



e-Paper ESP8266 Driver Board

用户手册

产品简介

- 本产品 是电子墨水屏网络驱动板。通过 WIFI 网络，用户可以使用 PC 或者是手机对微雪电子墨水屏进行图片信息更新
- 这款驱动板。体型小，板载 ESP8266。在驱动板的背面引出 ESP8266 的引脚，方便用户使用。支持 Arduino 开发。
- 上位机程序提供了图片的颜色校正功能。用户可以通过浏览器打开设置网页，将图片信息下载刷新到对应的墨水屏

产品特性

- 板载 ESP8266，支持 Arduino 开发
- 提供 HTML 上位机程序
 - ✧ 支持 Floyd-Steinberg 抖动算法，以获得更多的颜色组合，对原始图片进行更好的阴影渲染
 - ✧ 支持多种常用图片格式（BMP、JPEG、GIF 和 PNG 等）
 - ✧ 方便集成到各种网络应用中
- 出厂内置电子墨水屏驱动程序（开源）

应用

这款驱动板配合微雪电子墨水屏，可以应用于电子标签的无线更新

- 超市的电子价格标签
- 客户服务窗口的信息标签，比如名字标签
- 广告小标签

目录

产品简介	1
产品特性	1
应用	1
开发环境设置	3
Arduino IDE 的安装和配置	3
代码分析	4
墨水屏选择开关	5
如何使用	5
用户接口	6
图像处理	7
Level	7
Dithering	8
数据传输原理	10
命令	10
初始化算法	10
数据传输	13
像素数据格式	14
硬件	16
引脚	16
尺寸信息	16

开发环境设置

本文中提到的示例程序是基于 C 语言的。修改和编译示例程序，我们需要使用到 Arduino IDE 软件。为了方便查看串口数据，我们还需要用到串口助手软件（如果使用的是 Arduino IDE 的话，IDE 自带了串口监视器）。WIFI 路由器以及连接到该路由器的电脑或者手机。

