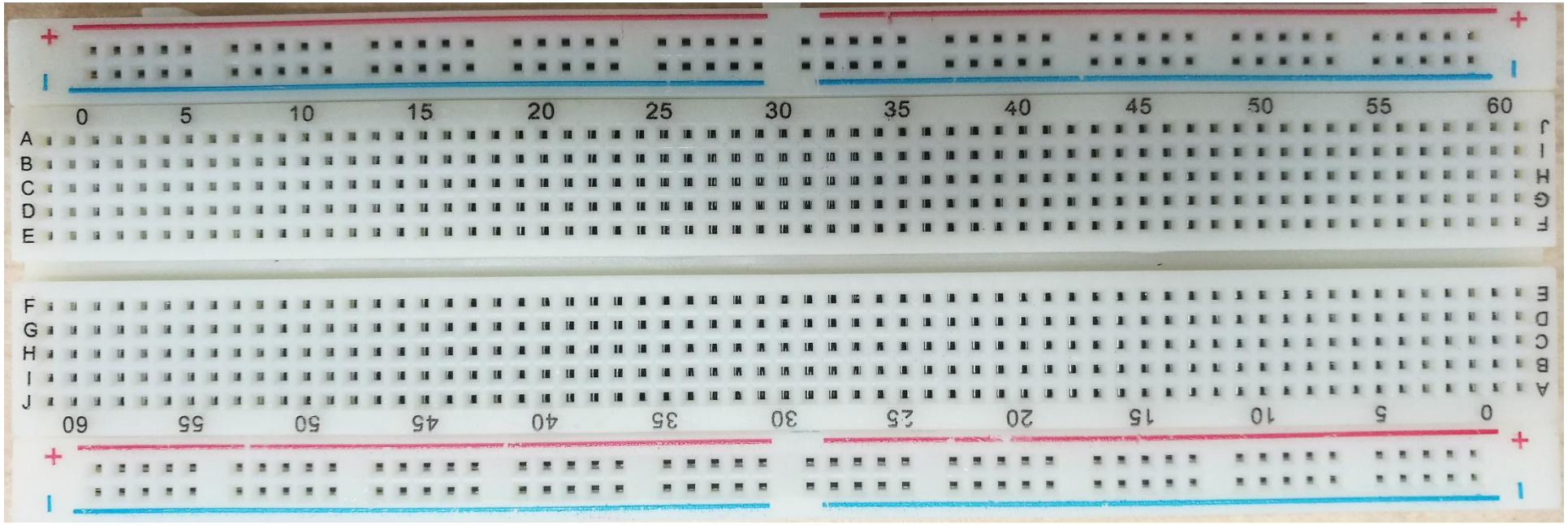
4 树莓派 linux 补充材料补充材料

Contents

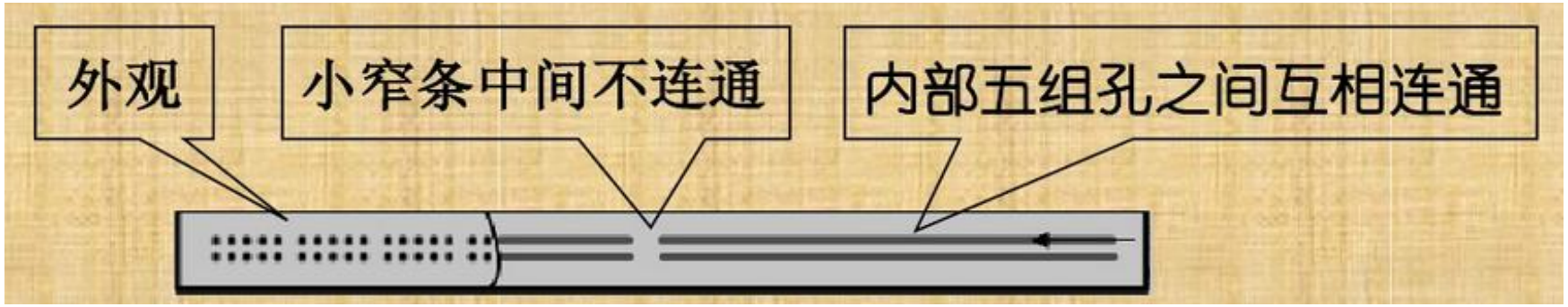
补充知识：面包板的使用

- 面包板正面
- 面包板的外观和内部结构如图所示，常见的面包板分上、中、下三部分，上面和下面部分一般是由一行或两行的插孔构成的窄条，中间部分是由中间一条隔离凹槽和上下各5行的插孔构成的宽条。



面包板的使用

- 窄条的结构



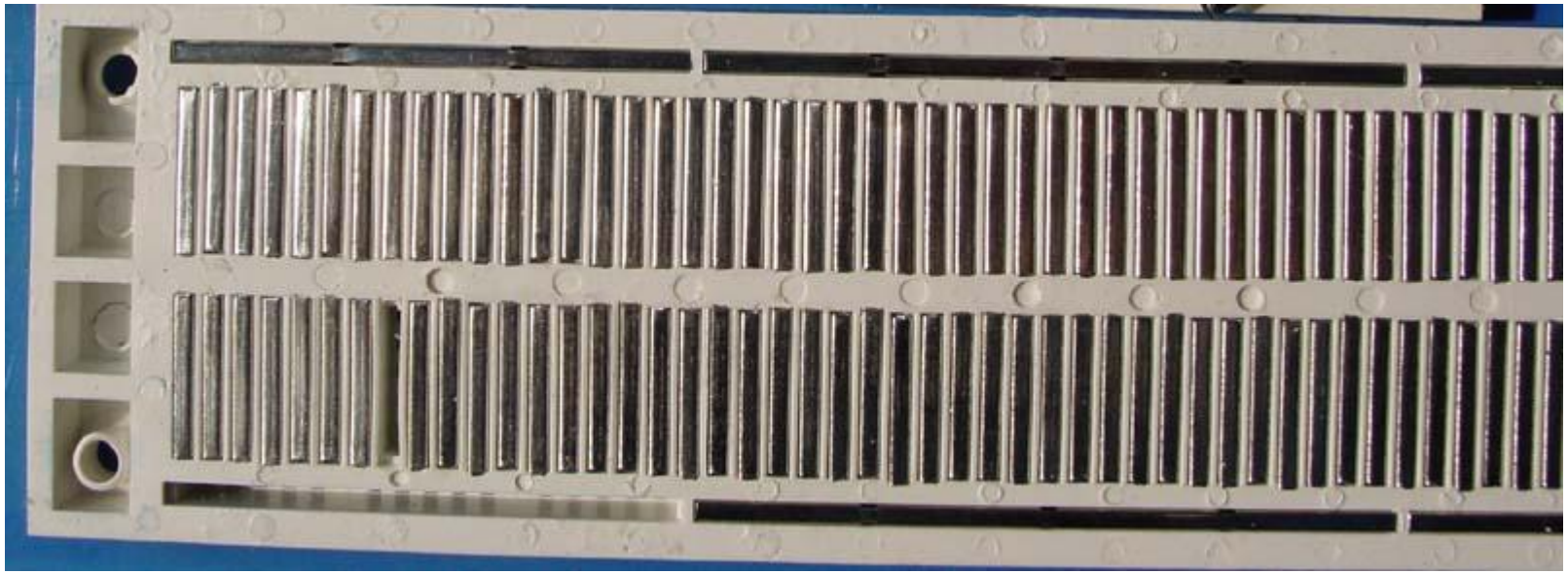
窄条上下两行之间电气不连通。每5 个插孔为一组（通常称为“孤岛”），通常的面包板上有10 组。这10 组“孤岛”一般有3种内部连通结构：① 左边5 组内部电气连通，右边5 组内部电气连通，但左右两边之间不连通，这种结构通常称为5-5 结构。② 左边3 组内部电气连通，中间4 组内部电气连通，右边3 组内部电气连通，但左边3 组、中间4 组以及右边3 组之间是不连通的，这种结构通常称为3-4-3 结构。③ 还有一种结构是10组“孤岛”都连通，这种结构最简单。

