

Digital Strobe Controller

CK-HDT48 Series



The CK-HDT48 series digital overdrive strobe controller is specifically designed for strobe control. The unique overdrive driving mode finds significant applications in high-speed detection fields such as semiconductor and metal size measurement, packaging foreign object identification.

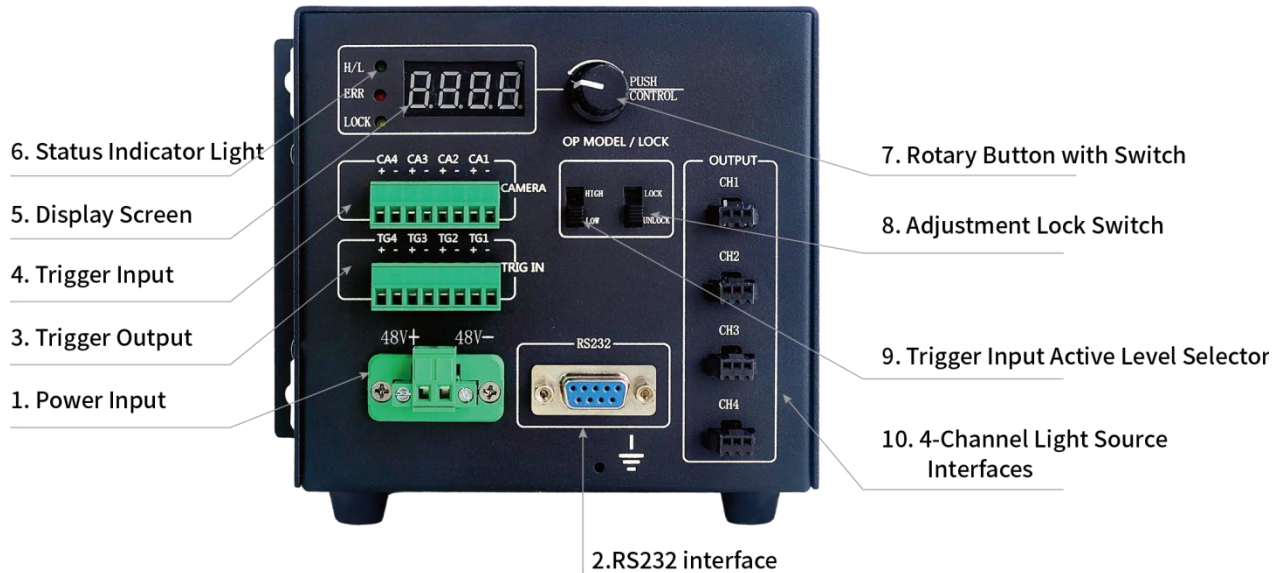
I. Features

- A. High Current Output: Each channel supports a peak current up to 3A.
- B. Brightness Control: Adjustable via a rotary knob or serial communication interface.
- C. Brightness Levels: Provides 256 brightness levels, ranging from 0 to 255.
- D. Opto-isolated External Trigger: Supports NPN/PNP trigger modes, with pulse widths as low as 10μs.
- E. Safety Features: Includes power-off memory, overload, and short-circuit protection.
- F. Based on the overdrive technology, it can increase the light source output by over 10 times compared to continuous mode, making it suitable for high-speed applications with microsecond-level exposure.
- G. We provide customized driver power supply services, with flexible specification adjustments according to requirements. The maximum power supports up to 600W, meeting the needs of diverse application scenarios.

II. Specifications

Model	CK-HDT48-200W
Drive Mode	Overdrive
Dimming Mode	256 levels PWM control
	Panel-mounted rotary knob/RS232
PWM Frequency	120KHz
Input Voltage	AC200~264V 50/60Hz
Channels	4
Output Voltage	DC 48V
Peak Output Current	3A max. (single channel)
Maximum Output Power	200W
Trigger Input Voltage	DC 5V~24V (approx.5.6mA)
Trigger Input mode	PNP/NPN/Soft trigger
Trigger Delay	30μs max. (load-dependent)
Lighting Time	Depends on the duration of the valid level
Lighting Delay	15μs max.
Operating Temp/Humidity	0~40°C, 20~85% RH (non-condensing)
Storage Temp/Humidity	-20~60°C, 20~85% RH (non-condensing)
Cooling Method	Natural cooling

III. Basic Operations



➤ Introduction to the Functional Areas of the Light Source Control Board

1. Power Input

- A. DC48V

2. RS232 Interface

- A. Connects to a serial communication cable for communication with an upper computer.
- B. Allows light brightness adjustment and soft triggering through specified instruction formats.

