

# FPGA Tips and Tricks for Nuclear Physics

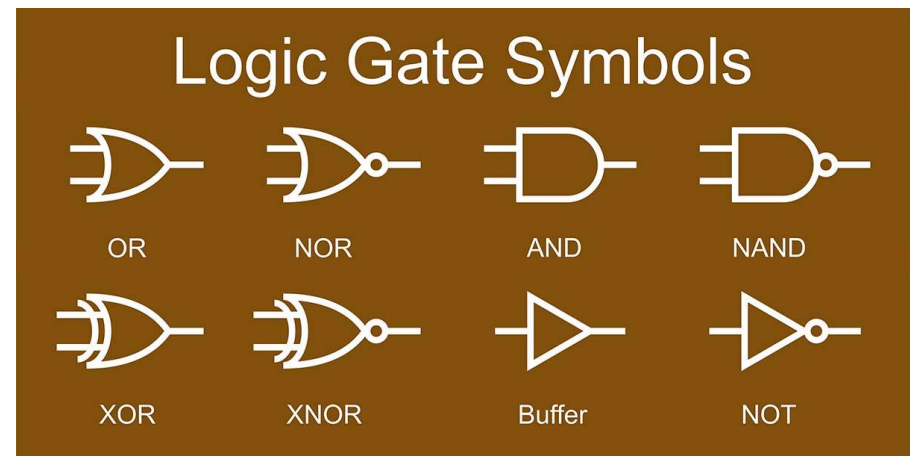
Marc-André Tétrault

IEEE NPSS School of Application of  
Radiation Instrumentation Dakar -  
Senegal

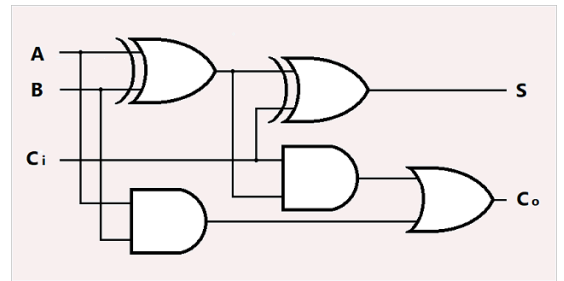
November 21st 2022

# Controlling and using data

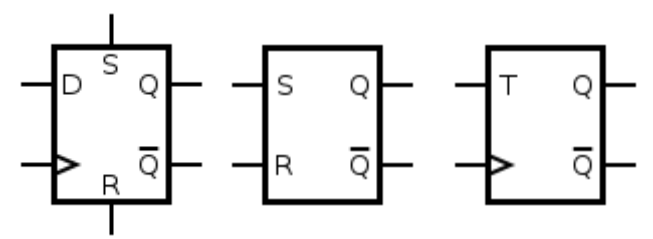
- Logic circuits are the building blocks for the digital age
  - Basic: OR, AND, NOT, XOR, Latch, Flops
  - Memories
  - Intermediate:
    - Adders
    - Comparators
    - Decoders
    - Multipliers
    - Accumulators
  - Advanced block
    - Video Decoder
    - Neural Networks
    - Crossbar (ethernet switch)
    - Processor
    - GPU



[www.nutsvolts.com](http://www.nutsvolts.com)



Full adder - [makecode.microbit.org](http://makecode.microbit.org)



Latches and Flops - wikipedia

# Implementing fully customized digital circuits

- Discreet components
  - For small projects, only a few gates
- Application Specific Integrated Circuit (ASIC)
  - Fully customizable, but very long, challenging and costly to build and validate
  - 10% of your time on the architecture and the logic design
  - 20% of your time on the physical implementation (mapping to chosen technology)
  - 70% of your time on making sure the specification, the implementation and the intent match (verification)
- FPGA
  - « Construction box » with many parts you can glue together in a design software
  - 50% of your time on the architecture and the logic design
  - 50% of your time on making sure the specification, the implementation and the intent match (verification)

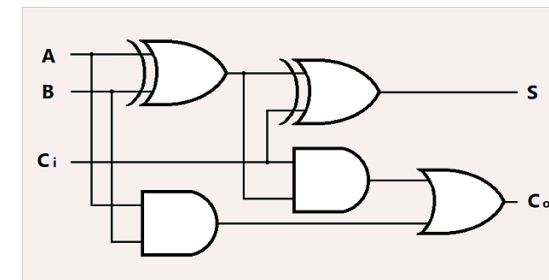
- Looking into different physics experiments, the same detection concepts are re-used in many places
  - But each experiment needs slightly different detector types, geometry, specifications, etc
  - Often the overall experiment uses a combination of very different detectors
- Detectors generate large amounts of data
  - 1- To keep everything, you need a high performance device to extract and record the data
  - 2- To find only the essential data, you need a high performance device to extract and record the data
- Systems use 100s to 1Ms of channels
  - The device has to support working in parallel
  - And generate a lot of data!
- We learn about physics and the experiment as we go, so having programmable hardware helps us try different ideas.



# Simple design example – Full adder

- Define your objective : add two 1-bit values and consider a carry-in
- Define your inputs/outputs
- Fill out a truth table
- Use math to simplify/reduce
- Draw circuit
- Implement
- Verify (simulate)

