Universitat Oberta de Catalunya

uoc.edu

Script AcopRFcheck.py

Oriol Serres Boldú
Màster Universitari en Enginyeria de Telecomunicacions
Àrea de Tecnologia d'Antenes

Nom Consultor/a: Dr. Jaume Anguera i Dra. Aurora Andújar
Nom Professor/a responsable de l'assignatura: Dr. Germán Cobo

Gener 2021

Agraïments:

A la família i amics.
Al grup d'acceleradors del sincrotró ALBA.
Al tutor/a: Jaume Anguera.
Aurora Andújar

Aquesta obra està subjecta a una llicència de

Reconeixement-NoComercial-SenseObraDerivada 3.0
Espanya de Creative Commons

FITXA DEL TREBALL FINAL

Títol del treball:	*Script AcopRFcheck.py*
Nom de l'autor:	*Oriol Serres Boldú*
Nom del consultor/a:	*Dr. Jaume Anguera i Dra. Aurora Andújar*
Nom del PRA:	*Dr. Germán Cobo*
Data de lliurament (mm/aaaa):	*Gener 2021*
Titulació o programa:	*Màster universitari en Enginyeria de Telecomunicacions*
Àrea del Treball Final:	*Antenes*
Idioma del treball:	*Català*
Paraules clau	*Accelerador de partícules ALBA, Física de partícules, programació en Python*

Resum del Treball (màxim 250 paraules): *Amb la finalitat, context d'aplicació, metodologia, resultats i conclusions del treball*

El principal objectiu de la realització d'aquest script sorgeix de la necessitat de revisar els múltiples paràmetres de configuració del sistema de radiofreqüència del sincrotró ALBA. Degut a la importància que té aquest sistema dins dels acceleradors de partícules es requeria una eina capaç de revisar tots aquests paràmetres, a part de realitzar tots els càlculs teòrics de potència *forward*, potència reflectida, fase síncrona i molts d'altres que seran els que quan es comparin amb els valors llegits del sistema de radiofreqüència ens verificaran que està llest per funcionar amb total normalitat.

Pel desenvolupament d'aquest *script* podem diferenciar dues parts: la primera part és el desenvolupament del marc teòric en què es basarà aquest, partint de la base física aplicada a la radiofreqüència dels acceleradors de partícules. La segona que serà la traducció d'aquest marc teòric al llenguatge de programació creant les funcions adequades.

Aquest *script* ha estat desenvolupat en Python, que és el llenguatge en que està desenvolupat el sistema de control del accelerador i que per tant, a part del desenvolupament matemàtic teòric, ens permetrà la comunicació via *device* amb els diferents elements que integren el sistema.

Així doncs, després de la realització del treball he assolit els objectius aconseguint aquesta eina que ens verifica el sistema i que en cas de que algun paràmetre estigui mal configurat ens retorna l'opció a modificar.

Abstract (in English, 250 words or less):

The main goal of this script arises from the need to review the multiple settings parameters of the ALBA synchrotron Radio Frequency system. Due to the importance of this system within particle accelerators, a tool capable of checking all these parameters was required, apart of performing all the theoretical calculations of forward power, reflected power, synchronous phase and many others that will be the ones that when compared to the read values of the Radio Frequency system they will check that it is ready to operate.

For the development of this script we have two parts: the first part is the development of the theoretical framework on which it will be based, starting from the physics applied to the Radio Frequency of particle accelerators. The second will be the translation of this theoretical framework into the programming language by creating the appropriate functions.

This script has been developed in Python. This is the language in which the accelerator control system has been developed and which, apart from the theoretical mathematical development, will allow us to communicate via device with the different elements that set up the system.

So, after the work has been done, I have achieved the objectives by providing us with a tool that verifies the system and that in case any parameter is misconfigured gives us back the option to modify.

Índex

Llista de figures

* La majoria de les figures son pròpies de les diferents divisions d'ALBA o bé de comunitats lliures en les que ALBA es membre, per això no estan citades.

1. Introducció

1.1 Context i justificació del treball

L'objectiu d'aquest treball final de màster, sorgeix de la necessitat que hi ha dins l'accelerador de partícules d'ALBA, de verificar el sistema de radiofreqüència abans de que es deixi l'accelerador llest per als usuaris. Degut a la importància d'aquest sistema, és necessari assegurar-ne el seu correcte funcionament per evitar possibles fallades que perjudiquin els usuaris.

Actualment la revisió és realitza de forma manual per l'equip de radiofreqüència, però degut al gran nombre de paràmetres i al temps límit de que es disposa per fer la posada a punt de tot el sistema, es converteix en una tasca carregosa, per tant automatitzar-ho mitjançant un *script* és una prioritat.

L'aportació d'aquest *script* farà que no calgui que sigui algú de l'equip de radiofreqüència que hagi de fer aquest tasca sinó que el propi operador que hi hagi a la sala de control pot acabar d'ajustar el sistema, a part de l'obvi estalvi de temps que aportarà també.

1.2 Objectius del treball

Els objectius del treball són la descripció dels següents conceptes:

- Explicació de la llum sincrotró.
- Parts de l'accelerador de partícules.
- Parts sistema de radiofreqüència en un accelerador de partícules.
    - Conceptes bàsics.
    - Distribució del sistema de RF.
    - Conceptes específics de RF aplicats als acceleradors de partícules.
- Sistema de control de l'accelerador.
- Explicació detallada de les parts de l'*script* AcopRFcheck.py encarregat de la revisió de la RF.

1.3 Enfocament i mètode seguit

L'enfocament del treball està dividit en parts diferenciades:

- Una primer part on es farà tot l'estudi teòric dels conceptes aplicats a la radiofreqüència.
- Una segona part on es traslladaran tots aquests conceptes, programant-los adequadament.
- Una tercera part on es revisaran que tots els resultats obtinguts són adequats en relació amb els resultats esperats.

1.4 Planificació del treball

La planificació ha sigut en un primer lloc l'adquisició dels coneixements teòrics del que es va dur a terme i paral·lelament, s'anava programant l'*script* en Python. A mesura que s'anaven creant les funcions es comentava amb l'equip de RF quin havia de ser l'*output* que havia de donar el programa en cas d'error. En un segon lloc, a mesura que es construïen aquestes funcions es comprovava que els càlculs teòrics fossin raonables i es comparaven amb els reals. Finalment, es va fer un test general en que es van tornar a comprovar tots els càlculs i els atributs que s'havien de revisar.

1.5 Breu sumari de productes obtinguts

Els productes obtinguts son:

- Estudi del marc teòric associat a la radiofreqüència dels acceleradors de partícules.
- Eina de revisió de la radiofreqüència.

1.6 Breu descripció dels altres capítols de la memòria

La memòria d'aquest projecte es divideix en 4 capítols:

- Primer capítol on es defineix el context, la justificació i els objectius del treball.
- Segon capítol amb la descripció del sincrotró.
- Tercer capítol amb la descripció del sistema de radiofreqüència.
- Quart capítol on es descriu el sistema de control i l'*script* realitzat.
- Cinquè capítol on es descriuen les conclusions.

2. Descripció Sincrotró

Un Sincrotró és una instal·lació que està formada per un complex d'acceleradors de partícules circular de 269 m de longitud, que genera electrons, els accelera fins a una velocitat relativista, amb una energia de 3GeV, i els emmagatzema. A part dels acceleradors hi ha les *beamlines* (línies de llum), que són els laboratoris on es realitzen els experiments a partir de la interacció de la llum sincrotró amb les mostres.

Hi ha una cinquantena de sincrotrons a tot el món, una vintena dels quals es troben a Europa. (Fig 1) Situat a Cerdanyola del Vallès, ALBA és l'únic sincrotró de l'Estat Espanyol.

Fig 1. Mapa Sincrotrons a Europa . Font: leaps

2.1 Llum sincrotró

La llum sincrotró, és un tipus de radiació que es produeix degut al fenomen en que si es canvia la trajectòria dels electrons, aquests generen fotons. Aquest canvi de direcció es realitza mitjançant la interacció dels electrons amb un camp magnètic. (Fig 2)

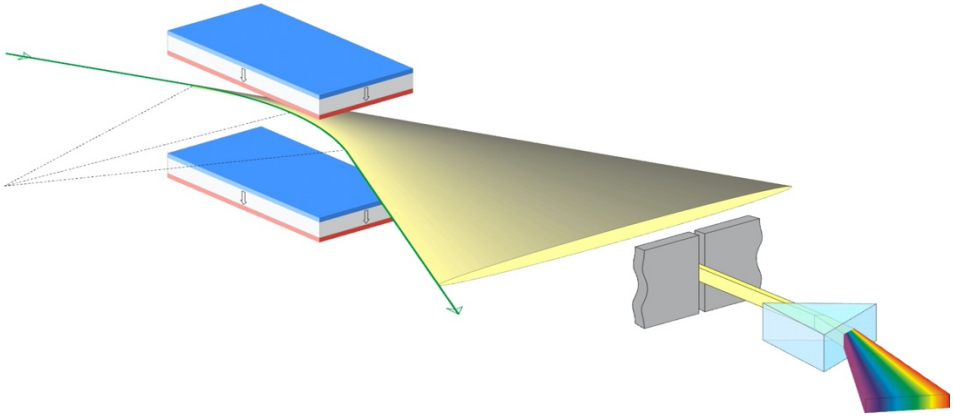


Fig 2. Emissió de llum quan una partícula amb càrrega elèctrica canvia la trajectòria

Com es veurà més endavant, el sincrotró té forma circular (Fig 4), de manera que aquest feix d'electrons que tenim movent-se al llarg de la circumferència genera fotons constantment. Així doncs, utilitzem aquesta llum que ens permet estudiar la matèria a escales atòmiques i moleculars fent experiments tan diversos com comprovar l'eficàcia d'un medicament, cosmètica, estudiar nous material per augmentar la memòria dels dispositius electrònics, o en el món de l'art, per veure, les capes de pintura, mineralogia, estudiar propietats de materials, entre d'altres coses.

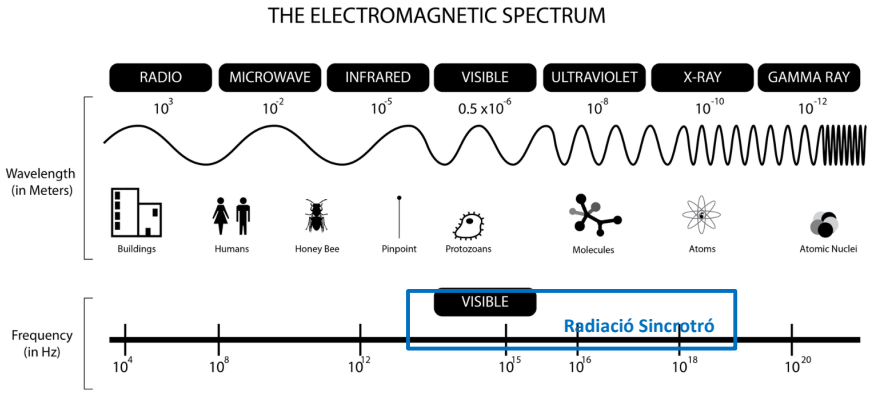


Fig 3. Espectre electromagnètic

2.1.1 Característiques de la llum sincrotró

- Ampli espectre d'energia: la llum sincrotró emet un rang d'energies que des de *infrared* fins als *hard X-Rays*
- Molt brillant: la llum sincrotró és extremadament brillant (100.000 cops més intensa que els tubs de raig X convencionals) i altament col·limada.
- Molt estable: a Alba l'òrbita del feix d'electrons es mou dins la micra, per tant tenim un feix de llum que és més intens i uniforme.
- Sintonitzable: podem obtenir un feix intens de qualsevol de les longituds d'ona que seleccionem.
- Altament polaritzable: els sincrotrons emeten llum altament polaritzada, la qual pot ser lineal, circular o el·líptica, les quals s'utilitzen depenent del tipus d'experiment a realitzar.
- Emissió de polsos molt curts: els polsos emesos són inferiors al nano segon.

2.2 Parts Sincrotró

El sincrotró ALBA està dividit en dues parts: els tres acceleradors on es genera la llum sincrotró i les *beamlines* on s'utilitza aquesta llum i es fan els experiments.

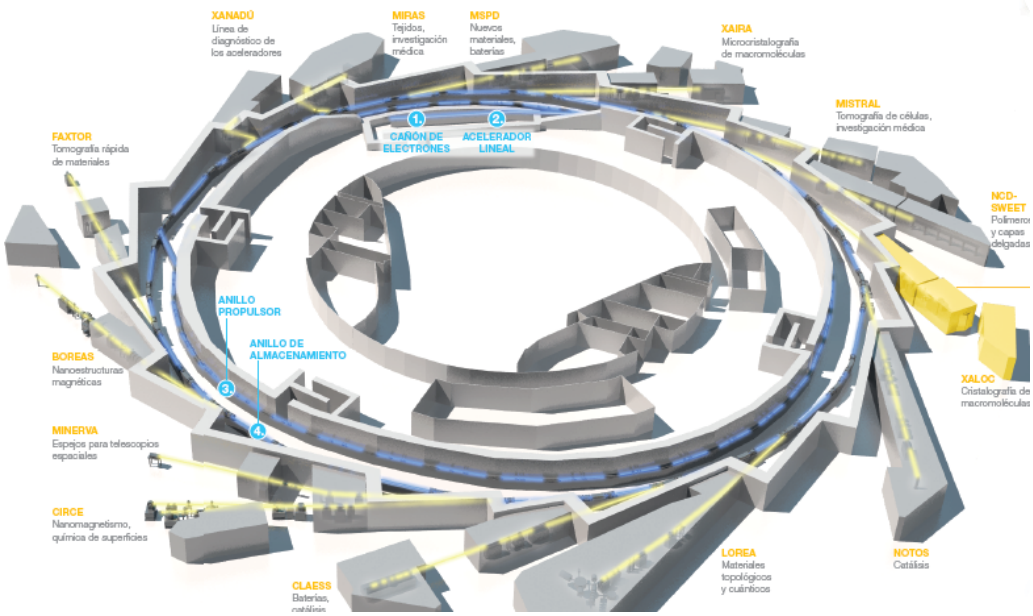


Fig 4. Esquema dels acceleradors

2.2.1 Acceleradors

El primer accelerador és un accelerador lineal (*LINAC)* que mesura aproximadament uns 10 metres, a l'inici del qual tenim un filament de tungstè (càtode) que escalfem a 1000 graus centígrads per extreure'n els electrons, el quals a continuació s'empaqueten (*pre-buncher i buncher*) i s'acceleren mitjançant dos cavitats de radiofreqüència a 3GHz, fins arribar al 99,98 % de la velocitat relativista de la llum amb una energia de 100 MeV.

Fig 5. Accelerador LINAC

El final del *Linac* està unit a l'accelerador propulsor (*Booster*), que és un accelerador de tipus circular de 249.6 m de longitud. En aquest accelerador hi ha electroimants que el que fan és focalitzar el feix d'electrons al llarg de les voltes, per tal de mantenir-los junts. Ara bé, el camp magnètic que produeixen aquests imants està sincronitzat amb el camp elèctric que induïm al feix d'electrons a través de la planta de radiofreqüència, de manera que quan entren els electrons al *Booster* tenen una energia de 100 MeV i una velocitat de 99,98 % en comparació amb la relativista, de tal manera que els imants utilitzaran un corrent determinat sincronitzat amb el voltatge de *RF*. A mesura que es van accelerant els electrons, va augmentant el corrent dels imants conjuntament amb l'augment de voltatge induït per la *RF.* Al cap de 160 ms extraiem aquests electrons que tenen una velocitat relativista (99,99998 %) i una energia de 3GeV.

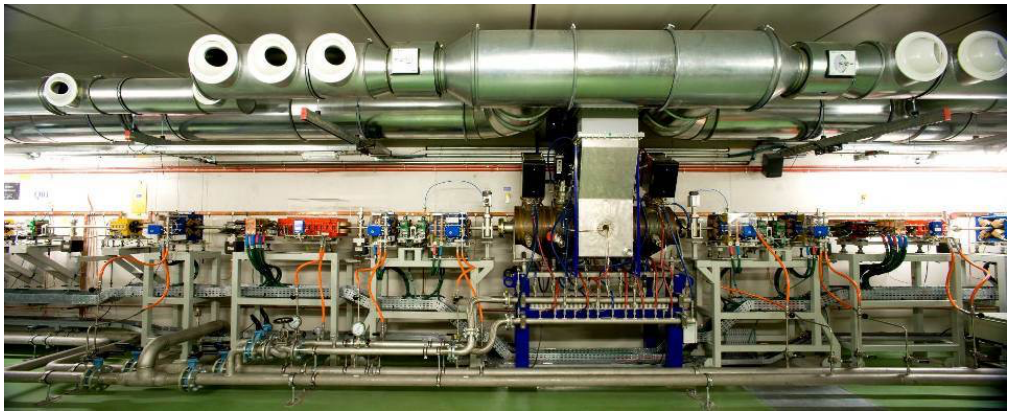


Fig 6. Accelerador Booster

Per últim, hi ha l'anell d'emmagatzematge (*Storage Ring*), que és l'anell de 268.8 m de longitud que està dividit en 16 sectors que al seu temps estan dividits en 2 cel·les per sector i serien conjunts d'elements de l'accelerador formats per imants de diferents tipus que es van replicant i aporten simetria de seccions rectes i seccions corbadores al llarg de l'anell. En aquest anell és on després d'accelerar els electrons i que tinguin l'energia desitjada els emmagatzemem. Degut a la generació de fotons quan corben la seva trajectòria, els electrons tenen una pèrdua d'energia i de velocitat. Per tal de compensar aquestes pèrdues i aconseguir que la llum emesa tingui les propietats adequades, s'utilitzen plantes de radiofreqüència que generaran el camp elèctric necessari per restituir aquesta energia perduda.

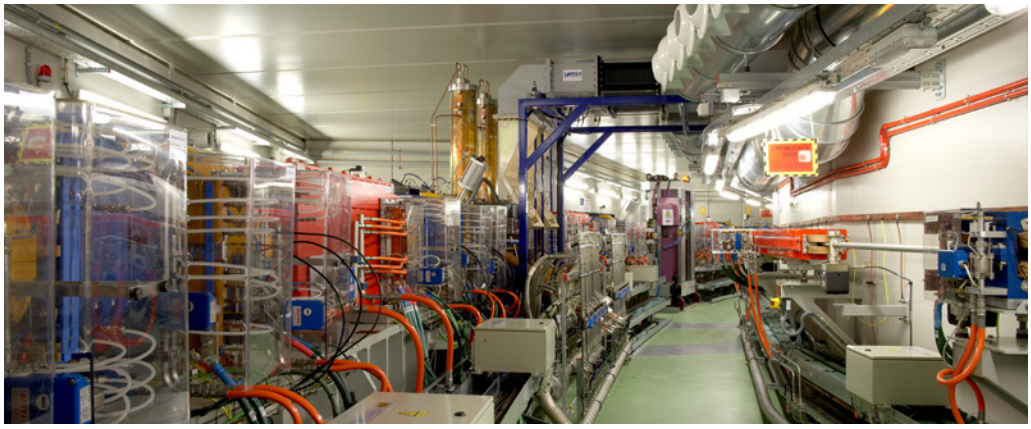


Fig 7. Storage Ring

Tots aquests acceleradors tenen diversos subsistemes en comú, que són essencials per al seu funcionament.

- ***Magnets**:* tenim 696 electroimants en el conjunt dels 3 acceleradors, els qual són de diversos tipus i per a diferents finalitats:
  - *Bending magnets o dipole magnets:* són uns imants bipolars que s'encarreguen de corbar la trajectòria del electrons al llarg dels acceleradors. A l'*Storage Ring* aquests imants generen un camp de B = 1.42T i tenen una longitud de L= 1.384 m, mentre que al *Booster* té un camp de B=0.89T i hi ha dues longituds diferents, el BM5 amb L= 0.972m i el BM10 amb L=1.972 m. Al *booster* s'utilitzen en *ramping mode* ja que van incrementant el corrent i per tant el camp a mesura que augmenta l'energia dels electrons.

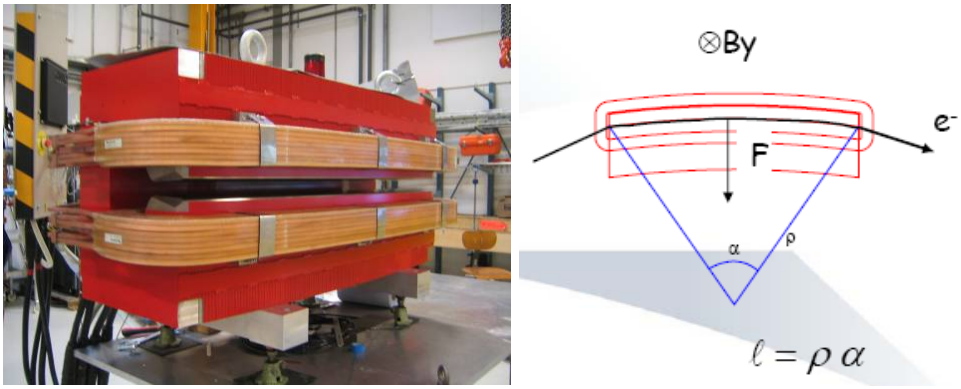


Fig 8. SR Bending Magnet, gràfic de trajectòria electró

Característiques dels *Bending magnets* del sincrotró:

		SR	BO (BM05)	BO (BM10)
Number of magnets		32	8	32
Bending angle	degrees	11.25	5	10
Curvature radius	m	7.047	11.4592	11.4592
Central gap	mm	36	22	22
Central magnetic field	T	1.42	0.89	0.89
Gradient	T/m	5.65	2.29	2.29
number of coils		2	2	2
number of turns		40	12	12
maximum current	A	527	660	660
dissipated power	W	9600	2000	4000
number of cooling circuits		2	2	2
for a P=7 bars, ΔT	ºC	8.6	10	10

Fig 9. Taula característiques bending magnets

- *Quadrupole magnets* : aquest tipus d'imants té 4 pols i s'utilitzen per focalitzar el electrons en el pla transversal. El camp varia linealment en funció de la distància al centre de l'imant. Aquest tipus d'imants focalitza en un pla mentre que defocalitza en l'altre pla. Aquest és el motiu pel que hi ha imants focalitzadors en horitzontal (*QH)* i focalitzadors en vertical (*QV).*

- *SR:* tenim 112 imants distribuïts en 10 famílies *QH* i 5 famílies *QV* . Les longituds poden ser 220, 260, 280 o 550 mm. El gradient màxim d'aquests quadrupols és 22.4T/m.

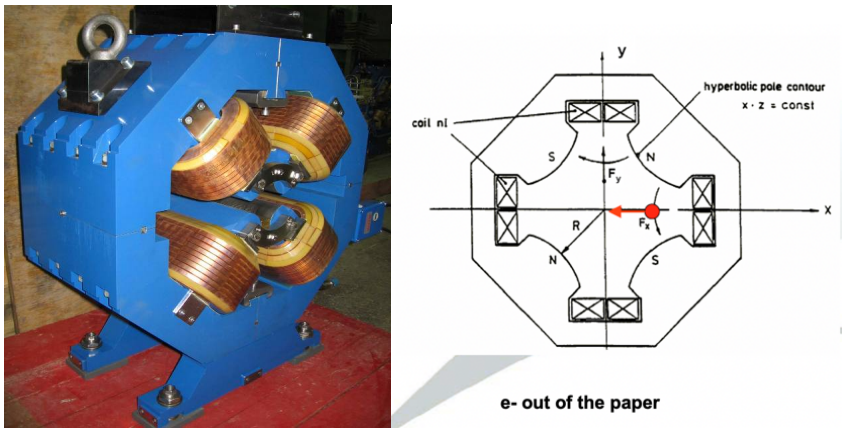


Fig 10. SR Quadrupole Magnet

- *Booster:* hi ha 60 imants distribuïts 2 famílies *QH* i 2 famílies *QV*, dels 60 imants, 36 són combinats amb un component sextupolar. Les longituds són 180 i 360 mm.

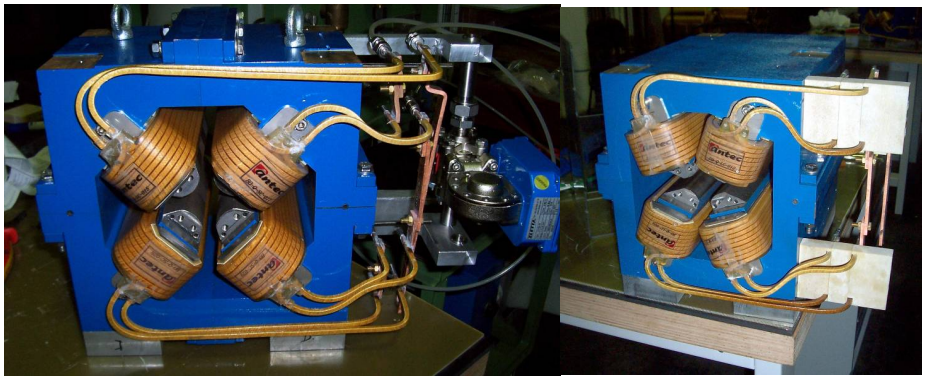


Fig 11. Booster normal quadrupole *i* combined quadrupole

- <u>*Sextupole magnets*</u>: són imants de 6 pols en els que el camp varia quadràticament amb la distància al centre de l'imant. S'utilitza per focalitzar els electrons segons la seva energia, compactant-los longitudinalment, així doncs, els electrons que tenen diferents energies veuen diferent força. Les partícules amb una energia més alta tenen més focalització i les que tenen menys energia redueixen la focalització.

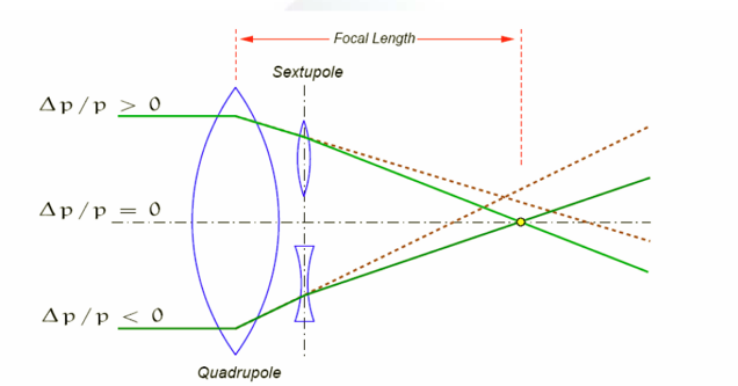


Fig 12. Gràfic focalització sextupol

- *SR:* hi ha 120 *sextupoles* dividits en 5 famílies *SH* i 5 famílies *SV* i tenen 2 longituds, 150 i 220 mm.

Fig 13. SR sextupole

- *Booster*: 16 imants on tenim una família SH i una família SV.

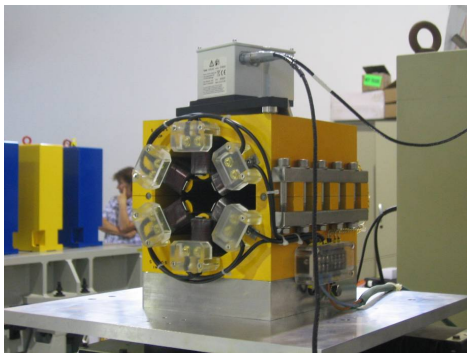


Fig 14. Booster sextupole

- *Corrector magnets:* són imants bipolars que la seva principal funció és la correcció de l'òrbita dels electrons perquè aquests es mantinguin el més a prop possible de la seva *Golden orbit*. En el cas del *SR* aquests imants estan inclosos dins als sextupols afegint-hi dos bobines més. Al *Booster* estan separats del sextupols i funcionen en dos modes: la gran majoria *DC mode,* en que tenen una energia constant i per tant un camp magnètic també constant, i uns quants que funcionen en *AC mode,* que rampejaran a la mateixa velocitat que incrementa l'energia dels electrons.

