Version 0.04 20/11/2007 16:34

LASER->USBDAQ communication

P. Dauncey[1], M. Stanitzki[2], G. Villani[2]

1. Introduction

This Document outlines the communication between the Laser TestStand Software written in LabView and the Software controlling MAPS USB_DAQ written in C++. The communication is realized using TCP/IP by sending fixed-size messages via an Ethernet connection

2. Network Setup

As already been stated, it was decided to use the TCP/IP protocol for the communication between the two software packages. The default port to be used is 15000 (well above the 1024 privileged port limitation), but can also be configured to any port beyond 1024. In order to be free of endian-ness problems and to make the LabView implementation easier, it was decided to use strings. To ease the LabView programming, the message frame size is fixed to 36 bytes for all messages

3. Configurable Parameters

3.1 By an Laser Operator

- Shutter size x (0-100) 3 bytes
- Shutter size y (0-100) 3 bytes
- Software Version 10 bytes (Arbitrary string)

3.2 By the DAQ

- Laser Intensity (0-100) 3 bytes
- Laser position x (-1000 - 1000) 7 bytes (6 bytes for the value, 1 for the sign)
- Laser position y (-1000 - 1000) 7 bytes (6 bytes for the value, 1 for the sign)
- Laser Fire Mode 1 byte
    - Continuous 0
    - Single pulse 1
    - Burst 2
- Laser Trigger Mode 1 byte
    - 0 Hardware TTL triggered by DAQ
    - 1 Software Triggered
- Laser Repetition Rate (0-50) 2 bytes
- Laser Burst pulses (0-100) 3 bytes

4. Message types and sizes

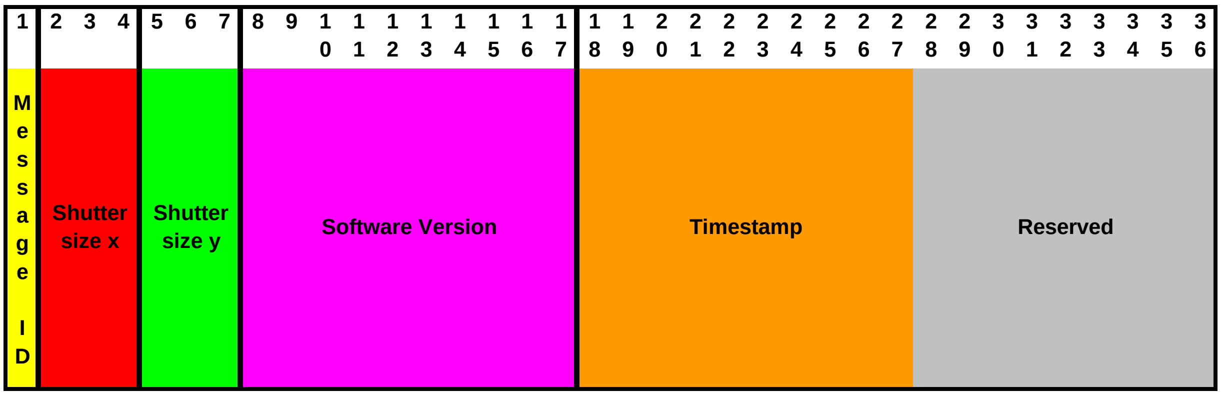
All messages have a unique message ID that specifies the message type; up to know 9 messages are supported. The following Message IDs have been allocated so far