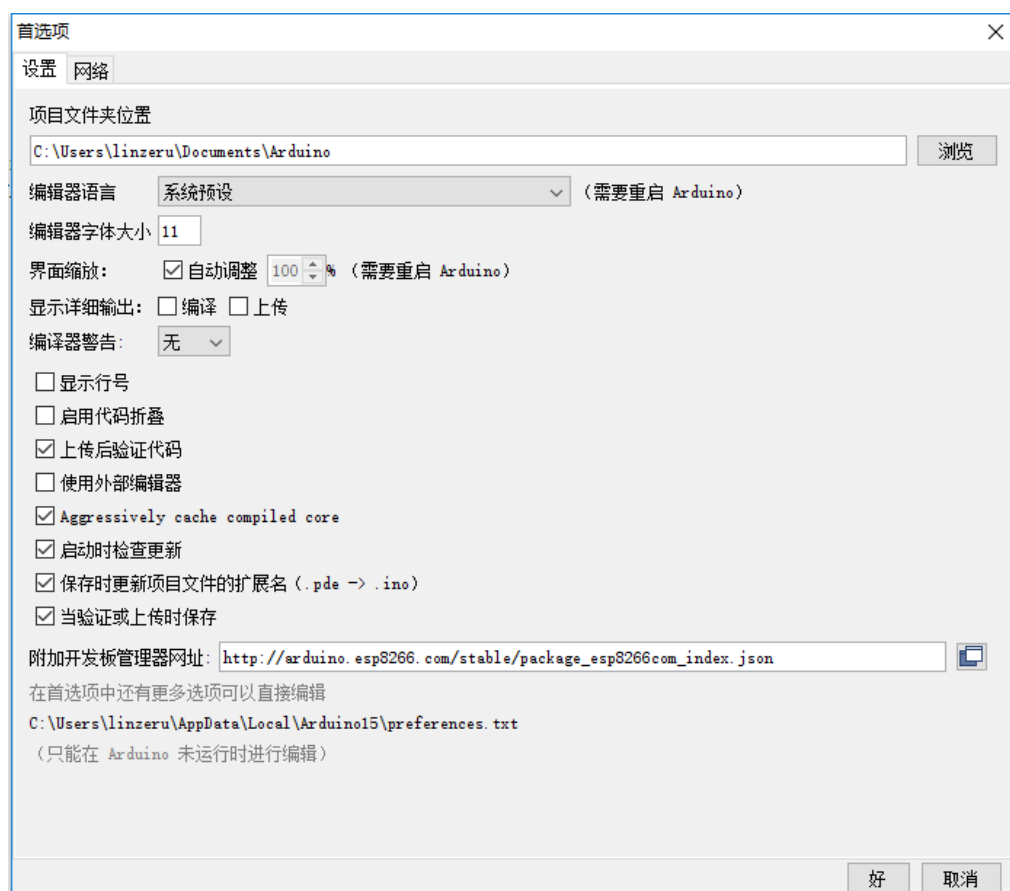
ARDUINO IDE 的安装和配置

如果你已经安装好 Arduino IDE 并且配置好了 ESP8266 开发环境的话，可以直接跳过该步骤

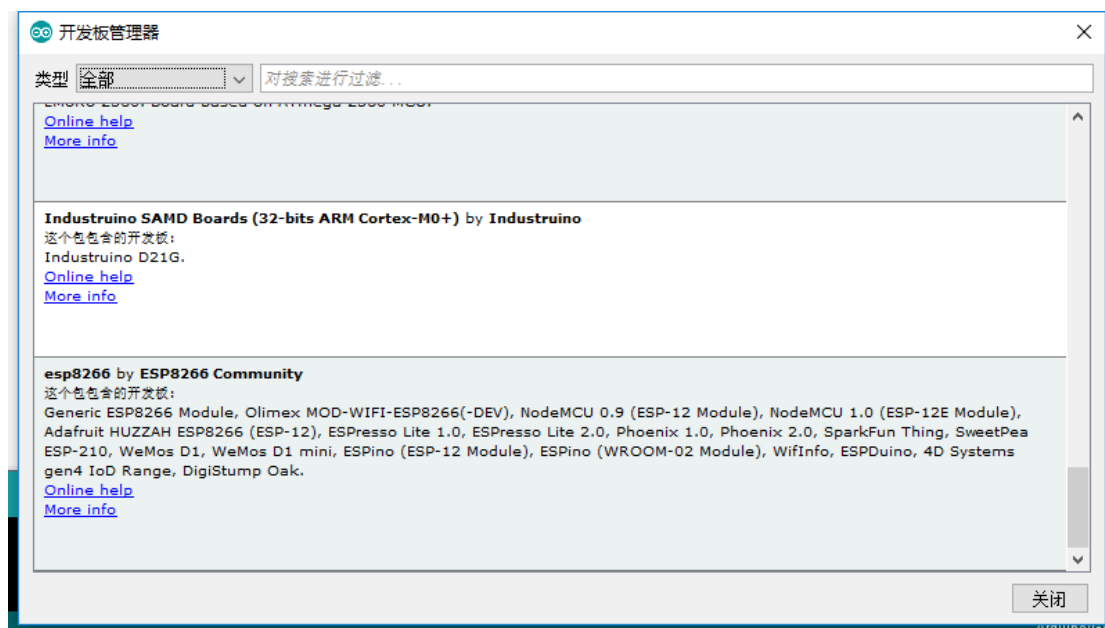
1. 到 Arduino 官方网站下载 [Arduino IDE](#)。用户可以根据自己的系统选择对应的版本下载。

注意：安装新版本的 IDE 并不会覆盖掉之前已保存的工程和文件

- 2 打开 IDE， 打开文件->首选项。在设置界面，将网址 http://arduino.esp8266.com/stable/package_esp8266com_index.json 添加到附加开发板管理器网址框中。然后点击“好”完成设置。



3. 打开工具->开发板管理器。找到 ESP8266by ESP8266 Community 选项并安装。安装完成后，就可以在工具->开发板里面找到 ESP8266 的相关选项了



代码分析

提供的示例程序代码里面主要有以下文件：

loader.ino: Arduino IDE 的工程文件，也是工程的 main 文件。文件中有两个函数 setup 和 loop。这里不另外多做说明

svr.h: 该文件定义了 ESP8266 WIFI 服务的函数。用户需要将 your you're your password 替换成自己 WIFI 的用户名和密码

```
/* SSID and password of your WiFi net -----*/  
const char* ssid    = "your ssid";  
const char* password = "your password";
```

