# Système d’acquisition de données sur RedPitaya avec module de traitement High Level Synthesis (HLS)

*Michel Gros, Hervé Le Provost,*

*Journées des Métiers de l'Electronique de l'IN2P3 et de l'IRFU, juin 2023*

# History - Motivation

Boites cali

4 ADC channels– 16-bits - optimal sampling @5 MHz

Mother board CEA + Mini-Module AVNET Virtex-5 FX

Data acquisition over Ethernet - software client/server

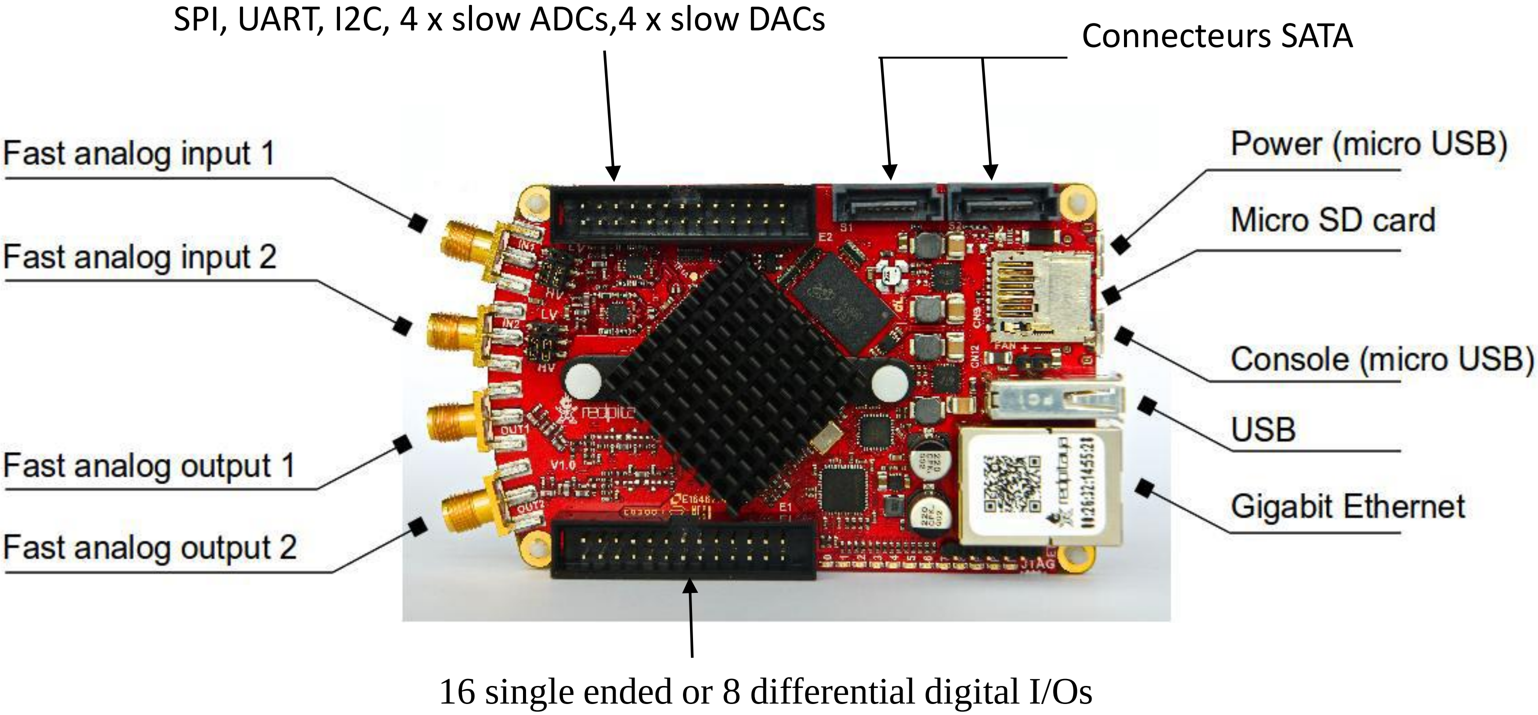
Samba CEA –no embedded OS –

mini-modules AVNET Virtex-5 FX obsolete

Replaced by mini-modules AVNET Zynq with firmware/software update – Acquisition validated but DDR bug on mini-module – solution abandoned

=> Choice to evaluate the RedPitaya ecosystem – same type as Raspberry pi but with programing logic

# RedPitaya



SPI, UART, I2C, 4 x slow ADCs,4 x slow DACs

Connecteurs SATA

Fast analog input 1

Fast analog input 2

Fast analog output 1

Fast analog output 2

Power (micro USB)

Micro SD card

Console (micro USB)

