被反射了出来，于是我们只能看到红色了。

其他颜色也是同样的道理，由此我们可以得出结论，物体的颜色与所反射的光色有关。

如果按照这个结论，如果要检测物体的颜色，我们应该物体打一束白光，并且在传感器上放置无数个波长不同的探头，用于检测反射光的颜色，这样就可以了。但实际上我们不需要的这么多探头，我们人类进化出来的眼睛提供给了我们思路。

3.1.2 人眼中的颜色

在我们的眼睛当中，有三种用于感知颜色的视锥细胞，他们分别对红绿蓝光敏感，他们敏感的曲线如图所示：

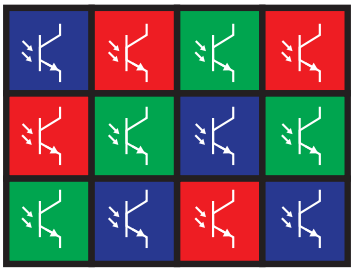


可以看到，眼睛中的三种视锥细胞完整覆盖了整个波长范围。仅需要分析各个三个视锥细胞的信号值，就可以得到完整的颜色信息。这三个视锥所敏感的颜色，我们称之为三原色。在这个理论基础上，发明了彩色电视、照相机等设备。

[2]被动测量：被动式测量很常见，摄像头就是一种被动测量，他会捕捉环境中的所有光线，并记录下来，从而分析颜色

3.2 被动光式传感器原理

传感器也在此基础上获得灵感，在感光元件顶端镀上一层滤光膜，他们仅允许红绿蓝三色光通过，分布在测光单元上组成阵列（RGB 阵列的目的是增加稳定性），由此来判断颜色。这就构成了被动光式颜色传感器，如图所示。

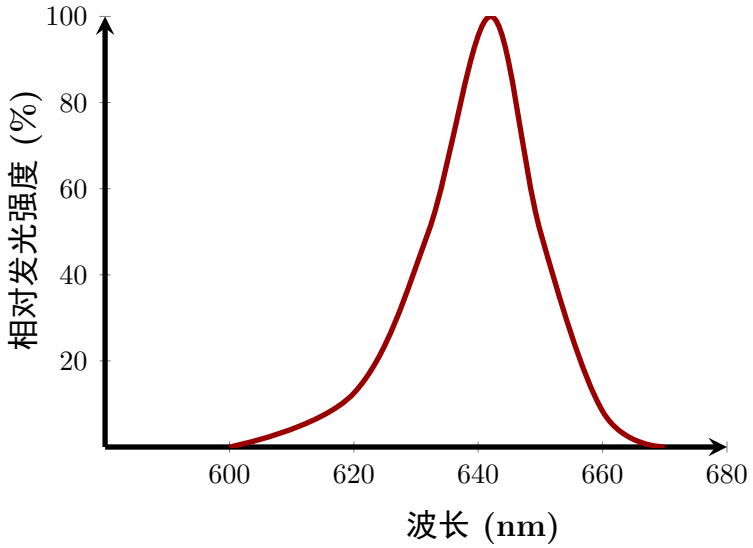


但是你有没有发现，如果照在物体上的光不是白色，例如说蓝色，那么是不是严重就偏色了呀，这是被动式传感器最大的问题，即受环境光影响较大，强依赖于提供的环境光。总结一下，被动光式颜色传感器，从原理上，是将一束白光照射到物体表面，然后经卡纸颜色的吸收，最终反射到传感器上，传感器有至少 3 种 RGB 镀膜的感光元件. 他们分别感知着 RGB 三种颜色，这种测量方式强依赖于环境光，极易受影响。

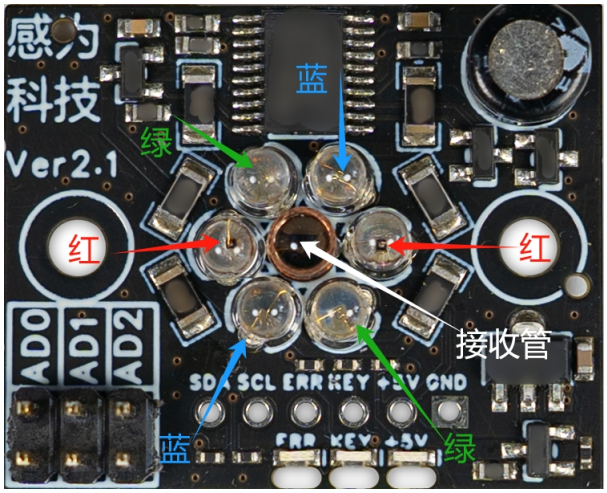
3.3 主动光式传感器原理

3.3.1 初识主动光式原理

我们了解完被动光式颜色传感器后，我们再学习主动光式颜色传感器就不难了。我们都知道 LED 的发光光谱实际上并不是一种波长，而是多种频率复合的一条曲线。这个曲线是不是看起来跟人眼红色视锥细胞曲线相似？



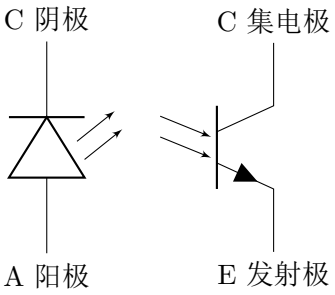
是的没错，如果将 RGB 三盏灯的曲线都铺在坐标轴上，那么跟视锥细胞的曲线就相似了。既然这样，我们可以换个思路：分别照射红绿蓝三色光，用一种全光谱敏感的感光元件来检测这三种波长的强度，感觉从原理上讲，一样可以检测出物体的颜色，于是就有了这款颜色传感器。



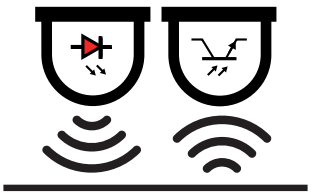
这种检测原理，我们称之为主动光式，接下来我们仔细研究一下如何在工程中实现。

3.3.2 如何实现测量

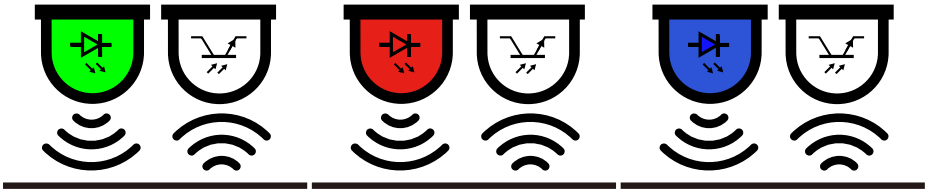
我们直接发射三束 RGB 光，不同颜色的物体会对三种波长有不同的吸收值，此时我们检测每种波长的反射大小，这样就可以测得物体的颜色了。对于物体的反射强度，我们通过光电对管来检测，光电对管是由一组发光二极管与光电三极管组成：



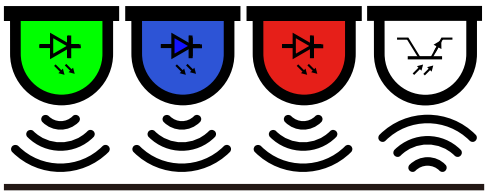
当发光二极管发出的光被检测物反射后，反射出来的光有一部分光会被光电三极管接收，而光电三极管是对光敏感的器件，当光强变大时，输出的电流将会变大，从而将光信号转化为电信号。



在颜色传感器中，像这样的对管共有 3 组，分别测量在 620nm、520、470nm 波段的反射强度，他们分别对应着红光、绿光、蓝光的值。



我们发现，在三组光电对管中，光电三极管都是相同的。那么我们可以通过分时复用的方式，节省 2 个光电三极管。三个 LED 交替亮起，从而测量出每一个波长。



这样的做法有一个很好的优点，即光色的一致性较高，我们都知道，在工业制造的 LED，其光强一般来说波动不会很大，一致性较好。但光电三极管的一致性相对较差，通过共用一个光电三极管，可以有效的避免 RGB 三个信号带来的强度波动，大大的增加了产品的一致性。

3.4 RGB 与 HSL 色彩格式

3.4.1 RGB 格式

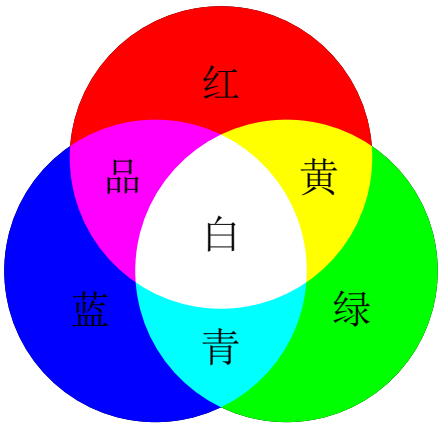
经传感器的检测后，我们能得到一组红绿蓝数据，我们称之为 RGB 格式，这个格式的数据可以通过 I^2C 直接读出，数据为 8bit 色深，即 0~255，这与大多数显示器、照相机的数据格式相同。

3.4.2 HSL 格式

根据格拉斯曼颜色混合定律，人的视觉智能分辨颜色的三种变化：色相 H、饱和度 S、明度 L，这三种特性可以统称为颜色的三属性。

H：色相

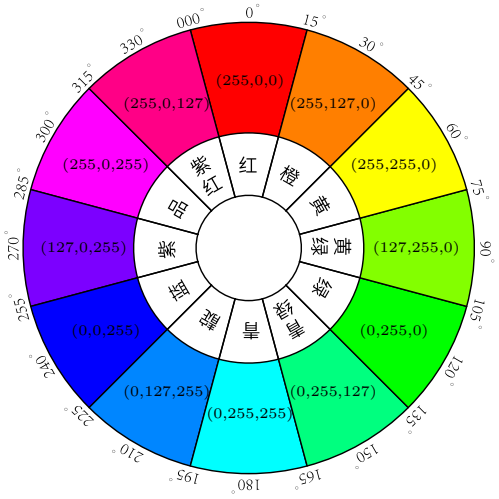
色相 H 的取值是 0-360 度，这源于色相是由三原色掺杂而产生的，最终的色谱构成一个圆的样式，所以色相取值为 0-360 度，所构成的色谱叫做色相环。其原理如图所示：



