ETH zürich

Data Acquisition with the Spectrum M4i.4420-x8 Digitizer

Klemens Flöge, Nicolas Schmid and Quentin Bordier

Group Project HS2021

16.12.2021, Zürich

Motivation:

- Data acquisition and data processing are often done separately, which can be time consuming
- Short experiments can generate huge amounts of data if not no real-time processing is done
- Oscilloscopes/FPGAs are expensive and not very flexible

Objectives:

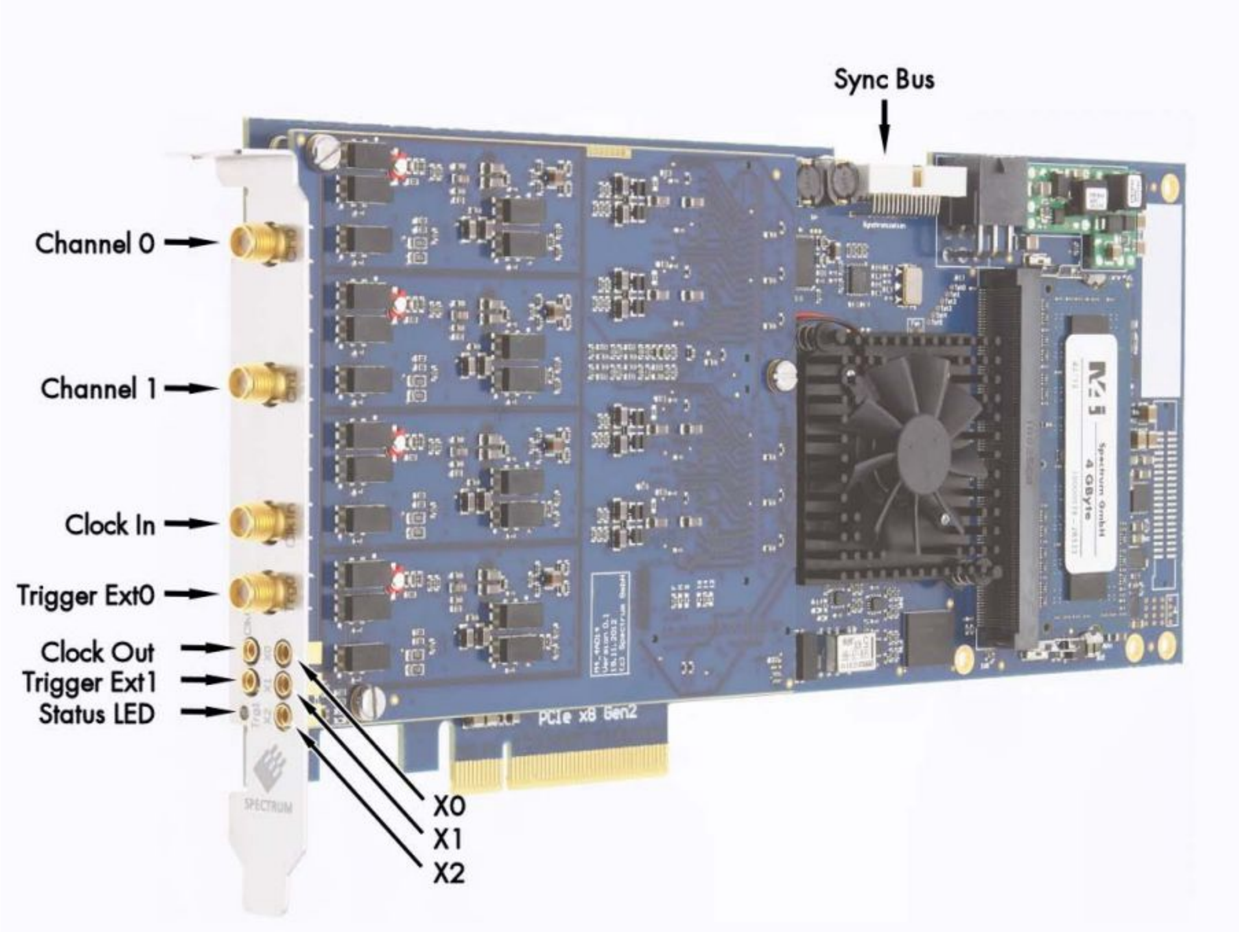
- Learn to operate the Spectrum digitizer
- Developing real time data processing on the GPU
- Explore different functionalities of the Spectrum card
- Use the card in a real experiment

ETH zürich

Spectrum Card

- Card overview
- SCAPP
- SCAPP applications
- Software overview
- Setup

Introduction to the Spectrum M4i.4420-x8 Digitizer



Main used ports :

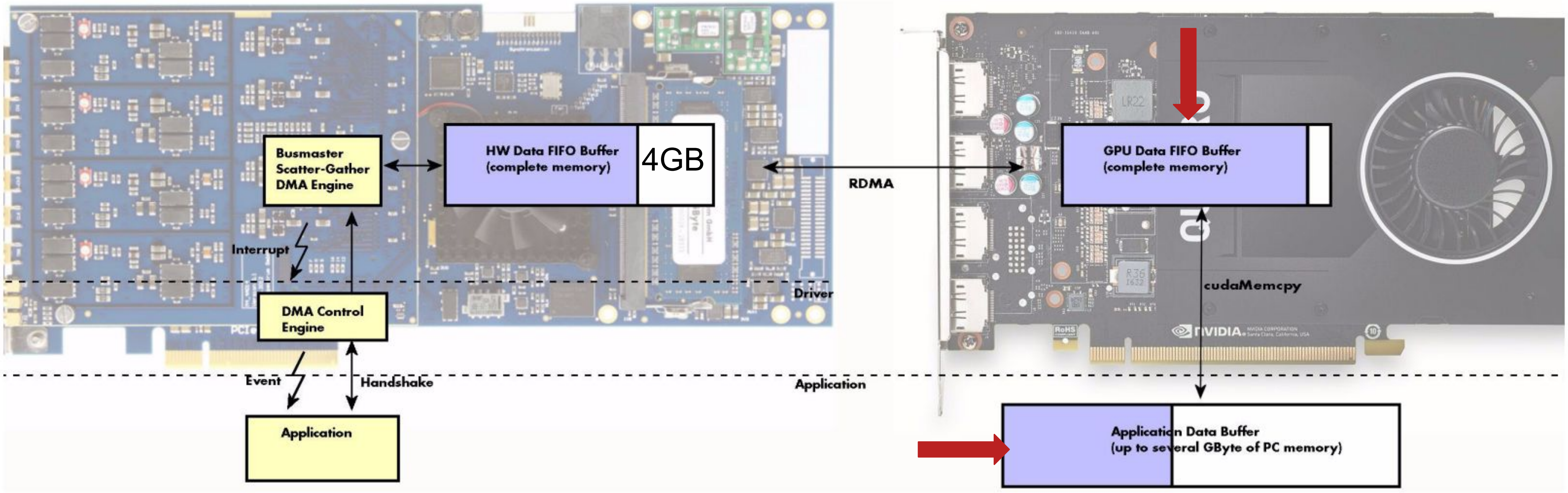
- 2 input channels
- external trigger

Main Features :

- The card is connected to a PCIe slot.
- 250MS/s with 16 bit resolution
- 500MS/s with 14 bit resolution
- Programmable input impedances, amplification and filter

