I2C Interface API

Implementation of the I^2C interface for communicating with the Pump Diode Board of the FFC-CM has been left to the user. However, a brief explanation of the format and implementation has been provided. Feel free to contact Vescent for support at:

info@vescent.com

Links

Here is a link
Here is a another

I2C interface

Communication with the Gen2 Laser Driver boards is achieved using a 2 wire I2C interface. The Gen2 Laser Driver board firmware contains metadata which can be extracted to expose all available commands by using a sequence of device enumeration commands described in this document. The Gen2 Laser Driver is compatable with I2C Fast Mode 400 kbit/s speed and uses 7 bit addressing with the address determined by the board's installation position on the pump laser board.. Little Endian (least significant byte first) data format is used for multi-byte data types.

I2C Addresses

The I2C address for the Gen2 Laser Driver board is determined by it's position on the pump laser board. The address is fixed and cannot be set in the firmware.

I2C Format

Commands are between 2 and 8 bytes use the following format:
Byte 0 = I2C Address
Byte 1 = Command Index Number
Bytes 2 – 7 = Argument Bytes
Bytes 2 through 7 are used for parameters in the order defined for the command. Unsigned 16 bit integer and Floating point values must be in Little Endian format (Least Significant Byte First)

I2C Command Metadata

The first two API commands can be used to extract command metadata for all Laser Driver board commands.
The command Metadata for each command contains the following information:

1) ASCII command string – Up to 8 ASCII characters which can be used to allow communication

through an optional Serial to i2c translation program which can be written to facilitate control through a terminal program such as PuTTY or Tera Term.

2) Command Index - 1 byte integer value used for identifying the command

3) Number of argument bytes – 1 byte integer value

4) Parameter type byte – 1 byte integer value defined in the next section

5) Return type – 1 byte integer value defined in the next section

Extracting the metadata is a two step process which requires the use of the first two API commands.
1) The first Command obtains the device type and the number of commands available in the API I2C format: (“ENUMDEV” is the ASCII identifier 0 is the I2C command index)

Byte 0 = I2C Address
Byte 1 = 0

The return value from this command contains 8 bytes. Bytes 0 and 1 are the important ones:

Byte 0 = device type (Gen2 Laser Driver is type 15)
Byte 1 = number of commands
2) The second command is called twice iteratively for each of the number of commands returned by the ENUMDEV command.

First Call:
I2C format: (“_ENUMCMD” is the ASCII identifier 1 is the I2C command index)
Byte 0 = I2C Address
Byte 1 = 1 (“ENUMCMD” command index)
Byte 2 = command index for the metadata requested
Byte 3 = **0** (Defines which return information to return for the command)

The return value from the first iteration is:
Byte 0 = command index
Byte 1 = number of parameter bytes the command expects
Byte 2 = The types of parameters expected (See Parameter Types section below)
Byte 3 = The data type of the return value.

Second Call:
I2C format: (“_ENUMCMD” is the ASCII identifier 1 is the I2C command index)
Byte 0 = Address
Byte 1 = 1 (“ENUMCMD” command index)
Byte 2 = command index for the metadata requested
Byte 3 = **1** (Defines which return information to return for the command)

The return value from the second iteration is:
The ASCII command string (up to 8 bytes)

I2C Parameter Types

Byte 3 of the return from the first iteration of the ENUMCMD command defines the data type of any parameters used by the command. Two bits of the byte are used to identify the data type in each

position of the command. These definitions are or'd together to allow up to 4 parameters to be defined within the same byte. I.e. A command that uses a one byte channel parameter followed by a 4 byte floating point parameter will use (CMD_ARG0_UINT8 | CMD_ARG1_FLOAT) for the byte 3 parameter type definition. The parameters are defined by their position using the two bit values shown here:

```
#define #define CMD_ARG0_UINT16 (CMD_UINT16 « 6)
#define CMD_ARG0_INT16 (CMD_INT16 « 6)
#define CMD_ARG0_FLOAT (CMD_FLOAT « 6)

#define CMD_ARG1_UINT8 (CMD_UINT8 « 4)
#define CMD_ARG1_UINT16 (CMD_UINT16 « 4)
#define CMD_ARG1_INT16 (CMD_INT16 « 4)
#define CMD_ARG1_FLOAT (CMD_FLOAT « 4)

#define CMD_ARG2_UINT8 (CMD_UINT8 « 2)
#define CMD_ARG2_UINT16 (CMD_UINT16 « 2)
#define CMD_ARG2_INT16 (CMD_INT16 « 2)
#define CMD_ARG2_FLOAT (CMD_FLOAT « 2)

#define CMD_ARG3_UINT8 CMD_UINT8
#define CMD_ARG3_UINT16 CMD_UINT16
#define CMD_ARG3_INT16 CMD_INT16
#define CMD_ARG3_FLOAT CMD_FLOAT
```

Where the bits are set according to these **Basic Types**:

```
#define CMD_UINT8 0x00
#define CMD_UINT16 0x01
#define CMD_INT16 0x02
#define CMD_FLOAT 0x03
#define CMD_RAW 0x04 (string of up to 8 bytes)
#define CMD_STATUS 0x05
#define CMD_ASCII 0x06
#define CMD_TEST 0x07
#define CMD_UINT32 0x08
#define CMD_NO_ARGS 0xFF
#define CMD_NONE 0xFF
```

Note **Basic Types** define the return type