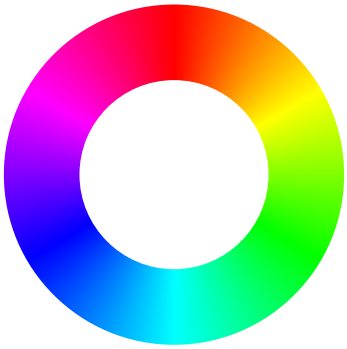
色相环是由红、绿、蓝三原色经过多次融合而形成的环状色相图。以下色彩均由 RGB 格式表示：

首先红 (255,0,0)、绿 (0,255,0)、蓝 (0,0,255) 三原色相互融合生成黄 (255,255,0)、青 (0,255,255)、品 (255,0,255)，这三种颜色称之为三间色。

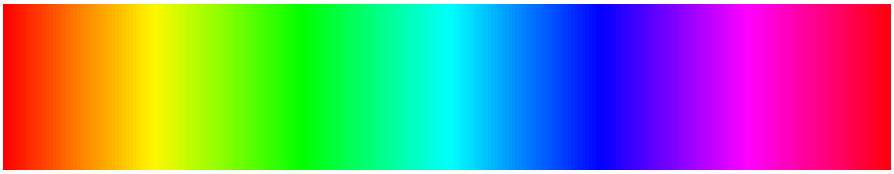
再将三原色与三间色相邻相容，形成二次间色，橙 (255,127,0)、黄绿 (127,255,0)、青绿 (0,255,127)、靛 (0,127,255)、紫 (127,0,255)、紫红 (255,0,127)。例如红 (255,0,0) 与黄 (255,255,0) 相容成为橙 (255,127,0)；青 (0,255,255) 与蓝 (0,0,255) 相容成为靛 (0,127,255)。



以此类推相邻相容，无限细分，就形成了标准的 RGB 色相环，但通常上，12 个色相就足够满足工程上的需要，无需再度细分。

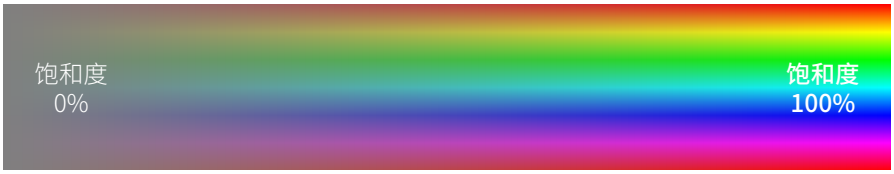


我们可以由 0 度切开——拉直，就形成了 0-359 度的线性图，如下图所示：



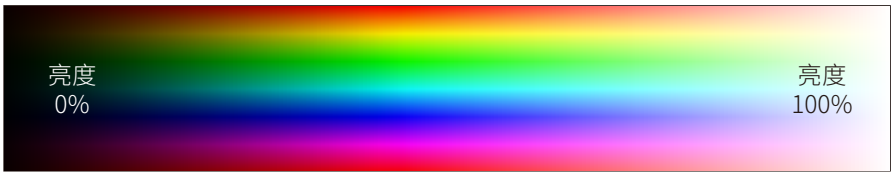
S：饱和度

指的是色彩的饱和度，它用 0% 至 100% 的值描述了相同色相、明度下色彩纯度的变化。数值越大，颜色中的灰色越少，颜色越鲜艳，呈现一种从灰度到纯色的变化。



L: 亮度

HSL 的 L(lightness) 分量，指的是色彩的明度，作用是控制色彩的明暗变化。它同样使用了 0% 至 100% 的取值范围。数值越小，色彩越暗，越接近于黑色；数值越大，色彩越亮，越接近于白色。



3.4.3 RGB 转 HSL 公式

如果您使用感为颜色传感器，您无需了解 RGB 是如何转为 HSL 的，因为传感器内部有转换算法，您仅需要发送相应的命令 (0xD1)，即可直接读取。

但是如果您用的不是感为颜色传感器的话，您就有必要在您的项目中应用这个算法了，在计算之前，请注意以下问题。

- 当您读取到一组 RGB 数据之时，您首先要对数据做归一化处理，做个映射，即 0-255 映射为 0-1，接下来的公式 RGB 的取值均为 0-1。
- max 为 RGB 中最大的值，min 为 RGB 中的最小值。
- 该映射会使 int8_t 的数据变为 float 数据，可能会影响计算速度，您如果想提高速度，需要等比例扩大该公式，这需要您自行推导。
- 色相值由于内含除法，量纲被约分，您可以直接带入 0-255 的数据，不会影响输出数据值。而饱和度、亮度就不行了，需要等比例扩大公式，这需要您自行推导。
- $r, g, b \in [0, 1]$
- $H \in [0, 360)$
- $S, L \in [0, 1]$

$$H = \begin{cases} 0° & if\ max = min \\ 60° \times \frac{g-b}{max-min} + 0° & if\ max = r\ and\ g \geqslant b \\ 60° \times \frac{g-b}{max-min} + 360° & if\ max = r\ and\ g < b \\ 60° \times \frac{b-r}{max-min} + 120° & if\ max = g \\ 60° \times \frac{r-g}{max-min} + 240° & if\ max = b \end{cases}$$

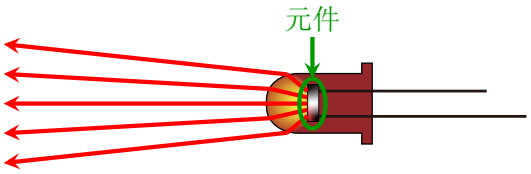
$$L = \frac{1}{2}(max + min)$$

$$S = \begin{cases} 0 & if\ L = 0\ or\ max = min \\ \frac{max-min}{2L} & if\ 0 < L \leqslant \frac{1}{2} \\ \frac{max-min}{2-2L} & if\ L > \frac{1}{2} \end{cases}$$

3.5 参数对检测效果的影响

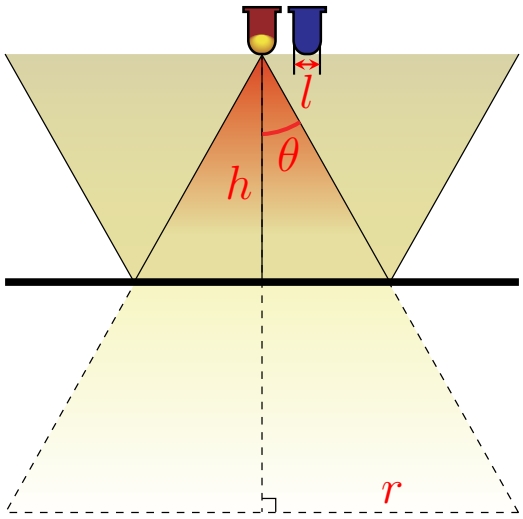
3.5.1 数学模型

要想研究检测距离对检测值的影响，必须了解光电对管的内部构成。其实不管是光电对管的发射管还是接收管，他们内部的发光或受光元件体积很小，且光路都是向外发散的。它们必须经过一个透镜，才能使用。经过透镜后，光路被束缚在一个较小的范围内，这个范围角度我们称之为光角。



加上透镜后，发射管能“照”的更远、“照”的更亮；接收管则在沿光路的方向上受光面积增大，而超出光角范围受光面积减小，从而能在特定的方向上捕捉更多的光线。这就像戴着老花镜的老奶奶，戴上老花眼镜后眼睛会变大一样。

为了进一步研究光在反射后的效果，我们假设检测面是光滑且能全反射的镜面。我们可以把光路的图像[3]绘制为如下所示。其中虚线部分为镜子反射的虚像，实线部分为实像，h 为对管距检测面的高度；r 为光路到达接收管时，光角扫过长度的一半；θ 为光角的一半，l 为接收元件经过透镜放大后的最大直径，即为透镜的直径。



假设反射的光线是均匀的，如图可知，如果 2 倍的 r 是发射光线的总和，则 l 可代表接收光线的总和。将 $\frac{l}{2r}$ 称之为相对进光量，即接收光/发射光，为一个百分数，比值接近 100% 时，证明发射的光线被全部接收。

$$\begin{aligned}\frac{l}{2r} &= \frac{l}{2\times(2\times h\times\tan\theta)}\\ &= \frac{l}{4h\tan\theta}\\ &= \frac{l}{4\tan\theta}\times\frac{1}{h}\end{aligned}$$

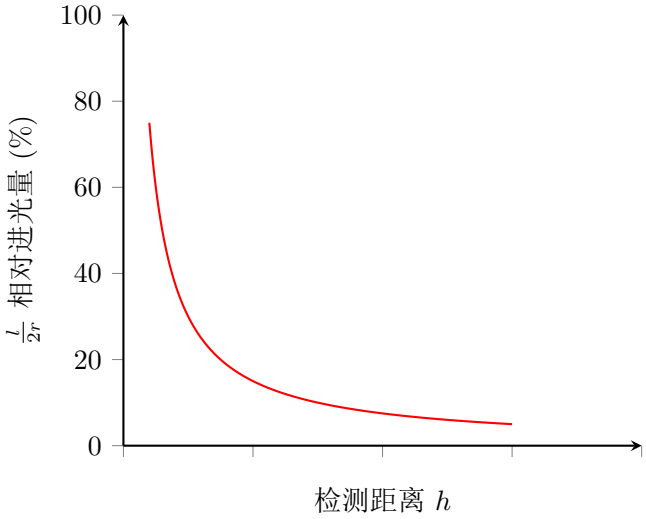
[3]这是在二维平面上的推导过程，在三维空间中，光角扫过的是一个圆锥体，在检测面上的投影是一个圆形，但这并不影响推导结果，差异被常数 c 吸收，趋势未发生变化，有兴趣的朋友可以自行推导。

