Xバンド加速管における放電現象の研究

末原 大幹 (Taikan SUEHARA)[1]

平成17年2月4日

[1]Dept.of Physics, Graduate School of Science, the Univ. of Tokyo

目 次

第 1 章　研究概要 **6**
1.1 最高エネルギー実験における加速器 6
1.2 高電場勾配のための加速器の要件 8
1.3 X-band 加速技術 8
1.4 加速管放電現象の測定と放電機構の解析 9
1.5 XTF(旧 GLCTA) と GLC 計画 10
1.6 本論の構成 10

第 2 章　XTF **13**
2.1 XTF の目的と機能 13
2.1.1 RF 源の開発・試験 13
2.1.2 放電の少ない加速管の製作 13
2.1.3 高電場加速の実証 14
2.2 XTF の設備 14
2.2.1 設備の概要 14
2.2.2 モジュレータ・直流安定化電源 18
2.2.3 RF 源と RF Power Control 18
2.2.4 クライストロン 22
2.2.5 導波管 25
2.2.6 加速管 26
2.2.7 真空系 28
2.2.8 各種センサー 29
2.3 XTF のコントロール系 29
2.3.1 計算機設備 29
2.3.2 コントロール系の概要 33
2.3.3 トリガー系 36
2.3.4 インターロック系 37
2.3.5 XTF コントロールソフトウェア 41
2.3.6 ステータスデータの記録 41

2.3.7 RF Processing 44
2.3.8 コントロールインターフェース 44

第 3 章 放電検出・記録システム (XTF-BDMS) 47
3.1 概要 47
3.1.1 放電現象の観測 48
3.1.2 検出システムの要件 50
3.1.3 XTF-BDMS 51
3.2 放電検出器 51
3.2.1 RF 検出器 51
3.2.2 γ 線検出器 58
3.2.3 音響検出器 71
3.3 データ収集 (DAQ:Data Aquisition) 回路 72
3.3.1 RF 波形記録 76
3.3.2 γ 線時間・電荷記録 82
3.3.3 音響波形記録 84
3.3.4 オシロスコープ 86
3.3.5 トリガー管理 88
3.4 データ収集系 90
3.4.1 概要・特長 90
3.4.2 BDMS-CAMAC 92
3.4.3 CC/NET による CAMAC モジュールとの通信 . . . 95
3.4.4 Trigger 応答 99
3.4.5 Event 番号管理 100
3.4.6 データ収集・放電検出 100
3.4.7 データ保存 103
3.4.8 通信機能 107
3.4.9 BDMS-VME 111
3.4.10 Acoustic 応答 112
3.4.11 オシロスコープ 116
3.4.12 画面表示 116
3.5 放電検出 116
3.5.1 放電検出の概要 117
3.5.2 RF 検出器による放電検出 117
3.5.3 γ 線検出器による放電検出 123
3.5.4 放電検出ルーチン 124

3.5.5 ユーザーインターフェース 125
3.6 オフライン解析 125
3.6.1 フレームワーク 127
3.6.2 Event Display 127
3.6.3 root による解析 128
3.6.4 データのエクスポート 129
3.7 まとめ 132

第 4 章 データ収集の経過 134
4.1 運転の手順・方法 134
4.1.1 立ち上げ、立ち下げ 134
4.1.2 計算機類の起動 136
4.1.3 RF パラメータの設定 137
4.2 運転履歴と設定の変遷 137
4.3 Processing History と RUN # 140

第 5 章 放電データの解析 141
5.1 解析の概要 141
5.2 放電の頻度 142
5.2.1 放電頻度と加速管の特性 143
5.2.2 放電の定義 147
5.2.3 加速管放電イベントの選択 147
5.2.4 各 Run での入力 RF パワーの評価 149
5.2.5 Run 内での放電頻度の変化 150
5.2.6 各 Run の比較 153
5.2.7 まとめ 156
5.3 放電の位置 157
5.3.1 位置解析の意義 157
5.3.2 RF データからの位置検出 157
5.3.3 γ 線データからの位置検出 161
5.3.4 音響データからの位置検出 161
5.3.5 データ間の相関 166
5.3.6 位置分布と電界強度 166
5.4 まとめ 168

第 6 章　まとめと今後の展望 **169**
6.1 XTF 169
6.2 放電検出システム 170
6.3 放電解析 171

第 7 章　謝辞 **172**

付 録 A 放電検出システム (CAMAC 部) ソースコード **176**
A.1 bdmonitor.h 176
A.2 bdmonitor.cpp 177
A.3 Camac-slot.h 177
A.4 CMasterCtrl.h 178
A.5 CMasterCtrl.cpp 178
A.6 CCamacCtrl.h 183
A.7 CCamacCtrl.cpp 184
A.8 pcc2.h 188
A.9 CSingleEvent.h 188
A.10 CSingleEvent.cpp 189
A.11 CCsyCtrl.h 189
A.12 CCsyCtrl.cpp 189
A.13 CEventAnalyzer.h 192
A.14 CSocketCtrl.h 192
A.15 CSocketCtrl.cpp 192
A.16 atf_socket.h - modified from original 195
A.17 CSocketReceiver.h 196
A.18 CSocketReceiver.cpp 197
A.19 streamers.h 197
A.20 LinkDef.h 200
A.21 CAnalyzeFADC.h 201
A.22 CAnalyzeFADC.cpp 201
A.23 CAnalyzeXRay.h 211
A.24 CAnalyzeXRay.cpp 211
A.25 Makefile 214

付 録 B 放電検出システム (VME 部) ソースコード **216**
B.1 bdmonitor.h 216

B.2 bdmonitor.cpp 216
B.3 CMasterCtrl.h 217
B.4 CMasterCtrl.cpp 218
B.5 CVmeCtrl.h 220
B.6 CVmeCtrl.cpp 220
B.7 common_include.h 221
B.8 myvme.h 222
B.9 myvme.cpp 223
B.10 CEventAnalyzer.h 225
B.11 CSocketCtrl.h 225
B.12 CSocketCtrl.cpp 225
B.13 atf_socket.h 225
B.14 streamers.h 226
B.15 LinkDef.h 226
B.16 CAnalyzeAcoustic.h 227
B.17 CAnalyzeAcoustic.cpp 227
B.18 Makefile 232

付 録C BDMS-CAMAC 通信メッセージ一覧 233

付 録D BDMS-VME 通信メッセージ一覧 237

第1章　研究概要

1.1　最高エネルギー実験における加速器

エネルギーフロンティアの加速器を用いた高エネルギー実験は、標準理論の検証および標準理論を越える新たな理論を発見するもっとも直接的な方法である。現在、標準理論で予言され、唯一発見されていない粒子である Higgs 粒子、また標準理論を超える理論として有力視されている超対称性理論がその存在を予言する超対称性粒子等の直接探索のため、TeV 領域の衝突エネルギーを達成できる次世代加速器は、素粒子物理学の発展のために不可欠なものとなっている。

現在、この TeV 領域の次世代加速器として、陽子陽子衝突型円形加速器の LHC(Large Hadron Collider)[1]、電子陽電子衝突型線形加速器 (LC:Linear Collider)[2] の二つの計画が進行中である。LHC は 2007 年の運転開始を目指して現在 CERN(European Organization for Nuclear Research:ヨーロッパ素粒子物理学研究所)[3] にて建設中である。重心系エネルギーは 14TeV[4] の予定。LC はまだ計画段階だが、世界各地で精力的に研究開発が進められている。重心系エネルギーは 1TeV[5] までが想定されている。

LHC と LC は相補的な役割が期待されている。LHC は衝突エネルギーが高いため重い超対称性粒子の発見等に適している。一方、LC は重心エネルギーは LHC に比べ低いが、複合粒子の陽子に比べ単一粒子の電子・陽電子の衝突であるため、バックグラウンドが少なく結合定数の精密測定等に適している。

電子・陽電子は陽子と比べて質量が 10^{-3} 程度であり、シンクロトロン放射 (損失エネルギーは γ^4 に比例) が高エネルギーで極端に大きくなるため、円形加速器では現実的には一定のエネルギー以上に加速することができない。[6] これまでの実験でもっとも大きな円形加速器の LEP では直径 4243m のリングで 1 周あたり 3GeV 弱の加速電圧を維持し、重心系エネルギー約 200GeV を達成したが、TeV 領域まで円形加速器でエネル

	LHC	LC	
		500 GeV	1 TeV
Light Higgs boson (120-140 GeV)			
Detection	○	○	–
Width (Γ_H)	△	○	–
J^P	△	○	–
Coupling ($g_{VVH}, Y_{f\bar{f}H}$)	○	◎	–
Top Yukawa C.C.($Y_{t\bar{t}H}$)	△	×	○
Self-coupling(λ_{HHH})	×	△	○
500 GeV SM Higgs boson			
Detection	○	×	○
Top quark			
Δm_t	$\sim$ 1 GeV	$\lesssim$ 100 MeV	–
Width (Γ_t)	×	a few %	–
Supersymmetry			
Squark mass reach	$\lesssim$ 2.5 TeV	$\lesssim \sqrt{s}/2$	
Slepton/Chargino/Neutralino	Cascade decay	Pair production	
Mass measurement	○	◎	
Proving SUSY (Spin, Coupling)	×	◎	
Testing SUSY breaking model	○	○	
MSSM Heavy Higgs	high $\tan\beta$	$\lesssim \sqrt{s}/2$	
Indirect constraint			
on SUSY parameters	△	○	○
Large Extra Dimension			
KK graviton	○	△	○
Black hole production	○	×	△
Z', KK graviton of RS model,	Direct production	Contact interaction	
KK mode of W and Z, etc.			
Mass reach	○	○	◎

表 1.1: Research potentials expected for the LHC and e^+e^- linear colliders. LHC with an integrated luminosity of $\sim$ 100 fb^{-1} and a 500 GeV e^+e^- linear collider with $\sim$ 500 fb^{-1} are compared. The merits of the energy extension of the linear collider to 1 TeV are shown in the column of 1 TeV LC. "Double circle", excellent; "circle", good; "triangle", fair; "cross", not useful. "–" means that this category is already fully covered at the 500 GeV e^+e^- linear collider.

ギーを上げるのは現実でない (単純な比較では 625 倍の加速勾配または周長が必要になってしまう)。

このため、TeV 領域の電子・陽電子衝突型加速器は必然的に線形となる。線形加速器は加速管内を粒子が 1 度しか通過しないため、現実的な長さの加速器を作るためには加速勾配を大きくする必要がある。具体的には、LC で想定している 1TeV を全長 30km の主線形加速器部で行うには 33MV/m の加速勾配が必要である。実際には加速管の接合部付近では加速できないため、加速管内で必要な加速勾配はさらに上昇する。

現在実用化されている線形加速器の最大加速勾配が 26MV/m(ATF Linac: 平均加速勾配)[7] であることを考えると、LC では極めて高い加速勾配が必要であることがわかる。

1.2 高電場勾配のための加速器の要件

このような高い加速勾配では、加速管の放電が極めて深刻な問題となってくる。一般的な加速管では、一定以上の電場勾配にすると放電頻度が急激に上昇し、定常的な加速に利用できなくなる。この関係性については第 5 章で詳しく述べる。

この加速管の放電限界は、加速周波数を上げることによって増やすことができることが経験的に知られている。[8, 放電頻度と周波数の関係性の仮説の 1 例として] これは加速周波数を上げると加速管内部の浮遊電子の加速電場による振動の振幅が小さくなるためと考えられているが詳しくはわかっていない。

そこで LC の加速管としては、通常の線形加速器によく用いられる 2.956 GHz (S-band) ではなく、さらに高い周波数で加速を行うアイデアが提唱された。LC 用に研究が進められてきたのは C-band (5.912 GHz) および X-band (11.424 GHz) の加速技術である。

1.3 X-band 加速技術

このうち X-band の加速技術は 20 年近くにわたり SLAC(Stanford Linear Accelerator Center)[9] と KEK(高エネルギー加速器研究機構)[10] の共同で開発が進められてきた。現在、X-band の加速管はビームなしで 65MV/m(ビーム負荷ありで約 50MV/m) の加速勾配が実現している。た

だし、実用的な高い放電安定性を持つ加速管の開発には加速管の設計、加工の工程などまだ調整が必要である。X-band は周波数が従来型の S-band の 4 倍であり、波長が 1/4(26.2mm) となるため、加速管、導波管など共鳴空洞を用いるコンポーネントはすべてサイズが小さくなり、従来より精度の良い加工が必要である。

また、モジュレータ、クライストロン、パルス圧縮器 (SLED-II) 等の加速管に電力を供給するのに必要なコンポーネントの開発も併せて行われており、X-band の高い加速勾配での加速器の運転が現実的なものとなりつつある。

XTF(X-band Test Facility)[11] は、X-band 加速技術の実証と試験をかねて KEK のアセンブリホールに設置されているテストステーションである。上記モジュレータ、クライストロンを用いて加速管に実際に高電力を供給し、加速管および電力コンポーネント、導波管等の性能評価、動作試験を行うことを目的としている。

1.4 加速管放電現象の測定と放電機構の解析

X-band 加速技術の最大の課題は、上述のように高い加速勾配において放電をいかに抑え安定に運転できる加速管を製作するか、にある。加速管放電を抑えるためには、加速管に実際に高電力を供給し、放電の特性を調べ、その加速管の問題点を見極める必要がある。

XTF の目的の一つは加速管の放電データの収集・解析し放電の少ない加速管を製作するための基礎データを得ることにある。そのため XTF の加速管周辺には各種放電検出器が設置され、加速管を常時監視している。

加速管放電の研究は、X-band 加速管に限った問題ではなく、一般に加速管の製作にとって本質的な問題であるため、この放電検知・記録システムおよび放電データは、多くの加速管製作・設計・運用上重要なデータとなると考えられる。

本研究はこの XTF 加速管放電検知・記録システム (以下 XTF-BDMS または BDMS と略す) の設計・構築・運用とそこで得られたデータの解析を主なテーマとしている。

1.5 XTF(旧GLCTA)とGLC計画

LCの計画はDESY(Deutsches Elektronen-Synchrotron:ドイツ電子シンクロトロン研究所)[12]を中心に開発された、L-band(1.3GHz)超伝導空洞を用いるTESLA(TeV Superconducting Linear Accelerator)[13]計画と、KEK,SLACを中心に開発されたX-band,C-band常伝導空洞のGLC(Global Linear Collider)/NLC(Next Linear Collider)計画の二つに大きく分かれて進められてきた。このうちGLCはかつてJLCと呼ばれ、約20年前から日本国内で構想され、基礎研究が進められてきたものである。

この二つの計画は、得られる物理的成果がほぼ同じであり、どちらも莫大な人的、金銭的負担がかかるため、計画を統一する試みがここ数年進められてきた。その結果、2004年夏に、L-band超伝導空洞を用いたLCを採用することが決まった。[14]今後、具体的な加速器、各種コンポーネントの設計が進み、2015年の運転開始を目指して建設が行われることになっている。

XTFは当初GLCTA(GLC Test Accelerator)と呼ばれ、GLCに必要な仕様の加速管、高周波コンポーネントの実証試験を最大の目的として建設された。そのため、XTFの設計はGLC設計パラメータとして策定されていたものを踏襲している。現在、X-bandでLCを建設する試みが白紙に戻ったため、XTFも現在設置されているKX01加速管、現在製作が進められているKX02,03加速管の試験終了後シャットダウンすることになった。

ただし、X-bandの加速技術自体はLCに限らず広い応用範囲を持つため、XTFで試験が行われる加速管のデータも今後有用なものとなることは間違いない。

1.6 本論の構成

第2章以降の本論の構成は以下のようになっている。

- 第2章 XTF

  XTFのモジュレータ・クライストロン・導波管・加速管などのパラメータ・運転状況、運転監視・コントロールソフトウェアについて概説する。

- 第3章 放電検出・記録システム (XTF-BDMS)

Item	Stage I	Stage II	Unit
Center-of-mass energy (E_{CM})	500	1000	GeV
Luminosity	25	25	$10^{33}\mathrm{cm}^{-2}\mathrm{s}^{-1}$
Repetition rate	150	100	Hz
Bunch population	0.75×10^{10}		
Number of bunches / RF pulse	192		
Bunch separation	1.4		ns
Bunch train length	268.9		ns
Injected $\gamma\epsilon_x/\gamma\epsilon_y$	300 / 2		10^{-8} m-rad
Injected beam energy	8		GeV
$\gamma\epsilon_x/\gamma\epsilon_y$ at IP	360 / 4	360 / 4	10^{-8} m-rad
β_x/β_y at IP	8 / 0.11	13 / 0.11 mm	
σ_x/σ_y at IP	243 / 3	219 / 2.1	nm
σ_z at IP	110		μm
$<\Upsilon>$	0.13	0.28	
Pinch enhancement	1.49	1.42	
Beamstrahlung	4.6	7.5	%
Photons per e^+, e^-	1.26	1.30	
Loaded gradient	49.8	49.8	MV/m
Linac length / beam	7.25	14.11	km
Beam delivery length / beam	1.9	1.9	km

表 1.2: Basic design parameters of GLC.

XTF に設置されている放電検出・記録システムについて、放電検出器、データ収集系、放電検出法などについて詳しく述べる。

- 第 4 章 データ収集の経過

  XTF の運転履歴および、測定されたデータの仕様について概説する。

- 第 5 章 放電データの解析

  XTF-BDMS で収集された放電データの解析を行い、放電頻度、放電位置、放電メカニズムの解明について得られた知見を詳しく述べる。

- 第 6 章 まとめと今後の展望

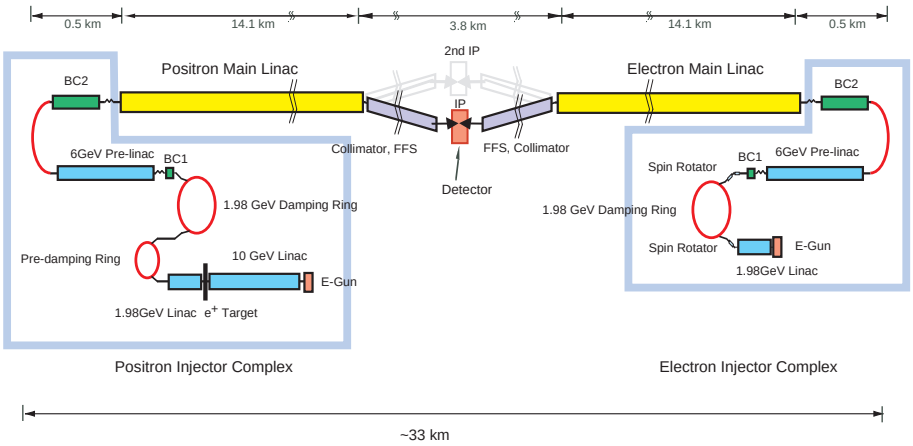

図 1.1: GLC の概念図。入射用の加速器群は見やすいように拡大されている。

2〜5 章の内容を整理し、あわせて今後の予定を述べる。

- 付録

  付録として、XTF-BDMS のデータ収集・放電検出を含む CAMAC,VME 放電部のソースコード (C++) を付記する。

このうち、筆者は第 2 章の XTF の建設・ソフトウェアの仕様策定に参加し、第 3 章・放電検出・記録システムでは測定器の設置・Calibration・データ収集系の設計・コーディング等すべてに中心的な役割を果たしている。付録に付記したソースコードは、ごく一部を除き全て筆者の手によるものである。第 4 章・XTF の運転管理にも主要な役割を果たしてきた。第 5 章・放電データの解析も、筆者が中心となってまとめたものである。

第2章　XTF

本実験では、XTF(X-band Test Facility) の加速管高電界設備を利用し、加速管に高電力を印加して、その放電特性を調べている。

XTF は、LC に必要な加速勾配 (ビームなしで ∼ 65 MV/m) を X-band で実現できる加速管、RF システムの実現・実証を目指し、2004 年春から運転を行ってきた。

本章では、XTF の目的と機能、設備、コントロール系について概説する。

2.1　XTF の目的と機能

XTF の目的および機能としては、以下があげられる。

2.1.1　RF 源の開発・試験

高電場を加速管に印加するためには、高電力 RF を効率的かつ安定に供給する RF 源が必要である。現在実用的な線形加速器のほとんどでは、RF 源としてクライストロンが用いられている。XTF の目的の一つは、X-band のクライストロンにて、高電力 RF を安定に生成するための技術開発・試験である。クライストロンには電力供給のためのパルス電源 (モジュレータ) も重要であり、安定、効率的なモジュレータの開発も並行して行っている。

2.1.2　放電の少ない加速管の製作

X-band LC の加速管には、現存する高エネルギー実験用線形加速器の 2〜3 倍に及ぶ高い電場での加速を設計している。高い電場勾配においては、加速管の放電をいかに抑えるかが極めて重要な問題である。XTF では、60cm X-band 加速管１基に 70MW 程度の RF を印加し、放電耐性や

放電の性質等について調べる。いくつかの仕様の加速管を差し替えて、その性能の比較も行っていく予定である。

2.1.3 高電場加速の実証

高エネルギー物理実験用として使用できる規模の X-band の加速器はまだ実用化されていない。XTF は、X-band 加速器の LC としての実用性を実証し、実用化をうながすための Demonstration を行い、さらに量産に向けての開発研究を推進する施設としての役割を課せられている。

2.2 XTF の設備

本節では、XTF の設備 (ハードウェア) について解説する。

2.2.1 設備の概要

XTF は、KEK(高エネルギー加速器研究機構) のアセンブリホール内に設置されている。アセンブリホール内では、XTF 以外に、LC 開発用の ATF(Accelerator Test Facility) の Linac およびダンピングリング、KEKB Crab Cavity の開発などが行われている。(図 2.1 を参照)

XTF の設備は 1,2 号 station,3 号 station の二つに大きく分けられる。1,2 号 station は加速管および RF コンポーネントの高電界試験用であり、3 号 station はクライストロン試験用の設備である。3 号 station は本研究と直接関係がないのでここでは詳しく述べない。

XTF1,2 号 station(以下、単に XTF と呼ぶ) は主に、直流電源およびモジュレータ、ソレノイド収束型クライストロン、電力輸送路 (主に導波管)、加速管から成り立っている。

XTF の全体図を図 2.2 に示す。XTF1,2 号 station の高電力 RF は 1,2 号機クライストロンで生成される。モジュレータはそれぞれのクライストロンに高電力を供給している。2 機のクライストロンの高周波 (RF) 出力は 3dB Coupler により合成されたのち、15m の電力輸送路 (矩形導波管 WR90 を使用している) でシールドルーム内に伝送される。

シールドルーム内では、導波管から送られた RF がまず 3dB Coupler で２つに分けられたあと、加速管の２つのポートに導入される。現在設置

されているのは、60cm 長加速管の KX01,1 基である。加速管内を伝わったRF は、２つの出力ポートより取り出され、ロードで吸収させている。

　加速管の周辺には、RF および放電イベントを観測するための種々の検出器が設置されており、パルス波形等を記録している。これらについては、3 章で詳しく述べる。

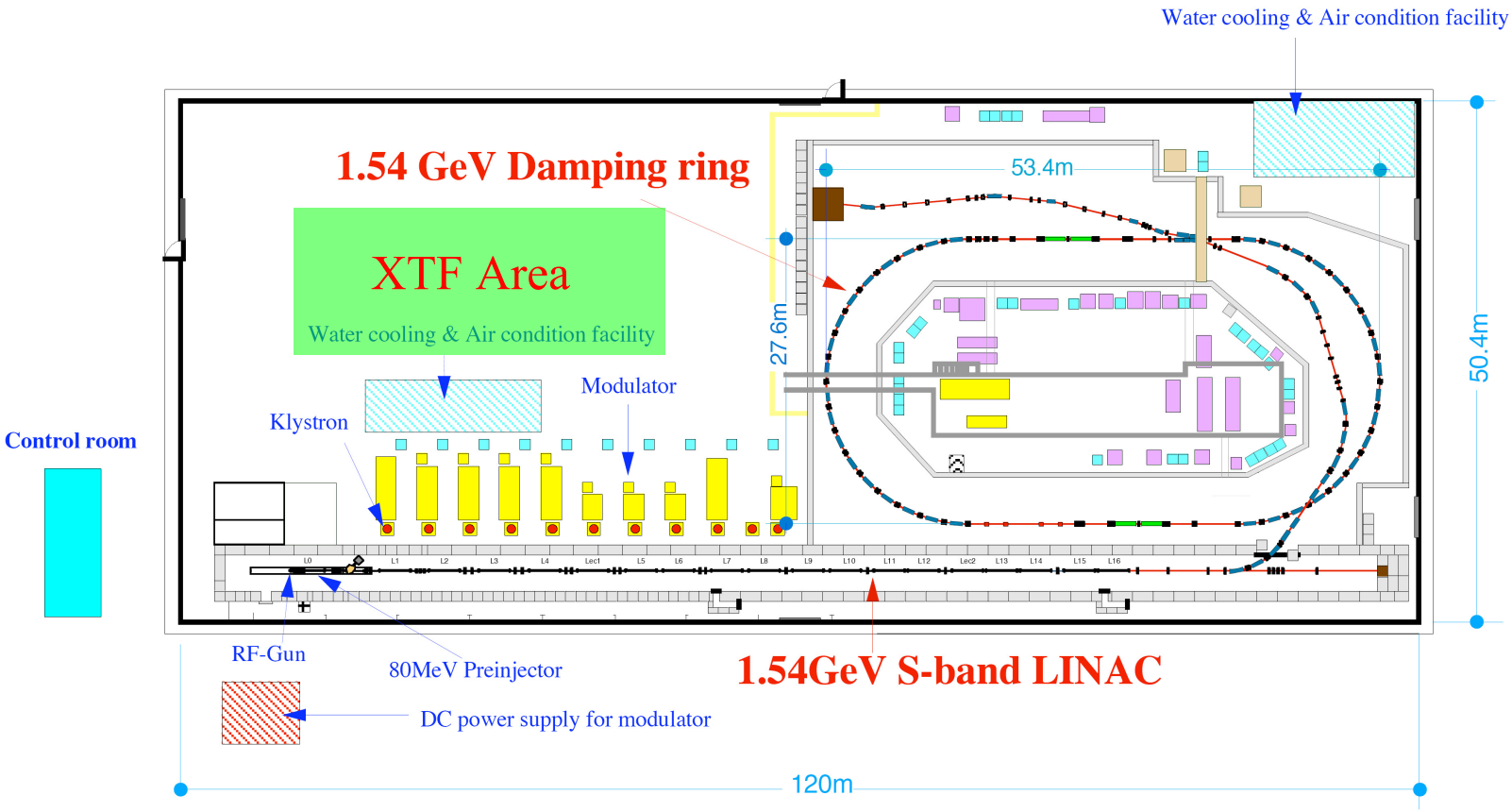

図 2.1: KEK アセンブリホールのレイアウト図。右半分のリングおよび下部の Linac が ATF の設備で、XTF はホールの左側、ATF Linac の上部に設置されている。

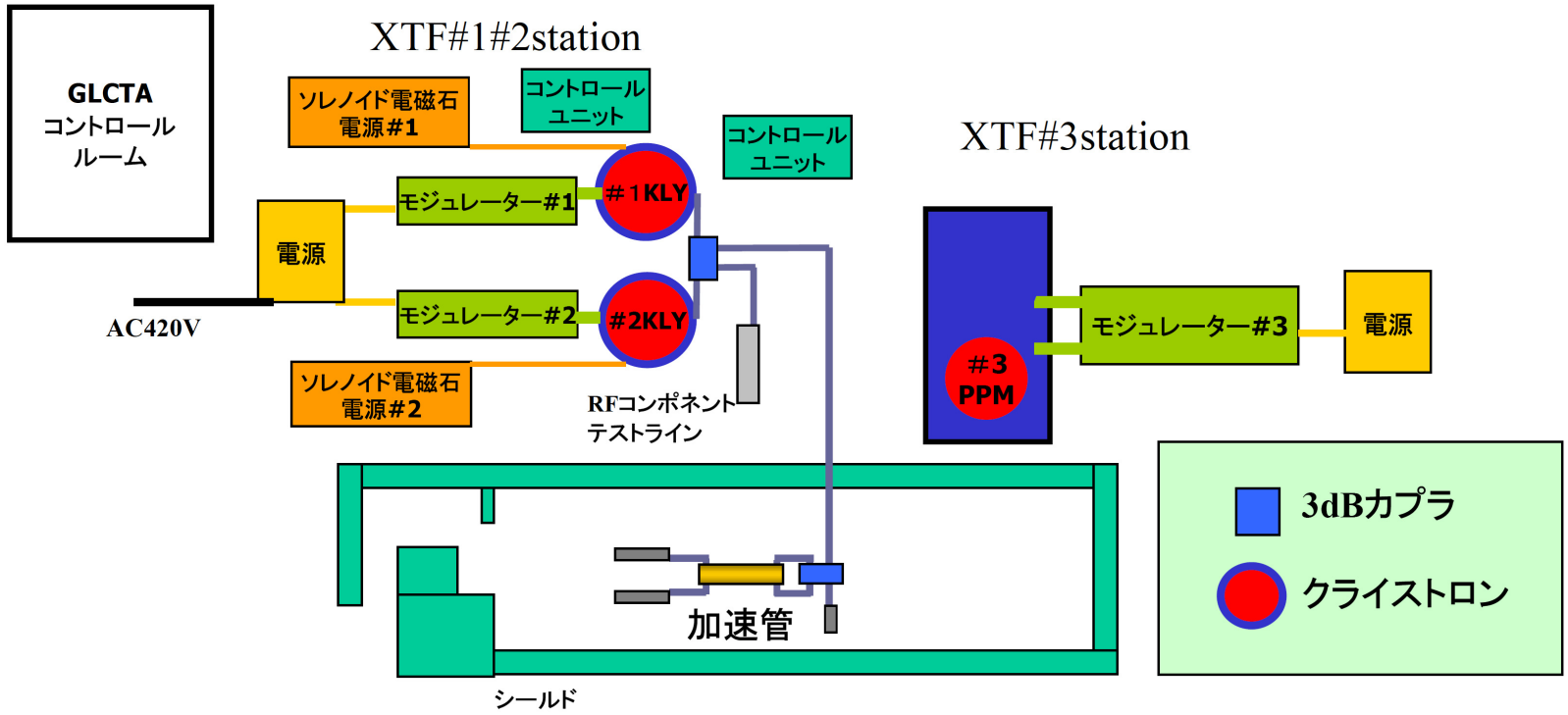

図 2.2: XTF のレイアウト図。

型番	XB-72 (ニチコン製)
LC 回路	6 段 (パルス幅:1.8 μs)
キャパシタンス	28 nF
リアクタンス	830 nH
出力インピーダンス	5.5Ω
最大充電電圧	90 kV
繰り返し周波数	0.625〜200 pps

表 2.1: PFN ユニット特性

2.2.2 モジュレータ・直流安定化電源

モジュレータおよび直流安定化電源は、クライストロンに高電力を供給するための電源設備である。XTF で使用されているのは、PFN(Pulse Forming Network) モジュレータと呼ばれるタイプのものである。

PFN モジュレータでの高電圧 (HV) 生成過程は次のようになる (図 2.5)。まず、商用 AC 三相 420V を安定電圧化し、直流化する。直流電流はチャージングコイルを通って PFN 内のコンデンサーに共振充電される。PFN ユニットのスペックを表 2.1 に示す。PFN に蓄えられた電荷はサイラトロンでのスイッチング動作により数 μs の間に直流パルス電流として取り出され、クライストロンのパルストランス部に送られる。

ここでは、安定な HV の供給を行うために共振充電における最大値を用いず、ある基準電圧に達したところで充電を抑えるようにチャージングコイル 2 次側のコイルに電流を流し込む De-Q 回路が組み込まれている。この De-Q 回路により PFN 充電電圧の安定化を計っている。

クライストロンのパルストランス部に送られたパルスは 1:12 に変圧され、クライストロン用熱電子銃のアノード（陽極)、カソード（陰極）間に印加される。

2.2.3 RF 源と RF Power Control

クライストロンに入力する RF 源としては、Signal Generator(SG) で生成した RF をパルス整形後、TWTAmp で増幅して用いている。

Signal Generator(SG) は Agilent E8241A[1]を採用している。周波数は

[1]http://literature.agilent.com/litweb/pdf/5988-2411EN.pdf

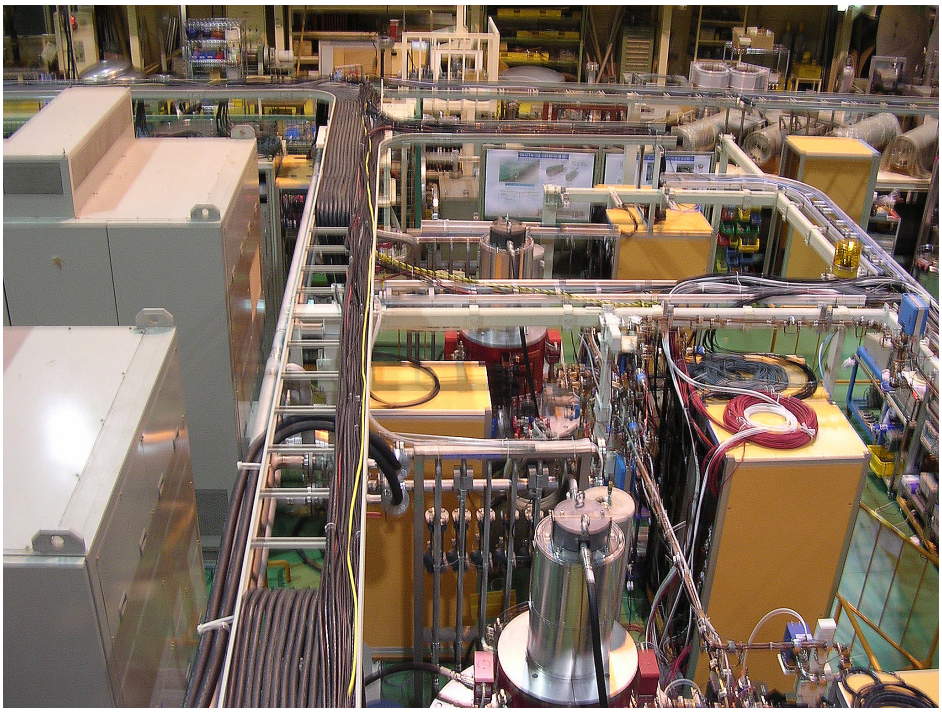

図 2.3: XTF シールド外部全体図。左側の白いボックスがモジュレータ、中央の円筒状のものがクライストロン。写真手前 (写っていない) がシールドルーム。

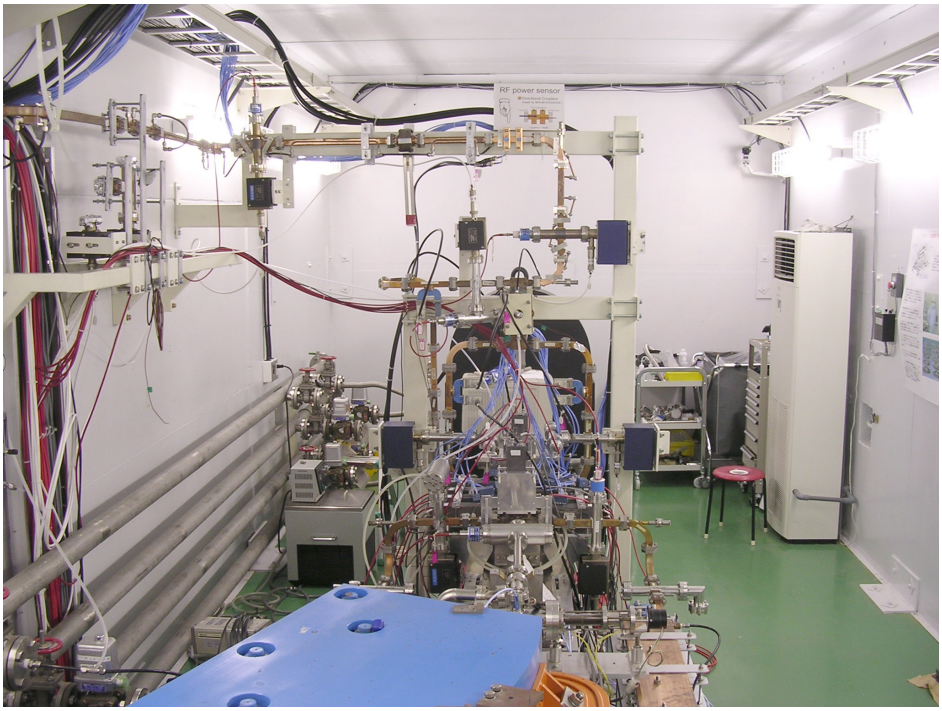

図 2.4: XTF シールド内部図。

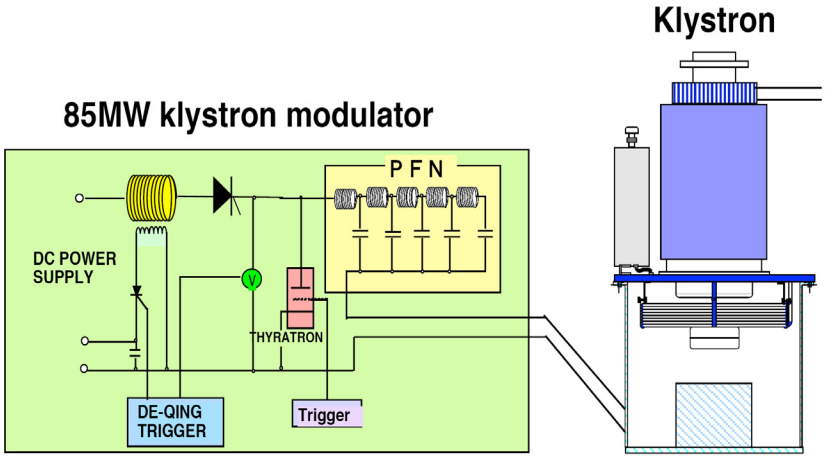

図 2.5: モジュレータ概念図

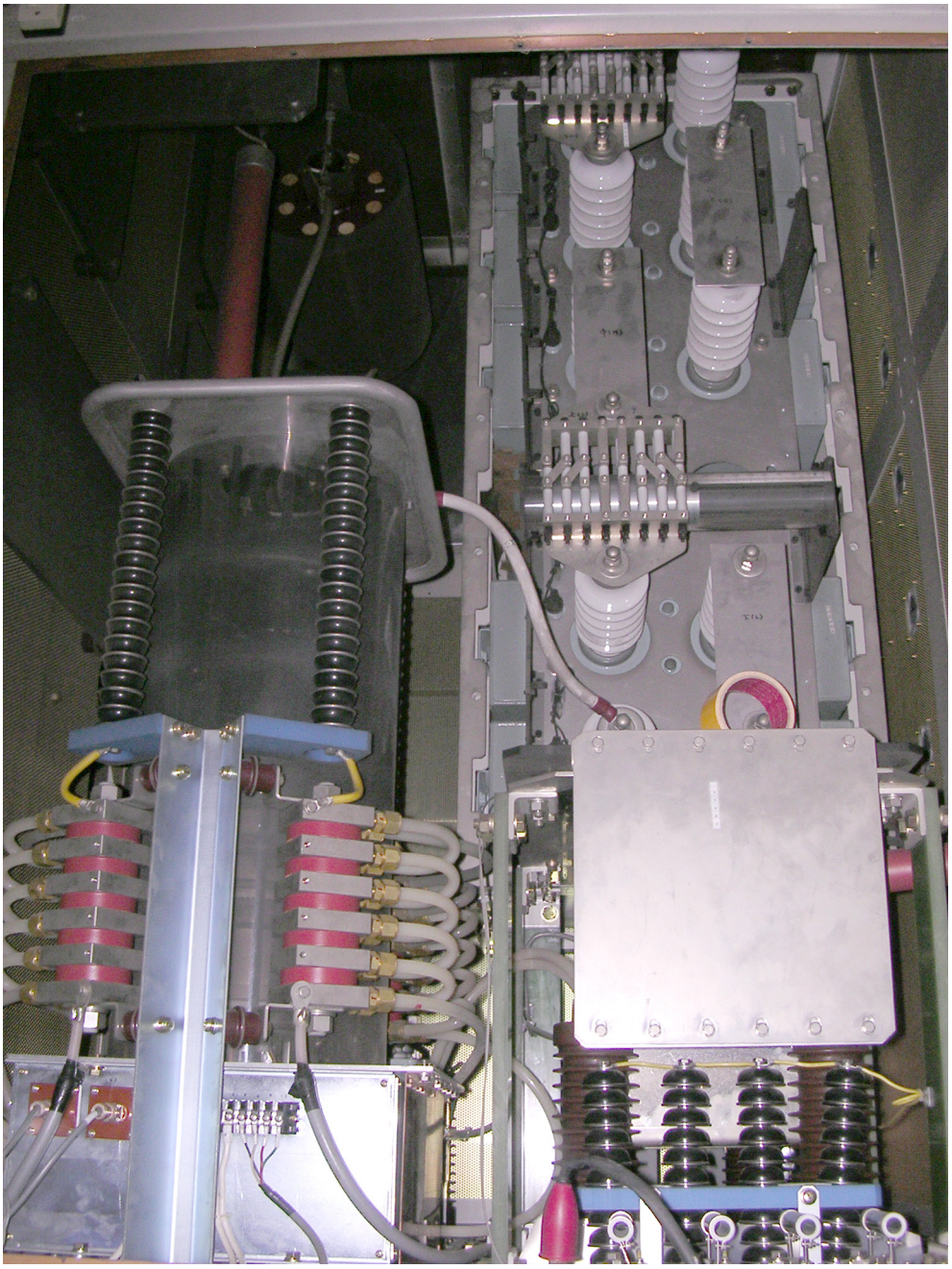

図 2.6: モジュレータ内部写真。右上部の白い円筒状のものが PFN ユニット、左の黒い筒状のものがスイッチ動作を行うサイラトロン。

11.424GHz の CW であり、運転出力は 6.00dBm となっている。

この SG の出力は、パルス成形モジュールでパルス化される。このパルス化のタイミングを与えるトリガーの ON/OFF はソフトウェアで管理されており、BDMS(3 章を参照) によりコントロールされている。パルス幅、トリガーからパルス切り出しまでの Delay はモジュールで設定可能である。

パルス化された RF は、RF Amp. で 13dBm まで増幅した後,RF 可変減衰器でレベルコントロールを行い、TWTA(Traveling Wave Tube Amplifier) で SG の出力をクライストロンの RF 入力として使用可能なレベルまで増幅される。RF 可変減衰器の減衰比は XTF コントロールにより UP/DOWN コントロールを通じてリモート制御されており、ここでクライストロンからの出力 RF のコントロールを行っている。

クライストロンの出力 RF パワーを制御するには、クライストロンの電子銃に印加する HV(モジュレータの出力) を調節する方法もある。この方法の方が安定した出力を期待できるが、XTF では制御を簡便にするためクライストロンの入力空洞に導入する RF パワーを上記の方法で制御する方法を用いている。

XTF の TWTA は Applied Systems Engineering の model 117[2]を採用しており、増幅率は 10^6 倍程度、最大出力は 1KW である。

これらの SG,TWTA は 1 号機,2 号機で共通となっており、TWTA 出力を分岐した後、1 号機については TWTA とクライストロンの距離がややあるため導波管 (WR90, 常圧) を用いて伝送している。導波管には位相調整ユニットが設置されており、1 号機と 2 号機の相対位相を調節して、加速管に最大の電力を送るよう設定している。

2.2.4 クライストロン

XTF では、高電力 RF の生成にクライストロンを用いている。

クライストロンは電子ビームの RF による速度変調を用いて RF の増幅を行うマイクロ波管の一種で、高電力が必要な線形加速器のほとんどで RF 源として利用されている。

クライストロンは図 2.10 に示されているように、主に熱電子銃、入力空洞、中間空洞、出力空洞、コレクターからなっている。電子銃で熱電子放出される電子ビームは印加されたパルス電場により加速され、入力空

[2]http://www.applsys.com/model117data.PDF

図 2.7: Signal Generator Agilent E8241A

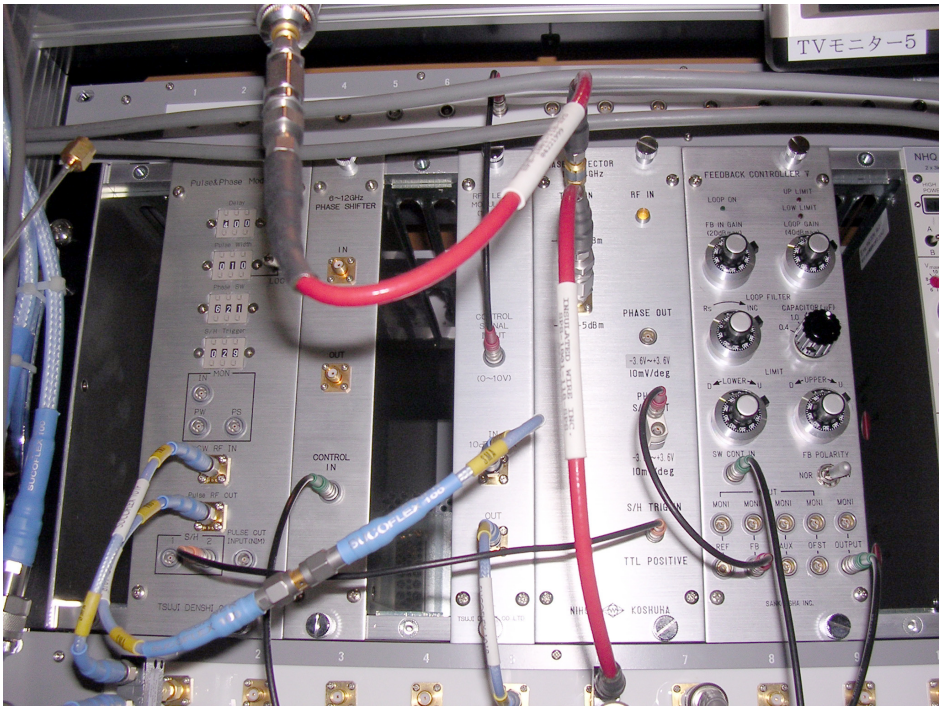

図 2.8: RF パルス整形回路。左のモジュールがパルス成形モジュール。左から 3 つめのモジュールが RF パワーコントロール用の可変減衰器。

図 2.9: TWTA Applied System model 117

動作周波数	11.424 GHz (X-band)
RF 出力	∼ 50 MW
パルス幅	∼ 800 ns
繰り返し周波数	∼50 Hz
印加電圧	∼500 kV

