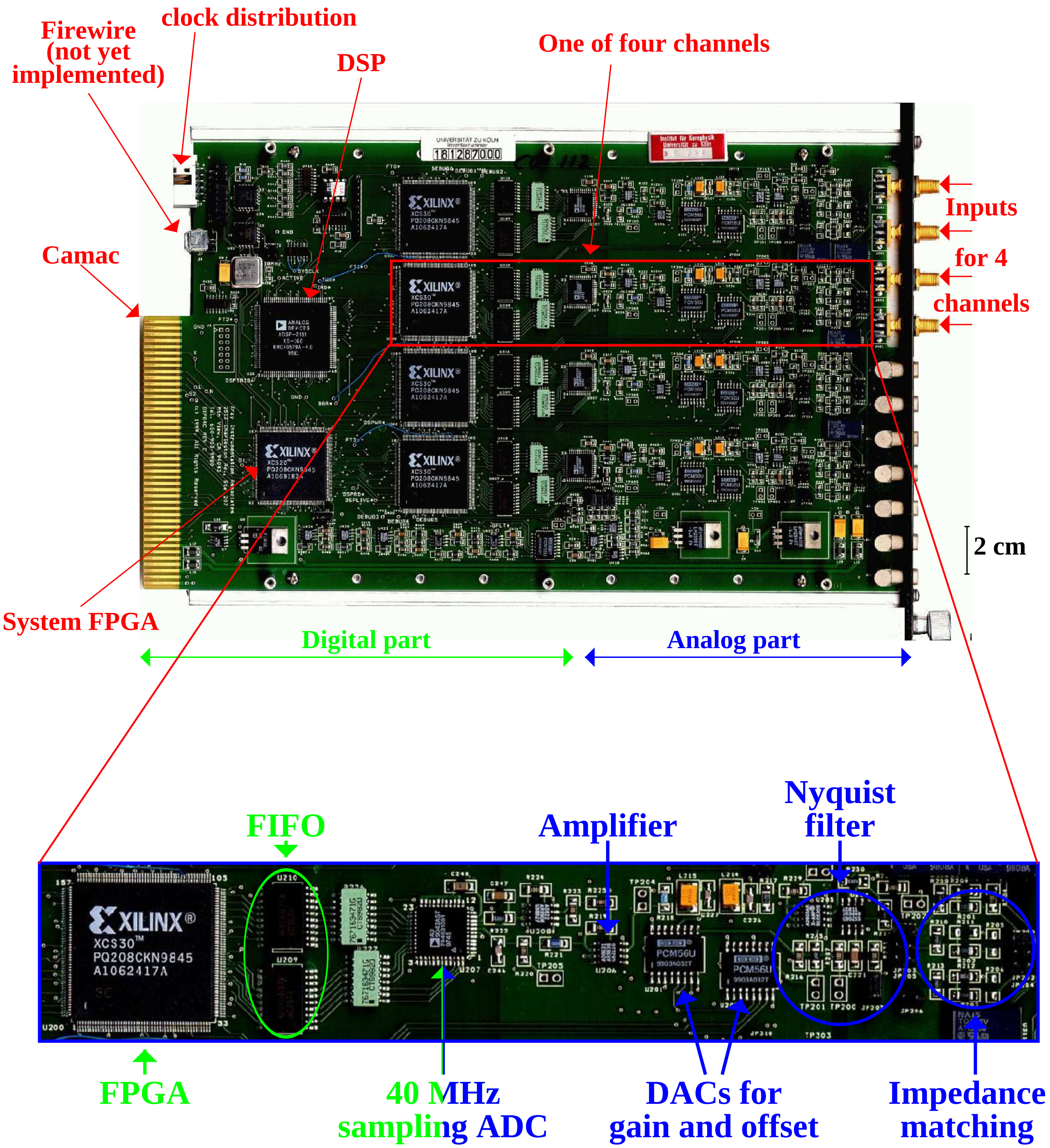
The Digital Gamma Finder (DGF)