3.5.2 检测距离对颜色亮度的影响

对上述公式，我们运用控制变量法，仅观测距离对检测值的影响。$\frac{l}{4\tan\theta}$ 中的 l、θ 为已知常量，可以将整体看做一个常数 c，则：

$$\frac{l}{2r}=\frac{c}{h}$$

我们发现，相对进光量与距离成反比例函数，也就是说，距离越远，颜色越黑。在 HSL 色彩格式中，表现为越远亮度 L 数值越低。



不难发现，当我们谈论 HSL 格式时，距离对检测结果仅仅影响了亮度 L 的变化，这也是我为什么极力推荐大家使用 HSL 格式的原因，他的抗干扰性比 RGB 好很多。

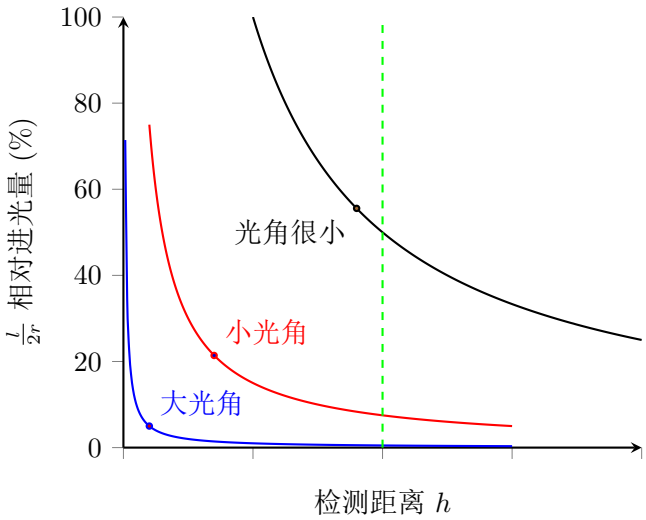
3.5.3 产品参数对检测距离的影响

在公式中，我们发现 $\frac{l}{4\tan\theta}$ 是反比例系数，这个系数与探头采样面积正相关，与光角成反比例。

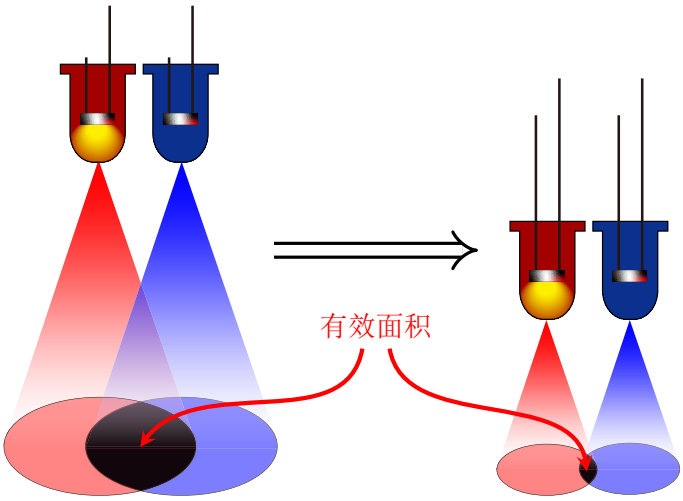
由于每个厂商的检测面积都差不太多，也就是 l 的值差不太多，但是光角是有所差异的，所以我们仅关注光角即可，我们可以通过下面的曲线直观的看到光角不同带来的区别。

我们希望的是检测结果随距离影响越小越好，而不是稍微挪动就近似为零。在下图的绿线可以看到，当不同光角处于同一距离 h 时，光角越小纵轴 $\frac{l}{2r}$ 值越大，所以应该减小光角以提高检测的精度。

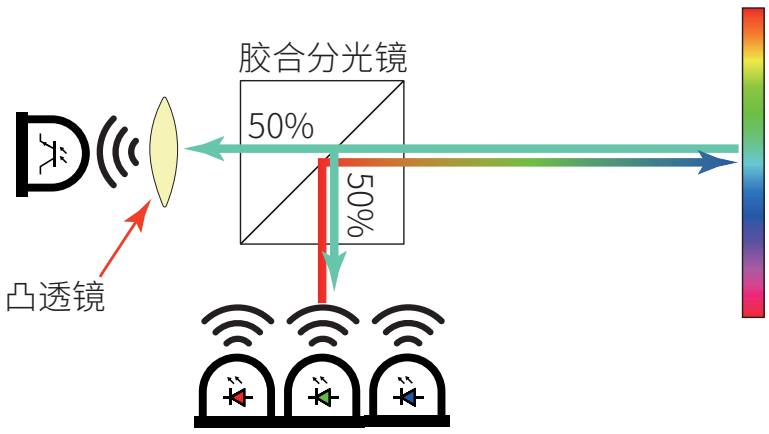
顺着思路我们极限假设一下，在理想情况下，激光近似为平行光。我们可以用激光进行测量，这时候由于测量光属于平行光，反比例系数近似无穷大。在公式 $\frac{\infty}{h}$ 中不管 h 为多大，$\frac{l}{2r}$ 还将是无穷大，表现为颜色传感器，无论 h 多大，相对进光量均为 100%，所以几乎不与检测距离相关，在同一检测面下，RGB 值几乎不会随距离发生变化，传感器也将无需校准，这就是激光测量的魅力所在。



不难发现，当同样检测高度下光角越小，相对进光量越大，越符合我们对于理想传感器的假设。如果是平行光，那么距离几乎不会干扰我们测量颜色，光被物体反射什么颜色，就会得到相应的数值，与距离无关，但是实际模型是这样的。



首先平行光需要非常精密的透镜产生，其次光的混合还需要类似于单反相机的棱镜产生，否则平行光的发射与接收也平行，彼此收不到信号，所以无法检测。但是还是有技术可以解决的，下图展示其中一种思路，但是生产成本会非常高，对于我们的应用场景来说，实在没必要。



4 设备的 I^2C 通讯

4.1 简介

颜色传感器使用 I^2C 总线通讯, 意在节约用户 I/O 资源。I^2C 总线仅需要两个 I/O 即可完成对 127 个设备的通讯。

该设备运行在 7 地址从模式下，支持配置 3 个硬件地址位、4 个软件地址位，全地址可设置，最多可配置 127 个颜色传感器。

4.2 设备的功能

- 读取 RGB 数据 4.6
- 读取 HSL 数据 4.7
- 配置软件地址 4.8
- ping 网络诊断工具 4.10
- 读取错误信息 4.11
- 设备软件复位重启 4.12
- 固件版本号查询 4.13

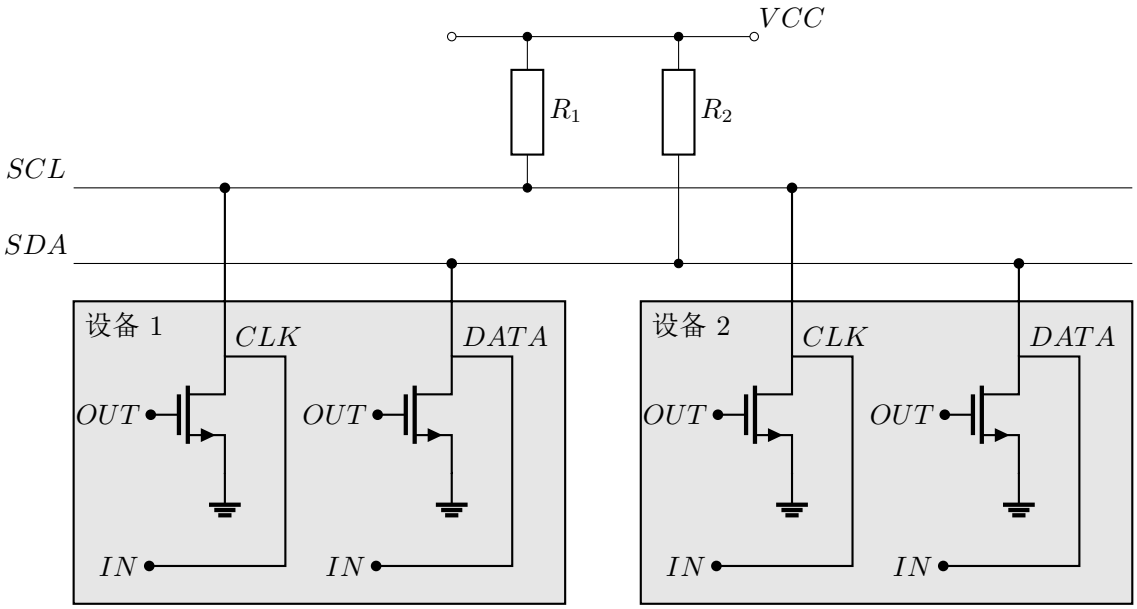
4.3 I^2C 协议回顾

这一章节中，将笼统的讲述 I^2C 的预备知识，为了方便速读或翻阅，我将这一章节的重点画出了出来，供大家参考。如果您对 I^2C 是零基础，阅读这些内容是远远不够的，您还需要自行查阅协议的具体内容，或者去互联网搜索相关的视频教程。

