Technical note
Quad-channel LArPix R/O Unit V3.0
(Firmware larpix4.11fifos.bit)

Igor Kreslo*

January 7, 2019

1 Introduction

The Unit is intended to drive operation of a number of LArPix ASICs, connected in daisy-chains. General view of the detector readout is shown in figure 1. Up to four daisy chains can be served by a single R/O Unit. The Unit is composed of commercial Arty-Z7 FPGA evaluation module from Digilent and a custom-designed mezzanine (Figure 2). Up to four LArPix chains can be connected via standard DIL-10 pin headers and ribbon cables. Signals from several units, in turn, can be consolidated into one Gigabit optical link with the use of commercial OEM network switch with media converter.

The Unit design allows to build readout systems scalable from a single laptop-controlled device for a table-top quick test, up to a full size multi-million pixel DUNE-ND detector readout. Current layout of mezzanine board is designed to match connectors of the first test pixel charge readout boards. It will, of course, be optimized for any larger-scale applications.

The Unit requires one +5V DC power supply, capable of supplying 1A of current. Typical current consumption (28 ASICs in a chain) does not exceed 620 mA, ating at about 470 mA for normal DAQ data rate (0.1 Hz/pixel). The Unit provides analog and digital control signal to the ASICs, the back end data communication is provided via Ethernet, namely implementation of Zero-MQ protocol.

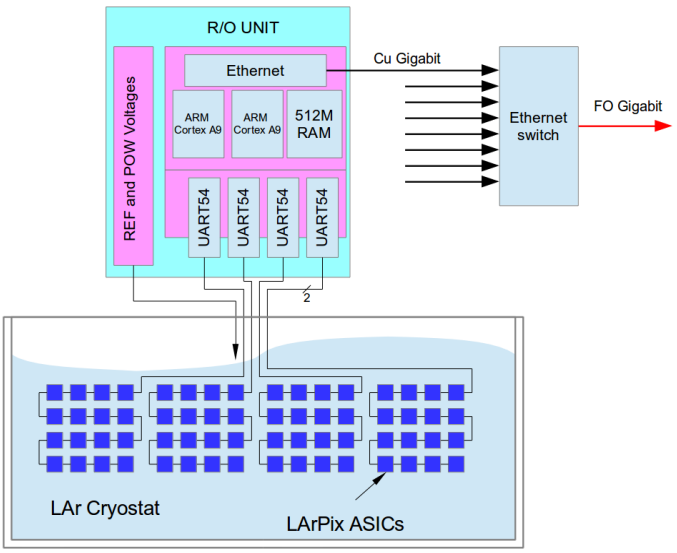

Figure 1: General concept of the LArPix-based Liquid Argon TPC readout.

*Igor.Kreslo@lhep.unibe.ch; LHEP, University of Bern, Switzerland.