面包板的使用

- 面包板的背面

插槽和插孔

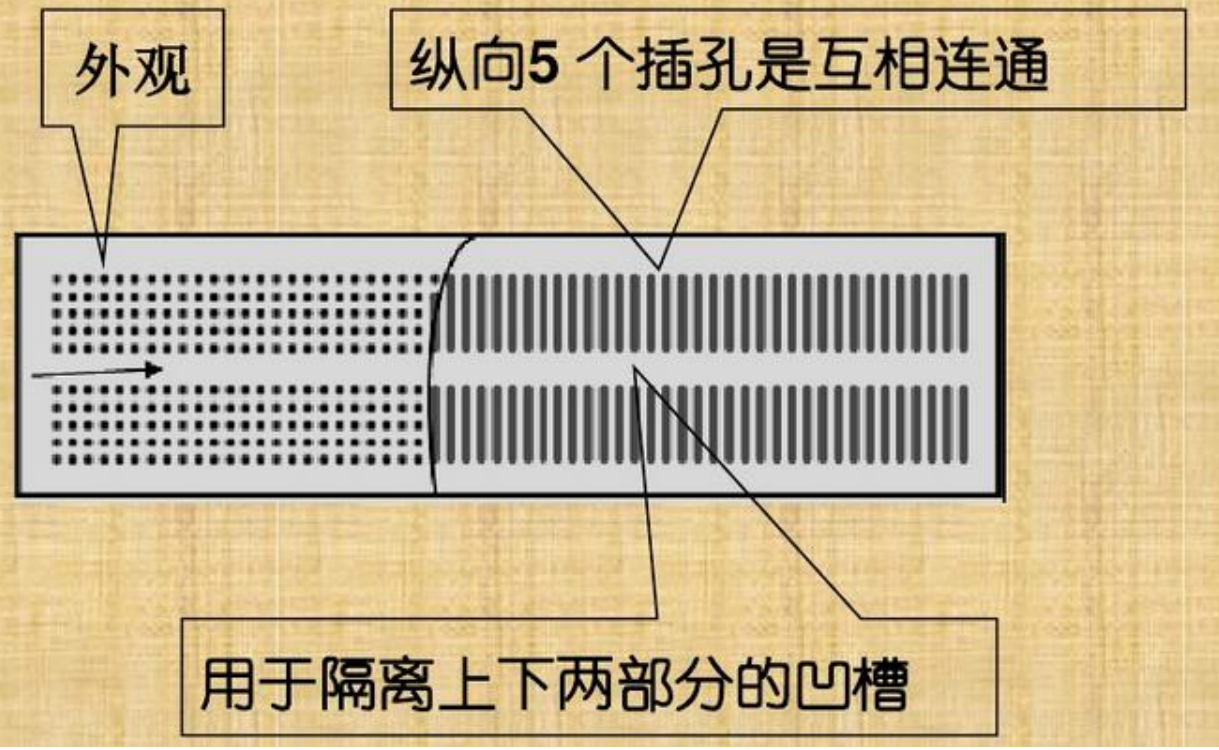
每一条金属片插入一个塑料槽，在同一个槽的插孔相通，不同槽的插孔不通。



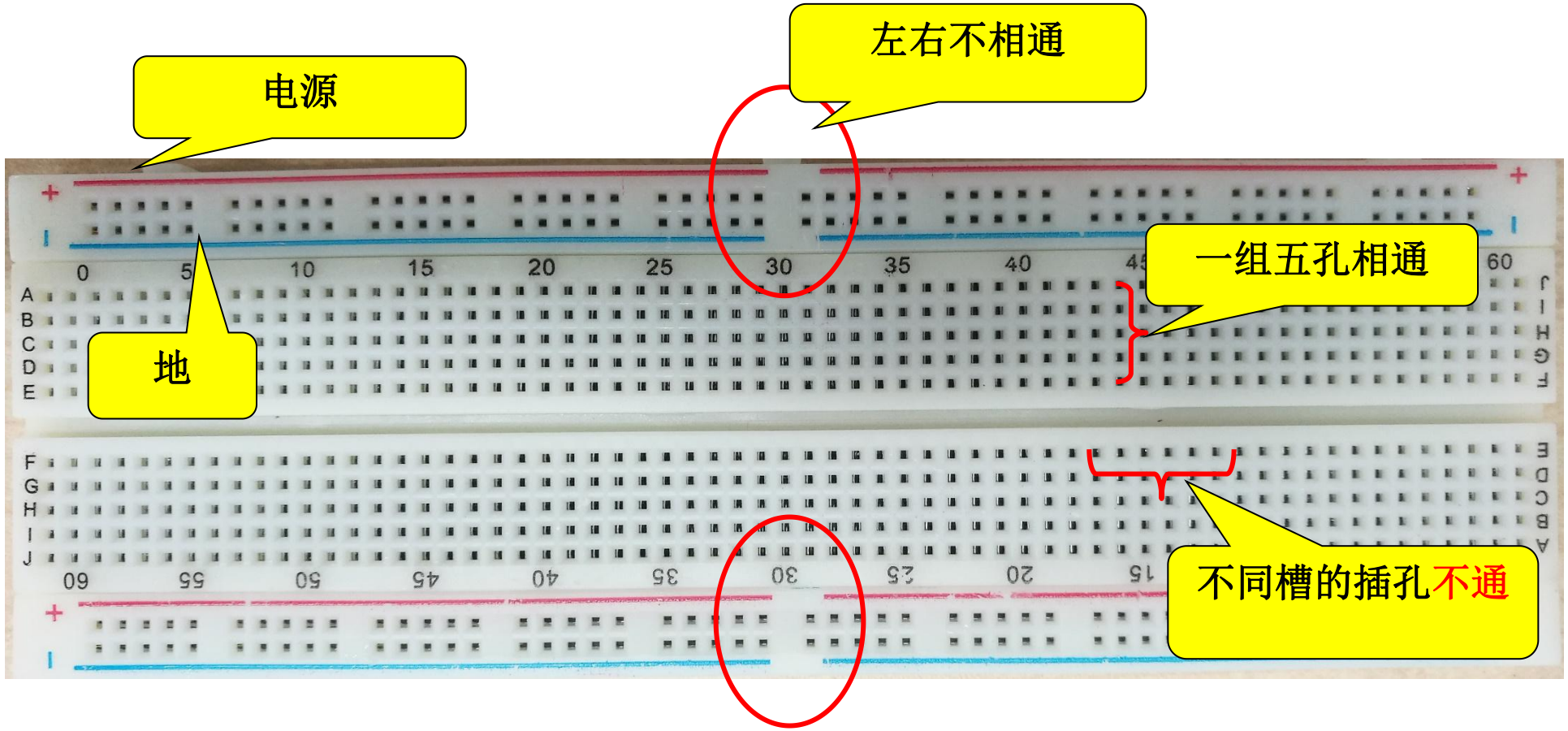
面包板的使用

质文档

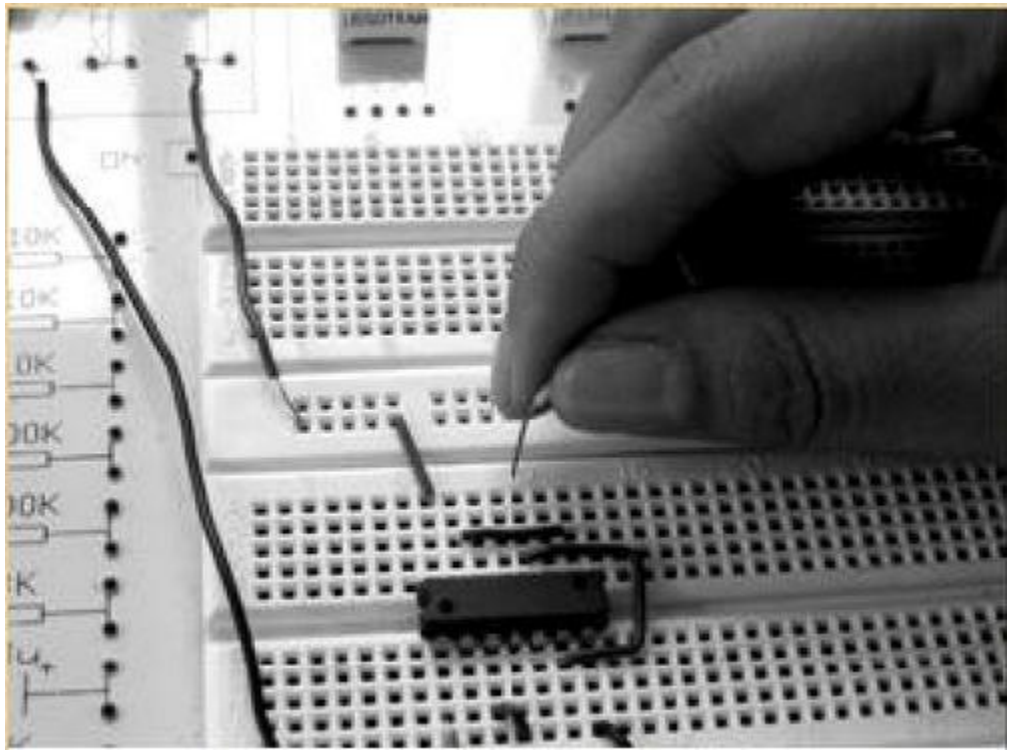
中间部分宽条是由中间一条隔离凹槽和上下各5行的插孔构成。在同一列中的5 个插孔是互相连通的，列和列之间以及凹槽上下部分则是不连通的。外观及结构如下图：