4.3.1 I^2C 硬件

I^2C 总线是 由飞利浦 (Philips) 公司开发的一种半双工[4]总线。它只需要两根线即可在连接于总线上的器件之间传送信息，这两根线分别是 SDA(串行数据线) 和SCL(串行时钟线)。一般情况下，在总线上的设备，I/O 均为开漏[5]状态，需要外接上拉电阻来实现通讯。

为什么是开漏的呢？因为设备间的运行电压可能不同，例如，有些设备是 5V 供电，有些设备是 3.3V 供电，这时候双方 I/O 输出的电压不相同，可能造成通讯失败。而 I/O 开漏仅会对总线进行拉低操作，也就是说逻辑“1”为总线电压，“0”为 0V，电平得到了统一。所以只要调配好总线为 5V，设置好总线的限流电阻 R_1, R_2(一般情况下为 $10k\Omega$)，即使总线电压略有差异，由于总线的总电流仅为 $500\mu A$，也可以安全通讯。



4.3.2 I^2C 基础

I^2C 总线是一种半双工通讯协议，同一时间只能由总线中的一个设备发送，为保证双向通讯，I^2C 采用分时复用的原理设计，所以它的通讯协议相较于其他协议，稍稍复杂些。为了准确书写 I^2C 代码，我们需要简单回顾一下。

I^2C 总线的通讯方，分为主设备 (Master) 与从设备 (Slave)，我们在日常应用中，单片机作为一个设备的控制中枢，常设置为主设备，因为主设备有权发起一次通讯，可以随时向从设备要数据。而作为一个提供服务的设备，则需要运行在从设备下，为主设备提供服务。

[4]半双工：数据可以在一个信号载体的两个方向上传输，但是不能同时传输。

[5]开漏：含义是 mosfet 的漏极开放，不接上拉电阻。这里的 mosfet 漏极类似于三极管的集电极，为了方便理解，请查阅“共射放大电路”，将 R_c 电阻去掉即为开集 (mosfet 的开漏)

I^2C 协议规定，仅主设备有权发起、结束一次通讯，其向总线发送的发起标志，称为起始位，发送的结束标志，称为停止位。每一次通讯必须含有起始位与停止位, 但不限制起始位与停止位之间传输的帧数。

在 I^2C 通讯中，主设备是通过地址，呼叫从设备的。这就像打电话，电话号码为手机呼叫的身份标志，建立通讯。I^2C 也需要一个身份标志完成通讯，这个身份标志就是地址。地址为 char 型[6]，其组成为：7bit[7] 地址位 +1bit 读写位，其中读写位决定了主从设备的传输方向，“0”表示：主机发送数据，“1”表示：主机接收数据。

char型bit位	7	6	5	4	3	2	1	0
作用及名称	从机地址							读/写

I^2C 通讯数据长度为 8bit，即每一帧数据长度为一个 char 型。在收到数据后，对方要回应一句“收到”，这个回应标志，称为 ACK 位。

4.3.3 I^2C 时序

I^2C 的时序，对于通讯来讲可分为为三种：主机向从机发送数据、从机向主机发送数据、主机先向从机发送数据然后从机再向主机发送数据，在下图中，展示了详细的时序图，其中，灰格子为主机行为，白格子为从机行为。

1、 主机向从机发送数据

S[i]	从机地址	0[ii]	A[iii]	数据	A	数据	$A/\overline{A}$	P[iv]

2、 从机向主机发送数据

S	从机地址	1	A	数据	A	数据	$A/\overline{A}$	P

3、 主机先向从机发送数据，然后从机再向主机发送数据

S	从机地址	0	A	数据	$A/\overline{A}$	S	从机地址	1	A	数据	$A/\overline{A}$	P

[i] **S**：I^2C 起始位。表格中灰色格子为主机发送。

[ii] **读写位**：“0”表示主机发送数据，’1’ 表示主机接收数据。

[iii] **A**：I^2C 应答位。

[iv] **P**：I^2C 停止位。

1(主–从)、2(从–主) 完全按照章节：4.3.2 中描述的样子运行，不难理解。而第 3 种(主–从–从–主) 的形式比较难以理解，其实这种形式，也仅仅是将 1(主–从)、2(从–主) 两种形式，“复制–粘贴”式的拼合起来。

唯独不同的是，中间切换过程并未发送停止位，原因是，一旦中途发送停止位，总线 就从繁忙变为空闲状态，这时如果有其他设备来抢总线，极易出现传输失败的情况。

这三种都是标准形式，我们应该养成按照标准写程序的习惯，尽管您的总线中可能只有一个主设备。按 照标准写程序，在很大程度上，能避免 bug 的出现。

[6]char：C 语言中的一个数据类型，由 8 个比特组成。

[7]bit：二进制的一位为 1 比特，名称为 bit，与十进制的个十百千万类似

4.4 通讯语法及结构

4.4.1 语法简介

通讯语法，是为了丰富通讯而建立的。如果单单读数据或是写数据，双方没有约定内容，那么设备就不清楚要读什么数据，或者要写什么数据了。这时候我们需要建立出一个会话协议出来，让对方知道在说些什么，这样就明白对方要传输什么样的数据了。

4.4.2 语法

语法十分简单：先传输一个命令，再传输命令所需的任意个的数据，其中“命令”仅允许主设备发往从设备，“数据”主、从设备均可发送。

命令 (主设备发送)	数据 (从、主设备发送)	数据 (传输方向与上一帧相同)	...

举个例子：以老板命令员工扫落叶为例，老板先告诉员工：“把落叶扫了，然后装垃圾袋里交给我”员工收到命令后，把地上的落叶扫了起来，用垃圾袋打包好，递交给老板，从而执行完一次命令。这里老板就是主设备，员工就是从设备。这时候我们就可以抽象为，命令 [主设备发送]+ 数据 [从设备发送] 的语句了。

把例子等价为颜色传感器的通讯：以传输数字数据为例，首先主设备发送一个命令给从设备，代号为 0xD0，告诉颜色传感器：“把 RGB 数据，打包起来发送给我”，从设备接收到 0xD0 后，立即将该数据打包上传给主设备，这就完成了一次简单的通讯。

4.4.3 结构

那么语法在 I^2C 协议中结构是如何的呢？或者说我们该如何写程序呢？我们来探究一下以防编程出错。

在前一节：I^2C **时序4.3.3**中提到，I^2C 通讯传输分为三种，分别是：主设备读、主设备写、主设备先写再读。在本章节中，我们将仅用到主设备写、主设备先写后读的功能。

1、主设备写：

主设备写数据，常用在类似于更改软件地址等参数中，其语法为: 命令 + 数据，结合 I^2C **时序4.3.3**中的内容，其通讯结构为：

S	从机地址	0	A	命令	A	数据 1	A	数据 2	A	...	数据 N	$A/\overline{A}$	P

注：图中灰色为主机行为，白色为从机行为。从机地址后紧跟着读写位，主机写为“0”，主机读为“1”。

2、主设备先写再读：

主设备先写再读，常用在类似于传输 RGB 数据，HSL 数据等应用中，其语法为: 命令 + 数据，其通讯结构为：

S	从机地址	0	A	命令	A	S	从机地址	1	A	数据 1	A	...	数据 N	$A/\overline{A}$	P

↑ 写数据　　↑ 读数据

需要注意的是，在这个模式中，我并没有在写数据与读数据之间加停止位，原因是，如果加停止位，总线将被释放，如果总线被其他设备抢走，这时候需要等待总线空闲后，才能读取后续数据。

4.5 地址位设置及实例

4.5.1 地址位设置

在这个章节中，我们探讨硬件地址的设置方法，如需软件配置地址请参考章节:4.8

在 I^2C 总线协议中，设备间的通讯，是通过主设备呼叫从设备的地址实现的。使用该传感器时，需要将您的设备设置在主模式下，而颜色传感器则运行在从设备模式下。从设备的地址是由 7bit 地址位 +1bit 读写位组成的，在地址位中高 4bit 为软件地址位，由软件配置，出厂数据为 0b1001，低 3bit 为硬件地址位，由安装跳线帽[8]配置，读写位决定了传输的方向。其组成形式如下所示：

S地址位7[i]	S地址位6	S地址位5	S地址位4	H地址位3[ii]	H地址位2	H地址位1	读写位0
1	0	0	1	AD2[iii]	AD1[iv]	AD0[v]	X[vi]

[i] S 地址位: 含义为软件地址位 (software address)，由软件配置, 出厂时其地址位 1001 XXXX，具体细节参考章节:4.8 。

[ii] H 地址位: 含义为硬件地址位 (hardware address), 由板载插针、跳线帽配置，不装跳线帽时为“0”。

[iii] AD2: 电路板上跳线帽安装位置，有对应的丝印层标记，其名称为 AD2，安装跳线帽后，由“0”变为“1”。

[iv] AD1: 电路板上跳线帽安装位置，有对应的丝印层标记，其名称为 AD1，安装跳线帽后，由“0”变为“1”。

[v] AD0: 电路板上跳线帽安装位置，有对应的丝印层标记，其名称为 AD0，安装跳线帽后，由“0”变为“1”。

[vi] X: 其含义为任意数值，不管该位是“0”或“1”。这一节讲地址位，与读写位无关，但其与地址位组成一帧数据，故必须示意，但不予理睬。

4.5.2 地址程序实例

上一节中，我们讲述了如何用硬件配置 I^2C 地址，即通过安装跳线帽的方式，来改变设备的地址。在这一节中，我们将讲述如何在不同的编程习惯下，正确写入设备的地址，首先我们假设插针 AD2、AD1、AD0 均安装有跳线帽，这时 H地址3=1H地址2=1，H地址1=1。按照上述表格，地址为：0b1001111X[9]

