MCM test setup 20210802

Updated 20211123 TLU/SRU
Updated 20211215 TLU software/pulsegenerator

Consists of (see also overview and picture below):

MCM: MultiChipModule with 8 SALTRO and one CPLD

Adapter: between MCM and flexcable

Flexcable: between adapter and Low Voltage card

Low Voltage card: supplies voltage to 5 MCM

5to1 card: support for 5 low voltage card. Used to control via I2C settings on the low voltage card and MCM.

I2C cable: between 5to1 card and low voltage card, local power to low voltage card.

USB to RS232 dongle: communication between 5to1 card and computer

SRU: readout and local control of MCM via low voltage card.

Shielded CAT6 RJ45 cable: between SRU and low voltage card.

Switch: network between SRU and computer.

Computer: laptop with programs to readout data and control of system.

RJ45 cables: 2 (data and control) between switch and SRU. 1 betweenn switch and computer.

Fan: to cool the MCM. The fan is needed unless one has switched off the ADC and preamplifier in the SALTROs.

Lattice programmer (old type): To program MCM CPLD.

Xilinx programmer: To program SRU Xilinx FPGA

Picprogammer: To program microchip processor on 5to1 card

Power supplies:

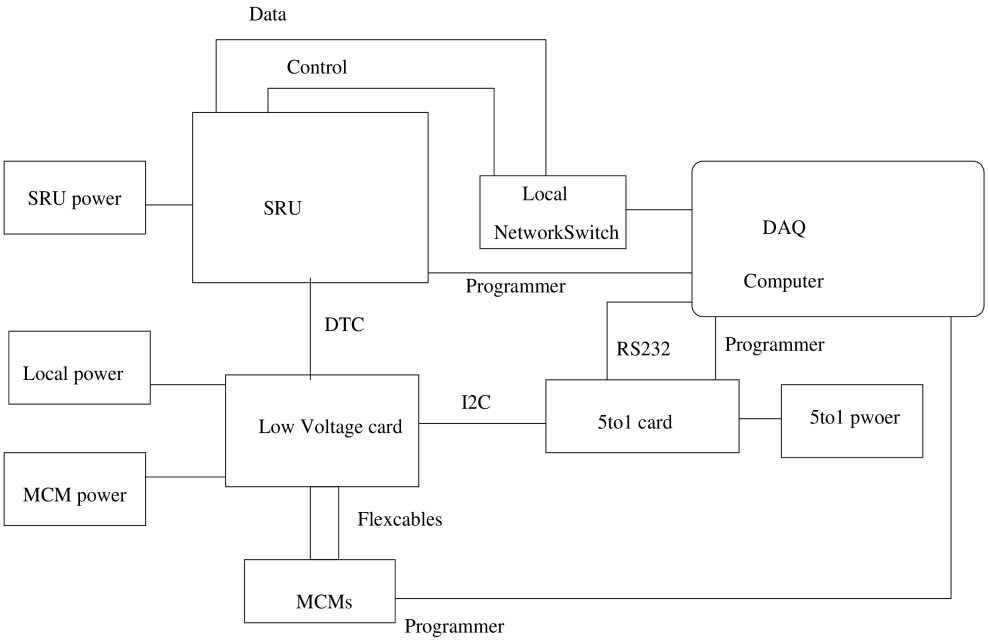
SRU power: special supply

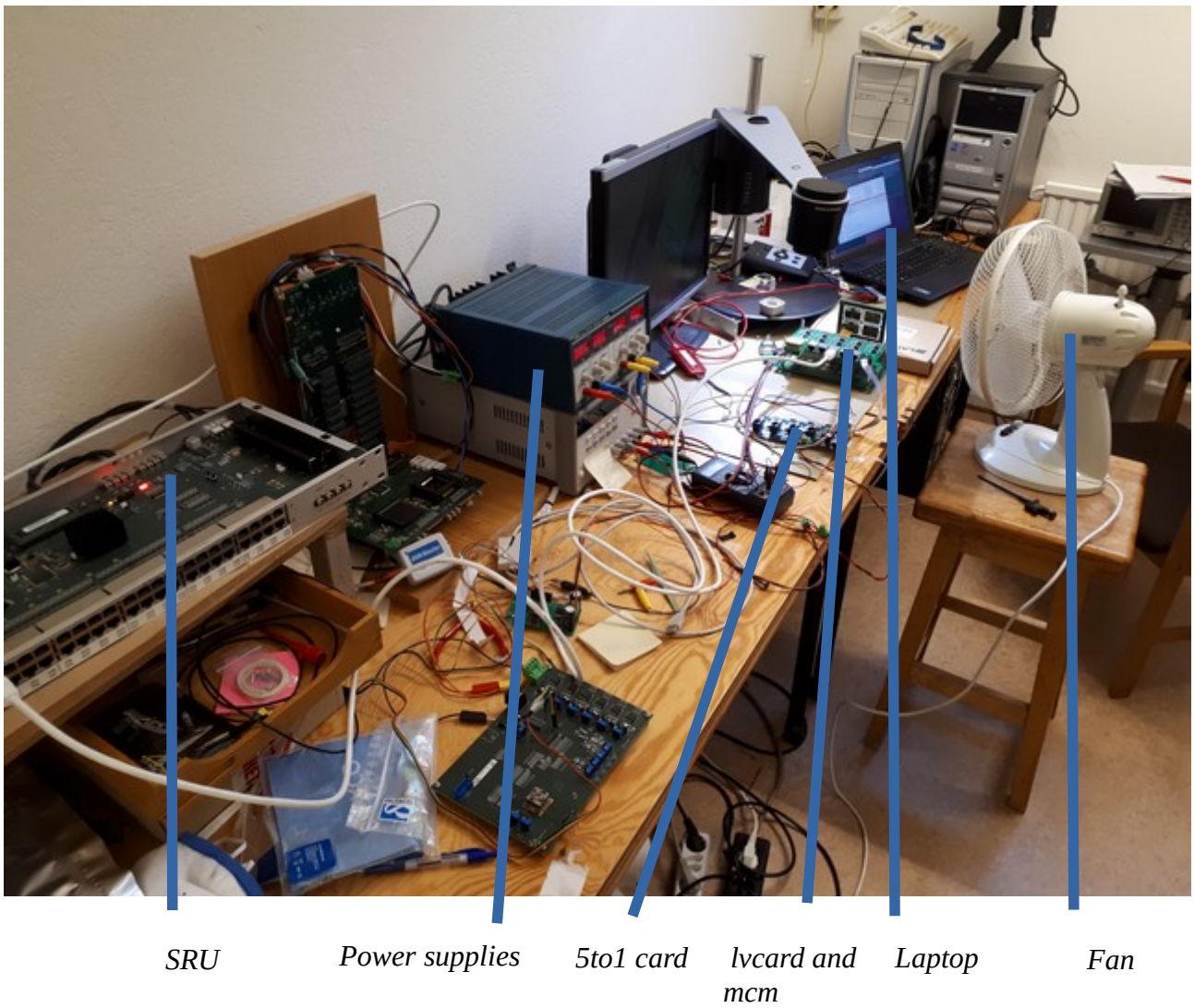
MCM power: 3.1V and 2.1V (each MCM takes >= 3.3A)

Local low voltage power: 6.5V (0.7A)

5to1 power: 5V (0.13A)

Overview of system





SRU

SRU (Scalable Readout Unit) is the unit that handles readout and control of the MCM.

It has two network interfaces on the back side, one for data transfer and one for trigger/control. Both of these are currently 1Gb/s. These are connected to a local netweork with a switch to which a laptop is connected (IP number 192.168.4.11) , the network addresses of the SRU are:

192.168.4.4	Trigger/Control, set when manufacturing the SRU.
192.168.4.5	Data, can be set by software via trigger/control line.

It has eight NIM input/outputs on the back side:

PX7-PX9	Undefined output
PX10	NIM signal issued a programmable time after a trigger
PX11-PX14	Undefined inputs

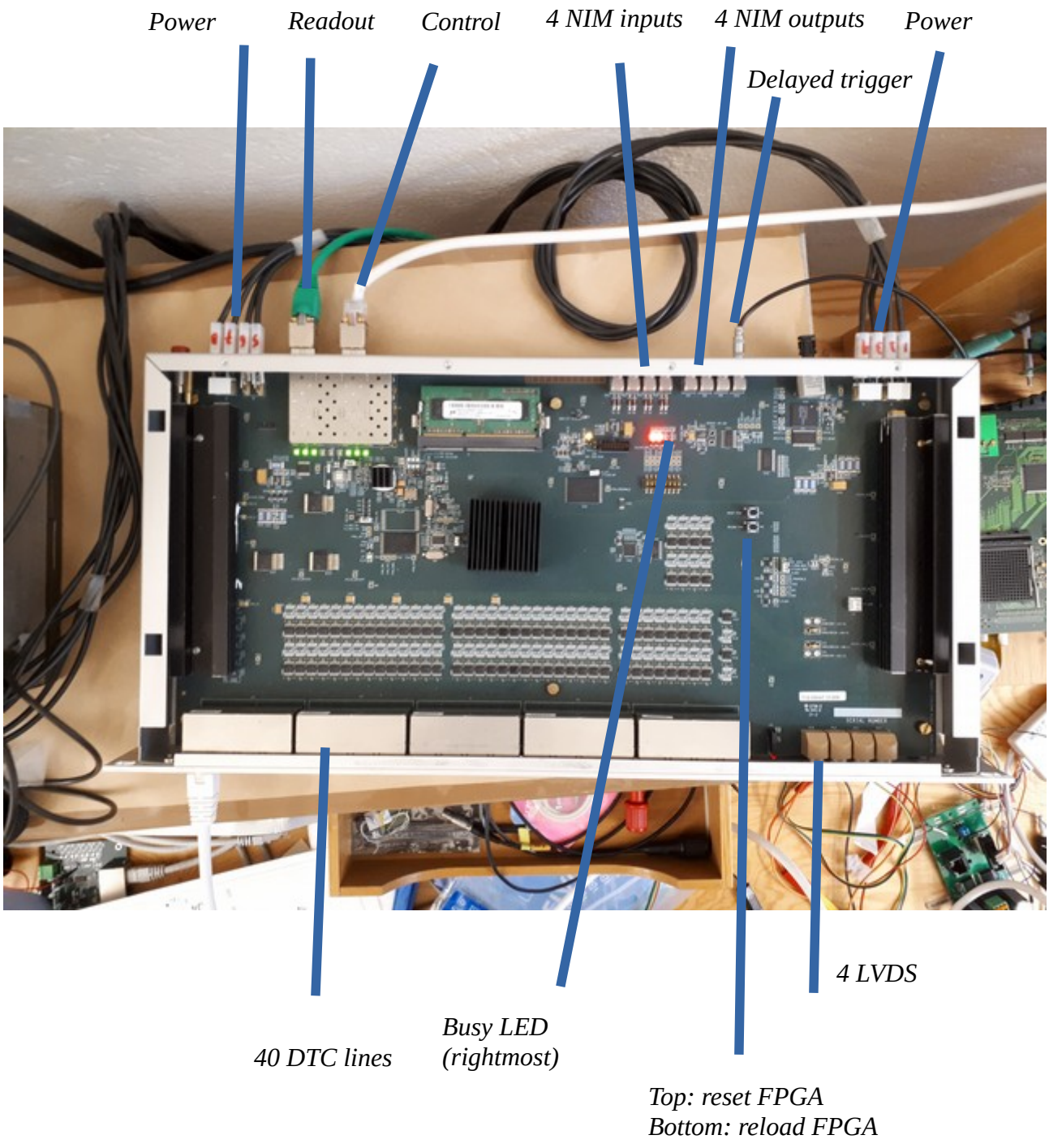
I has four LVDS input/outputs on the front panel:

PX15	Output
PX17	Input
PX16	Output
PX18	Input

It has 40 DataTriggerControl lines (DTC0-39) on the front, RJ45 connectors. We use DTC0 for communication with the MCM.

