Modbus communication protocol

Definitions and info for LED-controller CTR-52

1. Hardware

The standard communication protocol provided with the LED controller CTR-52 product line is based on Modbus TCP. Modbus is a data communications protocol for use with its programmable logic controllers (PLCs).

2. Modbus Setup Information

The LED controller CTR-52 is a Modbus device that allows you to access the light settings via Ethernet and communicates using a master-slave technique in which only one device (the master) can initiate transactions (called queries). The other devices (slaves) respond by supplying the requested data to the master, or by taking the action requested in the query. The CTR-52 controller is implemented as a Modbus slave (server). The server receives messages from the master (client), processes them and responds to them. The product does not send messages by itself.

Modbus Type:	**Slave (Server)**
Modbus Format:	**Modbus TCP**
Output Data Mode:	**Auto**

3. Modbus Setup Information

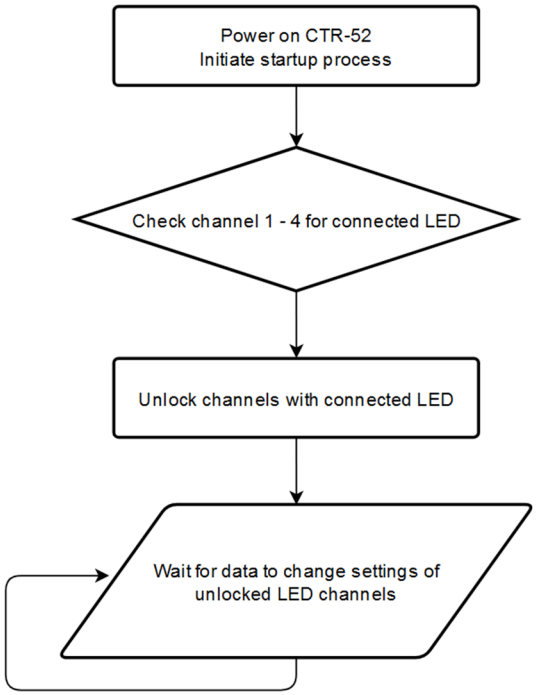

Figure 1 - Startup process CTR-52

At startup the CTR-52 looks for connected LED's. The channels with LED's connected will be unlocked and available in the following setup via modbus. **If you want to set up more LED's or other channels after starting the CTR-52 you will have to reboot the device so that the CTR-52 can detect and unlock the newly attached LED's.**

4. Modbus Register Table

3.1 General information

Field name	Description
Address	The Modbus register address.
Value range	The values which can be read or written from/to the address.
Default	The default value.
Description	Describes what's the function of the address and its values. Includes notes, where necessary.

3.2 Read discrete inputs

Address	Value range	Description
00	False, True	The alive bit toggles every 500ms to determine if the CTR-52 is connected and running.

3.3 Read discrete inputs – channel specific

Address	Value range	Description
20	False, True	Determines if light source is detected on channel 1.
30	False, True	Determines if light source is detected on channel 2.
40	False, True	Determines if light source is detected on channel 3.
50	False, True	Determines if light source is detected on channel 4.

3.4 Read / Write coil

Address	Value range	Default	Description
00	1 (Only write)	-	Restores all settings to default (does not modify EEPROM). Is set back to 0 by the controller.
02	0, 1	0	Switching on and off the voltage control for Flash-Mode.
12	1 (Only write)	-	Resets all error codes.

3.5 Read / Write coil – channel specific

Address	Value range	Default	Description
20	0, 1	0	Softwaretrigger for channel 1. Overrules hardware input only if signal level is low. Set back to 0 after processed.
21	1 (Only write)	-	Start tuning of channel 1. Tuning is the process to determine the necessary voltage for the desired current set for the LED.
22	0, 1	0	Invert the trigger-input signal of channel 1. LED will flash, when trigger is low.
30 – 32	…	…	Address 30 - 32 for control of LED channel 2. See LED channel 1 example.
40 – 42	…	…	Address 40 - 42 for control of LED channel 3. See LED channel 1 example.
50 – 52	…	…	Address 50 - 52 for control of LED channel 4. See LED channel 1 example.

