Terascale detector workshop

Hands-On part

“Simple FPGA-Based Trigger systems”

-

Digital signal processing on FPGAs

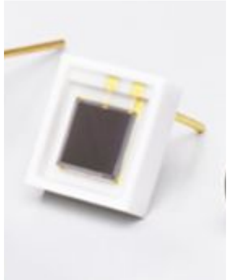
Konrad Briggl, Tigran Mkrtchyan, Vera Stankova

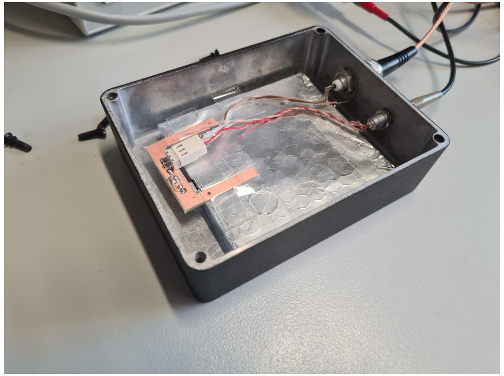
KIP, Heidelberg

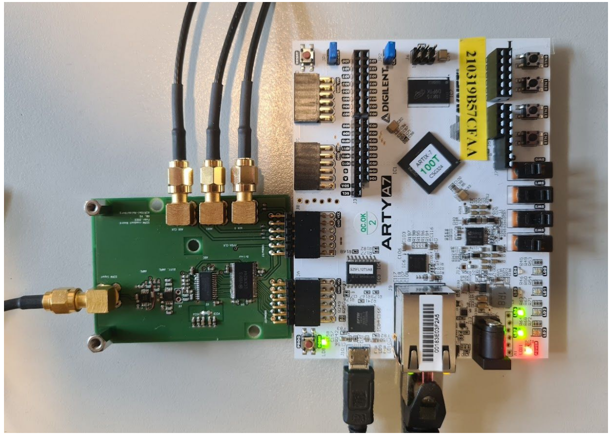
28.2.2023 - 1.3.2023

Setting the scene

- Scintillator coupled to SiPM
- SiPM signal
    - converted to differential voltage signal
    - Minimal pulse shaping to comply with ADC requirements
- Analog signal is digitized with 40MSPS by an ADC
- ADC is read out by an Arty7 FPGA (Xilinx)
- Readout of the FPGA via ethernet (IPbus protocol)



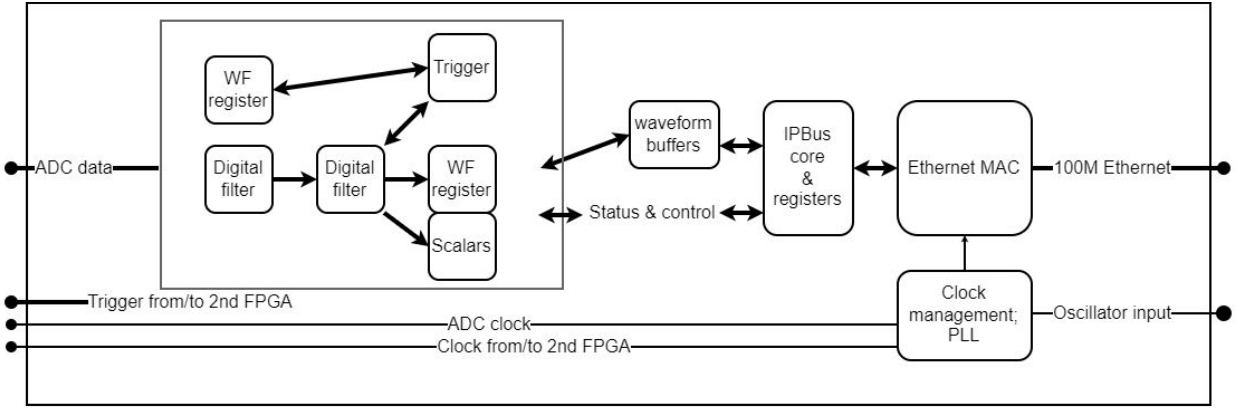




Setting the scene

This hands-on course: Development of FPGA firmware

- Sampling of ADC data
- Triggering
- Digital processing of ADC samples
- Combination of two systems - Trigger & Validation



Digital signal processing - Digital filters

LTI: Linear time invariant systems

- Linear filter response
    - Neglecting effects from dynamic range and quantisation
- Invariant under time shift
    - i.e. not dependent on some global time reference, starting point, etc.