表 2.2: XB72K XTF クライストロン性能表。設計仕様とは異なるが実際に運転可能なパラメータを示した。

胴、中間空胴、出力空胴の空胴を通過していく。電子ビームは入力空胴を通過するときに、入力 RF による電場により速度変調を受けバンチングされる。このバンチングされた電子ビームが出力空胴を通過するとき空胴内に電磁場を励起し、結果として大電力の RF パルスが取り出される。XTF1,2 号 station で使われているクライストロンは電子ビームの収束に電磁石を用いるソレノイド収束型クライストロンである。この方式では電磁石の電力消費が大きいため、現在電磁石の代わりに永久磁石を用いた PPM(Periodic Permanent Magnet) クライストロンが実用化されている。XTF3 号 station のクライストロンは PPM 型である。

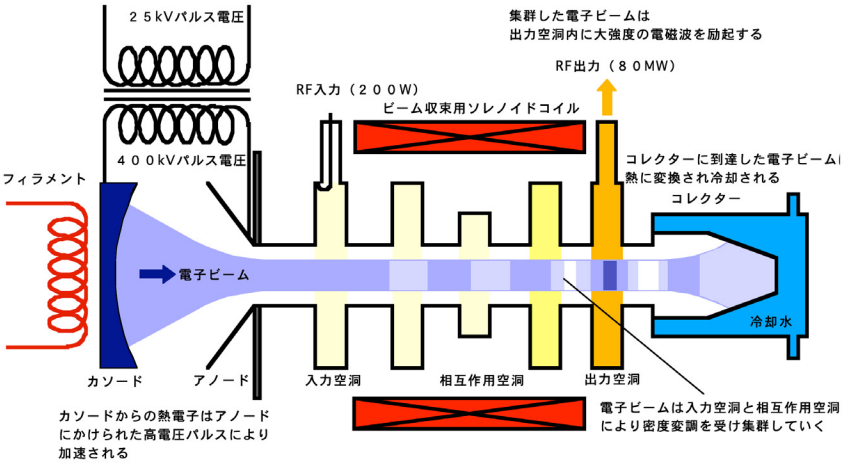

図 2.10: ソレノイド収束型クライストロン動作模式図

2.2.5 導波管

2つのクライストロンの出力は、合成されたあと、約15mの導波管WR90によって、シールドルーム内の加速管まで伝送される。出力の合成には3dB Coupler(図2.11)を用いている。

WR90(図2.12)は、$a = 22.86\text{mm}, b = 10.16\text{mm}$の矩形導波管である。11.424 GHzはTE10モードで伝送される。伝送損失αは次式で表される。[15]

$$\alpha = \frac{R_s}{b\zeta\sqrt{1-\left(\frac{\lambda}{\lambda_c}\right)^2}}\left[1+\frac{2b}{a}\left(\frac{\lambda}{\lambda_c}\right)^2\right] \tag{2.1}$$

$R_s = 7.8\times10^{-4}[\Omega]$は表皮抵抗(銅)、$\zeta = 377[\Omega]$は(真空の)波動インピーダンス、$\lambda = 26.26[\text{mm}]$は伝送波長、$\lambda_c = 45.8[\text{mm}]$はカットオフ波長である。

これらの値を代入して、

$$\alpha \sim 0.100[\text{dB/m}] \tag{2.2}$$

を得る。今回利用する導波管全長がおよそ15mであるので、全体での損失は

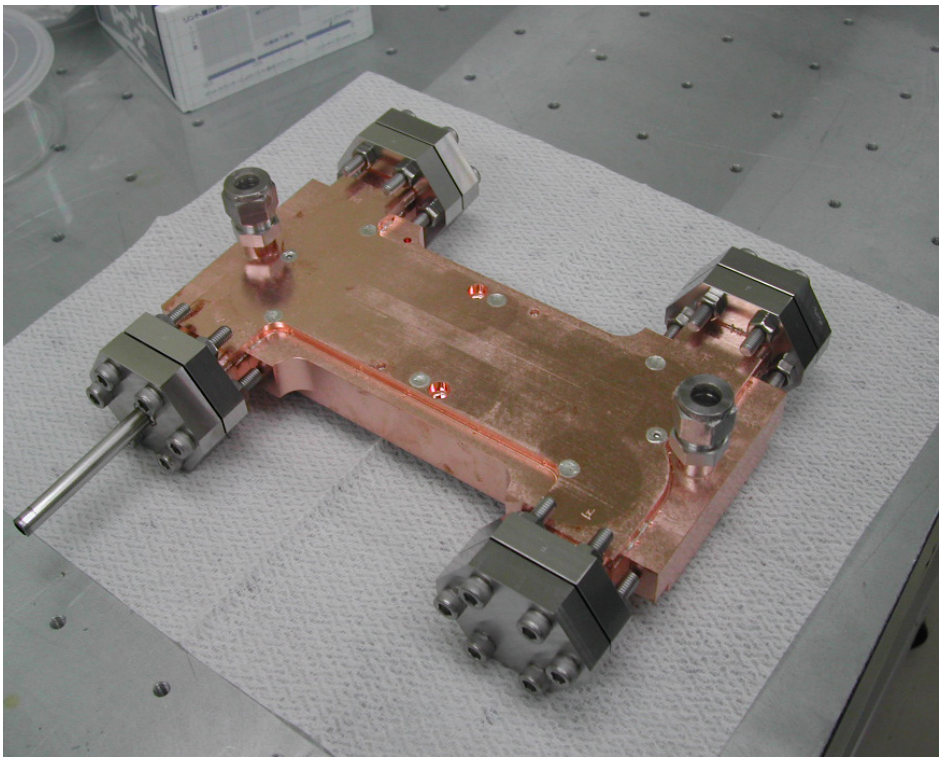

図 2.11: 3dB Coupler

$$1 - (10^{-0.100 \times 15}) \sim 29.2\% \tag{2.3}$$

となる。ただし、実際には導波管のフランジの SUS による損失も大きく、30%以上の損失となっている。

現在、この導波管をより伝送損失の少ない円形のものに交換する予定となっており、それにより加速管に供給する電力が大きく増えると期待されている。

2.2.6 加速管

加速空洞は、入射ビームの進行方向にビーム入射時に電場が立つような RF を保持することで、ビームを加速する。加速空洞 (セル) の集合体が加速管である。加速管では一般に構成するセルを電場または磁場で相互結合することで、単一の入射ポート (カプラ) で多数のセルに電場を立

図 2.12: X-band 導波管 WR90

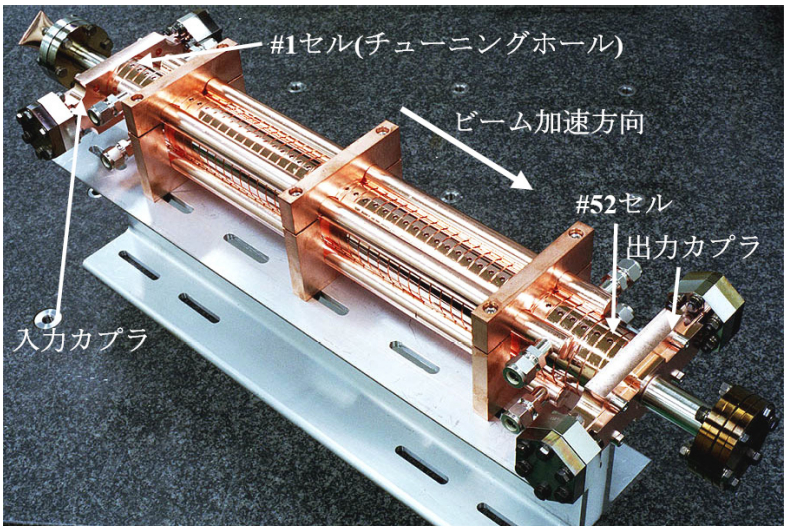

図 2.13: X-band 加速管 KX01

てる構造になっている。

加速管には進行波型と定在波型加速管があるが、高電場勾配には進行波型が有利であり、XTF の加速管も進行波型である。加速管長は長い方が接合部の割合が少なくなるため加速に有利だが、RF 技術上難しい問題がいろいろと生じる。

XTF には、入出力 2 ポートタイプの X-band 加速管を 1 基設置することが可能である。X-band 加速管は 1 章で述べたように KEK と SLAC で長年共同開発されており、XTF でも KEK-SLAC で現在研究開発がすすめられている加速管の試験を行っている。加速管長は 60cm である。

XTF に現在設置されている加速管 (KX01:図 2.13) の仕様を表 2.3 に示す。

加速管には放電監視用の各種センサーが設置されているが、それについては 3 章で詳しく述べる。

2.2.7 真空系

高電場を伝送するクライストロン〜加速管までの導波管、クライストロン、加速管は、放電を最小限に抑えるため超高真空を保つ必要がある。このため、XTF には全部で 26 基のイオンポンプ (排気量 10〜20[l/s]:図 2.14) が設置されている。到達真空度は、2×10^{-7}[Pa] 以下である。

全長	60 cm
セル数	53
運転周波数	11.424 GHz
セル当たりの位相進み	$5/6\pi(150°)$
減衰定数	0.54
v_g/c	$1 \sim 4\%$
充填時間	104 ns
ビームなし加速電場	65 MV/m
パルス幅	400 ns
入力 RF	63 MW

表 2.3: KX01 パラメータ

真空の計測は、CCG(Cold Cathode Gauge) で行われており、ペンレコーダーで記録されるとともに、モジュレータのインターロックに入力されている。一定の真空度を上回ると、モジュレータが自動的に停止する仕組みである。

2.2.8 各種センサー

XTF では、その他、加速管、導波管等各所の温度センサー、冷却水の流量モニター等が設置されており、パラメータ記録、異常検知を行っている。

2.3 XTF のコントロール系

次に、XTF のコントロール系 (ソフトウェア) について述べる。ここで述べるのは、XTF の運転に必要な、ステータス記録、インターロック管理、自動 Processing 等の機能である。XTF では、これに 3 章で述べる放電モニターを加えて、通常の運転がなされている。

2.3.1 計算機設備

はじめに、XTF で使われている計算機設備を概観しておく。

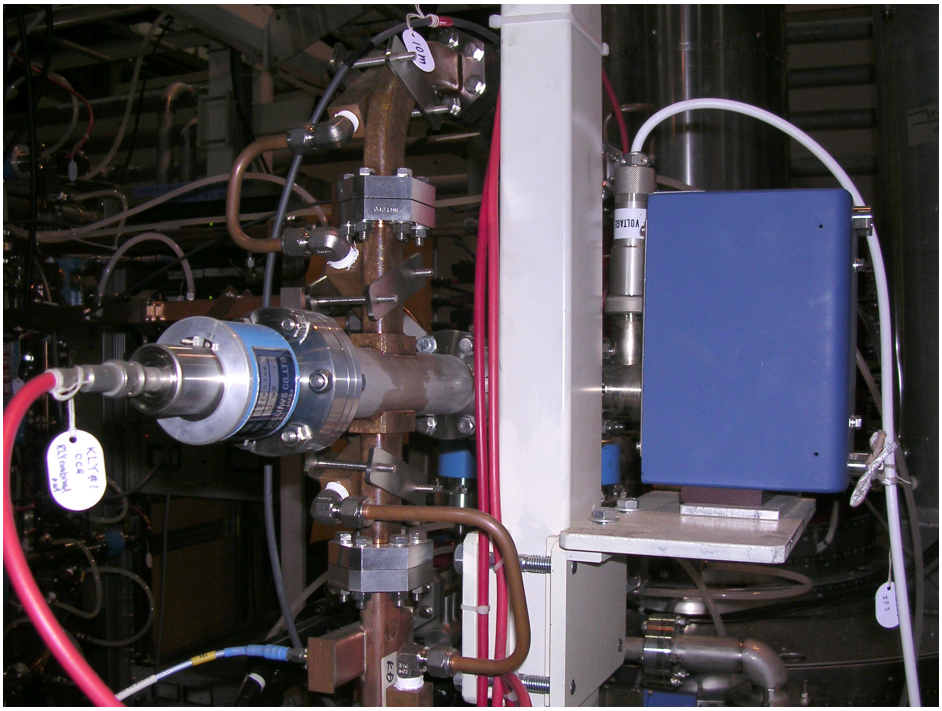

図 2.14: 導波管真空ポート周辺。右の青いボックスがイオンポンプ。左の円筒状のものが CCG。

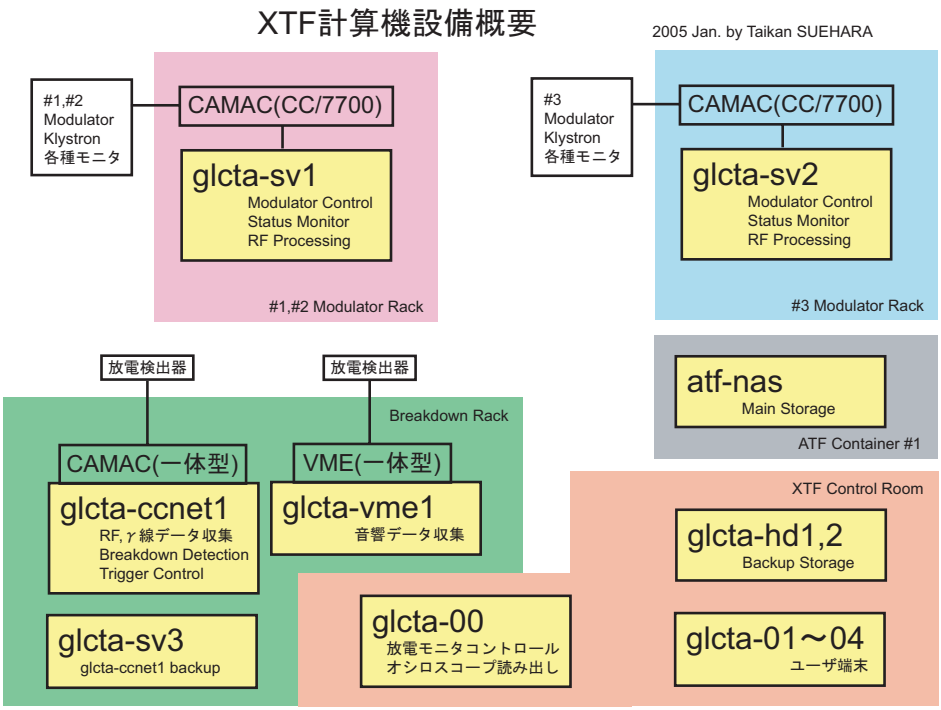

図 2.15: XTF 計算機設備概略図

DNS 名 (IP)	用途	計算機仕様
glcta-00 (130.87.70.170)	コントロール用 常駐ソフトウェア	HP workstation xw6000 Xeon 2.8 GHz×2, 2GB Memory 160GB×2 (RAID0) HDD, Fedora core 2
glcta-[01〜04] (130.87.70. [117,118,171,172])	コントロール ユーザ端末	HP workstation xw4100 (Dual Display) Pentium4 2.8 GHz, 1GB Memory 40GB HDD, Fedora core 2
glcta-sv1 (130.87.70.125)	XTF station#1,2 Control, CAMAC 通信 (Modulator Status 等)	Celeron 1.4GHz 500MB Memory 40GB HDD, RedHat Linux 9 60°C 動作可能
glcta-sv2 (130.87.70.126)	XTF station#3 Control, CAMAC 通信 (Modulator Status 等)	Celeron 1.4GHz 500MB Memory 40GB HDD, RedHat Linux 9 60°C 動作可能
glcta-ccnet1 (130.87.70.135)	XTF-BDMS Control CAMAC 通信 (放電モニタ)	Toyo CC/NET CAMAC Crate Controller 一体型 PC Crusoe TM5400 500MHz, 512MB Memory 1GB CompactFlash, CC/NET 専用 Linux
glcta-sv3 (130.87.70.127)	XTF-BDMS Backup	Celeron 1.4GHz 500MB Memory 40GB HDD, RedHat Linux 9 60°C 動作可能
glcta-vme1 (130.87.70.140)	XTF-BDMS VME データ収集	VMIC VMIVME-7750 VME Crate Controller 一体型 PC Pentium3 733MHz, 512MB Memory 40GB HDD, RedHat Linux 9
atf-nas (130.87.70.185)	Data Storage	evServ NAS 660i Celeron 1.2 GHz, 256MB Memory 120GB×6 (RAID5:600GB) HDD NAS 用 OS(詳細不明)
glcta-hd[1,2] (130.87.70. [119,120])	Backup Data Storage	LHD-NAS250 C3 533MHz, 128MB Memory 250GB HDD, Synology OS BASIC

表 2.4: XTF 計算機設備概要

glcta-sv1,sv2には、CAMACのクレートコントローラ (Toyo CC/7700)が設置されており、それぞれ 1&2,3 号 station のコントロールを行っている。

glcta-ccnet1,glcta-vme1,glcta-00 は BDMS 用の PC である。

glcta-ccnet1は、CAMAC クレートコントローラ一体型 PC、CC/NETである。ハードディスクは搭載しておらず、内蔵の CompactFlash で起動する。glcta-vme1 は VME クレートコントローラ一体型 PC である。このPC は HDD を接続し起動している。glcta-00 は、放電検出ソフトウェアのコントロール部がおかれている。これについては 3 章で詳述する。

glcta-01〜04 はユーザー端末であり、２画面の液晶ディスプレイが接続されている。モジュレータコントロールのインターフェース、放電検出設定ソフトウェア、放電イベントディスプレイなどはここで動作する。また、オフライン解析もこのマシンを利用できる。

以上すべてのマシンは Linux 上で動作している (Distribution は表 2.4 に示した)。実験に用いるディスク領域は、RAID5ディスクマシン atf-nas から NFS でマウントして共通に用いている。glcta-hd1,2 は atf-nas のバックアップディスクである。

atf-nas の主なディレクトリ構成を表 2.5 に示す。atf-nas のマウントポイントは/mnt/nas であり、上記全マシン共通である。

2.3.2 コントロール系の概要

図 2.16 に XTF コントロール系のブロック図を示す。

赤矢印は大電力系、青矢印は X-band RF 系、緑矢印はトリガー系、紫はインターロック系の動作を示している。また、橙の枠は計算機およびそのインターフェースであることを示し、橙の矢印は計算機によるコントロールを示している。黄緑の枠は、計算機のインターフェースが直接通信する相手を示している。

大電力系、RF 系については前段ですでに説明したので、以下ではトリガー系、インターロック系および計算機制御について述べる。

ハードウェアによるコントロールとしては、トリガー系およびインターロック系が重要である。トリガー系は、モジュレータ、クライストロン等の大電力系、RF 系の動作タイミングの指示だけでなく、各種モニタ、放電検出系の読みとりタイミングの指示等にも使われ、それぞれ適切な遅延時間で各所に配られている。

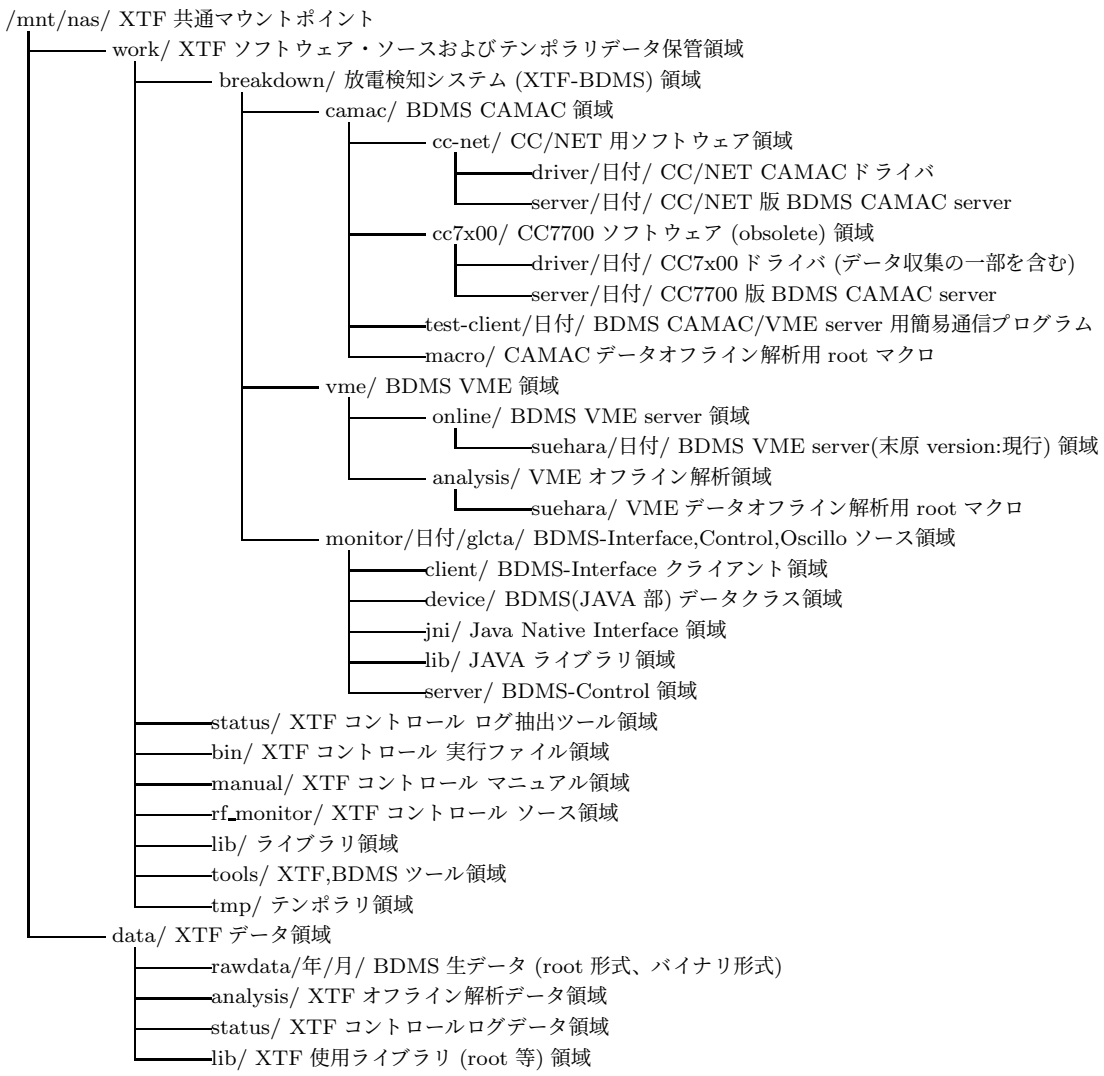

表 2.5: XTF コントロール ディレクトリ構成図

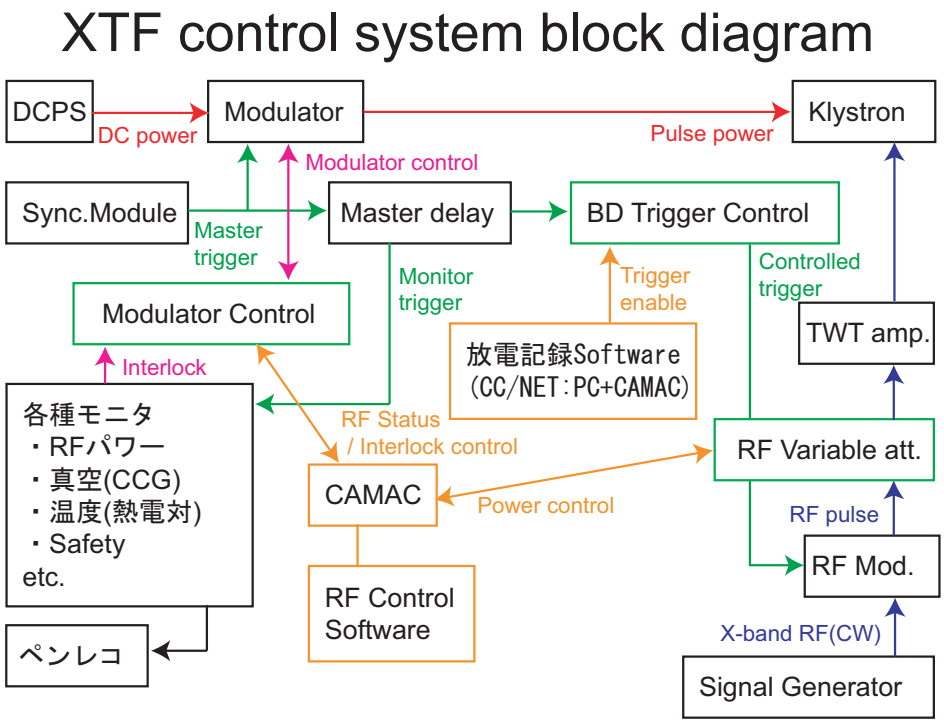

図 2.16: XTF コントロール ブロック図

インターロック系は、真空、温度、放射線等の異常を検知し運転を緊急停止するもので、このような大電力系では必須のシステムである。XTFでは、インターロック系はモジュレータコントロールに集約され、モジュレータに供給する次のトリガーを停止する。

XTF のインターロックは、加速管や導波管の Processing に伴う放電の真空悪化を検知するため、かなり頻繁に作動する点が特徴的である。このため、深刻でない (主に真空や RF 波形欠け等の) インターロックは RF Processing ソフトウェアによってインターロック発生から一定時間後、真空が一定値以上に回復していれば自動的にリセットされ、運転が再開される仕組みをとっている。

計算機制御は、大きく分けて RF Control 系、放電記録系の 2 つに分かれている。放電記録系の具体的な機能の説明は次章に譲るが、コントロール系への寄与としては、ブロック図に示されているように放電検出時に RF パルス成形器へのゲートをストップすることでクライストロンからの RF 出力を停止する点がある。

RF Control 系は、XTF コントロールシステムを統括しており、モジュレータコントロールを通して運転状態の管理、インターロックからの復旧を行う機能、各種モニタの値を記録する機能、RF 減衰器を通して出力 RF パワーを制御する機能等がある。また、これらを総合的に用いて加速管等の Processing を行う機能が用意されている。これらについても、後段で詳しく述べる。

2.3.3 トリガー系

ブロック図に示したように、トリガー系では、モジュレータ、RF パルス成形モジュール、各種モニターに動作タイミングを供給している。また、BDMS もこのトリガーを用いて動作している。

トリガー源としては、NIM の LINE SYNC モジュール 2.18が用いられている。このモジュールは、0.39〜150Hz($\frac{50}{2^n}(n = 0 \sim 6)$ と 100,150Hz,AC50V に同期) に対応したトリガーを発生する。XTF のトリガーはすべてこのモジュールから発信されたトリガーを分岐して用いている。

トリガー源を出たトリガーは、図 2.17 に示したように、6 つに分岐される。#1,#2 のそれぞれについて、モジュレータ, モニタ系へのトリガーと RF 系へのトリガー (TTL,NIM) である。#2 モニタへのトリガーは BDMS にも送られる。

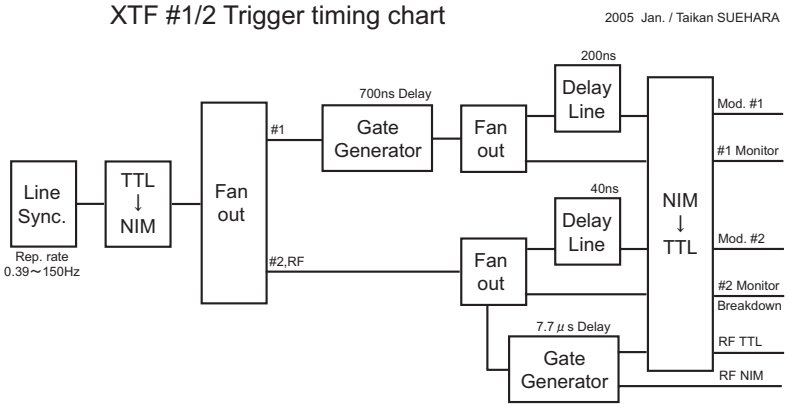

図 2.17: トリガータイミング図。Delay Lineはケーブルによる Delay,Gate Generator の上部は Gate による Delay を示している。

これらのトリガーはそれぞれ異なった Delay が設定されている。#1 と #2で Delay が異なるのは、モジュレータ、クライストロンの応答時間が #1 と #2では異なるためである。モニタ系へのトリガーは、Peak Hold モジュールのスタート信号として使われ、RF パルスのパワー測定に利用されているため、モジュレータへのトリガーよりやや早めに配られる。

RF 系へのトリガーに大きな Delay が入っているのは、モジュレータにトリガーを加えてからクライストロンへの印加電力が最大になるまで数 μs の時間を要するためである。

また RF 系へ送るトリガーは、一度 BDMS 上の CAMAC の CSY モジュールを経由し、放電検出時にパルスを停止するコントロールを行っている。BDMS によるトリガー管理については、3.3.5 節に詳述する。

2.3.4 インターロック系

先に述べたように、インターロック系はモニターしている真空、温度、各種電圧、水量、放射線、等に異常が生じた場合にモジュレータ等を緊急停止する機能である。

インターロック系は、LV,HV,Trigger の 3 系統に分かれている。LV,HV インターロックには各種ファンのステータス、冷却水の流量モニタ、温度センサ等が入力されており、オーバーヒートや電圧異常を防ぐ。LV インターロックは、モジュレータへの電力供給を停止し、HV インターロック

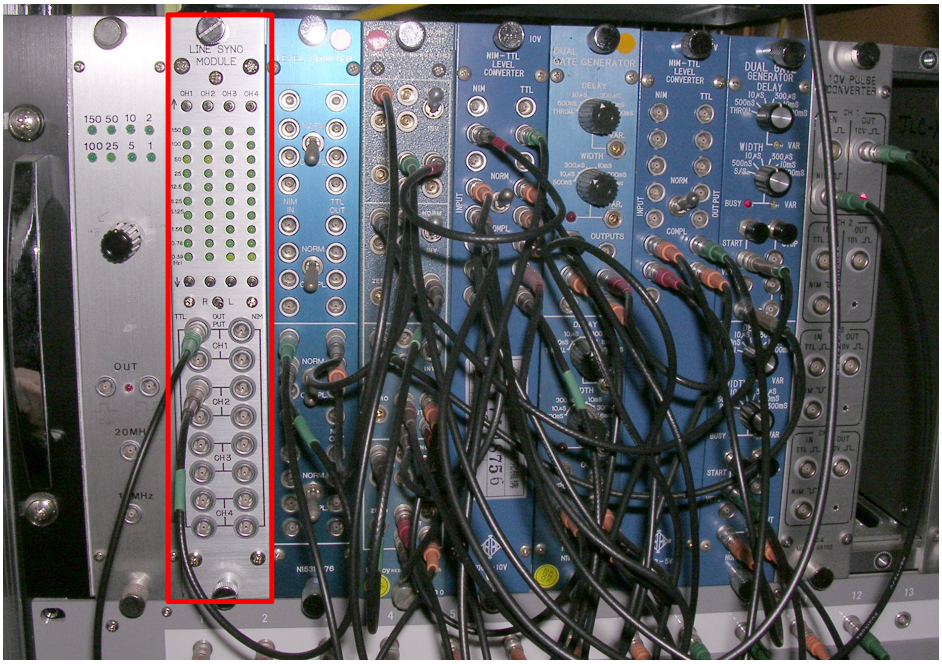

図 2.18: LINE SYNC モジュールとトリガー制御系。LINE SYNC モジュールは左から二つ目のモジュールで、設定した周波数のクロック信号を出力する。その他の各種モジュールは、モジュレータ、クライストロン、各種モニタに適切なディレイ・規格 (NIM,TTL) の信号を分配・生成し出力している。

は PFN への charging を停止する。

Trigger インターロックは、モジュレータのスイッチング動作を停止するものであり、主に各種真空系や RF 波形モニタ等を入れている。また、シールドルームのロック、放射線インターロックもこの Trigger インターロックに入力している。Trigger インターロックは加速管や導波管の放電に伴う真空悪化、シールドルームの開閉等により頻繁に作動する。

これらのインターロックは、各種モニタよりモジュレータコントロールに接点入力で配線されており、モジュレータコントロールによって常時監視されている。インターロックの接点が OPEN になると、該当する LV,HV または Trigger は即座に停止され、該当するリセット信号を受信するまでは再度 ON にすることができない。ただ、Safety を除く Trigger インターロックについては、後述する RF Processing ソフトウェアの指示により、自動的にリセット、復旧する。それ以外のインターロックはモジュレータコントロールパネルより手動でリセットする必要がある。

また、モジュレータインターロックとは別に、Fast RF Interlock というインターロック系統を設けている。これは、NIM モジュールでコントロールされており、インターロックの各チャンネルに到達するアナログ信号の強度があらかじめ設定されている閾値を超えた場合にインターロックを発行する。インターロックが発行されると、RF power control 用の減衰器の減衰値が最大になる (出力パワーは数 MW 以下に減少する)。

現在監視しているのは、加速管からの RF の反射強度である。Peak Hold モジュールを通すことにより、反射の最大値が一定以上の場合インターロックが作動する仕組みになっている。

さらに、放電モニタはインターロックとは別に、RF パルス成型モジュールの Gate を止めることにより RF の出力を止めることが可能である。この場合、モジュレータインターロックではないためモジュレータは停止しないが、現在のシステムではソフトウェアで一旦モジュレータを停止するシーケンスになっている。

以上のような各段のインターロックにより、XTF では異常時の確実なシステム停止を実現している。

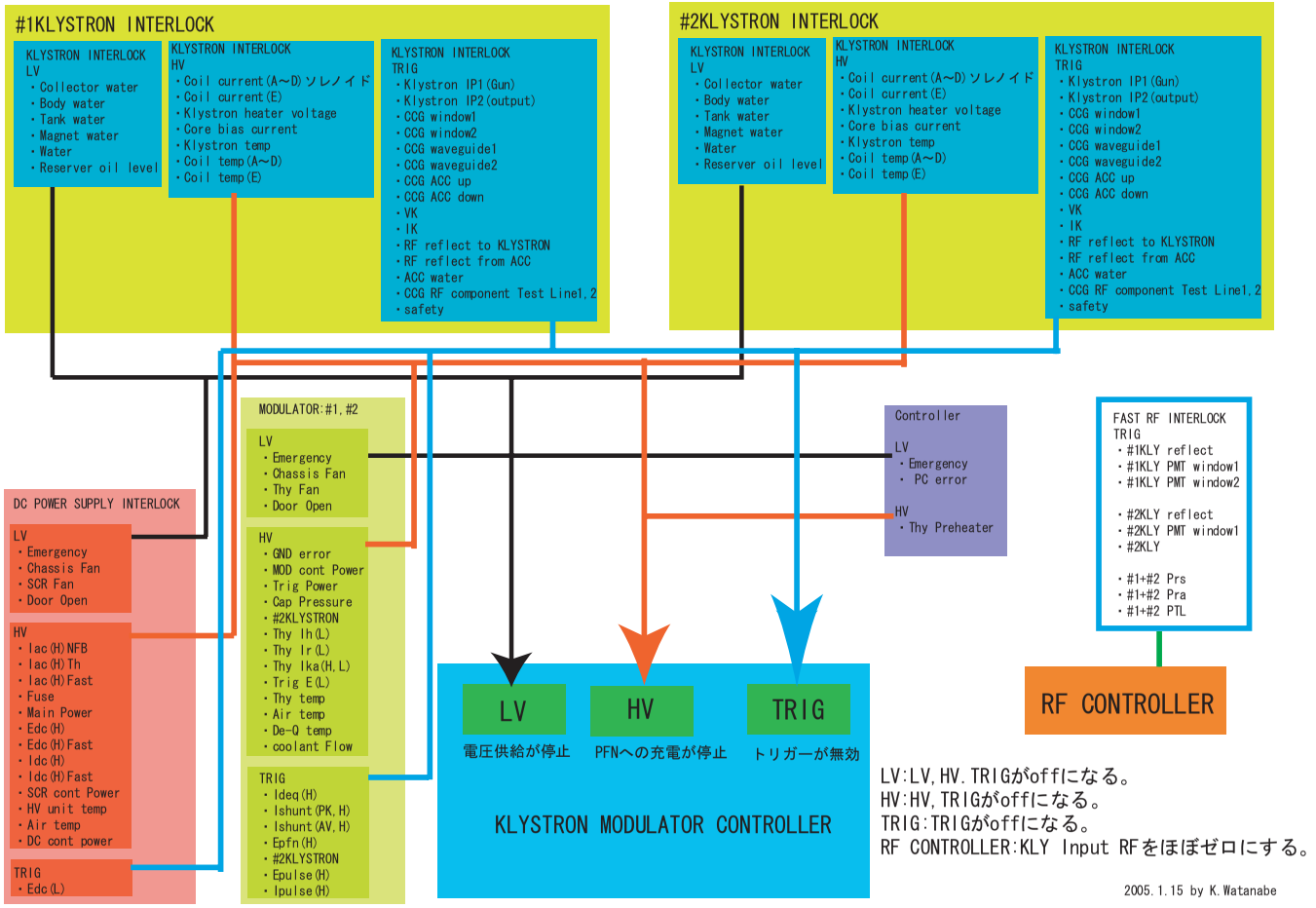

図 2.19: インターロック系模式図。冷却水やファン等の異常が大事故につながるインターロックは LV に、電源関係の異常は主に HV に Power や真空関係は Trigger に入力されている。RF インターロックは主に波形の評価で使用されている。

番地 (16 進)	サイズ (16 進)	形式	内容
0000	100 Bytes	string	記録開始日時
0100	100 Bytes	string	記録終了日時
0200	4 Bytes	int	項目数 (N)
0204+n×104+0($0 \leqq n \leqq N$)	100 Bytes	string	項目 n 名前
0204+n×104+100($0 \leqq n \leqq N$)	4 Bytes	int	項目型 (int or float)
0304+N×104+i×14C+0	30 Bytes	string	ログ時刻 t
0304+N×104+i×14C+4×n	4 Bytes	int or float	t での n の値

表 2.6: XTF Status data ファイルフォーマット。ファイルは 1 日に 1 つ、/mnt/nas/data/status/sv1/yyyymmdd.log の形式で作られる。記録間隔はほぼ 1 秒に 1 データ。

2.3.5 XTF コントロールソフトウェア

次に、XTF コントロールのソフトウェア部分について述べる。図 2.20 は、XTF コントロールのソフトウェア部のブロック図である。

コントロールの主要部は、glcta-sv1(以下、sv1) 計算機上で動作している。sv1 には、CAMAC クレートコントローラ (TOYO CC/7700) が接続されている。CAMAC には、各種ステータスを読みとるための 32ch ADC とモジュレータコントローラのインターロックステータス等を読みとるための SIG(Signal Input Gate) モジュールが搭載されている。CAMAC との通信は、sv1_control,sv1_monitor プロセスが担当している。

CAMAC を介して取得されたステータスデータは、一旦データベースに書き込まれ、データロギング、コントロール端末、Processing ソフトウェアはこのデータベース上からデータを取得する仕組みである。

2.3.6 ステータスデータの記録

ステータスデータは、ロギングソフトウェアによって、データベースから読み出し、保存されている。現在の保存レートは秒 1 回で、保存しているデータは表 (ファイルフォーマット) の通りである。一日ごとにファイルが作られるが、現在一日あたりのデータサイズは 100MB に及ぶ。これらのデータファイルは、後述のコントロールインターフェースから参照するほか、手動で解析も行っている。

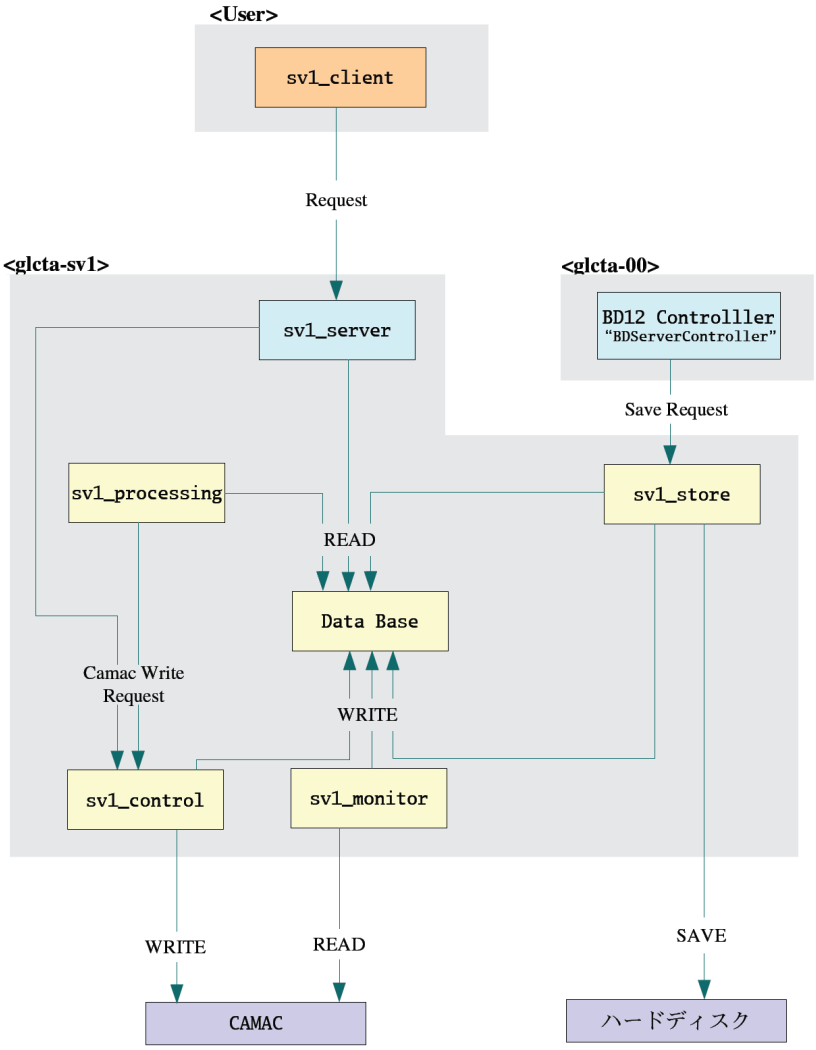

図 2.20: コントロールソフトウェア概念図

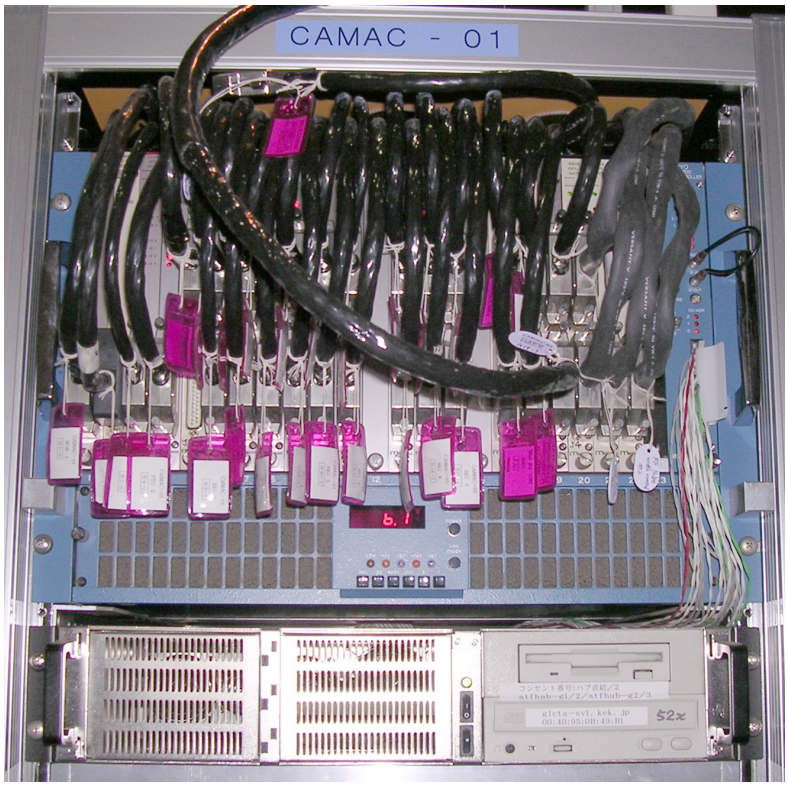

図 2.21: XTF#1,2 コントロール用 CAMAC クレート。SIG,ADC 等のモジュールが並んでいる。下部は 60°C 仕様の glcta-sv1 PC。

2.3.7 RF Processing

RF Processing ソフトウェアは RF コンポーネントの Processing を自動で行うためのソフトウェアである。主な機能としては、停止状態から設定した RF パワーまでゆっくりと (設定した速度で) パワーを上昇させていき、放電等によりインターロックがかかった場合、一定時間待機し、真空系が十分改善したことを確認した後、設定した比率でパワーを落として起動し、同様に設定パワーまで上昇させるというものである。

RF パワーのコントロールは sv1_control を通して RF 可変減衰器に接続された UP/DOWN コントロールを CAMAC で制御する方法で行っている (2.2.3 節を参照)。

2.3.8 コントロールインターフェース

XTF コントロールは、Dual Display に図 2.22(左画面), 図 2.23(右画面) を表示している。

左画面 (図 2.22) 最上部には、モジュレータ・RF 系の稼働状況・現在の入力パワー、パルス幅、繰り返し周波数などが表示されている。これらは sv1_monitor により CAMAC 経由で各種モジュール・測定器から読み出されたものである。

画面上部にはグラフが 2 つある。上のグラフは入力 RF パワーの推移を、下のグラフは真空度の推移を示している。途中パワーがなくなり、真空が悪くなっているところは、加速管・導波管等の放電箇所である。

下部は Processing の設定および上部グラフ表示の設定を行うインターフェースである。Processing 設定では UP/DOWN モジュールの目標値を左中央の Target フィールドに入力すると、その Target に向け UP/DOWN の値を徐々に上げていく。

右画面 (図 2.23) は主にインターロックの表示エリアである。上部は左画面と同様にモジュレータ・RF 系の稼働状況が表示されている。ここにある ON/OFF ボタンは Processing と独立に動作するので Processing の動作中にこのボタンを操作すると Processing が正しく行われない可能性がある。