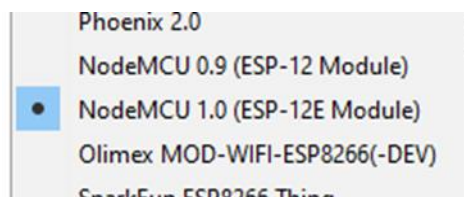
html.h, ccs.h 和 scripts.h: 是 html 上位机和相关函数。一般情况下用户不需要做改动

buff.h: 定义了一个 byte-buff 以及数据存放处理的相关函数。buffer 用来将接收到的图片信息数据里面的 1-, 2-bit 像素格式转化成便于 e-paper 显示用的 1-, 2-, 4-bit 的像素格式，

epd1in54.h, epd2in13.h, epd2.7.h, epd2in9.h, epd4in2.h, epd7in5.h 定义了相关墨水屏的初始化函数和应用程序，包括亮度和颜色饱和度的设置 (lut-massives)

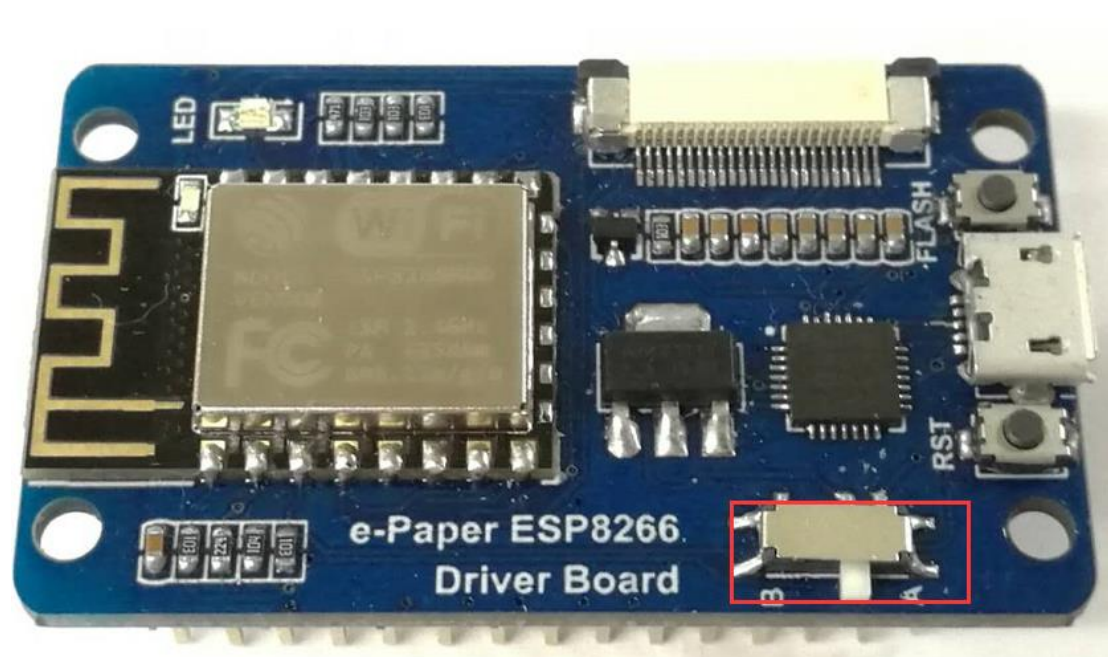
epd.h 定义了 e-paper 状态的控制函数（初始化，颜色通道选择，下传，刷新以及睡眠模式）

连接一个墨水屏到驱动板上，点击“下传”将程序下载到板子。如果下载程序过程出现报错，检查一下 IDE 的工具菜单中是否选择了正确的开发板和 COM 口



墨水屏选择开关

在驱动板上面有一个选择开关。用户需要根据接入的墨水屏型将开关拨到对应的位置，具体的型号对应参考下面的表格：



Trigger state	e-Paper type
A	1.54 inch, 2.13 inch, 2.9 inch
B	1.54 inch(b), 2.13 inch(b), 2.7 inch, 2.7 inch(b), 2.9 inch(b), 4.2 inch, 4.2 inch (b), 7.5 inch(b), 7.5 inch(b)

