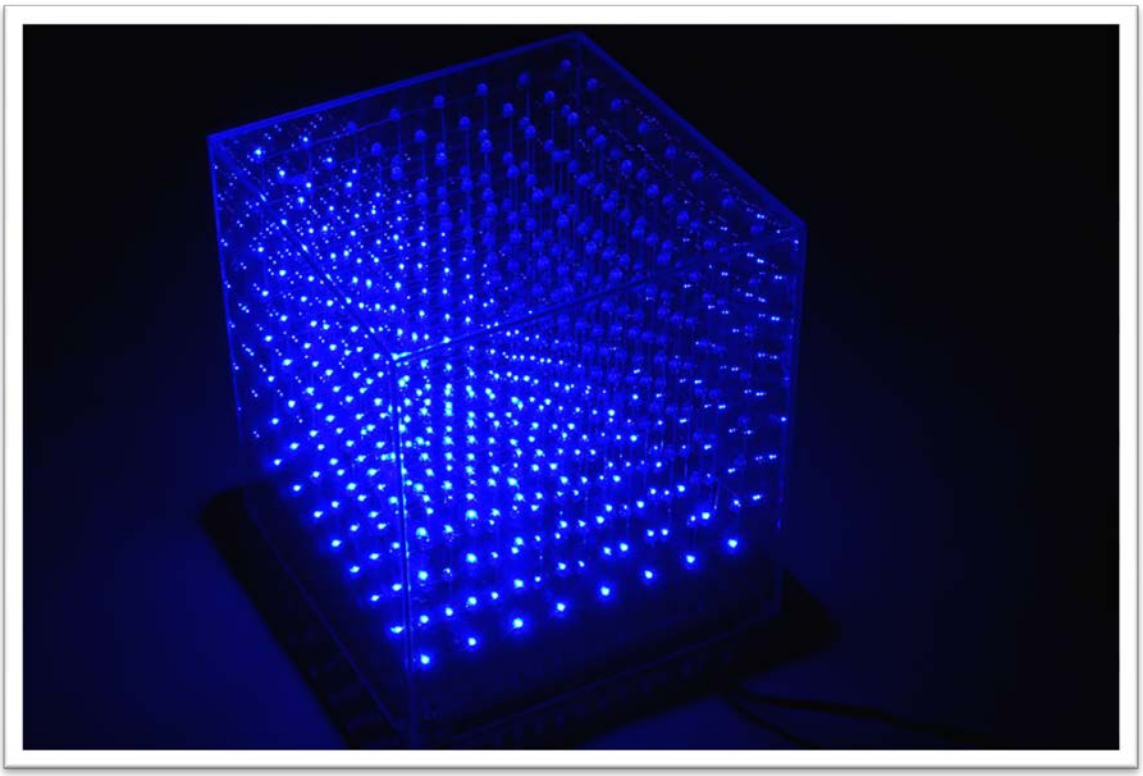
3D8 串口数据通讯协议 v1.6

aGuegu / 官微宏 2011-08-02（2012-01-02 更新）

本文介绍 3D8 8x8x8 LED 光立方的串口通讯办法，并提供例程。

背景知识要求：C 语言基础，单片机 C 语言程序设计基础。

目录



指令系统

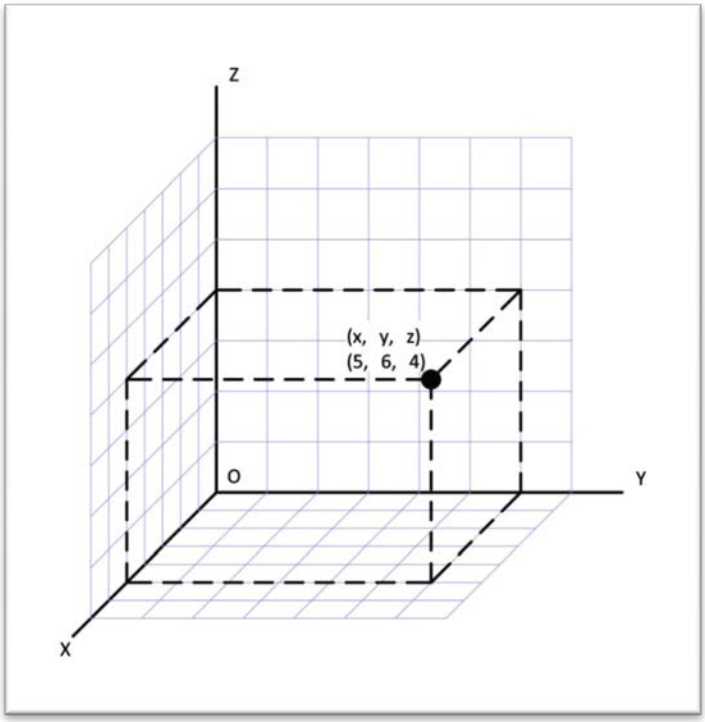
3D8 串口通讯协议，默认设置：波特率：115200；校验位：无；数据位：8；

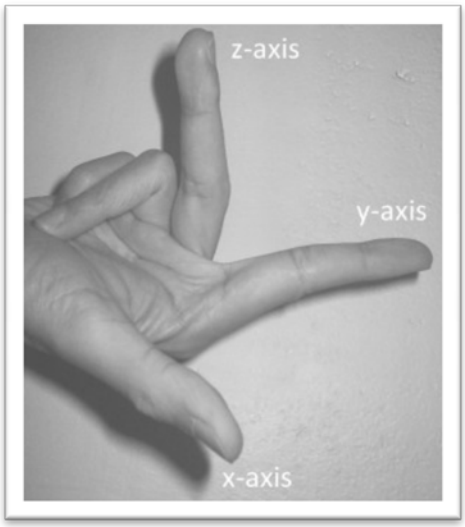
每次命令由 2 个字节构成。先发送高 8 位（H7-H0），再发送低 8 位（L7-L0）。考虑到系统运行效率，以及编程的简单性，不设校验位，合计三种代码格式，使用三种开始码与之对应、不设结束码。

因为不设置结束码，所以必须要保证每条代码发送的完整性。

功能	开始码	字节数	格式	示例
全局赋值	0xf0	2	0xf0 + Data	发送“0xf0 0x00”，则将所有束的值设为 0x00， 实现清屏操作。
单束赋值	0xf1	3	0xf1 + Column + Data	发送“0xf1 0x02 0xff”，则将第 2 束的值设为 0xff，即第 2 束全亮。
批量赋值	0xf2	65	0xf2 + Data[0] + Data[1] + … + Data[63]	发送“0xf2 0x3f 0x3e 0x3d …… 0x01 0x00”，即将 0x3f 赋值给第 0 束，即将 0x3e 赋值给第 1 束，即将 0x3d 赋值给第 2 束，……，将 0x3f 赋值给第 0 束。更新全屏画面。

- 坐标系：将 3D8 按照右手坐标系建立坐标系统，则我们可以定义 3D8 系统中任意一个光点（LED）的坐标为（x, y, z）。则可知 x,y,z 的值域（取值范围）皆为[0, 7]的整数域，即{0, 1, 2, 3, 4, 5, 6, 7}。





注：下文中提到的数字，皆为整数。

- 束坐标（Column，简写为 c）：是在 3D8 系统中，结合硬件系统定义的坐标值，其值 c = y*8 + x。由此可

得 c 的值域为[0, 63]；并且我们还能计算出 x = c % 8, y = c / 8。（熟悉编程的朋友都会知道，这里的“%”为取余运算，“/”为整除运算）。