Distinguish between FIR and IIR filter systems

- **FIR**: Output is a function of N last input sample
    - Finite impulse response
- **IIR**: Output is a function of N last input and M last output samples
    - Infinite impulse response

FIR filters

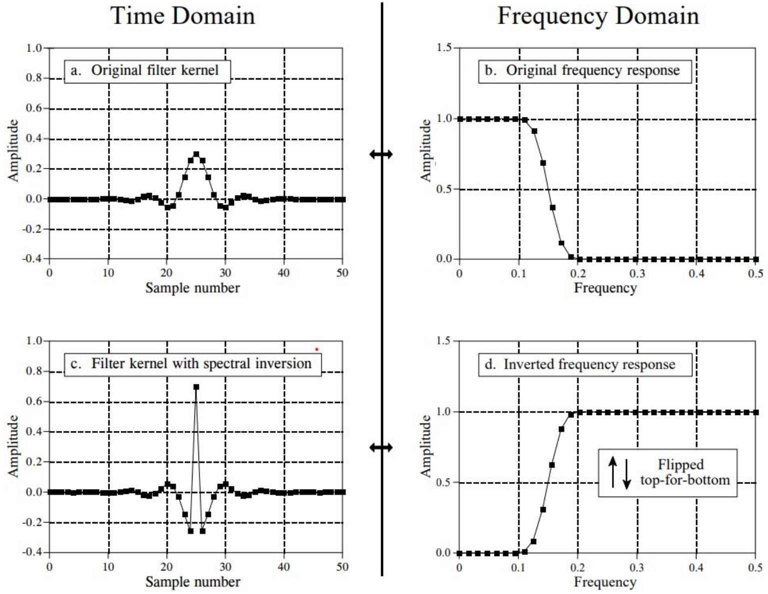
Reaction to an input x[n] is convolution x[n] with filter coefficients h[k]

Coefficients <-> Impulse response

Finite impulse response filters (h[k] has a finite length)

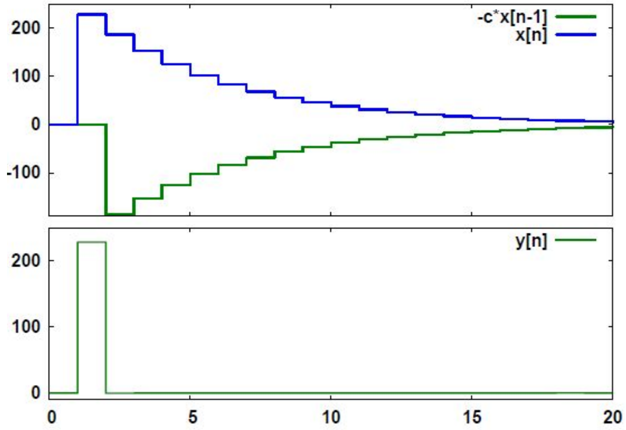
Convolution with discrete input series
    Product in frequency domain

