ATLAS DCS

Radiation Monitoring outside ID region

Document Version: 1.0
Document ID: none
Document Date: 07.10.2008
Document Status: In Work

Institutes and Authors:
CERN: S. Franz

Table of Contents

1. Introduction

The level of radiation level in the ATLAS detector will be measured at several places and by different ways. The purpose of this document is the description of the hardware and software implemented for the radiation monitors inside the detector but only outside the Inner Detector (ID) region. The hardware and software involved for the ID region is slightly different and is not the subject of this document.

These radiation monitoring sensors are meant to measure the TID[1] (in Gy) and the 1MeV equivalent neutron NIEL[2] (n/cm2) in the toroid area (Muon, PP2) and in the calorimeter (Tile Cal, Larg).

2. Hardware components

2.1. Sensors

The sensors are embedded on a small PCB (see Figure 1) which contains:

- A CMRP[3] diode to measure the Non-Ionizing Energy Loss (NIEL) deduced from the bulk damage in silicon which increases the forward voltage at constant current. This sensor has a full scale of 10^{15} n/cm² with a sensitivity of 10^{9} n/cm².
- A RADFET to measure the TID deduced from the threshold voltage increase over a 1.6 µm thick oxide. This sensor has a full range of ~10 Gy with a sensitivity of ~mGy.
- A NTC to measure the temperature for correction.

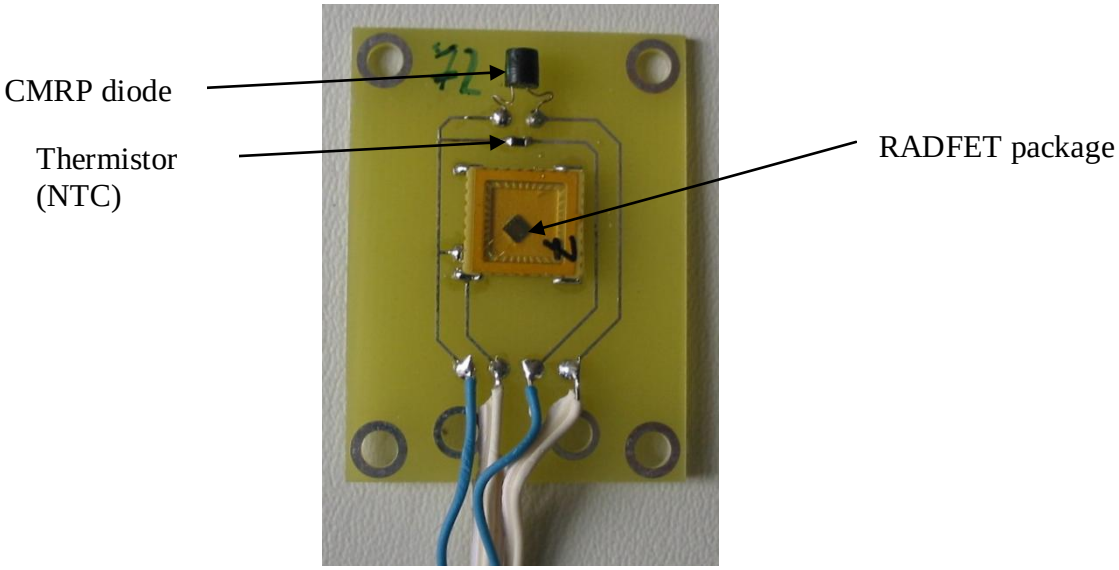


Figure 1: RadMon sensor board

[1] Total Integrated Dose
[2] Non Ionizing Energy Loss
[3] Clamped Mode Resonant Pole

2.2. Readout chain and powering

The Front End (FE) readout electronics are embedded in Aluminum boxes which are installed in the Toroid area and on the external structures in the experimental cavern (UX15) of ATLAS (see Figure 2).

Figure 2: Readout box

These readout modules provide the connectivity to the sensors through Lemo connectors and a 24 V supply to the sensors and are daisy chained on a field bus (CANOpen protocol). 6 boxes are installed in UX15 to read out 48 sensors:

- Tile Cal and Larg (22 sensors): 2 boxes on the voussoirs of sector 13 in the toroid, side A and C (respectively Ids RAD2-09.PVA13.XB and RAD4-09.PVC13.XB)
- PP2 areas (10 sensors): 2 boxes on the voussoirs of sector 5 in the toroid, side A and C (respectively Ids RAD2-10.PVA05.XB and RAD4-10.PVC05.XB)
- Muon wheels (16 sensors): 2 boxes on the HS structure, USA15 side, level 3 (Ids RAD5-17.HS-USA.X3 and RAD5-18.HS-USA.X3)

The readout module is made with 3 elements (see Figure 3):

- An Embedded Local Monitoring Board (ELMB) which provides:
  - The interface to the field bus with a CAN controller
  - 64 analog inputs (16 bits ADC) to read out the sensors
  - A Serial Protocol Interface (SPI) to drive an external DAC
- A DAC (type MAX525) which feeds current to the sensors for the readout
- A patch panel board which makes the connectivity between the ELMB, the DAC and the sensors

Using these elements, the readout procedure of the sensors is the following:

- Set high current in DAC channel which controls the switches
- Loop over sensors
  - Read ADC of the temperature sensor
  - Set the output current in the PIN DAC channel (100 µA)
  - Wait 100 ms
  - Read ADC of the PIN diode
  - Read ADC of the current control
  - Set the PIN DAC channel to 0
  - Set the output current in the RADFET DAC channel (1 mA)

- Wait 1 s
- Read ADC of the RADFET
- Read ADC of the current control
- Set the RADFET DAC channel to 0
- Go to next sensor with this loop

- Set current in DAC channel which controls the switches to 0

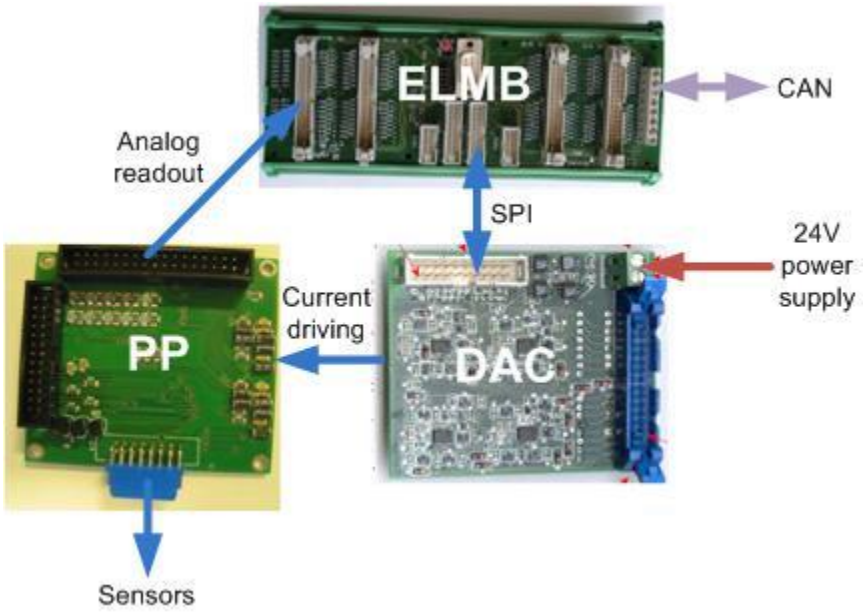


Figure 3: Readout components

3. Software

3.1. Front End computing environment

The FE electronics are driven by CAN bus from the USA15 counting room. A PVSS project contains all the software packages which are necessary to operate the Radiation Monitoring application. This application runs as a client of an OPC server which connects to the hardware. Both of them are installed on a single computer (A machine of the Common Infrastructure Control: PCATLCICUXDET).

IMPORTANT: An OPC server version 2.9.4.1 (release of 10 January 2008) or above is mandatory to ensure proper behavior of the complex RadMon readout scheme described above.

In order to match with the readout requirements (cf. chapter 2.2), the commands sent to the hardware are sent over the CAN bus using SDO (Single Digital Objects) which allow sequential operations on demand. The necessary EEPROM settings of the ELMB to match with SDO readout and analog outputs are shown in figure 4 and 5.