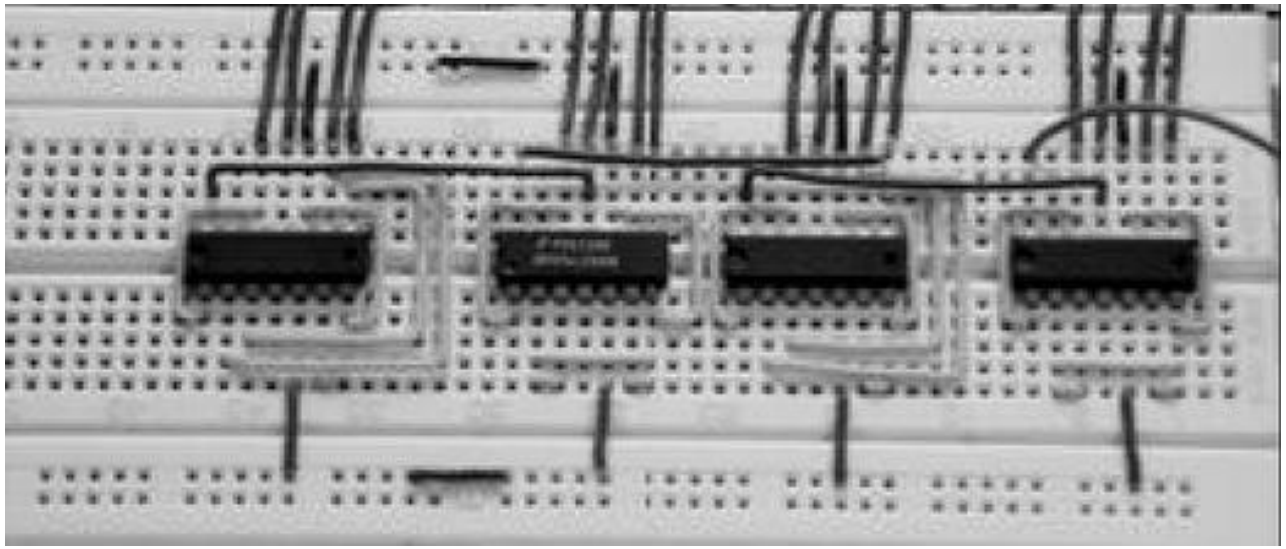


面包板的使用



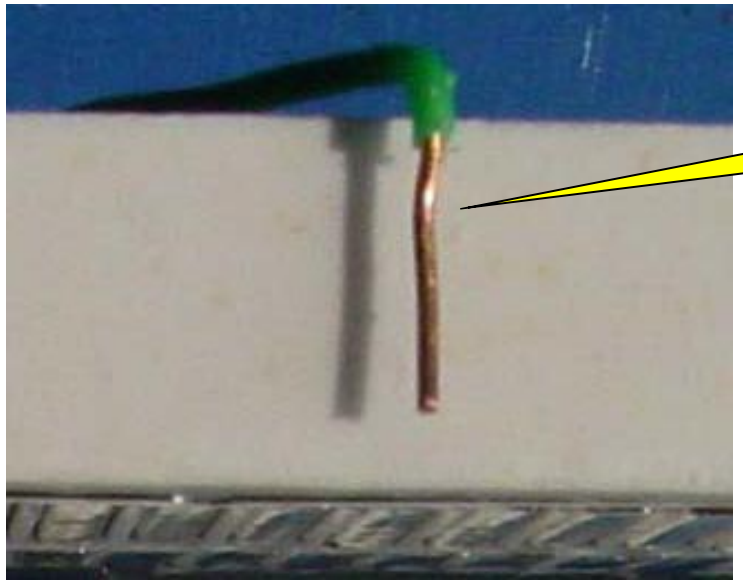
面包板的使用



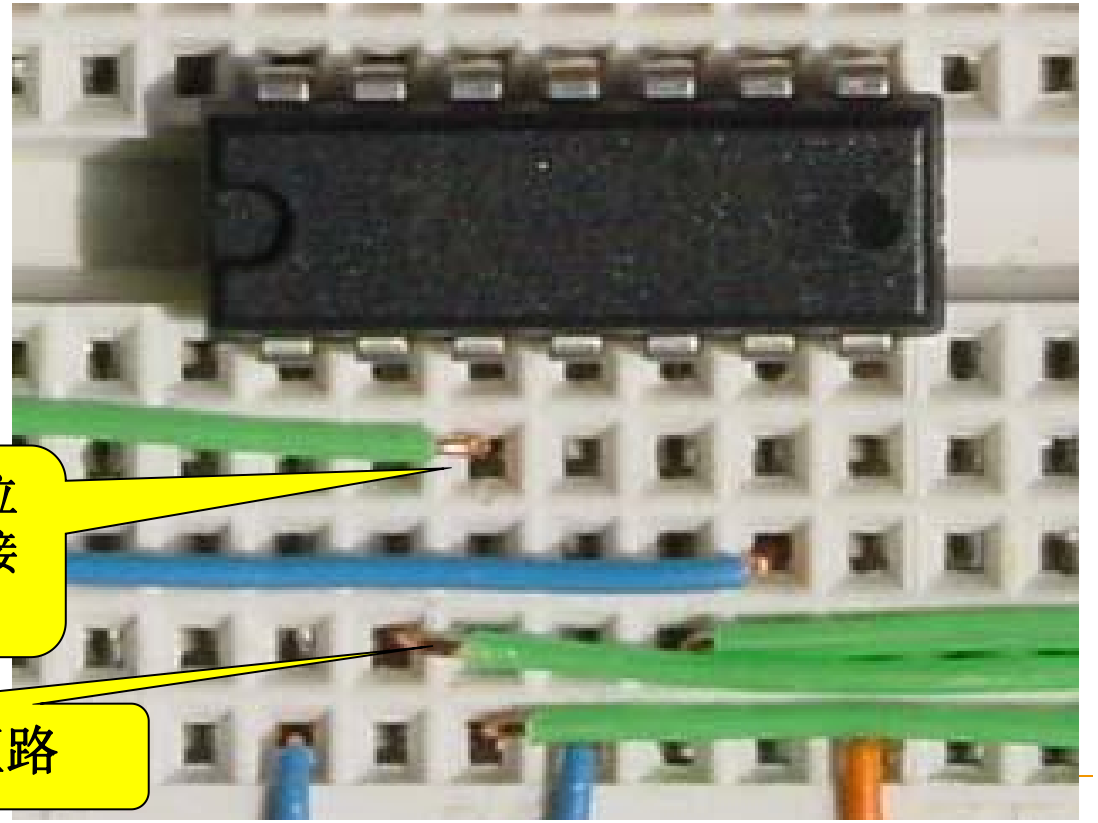


面包板的使用

- 导线的剥头和插法



导线剥头的长度比面包板厚度略短，转弯处留1mm绝缘层，绝缘层太长会造成绝缘层插入导电孔而不导通。铜线太短也会因接触不良而不导通。