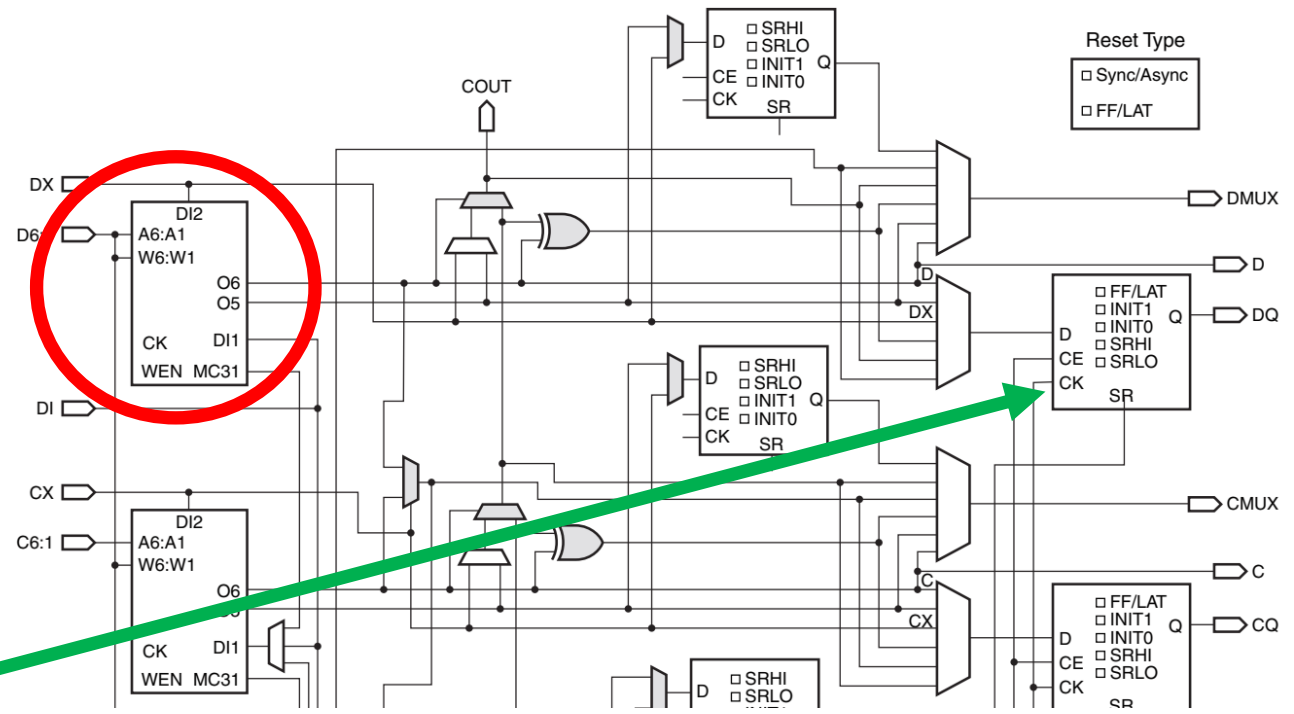
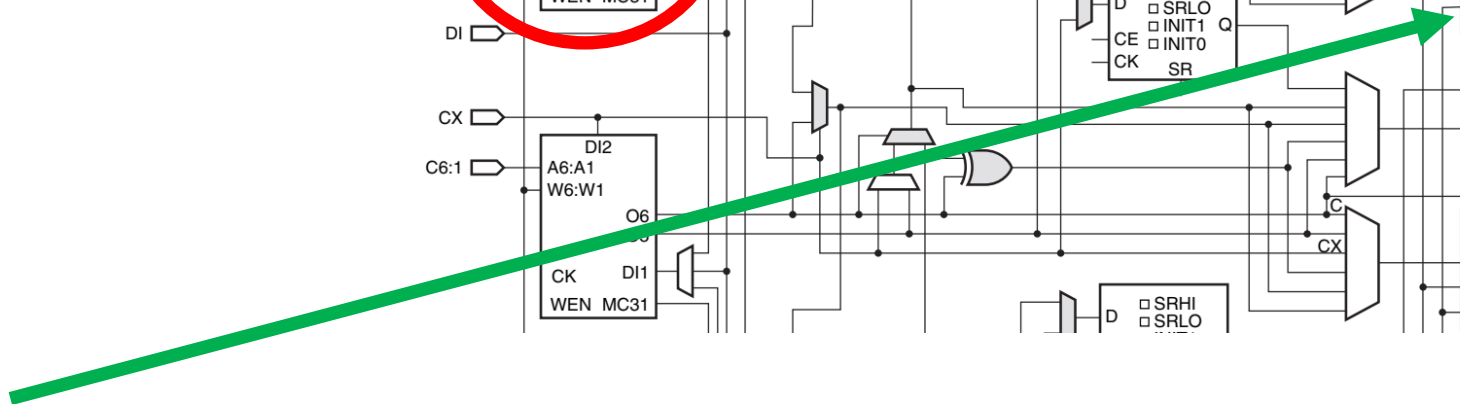
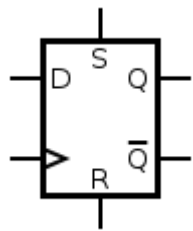
X	Y	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



# FPGA approach

- Instead of logic gates, uses look up tables to implement user intent
- Synthesis tool converts user logic to LUT

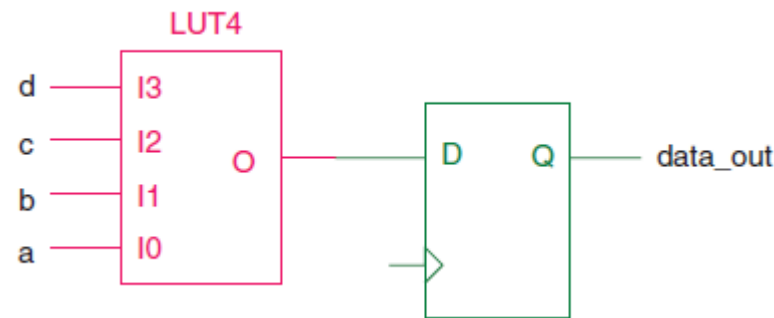
- Adding time



# Understand the size of your Look-Up Tables

- Case of 4-input LUTs

```
1
2 process (clk)
3 begin
4     if (clk'event and clk = '1') then
5         ChannelTrigger <= MasterEnable and ChannelEnable
6             and OverThreshold and not ChannelBusy;
7     end if;
8 end process;
```