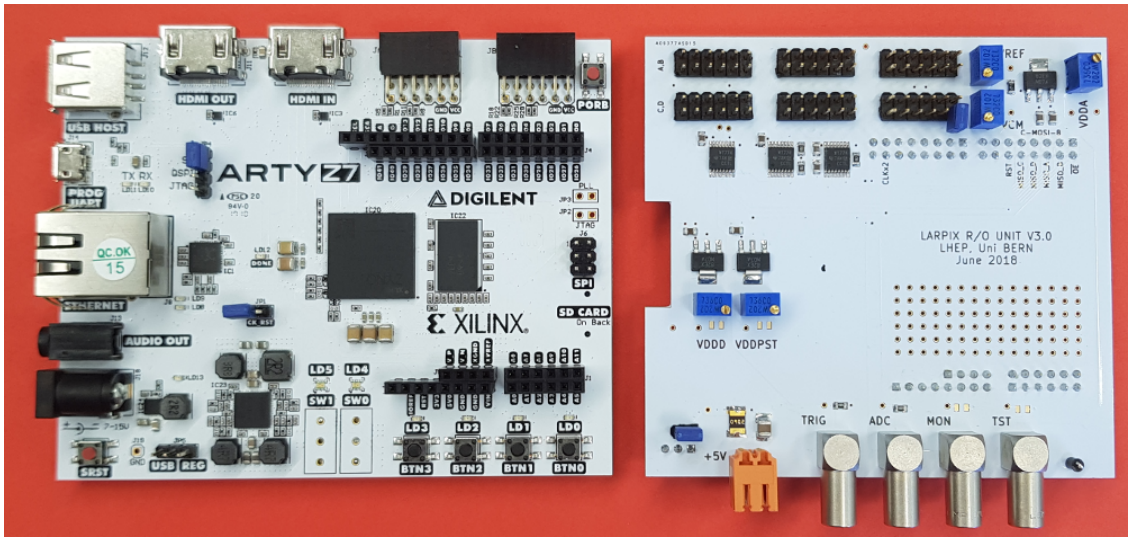

Figure 2: R/O Unit is composed of commercial ArtyZ7 board by Digilent (left) and custom mezzanine board placed on top of it (right).

2 The mezzanine

The mezzanine (see Figures 3, 7) contains voltage regulators to provide VDDD, VDDPST, VDDA power levels for the ASICS and dividers to provide VCM and VREF for their ADCs. The mezzanine contains a number of level adapters as well, allowing to couple 3.3V CMOS pins of FPGA to the ASICs, operating at VDDPST from 1.2V to 3.3V. The mezzanine also hosts main power connector (+5V) and a resettable fuse for 3A. The mezzanine supplies +5V power to the main Arty-Z7 board via a 2-pin header. A pixel readout board, that hosts LArPix ASICs daisy chains is connected to the mezzanine via standard DIL-10 pin headers and ribbon cables (see connector pinout on Figure 4).

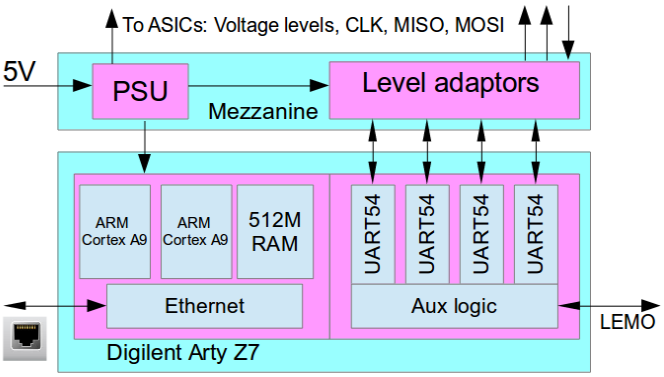

Figure 3: Structure of Readout Unit.

3 The main board and PL hardware level

The digital part of the Unit is represented by Xilinx Zynq-7020 system-on-chip. The chip contains PS and PL parts: PS is double-core high performance ARM processor system, PL is a programmable logic fabric similar to Spartan-6 family. The PL part of the design comprises 3 subsystems (Figure 5):

1. Programmable Clock and Reset Generator (CLKGEN);
2. 2-channel PWM RGB LED color modulator unit (PWMx6.0);

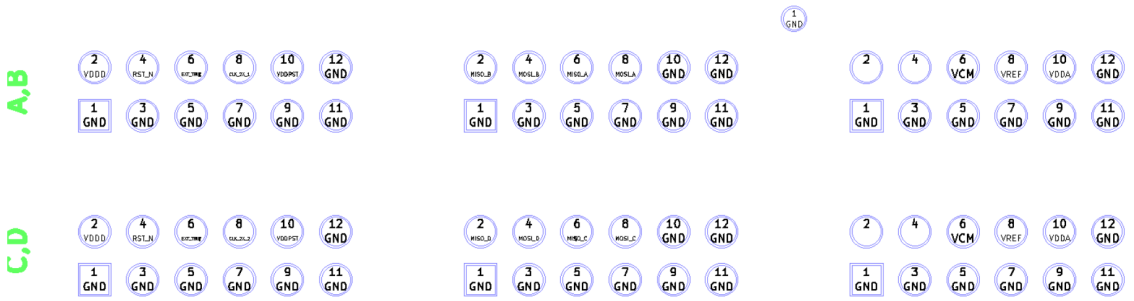

Figure 4: Connector for pixel readout board with ASIC daisy chains.

3. Four 54-bit synchronous UART units (UART54_A - UART54_C).

The UART54 block is further expanded in Figure 6.

