IOL HAT Software

Application manual

www.pinetek-networks.com

Printed on July 1, 2024

Table of Contents

SW1 – Introduction

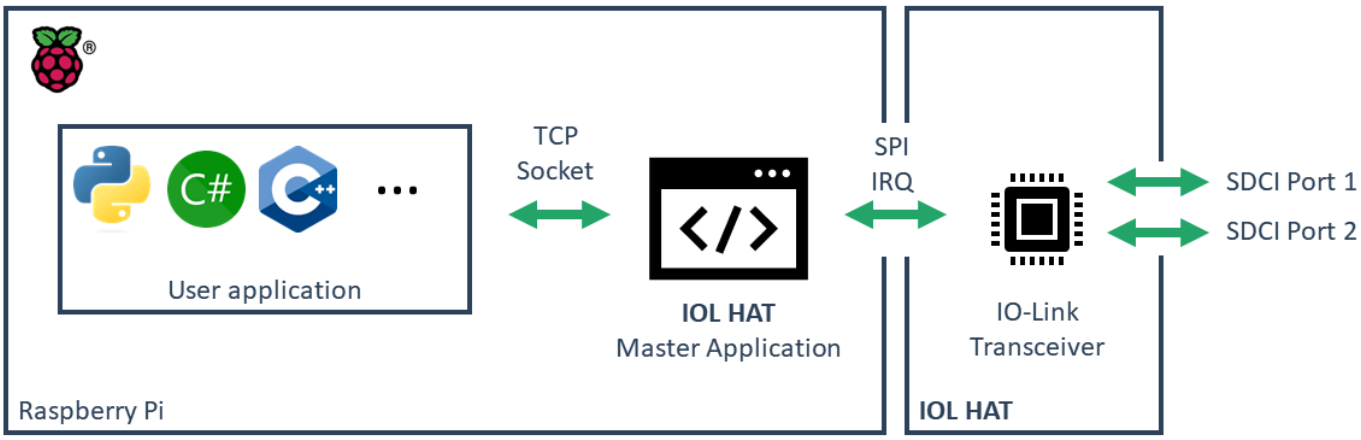
This section describes the methods and procedures that are used in connection with the open source software for the IOL HAT (“master application”) which is available in the Github repository: https://github.com/Pinetek-Networks/iol-hat

The software is based on the I-Link stack by RT-Labs. Please note that the software has been proven to work in different applications, however, it comes without guarantees of specific functions or functionality.

Information on how to obtain the software and how to build it for specific targets can be found in the GitHub. The following sections describe the application of the software on the Raspberry Pi or other Single Board Computers.

Theory of operation

The master application handles the raw communication to the MAX14819 transceiver on the IOL-HAT over SPI and provides the stack routines to operate the IO-Link communication. The master application offers a TCP socket interface to the user application for easy integration. The master application and user application need to be run independantly.



SW2 – Master Application

To operate the IOL HAT on a Raspberry Pi, the master application binary from the IOL HAT repository is used:
https://github.com/Pinetek-Networks/iol-hat/bin

The suitable master application file for the used Linux distribution (32bit/64bit) shall be used. The Linux version can be determined with

```
uname -a
```

If the Linux distribution is not available as binary, please refer to the building instructrions on the GitHub page to compile a matching version: https://github.com/Pinetek-Networks/iol-hat/tree/main/src-master-application

The settings how to operate the application are described on the GitHub page.

To operate two IOL HATs on one Raspberry Pi, two instances of the master application need to be executed. Only run one instance of the same master application at once, otherwise the behavior will be undefined.

SPI communication needs to be enabled on the host. For Raspberry Pi, this can be set using the raspi config tool:

```
sudo raspi-config
```

Upload the master application to the Raspberry Pi and run it. The master application will open a TCP server on the following ports:

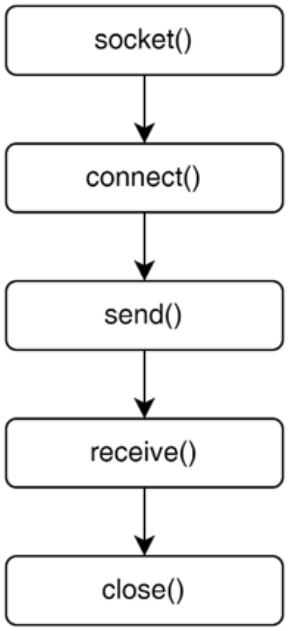
- SDCI 1+2: Port 14001
- SDCI 3+4: Port 14002

The client can connect to those ports and communicate through the binary prototcol as described below.