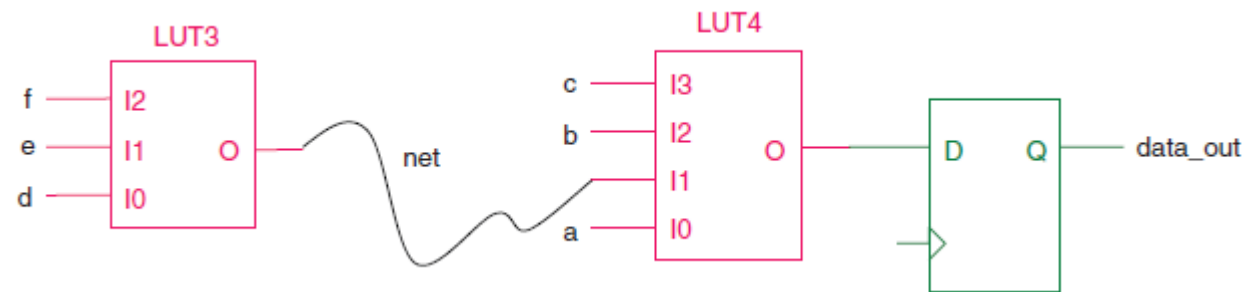


- K. Chapman, “Get your Priorities Right – Make your Design Up to 50% Smaller”, Xilinx White Paper wp275
- UG901 - Vivado Design Suite User Guide: Synthesis (UG901) (v2022.2)



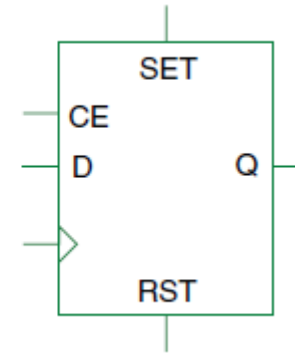
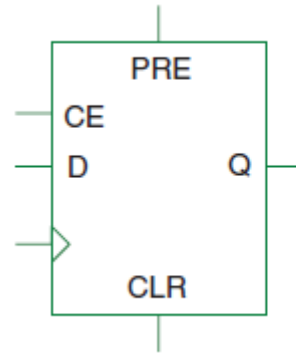
# Code while considering the size of your Look-Up Tables

- Case of 4-input LUTs
- ... but with 6 inputs?



# Coding while knowing the hardware - Flops

- Flip-Flop optional controls
  - Reset/Clear
  - Set/Preset
  - Chip Enable
- What is the difference between Reset and Clear?

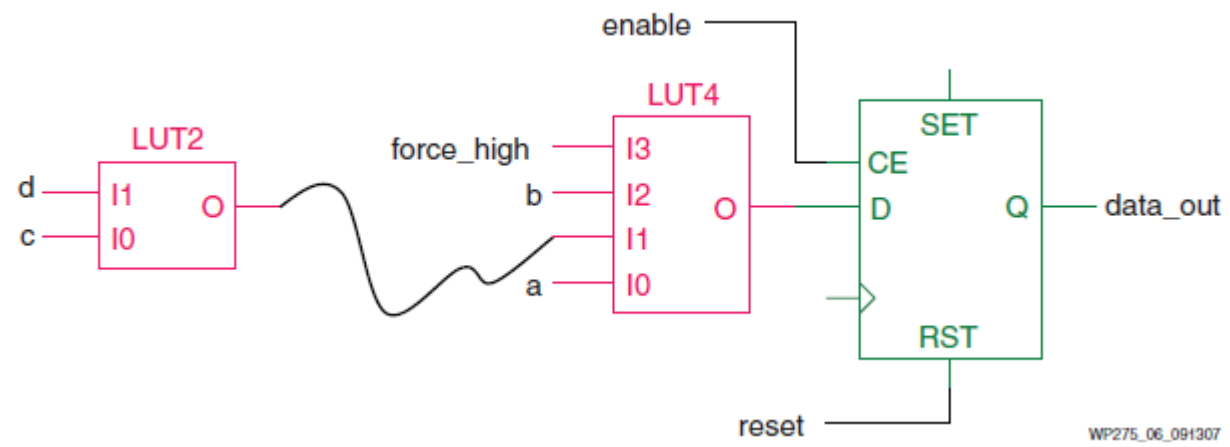


```

1
2 process (clk, reset)
3 begin
4     if(reset = '1')then
5         data_out <= '0';
6     elsif(clk'event and clk = '1') then
7         if(enable = '1')then
8             if(force_high = '1')then
9                 data_out <= '1';
10            else
11                data_out <= a and b and c and d;
12            end if;
13        end if;
14    end if;
15 end process;

```

- ← Reset (asynchronous)
- ← Clock
- ← Enable
- ← Set (synchronous)
- ← Logic Operation



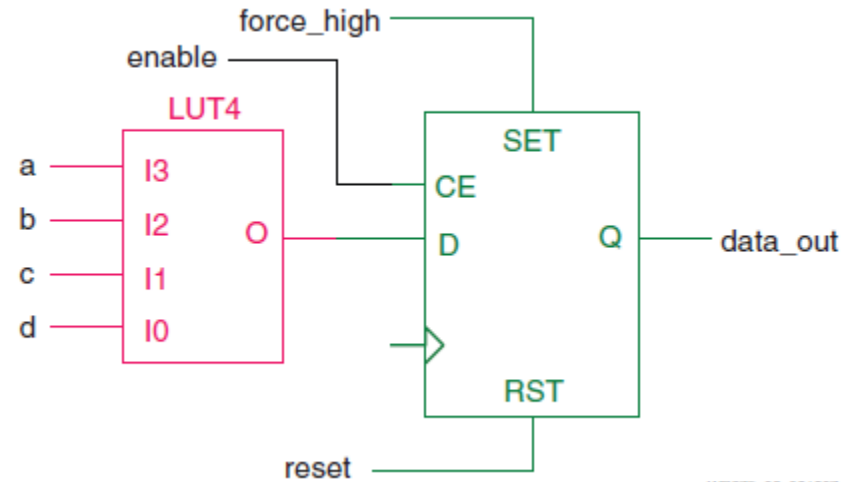
WP275\_06\_091307

```


1
2 process (clk, reset)
3 begin
4     if (clk'event and clk = '1') then
5         if (reset = '1') then
6             data_out <= '0';
7         elsif (force_high = '1') then
8             data_out <= '1';
9         elsif (enable = '1') then
10            data_out <= a and b and c and d;
11        end if;
12    end if;
13 end process;
14
15