USB

Gigabit Ethernet

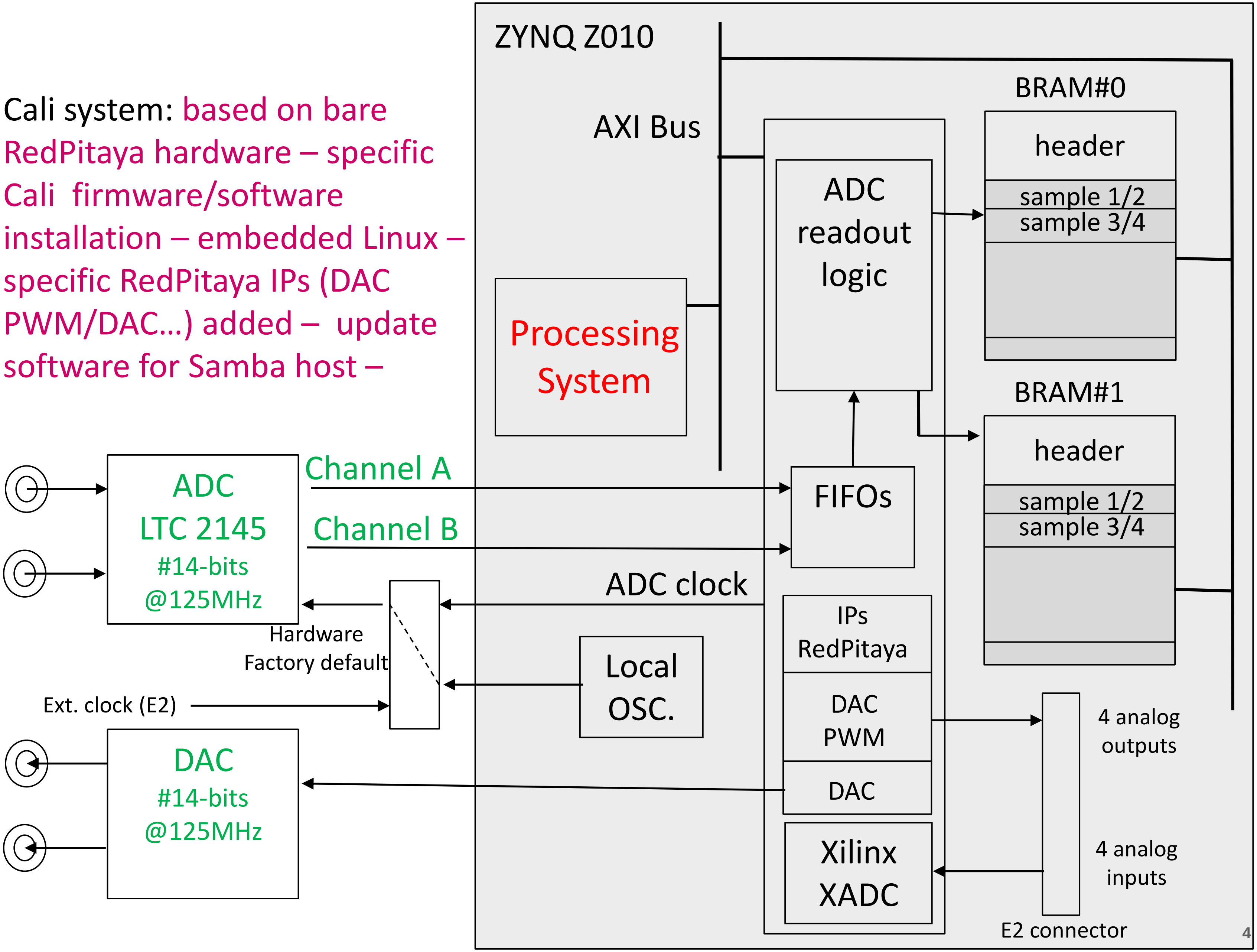
16 single ended or 8 differential digital I/Os

Documented in details here: https://redpitaya.readthedocs.io/en/latest/
Git for developers here: https://github.com/RedPitaya/RedPitaya

Full support for ecosystem: embedded linux– client/server RedPitaya protocol – Applications: oscilloscope, signal generator, spectrum analyzer…(RedPitaya) Power Analyzer…(community) oriented radio - a lot of accessories – claim to be open-source platform

# RedPitaya Cali - principle

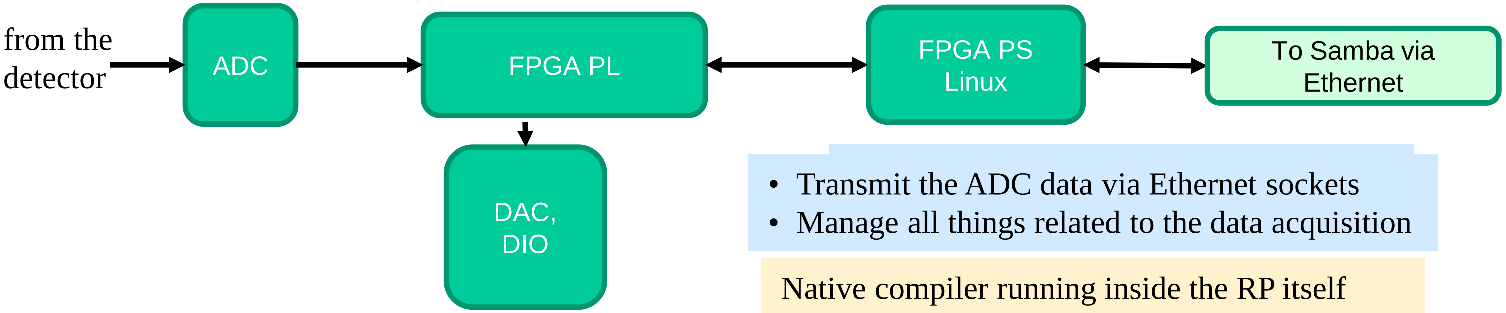
Cali system: based on bare RedPitaya hardware – specific Cali  firmware/software installation – embedded Linux – specific RedPitaya IPs (DAC PWM/DAC…) added –  update software for Samba host –