  Generen un camp de 0.050T i tenen una longitud de 70mm.

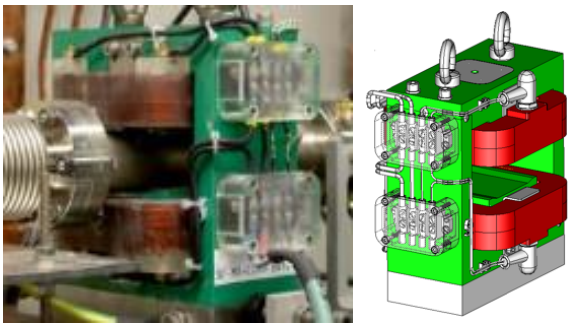


Fig 15. Booster corrector

- *Pulsed magnets:* són imants amb un pols característic molt curt que s'utilitzen per passar els electrons de *Linac a Booster* , de *Booster a SR* i per acumular electrons al *SR*. Tenim dos tipus d'imants polsats:
  - *Septum:* és un tipus d'imant polsat que pot ser un sinus d'una durada aproximada de 200µs , mig sinus o bé una forma d'ona DC. S'encarrega de corbar la trajectòria dels electrons per introduir-los al següent accelerador. Té la característica que té una regió amb un camp magnètic molt fort i una regió on el camp magnètic és pràcticament nul.

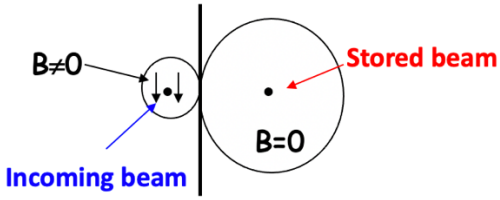


Fig 16. Gràfic funcionament Septum.

Fig 17. Septum magnet

  - *Kicker magnet:* s'utilitza per desplaçar el feix de la seva posició, tant en la injecció com en l'extracció d'electrons ja que te un pols inferior a 1µs. Hi ha un kicker a la injecció i 1 a la extracció del Booster, 1 a la injecció de *SR* i 4 dins al *SR* que son per generar el *bump* que s'utilitza durant el procés d'acumulació d'electrons, amb aquest *bump* el que fem és apropar el feix emmagatzemat al feix que s'està injectant.

- ***Vacuum* (Sistema de buit)**: els electrons que hi ha emmagatzemats a l'*storage ring* es mouen en sentit horari dins una cambra (*vacuum chamber*) d'alumini de forma ovalada,

que està instal·lada al llarg de la circumferència. Per evitar col·lisions dels electrons amb altres molècules s'ha de tenir aquesta cambra pressuritzada a *Ultra High Vacuum,* és a dir, d'entre $10^{-9} - 10^{11}$ mbar. Això permet assegurar que no tenim cap tipus de molècula a l'interior de la cambra i ajuda a augmentar el *life time* d'aquests electrons. Per aconseguir aquest nivell de buit s'utilitzen diferents tipus de bombes de buit (*pumps*), com *Ion Pumps, Cold Cathode Gauge* i *Turbomolecular Pumps*.

Fig 18. Vaccum chamber amb Ion Pumps

- ***Diagnostics and beam instrumentation:*** són els equips que s'utilitzen per a fer una anàlisi i un diagnòstic del feix d'electrons que està situat a la *vacum chamber* i que utilitza els camps produïts pel feix d'electrons. Hi ha diversos equips que permeten fer mesures directes de paràmetres del feix com la càrrega, la posició o el mida del feix:

  - *BPM (Beam position monitor):* determina la posició del feix i la informació que se n'obté és utilitzada per corregir l'òrbita amb els imants correctors. Està format per quatre botons que estan dins la *vacum chamber* i que utilitza el mètode *Delta over sum* per calcular la posició.

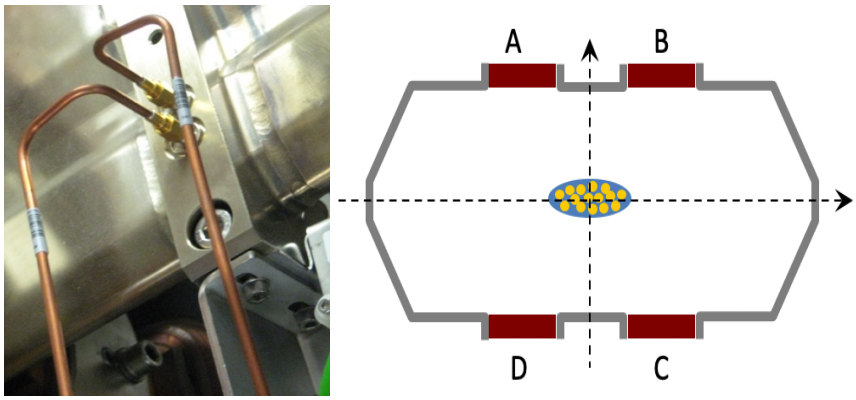


Fig 19. Beam position monitor

  - *FCT (Fast current transformer):* és un nucli ferromagnètic envoltat d'una bobina. Quan el feix d'electrons hi passa a través, indueix un camp al nucli com si fos un primari i aquest senyal és rebut per la bobina que actua com a secundària. Aquesta bobina es troba a la part exterior de la *vacum chamber*. Amb aquest dispositiu podem obtenir una imatge de la distribució del feix al llarg del *SR.*

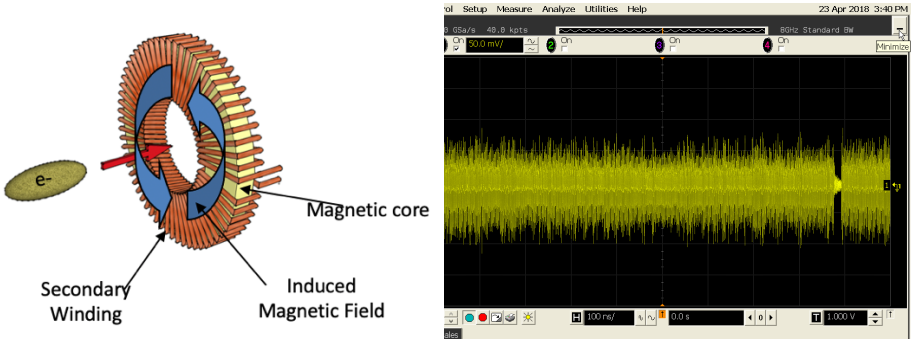


Fig 20. Picture of FCT , signal of FCT in scope

- *DCCT (DC Current Transformer):* té un funcionament similar al *FCT* però s'hi afegeixen components electrònics que agafaran la part *DC* del primari i permetran saber el corrent d'electrons que tenim al *SR*. També té les bobines situades a la part externa de la *vacum chamber* i té un tall de la conductivitat elèctrica de la càmera per capturar els corrents imatge amb la bobina. En el cas del *DCCT* s'utilitzen dues bobines per obtenir el flux magnètic compensat. Quan no hi ha corrent a l'*SR*, Vb – Va = 0, en canvi quan hi ha corrent el feix produeix un *offset* a les bobines, el corrent que es necessita per a compensar el *offset* fins a obtenir Vb – Va = 0, serà el corrent que tenim al *SR.*

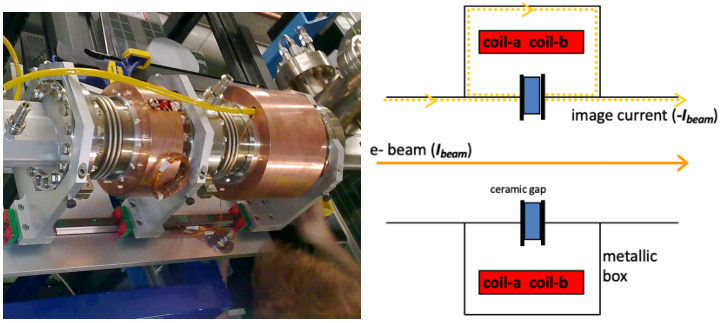


Fig 21. FCT and DCCT in SR , Picture of DCCT

- *Screen monitors*: aquests dispositius ens permeten veure la mida o la posició del feix en una determinada part de qualsevol dels acceleradors. Tenen un pistó neumàtic que introdueix la *fluorescent screen* a la *vacum chamber*, llavors el feix d'electrons xoca contra la *FS* i la llum es projectada cap a la càmera *CCD*. Utilitzant una anàlisi de la imatge adequada podem extreure la mida del feix. Hi ha dos tipus d'*screen monitors*: *YAG i OTR.* El primer tipus són utilitzats per feix de baixa intensitat ja que el material fluorescent produeix molta llum i poden saturar-se, en canvi el segon tipus produeixen poca llum i tenen una resposta lineal, la qual és òptima per a anàlisis quantitatives.

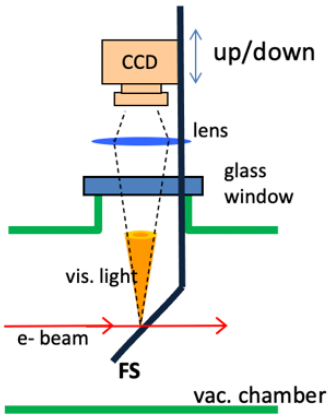


Fig 22. Picture of Screen monitor

- ***Insertion devices:*** són uns dispositius per extreure la llum amb la polarització que es vulgui i amb una energia més alta que si es fes amb un *bending magnet*. L'efecte dels *insertion devices* és el mateix que si s'incrementés el nombre de pols del *bendings* en un sol dispositiu, així doncs són estructures periòdiques de *dipole magnets*. Aquests dispositius poden ser *wigglers* o *undulators.*

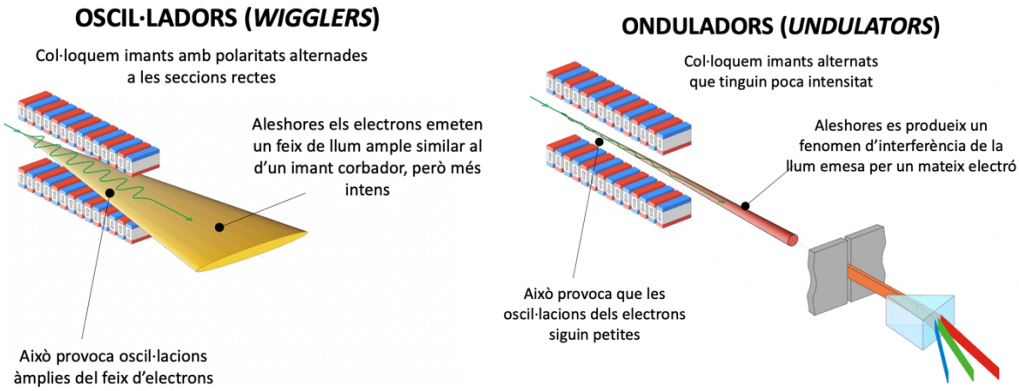


Fig 23. Pictures of a wiggler and undulator

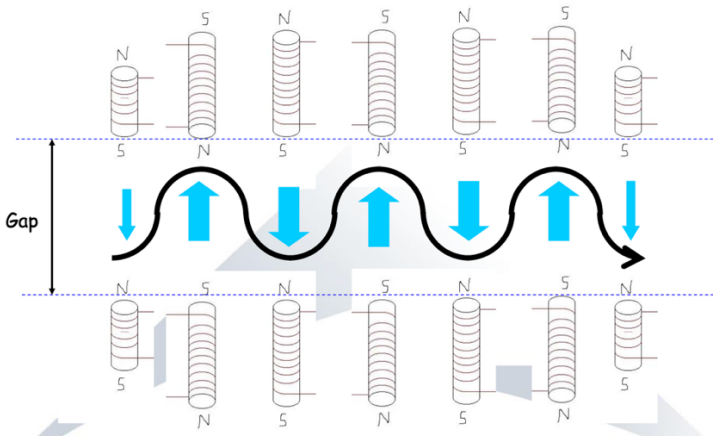


Fig 24. Picture of beam movement inside a wiggler

Mentre un *bending* normal genera 20keV d'energia un *SCW (Super Contacting Wiggler)* genera 30 keV. Dins al *wiggler* es mou el feix en forma de sinus obtenint un con de radiació molt més concentrat, 0.1º i per tant amb molt més flux de fotons i brillantor. La llum produïda en aquests *wigglers* tindrà una longitud d'ona relacionada amb el període del *wiggler* i la velocitat dels electrons. Hi ha diversos tipus *d'insertion devices* en funció de si el canvi de camp magnètic:

- Si el canvi es realitza modificant el corrent de les *coils* llavors són del tipus elèctric i poden ser *normal conducting* o *superconducting*.
- Si el canvi es realitza modificant el *gap* llavors són *permanent magnets* que poden ser *hybrid configuration* o *Pure Permanent Magnets.*

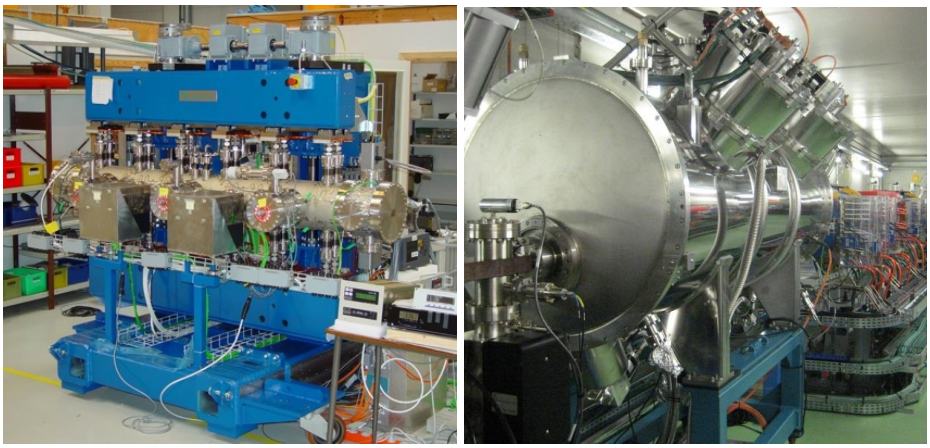


Fig 25. In vaccum ondulator, super contacting wiggler

- ***Radio Frequency**:* és el subsistema encarregat de donar l'energia que perden a cada volta els electrons, induint un voltatge en unes cavitats ressonants. Aquest concepte serà explicat extensament en l'apartat 3.

Fig 26. Radio Frequency Cavity

2.2.2 Beamlines

Per aprofitar els fotons generats, ja sigui amb *bending magnets* o bé amb *insertion devices*, a les tangents del *SR* (Fig 4) per on surten els fotons projectats, s'hi situen els laboratoris anomenats *beamlines* on s'hi posen les mostres que es volen estudiar i que disposen de múltiples instruments per modificar i adaptar les propietats de la llum, ja siguin miralls per enfocar la llum o bé monocromadors per seleccionar les diferents longituds d'ona, en funció de l'experiment que es vulgui fer. Hi ha dos tipus de *beamlines* en funció de si utilitzen *Hard-X rays o Soft X-rays*. Les *beamlines* estan dividides en dues parts: *Optical hutch* és la part on es tracta la llum que es rep per tal d'adequar-la al tipus necessari segons l'experiment, *i Experimental hutch* que seria la part on s'analitzen les mostres. A les *beamlines* amb *Hard-X rays* tant el *Optical hutch* com l'*experimental hutch* són laboratoris plomats per evitar els efectes de la radiació. En canvi a les *beamlines* de *Soft X-Rays* només l'*Optical hutch* és plomat.

Les *Beamlines* que hi ha a ALBA són les següents:

- **BOREAS**: és una *beamline* de *Soft X-Rays* d'espectroscòpia i reflectrometria que a través de l'absorció i reflexió de la llum polaritzada permet estudiar les propietats magnètiques dels materials.

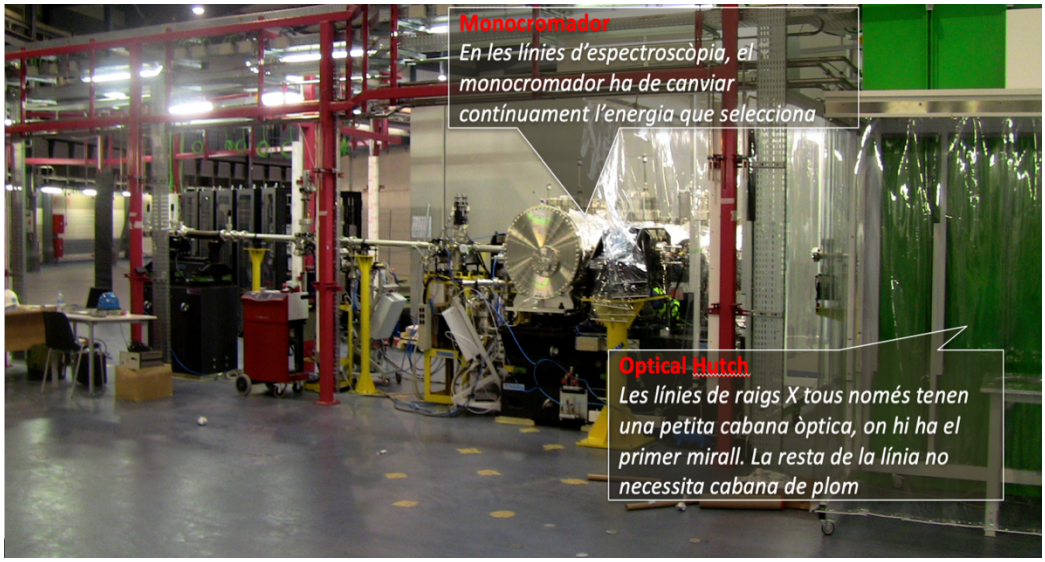


Fig 27. Vista general de Boreas

Posant com a exemple els disc durs per l'emmagatzematge de dades en que es guarden els en valors binaris, es poden determinar propietats magnètiques dels materials que ens permetrà desenvolupar dispositus més petits, més ràpids i més eficients.

- **CIRCE:** és una *beamline* de *Soft X-Rays* de microscòpia i espectroscòpia de fotoemissió que permet estudiar les propietats de les superfícies a partir dels electrons emesos efecte fotoelèctric. La fotoemissió es produeix quan un electró d'un àtom absorbeix un fotó i aquest li transfereix prou energia com per escapar-se de l'àtom. Mesurant d'on surten i amb quina energia surten es poden estudiar les propietats de la superfície dels diferents materials. CIRCE té dues estacions experimentals.

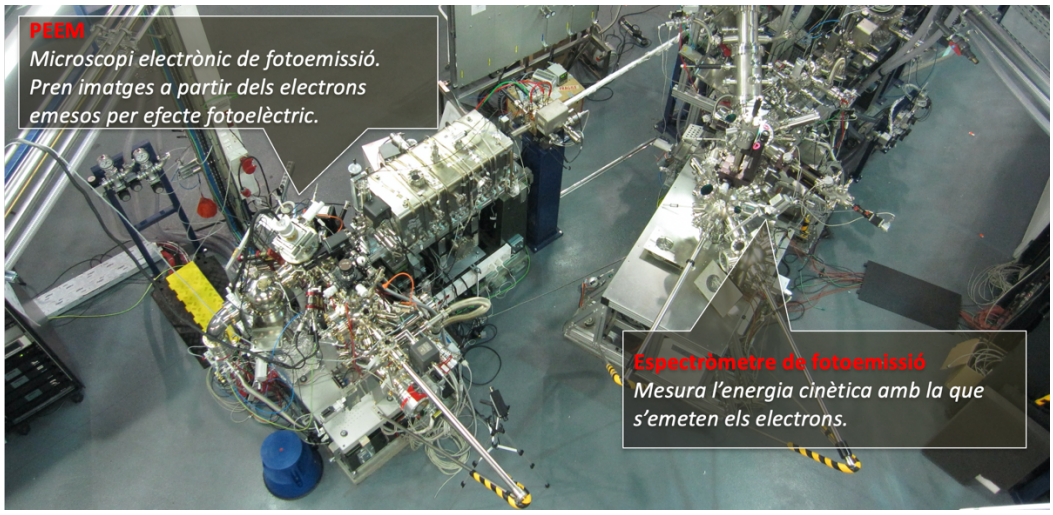


Fig 28. Estacions experimentals Circe.

Ja que els materials són diferents a la superfície, aquesta superfície en determinarà com reaccionen amb l'entorn (Fig 29).

Fig 29. Exemples d'estudis a Circe.

- **CLAES:** és una *beamline* de *Hard X-Rays* d'espectroscòpia de raigs X que permet determinar l'estructura i les propietats químiques dels sòlids. Als experiments de CLAES es fan servir gasos per estudiar cinètica de reaccions químiques.

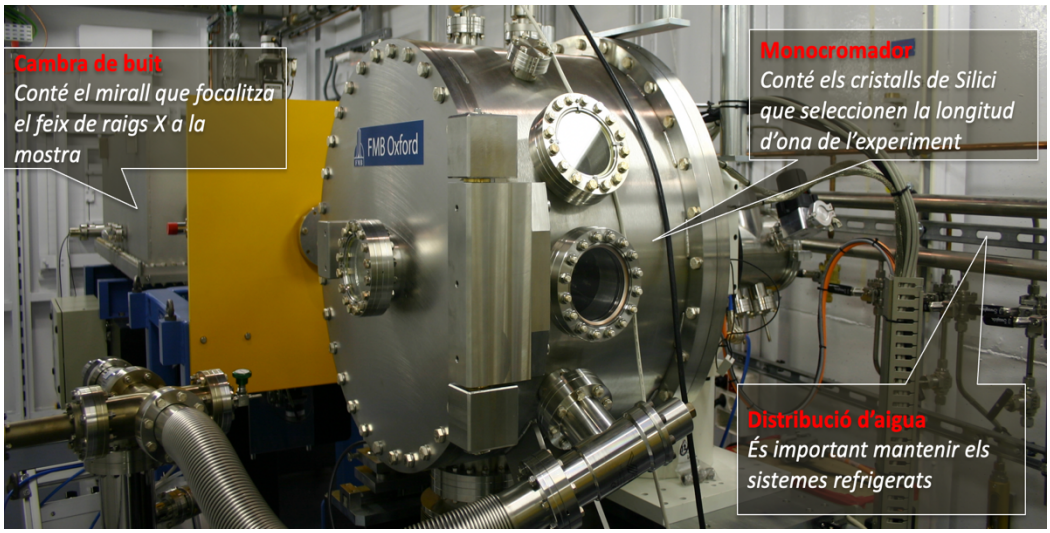


Fig 30. *Interior* Optical hutch CLAESS.

Un exemple del que s'estudia a CLAES és com reaccionen els gasos contaminants amb els diferents materials que hi ha al catalitzador (Fig 31) per tal de dissenyar-ne de nous i més eficients.

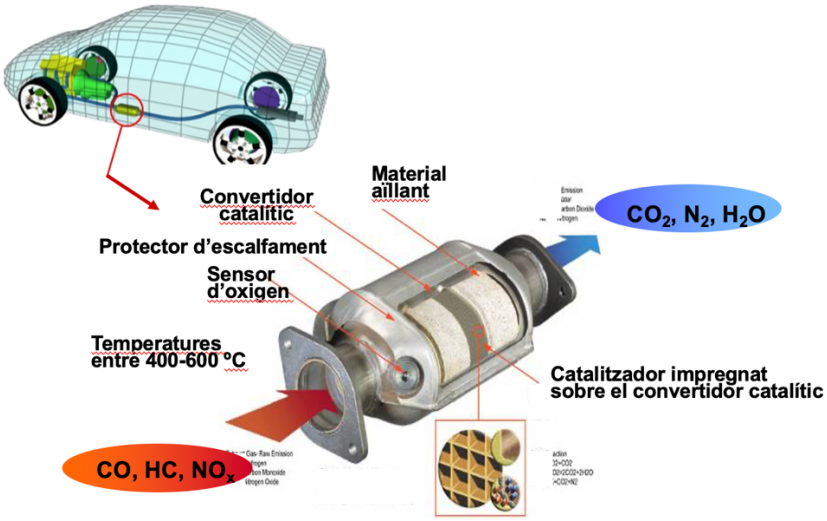


Fig 31. Exemple estudi CLAESS.

- **MISTRAL**: és una *beamline* de *Soft X-Rays* que fa microscòpia i tomografia de raigs X en criogènia, és a dir, és un microscopi que serveix per a prendre imatges tridimensionals d'objectes de mida cel·lular, amb el que es poden veure l'interior de la cèl·lula amb més resolució que amb els microscopis de llum visible, arribant a tenir un milió de vegades més resolució que un TAC convencional. Amb tot això es poden fer reconstruccions tridimensionals de les cèl·lules.

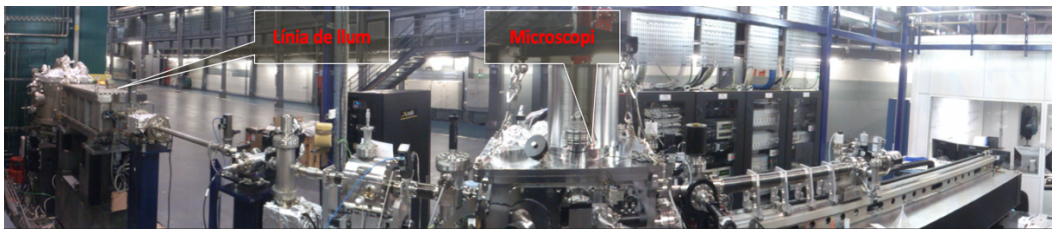


Fig 32.Estació experimental MISTRAL.

Un exemple d'investigació es veure com evolucionen els virus dins la cèl·lula, quan aquesta es contagia.

- **MSPD:** és una *beamline* de *Hard X-Rays*, en que s'utilitza la difracció de pols que ens permet estudiar l'estructura dels materials en el seu estat natural. S'utilitza un difractòmetre d'alta resolució. (Fig 33)

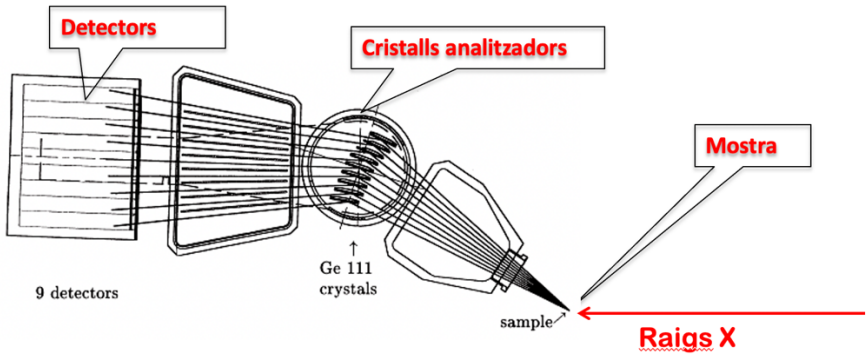


Fig 33.Esquema funcionament difractòmetre

Un exemple seria el comportament dels materials en diferents condicions, ja siguin d'alta pressió o temperatura, sotmesos a camps elèctrics o magnètics o bé amb agents químics reactius.