在编写程序时，如果您用寄存器编程，那么请在对应的寄存器写入该地址 0x9E[10], 即 0b1001111X，以给 stm8 编程为例:

```
/*I2C_OARL为stm8单片机的地址寄存器，需要将设备地址写入该寄存器。*/
I2C_OARL &= 0x01;//地址位清零,为了不影响0位，先将高7位清零。按位与0x01=0b0000 0001,结果为：0b0000 000X
I2C_OARL |= 0x9E;//写入地址,为了不影响0位，按位或0x9E=0b1001 1110。结果为：0b1001 111X
```

在程序编写时，如果您用库函数编程，那么地址为 0x4F。请注意, 由于大多数 I^2C 库函数的内部会将地址左移，目的是，在左移后的第 0 位插入读写位，以建立通讯方向，所以在提供地址时，应该相应的，要右移一位，即 0b1001 111X>>=1; 结果为 0b01001111=0x4F，以 ESP32 写法为例：

```
#include <Wire.h>
#define GW_GRAY_ADDR 0x4f //本应地址位0x9E，但Wire.h库会对地址左移一位，故要右移1位，即0x4f

/*函数setup与loop是arduinoIDE标准框架，setup是程序的入口，只运行一次，loop则是循环运行，类似while(1)*/
void setup(){
    Wire.begin();//Wire.h库中的初始化函数，运行它能让esp32使能IIC
}
void loop(){
    Wire.beginTransmission(GW_GRAY_ADDR);//通讯开始函数，它会对输入的地址进行左移操作。
    Wire.write(0xAA);//Wire.h库中的写函数，向从设备发送数据0xAA。
    Wire.endTransmission(1);//Wire.h库中的停止位函数，写入1为真，即产生停止位。
}
```

[8]跳线帽：默认不装地址跳线帽的情况下，为“0”，安装地址跳线帽后其地址位为“1”。

[9]0b：二进制标识符，与 0x 为十六进制标识符类似

[10]0x: 十六进制标识符，在 C 语言中，char 型有 8bit，可以由两个十六进制数表示。

4.6 读取 RGB 数据

这一章节中，我们将讲述如何读取 RGB 数据，我们都知道，RGB 数据是 3 个 8bit 数值，刚好相当于 3 帧数据，我们发送命令后

4.6.1 功能描述

读取 RGB 功能（命令符：0xD0），是将传感器得到的红绿蓝三通道的数值，通过 I^2C 的三帧发送出去，此数据称之为 RGB 数据。该功能为只读模式。

这三帧数据分别代表着红通道数值、绿通道数值、蓝通道数值。

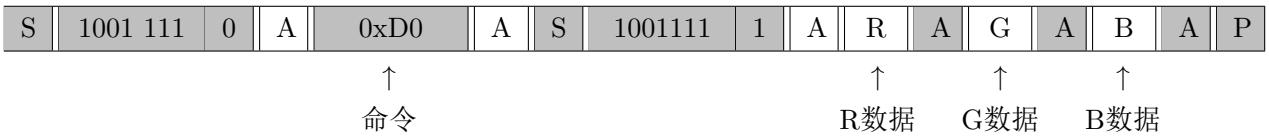
第一帧数据	第二帧数据	第三帧数据
红通道数值	绿通道数值	蓝通道数值

4.6.2 通讯方法

在上一章节: **语法4.4.2**中，我们提到，一条语句是由命令 + 数据组成的。如果您想获取 RGB 数据，需要在一条语句中先发送一个命令符：0xD0，再去读三帧数据，则所读得的数据即为 RGB 数据。

方法 1：

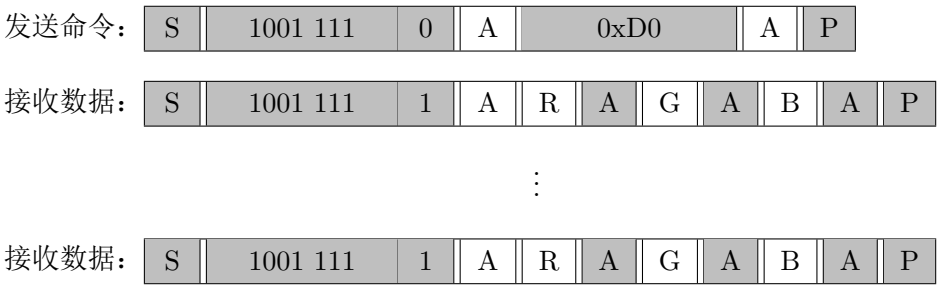
在以下说明中，我们都将从设备地址设置为 0b1001 111，以保证文章的一致性。方法 1，用的是标准的命令语句，是一种最基础的通讯方案。其结构为命令 + 数据的形式：



在整个语句中，在 I^2C 通讯中，占用了共计 54 个时钟时间，这个写法是没有问题的，完全能运行。但是我们所需的仅有 3 个 8bit 数据，却仅仅需只占 24 个时钟传输，外加应答位，共计 27 个时钟，最后浪费了 27 个时钟，接近 50% 是浪费的。如果效率这么低下，那么将会大大增加我们的通讯时间。

方法 2：

为了提高通讯效率，我们允许用户在只发一次该命令后反复读取该值，而不需要重复的发送命令。当然，前提是不要有新的命令覆盖 0xD0，所以如果您有覆盖该命令，请一定要在读取数据前发一条命令 0xD0，用于更新命令。方法 2 的结构为：



就这样，写一次命令，设备会记录当前命令，后续读取的数据皆为 0xD1 下的数据, 所需时钟为 38 个，只浪费了 11 个时钟。

方法 3：

当然，如果您想要更高的效率，您也可以先发送命令，再连续读取该数据。这种情况下，当读取到第 B 数据后，设备将从 R 读取。这样当读到第 4 帧时，该数据为第 R 数据。

发送命令：

S	0x9E	A	0xD0	A	P
	↑ 地址写		↑ 命令		

接收数据：

S	0x9F	A	0x34	A	0x47	A	...	A	0x39	A	0x34	A	...	A	P
	↑ 地址读		↑ R数据		↑ G数据				↑ B数据		↑ R数据				

方法 4：

命令所在的内存在初始化后，默认为 0xD0。所以您开机后, 命令默认是 0XD0，这时候，如果您仅仅要读取 RGB 数据，可以直接向设备要数据即可，结构为：

接收数据：

S	0x9F	A	R数据	A	G数据	A	...	A	B数据	A	R数据	A	...	A	P

4.6.3 代码示例

方法 1：

```
#include <Wire.h>
#define GW_GRAY_ADDR 0x4f //跳线帽全插的情况下
/*函数setup与loop是arduinoIDE标准框架，setup是程序的入口，只运行一次，loop则是循环运行，类似while(1)*/
void setup(){
    Wire.begin();//Wire.h库中的初始化函数
    ping();//ping函数，用于同步esp32与颜色传感器。文档在方法3之后有代码，请添加到您的程序中。
}
void loop(){
    unsigned char recv_value[3];//数据存放点
    //写命令
    Wire.beginTransmission(GW_GRAY_ADDR);//起始位函数
    Wire.write(0xD0);//写函数，发送命令0xD0
    Wire.endTransmission(0);//停止位函数，0为不产生停止位。
    // 读取数据
    Wire.requestFrom(GW_GRAY_ADDR/*地址*/,3/*读3个char*/,1/*停止位*/);//读数据的集成函数。
    for (unsigned int i = 0; i < 3; ++i) {
        recv_value[i] = Wire.read();//取出Wire.requestFrom运行后的数据。
    }
}
```

方法 2:

```
#include <Wire.h>
#define GW_GRAY_ADDR 0x4f //跳线帽全插的情况下

/*函数setup与loop是arduinoIDE标准框架，setup是程序的入口，只运行一次，loop则是循环运行，类似while(1)*/
void setup(){
    Wire.begin();//初始化函数
    ping();//ping函数，用于同步esp32与颜色传感器。文档在方法3之后有代码，请添加到您的程序中。
    //写命令
    Wire.beginTransmission(GW_GRAY_ADDR);//起始位函数。
    Wire.write(0xD0);//写函数，发送命令0xDD。
    Wire.endTransmission(1);//停止位函数，写入1为真，即产生停止位。
}
void loop(){
    unsigned char recv_value[3];//数据存放点。
    //读数据
    Wire.requestFrom(GW_GRAY_ADDR/*地址*/,3/*读3个char*/,1/*停止位*/);//读数据的集成函数。
    for (unsigned int i = 0; i < 3; ++i) {
        recv_value[i] = Wire.read();//取出Wire.requestFrom运行后的数据。
    }
}
```

方法 3:

```
    #include <Wire.h>
    #define GW_GRAY_ADDR 0x4f //跳线帽全插的情况下

    /*函数setup与loop是arduinoIDE标准框架，setup是程序的入口，只运行一次，loop则是循环运行，类似while(1)
        */
    void setup(){
        Wire.begin();//初始化函数
        ping();//ping函数，用于同步esp32与灰度传感器。
        //写命令
        Wire.beginTransmission(GW_GRAY_ADDR);//起始位函数。
        Wire.write(0xD0);//写函数，发送命令0xB0。
        Wire.endTransmission(1);//停止位函数，写入1为真，即产生停止位。
    }
    void loop(){
        unsigned char recv_value[24];//数据存放点

        //一次性连续读取8组RGB数据，共计24个，存放在recv_value中
        Wire.requestFrom(GW_GRAY_ADDR/*地址*/,24/*读24个char*/,1/*停止位*/);//读数据的集成函数。
        for (unsigned int i = 0; i < 24; ++i) {
            recv_value[i] = Wire.read();//取出Wire.requestFrom运行后的数据。
        }
    }
```