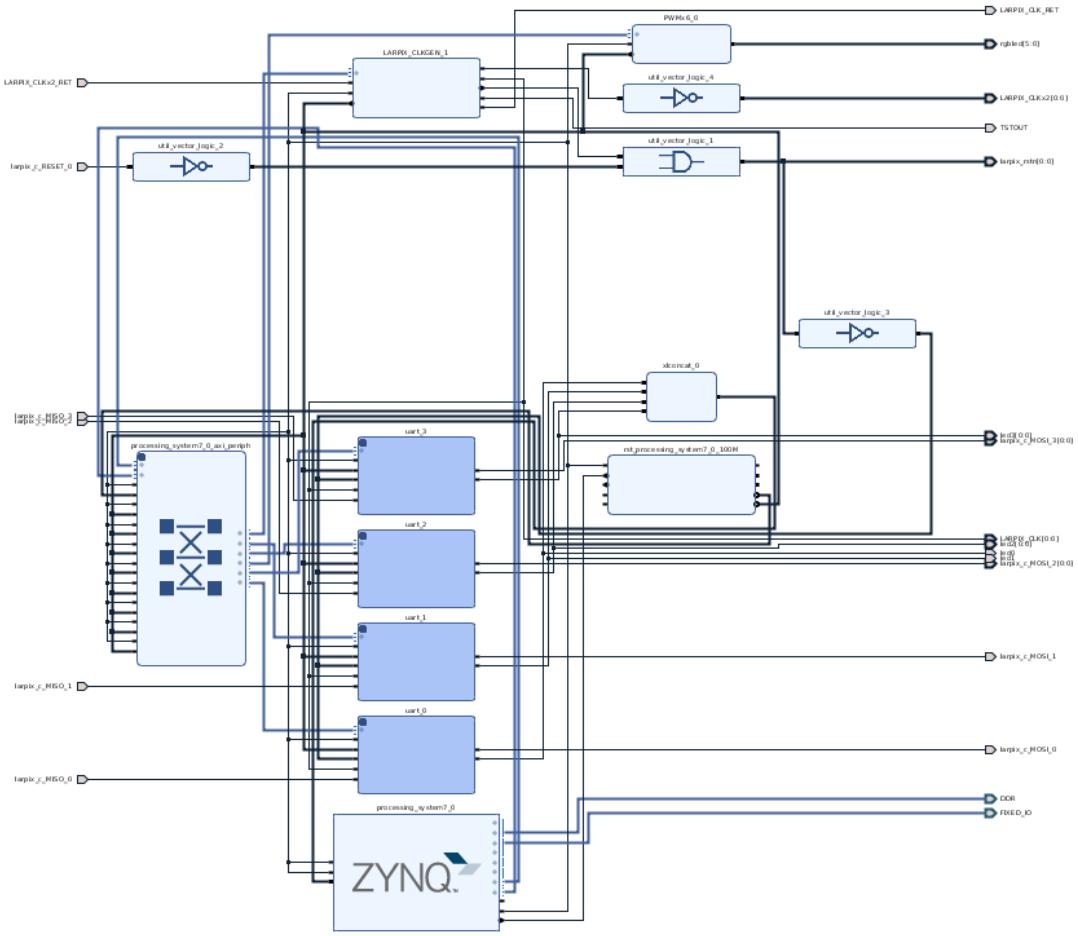

Figure 5: Block design for the Zynq-7000 Programmable Logic fabric.

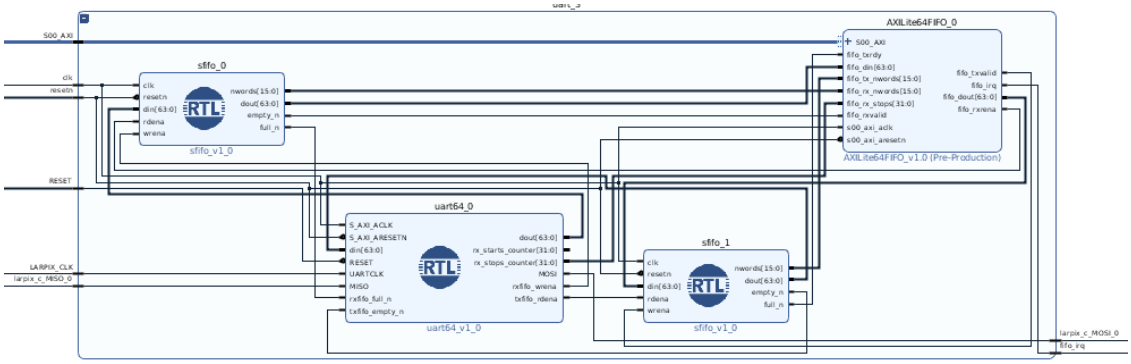

Figure 6: Block design for UART54.