- 数据值（Data），即定义每一束（Column）上的显示状态。因为会有多点同时亮起的情况出现，且 z 的范围为[0, 7]，所以一个字节（byte）的 8 个位（bit），恰好可用于定义这个状态。如果要求某束只有 z=0 点（第 0 层，即最底层）亮起，则可将 Data 值设置为 0x01（0000 0001 B）（后缀 B，表示该数为二进制数）。若要求某束只有 z=6 点（第 6 层，从上往下第 2 层）亮起，则 Data = 0x40（0100 0000 B）；若要求 z=0, z=6 点同时亮起，则 Data = 0x41（0100 0001 B）。
- 驱动流程：
  1. 主板上电以后，再给上位机上电。保证主板提前进入监听状态。
  2. 在上位机程序中，可以通过上述三种中指令的一种或多种组合，实现对一屏画面的赋值。
  3. 赋值结束后，主板内部的数据缓存会保存这些值，并不断对 LED 进行扫描显示。上位机可以通过延时函数，控制两帧画面之间的间隔。
  4. 循环步骤 2、3，实现动画效果。
- 系统重置

  系统重置在目前的通讯协议下显得尤为重要，因为指令有长有短，同时主板并不会去判断连接是否中断。所以当一条指令没有发送完整时，若出现通讯中断，主板会继续等待剩余指令发送完成。而如果此时重新建立连接，主板将不会把接收到的第 1 个字节作为开始码进行判断。这就导致通讯无法正常进行，此时需要重启主板，上位机重新开始发送数据，方可让通讯同步。

  简而言之，在断开上位机与主板的连接以后，重新连接时，应重启主板，而后上位机再开始发送数据。若使用控制棒，控制棒也将使用主板电源，所以此时若重启主板，主板和控制棒会同时重新启动。因为控制棒内部包含启动延时程序，所以使得在其发送数据之前，主板已经提前进入了监听状态。

更新手记：

V1.3 （2011-08-11）：修正了单束赋值命令中，束坐标可能超出范围的 bug。

V1.4 （2011-08-13）：修正了发送代码过长、错位而导致开始码错误进而系统不响应的 Bug，若开始码非 0xf0/0xf1/0xf2，主板将继续等待接收开始码。

V1.5 （2011-11-06）：修正了例程中的一些 bug。

V1.6 （2012-01-02）：修正文档中例程的 1 个 Bug（感谢群友“山东-小星星”指出），添加了 3D8-Lib 的链接。

底层驱动模块

```
//////////////////////////////////////////////////////////////////////////////
// 3D8 串口通讯底层驱动（程序模板）
// 基于上位机 12C5A60S2+22.1184M 外部晶振，若使用其它上位机，请在充分理解下列程序以后进行修改。
// aGuegu / 官微宏 2011-11-05 weihong.guan@gmail.com http://aguegu.net
//////////////////////////////////////////////////////////////////////////////

#include <REG51.H>
#include <intrins.h>
#include <stdlib.h>
typedef unsigned char uint8;
typedef unsigned int uint16;

#define LAYER_COUNT 8
#define COLUMN_COUNT 64

// 延时函数
void delay(uint16 a)
{
    while( a-- )
        _nop_();
}

// 长延时函数
void delayL(uint16 a)
{
    while( a-- )
        delay(-1);
}

// 串口初始化：波特率 115200（使用频率为 22.1184M 的外部晶振驱动）
void UART_Init (void)
{
    TMOD = 0x20;
    SCON = 0x50;
    TH1 = 0xff; TL1 = 0xff; // 设置波特率为 115200
    PCON = 0x80;
    TR1 = 1;
}

// 串口发送数据函数
```

```
void UART_Send (uint8 cData)
{
    SBUF = cData;
    while(!TI);
    TI = 0;
}

// 根据 x,y 的值计算 column 值
uint8 funGetColumn(uint8 x, uint8 y)
{
    return (8*y+x);
}
// 全局赋值函数
void funPrintUniqueValue2Cube(uint8 cData)
{
    UART_Send (0xf0);   // 开始码内置于函数中发送
    UART_Send (cData);
}

// 单束赋值函数
void funPrintColumn(uint8 cColumn, uint8 cData)
{
    UART_Send(0xf1);
    UART_Send(cColumn);
    UART_Send(cData);
}

void funPrintCube(uint8 *p)
{
    uint8 i;
    UART_Send(0xf2);
    for (i=0; i < COLUMN_COUNT; i++)
        UART_Send(p[i]);
}
////////////////////////////////////////////////////////////////////////////
// 以上函数作为固定驱动程序模块，一般不用修改
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// 动画程序段：此处插入打包以后的动画程序
////////////////////////////////////////////////////////////////////////////
```

```
// 主函数
void main(void)
{
    delay(-1); //其实这里的-1 等效于 65535，也就是这个延时函数所能提供的最大延时间隔
    delay(-1); //
    UART_Init(); // 初始化串口
    while(1)
    {

    }
}
//////////////////////////////////////////////////////////////////////////////
```

