JOINT INSTITUTE FOR NUCLEAR RESEARCH
Veksler and Baldin laboratory of High Energy Physics

FINAL REPORT ON THE SUMMER STUDENT PROGRAM

Research model of the SAMPA application specific integrated circuit for the MPD TPC-detector

Supervisor:
Sergey Alexeevich Zaporozhets

Student:
Yauheni Zenkovich, Belarus
Belarusian State University
of Informatics and
Radioelectronics

Participation period:
June 13 – August 6

Dubna, 2016

Contents

Introduction 3
Analog part 4
Auxiliary tools 4
Architecture and structure scheme 6
Changes of the SAMPA emulator 7
Waveforms 11
Operation of the ZSU module 15
Conclusions 16
References 17

Introduction

The report’s main objective is researching the design of a mixed signal application-specific integrated circuit(ASIC) for data readout, called SAMPA [1], which will be possibly implement on detector time projection chamber (TPC) at NICA project. The international mega-science project NICA (Nuclotron-based
Ion Collider fAcility) complex is aimed in the study in the laboratory of the properties of nuclear matter in the region of the maximum baryonic density. [2]

The ASIC SAMPA will be responsible for the amplification, sampling, digital filtering and formatting of the signals incoming from the detection chamber. Basically, this report is focused on the digital part and contains only general information on the analog part. Also, this report presents the research Verilog hardware description for the physics implementation of the SAMPA digital part.

SAMPA is improved version of the chip used now-a-days, called ALTRO. Among the list of upgrades, the most notable on the digital part for the MPD TPC-detector are: smaller chip’s size and serialization of the output, in opposite to the 40-bit bus on ALTRO / s-ALTRO. Also, the important advantage is configured the continuous readout option, with keeping the old triggered mode. The SAMPA ASIC adapts to different detector signals with programmable parameters. Amongst the other most notable upgrades full 32 channels integration, two times the number of the previous chips, lower power dissipation of 15mW / channel. Besides, SAMPA implements continuous data readout, with automatic data acquisition triggering and data sending. The chip outputs data through 4 serial links synchronous to a maximum clock of 320 MHz, while the digital signal processor (DSP) works nominally on 10 MHz and is the main ASIC’s clock (bigger clock domain). SAMPA has 3 clock inputs: 320, 40 and 10 MHz. However only the 320 MHz is necessary for operation, as the others may be derived from it by properly configuration of the clock generation block. This block also always generates a 32 MHz clock from the 320 MHz. The configuration is performed via instructions, which are send through an I^2C interface.

The whole architecture is based on 10-bit values and the DSP block has some data paths wider than 10-bit to achieve higher arithmetic precision. Obviously, the price to pay is a higher area and power consumption by the filters. Digital signal processing, zero suppression, data formatting, buffering (before sending the data out) and instruction processing is performed synchronous to the 10 MHz clock signal. The output block, responsible for sending information serially, is the only one working on 320 MHz and the interface between this block and the previous one, the data buffer (Ring Buffer) is performed on 32 MHz to match different clock speeds and memory sizes. Finally, a 20-bit 40 MHz global counter is implemented to provide a precise time information about the initial time of an event. [3]

Analog part

Right after colliding, the particle beam generates millions of other particles which create a voltage signal on the detection pads. These pads have a capacitance which accumulates charge for the next block preamplifier. This block, Preamplifier is an amplifier which is sensible to the charge on the capacitance and is responsible for amplifying the signal induced on a pad. Next, there's the Shaper, responsible for transforming the incoming pulse in a signal with a semi-gaussian shape. The idea behind the semi-gaussian shape is extending the time duration of the signal, lowering the sampling rate needed on the ADC to 10 MSPS. If the Shaper was not implemented, the sampling rate would have to be high enough to sample a very fast pulse, maybe reaching up to 1 GSPS , which would generate much more data and power dissipation issues. Finally, the ADC generates 10 bits data on 10MHz (default ADC clock) and provides it to SAMPA's digital part, which is where this work is focused.

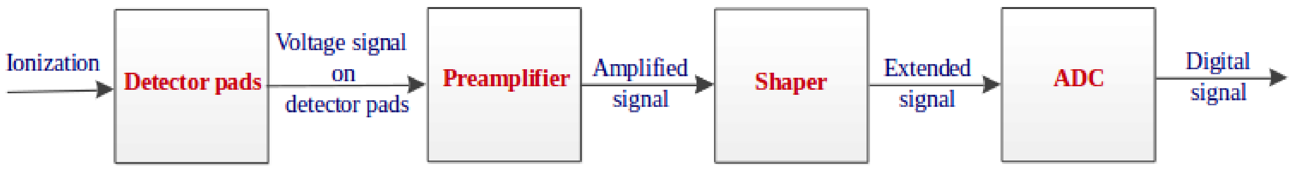

Figure 1 Structure scheme of the analog part

Auxiliary tools

Hardware design platform for research the ASIC SAMPA will use the SoCKit development board built around the Altera System-on-Chip (SoC) FPGA. The board has many features that allows users to implement a wide range of designed circuits. FPGA Device includes [4]:

- Cyclone V SX SoC—5CSXFC6D6F31C6N
- 110K LEs, 41509 ALMs
- 5140 M10K memory blocks
- 6 FPGA PLLs and 3 HPS PLLs
- 2 Hard Memory Controllers
- 3.125G Transceivers