# RedPitaya Cali – developments 1

Embedded linux: « Réalisation du système pour RedPitaya sur SD-card » G. Goavec-Mérou, J.-M Friedt – based on Buildroot – no difficulty

Native compiler for ARM: to develop RedPitaya application on MAC (no cross-compiler for ARM) or to easily compile on target– Native compiler for RedPitaya Cali via crossTool-ng – depends on embedded linux – Hard /long to generate – but provided with SD RedPitaya Cali card - https://github.com/crosstool-ng/crosstool-ng



- Transmit the ADC data via Ethernet sockets
- Manage all things related to the data acquisition

Native compiler running inside the RP itself

Language C/C++

- Read the ADC, create ethernet payload
- Output some signals (-> synchronisation)

Language VHDL/SystemVerilog

- Event detection
- Create Ethernet payload

Presentation Part 2

Language C/C++ (HLS) with pragmas for synthesis

# RedPitaya Cali – developments 2

- To **increase the number of channels** to read, use several RedPitaya working in parallel

- Some **FIFOs** in the process
  - samples from different RP arrive at different time in Samba.
  - we need to be sure that a given sample (i.e. with a given timestamp) from a given RP corresponds to the samples of the other RP which have the same timestamp

- Also, the clocks of all RP from the 125MHz local oscillator have **not exactly** the same frequency…

- Then we have to synchronize several RedPitaya, **both for the clock and for the timestamp**.

- 1 RP will be "*master*", and the others will be "*slaves*" (to be declared in the setup of Samba)

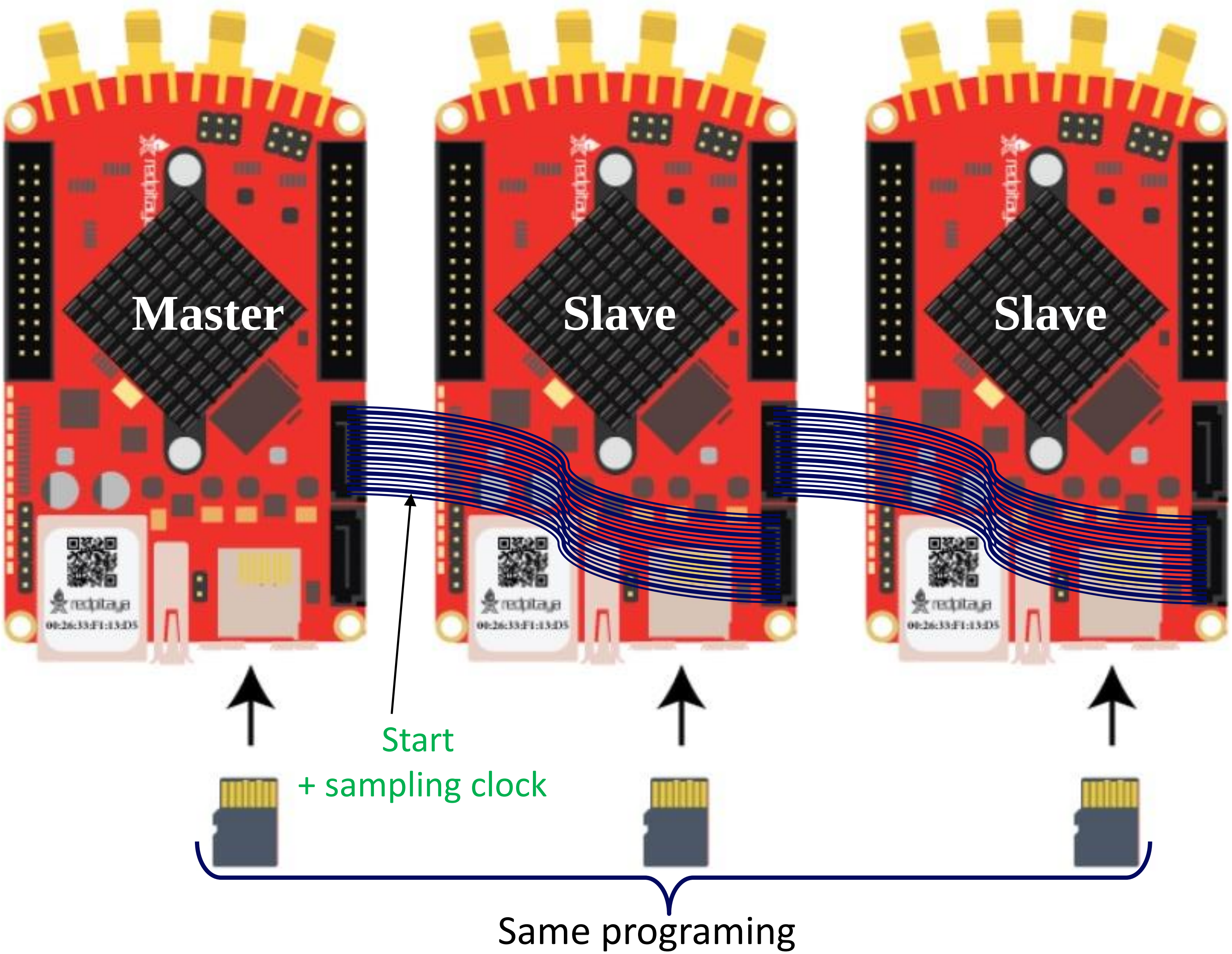
- Timestamp synchronization
  - when Samba sends a START, the master sends a *"timestamp reset"* to the slaves, and resets simultaneously its own timestamp;
  - the physical link is made by a common SATA cable