Firewire (not yet implemented)

clock distribution

DSP

One of four channels

Camac

Inputs for 4 channels

2 cm

System FPGA

Digital part

Analog part

FIFO

Amplifier

Nyquist filter

FPGA

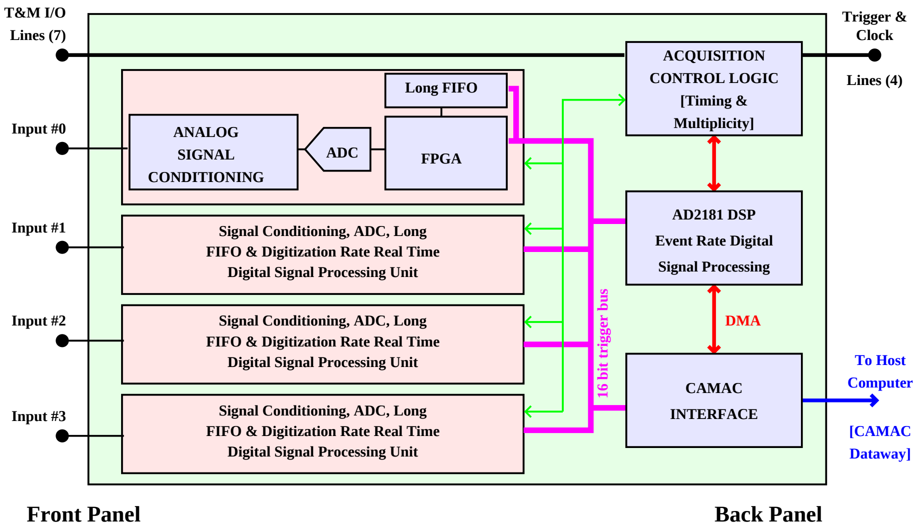
40 MHz sampling ADC

DACs for gain and offset

Impedance matching

The Digital Gamma Finder (DGF)[†]

- **A CAMAC module with four complete spectroscopic channels.**



- **Each channel processes *all* events independently (Analogue part, ADC, FIFO, FPGA).**

- **Logic generates a fast multiplicity signal, which can be used to decide if event is interesting.**

- **Events can be validated by an external NIM signal.**

  → **This validation must be provided *after* the slow filter time (i.e. ≈10 µs after fast trigger).**

- **A single (AD2181) DSP processes *validated* events.**

  → **Events which are not validated are thrown away with zero dead time.**

- **The DSP buffers data for CAMAC readout. The CAMAC protocol is handled by a fifth FPGA (the system FPGA).**

[†] Sold by X-ray Instrument Associates, California (USA) - www.xia.com

DGF Firmware and Software

- **Xia provide us with images for the FPGA chips and executables for the DSP.**
- **These are simply files which are either downloaded from their website or sent to us by e-mail.**
- **Without this firmware and software the DGF is only capable of responding to a couple of very simple CAMAC write cycles.**
- **The first thing to do, therefore, after switching on the CAMAC crate is to use those commands to "programme" the system FPGA. It is the system FPGA firmware which implements the full fast CAMAC level 1 protocol.**

  → **i.e. before the system code is uploaded to the DGF, it cannot understand CAMAC read commands and consequently prevents other CAMAC modules from working.**
- **Once the system code is loaded, the filter/trigger FPGAs can be "programmed". We have different FPGA codes for each decimation. We can also have specialised FPGA codes for "unusual" detectors.**
- **Finally, we programme the DSP. We can use the object code provided by Xia to link to code of our own if we want.**