如何使用

将示例程序下载到驱动板后，连接一个微雪电子墨水屏（裸屏）。使用 HTML 上位机（手机或者 PC 浏览器输入驱动板的 IP 地址），用户可以打开一个图片文件，选

择对应的墨水屏型号，比如选择 4.2b。然后点击 upload image 将程序下载到驱动板，并刷新到墨水屏。

用户接口

上位机网页的用户接口主要有以下的功能：

Select image file-选择打开一个图片文件

Level: black, Level: black and red, Dithering: black, Dithering: black and red-将打开的图片根据墨水屏型号进行处理

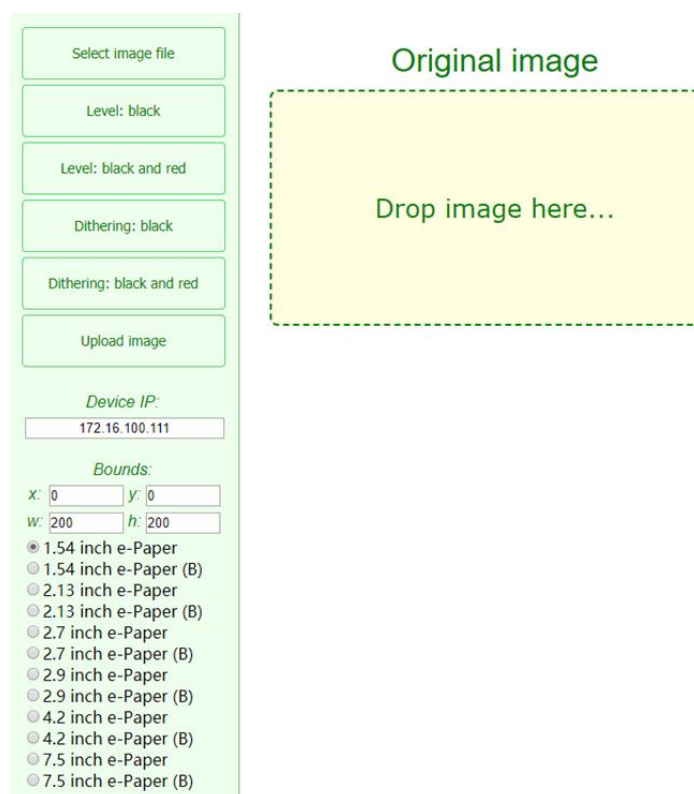
Upload image-向驱动板发起一个 POST 请求并等待驱动板回复“OK”，然后将处理过的图片数据发送到驱动板并刷新到墨水屏上

Device IP: 显示当前连接的驱动板的 IP 地址

Bounds: 显示图片偏移位置(X,Y) 和墨水屏分辨率 (W,H) 会根据对应的选择按钮 (1.5inch e-Paper... 7.5inch e-Paper(b)) 选择的型号显示对应的参数

Viewers: 原图片 (通过拖拽控制) 和处理过的图片会在这里显示 (一开始是没有的)

注意：当你更换连接的墨水屏型号的时候，Bounds 里面的 W,H 和 Device IP 会根据实际墨水屏型号自动更新。但是如果你想要测试一下软件或者是控制其他驱动板的时候，可以通过修改 Device IP 来控制不同的驱动板。



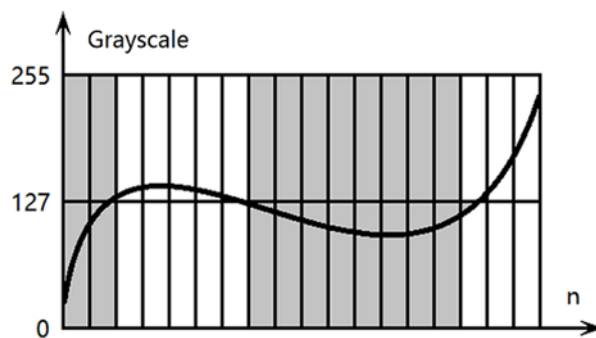
图像处理

在 HTML 页面的使用中，提供了两种图片处理方法: Level 和 Dithering

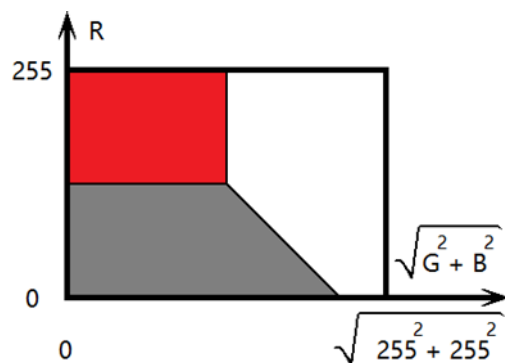
LEVEL

Level 的处理方法中，一张图像上面的所有像素可以被统分为几个大颜色区，图像上的每个像素的被归到最接近的颜色区内。这种处理方式比较适用于 2 种或者三种颜色的显示。