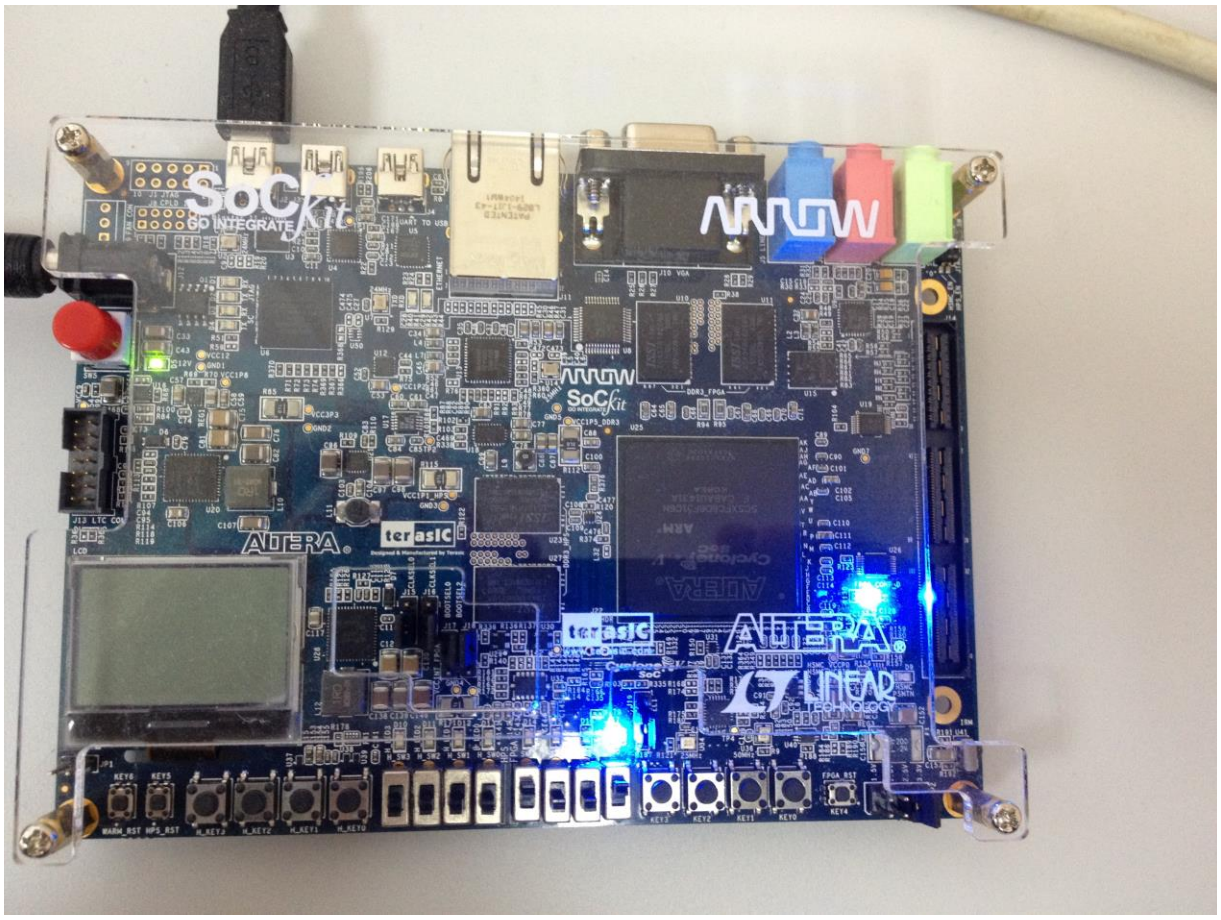

Figure 2 A photograph of the SoCKit development board for research SAMPA

Software will use Quartus II tools for use in debugging of hardware design. The Quartus II software includes many tools that are useful for a variety of purposes. In this report I will use two types of tools: *Netlist Viewers* and *SignalTap II Logic Analyzer*.

The Netlist Viewers provide a graphical indication of a synthesized circuit. A register transfer level (RTL) view of a designed circuit, generated after the initial synthesis, can be seen by using the RTL Viewer . A view of the final implementation, obtained after technology mapping , is available through the Technology Map Viewer . If a designed circuit involves a finite state machine, a diagram of this FSM can be examined by means of the State Machine Viewer[5].

The RTL Viewer provides a block diagram view of a circuit, at the level of registers, flip-flops and functional blocks that constitute the design. The displayed image is the circuit generated after the analysis and initial synthesis steps. It is not necessary to wait for the rest of the compilation process to be completed, which includes placing and routing the designed circuit.

The original circuit of SAMPA (version SAMPA_emulator.r698) is given in Figure 2.

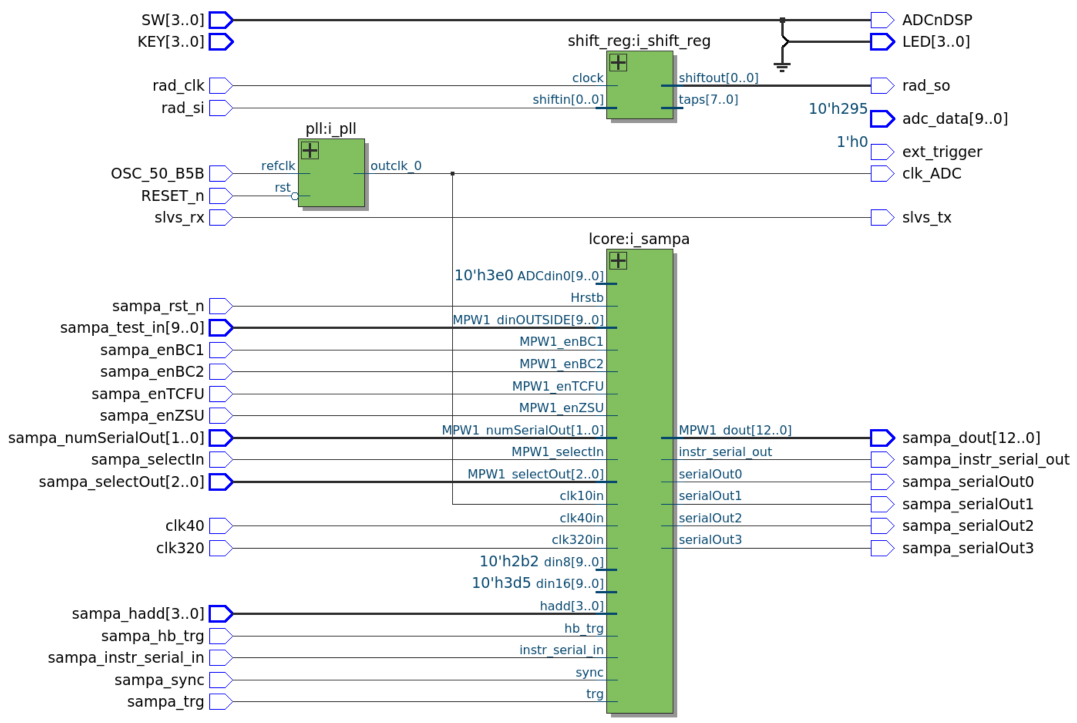

Figure 3 Original circuit of SAMPA (window RTL Viewer)

Quartus II software includes a software-implemented tool that acts as a virtual logic analyzer, which allows the user to examine signals that are going to occur anywhere within a circuit implemented in an FPGA chip. It is called the SignalTap II Logic Analyzer.