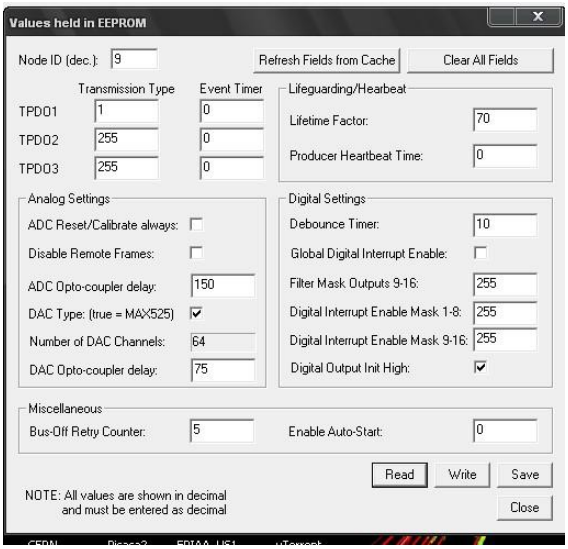


Figure 4: ELMB EEPROM settings for SDO R/O

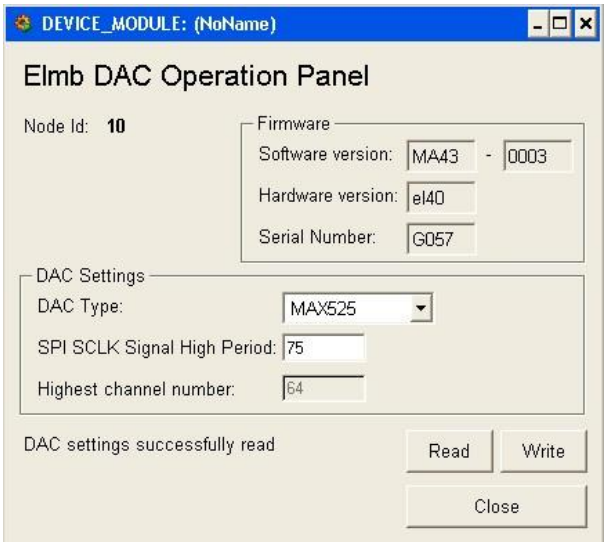


Figure 5: ELMB DAC settings

The calibration constants and the online values updated from the hardware are stored in a data-point structure partitioned for each individual sensor, following the structure (see Figure 4):

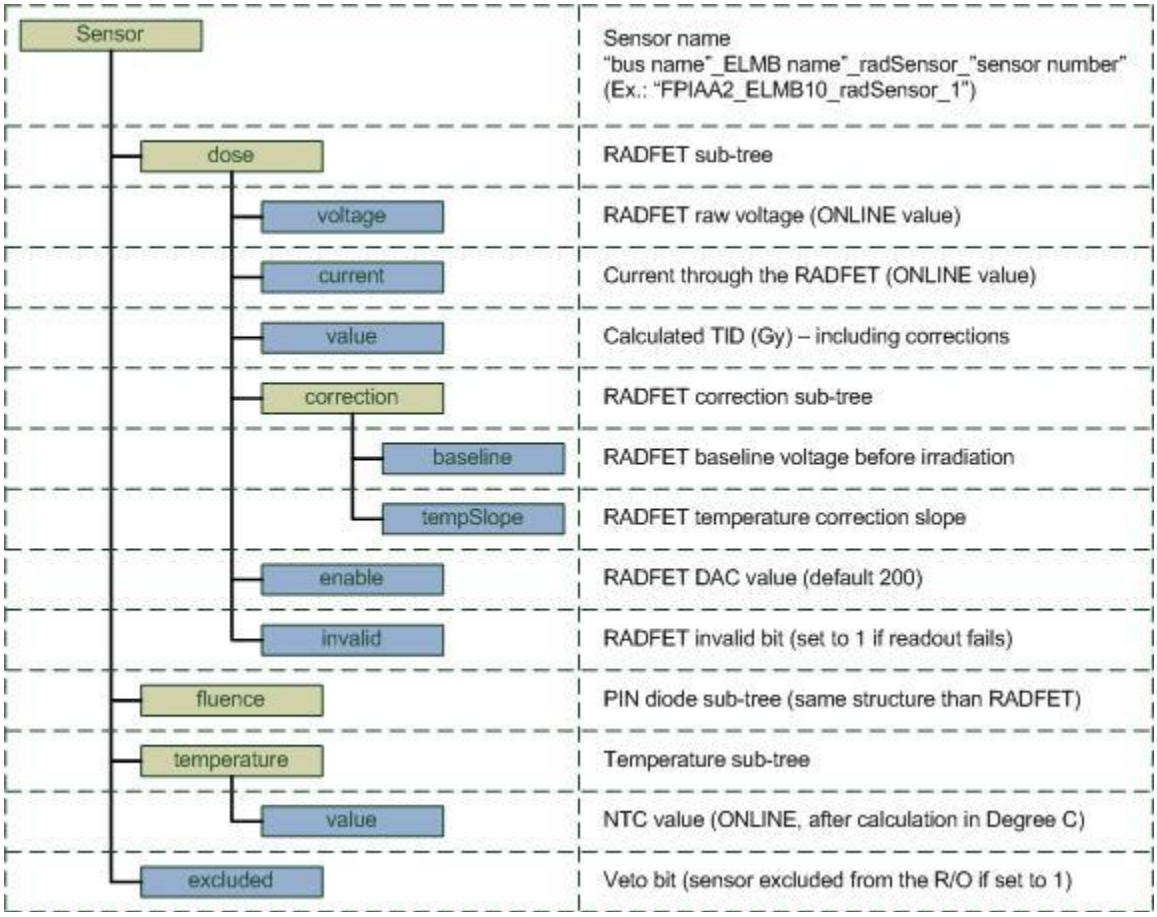


Figure 4: Radiation Monitoring DP structure

3.2. Raw data correction

Two correction constants are needed to operate the radiation monitoring sensors:

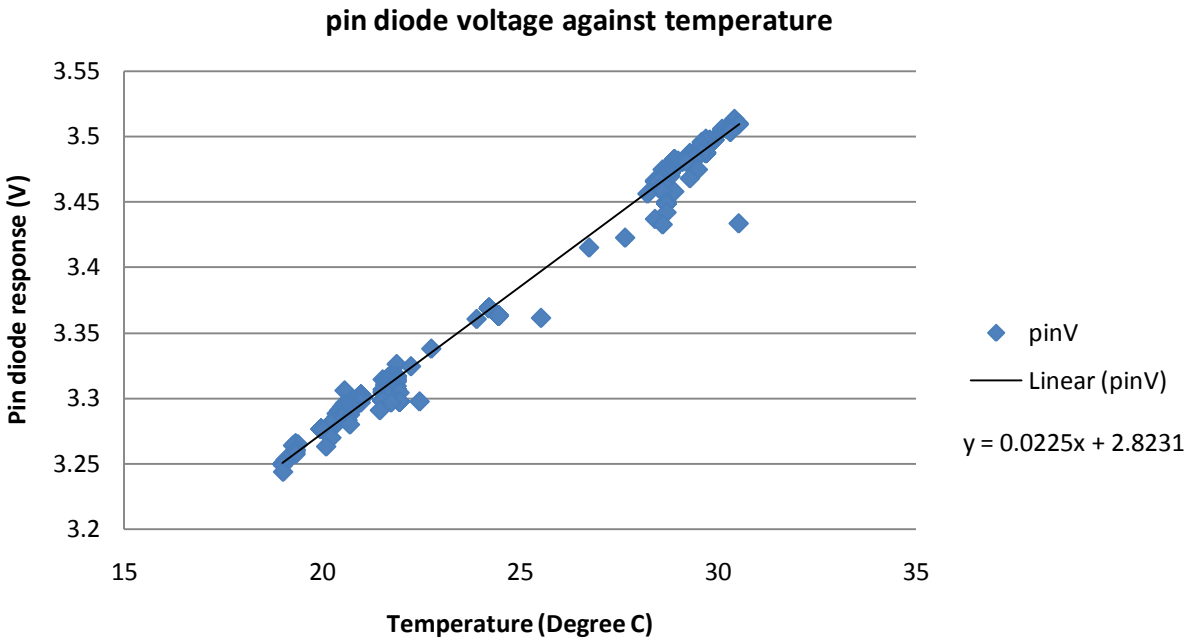
- The raw voltage baseline which corresponds to the forward voltage given by the RADFET and the PIN diode before irradiation
- The temperature correction slope, because the RADFET and the PIN diode response is temperature dependant.

