

Data Streams Implementation

Asynchronous packet flow

Sep 5, 1989

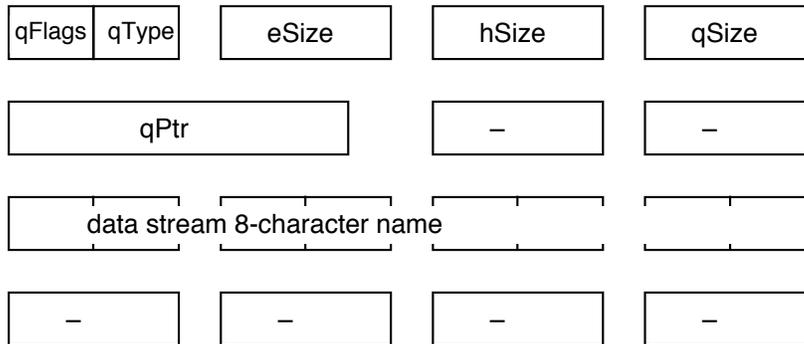
Introduction

Data streams are packets of data that are queued and made available to any data requester. The difference between a data stream and normal data acquisition is that a data stream packet may occur at arbitrary times asynchronous to normal data acquisition. A simple example is data which comes from a serial port. Another is 720 Hz sampled data collected by a ramp co-processor. Another is clock event data.

Normal data acquisition is done synchronously, and typically with only a single value which is collected at 15 Hz. On the other hand, a data stream can have packets added to it at any time even with varying amounts of data. Data stream support herein described makes this variable type of data accessible via a normal data request.

DSTRM system table

The DSTRM table provides for itemization of the various data streams that are supported. A data stream is identified by an index into this table, just as an analog channel is identified by an index into the ADATA/ADESC tables. The format of a DSTRM entry is as follows:



The `qFlags` include a bit (#6) to indicate that at reset time the queue associated with the data stream should be allocated from dynamic memory. Another flag bit (#7) indicates that the queue has been initialized. The `qType` is a small positive index which gives the type of queue header used, as different types of data streams may require different queue management. This index implicitly characterizes the means of queue initialization, packet entry, and packet extraction. The `Size` word is the entry size of the packets in the queue. For variable size packets, this word is zero. (In this case, the first word of each variable size packet is the size of the packet including the size word.) The `hSize` word is the amount of header space needed to support the data stream itself. It is referred to as the data stream-specific header. The `qSize` is the total size of the queue which is used to allocate the queue in the dynamic case and is also used to initialize the queue header. The `qPtr` is the pointer to the queue header. In the case of a dynamically allocated queue, this pointer points 8 bytes beyond the allocated area to allow for the common form of dynamic header:

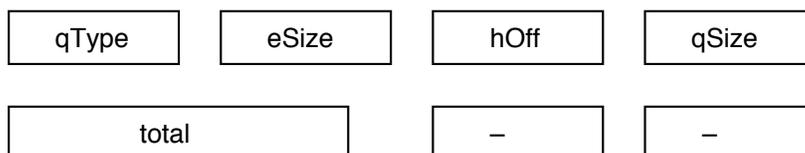


The `mSize` is the allocated size of the memory block, the `mNext` is a pointer to the next block in a chain (when used), and `mType` is the memory block type value of `0x000B` for this case. With the `qPtr` pointing just beyond this header, the same `qType` can serve either the dynamic or the static case. This first part of the DSTRM entry can be accessed using listype #53.

The 8-character data stream name can be used to identify the data stream mnemonically. It can be accessed using listype #54.

Queue format

The data stream queue format consists of 3 components. The first part is the same for all data stream queues. Its format is as follows:



Note that the values are copies of the `DSTRM` entry with a few exceptions. The first word is the `qType` without any flag bits. (This could be changed if the flag bits are needed, as there aren't expected to be many queue types.) The `hOff` is the sum of the header sizes of the first 2 components and is therefore the offset to the data stream-specific header. The `total` longword is the total number of packets ever written into the queue. For diagnostic purposes, the queue header can be accessed using listype #52.

The second component of the queue header is the `qType`-specific header. Its format for `qType=1` is as follows:



The `IN` word is the offset to the space for the next entry to be placed into the queue. The `LIMIT` word is set to the queue size. The `START` word is the offset to the first entry to be placed. It is initialized to point just after the total queue header.