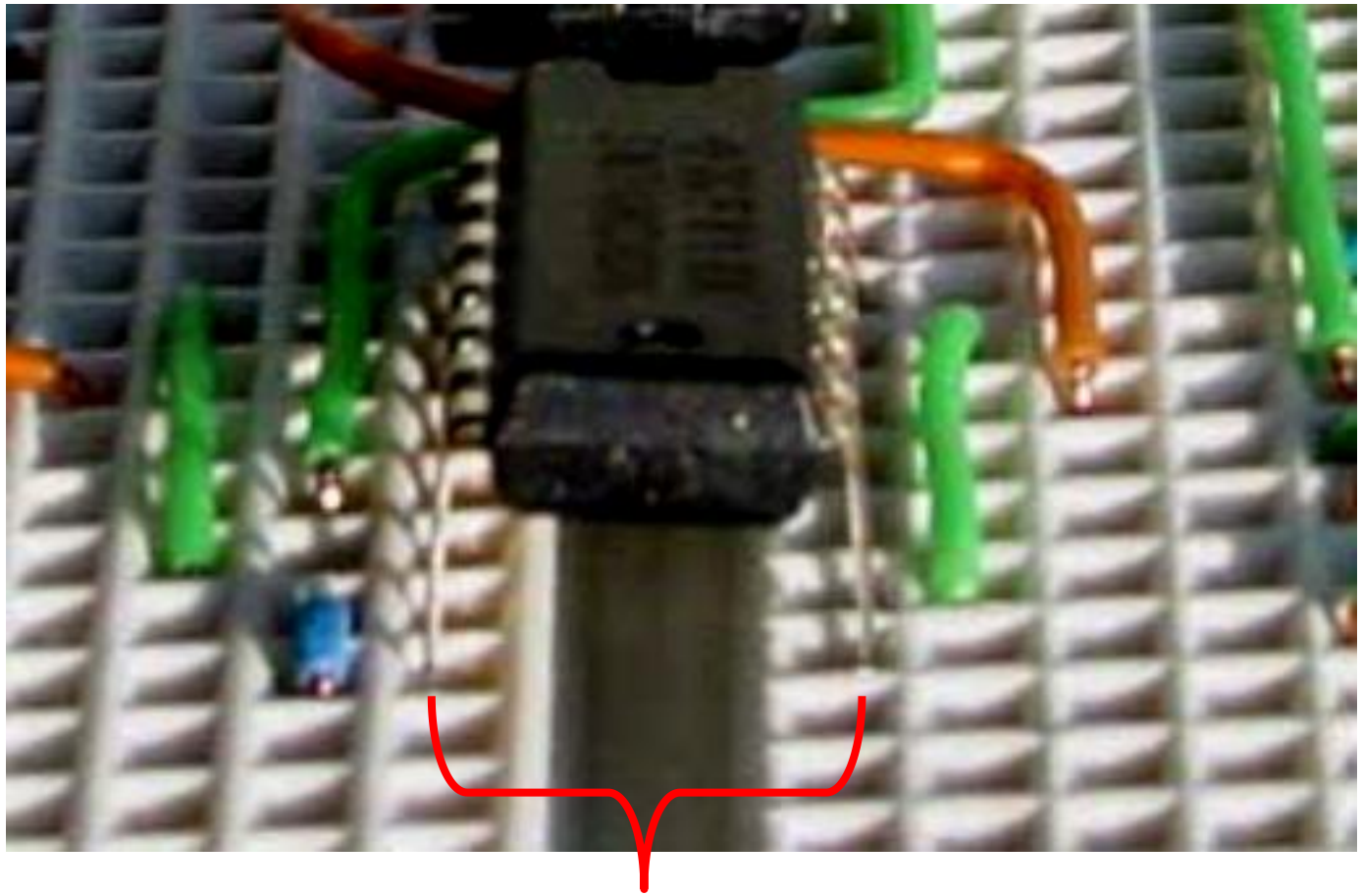


铜线必须插入金属孔中，特别在金属孔位置靠边时，容易插到边上空白处，引起接触不良，用万用表也难以测量。

铜线太长容易引起短路

面包板的使用

- 集成块的插法

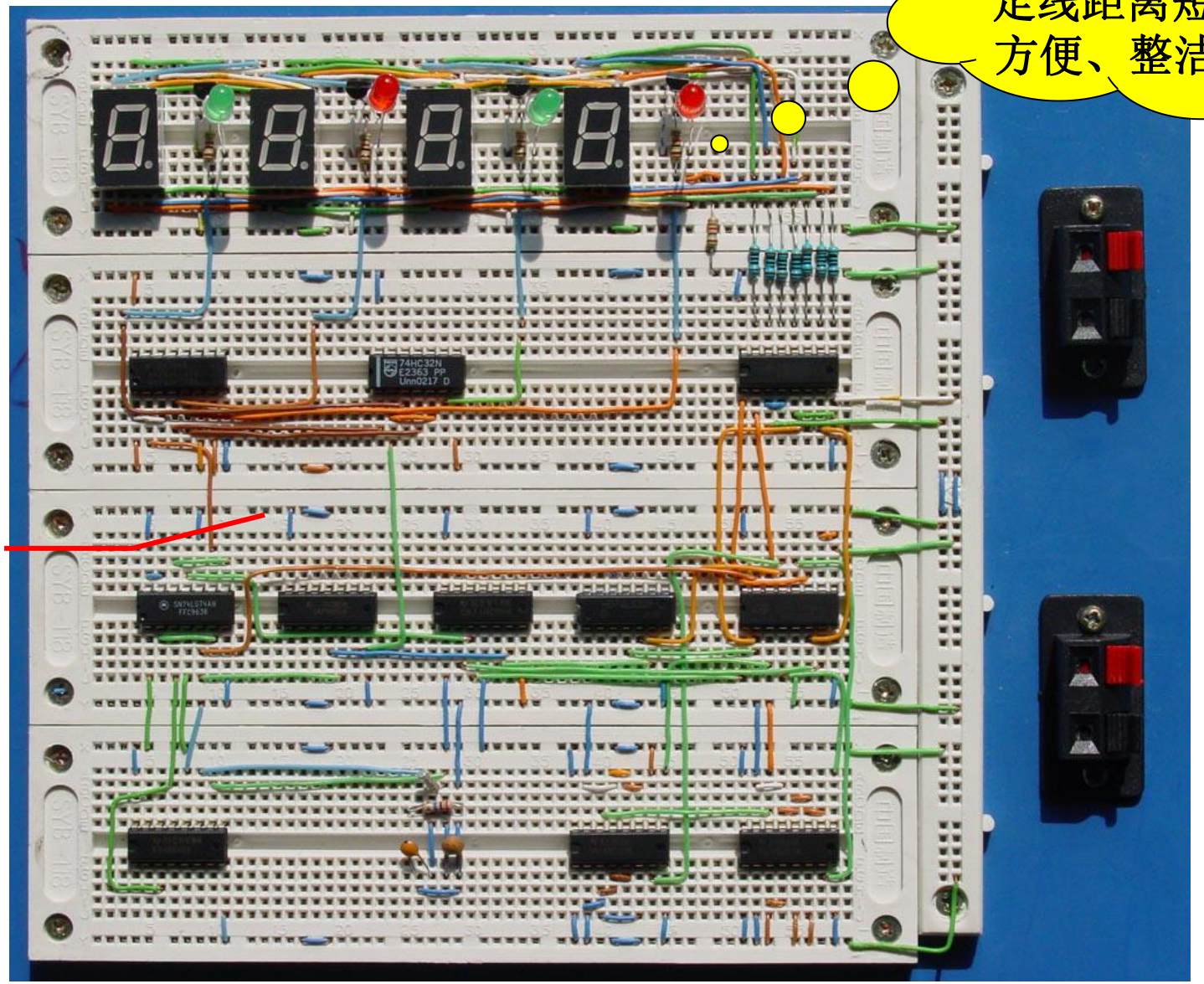