CLKGEN is a 32-bits AXI Slave, located at 0x43C00000 on the PS AXI Bus. It has three 32-bits wide control registers: CLOCKx2_DIVIDER TSTCLK_DIVIDER and SYSTEM_RESET.

Writing to the CLOCKx2_DIVIDER register sets the CLOCKx2 frequency to 100000/(CLOCKx2_DIVIDER+1) kHz.

Writing to the TSTCLK_DIVIDER register sets the pulse frequency at TST output LEMO to CLOCKx2/(TSTCLK_DIVIDER) kHz.

Writing 0x1 to SYSTEM_RESET register asserts RESET signals for all UARTS and ASICs. Writing 0x0 to SYSTEM_RESET releases normal operation.

PWMx6.0 is a 32-bits AXI Slave, located at 0x43C05000 on the PS AXI Bus. Six 4-bytes long registers control brightness of two RGB LEDs located on the main board. Register value range is 0x0000-0xFFFF.

Four UART54 units are controlled by a uart64n kernel driver, loaded at the on-board OS startup. This creates four character devices, which can be read/write from the user space: /dev/uart640 - /dev/uart643. For the user application there is no need to access UART54 hardware addresses, the driver does all required job. However for the sake of documenting below the hardware addresses assignment are given. Four UART54 are 64-bits AXI Slaves with base addresses at 0x43C01000, 0x43C02000, 0x43C03000 and 0x43C04000. Each UART54 has the following registers:

- DATA: interface to on-board data RX and TX FIFOs. Read and write operations are transparent. Content of FIFOs can be checked accessing STATS register. Each FIFO has depth of 256 64-bits words.
- IRQR: Bit 0 when set enables interrupt generation, bit 1 is the Interrupt signal, generated by UART on receive of 54-bits word from ASICs. Once set, interrupt can be reset by writing 0x0 into IRQR. In the firmware V4.11 Interrupt signal drives one of four green LEDs and routed to CPU Generic Interrupt Controller (inputs IRQ_F2P0 to IRQ_F2P3).
- STATS: 64-bits word, containing status of RX/TX FIFOs and UART received words for loss control:
  bits 0-15: RX FIFO words
  bits 16-31: TX FIFO words
  bits 32-63: UART54 received words since power up (can't be reset)

Normally, RX/TX FIFO words counters are read by the driver and there is no need to access them from the user space. The number of words received by UART, however, may be monitored by host-side software for word loss rate check, for instance.

4 PS Linux and hardware control low-level software

The PS part of the Zynq chip runs optimized version of Ubuntu Linux. The network interface is configured via DHCP of to a fixed IP by dedicated string in the on-board startup script, for instance:

Subsystem	Register	Address	Bytes	Function
CLKGEN	CLOCKx2_DIVIDER	0x43C00000	4	System clock divider
CLKGEN	SYSTEM_RESET	0x43C00004	4	System reset (active high)
PWMx6.0	LED1_B	0x43C05000	4	LED1 Blue brightness
PWMx6.0	LED1_G	0x43C05004	4	LED1 Green brightness
PWMx6.0	LED1_R	0x43C05008	4	LED1 Red brightness
PWMx6.0	LED2_B	0x43C0500C	4	LED2 Blue brightness
PWMx6.0	LED2_G	0x43C05010	4	LED2 Green brightness
PWMx6.0	LED2_R	0x43C05014	4	LED2 Red brightness
UART54_A	UART54_A_DATA	0x43C01000	8	Data RX/TX
UART54_A	UART54_A_STATS	0x43C01008	2	RX FIFO content
UART54_A	UART54_A_STATS	0x43C0100a	2	TX FIFO content
UART54_A	UART54_A_STATS	0x43C0100c	4	Words received by UART54 since startup
UART54_A	UART54_A_IRQR	0x43C01010	1	Interrupt register
UART54_B	UART54_B_DATA	0x43C02000	8	Data RX/TX
UART54_B	UART54_B_STATS	0x43C02008	2	RX FIFO content
UART54_B	UART54_B_STATS	0x43C0200a	2	TX FIFO content
UART54_B	UART54_B_STATS	0x43C0200c	4	Words received by UART54 since startup
UART54_B	UART54_B_IRQR	0x43C02010	1	Interrupt register
UART54_C	UART54_C_DATA	0x43C03000	8	Data RX/TX
UART54_C	UART54_C_STATS	0x43C03008	2	RX FIFO content
UART54_C	UART54_C_STATS	0x43C0300a	2	TX FIFO content
UART54_C	UART54_C_STATS	0x43C0300c	4	Words received by UART54 since startup
UART54_C	UART54_C_IRQR	0x43C03010	1	Interrupt register
UART54_D	UART54_D_DATA	0x43C04000	8	Data RX/TX
UART54_D	UART54_D_STATS	0x43C04008	2	RX FIFO content
UART54_D	UART54_D_STATS	0x43C0400a	2	TX FIFO content
UART54_D	UART54_D_STATS	0x43C0400c	4	Words received by UART54 since startup
UART54_D	UART54_D_IRQR	0x43C04010	1	Interrupt register

Table 1: PL Subsystems and their address map.

```
ifconfig eth0 130.92.139.27 netmask 255.255.255.0 broadcast 130.92.139.255
    gateway 130.92.139.1
ifdown eth0; ifup eth0;
```

This file should be edited nefore you run the system first time in your network. To access this file mount the microSD card at your PC and open **rootfs** file system. Navigate to **/home/ubuntu** and edit **setup** script. Commenting out ifconfig and ifdown lines will restore DHCP. Otherwise set up the IP you want to assign in the ifconfig line.

Username and password are written on the back of the main board. Connect to the board via SSH client. There is a bunch of executables in the home directory, that help to debug the system. Main interface library is pixlar.c(h). It encapsulates access to the PL level hardware and makes it more convenient.

In the previous version (V25.0/4) the uart54 access was realized from the user space via memory mapping functions. This is terribly slow approach and starting from version 3.0 it is replaced by a hardware kernel driver "uart64" (control of CLKGEN and PWMx6.0 remains via mapping as non-time critical). The driver takes advantage of using interrupts, generated by UART and crucially increases data throughput. the driver is compiled as a loadable module and must be loaded upon startup by (normally done by **setup** script):

```
sudo insmod uart64.ko
```

Once loaded, the driver creates four nodes /dev/uart640 to /dev/uart643, which can be accessed by standard linux char device access functions open(), read(), write() and close(). The uart54 transmit is triggered each time at least 8 bytes are written into device at once, only lower 54

bits are actually sent over MOSI-MISO lines. Reading is performed by 8-bytes groups as well. By default these nodes do not have w/r permission from normal user, so must be granted with (normally done by **setup** script):

```
sudo chmod 666 /dev/uart64*
```

By default interrupts are disabled for all four uarts. The driver enables them after first read request from user program. The uart64 driver has a buffer for 1024 64-bits words. When receive buffer becomes full (no reads from /dev/uart64x and LArPix data coming to unit) the driver disables interrupts. From this moment on, the incoming data will be lost until next read access. The read access to the corresponding /dev/ node re-enables the interrupts and re-allows data storage.

If the module is intended to operate in production mode, one needs to run only one executable (normally done by **setup** script):
sudo ./pixlar
This will start two services, one for command interface and one for data interface. After that the client may connect to this interfaces via Zero-MQ protocol and provide configuration or acquire data.

Additional service executable **./silence** can be invoked to disable all ASICs on the board. Update it with your corresponding list of ASICs! Normally this command is executed by **setup** script, but sometimes it needs manual execution. In this case connect by SSH and execute this command explicitly.

5 Zero-MQ server-client and protocol

The two services running on the module are pixlar_cmdserver and pixlar_dataserver. The first one listens for REQ-REP ZMQ connection at TCP:5555. The second one offers SUB-PUB socket at TCP:5556.

