Status Mini DAQ Software Control System

J.P. Cachemiche, P.Y. Duval, F. Hachon
R. Le Gac, F. Réthoré

CERN – 12 dec 2013 – LHCb upgrade AMC40 firmware

Overview

- LLI V1.0
- Linux driver
- User library
- Working tools: libraries and commands
- Conclusion

LLI and user driver status

The LLI V1.0 is available

Component	access	library	command	Config menu	status
minipods	PCIe/I2C	libminipods.a	minipods	NYI*	OK
FPGA transcievers	MM registers	libphy.a	phy	NYI*	OK
PLL	PCIe/SPI	libpll.a	pll	pll_control	OK
user registers	MM registers	libuser.a	demo application	NYI*	OK

* NYI: Not Yet Implemented

The LLI distribution V1.0

Available for dowloading in the AMC40 forge project at :

https://lbredmine.cern.ch/projects/amc40/wiki/Low_Level_interface_Software

Contains:

- the driver to access the firmware registers via the PCIe bus
- libraries to read/write the registers: user registers and LLI internal registers
- programs/commands to configure the AMC40 components
  - minipods
  - FPGA phy interface
  - pll
- a simple demo program to control the demo firmware application

At this URL you also have links to get:

- the firmware to load the demo application in the FPGA
- the LLI user guide

CCPC components (reminder)

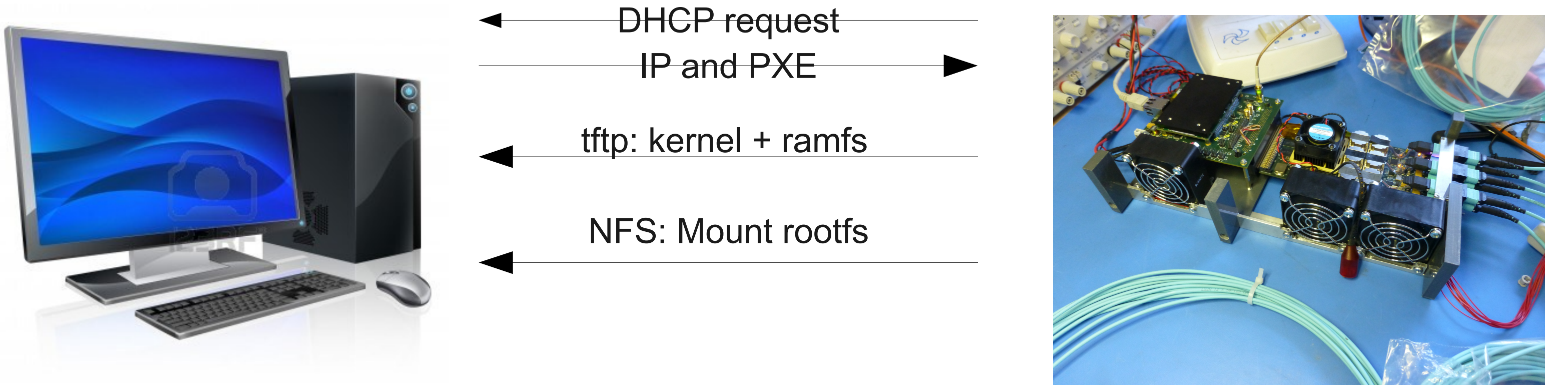
The CCPC is a diskless system with a linux 2.6.39 kernel and SL62 distribution using the PXE protocol to boot (supports the PCH_GBE controller).

It needs a server to provide:

- its IP address and boot code (DHCP)
- its kernel and initial file system in RAM (tftp)
- its final root file system to run (NFS)

The CCPC BIOS has been set to boot via PXE.
The needed files are available in the REDMINE project CCPC-Common
https://lbredmine.cern.ch/projects/ccpc-common
via the WIKI pages.

A guide explains how to configure the server for your CCPC.