$$y[n] = h * x = \sum_{k=0}^{M} h[k]\, x[n+k]$$

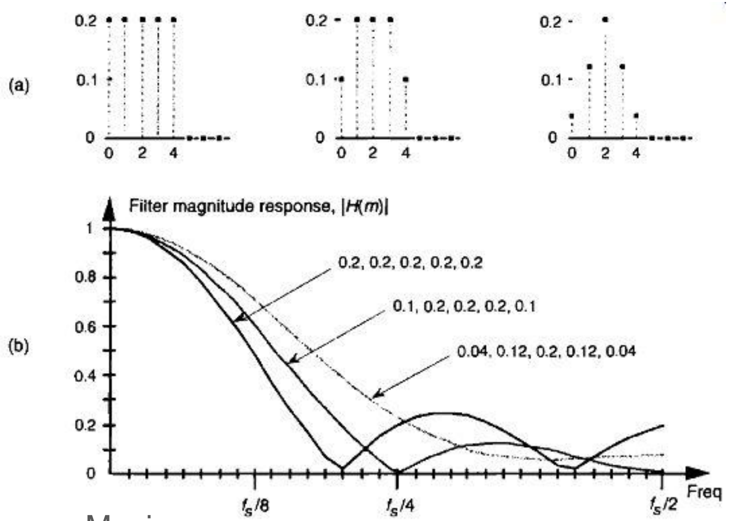


Examples of FIR filters

- Low pass / High pass
  Moving average
- Tail cancellation
- Correlation filters



Tail cancellation (1st order FIR)



Moving average

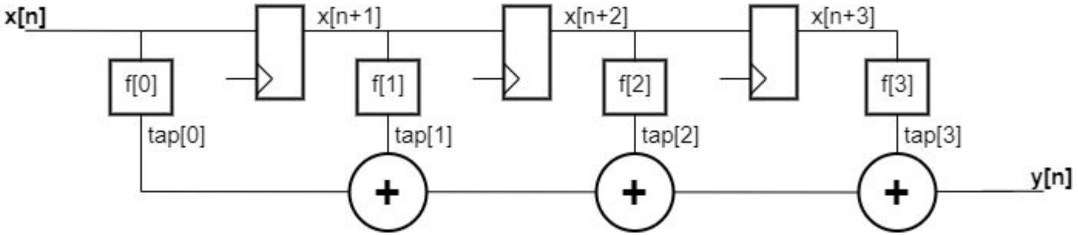
FIR filter implementations

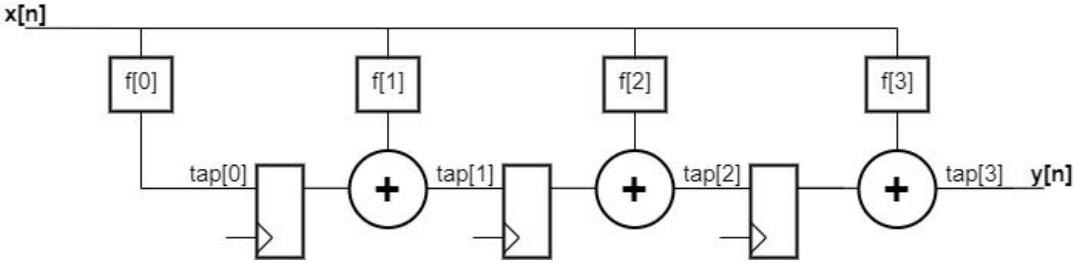
Canonical implementation

- Summation over N Taps
- N Multiplications
- N registers

Equivalent:

- Transposed implementation
- Summation is pipelined
- If coefficients are known and equal, can be shared
- Reduced resources



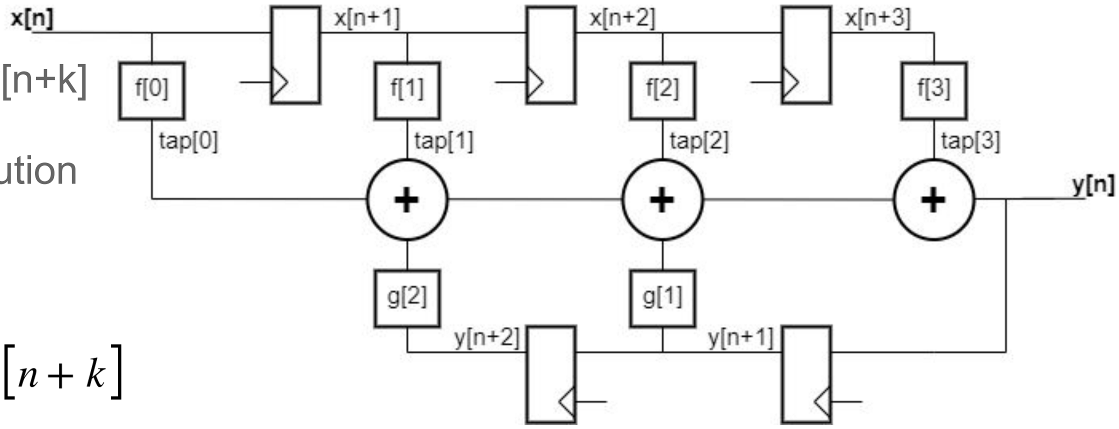


IIR digital filters

Infinite impulse response,
achieved by feedback of output, y[n+k]
Impulse response:
Only limited by quantization resolution