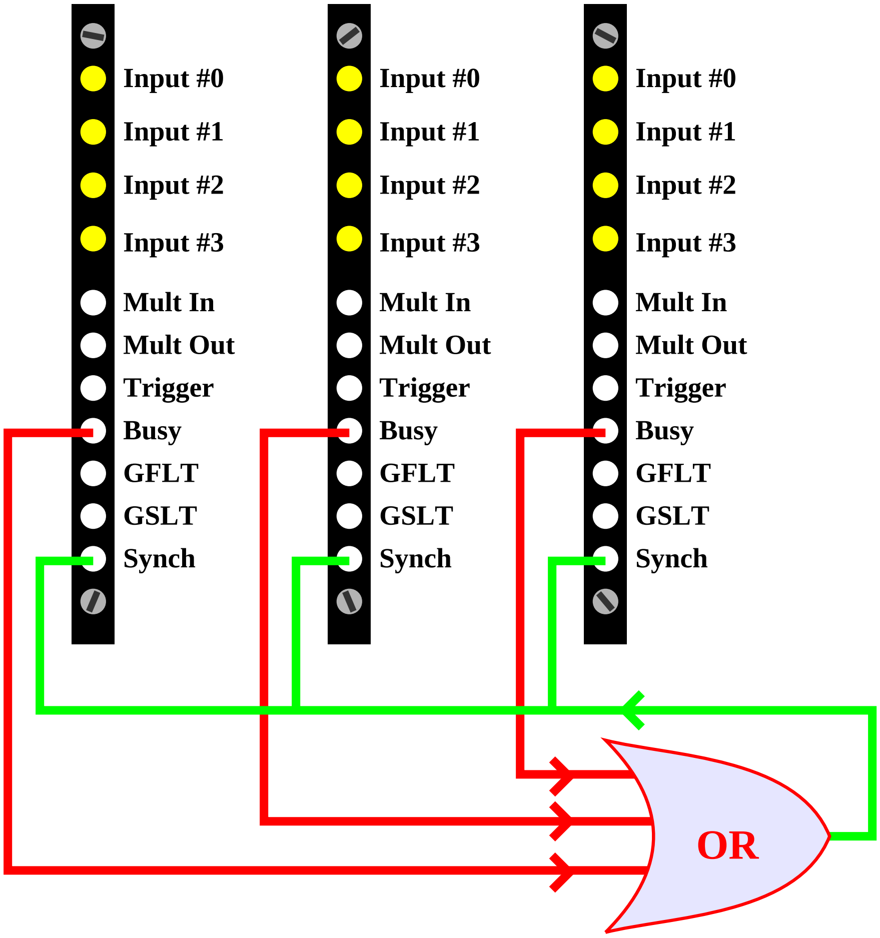
  → **e.g. the pulse-shape analysis code for Miniball.**

Timestamp synchronisation

- If we have more than four detectors, we need use more than one DGF.
- To find time differences between signals from different DGFs we simply substract the absolute values of their timestamps.
  - → Remember: the DGF has a counter which is incremented for each ADC sample (every 25 ns). The value of this counter (timestamp) for each trigger is written into the datastream.
- Clearly the timestamps for different DGF modules must be synchronized.
  - → We need to consider clock drift. Each DGF has an internal 40 MHz clock but if each module uses its own clock, imperfections in the clocks will cause them to drift apart.
  - → The solution is to use just *one* clock and to distribute it to all modules.
  - → Either we use the internal clock of one module and distribute it or we use an external clock.
  - → Note that the revision D version of the DGF has a bug. Only a special external clock developped by George Pascovici here in Cologne will work.
  - → Revision E version doesn’t have this bug.

Busy-Synch loop

- **We need to be able to zero all the counters on the same clock tick when we start.**
  - → **To achieve this, each DGF generates a BUSY signal when it is not acquiring, which is available on the front panel.**
  - → **We generate the logical OR of these BUSY signals and fan the result back to the SYNCH input of each DGF.**
  - → **When the last module starts, this signal goes from logic 1 to logic 0 and we program the DGFs to zero their clocks when this happens.**



Trigger distribution

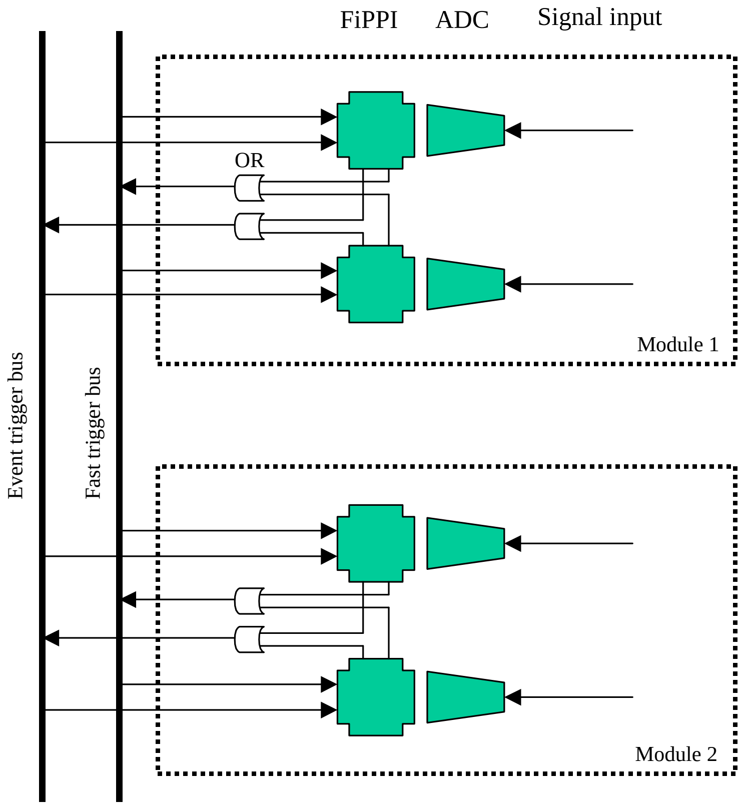
- **For some applications, it is necessary for one channel to trigger another.**