The CERN Root-based host-side library (PIX0MQ) should be compiled on a client PC with Root installed (**compile** script). PIX0MQ root object provides an API for all necessary configuration and data functions. The API is declared in PIX0MQ.hxx and most of functions names are self-explaining. For the LArPix ASIC-related functions, refer to an ASIC datasheet.

Typical use of the host-side ROOT test scripts:

```
root -l 'ScanThresholdsFine.C(207,27,10000)'
```

Two high-level GUI executables are provided for data quality monitoring and ASIC configuration.

```
root -l DisplayMain.C+g
```

opens Data Monitor window.

```
root -l Configurator.C+g
```

Opens ASIC configurator window. It assumes all configurations are saved in CONF directory and follows naming convention:

```
T0C0ASIC48.conf
```

T0 stands for tile 0; C0 stands for daisy chain 0 (can be in the range 0 to 3), ASIC48 speaks for itself.

Additionally, file chips.list defines population of the daisy chains and ASIC coordinates at the tyle:

```
48 0 1 7
```

is the line for ASIC 48, daisy chain 0, x=1, y=7.

6 Addendum

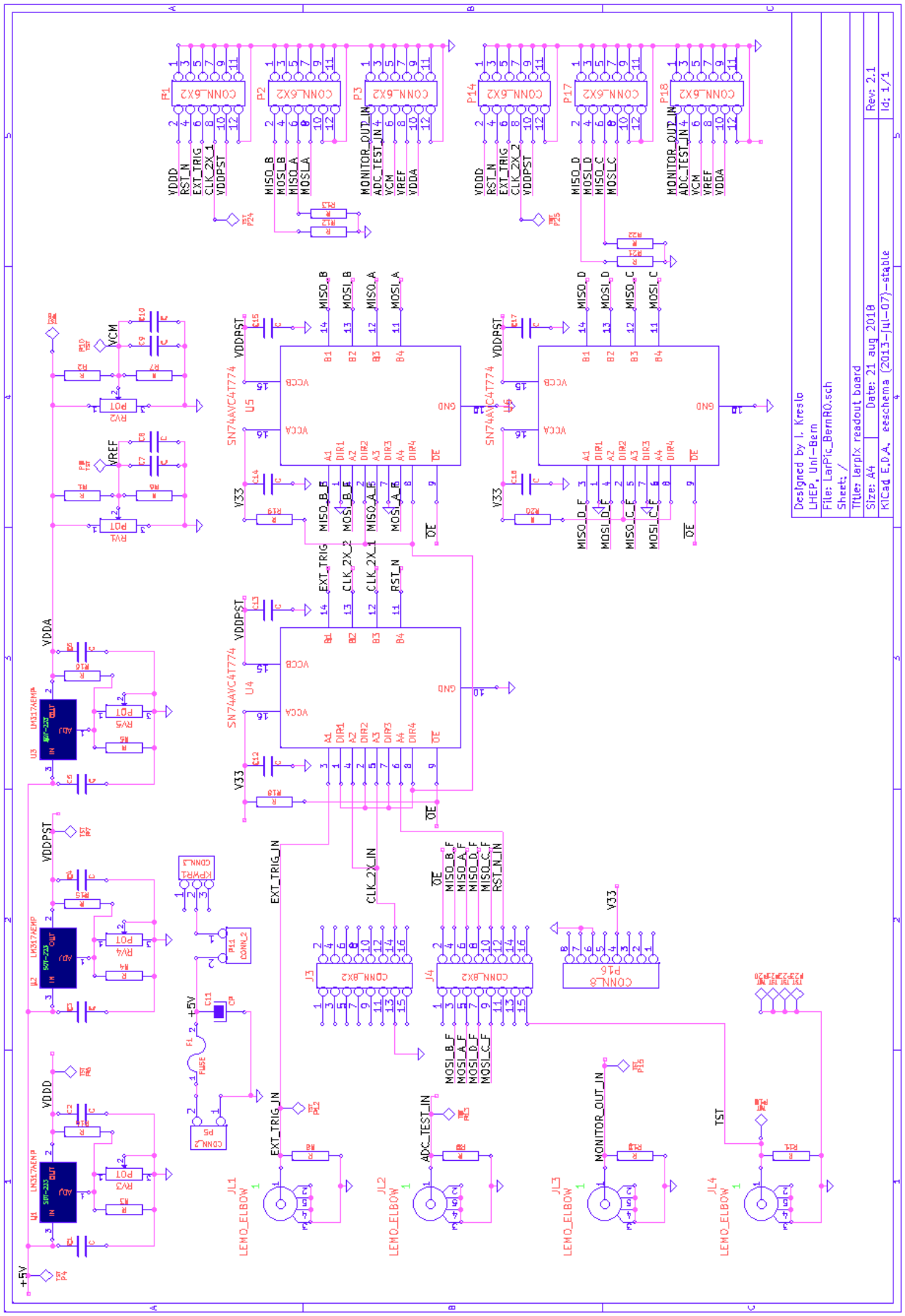

Figure 7: Schematics of the LARPIX R/O UNIT V2.0 Mezzanine.

6.1 Example ASIC configuration file T0C0ASIC48.conf

```
48  'ASIC ID (0-255)
0  'at tile Nr
0  'connected to UART channel (0-3)
15  'threshold_global (0-255)
24 29 24 27 23 28 27 26 24 23 18 29 25 26 24 23 26 22 30 25 25 28 23 23 21 23 21 23 26 24 26 25  '
1  'csa_gain: 0=45uV/e, 1=4uV/e
0  'csa_bypass_enable: 0 or 1; 1-Enables injection of ADC_TEST_IN to selected ADC
0x00000000  'csa_bypass_select - 32 bits, HIGH bits select channel for injection
0  'internal_bias: 0 or 1; UNUSED
1  'internal_bypass: 0 or 1; 1-Internal capacitor(?) on power line
0x00000000  'csa_monitor_select: 32 bits, only one HIGH to select channel to connect to analog MON
0xffffffff  'csa_testpulse_enable:  32 bits, LOW connects chan inputs to test pulse generator
0  'csa_testpulse_dac (0-255)
0  'test_mode: 0 - normal DAQ, 1 - UART test, 2 - FIFO burst
0  'cross_trigger: 0 or 1, 1- all not masked channels digitize when one is triggered
1  'enable_periodic_reset: 0 or 1, 1=periodic reset after reset_cycles
4096  'reset_cycles (0-2^24), in Master clock/4 cycles