DHCP request

IP and PXE

tftp: kernel + ramfs

NFS: Mount rootfs

Linux driver for user code registers

One driver to allow read and write accesses to registers mapped in PCIe address space.

The LLI V1.0 uses 2 BARs:

BAR 0: 32 bits non prefetchable memory space for user code registers (/dev/ecs_bar-0)
BAR 2: 32 bits non prefetchable memory space for internal LLI registers (/dev/ecs_bar-1)

BAR 0 is exported through a bridge to user code (see LLI specifications)

The driver is in lli_root/driver directory:

Go in this directory and execute
./start.sh (as root)

User library

The user library is used to write programs that access in read/write mode the registers implemented in user firmware (under the BAR0).

Very simple set of functions .

```
void lbPcie_user_init();
void lbPcie_user_close();

//Register
void lbPcie_user_readW(unsigned base_add, unsigned *val);
void lbPcie_user_writeW(unsigned base_add, unsigned *val);

int lbPcie_user_write(unsigned base_add, unsigned *val, int
size);
int lbPcie_user_read(unsigned base_add, unsigned *val, int size);
```

A simple demonstration program is included in the LLI distribution in directory tests/ecs.

Libraries for LLI internal resources

The LLI needs several user libraries to manage the board basic components:

- the minipods configuration (via an I2C bus interfaced to the PCIe)
- the PLLs configuration (via an SPI bus interfaced to the PCIe)
- the FPGA optical links PHY components (internal registers)

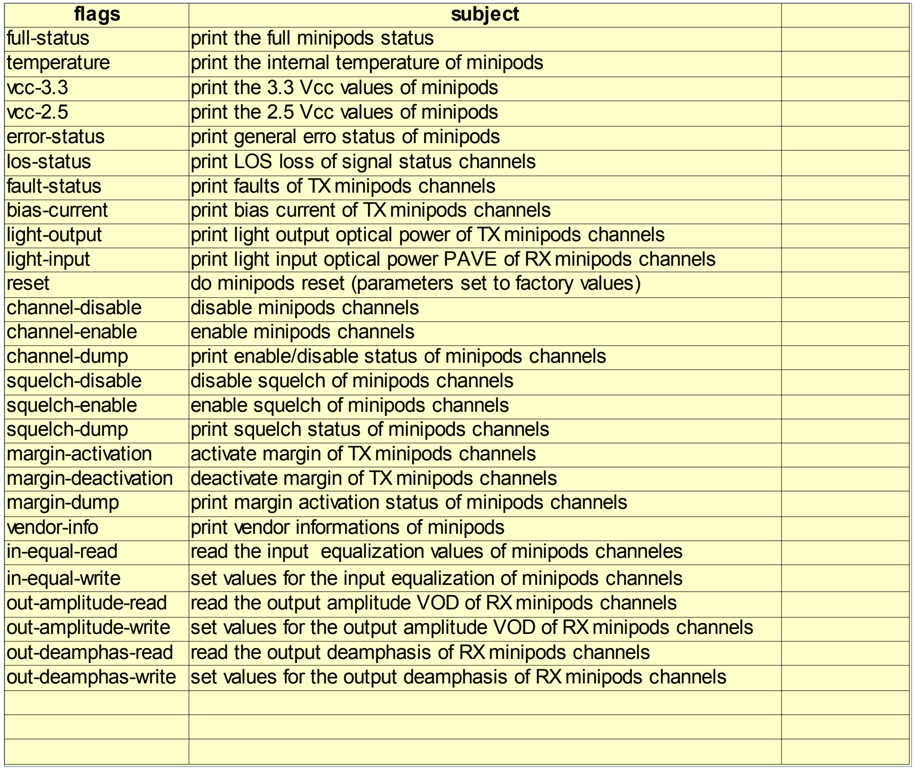
Those resources are mapped in BAR2 space.