SCAPP

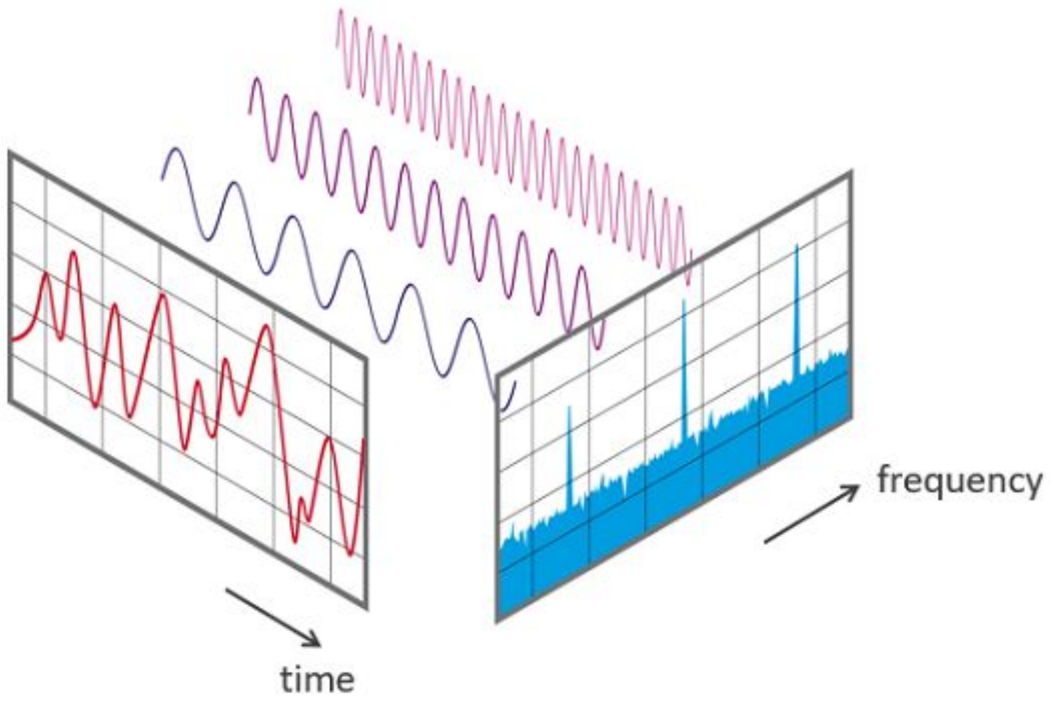
SCAPP allows us to send directly the data from the digitizer to the GPU



SCAPP Applications

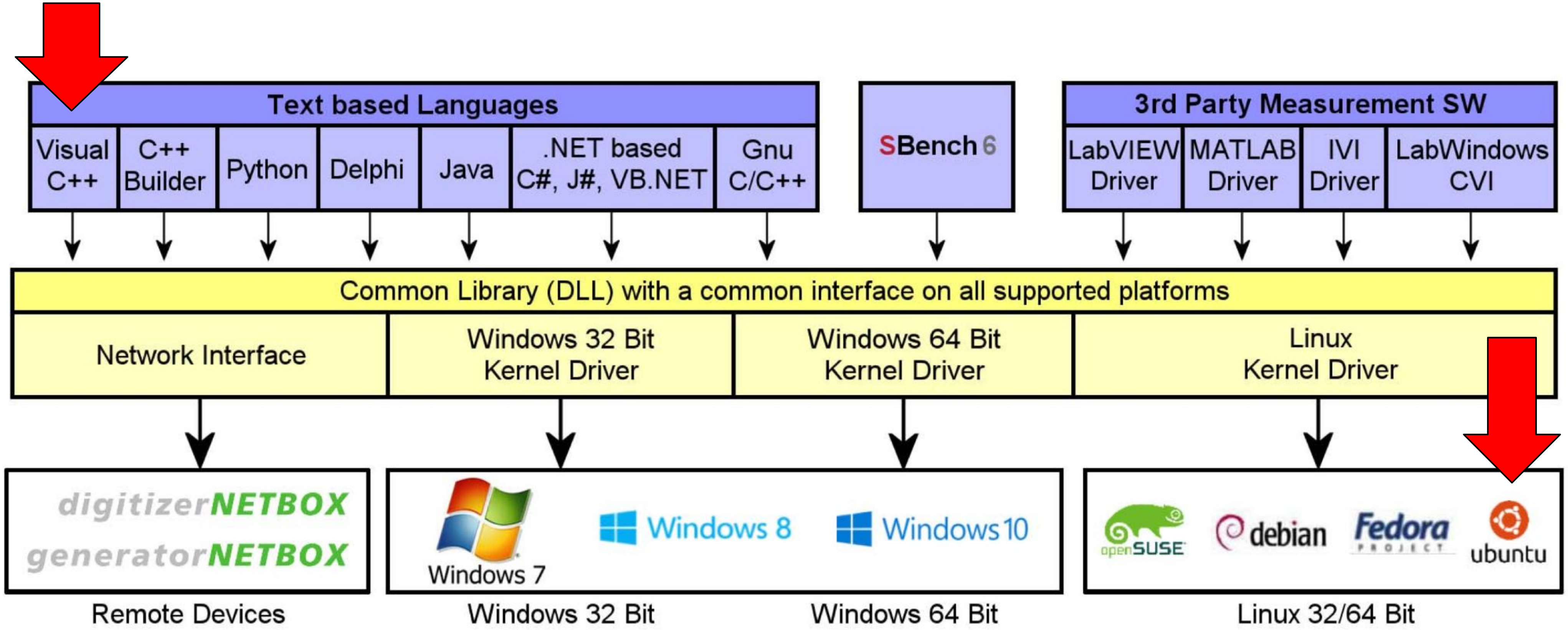
Ideal for signal processing applications such as:

- Data conversion
- Digital filtering
- Averaging
- Fast Fourier Transforms (FFTs)



https://www.nti-audio.com/portals/0/pic/news/FFT-Time-Frequency-View-540.png

Software Overview:



RDMA (Remote Direct Memory Access) between not possible between PCIe Cards and Nvidia GPU on any Windows OS

Set Up

The following steps were required for the setup :