中部はモジュレータ・クライストロン・導波管などの真空・温度等の値が表示されている。下部はインターロック表示で、インターロックが作動すると該当部分が赤く点滅する。

これらは JAVA で書かれており、JAVA 環境があれば KEK 内の計算機で

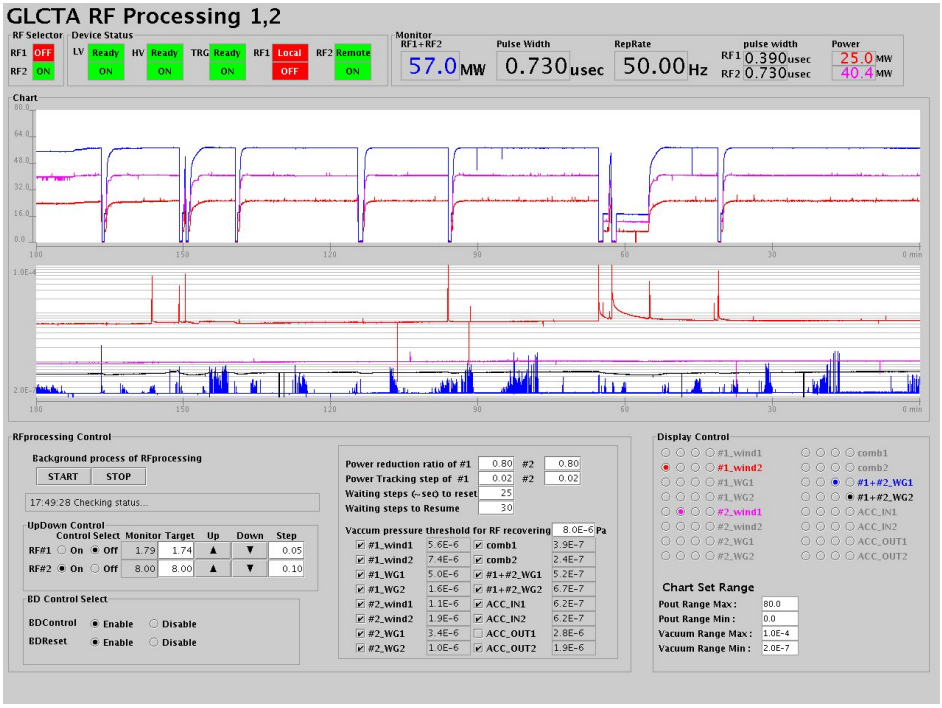

図 2.22: XTF コントロールインターフェース左画面。

あればどこからでも起動可能となっている。また、実際の XTF Control,RF Processing とは別のプロセスで起動しているため、インターフェースが強制終了するなどしても基本的にコントロールプロセスには影響を与えない。

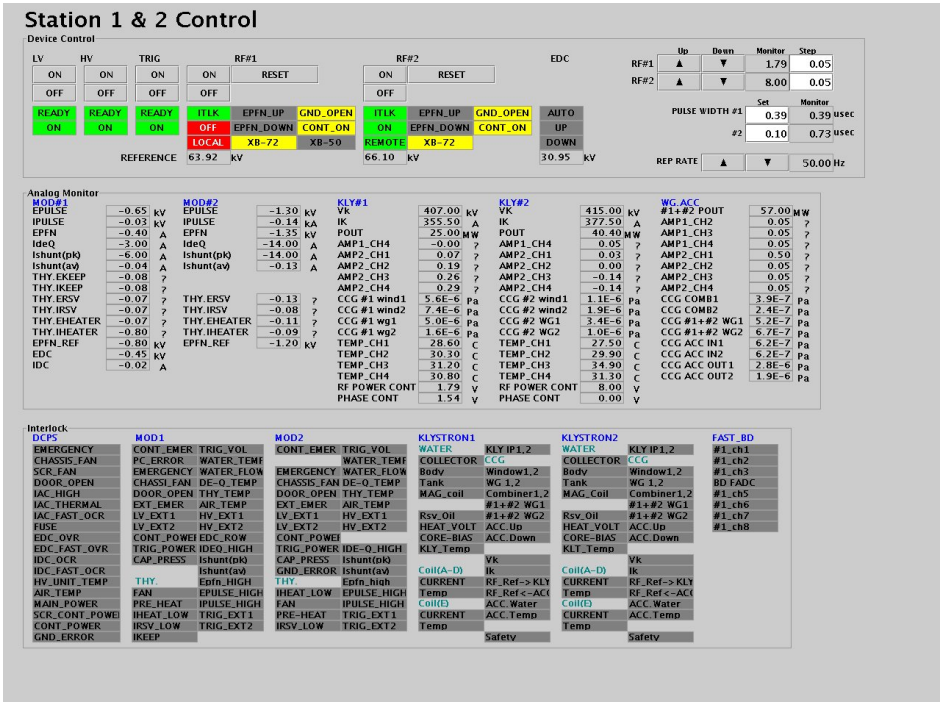

図 2.23: XTF コントロールインターフェース右画面。実際には左右画面で一つのウインドウである。

第3章　放電検出・記録システム (XTF-BDMS)

3.1　概要

高電場下で加速管の放電をいかに抑えるか、実用に耐えうる放電頻度を保ったまま電場強度をどこまであげられるかは、加速管設計・製作上最重要のテーマである。放電の影響として、以下の 3 点があげられる。

加速実効長の低下　加速管の放電は、加速管内の電磁場が突発的に乱れ、加速が正常に行われなくなるとともに、急激な真空の悪化を伴う現象である。これは、放電に伴い加速管の管内表面の物質がたたき出されたものと考えられ、真空が悪化した状態で RF を投入すればさらなる放電を引き起こす確率が極めて高いため、一定時間加速管を停止させる必要がある。

この停止時間の間は粒子が加速されず、いわばデッドタイムが生じる。このデッドタイムの運転時間に対する割合は、

$$R_{stop} = \frac{\tau_r}{\tau_r + \tau_b} \sim \frac{\tau_r}{\tau_b} \tag{3.1}$$

(τ_b:放電までの平均パルス数,τ_r:放電からパワー復旧までのパルス数)

で表される。実際には X-band LC は 8 本の加速管を 1 ユニットとして RF を供給するため、実際の停止時間は上式の 8 倍となる。この割合だけ予備の加速管を用意する必要がある。X-band LC において、2%のスペア加速管を用意すれば、放電後の復帰時間を ~ 10 秒として平均 2×10^6 パルスに 1 回の放電によるビーム最終エネルギーの不足に耐えられるという試算がある。[16]

放電痕等による加速管へのダメージ　加速管の放電は、先に述べたような粒子の放出を伴うため、加速管側へもダメージが予想される。実際に、

放電に伴う放電痕が運転後の加速管内に確認されることも多い。

これらの放電によるダメージが重なると、加速管内部が平滑でなくなるため、局所的に電場が強い部分が生じやすくなり、さらに頻繁な放電を誘発すると考えられる。また、加速管内部の形状が変化し、各加速セルの固有周波数変化による加速管の位相進みのずれが生じ、実効加速電場が減少する影響もある。

Linear Colliderのような高電界での長期運転を想定した加速管では、放電による劣化の加速管寿命・加速管パラメータへの影響を開発段階で正確に測定し・考慮する必要がある。

ビーム性質の悪化　放電に伴い生じる乱れた電磁場、またその後の真空悪化により加速ビームが散乱されることが予想される。これはビーム強度・エミッタンス等に悪影響をおよぼす可能性がある。

残留粒子との散乱にともなうビームエミッタンスの悪化については、加速管全長にわたって 10^{-4}[Torr] 程度以下であれば影響がないことが示されている [18]。しかし、電磁場の乱れによるもの、また突発的・局所的な真空悪化については、さらなる検討が必要である。

加速管の放電現象を観測し、加速管製作の改善につなげるためには、以上のような影響をふまえた上で、実際に観測・解析の機器・方法を構築する必要がある。

3.1.1　放電現象の観測

放電現象は直接的には加速管内のRF伝播の乱れと考えられるが、加速管内に測定器を簡単に設置できないため、加速管内の電磁場を直接観測することは難しい。そのため、放電現象を検知し、その特性を評価するには、放電に伴う加速管の外への影響をとらえる間接測定が必要である。

放電に伴う現象で外部から観測可能なものとしては、主に以下の現象が知られている。

- 加速管入力・出力部RFパルスの乱れ

  加速管内部のRFを測定するのは難しいが、加速管前後の導波管を通るRFならば容易に測定することが可能である。このRFのパルス内での時間変化から放電を検知するのが、加速管の放電検出としてもっとも一般的な方法である。

非放電時は、矩形の入力 RF パルスに対して、加速管内の RF 伝播速度 (KX01 では光速の約 3%) による遅延と加速管による減衰 (KX01 では約 35%が透過) を経てほぼ同型のパルスが出力カプラーから取り出される。また、ほぼ矩形の反射パルスが入力側へ戻るが、反射強度は一般に小さい (KX01 では 1%程度)。ただし、パルスの先頭部分、終端部分はやや反射が増大する。

それに対して放電時は、RF が出力カプラーまで正常に伝播されなくなるため、透過パルスの波形がある時点から弱くなる (パルス欠け) 現象が観測される。また、反射波形も放電時点以後大幅に増大する。この増大量はそれぞれの放電で異なるが、非放電時は安定した透過・反射波形が観測できるため、その差分を観測することで、容易に放電を検出できる。

- 加速管から放出される放射線

  放電時には、加速管の外へ大量の放射線が放射される現象が起きる。

  この現象は、放電に伴い加速管内にプラズマ状の粒子が発生し、それが加速管内の高電場により加速され、加速管内壁に衝突して制動輻射、コンプトン散乱等により生成された γ 線等とともに加速管外へ放出されるものと考えられる。

  加速管からは、非放電時も放射線が放射されている。これは電界放出 (Field Emission:FE) 電子が放電時と同様に高電場により加速、加速管内壁に衝突するためと考えられるが、放電時には非放電時と桁が違う量の放射線が放出されるため、その区別は容易である。

  また、放出される粒子の位置分布、エネルギー等を詳しく調べ、そのもととなった粒子の発生位置、エネルギー等を推定することにより、放電の様々な特性も評価できると考えられる。放電のメカニズムを知る手がかりとしても有力である。

- 放電に伴う加速管の熱振動

  放電時には、入力 RF パワーの一部が放電時に発生したプラズマ粒子に持ち去られる等して、散逸する (Missing Energy)。持ち去られたエネルギーの多くはプラズマ粒子の加速管の表面への衝突等により熱衝撃に変わる。この熱振動は (超) 音波となって加速管を伝わる。エネルギーの散逸と加速管表面への衝突は、熱衝撃の伝播より短い

タイムスケールで進行するため、加速管外壁に貼り付けた音響センサーで、この熱振動を測定することで熱衝撃の測定が可能である。

この測定では、RF の伝播と比べ音速によって決まる熱振動の伝播は極めて遅いため、放電位置を精密に観測することが可能と推測できる。

XTF では、以上の RF,γ 線, 音響の 3 種類に対応する測定器をそれぞれ設置し、オンライン・オフラインによる解析を行う放電検出・記録システムを製作・設置した。次節以降ではこの放電検出器および検出・記録システムの内容を詳細に述べることにする。

3.1.2 検出システムの要件

加速管放電のデータ収集には、通常の高エネルギー実験等と比べ、かなり異なった動作が測定システムに要求される。XTF の放電検知・記録システムは以下のような要件を満たすように設計した。

- 固定周波数によるデータ収集

  加速器のシステムは、一般に決まった繰り返し周波数 (rep.rate) により運転する。放電現象を確実に捉えるためには、データ収集、オンライン解析をこの 1 サイクル以内に終了させる必要がある。XTF は主に 50Hz で運転されており、100Hz までの運転が可能な仕様となっている。このためデータ収集・解析時間は最低 20ms, できれば 10ms 以内に抑える必要がある。

- ソフトウェアによるパルス毎の放電検出

  放電検出にはハードウェア (ロジック) とソフトウェアによる方法があるが、XTF では放電検出基準が複雑で、しかも開発段階や対称に応じて変更する必要があるため、ソフトウェアによる方法の方が適していると判断した。この場合、前項で述べたように時間的制限が厳しい。

- リングバッファを用いたデータ収集

  加速管の放電は非放電パルスと比べて非常に稀な現象であり、通常パルスはパルス毎の変化がほとんどない。このため、収集されるデータはほとんど同様の大量の非放電パルスと少量の放電パルスと

なる。また、音響センサ等では大量の波形を収集する必要があり、１パルスあたりのデータ量はかなり多い (現在収集しているデータは ∼70KB/pulse,50Hz で 3.5MB/sec となる)。従って、全パルスについて収集した全データを記録するのはディスク容量も膨大となり、保存時間も長くなり現実的ではない。

一方、このシステムの目的の一つとして、放電の予兆を調べることで放電を未然に回避できないか調べるというものがあり、放電パルスのみではなく、先攻するパルスも記録することが望ましい。このため、今回はリングバッファを用いて、放電前後の設定したパルス数を保存することにした。放電時には一度運転を停止するため、この停止中にデータをディスクに転送することができる。

また、全パルスに関する情報は、波形そのものではなく、その特性値を保存することにした。

- 使いやすい設定インターフェース

  長時間運転のため、システム制作者 (筆者) 以外も容易に操作が行え、しかも頻繁に設定変更できるよう、GUI による設定を可能とした。GUI 部は BDMS 本体とは完全に独立しており、ネットワークを通じて設定をやりとりする。

3.1.3 XTF-BDMS

図 3.1 は、XTF-BDMS のブロック図である。

以下、3 種の検出器について 3.2 節, システムについてはデバイス部を 3.3 節, ソフトウェア部を 3.4 節,3.5 節で解説する。

3.2 放電検出器

この節では、3 種類の放電検出器について述べる。

3.2.1 RF 検出器

RF 検出器は、導波管を通過する RF のパワーおよび位相を測定する。

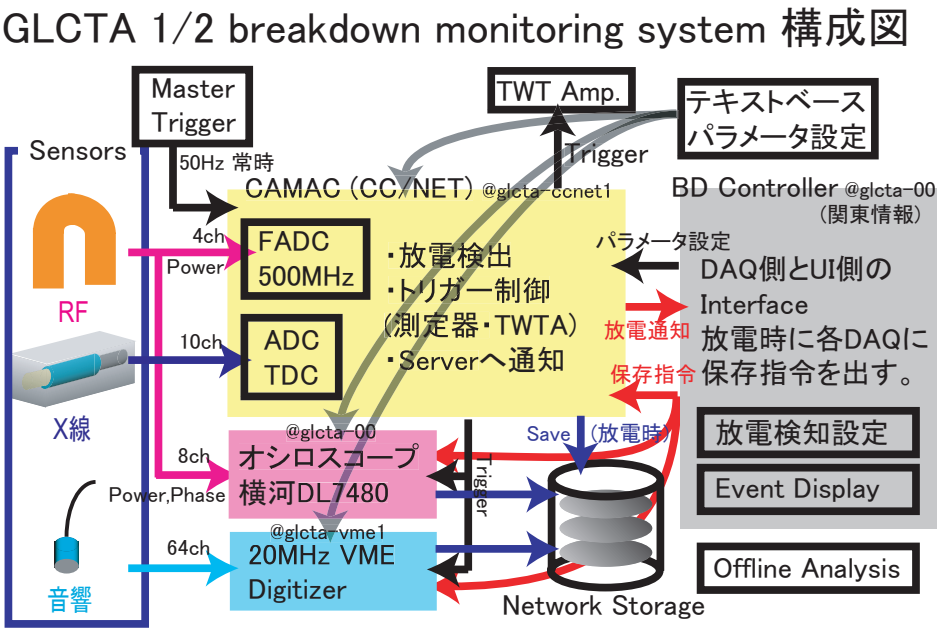

図 3.1: XTF-BDMS ブロック図

図3.2,3.3は、導波管を通過するRFを取り出す方向性結合器(DC:Directional coupler)と呼ばれる導波管パーツである。XTF-BDMSではこの2つのタイプのDCを使用している。図3.2の説明に記したように、DCは特定の方向へ進むRFを選択的に取り出すことができる。

DCの特性は、取り出したい方向の減衰(図の①→③,②→④)、取り出さない方向の減衰(図の①→④,②→③)で測定する。取り出さない方向の減衰は当然大きい方が望ましい。取り出す方向の減衰をCoupling(C)とよび、取り出さない方向の減衰-取り出す方向の減衰をDirectivity(D)と呼ぶ。原理的なC,Dの計算も可能だが、DCの性能評価も考えNetwork Analyzer(NA)を用いてC,Dの測定を行った。セットアップおよび測定結果を図3.5, 図3.6に示す。

Directional couplerは導波管の各所(図3.7)に設置されているが、BDMSで波形を記録しているのは、加速管入力RF①, 加速管出力RF②, 加速管反射RF[対称成分]③, 加速管反射RF[反対称成分]④の4つである。

DCから取り出されたRFはRFケーブルを用いてシールド外へ引き出される。3.2.1このRFケーブルによる減衰もSGとPower Meterを両端に設置し、SGの信号をPower Meterで測定し計測した。

RFケーブルで送られたRFは(特定チャンネルのみ)2分岐され、片側

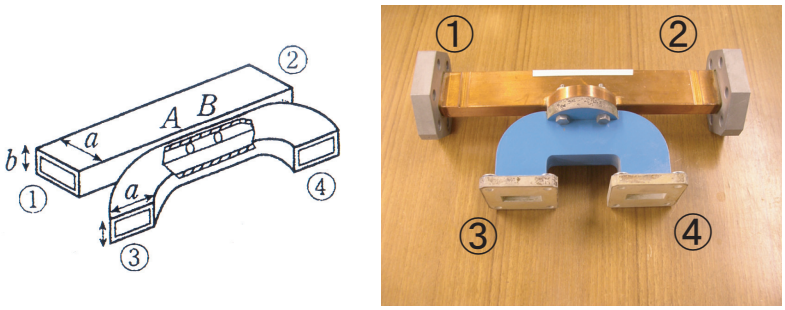

図 3.2: 2孔方向性結合器:模式図 [17] および写真。たとえば①から②へ RF が通過すると、小孔 A,B から導波管③④に RF が誘起される。A と B の間隔は $\lambda_g/4$(λ_g は管内波長で、11.424GHz に対し WR90(TE10 モード) では ~32[mm]) となっており、④へ進む RF は 2 孔で位相が揃うのに対し、③へ進む RF は 2 孔で位相が反転し うち消される。よって④からのみ RF が取り出される。同様に②から①へ進む RF は③とのみ結合する。

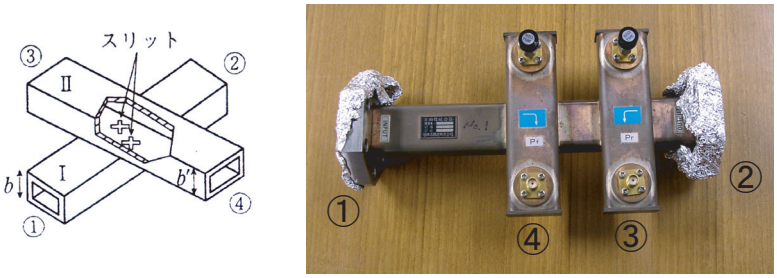

図 3.3: 十字型方向性結合器:模式図および写真。

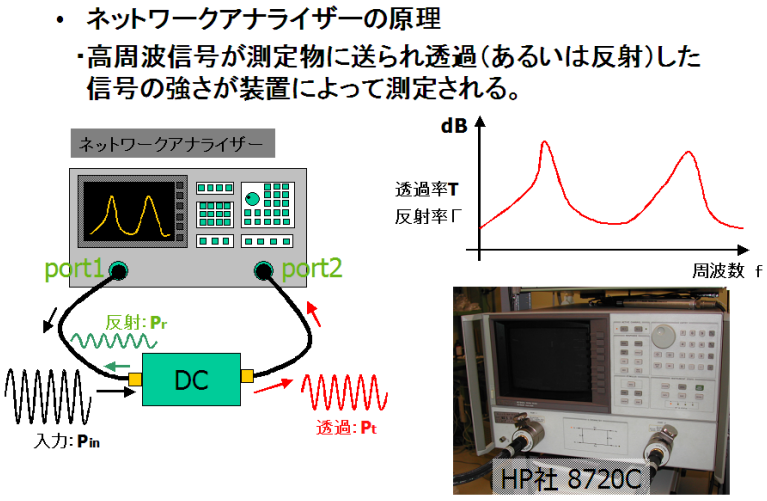

図 3.4: Network Analyzer およびその動作原理。

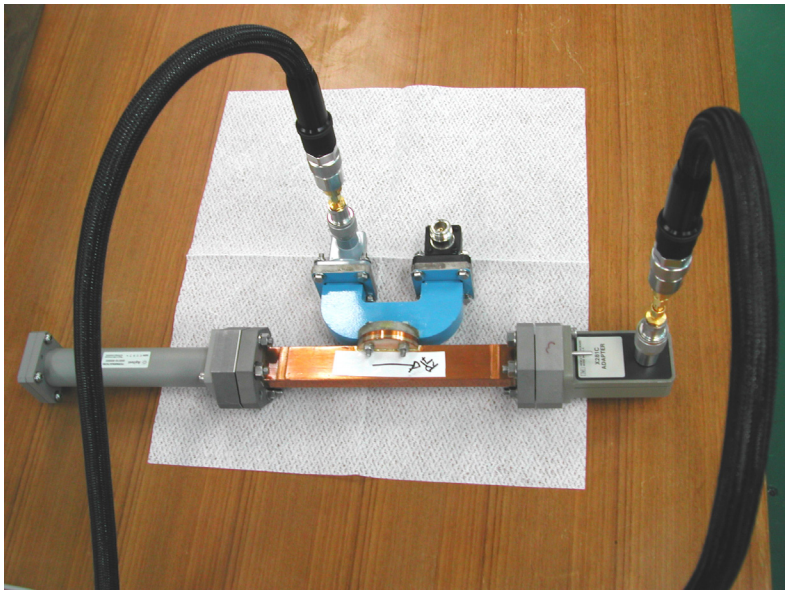

図 3.5: DC の C,D 測定。写真手前が NA で、2 ポート接続し、片側から RF を入力してもう片側への透過率を調べる。

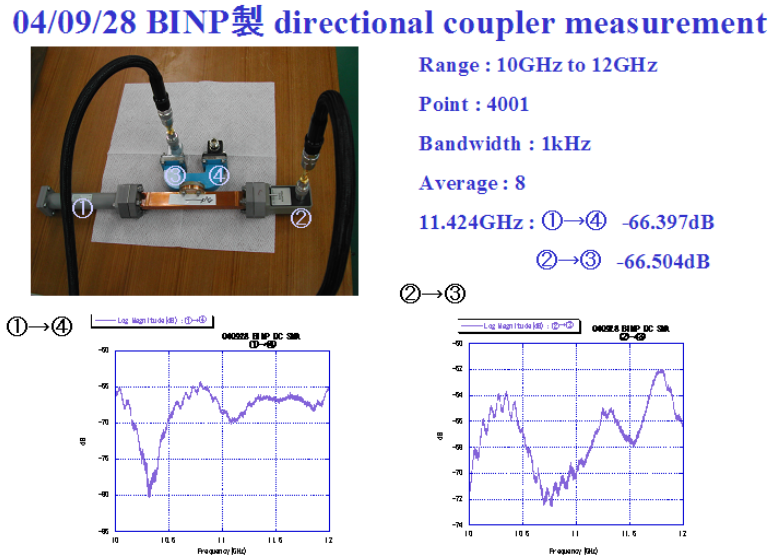

図 3.6: DC の C 測定結果例 (周波数特性:10〜12GHz)。XTF-BDMS で用いる DC はすべてこの測定を行い、11.424GHz 周辺で大きな周波数依存性がないことを確認し、11.424GHz での値を DC の C として計算に使用している。

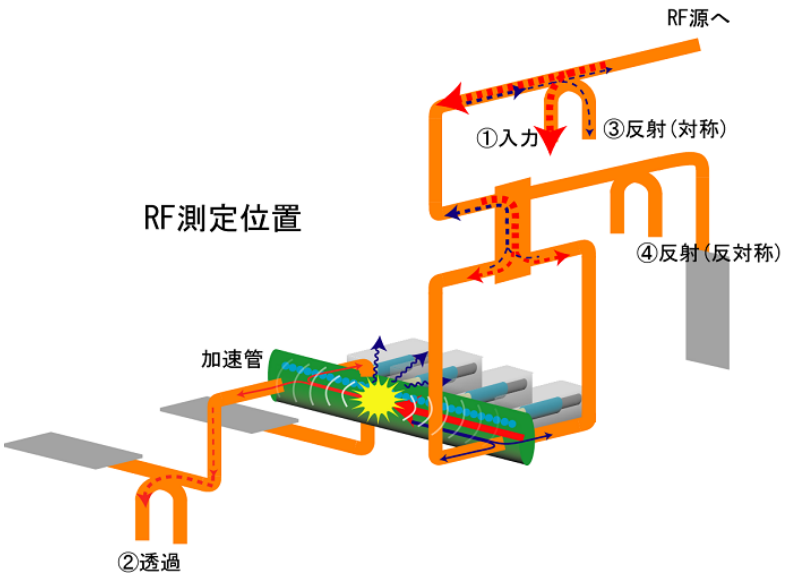

図 3.7: DC 測定位置

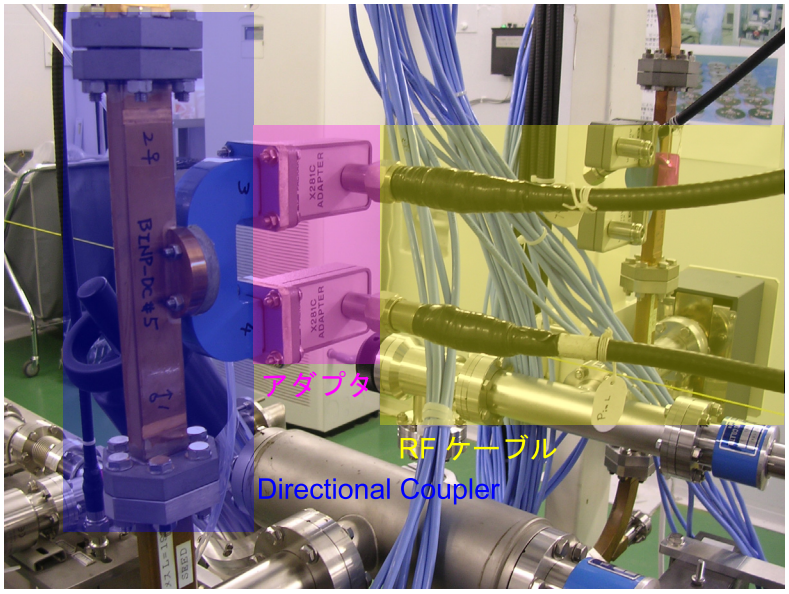

図 3.8: DC からアダプタを介して RF ケーブルへ出力

は Diode で DC 成分を、もう片側は Mixer を用いて位相成分を取り出す。Diode は Agilent 8473B(図 3.2.1, 表 3.2.1 参照) という製品で、RF エネルギーに対応した電圧を出力する。出力は線形ではなく、様々な RF を入力しその応答を調べる必要がある。これについても SG を用いて出力電圧をオシロスコープで観測し、Calibration を行った。(図 3.11)

BDMS で使用している 4 チャンネルについて、DC,Cable の減衰比、Diode の変換定数は表 3.2 に示した。Diode で直流に変換後に BDMS の Flash A/D Converter(FADC) に入力するまでの信号増幅・減衰比も併せて示した。

Mixer は基準 RF と Combine することで位相成分を取り出すもので、必要に応じて記録する。今回の解析には用いていないので、ここでは述べないことにする。

それぞれの箇所で測定された波形例を図 3.12 に示す。放電時の反射の増大、透過のパルス欠けがはっきりとわかる。

図 3.9: 8473B diode

動作周波数	0.01〜18 GHz
周波数安定性	Octave で ± 0.2dB
感度	$> 0.5mV/\mu W$
ノイズ	$< 50\mu W$
最大入力 RF	200mW
極性	負

表 3.1: 8473B diode 主要パラメータ

Ch.	DC C	RF Att.	LPF	Total Att.	Diode #	Diode c_2	Diode c_1	Elec.Att.
P_f	-63.7dB	-27.8dB	○	-91.5dB	0123	1.090×10^{-3}	4.154×10^{-2}	+4.0dB
P_{TL}	-69.4dB	-23.9dB	○	-93.3dB	0120	1.294×10^{-3}	5.070×10^{-2}	+24.0dB
P_{rs}	-62.5dB	-28.3dB	○	-90.8dB	0122	1.236×10^{-3}	5.559×10^{-2}	+10.0dB
P_{ra}	-67.0dB	-26.8dB	×	-93.8dB	0140	1.059×10^{-3}	4.446×10^{-2}	+20.0dB

表 3.2: RF パワー係数表 (2004/05/11)。RF Att. は Cable, 減衰器,Divider の合計 (合計で測定した)。Diode c_2,c_1 は Diode の変換係数 ($W_{in} = c_2 \times V_{out}^2 + c_1 \times V_{out}$,$W_{in}$:入力電力 [mW],$V_{out}$:出力電圧 [mV])。また、直流化後の減衰/増幅 (Elec.Att.) は 20dB で電圧 10 倍となる。P_{rs} の RF Att. には、測定後に追加された Divider(-4dB:カタログ値) を含む。

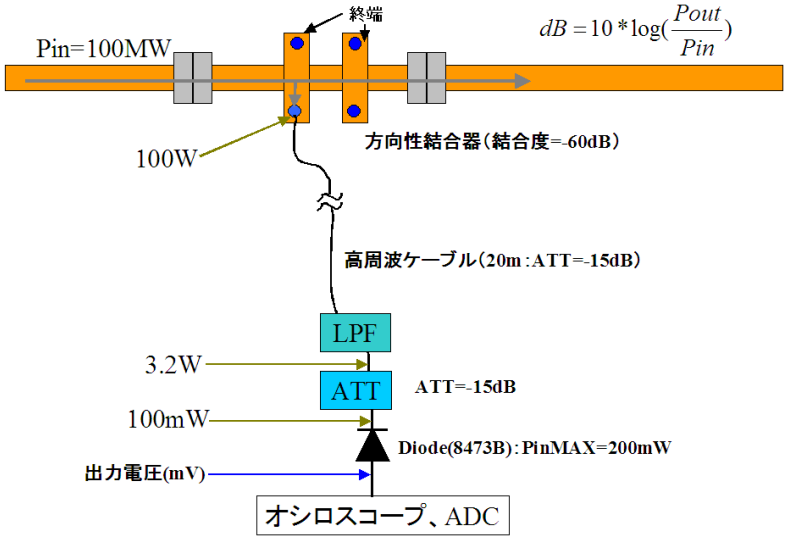

図 3.10: RF パワーレベルの推移。加速管に入出力する高電力は直接測定できないので、減衰比の大きな DC, 同軸ケーブル,RF 減衰器で減衰し、微少電力にして測定する。

3.2.2 γ 線検出器

γ 線検出器は、放電時に加速管から発生する粒子 (主に γ 線) を、プラスチックシンチレータと光電子増倍管 (PMT) を用いて検出する。

図 3.13 に、検出器の構造を示す。

プラスチックシンチレータは、ST406 というタイプを用いている。大きさは、模式図に示したように、直径 20mm, 高さ 20mm の円筒形である。

PMT は、GDB15(Hamamatsu R647 相当) を用いている。仕様は表 3.3 の通りである。PMT のソケットは Hamamatsu E849-68 を用いており、SHV および BNC コネクタにより高圧電源、測定系と接続されている。

PMT とシンチレータは optical gleese で密着されており、全体を黒ビニールテープで巻いて固定している。ただしシンチレータの前面のみ巻かれていない。PMT には主にノイズ対策としてアルミ箔が巻かれている。

コリメータは 70mm × 60mm × 200mm の直方体の鉛ブロックを円筒形にくりぬいたような形になっており、PMT とシンチレータがすっぽり入る形になっている。シンチレータの前面は、コリメータ前面から 30mm の深さに固定されている。コリメータの前面はアルミ箔およびビニールテープにより遮光されている。

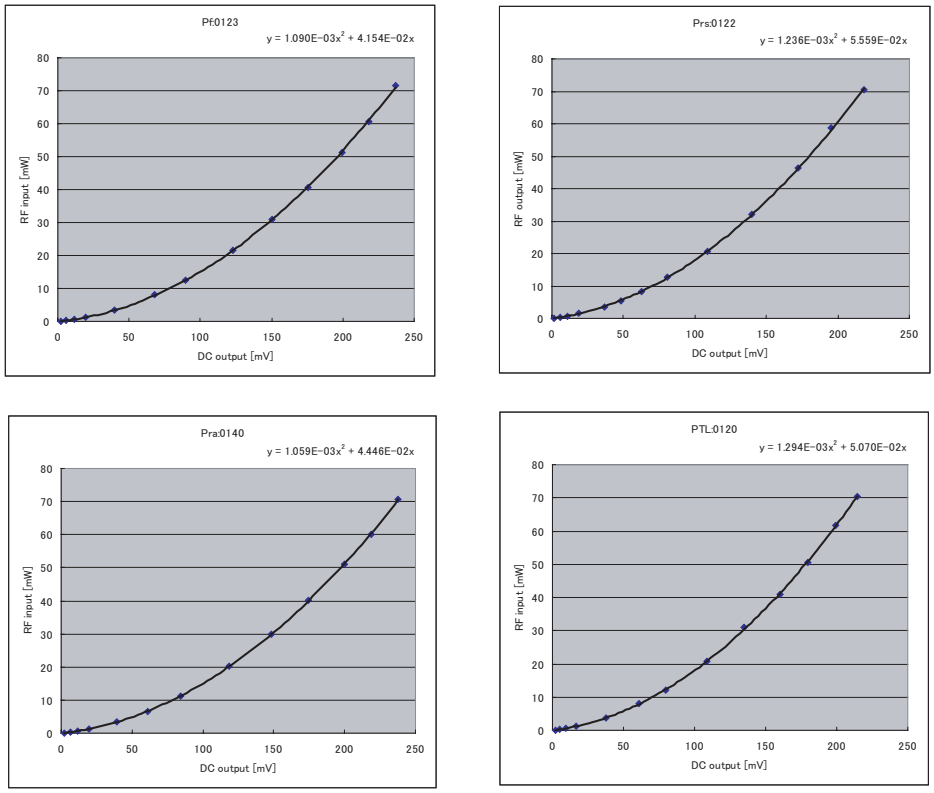

図 3.11: Diode Calibration 結果。BDMS で特に監視している 4 ポートに使用されている diode のデータを示す。2 次曲線の Fit を利用する。

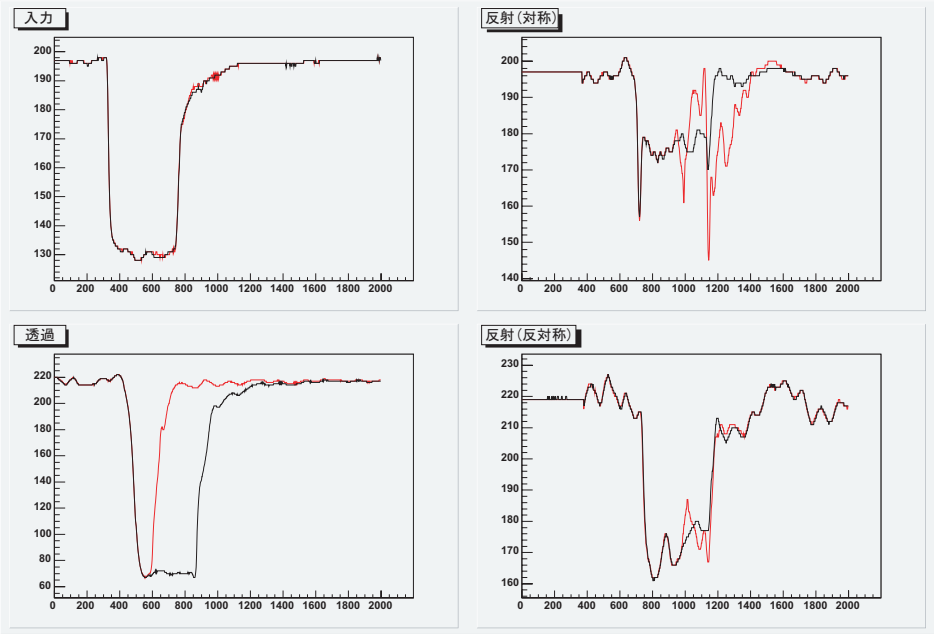

図 3.12: 典型的な加速管放電時と非放電時の波形。加速管透過 RF の欠けと反射 RF の増大が認められる。

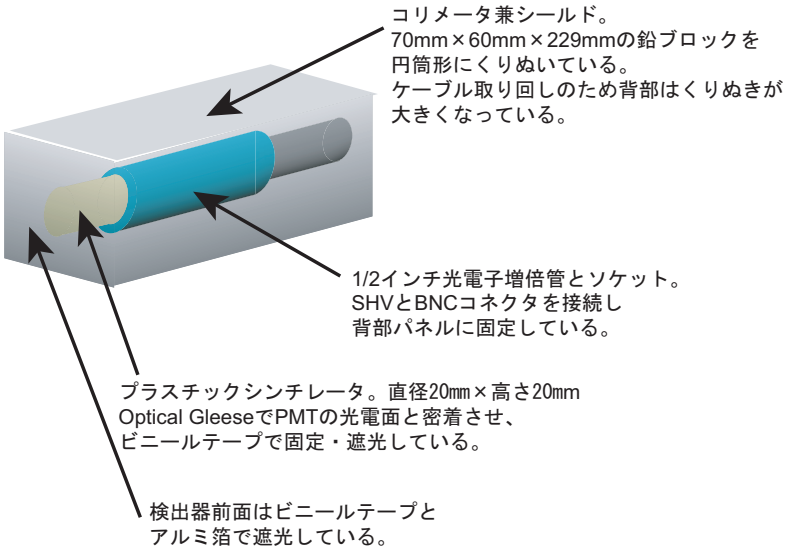

図 3.13: γ 線検出器模式図。

図 3.14: γ 線検出器内部写真。

型番	GDB15
寸法 [mm]	ϕ14.5, 高さ 73
Cathode 物質	KCsSb
Cathode 有効直径 [mm]	10
波長応答 [nm]	300〜650
最高感度波長 [nm]	400
Cathode 感度 (Luminous)[μA/lm]	30 min., 60 typ.
Cathode 感度 (Blue)[μA/lm]	7 min., 12 typ.
Anode 感度 [A/lm]	30(800V),150(1000V)
Anode 電圧 [V]	800〜1000 typ., 1100 max.
暗電流 [nA]	2 typ., 15 max
Rise time[ns]	2.3

表 3.3: GDB15 光電子増倍管仕様。

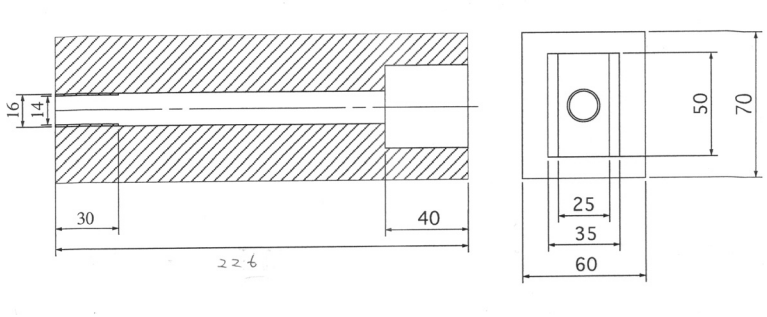

図 3.15: γ 線検出器コリメータ図面。

チャンネル数	32
設定可能電圧	0〜-3KV(-1〜-20V を除く)
出力電流	2mA MAX
電圧精度	$\pm$ 3V
リップル	200mVp-p
温度係数	0.3V/℃ (10 ℃〜40 ℃)

表 3.4: RPH-32010 主要パラメータ

この γ 線検出器ブロックは、中国の IHEP(高能研究所) と KEK で加速管の放電検出用に 1995〜97 年に共同開発されたものに若干の改良を施し使っているものである。施した改良は、(1) 高圧電源のコネクタを MHV から SHV に換装,(2) コリメータ前面の遮光を追加 (ビニールテープ) である。

PMT 用高圧電源としては、REPIC RPH-32010 を利用している。仕様は以下の通り。RPH-32010 はシールドルームの外、放電検出ラックに設置し、SHV ケーブルでシールドルーム内に引き込んでいる。

この γ 線検出器は、加速管の左右に 4 チャンネルずつと、構造の異なる coupler 周辺に input,output 各 1 チャンネル、計 10 チャンネル設置している。γ 線検出器設置用の架台は、今回新規に製作したものである。

Calibration　このプラスチックシンチレータおよび PMT の Calibration を、宇宙線および青色 LD(Laser Diode) 光源を用いて行った。

宇宙線を用いた Calibration のセットアップを図 3.19, 図 3.20 に示す。上部の 2 つのシンチレータの Coincidence をとり、トリガーとする。今回校

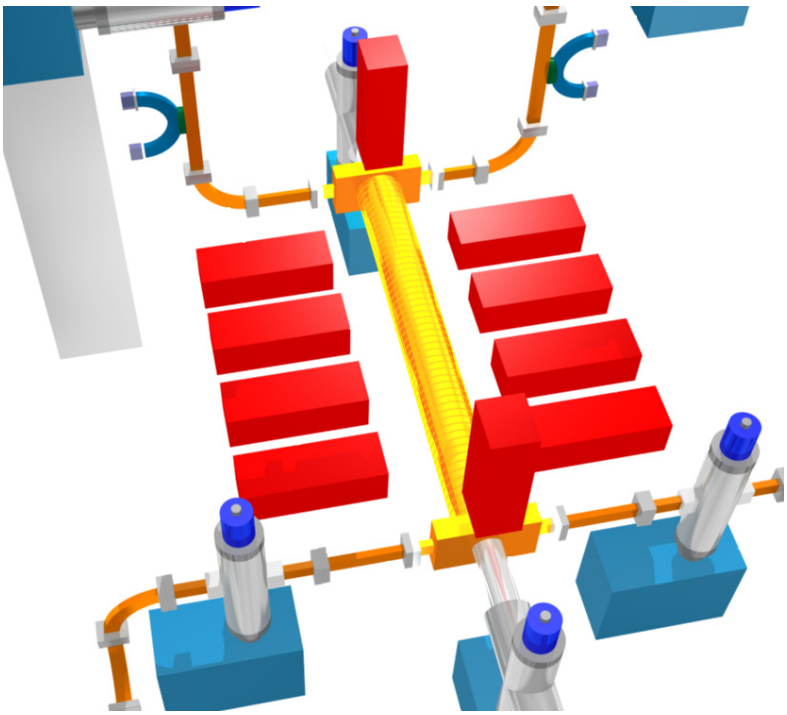

図 3.16: γ 線検出器設置図。赤く示したものが γ 線検出器。図奥が上流、手前が下流。加速管の左右に 4 つずつ、上流・下流の Coupler に各 1 つ設置されている。

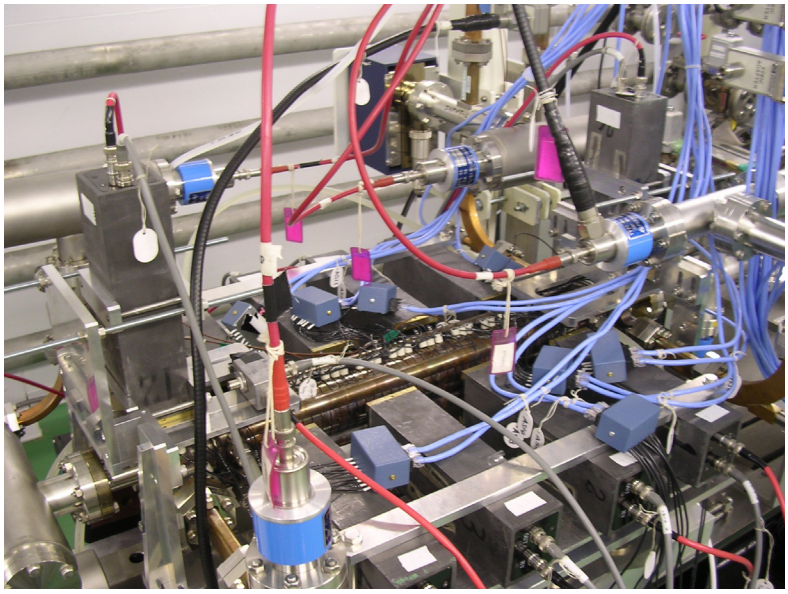

図 3.17: γ 線検出器設置写真。

正する 3 つのシンチレータを間に挟み、下部に板状のシンチレータを設置している。

測定は CAMAC の積分型 ADC(Repic RPC-022[1]) を用いて行った。以上 6 つのシンチレータの信号をそれぞれ ADC のチャンネルに入力し、CC7700 クレートコントローラを通じてデータを収集している。

収集ソフトウェアは XTF-BDMS の (CC7700 を使っていた以前のバージョンの) コードを簡素化・改良したものを用いた。CC7700 ドライバには、“cc7x00”[2]を使用している。

データ収集は約 280000 イベント行い、図 3.21 のヒストグラムを得た。トリガーレートは 2000count/hour 程度であった。上部二つのシンチレータのみをトリガとして用いた場合はピークが確認できないが、下部のシンチレータもトリガとした場合はピークが確認できた。

このピークは High energy muon の Energy deposit $\frac{dE}{dx} \sim 2[\mathrm{MeV/g{\cdot}cm^{-2}}]$ に対応すると考えられる。シンチレータの高さは 2.0cm なので、シンチレータの比重:$\sim 1.1[\mathrm{g/cm^{-3}}]$ として、ピークのエネルギーは $2 \times 1.1 \times 2.0 = 4.4[\mathrm{MeV}]$ となる。

[1]http://www.repic.co.jp/products/repic/camac/rpc-022/rpc_022.html
[2]ftp://iris.riken.go.jp/pub/cc7x00/