- Clock synchronization
  - 2 resistances to move to configure a RP as slave

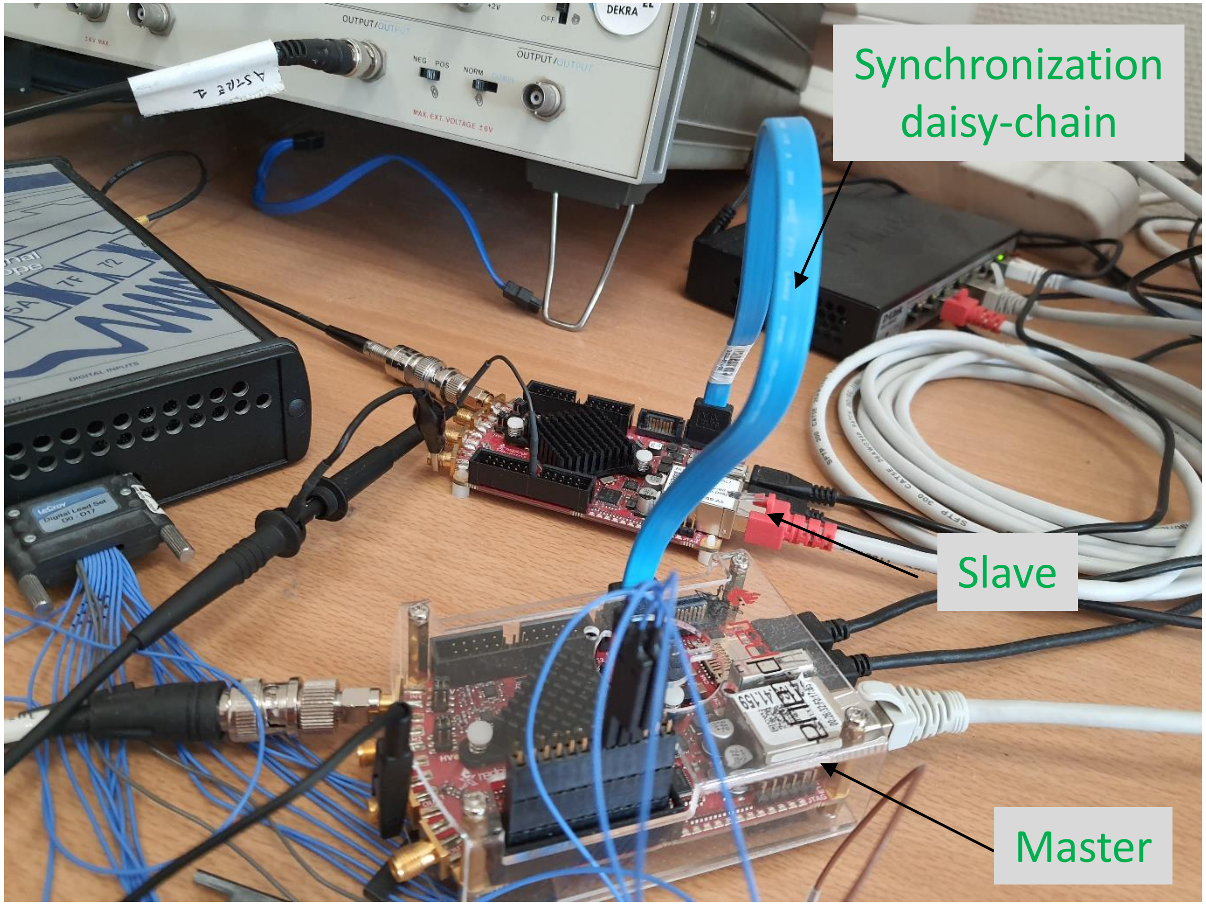
  Default out of box is master configuration

- Limitation on the total sampling rate: have to lower the clock (or, computers can also run in parallel)