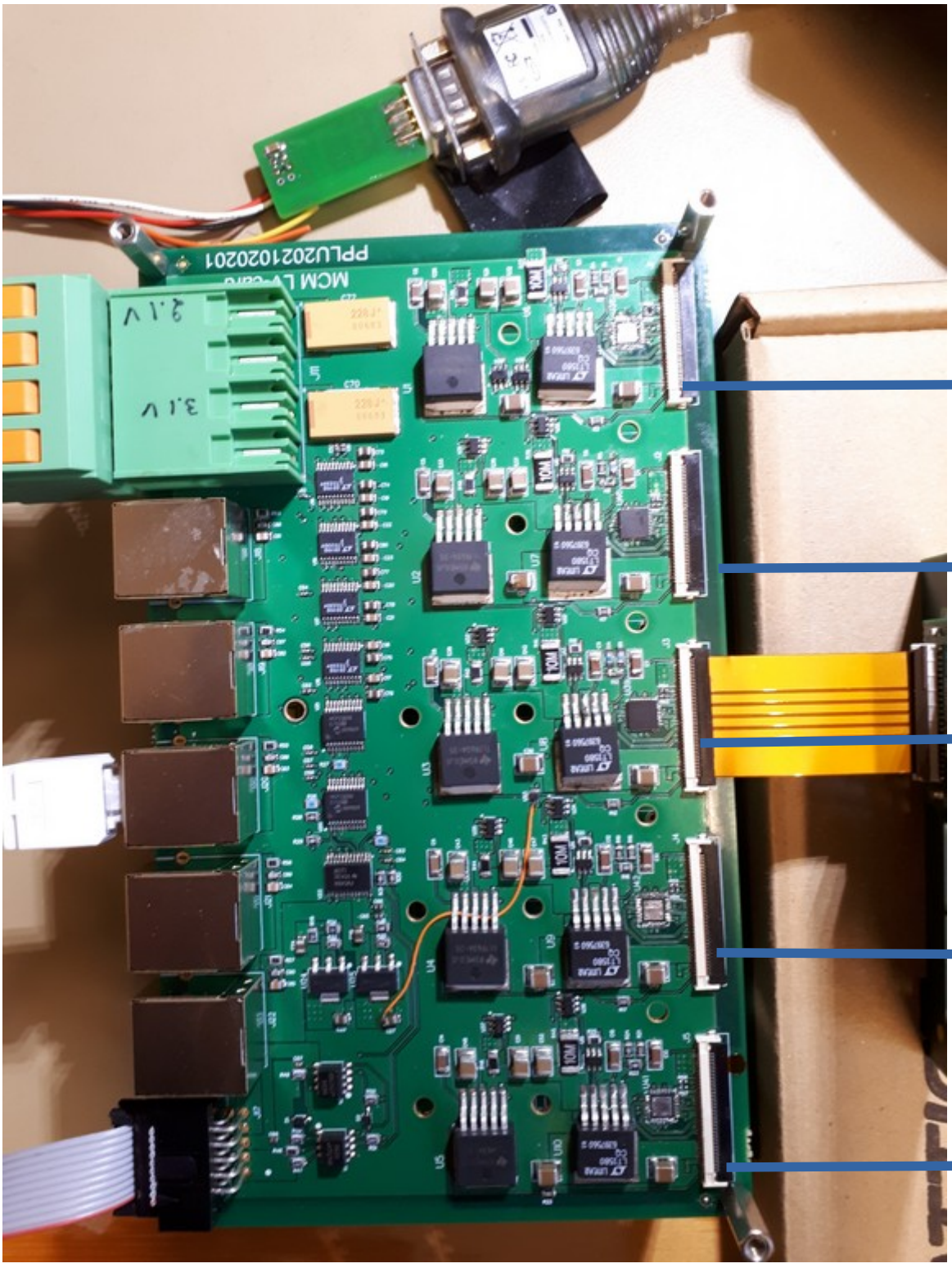
The FPGA (Xilinx Virtex) program can be reset by a button on the printed circuit board, another button is used to reload the FPGA program.

A LED indicates when the SRU is busy, e.g. when it after a trigger waits for data.


Power
Readout
Control
4 NIM inputs
4 NIM outputs
Power
Delayed trigger
4 LVDS
40 DTC lines
Busy LED
(rightmost)
Top: reset FPGA
Bottom: reload FPGA

Low Voltage Card – lvcard

The lvcard controls five MCM, labeled A-E as in the figure below.



The communication with the SRU:

is through five RJ45 CAT6 shielded connector, one for each MCM.

The Main Power:

(3.1V and 2.1V) supply the power for the 2.5V and 1.5V regulators needed for each of the MCMs.

The Local Power (>6.5V) give the power to the regulators (5V and 3.3V) for the local components on the lvcard:

4 ADCs: measures voltages and currents

1 temperature sensor: measures the temperature on the lvcard

5 LVDS buffer: SRU - MCM communication

1 I2C chain selector – selects local I2C or MCM A-E I2C chain

2 IO expanders – to enable/disable 1.5V and 2.5V regulators

1 high speed optocoupler – to isolate I2C on the lvcard from the incoming I2C line

T*he 5V from the 5to1 card give the power to:*

1 high speed optocoupler – to isolate I2C on the lvcard from the incoming I2C lines.

Limitations:

Only positions MCM B and C are fully mounted. The LVDS buffers for the communication SRU/MCM turned out to draw too much current for the voltage regulator. The LVDS buffers for position MCM A,D and E have been unmounted.

Power consumption

The total current of 3.1V + 2.1V with one MCM is 3.26A, this is dependent on the temperature on the MCM. 3.26A is with a temperature of 54 degrees as measured by the temperature sensor on the MCM.

The current for the 6.5V is 0.29A when two LVDS circuits are mounted. When all five LVDS circuits where mounted the current was 0.67A, which was more then the 3.3V regulator can handle.

The I2C chain from the 5to1 card:

There are six I2C chains, one on the lvcard and one for each MCM (A-E). The chain is selected by writing a bit pattern to the I2C chain selector on the lvcard, I2C address 0x70.

Local I2C chain:

Chain Selector	I2C ADDRESS 0x70
Bit 0	Enable I2C lvcard
Bit 1	Enable I2C MCM A
Bit 2	Enable I2C MCM B
Bit 3	Enable I2C MCM C
Bit 4	Enable I2C MCM D
Bit 5	Enable I2C MCM E

LVCARD I2C chain (chain 0) contains:

Temperature sensor	I2C address 0x48

ADC 1.5V Currents	I2C ADDRESS 0x08
Channel 0	2V1 input
Channel 1	Current 1.5V MCM A
Channel 2	Current 1.5V MCM B
Channel 3	Current 1.5V MCM C
Channel 4	Current 1.5V MCM D
Channel 5	Current 1.5V MCM E

ADC 2.5V Currents	I2C ADDRESS 0x09
Channel 0	3V1 input
Channel 1	Current 2.5V MCM A
Channel 2	Current 2.5V MCM B
Channel 3	Current 2.5V MCM C
Channel 4	Current 2.5V MCM D
Channel 5	Current 2.5V MCM E

ADC 1.5V Power	I2C ADDRESS 0x0A
Channel 0	Power 1.5V MCM A
Channel 1	Power 1.5V MCM B
Channel 2	Power 1.5V MCM C
Channel 3	Power 1.5V MCM D

Channel 4	Power 1.5V MCM E

ADC 2.5V Power	I2C ADDRESS 0x0B
Channel 0	Power 2.5V MCM A
Channel 1	Power 2.5V MCM B
Channel 2	Power 2.5V MCM C
Channel 3	Power 2.5V MCM D
Channel 4	Power 2.5V MCM E

IO EXPANDER 1.5V	I2C ADDRESS 0x20
Bit 0	Enable MCM A 1.5V regulator
Bit 1	Enable MCM B 1.5V regulator
Bit 2	Enable MCM C 1.5V regulator
Bit 3	Enable MCM D 1.5V regulator
Bit 4	Enable MCM E 1.5V regulator

IO EXPANDER 2.5V	I2C ADDRESS 0x21
Bit 0	Enable MCM A 2.5V regulator
Bit 1	Enable MCM B 2.5V regulator
Bit 2	Enable MCM C 2.5V regulator
Bit 3	Enable MCM D 2.5V regulator
Bit 4	Enable MCM E 2.5V regulattor

MCM card

The main components on the MCM card are:

8 SALTRO: digitizer of incoming analog signals

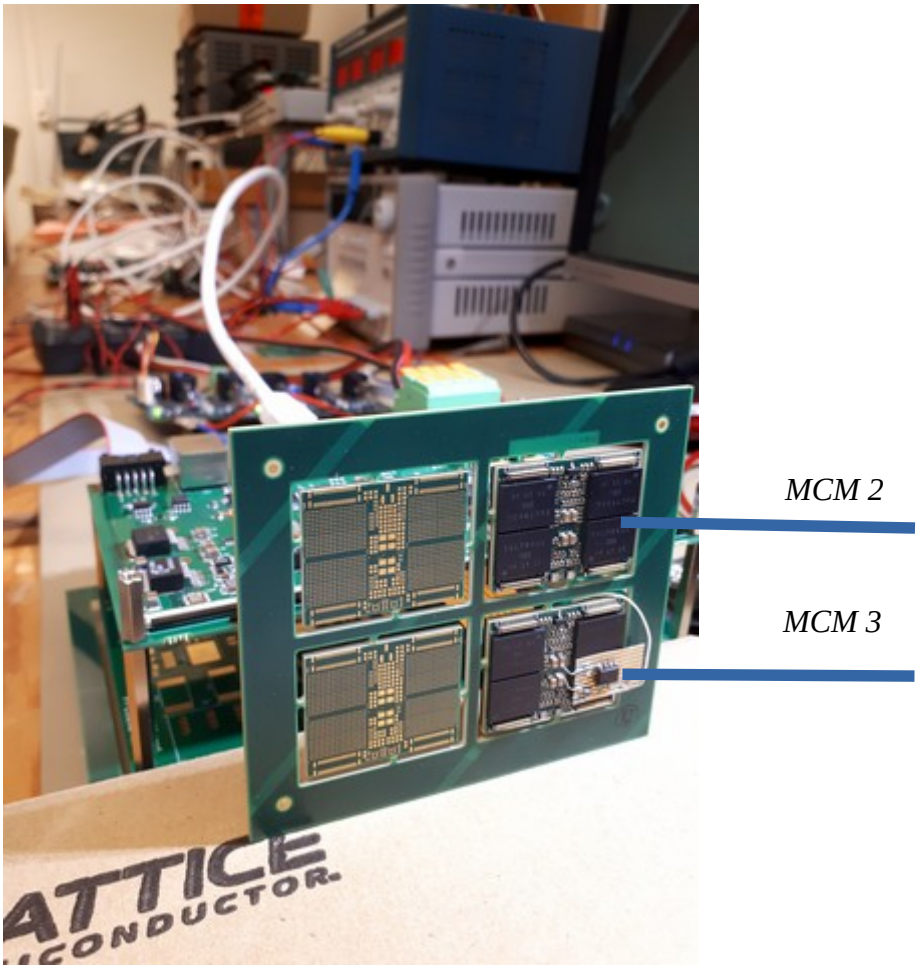
1 CPLD: communicates with SRU and all SALTRO

