数字频闪光源控制器

CK-HDT48 系列

CK-HDT48 系列数字频闪光源控制器，搭载行业领先的 Overdrive 增压模式，专为追求极致响应速度与光强输出的工业检测、高速成像场景量身打造，重新定义频闪控制的效率与精度边界。

一、功 能 特 性

1. 单通道带载输出电流高达 3A。
2. 旋钮/串口通信方式控制亮度，提供 0～255 共 256 级亮度调节。
3. 外触发模块采用光耦隔离设计，有效隔绝干扰信号，兼具超高响应速度与稳定可靠的信号传输表现稳定。
4. 具备掉电保存、过载/短路保护功能。
5. 先进的 Overdrive 模式，在不损伤光源寿命的前提下，将光源峰值亮度提升数倍（通常可达 10 倍以上），从而实现更高的瞬时光强输出。
6. 提供定制化驱动电源服务，可根据需求灵活调整规格，最大功率支持 600W，满足多元场景应用。

二、规 格 描 述

型号	CK-HDT48-200W-4DT
驱动方式	OverDrive
调光方式	256 级 PWM 控制
	面板旋钮/RS232
PWM 频率	120KHz
输入电压	AC200-264V 50/60Hz
通道	4
输出电压	DC 48V
最大输出电流	3A（单通道）
触发模式	高电平/低电平/软件触发

最大输出功率	200W
外触发电压	DC 5V～24V（电流约 5.6mA）
光源触发延迟	≤15μs
触发延迟	<30μs（与负载有关）
使用温湿度	温度：0～40℃、湿度：20～85%RH（非凝结）
保存温湿度	温度：-20～60℃、湿度：20～85%RH（非凝结）
冷却方式	自然冷却

三、使 用 说 明

➢ 光源控制板各区功能介绍

1. 电源开关：

左端三头插座，接交流输入，输入电压范围 AC100-240V 50/60Hz；右端开关打到“O”关闭总电源，打到“I”电源开关打开。

2. RS232 接口:

连接串口通讯线缆到上位机，使用指定的格式指令通过串口通讯软件可以调节光源亮度及软触发。

通信接口: RS232

波特率: 115200

数据位: 8 位

停止位: 1 位

校验位: 无

控制方式: 使用指定格式的指令通过串口通信软件进行控制。

字符集: ASCII 码

多通道指令分割符号: 英文半角逗号（,）。

详见通讯协议。

3. 触发输入口:

四路触发输入，按照顺序分别为 TG1、TG2、TG3、TG4。每一路的触发信号正接对应的“+”端，负接对应的“-”端。

控制器接收到触发端口的脉冲的上升沿信号（9 拨片开关打到上面），控制器驱动对应序号的光源，并输出脉冲信号到对应的触发输出（CAMERA）端口,触发相机拍照。

4. 触发输出口:

根据接收到的对应通道的输入信号，进行触发输出，输出脉冲的幅值为 12v，输出正接“CAMERA 触发输入”的正端，输出负接“CAMERA 触发输入”的负端。驱动开关为大于 100mA。

5. 数码显示屏：

四位七段数码显示，显示通道序号及对应的亮度值；左边第一位显示通道序号，右边三位显示亮度值，范围：0~255. 