# Many-channels DAQ: SATA daisy chain



# RedPitaya Cali – setup example



Synchronization daisy-chain

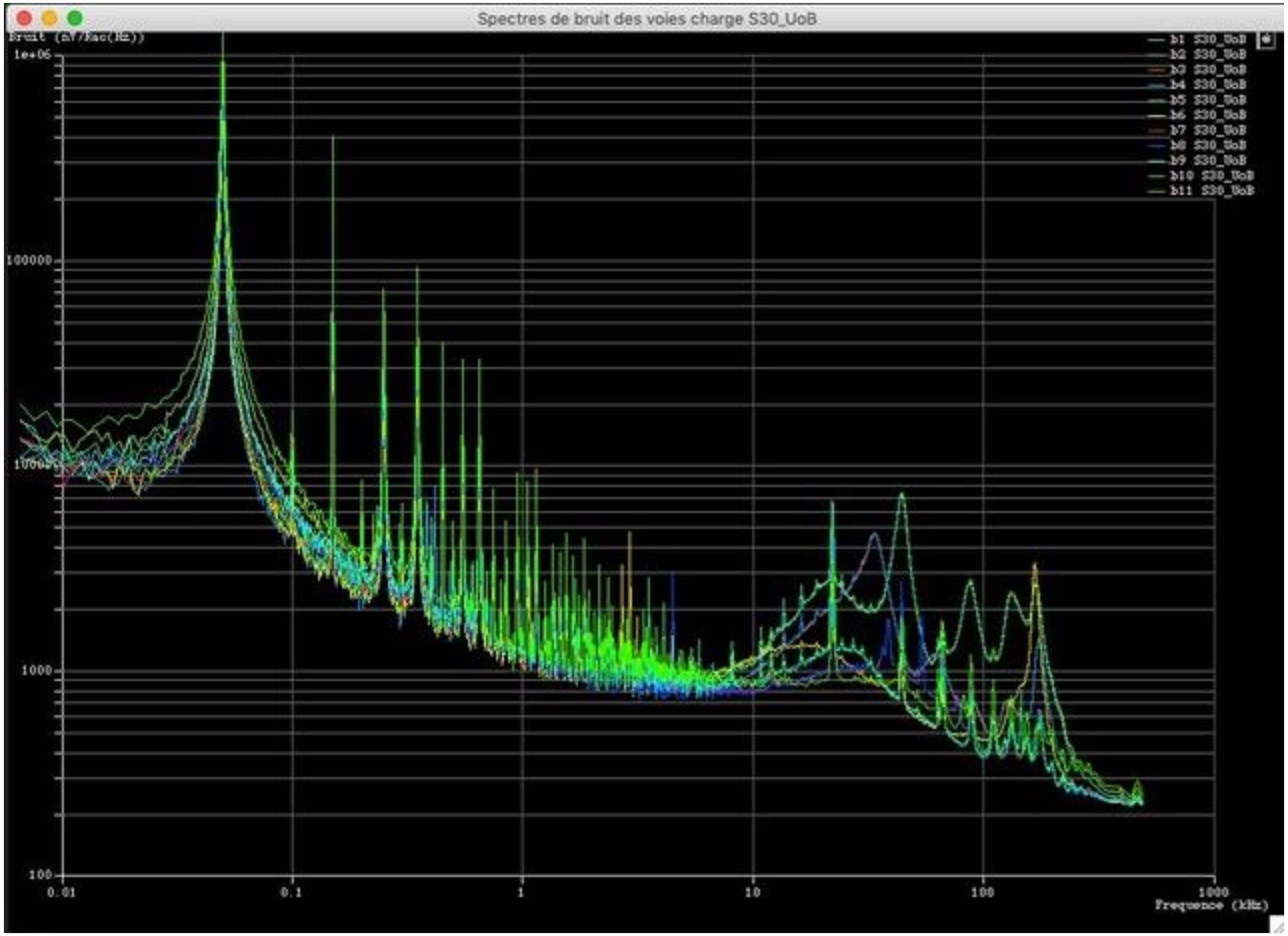
Slave

Master

# Many-channels DAQ: SATA daisy chain

NEWS-G  New Experiment for Wimp Search, with Gas

First installation with 11 channels / 6 RedPitaya June 2023 @University of Birmingham  (Michel) - Validated



Frequency spectrum : 11 channels – RedPitaya Cali/Samba DAQ

# RedPitaya Cali - Summary

No hardware to develop: a DAQ system with high speed DAC is up and running with off-the shelf hardware, a flashed SD card and a host configured with Samba (MAC) or Samix (Linux PC)
System delivered as a package - Samba/Samix is configured after installation for 1 board/2 channels

Active RedPitaya Eco-System

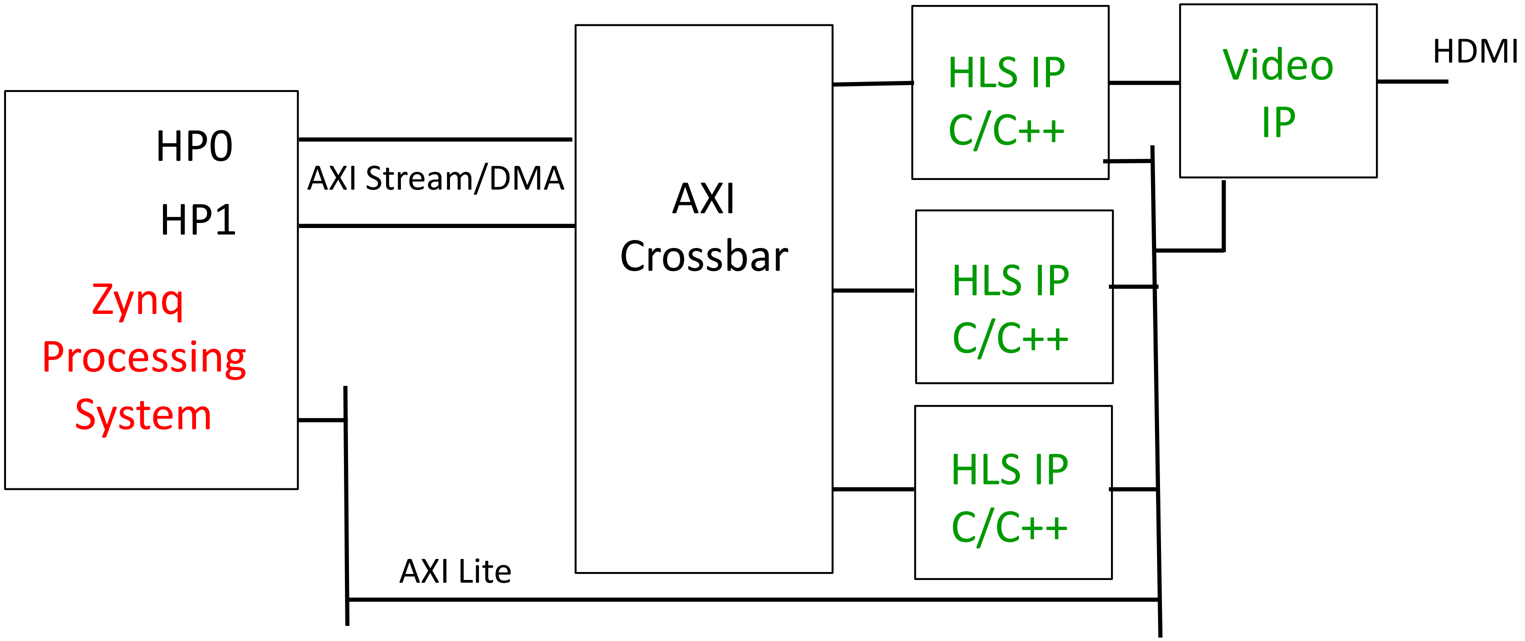
STEMlab 125-14 Low Noise Starter Kit
STEMlab 125-14 External Clock Starter Kit
STEMlab 125-14 X-Channel System (Scalable multi-channels )
SIGNALlab 250-12
SDRlab 122-16… (16-bits)