由于集成块引脚间与距离与插孔位置有偏差，必须预先调整好位置，小心插入金属孔中，不然会引起接触不良，而且会使铜片位置偏移，插导线时容易插偏。

此原因引起的故障占总故障的**60%**以上。

面包板的使用

- 接线样板

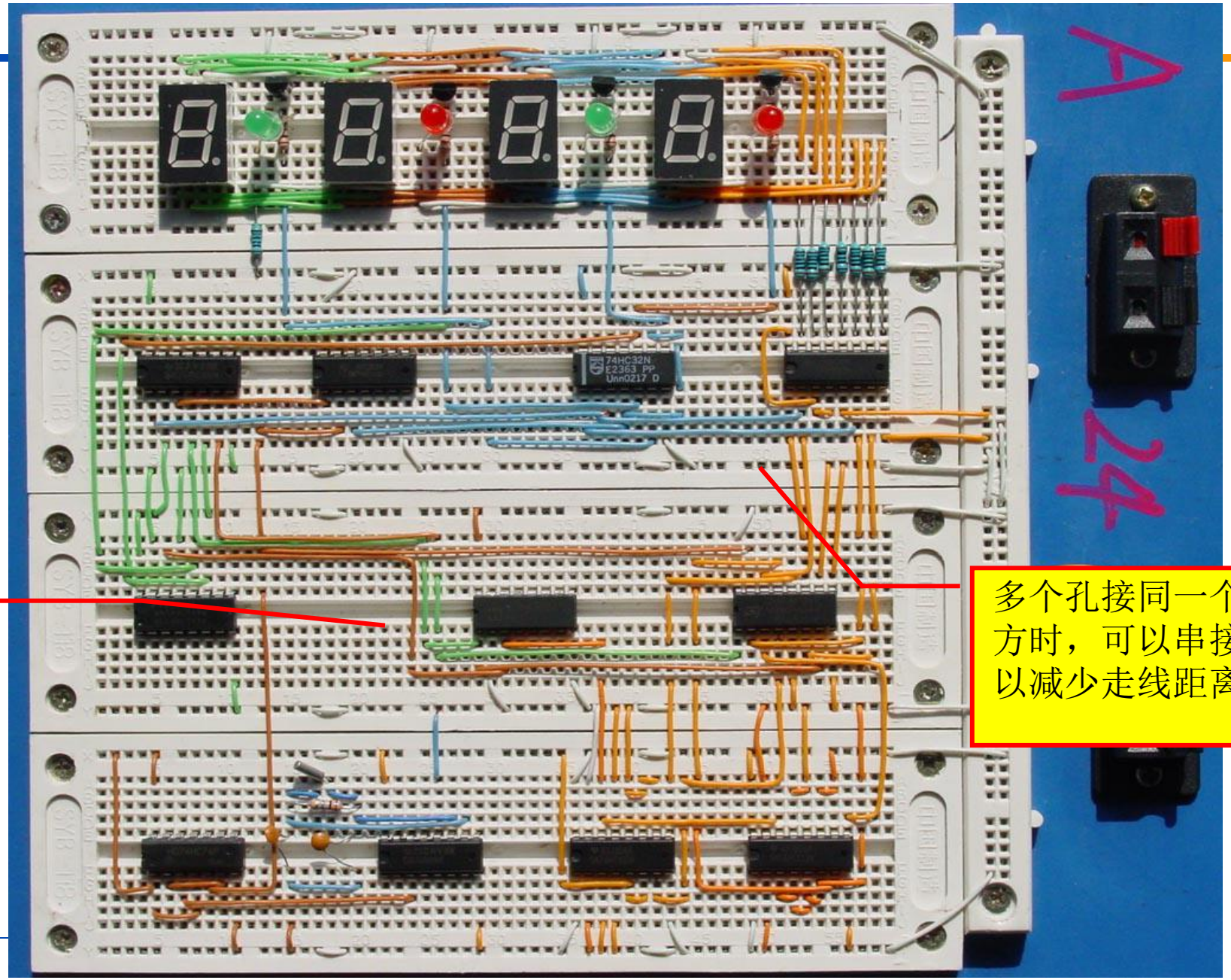


整块板上的元器件的布局要合理，使走线距离短、接线方便、整洁美观。

导线量好长度后，剥好线头、根据走线位置折好后插入面包板。走线方向为“横平、竖直”