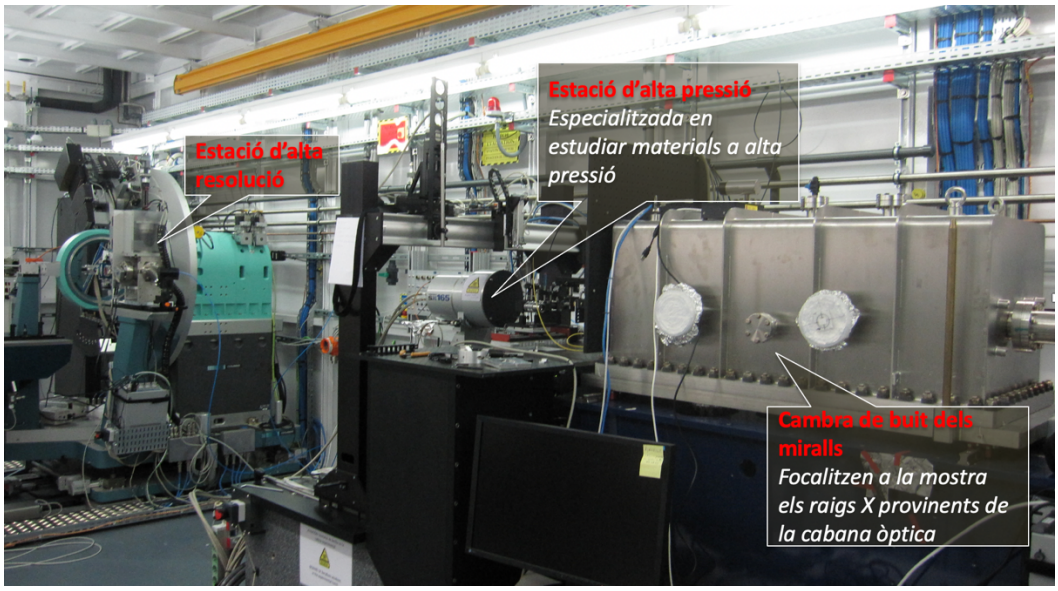


Fig 34. Experimental Hutch MSPD

- **NCD:** és una *beamline* de *Hard X-Rays* que realitza difracció no cristal·lina, dedicada a l'anàlisi de mostres de l'àmbit de la ciència dels materials i les ciències de la vida, en la qual la difracció no cristal·lina permet determinar l'estructura dels objectes de mida de nanòmetres.

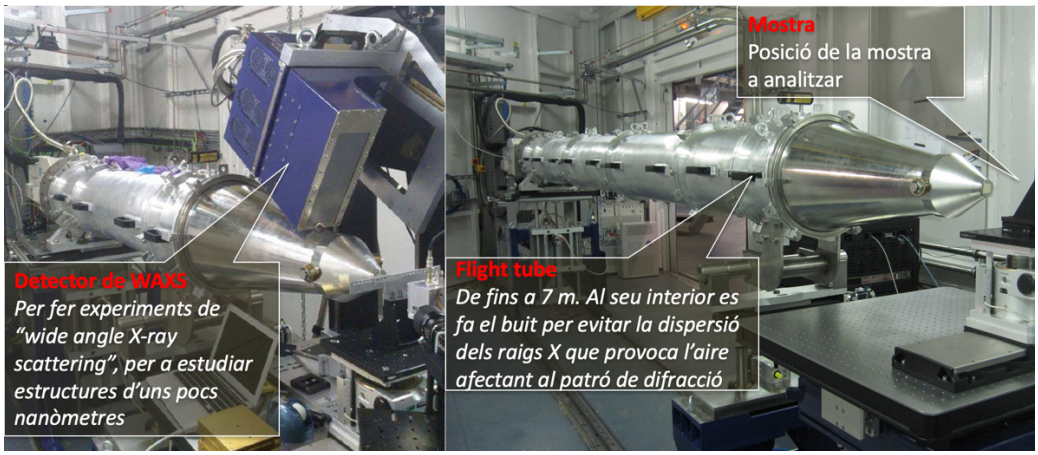


Fig 35. Experimental Hutch NCD.

Alguns exemples en el camp de ciències dels materials serien en aplicacions industrials com plàstics, gomes, lubricants, adhesius, etc. Mentre que altres exemples en el camp de les ciències de la vida serien l'anàlisi de l'estructura dels músculs, teixits biològics anòmals, implants bio-compatibles, etc.

- **XALOC:** és una *beamline* de *Hard X-Rays* que realitza cristal·logràfica de molècules on es pot determinar l'estructura de les molècules. A partir del patró de difracció d'un monocristall podem reconstruir amb precisió la ubicació de tots els àtoms de les molècules.

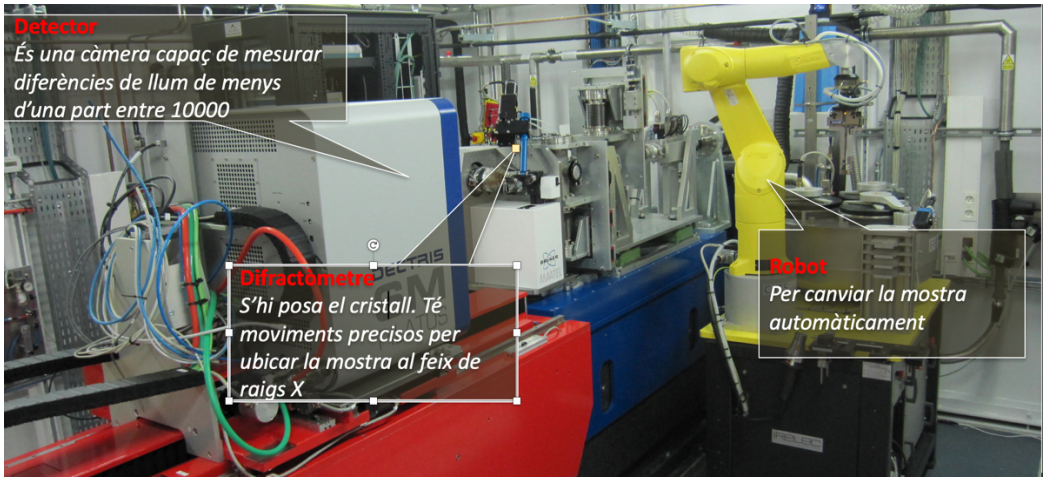


Fig 36. Experimental Hutch XALOC.

Un exemple d'estudi són les molècules que composen el éssers vius: proteïnes, enzims i àcids nucleics.

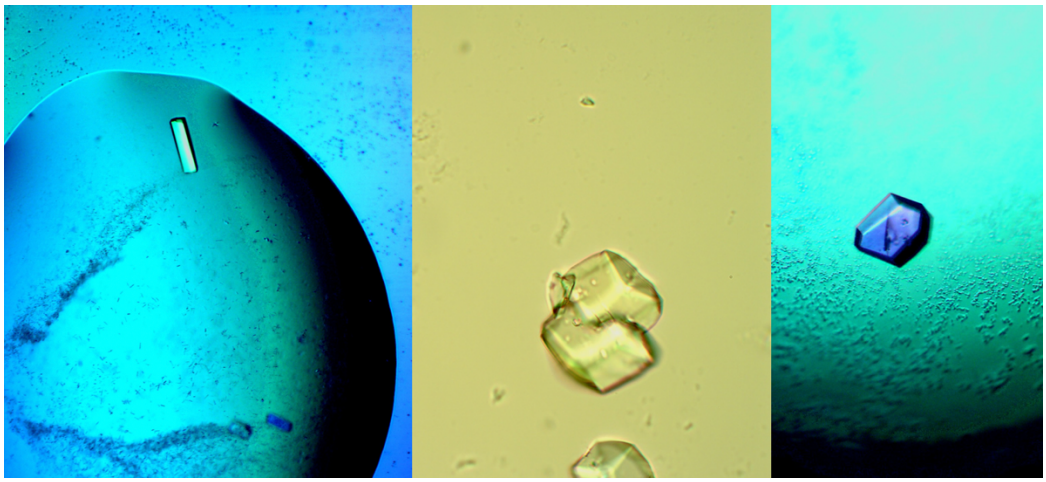


Fig 37. *Exemple de cristalls de proteïna que es troben dins d'una gota d'aigua congelada.*

- **MIRAS:** és una *beamline* de *Soft X-Rays*, és de *infrared* i permet estudiar un ampli ventall de sectors: farmacèutic, biologia, objectes antics, ciència de materials...

Fig 38. Experimental hutch MIRAS

S'estan construint noves *beamlines* :

- **LOREA:** és una *beamline* del tipus *Angle Resolved Photo Electron Spectrum* que entrarà en funcionament a principis de 2021.
- **NOTOS:** és una *beamline* per *XAS, XRD and metrology Applications* que entrarà en funcionament a la primavera de 2021.
- **XAIRA:** serà una *beamline* del tipus *Microfocus Macromolecular Crystallography* que entrarà en funcionament al 2022.
- **FAXTOR:** serà una beamline Fast Hard X-ray micro-Computed tomography & Radiography imaging que està previst que entri en funcionament a l'octubre del 2023.

2.3 Conclusions

Per una part s'han explicat tots els subsistemes que composen un accelerador de partícules i que ensenyen la complexitat que té per al correcte funcionament dels acceleradors degut a la gran varietat d'elements que els composen. Per altra banda, S'han vist totes les *beamlines* construïdes i per construir, les quals tenen un gran nombre i diversitat d´experiments que poden realitzar. Es pot concloure que dins un sincrotró es poden realitzar un gran nombre d'estudis amb àmbits molt diversos d'aplicació a la vida quotidiana.

3. Sistema Radio Freqüència

En aquest apartat es descriu el sistema de radio freqüència, explicant-ne els conceptes bàsics on es veu el perquè de la seva utilització. A continuació se'n descriuen les parts des de la generació de la potència RF, passant pel transport fins al seu lliurament a les cavitats. Finalment també s'expliquen els conceptes de RF aplicats exclusivament als acceleradors de partícules.

3.1 Parts sistema de radio freqüència

3.1.1 *Conceptes Basics*

- Quin són els objectius del sistema de radiofreqüència en un accelerador de partícules?
  Els principals objectius són accelerar els electrons fins a velocitats relativistes i restaurar l'energia perduda degut a la radiació sincrotró. Com s'ha explicat anteriorment, quan un electró corba la seva trajectòria i genera un fotó, aquest perd energia.

- Com s'obté aquesta energia?
  S'obté mitjançant un camp elèctric.

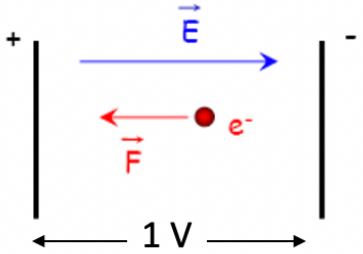


Fig 39. Camp elèctric

Així doncs, si es crea una diferencia de potencial de 1 V dins la cavitat, un electró obté una energia de 1eV, degut a que l'*Storage Ring* els electrons tenen una energia de 3 GeV necessiten 3000 MV per accelerar els electrons a una velocitat relativista. L'energia perduda a cada volta és de 1,1 MV aproximadament, en funció de les forces aplicades pels *insertion devices*.

- Com s'obtenen aquests 3000 MV?
  En un accelerador circular la partícula té un increment d'energia cada cop que veu un voltatge.

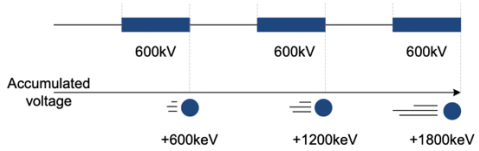


Fig 40. Voltatge acumulat

- Per què s'utilitza voltatge *RF* en comptes de *DC*?
  - El principal inconvenient del *DC* és que al voltatge que seria necessari es tindrien descàrregues electrostàtiques.
  - Des del punt de vista de *Beam Dynamics*, el voltatge de *RF* focalitza al pla longitudinal, ja que dins del mateix *bunch* les partícules amb diferents energies i

per tant trajectòries diferents, veuran diferents voltatges de manera que es compensaran i per tant s'agruparan (3.2.2 Synchronous phase).

- Per altra banda, com que el feix d'electrons no és continu sinó que està agrupat en *bunches,* no es necessita un camp elèctric continu. S'utilitza una freqüència de 500 MHz, i tenint una longitud de 268,8 metres de circumferència de l'*storage ring*, s'obté un número harmònic de 448, és a dir, es tenen 448 *buckets* amb una separació entre *bunches* de 0,6 metres.

$$Circumferència = h \cdot \lambda_{RF}$$

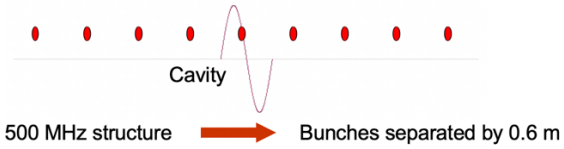


Fig 41. Bunches

- Quins són els principals components d'un sistema RF en un sincrotró?

  - *RF Amplifiers*: produeix la potència de les ones electromagnètiques.
  - *Waveguides & Auxiliaries*: s'encarreguen de transportar l'energia fins les cavitats.
  - *Cavities*: transformen les ones electromagnètiques en voltatges RF.
  - *LLRF:* és el sistema que serveix per controlar l'amplitud i les fases de les ones RF.

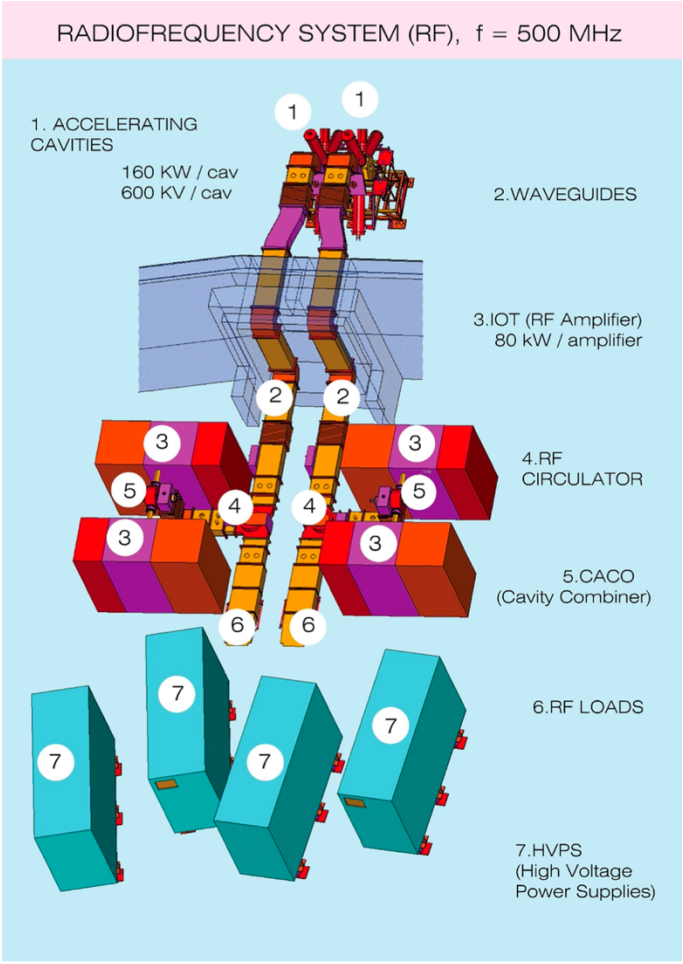


Fig 42. Elements del sistema de RF

3.1.2 ***Distribució del sistema RF***

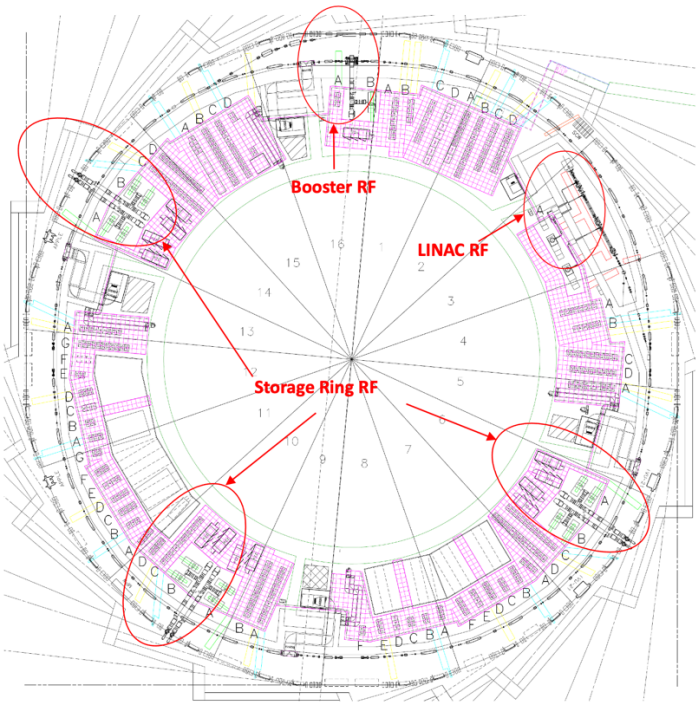


Fig 43. Distribució sistemes RF

Hi ha tres subsistemes de RF al sincrotró Alba:

- *Linac*: els amplificadors son 2 klystrons que funcionen a 3GHz que ens serviran per incrementar l'energia dels electrons de 90keV fins a 100 MeV. S'utilitza aquesta freqüència perquè a aquesta energia, aquests tipus d'amplificadors son viables i l'espai que ocupen aquestes cavitats és molt més reduït.
- *Booster*: És tracta d'un amplificador de SSPA (*Solid State Power Amplifier*) a 500 MHz unit mitjançant guies d'ona a una cavitat de 5 cel.les

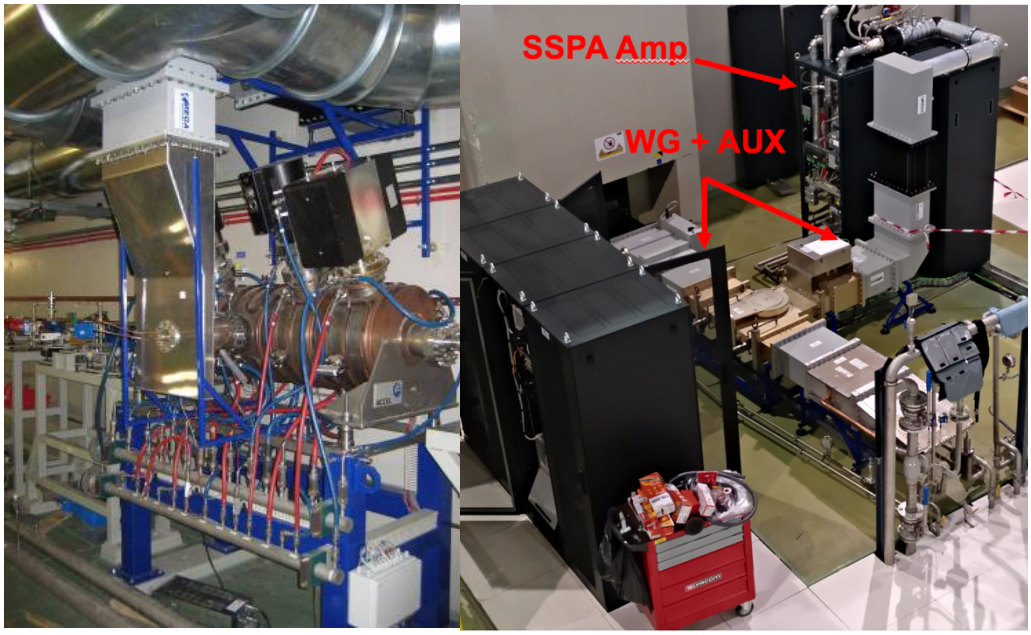


Fig 44. 5 cells cavity & SSPA + Wave guides

- *SR*: Com amplificadors tenim 12 IOTs (*Inductive Output Tube*) que s'uneixen mitjançant guies d'ones a 6 cavitats. Cadascuna d'aquestes part es explicada a continuació

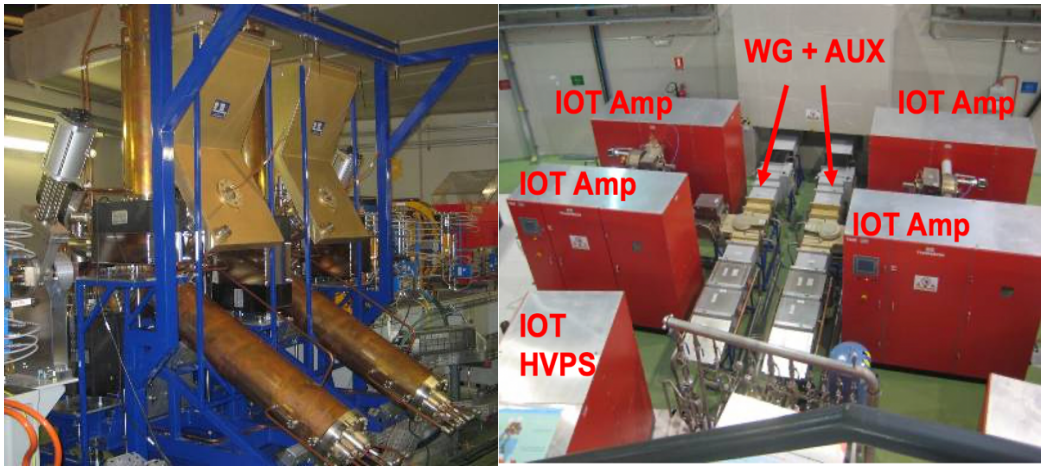


Fig 45. 1 cells cavities & SSPA + Wave guides

3.1.3 ***Cavities***

La manera de generar el voltatge és utilitzant cavitats acceleradores que actuen com a ressonadors de RF, son del tipus *Pillbox cavities normal conducting* ja que aquestes tenen el camp elèctric al *axis* és a dir al camí on passaran els electrons i el camp magnètic gira al voltant del camp elèctric tenint el seu màxim a les parets de la cavitat, aquest camp serà 0 al llarg del camí de les partícules.

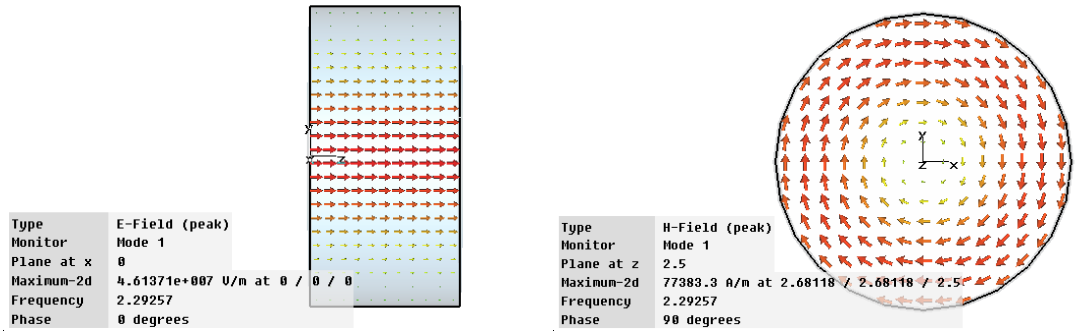


Fig 46. Electric field & Magnetic field. Font: Dr.G Burt, Lancaster University

La forma més simple d'un circuit ressonador és un circuit RLC on tindrem un voltatge màxim quan la freqüència és ω_0

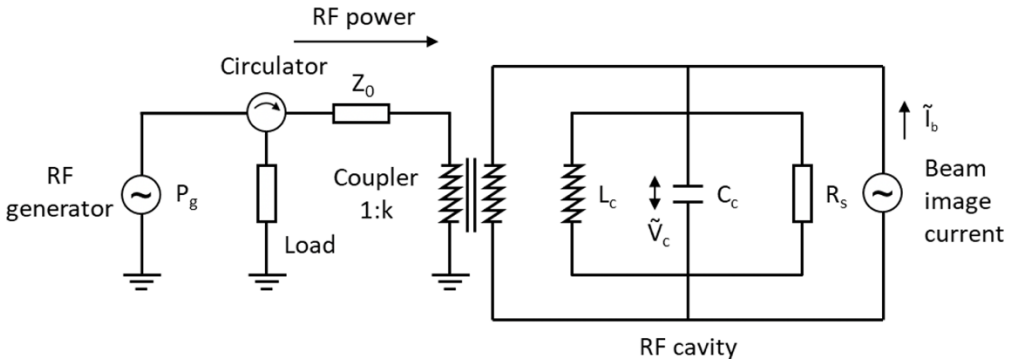


Fig 47. RLC Circuit. Font: Ignasi Bellafont

El TM_{010} *Fundamental mode* tindrà un camp elèctric purament longitudinal, el camp elèctric i magnètic no tindran dependència angular i la freqüència dependrà del radi independentment de la longitud. Aquest radi en el nostre cas serà adaptat per aconseguir 500 MHz i la longitud de la cavitat ens vindrà definida per:

$$L = \frac{\beta \cdot \lambda}{2} \text{ (3.2.1 Beta cavity)}$$

Equació 1. Longitud cavitat

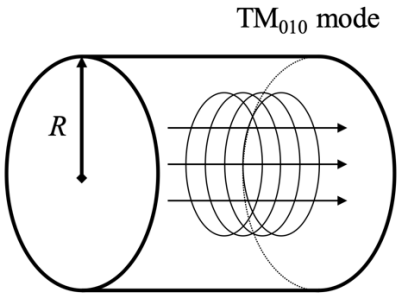


Fig 48.Fundamental mode. Font: *Jefferson Lab*

Per rebre el màxim voltatge, la partícula ha de travessar la cavitat en la meitat del període positiu.

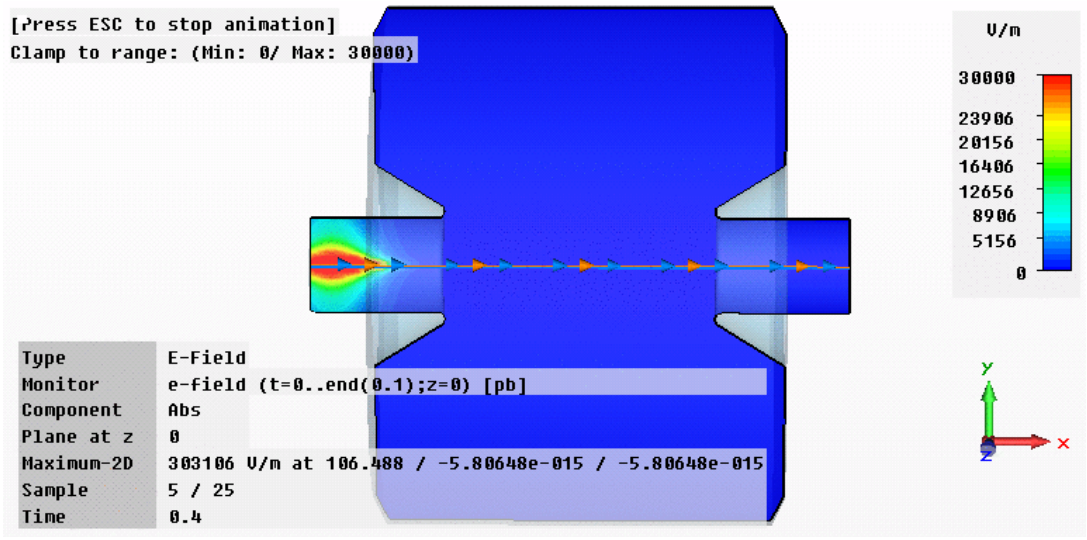


Fig 49. Partícula travessant cavitat. Font: Beatriz bravo

Els *High Order Modes* son altres freqüències que poden ressonar a l'interior de la cavitat induint nous camps elèctrics que poden ser perjudicials per al feix d'electrons. Per absorbir aquestes freqüències altes utilitzem *Dampers* a les cavitats. Els *dampers* no deixen de ser una linea de transmissió que acaba en una càrrega amb la geometria adequada, en que el voltatge induït al resistor degut *coupling* entre el camp magnètic i l'elèctric sigui 0 i per tant no atenuarà freqüència fonamental, en canvi, els *high order modes* el voltatge serà diferent de 0 i per tant s'atenuaran.