Closer to filters in the analog world

$$y[n] = \sum_{k=0}^{M} h[k]\, x[n+k] \; + \; \sum_{k=0}^{N} g[k]\, y[n+k]$$



Obtain long memory with small number of taps
(and thus sensitivity for low frequencies)

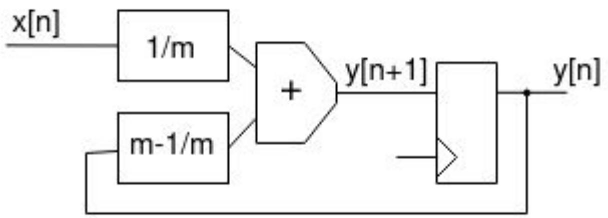
Possibility of divergence & oscillation

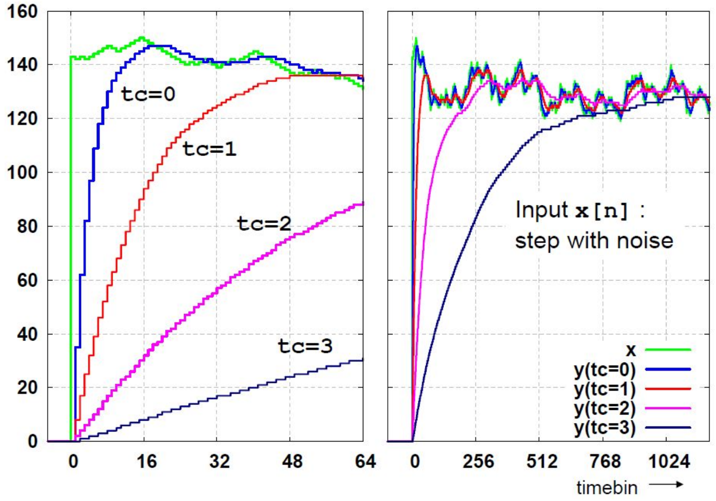
IIR filter: Lossy integrator & Optimization

Example of a single-tap IIR filter

- Low-pass filter with long integration time
- Not efficiently done as FIR

We want: $$y[n+1] = \frac{m-1}{m}\,y[n] + \frac{1}{m}\,x[n]$$

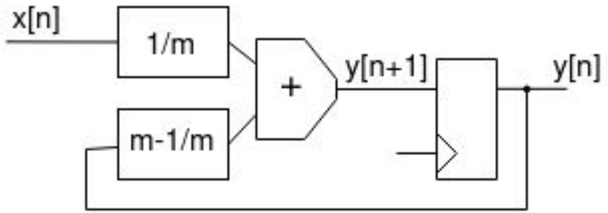




IIR filter: Lossy integrator & Optimization

Example of a single-tap IIR filter

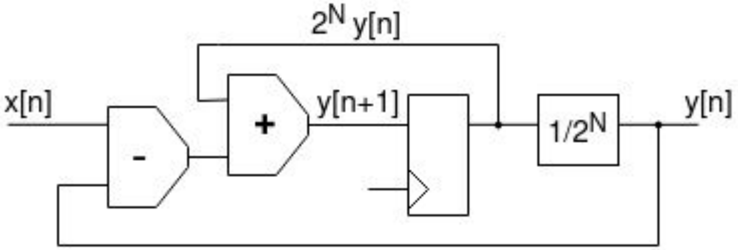
Low-pass filter with long integration time
Not efficiently done as FIR



We want: $$y[n+1] = \frac{m-1}{m}\, y[n] \; + \; \frac{1}{m}\, x[n]$$

But: division / multiplication by non-binary fractions

costly on FPGAs, to be minimized.