面包板的使用

- 接线样板

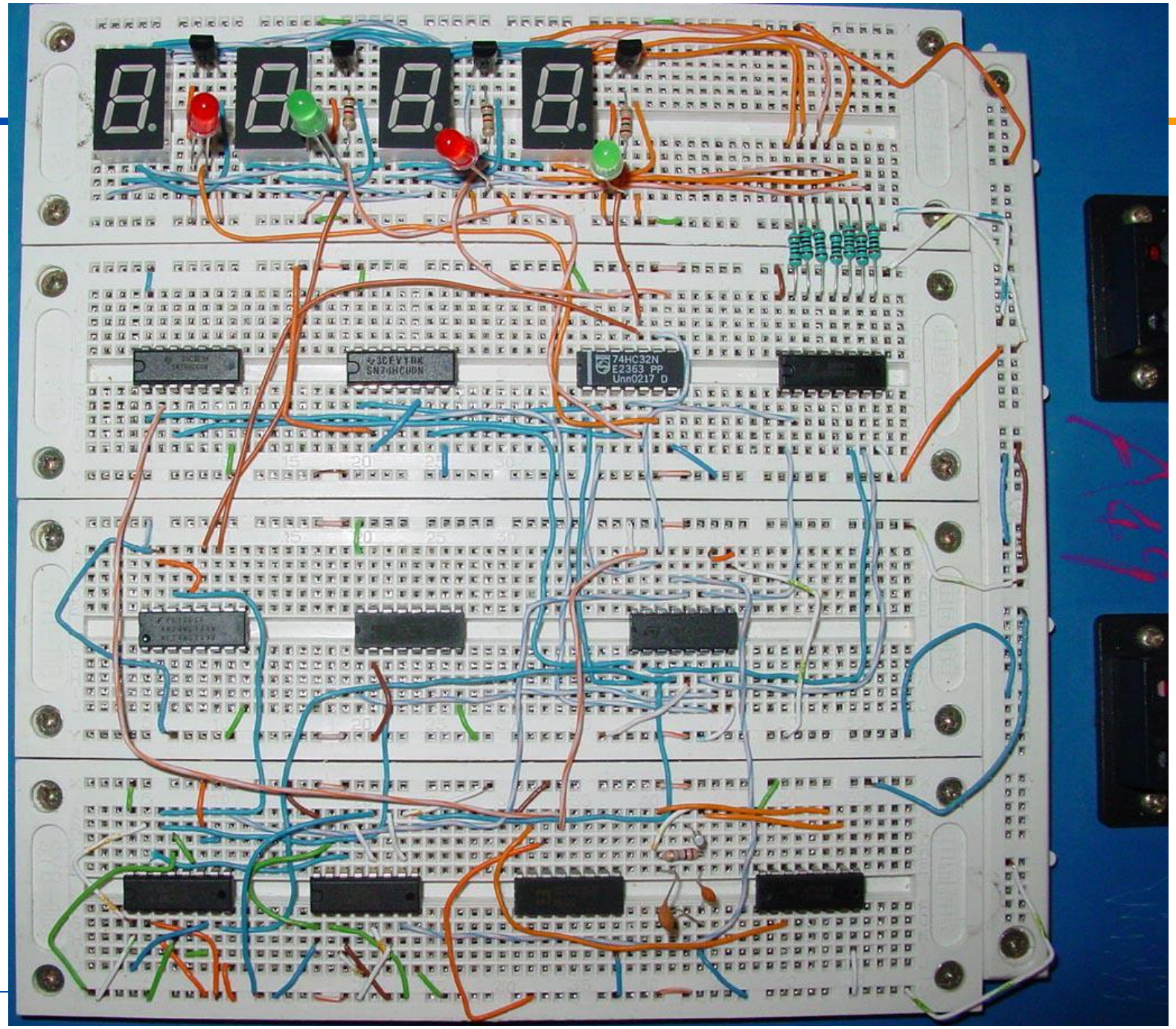


一根导线可以直通的地方尽量只用一根线，用多根导线转接费事又容易出错。

多个孔接同一个地方时，可以串接，以减少走线距离。

面包板的使用

- 接线样板（**反面案例**）



Contents

补充：Linux下用文件I/O的方式操作GPIO

- Linux 系统定义了一切皆文件的原则，甚至于硬件：磁盘，软盘，等……，这样的好处是，在不同的平台和硬件上都能形成统一的调用方式。
- 这里，我们从树莓派，探讨下GPIO的操作原理（非内核原理）：
  - 通过 sysfs 方式控制 GPIO
  - 接触过 Linux 的，应该都了解 *echo "0" > test.txt* 是将一个字符串 0 输入到当前目录的 test.txt 文件中，那么下面所有的操作，均基于这个操作形式。

补充：Linux下用文件I/O的方式操作GPIO

- 自 Linux 2.6.21 起，GPIO 作为一个基础组件包含到Linux 内，我们可以根据 GPIO 规范直接使用。
- 首先，进入 /sys/class/gpio 目录下, 我们可以看到以下文件：

```
# cd /sys/class/gpio
# ls -l
总用量 0
-rwxrwx--- 1 root gpio 4096 9月   8 01:37 export
lrwxrwxrwx 1 root gpio    0 8月   9 01:16 gpiochip0 -> ../../devices/platform/soc/3f200000.gpio/gpio/gpiochip0
-rwxrwx--- 1 root gpio 4096 9月   8 01:47 unexport
```

补充：Linux下用文件I/O的方式操作GPIO

- /sys/class/gpio目录功能具体如下：
  - gpio操作是通过 /sys/ 文件接口操作 IO 端口 GPIO 到文件系统的映射。
  - 控制 GPIO 的目录位于 /sys/class/gpio。
  - /sys/class/gpio/export 文件用于通知系统需要导出控制的 GPIO 引脚编号。
  - /sys/class/gpio/unexport 用于通知系统取消导出。
  - /sys/class/gpio/gpiochipX 目录保存系统中 GPIO 寄存器的信息，包括每个寄存器控制引脚的起始编号 base，寄存器名称，引脚总数

补充：Linux下用文件I/O的方式操作GPIO

- **导出引脚**
  - 向文件 /sys/class/gpio/export 写入引脚编号，即可激活引脚。例如：

 # echo 18 > /sys/class/gpio/export

  - 命令执行成功后，目录下，便会出现 gpio18 文件夹，如果没有出现，则表示引脚不可导出。

补充：Linux下用文件I/O的方式操作GPIO

控制方向

- 引脚导出成功后，即可通过写入 /sys/class/gpio/gpio18/direction 控制引脚【输入】 或【输出】。

```
# cd gpio18
# echo "out" > direction
```