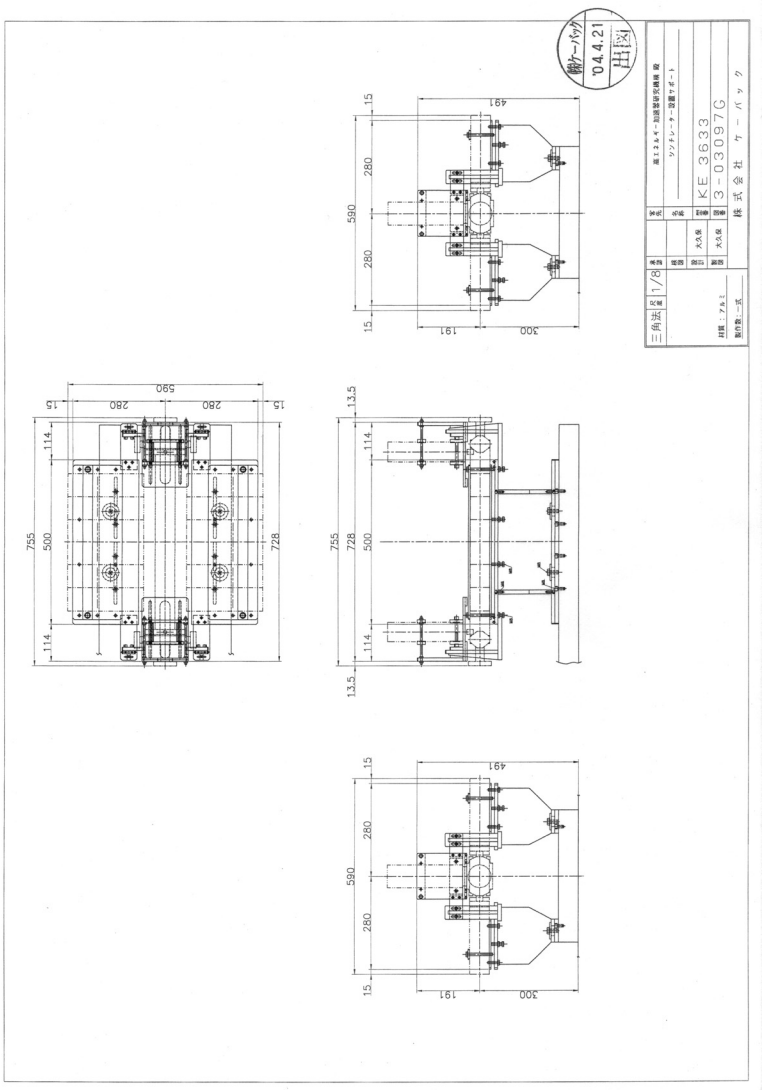

図 3.18: γ 線検出器架台図面。

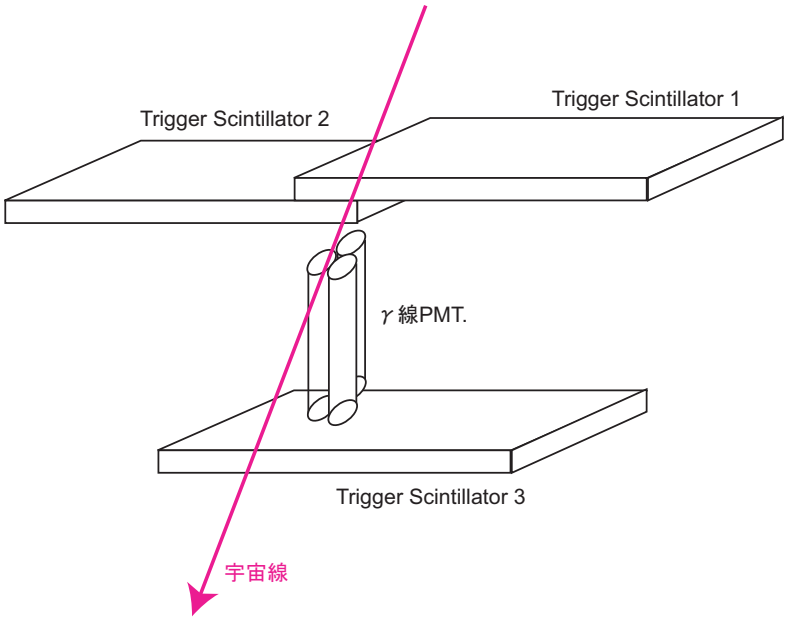

図 3.19: γ線 PMT. calibration 概念図。

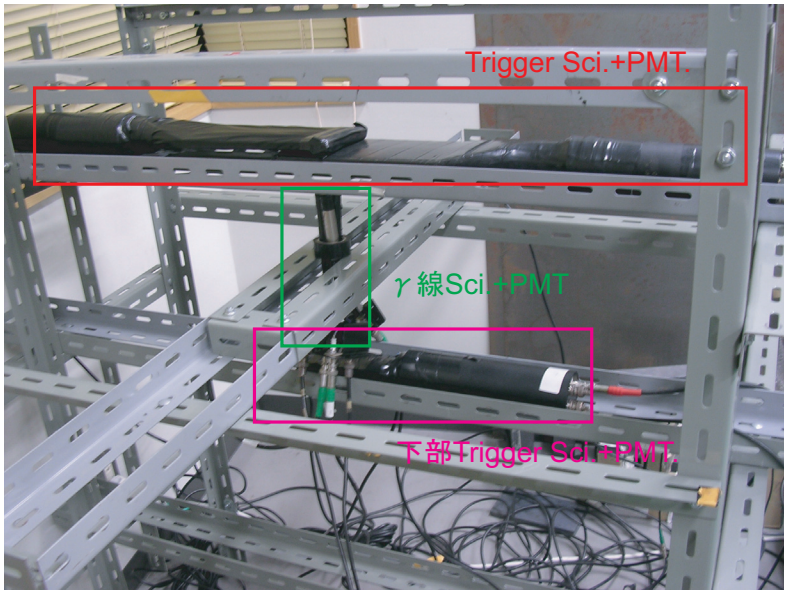

図 3.20: γ線 PMT. calibration

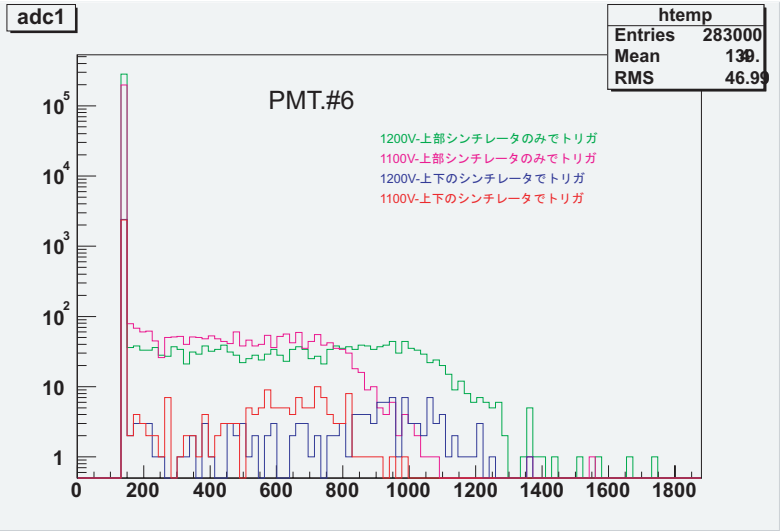

図 3.21: 宇宙線校正スペクトル。

この宇宙線による Calibration は 3 チャンネルについて行い、ピーク位置を比較した。結果を図 3.21 に示す。ADC の 1 カウントが 0.25pC,Baseline が 140 ± 2 カウントであったことを考慮し、各チャンネルの 1200V での Gain を求めた。

但し、イベント数が少ないため分解能が悪く、またチャンネル依存性が線源の Calibration 結果とも一致しないため、この解析では 3 チャンネルの Gain の平均 (41.1[pC/MeV]) を 3 チャンネルの平均ゲインとして次の線源 Calibration で用いることにする。

次に、青色 LD を使った Calibration を行った。

青色 LD は Nichia NDHV310APC[3]を用いた。波長は 408±8[nm] で、シンチレーション光に近い。パルス電源を接続し、5[ns]〜100[ns] まで任意の幅、強さの電圧入力を可能とした。セットアップは、LD に紙による適当な遮蔽を行った PMT. を密着させる簡易なものであり、1 度に 1 チャンネルの校正を行った。チャンネルごとの光量の一定性が保証されないため、まず高い電圧で Single Photon の Gain を調べてチャンネル間の相対 Gain を得た後、光量を増加させ電圧依存性をチャンネルごとに調べる方針をとった。

光量の調整は LD に供給する電圧調整により行っている。Single Photon

[3]http://www.nichia.co.jp/specification/ld/NDHV310APC.pdf

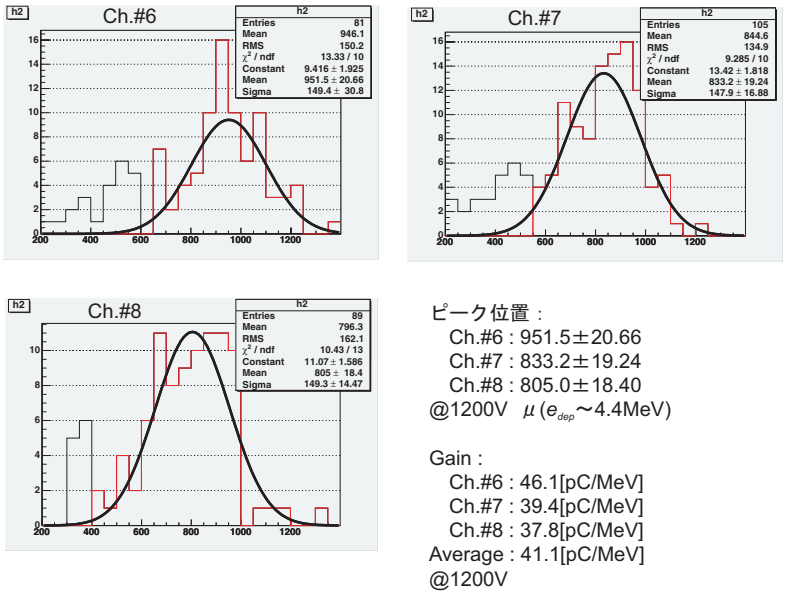

図 3.22: 宇宙線校正によるチャンネル毎の感度差。

の校正では、PMT の光電面に入射する光子の数が Gate(100ns) あたり 0 ～数個程度になるよう光量の調節を行った。

PMT. の電圧は 1200V(Gain の低い Ch.3 については 1250V) で行い、以下の関数でフィッティングを行った。

$$N(n) = N_0 \sum_j \left\{ P_{po}(j,\lambda) P_g(n, b+jg, i\sigma_0^2) \right\} \tag{3.2}$$

自由変数:λ,g,σ_0
n:ADC count
$N(n)$:count= n のイベント数
N_0:全イベント
$P_{po}(x,\lambda)$:平均 λ の Poisson 分布関数 (整数版)
$P_g(x,\mu,\sigma^2)$:平均 μ, 分散 σ^2 の Gauss 分布関数
b:ADC offset
g:Single photoelectron(p.e.) gain
σ_0^2:Single p.e. 分布の分散

図 3.23 に典型的なフィッティング結果を示す。フィッティング関数と得られたスペクトルがよく一致していることがわかる。

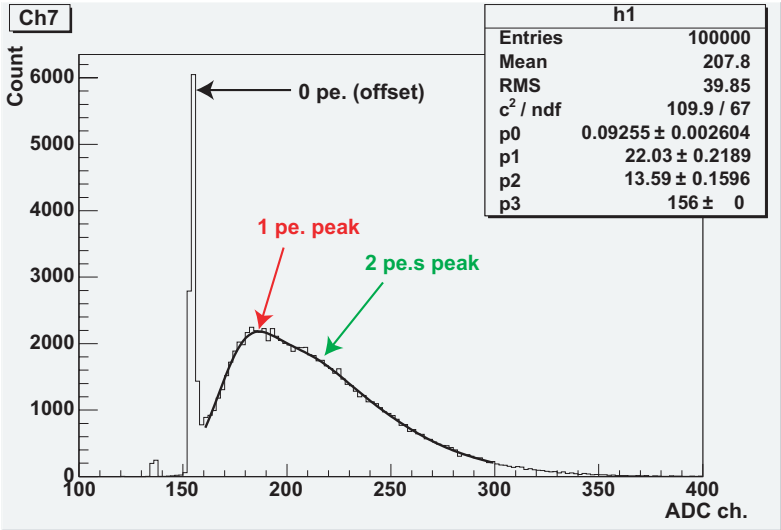

図 3.23: LD による Single photoelectron のスペクトル例。p1 が gain(g) を示す。1 pe. と 2 pe.s のピークが見えている。

電圧依存性は、チャンネルごとに電圧を変えながら、得られた ADC スペクトルの Gaussian Peak 位置をフィッティングで求め、Gain の比較を行う。ダイナミックレンジが広いため、光量を 3 段階に変えながら、400V ～1200V(Ch.3 は 1250V) での相対 Gain を測定した。

図 3.24 に、Single p.e. のチャンネルごとの Gain と各チャンネルの電圧依存性を合わせた Gain の電圧・チャンネル依存性を示す。縦軸は、宇宙線のデータも用いて Energy deposit あたりの電荷量に直してある。

加速管の放電時には、大量の γ 線が放出されるため、PMT の電圧を 1 粒子の観測に適した設定より大幅に下げる必要がある。XTF での測定では 550V(Run #6:2004/7/14～7/30 の時の値) 程度の電圧を印加している。

また、非放電時にも γ 線が放出され、これを観測するには PMT の電圧を 1 粒子の観測に適した程度まで上昇させる必要がある。これは、Field emission を観測する場合に随時行う。

図 3.25, 図 3.26 に通常時、放電時の PMT 出力波形および ADC データを示す。通常時はシグナルが pedestal 程度に収まっているのに対し、放電時では一定の強さのシグナルが放電時間に対応して継続的に出ているのがわかる。放電検出には非常に有用である。

また、前節で述べたように、γ 線放出の特性を詳しく調べることは、放電の特性を評価する重要なプローブとなりうる。この γ 線の特性につい

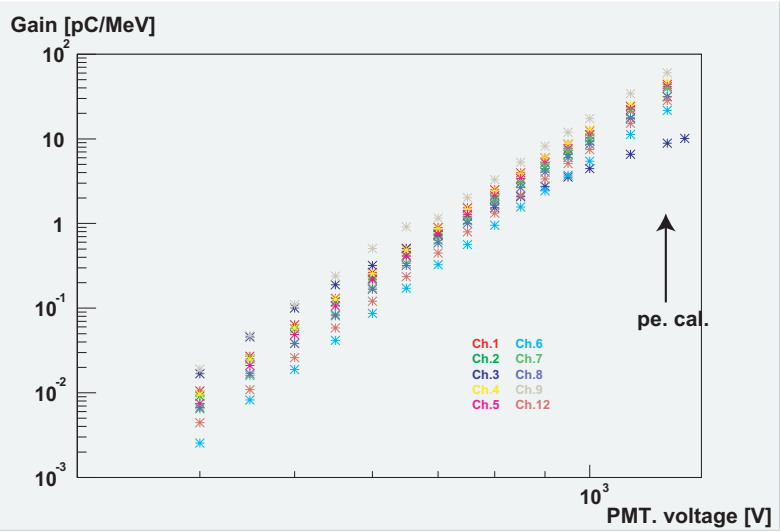

図 3.24: γ 線 PMT.Gain のチャンネルおよび電圧依存性。電圧依存性は log-log プロットでおおむね直線上にある。チャンネルごとの相対強度は 1200V のみであわせているため、低電圧側は信頼度が低い。Ch.3 は高電圧側での Gain が低く、PMT. に異常がある可能性が高い。

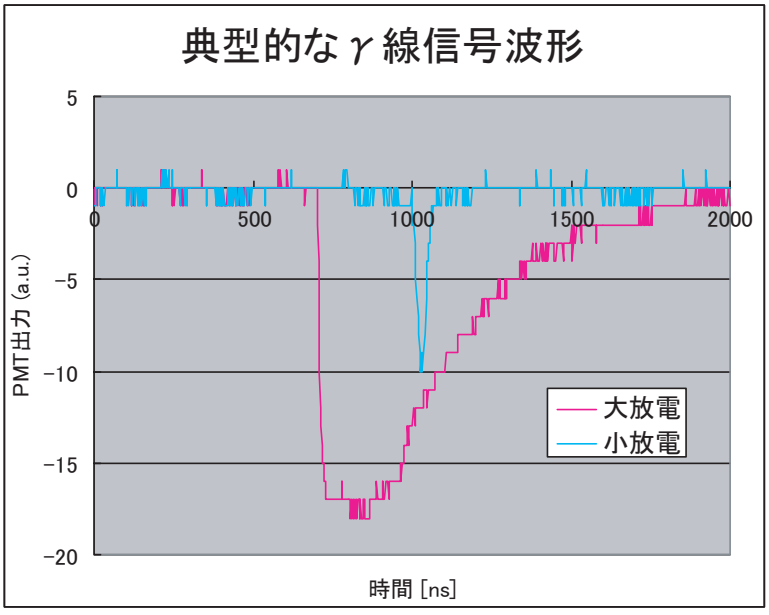

図 3.25: 典型的な放電の PMT 出力波形

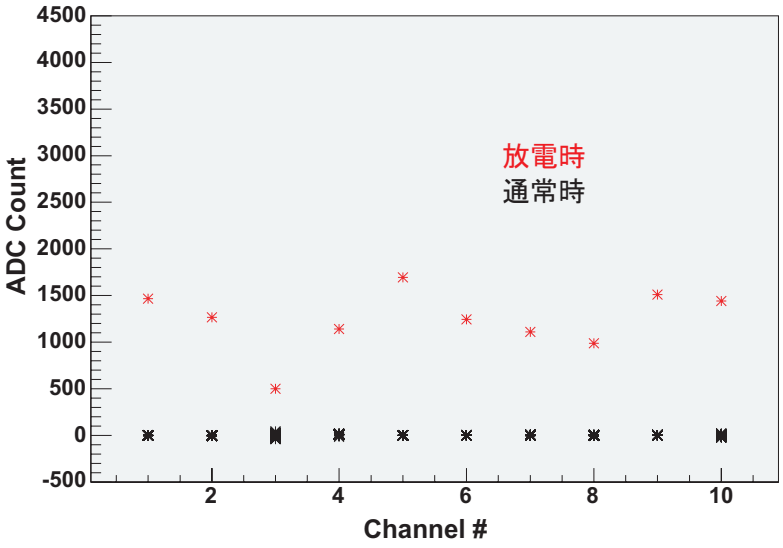

図 3.26: 典型的な放電の γ 線 ADC 分布

ては、第 5 章で詳細に評価する。

3.2.3 音響検出器

音響検出器は加速管に貼り付けた音響センサで放電時の加速管の熱衝撃を直接測定するものである。この検出法は SLAC にて最近開発されたもので、XTF でも SLAC からセンサーおよび測定モジュールを譲り受けて使用している。

加速管は純銅でできており、音速は P 波が約 4700[m/s],S 波が約 2300[m/s], 表面波が約 2100[m/s] である。今回用いる音響センサは、1MHz〜数百 KHz に感度があるため、その最短波長は

$$\frac{4700[\mathrm{m/s}]}{1\times10^{6}[\mathrm{Hz}]} \sim 4.7\times10^{-3}[\mathrm{m}] \tag{3.3}$$

より、5[mm] 程度となる。X-band 加速管の 1 セルはおよそ 10[mm] 幅のため、理論的にはセルレベルの分解能が期待できることになる。

音響センサは下図のようなもので、直径 10mm 程度のセラミックの圧電素子を用いている。主にコストダウンのためトランスでインピーダンスを下げた後 RJ45(ネットワークケーブルとして用いられるのと同じも

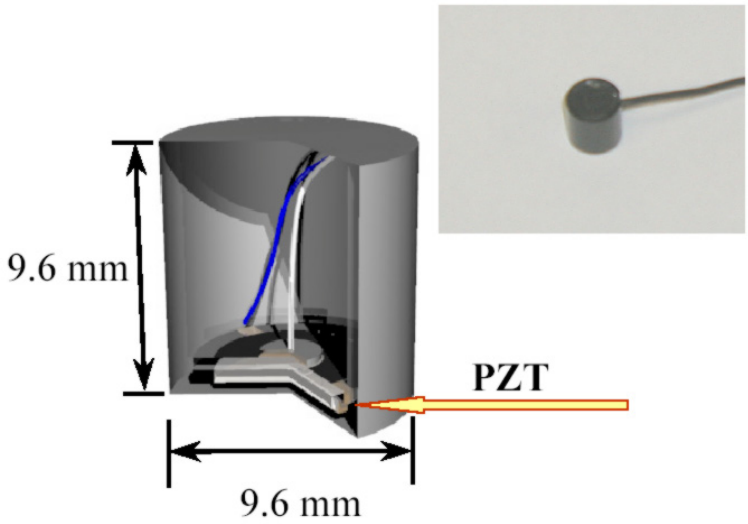

図 3.27: 音響検出器模式図

の) の UTP(Un-Twisted Pair:より対線ケーブル) で送信される。

このセンサーは加速管の各セルおよび coupler 周辺に全部で 64 チャンネル設置されている。設置は加速管表面にのり付けであり、チャンネル毎に同じ感度が保証できるセットアップにはなっていないが、波形から大体の感度を推定する。

典型的な非放電時、放電時のシグナル波形を示す。非放電時でもかなりの強度のシグナルが検出されており、放電時のシグナルとの分離が解析課題となる。

3.3 データ収集 (DAQ:Data Aquisition) 回路

本節では、XTF-BDMS の信号処理系 (NIM モジュール等) および CAMAC,VME 等のデータ収集デバイスの動作について述べる。

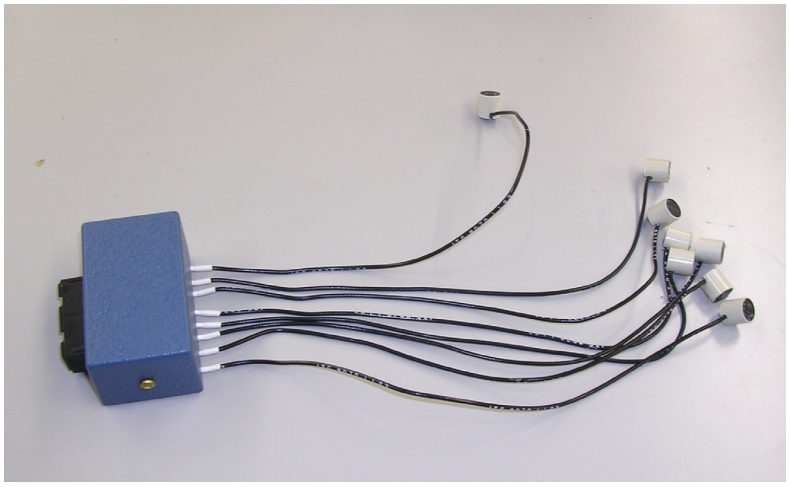

図 3.28: 音響検出器。ボックスの部分でインピーダンス変換と RJ45 への変換を行っている。

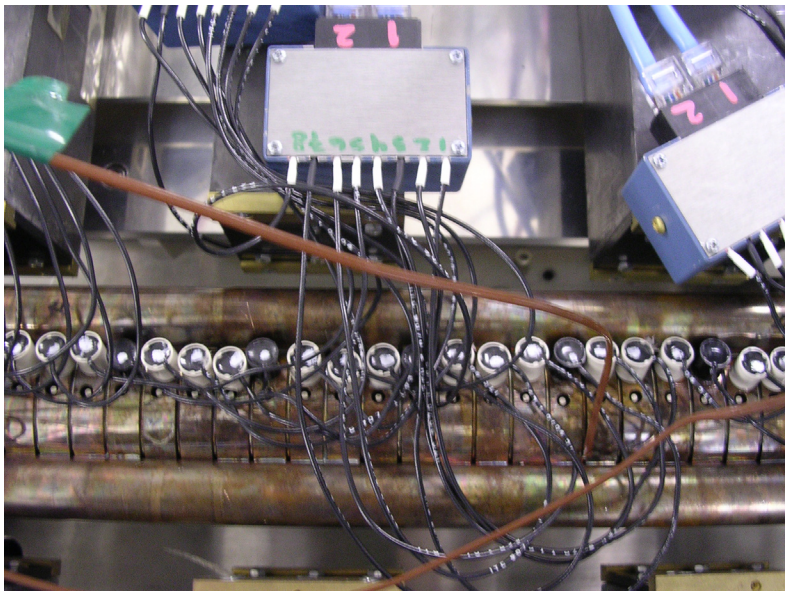

図 3.29: 音響検出器設置写真

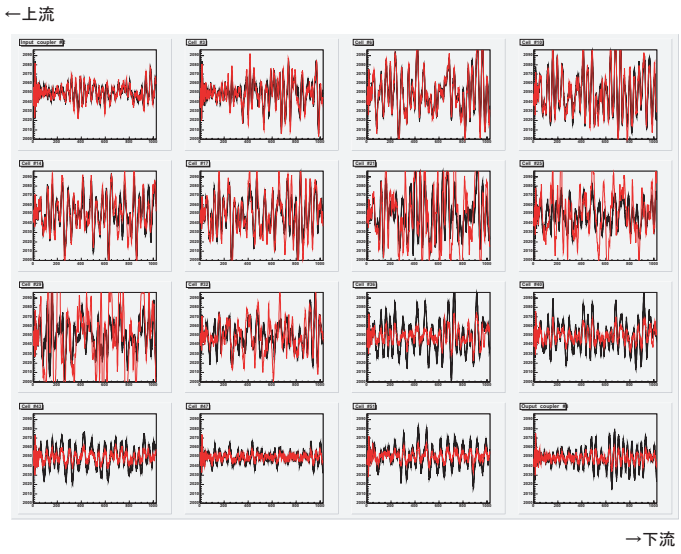

図 3.30: 典型的な放電時と非放電時の音響波形。赤が放電時・黒が非放電時の波形。

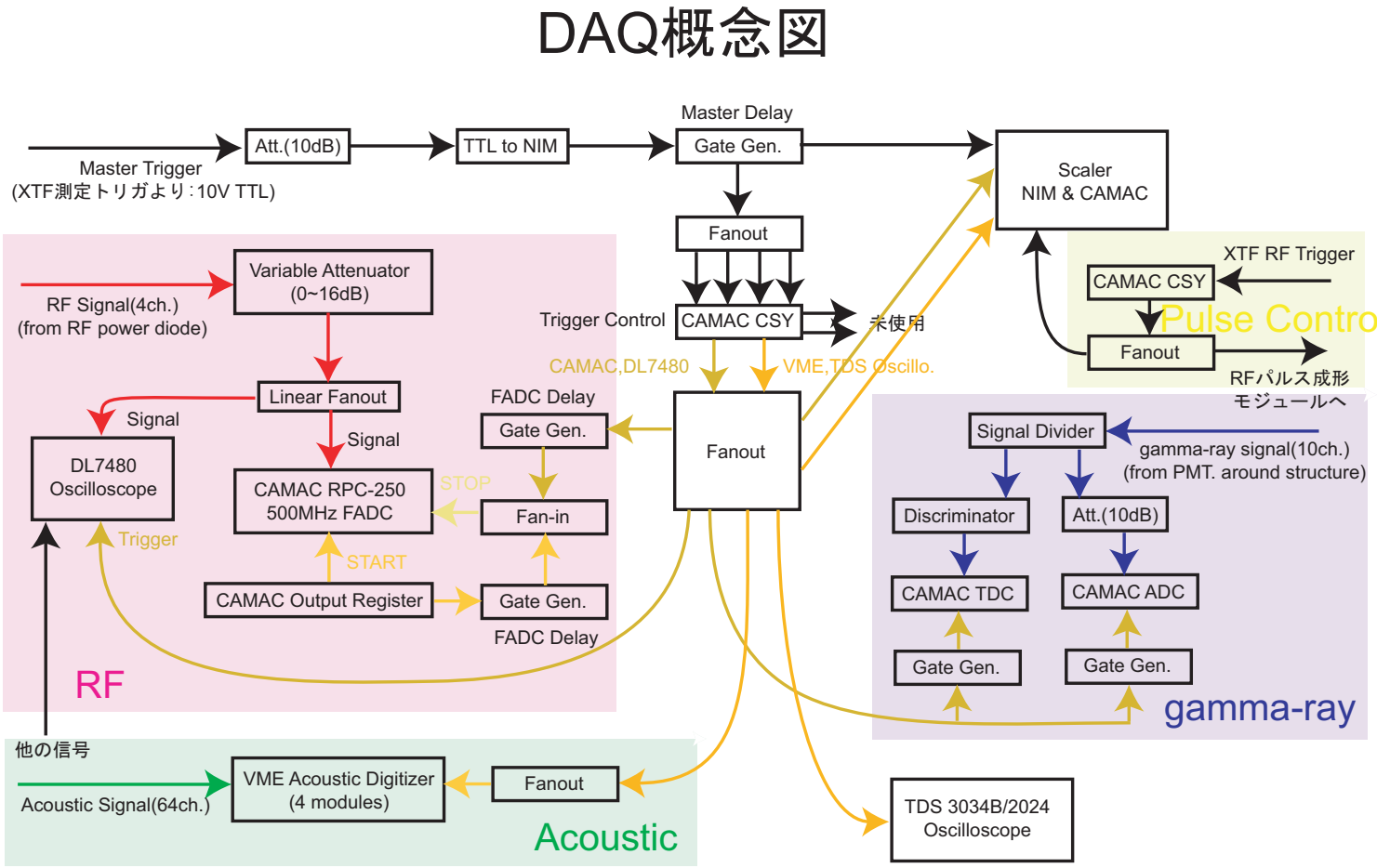

図 3.31: DAQ 回路概念図。

名称	RPC-250
動作周波数	500MHz
チャンネル数	2
メモリ	8KWord / ch
ADC 分解能	8bit
ADC 入力電圧	0〜2V
Amp. Gain	〜2 倍
Offset	-3.3V〜+0.8V
入力信号	Start,Stop,Signal(2ch),Ext.Clock
LAM(Look At Me)	Enable/Disable を内部のスイッチで切替

表 3.5: RPC-250 主要仕様 Amp. および Offset は ADC 入力前のアナログ処理であり、モジュール内の可変抵抗にて設定可能。

3.3.1 RF 波形記録

RF 検出器の信号は、CAMAC の 500MHz Flash ADC で波形を収集する。

使用しているのは、REPIC[4] RPC-250 × 2 台である。仕様を表 3.5 に示す。

図 3.34 に示されているように、RPC-250 は FIFO(First In First Out) バッファを 2 つ持っている。データ収集モード中は STOP 信号が入力されるまでこのバッファをリングメモリとして用いてデータを書き込み続ける。STOP 信号を入力すると、測定停止と共に FIFO カウンタが停止し、以後の CAMAC コマンドによる読み出しはこの FIFO カウンタを用いて行われる。

この FIFO バッファはチャンネルあたり 8192word ある。CAMAC バスの周波数は 1[MHz] であり、1 点の取得に 1 命令必要な FIFO バッファでは 8192 点の転送には理論上 8.192[msec] 以上かかる。RPC-250 は 4 チャンネル読む必要があるので、100Hz 運転に必要な 10[ms] 以内に全データを読みとるのは不可能である。

一方、XTF の RF パルス幅はクライストロン・モジュレータの制限により最大で 1[μs] 程度である。500MHz の FADC なので加速管からの反射など遅延がある信号を考慮して 2[μs] の波形を取得しても必要な点数はチャ

[4]林栄精機 http://www.repic.co.jp/

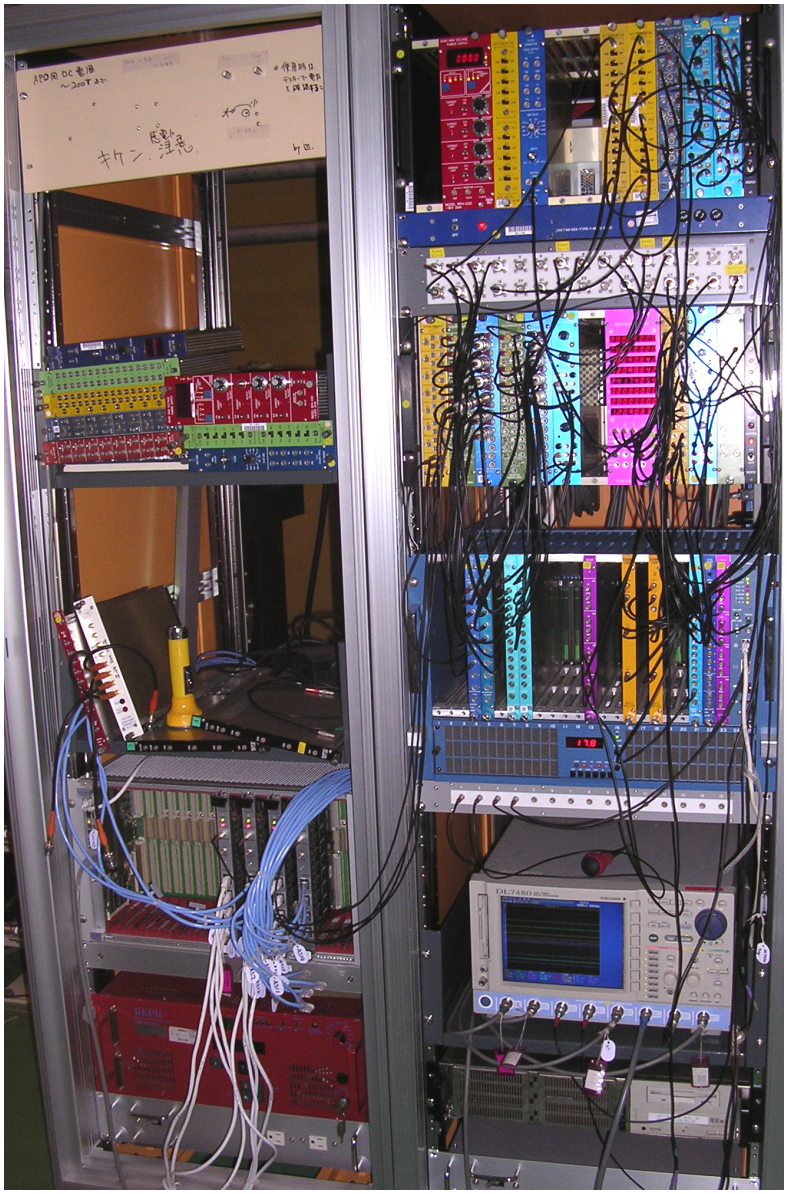

図 3.32: BDMS ラック写真。右のラックは上部から NIM2 段と CAMAC があり、主にトリガ調整回路と RF,γ 線検出器の信号処理を行っている。下段には DL7480 オシロスコープ、glcta-sv3 PC がある。左ラックの下部には VME クレートがあり、音響検出器のデータを収集している。最下段は γ 線検出器の PMT 用の高圧電源である。

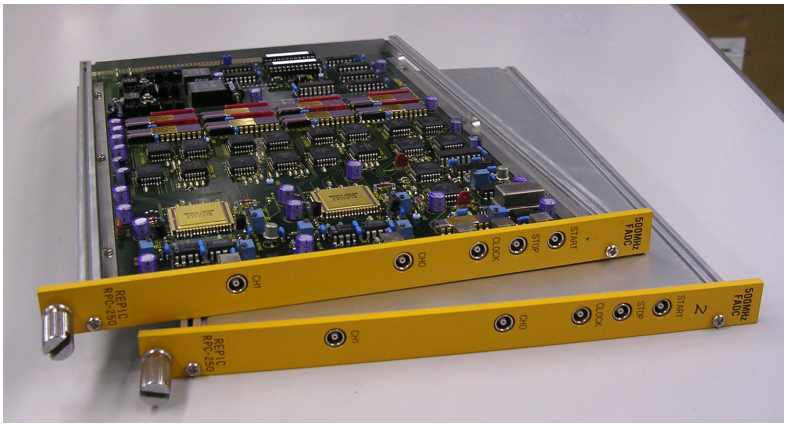

図 3.33: RPC-250 500MHz Flash ADC

コマンド	機能
F(0)A(0,1)	chA の FIFO から 1 つデータを読み出し、カウンタを一つ進める。
F(8)	LAM(Look At Me) チェック
F(9)	LAM をクリアし、データ収集を開始
F(10)	LAM をクリア
F(25)	データ収集開始

表 3.6: RPC-250 CAMAC コマンド一覧。実際には F(9) と F(25) の動作は同じと思われる。

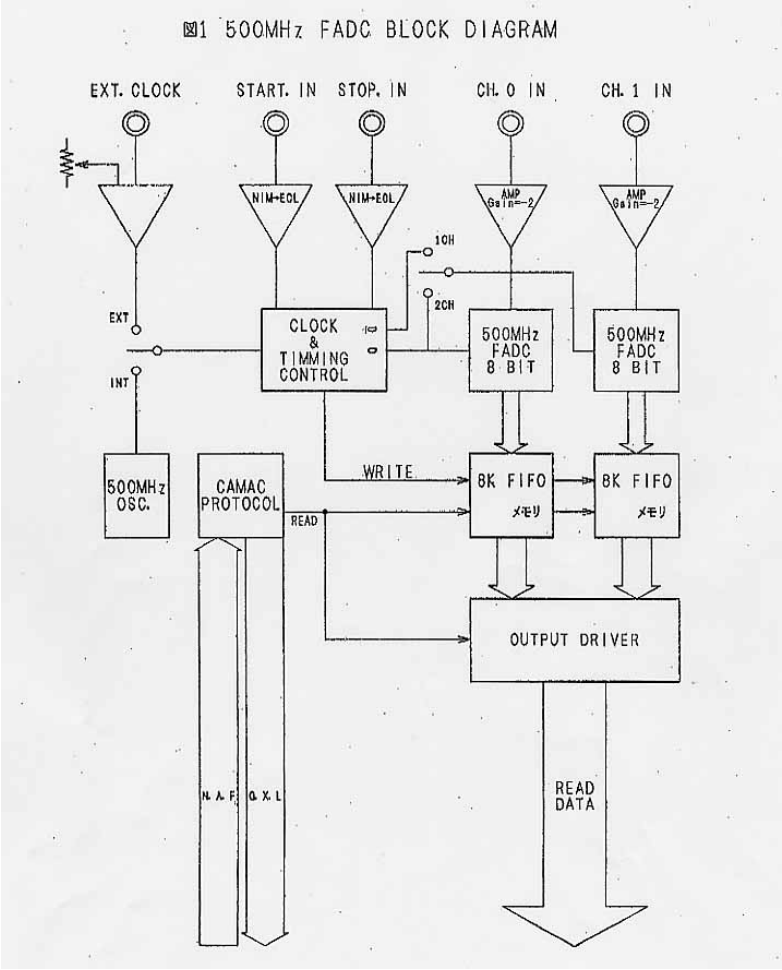

図 3.34: RPC-250 ブロック図

ンネルあたり 1000 点である。この場合、4 チャンネルでも理論上 4[ms] でデータ収集できることになり、トリガーレートにおいつける可能性がある。

ただし、RPC250 は 2 チャンネルに共通の FIFO カウンタを用いており、通常の方法でははじめのチャンネルで完全にデータを読み終わらないと次のチャンネルを読むことができない。また、FIFO カウンタをリセットする手段 (CAMAC コマンド等) も提供されていないため、当初 RPC-250 で 2 チャンネルを同時に使用するのは不可能と思われた。

しかし、様々な調査を行った結果、以下のようなことがわかり、目標通り 2 チャンネルとも予定した点数のみ読み込むことが可能となった。

- F(0) を入力すると、FIFO バッファ内の古いデータから順に読み出される。従って、最初に読み出されるデータは STOP 信号の $8192 \times 2[\mathrm{ns}] = 16.4[\mu\mathrm{s}]$ 前となる。
- F(0) で読むチャンネルを途中で変えると、FIFO カウンタはそのままで、読み出しチャンネルが変わる。

  たとえば、F(0)A(0) で ch0 を 1000 点読んだ後 F(0)A(1) をすると ch1 の 1001 番目のデータを取得する。
- F(9) は FIFO バッファをクリアしない。
- F(9) は測定を開始するのみで、FIFO カウンタは変化しない。測定データは現在の FIFO カウンタの位置から蓄積される。モジュールのスタート入力信号の動作も同様である。
- F(10) は LAM をクリアするのみで FIFO バッファ、カウンタには一切影響しない。

実際に、各チャンネルのデータを読み込む手順は次のようにした。

1. STOP 信号をモジュール前面の QLA コネクタより入力する。STOP 信号は NIM の GATE モジュールで RF 波形の立ち上がりから $15.5[\mu\mathrm{s}]$ 程度遅らせておく。
2. 1 チャンネル目のデータを取得 ($1000[\mathrm{ch}] = 2[\mu\mathrm{s}]$ 分) する。

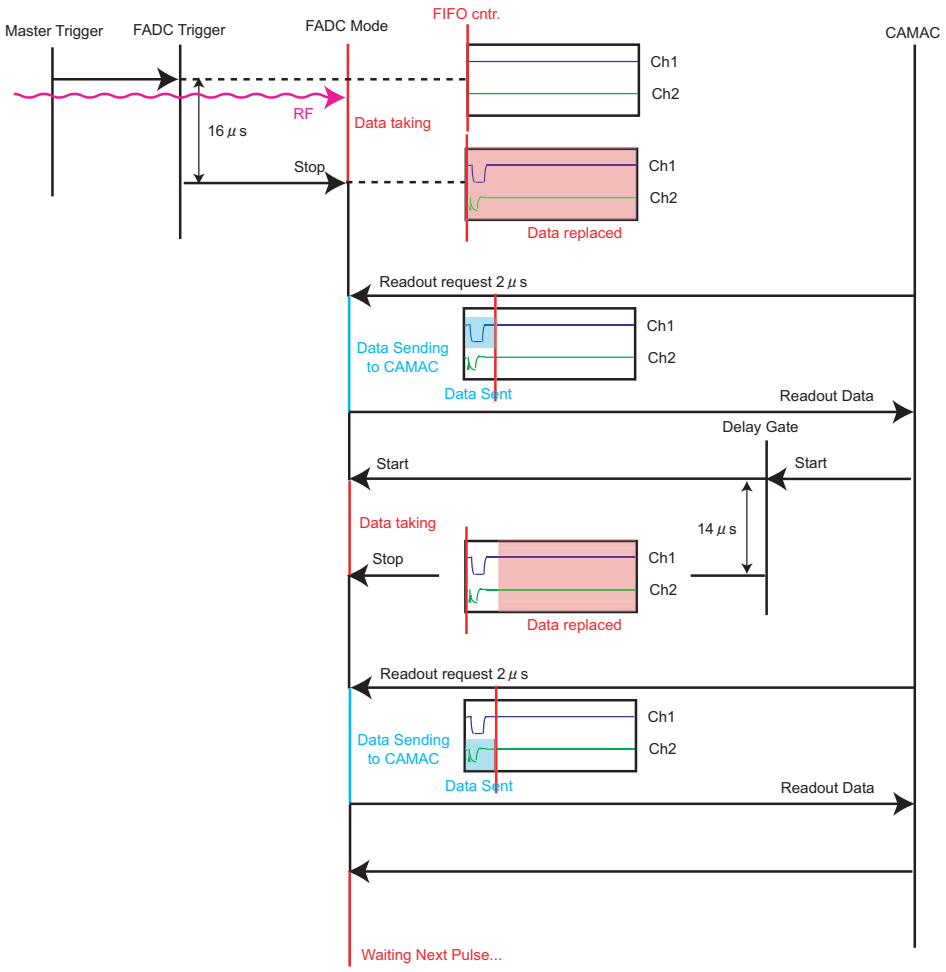

図 3.35: FADC 読み出し時の信号タイミング図

3. 読み出しが終了したら CAMAC のアウトプットレジスタよりパルスを出す。このパルスは FANOUT モジュールで分岐され、片方は FADC の STOP 信号へ、もう片方は GATE で 14.4[μs] 程度遅らせたあと FADC の STOP 信号へ (初めの STOP 信号と LOGIC FANIN で混ぜた後) 入力される。これで、FIFO カウンタが 14.4[μs] すすみ元の位置にもどる。

4. 2 チャンネル目のデータを取得する。

FADC の START,STOP 信号の供給図を図 3.35 に示す。
この方法だと、GATE モジュールのジッタが信号に影響してしまう。

名称	RPC-021
チャンネル数	12
フルスケール	-1000[pC]
ダイナミックレンジ	10[bit]
分解能	-0.25[pC]
Conversion Time	52[μs] max.
Linearity	$\pm$2[count]
温度依存性	$\pm$1[count/°C]

表 3.7: 12ch CS(Charge-Sensing) ADC RPC-021 主な仕様

GATE の DELAY 値がかなり大きいため、ジッタもそれなりに大きい。特に 2 チャンネル目のデータは 2 回の DELAY を挟むので影響が大きくなってしまう。実際には、現在測定しているデータは 10[ns] 程度のジッタを持っており、オフライン解析の際にはこれをキャンセルする機能が必要となる。

また、ケーブルを用いて片側のチャンネルの信号を遅らせることで 2 チャンネル読めるようにする方法もあるが、今回は少なくとも 2[μs] の Delay が必要であり、ケーブル長が約 400m という長大なものとなってしまい、また取得点数を変える際の調整が面倒なため採用しなかった。

3.3.2 γ 線時間・電荷記録

γ 線検出器の信号は、CAMAC の積分型 ADC,TDC(Time to Digital Converter) で立ち上がり時間と電荷量を測定する。CAMAC の ADC は REPIC RPC-021 を 2 モジュール、TDC は Kaizu KC3781A を 2 モジュール用いている。検出器は 10 チャンネル設置されているが、ADC,TDC は最大 16 チャンネルデータ収集可能である。

モジュールの仕様を表 3.7,3.9、CAMAC 命令一覧を表 3.8,3.10 に示す。

検出器からの信号は、まず Signal Divider で ADC 信号と TDC 信号に分けられる。ADC 信号は BNC Attenuator(SUHNER+HUBER 6910.01.A[5], 10dB) で減衰させた後、ADC の Signal IN に入力する。TDC 信号は Discriminator(KEK N0625, 閾値:100mV) で GATE を生成し、TDC の Stop

[5]http://www.hubersuhner.com/hs-p-rf-comp-atten.pdf

コマンド	機能
F(0)A(0〜11)	Data Read
F(2)A(0〜11)	Data Read & Clear(A(11)で全 ch Clear)
F(8)	LAM Test
F(9)	Data Clear
F(10)	LAM Clear
F(24)	LAM Disable
F(25)	LAM Enable