Refer to the communication protocol for details.

3. Trigger Output


- A. Four input ports labeled TG1, TG2, TG3, and TG4 in sequence.
- B. For each input, connect the positive signal to the "+" terminal and the negative to the "-" terminal.
- C. Upon receiving a rising edge signal (DIP switch 9 set to the upper position), the controller drives the corresponding light source and outputs a pulse signal to the Camera Trigger Input port.
- D. Drive capability: Min. 100mA.



4. **Trigger Input**

- A. Outputs corresponding pulse signals based on received input signals.
- B. Output pulse voltage: 12V.
- C. Connect the positive output to the camera's trigger input positive terminal and the negative output to the negative terminal.

5. **Display Screen**

- A. A four-digit, seven-segment display shows the channel number and corresponding brightness values.
- B. Display Pattern:
 - 1. Leftmost digit: Channel number.
 - 2. Right three digits: Brightness (0–255). 
 - 3. In idle mode, cycles through:
 - a. Current Mode → Voltage → Brightness of Channel 1 → Channel 2 → Channel 3 → Channel 4.
 - 4. In “LOCK” mode, displays “lck” if further operations are attempted.

6. **Status Indicator Light**

- A. H/L Indicator: trigger mode status (rising edge active).
- B. ERR Indicator: Lights up for system errors (e.g., short circuits in wiring).
- C. LOCK Indicator: Indicates that the rotary knob is locked and cannot adjust brightness.

7. **Rotary Button with Switch (PUSH/CONTROL)**

- A. Push to toggle channels or modes.
- B. Rotate to adjust brightness:
 - 1. Clockwise: Increases brightness.
 - 2. Counterclockwise: Decreases brightness.

8. **Adjustment Lock Switch**

- A. LOCK Position: Disables brightness adjustments, enabling trigger operation mode.
- B. UNLOCK Position: Allows channel selection and parameter adjustments using the rotary knob.

9. **Trigger Input Active Level Selector**

Switches between the active high/low level of the input signal.

10. **4-Channel Light Source Interfaces**

Four output ports labeled CH1, CH2, CH3, and CH4 in sequence, corresponding to their respective trigger inputs and outputs.

➤ Mode Introduction

1. Overdrive Strobe Mode

When a single valid trigger signal (high/low level) is received, the controller drives the light source of the relevant channel and simultaneously outputs a trigger signal to the corresponding trigger output port. To use this mode, first press the rotary knob to switch to the N000 mode (default mode).

- A. When using the PNP trigger input (active high), set the DIP switch to the "H" position.
- B. When using the NPN trigger input (active low), set the DIP switch to the middle position. In particular, in this mode, if no external trigger is connected, the light source of the corresponding channel will default to the steady-on mode, which may damage the light source—thus, it must be used with caution.
- C. Set the DIP switch to the "L" position to control the light source and trigger output via software commands.

2. Special Mode Configurations

A. N001 Mode

Set the DIP switch to the "H" position to enable this function. In the absence of an external trigger, the light source flashes periodically, rapidly and automatically, with a cycle of 1ms and a peak illumination time of approximately 180μs. This is a light source test mode—please use it with caution.

B. N002 Mode

In this mode, the light source will only turn on when the rotary knob is switched to the corresponding channel, and it will be in the steady-on mode—please use it with caution.

➤ Control Methods

■ Direct Control of the Light Source via Local Hardware

Use the rotary button with a switch located to the right of the LED display for control. Press the button to switch between light source channels and rotate it to adjust the brightness.

■ The controller supports remote operation using a communication protocol via the RS232 serial interface.

A. Protocol Features:

1. Control the on/off state of individual or multiple light source channels.
2. Adjust brightness levels for each channel.

B. Command Structure:

1. The protocol employs predefined instructions in a specific format for communication between the control system and the light source.
2. This method enables seamless integration with external systems, such as PCs or automated control setups.