无操作时轮流显示：当前模式->当前电压->通道 1 亮度->通道 2 亮度->通道 3 亮度->通道 4 亮度。

当处于 Lock 档位时，若再有其他操作，则恒显示 lck。

6. 状态指示灯：

H/L 指示：启用硬触发，上升沿有效；

ERR 指示：系统错误提示，检查接线是否短路；

LOCK 指示：旋钮锁定，无法通过旋钮调节光源亮度；

7. 带按键功能的旋钮 PUSH/CONTROL

未操作的情况下，当每按下一次按钮，就会从 n000 一直往下切换；如若已有操作，则从上一次操作的地方开始。

旋钮调节光源亮度值，顺时针旋转，亮度增加；逆时针旋转，亮度减小。

8. 锁定(LOCK/UNLOCK)切换开关

开关打到 LOCK，此时不能修改调节亮度，光源控制器进入触发工作模式；

开关打到 UNLOCK 状态，此时可以通过按钮旋钮选择不同的通道和设置参数。

9. 触发模式切换开关

开关打到“H”，高电平触发有效；

开关打到中间位置，低电平触发有效；

开关打到“L:”，硬触发模式关闭，启用软触发模式。

10. 四路光源接口

光源接口顺序，从上到下依次是 CH1、CH2、CH3、CH4 与触发输入和输出序号对应。

➢ 工作模式

■ **过驱动频闪模式**

当接收到单个有效触发信号（高 / 低电平）时，控制器驱动相关通道的光源，同时向对应触发输出端口输出触发信号。使用此模式时，需先按下旋钮切换至 N000 模式 (默认模式)。

1. 若使用 PNP 触发输入（高电平有效），请将拨码开关拨至"H"位置。
2. 若使用 NPN 触发输入（低电平有效），请将拨码开关拨至中间位置。特别注意，此模式下若未连接外部触发，对应通道的光源将默认进入常亮模式，可能导致光源损坏，使用时务必谨慎。
3. 将拨码开关拨至"L"位置，可通过软件指令控制光源及触发输出。

■ **特殊模式**

1. N001 模式将拨码开关拨至"H"位置可启用此功能。无外部触发时，光源会周期性自动快速闪烁，周期为 1ms，峰值照明时间约 180μs，此模式可用于光源测试。
2. N002 模式此模式下，仅当旋钮切换至对应通道时，光源才会开启并处于常亮状态，使用时请谨慎操作。

➢ 控制方式

一、通过本机硬件直接控制光源

通过多功能旋钮进行控制，按压切换光源通道，旋转控制亮度。

二、通讯协议的使用或封装

该协议用于通过 RS232 串口与光源控制器进行通信，以调节光源的开关和亮度。通过发送指定格式的指令控制多个通道的光源状态。

RS232 串口参数配置：

端口号	**波特率**	**数据位**	**停止位**	**校验位**	**控制字符**	**结束符**
COM*	115200	8	1	无	ASCII 码	回车（\r）