方法 4：

```
#include <Wire.h>
#define GW_GRAY_ADDR 0x4f //跳线帽全插的情况下

/*函数setup与loop是arduinoIDE标准框架，setup是程序的入口，只运行一次，loop则是循环运行，类似while(1)*/
void setup(){
    Wire.begin();//初始化函数
    ping();//ping函数，用于同步esp32与颜色传感器。文档在方法3之后有代码，请添加到您的程序中。
}
void loop(){
    unsigned char recv_value[3];//数据存放点。
    //读数据
    Wire.requestFrom(GW_GRAY_ADDR/*地址*/,3/*读3个char*/,1/*停止位*/);//读数据的集成函数。
    for (unsigned int i = 0; i < 3; ++i) {
        recv_value[i] = Wire.read();//取出Wire.requestFrom运行后的数据。
    }
}
```

在上述例程中，均在初始化时，有一个 ping 函数，详情请参阅4.10。其目的是解决一个 bug，当您的主控与颜色传感器同时上电时，您的主控初始化可能会比颜色传感器初始化快，这时候如果发命令，命令可能传达不到，收数据可能数据不对。所以需要在您主控的初始化函数中，写入 ping 函数。此函数会不停的发送命令 0xAA，然后去读取数据。当颜色传感器初始化完成后，收到了 0xAA 命令，它会理解为 ping 命令，这时如果它在线，它会返回 0x66。此时您的主控如果读到 0x66，则会跳出循环。这个方法可以确保您的主控与颜色传感器是同步初始化的，这样可以减少了数据错误的发生，其函数为：

```
void ping(void)
{
    char sensors_status;
    while(sensors_status!=0x66)//开机判断，如果读出的数据为0x66，则停止循环。
    {
        Wire.beginTransmission(GW_GRAY_ADDR);//起始位函数。
        Wire.write(0xAA);//写函数，发送命令0xAA
        Wire.endTransmission(0);//停止位函数，0为不产生停止位。
        Wire.requestFrom(GW_GRAY_ADDR/*地址*/,1/*读1个char*/,1/*停止位*/);//读数据的集成函数。
        sensors_status = Wire.read();//取出Wire.requestFrom运行后的数据。
    }
}
```

4.7 读取 HSL 数据

在上一章节4.6中，详细的讲述了 RGB 是如何读取的，这一章节讲述的内容与读取 RGB 数据相似。

4.7.1 功能描述

读取 HSL 功能（命令符：0xD1），是将传感器计算出来的色相、饱和度、亮度三个值，通过 I^2C 的三帧发送出去，此数据称之为 HSL 数据。该功能为只读模式。

这三帧数据分别代表着色相数值、饱和度数值、亮度数值。

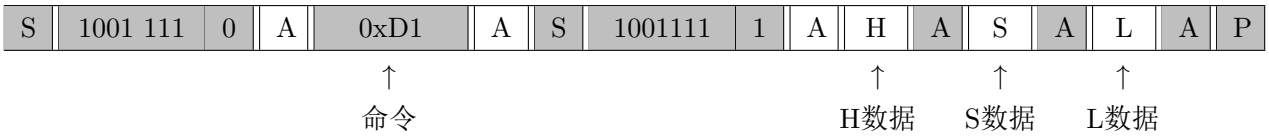
第一帧数据	第二帧数据	第三帧数据
色相数值	饱和度道数值	亮度道数值

4.7.2 通讯方法

在上一章节: **语法4.4.2**中，我们提到，一条语句是由命令 + 数据组成的。如果您想获取 HSL 数据，需要在一条语句中先发送一个命令符：0xD1，再去读三帧数据，则所读得的数据即为 HSL 数据。

方法 1：

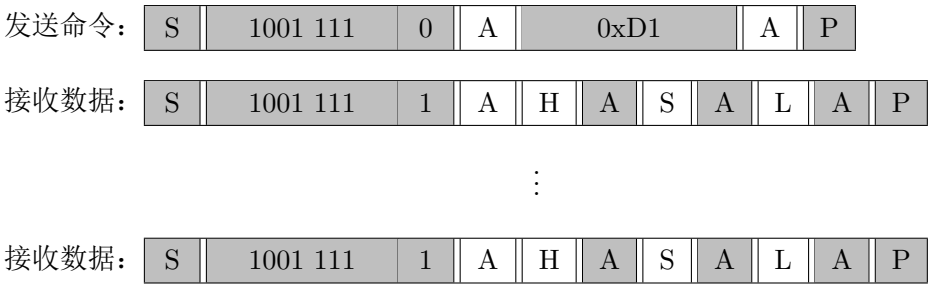
在以下说明中，我们都将从设备地址设置为 0b1001 111，以保证文章的一致性。方法 1，用的是标准的命令语句，是一种最基础的通讯方案。其结构为命令 + 数据的形式：



在整个语句中，在 I^2C 通讯中，占用了共计 54 个时钟时间，这个写法是没有问题的，完全能运行。但是我们所需的仅有 3 个 8bit 数据，却仅仅需只占 24 个时钟传输，外加应答位，共计 27 个时钟，最后浪费了 27 个时钟，接近 50% 是浪费的。如果效率这么低下，那么将会大大增加我们的通讯时间。

方法 2：

为了提高通讯效率，我们允许用户在只发一次该命令后反复读取该值，而不需要重复的发送命令。当然，前提是不要有新的命令覆盖 0xD1，所以如果您有覆盖该命令，请一定要在读取数据前发一条命令 0xD1，用于更新命令。方法 2 的结构为：

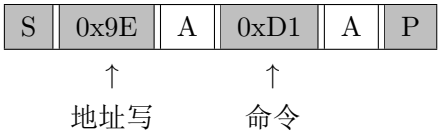


就这样，写一次命令，设备会记录当前命令，后续读取的数据皆为 0xD1 下的数据, 所需时钟为 38 个，只浪费了 11 个时钟。

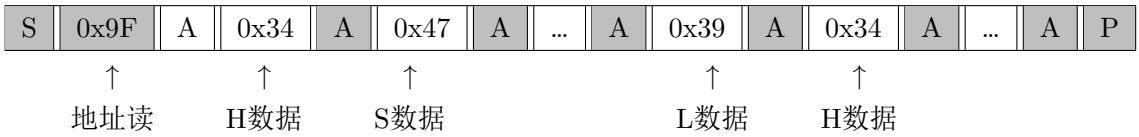
方法 3：

当然，如果您想要更高的效率，您也可以先发送命令，再连续读取该数据。这种情况下，当读取到第 L 数据后，设备将从 H 读取。这样当读到第 4 帧时，该数据为第 H 数据。

发送命令：



接收数据：



4.7.3 代码示例

方法 1：

```
#include <Wire.h>
#define GW_GRAY_ADDR 0x4f //跳线帽全插的情况下
/*函数setup与loop是arduinoIDE标准框架，setup是程序的入口，只运行一次，loop则是循环运行，类似while(1)
    */
void setup(){
    Wire.begin();//Wire.h库中的初始化函数
    ping();//ping函数，用于同步esp32与颜色传感器。文档在方法3之后有代码，请添加到您的程序中。
}
void loop(){
    unsigned char recv_value[3];//数据存放点
    //写命令
    Wire.beginTransmission(GW_GRAY_ADDR);//起始位函数
    Wire.write(0xD1);//写函数，发送命令0xD1
    Wire.endTransmission(0);//停止位函数，0为不产生停止位。
    // 读取数据
    Wire.requestFrom(GW_GRAY_ADDR/*地址*/,3/*读3个char*/,1/*停止位*/);//读数据的集成函数。
    for (unsigned int i = 0; i < 3; ++i) {
        recv_value[i] = Wire.read();//取出Wire.requestFrom运行后的数据。
    }
}
```

方法 2：

```
#include <Wire.h>
#define GW_GRAY_ADDR 0x4f //跳线帽全插的情况下

/*函数setup与loop是arduinoIDE标准框架，setup是程序的入口，只运行一次，loop则是循环运行，类似while(1)*/
void setup(){
    Wire.begin();//初始化函数
    ping();//ping函数，用于同步esp32与颜色传感器。文档在方法3之后有代码，请添加到您的程序中。
    //写命令
    Wire.beginTransmission(GW_GRAY_ADDR);//起始位函数。
    Wire.write(0xD1);//写函数，发送命令0xD1。
    Wire.endTransmission(1);//停止位函数，写入1为真，即产生停止位。
}
void loop(){
    unsigned char recv_value[3];//数据存放点。
    //读数据
    Wire.requestFrom(GW_GRAY_ADDR/*地址*/,3/*读3个char*/,1/*停止位*/);//读数据的集成函数。
    for (unsigned int i = 0; i < 3; ++i) {
        recv_value[i] = Wire.read();//取出Wire.requestFrom运行后的数据。
    }
}
```

方法 3：

```
#include <Wire.h>
#define GW_GRAY_ADDR 0x4f //跳线帽全插的情况下

/*函数setup与loop是arduinoIDE标准框架，setup是程序的入口，只运行一次，loop则是循环运行，类似while(1)*/
void setup(){
    Wire.begin();//初始化函数
    ping();//ping函数，用于同步esp32与灰度传感器。
    //写命令
    Wire.beginTransmission(GW_GRAY_ADDR);//起始位函数。
    Wire.write(0xD0);//写函数，发送命令0xB0。
    Wire.endTransmission(1);//停止位函数，写入1为真，即产生停止位。
}
void loop(){
    unsigned char recv_value[24];//数据存放点

    //一次性连续读取8组RHSL数据，共计24个，存放在recv_value中
    Wire.requestFrom(GW_GRAY_ADDR/*地址*/,24/*读24个char*/,1/*停止位*/);//读数据的集成函数。
    for (unsigned int i = 0; i < 24; ++i) {
        recv_value[i] = Wire.read();//取出Wire.requestFrom运行后的数据。
    }
}
```