The third component of the header is specific to the data stream itself. This is the component whose size is declared in the `DSTRM` entry. An example of the format of the third component is that used for the Clock Event Queue:

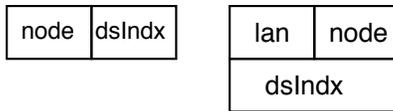


The first 3 words are diagnostic counts which give the number of times the clock event hardware fifo was found to be full (and subsequently cleared), the number of times it was found to be empty, and the number of clock events found in that fifo the last time it was accessed to copy events into the Clock Event Queue. The last word is the time stamp associated with cycle reset that is used to convert the hardware free-running time stamps into ones that are relative to cycle reset. A Data Access Table entry routine manages this header component for the Clock Event Queue.

Additional queue header forms can be designed for other queue types and for other types of data streams.

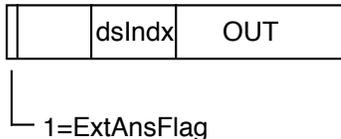
Data requests

A listype (#50) will be used to access data stream packets. The form of ident used is as follows:



Both the short and long ident forms are shown. The requester identifies the data stream index to select the data stream to be accessed. Another listype (#51) is used to request “old” packets—packets which had been placed into the queue prior to the time of the request.

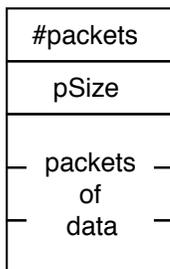
The format of the internal pointer that is kept during request processing is as follows:



Note that the `OUT` word, which is the offset into the queue of the last entry extracted is part of the internal pointer and not part of the queue header. This means that different user requests for the same data stream do not interfere with each other. This is a principal feature of the data stream approach. The `dsIdx` value allows access to the queue header pointer via the `DSTRM` table for fulfilling the request. When the `ExtAnsFlag=1`, the rest of the longword is a pointer into an external answer fragment buffer kept with the request, which just refers to the fact that the data has already been delivered from another node to this node. This last feature is only used for locally initiated requests and for data server requests, not for ordinary network data requests.

Returned data format

The format of the data that is returned in response to a data request of packet data from a data stream queue is as follows:



The first word gives the number of packets that are included in the response data. If it is zero, the queue had nothing in it this time. The second word gives the packet size. If it is zero, the queue uses a variable packet size, and each packet of data will begin with a size word, so the user can process them.

When making a request for previously-written packets using listype #51, the amount of previous packets that can be returned is limited by the size of the requested #bytes. Such requests might be one-shot requests and indicate a large buffer. Requests for only future data might typically be repetitive requests using a moderate size request buffer. A one-shot request to listype #50 would by definition return no information beyond the packet size.

Settings

One can make a data setting to write a packet into a data stream queue. If the queue has variable length entries, a size word (`=#dataBytes+2`) is inserted ahead of the setting data to

form the packet. If the queue has fixed size entries, the length of the setting data must be a multiple of the packet size to be accepted. Either listype #50 or #51 can be used to write a packet into a queue.

The routine `DSWrite` is used to write packet(s) into a data stream queue. It is declared as follows:

```
Procedure DSWrite(dsIndx, nBytes: Integer; VAR data: DType);
```

The `dsIndx` argument is the index part of the ident in the setting request. The `nBytes` word is the number of data bytes, and the `data` parameter is a pointer to the array of data bytes of the packet. If the queue uses variable size packets, only one packet can be written with a single call to `DSWrite`. Note that in this case, a size word is not included as the first word of the data array. The size word is written (with the value `nBytes+2`) into the queue preceding the packet data.

Settings should not be used to data stream queues other than those which are normally written to by a task. Queues which are written to by interrupt activities or by another processor on the same backplane should not be written to by a data setting.

Software modularization

Most data stream logic is centralized into the `DStream` module. The branch tables indexed by `qType` are all in this module. This includes routines which handle queue initialization, read access and write access. Generation of internal pointers is done as usual by code in the `ReqDGenP` and `PreqDGen` modules.

Data-stream specific code—that used to write into a data stream queue—knows about the `DSTRM` table entry format and the first and third components of the queue header. It does not need to know about the `qType`-specific header component.

Variable size packets

As stated above, variable size packets are recorded in the queue using a size word preceding the data. The size word is sufficient to allow data request processing of the packets using listype #50. But looking backwards to retrieve packets written previous to the request, in order to fulfill a listype #51 request, is quite another matter. In order to make this possible, there is an extra word in the queue that precedes the size word. This word contains the offset from the start of the queue header to the previous packet's size word. This allows backwards traversal of the queue's packets. When a variable packet size queue is initialized, the `START` word points just beyond any data stream-specific header. A zero word is placed there, and the `IN` word points to the next word, which will become the size word of the first packet placed into the queue. The extra previous pointer word that precedes the size word is not returned when packets are delivered in response to a data request.

Data stream-specific header initialization

When a data stream queue is initialized, all data stream-specific header space is set to zero. If nonzero values need to be entered there, the data stream-specific code can notice a cleared value and set up any nonzero initialized values needed.