表 3.8: RPC-021 CAMAC コマンド一覧

名称	KC3781A
チャンネル数	12
フルスケール	100,200,500[ns](スイッチ)
ダイナミックレンジ	12[bit]
分解能	25,50,125[ps](フルスケールに応じて)
Conversion Time	100[μs]

表 3.9: KC3781A 主な仕様

コマンド	機能
F(0)A(0〜7)	Read Data
F(2)A(0〜7)	Read & Data Clear(A(7)で全 ch Clear)
F(8)	LAM Test
F(9)	Data Clear
F(10)	LAM Clear
F(24)	LAM Disable
F(25)	Test Module
F(26)	LAM Enable

表 3.10: KC3781A CAMAC コマンド一覧

図 3.36: γ 線検出用 ADC,TDC

IN に入力している。

ADC Gate および TDC Start は Master Trigger の信号を Gate で調整して入力している。

ADC の Gate は RF 入力の立ち上がりに立ち上がりがほぼあわせてあり、Gate 幅は 1[μs] となっている。(ADC の最大入力ゲート幅の制限による)

TDC の測定最大時間は 1[μs] なので、Start 信号は RF 入力立ち上がりが測定可能な最小時間となる程度に調整されている。

3.3.3 音響波形記録

音響検出器の信号は、VME の 10MHz Digitizer で波形を記録する。この VME Digitizer Module は、SLAC にてこの音響検出器専用に開発されたものである。各モジュール 16 チャンネルの読み込みが可能で、計 4 台のモジュールを用いて 64 チャンネルの波形測定が可能となっている。

表 3.11 に、Digitizer の仕様を示す。

VME では、デバイスのメモリにコントローラ (PC) 側からアクセスする形でデータ収集を行う。このモジュールも、表 3.12 にあるように各種設定項目、FIFO の現在位置への参照がレジスタに割り当てられ、そこからデータ収集を行う。データ収集については後述する。

この Digitizer は、1024 チャンネルの波形 3 つを保存することができる。データ収集には時間がかかる (1024 チャンネルでは 20ms 程度) ので、通常時はモジュールからのデータの読み出しは行わず、放電時にモジュー

チャンネル数	16(4×RJ45)
周波数	20〜1000[KHz]
入力電圧	10[mVpp]〜2[Vpp]
インピーダンス	100[Ω]
メモリ	4096[words/ch]
ダイナミックレンジ	12[bit]
ADC 周波数	〜20[MHz]
Input Amp.	Analog Divices AD605
ADC	Burr-Brown ADS2806 dual 12bit 32MHz
FIFO(First In First Out)	IDT 72V3660 4096x36 10nS
FPGA	Xilinx XC2S100

表 3.11: Acoustic Digitizer 仕様

アドレス	サイズ [Bytes]	R/W	機能
0x00〜0x1F	2×16ch	R	Data
0x20	4	R/W	Mode Control/Status
0x24	4	R/W	ADC Clock Control/Readback
0x28	4	R/W	ADC Gain DAC Control/Readback
0x2C	4	R/W	Loopback Register
0x30	4	R	ADC board version(0x10)
0x34	4	R	Trigger freq.
0x38	4	R	Clock freq.
0x3C	4	R	FIFO word count

表 3.12: Acoustic Digitizer 機能一覧。VME のレジスタアドレスは 24bit で、上位 16bit がモジュールアドレス、下位 8bit を各レジスタに割り当てている。ここでは下位 8bit のみを示している。

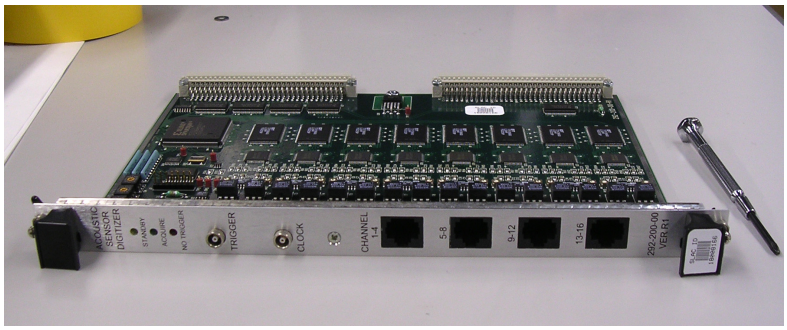

図 3.37: Acoustic Digitizer

ルに蓄えられた 3 パルス分の波形を読み出す。

ソフトウェアはモード切替により全パルス収集することもできるよう開発しているが、50Hz 運転では読み出しが追いつかないため通常の運転時は使われていない。

3.3.4 オシロスコープ

XTF には 1 台の Yokogawa DL7480 (500MHz 帯域,2GS/s,8ch) と 1 台の Tektronix TDS3034B (300MHz, 2.5GS/s, 4ch),2 台の Tektronix TDS2024 (200MHz, 2GS/s, 4ch) シリーズのオシロスコープが測定系に組み込まれており、各種データ収集に用いられている。

DL7480 は History Memory 機能を持ち、1000 程度の波形を蓄えておくことができる。読み出しには時間がかかる (〜1 分) ので放電検出後に、蓄えたパルスを PC に読み出す。現在は、クライストロンの出力波形、加速管出力、反射 RF の位相等の測定に使われている。定常的なデータ収集よりは、必要に応じて読み出す波形を変えて汎用的に用いられている。

DL7480 の仕様を表 3.13 に示す。

この DL7480 は BDMS に組み込まれ、放電検出時にネットワークを介して自動でデータを読み出し、ファイルに保存している。トリガは BDMS で使用しているものをそのまま配信している。

TDS3034B,TDS2024 は主にモジュレータ、クライストロンの出力波形、加速管の入出力波形のモニタに使われている。ヒストリーメモリ機能はないため、セルフトリガで使用していることが多い (BDMS のトリガも使用可能)。BDMS には組み込まれていないが、個別のデータ収集ソフト

チャンネル数	8
周波数帯域	500[MHz](50Ω 入力,1V/div 以下)
サンプルレート	1[GS/s](インタレース OFF)
分解能	8[bit]
内蔵メモリ	4[Mword/ch]

表 3.13: Yokogawa DL7480 基本性能

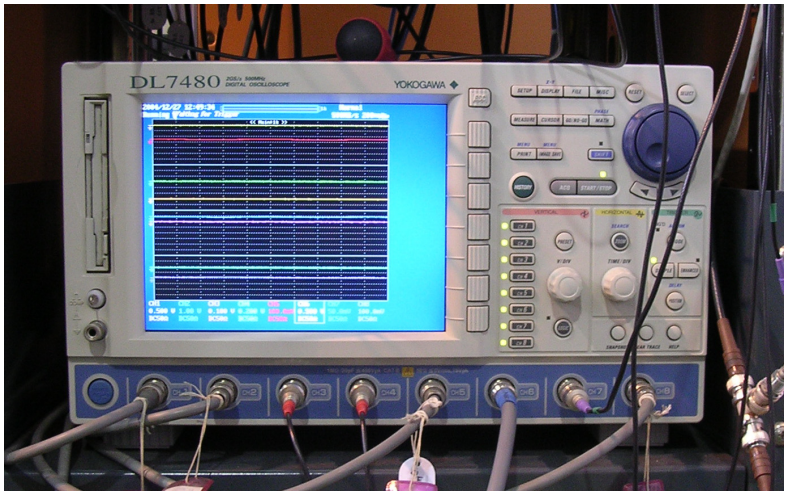

図 3.38: DL7480

ウェアを用いてネットワークからデータを読み出し、ファイルに保存することはできる。

3.3.5 トリガー管理

BDMSのトリガは、すべてXTFコントロールのマスタトリガから分配している。トリガの供給先は以下の通り。

- RFパルス成形モジュール (クライストロン制御)
- RF FADC
- γ線 ADC・TDC
- CAMAC Controller(CC/NET)
- VME Acoustic Digitizer
- オシロスコープ (DL7480,TDS3034B,TDS2024)

XTF-BDMSでは、測定系に通常の (リアルタイムOSでない)Linuxを用いているため、ネットワークその他の割り込みにより稀にパルスを取りこぼす (次のパルスまでにデータ収集・オンライン解析が終わらない) ことがある。そこで取りこぼしが発生した際に測定系が混乱するのを防ぐため、データ収集完了まで次のパルスを許可しない機能を付加している。

この機能はCAMAC CSY(Clear SYncronization with CAMAC access)モジュール (Technoland[6] C-TS 802) により実現されている。このモジュールはトリガーにVETOをかけるためのモジュールで、トリガが許可されているときは入力パルスを (約500nsの遅延ののち) そのまま出力し、不許可の時は出力しない。チャンネル数は4chで、独立に許可/不許可が設定できる。

CSYにはThroughモードとControlモードがあり、Controlモードでは毎回のCAMAC命令で次の入力1パルスのみを出力ポートへ中継する。データ収集が完全に終了してからこのCAMAC命令を発行することにより、取りこぼしの際の混乱を防ぐことができる。

ただし、RFパルス成形モジュールに供給しているパルスは、不意のパルス抜けがクライストロン等の運転に悪影響を及ぼすおそれがあるため、

[6]http://www.tcnland.co.jp/

コマンド	機能
F(16)Data(0x0〜0xF)	各チャンネルの許可/不許可
F(24)	全チャンネルを Control モードに設定
F(26)	全チャンネルを Through モードに設定

表 3.14: CSY モジュール コマンド一覧。F(16) は Control モードで下位から (0〜3)bit を立てることで該当チャンネルの次のパルスが許可になる。

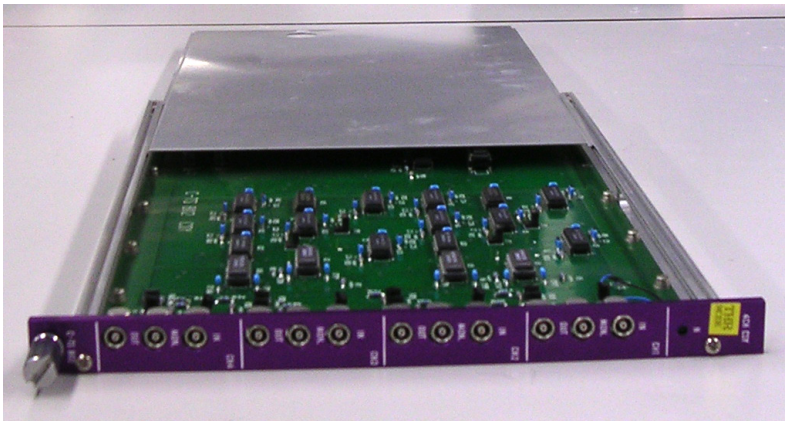

図 3.39: CSY Module

モジュール	チャンネル	割り当て
1	1	CAMAC(RF,γ),DL7480 オシロスコープ
1	2	VME,TDS オシロスコープ
1	3	unused
1	4	unused
2	1	unused
2	2	unused
2	3	#1 RF パルス成形化モジュール (現在は未使用)
2	4	#2 RF パルス成形化モジュール (現在は#1#2 両方に使用)

表 3.15: CSY モジュール チャンネル割り当て表。モジュール 1 は運転中は常時 Control モード、モジュール 2 は常時 Through モードで放電検出時のみ Control モードになる。

別の CSY モジュールを使い別個に管理している。この CSY モジュールは Through モードで使用されており、放電を検知したときのみ CAMAC の中継不許可命令を発行して以降のパルスを抑制する。

CSY モジュールは 4 チャンネルあり、表 3.15 のような割り当てになっている。CSY モジュールを通過したパルスは、Logic Fanout で複製したあと、それぞれの Delay や Gate を通して、各モジュールに送られる。

3.4 データ収集系

本節では、放電検知・記録コントロールソフトウェアについて述べる。

3.4.1 概要・特長

XTF の放電検知･記録ソフトウェアは、次の機能を主眼として作られた。

- 効率の良いデータ収集

  運転する繰り返し周波数に合わせ、決まった測定レートで各種データを収集する必要があるため、データ収集の効率が要求される。とくに FADC の波形を毎パルス取得する CAMAC がシビアである。また、CAMAC 以外にも様々なデバイスを用いてデータを収集しているので、その連携も重要となる。

- パルスごとの柔軟かつ高速な放電検知

ソフトウェアで放電検出し次のパルスを停止するため、毎パルス放電検知を行う必要があり、時間的制約が厳しい。また試験設備のため、放電検知の方法も柔軟に変更できるようにする必要がある。

VME, オシロスコープは毎パルスのデータ収集が不可能なため、放電検出は CAMAC のデータ収集ソフトウェアのみで行う。

- 効率よく柔軟なファイル形式

  収集すべきデータも途中で変わる可能性があるため、柔軟なファイル形式が要求された。今回のシステムでは、root[19] のファイル形式を基本とし、必要に応じて独自バイナリ形式でもファイル出力を行う。

  このため、root ライブラリを測定プログラムにリンクする。

- ネットワークでの連携

  測定の開始・停止、パラメータ変更、イベントの表示など、測定 PC が設置してある場所から離れたコントロールルームで運転制御を行うため、さまざまな設定はネットワークを介して行っている。通信には ATF で標準的に使われているライブラリを用いる。

- ユーザーインターフェースとの通信

  ユーザーインターフェースはコントロールソフトウェアとは完全に別プログラムになっており、前項で述べた通信のみを用いてユーザーインターフェースソフトウェアがコントロールソフトウェアに指令を伝える。ユーザーインターフェースのソフトウェアは JAVA で書かれており、コントロールが C++なので言語間の連携も必要となる。

BDMS は、

1. BDMS-CAMAC / CAMAC データ収集および放電検知
2. BDMS-VME / VME データ収集
3. BDMS-Oscillo / オシロスコープデータ収集
4. BDMS-Interface / 設定、イベント表示インターフェース
5. BDMS-Control / コントロールサーバ

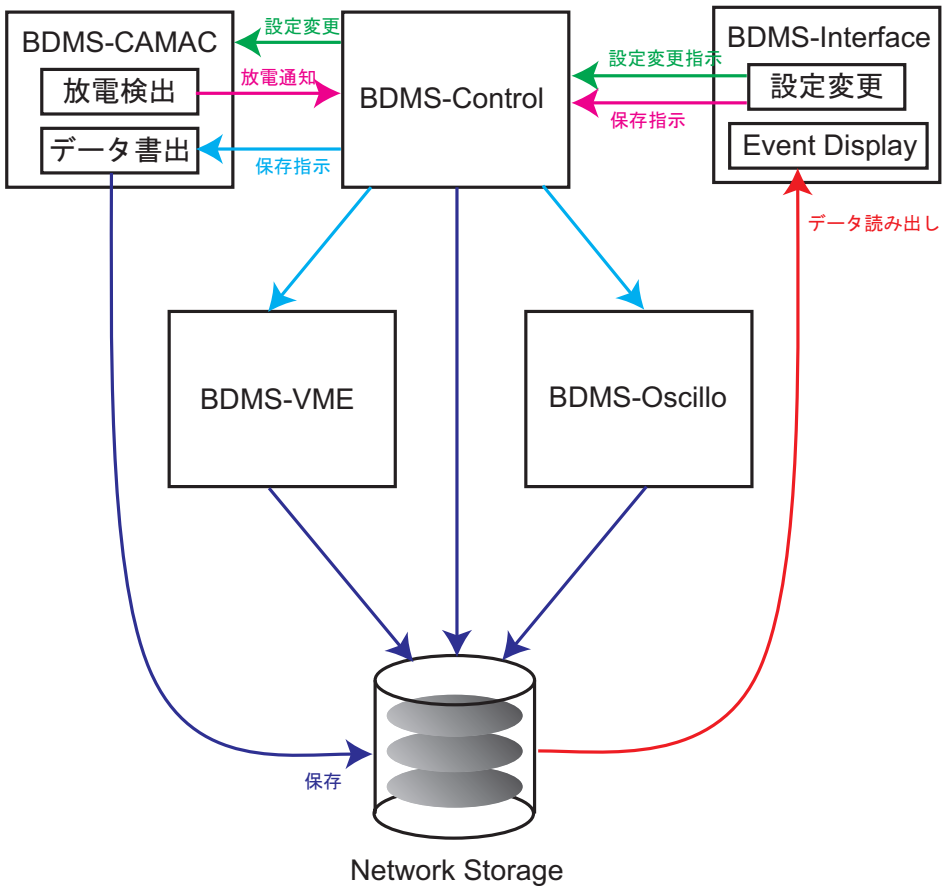

図 3.40: XTF-BDMS プロセス関係図

の主に 5 つのプロセスに分かれている。図 3.40 に、XTF-BDMS のプロセス間の関係を示す。

以下、それぞれの項目について詳細に述べていく。

3.4.2 BDMS-CAMAC

BDMS-CAMAC は、XTF-BDMS の心臓部であり、RF および γ 線検出器のデータ収集／保存・オンライン放電検出・測定系トリガ管理を行っている。

このソフトウェアは C++を使って書かれており、root ライブラリ、CC/NET ドライバ、ATF 通信ライブラリを使用 (リンク) している。オブジェクト

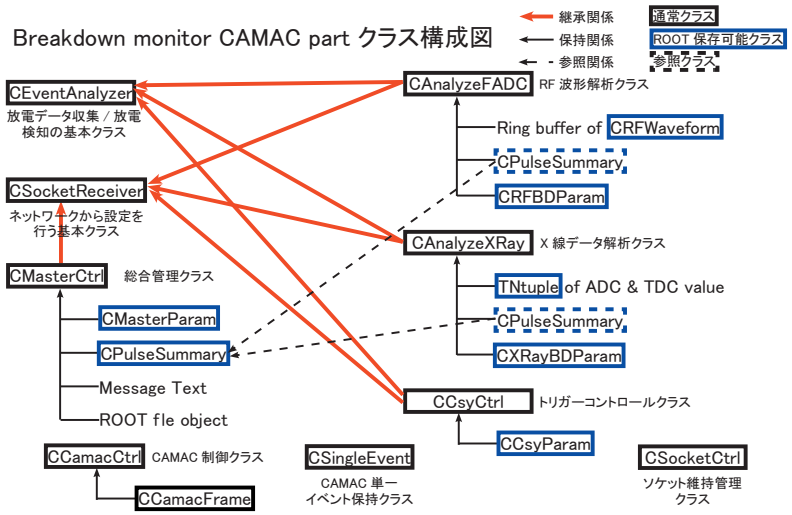

図 3.41: BDMS-CAMAC クラス構成図

指向にのっとり、機能毎のクラス分け・ファイル分けがなされている。

ソフトウェアが保存されているのは/mnt/nas/work/breakdown/camac/cc-net/server/以下のディレクトリである。

図 3.41 にクラス構成図、図 3.42 にフローチャートを示す。

以下、コンパイル方法、起動方法を示す。

コンパイル方法

```
suehara@glcta-02># root ライブラリ読込 /cern/root を
suehara@glcta-02># /mnt/nas/data/lib/root に ln -s しておく
suehara@glcta-02>. /cern/root/start-root.sh
suehara@glcta-02># ソース Directory に移動して make
suehara@glcta-02>cd /mnt/nas/work/breakdown/camac/
suehara@glcta-02>cd cc-net/server/current
suehara@glcta-02>make
suehara@glcta-02># 完成
```

start-root.sh は root を使用・コンパイルするのに必要な環境変数を定義するスクリプトである。

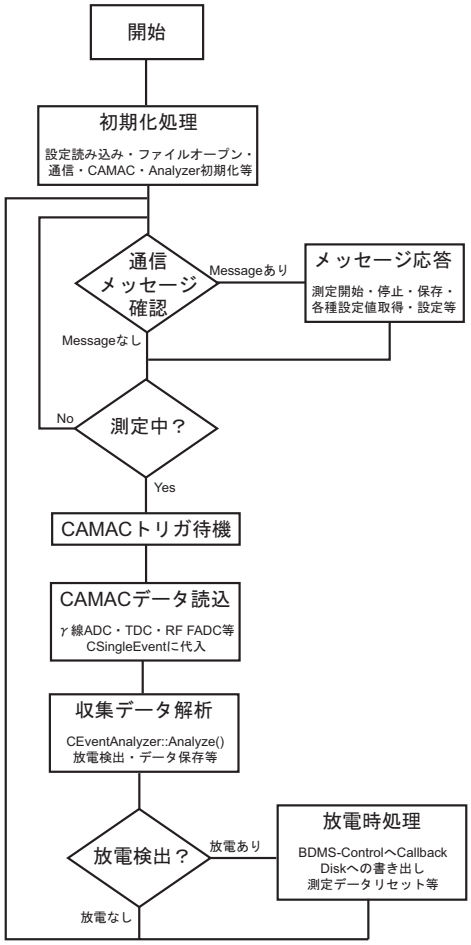

図 3.42: BDMS-CAMAC フローチャート

起動方法

```
suehara@glcta-02>ssh glcopr@glcta-ccnet1
Password: xxxxxxxx
glcopr@glcta-ccnet1>./bdmonitor start
```

./bdmonitorはBDMS-CAMAC起動スクリプトで、プログラムの実体は`/mnt/nas/work/breakdown/camac/cc-net/server/current/bdmonitor`である。

3.4.3 CC/NETによるCAMACモジュールとの通信

CC/NET[20]は、CAMACクレートコントローラ内蔵PCで、本システムのCAMACインターフェースに使われている。

CC/NETの高速通信 前節で述べたように、CAMACはパルス毎にデータ取り込み、放電検知を行うためデータ取り込み速度が極めて重要である。本システムでも当初CC7700クレートコントローラとLinux PC(glcta-sv3)を用いていたが、速度が不十分なためFADCの測定点数を500点以上に上げることができなかった。CC/NETはPCを内蔵し、CAMACコマンドのパイプライン処理(後述)という方式を採用することにより、CAMACバスの周波数1MHzの制限からはほぼ理論値に当たる1.04[μs/命令]という実行速度を達成している。[21]このため、BDMS-CAMACで最も通信量の多いRF波形の取得に必要な1000点$\times$4chの通信時間は4.16[ms]となり、50Hz,100Hzでの運転が十分可能となる。

CC/NETの運用 CC/NET上で動作するしているのはRedHat8.0をCC/NET用にカスタマイズして作られたもの[7]で、1GBのCompactFlash上に書き込まれCompactFlashから起動する(標準構成ではCFは512MBだがアップグレードしてある)。XTFコントロール端末のFedora core2とはいまのところバイナリ互換性がある(カーネルのバージョンは違うためドライバ類のコンパイルを除く)。

そこで、CC/NET上で動作する放電検知・記録ソフトウェアはコントロール端末でコンパイルし、NFSマウント上のディスクから起動のみCC/NET上で行う。これは、CC/NETのCPUが遅いためコントロール端末上と比べてコンパイル時間が3倍以上違うためである。

[7] http://www-online.kek.jp/~inoue/Parallel-CAMAC/Work/

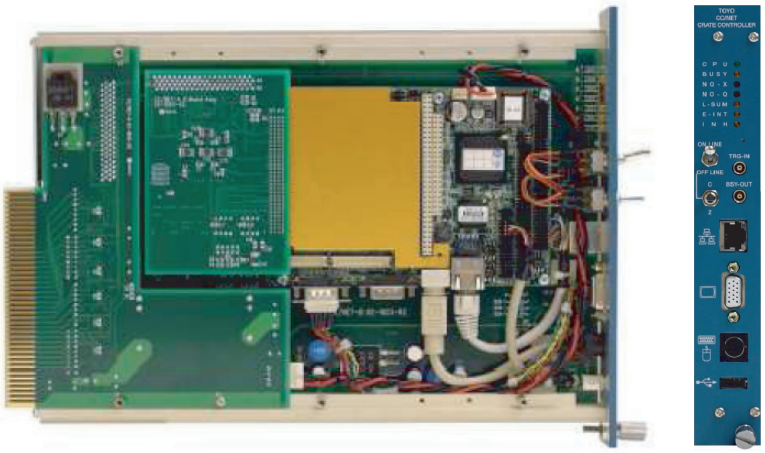

図 3.43: CC/NET。PC104+という Single Board Computer と CAMAC クレートコントローラが一体化している。フロントパネルには PC 用のインターフェースが一通りそろっている。XTF では普段は端末はつないでおらず、すべてフロントパネルのネットワークポートを介して外部から操作を行う。写真は [22] より。

パイプライン処理と CAMAC frame　CC/NET 用の CAMAC ドライバは CC/NET と共に供給されているものを使う。現在使用しているバージョンは camac.tar.gz-Dec192003[8]である。このドライバ [22] には、互換関数とパイプライン処理用のオリジナル関数が用意されている。今回は最高速度を出す必要があるため、パイプライン処理用の関数を用いている。

パイプライン処理は、いくつかの CAMAC 命令をまとめてドライバに送信し、戻り値をまとめてとってくる方式で、戻り値の評価を待たずに次の命令を発行するため高速な命令処理が可能である。パイプラインの概念図を図 3.44 に示す。

CC/NET の CAMAC ドライバは、Frame とよばれる処理単位を CAMAC コントローラとやりとりすることにより CAMAC 命令を処理する。Frame には command frame と reply frame があり、command frame は概念的には CAMAC 命令 (NAF,data) の配列であり、reply frame は応答 (data,Q,X) の配列である。

BDMS ではこの frame を CCamacCtrl::CCamacFrame にカプセル化している。frame を CCamacCtrl::Put() 関数で生成し、CCamacCtrl::Send()

[8]http://www-online.kek.jp/ yasu/Parallel-CAMAC/kits/camac.tar.gz

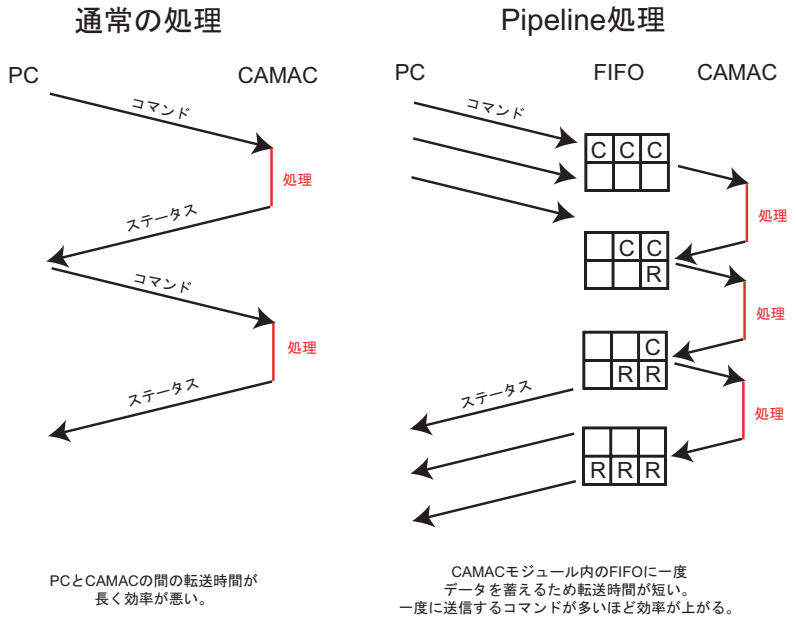

図 3.44: CC/NET のパイプライン処理

で送信、戻り値を取得する (生成した CCamacFrame は引数として渡す)。CCamacFrame のコンストラクタでは cam_gen_init() を呼び出し frame を初期化、CCamacFrame::Put() では cam_gen_cc() で frame に CAMAC 命令を push し、CCamacCtrl::Send() では cam_exec() を呼び出して frame を実際に送信している。

実際には 1 パルス分のデータを読み出すシーケンスは毎回変わらないため、1 度生成した command frame は毎回のデータ収集により再利用する。この frame 再利用を行うことで 10ms 以内の 1 パルスデータ収集が行えるようになった。

CC/NET の割り込み処理　次に、CC/NET における割り込みを説明する。

CAMAC における割り込みは通常 CAMAC モジュールの LAM ビットを用いるが、XTF-BDMS では LAM を発行すべきモジュールが複数にわたるのと、当初 CC/NET のドライバが LAM を正しくハンドリングできないバグを持っていたため、LAM は使用していない。その代わり、CC/NET の Trig In ポートに、各モジュールの変換時間に余裕をみた Delay をかけた Master Trigger を入力し、この Trig In で割り込みをかけている。

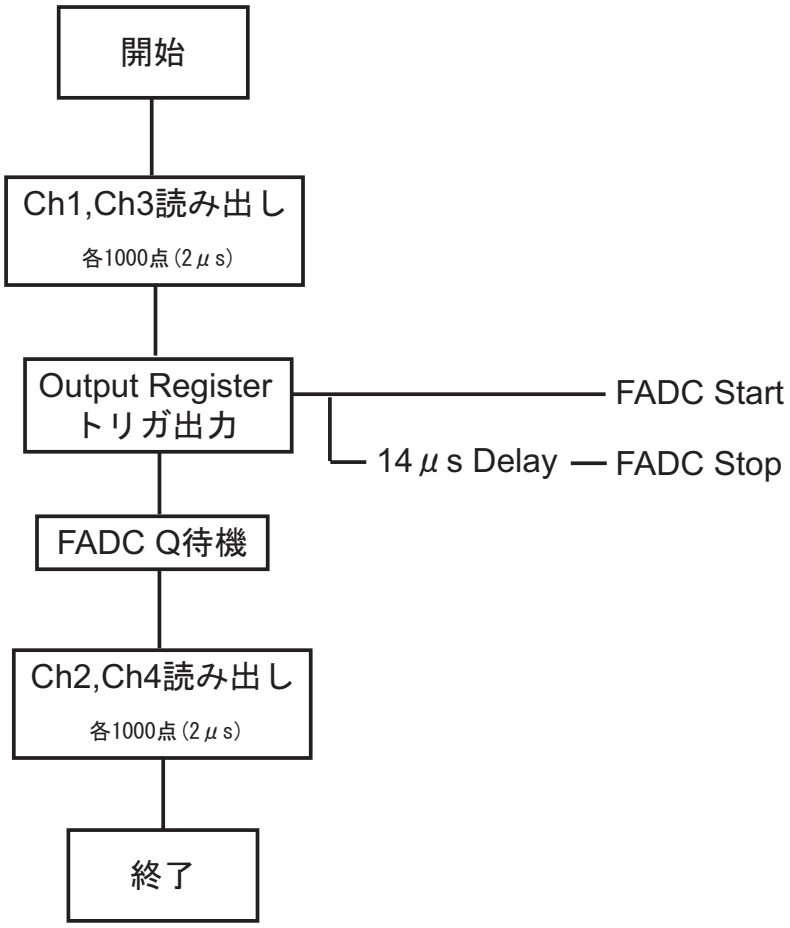

図 3.45: FADC 読み込み部の流れ

FADC のデータ取得　また、前節で述べたように、FADC の指定チャンネル数読み込みにはチャンネル切替の際にパルスを出す必要がある。これは output register を用いて 1 チャンネルの読み込みが終了した段階でパルスを発生させ、LAM ビットが立つのを待ってもう一つのチャンネルを読み込む方法によって対応する。

FADC 読み込み部の流れ図を図 3.45 に示す。

CSingleEvent　CAMAC から取得したデータはいったん CSingleEvent クラス内の配列にモジュール毎に保持される。XTF で現在使用しているデバイスは FADC が 2 台、Charge ADC2 台、TDC2 台、Output Register1 台,Scaler1 台,CSY2 台である。放電検出・記録時にはいくつかのモジュール

Status	Module 1(測定)	Module 2(RF)
未測定時	Control(Stop)	Control(Stop)
測定中	Control(毎パルス Enable)	Through
放電検知時	Control(Stop)	Control(Stop)

表 3.16: CSY モジュール モードの使い分け。

のデータをまとめて記録するが、CAMACからの読みとり部ではモジュール毎にいったん配列に記録する方式をとっている。

これは、以前 CC7700 を用いていたころはドライバの都合で CAMAC 読みとり部と解析部が独立したプログラムになっており、また CAMAC 読みとり部はカーネルモードで動作するため C の標準関数すら多くが使えない状態で複雑な処理に向いていないため読みとった状態をそのまま解析部に転送する方式をとっていたためである。CC/NET は導入当時ドライバに問題が多く本システムに使用可能かどうかわからなかったため、移植する際に極力以前の状態と容易に互換がとれるよう設計したため、現在のような状態になった。

ただ、現システムはモジュール毎に読み込み部が独立しており LAM チェックをモジュールごとに行っているため、モジュールの一部を取り外したときに対応が比較的容易という特徴ももっており、たんなる下位互換のためのみに現仕様になっているわけではないとも言える。

3.4.4 Trigger 応答

前節で述べたように、BDMS-CAMACは、RF および測定系のトリガ管理を行っている。トリガ管理用の CSY モジュールの機能はすでに述べた。

CSY の Control モード、Through モードの使い分けは表 3.16 のようになっている。

トリガ管理は放電検出後の処理として、CCsyCtrl クラスにより行われている。

放電検出時の処理については後述する。

チャンネル	割り当て
0	Master Trigger
1	CSY1ch1:CAMAC,DL7480 トリガ
2	CSY1ch2:VME,TDS トリガ
3	CSY2ch3:#1 RF トリガ (未使用)
4	CSY2ch4:#2 RF トリガ

表 3.17: BDMS CAMAC Scaler 割り当て表

3.4.5 Event 番号管理

Trigger 管理によりすべてのパルスが放電測定ルーチンに回されないので、実パルス数と測定パルス数の間の関連づけをする必要がある。

この関連づけには CAMAC の 12ch Scaler モジュール・KEK C1013 を用いている。入力している信号は表 3.17 の通り。

BDMS-CAMACでは、Ch0(Master Trigger) のカウントを Event #,Ch1 (CAMAC Trigger) のカウントを Acq #として、2 つのパルス番号をデータに付加している。

Scaler のクリアはシステムの開始時・放電後の測定開始命令 (CMaster-Ctrl::Start()) に同期して行われる。Scaler の読み込みは CSingleEvent への CAMAC データ取り込みに先立って行われ、各 CSingleEvent に Event #.Acq #が書き込まれる。

3.4.6 データ収集・放電検出

放電検出ルーチンの呼び出し　CAMAC ドライバを介したデータ収集の方法はすでに述べた。

ドライバからの出力はまずモジュール毎に CSingleEvent 内の配列に読み込まれる。CC/NET ではトリガは共通の測定トリガをコントローラのソケットから直接入力しているため、トリガによる割り込みがかかった時点で全モジュールは読みとり可能状態になっており、CSingleEvent への取り込みが終わった時点では必要なデータはすべて収集されているものとする。

データ収集・放電検出を行うクラスはすべて CEventAnalyzer を継承する。現在の本システムでは CAnalyzeXRay,CAnalyzeFADC の二つがそれ

仮想関数	機能
Initialize	初期化。書き込みファイルオブジェクトと PulseSummary が渡される。
Start	データ収集開始。各種カウンタのリセット等を行う。
Analyze	データ収集メイン関数。1 パルスのデータ収集ごとに呼ばれ、データを保存・放電検知を行う。
Stop	データ収集停止時に呼ばれる。
Save	root ファイルへのデータ保存時に呼ばれる。保存関連の処理を行う。
SaveBin	バイナリファイル出力時に呼ばれる。バイナリファイルの内容を生成する。
SaveBinPeriodic	定時収集パルスを保存する時に呼ばれる。

表 3.18: CEventAnalyzer 仮想関数一覧

にあたる。CEventAnalyzer の主な virtual 関数を示す。

CEventAnalyzer::Analyze() はデータ収集・放電検知のメインルーチンであり、CSingleEvent へのデータ取り込みが終わった時点で順番に呼び出される。この呼び出しは g_liAnalyzer というグローバルの std::list<CEventAnalyzer *> を巡回して行われるため、各 CEventAnalyzer 派生クラスはこのリストへの登録を行う必要がある。CAnalyzerXRay,CAnalyzerFADC ではコンストラクタにて行っている。

γ 線データ収集・放電検出　CAnalyzeXRay は γ 線検出器のデータ記録・放電検出を担当している。CAnalyzeXRay はメンバとして二つの TNtuple クラスを保持している。保存しているのは同じデータだが、一つは各チャンネルの ADC と TDC の組を 1 エントリとしており、もう一つは全チャンネルの ADC,TDC 値合計 24 チャンネルで 1 エントリとしている。

放電検出は、1 パルス前の ADC 値と現在の ADC 値の比較で行っている。放電検出については後述する。

RF データ収集・放電検出　CAnalyzeFADC は RF 波形の記録・放電検出を担当している。RF 波形の記録は、TTree を用いている。データ型は

CRFWaveformというもので、streamers.hで定義されている。これはrootライブラリではなく自前の型なので、rootに認識させるためClassDef()というマクロを用い、rootcintを通してシリアライザを自動生成している。

make dictによりstreamers.hからroot class dictionaryを自動生成するよう設定してある。streamers.hを変更したときはmake dictが必要である。

CRFWaveformはFADCの波形4チャンネルとEvent no.等の付加情報からなっている。

放電検出は、FADCの波形にピーク値、積分値等で閾値を設けてその条件を超えたら放電とみなす方式になっている。詳細は後述する。

CPulseSummary　また、パルス毎の統計情報を全パルス記録するため、CPulseSummaryというrootクラスを定義している。このクラスには現在X線のADC,TDC情報とRFの放電検出条件に対応した値(波形自体は除く)を記録しており、treeとしてパルス毎に保存している。これも、各Analyzerで行っている。

放電検出時の処理　いずれのAnalyzerも放電を検出するとg_csy.Breakdown()を呼び出す。呼び出されたCCsyCtrlは放電シーケンスを開始する。

放電シーケンスを開始すると、CCsyCtrlは一定のパルスを素通りさせたあと測定系CSYへのenableを停止、RF系CSYをコントロールモードに変更し、パルスを停止させる。素通りさせるパルス数は設定で変更でき、現在は基本的に0(即停止)となっている。この素通り中は各Analyzerはデータ収集のみ行い、放電検出を行わない(CCsyCtrlに素通り中かどうかのフラグがあり、Analyzer側でチェックしている)。

パルスを停止すると、コールバック通知登録がなされているソケットに向けて放電通知を行い、各ソケットから返答が戻るまで待機する。このとき使用不能になっているソケットは自動的に破棄する。

通知が終了するとAutoSaveフラグ(通信により設定可能・現在はON)が立っていれば、データ保存(後述)を行う。その後AutoStartフラグ(設定可能・現在はOFF)が立っていれば、現在までのデータをクリア(CEventAnalyzer::Clear()呼び出し)し測定を再開(CEventAnalyzer::Start())する。AutoStartフラグが立っていない場合はネットワークからの測定開始指令を待機する。

3.4.7 データ保存

データ形式概要 それぞれの測定データは、放電検出時、RF パワー停止から復旧までの時間を使って保存される。保存形式は、主に root 形式ファイルと、生データの (独自仕様) バイナリファイルの 2 種類となっている。

root 形式のファイルは、1 ファイルの中に各種の名前つき (TNamed 継承) データオブジェクトを保存することができ、また保存する毎に新しいリビジョン番号で保存する。保存はオブジェクトを指定するだけで自動で行われ、(原理的には) 簡便である。さらに、自動でデータ圧縮する機能も備わっている。読み込みも、root では名前を指定するだけで保存時と同じクラスに自動的に代入され便利である。ただし複雑なデータ形式のため、読み込み・書き込みには root が必要である。イベントディスプレイは Java で作られており Java で root ファイルを読むのが難しいためイベントディスプレイ用にはバイナリファイルを供給している。

root 形式のデータ書き出し root はオブジェクトの配置をディレクトリ構造に行っており、root ファイルは root 内のディレクトリとして保存したいオブジェクトに先立ってオープンする必要がある (あとでオブジェクトを移動する方法もあるが)。そのためファイルのオープンは起動時の初期化の際に行っている。オープンすると root のカレントディレクトリがオープンしたファイルのディレクトリに変更され、以後生成されるオブジェクトは (デフォルトで) 今開いたファイルに保存されることになる。ファイル名は日付 (yyyymmdd.root) という規則で生成され、1 日に一つのファイルを生成する。ただしファイルは起動時のみの生成なので、ファイルを変更するには放電検知・記録ソフトウェアを再起動する必要がある。また同じ日に 2 度以上起動した場合は前のデータにその起動のデータが追加される。

ファイル名は、2004 年 10 月までは yyyymmdd-hhmm.root という形式で時分もつけていたため、以前のファイルを開くときはこの形式に従って指定する。1 分に 2 度起動したときは同じファイルになるが、普通 1 分に 2 度起動しないため、事実上 1 起動に 1 ファイルとなっていた。自動解析の際にファイルを探すのが不便なため変更した。

保存ディレクトリは/mnt/nas/data/rawdata/年/月 となっている。

TTree,TNtuple の利用 root ファイル内に保存できるのは、root の built-in クラスと root 規約に従って作ったクラスのみである。root は高エネル

ギー実験用に作られているため、高エネルギーでよく使われる”構造化されたデータ (Event) の繰り返し”を保存するように最適な設計となっている。この繰り返しには TTree またはその派生クラスの TNtuple,TNtupleD クラスを用いる。TTree は構造化データを自分であらかじめ定義し使うのに対し、TNtuple,TNtupleD はデータが単なる float(TNtuple) または double(TNtupleD) の配列であると想定し、簡易にデータ登録ができるようになっている。

TTree を使うには、まずデータ構造を登録する必要がある。これにはデータ型を一つずつ登録する方法とイベントクラスを定義しそれをまとめて登録する方法があるが、本システムでは後者を用いている。データ型の登録には、TTree::AddBranch() を用いる。データ型は複数登録できるが、本システムでは一つのイベントクラスにまとめている。

Tree に書き込むには、この登録時にポインタを渡したイベントクラスにそのイベントのデータを書き出したあと TTree::Fill() を呼ぶことで、その登録時のポインタをから Tree が内部の配列にデータをコピーすることで行われる。

本システムで Tree を使って保存しているのは、RF(波形と付加情報) データ (CRFWaveform)、パルス統計情報 (CPulseSummary) の二つである。

TNtuple,TNtupleD は、Fill() 関数に float(double) の配列を渡すか、配列長が 16 以内なら Fill() 関数の引数として直接値を並べることで簡単に登録が行われる。本システムでは γ 線のデータは Ntuple に保存している。

TTree::Fill() ではデータはメモリ内に保存されるが、ファイルへの保存は行われない。ファイルへの保存は TFile::Write() でまとめて行われる。これは、ファイルをオープンしたディレクトリ内のオブジェクトすべてについて再帰的に Write() を呼び出すことにより実現している。オブジェクトは名前をキーとして保存されるが、同じ名前のオブジェクトがすでにあるときは、オブジェクトの Cycle No. を一つ足して保存するため前のデータを上書きすることはない。本システムでは、保存データは、放電毎に同じ cycle になるようにしているため、日付 (ファイル名) と Cycle No.(その日内のイベント番号に相当する) でイベントを取り出すことができる。また、放電の発生時刻をオブジェクトのタイトルに埋め込んでいるため、それを鍵に探すこともできる。

データの保存は、上記の通り TFile::Write() を呼び出すだけで自動的に行われるが、保存前のタイトル設定等の処理を行う必要があるため、TFile::Write() 呼び出しの前に CEventAnalyzer::Save() を呼び出して各

Analyzer で処理を行う。この Save() には時間情報を含めた文字列が渡されるため、各 Analyzer で統一の発生時刻を付加することができる。

放電検出条件の保存　また、次節で説明する放電検出条件についても、同じファイル内に保存している。これは、CMasterParam,CRFParam,CXRayParam という名前のクラスになっており、Tree ではなく単体を保存している。この保存は放電時に加えてネットワークを介した条件変更の際にも行われるため、Cycle No. が上記のデータとは一致せず、名前 (発生時刻) で検索する必要がある。

これらの放電検出条件は、前回終了時のものを起動時に呼び出して適用する必要がある。この際前回起動のファイルを探す手間を省くため、起動の際に /mnt/nas/data/rawdata/recent.root というファイルを今回用いるファイルへのハード リンクとして生成し、次回起動時に参照できるようにしている。以前はシンボリックリンクを用いていたが、RAID の導入によりシンボリックリンクが貼れないファイルシステムに移動したためこのようになった。前回のリンクはリンク先を開いてパラメタクラスをコピーした段階で破棄される。

バイナリ形式のデータ書き出し　バイナリファイルは各 Analyzer の SaveBin() を呼び出すことによって保存している。SaveBin は Save() から自動で呼び出されるため、通常は root ファイルを保存した際同時にバイナリファイルも保存する。現在 CAMAC の放電検出・記録システムから書き出しているバイナリファイルは 3 つで

1. 放電前 500 パルスの RF 波形
2. 1 分に 1 パルスサンプルをとった RF 波形
3. γ 線 ADC,TDC データである。

ファイルは放電について 1 つずつ作られ、それぞれ、

1. [b,n]yyyymmddhhmmss.bdmon-camac.bin
2. [b,n]yyyymmddhhmmss.bdmon-camac.periodic.bin,
3. [b,n]yyyymmddhhmmss.bdmon-camac.xray.bin の名前がつく。