4.8 软件地址配置

在前一节4.5我们提到如何设置硬件地址，在这一节中，我们将讲述软件地址如何配置。

4.8.1 功能描述

配置软件地址 (命令符：0xAD 0xAD。命令符则需要连续发送两个 0xAD，其目的是解决命令误触导致地址无意中被改掉，加长命令有助于容错)，是将设备的 I^2C 地址，通过软件代码进行更改的一种功能。在 I^2C 协议中规定，地址帧由 8bit 组成，其中 7bit 为地址位，1bit 为读写位。7bit 地址位中，设备规定 4bit 高位为 S(软件) 地址位，3bit 低位为 H(硬件) 地址位，其中 H 地址位与 S 地址配置方式不同，相互独立，S 地址的默认值为 0b1001，其结构为：

S地址位7	S地址位6	S地址位5	S地址位4	H地址位3	H地址位2	H地址位1	读写位0
1	0	0	1	AD2	AD1	AD0	X

配置地址的方式很简单，仅需要先发送命令 0xAD 0XAD，再发送软件地址位信息即可。设备将只读取 8bit 之中的 4bit 高位数据，自动忽略其中的 4bit 低位数据。如果高 4bit 皆为 0，则此次传输失效，回滚为原地址，其结构为：

发送完 S 地址位信息后，需要将设备重启，以生效该地址, 可以发送命令:0xC0，即可重启设备，否则设备将延续旧的地址，直至重启，或者重新开机。

请注意：在改地址时，我们应该避免多个设备的地址相同，因为那样，可能在更改地址时，一次更改多个设备的S地址，造成设置混乱。所以在改地址时，我们应该将设备分配不同的H地址，超过 4 个后，需要分批挂载总线配置。

4.8.2 用法及例程

用法：

```
#include <Wire.h>
#define GW_GRAY_ADDR 0x4f //跳线帽全插的情况下
/*函数setup与loop是arduinoIDE标准框架，setup是程序的入口，只运行一次，loop则是循环运行，类似while(1)*/
void setup(){
    Wire.begin();//Wire.h库中的初始化函数
    ping();//ping函数，用于同步esp32与颜色传感器。
    Wire.beginTransmission(GW_GRAY_ADDR);//起始位函数
    Wire.write(0xAD);
    Wire.write(0xAD);
    Wire.write(0xA0); //举例为新地址，0b1010 xxxx
    Wire.endTransmission(1);//终止通讯，发送停止位
    //最后重启设备，地址生效。软件重启的命令后续会学到
}
void loop(){
    // ...
}
```

4.9 广播重置地址与扫描找回地址

如果您在使用过程中，忘记已经设置好的地址，或者是误发了设置地址命令，导致地址丢失。这时候您不必担心，我们提供两种方法解决此问题。

方法 1：通过广播重置地址

您可以通过呼叫广播地址 0x00，来重置出厂地址 (0b1001 XXXX)，要重置地址，您仅需要广播一个重置命令即可。

我们知道 I^2C 广播无需地址，任何挂载到总线上的设备都能收到，所以不知道数据是给谁的，这样的话会非常容易出 BUG，所以大多数设备都选择关闭广播呼叫的功能。出于此原因，我们将简单的一帧命令改为一串长度 8 帧的命令以增加复杂度，这样就可以防止误触发，其数据为：

0xB8	0xD0	0xCE	0xAA	0xBF	0xC6	0xBC	0xBC

传输这段数据串时，您不需要按照“命令+数据”的语法操作，您仅需要将这一串数据通过广播的方式，发送给所有设备，当总线上所有的感为颜色传感器收到这一串数据后，将全部自动为您重置地址，并重启。其结构为：

S	0x00	A	0xB8	A	0xD0	A	0xCE	A	0xAA	A	...	A	0xBC	A	P
	↑		↑												
	地址		数据串												

即便使用 64bit 的命令符，也应该谨慎对待广播功能，因为 8 帧的方式仅能防御感为的设备不被骚扰，而不能保证挂在总线上的其他设备是安全的。请在重置时，将其他除感为颜色传感器之外的设备移除；或将感为颜色传感器挂载到未连接任何从设备的主控身上，单独重置该设备。

```
#include <Wire.h>
/*函数setup与loop是arduinoIDE标准框架，setup是程序的入口，只运行一次，loop则是循环运行，类似while(1)*/
// 广播重置地址所需的字符串
unsigned char reset_magic_number[8] = {
    0xB8,
    0xD0,
    0xCE,
    0xAA,
    0xBF,
    0xC6,
    0xBC,
    0xBC,
};
void setup(){
    Wire.begin();//Wire.h库中的初始化函数
    Wire.beginTransmission(0x00);//起始位函数
    Wire.write(reset_magic_number,8);
    Wire.endTransmission(1);//停止位函数。
    ping();
}
void loop(){
    //...
}
```

方法 2：扫描地址

I^2C 总线的从设备地址，一共有 7bit，排除 0x00 广播地址外，共有 127 个地址。您可以将颜色传感器单独挂载到总线上，将 127 个地址全部扫描一遍，如果哪个地址下有设备回应 (ACK)，那么这个地址就是该设备的地址。

```
#include <Wire.h>
#define GW_GRAY_ADDR 0x4f //跳线帽全插的情况下

void i2c_scan()
{
    byte error, address;
    int nDevices;

    Serial.println("扫描中...");

    nDevices = 0;
    for(address = 1; address < 127; address++ )
    {

        Wire.beginTransmission(address);
        error = Wire.endTransmission();

        if (error == 0)
        {
            Serial.print("找到I2C设备，地址为 0x");
            if (address<16)
                Serial.print("0");
            Serial.print(address,HEX);
            Serial.println("");

            nDevices++;
        } else if (error==4) {
            Serial.print("发生错误：  地址为 0x");
            if (address<16)
                Serial.print("0");
            Serial.println(address,HEX);
        }
    }
    if (nDevices == 0)
        Serial.println("无i2c设备\n");
    else
        Serial.println("完成\n");
}

/*函数setup与loop是arduinoIDE标准框架，setup是程序的入口，只运行一次，loop则是循环运行，类似while(1)*/
void setup(){
    Wire.begin();//Wire.h库中的初始化函数
    Serial.begin(115200);
}

void loop()
{
    i2c_scan(); //扫描i2c地址
    delay(5000); //每5秒扫描一次
}
```

4.10 ping 网络诊断工具

4.10.1 功能描述

ping（读音“乒”）网络诊断工具 (命令符：0xAA)，用于 I^2C 调试时使用，或是用于初始化中使用。实际上，该命令并不具备网络诊断功能，这个名称的意义在于帮助软件工程师快速理解其命令的用法，就是我们常说的：“能不能 ping 通?”

如果总线连接完整，地址正确，I^2C 程序无误，给颜色传感器发送 0xAA 命令，颜色传感器将返回数据 0x66 以证明其工作正常、总线完整。这个功能为只读模式。其语法结构只能为：

S	1001 111	0	A	0xAA	A	S	1001111	1	A	0x66	A	P
				↑						↑		
				命令						返回值		

4.10.2 命令特性

特性 1：回滚特性

ping 命令可以随时被插入，而不会影响当前命令的进程，此功能仅限于读数据时应用。

按照标准语法，当 ping 命令执行完毕后，为了不打断当前的进程，我们需要重新发送 ping 之前的命令，使之恢复并继续执行。例如，读取 RGB 数据时4.6在读取到 G 数据时，此时 ping 了一下。随后需要再发送 RGB 数据命令 (命令符：0xD0)，使之重新从头读取数字数据。

0xD0	R 值	G 值	B 值	R 值	0xAA	0x66	0xD0	R 值	G 值	B 值
↑	↑				↑		↑	↑		
命令	0xD0 命令下的数据				ping 语句		命令	0xD0 下的数据		

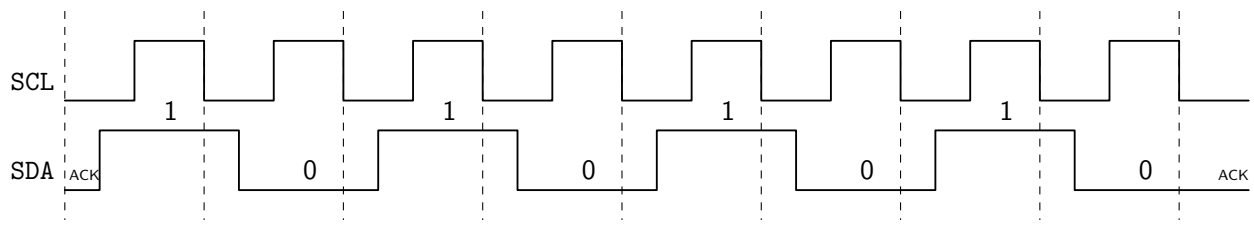
但您不需要这样做，ping 有命令回滚的特性。返回 0x66 后，您将不用重新发送命令 0xD0，颜色传感器将自动回滚为之前的命令 (0xD0)，并继续执行，而且颜色传感器会按照之前的序号继续读数据，而不是从头开始传输。

0xD0	R 值	G 值	B 值	R 值	0xAA	0x66	G 值	B 值	R 值	G 值
↑	↑				↑		↑			
命令	0xD0 命令下的数据				ping 语句		0xD0 下的数据			