Those libraries are not meant to be used directly by users but are building pieces of two types of programs:

- commands to be used in scripts for resources configurations
  (since the previous presentation the commands have been converted to long options)
- menu based programs to configure the resource in inter-active mode

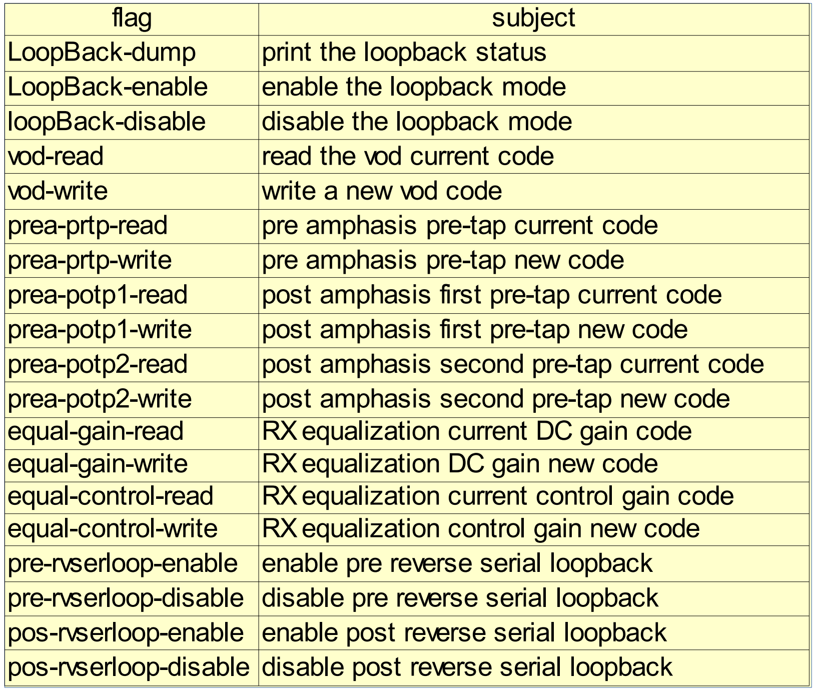
LLI commands (long options)

Minipods parameters



flags	subject	
full-status	print the full minipods status	
temperature	print the internal temperature of minipods	
vcc-3.3	print the 3.3 Vcc values of minipods	
vcc-2.5	print the 2.5 Vcc values of minipods	
error-status	print general erro status of minipods	
los-status	print LOS loss of signal status channels	
fault-status	print faults of TX minipods channels	
bias-current	print bias current of TX minipods channels	
light-output	print light output optical power of TX minipods channels	
light-input	print light input optical power PAVE of RX minipods channels	
reset	do minipods reset (parameters set to factory values)	
channel-disable	disable minipods channels	
channel-enable	enable minipods channels	
channel-dump	print enable/disable status of minipods channels	
squelch-disable	disable squelch of minipods channels	
squelch-enable	enable squelch of minipods channels	
squelch-dump	print squelch status of minipods channels	
margin-activation	activate margin of TX minipods channels	
margin-deactivation	deactivate margin of TX minipods channels	
margin-dump	print margin activation status of minipods channels	
vendor-info	print vendor informations of minipods	
in-equal-read	read the input equalization values of minipods channeles	
in-equal-write	set values for the input equalization of minipods channels	
out-amplitude-read	read the output amplitude VOD of RX minipods channels	
out-amplitude-write	set values for the output amplitude VOD of RX minipods channels	
out-deamphas-read	read the output deamphasis of RX minipods channels	
out-deamphas-write	set values for the output deamphasis of RX minipods channels	