The TCP connection is established for each communication with the master application, i.e., the connection is not kept open.

SW3 – Binary protocol

The sequence for communication over TCP is as follows:



The commands and responses are exchanged in the send() and receive() part of the sequence. The first octet in the command structure always defines the command ID.

Some commands use a port ID. This refers to the first or second port on the IOL HAT. Which IOL HAT is addressed is determined through the TCP port. Port ID 1 can refer to IOL HAT port 1 or 3, Port ID 2 to IOL HAT port 2 or 4, depending if TCP port 14001 or 14002 is used.

The commands have a common structure in the first two octets:

Octet #	Usage
0	Command ID
1	Length

In case of a command success, the return message as described in the commands is returned by the TCP server.

Error handling

In case of an error, the error message is returned:

Octet #	Usage
0	Error message ID = 0xFF
1	Error code

The following error codes are defined:

Error code	Usage
0x01	Message Length Error
0x02	Function ID unknown
0x03	Port power error, e.g. read while port disabled
0x04	Port ID error, i.e. port ID >2 called
0x05	Internal error, in this case octets 2..3 define the specific error code from the I-Link stack
0x06	Wrong status, e.g. data exchange when power off or no connection to the device

The error codes are listed in the appendix SW-Apendix-A.

SW3.1 – CMD_PWR

The power command switches the L+ power on the corresponding port. After switching on, the communication speed on the SDCI (COM1, COM2, COM3) is automatically detected and cyclic exchange of data is started.

Command:

Octet #	Usage
0	Command ID = 0x01
1	Port ID 0x00 or 0x01
3	Power status 0x00 = OFF 0x01..0xFF=ON

Return Success:

Octet #	Usage
0	Command ID = 0x01
1	Port ID 0x01 or 0x02
3	Power Status

SW3.2 – CMD_LED

The LED command switches the LED on the corresponding port.

Command:

Octet #	Usage
0	Command ID = 0x02
1	Port ID 0x00 or 0x01
3	LED Status (see below)

The LED status is 0x01 for the green LED and 0x02 for the red LED. With LED status 0x03, both LEDs are activated.

Return Success:

Octet #	Usage
0	Command ID = 0x02
1	Port ID 0x01 or 0x02
3	LED Status

SW3.3 – CMD_PD

This command exchanges the process data with the connected device. Process data is not validated in length or content.

Command:

Octet #	Usage
0	Command ID = 0x03
1	Port ID 0x00 or 0x01
2	Length Out
3	Length In
4..	Data Out

The ***Length Out*** field command is length for the output process data. The structure and length of the output data can be found in the SDCI device's documentation. The ***Length In*** field needs to be set according to the device's documentation. If the length fields are not correctly set, this may lead to invalid data or loss of communication.

Return Success:

Octet #	Usage
0	Command ID = 0x03
1	Port ID 0x01 or 0x02
2	Length Out (as given in the command)
3	Length In (as given in the command)
4..	Data In

SW3.4 – CMD_READ

This command reads a parameter with the index and subindex as given on the corresponding port.

Command:

Octet #	Usage
0	Command ID = 0x04
1	Port ID 0x00 or 0x01
2..3	Index, 16-bit value
4	Subindex
5	Length

The ***Length*** field is the desired/maximum length for the attribute. The structure of the attribute can be found in the SDCI device's documentation.

Return Success:

Octet #	Usage
0	Command ID = 0x0
1	Port ID 0x01 or 0x02
2..3	Index, 16-bit value
4	Subindex
5	Read Length
6..	Read Data

The structure for the Read Data can be found in the SDCI device's documentation.

SW3.5 – CMD_WRITE

This command writes a parameter with the index and subindex as given on the addressed port.

Command:

Octet #	Usage
0	Command ID = 0x05
1	Port ID 0x00 or 0x01
2..3	Index, 16-bit value
4	Subindex
5	Length
6..	Write Data

The structure of the ***Write Data*** can be found in the SDCI device's documentation.

Return Success:

Octet #	Usage
0	Command ID = 0x04
1	Port ID 0x01 or 0x02
2..3	Index, 16-bit value
4	Subindex
5	Length (as given in the command)

To verify the write operation, a read operation (Command ID 0x04) can be used on the same ***Index*** and ***Subindex***.