1 sensor: to measure temperature on the MCM

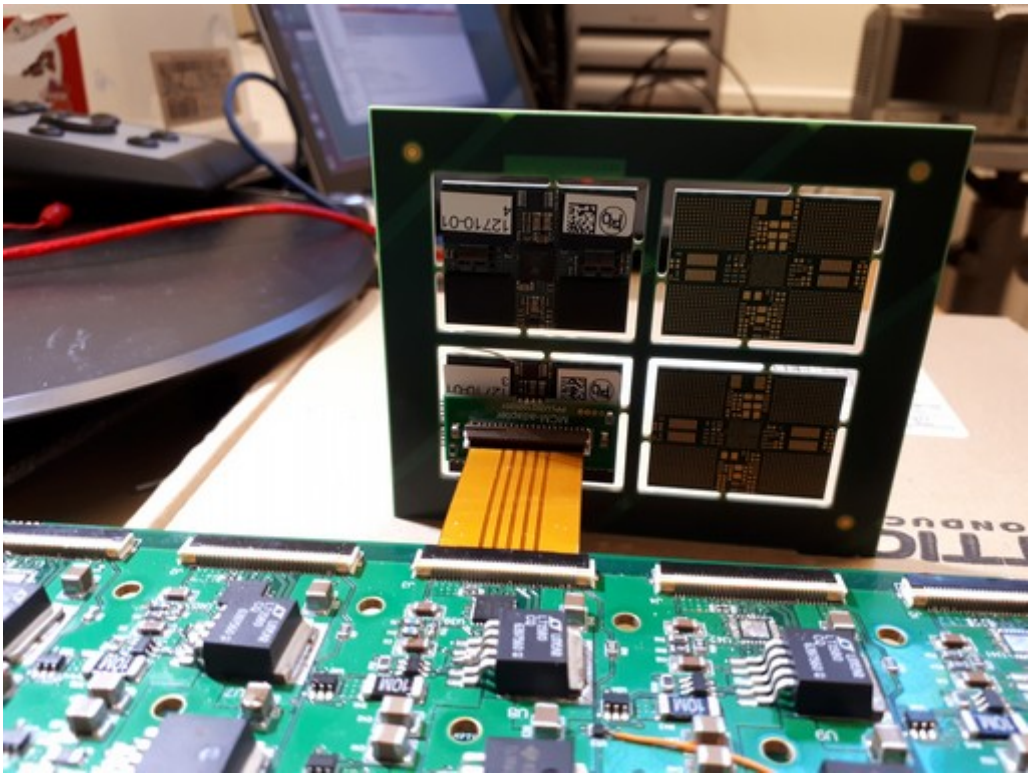
1 DAC: to set the four reference voltages and one decay bias, same on all SALTRO

1 transistor: bias control of all SALTRO (switch off/on the ADC on all SALTRO)

We have two MCM cards mounted, number 2 and 3 in the picture. We found that the opamplifiers for the reference voltages were oscillation with the original components. This showed up as an oscillation time sweep of the SALTRO data. MCM 3 was modified with another value of resistors in order to remove the oscillation. In this process one opamplifer were destroyed, and a small card with an opamplifier was attached on MCM 3.



The connection between the LVCARD and MCM is via a flexcable as in the picture below. Between the flexcableconnector and MCM connector is an adaptercard. This adapter do also contain JTAG for programming the CPLD.

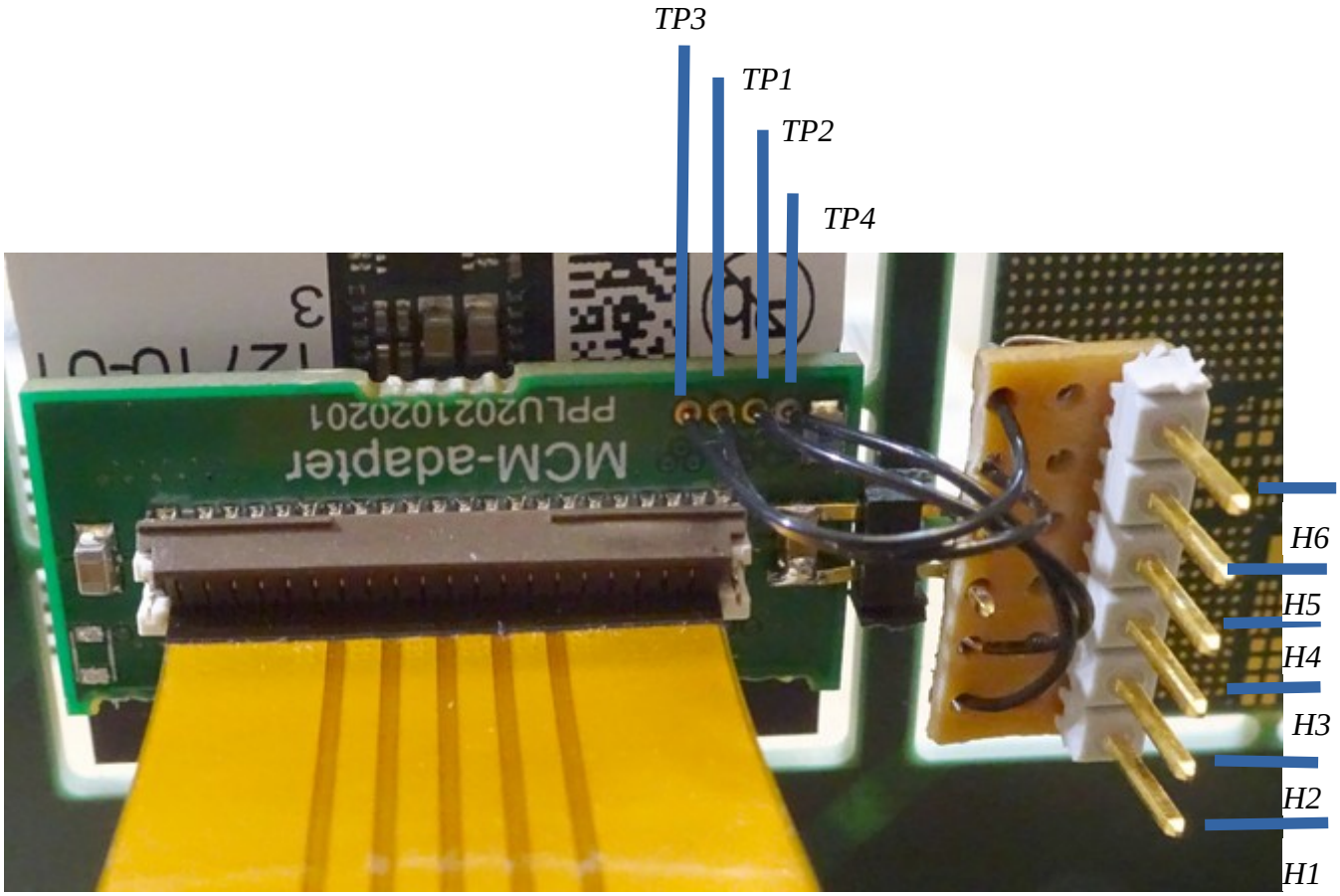


The temperature sensor and the DAC are controlled via I2C from the LVCARD.

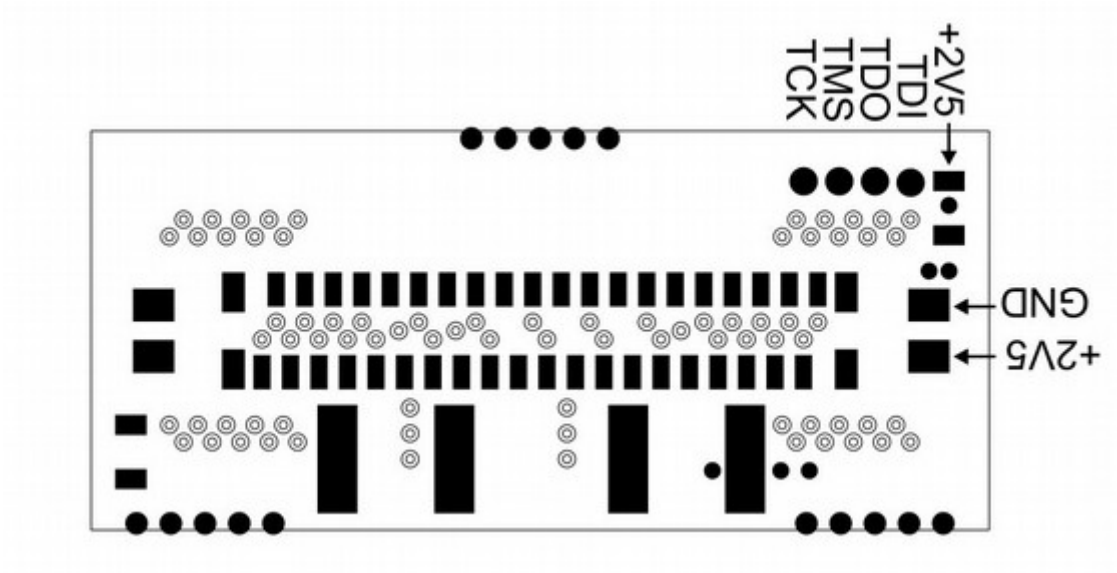
Temperature sensor	I2C address 0x48

DAC ref voltages	I2C address 0x0C
Channel A	Preamp BiasDecay
Channel B	RefP
Channel C	InCm/CmOut
Channel D	RefN

PROGRAMMING CPLD VIA ADAPTER1

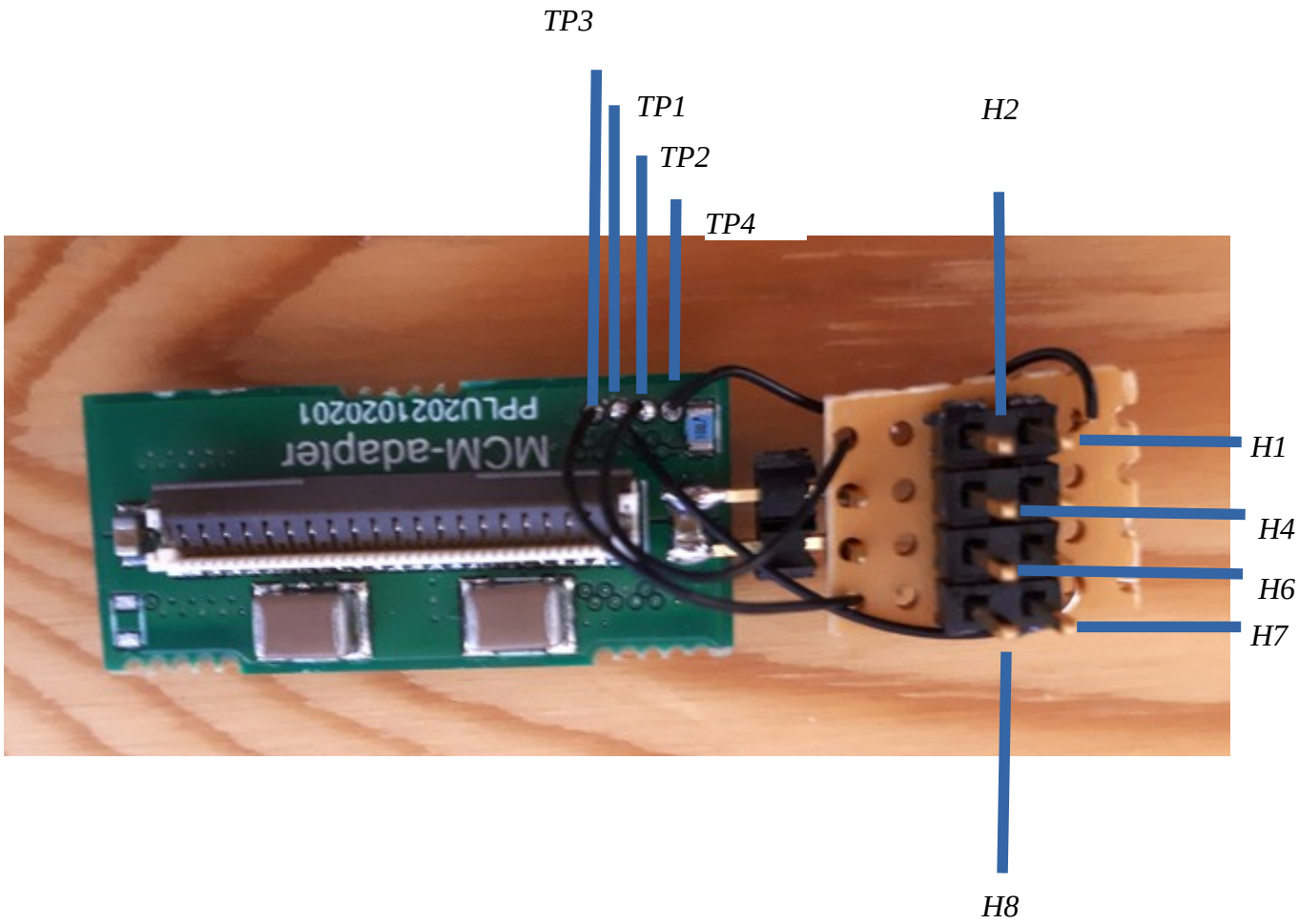


On the adapter between the flexcable and the MCM are four testpoints (TP1 – TP4) with the JTAG signals. From the testpoints are cables connected to a six pin (H1-H6) header. On the header pins are also ground and 2.5V, these are taken from a capacitor. The connection between JTAG signal, testpoint and header pins are given in the picture above, layout and the table below.