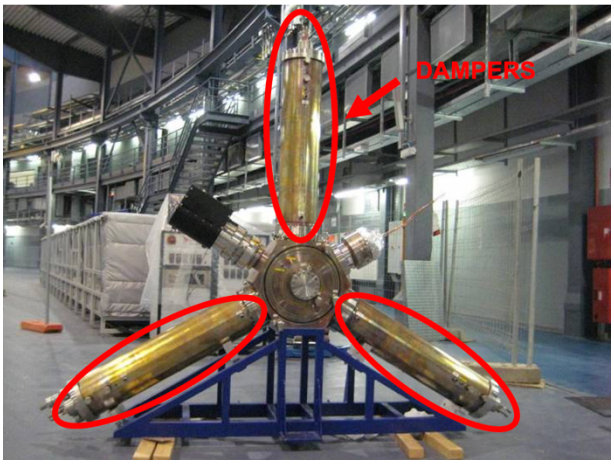


Fig 50. Dampers.

Tenim dos tipus de *cavity couplers:*

- *Input couplers* que s'encarreguen de portar la potència de RF a la cavitat.
- *Pickup loop* que utilitzem per llegir el voltatge de la cavitat agafant una petita porció del Voltatge, és a dir llegirem milivolts i amb la següent fórmula obtindrem el voltatge a la cavitat.

$$PickUpCoupling(dB) = 10\log\left(\frac{V_{pickup}^2/50}{V_{cav}^2/3.3\cdot 10^6}\right)$$

Equació 2. *Pickup Coupling*

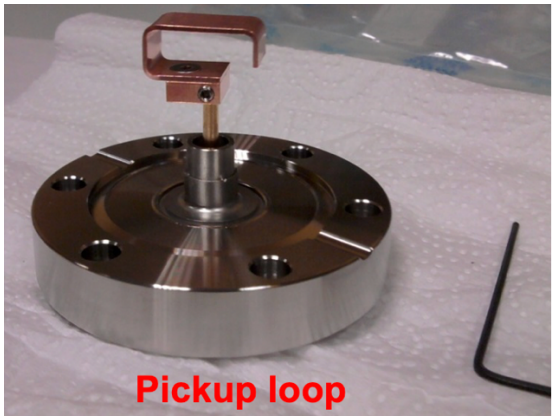


Fig 51. Pickup loop

Les característiques generals de les cavitats són:

Type	Single-cell	
HOM damped:		
Longitudinal	< 2	MΩ.MHz
Transverse	< 60	kΩ/m
Number	6	
Frequency	500	MHz
R_{shunt}	3.3	MΩ
Max. Voltage	> 700	kV
Input power	> 150	kW
Cooling capacity	> 80	kW

Fig 52. Característiques cavitats

3.1.4 ***RF Amplifiers***

L'amplificació del senyal de 500 MHz que prové del *master oscillator* i que ens dona el senyal de sincronisme a tot el sincrotró, necessita ser augmentat per poder assolir una voltatge lliurat a les cavitats de fins a 600 kV, per poder lliurar aquest voltatge, en el cas del *SR* l'etapa d'amplificació, es divideix en dos etapes:

- En una primera etapa tenim un *Solid State Amplifier* que amplificarà el senyal provinent del *Master Oscillator* fins a 500 W. Les característiques d'aquests *SSA* son:

    - *Gain* 55 dB
    - *Maximum Input* 3dBm
    - *Maximum Output* 600 W

- En una segona etapa tenim els *Inductive Output Tubes* (*IOT*) que s'encarrega de transformar alt voltatge DC en voltatge RF i s'encarreguen de produir els 80 kW que es poden lliurar a les cavitats. Les característiques dels IOTs son:

    - *Gain 23 dB*
    - *Maximum Input 500 W*
    - *Maximum Output 90 kW (Nomimal working point 80 kW)*

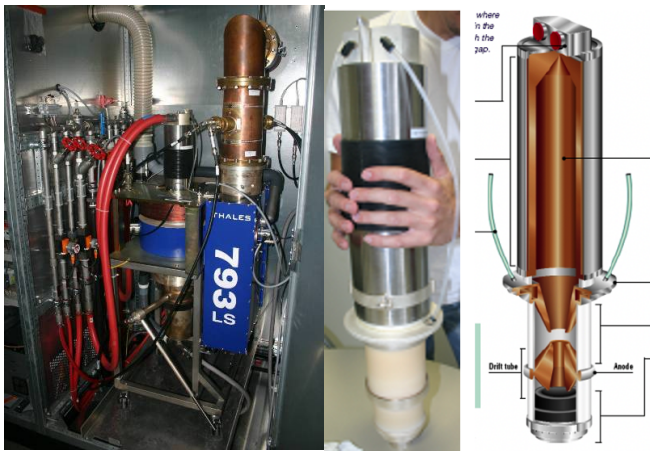


Fig 53. IOT

Els IOTs els podem considerar com uns petits acceleradors on tenim:

- Un *Vacuum Tube.*
- *E-Gun* que te un filament de grafit que s'escalfa amb una font de corrent de 12V/25A.
- Una diferencia de potencial entre l'ànode i el càtode de +36kV.
- *Grid* que rep una ona RF fins a 400 W i que es comporta de la següent manera:
    - Si la ona RF es positiva s'extreuen electrons del filament.
    - Si la ona RF es negativa no s'extreuen electrons de manera que creem una estructura de 500 Mhz.
- Accelerem els electrons amb una freqüència de 500 Mhz on al eix central del tub tindrem el camp elèctric i al voltant d'aquest eix és on tindrem el camp magnètic.
- Hi ha un *Output Cavity* i un *Coupling loop* per extreure aquesta potència RF del amplificador.
- Hi han *Focusing Coils* que serveixen per mantenir la trajectòria dels electrons a l'interior del tub.

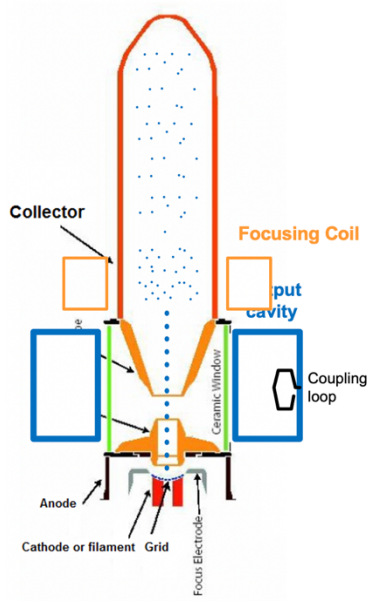


Fig 54. Esquema *IOT*

Aquestes dos etapes d'amplificació es troben instal·lades dins el mateix cabinet.

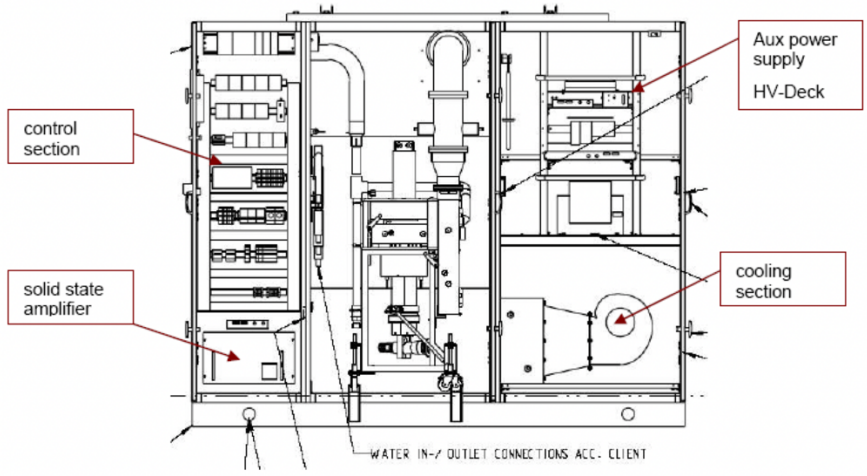


Fig 55. Esquema *cabinet IOT*

Aquests etapes amplificadores necessiten una energia que es subministra des dels *HVPS Cabinets (High Voltage Power Supplies)*. Tenim uns requeriments de 36kV i 3,2 A per obtenir 80kW.

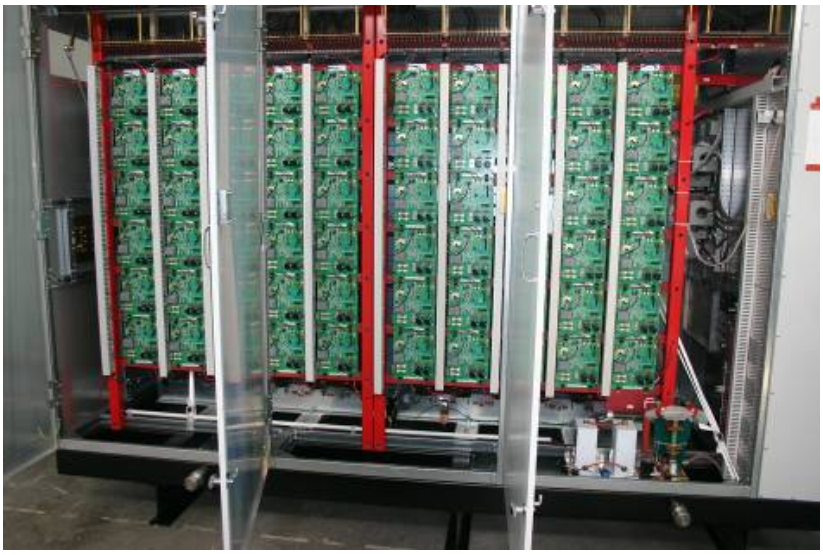


Fig 56. cabinet HVPS

Dins d'aquests *cabinets* tenim 60 mòduls de 700 Volts fabricats per Thomson, dels quals nomes en necessitem 52 per aconseguir els 36 kV, això dona redundància durant el funcionament, de manera que es poden avariar 8 mòduls i continua funcionant. Per evitar que uns es degradin més que els altres durant el funcionament van canviant els mòduls actius.

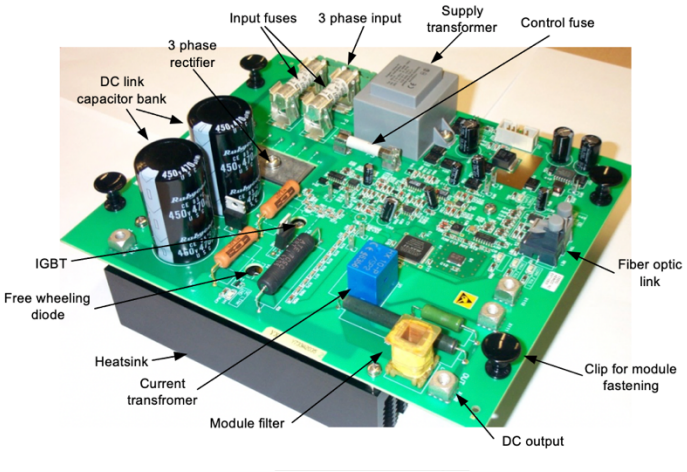


Fig 57. Mòdul *HVPS by Thomson*

L'amplificació de senyal al *Booster* consta d'una sola etapa on tenim un *SSPA* (*Solid State Power Amplifier).* L'etapa d'amplificació es fa amb 96 transistors MRFE8VP8600H del fabricant NXP, distribuïts en 12 mòduls, que es van combinant entre ells fins a obtenir una potència màxima de 50 kW.

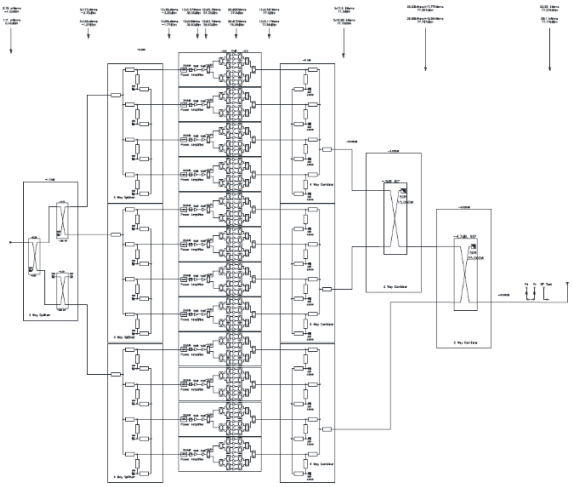


Fig 58. *SSPA* i esquema de l'equip

Com que al *Booster* només es necessiten 35kW, es té el benefici de la redundància ja que no es necessiten tots els mòduls per aconseguir la potència nominal de treball. En el cas d'aquest transmissor, dels 12 mòduls continua estan operatiu amb 10 i per tant qualsevol problema en 2 mòduls no afecta al funcionament del transmissor. A més, els mòduls es poden canviar en mode *hot swap*. El nombre de mòduls amb els que es pot funcionar depèn de la següent fórmula:

$$P_{generada} = P_{nominal} \cdot ({}^{(n-m)}/_{n})^2$$

Equació 3. Potència generada al SSA

On n és el nombre de mòduls total i *m* és el nombre de mòduls avariats.

3.1.5 ***Waveguides & auxiliaries***

Dins dels sistema de guies d'ona (*waveguides*) hi ha diversos elements:

- *Caco*
- *Costubs*
- *Circulator*
- *Load*
- *Waveguide*
- *Shutter*
- *Watrax*

CACO és un dispositiu de 3 ports que no deixa de ser una *Pillbox Cavity* on hi ha dos *IOTs* alimentant aquesta cavitat, i en que les seves ones incidents són combinades en aquesta cavitat i la ona que en resulta surt a traves d'una ona rectangular. La fase i l'amplitud provinents dels dos *IOTs* han de ser idèntiques per minimitzar el *VSWR*.

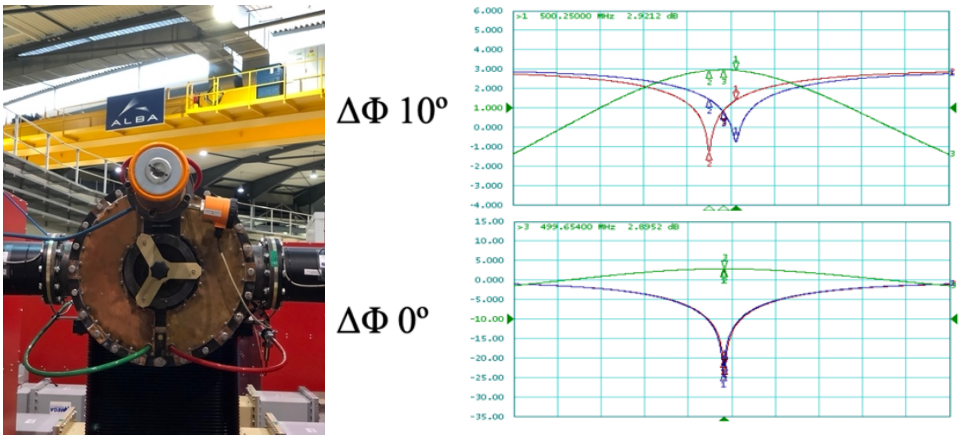


Fig 59. CACO & VSWR

Costub són curtcircuits instal·lats a les guies d'ona coaxials a la sortida dels *IOTs*. La principal funció és poder utilitzar la planta en cas d'alguna incidència en un dels *IOTs*. Si només funciona un dels dos *IOTs,* una ona estacionaria es creada entre el *CACO* i el IOT passiu amb una potència de fins a 64kV, això pot crear arcs molt perjudicials per a la ceràmica del *IOT* passiu. Així doncs els *costubs* aïllen l'*IOT* passiu tancant un curtcircuit, per tant permet un mode d'operació asimètric.

Fig 60. Costub

Circulator funciona com una 'rotonda' fent el símil amb el món automobilístic, on l'ona sempre ha de sortir per la primera sortida i serveix per aïllar els amplificadors *IOTs* de la cavitat, de manera que si es produeix potència reflectida des de la cavitat, aquesta no pugui perjudicar els *IOTs*.

Fig 61. Circulator

L'energia que torni de la cavitat, el circulador la projectarà cap a la ***Load,*** que s'encarregarà d'absorbir l'energia.

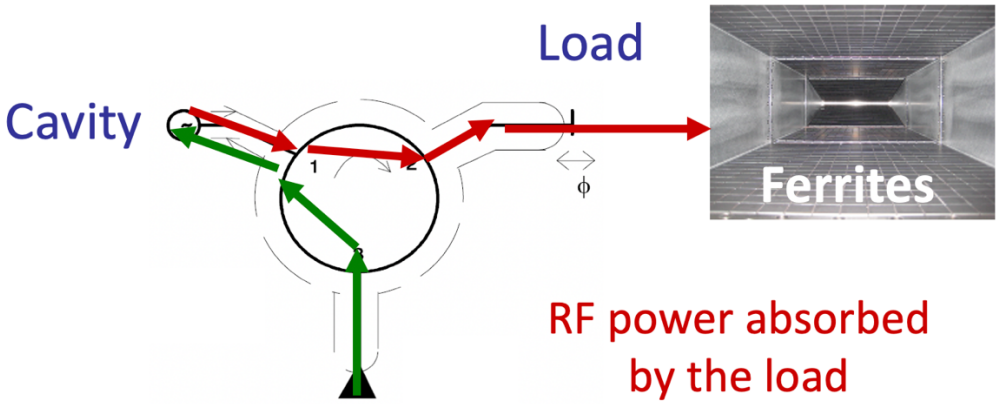


Fig 62. Funcionament del circulador

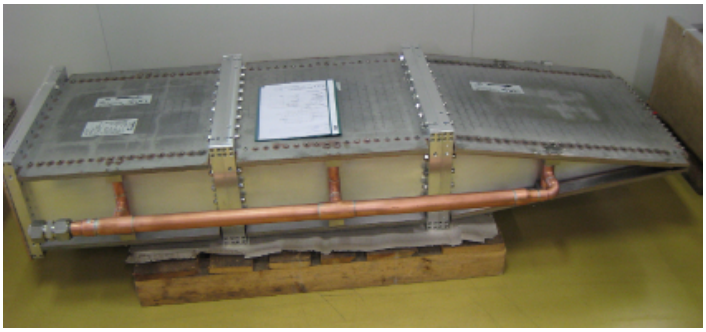


Fig 63. Load

Shutter serveix per aïllar la cavitat i per tant l'interior del túnel fent un curtcircuit a la guia d'ona abans d'entrar-hi. Permet un mode d'operació en el que es poden fer tests dels *IOTs* mentre el túnel està obert i per tant sense tenir permís per enviar potència a la cavitat, enviant la potència a la *load*.

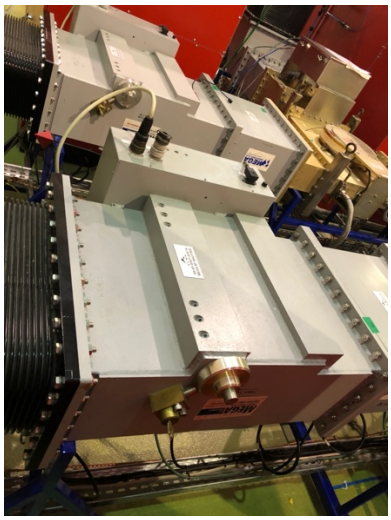


Fig 64. Diferents tipus de guies d'ona rectangular

***Waveguide*:** les guies d'ona són del tipus rectangular, com els *bidirectional couplers*. Son del tipus W1800 i suporten fins a 150kW en *continuos wave (cw)*. Hi ha diversos tipus:

Fig 65. Diferents tipus de guies d'ona rectangular

Els **bidirectionals couplers** s'utilitzen per saber tant la potència *forward* com la potència *reverse* que circular per la guia d'ona.

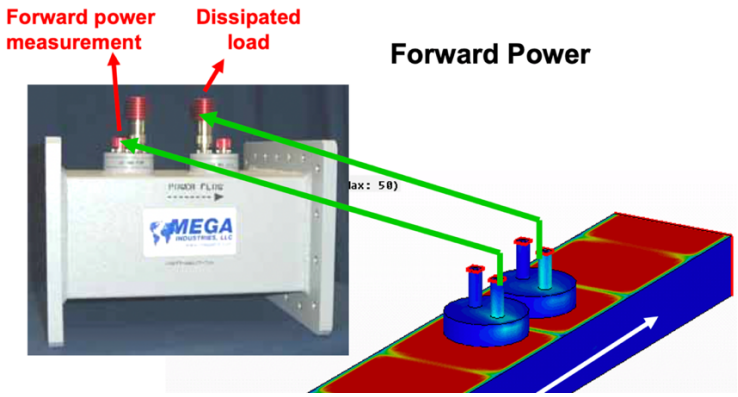


Fig 66. Bidirectional coupler

WATRAX (*Waveguide Transition To Coaxial)* s'encarrega d'agafar el senyal de RF provinent dels *IOTs* que es propaga per una guia d'ones rectangular i canviar-la a coaxial per introduir-lo a la cavitat. Principals característiques del *Watrax*:

- Maximum CW power: 150kW
- VSWR: 1.02
- Maximum peak ekectric field: 265 kV/m
- Power dispated: 132 W

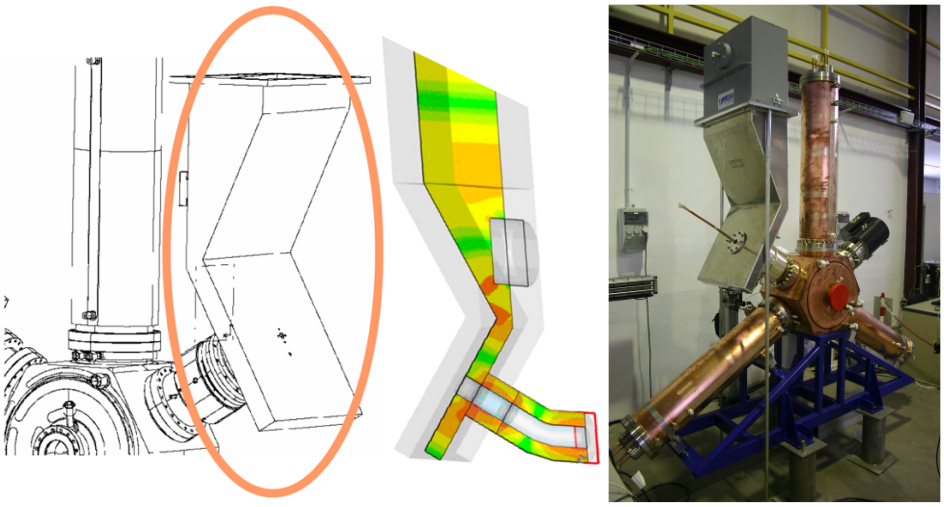


Fig 67. Watrax

3.1.6 ***LLRF***

EL *LLRF (Low Level RF)* és un sistema de control local de la cavitat de RF que té com a principals funcions:

- Control de l'amplitud i fase del voltatge de RF sincronitzat amb els *bunches* d'electrons.
- Control de la freqüència de ressonància de la cavitat.
- Diagnòstic de RF.

Es necessita el *LLRF* en primer lloc perquè els amplificadors tenen *ripple*, en segon lloc perquè el seu guany i fase no sempre és lineal i, finalment, perquè el feix d'electrons produeix *beam loading* interaccionant amb la cavitat i per tant canvia el voltatge.

