FP451x

Communications Protocol

Rev. 0v0

Freedom Photonics LLC
41 Aero Camino
Santa Barbara, CA 93117
(805)967-4900

www.freedomphotonics.com

Contents

Introduction

The protocol used by the FP451x series of laser controllers is loosely based on the OIF-ITLA-MSA-01.1 protocol developed for ITLAs, but they are not fully backwards compatible. The controllers are equipped with an RS232 port for communicating with a host (typically a PC using a USB<->RS232 converter). The protocol is a non-ASCII based command-response structure where the host issues a command and the controller always responds, regardless of the action taken. The host must enforce the avoidance of interleaving multiple commands – no additional commands can be issued until the previous response has been received. Each individual command is either a read from or a write to a 16-bit register. An Extended Addressing (EA) mechanism is provided for expanding the reads and writes to multi-byte arrays or strings, but the overhead of 16-bits (or two bytes) per transaction remains. This will be further explained in a later section.

Serial Port Configuration

The FP451x serial port uses the following configuration:

9600bps, 8 data bits, no parity, 1 stop bit, no flow control

Command Structure

Commands are four-byte data packets that are sent from the host to the controller. Byte 0 is the first byte transmitted and Byte 3 is the last byte transmitted. Though Byte 0 is the first transmitted, its value is last determined as it is based on the values of the other three bytes. For register read operations, Byte 2 and Byte 3 should both be 0x00.

Command Data Packet			
Byte 0	Byte 1	Byte 2	Byte 3
Checksum & R/W Bit	Register Address	Write Value (MSB)	Write Value (LSB)

The tables below describe the individual bits of the four-byte command packet. The bit positions have been numbered from the perspective of the complete packet.

Byte 0							
31	30	29	28	27	26	25	24
CHECKSUM				0	0	0	R/W

Byte 1							
23	22	21	20	19	18	17	16
Register Number							

Byte 2							
15	14	13	12	11	10	9	8
Data [15:8]							

Byte 3							
7	6	5	4	3	2	1	0
Data [7:0]							

Byte 0: LSB of data to be written. Should be 0x00 for register read operations.

Byte 1: MSB of data to be written. Should be 0x00 for register read operations.

Byte 2: The address of the register to be read from or written to.

Byte 3: [24] is read/write bit. Read = 0, Write = 1.

[27:25] always 0x0

[31:28] is checksum

The checksum is present for a basic message consistency check. It is a four-bit field calculated over the entire command packet. The packet is initially formatted with all required fields populated and bits [28:31] = 0x0. Each of the four bytes is then xored together resulting in an 8-bit value. Finally, this result is split into high and low nibbles which then get xored together. This is the final 4-bit value that is placed in the checksum field. The following code excerpt performs this calculation on an already formatted packet which is passed by reference:

```
UINT_8 OIFcalcBIP4(UINT_8 *data)
{
 UINT_8 bip8 = ((data[0] & 0x0F) ^ data[1] ^ data[2] ^ data[3]);
 return (((bip8 & 0xF0) >> 4) ^ (bip8 & 0x0F));
}
```

Response Structure

Once the command has been received from the host, the controller will perform a message consistency check by computing the checksum and verifying it against the checksum specified by the host. If the two are consistent, the controller assumes the message to be valid and continues to process it. If there is a mismatch, the controller will discard the message and respond to the host with an error indicating that is the case. If processing proceeds, the controller will verify that the register specified exists, that the read or write operation is allowed, and in the case of a register write, that the value being written is

within the range of acceptable values. Once the action is complete, the controller will respond to the host in the following format:

Response Data Packet			
Byte 0	Byte 1	Byte 2	Byte 3
Checksum & Status	Register Address	Value (MSB)	Value (LSB)

The tables below describe the individual bits of the four-byte response packet. The bit positions have been numbered from the perspective of the complete packet.

Byte 0							
31	30	29	28	27	26	25	24
CHECKSUM				CE	0x1	Status	

Byte 1							
23	22	21	20	19	18	17	16
Register Number							

Byte 2							
15	14	13	12	11	10	9	8
Data [15:8]							

Byte 3							
7	6	5	4	3	2	1	0
Data [7:0]							

The response checksum is calculated the same as the command checksum. Byte 0[27] is a communications error flag (CE) that the controller uses to notify the host in the event that there was a checksum mismatch in the last command received. The host should always check to see that the value of the bit is 0x0 after verify the response checksum. If this bit is set, the host should assume that the last command was discarded and re-send it. Byte 0[25:24] are status bits to communicate the execution status of the command being responded to. The following values are possible:

Byte 0[25:24] = 0x00 – The command was executed and the response data (if any) is valid

Byte 0[25:24] = 0x01 – An execution error occurred. The command was not executed and any response data is invalid.

Byte 0[25:24] = 0x02 – EA flag. The command was executed and the response data is accessible through the EA mechanism. In this case, the data returned in this response is the number of bytes to be accessed through EA.