端口号*选择提示：

端口应为光源控制器的 RS232 线连接电脑后端口号，可在设备管理器中查看。

可参考，通过当前光源控制器 RS232 线连接的电脑桌面，【我的电脑】右键菜单中打开【管理】，进入【设备管理器】中，查看【端口】。

1. 通讯指令格式

光源控制指令通过串口发送，指令格式为：

M<通道号>=<开关状态>,I<通道号>=<亮度值><回车符>

M<通道号>：控制通道开关，值为 0 表示关闭，值为 1 表示开启。

I<通道号>：控制通道亮度，亮度值范围是 0-255。

结束符：每条指令后必须加回车符（\r）。

注意：多个通道指令同时控制使用半角逗号符号隔开。

示例指令

打开通道 1，且使通道 1 的光源亮度设置为 100：

M10=1,I10=100\r

关闭通道 2，使通道 2 的光源亮度设置为 50（关闭通道，设置该值实际无意义）：

M20=0,I20=50\r

使用半角逗号符号隔开，同时控制多个通道：

M10=1,I10=100,M20=1,I20=150\r

2. 通道配置

光源控制器支持最多 4 个通道，通道编号为 1 至 4，指令中的<通道号>值取 1、2、3、4。

M10、M20、M30、M40：分别表示通道 1 至通道 4 的开关状态。

I10、I20、I30、I40：分别表示通道 1 至通道 4 的亮度调节值，范围从 0 到 255。

注意：使用指令控制时，先将“锁定”（LOCK/UNLOCK）开关切换至“UNLOCK”状态，才能进行控制。

3. 开发注意事项

波特率设置：确保软件与光源控制器的波特率一致，默认值为 115200。

指令结束符：每个指令后必须加回车符（\r），确保正确发送。

串口配置：确保串口参数配置（数据位、停止位、校验位等）正确，通常为 8 位数据位，1 位停止位，无校验。

4. 故障排除

1) 无法发送指令：

- 检查串口连接是否正常。
- 确认串口工具的波特率、端口号设置是否正确。

2) 光源不响应：

- 确保光源控制器电源已打开。
- 确认指令格式正确，且包含必要的结束符（回车符）。

3) 亮度调节无效：

- 确保指令中的亮度值在 0-255 的范围内。
- 切换光源控制器的状态为“UNLOCK”以启用控制。

本通讯协议提供了通过 RS232 串口调节光源控制器的方式，支持控制开关和亮度调节。开发者可以根据该协议封装软件，实时控制多通道光源的状态。

四、 编写软件控制光源控制器示例

基于当前提供的 RS232 串口通讯协议，我们可以编写一个 C++程序来通过串口与光源控制器设备进行通信，以控制光源的通道开关和亮度。

以下是使用 Windows API 封装的代码示例，它能够通过串口发送命令来控制设备的开关和亮度，命令格式类似于 M10=1,I10=100，用来控制第 1 通道的开关和亮度。

1. RS232 控制协议解析

M10=1：控制第 1 通道的开关，1 表示打开，0 表示关闭。

I10=100：设置第 1 通道的亮度，亮度值范围为 0 到 255。

波特率：115200，意味着数据传输的速度是 115200 位每秒。

结束符：命令以回车符（'\r'）结束。

2. C++ 串口封装类（使用 Windows API）

下面的代码展示了如何通过串口控制设备的亮度和开关，具体实现了通过 M 和 I 指令来控制通道的开关状态和亮度值。代码示例 Demo.cpp 如下：

```
#include <iostream>
#include <windows.h>
#include <stdexcept>
#include <string>
#include <sstream>

class SerialPort {
public:
    // 构造函数，初始化串口
    SerialPort(const std::string& portName, DWORD baudRate = CBR_115200) {
        // 打开串口
        hSerial = CreateFile(
            portName.c_str(),              // 串口名，例如 "COM1"
            GENERIC_READ | GENERIC_WRITE, // 读写权限
            0,                            // 独占访问
```

```
    NULL,                          // 默认安全属性
    OPEN_EXISTING,                 // 如果不存在则返回错误
    0,                             // 无重叠模式
    NULL                           // 默认模板文件
);

if (hSerial == INVALID_HANDLE_VALUE) {
    throw std::runtime_error("无法打开串口 " + portName);
}
// 配置串口参数
DCB dcbSerialParams = { 0 };
if (!GetCommState(hSerial, &dcbSerialParams)) {
    CloseHandle(hSerial);
    throw std::runtime_error("无法获取串口状态");
}
// 设置串口参数
dcbSerialParams.DCBlength = sizeof(dcbSerialParams);
dcbSerialParams.BaudRate = baudRate;  // 波特率
dcbSerialParams.ByteSize = 8;         // 数据位 8
dcbSerialParams.StopBits = ONESTOPBIT; // 停止位 1