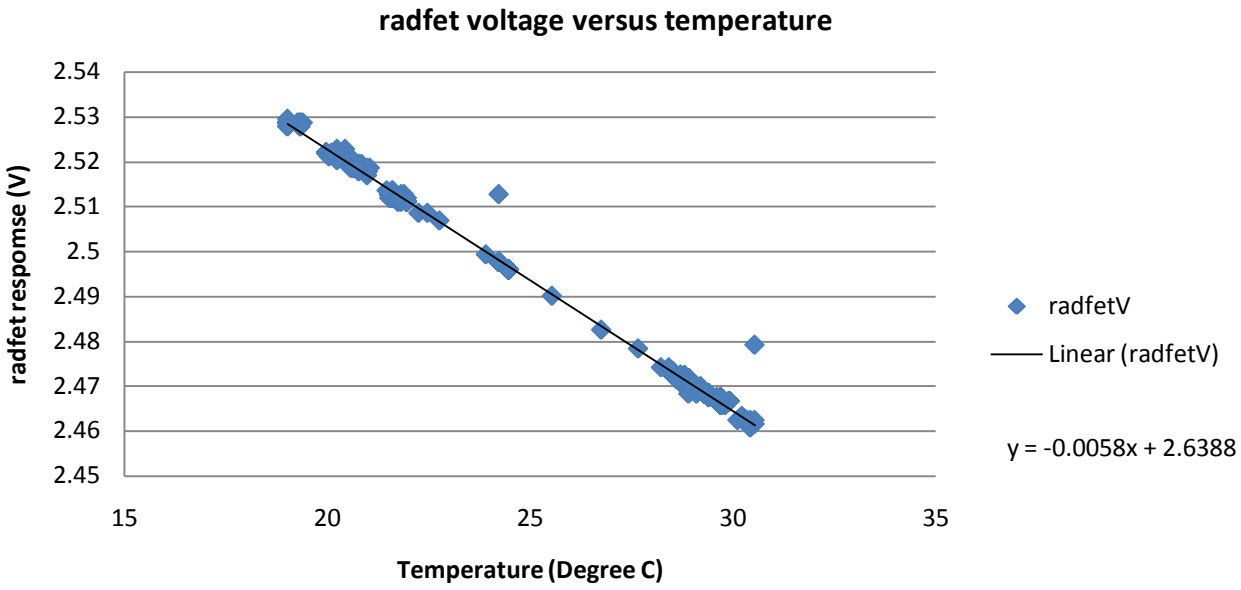
In order to get these correction constants, the sensors have been read out during 1 month and a half, and the data archived. The data analysis of this run is given below for one sensor (A.FS07.L29 from Larg) and the full table of correction constants is given in Annex 1. For some of them, the analysis failed, either because the sensor is malfunctioning (bad connection, to be solved at next ATLAS opening) or because the measured temperature of the concerned sensor has change in a too small scale during the data taking (i.e. the voltage drop due to temperature effect is of the same order than the instability of the readout).

Raw data (average over run period):

RAW	PIN (V)	PIN (uA)	Radfet (V)	Radfet (uA)
mean	3.372	1044.892	2.497	99.151
stdev	0.090	0.440	0.023	0.488

PIN diode and RADFET voltages against temperature:



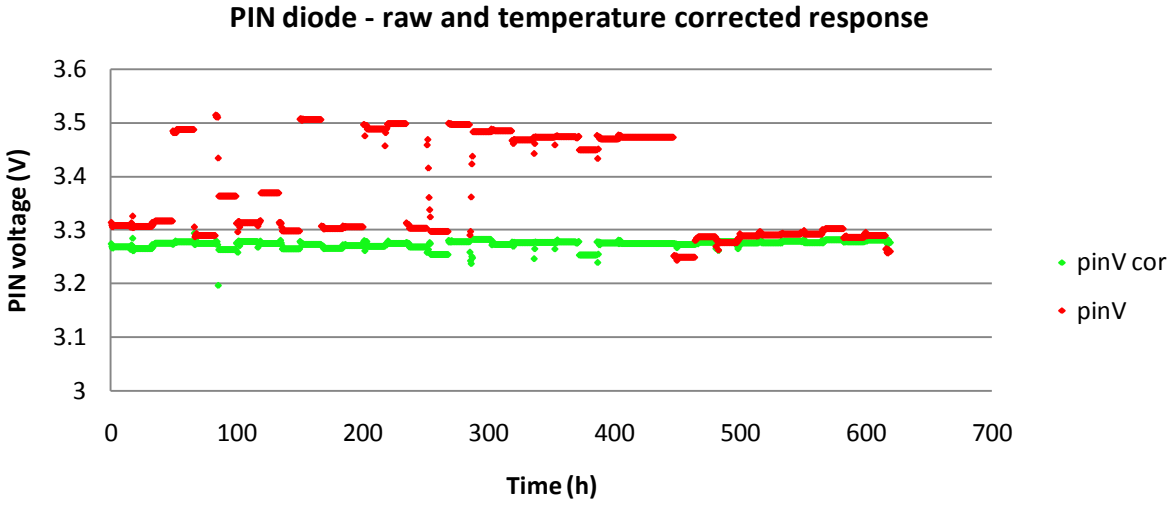


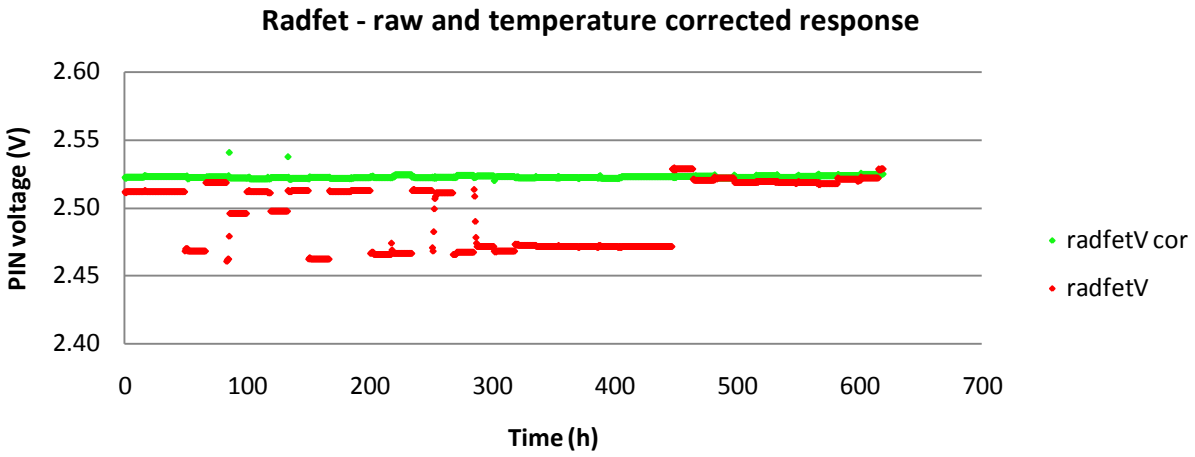
Data after temperature correction (average over run period):

Corrected	PIN (V)	PIN (uA)	Radfet (V)	Radfet (uA)
mean	3.273	1044.891	2.523	99.154
stdev	0.007	0.441	0.001	0.485

These values will be taken as baseline voltages for the TID and the NIEL at reference temperature of 20 Degree C.
The standard deviation of the voltages of the RADFET and the PIN diode indicate, after calculation, a resolution of respectively few mGy and ~2E9 n/cm², which is at the order of magnitude of the sensors sensitivity.

Raw and corrected data over the run period:





Finally the 2 plots above show that the temperature correction makes its job to get a stable readout of the voltages.