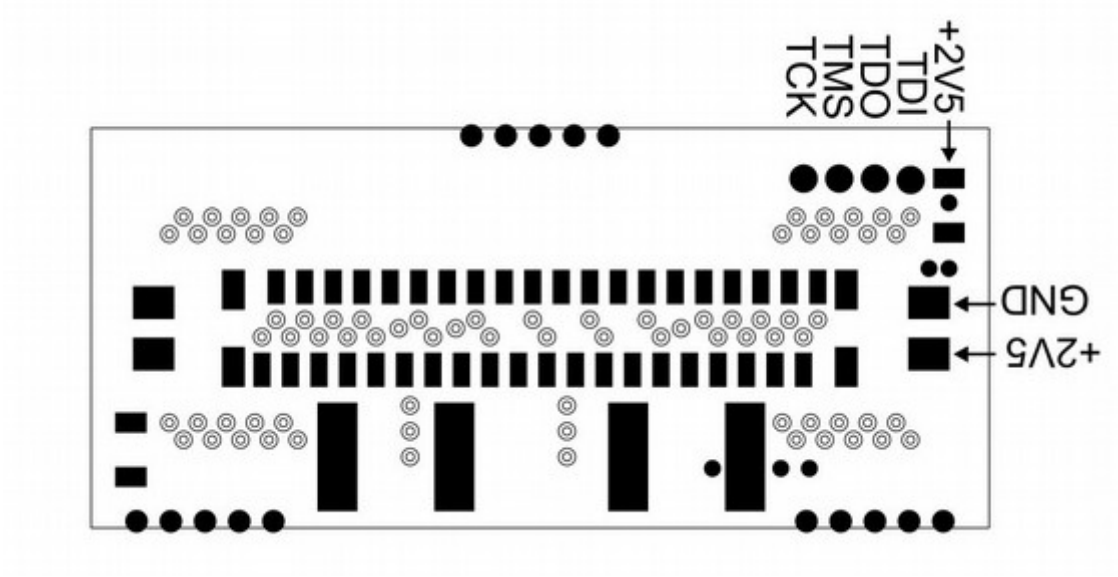


Header	*Testpoints*	*JTAG*	*Pin on J3 on MCM*
H1	TP4	TDI	9
H2	TP2	TDO	6
H3	Capacitor	2.5V	-
H4	Capacitor	GND	-
H5	TP1	TMS	8
H6	TP3	TCK	10

PROGRAMMING CPLD VIA ADAPTER2



On the adapter between the flexcable and the MCM are four testpoints (TP1 – TP4) with the JTAG signals. From the testpoints are cables connected to a six pin (H1-H6) header. On the header pins are also ground and 2.5V, these are taken from a capacitor. The connection between JTAG signal, testpoint and header pins are given in the picture above, layout and the table below.



Header	*Testpoints*	*JTAG*	*Pin on J3 on MCM*
H1	TP4	TDI	9
H2	TP2	TDO	6
H4	Capacitor	GND	-
H6	Capacitor	2.5V	-
H7	TP3	TMS	8
H8	TP1	TCK	10

5to1 card

One 5to1 card controls fve low voltage cards. For each low voltage card is one microprocessor that via I2C communicates with one low voltage card. Commands and data to/from the microprocessor is done in two ways:

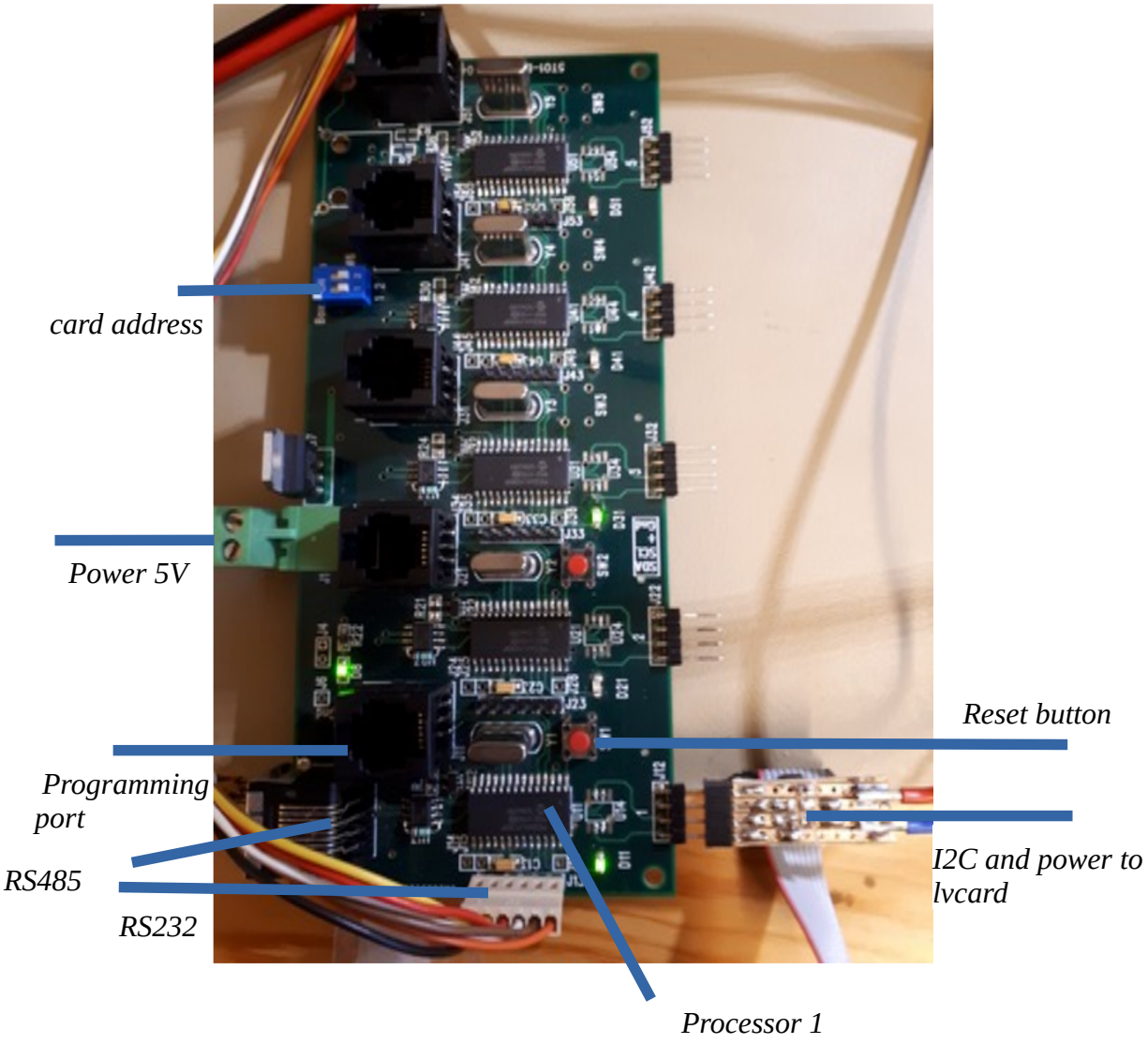
1. simple way via a console serial link (R232), one link per processor.

2. master control, a master box is connected via one RS485 link to a main controller. The master can handle four 5to1 cards. A switch on the 5to1 card gives the card address of the 5to1 card.

For the test setup is only option 1 used. Option 2 exist for an older version of the system. The microprocessor can monitor and control lvcards n standalone mode, but this also only exist for an older version and can not be used for the test setup.

Of the five processor is only one used, number 1 in the picture below.

The card needs 5V power, the current is then 0.10-0.12A.



With the current version of the lvcard only a few commands exist, and then via the serial RS232 interface. There are no default settings loaded after power on, so all has to be initialized by the operator after power on. Commands are case sensitive. There are more commands and functionality but these do not work work with this lvcard.

I2C chain select commands (NOTE: TCA9548A chip on lvcard):

SCI	Initialize chain selector control register
SCSn	Select chain n in control register 0=none, 1=LV card, 2-6=MCM A-E, 7-8=not used
SCR	Read switch control register

I*2C temperature commands (requires chain lvcard/MCM is set. NOTE: TMP112 chip on lvcard/MCM):*

TRT	Tmp101/112 read temperature (temperature is output in value and degrees)
TRH	Tmp101/112 read tHigh register
TRL	Tmp101/112 read tLow register
TRC	Tmp101/112 read config
TWCxx	Tmp101/112 write config with hex pattern xx

I2C ioexpander regulator commands (requires chain lvcard is set. NOTE: MCP23008 chip on lvcard):

ROIn	Initialize regulators 1: 1.5V 2: 2.5V 3: both (all set to off)
ROAxx	set all regulators (1.5V/2.5V) to hex value bit=0 off bit=1 on bit 0: MCMA, bit 1 MCMB, bit 2: MCMC, bit 3: MCMD, bit 4: MCME
RODn	Dump rall egulator registers 1: 1.5V 2: 2.5V 3: both

I2C ADC commands (requires chain lvcard is set. NOTE: LTC2309 chips on lvcard):

ARAn read and dump all channels in adc n (1-4)

n=1 :: channel 1: input 2V1, channel 2-6: current 1V5 MCMA-E
n=2 :: channel 1: input 3V1, channel 2-6: current 2V5 MCMA-E
n=3 :: channel 1-5: output 1V5 MCMA-E
n=4 :: channel 1-5: output 2V5 MCMA-E

I2C DAC commands (exist only in MCM chains, requires chain is set. NOTE: AD5315 chip): Each DAC unit corresponds to 2mV output.

DIN Write default values to DAC channels A-D
Default values are hardcoded in the program as:
Preamp BiasDecay: 0250, RefP: 0625, inCm/CmOut: 0375, RefN: 0125.
These corresponds to 0.5V, 1.25V, 0.75V, 0.25V which are the nominal values given in the SALTRO documentation.