Rewriting: $$y[n+1] \cdot m = y[n] \cdot m \; - \; y[n] + x[n]$$

If m = 2^N with natural N:
Implement using only addition, subtraction and bit shift operations

VHDL as a hardware description language

V (VHSIC, Very High Speed Integrated Circuit) **HDL** (Hardware Description Language)

- Developed in the 80’s
- Hardware description language,
- Generic (portable); also used for simulation/creation of testbenches
- Hierarchical designs
- Allows to describe analog/digital and mixed mode circuits

VHDL - file structure

Everything is an entity

- hierarchical design
- definition of signals through hierarchies (ports)
- Architecture: Implementation Signal declarations
- Hardware description code

```
LIBRARY library_name;
USE library_name.package_name. package_parts;

ENTITY entity_name IS
     PORT ( port_name : port_mode port_type;
            port_name : port_mode port_type;
            port_name : port_mode port_type;
                .
                .
                .
          );
END entity_name;

ARCHITECTURE archi_name OF entity_name is
   declarations
     BEGIN
          code (sequential or concurrent statements)
          .
          .
          .
END archi_name
```

VHDL: Combinatorial logic, signal assignments

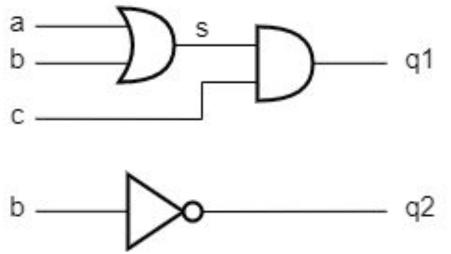
Signal assignments

- Corresponds to “connecting wires”
- Arithmetic operations on signals
  Very explicit operators:
  depending on signal type & Interpretation

- Inside or outside process block (later)

This example:

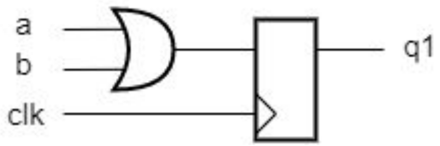
- 2 signal paths implemented
  q1 and q2 are processed at the same time
- code ordering does not matter here




```
q1 <= (a or b) and c;
q2 <= not b;
```

VHDL: Process blocks & Sequential logic

Sequential logic:
Introduces discrete time steps for the computation
- Defined by clock transitions (e.g. ‘0’ -> ‘1’)
- Results in storage elements
  (Flip-Flops, Latches) to be implemented



Assignment realized only if clock transition is seen

Signal changes only if clock transition

```
p_seq1: process(clk)
begin
        if(rising_edge(clk)) then
                q1 <= a or b;
        end if;
end process;
```

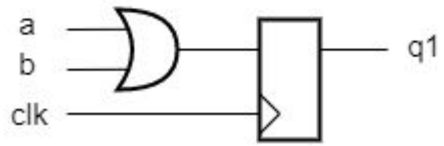
VHDL: Process blocks & Sequential logic

Process block: Basic building block of behavioral descriptions

- Sensitivity list at beginning of process defines execution of simulation: Compute only when signals in list change

- Can contain sequential statements
    - e.g. “if (rising_edge(clk)) then …”
- Signals can only be assigned in one process

- All assignments are evaluated at end of process
    - Multiple assignments, esp. conditional assignments: override previous assignments

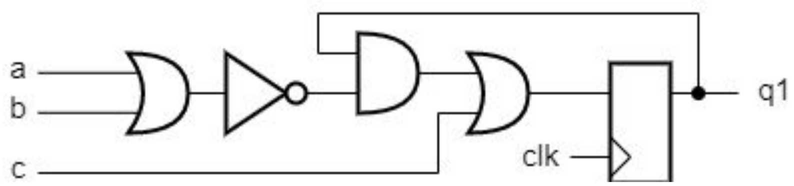



```
p_seq1: process(clk)
begin
        if(rising_edge(clk)) then
                q1 <= a or b;
        end if;
end process;
```