1. RS232 Serial Port Configuration

To enable communication between the light source controller and an external device using the RS232 protocol, configure the serial port parameters as follows:

Parameter	Configuration
Port	COM* (User-defined)
Baud Rate	115200
Data Bits	8
Stop Bits	1
Parity Bit	None (No parity)
Control Character	ASCII Code
Termination	Carriage Return (\r)

2. Explanation of Parameters

- A. Port: Replace COM* with the specific port number assigned to the controller during setup.
- B. Baud Rate: Ensures the speed of communication matches between devices.
- C. Data Bits & Stop Bits: Define the format of the transmitted data.
- D. Parity Bit: No error-checking parity is used.
- E. Control Character: ASCII-encoded commands are used to communicate.
- F. Termination: Each command should end with a Carriage Return (\r) to signal the end of a command.

3. Communication Command Format

The light source controller receives control commands through the serial port in the following format:

A. Command Structure

1. Channel On/Off Control

Format: M<Channel Number>=<Switch State>

- a. <Channel Number>: Specifies the channel to control (e.g., 1, 2, 3, or 4).
- b. <Switch State>:
 - 0: Turns the channel off.
 - 1: Turns the channel on.

2. Channel Brightness Control

Format: I<Channel Number>=<Brightness Value>

- a. <Channel Number>: Specifies the channel to adjust (e.g., 1, 2, 3, or 4).
- b. <Brightness Value>: Sets the brightness level (range: 0 to 255).

3. Command Ending

Each command must end with a Carriage Return (\r).

4. Combining Commands

Commands can be combined using a comma (,) to control multiple channels simultaneously.

B. Example Instructions Explanation

1. Turn on Channel 1 and set its brightness to 100

Command: M10=1,I10=100\r

- M10=1: Turn on Channel 1.
- I10=100: Set the brightness of Channel 1 to 100.
- \r: End of the command (carriage return).

2. Turn off Channel 2 and set its brightness to 50

Command: M20=0,I20=50\r

- M20=0: Turn off Channel 2.
- I20=50: Set the brightness of Channel 2 to 50 (the brightness value is ignored when the channel is off).
- \r: End of the command.

3. Control multiple channels simultaneously

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

- M10=1: Turn on Channel 1.
- I10=100: Set the brightness of Channel 1 to 100.
- M20=1: Turn on Channel 2.
- I20=150: Set the brightness of Channel 2 to 150.
- Use a comma , to separate multiple commands, and end the sequence with \r.

4. Notes

A. Channel Numbers and Format

- Channel numbers must be clearly indicated with the prefix M (for switching) or I (for brightness), e.g., M10 and I10 for Channel 1.
- Ensure commands are written in the correct format without extra spaces or invalid characters.

B. Brightness Value Range

Brightness values must be between 0-255. Any value outside this range is invalid.

C. Combining Commands

Multiple commands can be combined using a comma , to control multiple channels in one instruction. Ensure all commands are properly formatted.

■ Channel Configuration

The light source controller supports up to 4 channels, numbered 1 through 4. The <Channel Number> in the instructions can take values of 1, 2, 3, or 4.



A. Channel Command Details

1. Switch Control

a. M10, M20, M30, M40: Represent the on/off state for Channels 1, 2, 3, and 4, respectively.

- M10=1: Turns Channel 1 ON.
- M10=0: Turns Channel 1 OFF.

B. Brightness Adjustment

1. I10, I20, I30, I40: Represent the brightness adjustment for Channels 1, 2, 3, and 4, respectively.

- The brightness value range is from 0 to 255 (where 0 is the lowest brightness, and 255 is the highest).

5. Important Notes

A. Unlocking the Controller

1. Before sending commands, ensure the "LOCK/UNLOCK" switch is toggled to the UNLOCK state.
2. When in the LOCK state, commands sent via RS232 are ignored.

B. Command Format

Use the specified channel numbers and formats (M10, I10, etc.) in your commands.

C. Combined Commands

You can control multiple channels simultaneously by separating commands with commas ,. Ensure proper syntax and format.