3.6 Read / Write (holding) register

Address	Value range	Default	Description
00 – 01	xxx.xxx.xxx.xxx	192.168.0.99	IP address of the device. Valid after reboot when changed. 00 high byte: 99 00 low byte: 0 01 high byte: 168 01 low byte: 192
02 – 03	xxx.xxx.xxx.xxx	255.255.255.0	Subnet mask of the device. Valid after reboot when changed. 02 high byte: 0 02 low byte: 255 03 high byte: 255 03 low byte: 255
04	12345 (Only write)	-	Save current settings to EEPROM with 12345 or 0x3039.
05 – 06	10 – 590000 [µs]	20 [µs]	Minimal gaptime in µs to next trigger after flash is off. Every received trigger while gaptime is on gets ignored.
08	0, 1, 2	0	Activate flash sequence mode. Only reactive to trigger channel 1. Automatically activates flash mode on every channel. No mode-switch possible as long as activated. 0: off 1: on (Channel 1 + Channel 3, Channel 2 + Channel 4, Channel 1 + Channel 3 …) 2: on (Channel 1, Channel 2, Channel 3, Channel 4, Channel 1 …)

3.7 Read / Write (holding) register – channel specific

Address	Value range	Default	Description
20	0, 1, 2, 3, 4	0	Operation mode for channel 1. 0: Off 1: Steady (Continuous light) 2: Flash (On rising or falling edge of trigger) 3: Auto (Light follows state of trigger channel) 4: Discharge (Capacitors discharged, LED can be changed without risk)
21	0, 1, 2, 4, 8	1	Trigger masking “OR” for channel 1. Start the flash when **either** of the defined trigger channel’s state is high or low. E.g.: Respond to Trigger line 0 or 3: 1+8 = 9 **Note:** Valid for Flash-mode and Auto-mode. If set, register address 22 is set to 0.
22	0, 1, 2, 4, 8	1	Trigger masking “AND” for channel 1. Start the flash when **all** of the defined trigger channel’s state is high or low. E.g.: Respond to Trigger line 0 or 3: 1+8 = 9 **Note:** Valid for Flash-mode and Auto-mode. If set, register address 21 is set to 0.
23	50 – 30000 [mA]	100	Target current for channel 1 in mA. Gets tuned in Auto or Steady mode immediately Gets tuned in Flash mode for next flashes.
24 – 25	10 – 590000 [µs]	200	Target flash length for channel 1 in µs.
26 – 27	1500 – 21600 [mV]	0	Target voltage for channel 1 in mV. Note: Valid for Flash-mode. See coil address 02.
30 – 37	…	…	Address 30 - 37 for control of LED channel 2. See LED channel 1 example.
40 – 47	…	…	Address 40 - 47 for control of LED channel 3. See LED channel 1 example.
50 – 57	…	…	Address 50 - 57 for control of LED channel 4. See LED channel 1 example.

3.8 Read input register

Address	Value range	Description
00	4	Number of max. light channels of the CTR.
01	High byte: Major (e.g.: 3) Low byte: Minor (e.g.: 12)	Number hardware version. Major and minor release combined.
02	High byte: Major (e.g.: 1) Low byte: Minor (e.g.: 7)	Number firmware version. Major and minor release combined.
03	High byte: Major (e.g.: 1) Low byte: Minor (e.g.: 0)	Number IO protocol. Major and minor release combined.
04	High byte: Year Low byte: Week	Production week (batch number). E.g.: 20 35 (Year 2020, Calendar week 35)
05	Bit 1: Error channel 1 Bit 2: Error channel 2 Bit 3: Error channel 3 Bit 4: Error channel 4 Bit 8: Error DAC	Error code. If present code can be broken down via input register 24.
08	Bit 0: Trigger 1 … Bit 3: Trigger 4	Hardware trigger input status.
09	50	Min. current possible for each channel in mA.
10	1500	Max. current possible for each channel in mA in continuous or auto mode.
11	30000	Max. current possible for each channel in mA in flash mode.
12	3	Min. flash time in µs.
13 – 14	59000000	Max. flash time in µs.
15	-	Max. capacity per channel in flash mode in mAs (current * flash time)
16	-	Number of boot sequences since first start.

3.9 Read input register – channel specific

Address	Value range	Description
20	XX	LED current channel 1 in mA. In steady mode: live measured current In flash mode: measured current of last flash
21	XX	Last flash time channel 1 in µs.
22	XX	Charging voltage of the capacitors channel 1 in mV. In steady mode: live measured voltage In flash mode: measured voltage of last flash
23	0: tuning never started since boot 1: tuning in progress 2: last tuning successful 3: last tuning failed	Tuning status of channel 1.
24	0x00 (0): no error 0x80 (128): current not reached 0xff (255): short circuit	Error code for channel 1 if present.
20 – 24	…	Address 20 - 24 for control of LED channel 2. See LED channel 1 example.
30 – 34	…	Address 30 - 34 for control of LED channel 3. See LED channel 1 example.
40 – 44	…	Address 40 - 44 for control of LED channel 4. See LED channel 1 example.