3.3. Readout software structure

A Control manager has been set up in the PVSS environment to read out the sensors (see Figure 5), there are 3 elements (Scripts are given in Annex 2):

- A readout module which loops over every sensor and reads sequentially the raw voltages giving the temperature, the TID and the NIEL.
- A data quality function which checks the consistency of the newly read out data and eventually triggers a new data acquisition if the data are not consistent.
- Data computing which makes the temperature correction and calculates the TID and the NIEL (see 3.4. for calculation steps)

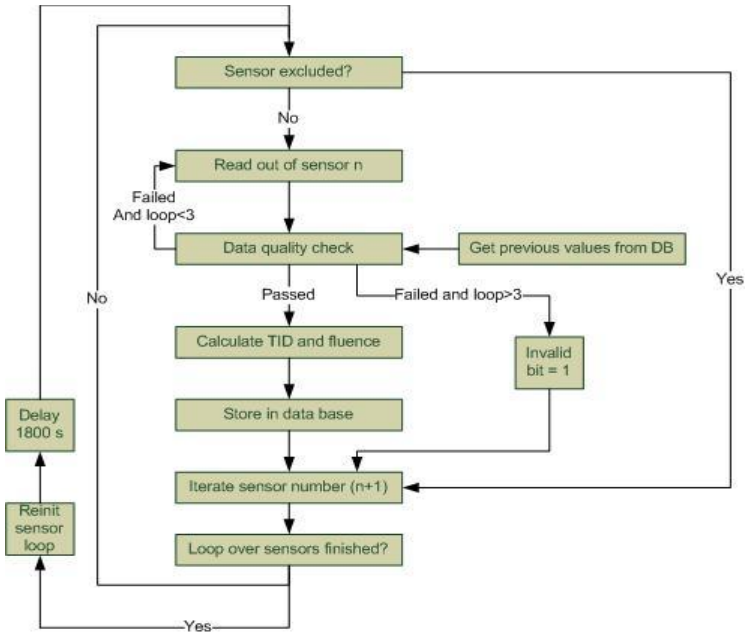


Figure 5: Readout software block diagram

3.4. TID and NIEL calculation

First the raw data are corrected in temperature:

$$corPinV = rawPinV - pinS \times (T - T_{ref}) \quad (V)$$
$$corRadV = rawRadV - radS \times (T - T_{ref}) \quad (V)$$

With:

- rawPinV and rawRadV respectively the PIN diode and RADFET raw voltages (V)
- corPinV and corRadV respectively the PIN diode and RADFET corrected voltages
- pinS and radS respectively the PIN diode and RADFET temperature correction slopes (V/Degree C)
- T = Temperature (Degree C)
- Tref = 20 Degrees C (reference temperature for correction)

Then the TID (D) and the fluence (Φ) are calculated from the corrected voltages:

$$D = e^{\frac{1}{Cdb} \times [\ln(corRadV - radV_0) - \ln(Cda)]} \quad (Gy)$$
$$\Phi = C_f \times (corPinV - pinV_0) \quad (n/cm^2)$$

With:

- $pinV_0$ and $radV_0$ the PIN diode and RADFET baseline voltages (V)
- Cda = 0.515 and Cdb = 0.481
- Cf = $1.7e^{11}$ n/cm^2

3.5. FSM and display

An FSM (Finite State Machine) tree has been built in order to include the Radiation Monitoring application in the DCS (Detector Control System) for monitoring in the ATLAS control room with the following structure (see figure 6):

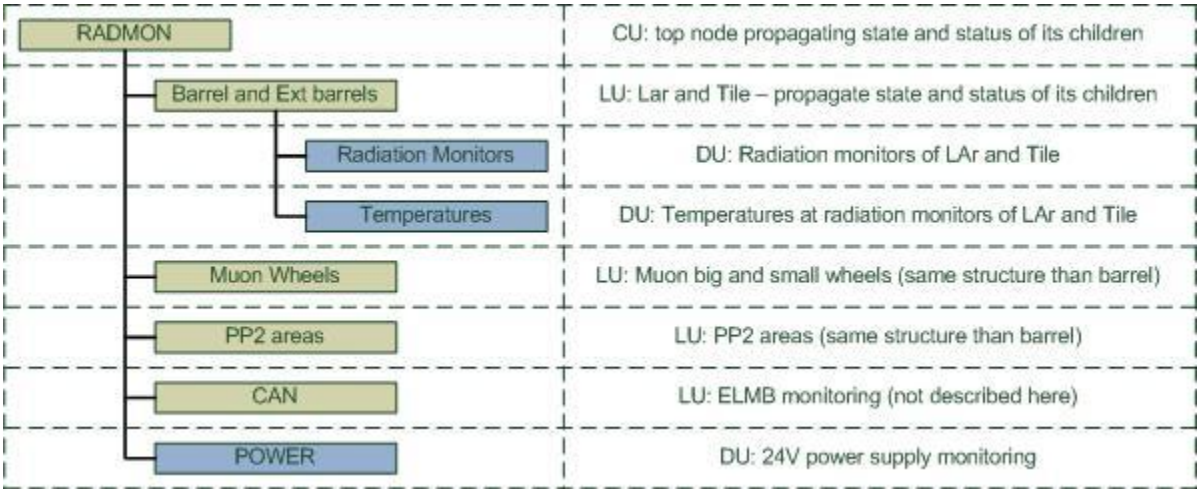


Figure 6: RadMon FSM hierarchy

The top node of the FSM propagates the state (“Ready”, “Not Ready”, “Unknown”) and the status (“OK”, “Warning”, “Error”, “Fatal”) of all its children. Then, the associated panel shows the status of the control manager which reads out the sensors periodically (see figure 7).

Figure 7: Radiation Monitoring FSM top page

This top node is referenced in the CIC (Common Infrastructure Control) FSM tree, which is then the entry point to go to the RadMon tree.

Then, by navigating down in the tree (see figures 8, 9, 10), the user will find a panel for each subsystem which displays the TID and fluence of each active sensor (excluded sensors are grey). A right click on an individual value pops up a plot with the archived data of the corresponding sensor.

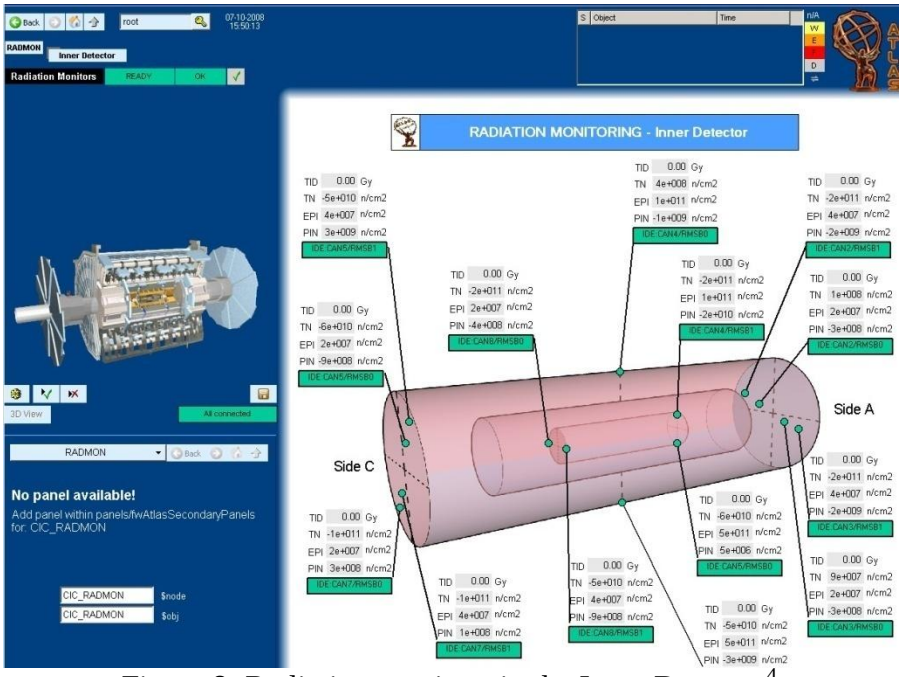


Figure 8: Radiation monitors in the Inner Detector[4]

[4] These sensors are not read out in the same way – see ID documentation for more information