驱动例程

- 例程 1 ——全屏闪动

// 将模板中主函数改为如下所示

```
void main(void)
{
    delay(-1);
    delay(-1);

    UART_Init(); // 初始化串口

    while(1)
    {
      funPrintUniqueValue2Cube(0xff);  // 开启所有显示
      delayL(4);
      // 显示时长，可以通过减小参数值，或增加 delay()运行的次数，改变闪烁间隔
      funPrintUniqueValue2Cube(0x00);  // 关闭所有显示
      delayL(4);
    }
}
```

小贴士：

- 单帧显示时长由延时函数 delay(uint16); 控制，参数为(-1)的话，间隔时间比较长，如果变为 0x20，甚至更小，甚至干脆不要延时。则会因为显示太快而导致肉眼察觉不出变化，反而是看到全屏都被点亮。这个时候并不是主板运行不正常，而是扫描太快，导致肉眼看不出变化。
- 控制帧速（单帧显示时长的倒数），除了用延时函数的方法，当然还可以用定时器中断的方法，有兴趣的朋友不妨试一下。☺

- 例程 2 ——三面扫描

在动画程序段插入如下函数：

```
// z 方向，逐面扫描
void funScanByZ(void)
{
    uint8 z;
    funPrintUniqueValue2Cube(0x00);
    for (z=0; z< LAYER_COUNT; z++)
    {
        funPrintUniqueValue2Cube(0x01 << z);
        delay(-1);
    }
}

// x 方向，逐面扫描
void funScanByX(void)
{
    uint8 x,y;
    funPrintUniqueValue2Cube(0x00);
    for (x=0; x< LAYER_COUNT; x++)
    {
        funPrintUniqueValue2Cube(0x00);       // 若屏蔽此行，则函数变为逐面点亮。
        // 因为单束赋值并不会对别的束的值发生影响，所以在每一帧显示开始时，对全屏进行清屏操作。
        for (y=0; y<LAYER_COUNT; y++)
            funPrintColumn(funGetColumn(x,y), 0xff);
        delay(-1);
    }
}

// y 方向，逐面扫描
void funScanByY(void)
{
    uint8 x,y;
    funPrintUniqueValue2Cube(0x00);

    for (y=0; y< LAYER_COUNT; y++)
    {
        funPrintUniqueValue2Cube(0x00);       // 若屏蔽此行，则函数变为逐面点亮。
        // 因为单束赋值并不会对别的束的值发生影响，所以在每一帧显示开始时，对全屏进行清屏操作。
        for (x=0; x<LAYER_COUNT; x++)
            funPrintColumn(funGetColumn(x,y), 0xff);
        delay(-1);
    }
}
```

将主函数更新为

```
void main(void)
{
    delay(-1);
    delay(-1);

    UART_Init(); // 初始化串口

    while(1)
    {
        // 依次运行三个方向的扫描程序
        funScanByX();
        funScanByY();
        funScanByZ();
    }
}
```

小贴士：

- 在这里已经就利用了建立子函数的方法，将动画进行“打包”，使得主函数简洁明了。
- 在主函数的 while(1)循环内，依然可以使用 for,while 循环语句，来决定动画如何进行循环。
- 目前这几个动画函数都没有参数，有兴趣的朋友不妨试试看能否把内部 delay()函数的参数作为整个函数的参数，这样当外部调用的时候，就可以直接控制帧速了。☺