Architecture and structure scheme

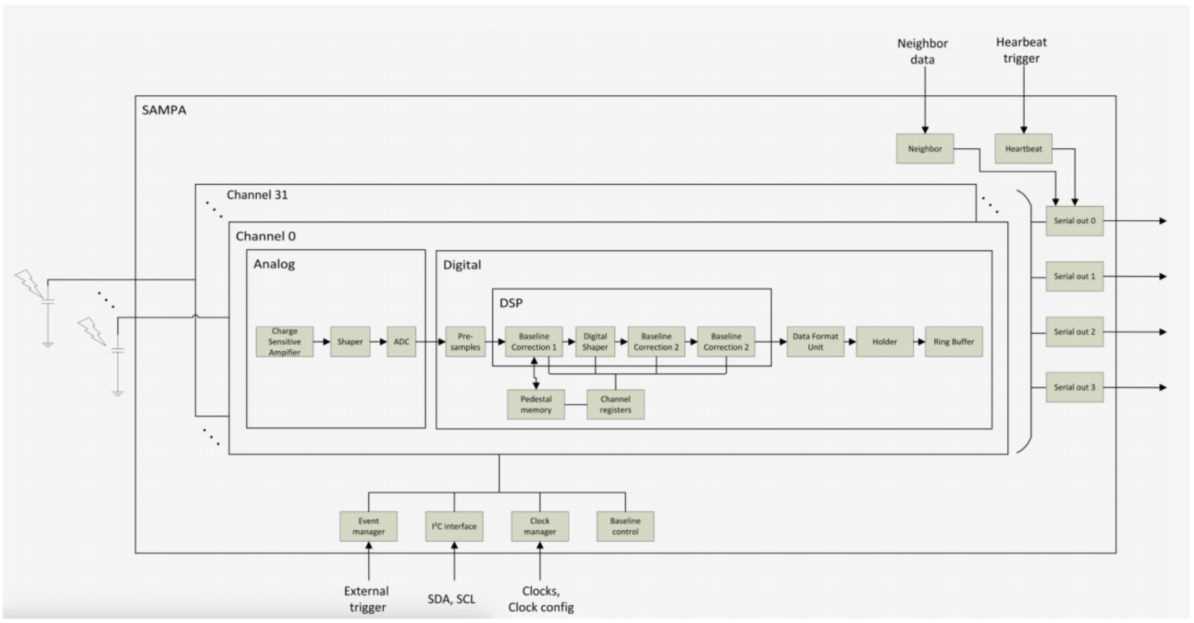

Figure 4 Architecture SAMPA

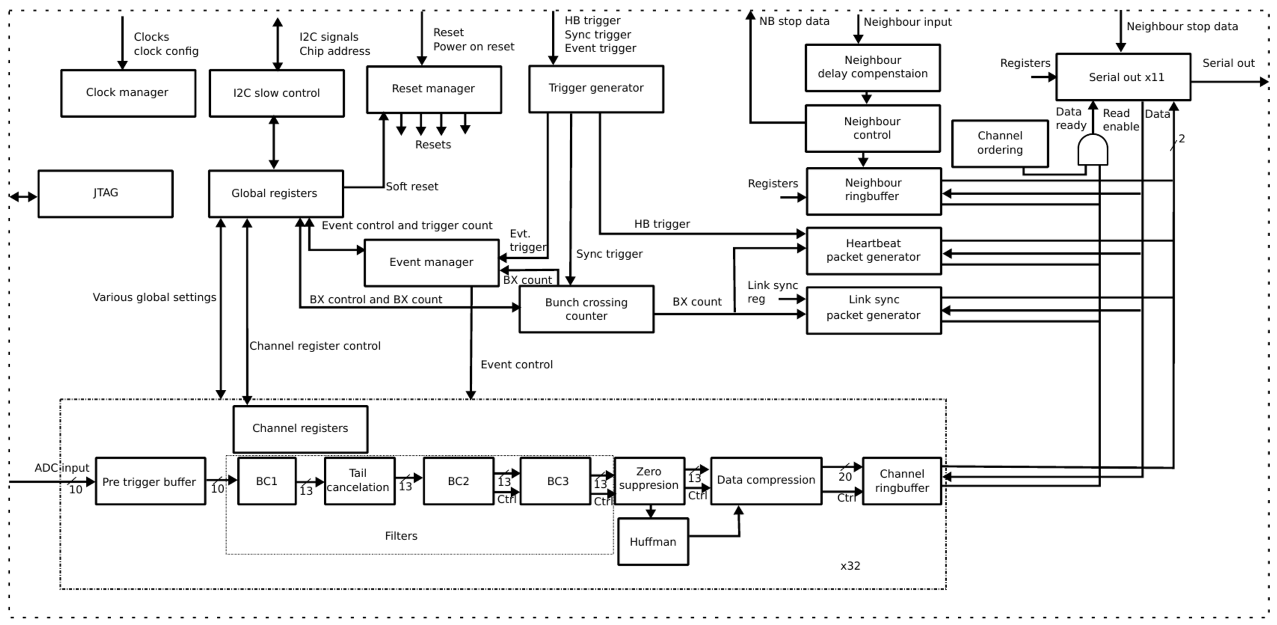

Figure 5 Structure scheme of the digital part SAMPA

Changes of the SAMPA emulator

The original version firmware which is an emulator SAMPA's chip has some defects or just unfinished parts.

1) Module oneshot was inserted to avoid the bounce of the buttons . Nothing was done with the bounce of the switches (there was no need) .

2) Assignments for some of the most important inputs to the buttons and the switches:

Hrstb   = key_os[0];   // Hard reset of all the system (asynchronous)

hb_trg = key_os[1];   // Being installed provides HeartBeat signal from out SerialOut0

sync = key_os[2];   // Being installed resets Bcrosscount

trg   = key_os[3];   // External trigger for Trigger mode (may be not used due to Nios (embedded processor), which may generate this signal)

SW[0] // Was used for temporal assignments;

instr_serial_in = SW[1]; // The value of the constant 1 generates constant SoftReset, which obstructs the analysis of the work of the system. This input was set to 0 by a switch. However it is important to improve understanding of of this input operation, as it is used to enter the commands in SAMPA.

selectIn   = SW[2]; // Selects: to send a constant or a signal to ZSU input

enZSu   = SW[3]; // Selects: to set zero suppression threshold to 0 or to 1.

3) Module *pll*, which is already embedded in the project, was used to generation of the frequency 10 MHz. However also the frequencies 40 and 320

MHz are necessary to set initial mode (initial reset). These frequencies were not embedded in the original project. The needed frequencies were added into the already prepared module *pll*, and then this module was used to frequency generation.