DWRn1234	Write DAC decimal value 1234 to channel n (0-3) channel 0: Preamp BiasDecay 1: RefP, 2: InCm/CmOut, 3: RefN
DRDn	Read DAC value channel n (0-3) channel 0: Preamp BiasDecay 1: RefP, 2: InCm/CmOut, 3: RefN

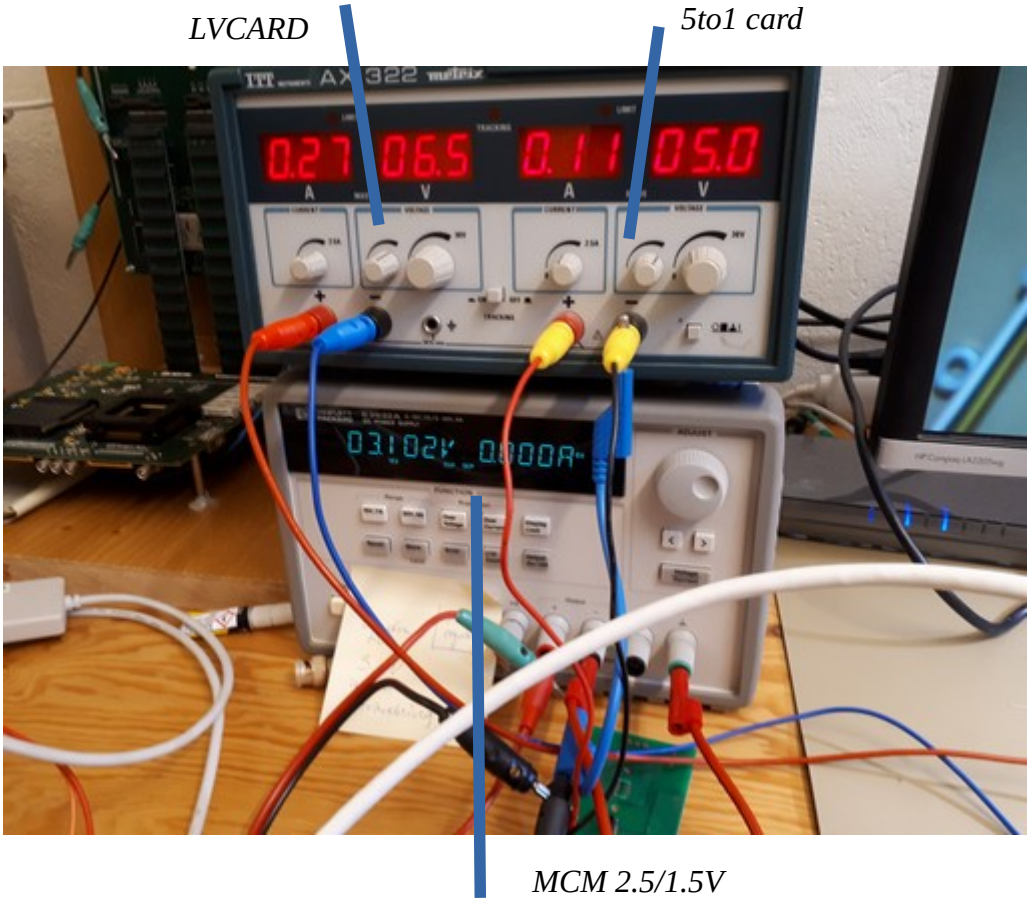
Power on

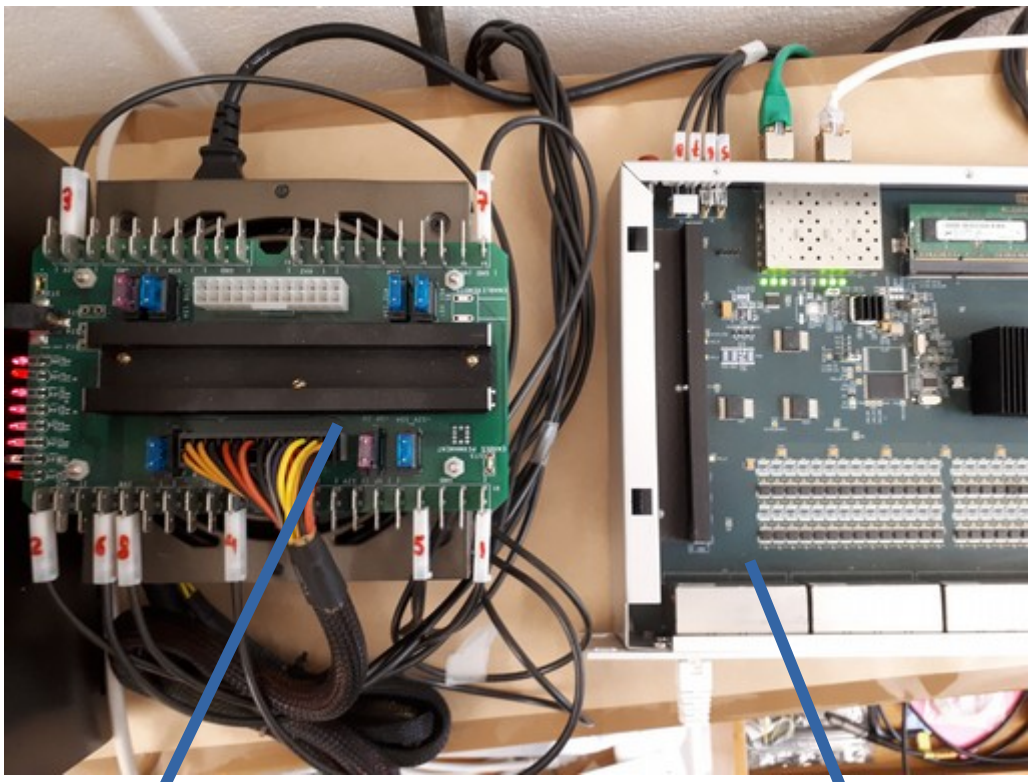
1. 5to1 card: should have 5V and the current is 0.10 - 0.12A

2. MCM – both 2.5V and 1.5V are connected to the same power supply. Set to 3.1V, when all is powered up with one MCM the current is ca 3.25A at 54 degrees.

3. LVCARD needs a separate power of >= 6.5V for the local components. With all LVDS buffers mounted at consumed 0.67A, too much for the regulator. We have removed three of the buffers (MCMA, MCMD, MCME). After this the current is 0.27A.

4. SRU. Turn on the special SRU power supply.





SRU power supply

SRU

Recommended power on in the order:

1. SRU

2. 5to1 card and local lvcard

Initailize in putty window the lvcard I2C settings, as also described further below:

Initilize the I2C chain selector
>SCI
Select chain 1 (li.e. the vcard)
>SCS1
Initialize ioexpanders for the regulators
>ROI3
Set all regulators off (to be don ebefore power on 3.1V supply):
>ROA00

3. Power on 3.1V supply

Power on the regulators
>ROA04

4. Initialize the reference voltages on MCM
>SCS4 (if MCMC)
>DIN

Program startup

From a terminal window type:

ilc-start.sh

This will open a number of windows/tabs and programs. The locations refer to where they are placed on the screen. It is a script located ~/bin/.

TODO: PICTURE

Upper left is a window in the preselected catalogue ilc/ilcdaq/ilcserver-v10.2/

This is where the readout program is started/stopped.

Start the readout program:

./ilcserver

To stop it:

Ct**rl-C**.

When the system hangs must the program be stopped and restarted.

This program loads parameters to and initialize the SRU and the communication with the CPLD. By default is it using the DTC line 0. It reads the SRU parameters from the file:

wdir/ilcdaq/config.v10-1/srupup-0.cfg

This file needs to be edited if the initialization is to be changed.

The program also write default parameters to the CPLD and SALTRO registers. There are various configuration files in the same directory for this. The file ‘sru-0.cfg’ contains information which SALTRO and channels that are initialized. The file ‘global-altro.physics’ gives the SALTRO parameters. The testscript described below load its own parameters.

If the system crash it might be that one can not start the ilserver again, this because the system thinks the it already is running, In this case remove the file:
~/wdir/ilcdaq/log-v10.1/ilcserver.pid

Source codes:
The source code of the ilcserver is in the directory: ~/ilc/ilcdaq/ilcserver-v10.2/
The program ilcserver forks another program, ilcrorc-readout.
The binary, source code and the compilation is done in the directory: ~/ilc/ilcdaq/readout-v10.1/

Upper right is a window with two tabs, both tabs in the preseleted catalogue ilc/ilcdaq/test-v10.0/

This is where testprograms are executed. Use one tab for testing if the there is a response from the MCM. The other tab is for running the testscripts.

Lower left is a window with two tabs. One with the preselected catalogue ilc/ilcdaq/gui-v10.0/. This is the place where the program to initialize the DAQ is executed:

j**ava DaqPanels**

This must be done each time the **ilcserver** above has been started and before running any testscript.

In the GUI that appears do:

1. Power on

2. PCA load

3. DAQ start

4. Exit

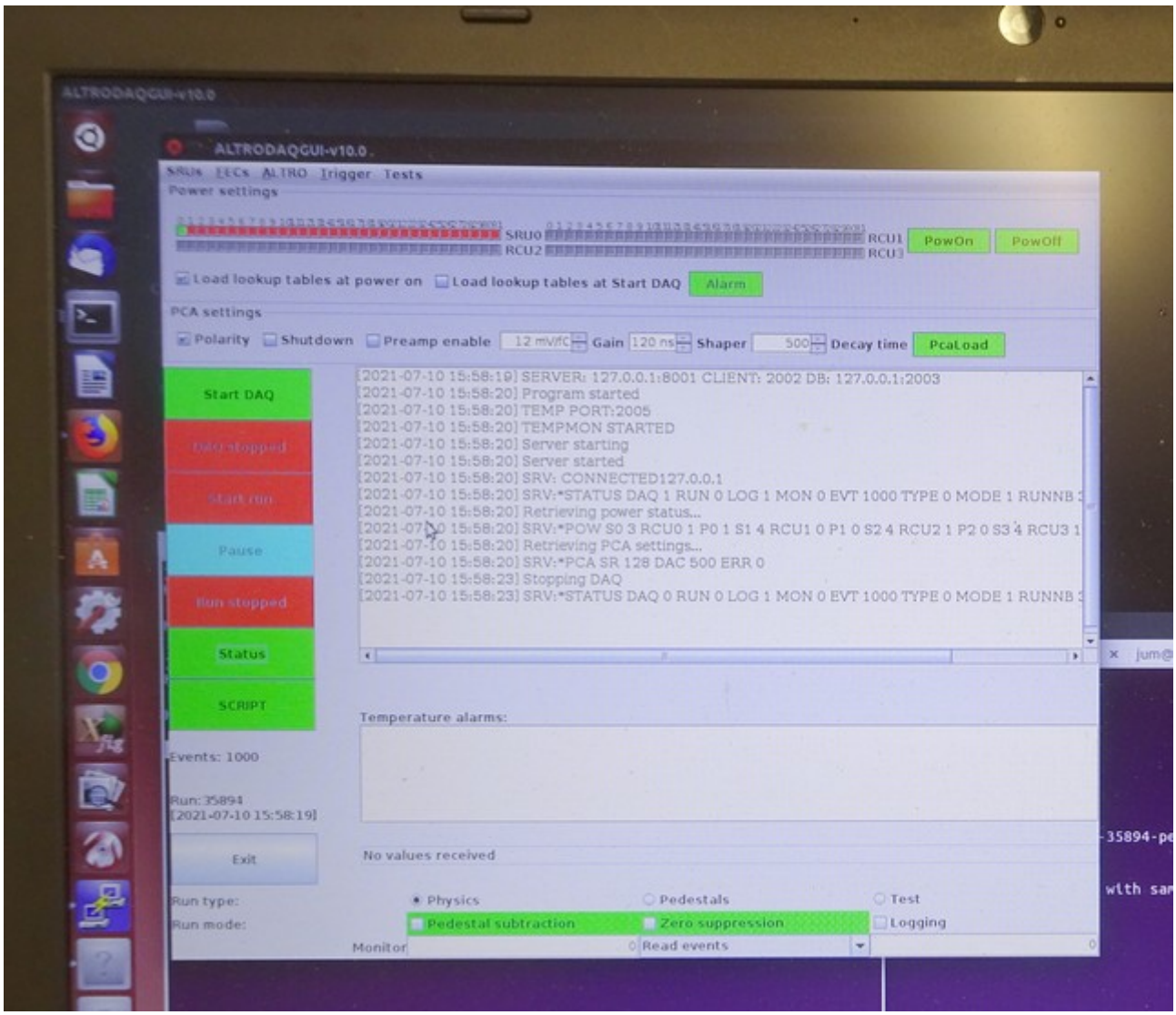
If the STATUS box appear red after any of 1-3, then one need to check why, reset or restart the SRU. More about this below. If the DAQ needs to be stopped and started again do:

1. Stop DAQ

2. Start DAQ

3. Exit

The GUI is shown in picture below.