[1] Imperial College, London
[2] Rutherford Appleton Laboratory

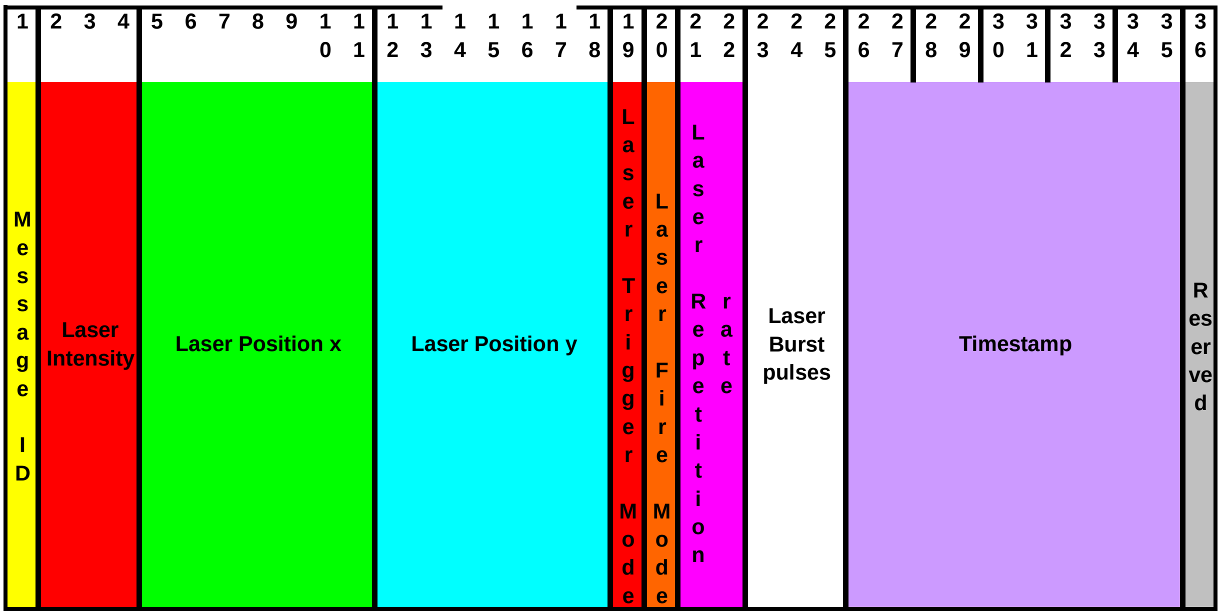
Message ID	Message Type
1	SetConfiguration
2	GetConfiguration
3	GetRunData
4	FireLaser
5	Reserved
6	Reserved
7	Reserved
8	Reserved
9	Acknowledged Message

The total length of a message is 36 bytes. There are three message formats, the RunData, the Configuration Data and the Acknowledge message. Messages 1 and 4 are echoed by a return message using the Acknowledged Message format.
The character "*" is used for padding.

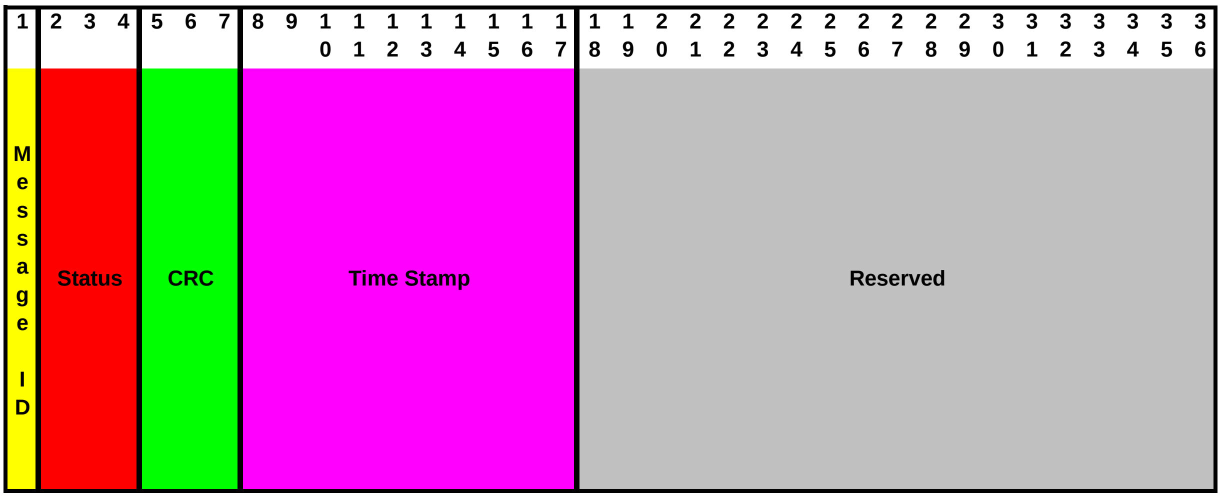
4.1 RunData Message Format



4.2 ConfigurationData Message Format



4.3 Acknowledged Message Format



4.4 SetConfiguration Message

This message uses ID 1 and sets all its included values

4.5 GetConfiguration Message

This message uses ID2 and reads back all the configuration values and adds a timestamp, following the POSIX convention. To request a GetConfigurationData Message sending a Message with ID 2 is sufficient.

4.6 GetRunData Message

This message uses ID 3, it will read back the shutter data and the software version, which we consider as unchangeable during a run. A Timestamp is added at the end, following the POSIX convention. To request a GetRunData Message sending a Message with ID 3 is sufficient.

4.7 FireLaser Message

This message has the ID 4, only the first byte is significant, all others can be neglected.

4.8 Acknowledge Message

This has the message ID 9 and the remaining Bytes contain a Status field (3 bytes) a 3 byte CRC field (not used) and a 12 byte timestamp, when the message was send.
The timestamp follow the POSIX convention of second elapsed since 1/1/1970 00:00 UTC.