  → **For segmented detectors (e.g. Miniball) there is useful information (e.g. mirror charges) on segments even if they have no net signal.**

  → **So we need to use the core as a trigger and always read out the segments.**

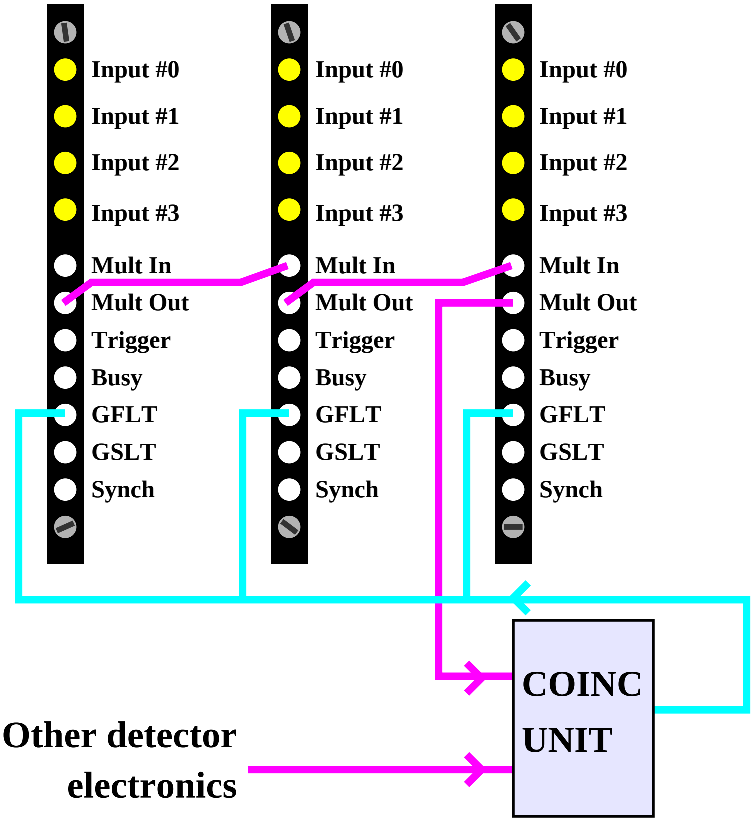
- **As there are more than four signals, we have to make one DGF (with the core signal) trigger another (with a segment).**

  → **The DGF provides a bus on the back panel which can be used to connect the fast trigger and event trigger of adjacent modules.**

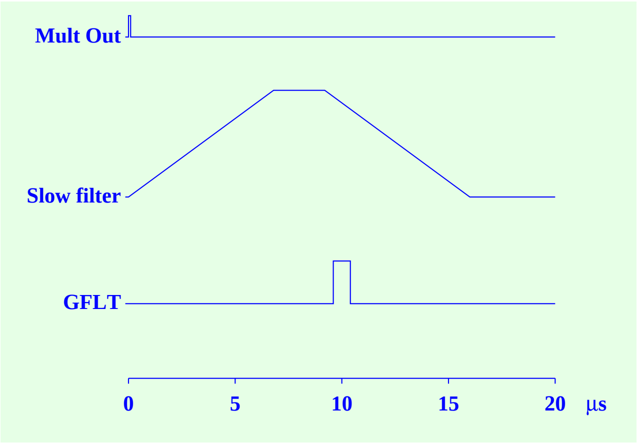


Validation

- Validation is performed using the Global First Level Trigger (GFLT) together with external electronics which can use the Mult Out signal and possibly signals from other systems to decide if the event is interesting.



- The GFLT has to be applied at the time when the DSP receives an interrupt from the FPGA (DSP trigger) at the end of the slow filter.



Reading samples with the DGF

- The most basic function we can perform with digital electronics is reading a series of samples.
- The DGF can acquire traces without a trigger. The untriggered traces can be used to monitor a signal, determine the level of the baseline etc.