4) The project code was simplified, by reason of long compilation (compilation must be done an every time after the code and settings SignalTapII changes, even if the signal is added or removed). The original module lcore had 3 input channels: din0, din8, din16. After the changes in the modules: *lcore, dfu_rb_so, serialout0;* the project became only for one channel. Also the next modules: *serialout13, serialout2;* were eliminated from the project and only one serial out: *serialout0*; was remained to simplify the research. Also the changes were added to the module *lcore.*

5) The first module, where the data is coming after ADC is called *presamples.* Presamples accumulates a number of samples so that the chip has access to before-trigger samples (useful for triggered run only). Also, the analog part provides data from ADC in 10-bit 2's complement. However, the rest of the digital part (read, DSP) is design to work with 10-bit data going from 0 to 1023, in opposite to -512 to 511 as in 2's complement format. This is compensated by adding 512 to the input by inverting the MSb as below:

```
assign din_unsigned = {~din[9], din[8:0]};
```

In this project our data (input signal) will not come from ADC, because now there is no real signal from TPC. Input data are generated by embedded Nios processor and stored in shift register. Therefore we don't need the module *presamples* now, and the data will go to the output of the module: assign dout = din;

6) The unit DSP (including units *BC1,DS (digital shaper or tail cancellation filter), BC2) was not researched* and the data do not go through this unit. I.e the input signal comes straight to the input of the module ZSU. This change was implemented at the module *filters.*

7) Optionally: the settings for ZSU are located in the module chrgu. Presamples was set to 3, and I set it to 2. Postsamples was set to 7, and I set it to 3. The glitch was remained at 2, a threshold was remained at 10 (or changed to 37). The global settings of Sampa are provided through the module tbunit. ns_e (number of sample per event (width of time window)) was 1021 (max vulue) I set 160. Contmode may be deactivated. Other settings of tbunit have not changed.

8) Changes in the code of the module ring_buffer:

a) If the positive edge of frequency is 10 MHz and if the option *d_wr_en* is set, then the data will be written to the memory and also the register of num10bit will count the quantity of the recorded data during the whole time window. Further when the time window ends, the value from num10bit transmits into the register, called size, which defines the quantity of the data must be sent from the output of the module ring_buffer. The output state machine of the module ring_buffer provides data transfer. Frequency of data transfer is 32 MHz. Below the part of the original code is shown (register num10bit):

```
always @(posedge clk10)
begin
        …………………………
        …………………………
                // Stores data and increases the number of 10-bit words on packet
                if(d_wr_en)
                        num10bit <= num10bit + 1;
                else
                        num10bit <= 0;
        …………………………
        …………………………
end
```

When a constant (above threshold, if MPW1_enZSU is set) comes to the input of the module ring_buffer, then no problems with observing the output signal appear (*d_wr_en* is set during all the time window), in that case the correct value num10bit will be sent to the register size. However if a signal, generated by a Nios, comes to the input of the module ring_buffer, then in the beginning the quantity of counted data will be correctly copied into num10bit, but further, when signal ends (*d_wr_en* will set to 0), num10bit will reset, and the output state machine will not send data, because size equals zero. To avoid this mistake the following condition was added:

```
always @(posedge clk10)
begin
        …………………………
        …………………………
                // Stores data and increases the number of 10-bit words on packet
                if(d_wr_en)
                        num10bit <= num10bit + 1;
                else if (outputState == s_sendH5)
                        num10bit <= 0;
        …………………………
        …………………………
end
```

It is not necessary that the condition was just such. The main thing is that the counted value num10bit doesn't have to change until the transfer of the output state machine to the condition s_sendH3 (look, how the value size is formed in the code).

b) After the added changes, described in the point above, the data (if they are) will appear always on the output, but not always these data will be correct. This is due to the fact, that a signal can come at any time. For example, if a signal doesn't appear at an every time window then, if the following string:

```
        assign d_rd_en = (outputState == s_sendData);
```

is not changed to:

```
        assign d_rd_en = (outputState == s_sendData) & reading;
```

data on the output will be displayed incorrectly. Unfortunately, other causes of malfunctions of the code are possible. For instance, if the time window truncates the input signal. To avoid such a kind of the problem it is necessary to improve debug relation between the pointer to reading and the pointer to writing of the data.

9) In the original project Sampa a constant comes to the input of the module *Icore*. To get more interesting results and to extend capability to manage by Sampa using this chip, Nios processor was added, which executes the following functions: forming input signal, setting of the external trigger signal. Subsequently Nios perhaps will be able to read and write global registers of Sampa. Also shift register, used for matching frequency Nios and Sampa, was added at this project. Nios writes input data in shift register and subsequently this data are sent in Sampa on the frequency 10 MHz. Shift register sends the data in Sampa only if the time window is set, and enables signal (from Nios) reset. The changing of the code of Nios gives the opportunity to get a various input signal for Sampa and to do this quickly without full compilation of the project.

The circuit of Sampa after the added changes is given in Figure 2.

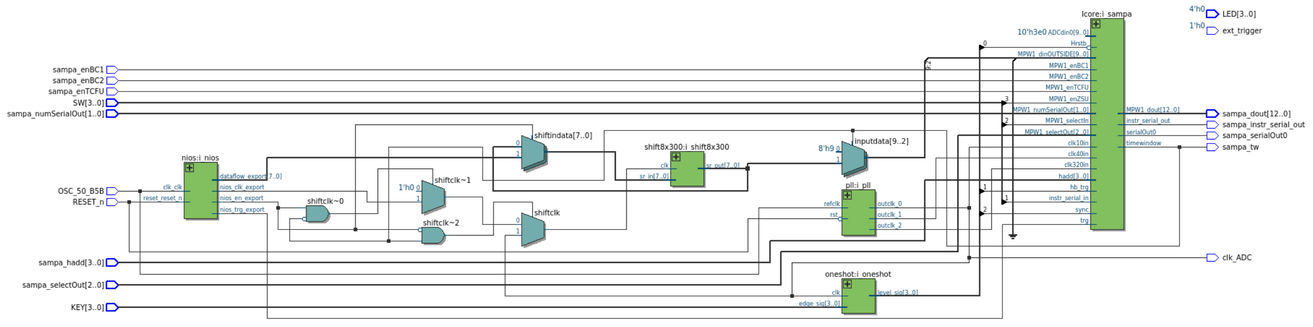

Figure 6 Circuit of Sampa with the added changes (window RTL Viewer)

Waveforms

The time window is defined as a configurable amount of samples which defines the size of a packet. This is the period, when the digital part is acquiring the samples from the ADC and feeding it to the DSP. At an every time, when the time window finishes, a new packet is generated. The chip has two main triggering methods:

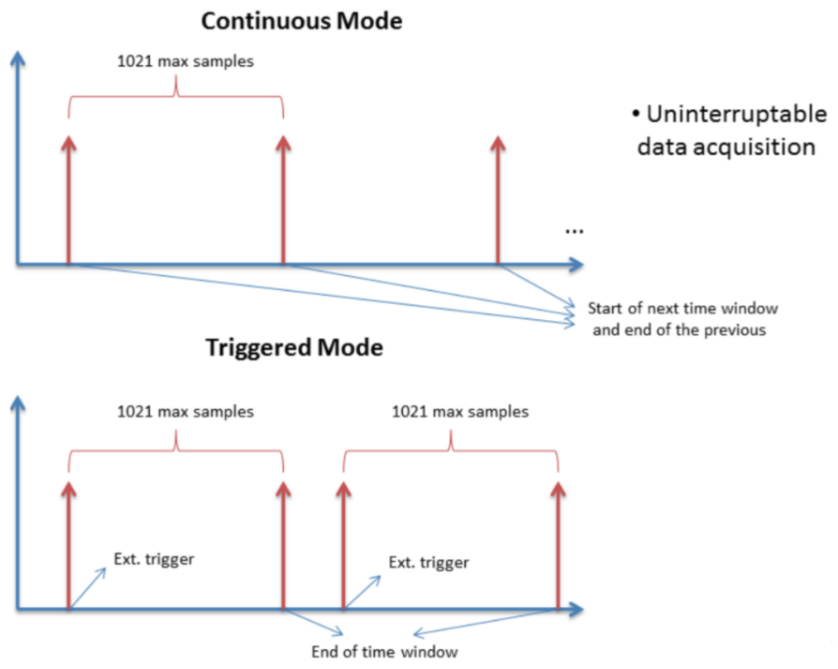

Figure 7 SAMPA's operation modes

1) Triggered: when an external signal (through dedicated trigger pin) is asserted.
2) Continuous: the time window starts automatically when continuous mode is activated.

For all waveforms, which will be shown below in this point, Sampa's operation mode will be continuous.

Example of the data, which will be coming on input of the ZSU module::

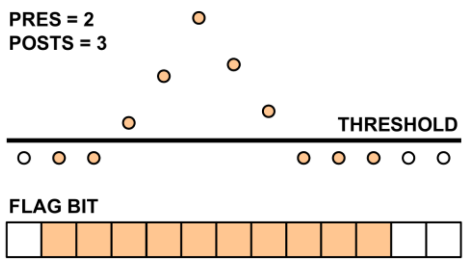

Figure 8 Example signal, which will be generated from Nios

Waveforms for ZSU module given in Figure 9.

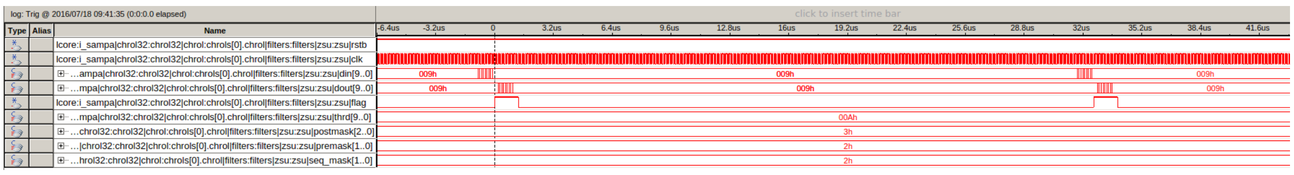

Figure 9a ZSU waveform test

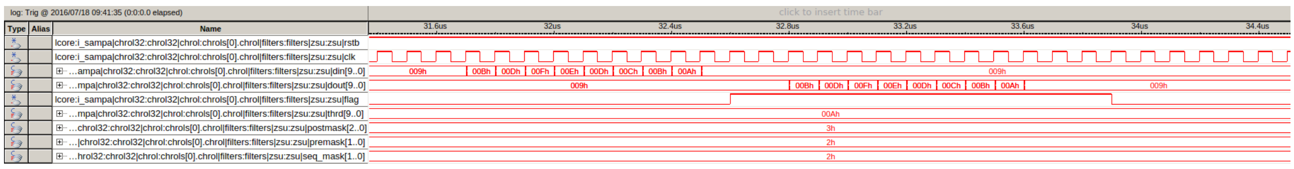

Figure 9b Processing the input data

Waveforms for DFU module given in Figure 10.

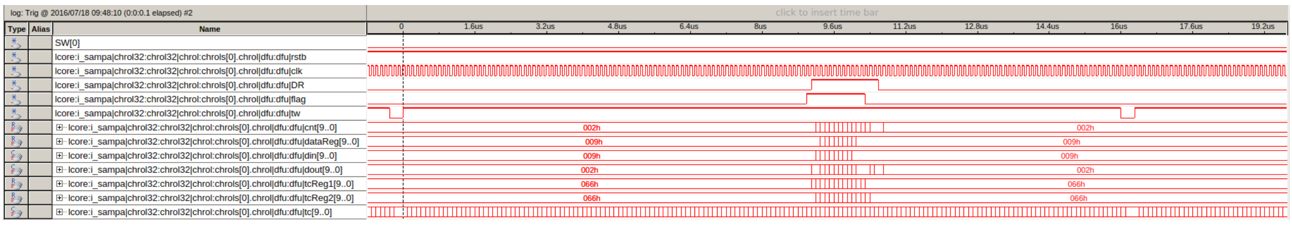

Figure 10a DFU waveform test

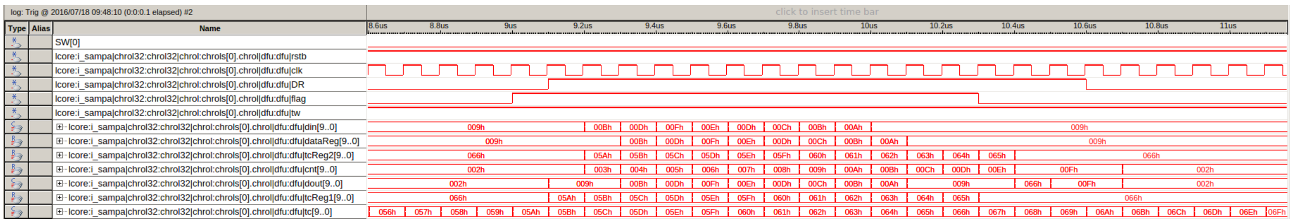

Figure 10b Assertion of flag coming from ZSU. Formation of DFU data

Waveforms for ring_buffer module given in Figure 11.