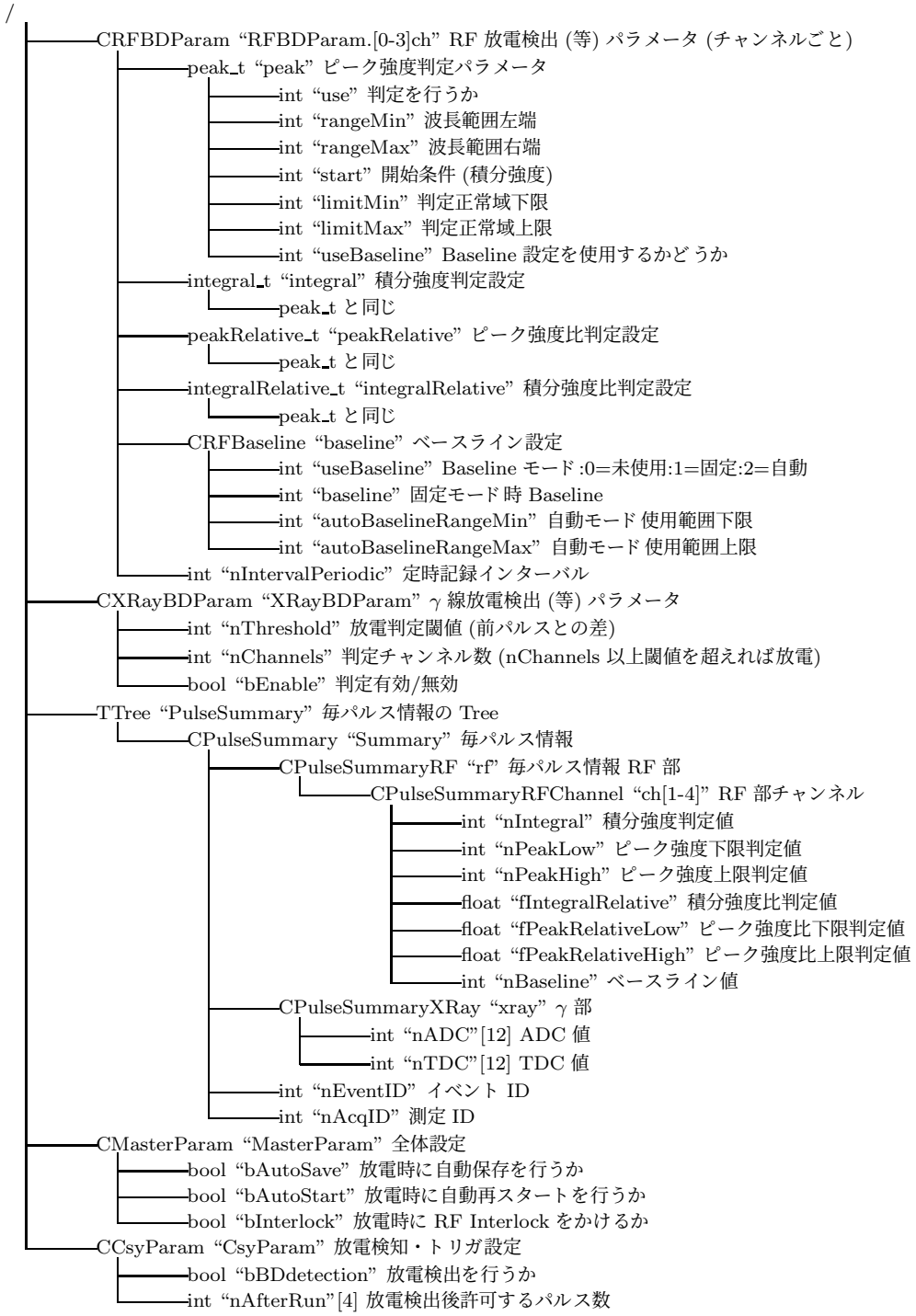

表 3.19: BDMS-CAMAC 保存 root オブジェクト一覧 (1)

/
- TTree “rf” RF 波形の Tree
  - CRFWaveform “rf” RF 波形
    - TArrayI “arrData”[4] 各チャンネル FADC 波形配列
    - int “nEventID” イベント ID
    - int “nAcqID” 測定 ID
    - TString “strTime” 取得時刻
- TTree “rf_periodic” 定時保存 RF 波形の Tree
  - “rf”と同じ
- TNtuple “acc-gamma” γ 線データ Ntuple
  - float “eventid” イベント ID
  - float “acqid” 測定 ID
  - float “adc[1-12]” ADC 値
  - float “tdc[1-12]” TDC 値
- TNtuple “acc-gamma2” γ 線データ Ntuple 別形式
  - float “eventid” イベント ID
  - float “acqid” 測定 ID
  - float “ch” チャンネル
  - float “adc” ADC 値
  - float “tdc” TDC 値

表 3.20: BDMS-CAMAC 保存 root オブジェクト一覧 (2)

ファイル形式を以下に示す。ファイルサイズは、放電前 RF 波形が 6.2MB 程度、定時保存,γ 線データは放電間隔により異なる (放電前 RF 波形より小さいものが多い)。

RF Processing はデータ保存が終わるまで待機するので、このデータを保存している間は Processing を行うことができない。保存時間は periodic イベントの総数、すなわち前回放電との時間差等により変化する。

3.4.8 通信機能

CAMAC 放電検知・記録ソフトウェアは、放電監視の開始・停止、放電検出条件の変更などの設定を外部からネットワークを通して行う。通信には atf_socket ライブラリ (/mnt/nas/data/lib/atf_socket に保存されている) を用いており、ATF のプロトコルにより 64KB までのデータの送受信が簡易に行えるようになっている。

ATF ライブラリ内で使用している関数を表 3.22 に示す。

ATF ライブラリには上位の書き込み・読み出し関数として、特定の型の配列を読み込む機構があるが、配列の範囲チェック等に不備があるため、本システムでは使用していない。ATF ライブラリでは表 3.23 の構造体をパケットとして送受信している。詳細なハンドリング手順はライブラリ内部の仕様なのでここでは省く。

サイズ	型	内容
-	String	“<MESSAGE>”
-	String	放電検知メッセージ
-	String	“</MESSAGE>”
-	String	“<RF-FADC>”
-	String	“Time: ” 時刻 (yyyy/mm/dd hh:mm:ss)
-	String	“Title: Time,Pf,Prs,Pra,Ptr”
-	String	“DataSize: ” 時間軸点数 (1000)
-	String	“DataCount: ” パルス数
-	-	以下パルス数分繰り返し
-	String	“Data ” イベント ID “ ” 測定 ID “ Start”
-	-	以下時間軸点数分繰り返し
2	short	時間 (0〜時間軸点数-1)
4	char[4]	FADC 値 (各チャンネル 1byte)
-	-	以上時間軸点数分繰り返し
-	String	“Data ” イベント ID “ ” 測定 ID “ End”
-	-	以上パルス数分繰り返し
-	String	</RF-FADC>

表 3.21: BDMS-CAMAC バイナリファイル形式。String は改行がデリミタ。

関数	機能
atf_socket_server_open	サーバーソケットを開く
atf_socket_server_accept	接続を待機
atf_socket_client_connection	クライアント接続
atf_socket_read	ソケット読み込み
atf_socket_write	ソケット書き込み

表 3.22: atf_socket ライブラリ BDMS での使用関数一覧

位置	サイズ	内容
0	4	HEADER_CODE=123456
4	4	送受信 ID
8	4	受信状態 (未使用)
12	4	型 (String=0 固定)
16	4	データサイズ (Header 含む)
20	108	未使用
128	-	データ

表 3.23: atf_socket ライブラリ 送受信パケット。受信状態・型は BDMS-CAMAC/VME で使っている場合。

この ATF ライブラリのプロトコルの上部に、本システム独自のプロトコルを実装している。このプロトコルでは、スペース文字を空白 (0x20) ないし改行 (0x0a) として、コマンドをスペースで区切って文字列のまま送受信する。

送信する文字列は以下のようになっている。

本ソフトウェアは、通信サーバ側として機能するため能動的に接続を行うことはないが、port5555 および 5556 を listen しており、クライアントからの接続要求を受け付ける。5555 は設定変更用のソケットを受付、5556 は放電時のコールバックを発行するためのソケットを受け付ける。

クライアントは基本的にはコントロールサーバであるが、特に制限はなく、後述のテキストベース通信クライアントでキャラクタベースの設定変更も行える。接続時には特に手順はなく、接続を受け入れるとそのままコマンド待機状態になる。1 つのポートにつき接続制限はないため、複数のクライアントと同時に通信できる。受け入れた接続はソケットのリストに登録される。

port5555 は 1 パルスに 1 回メッセージが着信していないか監視している。メッセージが到着していなければそのまま CAMAC データ取得に移り次のパルスを待つ。メッセージが到着していた場合は指定された処理を行い、応答を返してからふたたびメッセージ到着を確認する。

放電時のコールバックは、放電検出シーケンスの一部としてデータ保存の直前に行われる。これは、port5556 に登録されていて通信可能なすべてのソケットに対して行い、応答が帰るまで一ソケットずつ待機する。

これらのメッセージを簡単に送受信するため、テキストベースの通信

型	機能
int	48384 (Header)
int	1 (Packet Category)
int	ID
String	Message Category
String	Message
いろいろ	パラメータ (Messageごとに異なる)
int	48639 (Footer)

表 3.24: BDMS 送受信パケット仕様 1:Request Packet。空白文字 (スペース・改行) で切り分ける。よって String に空白は不可。int も文字として読める形で送信される。

型	機能
int	48384 (Header)
int	2 (Packet Category)
int	ID (Request に対応)
いろいろ	パラメータ (Request Messageごとに異なる)
int	48639(OK) or 48638(NG) (Footer)

表 3.25: BDMS 送受信パケット仕様 2:Reply Packet

型	機能
int	48384 (Header)
int	3 (Packet Category)
int	ID
String	Message
いろいろ	パラメータ (Messageごとに異なる)
int	48639 (Footer)

表 3.26: BDMS 送受信パケット仕様 3:Callback Packet

図 3.46: test-client 操作画面。0 は自由コマンド送信で、続いて送信するコマンドを (カテゴリから) 入力する。1,2 は測定開始・停止の定型コマンド送信。9 は終了。

クライアントが用意されている。これは主に通信テストとユーザーインターフェースから設定できない高度なメッセージを送信するために使う。

実行環境は/mnt/nas/work/breakdown/camac/test-client/日付/bdmon-ctrlで、同じディレクトリにソースもある。引数にサーバのアドレスを入力して起動する。

3.4.9 BDMS-VME

次に、BDMS-VME について説明する。

BDMS-VME は BDMS-CAMAC と同じフレーム上に構築されている。データ収集の開始・停止等は BDMS-CAMAC とほぼ同じ仕様の CMaster-Ctrl を使っているため、同様の操作で利用できる。CEventAnalyzer も基本的には同じで、実装されている CAnalyzeAcoustic で実際の音響検出器からのデータ収集を行う。root ファイルおよびバイナリファイルにデータを書き出している点も同様である。通信のプロトコルも基本的に同様であり、実装されているメッセージの種類が異なるだけである。

ただし、VME は基本的に放電検出後にデータ収集を行うため、放電検知関係の機能は実装されておらず、純粋にデータ収集だけのシステムとなっている。放電検知機能がないので、コールバック用ソケット (port5556) も存在しない。

クラス構成図、動作概要、ソース、コンパイル・実行方法・通信メッ

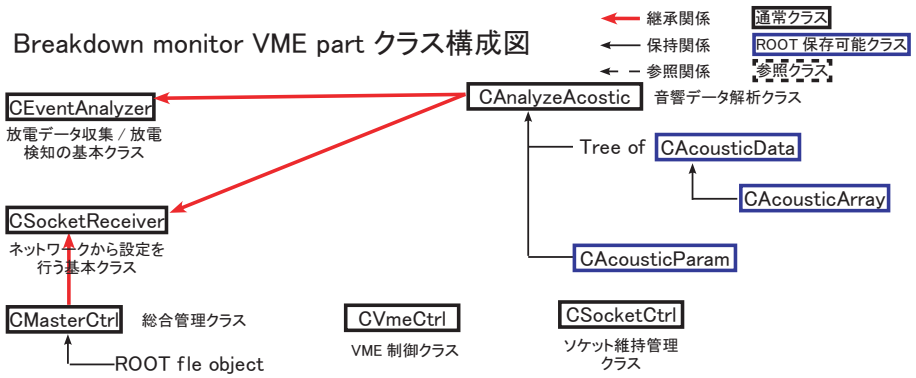

図 3.47: BDMS-VME クラス構成図

セージおよび、root ファイルのオブジェクト構成、バイナリファイルのフォーマットを説明する。

コンパイル方法

```
suehara@glcta-02># root ライブラリ読込 /cern/root を/mnt/nas/data/lib/root に ln -s して
おく
suehara@glcta-02>. /cern/root/start-root.sh
suehara@glcta-02># ソース Directory に移動して make
suehara@glcta-02>cd /mnt/nas/work/breakdown/vme/online/suehara/current
suehara@glcta-02>make
suehara@glcta-02># 完成
```

起動方法

```
suehara@glcta-02>ssh glcopr@glcta-vme1
Password: xxxxxxxx
glcopr@glcta-vme1>./bdmonitor start
```

./bdmonitor は BDMS-VME 起動スクリプトで、プログラムの実体は`/mnt/nas/work/breakdown/vme/online/suehara/current/bdmonitor`である。

3.4.10 Acoustic 応答

ここでは、VME からのデータ収集方法について述べる。

本システムで用いている VME クレートコントローラは、PC 一体型の VMIVME-7750 である。RedHat Linux 9 がインストールされており、コントロール系の Fedora Core 2 とはバイナリ互換性がある。このクレー

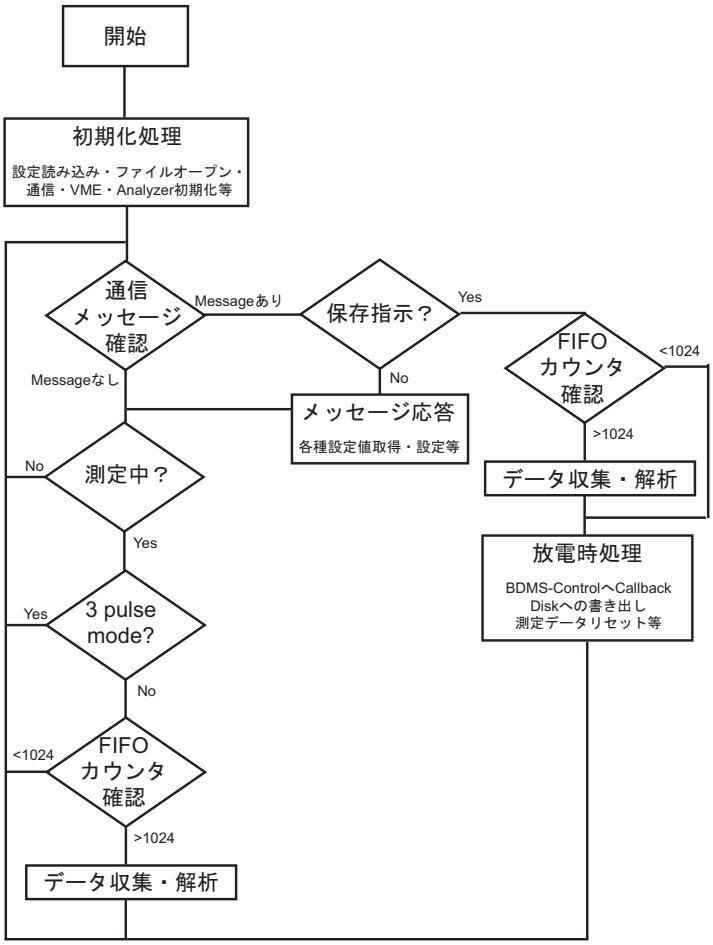

図 3.48: BDMS-VME フローチャート

/
- CAcousticParam “AcousticParam” データ収集パラメータ
  - int “m_nBufSize” 収集するイベント数 (リングバッファ)
  - int “m_bEveryRead” true:continuous モード ,false:3pulse モード
- TTree “acoustic” 音響波形データ Tree
  - CAcousticData “acoustic” 音響波形データ
    - int “nEventID” イベント ID
    - int “nAcqID” 測定 ID
    - CAcousticModule “module”[4] 各モジュールのデータ
      - int “nModule” ModuleID(0-3)
      - int “nGain” Gain 設定
      - int “nClock” Clock 設定
      - TArrayI “arrData”[16] 各チャンネルの波形

表 3.27: BDMS-VME 保存 root オブジェクト一覧

位置	サイズ	型	内容
0000	17	String	“VME Acoustic Data”
0017	4	int	Version (7)
0021	4	int	Number of Records : 1024 固定
0025	4	int	ADC clock Setting Ch1
0029	4	int	ADC gain Setting Ch1
0033	4	int	Number of Modules : 4 固定
0037	4	-	未使用
-	-	-	以下モジュール毎に繰り返し
0000	4	int	123123 (Separator)
0004	4	int	ADC clock Setting
0008	4	int	ADC gain Setting
0012	4	int	Trigger Frequency
0016	4	int	Clock Frequency
0020	4	int	Module Address
0024	4	int	Number of Pulses
0028	4	-	未使用
0032	-	short[]	生データ (2bytes×nRecords×nPulses)

表 3.28: BDMS-VME バイナリファイル形式

関数	機能
vme_init	初期化
vme_master_window_create	VME のアドレスの PC へのメモリマップ作成
vme_master_window_release	メモリマップ削除
vme_master_window_map	メモリマップをローカルメモリに適用
vme_master_window_unmap	ローカルメモリのメモリマップを削除

表 3.29: vme_universe BDMS-VME 使用関数一覧

トコントローラの CPU も (CC/NET よりは速いものの) 非力なのでコンパイルにはコントロール系の CPU を利用する。

VME ドライバは vme_universe と付属のドライバを利用している。このドライバの主な関数を以下に示す。

VME のモジュールにはそれぞれ 24 ビットのアドレスが割り当てられ、指定範囲のアドレスへのアクセスがモジュールに転送される。今回用いる SLAC の Digitizer モジュールは Dip スイッチでアドレスを指定するようになっている。今回 4 台のモジュールをそれぞれ 0x222200, 0x222300, 0x222400, 0x222500 というアドレスに設定し用いている。

レジスタの割り当ては表 3.12 に示してある。

モジュールは 16 チャンネルの入力それぞれに 3072word の FIFO バッファを持っており、それぞれアドレス 0x00〜0x1F に割り当てられている。ここに CPU からアクセスすると現在の FIFO カウンタの値が取り出されるとともに FIFO カウンタが一つ進み次のバッファを読み出すことができる。

モジュールには acquire モードと measure モードがあり、読み出しには acquire モード、データ収集には measure モードを用いる。この切り替えはレジスタ 0x20 への書き込みにより行われる。

その他、ADC gain,clock 等をレジスタを通して設定できる。

モジュールのレジスタの読み出し・書き込みを行うためには、PC のメモリを VME のアドレスに関連づける。これは vme_master_window_create() の呼び出しにより行う。その後関連づけられたメモリを読み書きすると自動的に VME のレジスタの読み書きが行われる。

また、VME からのデータ読み出しには continuous モード ,3 パルスモードの二つのモードを用意している。continuous モードは 1 パルス毎にデータ読み出しを行うモードで、毎パルスのデータを保存できるが、読み出

しに 20ms 強かかるため、50Hz 等の高い繰り返しではかなりのとりこぼしがある。CAMAC と同様リングバッファに保存するが、サイズは通信により指定できる。一方 3 パルスモードはモジュールに蓄えられる 3 パルスのデータを放電後に読み出すモードで、放電前 3 パルスしか読みとれないもののとりこぼしがない。現在の運転では基本的に 3 パルスモードを用いており、必要に応じて切り替えている。

3.4.11 オシロスコープ

前述のように XTF 放電検出・記録システムでは Yokogawa DL7480 1 台、Tektronix TDS3034B 1 台、Tektronix TDS2024 2 台を用いてモジュレータ・クライストロン・加速管 RF の波形等を記録している。データの読み出しは JAVA のライブラリを用いてネットワーク越しに glcta-00 より行っている。

どちらも速度上毎パルスデータを収集するのは不可能なため放電検出時にトリガをとめてデータの読み出しを行う。DL7480 はヒストリメモリ機能があり、CAMAC の RF 波形と同様に 500 パルス前までのデータを保存している。

データの呼び出しは独立した JAVA のプロセスが担当しており、CAMAC からの放電検知コールバックをうけたコントロールサーバが VME と同時に保存指令を発行している。

3.4.12 画面表示

図 3.49 は、XTF-BDMS のインターフェース画面である。BDMS のコントロールとしては、右の BD Condition ウインドウ上部に測定開始/停止と、現在の波形保存用のボタンがある。下部の放電検出設定、左ウインドウのイベントディスプレイについては後述する。

3.5 放電検出

本節では、本システムの放電検知法について述べる。

図 3.49: XTF-BDMS インターフェース。左がメイン画面で赤印の “BD Condition”ボタンを押すと右のウインドウが現れる。

3.5.1 放電検出の概要

放電検出にはソフトウェアとハードウェアによる方法があり、本システムではソフトウェアによる方法を用いていることはすでに述べた。

ただし、実際にはハードウェアによる放電検出も併用して行っている。これは、ソフトウェアがバグ等によりうまく動かなかった場合に加速管・クライストロン等へのダメージを防ぐためのもので、実際には大きな放電のみを検出する。これは、加速管からの RF 反射パルスを Peak Hold Module に入れ、Fast RF Interlock で Threshold を設けて監視しているものである。この放電検知はインターロック系に作用するため、真空等のインターロックと同様に機能する。インターロックによる停止もソフトウェアの放電条件に適合しシステム上放電として扱われるため問題はない。

ソフトウェアによる放電検知は、前述の通り CAMAC からの RF および γ 線のデータを用いて行われている。放電検知のパラメータはインターフェースを用いて柔軟に変更することができる。

以下、ソフトウェアによる放電検出条件について述べていく。

3.5.2 RF 検出器による放電検出

放電がおきると加速管からの透過出力波形、反射波形に乱れが生じる。これをとらえることで放電を検知することができる。

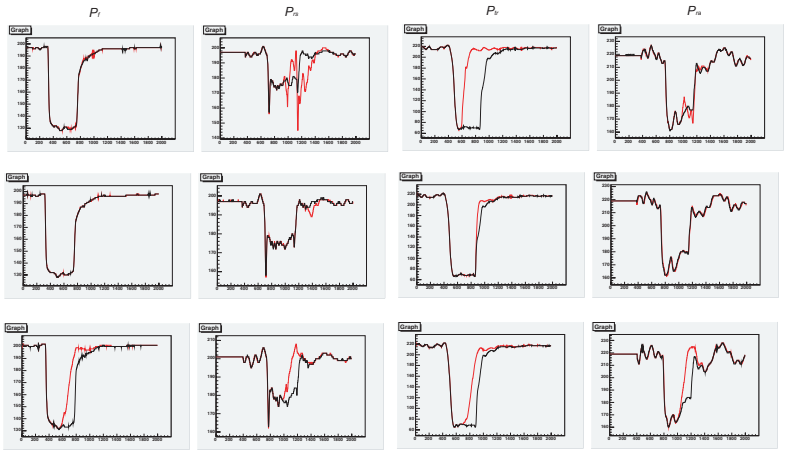

図 3.50: 代表的な放電例。内容は本文参照のこと。

代表的な放電例　図 3.50 に、代表的な放電例を示す。

非放電時の波形 (放電の一つ前のパルス) を黒で、放電時の波形を赤で示している。非放電時の波形は、入射パルスが若干の遅延を経てほぼ同じ波形で出力カプラから出力される。反射もパルス内でほぼ一定である。

1 段目は入力カプラ近くでの大放電の例である。透過パルスは先頭部分以外はほぼなくなり、大きな反射も観測される。

2 段目は出力カプラ近くでの小放電の例である。透過パルスは最後の部分がやや欠けているが違いはわかりにくい。反射はやや遅れて出力されるが、これも自動判別はやや難しい。

3 段目は導波管放電の例である。入力パルスにパルス欠けが見られ、加速管に入力されるパルスがすでに通常と違うことがわかる。透過パルスはほぼ入力パルスと同型になっている。

4 種の放電検出条件　非放電パルスと放電パルスを見分けるために、本システムでは各パルスのピーク強度、積分強度の両方に範囲を設け、その範囲を超える波形が検出された時に放電とみなしている。具体的には、以下の 4 つの判定基準を入力、透過、反射 (対称・反対称) の 4 つのチャンネルそれぞれに設け、そのどれか 1 つ以上が通常パルスの基準を満たさなかった場合に放電と認識する。

1. ピーク強度

   ピーク強度による判定基準は、FADC 波形を一定の時間軸で切り、

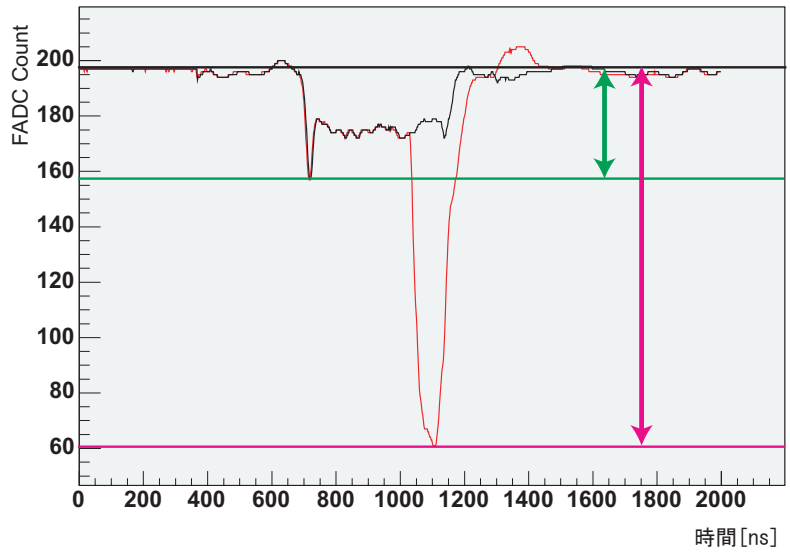

図 3.51: ピーク強度・ピーク強度比による放電検出。ピーク強度では図の赤線が基準値を越えたら放電とみなす。ピーク強度比では緑 (1 パルス前) と赤線とベースラインの差の比を監視する。

その範囲内での最小値、最大値が定められた閾値を超えているかを調べて放電を認識するものである。

これは主に反射波形にかけるための基準で、前出の例では 2 段目のような非放電時よりはるかに大きな反射パルスが出る場合に有効である。

具体的な判定方法を図 3.51 で示す。

ピーク強度判定は最小値、最大値を用いるため波形測定の際の異常値が大きな影響を及ぼしてしまう。実際使用している FADC モジュールは特定のチャンネルで値が大きく飛ぶ点が散見され、これが放電と認識されてしまう問題点があった。そのため現在の判定は、最小値、最大値とも取得した波形中で 2 番目に大きい・小さい点を基準に用いている。これによりほとんどの異常値による放電誤認を抑制できた。1 波形中に 2 点異常値が含まれる可能性も皆無ではないが、圧倒的にイベント数が少ないためオフライン解析で容易に除去できる。

2. 積分強度

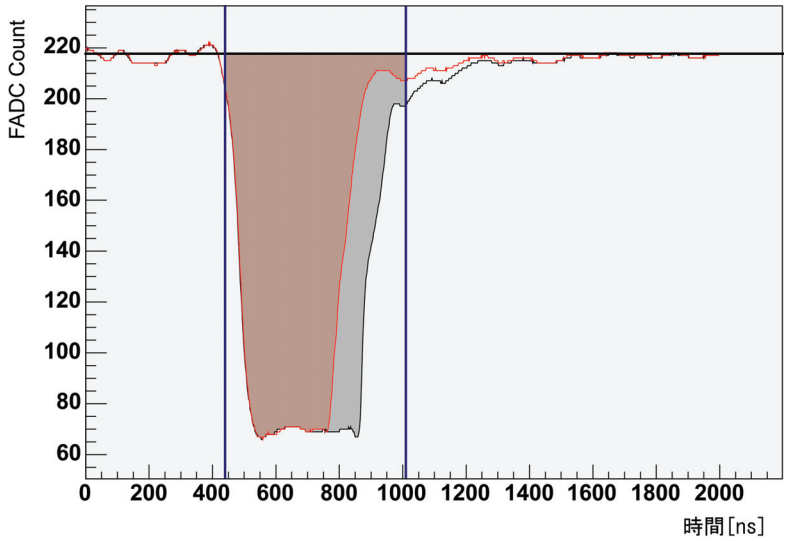

図 3.52: 積分強度・積分強度比による放電検出。積分強度は赤で塗った部分の面積が一定値を越える、または下回ったとき放電と見なす (反射では前者、透過では後者を用いる)。積分強度比は黒部分 (この図では赤部分を含む) の面積と赤部分の面積の比が一定以上ないし一定以下になったときに放電と見なす。

積分強度による判定基準は、ピーク同様 FADC 波形を一定の時間軸で切り、その範囲内でパルス強度を積分したものが、定められた範囲内にあるかどうかを調べるものである。この方法は透過のパルス欠けを判定するのに有効だが反射の増大の検知にも用いることができる。

具体的な判定方法を図 3.52 に示す。

この検出方法は、ピーク強度とは異なりベースラインの位置に大きく依存するため、ベースラインを正確に設定する必要がある。ベースラインの設定は各判定方法に共通であり、以下の項で詳しく説明する。

3. ピーク強度比

   上記二つはいずれも積分値、ピーク値の絶対値 (ADC 値) で判定基準を設定するが、実際には様々な RF パワーで運転を行うため非放電時のパルス高さ等も異なり絶対値での設定は不便である。また、

一度放電等で運転中断した後の再開は低いパワーから徐々に行うため、絶対値の設定ではパワーを上げている途中の放電を見逃す可能性が高い。このため、本システムでは１つ前のパルスとの比を判定基準とする設定も用意している。

ピーク強度比による判定基準では、上２つと同様に時間軸を区切り、前パルスの最小値と今パルスの最小値の比が一定以下、前パルスの最大値と今パルスの最大値の比が一定以上のときに放電とみなす。

この判定基準も、ピーク強度の基準と同様、主に反射の増大の検出に使われている。またピーク検出の基準と同等の異常値除外アルゴリズムを用いている。

具体的な判定方法は図 3.51 に示す。

4. 積分強度比

   積分強度比の判定基準は、これまでと同様時間軸を区切り、前パルスの範囲内積分値と今パルスの範囲内積分値の比が一定の範囲を超えたとき放電とみなす。

   透過パルスの波形欠け、反射の増大どちらにも用いられており、最も使いやすい判定基準である。とくに透過パルスの反射欠けはかなり厳しい条件で用いることができ、放電検知基準としては有力である。また、インターロック等でパルスがなくなったときもこの強度比の変化によって放電と認識する (1 パルス遅れるがリングバッファなので大きな問題はない)。

   積分強度の判定基準と同様、ベースラインに影響されるため慎重にベースラインを設定する必要がある。

   具体的な判定方法は図 3.52 に示す。

立ち上がり時の補正　本システムでは、以上４つの判定基準により RF 波形からの放電検出を行っている。ただし、放電後の立ち上がり時においては、先に述べたようにパワーが徐々にあがるため判定に問題が生じる。この問題を解決するために導入した前パルスとの比による比較も、パワーがなかったり少ないところでは測定系のノイズやジッタの影響が大きくなり適切に働かない問題がある。

このため立ち上がり時には、一定の強度に達するまで判定基準を一時的に OFF にする機能を追加している。この基準 (スタートレベル) は判定

基準の種類によらずすべて積分強度を監視する。積分強度がこのスタートレベルに達するまでは各判定基準は放電検知を行わない。このスタートレベルは判定基準ごとに設定できる。

ベースライン　また、先にも述べたように、FADC にはパルスがないときでも一定の値 (ベースライン) が出力されている。これは温度等によって変化し、その都度キャンセルする必要がある。このベースラインは以下の 3 つのいずれかのモードで設定を行っている。

1. ベースラインなし

   ベースライン補正を行わない。ベースラインは 0(実際は極性が負なため 255) とみなす。現在はほとんど使われていない。

2. 固定ベースライン

   あらかじめ固定の値をベースラインと指定しておく。

   これも温度ドリフト等に応じて値を変える必要があり、現在ではほとんど用いられていない。

3. 自動ベースライン

   時間軸の範囲を指定し、その範囲の平均値をベースラインとする。

   現在ほとんどの場合このベースラインが用いられているが、設定する範囲に信号成分が混入すると正しくベースラインの評価が行えないので注意が必要である。

   基本的にパルスの後方はノイズや反射等で安定しないため、パルスの前方を用いる。

   異常値対策のため最大値、最小値各 1 点は平均値算出に使用しない。そのためこの範囲は 3 点以上必要である。ややパターン化されたノイズが入るチャンネルもあるため、できるだけ長い範囲を設定する。

現在の放電検出条件　現在 XTF で主に用いられている放電検出条件を表 3.30 に示す。

　オンライン放電検出の段階では導波管放電、加速管放電、インターロック等を区別しないため、入力チャンネルには基準を設けていない。反射

チャンネル	波長下限	波長上限	開始条件	下部閾値	上部閾値	ベースライン
P_{rs}-peak	0	1000	0	0	150	なし
P_{rs}-peakRel.	0	1000	4000	0	140%	自動 (0-100)
P_{rs}-integralRel.	0	1000	4000	0	200%	自動 (0-100)
P_{TL}-integralRel.	300	770	12000	90%	150%	自動 (0-80)

表 3.30: XTF-BDMS 放電検出条件

(反対称) も波形が変化するときとしないときがあり一定しないため基準は設定していない。

透過波形については、積分強度比による検出のみを行っている。RF による放電検出としてはもっとも確実である。パルス欠けの基準は前パルスの 90%に設定してある。これを大きくすると低いパワーでノイズを誤認してしまうため注意が必要である。基準を上げるときは同時にスタートレベルを上げる。上限は 150%としてあるが、急に大きなパワーが入ることはシステム上ないので事実上意味はない。

反射 (対称) についてはピーク強度、ピーク強度比、積分強度比の 3 つの判定を複合的に行っている。

ピーク強度は閾値が十分高く設定してあるので、特に大きな放電のとき以外は反応しない。保険的な要素である。ピーク強度比は 200%となっている。入力カプラ付近の放電では加速管での減衰が少なく大きな反射が観測されるためこの基準にかかることも多い。非放電時の反射は入力パワーの 1%程度であり、放電時は 10%以上のこともある。積分強度比は 150%となっている。割合小さな放電まで見分けることができるが出力カプラ周辺の放電は信号が十分小さくなるため反射での検出は難しいものもある。

3.5.3 γ 線検出器による放電検出

放電時には加速管から大量の γ 線が放出される。γ 線による放電検知は、RF のように波形にたいして複雑な基準を設ける必要がなく単純に ADC 値の比較により行え確度も高いため有力である。

非放電時および代表的な放電時の ADC 分布は図 3.26 にすでに示した。

非放電時はほとんど一定の値になっており、パワーを止めてもあまり変わらない。信号はベースラインに比べて十分小さいと考えられる。放電時は非放電時と明らかに区別できる ADC 値がほぼ全チャンネルにわたっ

て得られる。

放電検出は前パルスの ADC 値との差が閾値を超えていないか監視することによって行っている。宇宙線の入射等の外乱の影響を防ぐため、いくつかのチャンネルで同時に ADC 値の変化を観測した場合のみ放電とみなすようになっている。

現在の閾値は 100 カウント、2 チャンネル同時となっている。非放電時のノイズは 10 カウント以内程度であり、放電時は通常 1000 カウント程度の信号が見えるためチャンネル毎の感度差などは考慮しなくても十分な判定基準となりうる。

また、γ 線による放電検出は RF の波形による放電検出によるものよりも感度が高く、RF では検知できない微弱な放電も検出できる。そのような事象の RF 波形を見ると、判定基準にひっかからないほどのわずかな透過パルスの欠け、反射の増大が確認できる。

一応ベースライン (固定) 等の機能もあるが現在は必要ないため利用していない。

当然、導波管の放電やインターロック等の加速管以外の原因による停止では信号が発生しないためこれらを検知することはできない。(導波管と加速管が同時に放電を起こすこともあり、これは当然検知できる)

加速管放電の様子はよくわかっていないため、γ 線の発生を伴わない放電もある可能性がある。このため RF による放電検出も同時に行い、加速管放電で γ 検出器の信号がないものを探しているが、導波管の放電等を除いて今のところ見つかっていない (運転統計は次章に示す)。現時点では γ 線検出器による放電検知は極めて効率的に動作していると推定できる。

3.5.4 放電検出ルーチン

放電検出ルーチンは、CEventAnalyzer::Analyze() の中に組み込まれている。

RF の放電検出は CAnalyzeFADC::Analyze() の 418 行目以降に書かれている。BDCheck[Peak,Integral,PeakRelative,IntegralRelative]() はそれぞれの判定基準に従って判定するルーチンであり、各チャンネルについて 4 種類それぞれを呼び出す。ここだけデータが root 形式でなく C の配列で渡されているのは共同開発者の都合による。

現在の放電検出パラメータは CRFBDParam にまとめて CAnalyzeFADC

が保持しているので、その中から必要な分を抜き出して使用している。相対判定では一つ前のパルスの情報が必要だが、それは CRFBDParam の中に一時的に保持する形を取っている。

γ 線の放電検出は CAnalyzeXRay::Analyze() 内 (135 行目以降) にある。単純な判定なので関数内にそのまま記述している。一つ前のパルスの情報はクラスのメンバ変数に保存している。

3.5.5 ユーザーインターフェース

図 3.53 は、放電検知パラメータを設定するインターフェースである。最上部のコントロールはすでに説明した。下部の BD Condition の枠内が放電検知関係の設定である。

枠内最上部の左側にチャンネル選択ボタンがある。放電検知条件はそれぞれのチャンネルで独立にかけるので、ここでまずチャンネルを選択する。

右上の Reload ボタンは現在の設定を BDMS から読み出しダイアログの値を置き換える。

その下の Auto Scanning は左部に表示されている放電波形の表示コントロールである。

その下にはベースライン設定がある。ベースラインはチャンネル毎に共通パラメータを設定し、各放電条件ごとに ON/OFF を切り替える。

その下部の 4 つのウインドウで、上記 4 つの放電検出条件を設定する。Range は横軸の監視範囲を、Start Level, Lower Limit, Upper Limit が縦軸の検出条件である。Relative の Upper/Lower Limit は%指定になっている。

左側の波形表示エリアには、今入力されている検出条件が図示できるようになっている (Show Line ボタンで表示/非表示を切り替える)。

設定の適用は各項目の Set ボタンで行う。

3.6 オフライン解析

この節では、放電の特性を理解するためのオフライン解析ソフトウェアの概要を説明する。

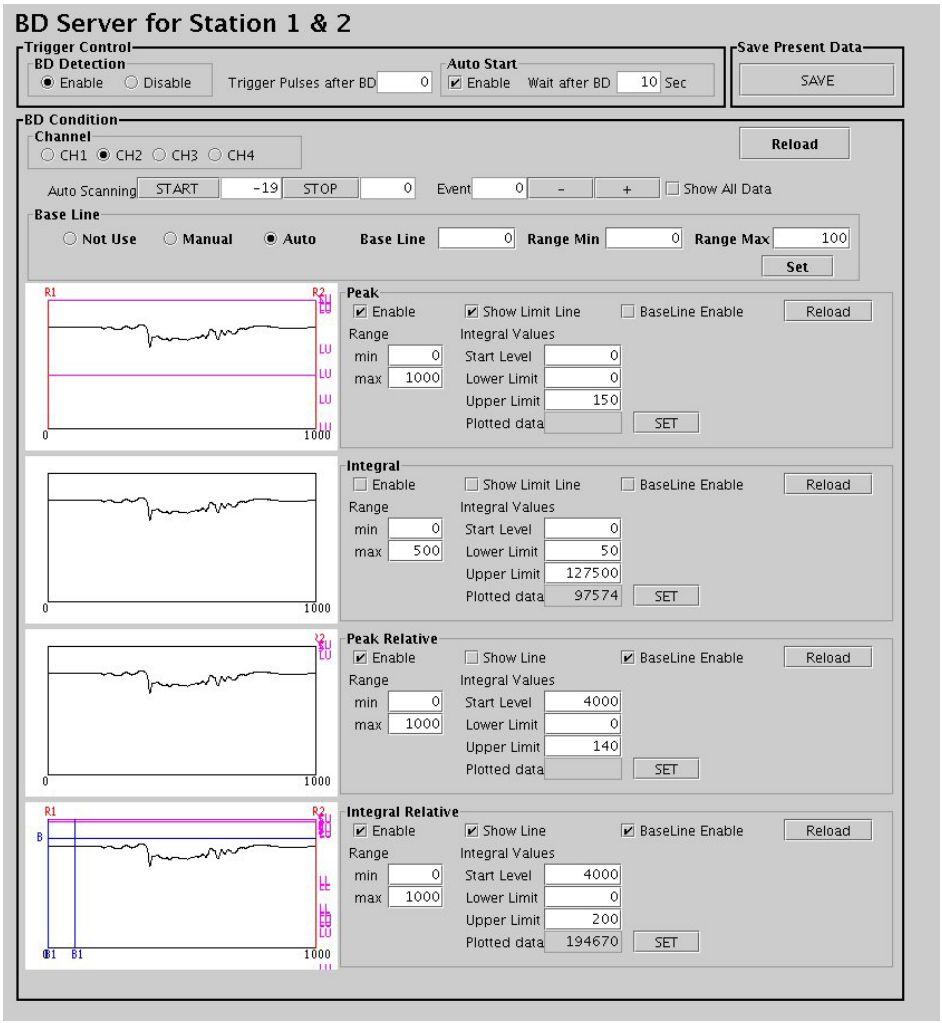

図 3.53: 放電検知パラメータ設定インターフェース

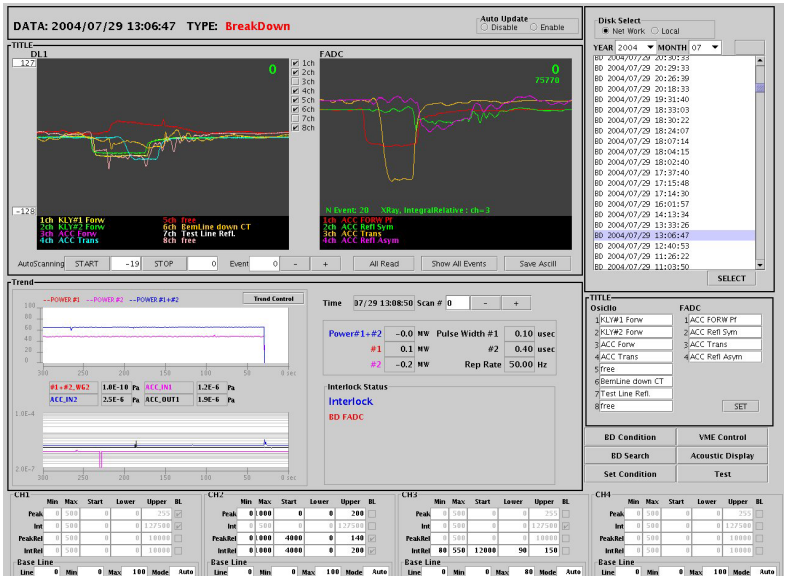

図 3.54: イベントディスプレイ画面

3.6.1 フレームワーク

XTF-BDMS はオフライン解析はイベントディスプレイと root による解析の 2 本立てになっている。

イベントディスプレイは各放電イベントの RF 波形を中心とした情報をインタラクティブに表示しイベントの傾向をつかむのに用いる。JAVA で書かれており、GUI で一通りの操作が可能である。

より詳細な解析や統計的な処理を行う場合は root を用いる。イベントの生データは root 形式で保存してあるので root のクラスにそのまま読み込んで解析が可能である。よく行う解析はマクロで保存し繰り返し使えるようにする。

ただ、root による解析はある程度の root および C++の知識が必要となるため、他の解析ツールも使えるようテキストにエクスポートする機能もある。これはイベントディスプレイ、root マクロ双方から行えるようになっている。

3.6.2 Event Display

図 3.54 は、本システムのイベントディスプレイの画面である。

中央上部のグラフが FADC(右) およびオシロスコープ (左) の波形で、主に RF 関係の波形を表示している。波形は 500 パルス分保存されているので左右の矢印で選択する。自動送りや重ね書き表示も可能である。高速化のためイベントを開いた際に読み込まれるのは放電前 20 パルスのみとなっており、All Read をクリックすると全パルスのデータを読み込む。

下部のグラフは入力パワーや真空度の推移を表示できる。Trend Control ボタンから表示項目を選択できる。

右上のリストはイベント一覧であり、ここから放電イベントを選択する。データは月ごとにフォルダがわかれているので、年月を上部のプルダウンから選択し右のボタンを押すとその年月のイベント一覧が表示される。

最下段はイベント発生時の放電検出条件が表示されている。

3.6.3 root による解析

root による解析は非定型なものが中心となるため、コマンドラインによる操作またはマクロを記述しての解析が中心となる。root のマクロは C++のインタプリタを使っておりクラスライブラリとして root をリンクする際の C++のコードがほとんどそのまま使える。

本システムでは独自のイベントクラスを用意しているのでファイルを開く際にはこのクラスの宣言を読み込む必要がある。宣言の読み込みから一通りのデータ読みだしまでは以下のようなコードになる。

読み出しコード例

```
  // File Open
  TFile *pf = new TFile(pStrFilename);
  pf->SetName("pf");

  TKey *pOldKey = NULL;
  for(int j=nStart;;j++){ // jstart 以降の各 cycle#について、
    int i;

    // まず root TTree を取得
    // Object の"Key"を名前と cycle から取得
    TKey *pKeyTree = pf->GetKey("rf",j);
    // Key がなければ cycle がもうないので読み出し終了
    if(pKeyTree == NULL)break;
    if(pKeyTree == pOldKey)break;
    pOldKey = pKeyTree;

    // Key から Object を取得
    TTree *pTree = (TTree *)pKeyTree->ReadObj();

    // 以下、RF のデータ構造を読み出し
    // データ読み出しバッファ
    CRFWaveform *p = new CRFWaveform;

    // RF 波形データは"rf"という Branch に格納されている
    TBranch *pBranch = pTree->GetBranch("rf");
    // SetAddress() は Tree と要素の読み出し用 Instance を結びつける
    pBranch->SetAddress(&p);

    // RF の各チャンネルの波形格納用配列
    TArrayI arrNormalT,arrBDT,arrNormalRs,arrBDRs;

    // GetEntry() は Tree の要素を SetAddress() で結びつけた Instance にコピーする。
    // 引数は要素の番号。GetEntries()-2 は放電の一つ前のパルス
    pTree->GetEntry(pTree->GetEntries()-2);
    // この段階で p の中身に放電一つ前の RF 波形がロード されている。
    // 配列にコピーする
    arrNormalRs = p->arrData[1];
    arrNormalT = p->arrData[2];

    // 同様に放電パルスも取得する
    pTree->GetEntry(pTree->GetEntries()-1);
    arrBDRs = p->arrData[1];
    arrBDT = p->arrData[2];

    // 以後いろいろ解析
    // ...
  }
}
```

また、放電位置の解析等の定型処理は root のマクロを用意している。

3.6.4 データのエクスポート

放電データをテキストファイルへエクスポートするには、イベントディスプレイを使う方法と root のマクロを使う方法がある。

イベントディスプレイからのエクスポートには、イベントを選択した状態で Save Ascii ボタンを押す。ファイル選択ダイアログが現れるのでファイル名を選択する。