Scalable DAQ

Data throughput: about 400 Mbps (TCP/IP or UDP) limited by Zynq PS (no jumbo frames)

Next step is to add a trigger: Host running Samba may be not powerful enough to filter data for high data rate/multiple channels - not enough power with the ARM processor@666 MHz – Study to implement it in the PL with HLS C/C++ synthetizer

# High Level Synthesis (HLS) typical design



HLS typical design: Co-processing for Zynq PS – input/output stream processing and IP configuration via AXI lite

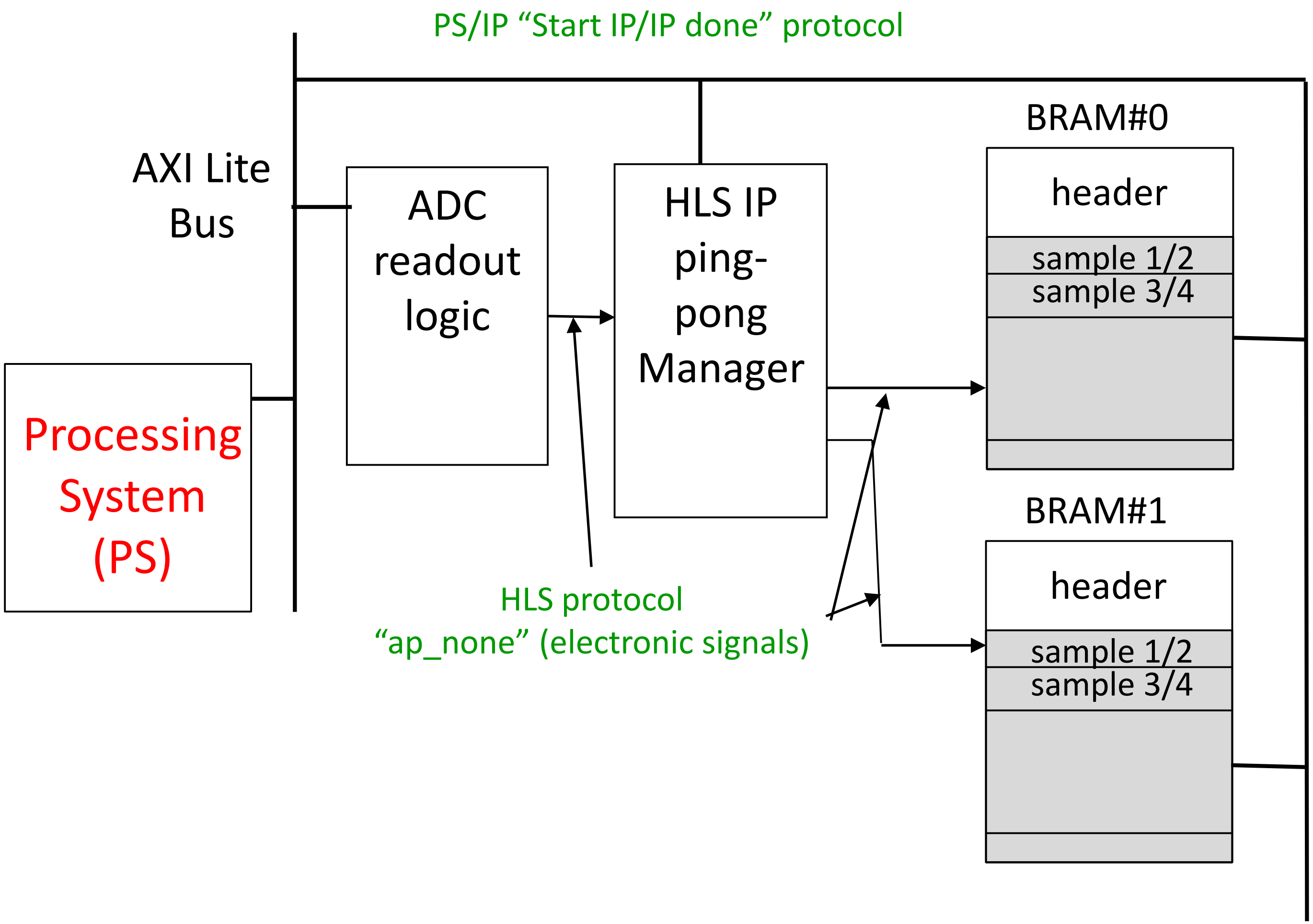
A lot of support/example designs from Xilinx and developers.