- 命令无报错，即为操作成功；可输入值有以下几种：
  - in　　引脚输入信号
  - out　输出控制到引脚
  - high/low　　复合操作，设置方向为输出，同时将value设置为相应的1/0

补充：Linux下用文件I/O的方式操作GPIO

- **输入输出值**
  - 对GPIO输入输出的值，通过 gpioX 下的 value 文件控制
  - 当为输入时，# echo /sys/class/gpio/gpio18/value 读入value中的值;
  - 当为输出时，# echo 1 > /sys/class/gpio/gpio18/value 通过value输出值;
- **取消引脚导出**
  - 当控制完成时，需要释放掉端口的控制，此时如导出引脚时一样，将引脚编号输出到 /sys/class/gpio/unexport 即可。
  - # echo 18 > /sys/class/gpio/unexport

Contents

GPIO Without Root

- Avoiding root privileges will make the system more secure and more stable.
- In modern versions of Linux, devices attached to your system are controlled by udev.
- Udev is a rule-based system that can run scripts when devices are plugged in.
- If you have developed apps for Android under Linux, you may have created a udev rule to modify permissions when your cell phone is plugged into the computer.
- If you have developed with Arduino on Linux, you have probably added yourself to the dialout group to get access to serial over USB.

补充：Udev

- 在对待设备管理这块，Linux改变了几次策略，但基本都是通过设备文件管理的。
- Linux早期，设备文件仅仅是一些带有适当的属性集的普通文件，它由mknod命令创建，文件存放在/dev目录下。
- 后来，采用了devfs（设备文件系统），一个基于内核的动态设备文件系统，他首次出现在2.3.46内核中。devfs在设备初始化时创建设备文件，设备驱动程序可以指定设备号、所有者、用户空间等信息。

补充：Udev

- 在此之前的设备文件管理方法(静态文件和devfs)有几个缺点：
  - 不确定的设备映射。特别是那些动态设备，比如USB设备，设备文件到实际设备的映射并不可靠和确定。举一个例子：如果你有两个USB打印机。一个可能称 为/dev/usb/lp0,另外一个便是/dev/usb/lp1。但是到底哪个是哪个并不清楚，lp0,lp1和实际的设备没有一一对应的关系，可能因为发现设备的顺序，打印机本身关闭等原因而导致这种映射并不确定。理想的方式应该是：两个打印机应该采用基于他们的序列号或者其他标识信息的唯一设备文件来映射。但是静态文件和devfs都无法做到这点。
  - 没有足够的主/辅设备号。我们知道，每一个设备文件是有两个8位的数字：一个是主设备号，另外一个是辅设备号来分配的。这两个8位的数字加上设备类型(块设备或者字符设备)来唯一标识一个设备。不幸的是，关联这些身边的的数字并不足够。

补充：Udev

- 在此之前的设备文件管理方法(静态文件和devfs)有几个缺点：
  - /dev目录下文件太多。一个系统采用静态设备文件关联的方式，那么这个目录下的文件必然是足够多。而同时你又不知道在你的系统上到底有哪些设备文件是激活的。
  - 命名不够灵活。尽管devfs解决了以前的一些问题，但是它自身又带来了一些问题。其中一个就是命名不够灵活；你别想非常简单的就能修改设备文件的名字。缺省的devfs命令机制本身也很奇怪，他需要修改大量的配置文件和程序。
  - 内核内存使用，devfs特有的另外一个问题是，作为内核驱动模块，devfs需要消耗大量的内存，特别当系统上有大量的设备时(比如上面我们提到的系统一个上有好几千磁盘时)

补充：Udev

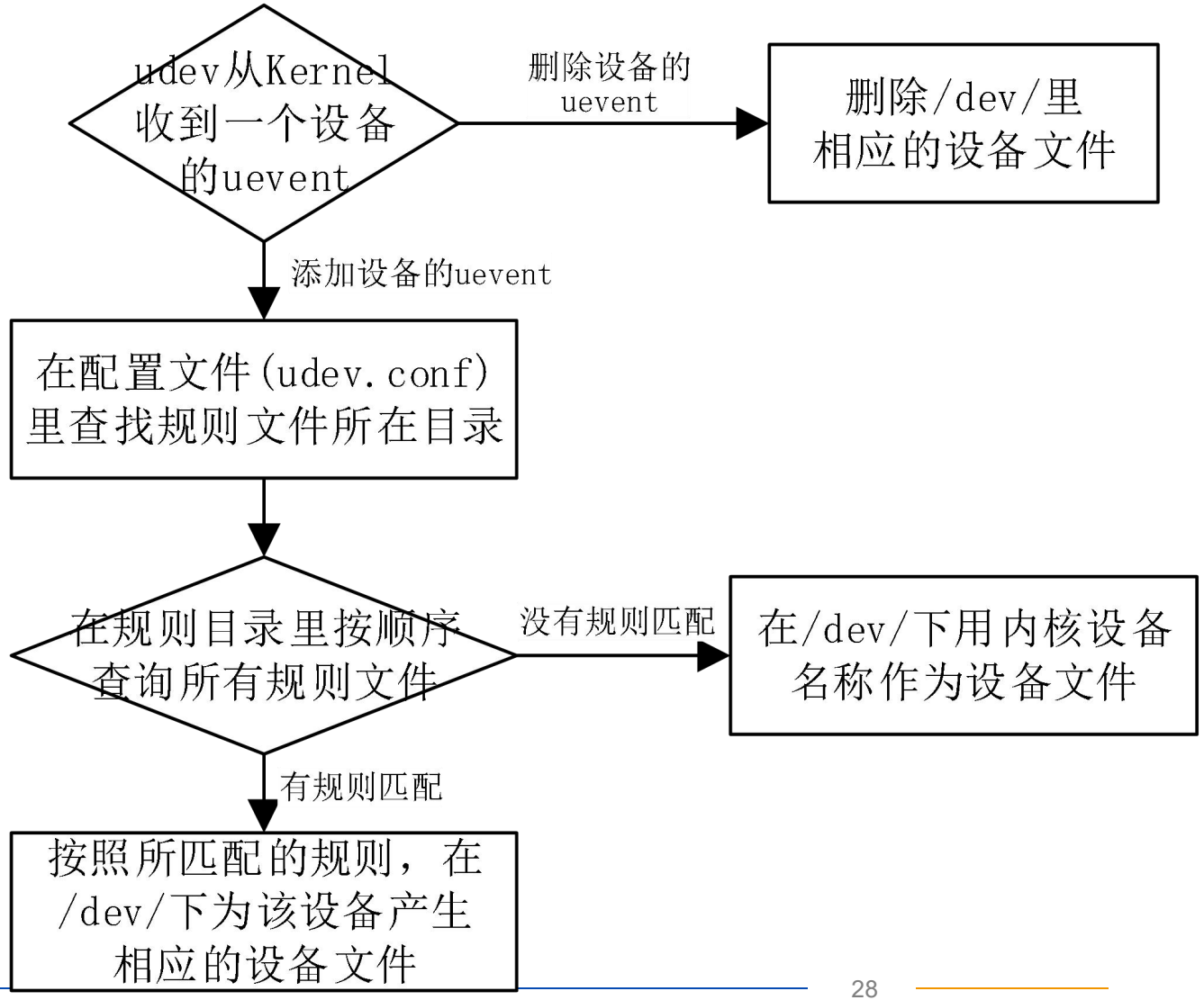
- 而自2.6 内核开始，引入了sysfs 文件系统。sysfs 把连接在系统上的设备和总线组织成一个分级的文件，并提供给用户空间存取使用。udev（热插拔设备） 运行在用户模式，而非内核中。udev 的初始化脚本在系统启动时创建设备节点，并且当插入新设备-->加入驱动模块-->在sysfs上注册新的数据后，udev会创新新的设备节点。
- udev 是一个工作在用户空间的工具，它能根据系统中硬件设备的状态动态的更新设备文件，包括设备文件的创建，删除，权限等。这些文件通常都定义在/dev目录下，但也可以在配置文件中指定。udev 必须内核中的sysfs和tmpfs支持，sysfs 为udev 提供设备入口和uevent 通道，tmpfs 为udev 设备文件提供存放空间。

