# **File Format of Germanium Detector UDP Streaming**

The Germanium detector delivers timestamped raw data via UDP streaming by UDP engine implemented in the Zynq SoC. On the receiver side, a UDP agent program receives UDP data and saves it as data files to storage. This document describes the format of such data files.

Fig. 1 depicts how detector data flows from the detector to storage via UDP streaming.

## 

Fig. 1 Germanium detector UDP data flow

## 1. Event Data

Event data is the basic unit transmitted by UDP, and each event contains one measurement from the detector, called photon event, which includes photon energy and time information as well as the strip address of the photon event. A complete event consists of two 32-bit words as depicted in Table 1 and Table 2. The first 32-bit word contains the event identifier (‘0’ on bit 31), the strip address (chip number and channel number), along with the photon count and Time of Arrival or Time over Threshold, selected by PV $(Sys)$(Dev).TDM.

Table 1. Photon event format

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bit | 31 | 30:27 | 26:22 | 21:12 | 11:0 |
| Content | 0 | chip\_addr | channel\_addr | td | pd |

The second 32-bit word starts with “100” indicate the content of the word, followed by a 29-bit timestamp. The timestamp a resolution of 10 ns and rolls over in about 42 seconds.

Table 2. Timestamp format

|  |  |  |
| --- | --- | --- |
| Bit | 31:29 | 28:0 |
| Content | 100 | timestamp |

**2. UDP Packet**

The UDP engine transmits event data as UDP packets with a payload limited to 1024 bytes, which is started when a new run/frame is initiated. Each packet starts with a 32-bit running counter/packet counter number that increases for each packet, followed by 32-bit 0s padding. In the first packet of each frame, a 32-bit Start-of-Frame-Marker **0xfeedface** is presented as the third 32-bit word, followed by the 32-bit frame number. After that, the UDP engine sends event data in UDP packets. When the end of a frame is reached, the UDP engine appends a 32-bit number corresponding to the number of events lost due to buffer overflow inside the detector, and a 32-bit End-of-Frame-Marker **0xdecafbad**. Based on this definition, there are three types of packets in each frame, whose structures are depicted in Table 3, 4, and 5 separately.

Table 1 1st UDP packet at Start of Frame

|  |  |  |
| --- | --- | --- |
| 32-bit Word Number | Function | Value |
| 0 | Packet Counter | Running counter |
| 1 | Padding | 0 |
| 2 | Start of Frame Marker | 0xfeedface |
| 3 | Frame Number | Running counter |
| 4...1023 | Event Data + Timestamp | See Table 1. 2. |

Table 2 2nd – (n-1) UDP packet

|  |  |  |
| --- | --- | --- |
| 32-bit Word Number | Function | Value |
| 0 | Packet Counter | Running counter |
| 1 | Padding | 0 |
| 2..1023 | Event Data + Timestamp | See Table 1. 2. |

Table 3 lastst UDP packet at End of Frame

|  |  |  |
| --- | --- | --- |
| 32-bit Word Number | Function | Value |
| 0 | Packet Counter | Running counter |
| 1 | Padding | 0 |
| 2..(n-2) | Event Data + Timestamp | See Table 1. 2. |
| n-1 | Events lost to overflow | count |
| n | End of Frame Marker Low | 0xdecafbad |

## 3. UDP Data File

The UDP agent program writes received data as data files into storage, and UDP packets are kept unchanged for further analysis. Since a run can generate Gigabytes of data, UDP data generated in each run might be divided into small segments. Based on the description of Section 2, due to the difference among the first packet (appears in the starting data file), the last packet (appears in the ending data file), and the intermediate packets (appear in all files). Reference to packet structures for data file formats. Examples of a starting data file, an intermediate data file, and an ending data file can be found as *test.0000000042.0000000000.bin*, *test.0000000042.0000000052.bin*, and *test.0000000042.0000002631.bin* separately.

## 4. UDP Related PVs

UDP transmission relies on the following PVs:

* *$(Sys)$(Dev).IPADDR:* IP address of the UDP interface.
* *$(Sys)$(Dev).FSIZ*: defines the size of data file. Since a run can generate Gigabytes of data, it is divided and saved as small segments limited by *$(Sys)$(Dev).FSIZ*.
* *$(Sys)$(Dev).RUNNO*: run number, or frame number,.
* *$(Sys)$(Dev).FNAM*: defines the directory and name of the data files. UDP agent appends run number and segment number.
* *$(Sys)$(Dev):ONLINE*: indicating the status of UDP agent. Be sure to check *$(Sys)$(Dev):ONLINE* before start receiving UDP data to ensure delivery.