Source code av java program: same directory.

The other tab is with the preselected catalogue wdir/ilcdaq/config.v10-1/. This is used to edit DAQ configuration files. Normally not needed, expert use. Needs to be done when changing from using peanplifier input as data or a memory in the SALTRO.

The SRU initialization is in the file:
srupup-0.cfg

Lower right is a window with three tabs. Tab 1 is for running **root**, which is started automatically in this tab. Tab 2 is for editing root scripts (need rudimentory knowhow of root C++ to do that), tab 3 to create the directory for testresults (normally the current date and the subdirectory root in that directory) and to copy files to this directory ***if test data is to be saved***. E.g.

mkdir 20201029

mkdir 20201029/root

Put log outputs and descriptions in the file readme.txt.

Copy pdf and root files with these commands:

```
*** COPY COMMANDS Examples ***
ls -l *.pdf | grep "okt 29" |  awk '{print $9}' | xargs cp -a -t 20201029/
ls -l *.root | grep "okt 29" |  awk '{print $9}' | xargs cp -a -t 20201029/root
```

Replace "okt 29" and 20201029 with appropriate date.

Putty window: Putty is used to communicate with the 5to1 card. Putty is automatically started. If starting it by hand then do:
1. Select 5to1
2. Load it
3. Open it

Before doing it must the 5to1 card be powered up and has been reset as described above.

Putty starts a window for writing commands to the 5to1 processor. The commands currently needed is given below.

All commands should reply:
S00009E0000

Initialize and select/read the I2C chain on LVCARD
Initialize select chain register, no chain selected.

>SCI
S00009E0000
Select I2C chain
>SCSn n: 0=no chain, 1=lvcard 2-6=MCMA-E

To select lvcard I2C chain:
>SCS1
S00009E0000
To select MCM C chain (=currently where the MCM is connected)
>SCS4
S00009E0000
Read back chain select control register:
>SCR
S00009E0000
0004
The first line returned is I2C status. The second line is the absolute value in hex read from the selected chain register

Poweron/off the MCM

Initialize the ioexpanders on the lvcard to be outputs and all outputs off:
>SCS1

S00009E0000
>ROI3
S00009E0000
S00009E0000

Dump all registers in both ioexpanders
>ROD3
0000
0000
0000
0000
0000
0020
0000
0000
0000
0000
0000
S00009E00000000
0000
0000
0000
0000
0020
0000
0000
0000
0000
0000
S00009E0000

Power on MCM C
>ROA04
S00009E0000

Read the currents and power for MCMs

The currents and voltages are measured be ADCs which give the values in ADCs units. The commands to read an is ARAn, where n=1-4. Output is one line per ADC channel, the first number is the value read from the ADC in hex, the second n decimal which represents the input to the channel in mV. The channel input is described in the part about the 5to1 card. In the commands below is MCM C used (fourth line in output). Two last channels are not used.

>ARA1
BCD0 03021
0000 00000
0000 00000

6200 01568
0080 00008
0030 00003
69E0 01694
6DF0 01759

>ARA2
C000 03072
0000 00000
0000 00000
0590 00089
0010 00001
0010 00001
6A50 01701
6DE0 01758

The ADC values are directly measured in mV. For 1.5V (ARA1) is the current measured with a 10 milliohm resistor and an amplification of 50 before the ADC:
1568 = 50*10* I(1.5V) =>
I(1.5V) = 3.136A

For 2.5V (ARA2) is the current measured with a 20 milliohm resistor and an amplification of 50 before the ADC:
89 = 50*20* I(2.5v) =>
I(2.5V) = 0.089A

The power supply showed ca 3.22A in this case.

The output voltages from the regulators are read with the ADCs for 1.5V and 2.5V respectively (the last three channels are not used):

>ARA3
0000 00000
0000 00000
5EA0 01514
0000 00000
0000 00000
6A80 01704
6AD0 01709
6EB0 01771

>ARA4
0740 00116
0090 00009
D30 02515
0AE0 00174

11C0 00284
6C80 01736
6B90 01721
6E80 01768

Again MCM C is used, i.e. channel 3. As seen this shows the correct voltages in mV (1.5V resp 2.5V).

Read the LVCARD temperature

>SCS1
S00009E0000
>TRT
S00009E0000
1F10 031.00

The first line returned is I2C status.
The second line is first the absolute value in hex read from the sensor, the second the temperature in degrees Celcius.

Read the MCM temperature

>SCS4
S00009E0000
>TRT
S00009E0000
3240 050.25

The first line returned is I2C status.
The second line is first the absolute value in hex read from the sensor, the second the temperature in degrees Celcius.

To initialize/change the DAC settings on the MCM.
This is only needed after each power on of the MCM.

Initilaize the DAC with default values (0075=150mV refN, 375=750mV cmin/cmout, 675=1350 mV refP), at the prompt '>' type (if there is no prompt hit CR):

>DIN

The reply should be (one line per channel):
S00009E0000
S00009E0000
S00009E0000
S00009E0000

When running with preamplifier gain = 3 (27mV/fC) you need to change refN and refP, else the SALTRO baseline will be outside the ADC scale.

To change refN to 0.150V (=DAC channel 3):
>DWR30075

To change refP to 1.350V(=DAC channel 1):
>DWR10675

The replies should be:
S00009E0000

The argument is a four character decimal value, each step corresponds to 2 mV of the DAC output.

Read the settings from the DAC, returned is the hex value in the register:
>DRD0
S00009E0000
23E8
>DRD1
S00009E0000
29C4
>DRD2
S00009E0000
25DC
>DRD3
S00009E0000
21F4

MCMTESTS

When everything is powered up and initialized one can try to communicate with the SRU, MCM and startup of DAQ.

Source code of scripts and programs: ~/ilc/ilcdaq/test-v10.0/

SRU REGISTERS

In the window for test communication with the SRU/MCM do:

To dump all sru registers used.
./srudumpregs

The output is a list of SRU register and values in hex, status is the status of the read command:

jum@knatte:~/ilc/ilcdaq/test-v10.0$ ./srudumpregs

Parameters: REG 0000 LOOP 1 VER 0 WRITE 0
SRU_REG_SN (1) value 0 status 0
SRU_REG_IP (2) value C0A80404 status 0
SRU_REG_FMVER (3) value 20160719 status 0
SRU_REG_SRUST (4) value 38 status 0

SRU_REG_DTCERRFLAG1 (5) value 0 status 0
SRU_REG_FPGA_MON (7) value A05D status 0
SRU_REG_READOUT_IP (8) value C0A8040B status 0
SRU_REG_CFG (20) value 1 status 0
SRU_REG_DTC_CLKEN1 (21) value 1 status 0
SRU_REG_DTC_CLKEN2 (22) value 0 status 0
SRU_REG_DTCRDOMASK1 (23) value FFFFE status 0
SRU_REG_DTCRDOMASK2 (24) value FFFFF status 0
SRU_REG_SALTRO_CFG1 (27) value 0 status 0
SRU_REG_SALTRO_CFG2 (28) value 0 status 0
SRU_REG_TRIG_L1_TW (29) value 13880078 status 0
SRU_REG_TRIG_L2_TW (2A) value 1F4000C8 status 0
SRU_REG_PTRIG_CFG value (50) 4E20 status 0
SRU_REG_TEST_TRIG_CFG0 (51) value 119400C8 status 0
SRU_REG_TEST_TRIG_CFG1 (52) value D1770 status 0
SRU_REG_TEST_PULSE_CFG (53) value 4C800C8 status 0
SRU_REG_TEST_PULSE_CFG: START 200 WIDTH 200 EN 1
SRU_REG_TX_L0L1_CNT (64) value 0 status 0
SRU_REG_TX_L2A_CNT (65) value 0 status 0
SRU_REG_TX_L2R_CNT (66) value 0 status 0
SRU_REG_TRIGERRCNT (67) value 0 status 0

INIT SRU DATA PATH

When the SRU is starded does it not know where to send over the data network. It needs to be told by sending a command to the UDP data interface in the SRU code.

In the window/tab for test communication with the SRU/MCM do:
./sru-init –init

This will tell if the SRU data transfer line is active. The output should be:

TO BE ADDED

Retry until correct reply. If no luck, try with resetting SRU. Should be done after each power on/reset av SRU.

TEST COMMUNICATION

In the upper left window start the ilcserver, if not already done. This is needed before any commands should sent to the MCM, since it initializes the SRU registers.

To test the communication with MCM do in the window/tab for test of communication:

./mcmdumpregs –fec 0

There should come a dump of the MCM registers if all is fine e.g.

TO BE ADDED

if there comes any error -3 as a reply, then try:

./srureset.sh

this partially resets the SRU and resyncs the data line with the MCM. Try again the mcmdumpregs. If there is a run going on then the ilcserver/daq needs to be restarted.

If this not helps:

stop the ilcserver (with Ctrl-C) and reset the SRU FPGA. Repeat sru-init and start the ilcserver/daq again.

SALTRO GLOBAL RESET

Before doing any access to the SALTRO after a power on do they need a global reset:

./grst.sh 0 1

Where the first argument is:
0 = preamplifier on
1 = preamplifier shutdown

Second argument:
0 = ADCs is (biasctrl is off)
1 = ADCs on (biasctrl is on)

After a global reset always stop and restart the ilcserver. (MAY CHANGE THIS PROCEDURE).

For the currents for the differents combinations, see next section (SALTRO ON/OFF)

SALTRO ON/OFF

If you do not want to reset the SALTROs, but only switch on/off preamplifer/ADCs use:

./biasctrl.sh <shutdown> <bias>

where

<shutdown> is:
0 = preamplifier on
1 = preamplifier shutdown

<bias> is:
0 = ADCs off (biasctrl is off)
1 = ADCs on (biasctrl is on)

The currents for different combinations is roughly:

shutdown	biasctrl	Current
0 on	0 off	1.4A
0 on	1 on	3.2A
1 off	0 off	0.4A

1 off	1 on	2.4A

DAQ START

To allow for run and data taking start/exit the DAQ via DaqPanels as described in "*Lower left* is a window" section above. If status goes red in the GUI, you have to try the options in the section "TEST COMMUNICATION" until it all works.

TEST RUNS

Go to the upper right windowtab used for runs. One simple action is to take run for pedestals of one chip with 135 samples. So this is given here as the example. Other tests needs some more editing of testscripts to be used by nonexpert.

./mkpedestal-shift.sh <mcm>00<chiponmcm> <gains> <shaper> <samples> <channels>

where the arguments are:

<mcm> mcm number is 2 or 3 with currently mounted MCMs.
<chiponmcm> 0-7.
<gain> 0 = 12mV/fC, 1=next highest 19 mV/fC, 2=next lowest 15 mV/fC, 3=27mV/fC
<shaper> 0 = 120 ns, 4=90 ns, 6= 60 ns, 7 = 120 ns
<samples> number of time samples in decimal (0-1000)
<channels> bit pattern in hex of which channels NOT TO READ.

The strange bit ordering of gain/shaper is due to a reordering of bits between MCM and SALTRO, will do the reorder in a program when all bits is tested to work as expected.

Example:

./mkpedestal-shift.sh 2001 0 0 135 0

will use chip 1 with lowest gain and longest shaping, 135 samples and all channels.

This script sets the number of samples in all chips to 135, activates the selected chip for readout in the CPLD, and takes a default of 500 events if it all works. To change the number of events you need to edit the script.