Digilent zybo-z7 (oriented video processing) - OpenCV Video Libraries in Vivado HLS. https://digilent.com/reference/programmable-logic/zybo-z7/start

Students Master Paris-Saclay/Ensta: efficient to develop with HLS complex algorithms even without electronic background – HLS reports for pipeline/latency are detailed – experience in HDL programing may be a bonus

But what is the step to replace VHDL/Verilog/SysVerilog by HLS ?

# HLS Cali Redpitaya – First step 1



How to write a Finite State Machine (FSM) as in VHDL/Verilog?

# HLS Cali Redpitaya – First step 2

```
void bramRouting (uint4 *Web_in, uint32 *Ain, uint32 *Din, uint32 *localErr, uint32 *Dout…) {

#pragma HLS INTERFACE ap_none  port= Web_in
#pragma HLS INTERFACE ap_none  port= Ain
#pragma HLS INTERFACE ap_none  port= Din
#pragma HLS INTERFACE ap_none  port= Dout
…
#pragma HLS INTERFACE s_axilite port=localErr bundle=PARMS
#pragma HLS INTERFACE s_axilite port=return bundle=CONTROL_BUS
#pragma HLS INTERFACE ap_ctrl_hs port=return
```

Electronic signals bus size defined by type

Parameter passed by processor via AXI memory access

Start/done IP via AXI memory access

Identify structure to add constraints via pragmas in HLS

Initiation Interval – While loop should be evaluated each clock cycle – depending on design, no clock cycle can be lost or it may be broken

```
        mainLoop: while (1) {
        #pragma HLS PIPELINE II=1
                        wrBuffers( Web_in, Ain, Din, localErr, Dout…);
        }//infinite mainLoop
}
```

IP Done=1 If IP is configured in autoStart mode via AXI at initialization, it will restart automatically but with one clock delay

HLS PIPELINE II=1 is a constraint for the synthesizer – it does not mean, it will be satisfied but if violated, it will be reported with explanations

Output of the synthesis is a Vhdl/Verilog code – generated code format can be parametrized but it is hard to debug at this level and not foreseen by the XILINX design flow

# HLS Cali Redpitaya – First step 3

```
void wrBuffers (uint4 *Web_in, uint32 *Ain, uint32 *Din, uint32 *localErr, uint32 *Dout…) {

static STATE_BRAM_CIBLE stateBram = STATE_BRAM_0;
```

Keep the value between two calls – initialized to STATE_BRAM_0

```
uint4 W;
uint32 A; uint32 D, D0 , D1;
uint32 val_in; uint16 v;
```

4-bits bus

Tested and working with Cali RedPitaya/Samba acquisition

```
//inputs
A = *Ain; W = *Web_in; D = *Din; D0 = *Din_0; D1 = *Din_1;
```

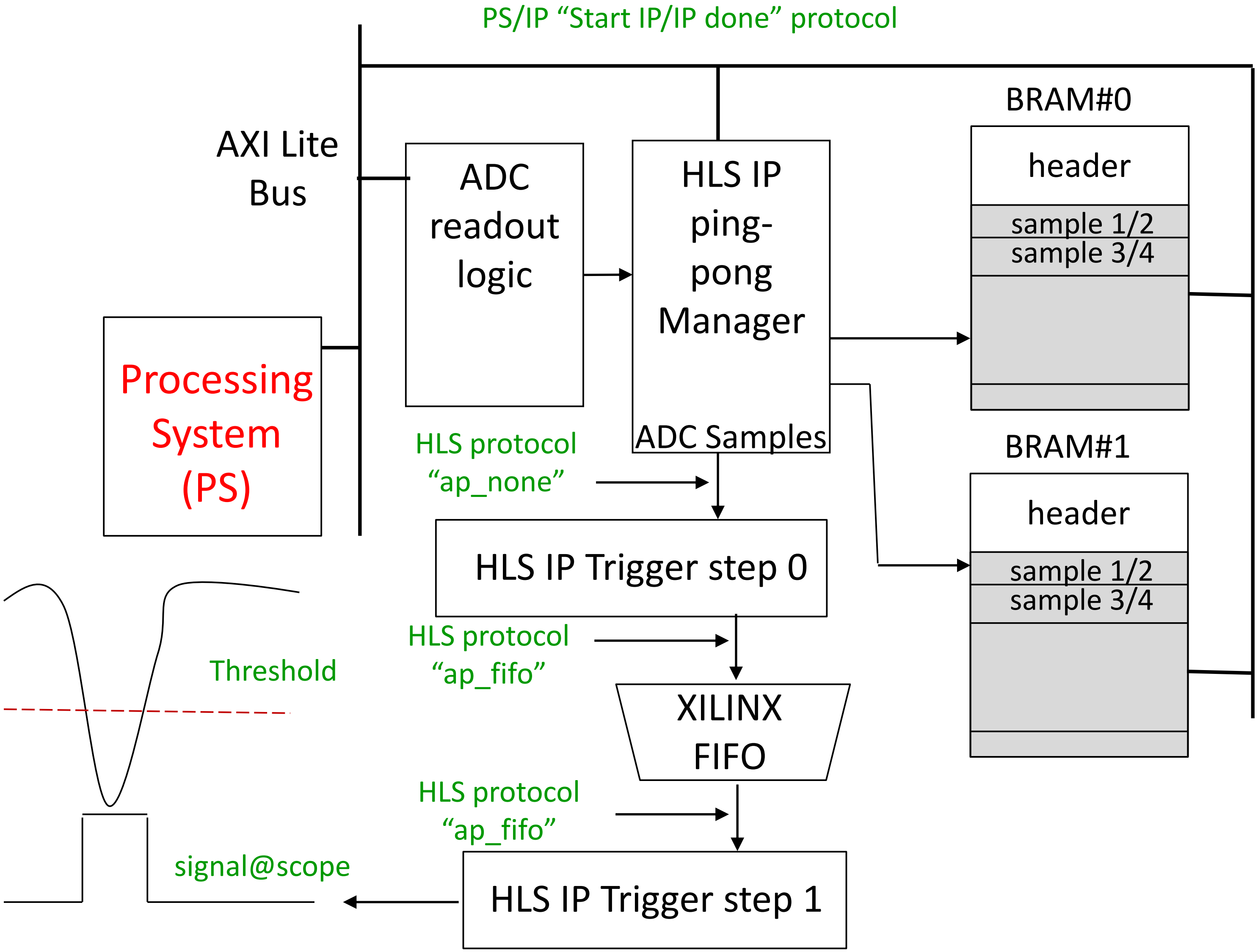
Electronic signals

```
//FSM to control the BRAM_0 & BRAM_1 data routing
Switch (stateBram) {
        case STATE_BRAM_0:
                *Aout_0 = A; *Web_out_0 = W; *Dout_0 = D;
                //Once Cali wrote a packet, it should update the BRAM status to full @0
                if ((A == 0x00000000) && (W != 0)) {
                        if ( D != BRAM_FULL) {
                                *localErr = ERR_WR_BRAM_0_FULL;
                        }
                        *Dout = BRAM_FULL; //block Cali in the mean time
                        stateBram = STATE_BRAM_READ_STATUS_PIPE_0;
                } else {
                        *Dout = BRAM_FREE;
                        stateBram = STATE_BRAM_0;
                }
                break;
        case STATE_BRAM_1:
                *Aout_1 = A; *Web_out_1 = W; *Dout_1 = D;
```

