LDA-DIF link ideas.

These are some comments on the DIF command document from an LDA person, and of course are not anything other than my opinions.

The reservation I have with this is that the original "FAST COMMANDS" structure using the K Comma and D words was designed to get around a specific problem, namely slow clock synchronisation. With the option to be extended to any other fast situations that might arrive.

It was always envisioned that the Physical/Logical Link Layer would be as separated from the Data layer as possible. This would allow us to remove any technology dependency from higher level firmware as much as we could. By embedded the DIF commands into K and D words this is no longer true, things become intimately linked. The worry is that any changes done to the link would cause major headaches to people if we go down this route. Of course, it is still and option, just want people to understand that it is not an ideal solution.

Since the link is already packet based, and the LDA is in essence a simple packet forwarding device, then it makes sense to use this existing system. From what I see in the list of proposed commands there is nothing stopping these being sent via this route.

The LDA already specifies its ODR-LDA link as a packet based on Ethernet along the following lines:

```
struct lda_packet {
  struct ethhdr ethernet_header; /* Defined in linux/if_ether.h */
  u_int16_t lda_packettype;
  u_int16_t lda_typemodifier;
  u_int16_t lda_pktID;
  u_int16_t lda_data_length;
  u_int8_t  lda_packetdata[];
} __attribute__((packed));
```

One of the forms of this this encapsulates a DIF packet that the LDA forwards. Data of length lda_data_length in lda_packetdata[] is sent to the requested DIF. This is sent, between the /S/ and /EPD/ framing signals. There is no reason that what is send here needs to be just "block data" it would make sense to have some initial header structure on the data, to allow it to be extended. A very simplistic idea follows.

```
struct dif_packet {
  u_int16_t dif_packettype;
  u_int16_t dif_pktID;
  u_int16_t dif_typemodifier;
  u_int16_t dif_data_length;
  u_int8_t  dif_packetdata[dif_data_length];
  u_int16_t dif_crc;
} __attribute__((packed));
```

You could define dif_packettype to have several values. Allowing you to define packets such as "Block data", "Generic Command", "ECAL Command", "AHCAL Command" and so on. This would prevent miss configured systems from sending AHCAL specific commands to ECAL for instance, as the ECAL would ignore it, this might prevent possible bad things happening in that situation.

dif_pktID could be used to match things together, if a previous packet had some data that this packet is to operate on. Or could be used to track command streams, to make sure there has been no loss in the system, eg, if you see non sequential ID's then something has gone wrong. Also, for DAQ system register reads, you could reply with the same ID, and this allows the DAQ software to match reply's to requests.

dif_typemodifier could be where you could place all the command's you previously defined. You now have 16 bits to use, in what ever way you want. You could maybe use this to hold modifiers such as "pass this to neighbour DIF for processing" or such, when in a fail over mode.

dif_datalength and dif_data[] would be used if the packet is actually carrying block data, define dif_datalength=0 and you are saying the there is none, and the CRC follows directly.

This is also vastly expandable, as people can in a way do what they like with the packet format. The nice thing from the LDA side is that nothing is assumed. This is just a packet that is sent from place to place. It also makes it easier for the LDA-DIF link aggregation board that the HCAL people want to design/operate, since it could define a dif_packettype(s) for it and then in essence wrap the real dif_packet inside this dif_packet. As shown below.

```
struct dif_packet {
  u_int16_t dif_packettype;                  // Concentrator type
  u_int16_t dif_pktID;
  u_int16_t dif_typemodifier;                // Interpreted by concentrator
  u_int16_t dif_data_length;                 // size of real packet
  u_int8_t  dif_packetdata[dif_data_length]; // another dif_packet
  u_int16_t dif_crc;
} __attribute__((packed));
```

I admit, this means the state machine on the DIF side of the link does need some intelligence. However, it is nothing too tricky. A simple packet buffer sitting on the RX side of the SERDES can hold full packets, and the state machine can then process them. The advantage is you are now totally decoupled from the physical layer, the LDA-DIF SERDES does all the link work, hiding all the horrid things from you and it spits out the stuff that allows you to fill your packet buffer, and then your state machine works on that. The SERDES still decodes the slowclock synchronisation and provides those signals

separate to the normal packet data ones. Since the LDA side already needs these RX buffers and such, those can be provided as part of the LDA-DIF link package, along with the SERDES.

If we assume your command values you already defined as Dx.x are used, then what needs to be defined are the values for dif_packettype. For example:-

- dif_generic
- dif_ecal_command
- dif_dhcal_command
- dif_ahcal_command
- dif_concentrator_command
- dif_block_data
- ... You could define "neighbour mode" dif packets separate, or there by allowing the dif to pass it over very easily without having to look too deep into it.
- dif_generic_neighbour
- ...

The same packet could also be used for sending the event data back to the LDA, define a packettype, and use the fields to hold some ID stuff and then use the dif_data_length and dif_data[] to hold the event size, and the data.

There is of course an over head with using packets for commands, which is that you have a minimal sized packet. However consider that at the ODR-LDA link you already have such a limit with Minimal Ethernet Packet size, you are already suffering from this.