5. Errorcodes

Permanent off:	**No LED attached or channel is discharged.**
Permanent on:	**Everything ok, LED detected and tune process was successful.**
Blinking:	**An error has occurred on this channel. The blink code remains until the device is power cycled, even if the problem is corrected.**

6. Python scripts

If you'd like to include the control of the CTR-52 in your own software code then please use the python scripts to be found on our website https://www.mbj-imaging.com/produkte/led-controller as examples on how to set up the controller the right way.

The scripts are using two modules:
Time, **Struct** & **pyModbusTCP**

The module pyModbusTCP communicates via the modbus read/write codes with the controller.
Functions are defined to read/write the data necessary to set up the LED's current, it's mode, trigger behavior and sequence.
Testcase functions at the end are showing the order in which the commands have to be sent.
The following shows the workwise of the script with PyCharm 2023.1.3 and the order in which the commands should be sent.
The script is known to be working with Python version 3.11.

```
import struct
import time
from pyModbusTCP.client import ModbusClient

# define some constants for the test
ipaddr = '192.168.0.99'  # Standard address is 192.168.0.99
c = ModbusClient(host=ipaddr, port=502, auto_open=True, auto_close=True, debug=False)
```

Figure 2 - Import and initialization

Modules get imported. The default IP address is set to "192.168.0.99". You can change it via the function "change_ip_address".

```
# Coils general (Function code read 01 / write 05)
coil_restoresettings = 0
coil_flash_voltagemode = 2
coil_reseterrors = 12
# Coils channel specific (Function code read 01 / write 05)
coil_channel_sw_trigger = 20
coil_channel_tune_current = 21
coil_channel_invert_trigger = 22

# Discrete channel specific (Function code read 02)
discrete_channel_lightdetected = 20

# Holding general (Function code read 03 / write 06)
holding_ipadress = 0
holding_ipadress_low = 1
holding_subnetmask = 2
holding_storevalues = 4
holding_gaptime_low = 5
holding_gaptime_high = 6
holding_sequencemode = 8
```

Figure 3 - Address definition

All modbus addresses are defined. Those will be used by the functions and the pyModbusTCP module.

```python
def channel_write_pulselength(channel, pulselength):
    c.write_multiple_registers(holding_channel_flashlength + channeloffset * channel, [0, pulselength])
    print('Write Channel', channel, 'pulselength:', pulselength, 'us')
    time.sleep(0.2)


def channel_read_pulselength(channel):
    flashlength = c.read_holding_registers(holding_channel_flashlength + channeloffset * channel, 1)
    return flashlength
```

Figure 4 - Examples functions

“Read”-functions will generally return the value that was read out from the register address.
“Write”-functions will write the desired value e.g. the pulselength to the specified channel. If the function is general – meaning it is valid for all channels automatically – there will be no channel to specify.

```python
def testcase_steady():
    deactivate_voltagemode()

    channel_change_mode(0, 0)
    channel_change_mode(1, 0)
    channel_change_mode(2, 0)
    channel_change_mode(3, 0)

    channel_change_mode(0, 1)
    channel_change_mode(1, 1)
    channel_change_mode(2, 1)
    channel_change_mode(3, 1)
```

Figure 5 - Testcase

The testcase-functions define the order in which the commands should be sent to the controller. Always initiate a zero state (channel_change_mode(x, 0)). This ensures that the controller switches the modes correctly.

```python
if __name__ == '__main__':
    # testcase_flashmode()
    complete_information()
    # change_ip_address()
    # store_all_values()
    # restore_settings()
    testcase_steady()
    # reset_errors()
    # testcase_steady()
    # testcase_voltagemode_flash()
    # testcase_sequence()
```

Figure 6 - Call functions and run script

If you are happy with the values to be given to the controller you may run the script. You can store the values at the end so the controller will safe the complete value set. Those values will remain after a reboot of the device.

If something went wrong you may reset errors or restore settings.

7. History

Version	Date	Changes
1.00	04.07.2023	Initial version. Added python script explanations and flowchart.