补充：Udev

- udev的目标是想解决静态文件和devfs提到的这些问题。
  - 通采用用户空间(user-space)工具来管理/dev/目录树
  - 和文件系统分开。
  - 知道如何改变缺省配置能让你知道如何定制自己的系统，比如创建设备字符连接，改变设备文件属组，权限等。

补充：Udev

- udev 是通过对内核产生的设备文件修改，或增加别名的方式来达到自定义设备文件的目的。但是，udev是用户模式程序，它不会更改内核行为。也就是说，内核仍然会创建sda，sdb等设备文件，而udev可根据设备的唯一信息来区分不同的设备，并产生新的设备文件（或链接）。而在用户的应用中，只要使用新产生的设备文件即可。



补充：Udev

- 目前很多的Linux分发版本采纳了udev的方式，因为它在Linux设备访问，特别是那些对设备有极端需求的站点(比如需要控制上千个硬盘)和热插拔设备(比如USB摄像头和MP3播放器)上解决了几个问题。
- 下面我们来看看如何管理udev设备。

  实际上，对于那些为磁盘，终端设备等准备的标准配置文件而言，你不需要修改什么。
- 但是，你需要了解udev如何配置使用新的或者外来设备，如果不修改，这些设备可能无法访问，或者说Linux可能会采用不恰当的名字，属组或权限来创建这些设备文件。
- 你可能也想知道如何修改RS - 232串口，音频设备等文件的属组或者权限。这点在实际的Linux实施中是会遇到的

udev配置文件

- 主要的udev配置文件是/etc/udev/udev.conf。这个文件通常很短，它可能只是包含几行#开头的注释，然后有几行选项：
  - udev_root=“/dev/”

    udev_rules=“/etc/udev/rules.d/”

    udev_log=“err“
- 上面的第二行非常重要，因为它表示udev规则存储的目录，这个目录存储的是以.rules结束的文件。
- 每一个文件处理一系列规则来帮助udev分配名字给设备文件以保证能被内核识别。

GPIO Without Root

$ sudoedit /etc/udev/rules.d/88-gpio-without-root.rules

Example：88-gpio-without-root.rules

SUBSYSTEM=="gpio", RUN+="/bin/chown -R root.dialout /sys/class/gpio/“

SUBSYSTEM=="gpio", RUN+="/bin/chown -R root.dialout /sys/devices/virtual/gpio/"

SUBSYSTEM=="gpio", RUN+="/bin/chmod g+s /sys/class/gpio/“

SUBSYSTEM=="gpio", RUN+="/bin/chmod g+s /sys/devices/virtual/gpio/"

SUBSYSTEM=="gpio", RUN+="/bin/chmod -R ug+rw /sys/class/gpio/“

SUBSYSTEM=="gpio", RUN+="/bin/chmod -R ug+rw /sys/devices/virtual/gpio/"

udev配置文件

- 规则文件里的规则有一系列的键/值对组成，键/值对之间用逗号(,)分割。
- 每一个键或者是用户匹配键（==，!=），或者是一个赋值键（=，+=，:=）。
  - 匹配键确定规则是否被应用，而赋值键表示分配某值给该键。
  - 这些值将影响udev创建的设备文件。赋值键可以处理一个多值列表。
  - 匹配键和赋值键操作符解释见下表：

操作符	匹配或赋值	解释
==	匹配	相等比较
!=	匹配	不等比较
=	赋值	分配一个特定的值给该键，他可以覆盖之前的赋值。
+=	赋值	追加特定的值给已经存在的键
:=	赋值	分配一个特定的值给该键，后面的规则不可能覆盖它。

GPIO Without Root

- Sets the owner of the two directories to be root, and the group to be dialout.

  SUBSYSTEM=="gpio", RUN+="/bin/chown -R root.dialout /sys/class/gpio/“

  SUBSYSTEM=="gpio", RUN+="/bin/chown -R root.dialout /sys/devices/virtual/gpio/"

- Sets the sticky bit flag on these two directories.

  SUBSYSTEM=="gpio", RUN+="/bin/chmod g+s /sys/class/gpio/“

  SUBSYSTEM=="gpio", RUN+="/bin/chmod g+s /sys/devices/virtual/gpio/"

- Configures the permissions on the directories to give members of the dialout group read and write permission.

  SUBSYSTEM=="gpio", RUN+="/bin/chmod -R ug+rw /sys/class/gpio/“

  SUBSYSTEM=="gpio", RUN+="/bin/chmod -R ug+rw /sys/devices/virtual/gpio/“

GPIO Without Root

- 创建好了新的规则文件之后，使用如下命令重启udev服务程序，触发并使用新的规则：
  - $ sudo service udev restart
  - $ sudo udevadm trigger --subsystem-match=gpip
- 在较新的Linux版本中，所有的服务程序（服务器）都是由脚本控制的。
- 所以，可以简单的使用命令 $ sudo service apache2 reload 告知Apache web服务器重新读取配置文件，你也可以使用命令 $ sudo service ssh restart 让SSH服务器重新启动（先关机再启动）

GPIO Without Root

- 下一步，检查所有关系是否正确：
  - $ ls -lR /sys/class/gpio/
  - ls命令输出的信息会多次提及“dialout”用户组。
  - 参数 -l 表示累出文件的详细信息（包括拥有者、用户组、权限）
  - 参数 -R表示同时也列出所有子目录的详细信息。

GPIO Without Root

- 至此，我们做完了所有使用非root权限操作设备GPIO的准备，下面就尝试一下：

```
$ echo “27” > /sys/class/gpio/unexport          #先取消导出，下一句就不报错
$ echo “27” > /sys/class/gpio/export            #导出GPIO 27
$ echo “out” > /sys/class/gpio/gpio27/direction     #设置该I/0口为输出
$ echo “1” > /sys/class/gpio/gpio27/value           #设置该I/O口为高电平，灯亮
$ echo “0” > /sys/class/gpio/gpio27/value           #设置该I/O口为低电平，灯灭
```

- 以上命令提示符是“$”，表示当前为普通用户
- 这样几乎所有的编程语言都能使用GPIO了。