dcbSerialParams.Parity = NOPARITY;    // 校验位 无
```

```
        if (!SetCommState(hSerial, &dcbSerialParams)) {
            CloseHandle(hSerial);
            throw std::runtime_error("无法设置串口状态");
        }
        // 设置超时
        COMMTIMEOUTS timeouts = { 0 };
        timeouts.ReadIntervalTimeout = 50;
        timeouts.ReadTotalTimeoutConstant = 50;
        timeouts.ReadTotalTimeoutMultiplier = 10;
        timeouts.WriteTotalTimeoutConstant = 50;
        timeouts.WriteTotalTimeoutMultiplier = 10;

        if (!SetCommTimeouts(hSerial, &timeouts)) {
            CloseHandle(hSerial);
            throw std::runtime_error("无法设置串口超时");
        }
    }
    // 析构函数，关闭串口
    ~SerialPort() {
        if (hSerial != INVALID_HANDLE_VALUE) {
            CloseHandle(hSerial);
```

```
        }
    }
    // 发送数据到串口
    void WriteData(const std::string& data) {
        DWORD bytesWritten;
        if (!WriteFile(hSerial, data.c_str(), data.length(), &bytesWritten, NULL)) {
            throw std::runtime_error("写入串口数据失败");
        }
     }
    // 从串口读取数据
    std::string ReadData(size_t length) {
        char buffer[1024] = { 0 };
        DWORD bytesRead;
        if (!ReadFile(hSerial, buffer, length, &bytesRead, NULL)) {
            throw std::runtime_error("读取串口数据失败");
        }
        return std::string(buffer, bytesRead);
    }
    // 发送命令控制光源的开关和亮度
    void ControlLight(int channel, bool switchOn, int brightness) {
        if (channel < 1 || channel > 4) {
```

```
            throw std::invalid_argument("通道号必须在 1 到 4 之间");
        }
        if (brightness < 0 || brightness > 255) {
            throw std::invalid_argument("亮度值必须在 0 到 255 之间");
        }
        std::ostringstream command;
        // 设置开关命令
        command << "M" << channel << "=" << (switchOn ? 1 : 0) << "\r";
        WriteData(command.str());
        // 设置亮度命令
        command.str("");  // 清空之前的命令
        command << "I" << channel << "=" << brightness << "\r";
        WriteData(command.str());
    }

private:
    HANDLE hSerial; // 串口句柄
};

int main() {
    try {
```

```
        // 创建串口对象，指定 COM1 端口
        SerialPort serialPort("COM1");
        // 控制第 1 通道：打开并设置亮度为 100
        serialPort.ControlLight(1, true, 100);
        // 控制第 2 通道：关闭并设置亮度为 0
        serialPort.ControlLight(2, false, 0);
        // 控制第 3 通道：打开并设置亮度为 150
        serialPort.ControlLight(3, true, 150);
        std::cout << "控制命令已发送" << std::endl;
    }
    catch (const std::exception& e) {
        std::cerr << "错误: " << e.what() << std::endl;
    }
    return 0;
}
```