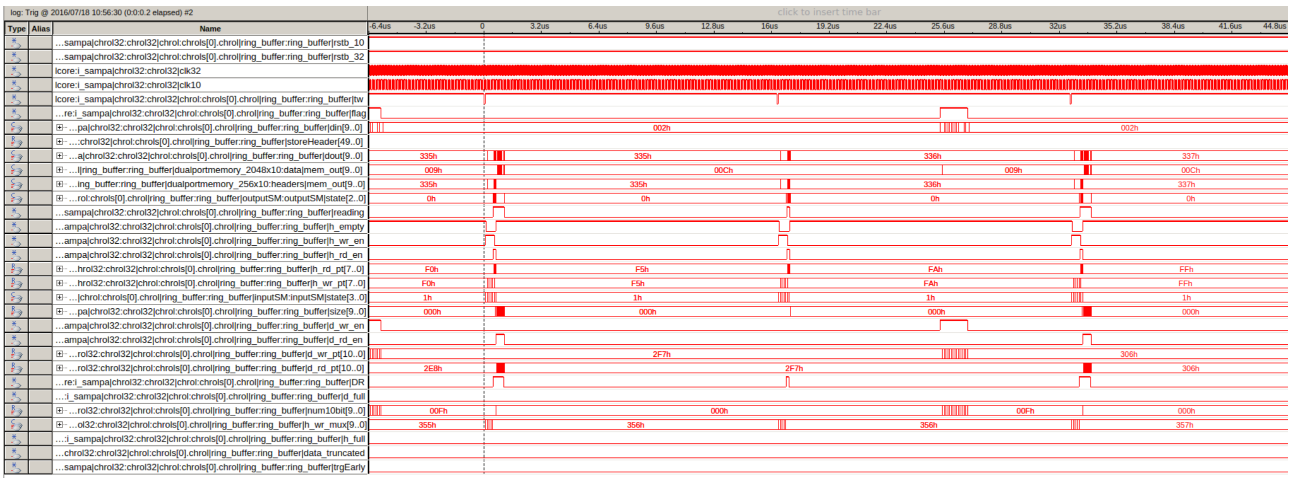

Figure 11a ring_buffer waveform test

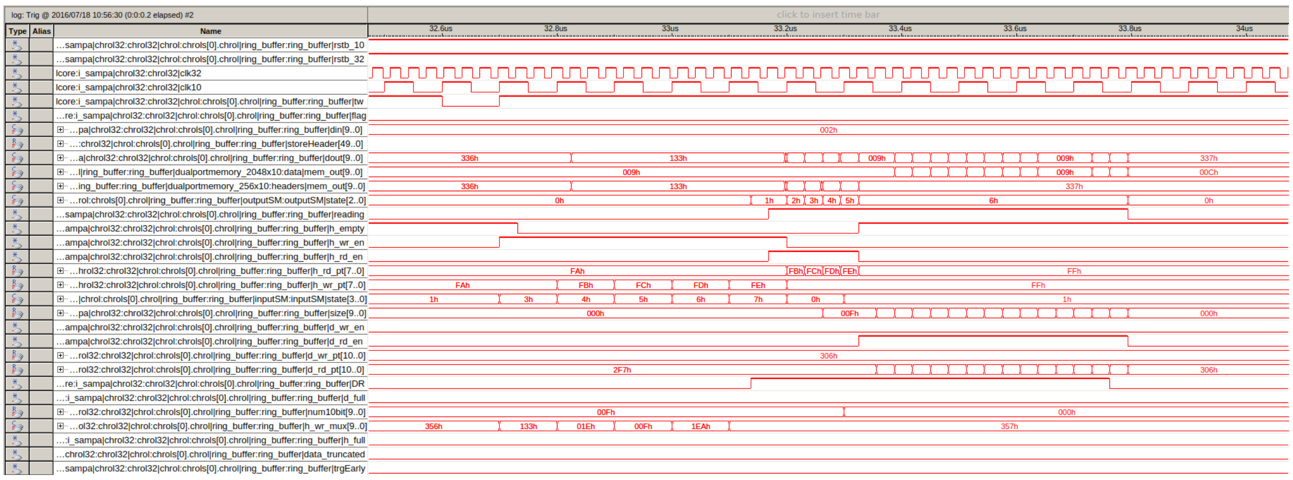
log: Trig @ 2016/07/18 10:56:30 (0:0:0.2 elapsed) #2
click to insert time bar
Type
Alias
Name
...sampa|chrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|rstb_10
...sampa|chrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|rstb_32
Icore:i_sampa|chrol32:chrol32|clk32
Icore:i_sampa|chrol32:chrol32|clk10
Icore:i_sampa|chrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|tw
...re:i_sampa|chrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|flag
...pa|chrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|din[9..0]
...:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|storeHeader[49..0]
...a|chrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|dout[9..0]
...l|ring_buffer:ring_buffer|dualportmemory_2048x10:data|mem_out[9..0]
...ing_buffer:ring_buffer|dualportmemory_256x10:headers|mem_out[9..0]
...rol:chrols[0].chrol|ring_buffer:ring_buffer|outputSM:outputSM|state[2..0]
...sampa|chrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|reading
...ampa|chrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|h_empty
...ampa|chrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|h_wr_en
...ampa|chrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|h_rd_en
...hrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|h_rd_pt[7..0]
...hrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|h_wr_pt[7..0]
...|chrol:chrols[0].chrol|ring_buffer:ring_buffer|inputSM:inputSM|state[3..0]
...pa|chrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|size[9..0]
...ampa|chrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|d_wr_en
...ampa|chrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|d_rd_en
...rol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|d_wr_pt[10..0]
...rol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|d_rd_pt[10..0]
...re:i_sampa|chrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|DR
...i_sampa|chrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|d_full
...rol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|num10bit[9..0]
...ol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|h_wr_mux[9..0]
...i_sampa|chrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|h_full
...chrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|data_truncated
...sampa|chrol32:chrol32|chrol:chrols[0].chrol|ring_buffer:ring_buffer|trgEarly
32.6us
32.8us
33us
33.2us
33.4us
33.6us
33.8us
34us