El LLRF està basat en *down and up conversion* i en modulació i desmodulació digital IQ on s'utilitza:

- *cPCI with 16 ADCs, 8 DACs* i *Virtex-4 FPGA.*

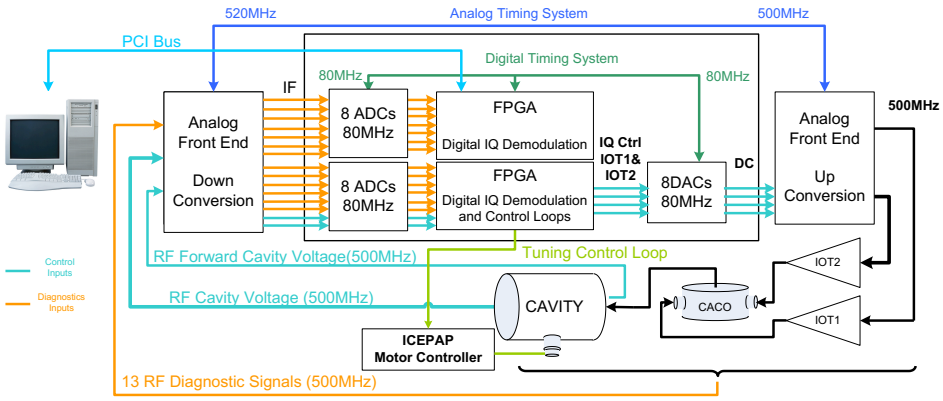


Fig 68. LLRF

- *Analog Front Ends for Downconversion (RF to IF).*

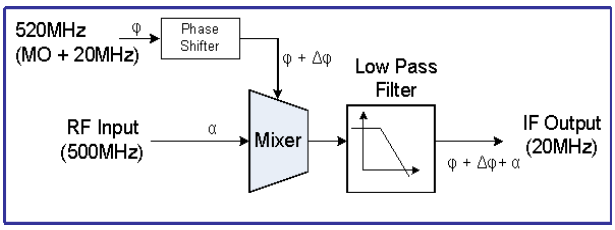


Fig 69. Downconversion

- *Analog Front Ends for Upconversion (DC to RF).*

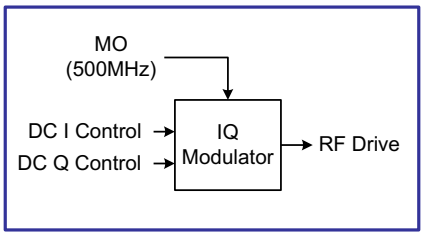


Fig 70. Upconversion

- *Timing systems: 520MHz (500 + 20 MHz) for downconversion synchronized with digital 80MHz clock for digital acquisition.*

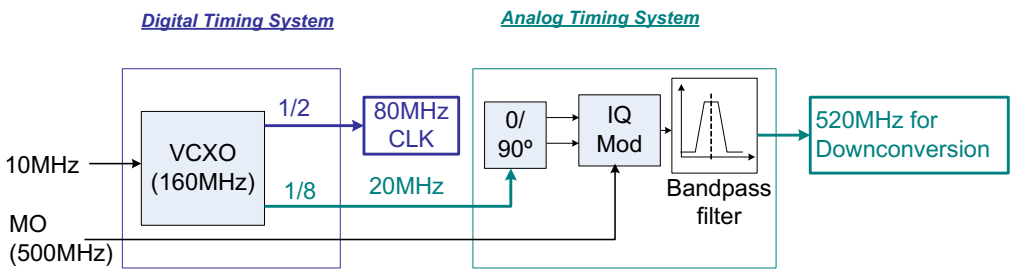


Fig 71. Timming System

3.2 Conceptes de *RF* específics de sincrotró

3.2.1 *Beta cavity*

Aquest paràmetre indica com d'acoblada està la cavitat al sistema de generació extern que proporciona la potència, i ens mostraria la relació entre la potència dissipada a la cavitat i la potència dissipada a l'exterior de la cavitat, quan es perd la potència i la cavitat es descarrega.

$$\beta = \frac{Q_o}{Q_e} = \frac{P_e}{P_{diss}}$$

Equació 4. *Beta*

3.2.2 *Synchronous phase*

La fase síncrona és la fase en que els electrons veuen el voltatge necessari per recuperar l'energia que perden al fer una volta. Els electrons que es troben en aquesta fase són les partícules síncrones. Com que la cavitat de *RF* veu la meitat positiva de la forma d'ona, és a dir, la meitat del període hi ha dues fases on els electrons veuen el mateix voltatge però només un d'aquest és estable. Hi ha partícules dins els *bunch* d'electrons que la seva trajectòria no és exactament la de la fase síncrona, per tant aquestes partícules veuran un voltatge diferent i més o menys energia que les partícules síncrones. Aquestes partícules que veuen diferent voltatge, si n'han vist més del que tocava a la següent volta arribaran abans i per tant veuran menys del que els hi tocaria com a partícules síncrones, i si n'han vist menys, a la següent volta arribaran més tard i per tant rebran més voltatge, de manera que la fase síncrona ens ajuda a compactar longitudinalment els electrons. Si no estem a la part estable de l'ona aquest compactament no es produiria sinó que cada cop les partícules que arriben més tard arribarien més i més fins que es perdrien. En el diagrama de fasors (Equació 8), Φ_s és l'angle format entre I_g i V_c + 90º.

$$\boldsymbol{\Phi_s} = \pi - \operatorname{asin}\left(\frac{U_o}{V_{total}}\right)$$

Equació 5. *Synchronous phase*

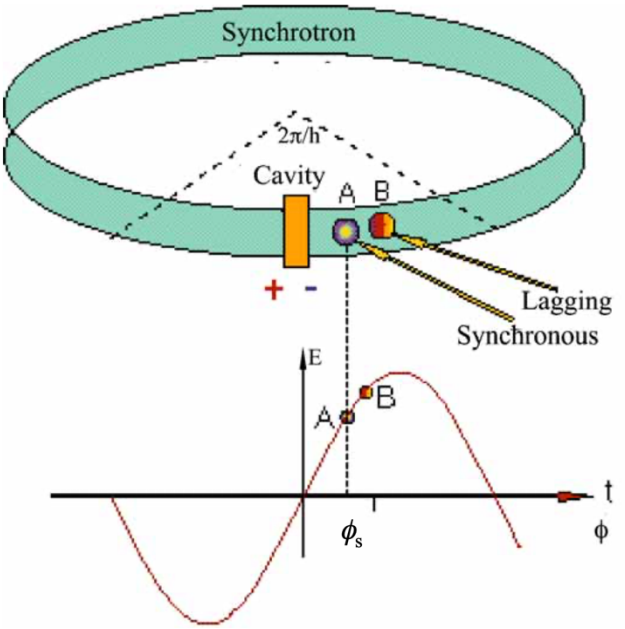


Fig 72. Synchronous phase. Font: *uspas*

3.2.3 *Shunt resistance*

Defineix el voltatge produït a la cavitat donada una potència de dissipació. No es tracta d'una resistència real sinó d'un factor que ens determina l'efectivitat amb la qual l'accelerador pot convertir la potència *RF* en gradient accelerador.

$$R_s = \frac{1}{2}\frac{|V|^2}{P_c}$$

Equació 6. Resistència Shunt

3.2.4 *Quality factor (Q_o)*

El *Quality factor* caracteritza les pèrdues de RF a la cavitat. Dona informació de l'energia que la cavitat pot emmagatzemar en relació a l'energia dissipada a la mateixa. Una cavitat RF amb un factor Q alt, és més eficient, malgrat això si s'utilitza un factor Q més baix es pot utilitzar un espectre de freqüències més ampli, sent més estable i menys sensible a les inestabilitats. El factor Q canvia significativament en funció de la resistència del material.

$$Q_o = \frac{\omega_o \cdot W}{P_c}$$

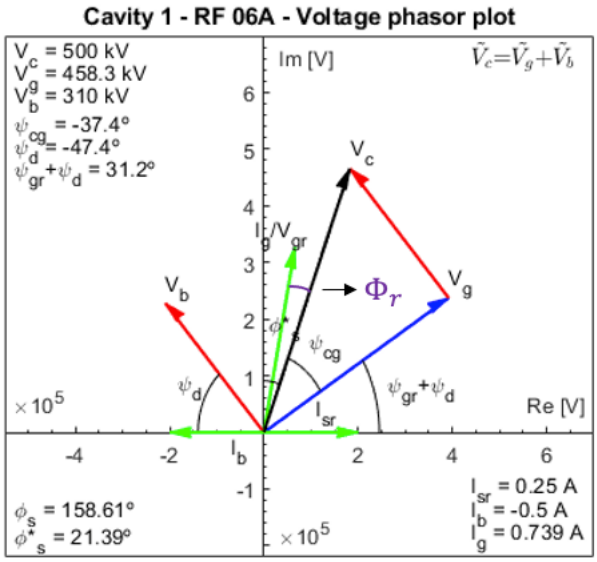
Equació 7. Quality Factor

On W és l'energia emmagatzemada a la cavitat i P_c és la potencia dissipada a la cavitat.

3.2.5 *Power Beam*

És la potència lliurada pel feix i es pot extreure la seva fórmula mirant el diagrama de fasors d'una cavitat, on V_c és el voltatge a la cavitat i I_{sr} és el corrent al *SR* i la

Φ_s l'angle format entre I_g i V_c + 90º. (*Equació 8*).



Equació 8. Diagrama de Fasors. Font: Ignasi Bellafont

$$P_{beam} = V_{acc}\, I_{sr} = V_c\, I_{sr}\, cos\ \Psi_{V_c - I_{sr}} = V_c\, I_{sr} \sin \Phi_s$$

Equació 9. *Power Beam*

3.2.6 Robinson detuning

$\boldsymbol{\Phi_r}$ és la desintonització de la cavitat que indica quants graus està desintonitzada la cavitat respecte l'angle de sintonització òptim quan $\boldsymbol{P_f}$ $\boldsymbol{i}$ $\boldsymbol{V_c}$ (en el diagrama de fasor $\boldsymbol{P_f}$correspon a $\boldsymbol{I_g}$) (*Equació 8*) estan en fase i serveix per tenir més estabilitat.

3.2.7 Coupling

És el coupling extra que necessita la cavitat per assolir el coupling òptim quan la cavitat té *beam loading*. Hi ha una $\beta_{op\ abs}$ que seria la ß necessària per aconseguir 0 W de potència reflectida amb un *detuning* òptim. $\beta_{op\ abs}$ = 1 ja que sense feix $\beta_b = 0$

$$\beta_{op\ abs} = 1 + \beta_b \rightarrow \beta_b = \frac{P_b}{P_c}$$

Equació 10.Coupling

3.2.8 *Power Forward*

Correspon a la potència enviada a la cavitat i ve determinada per:

$$P_{forward} = \frac{P_{cavity} \cdot (1 + \beta_b)}{1 - r_{rev}} \rightarrow P_{forward} = P_{beam} + P_{cavity} + P_{reflected}$$

Equació 11. *Power Forward*

3.2.9 ***Power Reverse***

És la potència que retorna reflectida des de la cavitat

$$P_{reflected} = P_{forward} \cdot r_{rev}$$

Equació 12. Potència reflectida

3.2.10 ***Reflection coeficient***

És el percentatge reflectit de la potència generada i ve donat per:

$$r_{rev} = \frac{(1 - \beta + \beta_b)^2 + \tan^2 \Phi_r \cdot (1 + \beta + \beta_b)^2}{(1 + \tan^2 \Phi_r) \cdot (1 + \beta + \beta_b)^2}$$

Equació 13. Coeficient de reflexió

3.2.11 ***Detuning frequency***

Degut a que la cavitat actua com un circuit RLC, seria la pròpia freqüència de ressonància del circuit.

$$f_{detuning} = f_{RF} \cdot \frac{\tan \psi_d}{2\, Q_0}$$

Equació 14. *Detuning frequency*

3.2.13 ***Beam loading detuning***

Obté la desintonització òptima, que és quan $\boldsymbol{P_f}$ $\boldsymbol{i}$ $\boldsymbol{V_c}$ (en el diagrama de fasors, $\boldsymbol{P_f}$ correspon a $\boldsymbol{I_g}$) estant en fase (*Equació 8*)

$$f_{beam\_loading} = \frac{f_{ressonance} \cdot I \cdot R_s \cdot \cos \Phi_{synchronous}}{Q_0 \cdot V_c}$$

Equació 15. *Beam loading detuning*

3.2.14 ***Loaded quality factor***

És el factor de qualitat global de la cavitat quan hi ha *beam loading*.

$$Q_l = \frac{Q_0}{\beta + 1}$$

Equació 16. *Loaded Quality Factor*

3.2.15 **Posició del plunger**

Dona la proporció que s'ha de moure el *plunger* per obtenir la variació de freqüència que volem.

$$\Delta y_{plunger} = \Delta f \frac{2cm}{1.1e6\ Hz}$$

Equació 17. *Posició del plunger*

3.3 Conclusions

Després d'aquesta secció s'extreuen les següents conclusions: la complexitat del sistema i dels múltiples paràmetres i elements que en formen part i la dificultat que tots els elements funcionin adequadament, ja que hi ha un sistema de control de la radiofreqüència que fa funcionar les sis cavitats independents però alhora fa necessària l'actuació sobre totes elles, per compensar els efectes que es produeix una sobre totes les altres. Així doncs, a grans trets ens justifica la necessitat de la creació d'aquest *script* per revisar tots aquests paràmetres.

4. Control System & scripting

En aquest apartat es descriurà tot el sistema de control dels acceleradors d'ALBA i s'explicaran detalladament totes les funcions de l'*script* AcopRFcheck.py

4.1 Control system

El sistema de control és el programari encarregat d'interaccionar entre els elements *hardware* de l'accelerador o les *beamlines* i els usuaris. El sistema de control a un nivell inferior, utilitza la plataforma Tango per controlar *software i hardware devices.* A les capes superiors, utilitza llenguatges de programació orientats a objectes, com *C ++, Python* o *Qt*, per implementar la interfície d'usuari, gràfica o amb la consola de comandes.

4.1.1 Tango

Tango és una plataforma de *software* lliure que es desenvolupa i manté mitjançant la col·laboració dels diferents centres de recerca que l'utilitzen: ALBA, ESRF, ELETTRA,SOLEIL, MAX-IV, DESY, SOLARIS. Es basa en un *Object Request Broker* (ORB) que permet tenir en una única base de dades (DB) la majoria dels dispositius de l'accelerador (*power suplies, RF*, etc.) o de les *Beamlines* (motors, estacions experimentals, detectors, etc.). A nivell de comunicació utilitza *CORBA* (*synchronous and asynchronous communication*) i *zeromq (*comunicació basada en events), donant control a diferents tipus de *devices*, utilitzant diferents tipus de *bus* (*serial, ethernet, GPIB) i* amb diferents protocols (*modbus over TCP, SCPI, vendor dependant*). Tango es desenvolupa amb *C ++* i també *Python* (*PyTango*) on tots els objectes són un *device* i es connecten als anomenats *device servers* (DS). Tots els *device servers* poden tenir associats un o diversos dispositius. Per exemple, molts motors (on cada motor és un dispositiu) es poden associar a un *Sardana Pool Device Server*.

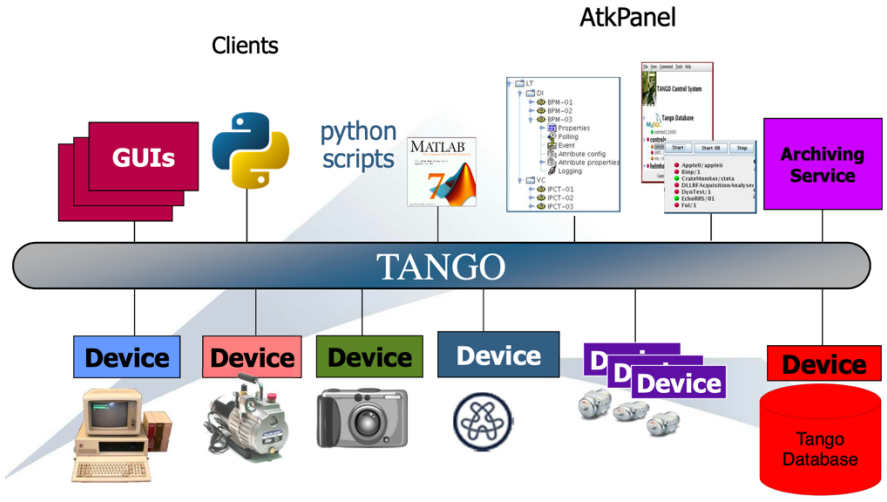


Fig 73. Tango

L'accelerador té el seu propi *TANGO DB* i cada *Beamline* també. Els *Softwares* relacionats amb *Tango* són *Astor* i *Jive*, on *Astor* permet inicialitzar i parar *Tango Device Servers* i *Jive* permet visualitzar l'organització de tots els *Tango Devices* i els seus atributs donat un *Tango Data Base*.

A continuació s'observa a l'*Astor* els *Tango Device Servers* associats a la RF (*Fig 74.* Astor)

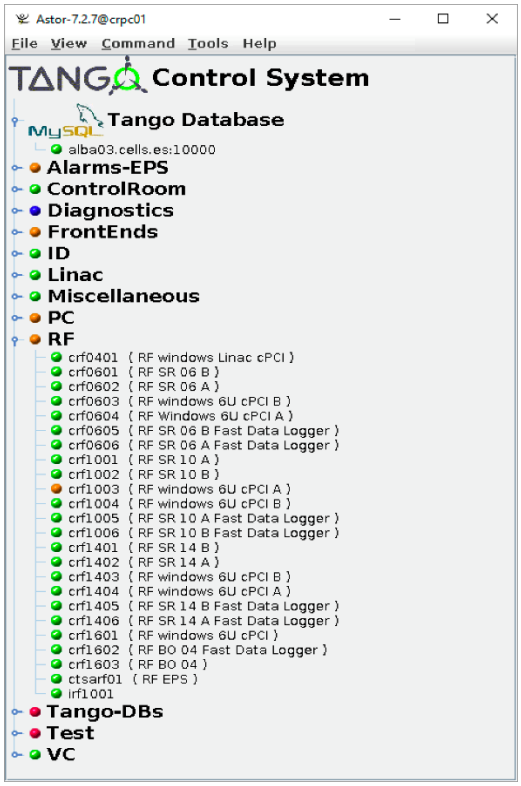


Fig 74. Astor

La següent imatge mostra el *Jive* amb els *devices* associats a la RF del sector 6.

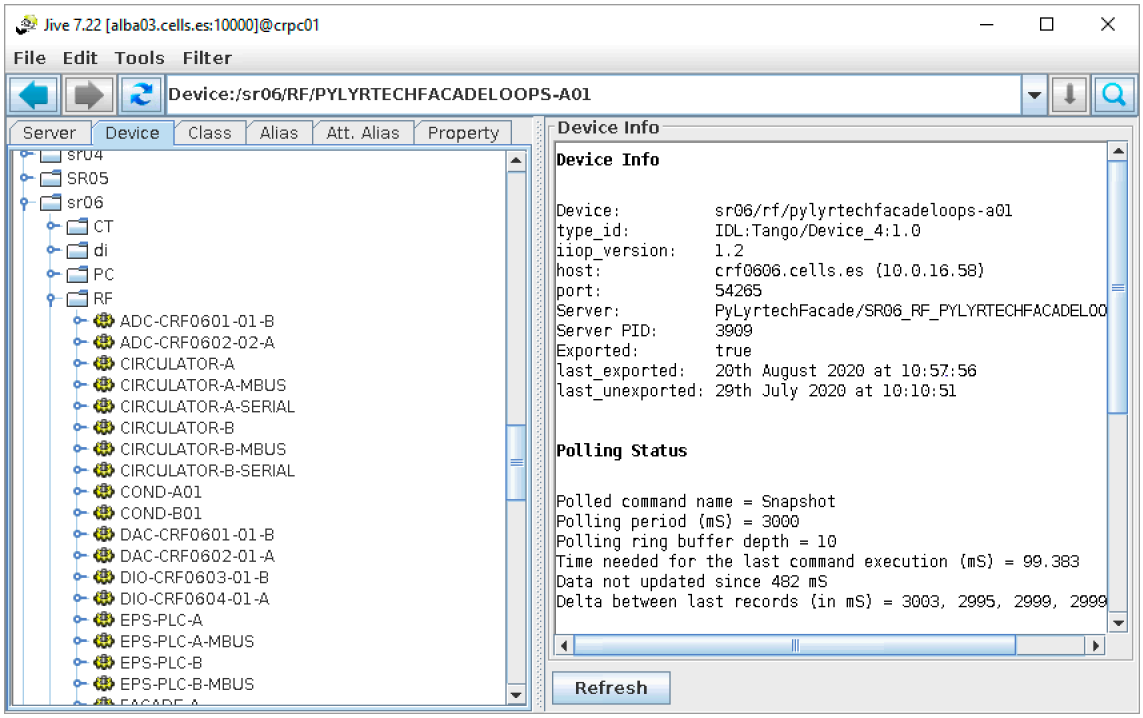


Fig 75. Devices al *Jive*

Exemple d'atributs de la RF al *Jive* on es veuen les característiques de l'atribut i el valor llegit.

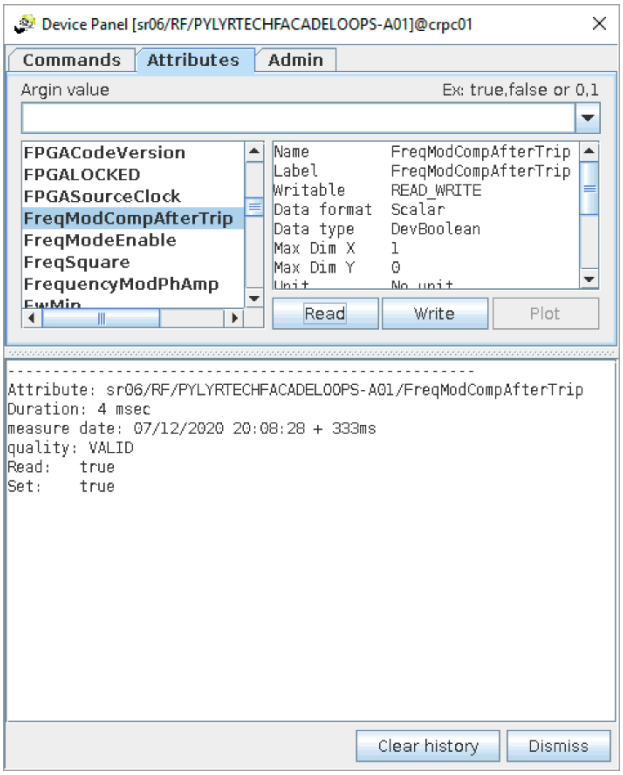


Fig 76. Atributs al *Jive*

4.1.2 Sardana

Sardana és un projecte *software* desenvolupat a ALBA més orientat a *beamlines* que pertany a la capa superior de *Tango*. Aquest *software* permet reunir diferents grups de motors i canals experimentals per a l'adquisició de dades, per tal de realitzar *scans* (en un *scan*, per exemple, es pot analitzar l'absorció de diferents longituds d'ona per una mostra determinada). El marc Sardana està dissenyat per amagar cert control intern (no cal conèixer el control intern per part de l'usuari) d'un grup de dispositius, això proporciona un conjunt d'operacions ampli que un usuari pot necessitar.

Així doncs, Sardana permet, per exemple, crear *pseudomotors*. Els *pseudomotors* són una unió de diversos motors físics. Per exemple, *l'scrapper* que s'utilitza a la sala de control quan es vol matar el feix, està formada per dos *jaws*, cada mandíbula es pot moure independentment gràcies a dos motors físics. Però normalment es vol controlar el *Gap* i el *Offset* de les mandíbules per tal d'anar reduint el *Gap* i anar matant el feix. Per aquest motiu, es fa servir Sardana, per agrupar els dos motors que mouen les mandíbules (motors físics) i crear dos *pseudomotors* anomenats *gap* i el *offset.*

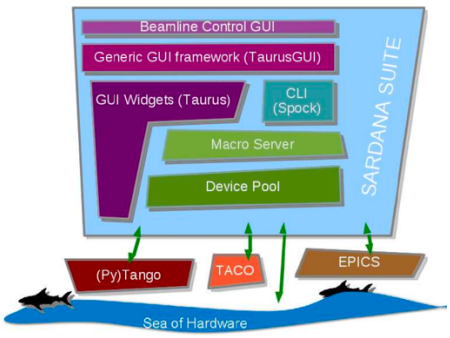


Fig 77. Esquema control *Beamline.*

La interfície de la línia de comandes (cli) de Sardana s'anomena *Spock*, que és un *script* que funciona com a *Command Line Interface Macroserver* client. Utilitza *IPython* per interactuar amb l'usuari, permetent-li utilitzar *Macros*. A Sardana hi ha:

- *Device Pool Server:* és el servidor que conté tots els *Devices* que són enmagatzemats al *Pool*.
- *MacroServer:* Dona la possibilitat d'executar *Macros* realitzant accions amb els elements del *Pool*.
- *Doors*: és el punt d'accés de l'usuari al sistema.

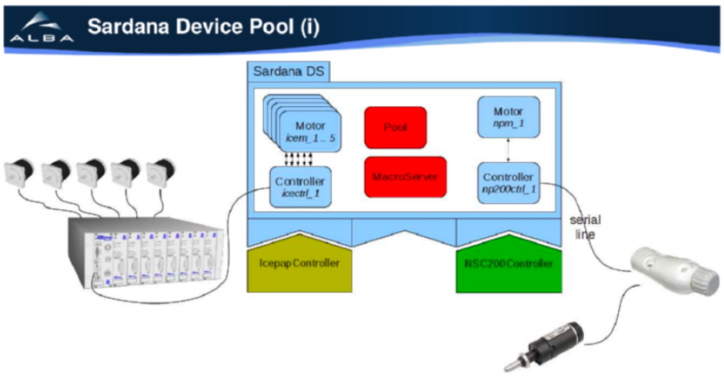


Fig 78. Esquema Sardana Device.

4.1.3 Taurus

Un altre element dins el sistema de control és *Taurus* que permet crear de manera senzilla *Graphical User Interfaces (*GUIs) per a l'accelerador i les *beamlines* per tal de poder visualitzar i controlar *devices*, com per exemple posicions del motor, realitzar *scans*, visualitzar dades en una tendència, executar macros de manera visual, etc. Taurus es desenvolupa utilitzant *Python* i en el cas de Taurus s'utilitza *Qt* per desenvolupar les aplicacions gràfiques, a través de la biblioteca *Python Qt*.

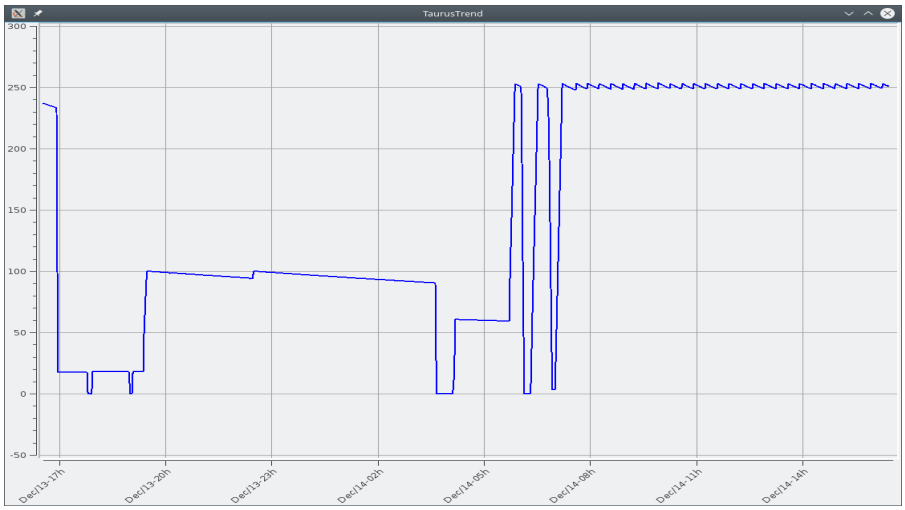


Fig 79. Taurustrend DCCT SR

4.2 AcopRFcheck.py

Aquest script de Python, està construït de forma modular i està dividit en dos classes que a la seva vegada contenen múltiples funcions. Hi ha una classe, *class Plant (object),* que serà on es crearan els objectes de cada planta de RF, és a dir, es crearan els objectes on es tenen tots els atributs i càlculs corresponents a cada una de les plantes (SR06A, SR06B,...) i es tindrà una altra classe, la *class checkRF (object)* que serà on s'executarà pròpiament l'*script*, és a dir, on es realitzaran tots els càlculs de tot el conjunt i on es farà el *check* de tots els atributs necessaris.

4.2.1 class check_RF()

Quan s'executa el fitxer es crida la classe *checkRF:*

```
if __name__ == '__main__':
    checkRF = checkRF()
```

Una vegada s'entra dins la classe, es crea una llista amb totes les plantes que es disposa i un diccionari que conté les cavitats i la seva *Beta* corresponent.

```
class checkRF (object):
    def __init__(self):

        ## List of RF plants ##
        self.Plants = ['sr06a', 'sr06b', 'sr10a', 'sr10b', 'sr14a', 'sr14b']

        ## Cavities Beta ##
        self.Beta = {'sr06a':2.701,
              'sr06b':2.687,
              'sr10a':2.4582,
              'sr10b':2.5467,
              'sr14a':2.678,
              'sr14b':2.5639,
              }
```

Es mira si hi ha corrent emmagatzemat al *SR* ja que no té sentit fer càlculs sobre el feix si no hi ha feix, i a part els càlculs no són fiables si no hi ha mínim 50 mA al *SR*.

En cas que no hi hagi corrent o sigui inferior a 50 mA, l'execució de l'*script* pot ser interessant per mirar que la configuració dels atributs de les plantes és correcte. Per tant *l'script* informa que el corrent és molt baix i que si es vol **només** fer un *check* dels atributs de les plantes, en cas negatiu acabaria *l'script* i en cas afirmatiu s'importen els atributs de les plantes cridant la classe *Plant* i les funcions *self.check_Attr()* i *self.printing()* i es fa el *check* corresponent. L'explicació de com s'importen els atributs i es fa el check s'explicarà extensament al llarg del document. S'activa un *flag* anomenat *test_mode* que posteriorment servirà per definir de diferent manera els resultats.

```
### check current in SR ##
self.current = taurus.Attribute('sr/di/dcct/averagecurrent').read().rvalue.magnitude
print(self.current)
if self.current < 30.0
    print("\nThe Current is too low, please inject more current before launch the script if you
    want to check the power beam parameters\n")
  reply = input("Doy you want to check the LLRF attributes  (yes/no) ")
  if reply == 'yes':
    self.sr06a = Plant('06a')
    self.sr06b = Plant('06b')
    self.sr10a = Plant('10a')
    self.sr10b = Plant('10b')
    self.sr14a = Plant('14a')
    self.sr14b = Plant('14b')
    self.rfPlants = ["sr06a","sr06b","sr10a","sr10b","sr14a","sr14b"]
    self.comparer = dict.fromkeys(self.rfPlants,{})
    self.booster_on = False
    self.test_mode = True
    self.check_Attr()
    self.printing()
    return None
  else:
    return None
```

El segon cas, amb corrent superior a 50 mA, seria el normal d'operació és quan ja es té suficient feix al *SR* i es volen realitzar tots els càlculs.