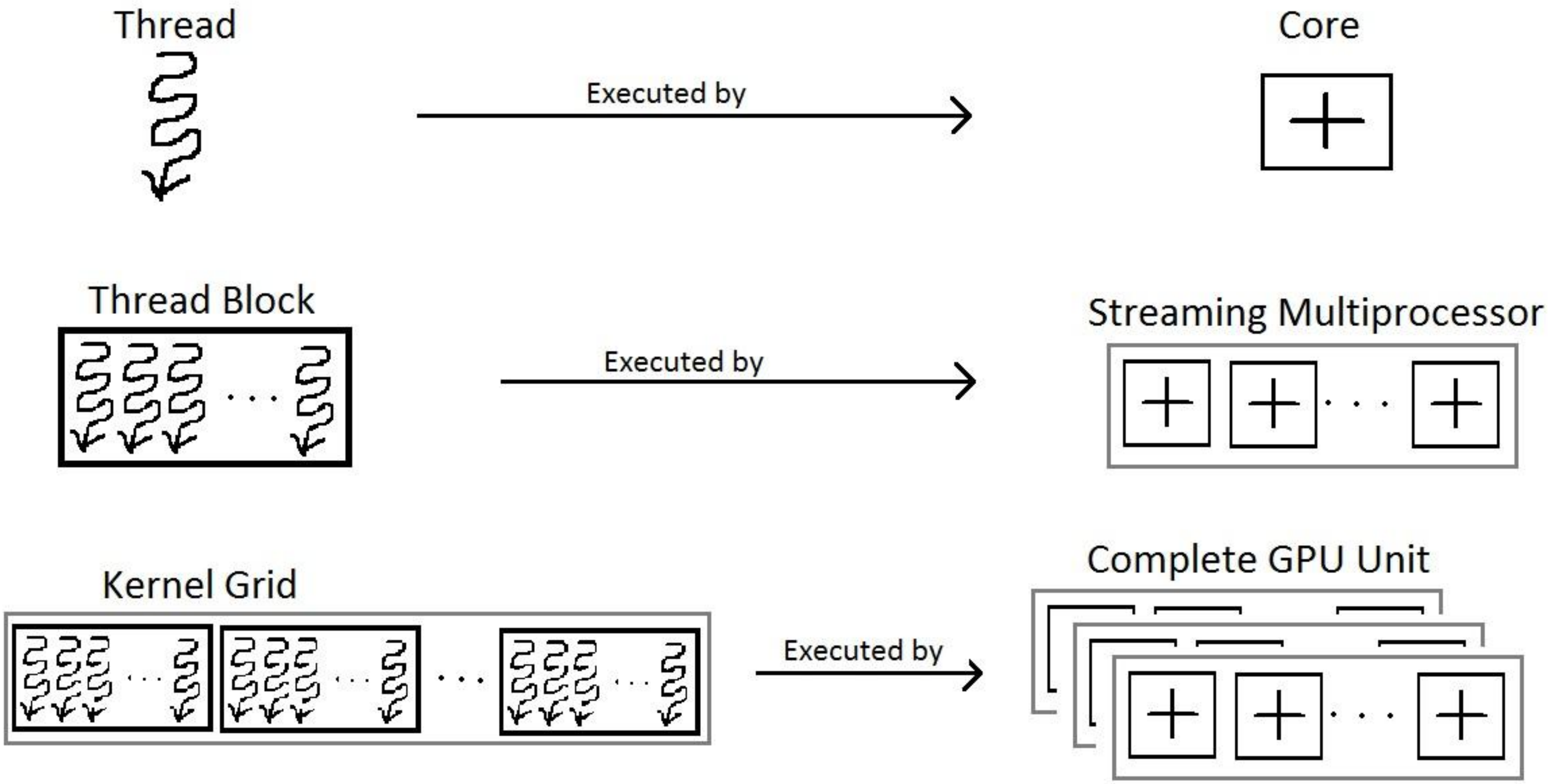
- OS installation
- Drivers Installation:
  - Spectrum drivers
  - SCAPP drivers
  - GPU Driver
- Coding environment:
  - Visual Studio Code

https://icon-icons.com/icon/file-type-vscode/130084
https://www.osboxes.org/wp-content/uploads/2018/04/ubuntu-server-desktop-post.png

Programming the Card

- GPU
- CUDA
- Spectrum
- Memory
- Averaging and FFT’s

Graphic Processing Unit (GPU)



https://upload.wikimedia.org/wikipedia/commons/a/af/Software-Perspective_for_thread_block.jpg

CUDA Kernels: the road to multi-threading

- Kernels are used to access the thousands of processing cores within a GPU
- This is done via kernels in the CUDA programming API
- Example of a kernel which adds two data vectors:

```
__global__ void vecAdd(float *a, float *b)
  {
    // Get our global thread ID
    int id = blockIdx.x*blockDim.x+threadIdx.x;

    // Make sure we do not go out of bounds
    a[id] = a[id] + b[id];
  }
```

Executing a Kernel in CUDA

Example of a Kernel call:

- Number of Operations = Number of blocks * Number of Threads per Block
- lSegmentSize: size of data vector ~ Number of Operations
- number_of_threads: number of threads per block
- Code:

```
vecAdd<<<lSegmentSize / number_of_threads, number_of_threads>>> ((float*)a, (float*)b);
```

Number of Blocks

Number of Threads per Block

The Spectrum API

The spectrum card is coded through a few functions, namely :

- spcm_dwSetParam_i32();
- spcm_dwGetParam_i32();
- spcm_dwDefTransfer_i64();

The functions work by changing the value of registers within the card. The registers are all defined in the manual.

```
uint32 _stdcall spcm_dwSetParam_i32 (    // Return value is an error code
    drv_handle  hDevice,                 // handle to an already opened device
    int32       lRegister,               // software register to be modified
    int32       lValue);                 // the value to be set
```

Register Definition in the Manual

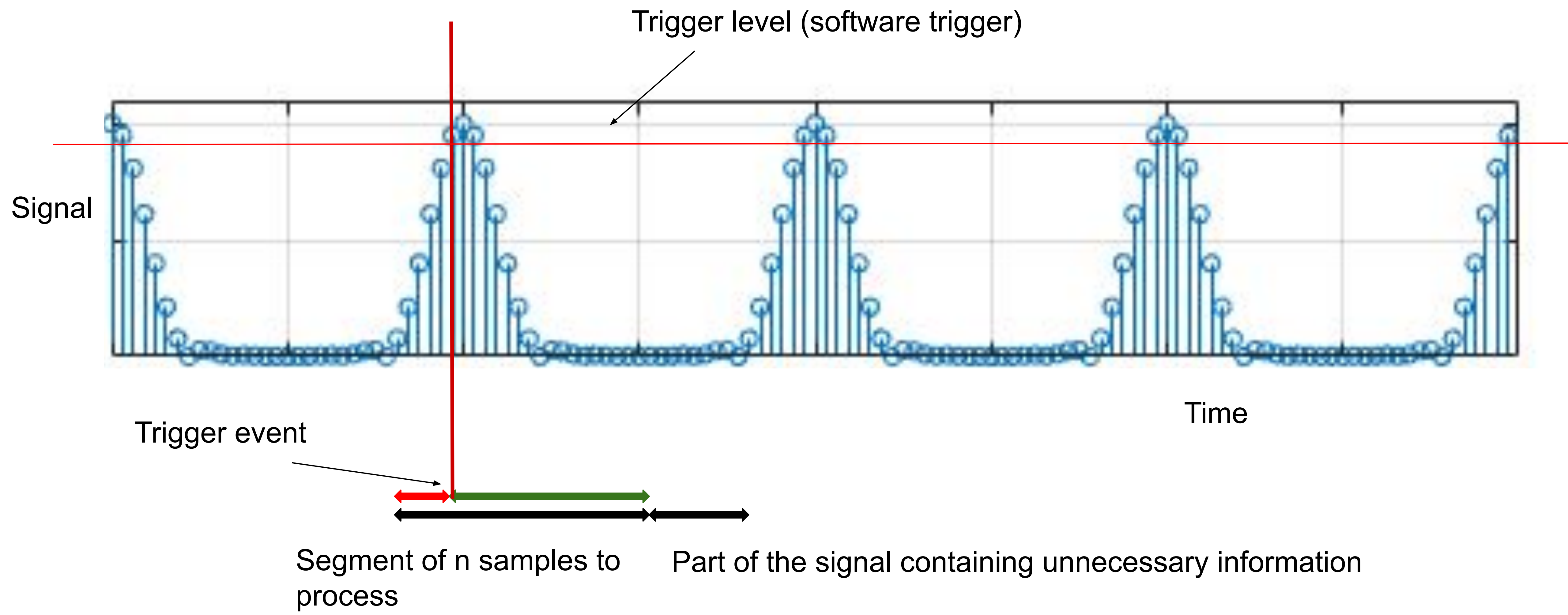
Register	Value	Direction	Description
SPC_M2CMD	100	w	Command register of the board.
M2CMD_CARD_START	4h		Starts the board with the current register settings.
M2CMD_CARD_STOP	40h		Stops the board manually.

```
spcm_dwSetParam_i32(hDevice, SPC_M2CMP, M2CMD_CARD_START)
```