6. Examples

A. Turn ON Channel 2 and set its brightness to 200:

Command: M20=1,I20=200\r

B. Turn OFF Channel 3 and set its brightness to 50:

Command: M30=0,I30=50\r

C. Simultaneously control multiple channels (e.g., Turn ON Channel 1 and set brightness to 150, Turn OFF Channel 4):

Command: M10=1,I10=150,M40=0\r

7. Development Considerations

- A. Baud Rate Settings: Ensure the software and light source controller have matching baud rates. The default baud rate is 115200.
- B. Command Terminator: Each command must end with a carriage return (\r) to ensure correct transmission.
- C. Serial Port Configuration: Ensure the serial port parameters (data bits, stop bits, parity, etc.) are correctly set. Typically, this will be 8 data bits, 1 stop bit, and no parity.

8. Troubleshooting

A. Cannot Send Command:

1. Check Serial Connection: Verify the physical serial port connection is secure.
2. Verify Serial Port Settings: Ensure the correct baud rate and port number are configured in the serial communication tool.

B. Light Source Not Responding:

1. Check Power Supply: Ensure the light source controller is powered on.
2. Verify Command Format: Ensure the command format is correct and includes the necessary terminator (carriage return, \r).

C. Brightness Adjustment Not Effective:

1. Check Brightness Range: Ensure the brightness value in the command is within the 0-255 range.
2. Unlock the Controller: Ensure the light source controller is in the UNLOCK state to allow control. The controller will not accept commands if it is locked.

This communication protocol enables developers to control light source controllers through an RS232 serial interface, allowing for channel switching and brightness adjustments. Developers can implement and package software to enable real-time control of multiple channels.

IV. Example of writing software to control the light source controller

Here is a C++ code example based on the provided RS232 serial communication protocol, which uses the Windows API to send commands to a light source controller to control the channel switch and brightness. The command format is similar to M10=1,I10=100, used to control the switch and brightness of channel 1.

➤ RS232 Control Protocol Analysis

The RS232 control protocol is used to communicate with the light source controller via the serial port (RS232), allowing users to control the switches and brightness of the light source channels. Below is a breakdown of the protocol:

A. M Command (Channel Control)

M10=1: This command is used to control the switch of channel 1.

1. M10: Refers to the command for controlling channel 1.

2. =1: Turns channel 1 ON.
3. =0: Turns channel 1 OFF.

Example:

1. M10=1: Turns ON channel 1.
2. M10=0: Turns OFF channel 1.

B. I Command (Brightness Control)

I10=100: This command is used to set the brightness level of channel 1.

1. I10: Refers to the command for controlling the brightness of channel 1.
2. =100: Sets the brightness of channel 1 to 100, where the brightness value can range from 0 to 255.
 - a. 0: Minimum brightness (OFF or no light).
 - b. 255: Maximum brightness (full intensity).

Example:

1. I10=100: Sets the brightness of channel 1 to 100.
2. I10=255: Sets the brightness of channel 1 to the maximum (255).

C. Baud Rate

The baud rate is 115200, which means the data is transmitted at a rate of 115200 bits per second (bps). This is the speed at which data is sent over the RS232 serial connection.

D. Command Terminator

The command must end with a carriage return (CR) character: \r. This is essential for properly terminating each command and allowing the controller to recognize and process it.

➤ Example Code (Demo.cpp):

Here is an example of a C++ serial port encapsulation class using Windows API. This class allows you to send commands to control the light source controller's switch and brightness. The commands use the M and I instructions to control the channel's switch state and brightness value.

