**UDP DATA Streaming on Germanium Detector**

The Germanium detector encapsulates original detector data (energy, timing) into events and sent them out as UDP data. On a remote computer, a UDP agent software receives the UDP data and write it to binary files. The complete data flow is depicted in Fig. 1.

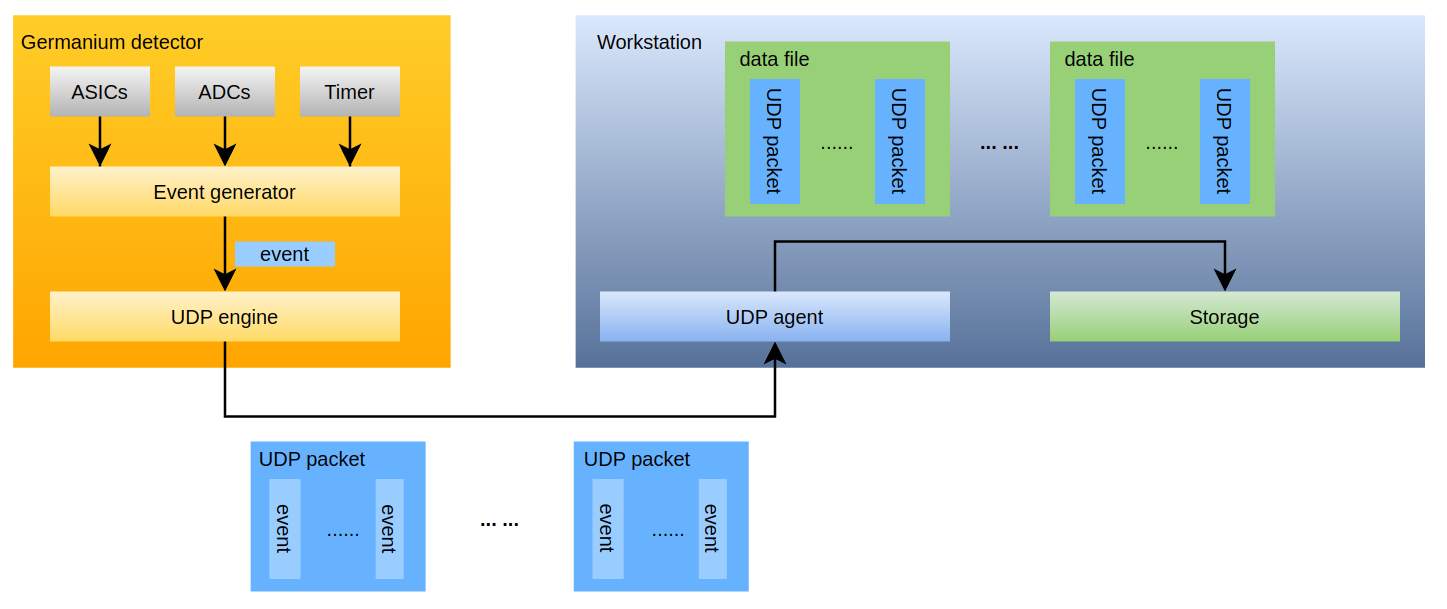


Fig. 1 Germanium detector UDP data flow

# **UDP data streaming configuration**

UDP data receiving depends on two software modules in addition to the EPICS IOC in the detector

* **germDaemon** (<https://github.com/NSLS-II/germDaemon>): an EPICS IOC provides additional PVs (e.g., *$(Sys)$(Dev):IPADDR*) assisting data receiving.
* **germ\_agent** (<https://github.com/NSLS-II/germ_agent>): an EPICS CA client that receives data and writes binary files. Once the main IOC (runs on the detector) is restarted, or the IP address of the UDP interface is changed, germ\_agent needs to be restarted.

UDP data streaming configuration can be done from the Phoebus screen, which is available in **opi/** in **germDaemon** directory. The following PVs must be set/monitored for UDP data streaming to work properly (also shown in Fig. 2):

* ***$(Sys)$(Dev).IPADDR****:* IP address of the UDP interface.
* ***$(Sys)$(Dev).FSIZ***: maximum size of a data file. Since a run can generate Gigabytes of data (a frame), the frame is divided and saved as segments whose sizes are limited by *$(Sys)$(Dev).FSIZ*. The number should be evaluated to achieve optimal performance for data receiving and file saving.
* ***$(Sys)$(Dev).RUNNO***: count number, or frame number. It can be set and it increments automatically for successive counts.
* ***$(Sys)$(Dev):FNAM***: name of the data files (including full path). UDP agent appends run number and segment number.
* ***$(Sys)$(Dev):ONLINE***: indicating the status of UDP agent. Be sure to check this PV before start counting to ensure UDP data being received properly. **germ\_udp** clears a watchdog in **germDaemon** regularly to maintain its online status.
* ***$(Sys)$(Dev):TOLERANCE***: a time frame (in seconds). It keeps **germ\_agent** alive in case of network latency or other factors that causes the watchdog not being cleared within this time frame.



Fig. 2 Phoebus screen for UDP streaming

# **UDP data file analysis**

The data files contain multiple UDP packets whose sizes are limited to 1024 bytes. The formats of UDP packets depends on whether a packet is the first one, an intermediate one, or the last one of a count. 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 last UDP packet at End of Frame*

|  |  |  |
| --- | --- | --- |
| 32-bit Word Number | Function | Value |
| 0 | Packet Counter | Running counter |
| 1 | Padding | 0 |
| 2..(n-3) | Event Data + Timestamp | See Table 1, 2. |
| n-2 | Event lost due to overflow | Counter |
| n-1 | End-of-Frame marker | 0xdecafbad |

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 40 ns and rolls over in about 21 seconds.

*Table 2. Timestamp format*

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