The raw data file, if data logging is used, is converted to root ntuples by the test script.

The data file written is:
wdir/ilcdaq/data/readout-<runnumber>_0.dat

The root ntuple file is:
wdir/ilcdaq/data/readout-<runnumber>_0.root

Now and then should old files be removed, else will the disk be full.

In the window with the ilcserver running should one see a line for each event taken, the test script will output a line for each hundred event. The script enables by default data logging in a file. At the end of a run there is a crash dump of the program that converts the data to root ntuples, ignore that dump. At the end is a line output with a root command to execute for pedestal plots. This is to be executed in the windowtab where root is running.

E.g for the example above:

```
.x plotallped.cxx(2001,34643,"gain0-shaper0")
```
(34643 is the runnumber, which is of course different for each run).

It will generate an output pdf-file:

```
ch-all-chip2001-34643-ped-gain0-shaper0.pdf
```

To see the time sweep of all channels execute (again this example) the root command:

```
.x plotallchchip0x.C(10,2001,"34643-gain0-shaper0","onerun.txt")
```
10 is the event number to do it for. The file onerun.txt must contain the root file to use i.e.

readout-<runnumber>_0.root

To edit select the windowtab in the “root” window with a a normal terminal prompt. The script updates this file, but you may want to use another run number than the last run.

The output pdf-file will be:

```
chip0x-all-chip2001-evt10-34643-gain0-shaper0.pdf
```

Example code of mkpedestal-shift.sh:

```
#!/bin/bash
#args = <mcm>00<chipnb> <gain> <shaper> <samples in decimal> <channels in hex>
mydir=/home/jum/wdir/ilcdaq/root/files
myroot=/home/jum/wdir/ilcdaq/root/
mydata=/home/jum/wdir/ilcdaq/data/
events=500
mymcmchip=$1
mychip=$((mymcmchip & 0x7))
samples=$4
channels=$5
shift1=`printf '%x' $samples`
shift2=`printf '%x' $((samples + 0x400))`
shift3=`printf '%x' $((samples + 0x800))`
shift4=`printf '%x' $((samples + 0xC00))`

./mcmwrreg --fec 0 --register 10 --write --value FFFF
./mcmwrreg --fec 0 --register 11 --write --value FFFF
./mcmwrreg --fec 0 --register 12 --write --value FFFF
./mcmwrreg --fec 0 --register 13 --write --value FFFF
./mcmwrreg --fec 0 --register 14 --write --value FFFF
./mcmwrreg --fec 0 --register 15 --write --value FFFF
./mcmwrreg --fec 0 --register 16 --write --value FFFF
./mcmwrreg --fec 0 --register 17 --write --value FFFF
let mcmreg=${mychip}+10
#./mcmwrreg --fec 0 --register $mcmreg --write --value FFFC
./mcmwrreg --fec 0 --register $mcmreg --write --value $channels
#
# read partly sample memory (135 samples = 0x87) - global register
```

```
#
echo "" >> $mydir/readme.txt
date >> $mydir/readme.txt
echo "=====" >> $mydir/readme.txt
echo "MKPEDESTAL CHIP $1 GAIN $2 SHAPER $3" >> $mydir/readme.txt
echo "CHIP: " $mychip >> $mydir/readme.txt
echo "SAMPLES: " $samples  >> $mydir/readme.txt
echo "CHANNELS: " $channels >> $mydir/readme.txt
echo "SHIFT: " $shift1,$shift2,$shift3,$shift4 >> $mydir/readme.txt
# bit 0 - 9, acq_end
# bit 10-19, acq_start
# 0x0000 = start=0, 0xx0400 start=1, 0x0800 start=2, 0x0C00 start=3
#
#./altrowrrdreg --fec 0 --chip 0 --register A --channel 0 --write 3E8 --verify
./altrowrrdreg --fec 0 --chip 0 --register A --channel 0 --write $shift1 --
verify
./altrowrrdreg --fec 0 --chip 1 --register A --channel 0 --write $shift2 --
verify
./altrowrrdreg --fec 0 --chip 2 --register A --channel 0 --write $shift3 --
verify
./altrowrrdreg --fec 0 --chip 3 --register A --channel 0 --write $shift4 --
verify
./altrowrrdreg --fec 0 --chip 4 --register A --channel 0 --write $shift1 --
verify
./altrowrrdreg --fec 0 --chip 5 --register A --channel 0 --write $shift2 --
verify
./altrowrrdreg --fec 0 --chip 6 --register A --channel 0 --write $shift3 --
verify
./altrowrrdreg --fec 0 --chip 7 --register A --channel 0 --write $shift4 --
verify
./mcmpca16 --fec 0 --preamp 0 --gain $2 --shaper $3 --shutdown 0 --polarity 1 --
write --verify
#./mcmwrreg --fec 0 --register $mcmreg --write --value 0
./swnettrig --loop 0 --events=$events
lastfile=`ls -rt $mydata | tail -n 1`
pushd /home/jum/ilc/ilcdaq/monitor-v10.1
./mon-file.sh $lastfile > slask0.out
grep "ERROR" slask0.out >>  $mydir/readme.txt
popd
echo $lastfile "Pedestal Channel 0-15 nominal gain$2-shaper$3" >>
$mydir/readme.txt
fileout=`echo $lastfile | sed -e s/_0.dat//g | sed -e s/readout-//g`;
echo ".x plotallped.cxx($1,$fileout,\"shift-gain$2-shaper$3\")"
fileout=`echo $lastfile | sed -e s/.dat/.root/g`;
echo $fileout > $myroot/onerun.txt
```

TRIGGERING

The trigger used is a software trigger. It works by a command sent to the SRU on the control network, the SRU then sends a first level trigger command to the CPLD, automatically followed by a second level trigger. The time between the first level and second level trigger is selectable via a SRU register. When the SRU receives the it sends it over the data network to the DAQ prorgam on the computer. The DAQ program, when ready for a new event, sends a ready command to the program issuing the software trigger. The trigger is issued by the program:

./swnettrig --loop 0 –events=<events>

where <events> is the number of triggers to issue. The program do as the first thing issue a START-OF-RUN command the DAQ program, then <events> triggers, when all triggers have been issued an END_OF_RUN is sent to the DAQ program.

DATA LOGGING

By default logging of data to a file is enabled. To disable data logging add the option --nologging to the command line of swnettrig.

./swnettrig --loop 0 –events=<events> --nologging

The data file written is:
wdir/ilcdaq/data/readout-<runnumber>_0.dat

NOTE 1: the testscript needs to be edited to change logging mode.
NOTE 2: the testscript will analyse the previous last data file anayway.

MONITORING

There is no proper data dump of an event. As shown in the script above is there a program to analyze a data file. The script and program to analyze a data file is in the directory:

ilc/ilcdaq/monitor-v10.1
./mon-file.sh $lastfile > slask0.out

where $lastfile is the name of data file (readout-<runnumber>_0.dat)

The root output file will be in:
wdir/ilcdaq/root/eadout-<runnumber>_0.root

The file slask0.out contains debugoutputs and data dumps for all events as raw data and unpacked data, example, first raw data and then unpacked for same channel:

```
>======= CHANNEL =======
Error 2
Channel 48 (30)
Length 134 (86)
Channels processed 1
START-1 60860030 START-2 80000410

  0: 08625443 04511044 04311444 04411044 04411043 04411445 04411044 04411044
  8: 04411443 04411045 04411044 04410C44 04411043 04411043 04411044 04411044
 16: 04511045 04511445 04411444 04511444 04411444 04411444 04411045 04511445
 24: 04511445 04511444 04511445 04411446 04511445 04511445 04511846 04611446
 32: 04611845 04511445 04511444 04511045 04511445 04511045 04411445 04411443
 40: 04110843 04511446 04611844 04511044
<====== CHANNEL =======
>====CHANNEL DATA=====
Channel 48 (30)
  0:0086 0095 0043 0045 0044 0044 0043 0045
```

```
  8:0044 0044 0044 0044 0044 0044 0043 0044
 16:0045 0045 0044 0044 0044 0044 0044 0044
 24:0044 0045 0043 0044 0044 0045 0044 0044
 32:0044 0044 0043 0044 0044 0044 0043 0044
 40:0044 0043 0044 0044 0044 0044 0044 0044
 48:0045 0044 0045 0045 0045 0045 0044 0045
 56:0044 0045 0045 0044 0044 0045 0044 0044
 64:0045 0044 0044 0044 0045 0045 0045 0045
 72:0045 0045 0045 0045 0045 0044 0045 0045
 80:0045 0044 0045 0046 0045 0045 0045 0045
 88:0045 0045 0045 0046 0046 0046 0045 0046
 96:0046 0046 0045 0045 0045 0045 0045 0045
104:0044 0045 0044 0045 0045 0045 0045 0045
112:0044 0045 0044 0045 0045 0044 0045 0043
120:0041 0042 0043 0045 0045 0046 0046 0046
128:0044 0045 0044 0044 0043 0043
```

VARIOUS USEFUL COMMANDS

Dump all register in CPLD:
./mcmdumpregs --fec <DTCnb>

Write to register in CPLD on MCM:
./mcmwrreg --fec <DTCnb> --register <mcmregister> --write --value <hex pattern>

Set preamplifier parameters (a register in CPLD):
./mcmpca16 --fec <DTCnb> --preamp 0 --gain <0-3> --shaper <0-7> --shutdown <0,1> --polarity <0,1> --biasctrl <0,1> --grst <0,1> --write –verify

<preamp> always 0
<gain> 0 = 12mV/fC, 1=next highest, 2=next lowest, 3=27mV/fC
<shaper> 0 = 120 ns, 4=90 ns, 6= 60 ns, 7 = 120 n

<shutdown> 0 – preamplifer on, 1 – preamplifer off
<polarity> 0 – positive, 1 – negative
<biasctrl> 0 – ADCs off, 1 ADCs on
<grstt> 0 - set tow Global Reset to SALTRO low, 1 – set Global reset to SALTRO high
<write> perform write to CPLD
<verify> read back and verify written value

Write to SALTRO register:
./altrowrrdreg --fec <DTCnb> --chip <0-7> --register <register in hex> --channel <0-15> --write <hex value> --verify

if –write is omitted is the register only read.

Dump all registers for a specific channel:

/altrodumpreg --fec <DTCnb> --chip <0-7> --channel <0-15>

Write a pattern to the pedestal memory:
./saltrowrmem --fec 0 –chip <0-7> --pattern <pattern>
Possible patterns are as ‘defined’ in saltrowrmem.c:
#define ALTRO_LOOKUP_NONE 0
#define ALTRO_LOOKUP_RAMP 1
#define ALTRO_LOOKUP_INVERT 2
#define ALTRO_LOOKUP_INVERT_8_15 2
#define ALTRO_LOOKUP_INVERT_0_7 3
#define ALTRO_LOOKUP_2AA 4
#define ALTRO_LOOKUP_155 5
#define ALTRO_LOOKUP_PEAKS 6
#define ALTRO_LOOKUP_FILE 7
#define ALTRO_LOOKUP_ZERO 8
#define ALTRO_LOOKUP_3FF 9
#define ALTRO_LOOKUP_WALK1 10
#define ALTRO_LOOKUP_RANDOM 11
#define ALTRO_LOOKUP_CH 12
#define ALTRO_LOOKUP_155_2AA 13
#define ALTRO_LOOKUP_VALUE 14
#define ALTRO_LOOKUP_3FF_000 15

Normally pattern = 1 ramp (between 0-999) used.