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

```
// Constructor to initialize the serial port
SerialPort::SerialPort(const std::string& portName, DWORD baudRate = CBR_115200) {
    // Open the serial port
    hSerial = CreateFile(
        portName.c_str(),          // Port name (e.g., "COM1")
        GENERIC_READ | GENERIC_WRITE, // Read/write access
        0,                        // Exclusive access
        NULL,                     // Default security attributes
        OPEN_EXISTING,           // Error if port doesn't exist
        0,                       // No overlapped mode
        NULL                      // Default template file
    );

    if (hSerial == INVALID_HANDLE_VALUE) {
        throw std::runtime_error("Unable to open serial port " + portName);
    }

    // Set up the serial port parameters
    DCB dcbSerialParams = { 0 };
    if (!GetCommState(hSerial, &dcbSerialParams)) {
        CloseHandle(hSerial);
        throw std::runtime_error("Unable to get serial port state");
    }
    dcbSerialParams.DCBlength = sizeof(dcbSerialParams);
    dcbSerialParams.BaudRate = baudRate; // Set baud rate
    dcbSerialParams.ByteSize = 8;        // 8 data bits
    dcbSerialParams.StopBits = ONESTOPBIT; // 1 stop bit
    dcbSerialParams.Parity = NOPARITY;   // No parity
    if (!SetCommState(hSerial, &dcbSerialParams)) {
        CloseHandle(hSerial);
        throw std::runtime_error("Unable to set serial port state");
    }

    // Set timeouts
    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("Unable to set serial port timeouts");
        }
    }

// Destructor to close the serial port
SerialPort::~SerialPort() {
    if (hSerial != INVALID_HANDLE_VALUE) {
        CloseHandle(hSerial);
    }
}

// Send data to the serial port
void SerialPort::WriteData(const std::string& data) {
    DWORD bytesWritten;
    if (!WriteFile(hSerial, data.c_str(), data.length(), &bytesWritten, NULL)) {
        throw std::runtime_error("Failed to write data to serial port");
    }
}

// Read data from the serial port
std::string SerialPort::ReadData(size_t length) {
    char buffer[1024] = { 0 };
    DWORD bytesRead;
    if (!ReadFile(hSerial, buffer, length, &bytesRead, NULL)) {
        throw std::runtime_error("Failed to read data from serial port");
    }
    return std::string(buffer, bytesRead);
}
```

```
// Control light source: switch on/off and set brightness
void SerialPort::ControlLight(int channel, bool switchOn, int brightness) {
    if (channel < 1 || channel > 4) {
        throw std::invalid_argument("Channel must be between 1 and 4");
    }
    if (brightness < 0 || brightness > 255) {
        throw std::invalid_argument("Brightness must be between 0 and 255");
    }
    std::ostringstream command;
    // Set switch command
    command << "M" << channel << "=" << (switchOn ? 1 : 0) << "\r";
    WriteData(command.str());
    // Set brightness command
    command.str(""); // Clear previous command
    command << "I" << channel << "=" << brightness << "\r";
    WriteData(command.str());
}

int main() {
    try {
        // Create SerialPort object with COM1 port
        SerialPort serialPort("COM1");
        // Control Channel 1: turn on and set brightness to 100
        serialPort.ControlLight(1, true, 100);
        // Control Channel 2: turn off and set brightness to 0
        serialPort.ControlLight(2, false, 0);
        // Control Channel 3: turn on and set brightness to 150
        serialPort.ControlLight(3, true, 150);
        std::cout << "Commands have been sent successfully!" << std::endl;
    }
    catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << std::endl;
    }
    return 0;
}
```

➤ Example Code Function Analysis

1. Serial Port Configuration:

A. Opening the Serial Port:

1. The serial port is opened using CreateFile, with the following configurations:
 - a. Baud rate: 115200 (bits per second)
 - b. Data bits: 8
 - c. Stop bits: 1
 - d. Parity: None

B. Parameter Configuration:

Serial port parameters are set using the DCB (Device Control Block) structure, which ensures proper communication settings.

C. Timeout Configuration:

The program uses COMMTIMEOUTS to set read/write timeout policies for the port.

2. Light Control:

A. ControlLight(int channel, bool switchOn, int brightness):

1. Controls the specified channel's switch state (on/off) and brightness (0–255).
2. It generates formatted commands and sends them to the serial port:
 - a. Example: M10=1,I10=100\r
 - M: Controls the switch (1 for ON, 0 for OFF).
 - I: Controls brightness (range: 0–255).

3. Command Transmission:

A. WriteData:

This method sends commands to the serial port. It ensures all commands end with a carriage return (\r).

B. Dynamic Command Generation:

The ostringstream class is used to dynamically construct command strings, ensuring correct formatting and easy readability.

4. Error Handling:

A. Exception Handling:

The program throws exceptions in the following cases:

1. Failure to open the serial port.
2. Incorrect parameters for channel or brightness.

3. Command transmission failure.

B. Error Display:

Errors are caught in the main function and displayed to the user.

This example demonstrates a foundational implementation of serial communication for controlling devices. Its modular design allows for flexible adaptation to other control scenarios.

V. Dimensions