VHDL: Process blocks & Sequential logic

Assignment realized only if clock transition is seen

- Signal changes only if clock transition
- Using feedback (implicit): Keep old value
- Assignment preference by code order




```
p_seq2: process(clk)
begin
        if(rising_edge(clk)) then
                if (a='1' or b='1') then
                        q1 <= '0';
                end if;
                if (c = '1') then
                        q1 <= '0';
                end if;
        end if;
end process;
```

VHDL: Assignment loops, variables

Assignments can also be written as a loop

Generalization of code, less replications
Generally results in repeated implementation

For signals:
“Evaluated at the end” still applies.
Complex calculus, e.g. summation,
needs to be explicit

Solution: **VHDL variables**
- Local to process
- Keep “Memory”
- Assignment by “:=” operator

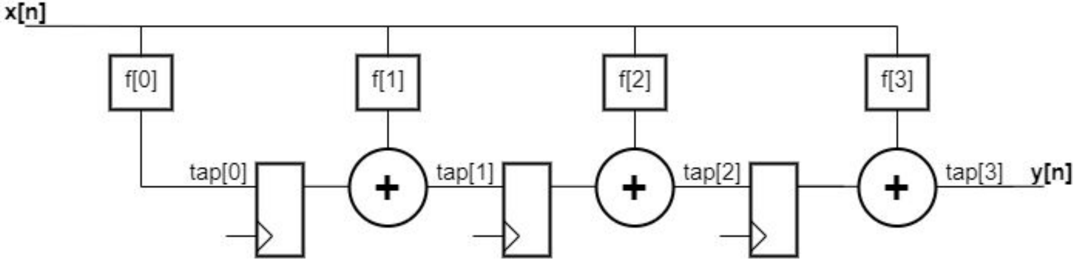
```
signal a : some_signed_array (4 downto 0);
signal b : some_signed_array (4 downto 0);

p_for: process(clk)
begin
        for i in (4 downto 0) loop
                b(i) <= a(i) + 1;
        end loop;
end process;


p_var1: process(clk)
variable sum  : integer;
begin
        a:=1;
        a:= a + 2;
        somesignal <= a; - - signal will be 3
end process;
```

VHDL: FIR filter implementation

4-Tap FIR filter in VHDL:

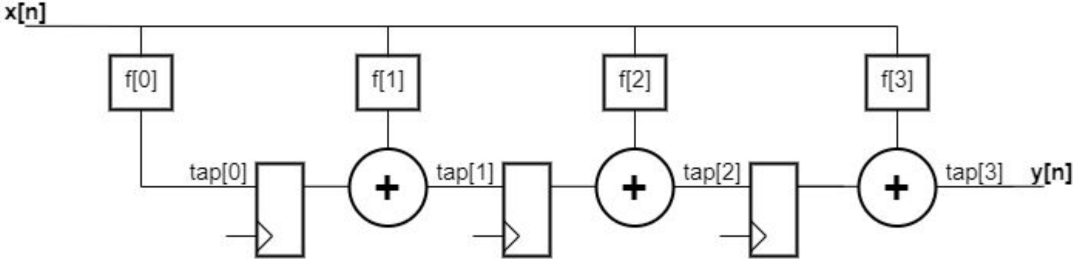



```
p_FIR: process(clk)
begin
        if(rising_edge(clk)) then
                tap(0) <= f(0) * x;
                tap(1) <= tap(0) + f(0) * x;
                tap(2) <= tap(1) + f(1) * x;
                tap(3) <= tap(2) + f(2) * x;
        end if;
        y <= tap(3);
end process;
```

VHDL: FIR filter implementation

More generic implementation
(regarding number of taps)

```
p_FIR: process(clk)
begin
        if(rising_edge(clk)) then
                tap(0) <= f(i) * x;
                for (i in 1 to tap'high) loop
                        tap(i) <= tap(i-1) + f(i) * x;
                end loop;
        end if;
        y <= tap(tap'high);
end process;
```