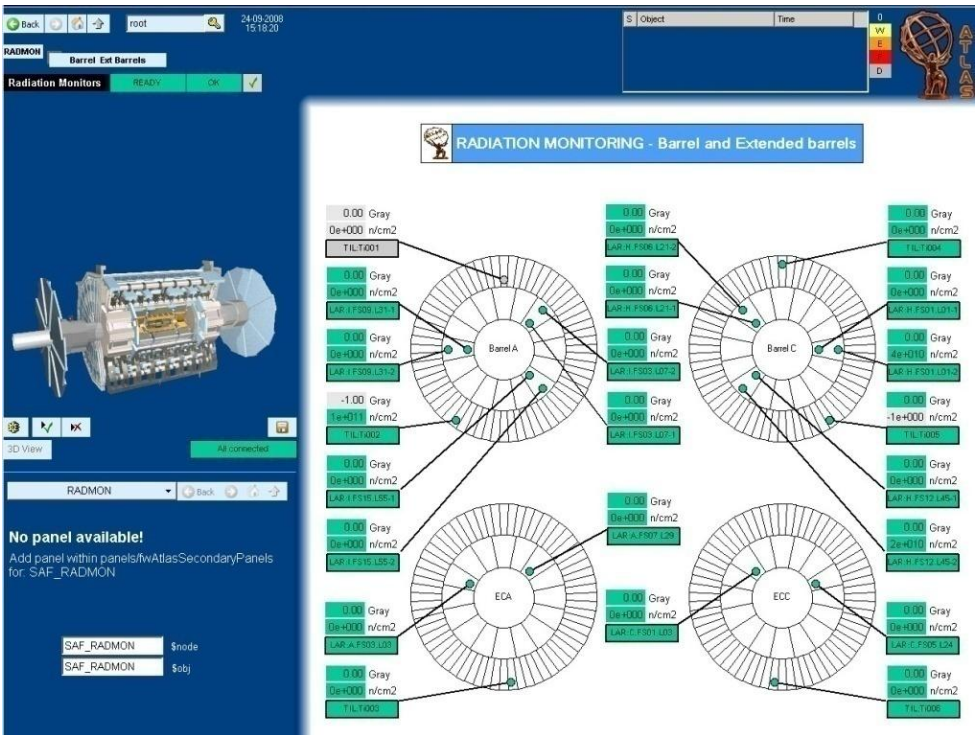


Figure 8: Radiation monitors of LAr and TileCal

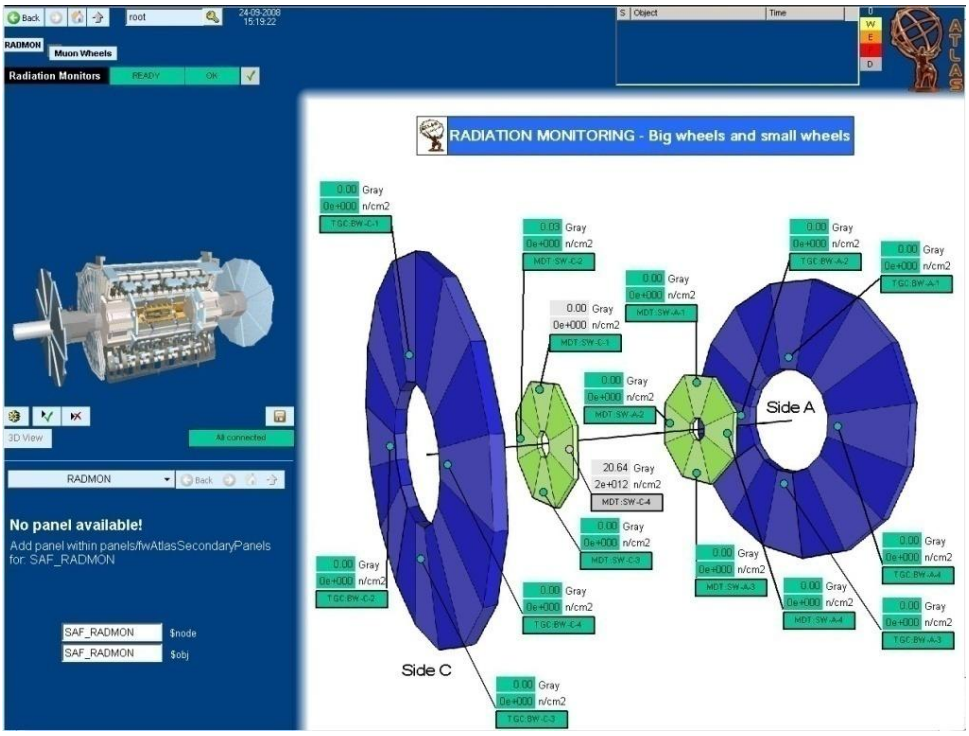


Figure 9: Radiation monitors of the Muon wheels

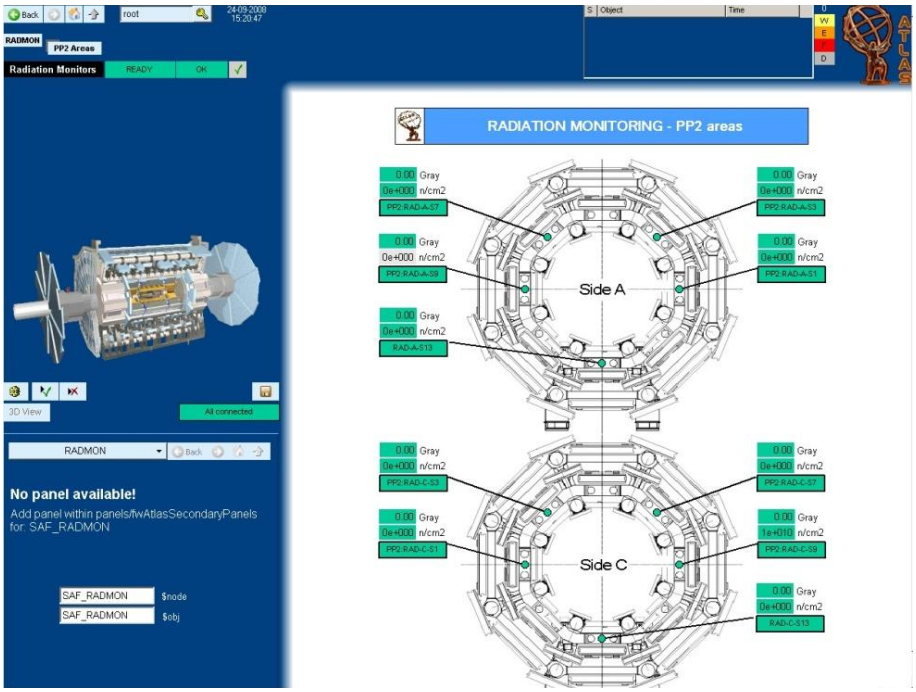


Figure 10: Radiation monitors of PP2 areas

Note that, for each subsystem, each panel is duplicated on a second device unit to display the temperature instead of the TID and fluence, as it is shown in the example, figure 11.

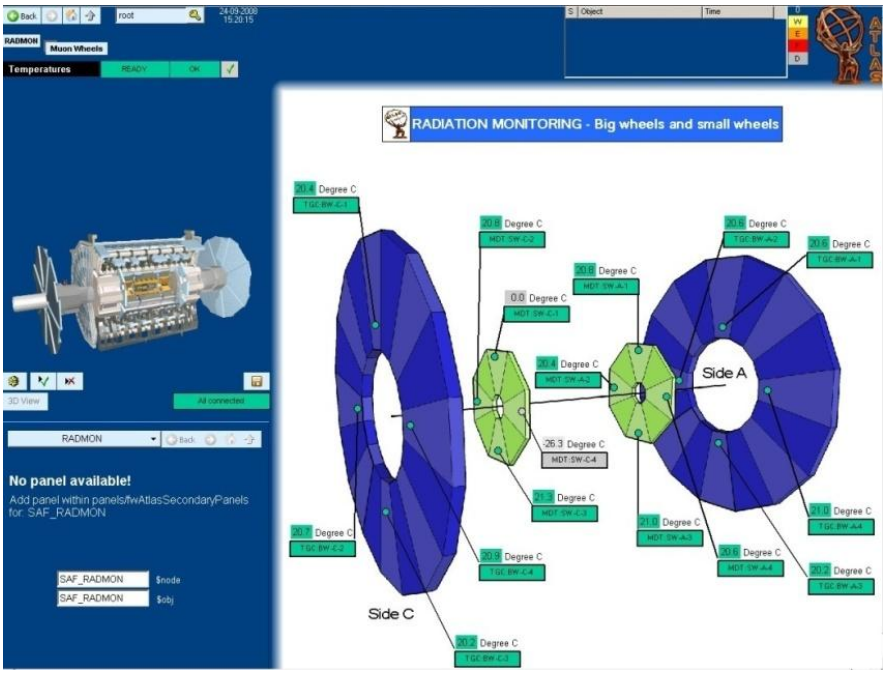


Figure 11: Temperatures at the radiation monitors level – Muon wheels

The hardware equipment is also supervised from the FSM:

- the functional parameters of the ELMB / CAN buses are supervised (figure 12)
- the 24V power can be switched on/off (figure 13).