FPGA PHY parameters



flag	subject
LoopBack-dump	print the loopback status
LoopBack-enable	enable the loopback mode
loopBack-disable	disable the loopback mode
vod-read	read the vod current code
vod-write	write a new vod code
prea-prtp-read	pre amphasis pre-tap current code
prea-prtp-write	pre amphasis pre-tap new code
prea-potp1-read	post amphasis first pre-tap current code
prea-potp1-write	post amphasis first pre-tap new code
prea-potp2-read	post amphasis second pre-tap current code
prea-potp2-write	post amphasis second pre-tap new code
equal-gain-read	RX equalization current DC gain code
equal-gain-write	RX equalization DC gain new code
equal-control-read	RX equalization current control gain code
equal-control-write	RX equalization control gain new code
pre-rvserloop-enable	enable pre reverse serial loopback
pre-rvserloop-disable	disable pre reverse serial loopback
pos-rvserloop-enable	enable post reverse serial loopback
pos-rvserloop-disable	disable post reverse serial loopback

LLI commands examples (long options)

The minipods can be controlled typing commands or in bash script like:

```
#!/bin/bash
#Print the squelch setting of all minipods
        minipods --channel-dump
#Print the squelch setting of minipod 4
        minipods --squelch-dump  -m4
#Disable the squelch of all channels of all minipods
        minipods -squelch-disable
#Disable the squelch of all channels 5 of all minipods
        minipods –squelch-disable  -c5
#Print the equalization setting of all channels of minipod 2
        minipods --in-equal-read -m2
#Print the equalization setting of channel 1 of minipod 0
        minipods --in-equal-read -m0 -c1
#Set 12 code values for the equalization setting of the 12 channels of all RX minipods
        minipods --in-equal-write  -l1,1,1,1,1,1,2,2,2,2,2,2
#Set 12 code values for the equalization setting of the 32 channels of minipod 4
        minipods --in-equal-write -m4  -l3,3,3,2,2,3,2,2,0,0,0,2
```

LLI commands examples

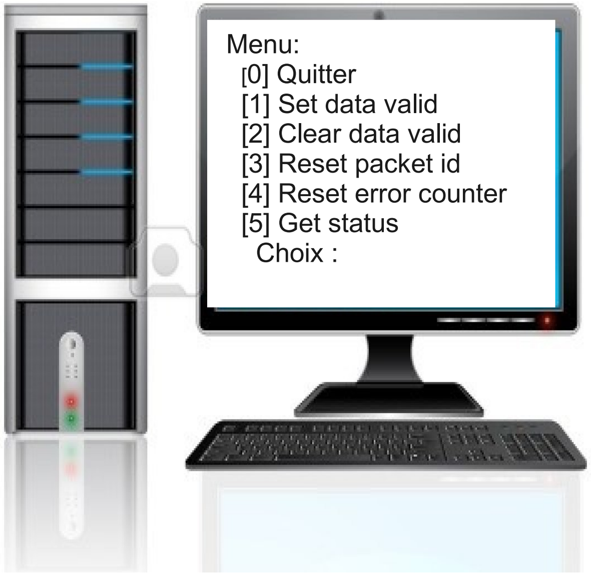
The minipods can be controlled via commands in python script like:

```
#/usr/bin/python
import os

f = os.popen(“minipods --channel-dump”)
print “Squelch setting of all minipods :”, f.read()

print “Disabling the squelch of all channels of all minipods”
f = os.popen(“minipodCmd -squelch-disable ”)

print “Set 12 code values for the equalization setting of the 12 channels of all RX minipods”
f = os.popen(“minipodCmd  --in-equal-write  -l1,1,1,1,1,1,2,2,2,2,2,2“)
```