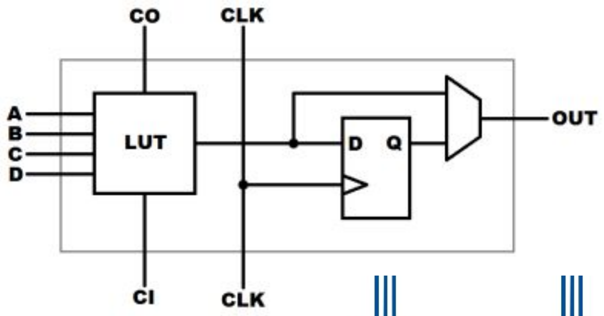


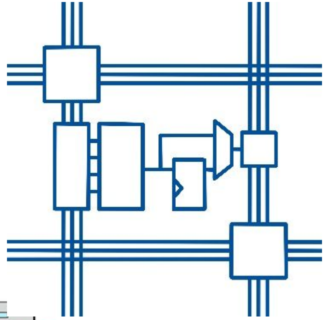
FPGAs

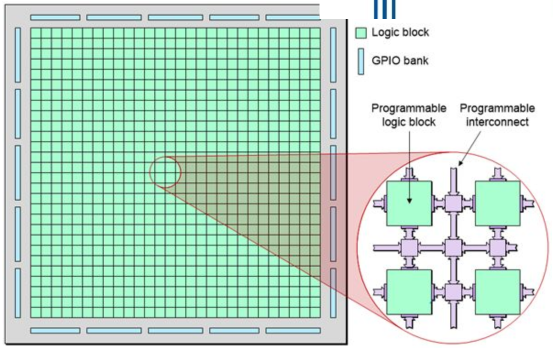
Logic block as basic building block
- LUT + Sequential cell(s)
- matrix describing behaviour
- esp. programmable routing
- matrix describing interconnection

Programmable logic
- Hard blocks and programmable cells
- optimized for market reach

Additional specialized blocks
- fast transceivers, plls,
- complex calculation, memory,
- ARM processors …







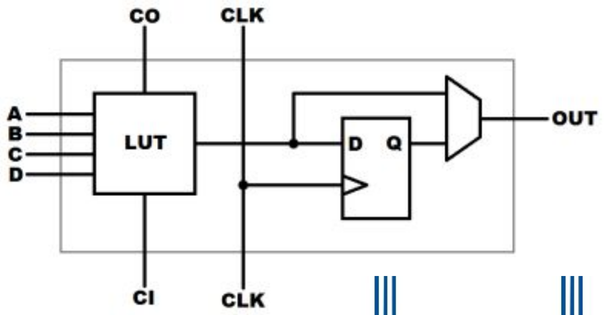
Bird's-eye view of FPGA

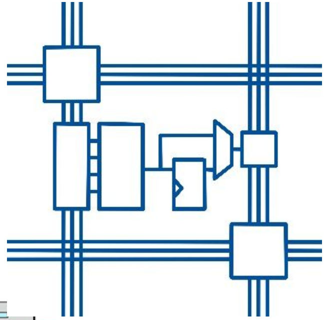
FPGA build process

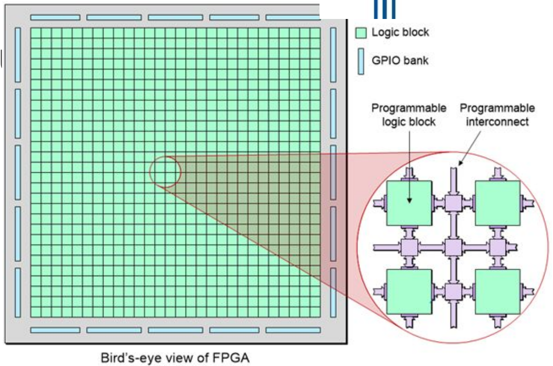
Translate generic VHDL to FPGA bitfile

- Mapping of vhdl assignments
  -> generic digital description (Synthesis)
- Mapping to FPGA building blocks (Mapping)
- Mapping to physical cells on FPGA (Placement)
- Connection of the blocks: Routing of the design
- Generation of a “bit file” : Can be uploaded to the FPGA

- Driven by constraints
  Timing (clock speed and relations vs propagation & ro
  IO constraints (pins on PCB, type of IO)