例如，如果灰度图的一个像素颜色值小于等于 127，那么可以说这个像素的颜色归为黑色，否则就是白色。



如果是彩色图像的话，绿色和蓝色的像素可以统称为蓝绿或者相对于红色像素来说的话说它是非红色。在下面的图片中，我们可以看到 R 值高以及 G 值低的像素显示为红色，其他的按照前面的方法区分为黑色和白色。



上面颜色的定义算法是基于差异计算：其他颜色值的平方和（除支持显示的颜色外的其他颜色）。墨水屏显示的时候，每个像素会显示成跟原颜色差异最小的那个颜色。关于颜色处理的代码：

```
// Returns the discrepancy between given (r, g, b)
// and available colors
function getErr(r, g, b, avlCol)
{
    r -= avlCol[0];
    g -= avlCol[1];
    b -= avlCol[2];
    return r*r + g*g + b*b;
}

// Returns the index of available color
// which has minimal discrepancy with the given one
function getNear(r,g,b)
{
    var ind=0;
    var err=getErr(r,g,b,curPal[0]);

    for (var i=1;i<curPal.length;i++)
    {
        var cur=getErr(r,g,b,curPal[i]);
        if (cur<err){err=cur;ind=i;}
    }

    return ind;
}
```

DITHERING

由于一般图片的颜色过渡相对平滑，而这些过渡的颜色可能同时趋近于多种颜色，因此如果使用上面 Level 处理方法的话会丢失掉很多颜色细节。比如摄像头拍摄的图片，过渡颜色区域（阴影）会占大部分。因此通常情况下我们可以选择在那些区域通过颜色抖动的方法来处理阴影。

通常情况下在小区域内的颜色，我们眼睛看到的颜色不一定是真实的颜色，大多是颜色抖动的效果。所以一般来说，对于颜色较少的系统，我们可以以牺牲分辨率为代价，通过抖动来增加可用颜色数量。优秀的颜色抖动算法能够尽可能的避免颗粒感。在这基础上我们采用 Dithering 算法。

示例程序采用 Floyd-Steinberg Dithering 算法。该算法是 Robert Floyd 和 Louis Steinberg 在 1976 年发表的著名的 2D 误差扩散方法。



Figure 1 Floyd-Steinberg 抖动算法的误差扩散

在上面的图片中，X 代表当前像素，误差是原像素和可用的灰度/颜色值之间的标量/向量差。这个误差同时会向右边，右下角，下边以及左下角做扩散，它们的权重分别为 7/16，1/16，5/16 以及 3/16。误差不会向上边以及左边的像素扩散，因为这些像素是已经被扫描过了的并被前面的像素用同样的方法校准过了。



Figure 2 原图



Figure 3 Level : 黑白处理和红黑白处理效果



Figure 4 Dithering : 黑白处理和红黑白处理

数据传输原理

ESP8266 模块可以用来发送一些比较小的数据包，无法发送较大的图片或者视频流。因此这里我们通过分开发送的方法来发送图片数据。如果说你对于 WIFI 的原理不是很了解，但是需要用到 WIFI 来通过 HTML 页面进行文件传输的话，你可以参考一下本文档。这里使用的是 POST 请求数据传输方法。你也可以根据自己的需求对示例程序里面的函数和命令进行修改

命令

本原理是通过在网页方面进行图片数据分割，然后在驱动板上来接收数据并重组。其中包括 4 个指令：

- EPDn-屏幕初始化（n 的范围为 a~l）
- LOAD-下传图片数据（黑色或者红色通道）
- NEXT-从黑色通道切换到红色通道
- DONE-刷新显示并让模块进入睡眠模式