L'*script* mira si el *Super Contacting Wiggler* està obert o tancat ja que determinarà l'energia perduda per volta, que és un factor essencial a l'hora de calcular la fase síncrona (concepte explicat posteriorment) i es llegeix la freqüència del *master oscillator*.

```
## Get energy loss per turn checking SCW ##
try:
  if taurus.Attribute('sr/id/SCW01/magneticfield').read().rvalue > 2.0:
    self.e_loss = 1.1  # [MeV]
  else:
    self.e_loss = 1  # [MeV]
except ValueError:
  print('SCW is on Fault')
```

```
        self.e_loss = 1  # [MeV]

    ## Get RF frequency ##
    self.frequency = taurus.Attribute('SR09/rf/sgn-01/Frequency').read().rvalue.magnitude
```

Procedeix a generar els objectes de cada planta que contindran tots els atributs necessaris pels càlculs executant la classe *class Plant()*

```
## Get attributes of the plants ##
    self.sr06a = Plant('06a')
    self.sr06b = Plant('06b')
    self.sr10a = Plant('10a')
    self.sr10b = Plant('10b')
    self.sr14a = Plant('14a')
    self.sr14b = Plant('14b')
```

S'accedeix a la classe *Plant()*

4.2.2 class Plant()

Aquesta classe s'inicialitza passant-li el nom de la planta que es vol obtenir i es crida l'única funció que conté aquesta classe que és *read_properties()*

```
class Plant(object):
  """
  Returns all properties/atributes for a given plant.
  :param cavity: 06A, 06B, 10A, 10B, 14A, 14B

  """
  def __init__(self, name):
    self.name = name
    self.read_properties()
```

4.2.2.1 *def read_properties*()

Accedeix a la funció *def read_properties()* i es procedeix a llegir atributs dels *tango devices*. Es fa utilitzant una biblioteca pròpia del grup de *RF* en la que utilitza la funció *rf_attr_finder* a la que se li passa la planta *self.name*, el tipus de *Device*, és a dir si és *eps, facade, iot o llrf* i el nom de l'atribut per exemple *'TxST1_RFON'*, retornarà el *path* sencer d'on es troba aquest atribut. S'utilitza aquesta funció perquè dins al grup de RF s'ha volgut estandarditzar la manera de llegir *devices* ja que si es canvia de nom o de lloc un atribut, només s'ha de canviar a la biblioteca pròpia del grup i no cal canviar tots els *scripts* que miren atributs un a un.

Primer mira si els transmissors estan *ON*, la posició dels *Costubs* i del *Shutter* i en cas que es produeixi una excepció (error de lectura) pregunta si la planta està encesa o no, ja que d'això en dependran la resta de càlculs totals.

```
  def read_properties(self):

    #### Read the attributes in tango devices ###

    # Check if all plants are in RFON #
```

```
try:
   dev = rf_attr_finder(self.name, 'eps', 'TxST1_RFON')
   self.TxST1_RFON = taurus.Attribute(dev).read().rvalue
   dev = rf_attr_finder(self.name, 'eps', 'TxST2_RFON')
   self.TxST2_RFON = taurus.Attribute(dev).read().rvalue
   dev = rf_attr_finder(self.name, 'eps', 'COSTUB_TX1')
   self.costub_TX1 = taurus.Attribute(dev).read().rvalue
   dev = rf_attr_finder(self.name, 'eps', 'COSTUB_TX2')
   self.costub_TX2 = taurus.Attribute(dev).read().rvalue
   dev = rf_attr_finder(self.name, 'eps', 'SHUT_ST_OUT')
   self.shut_st_out = taurus.Attribute(dev).read().rvalue

except Exception as e:
   reply = input("Is this plant "+self.name+" ON?  (yes/no)")
   if reply == 'yes':
      self.TxST1_RFON = True
      self.TxST2_RFON = True
   else:
      self.TxST1_RFON = False
      self.TxST2_RFON = False
```

Ara revisa si el *TX1 (transmissor 1)* o bé el *TX2* estan encesos i el *shutter* no està posat. Això ho fa per saber si realment algun transmissor esta encès i per saber si s'està enviant potència a la cavitat, ja que si el *shutter* estés posat, s'estaria enviant la potència a la *Load*.

Procedeix a llegir el voltatge i la potència a la cavitat. Com en el cas anterior, si salta una excepció de lectura preguntarà si la planta està encesa i en cas afirmatiu demana introduir el voltatge de la cavitat, ja que poder tenir un voltatge total real és fonamental. En cas que els transmissors estiguin *OFF* i el *shutter* posat, activa un *flag* (*self.plant_on = False*) que indica que la planta està apagada i que per tant no la tindrà en compte pels càlculs futurs.

```
if (self.TxST1_RFON or self.TxST2_RFON) and self.shut_st_out == True:

   try:  # Check cavity voltage and power disipated in cavity if RFON #
      dev = rf_attr_finder(self.name, 'facade1', 'CAV_VOLT')
      self.cav_volt = taurus.Attribute(dev).read().rvalue.magnitude
      dev = rf_attr_finder(self.name, 'facade1', 'PDIS_CAV')
      self.pdis_cav = taurus.Attribute(dev).read().rvalue.magnitude
      self.plant_on = True
   except Exception as e:
      reply = input("Is this plant "+self.name+" ON?  (yes/no)")
      if reply == 'yes':
         volt = input("Insert the cavity voltage in "+self.name+" ON?  (550)")
         self.cav_volt = float(volt)
      else:
         self.plant_on = False
         print("The plant "+self.name+" is OFF")
else:
   self.plant_on = False
```

Un vegada ha comprovat si la planta està operativa, procedeix a llegir la *power forward* i la *power reverse* de cada *TX.* Per fer-ho abans mira que el *TX* està encès i que el *costub* corresponent al *TX* no està posat. En cas contrari considerarà que *power forward and reverse* són 0.

```
if self.plant_on == True:

        ## getting power forward and reverse for every plant, checking if RFON and costub is
out (symmetric mode) ##
        if self.TxST1_RFON == True and self.costub_TX1 == 1:
           try:
              dev = rf_attr_finder(self.name, 'iot1', 'TbPfwd')
              self.fw_iot1 = taurus.Attribute(dev).read().rvalue.magnitude
              dev = rf_attr_finder(self.name, 'iot1', 'TbPrev')
              self.rev_iot1 = taurus.Attribute(dev).read().rvalue.magnitude
           except Exception as e:
              exc_type, exc_obj, exc_tb = sys.exc_info()
              print('Exception type {} in line {}'.format(exc_type, exc_tb.tb_lineno))
              print('Error! Code: {c}, Message: {m}'.format(c=type(e).__name__, m=str(e)))
        else :
           self.fw_iot1 = 0
           self.rev_iot1 = 0

        if self.TxST2_RFON == True and self.costub_TX2 == 1:
           try:
              dev = rf_attr_finder(self.name, 'iot2', 'TbPfwd')
              self.fw_iot2 = taurus.Attribute(dev).read().rvalue.magnitude
              dev = rf_attr_finder(self.name, 'iot2', 'TbPrev')
              self.rev_iot2 = taurus.Attribute(dev).read().rvalue.magnitude
           except Exception as e:
              exc_type, exc_obj, exc_tb = sys.exc_info()
              print('Exception type {} in line {}'.format(exc_type, exc_tb.tb_lineno))
              print('Error! Code: {c}, Message: {m}'.format(c=type(e).__name__, m=str(e)))
        else :
           self.fw_iot2 = 0
           self.rev_iot2 = 0
```

Llegeix la fase síncrona de la cavitat:

```
    ## Getting synchronous phase ##
    try:
       dev = rf_attr_finder(self.name, 'facade1', 'PHASE2')
       self.sync_ph = taurus.Attribute(dev).read().rvalue.magnitude
    except Exception as e:
       exc_type, exc_obj, exc_tb = sys.exc_info()
       print('Exception type {} in line {}'.format(exc_type, exc_tb.tb_lineno))
       print('Error! Code: {c}, Message: {m}'.format(c=type(e).__name__, m=str(e)))
       print("The reading of synchronous phase in SR",self.name," is not working")
```

Procedeix a llegir tots els atributs de la planta necessaris per fer el *check* i els càlculs. En cas que hi hagi algun error de lectura, se li assigna el valor que hauria de tenir per defecte.

```
## Getting attributes for comparison ##
Try:

    dev = rf_attr_finder(self.name, 'llrf', 'AmpRefMin')
    self.AmpRefMin = taurus.Attribute(dev).read().rvalue.magnitude
    dev = rf_attr_finder(self.name, 'facade1', 'CAV_FW')
    self.cavityforw = taurus.Attribute(dev).read().rvalue.magnitude
    dev = rf_attr_finder(self.name, 'facade1', 'CAV_RV')
    self.cavity_reverse = taurus.Attribute(dev).read().rvalue.magnitude
    dev = rf_attr_finder(self.name, 'facade1', 'PBEAM')
    self.Pbeam = taurus.Attribute(dev).read().rvalue.magnitude
    dev = rf_attr_finder(self.name, 'facade2', 'plunger')
    self.plunger = taurus.Attribute(dev).read().rvalue.magnitude
    dev = rf_attr_finder(self.name, 'facade1', 'Enc1')
    self.encoder = taurus.Attribute(dev).read().rvalue.magnitude
    dev = rf_attr_finder(self.name, 'llrf', 'PHASEOFFSETROBINSON')
    self.robinson = taurus.Attribute(dev).read().rvalue.magnitude
    dev = rf_attr_finder(self.name, 'llrf', 'PHREFCAV')
    self.phref_cav = taurus.Attribute(dev).read().rvalue.magnitude
    #dev = rf_attr_finder(self.name, 'circ', 'CoilCtrlBias')  ## CHeck after shutdown
    #self.coilCtrlBias = taurus.Attribute(dev).read().rvalue.magnitude
    dev = rf_attr_finder(self.name, 'llrf', 'FreqModCompAfterTrip')
    self.FreqModCompAfterTrip = taurus.Attribute(dev).read().rvalue
    dev = rf_attr_finder(self.name, 'llrf', 'TimeDecayAmpMod')
    self.TimeDecayAmpMod = taurus.Attribute(dev).read().rvalue.magnitude
    dev = rf_attr_finder(self.name, 'llrf', 'TimeDecayPhaseMod')
    self.TimeDecayPhaseMod = taurus.Attribute(dev).read().rvalue.magnitude
    dev = rf_attr_finder(self.name, 'llrf', 'LookRef')
    self.LookRef = taurus.Attribute(dev).read().rvalue
    dev = rf_attr_finder(self.name, 'llrf', 'MixerOutputEnable')
    self.MixerOutputEnable = taurus.Attribute(dev).read().rvalue
    dev = rf_attr_finder(self.name, 'llrf', 'PhaseShiftEnable')
    self.PhaseShiftEnable = taurus.Attribute(dev).read().rvalue
    dev = rf_attr_finder(self.name, 'llrf', 'DisRFOnItckFromEPS')
    self.DisRFOnItckFromEPS = taurus.Attribute(dev).read().rvalue
    dev = rf_attr_finder(self.name, 'llrf', 'ConditioningEnable')
    self.ConditioningEnable = taurus.Attribute(dev).read().rvalue
    dev = rf_attr_finder(self.name, 'llrf', 'FreqModeEnable')
    self.FreqModeEnable = taurus.Attribute(dev).read().rvalue
    dev = rf_attr_finder(self.name, 'llrf', 'PositivePhaseModulation')
    self.PositivePhaseModulation = taurus.Attribute(dev).read().rvalue
    dev = rf_attr_finder(self.name, 'llrf', 'PhaseStepModulation')
    self.PhaseStepModulation = taurus.Attribute(dev).read().rvalue
    dev = rf_attr_finder(self.name, 'llrf', 'TuningEnable')
    self.TuningEnable = taurus.Attribute(dev).read().rvalue
    dev = rf_attr_finder(self.name, 'llrf', 'TuningPosEn')
    self.TuningPosEn = taurus.Attribute(dev).read().rvalue
    dev = rf_attr_finder(self.name, 'llrf', 'AutomaticParkingDisable')
    self.AutomaticParkingDisable = taurus.Attribute(dev).read().rvalue
    dev = rf_attr_finder(self.name, 'llrf', 'ParkupEnable')
    self.ParkupEnable = taurus.Attribute(dev).read().rvalue
```

```
dev = rf_attr_finder(self.name, 'llrf', 'FieldFlatness')
self.FieldFlatness = taurus.Attribute(dev).read().rvalue
dev = rf_attr_finder(self.name, 'llrf', 'FieldFlatnessPos')
self.FieldFlatnessPos = taurus.Attribute(dev).read().rvalue
dev = rf_attr_finder(self.name, 'llrf', 'AmpCell2Gain')
self.AmpCell2Gain = taurus.Attribute(dev).read().rvalue.magnitude
dev = rf_attr_finder(self.name, 'llrf', 'AmpCell4Gain')
self.AmpCell4Gain = taurus.Attribute(dev).read().rvalue.magnitude
dev = rf_attr_finder(self.name, 'llrf', 'FieldFlatPercentage')
self.FieldFlatPercentage = taurus.Attribute(dev).read().rvalue.magnitude
dev = rf_attr_finder(self.name, 'llrf', 'MovePLG1')
self.MovePLG1 = taurus.Attribute(dev).read().rvalue
dev = rf_attr_finder(self.name, 'llrf', 'MovePLG2')
self.MovePLG2 = taurus.Attribute(dev).read().rvalue
dev = rf_attr_finder(self.name, 'llrf', 'FrequencyModPhAmp')
self.FrequencyModPhAmp = taurus.Attribute(dev).read().rvalue
dev = rf_attr_finder(self.name, 'llrf', 'AmplitudeAmpMod')
self.AmplitudeAmpMod = taurus.Attribute(dev).read().rvalue.magnitude
dev = rf_attr_finder(self.name, 'llrf', 'PhaseDelayAmpMod')
self.PhaseDelayAmpMod = taurus.Attribute(dev).read().rvalue.magnitude
dev = rf_attr_finder(self.name, 'llrf', 'TimeDecayAmpMod')
self.TimeDecayAmpMod = taurus.Attribute(dev).read().rvalue.magnitude
dev = rf_attr_finder(self.name, 'llrf', 'TimeDelay')
self.TimeDelay = taurus.Attribute(dev).read().rvalue.magnitude
dev = rf_attr_finder(self.name, 'llrf', 'ModulationType')
self.ModulationType = taurus.Attribute(dev).read().rvalue.magnitude
dev = rf_attr_finder(self.name, 'llrf', 'AmplitudePhaseMod')
self.AmplitudePhaseMod = taurus.Attribute(dev).read().rvalue.magnitude
dev = rf_attr_finder(self.name, 'llrf', 'TimeDecayPhaseMod')
self.TimeDecayPhaseMod = taurus.Attribute(dev).read().rvalue.magnitude
dev = rf_attr_finder(self.name, 'llrf', 'AbsoluteTimeDecay')
self.AbsoluteTimeDecay = taurus.Attribute(dev).read().rvalue.magnitude

dev = rf_attr_finder(self.name, 'llrf', 'AmpRefCav')
self.AmpRefCav = taurus.Attribute(dev).read().rvalue
dev = rf_attr_finder(self.name, 'llrf', 'FwMinAmpPh')
self.FwMinAmpPh = taurus.Attribute(dev).read().rvalue
dev = rf_attr_finder(self.name, 'llrf', 'FwMin')
self.FwMin = taurus.Attribute(dev).read().rvalue
dev = rf_attr_finder(self.name, 'llrf', 'ConditioningDutyCycle')
self.ConditioningDutyCycle = taurus.Attribute(dev).read().rvalue
dev = rf_attr_finder(self.name, 'llrf', 'LoopInput')
self.LoopInput = taurus.Attribute(dev).read().rvalue
dev = rf_attr_finder(self.name, 'llrf', 'RFSwitchControlAuto')
self.RFSwitchControlAuto = taurus.Attribute(dev).read().rvalue
dev = rf_attr_finder(self.name, 'llrf', 'PISlaveLoopsEnable')
self.PISlaveLoopsEnable = taurus.Attribute(dev).read().rvalue
dev = rf_attr_finder(self.name, 'llrf', 'CmdStart')
self.CmdStart = taurus.Attribute(dev).read().rvalue
dev = rf_attr_finder(self.name, 'llrf', 'Stepstodetunecavityaftertrip')
self.Stepstodetunecavityaftertrip = taurus.Attribute(dev).read().rvalue
```

```
        self.check_attrib = True
except Exception as e:
        exc_type, exc_obj, exc_tb = sys.exc_info()
        print('Exception type {} in line {}'.format(exc_type, exc_tb.tb_lineno))
        print("\n Check Plant ",self.name,"\n")
        print('Error! Code: {c}, Message: {m}'.format(c=type(e).__name__, m=str(e)))
        print("Check Plant ",self.name)
        self.check_attrib = False
        self.FreqModCompAfterTrip = True
        self.LookRef = False
        self.MixerOutputEnable = True
        self.PhaseShiftEnable = True
        self.DisRFOnItckFromEPS = False
        self.ConditioningEnable = False
        self.FreqModeEnable = False
        self.PositivePhaseModulation = False
        self.PhaseStepModulation = False
        self.TuningEnable = True
        self.TuningPosEn = False
        self.AutomaticParkingDisable = False
        self.ParkupEnable = True
        self.FieldFlatness = False
        self.FieldFlatnessPos = False
        self.AmpCell2Gain = 0
        self.AmpCell4Gain = 0
        self.FieldFlatPercentage = 0
        self.MovePLG1 = True
        self.MovePLG2 = False
        self.FrequencyModPhAmp = 1
        self.AmplitudeAmpMod= 1
        self.PhaseDelayAmpMod = 0
        self.TimeDecayAmpMod = 1.5
        self.TimeDelay = 0
        self.ModulationType = 2
        self.AmplitudePhaseMod = 1
        self.TimeDecayPhaseMod = 1
        self.AbsoluteTimeDecay = 1
```

Un cop ha llegit i emmagatzemat tots els atributs que contindrà cada objecte torna a la classe principal *check_RF()* retornant l'objecte planta i procedeix a executar les diferents funcions. Es crea el diccionari *self.comparer*, en el que es tindran totes les plantes i a mesura que es vagin executant les funcions, aquest diccionari tindrà les plantes amb els corresponents atributs i càlculs, i a la vegada dirà si estan *OK* o no. Si no ho estan contindrà el missatge que ha de mostrar i que indicarà a qui executi l'*script,* que ha de fer. Aquesta informació es mostrarà un cop s'hagin executat totes les funcions i finalment s'executi la funció d'imprimir.

4.2.1 class check_RF()

```
    self.comparer = {}
    self.test_mode = False
```

```
## Calculate parameters ##
self.rfplants_active()
self.total_voltage()
self.calc_pforward_plant()
self.calc_pbeam()
self.check_pforw()
self.status_RF()
self.check_Robinson()
self.calc_sync_ph()
self.calc_det_angle()
self.simulator()
self.check_diff_pforw()
self.check_diff_revers()
self.check_diff_plunger()
self.check_sync_Phase()
self.check_BO()
self.check_Attr()
self.printing()
```

4.2.1.1 def rfplants_active()

Mira les plantes que estan funcionant, hi ha diversos escenaris, ja que en el posterior *Status Summary* (4.2.1.12 status_RF()) on s'ensenya l'estat de totes les plantes, hi ha la possibilitat que la planta estigui enviant potència a la cavitat, en que es mostraria per pantalla el voltatge a la cavitat i la seva fase síncrona, la possibilitat que el *shutter* estigui tancat i per tant la planta estigui enviant potència a la *Load* o bé, que la planta estigui *OFF*.

```
## Check RF Plants running  and get plants info. for the summary##
  def rfplants_active(self):
    self.rfPlants = []
    self.status = {}
    for p in self.Plants:

      if getattr(self,p).plant_on == True and getattr(self, p).shut_st_out == True:
        self.rfPlants.append(p)
        self.status.update({p:{}})
        self.status[p]["Cavity Voltage"] = getattr(self, p).cav_volt
        self.status[p]["Sync.Phase"] = getattr(self, p).sync_ph

      elif getattr(self,p).plant_on == True and getattr(self, p).shut_st_out == False:
        self.status.update({p:{}})
        self.status[p]["Cavity Voltage"] = "to the load"
        self.status[p]["Sync.Phase"] = 0

      elif getattr(self,p).plant_on == False:
        self.status.update({p:{}})
        self.status[p]["Cavity Voltage"] = "Not in RFON"
        self.status[p]["Sync.Phase"] = 0
```

4.2.1.2 def total_voltage()

Calcula el voltatge total que es té, sumant els voltatges individuals de cada planta.

```
## the total voltage ##
   def total_voltage(self):
      self.voltage_total =[]
      for p in self.rfPlants:
         self.voltage_total.append(getattr(self, p).cav_volt)
      self.voltage_total = sum(self.voltage_total)
      print (self.voltage_total)
```

4.2.1.3 def check_pforw()

A la primera part de la funció fa un loop que calcula *mean power forward and Standard deviation* de totes les plantes. Això servirà perquè al següent *loop* que hi ha dins la funció, miri si la suma dels dos *TX* de cada planta està per damunt de la mitjana de les potències sumant-hi dos cops la desviació estàndard. Això es fa perquè teòricament totes les plantes han d'estar lliurant la mateixa potència aproximadament, llavors s'aplica un marge que seria 2 vegades la desviació estàndard sobre la mitjana. Si no és així segurament hi ha alguna cosa malament a la planta i s'ha de revisar. L'*script* quan fa aquest *loop* guarda a la variable *self.comparer* l'atribut *iots_fw.*

```
## Chek total IOT pforward, mean and stdv ##
   def check_pforw(self):

      self.total_pforw = []

      ## Loop to get Mean power and standard deviation
      for p in self.rfPlants:
         self.total_pforw.extend((getattr(self, p).fw_iot1, getattr(self, p).fw_iot2))
         self.total_pforw_mean = np.mean(self.total_pforw)
         self.total_pforw_std = np.std(self.total_pforw)

       ## loop to check the forward power
      for p in self.rfPlants:
         self.comparer.update({p:{}})

         msg = ""

         ##forward power of IOTs in comparison with the mean of the other plants

           if (getattr(self, p).fw_iot1 and getattr(self, p).fw_iot2) >
           (self.total_pforw_mean+(2*self.total_pforw_std)):

            msg += ('The forward power in IOTs are: '+str(round(getattr(self, p).fw_iot1,2)) +
                  ' and '+ str(round(getattr(self, p).fw_iot2,2)) +
                  ' the average power in the other plants is '+str(round(self.total_pforw_mean,2)))

            msg += ('Call RF group as soon as possible\n')
```

```
            #print ('test')
        else:
            msg = 'OK'
        self.comparer[p]["iots_fw"] = msg
```

4.2.1.4 def calc_sync_ph ()

Procedeix al càlcul de la fase síncrona (3.2.2 *Synchronous phase*) en funció d'*Energy loss per turn* i del voltatge total de *RF* que es té al *SR* i imprimirà per pantalla un missatge en que diu si els IDs estan oberts o tancats, el voltatge total i la fase síncrona que s'hauria de tenir a les cavitats.

```
    ## Calculate synchronous phase ##
    def calc_sync_ph(self):
        if self.current > 1:
            self.phase = np.arcsin (self.e_loss / (self.voltage_total / 1000))
            self.sync_ph_calc = (np.pi - self.phase)*180/np.pi
        if self.e_loss == 1.1:
            print ("\n**********************")
            print ('The IDs are closed, the Total Voltage is '+str(round(self.voltage_total,2))+' MV and
the theoretical synchronous phase is '+str(round(self.sync_ph_calc,1)))
            print ("**********************\n")
        else :
            print ("\n--------------------")
            print ('The IDs are open, the Total Voltage is '+str(round(self.voltage_total,2))+' MV and
the theoretical synchronous phase is '+str(round(self.sync_ph_calc,1)))
            print ("--------------------\n")
```

4.2.1.5 calc_pbeam ()

Calcula la potència lliurada al feix d'electrons.

```
    ## Power Beam ##
    def calc_pbeam(self):
        for p in self.rfPlants:
            getattr(self, p).pBeam = getattr(self, p).cav_volt * (self.current / 1000) \
                                * np.sin(((180 - getattr(self, p).sync_ph) * np.pi) / 180)
```

4.2.1.6 calc_det_angle ()

Calcula l'angle de desintonització de la cavitat, en el que fa un *loop* començant des d'una *Pforward* = 1 i un *Det_angle* = 0 i va fent iteracions mentre aquesta *Pforward* que s'anirà calculant, sigui inferior a la *Pforward* que es llegeix a la cavitat. Per calcular aquesta *Pforward* procedeix a calcular el *coupling* (*Equació 10*), a continuació el *reflection coeficient* (*Equació 13*) i finalment obté la *Pforward* (*Equació 11*), mentre es vagi complint la condició s'incrementa 0,01 el *Det_angle*, com que aquest factor s'utilitza per calcular el *reflection coeficient* a cada iteració s'obté una *Pforward* més gran fins a sortir del *loop*.

```
## Calculate detuning angle ##
    def calc_det_angle(self):
```

```
for p in self.rfPlants:
    Pforward = 1
    Det_angle = 0
    while ( Pforward < getattr(self, p).pForward and getattr(self, p).pForward!= 0 ):

        if getattr(self, p).cav_volt >= 0:

            Bbeam =  getattr(self, p).pBeam / getattr(self, p).pdis_cav

            Tg = np.tan((Det_angle * np.pi) / 180) ** 2

            R = ((1-self.Beta[p]+Bbeam)**2+(1+self.Beta[p]+Bbeam)**2*Tg) /
            (((1+self.Beta[p]+Bbeam)**2)*(1+Tg))

            Pforward = (getattr(self, p).pdis_cav * (1+Bbeam)) / (1-R)
        else:
            Pforward = Pforward + 1
        Det_angle = Det_angle + 0.01
    getattr(self, p).det_angle = Det_angle-0.01
```

4.2.1.7 simulator ()