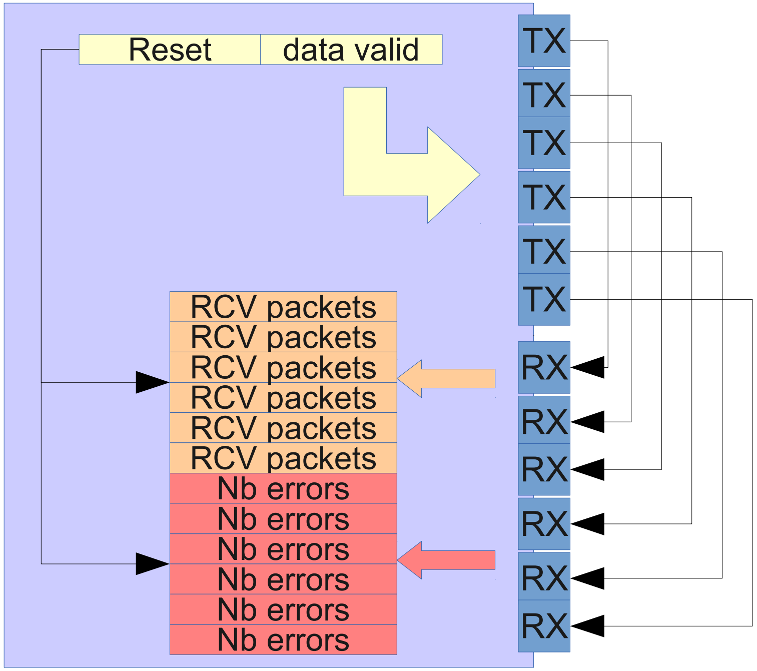
Demo application

LLI V1.0 contains a demo application in the directory: lli_root/user

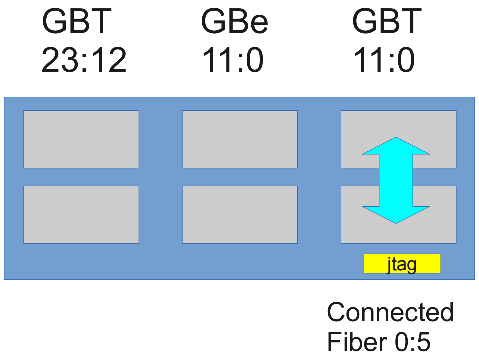
The application is a simple menu based program to start/stop emitting packets on 6 links and display the number of packets sent and errors detected.



```
Menu:
  [0] Quitter
  [1] Set data valid
  [2] Clear data valid
  [3] Reset packet id
  [4] Reset error counter
  [5] Get status
    Choix :
```