データフォーマットは次のようになっている。

BDMS export データ形式・FADC

```
! FileName
/home/.../b20041213115141.bdmon-camac.bin
! EventNo
0
! NChannel
4
! NData
1000
! Channel, Title
0,ACC FORW Pf
1,ACC Refl Sym
2,ACC Trans
3,ACC Refl Asym
! H Scale
! V Scale
!Number, ch1, ch2, ch3, ch4
0,211,213,218,220
1,211,213,218,220
2,211,213,219,220
3,211,213,220,220
...
998,211,213,219,220
999,211,213,220,220
<END>
```

BDMS export データ形式・オシロスコープ

```
! FileName
/home/.../b20041213115141.dl1
! EventNo
0
! NChannel
8
! NData
1000
! Channel, Title
0,KLY#1 Forw
1,KLY#2 Forw
2,ACC Forw
3,ACC Trans
4,BeamLine up FC
5,BemLine down CT
6,Test Line Refl.
7,BeamLine down FC
! HScale
0.0
! Channel, VScale
0,0.0
1,0.0
2,0.0
3,0.0
4,0.0
5,0.0
6,0.0
7,0.0
!Number, ch1, ch2, ch3, ch4, 5ch, 6ch, 7ch, 8ch
0,0,0,-1,-1,10,-1,-1,2
1,1,-1,-1,0,8,0,-1,2
2,0,0,-1,-1,9,0,-1,2
3,1,0,-1,0,7,0,-2,2
...
998,0,0,0,-1,4,0,0,1
999,0,0,-1,-1,4,-1,0,1
<END>
```

root からのエクスポートは ExportToCsv() というマクロを用いる。

ExportToCsv() 使用例

```
[glcopr@glcta-03 glcopr]$ . /cern/root/start-root.sh
[glcopr@glcta-03 glcopr]$ root -l
// ln -s /mnt/nas/work/breakdown/vme/analysis/suehara/ ~/vme/ してある
// オブジェクト定義ファイルをロード (Compiled mode)
root [0] .L ~/vme/streamers.h+
// 書き出しマクロファイルをロード
root [1] .L ~/vme/041026.C
// ファイルを開く
root [2] TFile f("/mnt/nas/data/rawdata/2005/01/20050113.vme.root")
// ファイル内オブジェクト一覧を表示 目的のイベントを見つける
root [3] f.ls()
TFile**         /mnt/nas/data/rawdata/2005/01/20050113.vme.root
 TFile*         /mnt/nas/data/rawdata/2005/01/20050113.vme.root
  KEY: CAcousticParam   AcousticParam;1
  KEY: TTree    acoustic;80     Acoustic signal at /mnt/.../b20050114135454.vme
  KEY: TTree    acoustic;79     Acoustic signal at /mnt/.../b20050114134332.vme
  KEY: TTree    acoustic;78     Acoustic signal at /mnt/.../n20050114134236.vme
...
  KEY: TTree    acoustic;45     Acoustic signal at /mnt/.../b20050113210354.vme
  KEY: TTree    acoustic;44     Acoustic signal at /mnt/.../b20050113203132.vme
  KEY: TTree    acoustic;43     Acoustic signal at /mnt/.../b20050113191917.vme
...
  KEY: TTree    acoustic;4      Acoustic signal at /mnt/.../b20050113112705.vme
  KEY: TTree    acoustic;3      Acoustic signal at /mnt/.../b20050113112506.vme
  KEY: TTree    acoustic;2      Acoustic signal at /mnt/.../b20050113092940.vme
  KEY: TTree    acoustic;1      acoustic signal
// cycle 番号指定で目的のオブジェクトを TTree に読み込む
root [6] TTree *p = f.Get("acoustic;44")
// 書き出しマクロを起動 引数は TTree とファイル名
root [7] ExportToCsv(p,"sample.csv")
Entry 0
Entry 1
Entry 2
// 書き出し終了
```

データ形式は以下のようになっている。

BDMS export データ形式・音響センサ

```
event, time, 1, 2, ..., 64
0, 0, 2048, 2053, ..., 2053
0, 1, 2047, 2057, ..., 2057
...
0, 1023, 2046, 2061, ..., 2048
1, 0, 2047, 2045, ..., 2048
...
2, 2023, 2047, 2055, ..., 2051
```

3.7 まとめ

XTF 放電検知・記録システムの概要について説明した。

本システムは RF,γ 線, 音響のデータを複合的にデータ収集、放電検知、保存、RF パワー系のフィードバックまで統一的に行うシステムとして、

定常的なデータ収集が可能となっている。この 1 年ほど改良を加えながら運用されてきており、安定度も高まってきている。

今後しばらく現状の性能を維持しつつ運転が続けられる予定で、システムの完成度が高まったことで次にインストールされる KX02 では今まで以上の放電データが蓄積でき、放電のメカニズムの解析、加速管の性能向上に役立つと期待されている。

第4章　データ収集の経過

この章では、XTF のこれまでの運転経過と、蓄積された放電データについて述べる。

4.1　運転の手順・方法

XTF の運転は基本的に無人で継続できるが、安全上の理由から定期的に監視をする必要があり、現在は 24 時間運転は行っていない。従って毎日マニュアルに従い立ち上げ、立ち下げを行う必要がある。ここでは現在の運転手順を記しておく。

4.1.1　立ち上げ、立ち下げ

立ち上げ　2004 年 7 月の KX01 継続運転の際のマニュアルから参照している。

1. シールド内待避・扉閉
2. 放射線安全システム制御盤 キーにて準備完了 (緑色) とする
   シールドルームの鍵を制御盤に差し込むと運転が可能となる。
3. 各種電源・冷却水の確認
   モジュレータの LV 系統、測定系、真空系は常時電源 ON となっている。
4. クライストロン収束ソレノイド電源を立ち上げ
   二台あるクライストロンにそれぞれ三台のソレノイド電源がある。
   それぞれ 440A,640A,740A の電流値にセットする。
5. 加速管冷却システムの電源を投入する。

6. モジュレータ操作盤にて、Esdc を 25kV にセットし、インターロックをリセットした後 HV を投入する
   Esdc は通常は 33kV〜36kV 程度で運転を行うが、立ち上げ時は暖気運転のため低い電圧から開始する。

7. コントロールソフトウェアが動作していることを確認

8. RF パワー制御用の Up/Down モジュールの値を 1.0V に下げ、運転開始
   Up/Down の値は RF Processing のインターフェースから設定できる。

9. クライストロンの Vk、RF 出力パワー、パルス幅、繰り返し周波数が設定通りであることを確認する。

10. しばらく安定に動作すれば、Up/Down を 7V 程度まで徐々にあげる。
    モジュレータの電圧が低いため、20MW 程度で RF パワーが上限に達する。

11. 上記の状態で 5 分以上運転して安定であれば Up/Down を 1.0V に下げ、Esdc を設定値まで上昇させる
    Esdc はコントロールルームからは行えないので、モジュレータ制御盤で行う。

12. 再び Up/Down を徐々に上昇させ、所定のパワーを供給する。

立ち下げ

1. 最終の運転パラメータを記録しておく

2. Up/Down を 1.0V に下げる

3. 放射線安全システム制御盤のキーを回し、Ready を落とす

4. HV を落とす

5. ソレノイド電源を落とす

4.1.2 計算機類の起動

コントロールソフトウェア、放電検知・記録ソフトウェアは通常は常時起動のため、定常運転中に再起動が必要になることはない。ただし、停電や PC のトラブル等で再起動が必要になることがある。その場合の手順を整理しておく。

計算機の起動 glcta-00〜04,glcta-ccnet,glcta-vme1,glcta-sv1 が起動していない場合は起動する。

glcta-00〜04 は通常の PC のため PC の電源ボタンで起動する。

glcta-ccnet1,glcta-vme1 はそれぞれ一体のクレートの電源投入で起動する。

周囲の NIM 電源が入っていない場合は併せて電源を入れる。

glcta-sv1 は PC と CAMAC の電源をそれぞれ投入する。

ソフトウェア類の起動

コントロールソフトウェアの起動 XTF コントロールソフトウェアの制御部はすべて glcta-sv1 上で動作している。

glcta-sv1 にログインし、以下のコマンドを入力する。

XTF コントロール起動

```
$ su
password: xxxxxxx
# cd /home/glcopr/cc7x00
# make install
# exit
$ all_glcta_setup_sv1
```

放電記録・検知ソフトウェアの起動 放電検知・記録ソフトウェアは、glcta-00,glcta-ccnet,glcta-vme1 上でそれぞれ動作する。ソフトウェアの起動にはそれぞれの計算機に glcopr でログインし、次のコマンドを実行する。

XTF コントロール起動

```
glcta-00 : glcta bd12server start
           glcta bd12controller start
 glcta tdsserver start
(上記 3 つのプロセスを別のコンソール上で実行する)

glcta-ccnet1 : ./bdmonitor start

glcta-vme1 : ./bdmonitor start
```

4.1.3 RF パラメータの設定

RF 源のパラメータ (パワー、パルス幅、繰り返し周波数) を変更するには以下の手順を行う。

パワーの設定 RF パワーの設定は、モジュレータの電圧、RF attenuator(Up/Down モジュール) の二つで行う。モジュレータの電圧は極力一定値とし、Up/Down モジュールの操作でパワー変更を行う。ただし、可能な最高のパワーで運転する場合はモジュレータの電圧を限界まで上げる必要がある場合がある。

Up/Down コントロールは RF Processing のインターフェースから設定できる。Processing の目標値に設定すると、そこまで徐々に Up/Down の値を上昇させていく。下げる場合は目標値がそのままセットされる。

モジュレータの電圧はモジュレータ制御盤から行う。Esdc を設定すると、そこを目標として Edc の値が変化する。Edc と Esdc は完全に一致しないので、目的の Edc になるよう Esdc を調整する。

パルス幅の設定 パルス幅の設定はクライストロン#1 コントロールラック最上段の Pulse&Phase Modulator モジュール (図 2.8 の左端のモジュール) を用いる。

この Pulse Width の値を設定すると、パルス幅が変わる。これも実際のパルス幅と完全に一致しないので、オシロスコープの波形を見てパルス幅をあわせる。

繰り返し周波数の設定 繰り返し周波数の設定には、モジュレータ制御盤の下部の NIM ラック上の Sync モジュール (図 2.18) を用いる。押しボタンで周波数が設定できる。

4.2 運転履歴と設定の変遷

XTF では、2004 年 4 月から、KEK 製 KX01 加速管を設置し、プロセシングおよび高電界試験を行ってきた。高電界試験はパルス幅 400ns,600ns,800ns について、50〜63MW のパワーで行った。試験中にいくつかトラブルがあり、常に安定して運転が行われたわけではないが、2004 年 7 月までの 4ヶ月程度の運転を行った。XTF のこれまでの運転履歴を図 4.1 に示す。

期間	内容
～2003/12	運転準備
2004/1～2004/3	SLAC 製 53cm 加速管を用いて測定系のテスト・チューニング
2004/4/6～	KX01 運転開始 50MW,50Hz パルス幅を 50ns,100ns,200ns とあげていく。
2004/4/26～	400ns 運転開始
2004/5/18～5/27	5/28 55MW,400ns,50Hz 定常運転 (24 時間運転)
2004/5/19	モジュレータ内部のコンデンサ 1 台故障。以後 Flat Top がやや短くなる
2004/6/7～6/14	600ns,55MW,50Hz 運転
2004/6/15	TWT アンプ 2 台故障。1 台の出力を分けて使うことに
2004/6/29～7/2	運転再開。600ns,55MW,50Hz
2004/7/5～7/9	800ns,55MW,50Hz 運転
2004/7/12	加速管上流の CT から真空がリーク
2004/7/14～7/30	400ns,63MW,50Hz 定常運転 (12 時間運転)
2004/8	夏期 Shutdown シールド扉交換、加速管冷却改善
2004/9/2～9/25	γ 線検出器のテストのため設定をいろいろ変えて運転
2004/10/1～	RF コンポーネントテストラインにパワー供給。加速管へは供給せず。

表 4.1: XTF の運転履歴と大きな設定変更・装置故障等の履歴

日付	内容
2004/4/6	KX01 運転開始。放電検出は RF 波形のみ。放電検出条件のチューニング。
2004/4/21	50Hz 移行により、VME の収集モードを 3 パルスモードに変更
2004/6/2	CAMAC 放電検出を新バージョンに移行
2004/6/9	CAMAC でパルス統計情報の収集開始。γ 線による放電検出を開始
2004/6/28	CC7700 から CC/NET に移行。RF 波形の時間軸を 500 点から 1000 点に増加。
2004/6/9〜29	積分強度比の放電検出がバグによりかからない状態になっていた
2004/7/2	音響データ root ファイルフォーマット変更
2004/7/14	RF データ root ファイルフォーマット変更
2004/7/15,16	落雷による瞬停。以後 7/21 まで γ 線 PMT の HV が OFF
2004/11/2	VME バイナリファイル書き出しのバグを修正。定常 (1 分に 1 パルス) 記録開始

表 4.2: XTF-BDMS 更新履歴

また、KX01 の運転は XTF ソフトウェアの動作確認・改良を含めて行った。そのため、運転中にも BDMS や XTF コントロールの修正・改良が行われた。図 4.2 には BDMS の改良・設定変更の履歴を示す。

現在オフライン解析で用いているマクロで読み込めるデータは VME が 7/2 以降、CAMAC が 7/14 以降のものである。それ以前のデータはコンバータを整備していないため現在 root ファイルを簡単に読み込むことができない。

イベントディスプレイ用のバイナリファイルのフォーマットは変わっていないため以前のものも読み込める。ただし音響データに関しては読み出しインターフェースがなかったため書き出しもデバッグを行わないまま放置してあり、2004/11/2 の修正以前のものは読み出すことができない。7/2 以降のものは root で読み込めるため簡単にコンバート可能である。

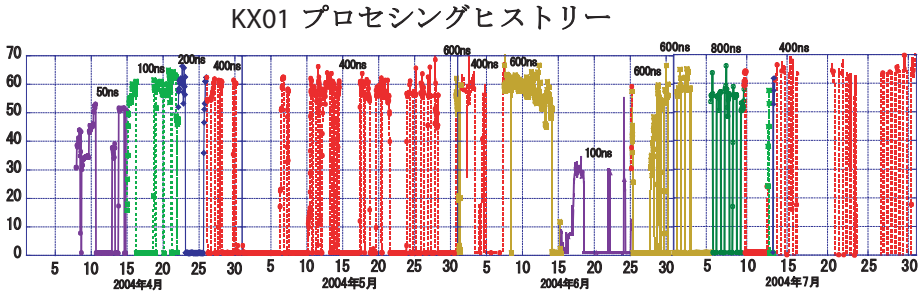

図 4.1: KX01 Processing History(2004/4〜2004/7)。グラフのドットは 1 時間ごとの最高パワーを示している。パルス幅は色で示している。

Run	期間	Power	Pulse 幅	Pulse 数	Data 数	解析	備考
-	4/ 6〜5/17	-	-	-	-	-	立ち上げ
1	5/18〜5/28	∼50MW	400ns	1.4×10^7	110	頻度のみ	-
-	5/31〜6/ 4	-	-	-	-	-	調整
2	6/ 7〜6/14	∼52MW	600ns	2.3×10^7	261	頻度のみ	-
-	6/15〜7/28	-	-	-	-	-	調整
3	6/29〜7/ 2	∼52MW	600ns	6.8×10^6	184	頻度のみ	-
4	7/ 5〜7/ 9	∼53MW	800ns	6.8×10^6	277	頻度のみ	-
5	7/14〜7/30	∼63MW	400ns	1.3×10^7	393	すべて可	-
-	9/ 2〜9/25	-	-	-	-	-	γ 線調査

表 4.3: RUN #の定義および各 RUN でのパラメータ・パルス数・収集データ数。収集データ数は BDMS で放電が検出されデータ収集が行われたものを全て含み、加速管放電だけでなく、導波管放電、インターロック、ソフトウェアの不正、設定ミス等のデータ数の合計である。Pulse 数は 40MW 以上の Power が入った秒数から計算。

4.3 Processing History と RUN

2004/4〜2004/7 の KX01 の Processing History を図 4.1 に示す。

XTF の運転は、放電頻度等の解析のため、RF パワーおよびパルス幅をある程度固定して行い、データを収集してきた。以下、パラメータを固定して安定運転を行った期間を RUN とし、RUN #を定義する。

RUN # の定義および期間・パラメータ・パルス数・収集したデータ数を表 4.3 に示す。現在詳細な解析を行っているのは RUN 5 のデータのみである。今後コンバータを整備しそれ以前のデータについても使える形にする予定である。

第5章　放電データの解析

本章では、3 章で述べた放電検知・記録システムのデータを用いて行った KX01 の放電解析の方法および結果を示す。

5.1　解析の概要

放電解析の目的は、加速管の性能評価および加速管の放電特性を調べ、それを元に放電頻度の少ない加速管を製作することであった。

X-band 加速管は、KEK と SLAC で 10 年以上協力して開発がすすめられてきた。高電界試験設備は SLACが先行して設置しており、加速管の放電についても、SLACで重点的に研究されている。本システムでも SLAC の研究成果を参照しつつ KX01 の放電解析を行っている。

具体的には、以下の放電特性データを調べている。

- 放電頻度

  実際に運転に用いる加速管では、運転パラメータにおける放電頻度が十分低いことが第一条件となる。そのため放電の頻度は加速管の性能を表す最も重要なパラメータといえる。よって、放電頻度を詳細に評価することが放電特性を調べる第一歩である。

  放電頻度を調べるのは放電検知が正常に機能しさえすれば可能なため解析としては易しいと言えるが、加速管の放電頻度は後述するように加速管に印加する電場強度、RF パルス幅により大きく変化し、また運転時間の経過にともなう変化もある。さらに、放電が連続して起こる現象等もあり、これらを考慮しつつ総合的に放電頻度を評価する必要がある。

  また、放電は運転に影響の少ない微少なものから大きなものまであり、どの程度の規模の放電を捉えるかで頻度が大きく変わってしまう問題もはらんでいる。

本解析では、KX01 の放電頻度について、その RF パワー、パルス幅、繰り返し周波数、運転時間の経過による変化等を調べ、また放電の時間的分布 (連続して起こるか) についても考察を行う。

- 放電位置

  放電位置の解析も、放電の少ない加速管を製作する上で重要である。放電位置の分布を知ることで、加速管の放電が起きやすい部位を知ることができるためである。加速管によっては入力、出力カプラ周辺等特定の部位に放電が集中することがあり、この場合その場所の構造を改良することで放電頻度を抑え、より高い電場に耐えられる加速管を製作できる可能性が高い。

  また、放電による加速管のダメージを調査する上でも、放電位置の分布を知っておくことが重要である。運転終了後に加速管内部の状態を調査する際に、放電分布との関係を知ることができる。

  本システムでは、RF、γ 線、音響のすべての検出器で放電位置に感度があることが予定されており、放電位置の詳細な解析が可能となる。

- 放電メカニズム

  加速管内で放電がどのように起きるかを知ることが、最終的に放電の少ない加速管の条件を知る上で重要である。

  加速管の放電メカニズムには仮説が提唱されているが、実際に測定により確かめる必要がある。本システムでは 3 種類の測定器の情報を複合的に用いて、総合的に加速管の放電メカニズムに迫っていく。

  具体的には、放電位置と時間、Missing Energy の相関、γ 線のエネルギー分布、位置分布等から放電の発生とその広がりについての情報が得られると期待できる。

本解析では、以上 3 つの点を中心に得られたデータの解析を行い、KX01 の放電特性、および加速管の一般的な放電メカニズムについても考察を行う。

5.2 放電の頻度

本節では、KX01 の放電頻度に関する結果を示す。

5.2.1 放電頻度と加速管の特性

加速管の放電頻度は、同じ加速管でも加速管に印加する RF 強度、パルス幅等に大きく依存しているため、さまざまな条件で放電頻度を測定し総合的に特性を評価する必要がある。以下、本解析で評価するパラメータについて概説する。

RF 強度　加速管放電のメカニズムについては解明されていない点が多いが、DC での放電現象の研究結果より、電界放出電子 (field emission electron)が放電のトリガーの役割を果たしていると考えられている。電界放出とは、RF を印加したときに空洞表面の金属壁からトンネル効果により電子が空洞内に放出されるものであり、その電流密度 (Fowler-Noldheim(FN) 電流密度) は電場強度に対して以下のような関係にある。

$$j_F = \frac{6.02 \times 10^{-12} \times 10^{4.52\phi^{-0.5}} E_{eff}^{2.5}}{\phi} \exp\left(-\frac{6.53 \times 10^9 \phi^{1.5}}{E_{eff}}\right) [\mathrm{A/cm^2}] \tag{5.1}$$

ϕ は仕事関数 [eV] で銅では 4.5〜5[23],E_{eff} は実効表面電場強度 [V/m] で、一般に加速勾配の数十〜数百倍である (加速管の構造、Processing の状況などにより変化する)。

この FN 電流密度と放電頻度の関係ははっきりとはわかっていないが、FN 電流密度が電場強度に $\exp(-1/E)$ という非常に強い依存性をもっているため、放電頻度と電場強度の関係も指数的であると予測される。

電場強度と放電頻度の関係に関するデータ収集は、SLAC の X-band 高電界試験 (NLCTA:Next Linear Collider Test Accelerator) で長期にわたり行われてきた。図 5.2 は主に 2003 年に試験が行われた加速管のデータである。電場強度に対して指数関数的に放電頻度が増えていく様子が確認されている。

パルス幅と放電頻度　パルス幅と放電頻度の関係性については、想定するモデルによって大きく異なり、まだ詳しく解明されていない。(たとえば、[25] には放電頻度は pulse 幅の 3 乗に比例する、とある)

NLCTA のデータでは、pulse 幅に対して指数的な応答を示すという結果が示されている。ただしデータ点数が少なく、さらなる検証が必要と考えられる。(図 5.3

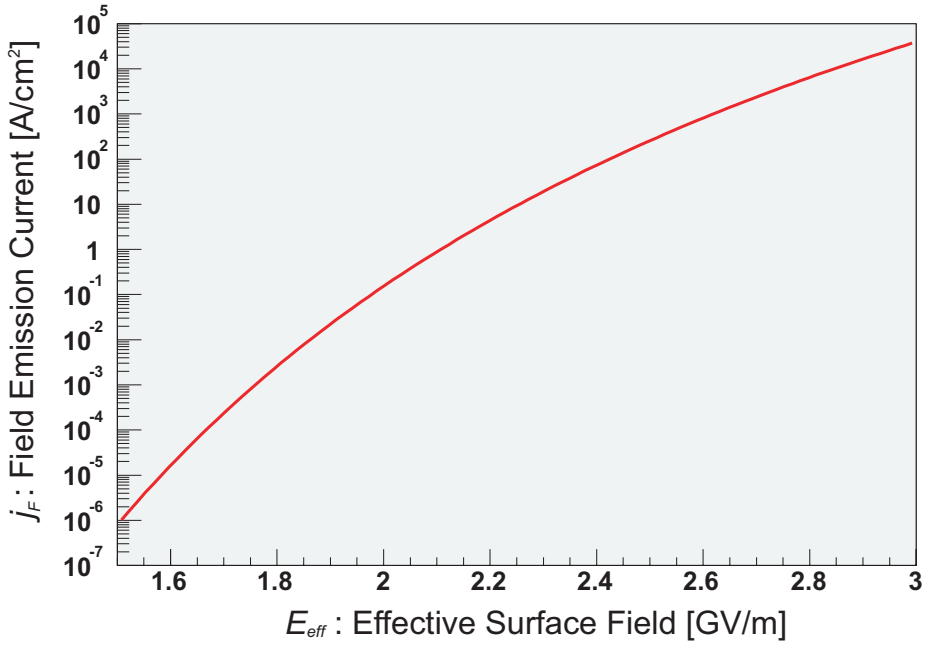

図 5.1: Fowler-Noldheim 電流密度と電場強度。exp(1/E) の関係が支配的である。

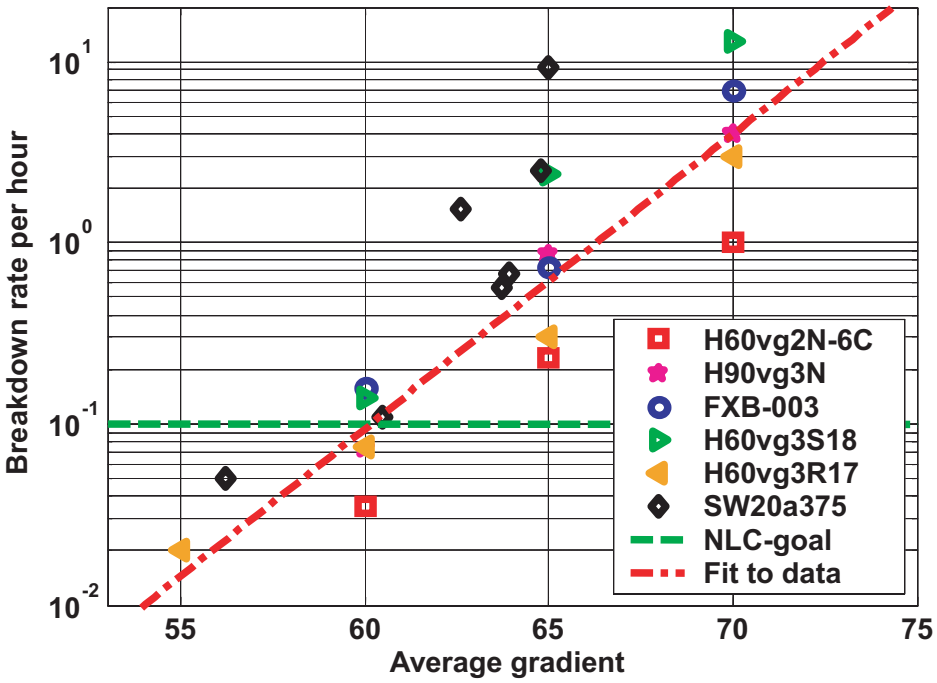

図 5.2: NLCTA における放電頻度と電場強度の関係性 [24]

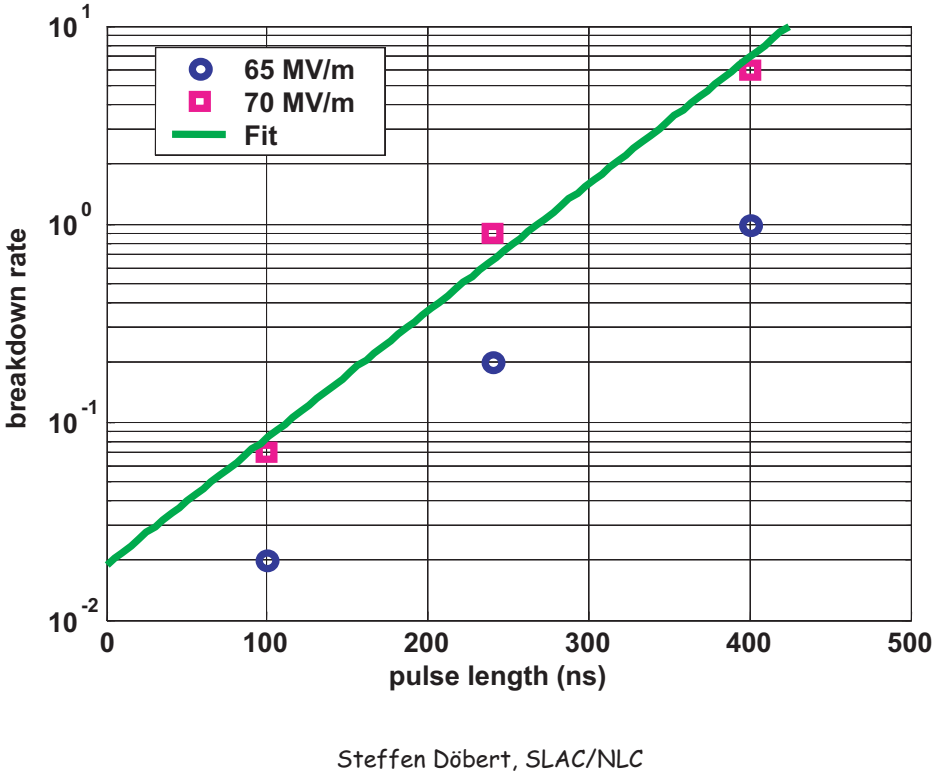

図 5.3: NLCTA における放電頻度とパルス幅の関係性 [24]

Processingによる放電特性の改善　加速管の特性として、製作、インストール直後は低いパワーでも放電がおきるが、しばらくRF供給を続け、放電を起こしていくうちに、徐々に高い電力を供給しても放電がおきにくくなる現象がある。このため、加速管は運転に使用する前に一定時間のProcessingを行う必要がある。

Processingによる放電特性の改善は、主に加速管の内壁に付着した不純物の蒸発、また放電により加速管内壁の凹凸部が溶解・蒸発し平滑度が増すこと等によると考えられている。この特性の改善スピードは加速管により異なり、短期間かつ一定期間で安定にProcessingされるかどうかは加速管の重要な性能要件である。

本解析では、KX01の放電頻度の変化を検証し、Processingの効果の評価も行う。

5.2.2　放電の定義

一口に加速管の放電と言っても、その特性はそれぞれでかなり異なる。図3.50にXTF-BDMSで捉えられた代表的な放電データの例をいくつかあげた。

以下のデータは、RFについては表3.30に示した検出条件、γ線については3.5.3節で述べたように、2ch以上の検出器で前パルスより100カウント以上多いADC値を検出した場合を放電検出条件として捉えられた放電に関するものである。検出条件によって放電頻度は大きく変わり、またその傾向性も変わる可能性がある。

5.2.3　加速管放電イベントの選択

加速管の放電頻度を解析するためには、まず加速管の放電イベントとそれ以外(導波管、インターロック)による放電検知を区別する必要がある。

本解析では、二つの方法でこの区別を行う。

- RF波形による方法

  先に述べたように、加速管の放電では入力RFの波形は変わらず出力波形、透過波形が変化する。それに対し導波管の放電では入力RFに欠けが見られ、またインターロックによる放電検知では最終パルスの入力RFがなくなる。

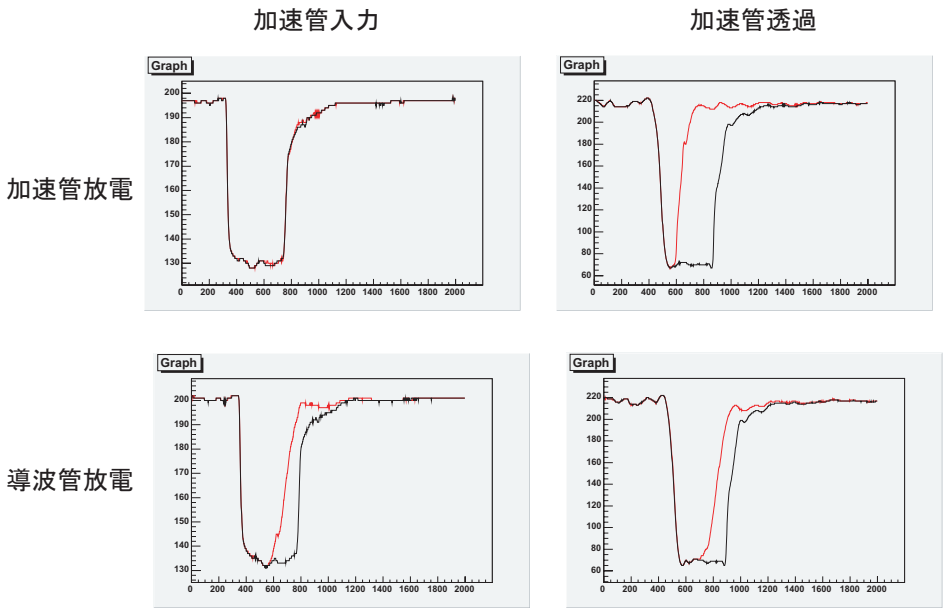

図 5.4: 加速管放電と導波管放電の RF 波形の違い。導波管放電は入力 RF に欠けが見られる。

従って、入力 RF の波形変化が少ないという条件で加速管の放電を見分けることが可能である。ただ大放電では反射 RF がやや入力 RF のポートに漏れ出てくるため、やや波形が変化するので注意が必要である。

- γ 線による方法

  γ 線は基本的に加速管の放電以外では反応しないため、γ 線で有意な信号が検出されているものをすべて加速管放電イベントとみなせる。

  検出条件としては放電検知システムと同じく、前パルスより 100 カウント以上とした。1 チャンネルだけの場合も加速管放電に含める (ただし、そのようなイベントはなかった)。

二つの結果が一致しないイベントについては、RF 波形等を実際に見て判断した。

この区別により、加速管放電イベント数は表 5.1 のようになった。

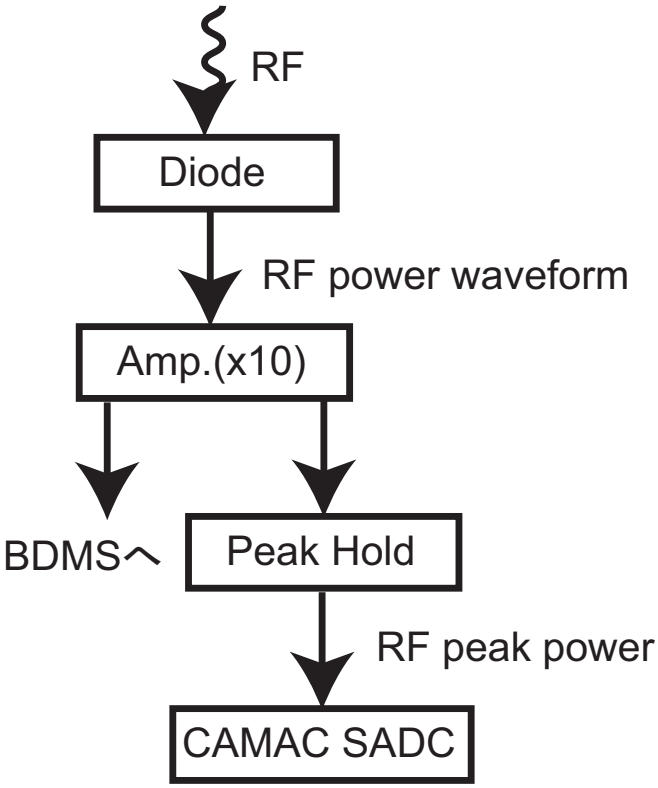

図 5.5: XTF コントロール RF パワー測定

5.2.4 各 Run での入力 RF パワーの評価

放電頻度の RF パワーに対する依存性を調べるには、各 Run における入力 RF パワーを詳しく評価する必要がある。

入力パワーは、XTF コントロール系で記録されている。図 5.5 に、コントロール系での RF パワー測定の Setup を示す。データの収集はコントロール系の CAMAC SADC により 1 秒周期で行われている。

この 1 秒ごとの入力 RF パワーの分布を各 Run でとったのが図 5.6 である。このとき、2004/7/13 以前のデータは Diode の Calibration が正確でない (線形近似になっている) ため、変換式を用いて変換している。

$$W_{in} = 1.090 \times 10^{-3} \times V_{out}^2 + 4.154 \times 10^{-2} \times V_{out} \tag{5.2}$$

$$W'_{in} = 0.33974 \times V_{out} + 1.0194 \tag{5.3}$$

$$W_{in} = 1.3595 \times 10^{-2} \times W'^2_{in} + 0.1407 \times W'_{in} - 2.9748 \times 10^{-2} \tag{5.4}$$

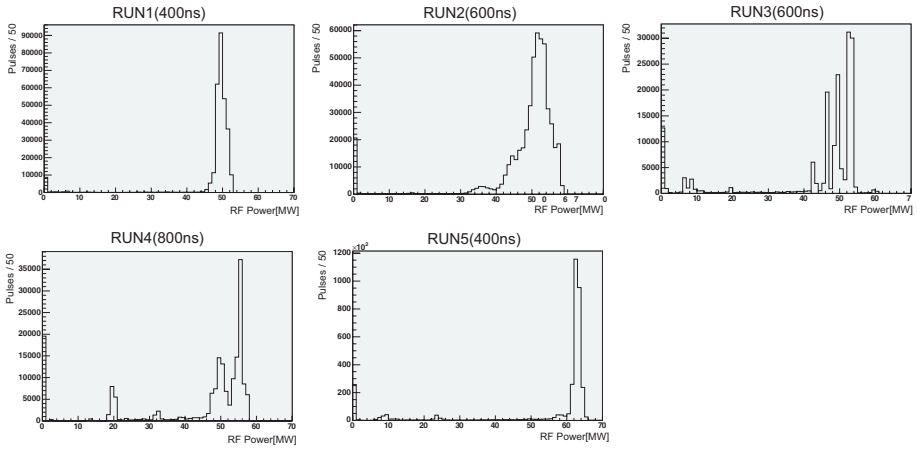

図 5.6: 各 RUN の入力 RF パワー分布。どの RUN もおおむねピークパワー付近に多くのパルスが集中しているが、ややばらけている RUN もある。パワーが 0 のところに分布があるのは、放電後パワーがなくなった時間や機器の調整の時間を含むためである。

(W_{in}:正しい入力電力 [mW],W_{in}:7/21 以前の表示用入力電力 [mW],V_{out}:出力電圧 [mV])

ヒストグラムを見ると、各 Run とも規定の RF パワー前後にピークを持つ分布となっているが、0MW にも分布があり、また規定の RF パワーより低いところにも少し分布がある。0MW は放電後のデータ保存やインターロック等によりパワーが入ってなかった時間を、低いパワーは放電後のパワー復帰中のものと思われる。放電頻度は印加 RF 電圧に exponential で依存性があることを考慮して、各 Run でパワー下限値を設定し、これ以下のパワーの時はパルス数にカウントしないことにする。また、下限値より RF パワーが大きいパルス (有効パルス) の全パルスの平均値を、その Run の印加 RF パワーとみなすことにする。

表 5.1 の Pulse 数,Power は、各 RUN について 40MW を下限値としてある。

5.2.5 Run 内での放電頻度の変化

各 Run の間の放電頻度の比較をする前に、Run 内での放電頻度の変化について検証する。同じ Run 内では基本的に RF パワー、pulse 幅等の設定が同じなため運転経過による Processing の効果が検証しやすい。

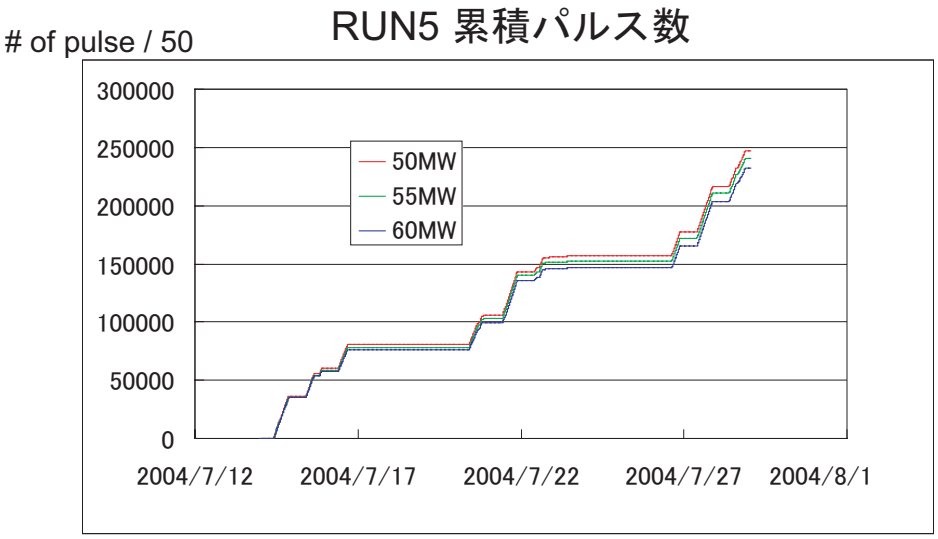

図 5.7: RUN5 入力 RF パルス数の推移

Run	Power	Pulse 幅	Pulse 数	加速管放電	頻度
1	∼50MW	400ns	1.4×10^7	7	0.090 BD/h
2	∼52MW	600ns	2.3×10^7	104	0.82 BD/h
3	∼52MW	600ns	6.8×10^6	35	0.93 BD/h
4	∼53MW	800ns	6.8×10^6	97	2.57 BD/h
5	∼63MW	400ns	1.3×10^7	57	0.79 BD/h

表 5.1: 各 RUN での加速管放電数および放電頻度。頻度は 50Hz での値。

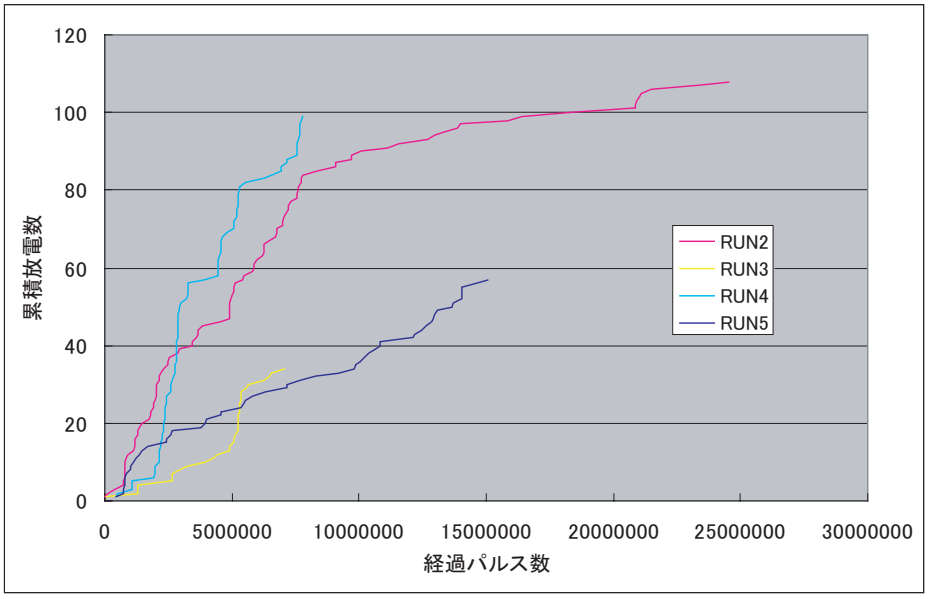

図 5.8: 各 RUN での累積放電数の推移

Run 内での放電頻度の変化を見るにはある程度の放電数が必要なため、放電数の少ない (7 回)Run1 を除いた 4 つの Run について放電頻度の変化を見る。

図 5.8 でわかるように、Run2 においては顕著な放電頻度の低下が見られるが、Run3〜5 については放電頻度が乱高下し、低下傾向ははっきりとは見られない。Run4 では初期の 6 時間ほどは極めて放電頻度が少なく、その後上昇している。このように、短期間での放電頻度の継続的な低下を見るのは難しいことがわかる。また、顕著な放電頻度の低下がみられる Run2 においても、その変化は連続的ではなく、ある 1 点ないし 2 点を境に急激に変化しているように見られる。

放電頻度が RUN 内で一定であれば、ある放電と次の放電の間の間隔の分布 (以下、放電間隔分布とよぶ) は指数関数的な減少を示すはずである。逆に放電間隔分布が指数関数からどれだけはずれているかを調べることで、放電頻度がどの程度一定であるかを調べることができる。

図 5.9 は、各 RUN における、放電間隔分布の指数関数でのフィットである。Fitting は各パルスで等確率で放電が起きる場合の式、

$$n(t) = \frac{NB}{\tau} \exp(-\frac{t}{\tau}) \tag{5.5}$$

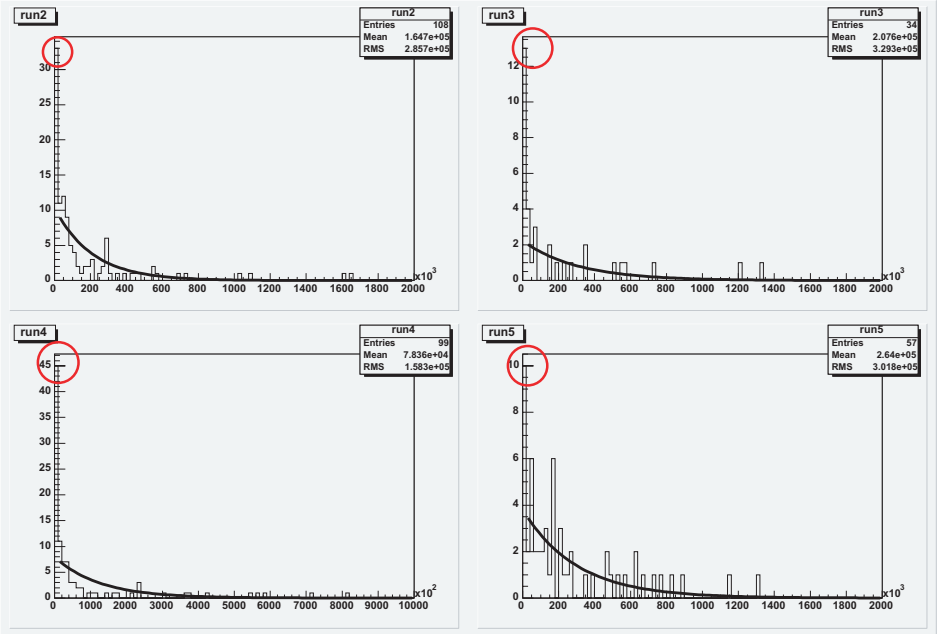

図 5.9: 各 RUN の放電間隔の分布。横軸は経過パルス数 (RUN2~4 は 40MW 以上、RUN5 は 50MW 以上のパワーのパルス数のみをカウント)、縦軸は放電数。

($n(t)$ はパルス数 t の bin に入る粒子数、N は全放電数、B は bin 幅、τ は時定数) によるものである。

放電間隔が最小 ($< 2 \times 10^4$ パルス) の bin に明らかに Fitting とはずれた値が見られ、前の放電直後は放電がおきやすいことがわかる。

NLCTA においても、放電が短時間に連続して起きる現象が確認されており、'Spitfest' と呼ばれている [26]。

このような過程の積み重ねで加速管の特性改善すなわち Processing が行われると考えられる。離散的な現象であるため、短期間での Processing 効果の定量的な検証は難しいことがわかる。KX01 の Processing 効果の検証は Run 間の比較により行う。Run2 の内部で著しい放電頻度の変化があることは以下の解析では注意する。