En aquesta funció el que farà és calcular a partir de les dades teòriques per a cada cavitat, els paràmetres *de Pfroward i Prevers* i posició del plunger (*r_plunger)* per contrastar-los amb les dades llegides de les cavitats. Els càlculs segueixen el següent ordre: calcula potència dissipada a la cavitat (), potència lliurada al feix (*Equació 9*), *coupling* (*Equació 10*), coeficient de reflexió (*Equació 13*), *Pforward (Equació 11), Prevers (Equació 12), Detuning frequency (Equació 14), Beam loading detuning (Equació 15), Posició del plunger (Equació 17)*

def simulator (self):

```
## Parameters #
Rs = 3.3e6 # Shunt resistance
Qo = 29500 # Quality factor
Working_Frequency = 499650000

## Simulated parameters ##
try:
   for p in self.rfPlants:

      ## Power dispated in the cavities #
      #print("cavity volt",getattr(self, p).cav_volt)
      getattr(self, p).pdis_cav_sim = ((getattr(self, p).cav_volt * 1000)**2 \
                           / ( Rs*2*1000))
      #print("pdis_cav",getattr(self, p).pdis_cav_sim)
      ## Power Beam ##
      getattr(self, p).pBeam_sim = (getattr(self, p).cav_volt * self.current \
                        * np.sin(self.phase)) / 1000

      ## Coupling ##
```

```
        getattr(self, p).b_beam_sim =  getattr(self, p).pBeam_sim \
                              / getattr(self, p).pdis_cav_sim
        #print ("b_Beam_sim in "+str(p), getattr(self, p).b_beam_sim)

        ## Reflection coefficient ##
        getattr(self, p).tg_sim = np.tan(getattr(self, p).det_angle * np.pi / 180)**2
        getattr(self, p).r_sim =((1-self.Beta[p]+getattr(self, p).b_beam_sim)**2 \
                        + (1+self.Beta[p]+getattr(self, p).b_beam_sim)**2 * getattr(self, p).tg_sim)
\
                        /((1+self.Beta[p]+getattr(self, p).b_beam_sim)**2*(1+getattr(self,
p).tg_sim))

        #print ("r_sim in "+str(p), getattr(self, p).r_sim)

        ## forward power simulated ##
           getattr(self, p).pForw_sim = getattr(self, p).pdis_cav_sim * ((1+getattr(self,
           p).b_beam_sim)\
                           /(1-getattr(self, p).r_sim))

        #print("pForw_sim", getattr(self, p).pForw_sim,"\n")

        ## reverse power simulated ##
        getattr(self, p).pRev_sim = getattr(self, p).pForw_sim * getattr(self, p).r_sim
        #print(p,getattr(self, p).pRev_sim)

        ## Detuning frequency ##
        getattr(self, p).det_Freq_sim = self.frequency * np.tan(getattr(self, p).det_angle *
np.pi/180)/2/Qo

        ## Ressonace frequency ##
        getattr(self, p).res_Freq_sim = Working_Frequency - getattr(self, p).det_Freq_sim ##
check

        ## Beam loading detuning ##
        getattr(self, p).beam_det_sim = ((getattr(self, p).res_Freq_sim * (self.current/1000) *
Rs)\
                              / (Qo * (getattr(self, p).cav_volt*1000))) * np.cos(self.phase)

        getattr(self, p).plunger_sim = ((2*getattr(self, p).beam_det_sim) / 1100000)

        ## plunger due to robinson + beam detuning
        getattr(self,p).q = Qo / (self.Beta[p]+1)
        #print(getattr(self,p).q)
        getattr(self,p).robinson_Freq = ((np.tan((getattr(self,p).robinson*np.pi)/180) *
                                 getattr(self, p).res_Freq_sim) / (-2*getattr(self,p).q))
        #print(getattr(self,p).robinson_Freq)
        getattr(self,p).Robinson_fcn = (2*getattr(self,p).robinson_Freq) / 1100000
        getattr(self,p).r_plunger = getattr(self,p).plunger + getattr(self, p).plunger_sim +
getattr(self,p).Robinson_fcn

        #print(getattr(self,p).r_plunger,getattr(self,p).plunger, getattr(self, p).plunger_sim
,getattr(self,p).Robinson_fcn )
```

```
except Exception as e:
    exc_type, exc_obj, exc_tb = sys.exc_info()
    print('Exception type {} in line {}'.format(exc_type, exc_tb.tb_lineno))
    print('Error! Code: {c}, Message: {m}'.format(c=type(e).__name__, m=str(e)))
```

4.2.1.8 check_diff_pforw ()

Mira la diferència entre la potència *forward* calculada i la real llegida a cada cavitat i en cas que la diferència de potència sigui superior al 10 % guardarà a la variable *self.comparer* el missatge corresponent a ser imprès posteriorment. Si ens inferior al 10 %, guardarà un OK corresponent a la diferència de potència.

```
    ## Check difference between calculated power forward and measured power forward ##
  Check after shutdown
    def check_diff_pforw(self):
        for p in self.rfPlants:
            diffPforwr = (abs(getattr(self, p).pForward - getattr(self, p).pForw_sim) / getattr(self,
  p).pForw_sim)*100
            if abs(diffPforwr) <= 10:
                self.comparer[p]['DiffPforwr'] = 'OK'
            else:
                self.comparer[p]['DiffPforwr'] = ('The Power forward is '+ str(round(getattr(self,
  p).pForward,2)) + ' and should be '+ str(round(getattr(self, p).pForw_sim,2)))
```

4.2.1.9 check_diff_revers()

Mira la diferència entre la potència *reverse* calculada i la real llegida a cada cavitat i en cas que la diferència de potència sigui superior a 1 kW, guardarà a la variable *self.comparer* el missatge corresponent a ser imprès posteriorment, si ens inferior a 1 kW, guardarà un OK corresponent a la diferència de potència.

```
    ## Check difference between calculated power reverse and measured power reverse ##
    def check_diff_revers(self):
        for p in self.rfPlants:
            diffrevers = (abs(getattr(self, p).cavity_reverse - getattr(self, p).pRev_sim))
            if abs(diffrevers) <= 1:
                self.comparer[p]['Diffrevers'] = 'OK'
            else:
                self.comparer[p]['Diffrevers'] = ('The Power reverse is '+ str(round(getattr(self,
  p).cavity_reverse,2)) +' and should be '+ str(round(getattr(self, p).pRev_sim,2)))
```

4.2.1.10 check_plunger()

Aquesta funció compara els valors teòrics de la posició del *plunger* i el valor real que tenim al *encoder.*

```
  ## Check difference between calculated plunger and measured plunger ##
    def check_diff_plunger(self):
        for p in self.rfPlants:
```

```
diffplunger = abs(getattr(self,p).r_plunger - getattr(self, p).plunger)
if abs(diffplunger) <= 0.08:
   self.comparer[p]['Plunger'] = 'OK'
else:
   self.comparer[p]['Plunger'] = ('The Plunger is '+ str(round(getattr(self, p).plunger,2)) +
                                  ' and should be '+ str(round(getattr(self, p).r_plunger,2)))
```

4.2.1.11 check_sync_phase()

Aquesta funció té dues parts: en un primera part mira la fase síncrona de cada planta i la compara amb la teòrica, si la diferència és mes gran de 3º, mostrarà per pantalla la fase síncrona que es té i la que hauria de tenir, sinó ens mostrarà la fase síncrona com *OK.* Com en tots els altres casos la informació es guarda a la variable *self.comparer*.

En la segona part, si hi ha més d'una planta amb la fase síncrona malament, es mira la que té la major diferència, degut a que no es poden moure totes les fases que estiguin malament a l'hora, quan es mou una fase, les altres tendeixen a equilibrar-se. Llavors *l'script* diu que es mogui la fase els graus necessaris en una cavitat i després que es torni a executar per veure la quantitat necessària de moure la següent fase.

```
## Check plants synchronous phase in comparison with calculated syn.ph ##
  def check_sync_Phase(self):
     diff_Sync ={}

     for p in self.rfPlants:
        msg =""
        diff_Sync[p] = self.sync_ph_calc - getattr(self, p).sync_ph

        if abs(diff_Sync[p]) <= 3:
           self.comparer[p]['diff_Sync'] = 'OK'
        else:
           msg+= ('Synchr. Phase is \033[1m'+str(round(getattr(self, p).sync_ph,2)) +
                 '\033[0m  and should be \033[1m'+str(round(self.sync_ph_calc,2)) +
                 '\033[0m. The difference is \033[1m'+str(round(diff_Sync[p],2))+'\033[0m\n')
           #self.comparer[p]['diff_Sync'] = msg
           self.comparer[p]['value_change_ph'] = msg
           if abs(diff_Sync[p]) >= 20 :
              msg +=('If the difference is higher than 20 degrees call to RF')
              self.comparer[p]['change_Sync'] = msg

     ## Calculating the phase's change

     v=list(map(abs,diff_Sync.values()))
     #print(v)
     k=list(diff_Sync.keys())
     #print(k)
     maxvalue = max(v)
     maxvalue_idx = v.index(maxvalue)
     #print (maxvalue_idx)
     p = k[maxvalue_idx]

     if maxvalue > 3:
```

```
        msg += ("Change the Cav Ph(Deg) setting \033[1m"+str(round(diff_Sync[p],2))+'\033[0m
    degrees in ctrgen GUI and launch the program again ')
        msg += ("\nRemember to update the whiteboard\n ")
        self.comparer[p]['value_change_ph'] = msg
```

4.2.1.12 status_RF()

Aquest funció s'encarrega d'imprimir per pantalla l'*Status de RF* que s'ha calculat a la funció 4.2.1.1 def rfplants_active()

```
  ## Check status RF, cavity Voltage and synchronous phase ##
     def status_RF(self):

        df = pd.DataFrame.from_dict(self.status, orient = 'index')
        df.style.format({'Cavity Voltage':"{:.2%}".format})

        print ("\n**********")
        print ('      STATUS RF')
        print ("**********\n")
        print (df)
```

4.2.1.13 check_Robinson()

Mira si l'atribut *PHASEOFFSETROBINSON* a les cavitats té un valor de -18 º, en cas negatiu guardarà el missatge que s'ha d'escriure -18 a la cavitat i *device* corresponent.

```
     # Checking phase robinson attr ##
     def check_Robinson(self):

        for plant in self.rfPlants :

           try:
              if plant == "sr10a":
                 self.devicellrf ="PICO-LLRF"
              else:
                 self.devicellrf ="PYLYRTECHFACADELOOPS"
              ### Check robinson attr##
              if getattr(self, plant).robinson < -17.5 or getattr(self, plant).robinson > -18:
                 msg = ""
                 msg += 'The Phase Offset Robinson is different from -10\n'
                       msg += 'Write -10 in jive in '+plant.upper()+'/RF/'+self.devicellrf+'-
           X01/PHASEOFFSETROBINSON'
                 self.comparer[plant]['robinson'] = msg
              else:
                 self.comparer[plant]['robinson'] = 'OK'

           except Exception as e:
              exc_type, exc_obj, exc_tb = sys.exc_info()
              print('Exception type {} in line {}'.format(exc_type, exc_tb.tb_lineno))
              print('Error! Code: {c}, Message: {m}'.format(c=type(e).__name__, m=str(e)))
```

4.2.1.14 check_BO()

Aquesta funció per revisar el *booster* està dividida en diverses parts: la primera part llegeix un fitxer tipus *yaml* , que és un fitxer de text ,que conté els atributs amb el seu valor per defecte, que més tard s'utilitzen per comparar amb els valors llegits dels *devices*. Es crea un diccionari del tipus que s'ha creat pels atributs del *SR* que es dirà *self.comparer_BO* que contindrà tots els atributs del *booster* amb el seus valors.

```
def check_BO(self):

    ## Load yaml file with attr values  ##
    try:
        yaml_file_path = Path('/data/test/Oriol/Python/acopRFcheck/bo_attr.yaml')
        self.bo_attr_Dict = load_yaml(yaml_file_path)
        #print (self.bo_attr_Dict)
    except Exception as e:
        exc_type, exc_obj, exc_tb = sys.exc_info()
        print('Exception type {} in line {}'.format(exc_type, exc_tb.tb_lineno))
        print('Error! Code: {c}, Message: {m}'.format(c=type(e).__name__, m=str(e))

    self.compare_BO = {}
```

A la segona part es llegeixen tots els atributs del *booster* i s'incorporen al diccionari abans esmentat.

```
try:

        self.compare_BO['CoilCtrlBias'] = taurus.Attribute('bo/RF/CIRCULATOR-
01/CoilCtrlBias').read().rvalue.magnitude
        self.compare_BO['ManualMode'] = taurus.Attribute('bo/RF/CIRCULATOR-
01/ManualMode').read().rvalue
        self.compare_BO['AmpRampInit'] =
taurus.Attribute('bo/RF/PYLYRTECHFACADELOOPS/AmpRampInit').read().rvalue.magnitude
        self.compare_BO['AmpRampEnd'] =
taurus.Attribute('bo/RF/PYLYRTECHFACADELOOPS/AmpRampEnd').read().rvalue.magnitude
        self.compare_BO['RAMPENABLE'] =
taurus.Attribute('bo/RF/PYLYRTECHFACADELOOPS/RAMPENABLE').read().rvalue
        self.compare_BO['AmpRefCav']=
taurus.Attribute('bo/RF/PYLYRTECHFACADELOOPS/AmpRefCav').read().rvalue.magnitude
        self.compare_BO['MIXEROUTPUTENABLE'] =
taurus.Attribute('bo/RF/PYLYRTECHFACADELOOPS/MIXEROUTPUTENABLE').read().rvalue
        self.compare_BO['TUNINGPOSEN'] =
taurus.Attribute('bo/RF/PYLYRTECHFACADELOOPS/TUNINGPOSEN').read().rvalue
        self.compare_BO['FIELDFLATNESSPOS'] =
taurus.Attribute('bo/RF/PYLYRTECHFACADELOOPS/FIELDFLATNESSPOS').read().rvalue
        self.compare_BO['CLKPERPULSE'] =
taurus.Attribute('bo/RF/PYLYRTECHFACADELOOPS/CLKPERPULSE').read().rvalue.magnitude

            self.compare_BO['PhRefCav'] =
taurus.Attribute('bo/RF/PYLYRTECHFACADELOOPS/PhRefCav').read().rvalue
```

```
        self.compare_BO['PhaseRampEnd'] =
taurus.Attribute('bo/RF/PYLYRTECHFACADELOOPS/PhaseRampEnd').read().rvalue
        self.compare_BO['RampIncRate'] =
taurus.Attribute('bo/RF/PYLYRTECHFACADELOOPS/RampIncRate').read().rvalue
        self.compare_BO['t1ramping'] =
taurus.Attribute('bo/RF/PYLYRTECHFACADELOOPS/t1ramping').read().rvalue
        self.compare_BO['t2ramping'] =
taurus.Attribute('bo/RF/PYLYRTECHFACADELOOPS/t2ramping').read().rvalue
        self.compare_BO['t3ramping'] =
taurus.Attribute('bo/RF/PYLYRTECHFACADELOOPS/t3ramping').read().rvalue
        self.compare_BO['t4ramping'] =
taurus.Attribute('bo/RF/PYLYRTECHFACADELOOPS/t4ramping').read().rvalue
```

La tercera part es creen dos llistes, una que contindrà els atributs, els quals posteriorment, en cas de ser erronis, s'imprimeix per pantalla el resultat i l'acció a realitzar per l'operador, és a dir el valor correcte a canviar en aquell atribut, la llista és *bo_good_attr*. La segona llista anomenada *bo_call_rf* que en cas de tenir un resultat erroni, també imprimirà per pantalla els atributs, amb el missatge de s'ha de trucar al grup de *RF* quan sigui possible.

```
bo_good_attr = ['ManualMode','AmpRampInit','AmpRampEnd','MIXEROUTPUTENABLE',
                'TUNINGPOSEN','FIELDFLATNESSPOS','CLKPERPULSE','CoilCtrlBias' ]
bo_callRF = ['MIXEROUTPUTENABLE', 'TUNINGPOSEN', 'FIELDFLATNESSPOS', 'CLKPERPULSE']
```

La part final crea un diccionari anomenat *bo_results* on guarda els atributs i el resultat de la revisió d'aquests atributs. A continuació fa un *loop,* que compara els valors teòrics amb els valors llegits i retorna un missatge dient què s'ha de canviar, si s'ha de trucar o bé, si està OK, ja no mostrarà res per pantalla. Finalment, guarda tots els resultats al diccionari general *self.compare.*

```
bo_results = dict()
#print(self.compare_BO['CoilCtrlBias'], self.bo_attr_Dict['CoilCtrlBias'])
for attr in bo_good_attr:
   if round(self.compare_BO[attr]) != self.bo_attr_Dict[attr]:
      msg = ""
      if attr in bo_callRF:
           msg += ('The Attribute '+ attr+ ' is different from '+ str(self.bo_attr_Dict[attr])+ '
           and should be  '+str(self.bo_attr_Dict[attr])+'\n')
         msg += ('The settings of LLRF are not well set. call to RF\n ')
         bo_results[attr] = msg

      else:
         msg += ('The Attribute '+ attr+ ' is different from '+ str(self.bo_attr_Dict[attr]) + '
  and should be  '+str(self.bo_attr_Dict[attr])+'\n')
         if attr == 'ManualMode' or attr == 'CoilCtrlBias':
            print("test control bias")
            devicellrf_bo = "CIRCULATOR-01"
            if 'CoilCtrlBias':
                  msg += ('Write '+str(self.bo_attr_Dict[attr]-50)+' in jive in
                  bo/RF/'+devicellrf_bo+'/'+attr+'\n')
               bo_results[attr] = msg

            else:
```

```
                    msg += ('Write '+str(self.bo_attr_Dict[attr])+' in jive in
                bo/RF/'+devicellrf_bo+'/'+attr+'\n')
            else:
                devicellrf_bo = 'PYLYRTECHFACADELOOPS'
                msg += ('Write -'+str(self.bo_attr_Dict[attr])+' in jive in
bo/RF/'+devicellrf_bo+'/'+attr+'\n')
                bo_results[attr] = msg

        else:
            bo_results[attr] = "OK"

    if self.compare_BO['RAMPENABLE'] == False:
        if 99 < self.compare_BO['AmpRefCav'] >  101:
            devicellrf_bo = 'PYLYRTECHFACADELOOPS'
            msg += ('Write '+str(self.bo_attr_Dict['AmpRefCav'])+' in jive in
bo/RF/'+devicellrf_bo+'/AmpRefCav\n')
            bo_results['AmpRefCav']= msg
        else :
            bo_results['AmpRefCav'] = "OK"

    self.comparer["Booster"] = bo_results
except Exception as e :
    self.booster_on = False
    print ('The LLRF in Booster is hang')
    exc_type, exc_obj, exc_tb = sys.exc_info()
    print('Exception type {} in line {}'.format(exc_type, exc_tb.tb_lineno))
    print('Error! Code: {c}, Message: {m}'.format(c=type(e).__name__, m=str(e)))
```

4.2.1.15 check_Attr()

A la primera part es llegeix un fitxer *yaml* que conté tots els atributs amb els valors que s'haurien de tenir per tal de que després es puguin comparar amb els valors llegits directament.

```
def check_Attr(self):

    ## Load yaml file with attr values  ##
    try:
        yaml_file_path = Path('/data/test/Oriol/Python/acopRFcheck/attr.yaml')
        self.attr_Dict = load_yaml(yaml_file_path)
        #print (self.attr_Dict)
    except Exception as e:
        exc_type, exc_obj, exc_tb = sys.exc_info()
        print('Exception type {} in line {}'.format(exc_type, exc_tb.tb_lineno))
        print('Error! Code: {c}, Message: {m}'.format(c=type(e).__name__, m=str(e)))
```

La següent part es creen dues llistes, una que contindrà els atributs dels quals posteriorment, en cas de ser erronis, s'imprimiran per pantalla el resultat i l'acció a realitzar per l'operador, és a dir, el valor correcte a canviar en aquell atribut, la llista es *self.good_attr* . Hi ha una altra llista anomenada *self.call_RF* que en cas de tenir un resultat erroni, també imprimirà per pantalla els atributs, amb el missatge de s'ha de trucar al grup de *RF* quan sigui possible.

```
self.good_attr = [ 'AmpCell2Gain','AmpCell4Gain','MovePLG1','MovePLG2',
'PhaseDelayAmpMod','TimeDelay','ModulationType','FreqModCompAfterTrip',
'FieldFlatness','FreqModeEnable','MixerOutputEnable','TuningEnable','LookRef',
'PositivePhaseModulation','FieldFlatnessPos','DisRFOnItckFromEPS','ParkupEnable',
'ConditioningEnable','AutomaticParkingDisable','PhaseStepModulation',
'TuningPosEn', 'RFSwitchControlAuto','LoopInput','PISlaveLoopsEnable','CmdStart']

self.call_RF = ['AmplitudePhaseMod','FrequencyModPhAmp','AbsoluteTimeDecay',
'AmplitudeAmpMod', 'TimeDecayAmpMod','TimeDecayPhaseMod', 'AmpRefCav',
'AmpRefMin', 'FwMinAmpPh','FwMin', 'ConditioningDutyCycle',
Stepstodetunecavityaftertrip']
```

La part final crea un diccionari que contindrà anomenat *results* on guarda els atributs i el resultat de la revisió d'aquests atributs. A continuació fa un *loop* que compara els valors teòrics amb els valors llegits i retorna un missatge dient que s'ha de canviar, si s'ha de trucar o bé si està OK, que ja no mostrarà res per pantalla. Finalment guarda tots els resultats de cada planta de RF al diccionari general *self.compare.*

```
## check attr ##

for plant in self.rfPlants :

  if plant == "sr10a":
    self.devicellrf ="PICO-LLRF"
  else:
    self.devicellrf ="PYLYRTECHFACADELOOPS"
  try:
    results = dict()
    pl = getattr(self, plant)
    for attr in  self.good_attr:
      if getattr(pl, attr) != self.attr_Dict[attr]:
        print(getattr(pl, attr))
        msg = ""
        msg += ('The Attribute '+ attr+ ' is  '+ str(getattr(pl, attr))+ ' and should be
        '+str(self.attr_Dict[attr])+'\n')
          msg += ('Write '+str(self.attr_Dict[attr])+' in jive in SR'+pl.name+'/RF/'+
          self.devicellrf+'-X01/'+attr+'\n')
        msg += ('Send an email to RF group\n ')
        results[attr] = msg
        print(results[attr])
      else:
        results[attr] = "OK"

    if self.test_mode == False:
      self.comparer[plant].update(results)
    else:
      self.comparer[plant]=results
    #print(self.comparer[plant])

  except Exception as e:
    exc_type, exc_obj, exc_tb = sys.exc_info()
    print('Exception type {} in line {}'.format(exc_type, exc_tb.tb_lineno))
```

```
        print('Error! Code: {c}, Message: {m}'.format(c=type(e).__name__, m=str(e

    try:
        pl = getattr(self, plant)
        for attr in  self.call_RF:
            if getattr(pl, attr) == self.attr_Dict[attr]:

                msg = ""
                msg += ('The Attribute '+ attr+ ' is '+ str(getattr(pl, attr))+ ' and should be different
                                    from '+str(self.attr_Dict[attr])+'\n')
                 msg += ('Call RF group as soon as possible\n')
                 results[attr] = msg
            else:
                 results[attr] = "OK"

        if self.test_mode == False:
            self.comparer[plant].update(results)
        else:
            self.comparer[plant]=results
        #self.comparer[plant].update(results)

    except Exception as e:
        exc_type, exc_obj, exc_tb = sys.exc_info()
        print('Exception type {} in line {}'.format(exc_type, exc_tb.tb_lineno))
        print('Error! Code: {c}, Message: {m}'.format(c=type(e).__name__, m=str(e)))
```

4.2.1.16 check_FPGA()

Aquesta funció s'encarrega de mirar que les FPGA que controlen el LLRF tenen els valors adequats.

def check_FPGA(self):

Mira si el *booster* està operatiu i l'afegeix a la llista de plantes a supervisar.

```
    #rf_Plants = self.rfPlants
    if self.booster_on == True:
        self.rfPlants.append('BO')
```

Les configuracions apropiades es troben guardades en un fitxer anomenat *fpga.yaml*.

```
     ## Load yaml file with attr values  ##
    try:
        yaml_file_path = Path('/data/test/Oriol/Python/acopRFcheck/fpga.yaml')
        self.fpga = load_yaml(yaml_file_path)
```

Es crearan dues llistes on es guardaran els atributs a supervisar. D'aquí es mira la versió de codi que tenen instal·lada, el seu estat, si tenen la RAM configurada i l'estat dels *DACs.*

```
        list_fpgacodeversion = ['Pylyrtechfacadeloops-a01/FPGACodeVersion',
```

```
                'Pylyrtechfacadediags-a01/FPGACodeVersion',
                'Pylyrtechfacadeloops-b01/FPGACodeVersion',
                'Pylyrtechfacadediags-b01/FPGACodeVersion',
                'Pylyrtechfacadeloops/FPGACodeVersion',
                'Pylyrtechfacadediags/FPGACodeVersion']

    list_attr = ['RAMConfigured','State','Adac_trig','Adac_trig_nok']
```

Es crea un *loop* on es mira cada planta de RF, primer llegint l'atribut en qüestió i després es compara amb el valor teòric. En cas de ser diferent es mostrarà per pantalla un missatge on diu a l'operador que ha de fer.

```
    for plant in self.rfPlants :
      print(plant)
      results = dict()
      dict_fpga = self.fpga[plant.upper()]
      for name, value in dict_fpga.items():
        if plant == "BO":
          attr = taurus.Attribute(plant+'/RF/'+name).read().rvalue
        else:
          attr = taurus.Attribute(plant[:-1]+'/RF/'+name).read().rvalue
        if name in list_fpgacodeversion:
          if attr ==  value:
            results[plant+'/'+name] = "OK"
          else:
            results[plant+'/'+name] = "\nThe FPGACodeVersion in "+plant+'/'+name+"
            should be different, send an email to RF group"

        elif name.split("/")[-1] == 'RAMConfigured':
          if attr != value:
            results[plant+'/'+name] = "OK"
          else:
            fdl = name.split("/")[:-1]
            #print(fdl)
            results[plant+'/'+name] = "\nRestart DS FDL "+fdl[0]+" in astor"

        elif name.split("/")[-1] == 'FDLState':
            if attr == value:
              results[plant+'/'+name] = "OK"
            else:
              fdl = name.split("/")[:-1]
              results[plant+'/'+name] = "\nIn LLRF GUI, push FDLStart button in the FDL
             tab, in "+fdl[0]

        elif name.split("/")[-1] == 'State':
            if attr == value:
              results[plant+'/'+name] = "OK"
            else:
              fdl = name.split("/")[:-1]
              results[plant+'/'+name] = "\nIn LLRF GUI, push Start button in the FDL  tab, in
             "+fdl[0]
        elif name.split("/")[-1] == 'Adac_trig':
```

```
            if attr == value:
                results[plant+'/'+name] = "OK"
            else:
                fdl = name.split("/")[:-1]
                 results[plant+'/'+name] = "\nAdac_trig is not OK in "+name+". Launch
                 ctmps_resetall in terminal"

        elif name.split("/")[-1] == 'Adac_trig_nok':
            if attr == value:
                results[plant+'/'+name] = "OK"
            else:
                fdl = name.split("/")[:-1]
                 results[plant+'/'+name] = "\nIn LLRF GUI, Push reset AdacTrig NOK button
                 inside the FDL tab, in "+fdl[0]
```

Finalment s'afegeixen al diccionari *self.comparer* els resultats obtinguts, posant un *OK* en cas positiu o bé el text necessari.

```
            self.comparer[plant].update(results)

        #pprint.pprint(self.comparer)
    except AssertionError:
        print("The file doesn't exist!!")
    except Exception as e:
        exc_type, exc_obj, exc_tb = sys.exc_info()
        print('Exception type {} in line {}'.format(exc_type, exc_tb.tb_lineno))
        print('Error! Code: {c}, Message: {m}'.format(c=type(e).__name__, m=str(e)))
```

4.2.1.17 printing()

Aquesta funció el que fa es imprimir per pantalla tots els atributs que no estiguin OK. A part guardem aquesta informació en un atribut anomenat *self.mail* que posteriorment utilitzarem per enviar aquesta mateixa informació per e-mail.

```
def printing(self):
    print ("\n**********")
    print (' RF PLANTS CHECKING')
    print ("**********\n")

    for plant in self.rfPlants :
        msg = ""
```

Es fa una llista amb els atributs que en cap cas no es mostraran per pantalla i la seva informació només serà enviada per *e-mail*.

```
        attr_avoid_print = ['Diffrevers','DiffPforwr']
```

Fa un *loop* mirant tots els atributs que no estan OK, aquesta informació es guarda a la variable *msg.* Si l'atribut erroni en qüestió no s'ha d'imprimir per pantalla, simplement es guarda la informació a la variable *self.mail*

```
        ##Check the attributes are diferent from OK ##

        for k,v in self.comparer[plant].items():
           #print(k, v)
           if k not in attr_avoid_print:

              if v != "OK":
                 msg += (v+'\n')
           else:
              if v != "OK":
                 self.mail += ( "The plant "+plant.upper()+'\n')
                 self.mail += (v+'\n')


        ## avoid print if there are no problems ##

        m = not msg
        if m == False:

           print("********************")
           print("The plant \033[1m"+plant.upper()+'\033[0m')
           print("********************\n")
           print(msg)
           self.mail += ( "The plant "+plant.upper()+'\n')
           self.mail += msg

        ## All values of the plant are OK ##
        [self.comparer[plant].pop(key, None) for key in attr_avoid_print]
        if all(value == "OK" for value in self.comparer[plant].values()) == True:
           print ("The \033[1m"+plant.upper()+"\033[0m plant is \033[1mOK \033[0m")

     df = pd.DataFrame.from_dict(self.comparer, orient = 'index')
     df.style.format({'sr06a':"{:.2%}".format
```