```

- ← Clock
- ← Reset
- ← Set
- ← Enable
- ← Logic Operation



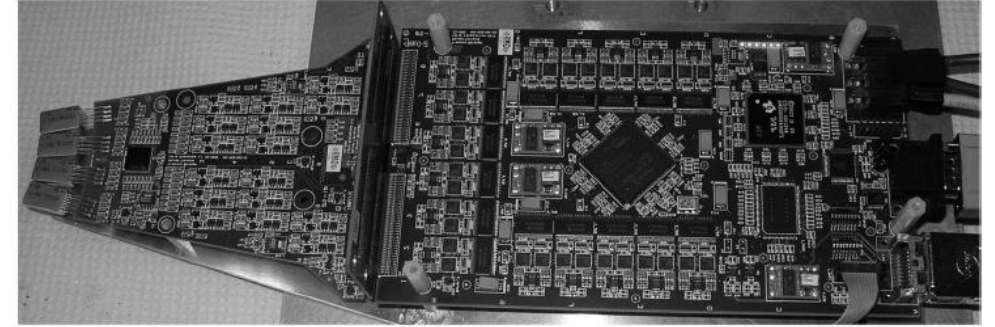
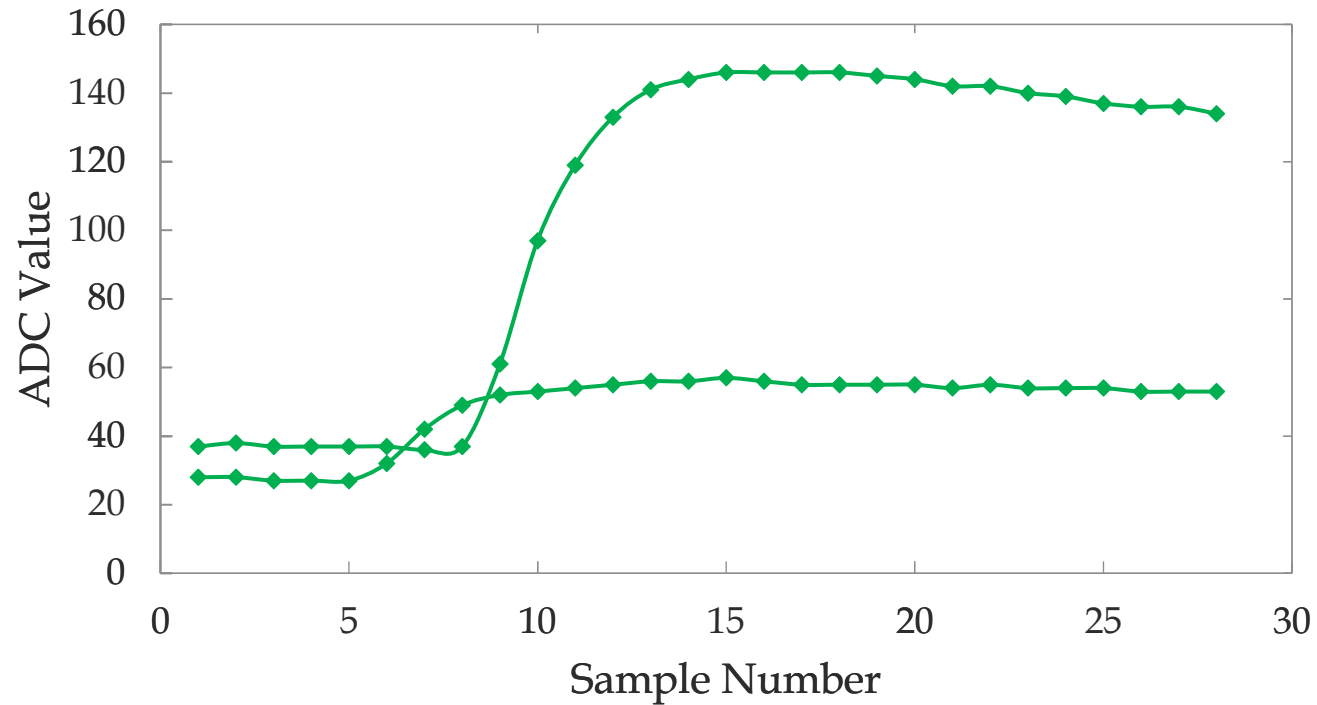
WP275\_08\_091307

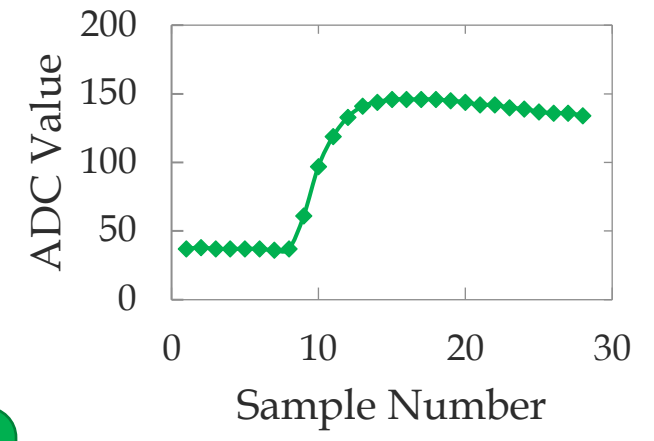
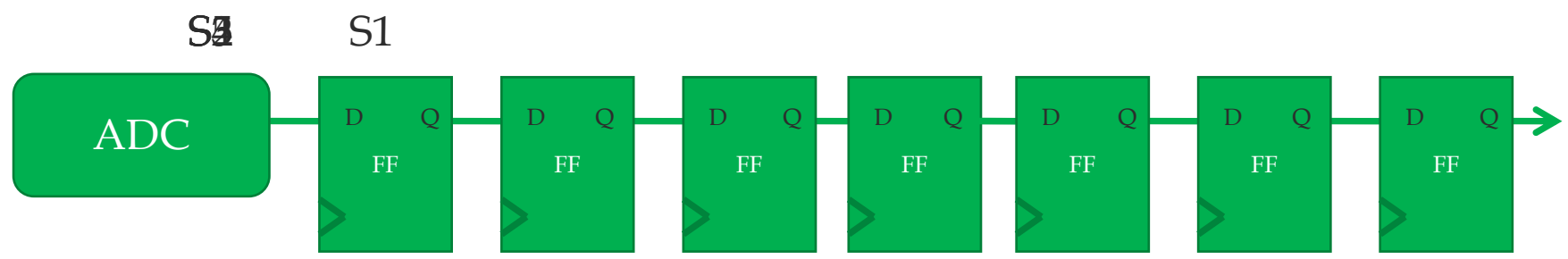


