The IPU Channel Data Format

J. Michel

July 7, 2009

1 IPU Channel Data Format

1.1 Overview

All data on the IPU channel is organized in 32 Bit wide words. On the network, hidden from the user, these are transported in two 16 bit words, MSB first.

The structure of the data stream will change slightly from the frontends through several hubs and to the final SubEvent structure.

The start of each transmission is a network header (5x 16bit) which does not carry necessary information on the IPU channel. The end is formed by the network termination packet (again 5x 16bit), which can transport some basic error information. Every hub and every frontend contributes to this error information. Since this 32 bit wide information has the same meaning in every stage, it is not shown in the sections dealing with frontends and hubs.

1.2 Data sent by Frontends

The data structure is shown in table 1.1. The first word that is sent by any frontend contains event information, namely the 16 bit event number, a 8 bit wide random code, the 4 bit wide trigger type (T) and a so-called "pack-bit" (P). The remaining three bits are reserved (R) for future use.

Bits	**31 .. 24**	**23 .. 16**	**15 .. 8**	**7 .. 0**
Event Information	R R R P T T T T	Random Code	Event Number	
Data Header	Length		Endpoint Address	
Data Words	Data			
	...			

Table 1.1: IPU Data sent from any frontend. Single letters are explained in the text

The second word contains the length of data following (counted in 32 bit words) and the TrbNet address of the frontend board. This dataword is generated automatically by the endpoint itself.

Afterwards, all event data is sent. The datastructure depends on the specific requirements of the frontend. It is suggested to put a general data descriptor in the first word, then followed by event data and / or debug information if needed.

The frontend has to send the event information and data header in any case, also when there is now event data available, e.g. because no hit was detected.

1.3 Data Stream generated by Hubs

A hub merges data from one or more senders into a single data stream. Table 1.2 shows the resulting data stream in case of a hub with two connected frontends, one delivering two data words, the other only one data word.

The event information word is read from all connected frontends, checked to be right for the expected event and then sent as a single word in the beginning of the data stream. Afterwards, the hub adds up the length information in the data header from all received data streams and generates a new data header, containing the overall length and the network address of the hub. Finally, all incoming data is forwarded starting with the data header of each data stream.

Bits	**31 .. 24**	**23 .. 16**	**15 .. 8**	**7 .. 0**
Event Information	R R R 1 T T T T	Random Code	Event Number	
Data Header Hub1	Length (5)		Hub Address	
Data Header FEE1	Length (2)		FEE1 Address	
Data Words	Data1 FEE1 Data2 FEE1			
Data Header FEE2	Length (1)		FEE1 Address	
Data Words	Data1 FEE2			

Table 1.2: Hub merging data from two frontends in non-packing mode

This operation mode is called non-packing mode. Opposed to that, the packing mode (table 1.3) would remove the data headers received from the frontends. This can only be done if the remaining data stream still contains all information needed to unpack and process the data.

Which of the two operating modes is used is chosen based on the “pack-bits” received in the event information word: If this bit is set in all received data streams, then (and only then) the packing is enabled. Data sent from frontends will always have the pack bit cleared, so that the address information is always kept.[1]

[1]Even though the RICH data format would allow to remove the length/source information from the frontends, it will be kept. The first argument is, that the overhead generated by these words is negligible. The second more importan point is, that the SubEventBuilder has to be able to add his own data words to the end of the data stream and therefore needs to put its own length/source information to the data stream. Without this information it would not be possible to detect the

The packing mode does not only reduce the amount of data, but has another advantage as well: In the non-packing mode, each additional layer of hubs would add another Data Header with a new overall length and a new hub address. Then the program unpacking the data is not able to know how many data headers will appear before the first data word without additional information (i.e. the number of hubs used) not contained in the data stream. Therefore, the first hub will set the pack-bit in its outgoing data stream in any case to allow (possible) next hub to remove its data header.

Under this assumption, the number of data headers stays constant, no matter how many hubs the data is passed through. If the frontend data does not allow packing (e.g. MDC), there are two data headers before the first data word. If the frontend allows packing (e.g. RICH), there is only one data header.

Bits	**31 .. 24**	**23 .. 16**	**15 .. 8**	**7 .. 0**
Event Information	R R R 1 T T T T	Random Code	Event Number	
Data Header Hub1	Length (3)		Hub Address	
Data Words	Data1 FEE1			
	Data2 FEE1			
	Data1 FEE2			

Table 1.3: Hub merging data from two frontends in packing mode

1.3.1 Debugging Information from Hubs

To send debugging information, the hubs are allowed to add an own data header and an arbitrary number of data words to the end of each transfer. This data will then appear as if it comes from another endpoint but has the network address from the hub. Clearly this feature can only be used when the hub is used in a non-packing mode, since otherwise the hub debug data can not be detected properly.

1.4 The SubEventBuilder

Before data transported on TrbNet can be forwarded via Ethernet to the Event Builder, a SubEventHeader has to be added. This will always be done as the last processing step inside an FPGA, no matter whether the data is then directly sent via Gigabit Ethernet or piped through an Etrax CPU. (At least, this is our plan, for the meantime building subevents is up to the software)

The SubEventBuilder reads the first two words, namely Event Information and the

end of event data and the start of SubEventBuilder information

Bits	**31 .. 24**	**23 .. 16**	**15 .. 8**	**7 .. 0**
Event Information	R R R 1 T T T T	Random Code	Event Number	
Data Header Hub1	Length (7)		Hub Address	
Data Header FEE1	Length (2)		FEE1 Address	
Data Words	Data1 FEE1			
	Data2 FEE1			
Data Header FEE2	Length (1)		FEE1 Address	
Data Words	Data1 FEE2			
Data Header Hub	Length (1)		Hub Address	
Data Words	Data1 Hub			

Table 1.4: Hub merging data from two frontends in non-packing mode and adding some debug information

Data Header from the last hub, and generates a SubEventHeader based on this information.

1.4.1 Network Termination Packet: Error and Status Information

The network termination packet transports an overview of errors that happened during readout. There are 16 bits of information as shown in table 1.6.

This information is transported in the same manner as hubs add debug data, namely with an additional data header in the end of the datastream.

Bits	**31 .. 16**	**15 .. 0**
SubEventHeader	SubEventSize	
	SubEventDecoding	
	SubEventID (fixed network address of SubEventBuilder)	
	SubEventTriggerNumber	
Data Header	Length FEE1	FEE1 Adress
Data Words	Data1 FEE1	
	...	
Data Header	Length FEE2	FEE2 Adress
Data Words	Data1 FEE2	
	...	
Data Header	Length SEB (1)	SEB Adress
Add. Information	SubEventInformation	

Table 1.5: Data output of the SubEventBuilder (here: non-packed format. In packed format, the data headers would be missing)

Bits	**Name**	**Description**
16	trg. number mismatch	Trigger number received does not match internal counter value
17	trg. code mismatch	Mismatch in upper 16bit word of event information
18	wrong length	Length submitted in HDR does not match real length
19	answer missing	a board sent a short transfer instead of DHDR
20		
21		
22		
23		
24	not found	sent trigger number does not match stored events
25	partially not found	parts of the data are missing
26	severe problem	serious sync problem detected - intervention needed
27	single broken event	event data corrupted, next event most likely not affected
28		
29		
30		
31		

Table 1.6: Error- and Status information contained in the network termination packet