

## Mipp Data Cable protocol

### Physical definition:

Each front end is fitted to two RJ-45 connectors, each of which contains four differential pairs. Two pairs are bussed. The controller transmits on these pairs. One is a synchronous link for trigger, timing and initialization, and the other an asynchronous link used for control and status. Two pairs are daisy-chained. The front ends transmit on these pairs. One is used for sending event data, the second for sending status information. The power up default condition for the boards is to listen for an initialization message on the synchronous link and transparently pass through anything received on the daisy chained links. The direction toward the readout controller is defined as "downstream". The first card is defined to be the card that transmits directly to the controller.

### Electrical definition:

The signal levels on the pairs within a cable are defined as LVDM which is designed to drive a doubly terminated line. Transformer coupling is used. The encoding scheme is FM with bits represented as two frequencies: 26.5MHz (logic 0) and 13.28MHz (logic 1). Front ends are required to phase lock their transmit channels to the synchronous bussed pair. Data transmitted by the front ends will be at known frequency, but unknown phase with respect to the synchronous link.

### Logical definition:

Each frame consists of a 21 bit sequence beginning with a start bit followed by 18 bits of data, a parity bit and a trailing 0. The first two bits of the data field are used to distinguish between control and data words. Two control codes will be used to define a begin block and end block character. The front ends will use the end block character along with their address to decide when it is their turn to send event data. The last card in the chain will send event data first. When the next card detects the end block character, it will transmit and so on. Any card that is not transmitting its own data must transparently pass the data received from its upstream neighbor to its downstream neighbor. When a card changes from one transmit state to the other, the phase of the FM cannot be disrupted, so as not to confuse the receiver on the downstream neighbor.

### Chain Address assignment sequence:

The daisy chained links allow for the automated card address assignment on a cable. When an assign address message is broadcast on the asynchronous bussed pair, all cards will block the re-transmission of data from their daisy-chained links until after they have received a response word from their upstream neighbor. The last card in the chain will send a response message on its status pair and assign itself the first address in the chain. The FM protocol on the cable means that all transmitters will be sending a pulse train. The absence of this pulse train on

its upstream connector will allow a card to identify itself as the last in chain. The response includes the card address. The remaining cards will receive responses from their upstream neighbors and add one to the address information received, assign themselves this address and send the modified response to their downstream neighbor. After a card has sent its configuration response, it must transparently link the upstream and down stream daisy chained ports. The first card will send its address not to another card, but to the readout controller. Thus the controller can know how many cards are on the chain.

## **Synchronous Link Messages:**

### **Reset:**

This message would be sent when the equivalent of a power up reset is desired.

### **Initialization:**

This message would presumably be sent once per run. This will refresh software setup parameters, clear any data in the event buffers, reset the event counters and any timers used, clear any latched error bits.

### **Begin Spill:**

At this time the 53MHz timers used for event synchronization would be reset.

### **Triggers:**

Trigger information will include event synchronization data and some trigger selection bits. The desire is to keep the length of the trigger message shorter than the minimum time between triggers for the sake of simplicity.

**End Spill:** Is this necessary/desirable?

### **Read next event:**

This message would include event synchronization information. Event data is restricted to sequential access, and cannot be accessed during the spill. Each front checks that the stored synchronization information corresponding to the event fragment ready to be sent in its buffer matches the synchronization information in the read next event message. There is sufficient logic in the front ends such that they will sequence themselves and transmit event data to the controller from all the front ends connected to a cable.

## **Asynchronous Link Messages:**

These messages are for writing front end setup parameters and returning slow control and monitoring information. The details of this data are front end dependent. The upper order bits of the data frame could be used to distinguish between addresses and data.

### **Error handling:**

There are two classes of errors: Those associated with particular events (e.g. synch errors) and those associated with the state of the front ends (e.g. over temp, link parity errors) Event specific errors are bits set in the status word in the event header. The other class of errors would be sent in response to status queries by the controller.

## **Event data format:**

### **From the front end:**

The event data block consists of an event word count, followed by synchronization information, a status word and finally the subsystem specific event data.

### **From the controller:**

The event data from all the front ends will be concatenated into a single word. Redundant information from the individual front ends will be stripped off. The block will consist of a word count, one set of synchronization information a generic header block, a subsystem specific header block followed by the data from all the relevant front ends attached to the controller.

## **Front end functional requirements:**

### **Addressing:**

Each front end must respond to a self-configure command and be capable of assigning itself an address based on its location within a chain of front ends on a cable and passing the appropriate message to its downstream neighbor.

### **Setup data:**

Each front end must respond appropriately to setup reads writes when addressed. It must have the ability to store in non-volatile memory this information to eliminate the need for sending large quantities of setup information before each initialization command.

### **Initialization:**

Each front end must respond to an initialization command by refreshing its setup parameters, resetting any counters, flushing its data buffers etc. When this process is complete, an initialization acknowledge message is sent on the synchronous daisy-chained link to indicate that the front end is ready to take data. The front ends must have the ability to order their transmissions so that one init message will result in the transmission of all the init acknowledges on the chain. This ordering is done in the same manner as for event data.

### **Triggering:**

On receipt of a trigger message, each front must be capable of cross checking the synchronization data contained in the message. Synchronization information must be stored with each event for further checks during readout. There must be a capability for delay adjustment of the on-card event synchronization timers to compensate for signal propagation differences of the data cables. Any synchronization errors must be reported in the event data header. Each front end must have the ability to store (e.g. pipeline) data long enough for a trigger decision to be made.

**Buffering and readout:**

Each front end must be capable of storing all the event data for a four second spill which is specified to be 20,000 events (12,000 events + contingency). Fiftysix seconds are allotted for transferring the data from the front ends to the controller (typically thirty seconds for 12,000 events). The front ends must have the ability to order readout data on the data cable according to their address in response to a single event readout broadcast command from the controller.