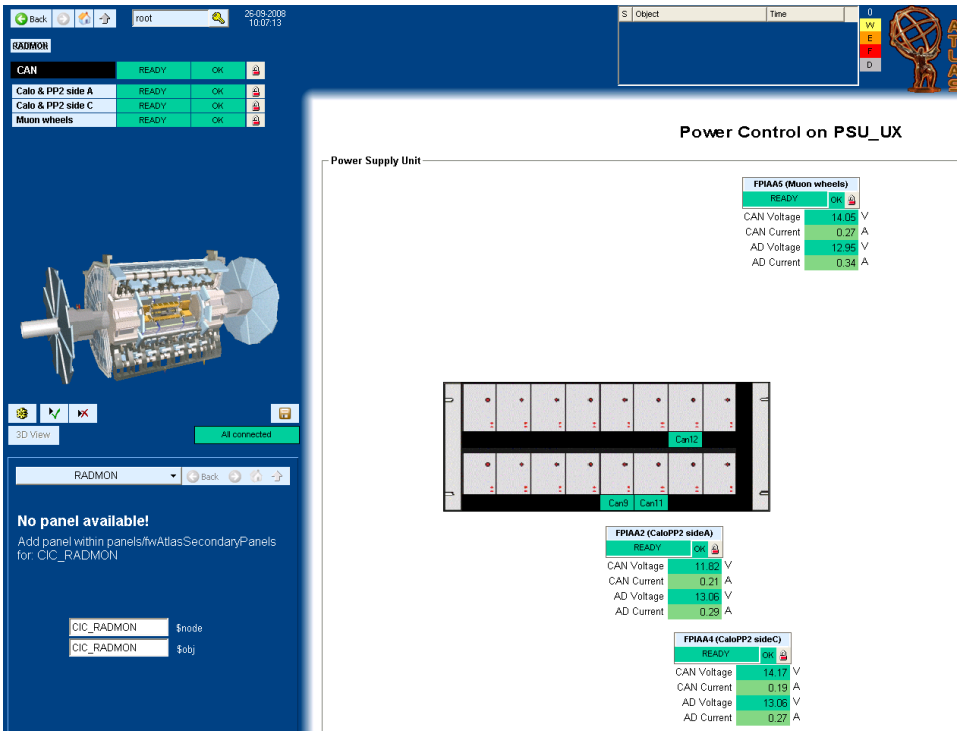


Figure 12: Parameters on the ELMB / CAN buses

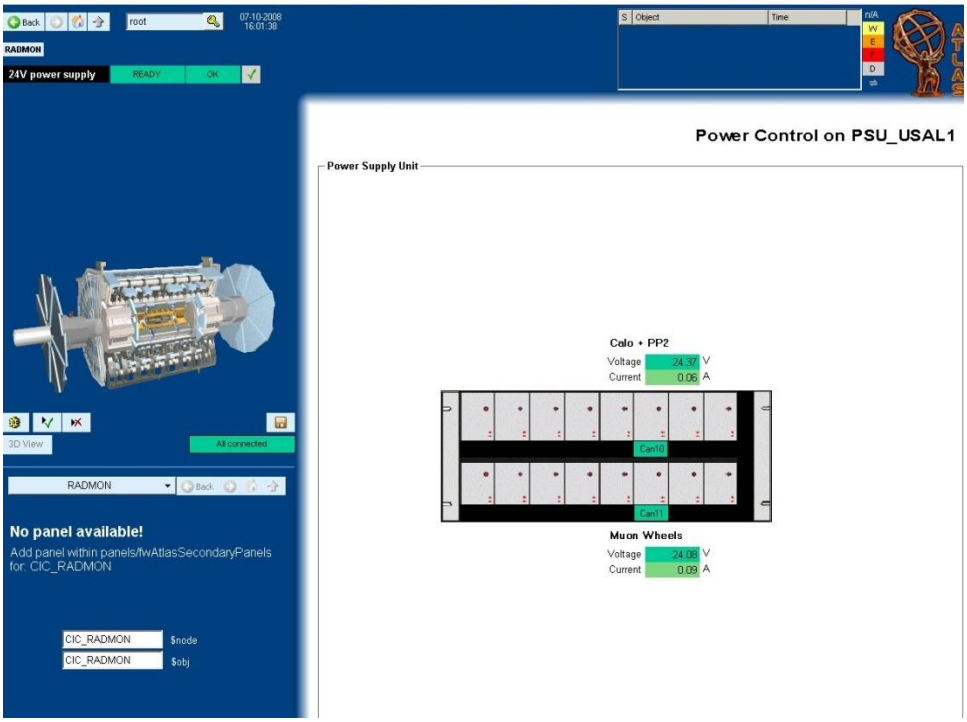


Figure 13: Powering of the sensors

3.6. FSM states and status

An alert handling mechanism has been setup on the radiation monitoring datapoints in order to show and propagate up in the FSM tree the state and the status of the application. It follows the rules below:

TID and NIEL	State	READY	R/O control manager is running
		NOT READY	R/O control manager stopped or crashed
	Status	OK	Valid Data - invalid bit = 0
		WARNING	Invalid Data - Invalid bit = 1
Temperature	State	READY	R/O control manager is running
		NOT READY	R/O control manager stopped or crashed
	Status	OK	10 °C < T < 40 °C
		WARNING	0 °C < T < 10 °C or 40 °C < T < 50 °C
		ERROR	T < 0 °C or T > 50 °C
Power	State	READY	Powering is OK
		NOT READY	Powering is partially and completely stopped
	Status	OK	No issue
		WARNING	Power failure
		ERROR	CAN / ELMB communication or power failure

4. Conclusion

The RadMon application is now integrated in the ATLAS Detector Control System with stable hardware and software settings. Some work has still to be done on the sensors, trying to repair the few malfunctioning sensors at the next ATLAS maintenance period. Then, some additional features will be added to the software:

- Come to an homogenous look and feel for the panels (all in 3D view)
- Add the control manager survey in FSM state propagation mechanism

Annex 1: Correction constants list

	PIN		RadFet	
Sensor	base	slope	base	slope
F5_E17_1	3.1611	0.0175	2.639	-0.0028
F5_E17_2	3.4147	0.0153	2.188	-0.0046
F5_E17_3	3.4035	0.019	2.6133	-0.0049
F5_E17_4	3.3377	0.0079	2.3705	-0.0022
F5_E17_5	3.3914	0.0097	2.2967	-0.0046
F5_E17_6	3.6844	0.0206	2.3171	-0.0026
F5_E17_7	3.8012	0.0199	2.1866	-0.0039
F5_E17_8	3.0307	0.0095	2.491	-0.0043
F5_E18_1	3.5556	0.031	2.5426	-0.0048
F5_E18_2	3.423	0.0285	2.3132	-0.0053
F5_E18_3	3.5957	0.0271	2.7309	-0.0039
F5_E18_4	3.3204	0.0281	2.4337	-0.0047
F5_E18_6	3.5817	0.035	2.1102	-0.0027
F5_E18_8	3.2495	0.0191	2.3848	-0.0039
F2E10_rad1	3.264	0.0224	2.435	-0.0034
F2E10_rad2	3.372	0.0225	2.497	-0.0058
F2E10_rad3	3.422	0.0157	2.108	-0.0025
F2E10_rad4	2.983	-	2.126	-0.0036
F2E10_rad5	3.261	0.0241	2.37	-0.0035
F4E10_rad1	3.458	0.0666	2.5	-0.004
F4E10_rad2	3.364	0.0206	2.228	-0.0024
F4E10_rad3	3.375	0.0183	2.485	-0.005
F4E10_rad4	3.31	0.023	2.68	-0.0051
F4E10_rad5	3.319	0.0192	2.317	-0.0031
F2E9_rad1	3.125	0.0211	2.517	-0.0082
F2E9_rad2	3.327	0.0194	2.099	-0.0027
F2E9_rad3	3.363	0.0208	2.538	-0.0041
F2E9_rad4	3.241	0.0182	2.521	-0.0041
F2E9_rad5	3.072	0.0146	2.125	-0.0025
F2E9_rad6	3.171	0.0137	2.574	-0.0047
F2E9_rad7	3.224	0.0137	2.685	-0.0044
F2E9_rad8	3.09	0.0145	2.345	-0.0044
F2E9_rad9	3.194	0.0188	2.157	-0.0025
F4E9_rad1	2.991	0.0131	2.412	-0.0016
F4E9_rad2	2.722	0.0621	2.175	-0.0023
F4E9_rad3	2.958	0.0088	2.259	-0.0041
F4E9_rad4	2.687	0.009	2.667	-0.0033

F4E9_rad5	2.884	0.0135	2.404	-0.0036
F4E9_rad6	2.835	0.0113	2.555	-0.0061
F4E9_rad7	2.855	0.0101	2.243	-0.0039
F4E9_rad8	2.983	0.0156	2.494	-0.0045
F4E9_rad9	2.908	0.0081	2.47	-0.0052

This table does not include the broken sensors and the one for which the correction constants are not yet calculated.