SALTRO register test:
./saltroregtest --fec <DTCnb> --chip <0-7> --pattern FFFF --loop <tests> --verify

<pattern> bit pattern of tests to do FFFF = all.
<loop> number of tests
<verify> read back and verify written value (if not it will only write and do not read back)

USING PEDESTAL MEMORY AS DATA

Instead of using the preamp inputs as data exists the possibility to use a memory in the SALTRO, this memory is normally used for time dependendant pedestal subtraction but can also be used as known data. To use pedestal memory as data - ramp only (0-9999) exist, set up dpcfg=vfped-fpd in wdir/ilcdaq/config-v10.1/global-altro.physics and redo startdaq in DaqPanels.

DPCFG START

Part of global-altro.physics to edit:

din - fpd do not use lookup table
DPCFG.FBCM 0

#-> din - vpd - fpd = subtract self ped – fixed
#DPCFG.FBCM 4
#-> f(din) - fpd = use lookup table subtract fixed pedestal
#DPCFG.FBCM 8
f(din-vpd) - fpd = use lookup table subtract self pedestal
#DPCFG.FBCM 9
f(t) - fpd = read pedestal memory as data
#DPCFG.FBCM 10

Normal running uncomment: and comment as:
DPCFG.FBCM 0
#DPCFG.FBCM 10

Using memory as data uncomment and comment as:
#DPCFG.FBCM 0
DPCFG.FBCM 10

As an example look in the script:
./testpedmen.sh <chip> <pattern>
pattern = 1 ramp

PROBLEMS

It takes sometimes long before the first event is transferred. If more than 5 minutes, you have to restart things. (see section “TEST COMMUNICATION”).

It sometimes just stops in the middle of a run, again see section “TEST COMMUNICATION”. If the communication works it might just be it has reached the event number 28560. Then it normally recovers by itself after a few minutes.

When the data is hangs like that this the SRU BUSY LED is lit, see section about SRU.

TODO

Scripts and programs have been developed as needed and are not all updated with latest or needed functionality, nor well written or documented. Can of course be worked on as needed.

TLU system

The TLU system software and documentation repository:
https://ohwr.org/project/fmc-mtlu

The documentations is also at:
https://www.hep.lu.se/eudet/saltro/developdoc/Documentation_Main_TLU.pdf

A copy of the software (20211026):
https://www.hep.lu.se/eudet/saltro/developdoc/fmc-mtlu-sw-master.tar.gz

With Lund changes:
https://www.hep.lu.se/eudet/saltro/developdoc/fmc-mtlu-sw-master-lund.tar.gz

The TLU requires the ipbus protocol to be used. The ipbus software can be fetched and installed as described at:
https://ipbus.web.cern.ch/doc/user/html/software/install/compile.html

The TLU controls clock, trigger, busy, spill, synk/T0 and sending a trigger number. This is done via a HDMI cable with LVDS levels. We have made an interface card to provide this to the SRU. The interface card split the HDMI signals to individual LVDS LEMO connectors.



Naming as given as in the TLU documentation and the picture above.

J1 : HDMI cable from TLU.
J2: CONT = spill signal to SRU. Se chapter 6 (e.g. Figurer 6.1 for timing) in TLU documentation.
J3: CLK = 40 Mhz clock to SRU
J4: BUSY = Busy from SRU
J5: TRIG = Trigger to SRU (see Figure 4.3 in documentation,)
J6: SPARE = synk/T0 and trigger number to SRU (see Figure 4.4 in documentation)

synk/T0 is issued before any triggers are sent in order to synchronize time stamps etc…
In the mode we will use (tAIDA mode with trigger output) he TLU is for each trigger a trigger number sent on the line.

The figures mentioned are replicated here.

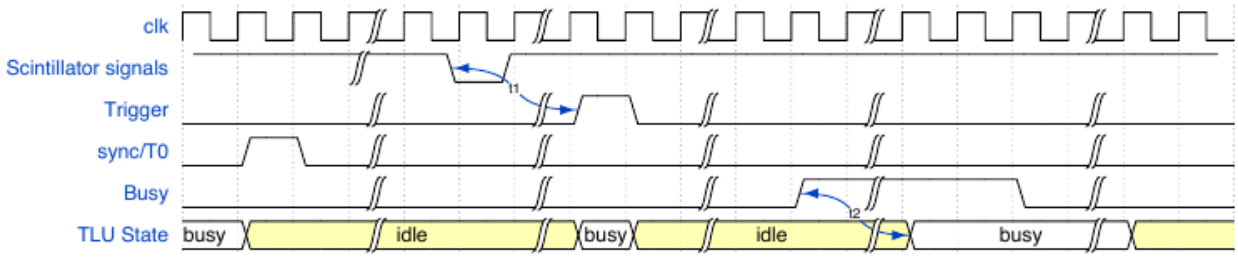


Figure 4.3: Synchronous (AIDA) Interface Mode

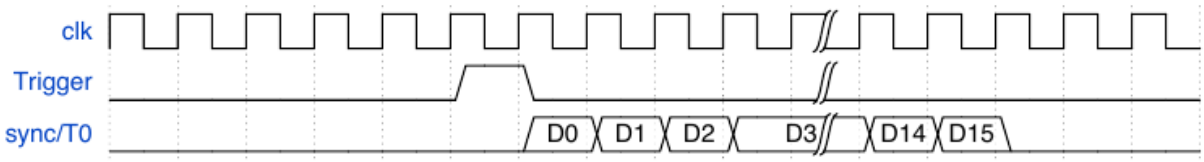


Figure 4.4: Synchronous (AIDA) Interface Mode With Trigger Number

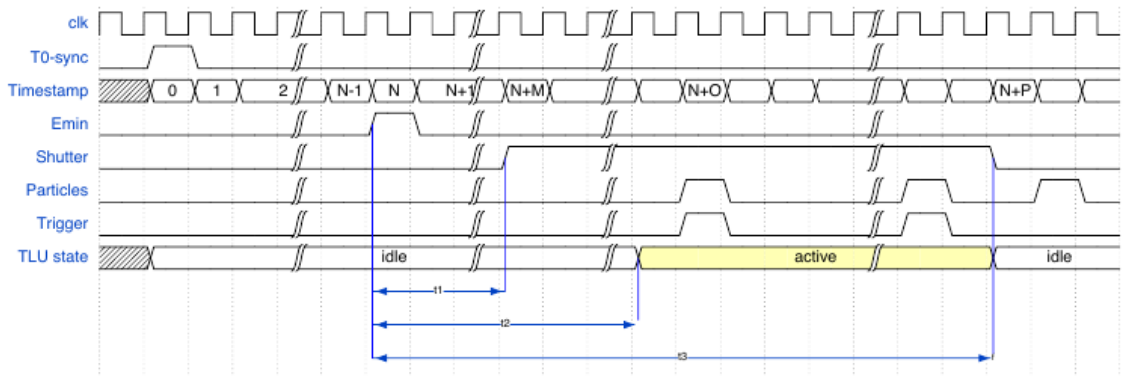


Figure 6.1: Shutter Timing: the E_{min} signal is fed to one of the trigger inputs and initiates the shutter sequence; after a programmable delay t_1 the TLU asserts the *shutter* signal. The unit will start to issue trigger signals to the DUT once a programmable time t_2 has elapsed. The window between t_1 and t_2 can be used to ensure the DUT is configured and ready to accept triggers. The unit will issue triggers until the end of the shutter window, determined by t_3.

HDMI PIN	HDMI Signal Name	Enable Signal Name
1	HDMI_CLK	ENABLE_CLK_TO_DUT or ENABLE_DUT_CLK_FROM_FPGA
2	GND	–
3	HDMI_CLK *	ENABLE_CLK_TO_DUT or ENABLE_DUT_CLK_FROM_FPGA
4	CONT	ENABLE_CONT_FROM_FPGA
5	GND	–
6	CONT*	ENABLE_CONT_FROM_FPGA
7	BUSY	ENABLE_BUSY_FROM_FPGA
8	GND	–
9	BUSY*	ENABLE_BUSY_FROM_FPGA
10	SPARE	ENABLE_SPARE_FROM_FPGA
11	GND	–
12	SPARE*	ENABLE_SPARE_FROM_FPGA
13	n.c.	
14	HDMI_POWER	–
15	TRIG	ENABLE_TRIG_FROM_FPGA
16	TRIG*	ENABLE_TRIG_FROM_FPGA
17	GND	
18	n.c.	
19	n.c.	

Table 2.1: HDMI pin connections.

The HDMI pin is the Calice HDMI pin-out, the TRIGGER lines use wires dedicated for I2C communication. These often aren’t very good quality. HDMI cables < 2 meters has always worked.

TLU control

The TLU is controlled via ipbus protocol over ethernet.
The predefined IP-address is: 192.168.200.3

Scripts and configurations files are in the directory:
~ilc/ilcdaq/TLU/fmc-mtlu-sw-master/TLU_v1e/scripts/

I have modified the python scripts that communicates with the TLU.

Main script modified for Lund use is:
tartTLU_v1e-lund.py

The main script need the definitions in:
TLU_v1elund.py

To execute the script:
python startTLU_v1e-lund.py

commands used in the script:

configure	reads configuration file and loads into TLU
startRun	start a run
triggers	list number of triggers so far
endRun	stop a run

The configuraion file which is loaded is:
localConf-lund.conf

Lund settings in the configuration file:

DUTMaskMode= 0x55	All DUTs : 01 AIDA mode with trigger number output
HDMI1_clk = 2	Clock from FPGA on DUT1
trigMaskHi = 0x00000000	
trigMaskLo = 0x00000002	Only input 1 used as trigger
DACThreshold0 = -0.4	Trigger threshold in V on input 1 (trigger input)
DACThreshold5 = -0.4	Trigger threshold in V on input 6 (spill signal input)
InternalTriggerFreq = 0	Generate internal triggers (in Hz, 0= no triggers)
# Shutter settings	
EnableShutterMode = 1	set to 0 to disable
ShutterSource = 5	trigger input 6 as source
InternalShutterInterval = 0	no internal shutter
ShutterOnTime = 40	from shutter signal to shutter is set 25 ns * 40 = 1 microsecond
# from shutter signal to triggers accepted 25 ns * 6000 = 150 microseconds	
# to allow power up on SALTRO.	
ShutterVetoOffTime = 6000	
# 25 ns * 80000 = 2.0 ms	
ShutterOffTime = 80000	spill off after 2 ms

Alternative is to use the full EUDAQ2 DAQ system, but it seems on overkill for our tests.

EUDAQ repository:
https://github.com/eudaq/eudaq

SRU connectors

On the SRU do we need:

one LVDS output for BUSY
four LVDS inputs for TRIG,CLK,CONT,SPARE

Since there is only four LEMO LVDS on SRU do we need to convert one signal to NIM, e.g. CONT which is a level.

Do we need to change LVDS on the SRU to allow 1 output and 3 inputs?

PULSE GENERATOR

3 st setups sparade.

För båda gäller: Spill on/off repeteras med internt trigger intervall 50millisek. (ändras med trigger interval under burstknappen meny 2). Spill on har delay noll relativt interna triggeren. Med en extra pulsgeerator kan man trigga externt om man vill.
Kanal 1 spill on/off
Kanal 2 trigger puls.

Setup 1:
Kanal 1 ger en lång nivå tänkt som spill on tiden som går till -800mV när spillet kommer på. Hur länge spillet är på sätter man med vidden på kanal 1 (i setup 1 satt till 9.99millisec). Denna mod motsvarar att vi kör utan TLU.
Kanal 2 är triggerpulsen. Gjort den 30ns lång. Delay relativt trigger (och därmed spill on) sätts med Trigger delay i setupen satt till 1millisec.

Setup 2
Kanal 1. Spill on satt till 30ns I övrigt som setup 1.
Kanal 2: Som setup 1.
Denna mod motsvarar att signalerna går till TLUn och att denna generera spill on och trigger.

Setup 3
som inverterar kanal 1 i setup 1 för den händelse att SRU skall ha 0V när spillet är på.

For our TLU setup should
Channel 1 (spill) be connected to trigger input 6
Channel 2 (trigger) be connected to trigger input 1.