# **Basic example in radiation detection instruments**

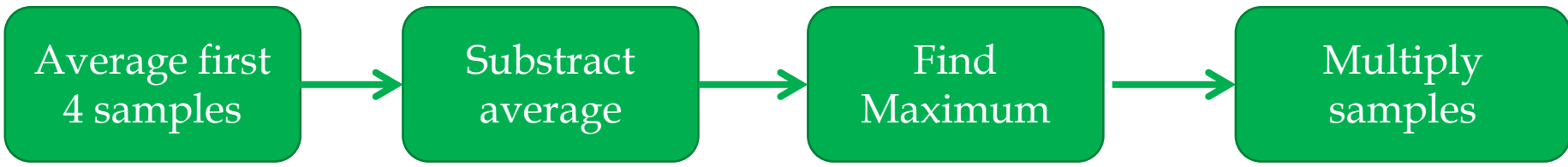
# Real world example

- Detector: Scintillator crystal, APD, Analog circuit, ADC
  - FPGA captures samples on a rising edge.
  - For signal processing, often need to « normalize » signals



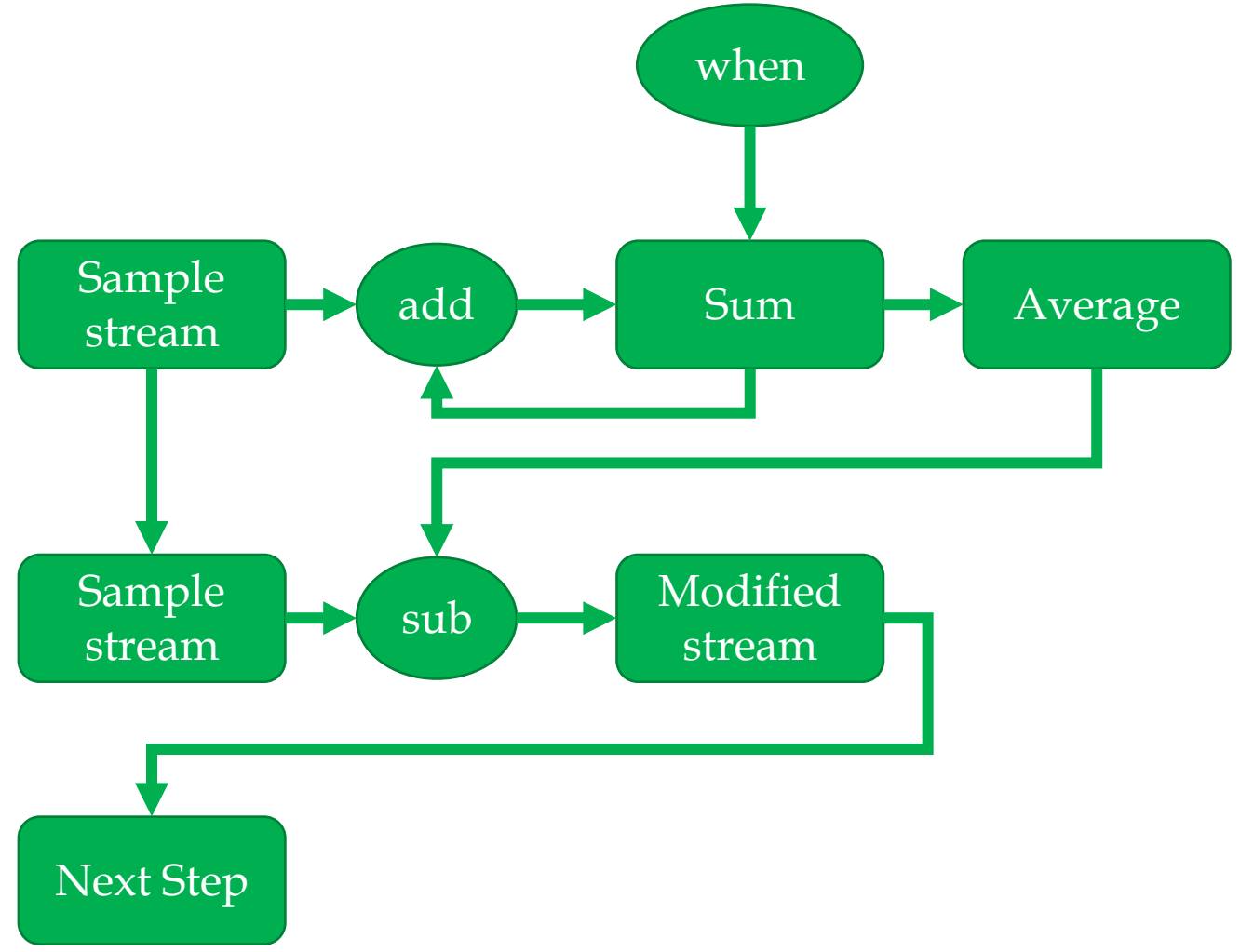
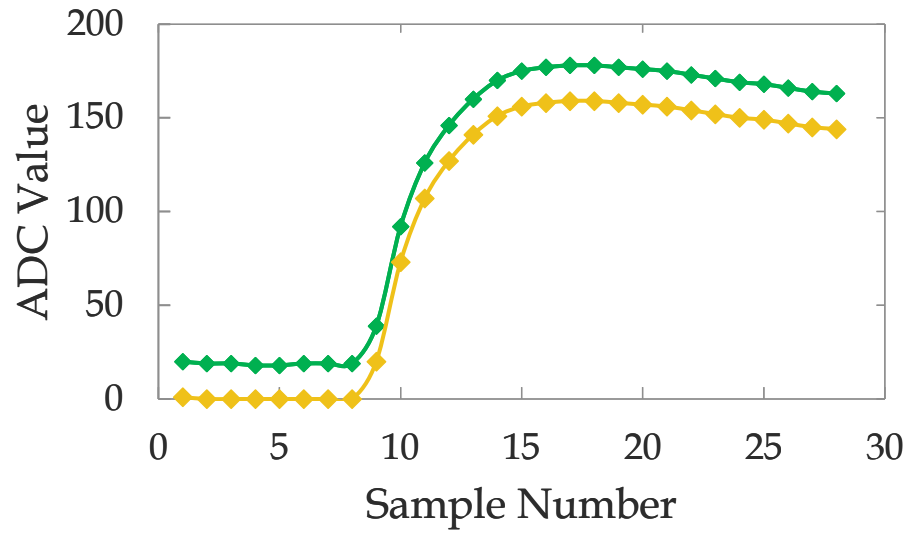