SW3.6 – CMD_STATUS

The status command returns the status of the addressed port.

Command:

Octet #	Usage
0	Command ID = 0x06
1	Port ID 0x01 or 0x02

Return Success:

Octet #	Usage	Comment
0	Command ID = 0x06	
1	Port ID 0x00 or 0x01	
2	Process data IN valid	0x00 = Not valid 0x01 = Valid
3	Process data OUT valid	
4	Transmission rate	0x00 = Detection Failure 0x01 = COM1 (4.8 kbps) 0x02 = COM2 (38.4 kbps) 0x03 = COM3 (230.4 kpbs)
5	Process data IN length	
6	Cycle time	
7	Process data OUT length	
8..9	Vendor ID	16-bit value
10..13	Device ID	32-bit value
14	Power	0x00 = Power OFF 0x01 = Power ON

SW-Appendix-A – Error messages

The following extended Error Messages are defined (see iolink.h in application source code):

```
IOLINK_SMI_ERRORTYPE_NONE = 0x0000,

/* Table C.1 ErrorTypes */
IOLINK_SMI_ERRORTYPE_APP_DEV                = 0x8000,
IOLINK_SMI_ERRORTYPE_IDX_NOTAVAIL           = 0x8011,
IOLINK_SMI_ERRORTYPE_SUBIDX_NOTAVAIL        = 0x8012,
IOLINK_SMI_ERRORTYPE_SERV_NOTAVAIL          = 0x8020,
IOLINK_SMI_ERRORTYPE_SERV_NOTAVAIL_LOCCTRL  = 0x8021,
IOLINK_SMI_ERRORTYPE_SERV_NOTAVAIL_DEVCTRL  = 0x8022,
IOLINK_SMI_ERRORTYPE_IDX_NOT_ACCESSIBLE     = 0x8023,
IOLINK_SMI_ERRORTYPE_PAR_VALOUTOFRNG        = 0x8030,
IOLINK_SMI_ERRORTYPE_PAR_VALGTLIM           = 0x8031,
IOLINK_SMI_ERRORTYPE_PAR_VALLTLIM           = 0x8032,
IOLINK_SMI_ERRORTYPE_VAL_LENOVRRUN          = 0x8033, /* Also in C.2 */
IOLINK_SMI_ERRORTYPE_VAL_LENUNDRUN          = 0x8034,
IOLINK_SMI_ERRORTYPE_FUNC_NOTAVAIL          = 0x8035,
IOLINK_SMI_ERRORTYPE_FUNC_UNAVAILTEMP       = 0x8036,
IOLINK_SMI_ERRORTYPE_PAR_SETINVALID         = 0x8040,
IOLINK_SMI_ERRORTYPE_PAR_SETINCONSIST       = 0x8041,
IOLINK_SMI_ERRORTYPE_APP_DEVNOTRDY          = 0x8082,
IOLINK_SMI_ERRORTYPE_UNSPECIFIC             = 0x8100,

/* Table C.2 Derivced ErrorTypes */
IOLINK_SMI_ERRORTYPE_COM_ERR           = 0x1000,
IOLINK_SMI_ERRORTYPE_I_SERVICE_TIMEOUT = 0x1100,
IOLINK_SMI_ERRORTYPE_M_ISDU_CHECKSUM   = 0x5600,
IOLINK_SMI_ERRORTYPE_M_ISDU_ILLEGAL    = 0x5700,

/* Table C.3 SMI related ErrorTypes */
IOLINK_SMI_ERRORTYPE_ARGBLOCK_NOT_SUPPORTED   = 0x4001,
IOLINK_SMI_ERRORTYPE_ARGBLOCK_INCONSISTENT    = 0x4002,
IOLINK_SMI_ERRORTYPE_DEV_NOT_ACCESSIBLE       = 0x4003,
IOLINK_SMI_ERRORTYPE_SERVICE_NOT_SUPPORTED    = 0x4004,
IOLINK_SMI_ERRORTYPE_DEV_NOT_IN_OPERATE       = 0x4005,
IOLINK_SMI_ERRORTYPE_MEMORY_OVERRUN           = 0x4006,
IOLINK_SMI_ERRORTYPE_PORT_NUM_INVALID         = 0x4011,
IOLINK_SMI_ERRORTYPE_ARGBLOCK_LENGTH_INVALID  = 0x4034,
IOLINK_SMI_ERRORTYPE_SERVICE_TEMP_UNAVAILABLE = 0x4036,
```