初始化算法

当用户点击“Upload image”按钮的时候，事件处理器会创建一个对象，发送指令到驱动板，并且监听驱动板返回的应答信息，这里发送的是 EPDn 指令。驱动板在接收到 EPDn 指令的时候，会初始化墨水屏并且同时确定刷新颜色准备写入数据。

黑白屏幕的内存写入指令（EPDn 和 NEXT）分别是：

```
EPD_SendCommand(0x24); //WRITE_RAM
```

对于红黑白三色墨水屏的写入命令是（黑色和红色）：

```
EPD_SendCommand(0x10); //DATA_START_TRANSMISSION_1,
```

`EPD_SendCommand(0x13); //DATA_START_TRANSMISSION_2,`

有几个例外的：

- 黑白屏幕 2.7 寸和 4.2 寸的使用的是红色通道 (0x13)
- 7.5 寸的屏幕同时载入红色和黑色数据

注意：在移植代码的时候，要注意屏幕的数据写入方式来移植相应的代码。

需要特别注意的是，在写入图片数据的时候，除了 2.13 寸屏幕是一行行写入的之外，其他屏幕都在写入每一行数据前，都必须先调用指令 `WRITE_RAM`, `DATA_START_TRANSMISSION_1` 或者 `DATA_START_TRANSMISSION_2`。