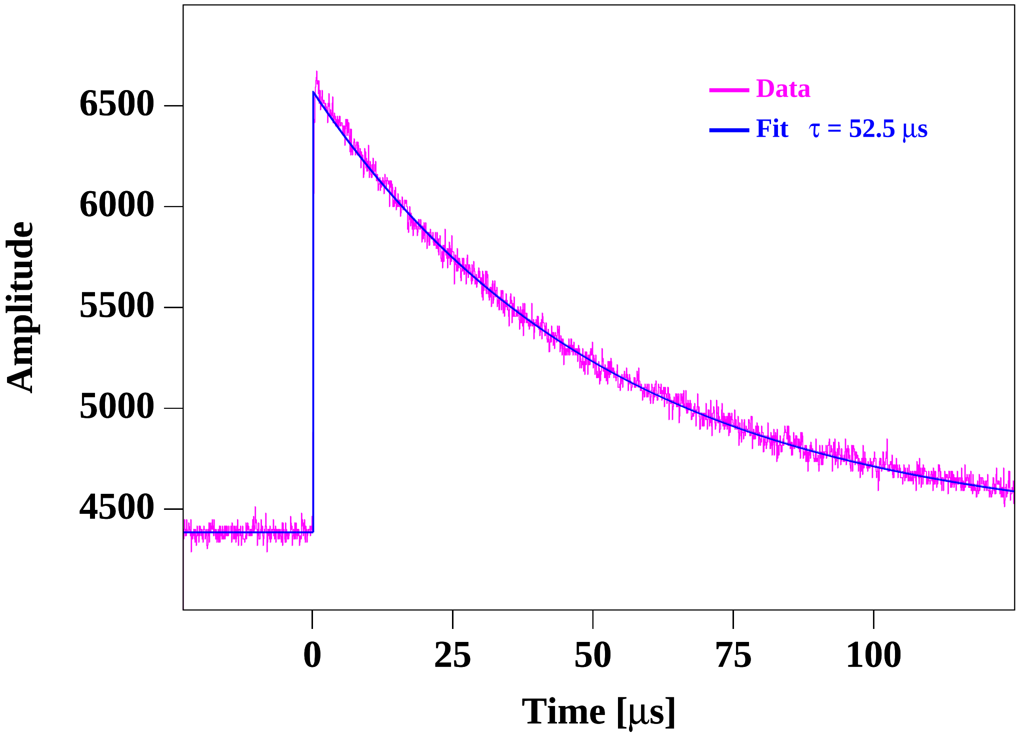
  → In this mode, we obtain up to 8192 samples are separated by `XWAIT` × 25 ns.
- It is also possible to acquire samples whenever a trigger occurs. This can be done in two ways:

  → Either we program the DGF to use its internal FIFO to store up to 4096 samples each separated by 25 ns (just over 100 μs). The parameter `TRACELENGTH` controls the number of samples to acquire. This mode is useful for studying the rising flank of the signals.

  → Or we get the DGF to acquire samples in real time. Then they are separated by (3 + `XWAIT`) × 25 ns. This mode is used to study the exponential decay of the signals and can be used to determine the τ constant of the preamplifier which the DGF needs to know.

Determining τ

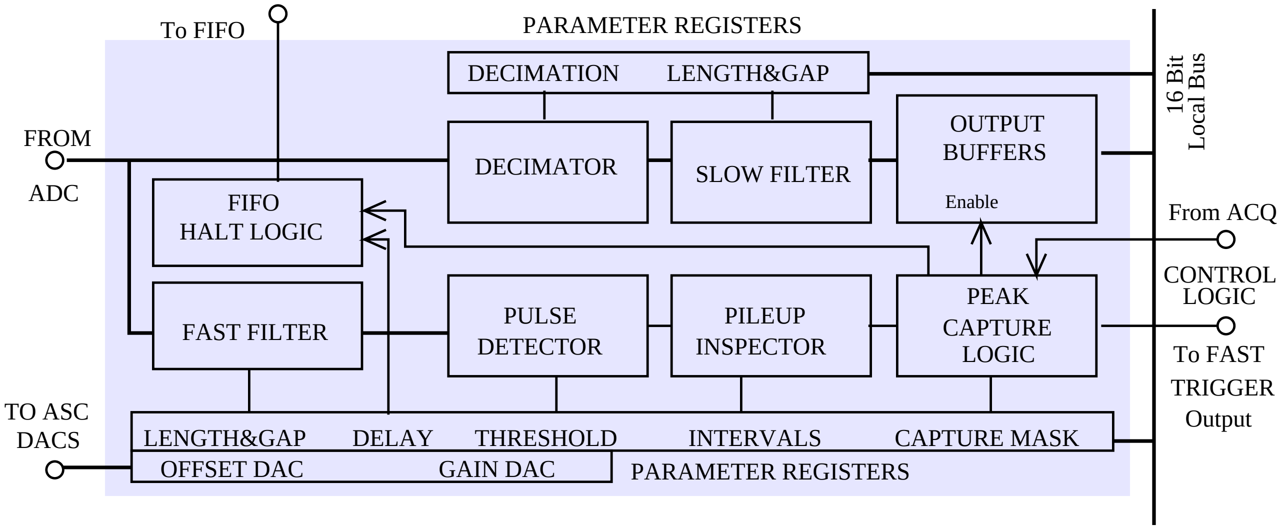
- The DGF needs to know the time constant τ of the preamplifer, which is given as the parameter `PREAMPTAU`.
- We can determine τ by acquiring a signal and performing a least-squares fit.