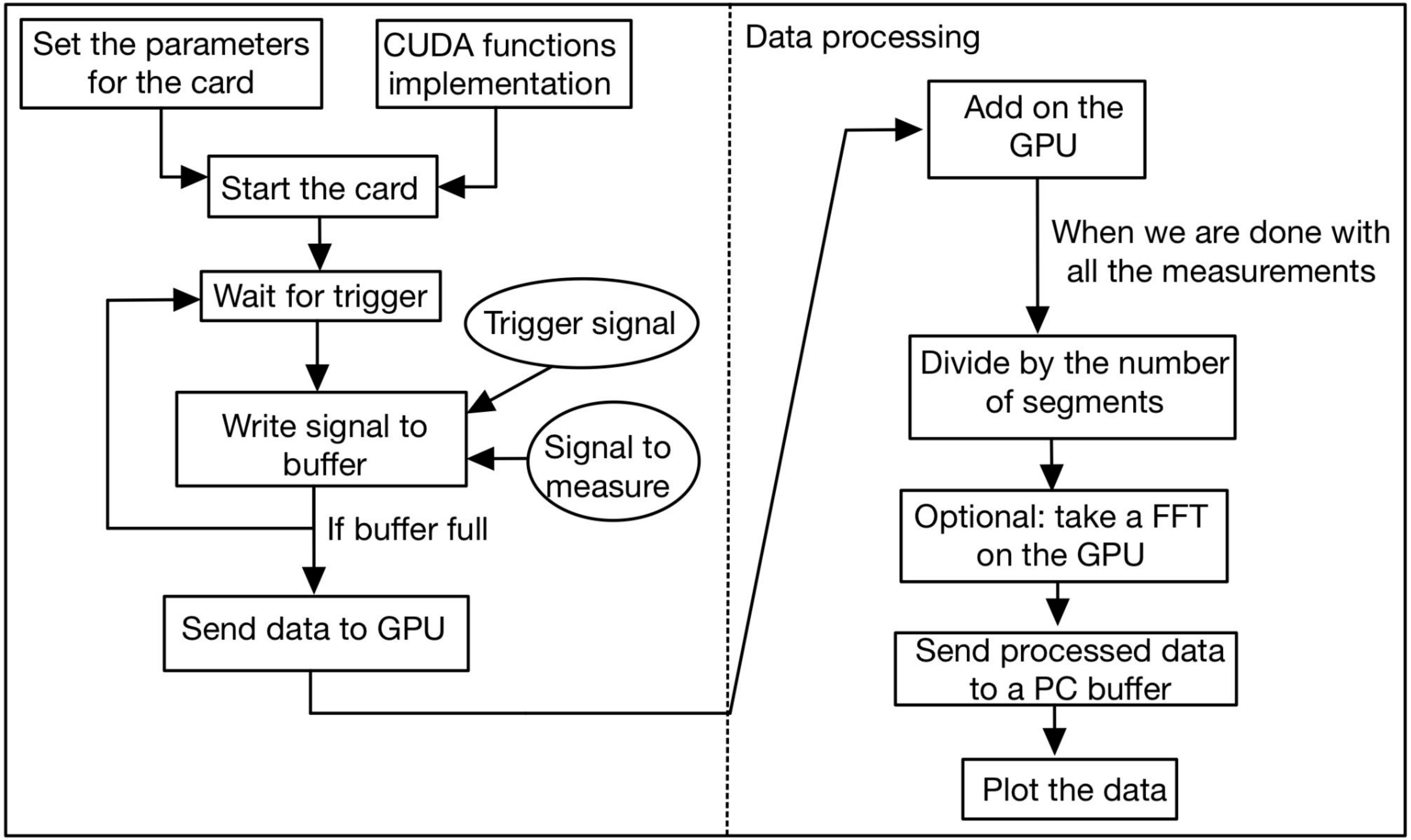
Register

Register value

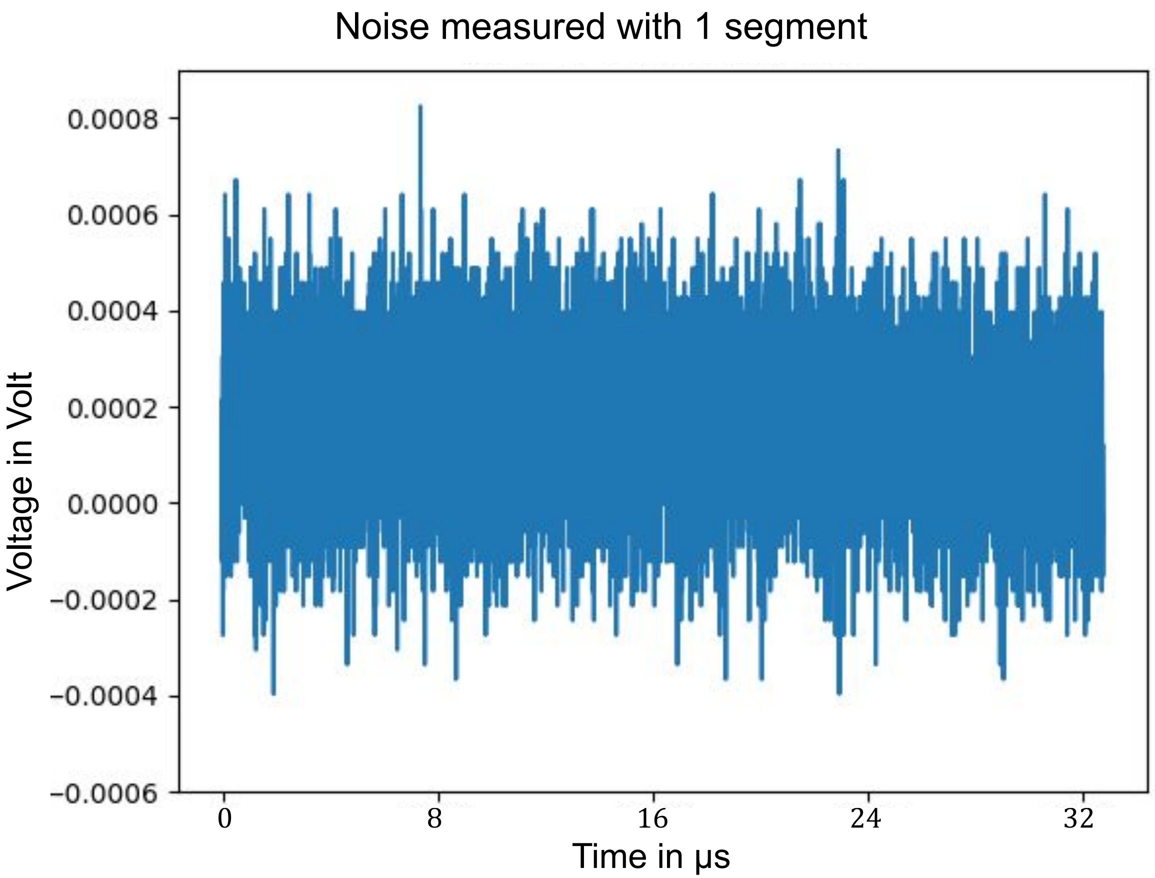
Data Acquisition:

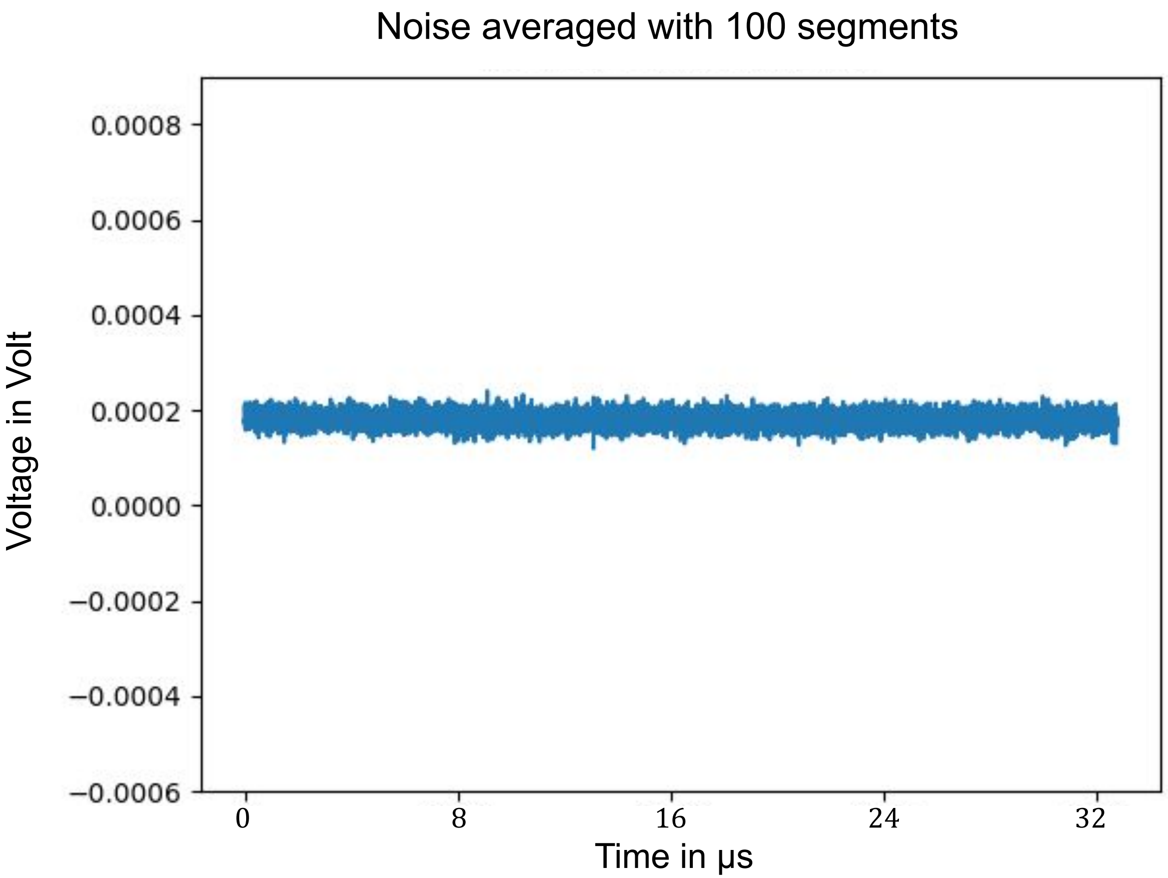


Flow graph of our Program

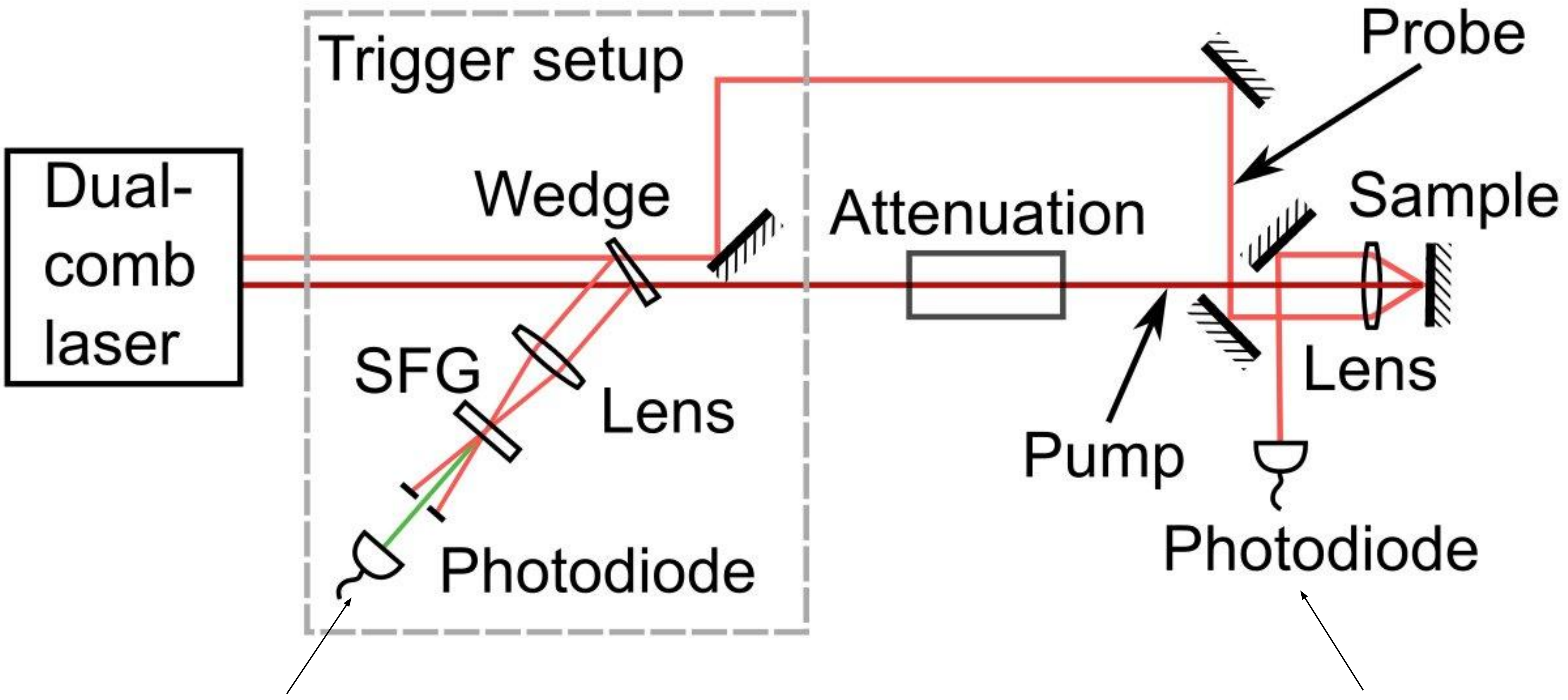


Averaging to reduce noise





The Pump-Probe Experiment



External Trigger channel

Channel 0

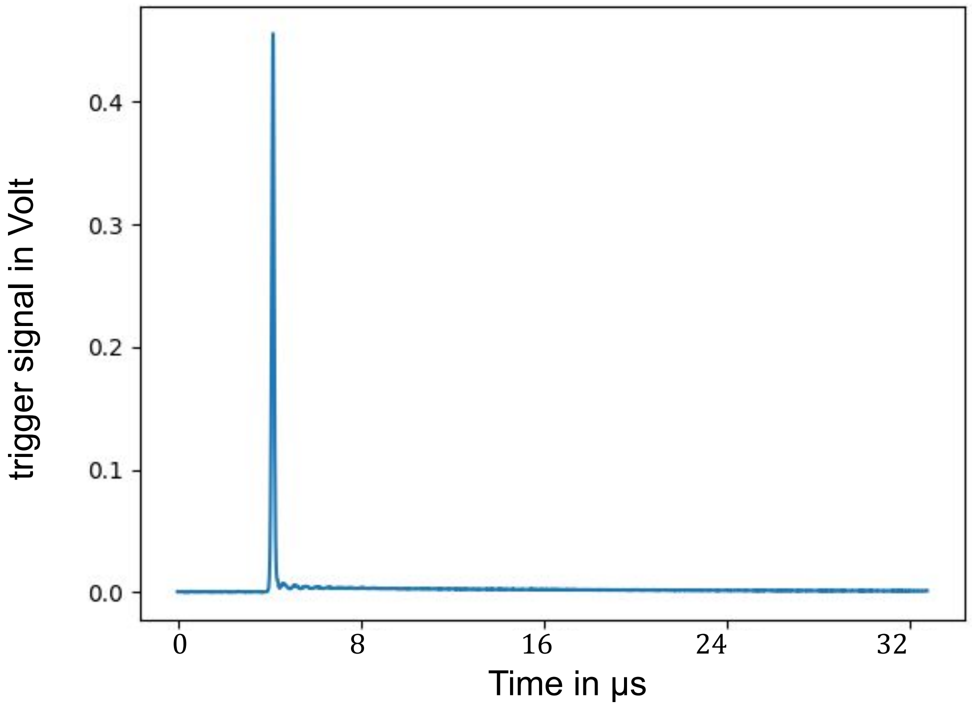
External Triggering on Spectrum Device

- Set the input resistance of the trigger:

  spcm_dwSetParam_i32 (hCard, SPC_TRIG_TERM, 1); // 1 is for 50 Ohm termination

- Set the level of the trigger:

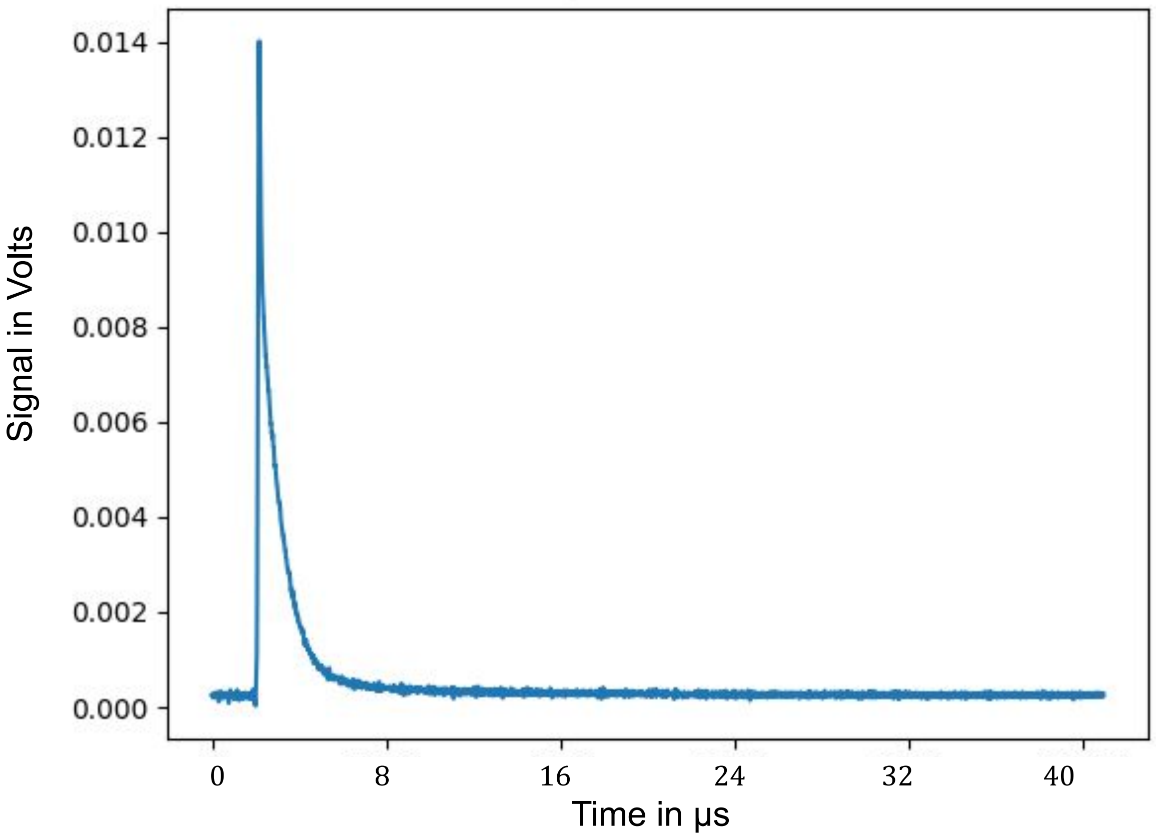
  spcm_dwSetParam_i32 (hCard, SPC_TRIG_EXT0_LEVEL0, ext_trig_level);



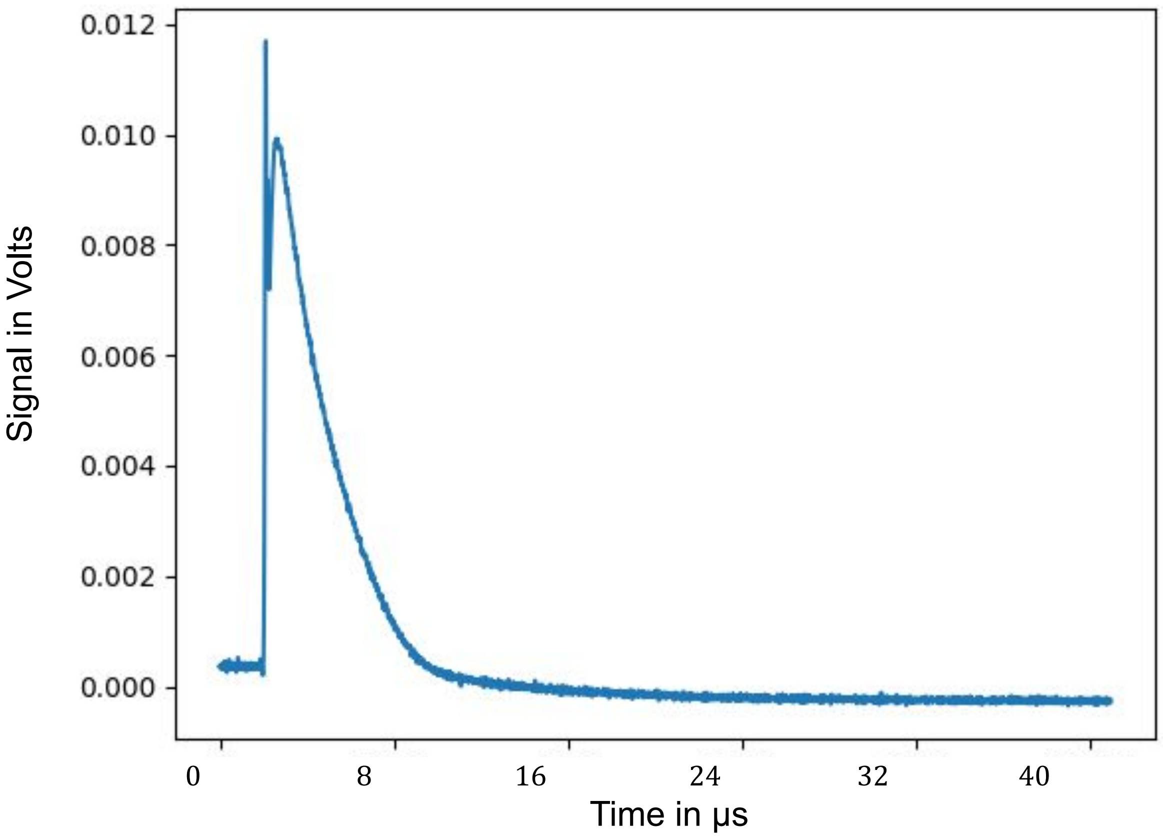
Results

Plots show voltage of reflected probe signal in photodiode: this is proportional to the reflectivity of the SESAM sample