0  'fifo_diagnostic_en: 0 or 1, Embeds FIFO counter to output event
1  'sample_cycles, 0-255. 0 or 1 - one system clock period used for sampling
255  'test_burst_length, 0-2^16, for FIFO burst test
0  'adc_burst_length: 0-255, number of ADC conversions per pix hit, 0 or 1 = one sample per hit
0x00000000  'channel_mask: 32 bits, HIGH to disable channel
0xffffffff  'external_trigger_mask: 32 bits, HIGH to disable external trigger to channel
```

6.2 Example application: ScanThresholdsFine.C

```
  PIX0MQ *p;
  TCanvas *c;
  TH1F *hrate;
  TGraph *grate;
#include "InitBoard.h"

void ScanThresholdsFine(uint64_t chipid=80, uint64_t gth=22, int kHz=10000)
{
  c=new TCanvas();
  c->SetLogy(1);
  c->SetGridx(1);
 c->SetGridy(1);
  uint64_t th;
  uint32_t mask;
  Float_t rate, rate_prev=1e7;
  p=new PIX0MQ("130.92.139.27:5555");
p->ControlSocketSetClock(kHz);
    if(InitBoard("CONF/chips.list")==0) return;
    if(chipchannel[chipid]<0) {printf("Nonexisting chip!\n"); return;}
  p->ActiveChannel=p->chipchannel[chipid];

  p->ASIC_threshold_global(chipid,gth);

  char nm[256];
  sprintf(nm,"ASIC %lu pix rate vs pix trim threshold, Glob_th=%lu, Fx2=%d kHz",chipid,gth,kHz);
  hrate=new TH1F("hrate",nm,32,0,32);
  hrate->GetXaxis()->SetTitle("pixel_trim_dac");
  hrate->GetYaxis()->SetTitle("Pixel event rate, Hz");
  hrate->SetMaximum(200000);
  hrate->SetMinimum(0.1);
```

```
  grate=new TGraph(0);
  mask=1;
  hrate->Draw("hist");

  for(int ch=0; ch<32; ch++) p->ASIC_pixel_trim_dac(chipid,ch,0x1f); //max threshold everywhere

  for(int ch=0; ch<32; ch++)
{
  hrate->Reset();
  grate->Clear();
  grate->Set(0);
  rate_prev=190000;
  for(th=0; th<32; th++)
  {
  p->ASIC_pixel_trim_dac(chipid,ch,th);
  p->ASIC_channel_mask(chipid,(~mask));
  rate=p->GetRate(p->chipchannel[chipid],200);
usleep(100);rate=p->GetRate(p->chipchannel[chipid],200); }
 rate=p->GetRate(p->chipchannel[chipid],200); }

  grate->SetPoint(th,th,rate);
  grate->Draw("lsame");

  c->Update();
  p->ASIC_channel_mask(chipid,0xffffffff);
  printf( "Pixel %d Threshold %lu rate %f\n",ch,th,rate);
  if(rate==0 && rate_prev==0) break; //to save time
  rate_prev=rate;
  }

  grate->SetLineColor(1+ch%5);
  grate->DrawClone("lsame");
  mask=mask<<1;
}
  p->ActiveChannel=-1;

  p->~PIX0MQ();
}
```