Normalization



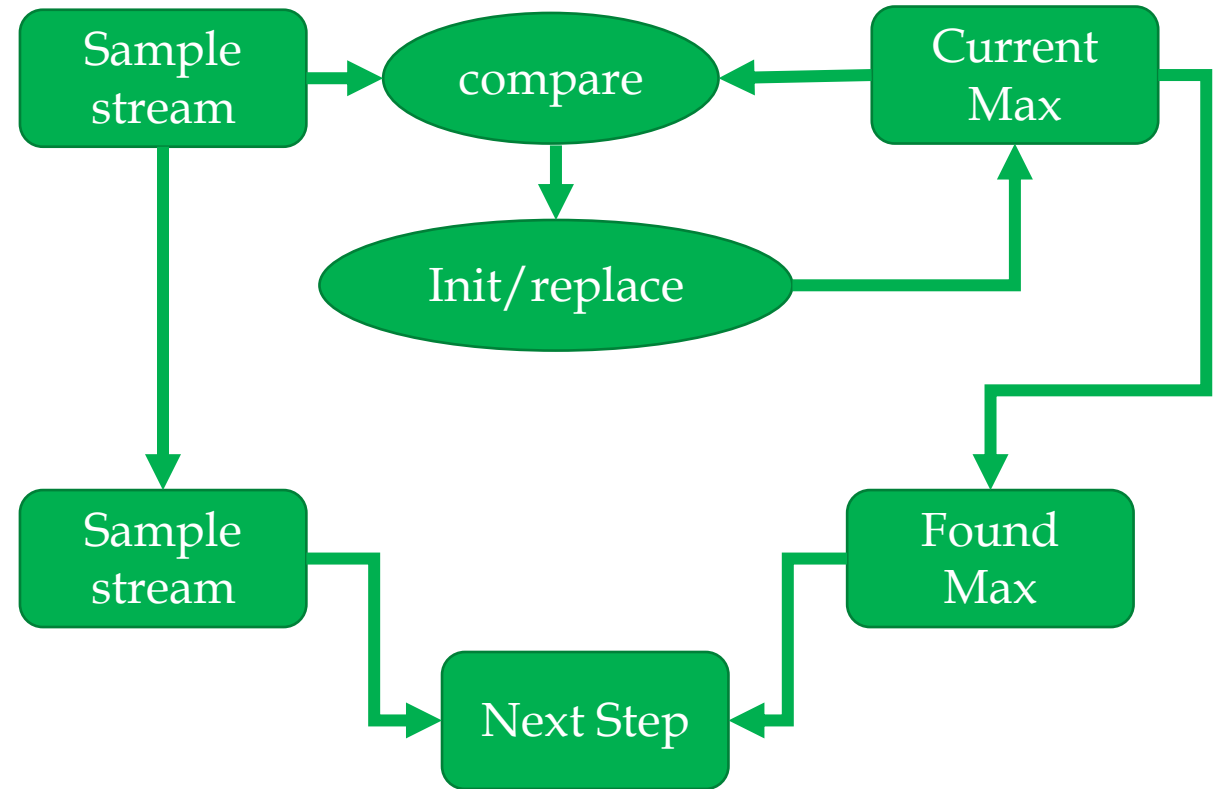
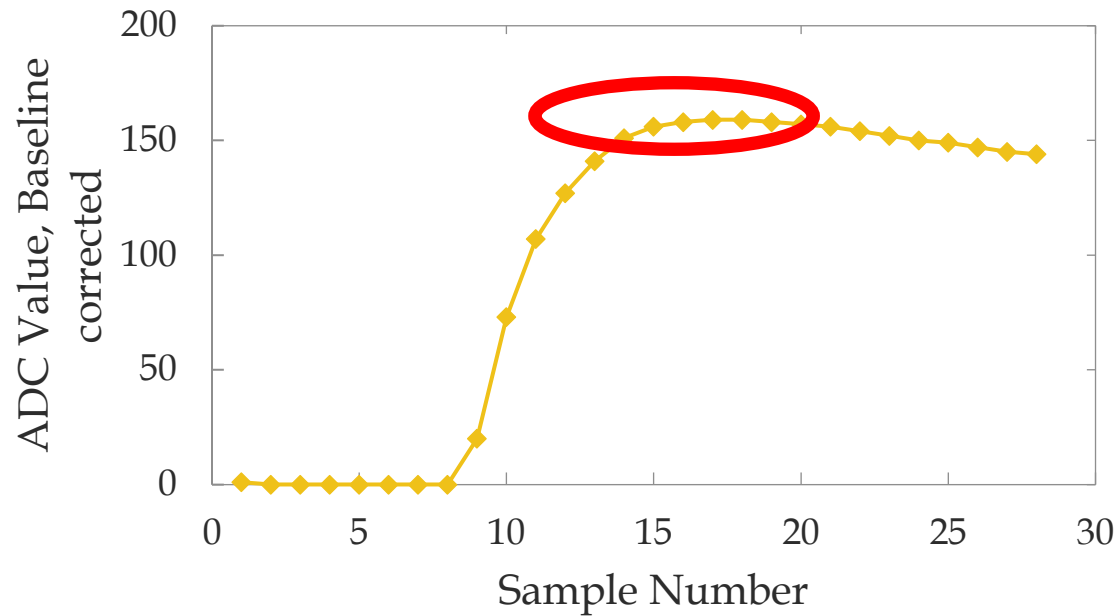
# Preprocessing