- 例程 3 ——上升流

在动画程序段插入如下函数：

```
void funDemoRise(uint8 *pCube)
{
    uint8 i, x, y, j;

    j = 0x80; // j 作为计数器，计算总共上升了多少次
    while(j--)
    {
        // 上升
        for(i=0; i< COLUMN_COUNT; i++)
        {
            pCube[i] <<= 1;
        }

        // 底面重新有 0-3 个新的点亮起
        for(i=0; i< rand() % 4; i++)
        {
            x = rand() % 8;
            y = rand() % 8;
            pCube[funGetColumn(x,y)] |= 0x01;
        }

        funPrintCube(pCube);
        delay(0x100);
    }
}
```

主函数改为：

```
void main(void)
{
    uint8 pCube[COLUMN_COUNT];

    delay(-1); delay(-1);

    UART_Init(); // 初始化串口
    srand(9); // 给个随机种子
    while(1)
    {
        funDemoRise(pCube);
    }
}
```

例程工程文件下载链接：

http://aguegu.net/wordpress/wp-content/uploads/2011/11/062-3D8-Controller-Demo.zip

小贴士：

- 这里只用到批量赋值命令，这也是我推荐大家使用的命令。其基本流程就是，在上位机内部建立一个数组，通过各种运算，改变这个数组里面的值，以达到理想的效果，在运算完成以后，使用批量赋值命令，将整体输出到 3D8 主板上。虽然这样传输的数据量最多，需要 65 个字节，但是我们的高波特率传输已经使得，肉眼察觉不到这个时间，而且这设计动画效果过程中，这样的思路最简单。
- 这里用到了随机函数，随机函数是最简单的让动画看起不那么千篇一律的方法，充分利用它吧。Rand() 的返回值范围为[0,255]，是用取余运算%8，可将其范围缩小到[0,7]。
- 位运算，在单片机程序中，位运算是经常用到的，这里用到了或运算|和左移位算<<。还有&、>>等，好好复习一下，未来都会用得到。
- 指针，这里我使用了指针，传统的单片机程序可能更习惯于使用全局变量（当然，这个变量也是个数组），但是这么做会增加冗余度，牺牲系统的稳定性，消耗大量的系统资源。其实完全可以避免，把 C 语言的教材翻出来，看看指针怎么用吧。☺
- 指针运算中，最怕的就是指着指着，指到别的地方去，万一做了什么修改会导致程序的崩溃、跑飞。所以我在宏定义中加入了 LAYER_COUNT 和 COLUMN_COUNT，对数组下标的上限进行了明确的定义。而且在指针中尽量使用 p[c]这样的形式来调用，比较直观。（当然，高手的话，就更喜欢用 p++之类更加晦涩的语句了☺）。
- 关于计数器 j，其实我不太喜欢这个东西，但这也是最简单的，控制动画整体播放长度的方法。如果只是单一的效果，不要这个循环也罢。但是如果有多个动画滚动显示，就需要这样的跳出机制。有兴趣的朋友试试看，使用按键等其它办法来实现它。

欢迎大家参与交流

3D8 光立方官方交流 QQ 群：1 群：165068863（已满员）、2 群：179638304

资料汇总：http://aguegu.net 进入“Macro”栏目即可。
淘宝店链接：http://item.taobao.com/item.htm?id=10959651858

参考资料：
视频：3D8 光立方 8x8x8 LED 通讯协议说明
http://v.youku.com/v_show/id_XMjkyNDQ0MzY0.html

视频：使用 51 单片机最小系统控制 3D8 光立方
http://v.youku.com/v_show/id_XMzE5NzU4NTIw.html

动画函数库：V0.1 介绍、视频、下载，可以更方便的编动画哦~
http://aguegu.net/?p=397