Execution Error Determination

If the host detects an execution error in response to an issued command (Byte 0[25:24] = 0x01), further module interrogation is required to determine the cause of the error assertion. First, the host should read NOP register (address 0x00). By inspecting bits [3:0], the following possible execution errors can be determined:

NOP[3:0] Value	Error Symbol	Interpretation
0x00	OK	No errors present
0x01	RNI	The addressed register is not implemented
0x02	RNW	The addressed register cannot be written
0x03	RVE	Register value written is out of range
0x04	CIP	Command ignored due to pending operation
0x05	CII	Command ignored while controller is initializing
0x06	ERE	Extended address invalid
0x07	ERO	Extended address is read only
0x08	EXF	Execution general failure
0x09	--	
0x0A	IVC	Invalid configuration
0x0B - 0x0E	--	
0x0F	--	

Extended Addressing

In the event that a register read operation returns a result through the EA mechanism (as indicated by response Byte 0[25:24] = 0x02), the data contained within the response is not the contents of the register being read, it is instead the byte count of the response to be read out through the EA data register. If this data is in the format of a string, the string will be null terminated and the count will include the termination character.

Upon receiving the EA flag, the host must read out the number of bytes specified by the controller in the response data two-bytes at a time through the EA data register (address 0x0B). After each read, the internal memory address accessed through the EA data register is automatically incremented. This allows for a sequence of reads of the EA data register with no other actions in between to extract a multi-byte data value. If the host attempts to read the EA data register beyond the byte count returned by the originating command's response, an execution error will result.

For an example of a register read operation utilizing the EA mechanism, let's consider reading the "MODEL" register (0x03) of the FP4516 laser controller. The host first issues a command to read register

address 0x03. The module response contains Byte 0[25:24] = 0x02 specifying an EA operation is required and the data value returned is 0x0008 for the null-terminated string “FP 4516”. The host will then read the EA data register (0x0B) four consecutive times to extract the full eight bytes (characters).

The table that follows shows the sequence of register operations for performing the register read. Host commands and controller responses are shown.

	Host Command			Controller Response		
	R/W	Address	Data[15:0]	Status	Address	Data[15:0]
1	R	0x03 (MODEL)	0x0000	0x02 (EA)	0x03 (MODEL)	0x0008
2	R	0x0B (EA Data)	0x0000	0x00 (OK)	0x0B (EA Data)	0x4650 (“FP”)
3	R	0x0B (EA Data)	0x0000	0x00 (OK)	0x0B (EA Data)	0x2034 (“ 4”)
4	R	0x0B (EA Data)	0x0000	0x00 (OK)	0x0B (EA Data)	0x3531 (“51”)
5	R	0x0B (EA Data)	0x0000	0x00 (OK)	0x0B (EA Data)	0x3600 (“6/0”)

Register Address Map

Address	Name	R/W	EA	Description
0x00	NOP	R/W		No operation on write, [3:0] are error field described above
0x01	DEVTYPE	R	EA	Returns the device type as a null terminated string
0x02	MFGR	R	EA	Returns the manufacturer as a null terminated string
0x03	MODEL	R	EA	Returns the model number as a null terminated string
0x04	SERNO	R	EA	Returns the serial number as a null terminated string
0x06	RELEASE	R	EA	Returns the firmware release information as a null terminated string
0x0B	EADATA	R/W		EA Data register
0xA8	GNI	R/W		Sets/Gets gain current in 10uA steps
0xA9	PHI	R/W		Sets/Gets phase current in 10uA steps
0xAA	FMI	R/W		Sets/Gets front mirror current in 10uA steps
0xAB	BMI	R/W		Sets/Gets back mirror current in 10uA steps
0xAC	SAI	R/W		Sets/Gets SOA current in 10uA steps
0xB0	LSRTECI	R		Reads laser TEC current. Itec = -(VALUE/26214 – 1.5) * 2.5
0xB1	BMV	R		Reads back mirror operating voltage. V = VALUE * 5.02196E-5
0xB3	GNV	R		Reads gain operating voltage. V = VALUE * 5.02196E-5
0xB4	CSETECI	R		Reads case TEC current. Itec = -(VALUE/26214 – 1.5) * 2.5
0xB7	FMV	R		Reads front mirror operating voltage. V = VALUE * 5.02196E-5
0xB8	SAV	R		Reads SOA operating voltage. V = VALUE * 5.02196E-5
0xB9	PHV	R		Reads phase operating voltage. V = VALUE * 5.02196E-5
0xE0	LSRTMPSP	R/W		Sets/Gets laser temperature setpoint in 0.01 °C steps
0xE1	LSRTMPOP	R		Gets laser temperature operating point in 0.01 °C steps
0xE3	CSETMPSP	R/W		Sets/Gets case temperature setpoint in 0.01 °C steps
0xE4	CSETMPOP	R		Gets case temperature operating point in 0.01 °C steps
0xE5	LSRTECEN	R/W		Laser TEC enable. All non-zero values enable TEC

0xE6	CSETECEN	R/W		Case TEC enable. All non-zero values enable TEC

Please note that additional registers exist for manufacturing purposes and attempts to read from or write to these undocumented addresses may result in undefined behavior.