Figure 11b When the time window ends then forms beginning of a packet

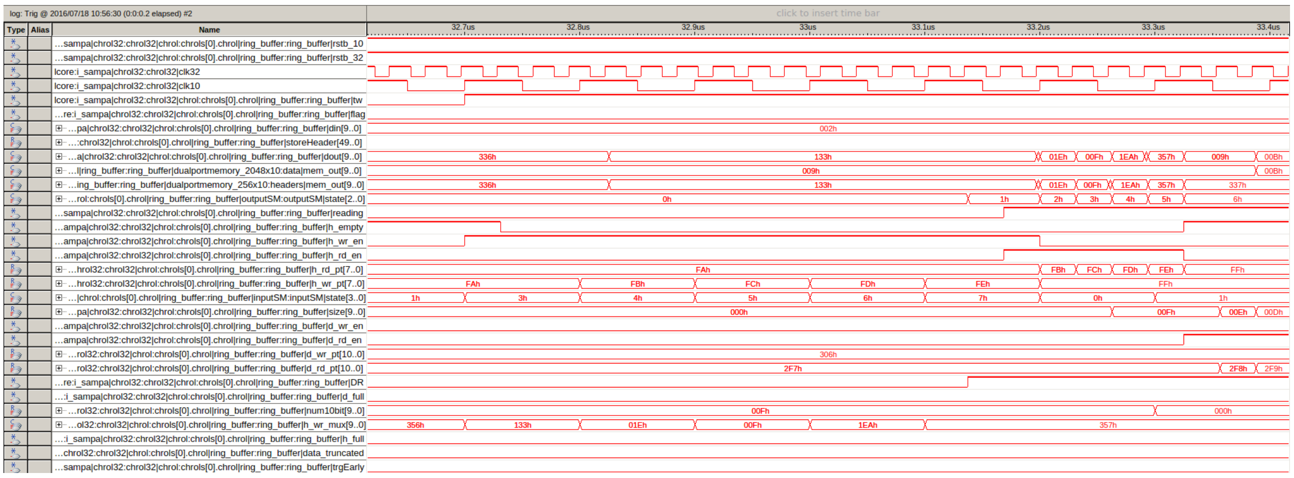

Figure 11c The end of the time window. Formation of header

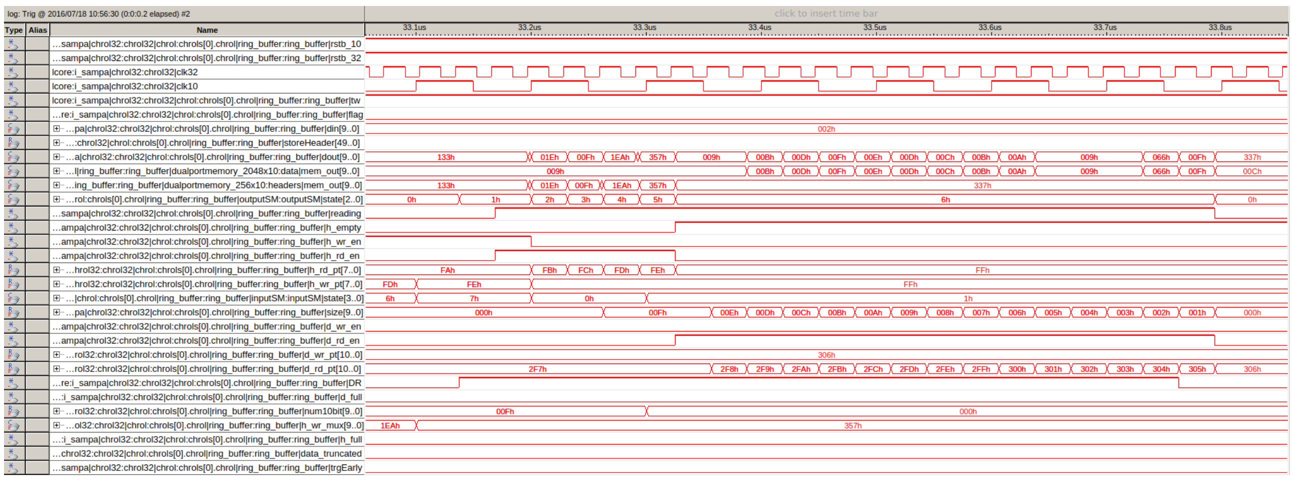

Figure 11d The end of the time window. Showed packet of data

Waveforms for serialout0 module given in Figure 12.

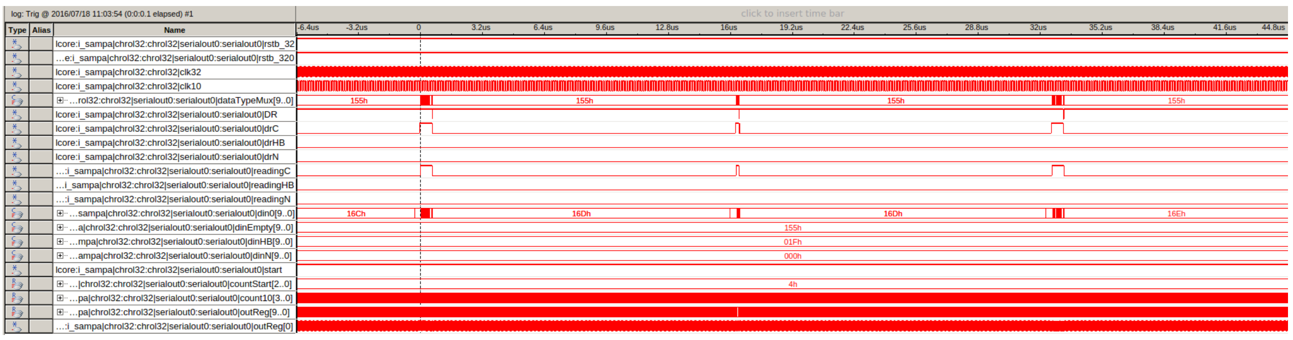

Figure 12a serialout0 waveform test

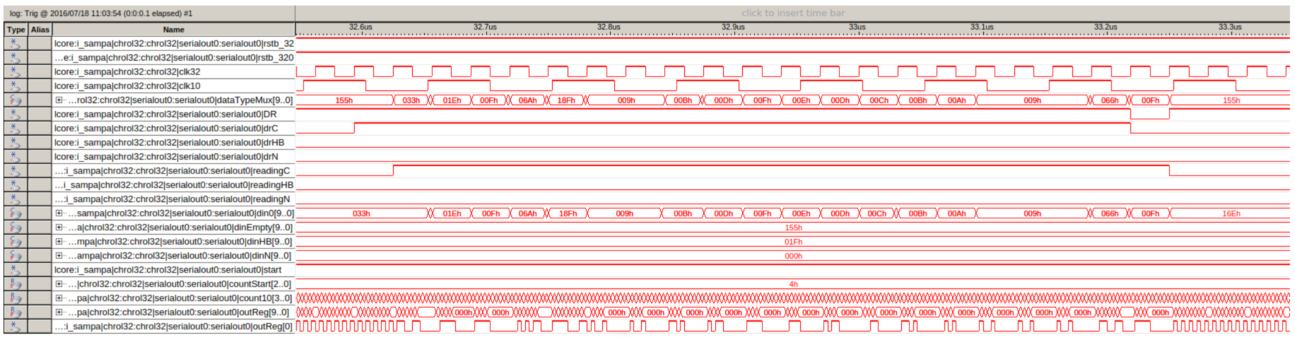

Figure 12b Getting data from the ring_buffer module. Copying data in register dataTypeMux