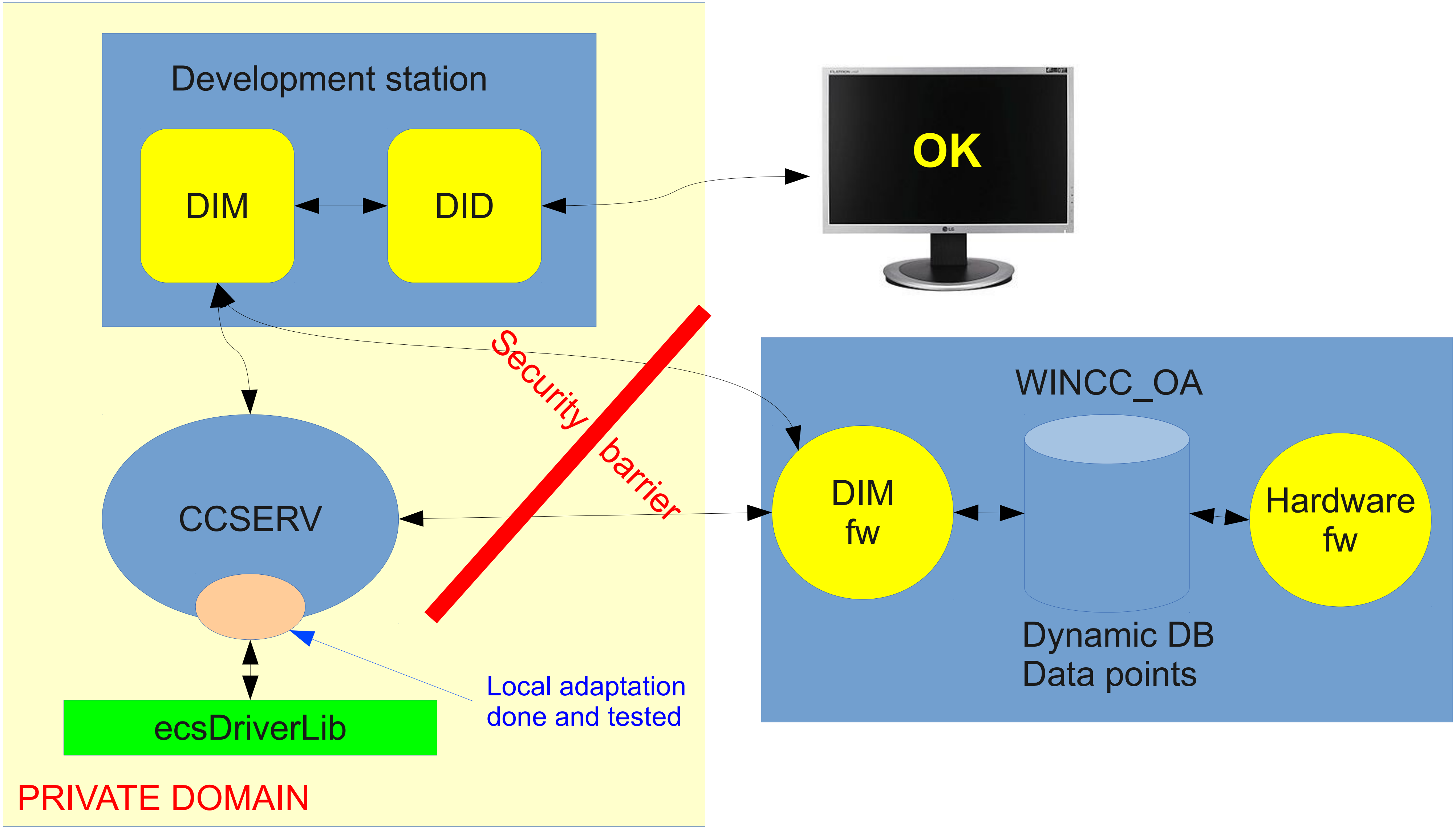
FPGA Logical view



AMC40 front pannel

ECS integration in WINCC-OA

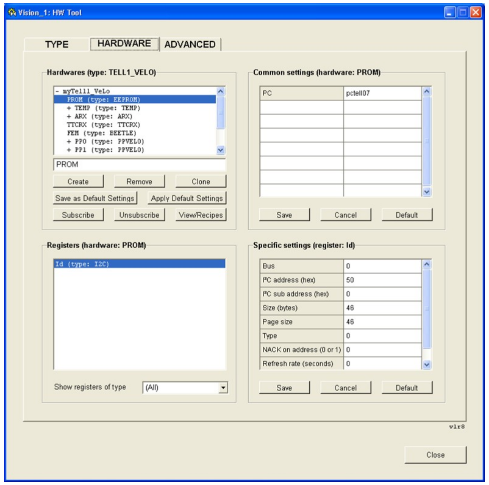
The CCSERV has been compiled and successfully tested with DIM on the AMCTP



ECS integration in WINCC-OA

Next steps:

- test with full fwHardware in the CERN wide network
- XML format to feed fwHw with registers descriptions



Hardwares (type: TELL1_VELO)

```
- myTell1_VeLo
    PROM (type: EEPROM)
    + TEMP (type: TEMP)
    + ARX (type: ARX)
    TTCRX (type: TTCRX)
    FEM (type: BEETLE)
    + PP0 (type: PPVELO)
    + PP1 (type: PPVELO)
```

PROM

Common settings (hardware: PROM)