Fluence: 0.2 mJ/cm²



Fluence: 3mJ/cm²



ETH zürich

ETH zürich

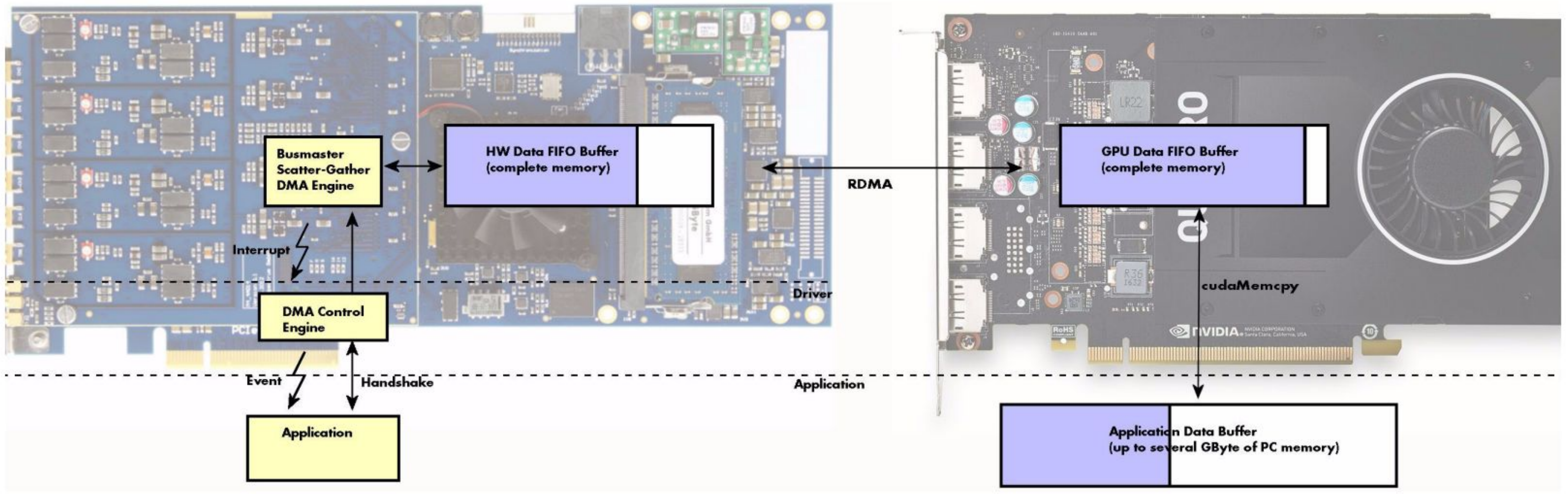
Thanks for your attention, Merry Christmas and a Happy New Year

https://media.cooperation.ch/images/2020/11/09/134704145-5e2adeea-12d4-440e-ab17-8520076e0d55.jpg

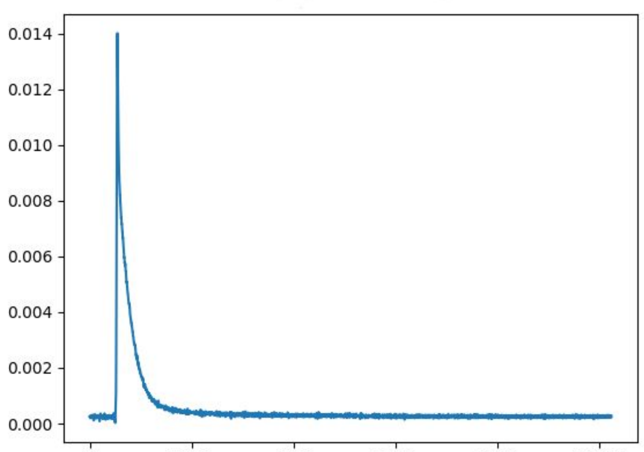
Summary

- Setup and Hardware
  - Software setup
  - Overview SCAPP
- Programming
  - Spectrum API
  - Cuda Toolkit
- Experimental results
  - Experiment
- Outlook
  - can be used for virtually any data processing task which involves a lot of data

SCAPP overview:



Experimental results:



Main Loop:

```
while (qwTotalMem < qwToTransfer){
    if ((dwError = spcm_dwSetParam_i32 (hCard, SPC_M2CMD, M2CMD_DATA_WAITDMA)) !=
ERR_OK){

        if (dwError == ERR_TIMEOUT)
            printf ("\n... Timeout\n");
```

```
else{ //if no problem with the DMA transfer

    //available user storage in the Card

    spcm_dwGetParam_i32 (hCard,
    SPC_DATA_AVAIL_USER_LEN, &lAvailUser); //absolute
    position of the data in the card

    spcm_dwGetParam_i32 (hCard,
    SPC_DATA_AVAIL_USER_POS, &lPCPos);


    if (lAvailUser >= lNumByteInSegment){

    //takes count of how much data we already received to
    know when to stop qwTotalMem += lNumByteInSegment;

    //put the data on the GPU

    cudaMemcpy (pvBuffer_gpu, (char*)pvDMABuffer_card +
    lPCPos, lNumByteInSegment,
    cudaMemcpyHostToDevice);

//scale the input data (form a int16 value to mV)

Scale<<<lSegmentSize /
number_of_threads,number_of_threads>>>
((int16*)pvBuffer_gpu, lIR,
lMaxADCValue,(float*)gpu_buff_b);

//Addition of buffer "a" with the scaled data

vecAdd<<<lSegmentSize /
number_of_threads,number_of_threads>>>
((float*)gpu_buff_a, (float*)gpu_buff_b);

counter += 1;

if(counter== Num_seg){

break; }

// now the processed data is in the host memory and can
be processed further, //e.g. written to disk

// mark the segment as processed

spcm_dwSetParam_i32 (hCard,
SPC_DATA_AVAIL_CARD_LEN, lNumByteInSegment);
```

FFT Code:

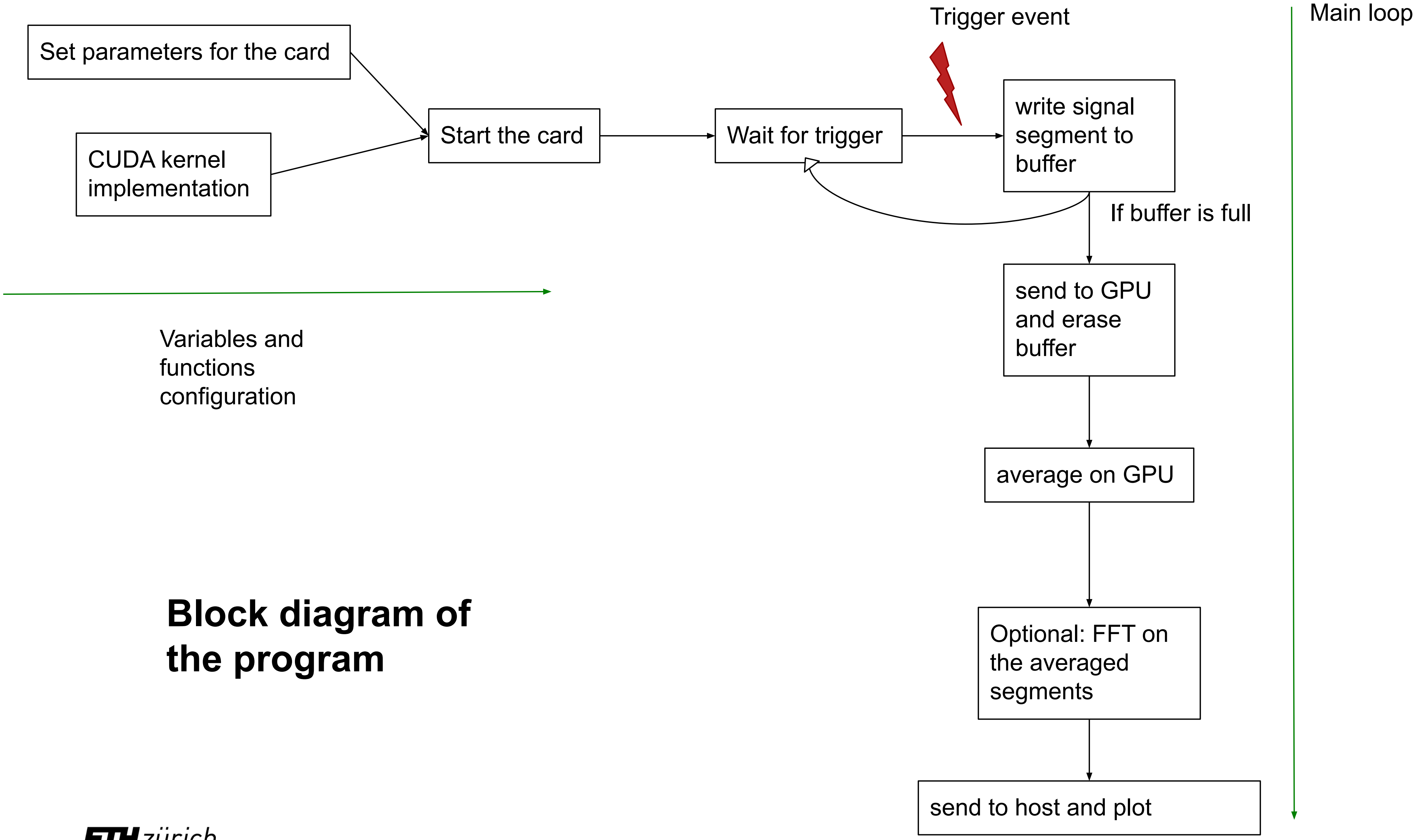
```
cudaMalloc(&data, NumFFTsamples * sizeof(cufftComplex));
if (cudaGetLastError() != cudaSuccess){
    fprintf(stderr, "Cuda error: Failed to allocate\n");
return;
}

if (cufftPlan1d(&plan, lSegmentSize, CUFFT_R2C, BATCH) != CUFFT_SUCCESS){
    fprintf(stderr, "CUFFT error: Plan creation failed");
    return;
}
```

```
// here the FFT is performed

if (cufftExecR2C(plan, (cufftReal*)gpu_buff_a, data) != CUFFT_SUCCESS){
    fprintf(stderr, "CUFFT error: ExecC2C Forward failed");
    return;
}

if (cudaDeviceSynchronize() != cudaSuccess){
    fprintf(stderr, "Cuda error: Failed to synchronize\n");
    return;

}
```

External Trigger specifications :

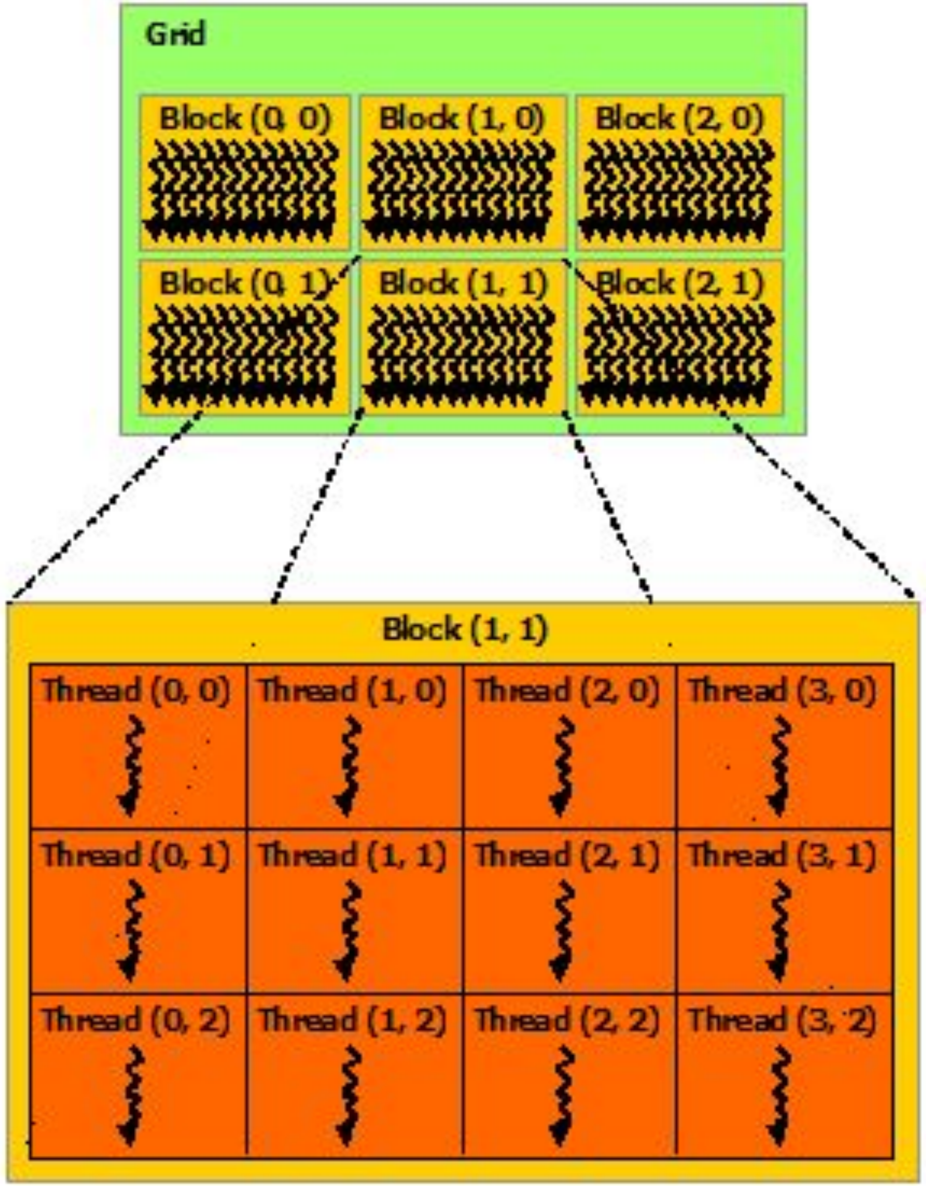
- different input resistances, for different input ranges of Trigger signal:
  - 1kΩ : Full Scale range is ±10V = 20V total
  - 50Ω : Full Scale range is ±2.5V total
- minimum requirement for height of input signal: 2.5% of Full Scale range

Block diagram of the program



Graphic Processing Unit (GPU)

- GPU's consists of thousands of processing cores
- Threads: smallest sequence of programmed instructions that can be managed independently by a scheduler
- to maximise parallelisation of tasks, the GPU groups threads together in so called blocks.
- The computational power of the device is the arranged in a grid of these blocks.



https://www.researchgate.net/figure/CUDA-programming-grid-of-thread-blocks-Source-NVIDIA_fig3_328752788