- Alternatively, we can use the nominal 50 µs value as a starting point and then adjust τ to get the best peak shape and resolution in the resulting spectrum.
- The latter method generally gives the best results.
- This value is particularly important for high count rates, because then, the probability of a second signal occuring on the falling flank of a signal is higher.

The Filter/Trigger FPGA

- **The DGF has one FPGA per channel.**
- **There is a fast branch, which applies a fast trapezoidal filter, used for pulse detection, the pileup inspector and the peak capture logic.**
- **There is also a slow branch, which decimates the samples, applies a slow trapezoidal filter and performs most of the energy determination.**



- **The FPGA analyses *all* signals continuously, detecting peaks, filtering and performing pileup inspection.**
- **When a peak is detected, if triggering is enabled on that channel, it sets a flag to indicate it has valid data. If validation is not required, it sends an interrupt to the DSP. If validation is required, this interrupt is gated by the validation signal. When the DSP receives an interrupt, it reads out all the channels which have data.**

The onboard MCA

- **Each channel of the DGF can work as a multichannel analyzer, acquiring a spectrum without help from the host computer with minimal dead time.**

  → **Additional onboard paged memory is used to store the spectra with 32768 bins each with 24 bits.**

- **The host computer configures the DGF and then starts the acquisition and does not need to communicate again with the DGF until it tells it to stop acquiring.**

- **After the measurement, the spectra can be downloaded to the host computer which doesn't need to communicate with the DGF during the measurement.**

- **Unfortunately, it is** *not* **possible to read the spectra without interrupting the measurement because a special DSP control task is needed to access the paged memory.**

- **This kind of measurement is useful for experiments where just a single spectrum is needed from each detector and no γγ-coincidences are required.**

Acquiring listmode data

- The main use of the DGF is to acquire listmode data.

- Conventional ADCs need to be read out for each event which means any delay in readout caused by latency problems in the host computer result in dead time.

  → A workaround is to use additional hardware to read out the ADCs and buffer the data and send it to the host computer (e.g. the FERA system used in Cologne).

  → Another solution is to use multi-hit ADCs. Such modules can buffer several ADC conversions which are then read out together.

- More modern systems like the DGF package data into large buffers and are designed to be read out in buffered mode.

  → The DSP of the DGF uses 8 kwords of its internal memory as a buffer into which it can put hundreds of events (in the shortest data format).

  → The host computer only has to read out data occasionally but then receives large packets of data. For most modern communications systems the handling of a large packet of data is more efficient than handling the same volume of data in small packets.

Data formats

- Not all users require the same data. Some need only the energies and times of each γ ray which was validated. Others need additional data such as pulse shape information.

  → The DGF implements three different data formats to suit different applications.

- All the formats start with a buffer header, which indicates from which module the data come, the format of the rest of the data and the 48-bit timestamp for the start of the buffer.

Word #	Variable	Description
0	BUF_NDATA	Number of words in this buffer
1	BUF_MODUM	Module number
2	BUF_FORMAT	Format descriptor
3	BUF_TIMEHI	Run start time, high word
4	BUF_TIMEMI	Run start time, middle word
5	BUF_TIMELO	Run start time, low word

- Then for each event, there is an event header, which indicates which channels fired and gives the last 32 bits

  → The 16 most significant bits have to be worked out knowing their value at the start of the buffer and that the timestamps always increase.