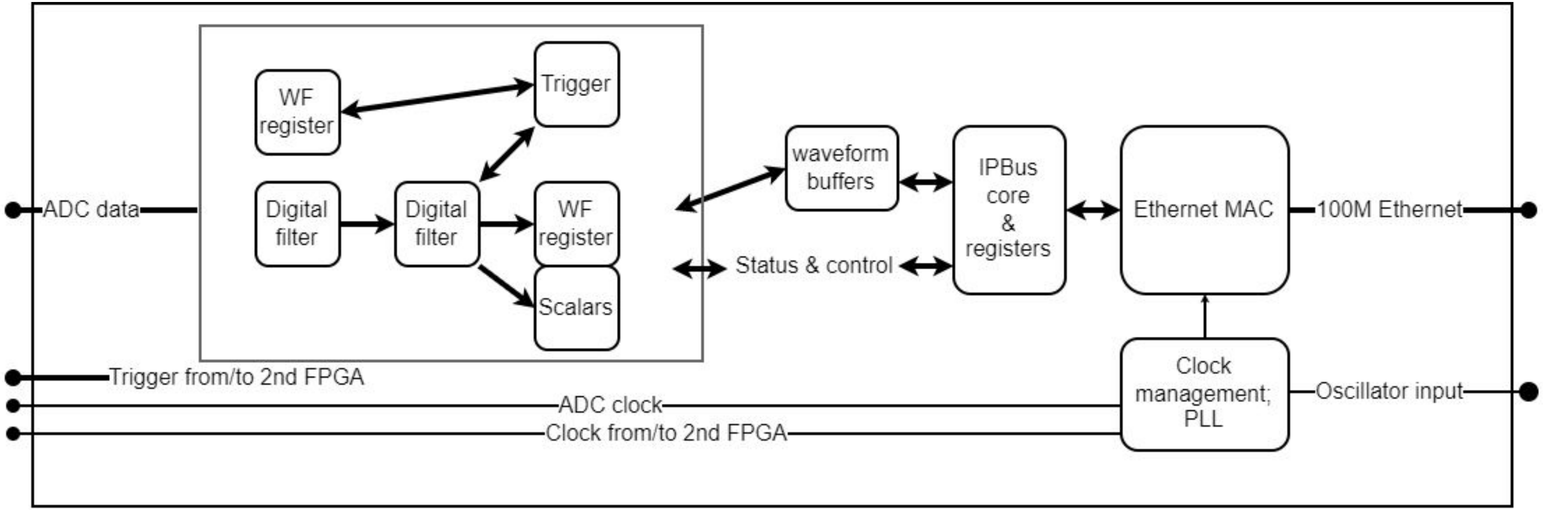






Hands-On: The starting point

- What is there?
- How to get started -> First exercise
- 7 Exercises ; we will discuss solutions on the way
- Safety precautions



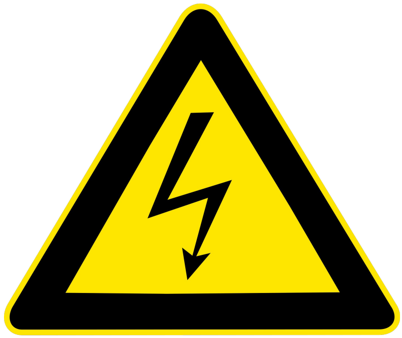
Safety and Hazards

- SiPMs need bias voltage to function (Here: up to 72V).
- The Supplies used can generate up to 120V and relatively high currents.
- Secured against direct touching if not tempered with.

Please:

- Do not touch electrical assemblies.
- Do not modify the connections of the system.
- Do not change power supply settings without consulting us.

A signature sheet will be handed around



Additional resources as starting point

Both topics have plenty of good resources online
Finding an optimized filter is a science by itself, many books cover this.

Digital signal processing

- Analog DSP book (Link: Chapter on digital filters)
  https://www.analog.com/media/en/technical-documentation/dsp-book
- Lecture by V. Angelov, Uni Heidelberg (also on VHDL)

VHDL:

- Lots of tutorials online, e.g. nandland