3. 示例代码功能解析

4) 串口设置：

- 串口通过 CreateFile 打开，配置了 115200 波特率，8 数据位，1 停止位，无校验位。
- 使用 DCB 结构体来设置串口参数。
- 设置了串口的超时策略。

■ **控制光源：**

- ControlLight(int channel, bool switchOn, int brightness) 用来控制指定通道的开关和亮度。
- 通过格式化字符串生成控制命令并发送到串口。
- 每个通道的控制命令格式为 M10=1,I10=100\r，其中 M 控制开关，I 控制亮度。

■ **发送命令：**

- WriteData 方法用于将命令发送到串口，注意命令以回车符（\r）结尾。
- 使用 ostringstream 生成动态命令字符串，确保每次发送时命令格式正确。

■ **错误处理：**

如果串口打开失败、命令发送失败或参数不正确，将抛出异常并打印错误信息。

4. 如何使用

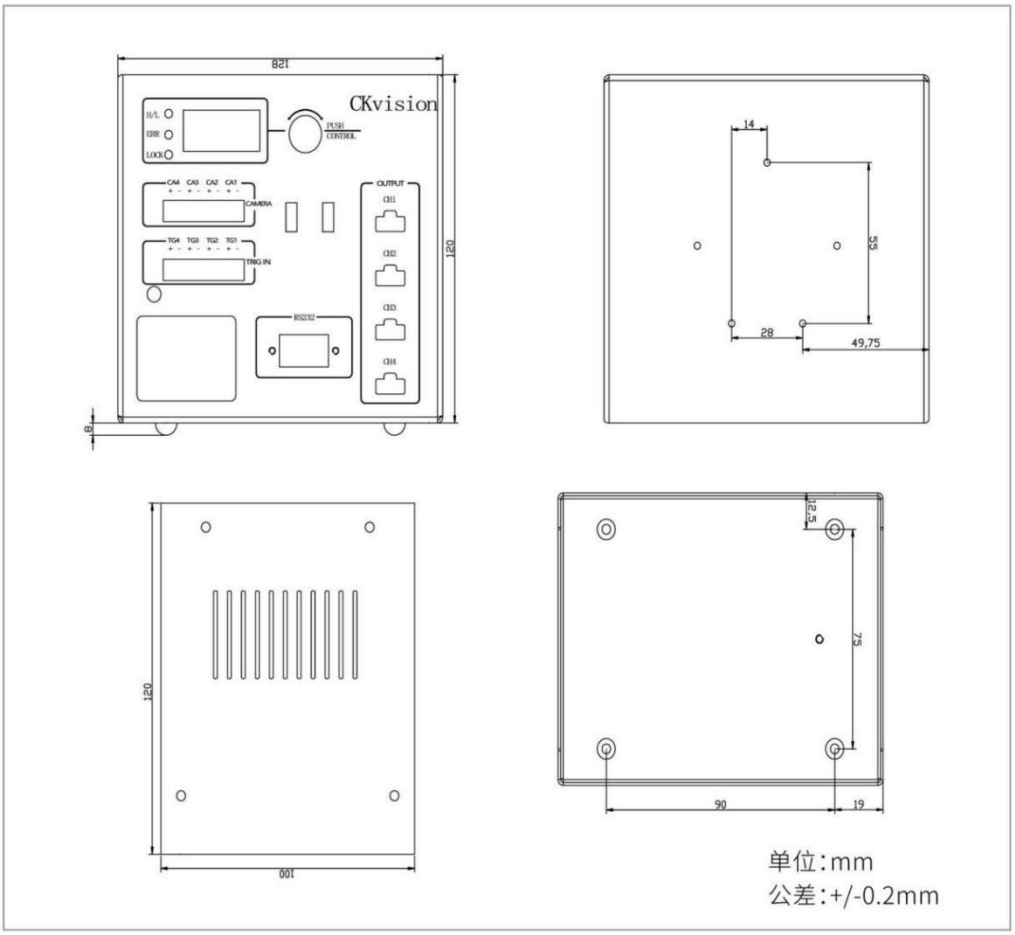
1) 确保设备已经连接到电脑的串口（如 COM1，COM2 等）。
2) 运行程序后，它将发送控制命令来控制指定通道的开关和亮度。例如：打开通道 1 并设置亮度为 100，关闭通道 2 等。
3) 你可以根据需要修改 ControlLight 的调用参数来控制不同通道的开关和亮度。

5. 拓展

- 如果需要更多的控制功能（如获取设备状态、设置其他参数等），可以在 SerialPort 类中添加相应的命令和解析方法。
- 为了提高系统的响应性，可以在读取数据时加入线程或异步操作，处理来自串口的数据反馈。

此代码示例为串口通信的基本封装，并且可以灵活扩展到其他控制应用中。

五、外 形 尺 寸


821
CKvision
120
8
14
55
28
49,75
120
100
12,5
75
90
19
单位:mm
公差:+/-0.2mm

公司简介

公司全称：深圳市创科自动化控制技术有限公司
公司总部：中国 • 深圳
成立时间：2003 成立技术团队，2005 年公司注册成立
公司愿景：成为机器视觉行业领导者。

业务范围：

智能机器视觉软件 | 智能相机 | 视觉控制器 | 工业相机 | 3D 相机 | 工业镜头 | 机器视觉光源 | Machine vision software | Smart camera | Vision controller | Industrial camera | 3D camera | Machine vision lens | Machine vision lights |

联系方式：

深圳市创科自动化控制技术有限公司
CK MACHINE VISION TECHNOLOGY CO.，LTD.

总部地址：深圳市宝安区新桥街道黄埔社区洪田路 155 号创新智慧港 1 栋 1105
华东分公司：昆山市伟业路 18 号现代广场 A2302
广州办事处：广州市番禺区东艺路金山谷意库 80 栋 603-604
惠州办事处：惠州市惠城区仲恺大道惠环段 269 号新港大厦 610
台湾办事处：新北市板桥区双十路二段 79 号 9 楼（捷运江子翠 3 号出口）
西南分公司：成都市郫都区创智南一路 38 号 2 栋 805

电话(Tel)：0755-33938281/33938283
传真(Fax)：0755-33938285
网址(http)：www.ckvision.net