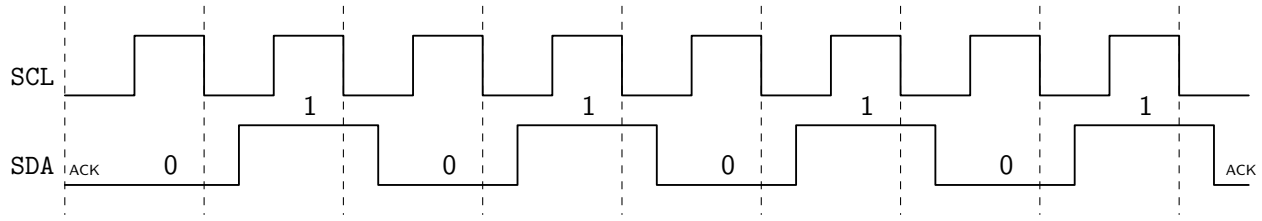
假如您需要在 ping 后重新从头读数据的话，您要重新发命令 0xD0。除此之外需要注意的是，如果您向传感器写数据，这种操作是不被允许的。

特性 2：信号校验特性

命令符：0xAA，它的二进制形式为，0b1010 1010。恰好为稳定的方波，方便示波器观看，以便调试总线。



返回值 0x66，它的二进制形式为，0b0101 0101。也为稳定的方波：



4.10.3 用法及程序实例

ping 功能常用在设备初始化中，用于同步您所用的主控与颜色传感器，由于颜色传感器开机到正常工作也需要一定的时间，如果您在未正常工作之前发送命令，可能会丢失通讯数据，或者通讯失败，所以需要用 ping 工具在用户主控程序初始化时，反复 ping，如果有正常返回值，则证明可以进行后续工作。

```
#include <Wire.h>
#define GW_GRAY_ADDR 0x4f //跳线帽全插的情况下
/*函数setup与loop是arduinoIDE标准框架，setup是程序的入口，只运行一次，loop则是循环运行，类似while(1)*/
void setup(){
    Wire.begin();//Wire.h库中的初始化函数
    Serial.begin(115200);
}
void loop(){
    char recv_value;//数据存放点
    //写命令
    Wire.beginTransmission(GW_GRAY_ADDR);//起始位函数
    Wire.write(0xAA);//写函数，发送PING命令0xAA
    Wire.endTransmission(0);//停止位函数，0为不产生停止位。
    // 读取数据
    Wire.requestFrom(GW_GRAY_ADDR/*地址*/,1/*读1个char*/,1/*停止位*/);//读数据的集成函数。
    recv_value = Wire.read();//取出Wire.requestFrom运行后的数据。
    if (recv_value == 0x66) {
        Serial.println("PING OK");
    } else {
        Serial.println("PING NOT OK");
    }
    delay(1000);
}
```

4.11 读取错误信息

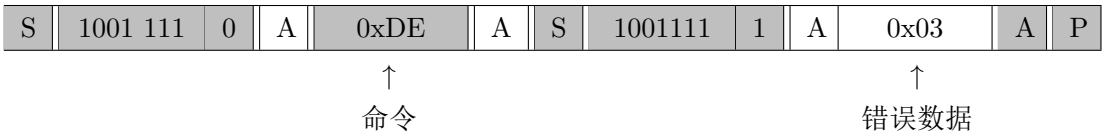
4.11.1 功能描述

读取错误信息（命令符：0xDE，即Data of Error），是将传感器使用过程中，出现的错误，将其汇总在 8bit 错误寄存器中。其中每 bit 代表一种错误，其详细构成为：

寄存器	7bit	6bit	5bit	4bit	3bit	2bit	1bit	0bit
使能位	保留			R 过曝	G 过曝	B 过曝	按键短路	对管过曝

位7:5	保留位，读出 0
位4	当传感器的 R 值超出 255 后，会提示 R 过曝。我们在使用过程中应该尽可能避免某一个值进入非线性区（过曝），从而能更好的还原色彩
位3	当传感器的 G 值超出 255 后，会提示 G 过曝。我们在使用过程中应该尽可能避免某一个值进入非线性区（过曝），从而能更好的还原色彩
位2	当传感器的 B 值超出 255 后，会提示 B 过曝。我们在使用过程中应该尽可能避免某一个值进入非线性区（过曝），从而能更好的还原色彩
位1	按键按钮按住时长超过 15 秒，系统判定为用户按键 pin 一直短接在 GND 上，此时置此位为 1。其目的是防止用户按键 pin 不小心搭接到 GND，而干扰传感器正常使用。
位0	对管接收光线量是有物理上限的，假如对管受日光或者环境光影响而达到了对管物理上限，则此位置 1。

该错误寄存器为只读寄存器，不可修改其中的内容。当有错误发生时，相应的位将被置位，当错误消失，相应位将被清零。其通讯结构仅为：



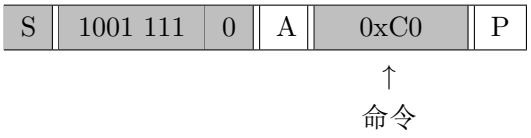
4.11.2 用法及程序实例

```
#include <Wire.h>
#define GW_GRAY_ADDR 0x4f //跳线帽全插的情况下
/*函数setup与loop是arduinoIDE标准框架，setup是程序的入口，只运行一次，loop则是循环运行，类似while(1)*/
void setup(){
    Wire.begin();//Wire.h库中的初始化函数
    ping();//ping函数，用于同步esp32与颜色传感器
}
void loop(){
    char recv_value;//数据存放点
    //写命令
    Wire.beginTransmission(GW_GRAY_ADDR);//起始位函数
    Wire.write(0xDE);//写函数，发送读取错误信息命令0xDE
    Wire.endTransmission(0);//停止位函数，0为不产生停止位。
    // 读取数据
    Wire.requestFrom(GW_GRAY_ADDR/*地址*/,1/*读1个char*/,1/*停止位*/);//读数据的集成函数。
    recv_value = Wire.read();//取出Wire.requestFrom运行后的数据。
}
```

4.12 设备软件重启

4.12.1 功能描述

设备软件重启命令 (命令符：0xC0, 即 Command top)，当您的主控程序出现 bug，或是更改设备软件地址后，需要对设备进行重启操作，这时候就可以发送命令 0xC0，即可完成设备重启。其通讯结构仅为：



4.12.2 用法及示例

当我们重启设备后，设备将首先初始化，这需要一些时间。如果您的主控在重启后，紧接着就发命令读写数据，那么大概率会漏掉信息，导致通讯错误。所以需要您使用 ping 函数，等待设备回应。

```
#include <Wire.h>
#define GW_GRAY_ADDR 0x4f //跳线帽全插的情况下
/*函数setup与loop是arduinoIDE标准框架，setup是程序的入口，只运行一次，loop则是循环运行，类似while(1)*/
void setup(){
    Wire.begin();//Wire.h库中的初始化函数
    ping();//ping函数，用于同步esp32与颜色传感器

    Wire.beginTransmission(GW_GRAY_ADDR);//起始位函数
    Wire.write(0xC0);//写函数，发送重启命令0xC0
    Wire.endTransmission(1);//停止位函数，0为不产生停止位。

    ping();//ping函数，用于同步esp32与颜色传感器
}
void loop(){
    // ...
}
```

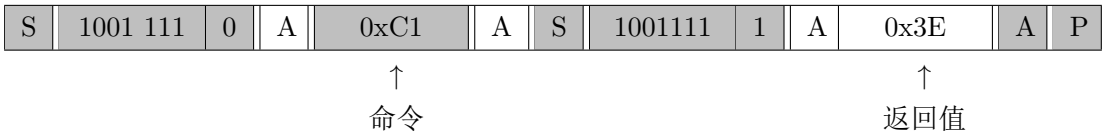
4.13 固件版本号查询

4.13.1 功能描述

固件版本号 (命令符:0xC1)，用于查询设备固件的型号，该型号由 8bit 组成，其中前 4bit 为一组 16 进制数，后 4bit 为 16 进制数，例如：版本 V3.14 的版本号为 0x3E

数据位	7bit	6bit	5bit	4bit	3bit	2bit	1bit	0bit
	0	0	1	1	1	1	1	0
版本号	高位版本：3				低位版本：14			

您在查询固件版本号时，请先发送 0xC1，颜色传感器将返回数据到您的主控上，其语法结构只能为：



4.13.2 用法及示例

```
#include <Wire.h>
#define GW_GRAY_ADDR 0x4f //跳线帽全插的情况下
/*函数setup与loop是arduinoIDE标准框架，setup是程序的入口，只运行一次，loop则是循环运行，类似while(1)*/
void setup(){
    Wire.begin();//Wire.h库中的初始化函数
    ping();//ping函数，用于同步esp32与颜色传感器
}
void loop(){
    char recv_value;//数据存放点
    //写命令
    Wire.beginTransmission(GW_GRAY_ADDR);//起始位函数
    Wire.write(0xC1);//写函数，发送读取固件版本命令0xC1
    Wire.endTransmission(0);//停止位函数，0为不产生停止位。
    // 读取数据
    Wire.requestFrom(GW_GRAY_ADDR/*地址*/,1/*读1个char*/,1/*停止位*/);//读数据的集成函数。
    recv_value = Wire.read();//取出Wire.requestFrom运行后的数据。
}
```

4.14 命令符

读取 RGB 数据	0xD0	读
读取 HSL 数据	0xD1	读
软件地址配置	0xAD 0XAD	写
广播重置地址	字符串:0xB8 0xD0 0xCE 0xAA 0xBF 0xC6 0xBC 0xBC	写
ping 网络诊断工具	0xAA	读
读取错误信息	0xDE	读
设备软件重启	0xC0	无
固件版本号查询	0xC1	读