Mira si la variable *self.mail* està plena, que voldrà dir que hi ha algun error i llavors cridarà la funció d'enviar *e-mails*.

```
     ## Send an email if there are error ##

     ma = not self.mail
     print(ma)
     if ma == False:
        self.sendmail(self.mail)
```

4.2.1.17 sendmail()

Aquesta funció envia un *mail* des d'un compte genèric de la *control room* definit al *sender* i es té una llista anomenada *receivers,* que inclou les direccions de correu destinatàries. A aquesta funció, al cridar-la, se li passa la variable *msg* que contindrà tots els missatge erronis abans esmentats.

```
def sendmail (self, msg):
    """This function sends an e-mail"""
    sender = 'root@crremote01.cells.es'
    receivers = ['oserres@cells.es']

    message = "From: acopRFcheck Script <root@crremote01.cells.es> \n"
    message += "To: Person <oserres@cells.es> \n"
    message += "Subject: RF errors \n\n"
    message += msg
```

S'assigna el servidor STMP necessari per enviar el correu i es posen excepcions perquè en cas que es produeixi un error mostri per pantalla que el correu no s'ha pogut enviar. En cas que s'enviï també ho mostra per pantalla.

```
 try:
        smtpObj = smtplib.SMTP('smtp.cells.es')
        smtpObj.sendmail(sender, receivers, message)

        print ("\nSuccessfully sent email\n")

    except smtplib.SMTPException:

        print ("Error: unable to send email")

    except smtplib.SMTPSenderRefused:

      print (" Sender Unable ")
```

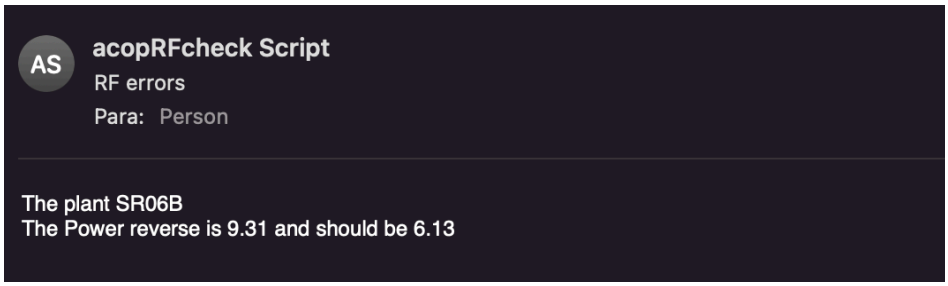
4.3 Resultats

Es van realitzar diverses proves per demostrar el funcionament de l'*script:*

- Aquí es mostra la primera part l'*Status RF* on es veu el voltatge a cada cavitat i la fase síncrona corresponent. A la part del mig es veu l'estat dels IDs, el voltatge total i la fase síncrona teòrica. A la part inferior es veu la part *RF Plants Checking* on es veu cada planta i que el seu estat és OK, i per tant tots els atributs de les plantes estan correctament configurats. Finalment, indica que s'ha enviat un *e-mail*, per tant vol dir, que algun paràmetre que s'ha de notificar al grup de RF però no a l'operador està mal ajustat.

```
********************
       STATUS RF
********************

       Cavity Voltage  Sync.Phase
sr06a       499.877989  158.909137
sr06b       500.151677  158.567942
sr10a       500.100823  159.999676
sr10b       499.727833  159.248521
sr14a       499.844029  159.370267
sr14b       500.282203  158.871984

********************
The IDs are closed, the Total Voltage is 2999.98 MV and the theoretical synchronous phase is 158.5
********************


********************
 RF PLANTS CHECKING
********************

The SR06A plant is OK
The SR10A plant is OK
The SR10B plant is OK
The SR14A plant is OK
The SR14B plant is OK
The BOOSTER plant is OK

Successfully sent email
```

- Aquest és l'*e-mail* que envia al grup de RF indicant que la *Power Reverse* no té el valor adequat.

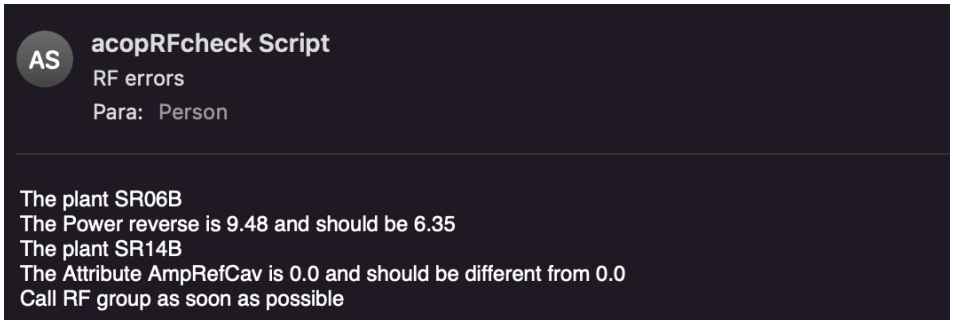


Posteriorment es van modificar diferents paràmetres per veure que l'*script* que era capaç de detectar-los.

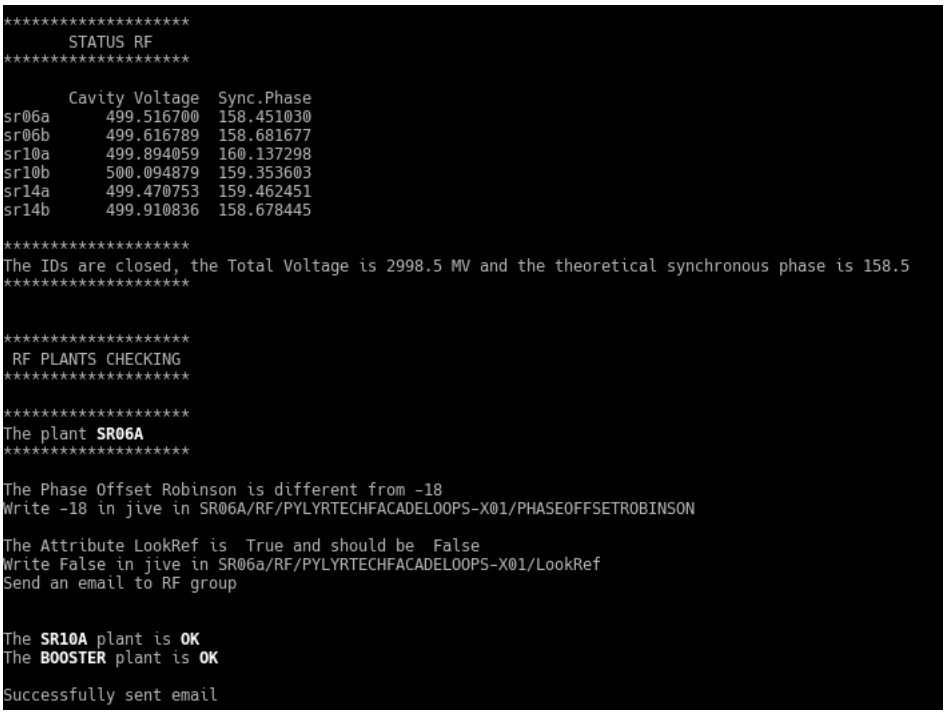
- Aquí es va modificar l'atribut *AmpRefCav* i es veu l'*script* el detecta i llença el corresponent missatge, en aquest cas s'ha de trucar al grup de RF.

```
********************
       STATUS RF
********************

       Sync.Phase  Cavity Voltage
sr06a  158.825444      499.809911
sr06b  158.353001      500.130483
sr10a  159.470986      500.557748
sr10b  158.748160      499.657969
sr14a  158.874847      499.888868
sr14b  159.839358      445.675268

********************
The IDs are closed, the Total Voltage is 2945.72 MV and the theoretical synchronous phase is 158.1
********************


********************
 RF PLANTS CHECKING
********************

The SR06A plant is OK
The SR10A plant is OK
The SR10B plant is OK
The SR14A plant is OK
********************
The plant SR14B
********************

The Attribute AmpRefCav is 0.0 and should be different from 0.0
Call RF group as soon as possible


The BOOSTER plant is OK

Successfully sent email
```

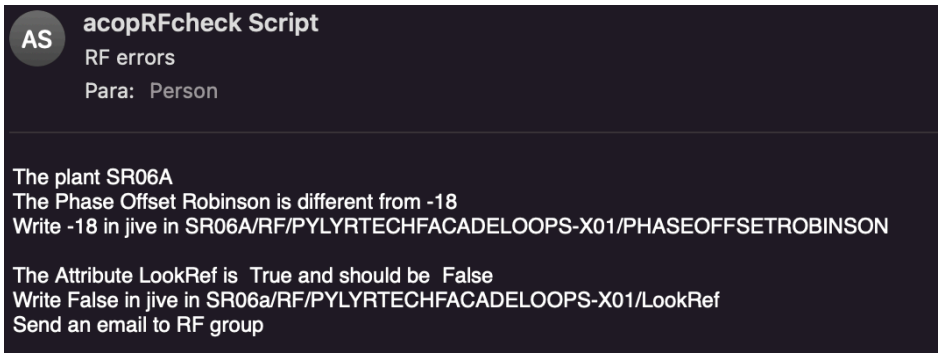
- En aquest tenim l'*e-mail* on es té el mateix error que abans que només ha de rebre el grup de RF i el missatge de l'error que també es veu a l'*script*.



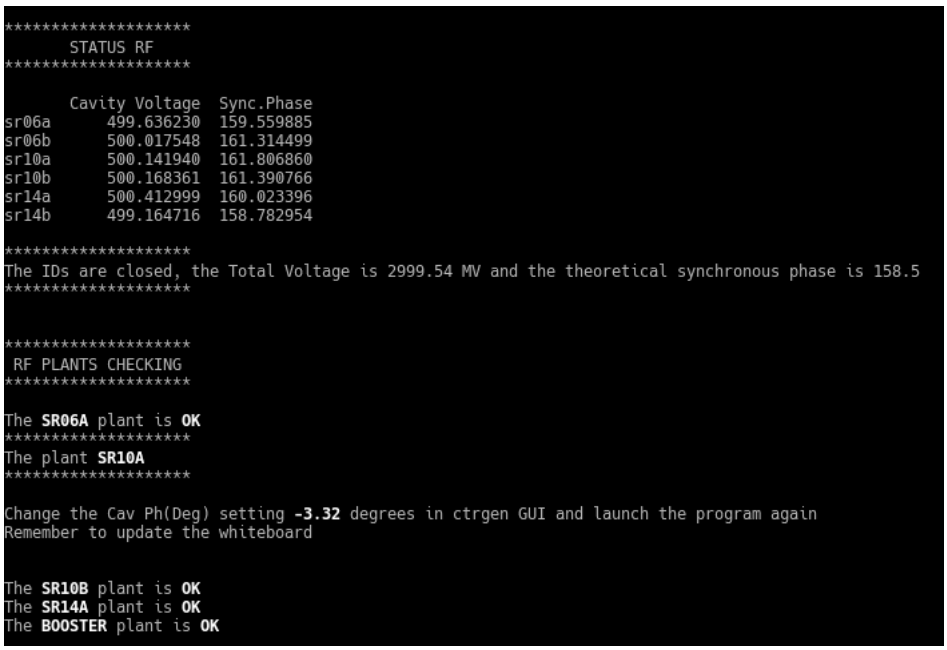
- S'han modificat l'atribut L*ookRef* i el paràmetre *Phase Offset Robinson detuning* amb els seus corresponents missatges, en el cas del *Robinson* ens indica que s'ha de fer per ajustar el paràmetre adequadament, en l'altre cas s'ha d'enviar un *e-mail* al grup de RF.



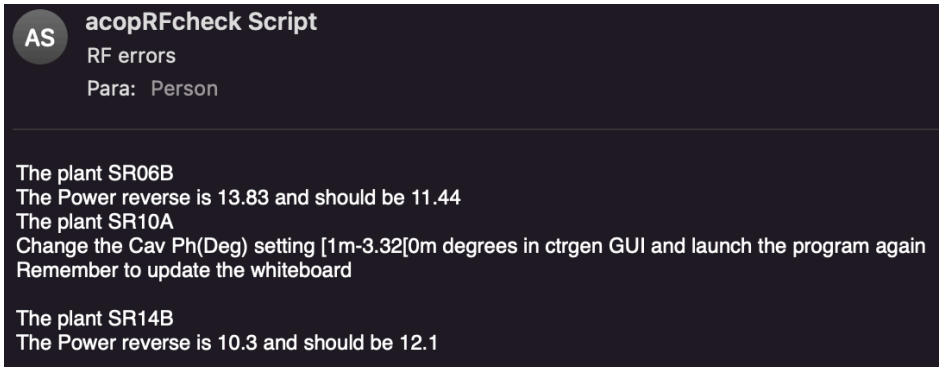
- *E-mail* corresponent



- Aquí es veu que la diferència de fase síncrona a la cavitat SR10A és superior a 3 graus per tant es produeix una alerta que diu el que s'ha de disminuir *Cavity Phase*.



- *E-mail* corresponent



- Per fer aquest test s'ha rebaixat el marge de comparació del *plunger* per tal de veure que *l'script* actua correctament.

```
********************
 RF PLANTS CHECKING
********************

********************
The plant SR06A
********************

The Plunger is -0.27 and should be -0.2

********************
The plant SR06B
********************

The Plunger is -0.33 and should be -0.26

********************
The plant SR10A
********************

The Plunger is -0.21 and should be -0.14

********************
The plant SR10B
********************

The Plunger is -0.23 and should be -0.17

********************
The plant SR14A
********************

The Plunger is -0.4 and should be -0.33

********************
The plant SR14B
********************

The Plunger is -0.28 and should be -0.21
```

- Finalment l'última prova realitzada és amb els atributs de les FPGA del LLRF.

```
********************
The plant SR14B
********************


In LLRF GUI, Push reset AdacTrig NOK button inside the FDL tab, in Pylyrtechfacadediagsfdl-b01

In LLRF GUI, push Start button in the FDL  tab, in Pylyrtechfacadeloopsfdl-b01

In LLRF GUI, push Start button in the FDL  tab, in Pylyrtechfacadediagsfdl-b01

In LLRF GUI, Push reset AdacTrig NOK button inside the FDL tab, in Pylyrtechfacadeloopsfdl-b01

Adac_trig is not OK in Pylyrtechfacadediagsfdl-b01/Adac_trig. Launch ctmps_resetall in terminal

Restart DS FDL Pylyrtechfacadeloopsfdl-b01 in astor

Restart DS FDL Pylyrtechfacadediagsfdl-b01 in astor

Adac_trig is not OK in Pylyrtechfacadeloopsfdl-b01/Adac_trig. Launch ctmps_resetall in terminal

********************
The plant BO
********************


Restart DS FDL Pylyrtechfacadediagsfdl in astor

Adac_trig is not OK in Pylyrtechfacadediagsfdl/Adac_trig. Launch ctmps_resetall in terminal

In LLRF GUI, Push reset AdacTrig NOK button inside the FDL tab, in Pylyrtechfacadediagsfdl

Restart DS FDL Pylyrtechfacadeloopsfdl in astor

In LLRF GUI, push Start button in the FDL  tab, in Pylyrtechfacadeloopsfdl

Adac_trig is not OK in Pylyrtechfacadeloopsfdl/Adac_trig. Launch ctmps_resetall in terminal

In LLRF GUI, Push reset AdacTrig NOK button inside the FDL tab, in Pylyrtechfacadeloopsfdl

In LLRF GUI, push Start button in the FDL  tab, in Pylyrtechfacadediagsfdl
```

4.4 Conclusions

En aquest apartat s'ha explicat tant el funcionament en que es basa el sistema de control de l'accelerador com totes les funcions que s'utilitzen en que es divideix l'*script* mostrant-ne finalment el resultats i veient-ne el seu correcte funcionament.

5. Conclusions

La principal conclusió és que s'ha aconseguit crear una eina per supervisar el sistema de radiofreqüència, el qual és un element crític ja que és el que s'encarrega de subministrar l'energia que van perdent els electrons i per tant és essencial per al bon funcionament del Sincrotró. Aquest sistema treballa a 500 MHz, que és la freqüència de disseny que ens permet tenir uns amplificadors de RF com són els *IOTs,* que a aquesta freqüència són relativament senzills de fabricar, ja que a freqüències superiors degut a la distància que ha de tenir entre el càtode i el *grid* representa una dificultat molt més gran de fabricació i, per tant, de generació de freqüències superiors. Per a poder generar aquests 500 MHz hem de tenir els paràmetres que controlen els *IOTs*, així com el CACO, que acobla i combina el senyal dels 2 *IOTs*, s'ha de controlar el circulador, i finalment la cavitat, que és on s'entrega aquest voltatge i que té un *plunger* que permet la perfecta sintonització de la cavitat. Tenint en compte aquests elements i molts d'altres que intervenen en el sistema i que a la seva vegada tenen múltiples paràmetres que s'han de controlar, va sorgir la necessitat de tenir un *script* que ens certifiqués que tots els elements estan ben configurats.

Per desenvolupar aquest *script*, primer s'ha fet un estudi teòric de tots els fenòmens que es produeixen i que s'han de tenir en compte en la generació d'aquest voltatge RF, i la seva interacció amb els feix d'electrons, adquirint notables coneixements teòrics del seu funcionament. En segon lloc, s'ha traslladat aquesta teoria a la pràctica traduint-ho en codi Python, que a la seva vegada s'ha après a utilitzar més àmpliament.

Així doncs, es pot dir que s'han assolit tots els objectius que es van plantejar a l'hora d'iniciar el projecte, inclús s'han afegit noves funcions no que estaven previstes en un primer moment, com podria ser el cas de la funció per revisar les FPGA.

Pel que fa a la planificació es pot dir que s'ha complert ja que era un projecte que tenia un termini de 6 mesos per realitzar a la feina i ara ja podem dir que està acabat, sempre obert a possibles ampliacions en un futur. No ha calgut introduir canvis en la metodologia ja que després d'adquirir les nocions teòriques tècniques prèvies, el fet que l'*script* estigui desenvolupat de forma modular ha facilitat molt la programació de cada funció de forma independent per després poder-les cridar totes, fet que també ha permès les últimes actualitzacions que s'han realitzat i també les futures.

Finalment, es pot dir que amb aquest *script* permet tenir controlat en qualsevol moment els elements de configuració del sistema i a part obtenir un *feedback* de la potència que s'està lliurant al feix d'electrons amb tots els seus càlculs corresponents. Així doncs, amb la posada en funcionament d'aquesta supervisió automàtica, s'ha resolt el problema que suposava de temps i de personal la supervisió manual.

Les línies de treball futures quedaran subjectes a les possibilitats potencials que es vegin des de el grup de RF, per a la seva ampliació.

6. Glossari

- RF: *Radio Frequency*
- AcopRFcheck: *Accelerators Operation Radio Frequency Check.*
- Linac: *Linear Accelerator*
- SR: *Storage Ring*
- FCT: *Fast Current Transformer.*
- DCCT: *Direct Current Tranformer.*
- BPM: *Beam Position Monitor.*
- ID: *Insertion Device.*
- LLRF: *Low Level Radio Frequency.*
- CACO: *Cavity Combiner.*
- HVPS: *High Voltage Power Supplies.*
- IOT: *Inductive Output Tube*.
- WATRAX: Waveguide Transition to coaxial.
- SSPA: *Solid State Power Amplifier.*
- FPGA: *Field Programmable Gate Array*.

7. Bibliografia

G.A.Krafft & L.Mermimga. USPAS Course on Reciculating Linear Accelerators. (2001). *Jefferson Lab*. Obtingut a https://www.jlab.org/ERL/USPAS/lect8.ps
Beam Loading and Robinson's Stability. (2000). *Fermilab.*
Obtingut a https://lss.fnal.gov/archive/2000/conf/Conf-00-142-T-7.pdf
Thomas Weiss. Interaction between RF-System, RF-Cavity and Beam.(2005). *Delta, Universitat Dormund.*
Obtingut a http://www.delta.tu-dortmund.de/cms/de/Studium/Homepage_Weis/Vortraege/Vortrag_BeamLoading_Riezlern2005.pdf
Curs intern d'operadors sincrotró ALBA. Magnets.
Curs intern d'operadors sincrotró ALBA. Vaccuum.
Curs intern d'operadors sincrotró ALBA. Diagnostics.
Curs intern d'operadors sincrotró ALBA. Insertion Devices.
Curs intern d'operadors sincrotró ALBA. Radio Frequency.

8. Annexos

A continuació es mostraran els fitxers de configuració que es llegeixen durant l'execució del *script.*

ATTR.yaml

```
# Call RF Attribute values, these values have to be different form 0.0#
TimeDecayAmpMod: 0.0
TimeDecayPhaseMod: 0.0
FrequencyModPhAmp: 0.0
AbsoluteTimeDecay: 0.0
AmplitudePhaseMod: 0.0
AmplitudeAmpMod: 0.0
AmpRefCav: 0.0
AmpRefMin: 0.0
FwMinAmpPh: 0.0
FwMin: 0.0
ConditioningDutyCycle: 0.0
Stepstodetunecavityaftertrip: 0.0

# The attributes should be this values #
AmpCell2Gain: 0.0
AmpCell4Gain: 0.0
MovePLG1: 1.0
MovePLG2: 0.0
PhaseDelayAmpMod: 0.0
TimeDelay: 0.0
ModulationType: 2.0
FreqModCompAfterTrip: True
FieldFlatness: False
FreqModeEnable: False
MixerOutputEnable: True
TuningEnable: True
LookRef: False
PositivePhaseModulation: False
FieldFlatnessPos: False
DisRFOnItckFromEPS: False
ParkupEnable: True
ConditioningEnable: False
AutomaticParkingDisable: False
PhaseStepModulation: False
TuningPosEn: False
RFSwitchControlAuto: True
LoopInput: 0.0
PISlaveLoopsEnable: False
CmdStart: 5
PHASEOFFSETROBINSON: -18
#FPGALOCKED: DCMS_LOCKED
```

BO.yaml

```
# Booster atributes #

ManualMode: True
AmpRampInit: 50
AmpRampEnd: 415
RAMPENABLE: False
AmpRefCav:  100
MIXEROUTPUTENABLE: True
TUNINGPOSEN: False
FIELDFLATNESSPOS: True
CLKPERPULSE: 4
#coilcontrolbias_RAMPING: -12000
CoilCtrlBias: -11950

# The following values have to be different from 0.0
PhRefCav: 0.0
PhaseRampEnd: 0.0
RampIncRate: 0.0
t1ramping: 0.0
t2ramping: 0.0
t3ramping: 0.0
t4ramping: 0.0
```

FPGA.yaml

```
SR06A:
   Pylyrtechfacadeloops-a01/FPGACodeVersion: 2020022800
   Pylyrtechfacadediags-a01/FPGACodeVersion: 2017053120
   Pylyrtechfacadeloopsfdl-a01/RAMConfigured: True
   Pylyrtechfacadediagsfdl-a01/RAMConfigured: True
   Pylyrtechfacadeloopsfdl-a01/State: 10
   Pylyrtechfacadediagsfdl-a01/State: 10
   Pylyrtechfacadeloopsfdl-a01/Adac_trig : False
   Pylyrtechfacadediagsfdl-a01/Adac_trig : False
   Pylyrtechfacadeloopsfdl-a01/Adac_trig_nok : False
   Pylyrtechfacadediagsfdl-a01/Adac_trig_nok: False


SR06B:
   Pylyrtechfacadeloops-b01/FPGACodeVersion: 2020022800
   Pylyrtechfacadediags-b01/FPGACodeVersion: 2017053120
   Pylyrtechfacadeloopsfdl-b01/RAMConfigured: True
   Pylyrtechfacadediagsfdl-b01/RAMConfigured: True
   Pylyrtechfacadeloopsfdl-b01/State: 10
   Pylyrtechfacadediagsfdl-b01/State: 10
   Pylyrtechfacadeloopsfdl-b01/Adac_trig : False
   Pylyrtechfacadediagsfdl-b01/Adac_trig : False
```

Pylyrtechfacadeloopsfdl-b01/Adac_trig_nok : False
Pylyrtechfacadediagsfdl-b01/Adac_trig_nok: False

SR10A:
Pico-llrf-a01/FPGACodeVersion: 2020100900
Pico-llrf-a01/FDLState: RUNNING

SR10B:
Pylyrtechfacadeloops-b01/FPGACodeVersion: 2020022800
Pylyrtechfacadediags-b01/FPGACodeVersion: 2017053120
Pylyrtechfacadeloopsfdl-b01/RAMConfigured: True
Pylyrtechfacadediagsfdl-b01/RAMConfigured: True
Pylyrtechfacadeloopsfdl-b01/State: 10
Pylyrtechfacadediagsfdl-b01/State: 10
Pylyrtechfacadeloopsfdl-b01/Adac_trig : False
Pylyrtechfacadediagsfdl-b01/Adac_trig : False
Pylyrtechfacadeloopsfdl-b01/Adac_trig_nok : False
Pylyrtechfacadediagsfdl-b01/Adac_trig_nok: False
SR14A:
Pylyrtechfacadeloops-a01/FPGACodeVersion: 2020022800
Pylyrtechfacadediags-a01/FPGACodeVersion: 2017053120
Pylyrtechfacadeloopsfdl-a01/RAMConfigured: True
Pylyrtechfacadediagsfdl-a01/RAMConfigured: True
Pylyrtechfacadeloopsfdl-a01/State: 10
Pylyrtechfacadediagsfdl-a01/State: 10
Pylyrtechfacadeloopsfdl-a01/Adac_trig : False
Pylyrtechfacadediagsfdl-a01/Adac_trig : False
Pylyrtechfacadeloopsfdl-a01/Adac_trig_nok : False
Pylyrtechfacadediagsfdl-a01/Adac_trig_nok: False
SR14B:
Pylyrtechfacadeloops-b01/FPGACodeVersion: 2020022800
Pylyrtechfacadediags-b01/FPGACodeVersion: 2017053120
Pylyrtechfacadeloopsfdl-b01/RAMConfigured: True
Pylyrtechfacadediagsfdl-b01/RAMConfigured: True
Pylyrtechfacadeloopsfdl-b01/State: 10
Pylyrtechfacadediagsfdl-b01/State: 10
Pylyrtechfacadeloopsfdl-b01/Adac_trig : False
Pylyrtechfacadediagsfdl-b01/Adac_trig : False
Pylyrtechfacadeloopsfdl-b01/Adac_trig_nok : False
Pylyrtechfacadediagsfdl-b01/Adac_trig_nok: False

BO:
Pylyrtechfacadeloops/FPGACodeVersion: 2020021801
Pylyrtechfacadediags/FPGACodeVersion: 2018091920
Pylyrtechfacadeloopsfdl/RAMConfigured: True
Pylyrtechfacadediagsfdl/RAMConfigured: True
Pylyrtechfacadeloopsfdl/State: 10
Pylyrtechfacadediagsfdl/State: 10
Pylyrtechfacadeloopsfdl/Adac_trig : False
Pylyrtechfacadediagsfdl/Adac_trig : False
Pylyrtechfacadeloopsfdl/Adac_trig_nok : False
Pylyrtechfacadediagsfdl/Adac_trig_nok: False