PC	pctell07

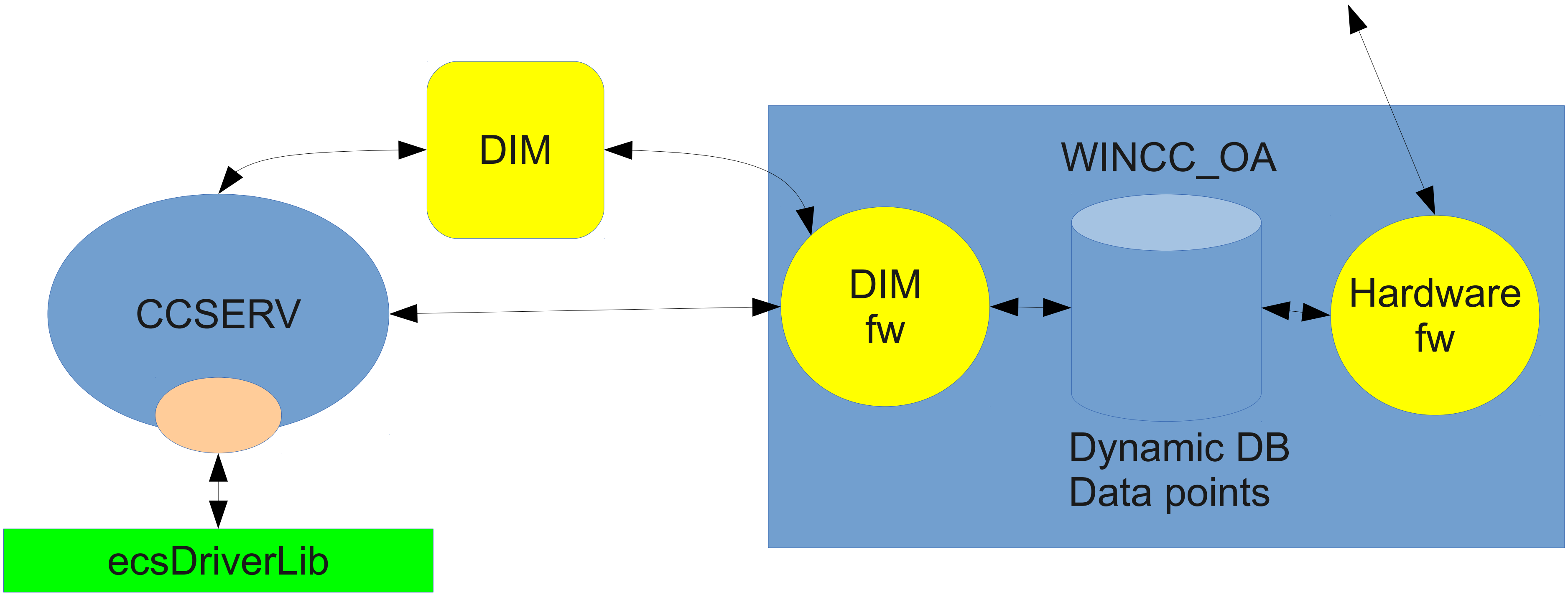
Registers (hardware: PROM)

Id (type: I2C)

Show registers of type (All)

Specific settings (register: Id)

Bus	0
I²C address (hex)	50
I²C sub address (hex)	0
Size (bytes)	46
Page size	46
Type	0
NACK on address (0 or 1)	0
Refresh rate (seconds)	0



Conclusion

All software pieces are there to build a control program to manage your firmware on the CCPC.

- a bootable Linux system
- the driver to link PCIe mapped resources into the OS file system
- the libraries to read/write into those resources/registers
- integration into CCSERV for WINCC-OA hardware framework done

1- Load the FPGA with the demo firmware and exercise the commands and programs
2- Analyse the example demo firmware to understand how to create PCIe mapped registers
3- Look at the (simple) example to see how to use the library to access those registers
4- Use the forge to share your feedback (*New issue* and *Issues* tabs) in AMC40 project

Useful links:

https://lbredmine.cern.ch/projects/ccpc-common/wiki/Kernel_and_distribution

https://lbredmine.cern.ch/projects/amc40/wiki/Low_Level_interface_Software