Annex 2: Software scripts

```
//--------------------------------------------------------------------------------------------
//Control script for Radiation Monitoring
//
//Sebastien Franz
//PH/ADO
//sebastien.franz@cern.ch
//--------------------------------------------------------------------------------------------


#uses "RadMonLib.ctl"
#uses "RadMonConstants.ctl"

main()
{
  initConstants();
  dyn_float newValues,previousValues,tempSlopes,baseVoltages,newValuesCor;
  dyn_string dsExceptionInfo;
  int pinValid,radfetValid;
  bool excluded;

  while(dynlen(dsExceptionInfo) == 0)
  {
    int maxLoop = 0;

    for(int i = 1; i<= 6; i++)
    {
      for(int k = 1; k < nSensor[i]; k++)
      {
        DebugN(targetDp[i],"Sensor",k);
        dpGet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".excluded", excluded);
        DebugN(excluded);
        if(!excluded)
        {
          getPreviousValues(i,k,previousValues,tempSlopes,baseVoltages);
          dsExceptionInfo = readSensors(i,k,newValues);
          newValuesCor = checkDataConsistency(previousValues,newValues,tempSlopes,baseVoltages,pinValid,radfetValid);
          if(pinValid == 2 || radfetValid == 2)
          {
            if(maxLoop<2)
            {
              DebugN("tentative " + (string)(maxLoop+1));
              k -= 1;
              maxLoop += 1;
            }
            else
            {
              DebugN("tentative " + (string)(maxLoop+1),"Stop loop");
              maxLoop = 0;
              computeData(i,k,newValuesCor,baseVoltages,pinValid,radfetValid);
            }
          }
          else
          {
            DebugN("R/O OK");
            computeData(i,k,newValuesCor,baseVoltages,pinValid,radfetValid);
          }
        }
        else
          DebugN(targetDp[i] + " Sensor " + k + " is excluded");
      }
    }
    delay(1800);
  }
}
```

```
//------------------------------------------------------------------------------------------
//Library of functions for radiation monitoring
//
//Sebastien Franz
//PH/ADO
//sebastien.franz@cern.ch
//------------------------------------------------------------------------------------------

#uses "RadMonConstants.ctl"

//Init constants values

void initConstants()
{
  m_sSystem = getSystemName();
  pinList =     makeDynInt(32,36,40,44,0,4,48,52,56,60,16,20);
  radfetList = makeDynInt(33,37,41,45,1,5,49,53,57,61,17,21);
  currentList = makeDynInt(34,38,42,46,2,6,50,54,58,62,18,22);
  ntcList = makeDynInt(35,39,43,47,3,7,51,55,59,63,19,23);
  pinEnableList = makeDynInt(16,18,20,22,24,26,0,2,4,6,8,10);
  radfetEnableList = makeDynInt(17,19,21,23,25,27,1,3,5,7,9,11);
  targetELMB = makeDynString("FPIAA2/ELMB_9","FPIAA4/ELMB_9","FPIAA2/ELMB_10","FPIAA4/ELMB_10",
                      "FPIAA5/ELMB_17","FPIAA5/ELMB_18");
  targetDp = makeDynString("FPIAA2_ELMB9","FPIAA4_ELMB9","FPIAA2_ELMB10","FPIAA4_ELMB10",
                     "FPIAA5_ELMB17","FPIAA5_ELMB18");
  nSensor = makeDynInt(11,11,5,5,8,9);
}



// Read out of a single sensor-----------------------------------------------------------

dyn_string readSensors(int i, int k, dyn_float &newValues)
{
  //variables
  float ntcValue, pinValue, radfetValue, currentValue;
  dyn_string dsExceptionInfo;

  //DAC values
  int switchEnable = 4095;
  int pinEnable; //default 195;
  int radfetEnable; // default 200;
  int switchDisable = 0;
  int pinDisable = 0;
  int radfetDisable = 0;

  //Enable switch
  dpSetWait(m_sSystem + "ELMB/" + targetELMB[i] + "/AO/ao_31.value", switchEnable);
  dpSetWait(m_sSystem + "ELMB/" + targetELMB[i] + "/AO/ao_15.value", switchEnable);

  //Read NTC
  fwElmb_elementSQ(m_sSystem + "ELMB/" + targetELMB[i] + "/AI/aisdo_" + ntcList[k] + ".rawValue", 2, ntcValue, dsExceptionInfo);
  if (dynlen(dsExceptionInfo) > 0)
    DebugN(dsExceptionInfo);
  dpGet(m_sSystem + "ELMB/" + targetELMB[i] + "/AI/aisdo_" + ntcList[k] + ".value", ntcValue);
  ntcValue = 1./297.+1./3435.*(log((ntcValue/2.5)/10000.));
  ntcValue = 1./ntcValue - 273.15;
  newValues[1] =ntcValue;
  dpSet(m_sSystem + targetDp[i] + "_radmon_" + (string)k + ".ntc",ntcValue);
  DebugN(ntcValue);

  //Read Pin diode
  dpGet(m_sSystem + targetDp[i] + "_radmon_" + (string)k + ".pinEnable",pinEnable);
  dpSetWait(m_sSystem + "ELMB/" + targetELMB[i] + "/AO/ao_" + pinEnableList[k] + ".value", pinEnable);
  delay(0,100);
  fwElmb_elementSQ(m_sSystem + "ELMB/" + targetELMB[i] + "/AI/aisdo_" + pinList[k] + ".rawValue:_online.._value", 2, pinValue,
dsExceptionInfo);
  if (dynlen(dsExceptionInfo) > 0)
    DebugN(dsExceptionInfo);
```

```
  dpGet(m_sSystem + "ELMB/" + targetELMB[i] + "/AI/aisdo_" + pinList[k] + ".value", pinValue);
  fwElmb_elementSQ(m_sSystem + "ELMB/" + targetELMB[i] + "/AI/aisdo_" + currentList[k] + ".rawValue:_online..._value", 2, currentValue,
dsExceptionInfo);
  if (dynlen(dsExceptionInfo) > 0)
    DebugN(dsExceptionInfo);
  dpGet(m_sSystem + "ELMB/" + targetELMB[i] + "/AI/aisdo_" + currentList[k] + ".value",currentValue);
  dpSetWait(m_sSystem + "ELMB/" + targetELMB[i] + "/AO/ao_" + pinEnableList[k] + ".value", pinDisable);
  pinValue = pinValue/1000000*11;
  currentValue = currentValue/100;
  newValues[2] = pinValue;
  newValues[3] = currentValue;
  dpSet(m_sSystem + targetDp[i] + "_radmon_" + (string)k + ".pinV",pinValue);
  dpSet(m_sSystem + targetDp[i] + "_radmon_" + (string)k + ".pinI",currentValue);
  DebugN(pinValue,currentValue);

  //Read RadFet
  dpGet(m_sSystem + targetDp[i] + "_radmon_" + (string)k + ".radfetEnable",radfetEnable);
  dpSetWait(m_sSystem + "ELMB/" + targetELMB[i] + "/AO/ao_" + radfetEnableList[k] + ".value", radfetEnable);
  delay(1);
  fwElmb_elementSQ(m_sSystem + "ELMB/" + targetELMB[i] + "/AI/aisdo_" + radfetList[k] + ".rawValue:_online..._value", 2, radfetValue,
dsExceptionInfo);
  if (dynlen(dsExceptionInfo) > 0)
    DebugN(dsExceptionInfo);
  dpGet(m_sSystem + "ELMB/" + targetELMB[i] + "/AI/aisdo_" + radfetList[k] + ".value",radfetValue);
  fwElmb_elementSQ(m_sSystem + "ELMB/" + targetELMB[i] + "/AI/aisdo_" + currentList[k] + ".rawValue:_online..._value", 2, currentValue,
dsExceptionInfo);
  if (dynlen(dsExceptionInfo) > 0)
    DebugN(dsExceptionInfo);
  dpGet(m_sSystem + "ELMB/" + targetELMB[i] + "/AI/aisdo_" + currentList[k] + ".value",currentValue);
  dpSetWait(m_sSystem + "ELMB/" + targetELMB[i] + "/AO/ao_" + radfetEnableList[k] + ".value", radfetDisable);
    radfetValue = radfetValue/1000000*11;
    currentValue = currentValue/100;
    newValues[4] = radfetValue;
  newValues[5] = currentValue;
  dpSet(m_sSystem + targetDp[i] + "_radmon_" + (string)k + ".radfetV",radfetValue);
  dpSet(m_sSystem + targetDp[i] + "_radmon_" + (string)k + ".radfetI",currentValue);
  DebugN(radfetValue,currentValue);


  //DAC disable
  dpSetWait(m_sSystem + "ELMB/" + targetELMB[i] + "/AO/ao_31.value", switchDisable);
  dpSetWait(m_sSystem + "ELMB/" + targetELMB[i] + "/AO/ao_15.value", switchDisable);

  return dsExceptionInfo;
}

//Check Data consistency----------------------------------------------------------------------------------------------------------------

dyn_float checkDataConsistency(dyn_float previousValues,dyn_float newValues,dyn_float slopes, dyn_float baseVoltages, int &test1, int
&test2)
{
  float newPinV,prevPinV,newRadfetV,prevRadfetV, diffBase, diffPrevious;
  dyn_float newValuesCor = newValues;

  //Fluence
  newPinV = newValues[2]-slopes[1]*(newValues[1]-20.0);