Word #	Variable	Description
0	EVT_PATTERN	Hit pattern
1	EVT_TIMEHI	Event time, high word
2	EVT_TIMELO	Event time, low word

- Then we have the data for each channel which fired (as specified by EVT_PATTERN) in the format specified by BUF_FORMAT. This data always contains the energies and times, but may also contain waveforms, times determined with a constant fraction algorithm or pulse-shape analysis information.

Run tasks

- **The DGF provides several different *run tasks* for acquiring in different ways.**
  - **→ A long format with full pulse-shape data and possibly waveforms.**

Word #	Variable	Description
0	CHAN_NDATA	Number of words for this channel
1	CHAN_TRIGTIME	Fast trigger time
2	CHAN_ENERGY	Energy
3	CHAN_XIAPSA	CFD time (0.1 ns bins)
4	CHAN_USERPSA	User PSA value
5	CHAN_GSLTHI	GSLT timestamp, high word
6	CHAN_GSLTMI	GSLT timestamp, middle word
7	CHAN_GSLTLO	GSLT timestamp, low word
8	CHAN_REALTIEMHI	High word of real time

  - **→ An intermediate format with just energy, time and and PSA data.**

Word #	Variable	Description
1	CHAN_TRIGTIME	Fast trigger time
2	CHAN_ENERGY	Energy
3	CHAN_XIAPSA	CFD time (0.1 ns bins)
4	CHAN_USERPSA	User PSA value

  - **→ A short format with just energy and time.**

Word #	Variable	Description
1	CHAN_TRIGTIME	Fast trigger time
2	CHAN_ENERGY	Energy

- **There is also a run task for the acquisition with *only* the MCA. (i.e. without any list mode data.)**
  - **→ Note, however, that MCA spectra can be acquired in parallel with list mode data in all the other run tasks.**

- **Run tasks are performed by setting up the DSP parameters, then writing a bit to the control-status register and the waiting for the DGF to reset that bit when it has finished acquiring.**

Control tasks

- **In addition to the run tasks (actual acquisition), it is often necessary to perform other tasks to control specific parts of the hardware. This is done via** *control tasks***. These tasks include:**
  - → **Programming the gain and offset DACs. (i.e. we copy the values from DSP memory to the DACs).**
  - → **Setting the filter parameters in the FPGA.**
  - → **Ramping the offset DAC to find the optimum value.**
  - → **Acquiring untriggered signals.**
  - → **Reading/writing the paged memory used to store the MCA histograms. Note that it is only possible to access one 4K page at a time. So eight control tasks are needed to read a whole MCA spectrum for one channel.**
- **These tasks are performed in a similar way to run tasks by setting the DSP parameters and writing a bit in the control-status register and waiting for the DGF to reset that bit.**

User DSP code

- The DGF allows the user to write code for the DSP which is called for each event. This code can be used to perform application-specific tasks. E.g., with Miniball user DSP code[†] was written to perform pulse-shape analysis.
- The user provides five routines:
  - → **UserBegin -** called when the DGF is initialised.
  - → **UserRunInit -** called when a run is started.
  - → **UserChannel -** called for each channel which is hit.
  - → **UserEvent -** called for each each event.
  - → **UserRunFinish -** called when a run is ended.
- UserBegin is always called, but the others are only called if a flag is set.
- The code has to be written in ADSP-2181 assembler:

```
UserBegin:
  ar=^RTBuf;              /* UserOut[0]=address of RiseTimeBuffer */
  ar=ar+0x4000;
  dm(  UserOut)=ar;       /* address as seen from the host        */
  ar=%RTBuf;
  dm(  UserOut+1)=ar;     /* UserOut[1]=length  of RiseTimeBuffer */
                          /* To communicate address               */
                          /* and size of buffer to host.          */
  ar=^PolyBuf;            /* UserOut[2]=address of PolygonBuffer  */
  ar=ar+0x4000;
  dm(  UserOut+2)=ar;     /* address as seen from the host        */
  ar=%PolyBuf;
  dm(  UserOut+3)=ar;     /* UserOut[3]=length  of PolygonBuffer  */
                          /* To communicate address               */
                          /* and size of buffer to host.          */
JUMP UserBeginReturn;
```

[†] Written by Martin Lauer, Max Planck Institut, Heidelberg.