Can be access from PS via memory access

Clock is not explicitily defined but if HLS PIPELINE II=1 is satisfied, it is same as process (clk) and case/when VHDL FSM

# HLS Cali Redpitaya – Second step 1



# HLS Cali Redpitaya – Second step 2

```
void adcTreatLvl2 (volatile uint32 *threshold, volatile uint32 *calculated, volatile uint32 *fifoIn, volatile
booleen *trigOut) {
```

Trigger output - oné wire

Fifo input interface (ef, data, read signals)

```
#pragma HLS INTERFACE ap_fifo  port= fifoIn
#pragma HLS INTERFACE ap_none  port= trigOut
#pragma HLS INTERFACE s_axilite port= threshold bundle=PARMS
#pragma HLS INTERFACE s_axilite port= calculated bundle=PARMS
#pragma HLS INTERFACE s_axilite port=return bundle=CONTROL_BUS
#pragma HLS INTERFACE ap_ctrl_hs port=return

                loopInfinite: while (1) {
#pragma HLS PIPELINE II=2
                                findSig(fifoIn, threshold, sample, calculated, trigOut);
                }//loopInfinite
}//adcTreatLvl2
```

At least 1 fifo read every 2 clock cycles requested

# HLS Cali Redpitaya – Second step 3

```
void findSig(volatile uint32 *fifoIn, volatile uint32 *threshold, volatile uint32 *calculated, volatile booleen *trigOut) {
        …
        float average,norme=1.0/(float)DIM_BUFFER;
```

Static: Keep the value from one call to the next one

Accumulate sample

```
        static int somme=0;
        static uint32 intAverage;
        static booleen toggle,plein=0;
        static short sampleTab [DIM_BUFFER];
        static int l=0,p=0;
```

Extract 2 samples (32-bits) from Fifo

```
        fifoData = *fifoIn;
        if ((fifoData & 0xffff) == 0)
                val = (short)((int)0x8000 - (int)((fifoData>>16) & 0xffff));
        else val= (short)((int)0x8000 - (int)(fifoData & 0xffff));
        sampleTab[l++] = val;
…
```

Average over last DIM_BUFFER sample

```
        if(plein) {
                if(l == 0) somme = somme - sampleTab[0] + sampleTab[DIM_BUFFER-1];
                else somme = somme - sampleTab[l] + sampleTab[l-1];
        } else  somme += sampleTab[l-1];
        average = somme * norme;
…
```

Threshold

Trigger output Tested @scope and working

```
        intAverage = (uint32)average;
        if ((intAverage > *threshold) && (intAverage!= 0x8000))
                *trigOut = 1;
        else
                *trigOut = 0;
}//findSig
```

# HLS Cali Redpitaya – Summary

Replace VHDL/Verilog/SysVerilog: after a learning curve, it is realistic even if you feel at start it would be much faster to code in HDL – in HDL, you code one implemented solution, in HLS you code an algorithm and explore with pragmas different implementations

Open FPGA programming to physicist and software developers: straightforward for AXI stream pipelined data processing – Ease exchange for instance to establish a model of a FPGA code in a software system

Next step for RedPitaya Cali: add some additional filtering to the threshold trigger – extract the ADC samples and build the Ethernet packet payload