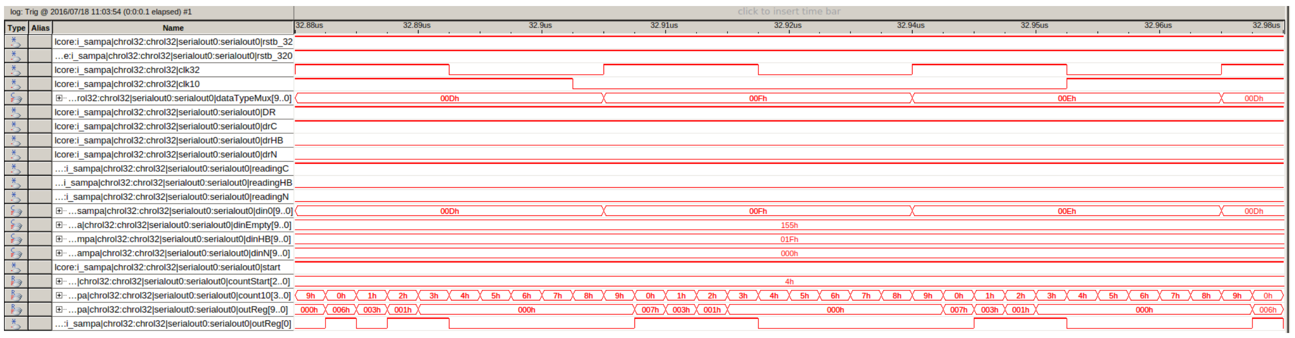

Figure 12c Sending data through serial interface

Waveforms for heartbeat module given in Figure 13.

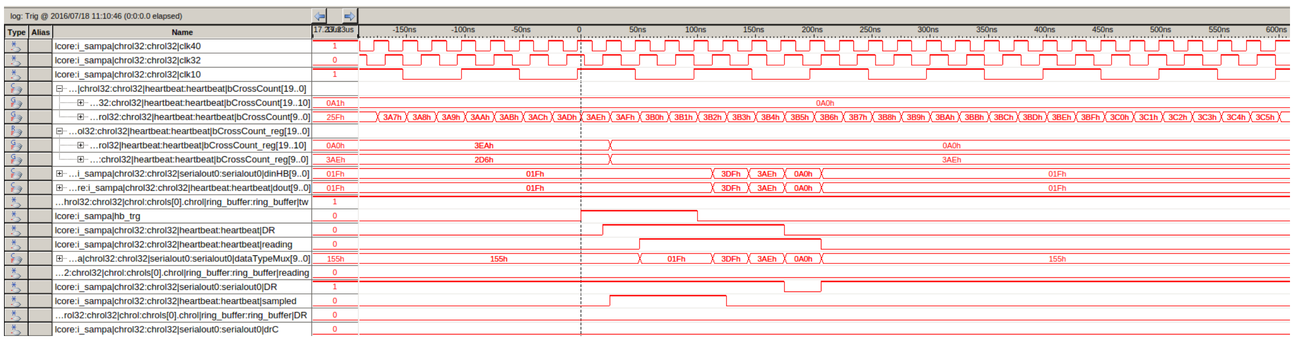

Figure 13 Heartbeat waveform test

Operation of the ZSU module

Initial conditions:
frequency = 10 MHz;
1 sample = 1 period = 0,1 us;
ns_e (number of sample per event) = 300;
the width of the time window = ns_e * period = 300 * 100 = 30 us;
ns_s (number of sample per signal)= 8 ;
The width of the signal = ns_s * period = 8 * 100 = 0,8 us;
Number of sample per header = 5;
Number of presamples[1] = 2;
Number of postsamples[2] = 3;

Number of signals on top base line	Number of sample per input signals	Number of sample per payload	Number of sample per packet when ZSU filter enabled	Number of sample per packet when ZSU filter disabled
0	0	0	5	307
1	8	15	20	307
2	16	30	35	307
3	24	45	50	307
4	32	60	65	307

packet = payload + header;
payload = Presamles + ns_s + Postsamples + TC[3] +CS[4];
payload 0 = 0 + 0 + 0 + 0 + 0;
payload 1 = 2 + 8 + 3 + 1 + 1;
payload 2 = 4 + 16 + 6 + 2 + 2;
payload 3 = 6 + 24 + 9 + 3 + 3;
payload 4 = 8 + 32 + 12 + 4 + 4;

1. Presamples is samples before the actual pulse arrives.
2. Postsamples is samples after the actual pulse arrives.
3. Cluster size (CS): 10-bit field containing a 0 to 1023 value that represents the number of 10-bit words on the cluster. It includes itself and the TC, as the sum of all clusters.
4. Time count (TC): 10-bit field, which contains the time from 0 to 1023 of the first valid sample, locating the cluster data in time, as the time information is lost due to Zero Suppression.

Conclusions

The limited set of SAMPA chip is manufactured and available for evaluation. The test platform including a processor is required for verification the chip and for check the chip conformance to specification. Present work is based on the chip HDL description (SAMPAemu) which targeted for design testing procedures. A concept for the test platform was designed using FPGA – equipped Altera development board. An Idea is to provide test platform with embedded RISC processor (Nios) as a board controller. Nios generates test vectors, configuring mode operation of the SAMPAemu and initialize data processing.

The processor provides opportunity for check the work of the SAMPAemu for any input data, i.e Nios can model signals from ADC. As firmware Nios and SAMPAemu both synthesized in FPGA test procedures flexibility are achieved.

Nios takes few FPGA resources and can be synthesized in the width range Altera chips, for example Cyclone III or MAX 10.

While developing test platform firmware system level debugging tool SignalTapII was used. The tool allows to examine signals that are going to occur anywhere within a circuit implemented in an FPGA chip. Important feature of SignalTapII is the fact that it doesn’t spoil parameters of measured signals. This is the guarantee that waveforms given in above is correct.

Further project development assumes to control SAMPAemu state via I^2C interface. Also it is important to study operation of DSP (including Base line correction and tail cancellation filter). The work performed above is base for future development.

At the present time the chip SAMPA is very perspective solution for TPC readout system design.

References

[1] https://indico.cern.ch/event/357738/contributions/848776/

[2] http://nica.jinr.ru/physics.php

[3] http://folk.uib.no/ave082/SAMPA/heitor_graduation_project_SAMPA.pdf

[4] http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=205&No=816&PartNo=4

[5]ftp://ftp.altera.com/up/pub/Altera_Material/16.0/Tutorials/VHDL/Debugging_Hardware.pdf