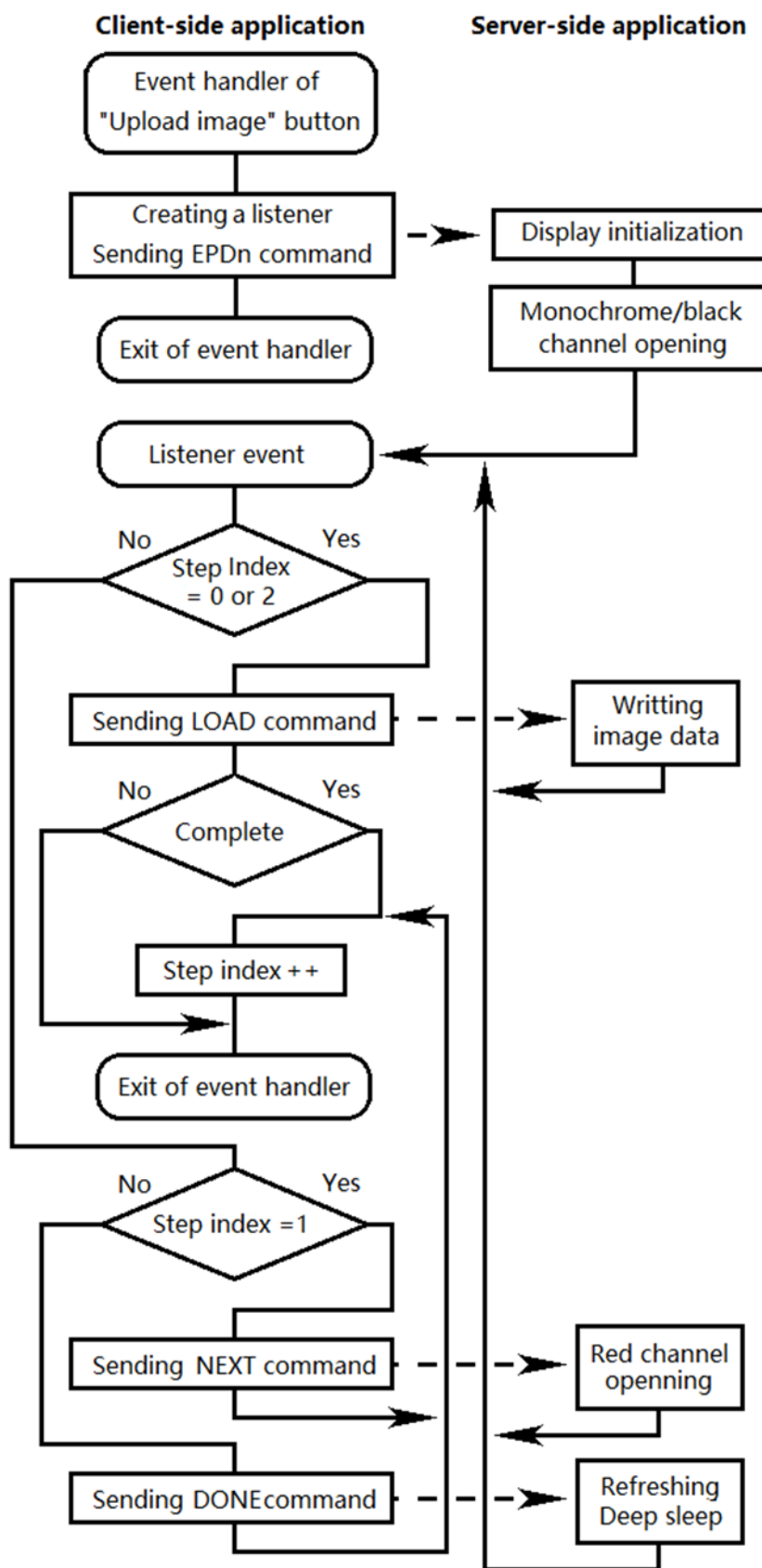


Figure 5 初始化和数据传输算法流程

数据传输

每次驱动板在收到请求命令的时候，都会回传“OK”作为应答。收到这个应答信息后，程序会初始化 `xhReq` 对象的 `onload` 事件并且网页端会根据 `stInd` 再次发送指令（参考上面的算法流程图和 `uploading.js` 文件）：

```
xhReq.onload = xhReq.onerror = function()
{
    if(stInd==0)return u_data(a,(epdInd==1)?-1:0,0,50);
    if(stInd==1)return u_next();
    if(stInd==2)return u_data(a,3,50,50);
    if(stInd==3)return u_done();
};
```

这里的 `epdInd` 就是在 `EPDn` 指令里面提到墨水屏型号 `n`

注意：黑白电子墨水屏跳过步骤 1 和 2

图片数据会作为 `POST` 请求被发送，驱动板在接收到图片数据后会先存放到一个缓冲区进行处理（参考 `buff.h` 文件）：

```
/* Size, current position index and byte array of the buffer -----*/
#define Buff__SIZE 2050
int    Buff__bufInd;
char    Buff__bufArr[Buff__SIZE];
```

每字节有 `2050>=` (大约 1000 字节数据会被发送到的 ESP8266 模块+ 15 字节)x2 char。

注意：使用该缓冲区，你也可以发送更多的字节，这种情况下就需要修改 `Buff_getByte` 函数（`buff.h`）和 `byteToStr` 函数（`uploading.js`）。另一个提高请求数据容量的方法就是估计请求的辅助数据，不过这种方法功能跟第一种是兼容的。

实际上，Arduino 有一个函数 `ReadStringUnit`，这个函数可以存放并返回整个请求的数据，但是这个函数使用起来非常的慢。示例程序中我们使用了一个 `while` 循环结构。结构里面会不断存储 `POST`-请求里面发送过来的数据，并且循环里面的每次迭代都有一个信号量（`styles.css` 和 `uploading.js`）用来判断该数据是否需要被存放起来。

```
// While the stream of 'client' has some data do...
while (client.available())
{
    // Read a character from 'client'
    int q = client.read();

    // Save it in the buffer and increment its index
    Buff_bufArr[Buff_bufInd++] = (byte)q;

    // If the character means the end of line, then...
    if ((q == 10) || (q == 13))
    {
        // Clean the buffer
        Buff_bufInd = 0;
        continue;
    }

    // If data accumulated in buffer end with "styles.css", then
    if ((Buff_bufInd > 13) && Buff_signature(Buff_bufInd - 14, "/styles.css"))
    {
        ...
        // If the buffer's length is larger, than 4 (length of command's name), then...
        if (Buff_bufInd > 4)
        {
            // It is probably POST request, no need to send the 'index' page
            isIndexPage = false;

            // e-Paper driver initialization
            if (Buff_signature(Buff_bufInd - 4, "EPD"))
            {
                ...
            }
        }
    }
}
```

像素数据格式

当用浏览器打开一张图片的时候，原图的像素格式默认是 24 bpp 的，经过处理后的图像会被降低到 1 bpp (黑白):

0 : 白色

1 : 黑色

或者是 2bpp (黑白红) :

0 : 白色

1 : 黑色

2 : 灰色 (只适用 1.54inch e-Paper B)