- Step 1: Remove baseline
  - Mean 4 first samples : 4 clocks
  - Subtract with saturation : 1 clock per sample



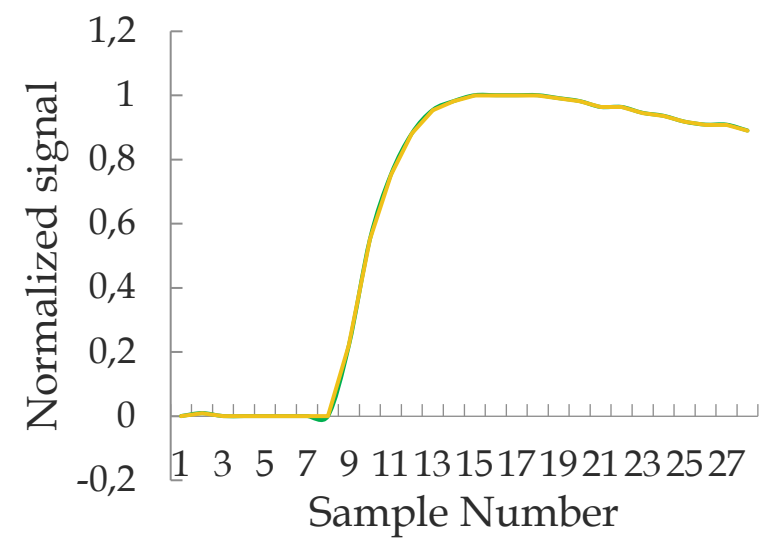
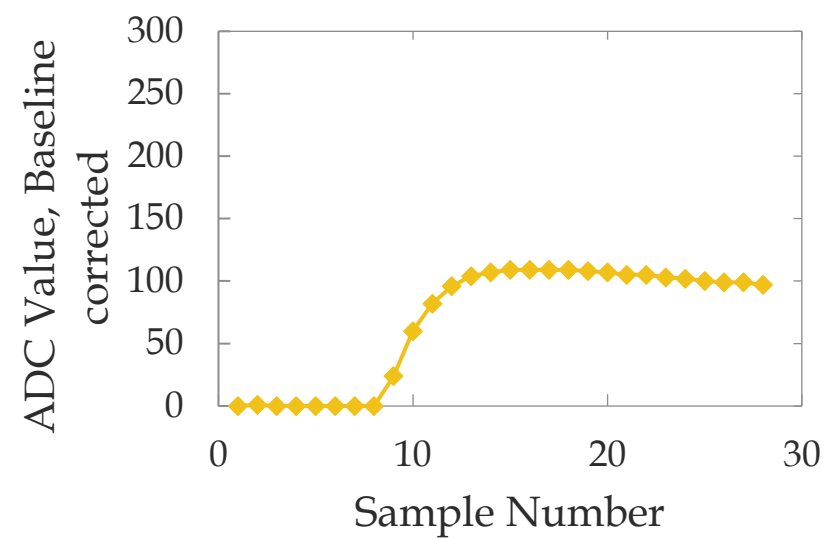
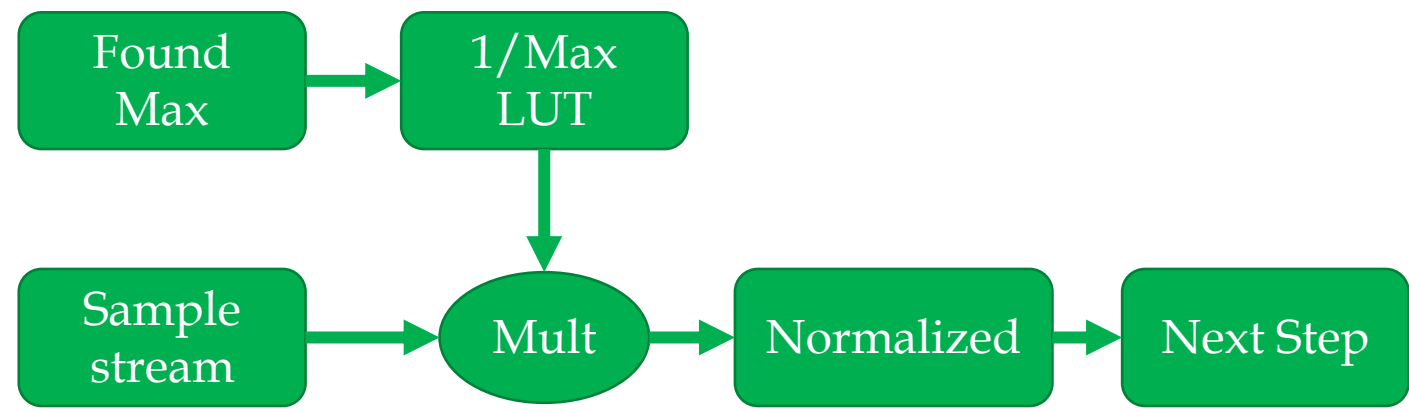