  prevPinV = previousValues[2]-slopes[1]*(previousValues[1]-20.0);
  diffBase = newPinV - baseVoltages[1];
  diffPrevious = newPinV - prevPinV;
  if(diffBase < -0.5 || diffBase > 16.0)
    test1 = 2;
  else if(diffBase >= -0.5 && diffBase <= 0.05)
    test1 = 1;
  else if(diffBase > 0.05)
  {
    if(diffPrevious < -0.5)
      test1 = 2;
    else if(diffPrevious >= -0.5 && diffPrevious <= 0.0)
    {
      test1 = 0;
```

```
      newValuesCor[2] = prevPinV;
    }
    else if(diffPrevious > 0.0)
    {
      test1 = 0;
      newValuesCor[2] = newPinV;
    }
  }

  //TID
  newRadfetV = newValues[4]-slopes[2]*(newValues[1]-20.0);
  prevRadfetV = previousValues[4]-slopes[2]*(previousValues[1]-20.0);
  diffBase = newRadfetV - baseVoltages[2];
  diffPrevious = newRadfetV - prevRadfetV;
  if(diffBase < -0.5 || diffBase > 20.0)
    test2 = 2;
  else if(diffBase >= -0.5 && diffBase <= 0.05)
    test2 = 1;
  else if(diffBase > 0.05)
  {
    if(diffPrevious < -0.5)
      test2 = 2;
    else if(diffPrevious >= -0.5 && diffPrevious <= 0.0)
    {
      test2 = 0;
      newValuesCor[4] = prevRadfetV;
    }
    else if(diffPrevious > 0.0)
    {
      test2 = 0;
      newValuesCor[4] = newRadfetV;
    }
  }
  DebugN("Data test",test1,test2);
  return newValuesCor;
}

//Get values of the previous readout----------------------------------------------------

void getPreviousValues(int i,int k, dyn_float &prevVal, dyn_float &slopes, dyn_float &baselines)
{
  dpGet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".temperature.value", prevVal[1]);
  dpGet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".fluence.voltage", prevVal[2]);
  dpGet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".fluence.current", prevVal[3]);
  dpGet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".dose.voltage", prevVal[4]);
  dpGet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".dose.current", prevVal[5]);
  dpGet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".fluence.correction.baseline",baselines[1]);
  dpGet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".fluence.correction.tempSlope",slopes[1]);
  dpGet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".dose.correction.baseline",baselines[2]);
  dpGet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".dose.correction.tempSlope",slopes[2]);
}

//Compute Data--------------------------------------------------------------------------

void computeData(int i, int k, dyn_float newValuesCor,dyn_float baseVoltages,int pinValid,int radfetValid)
{
  //Constants for Dose/fluence calculation
  float Cf = 1.7e+11; // n/cm2/V
  float Cda = 0.515; float Cdb = 0.481;
  float diffFluence, diffDose,newDose, newFluence;
  //Calculate Dose and fluence
  diffFluence = newValuesCor[2] - baseVoltages[1]; DebugN(diffFluence);
  diffDose = newValuesCor[4] - baseVoltages[2]; DebugN(diffDose);
  newFluence = Cf*(newValuesCor[2] - baseVoltages[1]);
  newDose = exp(1/Cdb*(log(newValuesCor[4] - baseVoltages[2])-log(Cda)));

  switch(pinValid)
  {
    case 2:
      DebugN("PIN diode excluded - no valid data");
```

```
    dpSet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".fluence.invalid",true);
    break;
   case 1:
    newFluence = 0.0;
    dpSet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".fluence.value",newFluence);
    dpSet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".fluence.invalid",false);
    DebugN("Fluence = " + newFluence);
    break;
   case 0:
    dpSet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".fluence.value",newFluence);
    dpSet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".fluence.invalid",false);
    DebugN("Fluence = " + newFluence);
    break;
  }

  switch(radfetValid)
  {
   case 2:
    DebugN("RadFet excluded - no valid data");
    dpSet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".dose.invalid",true);
    break;
   case 1:
    newDose = 0.0;
    dpSet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".dose.value",newDose);
    dpSet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".dose.invalid",false);
    DebugN("Dose = " + newDose);
    break;
   case 0:
    dpSet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".dose.value",newDose);
    dpSet(m_sSystem + targetDp[i] + "_radSensor_" + (string)k + ".dose.invalid",false);
    DebugN("Dose = " + newDose);
    break;
  }
}
```

```
//--------------------------------------------------------------------------------------------
//FSM device unit script for Radiation Monitoring
//
//Sebastien Franz
//PH/ADO
//sebastien.franz@cern.ch
//--------------------------------------------------------------------------------------------

RadMonSensors_DU_initialize(string domain, string device)
{
  string targetELMB = "";
  dyn_string dplistDose,dplistFluence,dplistTemp;
  dyn_string dplistRad;
  string nodeName = fwFsmAtlas_getNodeDPEName (domain, device);

  if(domain == "SAF_RADMON_MUONWHEELS")
  {targetELMB = "FPIAA5";}
  else if(domain == "SAF_RADMON_BARREL")
  {targetELMB = "ELMB9";}
  else if(domain == "SAF_RADMON_PP2")
  {targetELMB = "ELMB10";}

  dyn_string dplist = dpNames("*" + targetELMB + "*","RadMonSensors");

  string tmp = strsplit(device,"_")[4];

  for(int i = 1; i<= dynlen(dplist); i++)
  {
    if(dpExists(dplist[i] + ".dose.value:_alert_hdl._act_state_color"))
      dplistDose[i] = dplist[i] + ".dose.value:_alert_hdl._act_state_color";
    else
      DebugN(dplist[i] + ".dose.value:_alert_hdl._act_state_color" + " does not exist");

    if(dpExists(dplist[i] + ".fluence.value:_alert_hdl._act_state_color"))
      dplistFluence[i] = dplist[i] + ".fluence.value:_alert_hdl._act_state_color";
    else
      DebugN(dplist[i] + ".fluence.value:_alert_hdl._act_state_color" + " does not exist");

    if(dpExists(dplist[i] + ".temperature.value:_alert_hdl._act_state_color"))
      dplistTemp[i] = dplist[i] + ".temperature.value:_alert_hdl._act_state_color";
    else
      DebugN(dplist[i] + ".temperature.value:_alert_hdl._act_state_color" + " does not exist");
  }

  dplistRad = dplistDose;
  dynAppend(dplistRad,dplistFluence);

  if(tmp == "RAD")
    dplist = dplistRad;
  else if(tmp == "TEMP")
    dplist = dplistTemp;
  else
    DebugN("RadMon sensors INIT - No DPE match");

  dynAppend(dplist,nodeName);

  dpConnect("RadMonSensors_DU_callback", dplist);
}

RadMonSensors_DU_callback(dyn_string dplist, dyn_string values)
{
  //Get domain and device name
  string domain,device,nodeName;

  nodeName = dplist[dynlen(dplist)];
  fwFsmAtlas_getNodeNameComponents(nodeName,domain,device);

  //Set  Status
  string targetStatus = "OK";
```

```
  for(int i = 1; i < dynlen(values); i++)
    if(strpos(values[i], "Fatal")>-1) targetStatus = "FATAL";

  if(targetStatus != "FATAL")
    for(int i = 1; i < dynlen(values); i++)
      if(strpos(values[i], "Error")>-1) targetStatus = "ERROR";

  if(targetStatus != "FATAL" && targetStatus != "ERROR")
    for(int i = 1; i < dynlen(values); i++)
      if(strpos(values[i], "Warn")>-1) targetStatus = "WARNING";

  //Set State
  string state = "READY";


  //Put new values online
  fwDU_setState(domain, device, state);
  fwFsmAtlas_setStatus(domain, device, targetStatus);
}



RadMonSensors_DU_valueChanged( string domain, string device, string &fwState )
{
}


RadMonSensors_DU_doCommand(string domain, string device, string command)
{
}
```