3 : 红色

在 xhReq 对象发送 POST 请求的之前，程序会先将图片数据打包成字节或者字。打包后的数据的数据位顺序是按照屏幕显示顺序或者说是最快转换成原图显示的顺序排列的。下面的表格说明了如何数据格式的对应：

屏幕类型 (channel)	传输格式 (p – pixel, b - bits)	存放格式
1.54 (黑白) 1.54b (三色红) 2.13 (黑白) 2.13b (三色) 2.7b (三色) 2.9 (黑白) 2.9b (三色) 4.2 (黑白) 4.2b (三色)	p:01234567 - b:76543210 0 – 黑色或红色; 1 – 白色.	p:01234567 - b:76543210 0 – 黑色或红色; 1 – 白色. 注：没有变化
2.7	p:01234567 - b:76543210 0 – 黑色; 1 – 白色.	p:01234567 - b:76543210 0 – 白色; 1 – 黑色 (). 注: 逐位取反
7.5	p:01234567 - b:76543210 0 – 黑色或红色; 1 – 白色.	p:0 - b:7,6,5,4 (7 is high, 4 is low); p:1 – b:3,2,1,0; p:2 – b:15,14,13,12; p:3 – b:11,10,9,8; 0000 (0) – 黑色; 0011 (3) – 白色.
1.54b (黑色)	p:0 – b:1,0; p:1 – b:3,2; p:2 – b:5,4; p:3 – b:7,6; 00 (0) – 黑色; 01 (1) – 白色; 10 (2) – 灰色; 11 (3) – 红色, is read as 10 (2).	p:0 – b:7,6; p:1 – b:5,4; p:2 – b:3,2; p:3 – b:1,0; 00 (0) – 黑色; 01 (1) – 灰色; 11 (3) – 白色;
7.5b	p:0 – b:1,0; p:1 – b:3,2; p:2 – b:5,4; p:3 – b:7,6; 00 (0) – 黑色; 01 (1) – 白色; 10 (2) – 灰色; 11 (3) – 红色.	p:0 - b:7,6,5,4 (7 是高位, 4 是低位); p:1 – b:3,2,1,0; p:2 – b:15,14,13,12; p:3 – b:11,10,9,8; 0000 (0) – 黑色; 0011 (3) – 白色; 0100 (4) – 红色.

硬件

墨水屏网络驱动板是通过 WIFI 网络对微雪电子墨水屏进行图片刷新，模块板载 ESP8266 芯片，用户也可以使用本模块实现更多的功能：用户可以将图片保存到 SD 卡或者其他外部存储（要另外添加外部存储模块），读取传感器信息并显示到墨水屏上面等等。用户可以通过驱动板背面引出的两排排针连接其他的模块。

引脚

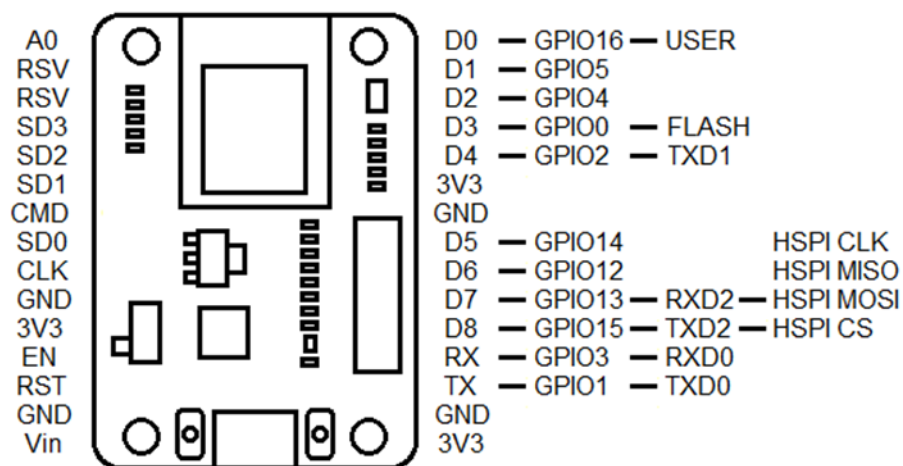


Figure 6 驱动板的引脚定义

注：不需要使用墨水屏或者刷新他的话，可以将墨水屏从驱动板上面取下

尺寸信息