- Step 2: find maximum value in signal to determine normalization factor
  - Travel samples until first local max is encountered

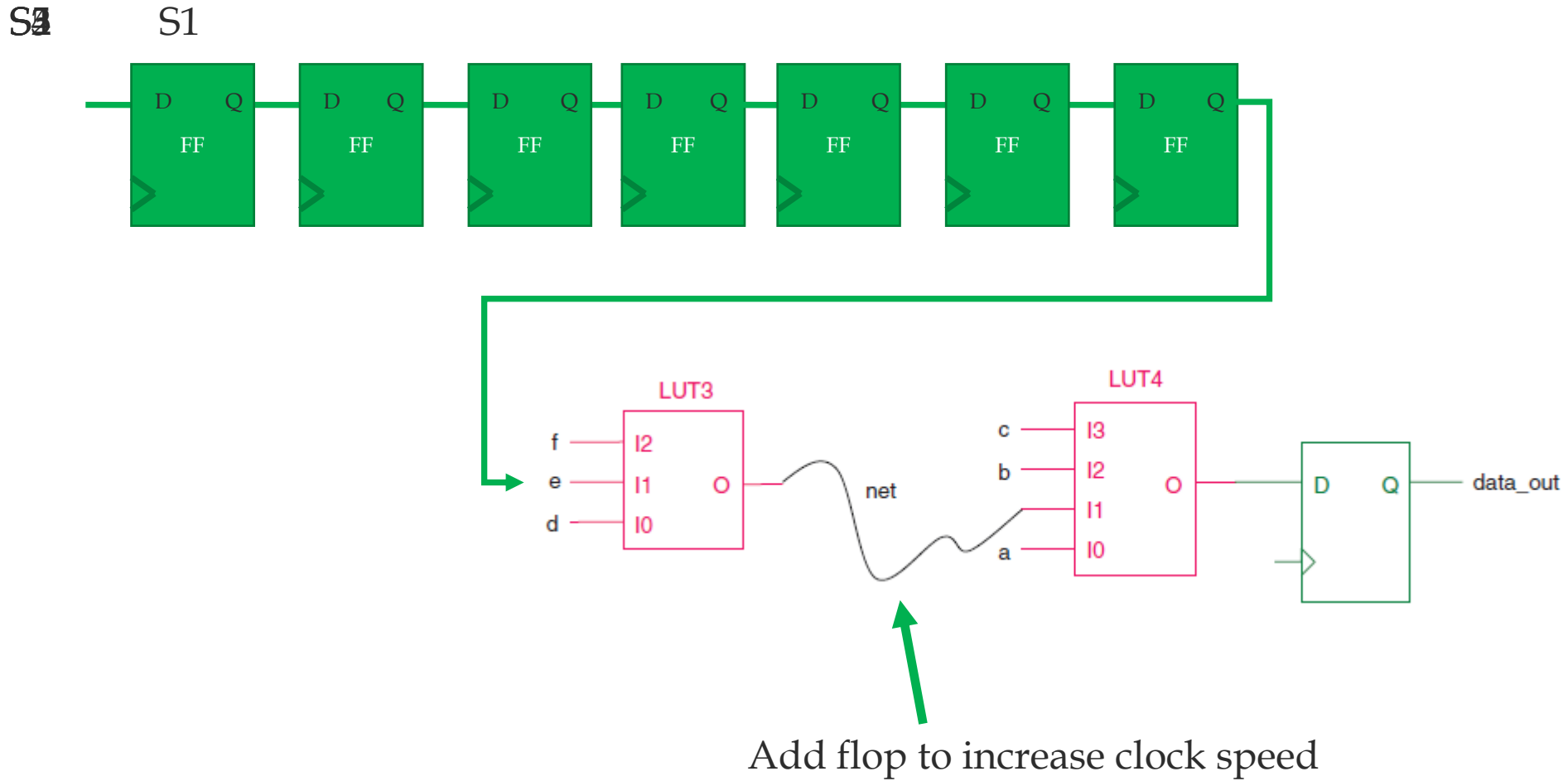


# Normalization

- Samples \* 1 / Max
  - Hardware multiplier, fixed point
  - Look-up table for 1 / Max, 8-bit resolution
- 1 clock per sample



- Logic performance depends on the longest logic/routing path
- Final result depends on
  - Worst case logic levels
  - Fan-out
- Set a goal for the tool
  - It will try changes for you, and maybe reach your goal
- If it cannot reach the goal, look for a report about “worst negative slack”
  - Net with longest delay
  - “Total negative slack” is the sum of all paths that fail the goal
- Look at the failing paths, review your code, rearrange, etc.



# Modern design approaches

- Question: « This looks very challenging to make, and worst, if I want to modify it! »
  - Answer: « You are right... »

- High Level Synthesis (HLS)

- AMD (Xilinx)
- Intel (Altera)
- Cadence/Synopsys/Siemens(Mentor)
  - → ASIC

- Almost like C/C++

- Have to include compiler directives for hardware « shaping » like pipelines

```
38  template <typename samples_t, typename meantype_t>
39  samples_t BaseLineMean(samples_t Samples[SAMPLE_COUNT],
40                          samples_t ReturnSamples[SAMPLE_COUNT])
41  {
42      meantype_t Mean = 0;
43      for(int x= 0 ; x < SAMPLE_COUNT; x++)
44      {
45          if(x < 4){
46              Mean += Samples[x];
47          }
48          ReturnSamples[x] = Samples[x];
49      }
50      Mean >>= 2;
51      return (samples_t)Mean;
52  }
```

- Included in
  - Vivado ML Standard Edition (previously « web pack »)
  - Quartus
- Many vendor examples and tutorials
- Case studies presented at the 2018 IEEE NPSS Real Time Conference
  
- Start with vendor examples, then with small design of your own
  - Always keep in mind how you expect the hardware might look when writing your code



# Thank you for listening!

Questions? 😊