6.3 Header file with service functions: InitBoard.h

```
#include <stdio.h>
  int nchips;
  int chipchannel[256];
  FILE *fp1;
  char line[128];

int InitBoard(const char *list)
{
 int chip, chan; //, na, x,y;

 p->InitChipList(list);
 if(p->nchips<=0) return 0;

 for(chip=0; chip<256; chip++) if(p->chipchannel[chip]>=0)
{
    chan=p->chipchannel[chip];
    printf("Configuring Chip %d channel %d...\n",chip,chan);
```

```
    p->ActiveChannel=chan;
    p->ASIC_csa_monitor_select(chip, 0);
    p->ASIC_channel_mask(chip,0xffffffff);
    p->ASIC_threshold_global(chip,250);
    p->ASIC_external_trigger_mask(chip,0xffffffff);
    p->ASIC_reset_cycles(chip,4096);
    p->ASIC_reg47(chip,0,0,1,0); //enable periodic reset
    p->ASIC_reg33(chip,1,0,1,1); //enable internal bypass
    p->ASIC_adc_burst_length(chip, 1); // 1 conversion per trigger
  for(int ch=0; ch<32; ch++) p->ASIC_pixel_trim_dac(chip,ch,0x0); //max threshold everywhere
}

  return 1;
}

void dump(unsigned char * ptr)
{
   int i;
     for (i = 0; i < 8; ++i)
        printf("%02x", *(unsigned char*)(ptr+7-i));
        printf("\n");

}



void dumpd(uint64_t word)
{
   int bc=0;
      printf("(");
      for (bc = 0; bc < 64; bc++)
     {
       if(bc==2) printf(") id:");
       if(bc==10) printf(" ch:");
       if(bc==17) printf(" ts:");
       if(bc==41) printf(" adc:");
       if(bc==51) printf(" fifo/2:");
       if(bc==52) printf(" ovfl:");
       if(bc==53) printf(" par:");
       if(bc==54) printf(" ");
       if( ((word>>bc) & 1) > 0 ) printf("1"); else printf("0");

     }
     //   printf("%02x", *(unsigned char*)(ptr+7-i));
        printf("\n");

}

void dumpc(uint64_t word)
{
   int bc=0;
      printf("(");

      for (bc = 0; bc < 64; bc++)
     {
       if(bc==2) printf(") id:");
       if(bc==10) printf(" addr:");
       if(bc==18) printf(" data:");
```

```
        if(bc==26) printf(" zeros:");
        if(bc==53) printf(" parity:");
        if(bc==54) printf(" ");
        if( ((word>>bc) & 1) > 0 ) printf("1"); else printf("0");

      }
      //   printf("%02x", *(unsigned char*)(ptr+7-i));
         printf("\n");

}

void dump_decoded(uint64_t word)
{
  if((word & 0x0000000000000003) == 0) {printf("DATA "); dumpd(word); }
  if((word & 0x0000000000000003) == 2) {printf("CONF_W "); dumpc(word); }
  if((word & 0x0000000000000003) == 1) {printf("TEST "); dumpd(word); }
  if((word & 0x0000000000000003) == 3) {printf("CONF_R "); dumpc(word); }
}
```