5.2.6 各 Run の比較

図 5.10 は上記放電頻度と RF パワー、pulse 幅を整理したものである。Pulse 幅、RF パワーに関して一貫性のある運転を行っていない上、Pro-

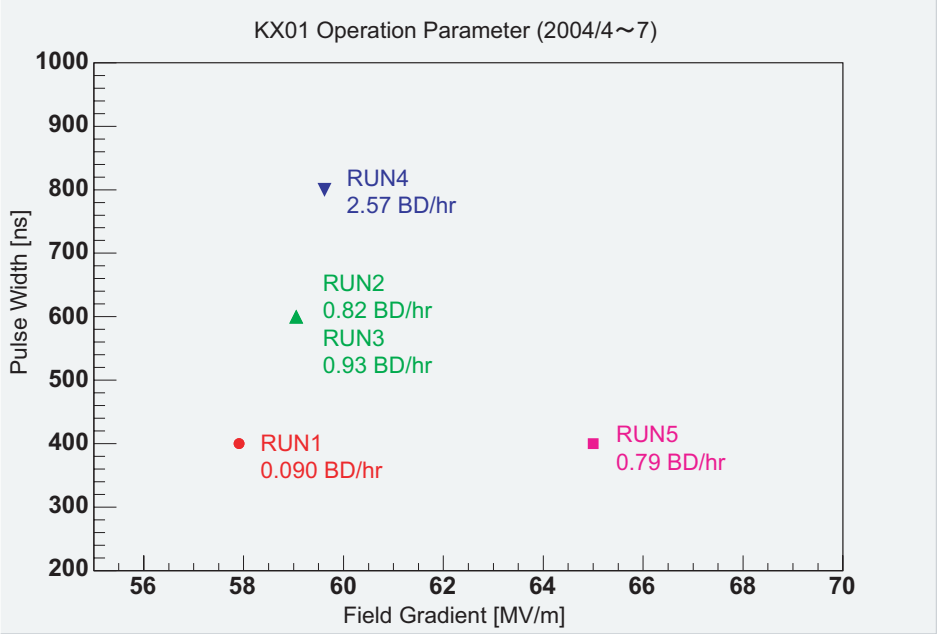

図 5.10: 各 RUN のパラメータ

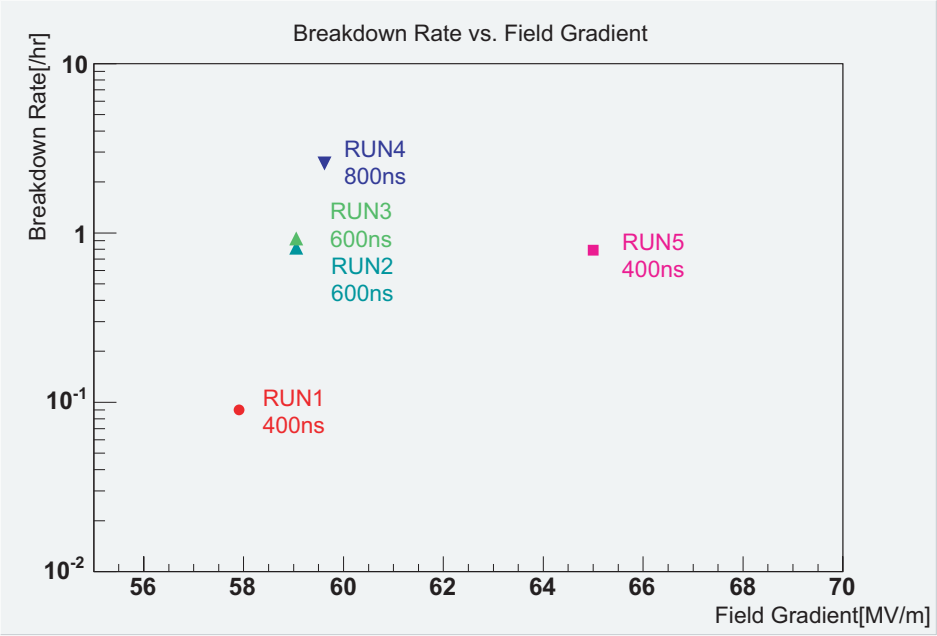

図 5.11: 各 RUN の放電頻度の比較。パルス幅が違うため単純な比較はできない。

cessing 要因も含まれる (同じ pulse 幅の Run1 と Run5 が同じ加速管の状態ではない) ため、解析的な比較をすることが難しい状況にある。そこで、NLCTA での結果もふまえた上でこのデータからわかることを列挙してみる。

表面電場と放電頻度　同じパルス幅、違う電場強度での運転は RUN1 と RUN5 のみである。ただし、この 2 つの RUN の間には 2ヶ月間の Processing 期間があり、加速管の表面状態も変わっている可能性が高い。

従って、表面電場と放電頻度の関係については、XTF で今まで得られたデータからは考察が難しい。以後は NLCTA で得られている KX01 と同型 (H60vg3) 加速管のデータを元に考察する。

$$R_{BD} \sim R_0 \times 10^{\frac{G-G_0}{6[\mathrm{MV/m}]}} \tag{5.6}$$

(NLCTA での放電頻度と加速電場の関係。R_{BD} が放電頻度,R_0 は基準加速電場での放電頻度,G は加速電場,G_0 は基準加速電場。放電頻度は 6[MV/m] 加速電場があがるごとに 10 倍変化する。)

400ns での Processing 効果　上記の依存性を仮定して、400ns での Processing 効果を調べてみる。(5.6) を RUN1,RUN5 のデータに適用する。KX01 の加速電場は、RUN5 の 63MW で 65MV/m,RUN1 の 50MW では

$$65 \times \sqrt{50/65} \sim 57[\mathrm{MV/m}] \tag{5.7}$$

となるので、放電頻度の比は

$$10^{(65-57)/6} \sim 21.5 \tag{5.8}$$

となる。実際の比は 0.79/0.090 ∼ 8.8 となるので、Processing の効果が現れていると考えられる。

pulse 幅と放電頻度　図 5.12 に pulse 幅と放電頻度の関係を示した。この際、同じ RF パワーで放電頻度を測定していないため、

$$R_c = R_{BD} \times 10^{\sqrt{P-63}/6} \tag{5.9}$$

(R_{BD} は実測放電頻度,P は RF パワー [MW],R_c は規格化放電頻度)
を用いて RF パワーに補正を行っている。

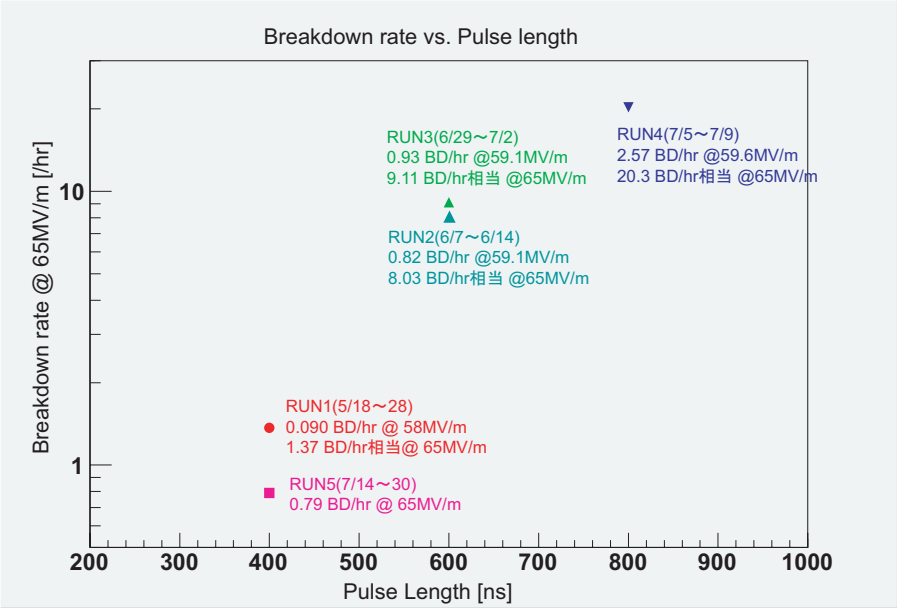

図 5.12: pulse 幅と放電頻度。電場については (5.6) に従いに放電頻度を 65MV/m に規格化した。

パルス幅が大きくなると放電頻度が大幅に上昇している様子がわかる。関係式についてはデータをもう少し追加する必要がある。またパルス内の放電時間の分布を調べることでも pulse 幅と放電頻度の関係を推測できる。

5.2.7 まとめ

本節で得られた結果を以下にまとめる。

- 各 RUN 内での放電頻度の変化

  RUN 内では基本的に同じ RF パラメータで運転を行っているが、放電頻度は必ずしも均等ではないことがわかった。特に放電直後にもう一度放電が起きる確率が高い 'Spitfest' 現象が確認できた。

  これらは、1 度の放電でも加速管表面への影響がかなり大きい場合があるということを示唆している。放電により加速管表面状態が変わり、場合によっては放電を誘発することになるのである。

- Processing の効果

RUN1 と RUN5 は同じパルス幅での運転で、NLCTA の表面電場と放電頻度の関係式を用いると換算した放電頻度は RUN1 の方がかなり大きくなる。

よって、NLCTA の仮定に依存するものの、Processing の効果がある程度確認できたと言える。

- pulse 幅依存性

  pulse 幅をのばすと飛躍的に放電頻度があがることは確認できた。指数的な関係かべき乗の関係にあるかは今回のデータだけでは不明である。

XTF は、今後導波管コンポーネントの入れ替えにより加速管供給可能パワーが〜80MW 程度に上昇する予定であり、データの追加によってパワー、pulse 幅等の放電頻度に対する依存性はより詳しく解析する予定である。

5.3 放電の位置

本節では、放電位置の検出方法と KX01 の放電位置分布に関する結果を示す。

5.3.1 位置解析の意義

加速管放電を減少させるためには、加速管内で放電の起こりやすい位置を特定し、その原因を究明することが必要である。放電位置の解析は、運転中に外部の放電検出器からのデータによって解析する方法と、運転終了後に加速管内部の放電痕から調べる方法と大きく二つに分かれるが、KX01 はまだ運転中のため、ここでは前者のみを用いた結果を示す。

図 5.13 に KX01 の表面電場 (理論値) の分布を示す。

5.3.2 RF データからの位置検出

放電位置の検出は RF,γ 線, 音響のそれぞれの検出器から独立に行うことができる。

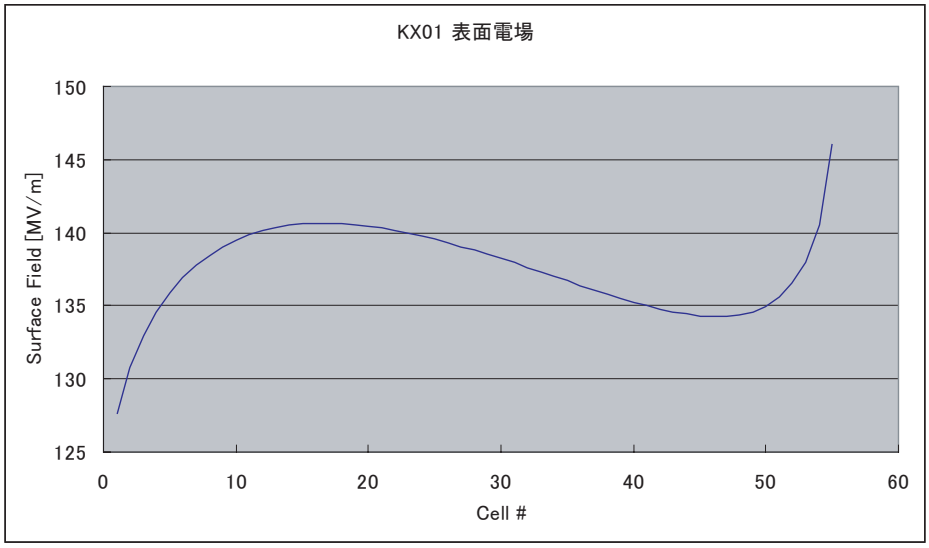

図 5.13: KX01 の表面電場 (理論値)。入力パワーは 63MW としている。

RF からの放電位置検出は、透過 RF 波形および反射 RF 波形を用いて行う (図 5.14)。

位置 Z においてパルスの先頭から時間 t の場所で放電が起きたとすると、t 以降の RF は正常に伝播されず、透過 RF パルスの幅が t に変化する ($t_{tr} = t$)。また反射は位置 Z において時刻 t で起こり、それが入射カプラに戻ってくる。非放電パルスでの反射は主に入射カプラで起こっていると考えられており、放電による反射との時間差は $t_{rs} = t + \frac{2Z}{v_g}$ となる。実際には v_g は加速管の位置により異なるのでその積分を計算することになる。この二つのデータから t,Z を求めることができる。

(実際のパルスでの放電位置測定例は図 5.19 を参照)

具体的には、以下の手順に従い放電位置を算出する。

1. 透過 RF, 反射 RF 波形それぞれについて、放電時の RF 波形 (1) と一つ前のパルスの波形 (2) を重ねる。

   このとき、波形全体の時間方向のジッタ (主に FADC のスタート信号のディレイゲートに起因する) をキャンセルするため、強度軸に基準値を設け、それぞれの波形がその強度軸をまたぐ時間が同じになるように補正する。

   また、FADC baseline もふらつくので揃える。これは RF が入る前の部分を切り出して平均する。

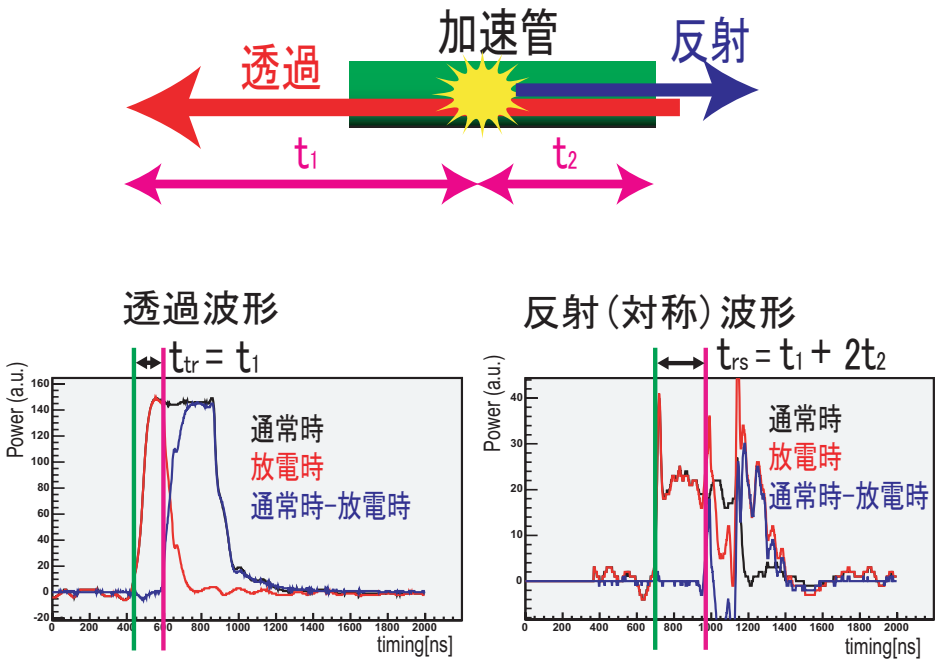

図 5.14: RF データからの放電位置検出

2. (1) と (2) の差分をとり、(3) とする。

3. (1)、(3) の波形が基準値を超える時間の差を t_{tr} または t_{rs} とする。

4. 上式に従って Z,t を算出する。

この位置算出の誤差要因としては以下が考えられる。

- RF 波形の立ち上がり誤差

  反射波形の小さなイベントでは、立ち上がりから基準値に達するまで時間がかかり、その間の時間が誤差となってしまう。この誤差はイベントによって異なるが、加速管の群速度 v_g はおよそ光速の $3\% \sim 1 \times 10^7$[m/s] であり、2ns(=FADC の 1 チャンネル) の誤差でも約 2cm〜2 セル分となってしまう。これを抑えるには、基準値をできるだけ低くするのが有効だがあまり低くするとノイズを拾ってしまう。実際にはノイズとぎりぎり区別できる程度の基準値でも反射の少ないイベントでは 20ns 以上かかることがあり、分解能が極めて悪くなってしまう。

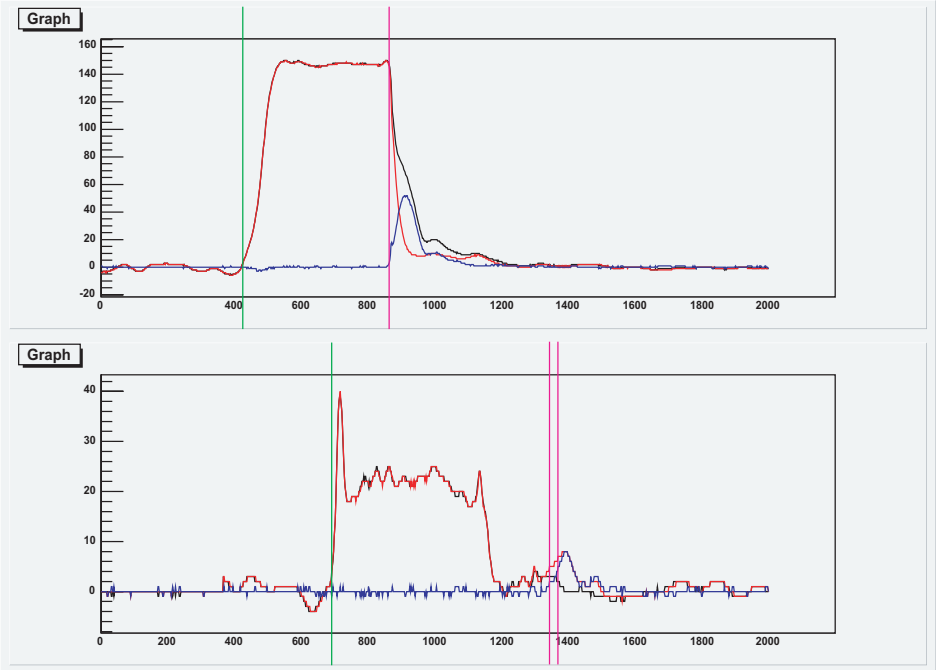

図 5.15: 反射が小さな放電では反射スペクトルの立ち上がり判定が難しい。この図では閾値 5 カウントで反射位置を見つけ (右の赤線)、その後信号が 2 カウント以下になるまで前方に戻る (左の赤線) ことにより正確な立ち上がり位置を取得している。

この誤差を緩和するために、現在の解析ルーチンでは基準値を 2 段階設け、はじめに高い方の基準値で立ち上がりを探し、それが見つかると低い方の基準値を下回るまでパルスの先頭方向に戻っていくという方式を採用している。

これにより、立ち上がり誤差は 10ns 以下程度に抑えられている。

- 非放電パルスの反射位置

  非放電パルスの反射波形は、パルスの先頭と最後尾にピークを持つ波形である (図 3.50 参照)。このピークは導波管と加速管を接合するカプラで RF の群速度が大きく変わることにより発生する反射と考えられているが、入力パルスも完全に矩形ではなく立ち上がり時間があり、反射ピークもまた時間的な広がりがあるため、ピークのどの位置を反射の先頭と考えるかに誤差が生じる。

  反射ピークの幅が 20ns 程度であるので、この誤差は 20ns より少な

いと考えられる。

これらを総合して、RF 位置検出の分解能は 5〜10 セル程度であると考えられる。

このように、この位置解析方法はさほど分解能は高くないが、従来から行われてきた方法であり、信頼性が高い点に特徴がある。

5.3.3 γ 線データからの位置検出

γ 線検出器の ADC 分布から直接、放電位置を検出することができる。

γ 線検出器は、シンチレータと光電子増倍管を鉛のコリメータで包んだ形状であり、コリメータが正しく機能すれば、前方のわずかな立体角から飛来した γ 線のみを検出するはずである。放電に伴う γ 線の放出が加速管の 1 点から起こっていれば、特定のチャンネルのみ信号が検出され、容易に放電位置を特定できるはずである。

しかし、実際の ADC 分布は図 5.16 のように、強度にやや分布はあるものの、すべてのチャンネルで信号が見られる。この現象については検証が必要だが、一応 ADC 分布は一点にピークを持つ分布を示すため、これから位置の特定が可能である。

ただし、γ 線検出器は加速管全長の 6 点しか計測していないため、分解能は極めて悪い。このため、現在の解析では γ 線 ADC のデータは放電位置の解析には用いていない。

また、全チャンネルで応答があるので、TDC 分布から放電位置を検出することも可能と考えられる。実際、TDC 分布は放電位置によってやや変わるように見えるが、まだ統計的に放電位置を検出・解析するには至っていない。

5.3.4 音響データからの位置検出

音響センサは放電位置を精密に特定する目的に特化したセンサーで、セル単位での放電位置を検出できると期待される。

音響センサからの放電検出にも、二つの方法が考えられる。

信号強度による検出　放電時は非放電時より大きな波形が記録されるはずである。その強度比をチャンネル毎に調べることで、放電位置を検出できると予測される。

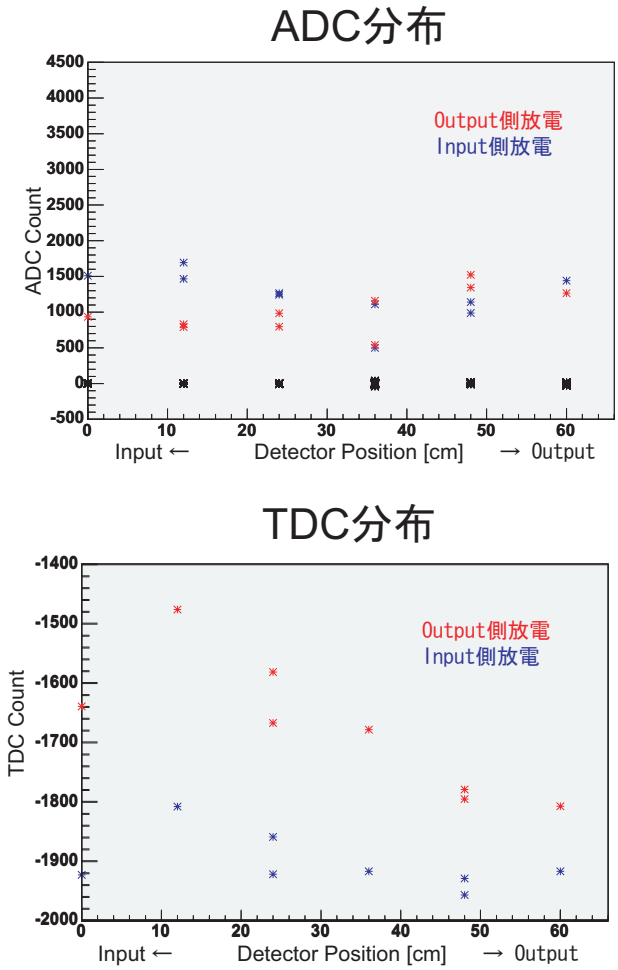

図 5.16: γ 線検出器・ADC 分布・TDC 分布。RF, 音響のデータから、上段は上流の放電、下段は下流の放電と推測される。ADC 分布では、上段の方が上流側の信号がやや強くなっている。また、TDC 分布はどちらも下流側の時間が早くなっているが、上流と下流の差は下段の方が大きい (全体として上流の方が TDC 値が小さくなっているのは、パルス内の放電時間による)。

信号の立ち上がり時間による検出　第 3 章で述べたように、熱衝撃の伝播速度は RF に比べて極めて遅い。そのため、1 点で熱衝撃が発生し、それが周囲に伝播していくモデルを想定すると、放電が発生したセルと隣接するセルの時間差は、δl をセル間隔 (〜1.0cm) として、

$$\delta t = \frac{\delta l}{v_{cu}} \sim 2[\mu s] \tag{5.10}$$

となる。これは 10MHz の Digitizer で十分検知できる範囲と期待される。

典型的な音響波形　まず、典型的な放電の音響波形 (図 3.30 を参照) を見てみると、非放電時にもある程度の大きさの波形が検知されている。これは、RF 導入に伴う熱衝撃によるものと思われる。この非放電時の波形は、位相も含めてパルス間でほとんど等しい。

次に、放電時の波形を見てみると、非放電時の波形から変化がみられる。非放電時の大きさとの比は、イベントにより 10 倍くらいのものからほとんど変化が見られない (〜数%) ものまである。

チャンネル分布については、特定のチャンネルで強度が高いイベントはあまりなく、数セル〜数十セルにわたって非放電時と違う波形が見られる。

また、非放電時より大きな波形が観測される位置 (放電位置と思われる) より下流については、通常時より小さな波形が観測されている。これは、非放電時に通過している RF が放電箇所で遮断され RF が通過しなくなったためと考えられる。

以上をふまえて、２種類の方法により信号強度による放電検出を行う。

差分法　非放電時の波形が位相込みでほとんど同じことを利用し、放電時の波形から非放電時の波形を引くことで波形の分離を試みる。

放電時の電圧波形を $V_n(t)$, 一つ前の波形を $V_{n-1}(t)$, 差分を

$$\delta V_n(t) = V_n(t) - V_{n-1}(t) \tag{5.11}$$

とする。

波形の強度は $\delta V_n(t)$ の二乗積分で与えることにする。ただし、これだと前記のように下流の波形が観測されてしまうため、以下のように補正をして非放電時より強度が弱くなった分はカウントしないことにする。

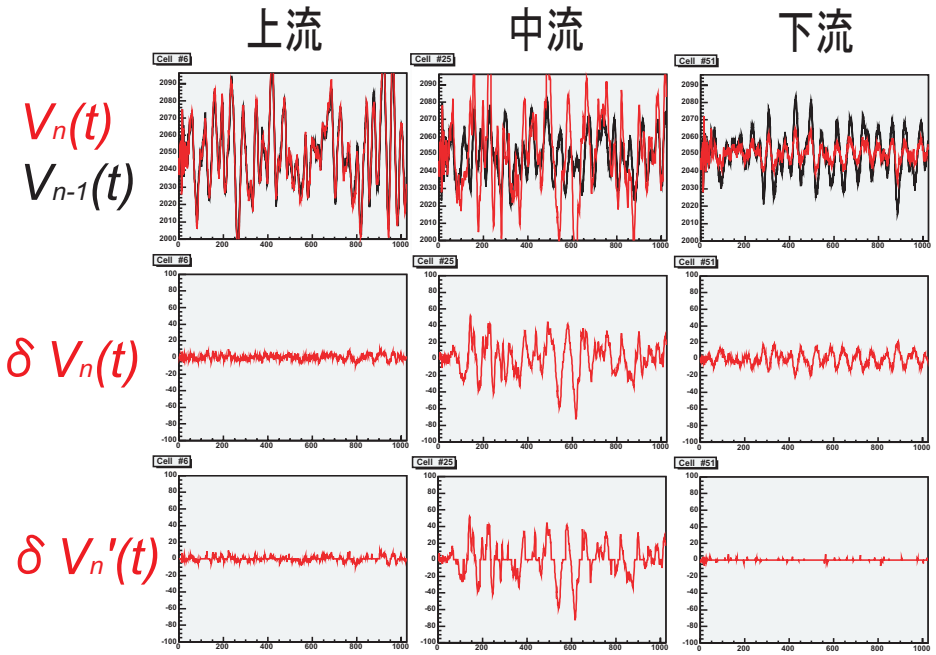

図 5.17: 差分法の波形処理。δV_n では下流側の RF が弱くなった部分も信号が見えてしまうのに対し $\delta V_n'$ ではキャンセルされていることがわかる。

$$\delta V_n(t)'(t) = \begin{cases} \delta V_n(t) & \delta V_n(t) \times V_{n-1}(t) > 0 \\ 0 & \delta V_n(t) \times V_{n-1}(t) \leqq 0 \end{cases} \tag{5.12}$$

$$I_{dif} = \int \delta V_n(t)'(t)^2 dt \tag{5.13}$$

(図 5.17 参照)

この強度のチャンネル分布をとり、Peak 値および Gaussian フィットの中央値を解析に用いる (目視で代表値の正確さを確認している)。gaussian はうまくフィットできるものとできないものがある。

(チャンネル分布の例は図 5.19 参照)

rms 法　この方法は波形の演算を行わず、単に通常時の rms と放電時の rms の比を用いる。

rms 法のチャンネル毎の強度は以下の式で計算される。

$$I_{rms} = \frac{\int V_n(t)^2 dt}{\int V_{n-1}(t)^2 dt} \tag{5.14}$$

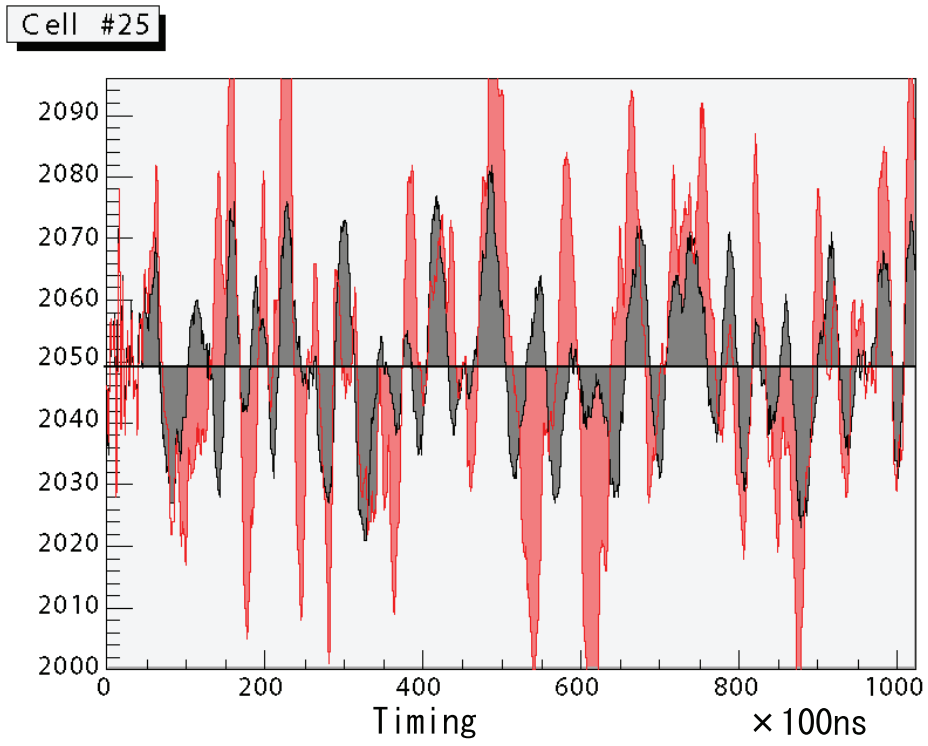

図 5.18: rms 法の波形処理。赤で塗りつぶした部分/黒で塗りつぶした部分が信号強度となる。非放電時の信号強度は 1 となる。

音響センサは Calibration を行っていないため、差分法はチャンネル毎の感度の影響を受けてしまうのに対し、rms 法は非放電波形のチャンネル依存性がないという仮定の下で Calibration の影響を受けない利点がある。ただし、実際の放電では隣のチャンネルにくらべ非放電波形が著しく小さいチャンネルでも放電時の波形強度はあまり変わらない場合も多く、Cancelation がうまくいかない場合もある。

また、波形の分離を行っていないため、波形変化が小さいイベントでは判断が難しくなる。

これらの二つの方法を利用し、音響センサから放電位置の解析を行っている。

立ち上がり時間　また、放電波形の立ち上がり時間は、顕著なものが確認されていない。このことから、熱衝撃が一点から発生しているという仮定に誤りがある可能性が高いと考えられる。NLCTA においては、特定

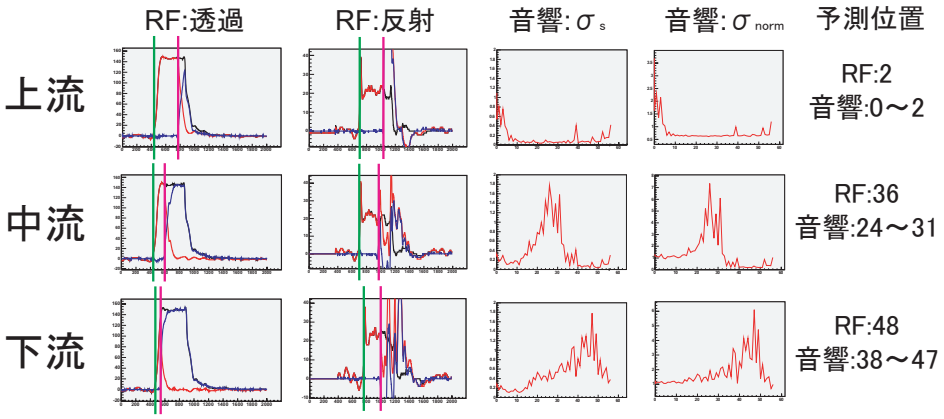

図 5.19: 典型的な放電での RF と音響からの放電位置

のイベントにおいて、放電セルと隣接数セルとの間で立ち上がり時間の遅延が見られたという報告もある。[27]

立ち上がり時間については、さらなる解析が必要である。

5.3.5 データ間の相関

RF から得た放電位置と、音響センサから得た放電位置の比較を行う。

ここで使用しているのは、Run5 の 7/16 14:38 以降の 39 イベントのデータである。

図 5.19 には、典型的な放電での RF 検出器からの放電位置と音響検出器からの放電位置の比較を示した。数セルのずれはあるが、だいたいの検出位置は一致していることがわかる。

この比較を全イベントについてプロットすると、図 5.20 を得る。RF と音響からの位置は大まかには一致しているイベントが多いが、一部音響からの位置は Input Coupler 周辺で、RF からの位置は中央部になっているものがある。(図 5.19 の○印)

これらは図 5.15 に示されているようなイベントで、主に RF の立ち上がり誤差がきいていると考えられる。

5.3.6 位置分布と電界強度

図 5.20, 図 5.21 の放電位置の分布を見ると、まず中央付近が少なく、Coupler 周辺での放電が多いことがわかる。Coupler は通常のセルとは構

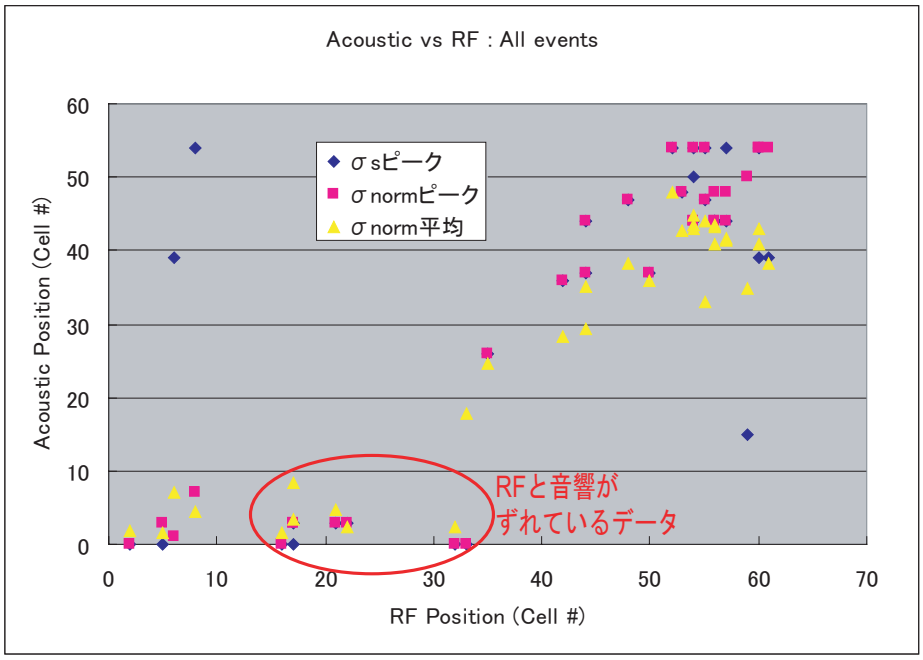

図 5.20: RF と音響からの放電位置の比較

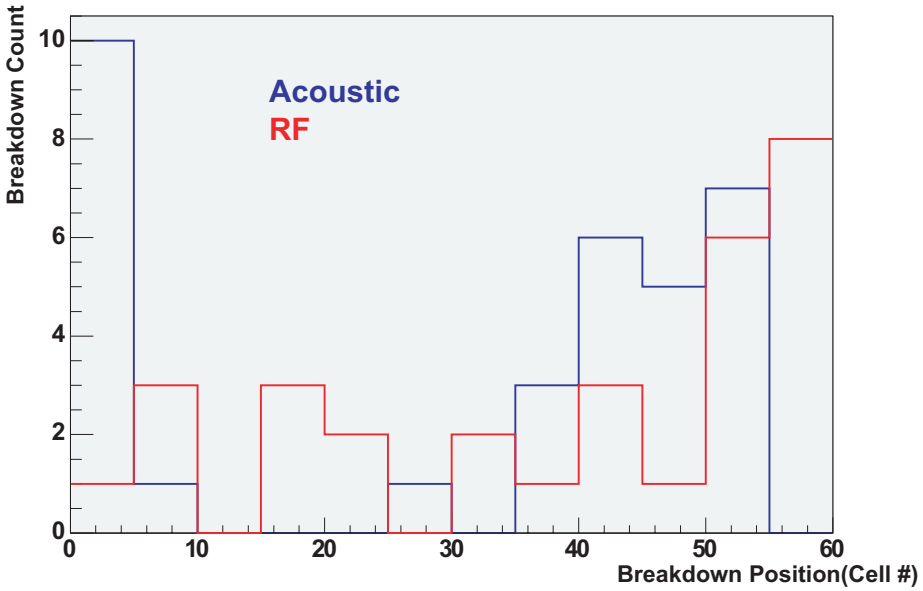

図 5.21: RUN5 での放電位置。入力、出力カプラ付近の放電が多いことがわかる。

造の異なるセルがいくつかあり、表面電場が高い部位がある可能性がある。とくに Output Coupler 付近が多く、図 5.13 に示されている Output Coupler 付近の電場の強さを反映しているものである可能性がある。

ここでは放電位置のみからの議論だが、放電強度との比較を行うことで深刻な放電を引き起こす部位について今後検討を進めていく。

5.4 まとめ

XTF において KX01 の高電界試験を行った際の加速管放電を、RF,γ 線, 音響の 3 つの検出器から得られたデータをもとに解析した。

放電頻度については、入力 RF パワー、パルス幅と放電頻度の関係性を調べ、また放電の時間局在性についても確認できた。また加速管 Processing の効果を確認した。

放電位置については、RF 検出器のデータからの予測位置と音響検出器のデータからの予測位置がおおまかに一致することを示し、また KX01 においては Output Coupler 付近での放電が多いことが確認できた。

今後は、Run4 以前のデータについても解析を行い、放電位置の Processing による変化等についても解析する。また、RF power とパルス幅の放電頻度に対する依存性については、より依存性がわかる設定でデータ収集を行う。

また、今後 XTF で高電界試験が行われる KX02,03 においても Processing の経過による放電特性の変化をより詳細に調べることになっている。

第6章　まとめと今後の展望

6.1　XTF

XTF は 2003 年夏から建設を開始し、2004 年 4 月から KX01 加速管を設置し、加速管の Processing, 高電界試験, 放電データの収集を行ってきた。当初目標としていた 65MV/m, パルス幅 400ns での放電頻度の測定を行い、0.79[BD/hr] という結果を得た。またパルス幅をのばした状態での運転も行った。

今後の予定は、1 月に低損失円形導波管の導入、クライストロンの交換により 80MW 程度に加速管供給可能パワーを増やし、より高いパワーの RF での放電特性を調べる。また、3 月頃からは KX02 のインストール、Processing を予定している。KX02 は、ビーム加速の際問題となる高調波を減衰するためのスロットを付加した加速管である。その後、XTF では現在製作中の KX03 の Processing を行い、シャットダウンする予定でとなっている。

冒頭に述べたように、XTF は当初 Linear Collider のための X-band 高周波加速技術の確立を目指していた。Linear collider 計画は 2004 年夏から L-band の超伝導空洞を加速管に用いるという方針転換がなされ、現在超伝導空洞および関連技術の研究が急速に進められている。XTF を含め、今まで X-band の常伝導空洞を前提に構築されてきた技術も、超伝導に転用できる部分は生かしていく。

X-band 加速技術は、Linear Collider としては白紙に戻されたが、小型かつ高い加速勾配が可能であるため、将来にわたって有望な加速技術であり、今後とも研究・開発の必要性は高い。XTF の今後の Run では、そのような幅広い応用も見据えて、必要なデータ収集を行っていく予定である。

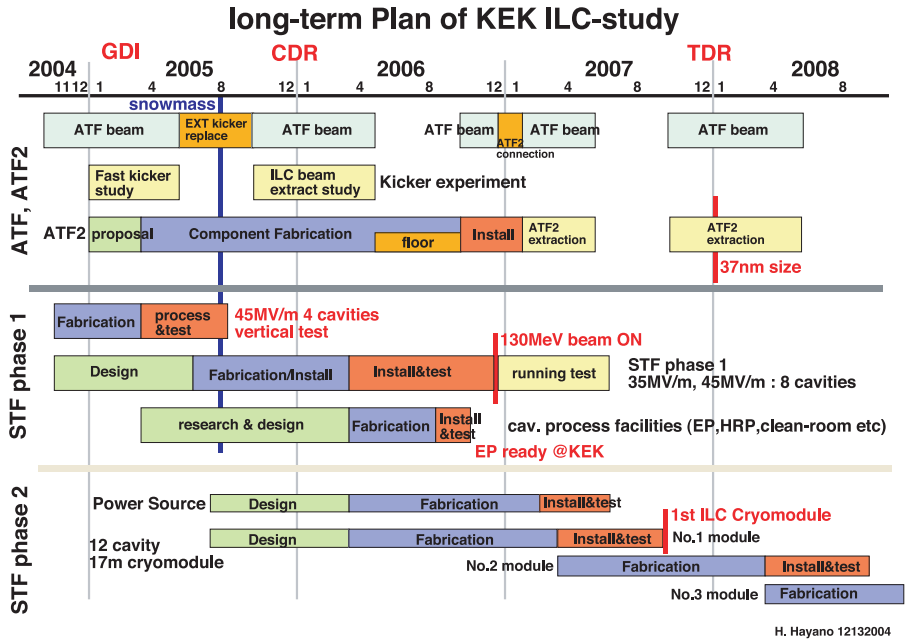

図 6.1: KEK での ILC 関係開発タイムスケジュール

6.2 放電検出システム

XTF の放電検出システムは、XTF の運転開始にあわせ、2003 年夏頃より開発が行われ、運転後も幾度かの改良を経て、現在安定にデータ収集を行える状態になっている。

放電検出器は、RF 検出器、γ 線検出器、音響検出器の 3 種類を設置しており、それぞれの検出器で、加速管の放電時に有意なデータが得られている。

システムとしては、50Hz での毎パルスのデータ収集と放電検出、放電前パルスを含めたイベントの保存、オンライン解析、ネットワークによる設定変更等が実装され、放電検出システムとしては一通りの機能を備えたものが完成した。

KX02,03 では、KX01 で開発途上だった Processing 初期のデータ収集をより精密に行い、Processing にともなう放電特性の変化をより詳細に追跡できる予定となっている。

改良点としては、音響センサの Calibration,γ 線検出器のコリメータの改善などが考えられ、現在検討中である。

6.3 放電解析

放電解析については、KX01 の放電データを元に、放電頻度、放電位置の解析を行い、一通りの結果を得た。

KX01 の放電頻度の RF パワーに対する依存性、パルス幅に対する依存性も確かめられた。また、Processing による放電頻度の改善も確認された。

放電位置については、Run5 のみの解析だが、coupler 周辺に放電が集中していることが確認された。

今後の解析では、Run1〜4 についても Run5 と同様の解析を行うとともに、エネルギー収支についてさらに解析を進める。

また、当初の放電解析の目的の一つに、放電の予兆を捉えるというものがあり、これは今後のデータ収集で積極的に行っていく。

これらを通して、加速管の放電メカニズムに関する検証も、今後より詳細に行っていきたいと考えている。

第7章　謝辞

本研究においては、以下の方々に多大なご協力をいただきました。深く感謝いたします。

- KEK の肥後寿泰先生には、この放電検出システムの構想・設計・製作にわたり全面的に指導をいただきました。
- KEK の早野先生、照沼先生、佐伯先生、東北学院大の渡辺君はじめ XTF(旧 GLCTA) および ATF グループの方々には、XTF の建設、維持および BDMS の運転等に関し全面的に協力いただきました。
- (株) 関東情報サービスの早川さん、塚田さんは XTF コントロール全般および BDMS ユーザインタフェース部、オシロスコープ読み出し部の開発に尽力いただきました。
- SLAC の NLCTA グループの方々には音響センサおよびモジュールを提供していただきました。また有用な解析データをいただきました。
- 駒宮先生・佐貫先生をはじめ駒宮研究室の方々には研究の全般にわたり指導・助言をいただきました。

関連図書

[1] LHC website

http://lhc-new-homepage.web.cern.ch/lhc-new-homepage/

[2] ILC website

http://www.interactions.org/linearcollider/

[3] CERN website

http://public.web.cern.ch/Public/Welcome.html

[4] LHC Design Report

http://ab-div.web.cern.ch/ab-div/Publications/LHC-DesignReport.html

[5] GLC Project Report 2003

http://lcdev.kek.jp/ProjReport/PDF/GLCReport.pdf

[6] Koji TAKATA, 高周波加速の基礎, KEK report 2003-11, P.35〜37,P.65

[7] ATF website

http://www-atf.kek.jp/atf/index.html

[8] P.B.Wilson, Frequency and pulse length scaling of RF breakdown in accelerator structures, SLAC-PUB-9114(2002)

[9] SLAC website

http://www.slac.stanford.edu/

[10] KEK website

http://www.kek.jp/intra-j/index.html

[11] KEK X-band R&D website(XTF を含む)

http://xband.kek.jp/

旧 GLCTA website

http://atfweb.kek.jp/glcta/

[12] DESY website

http://www.desy.de/html/home/fastnavigator.html

[13] TESLA website

http://tesla-new.desy.de/content/index_eng.html

[14] ITRP Final Report(2004)

http://www.ligo.caltech.edu/ skammer/ITRP/ITRP_Report_Final2.pdf

ITRP press release

http://www.interactions.org/cms/?pid=1014290

[15] 江面栄二, マイクロ波伝送と信号解析の基礎 *(*加速器設計シリーズ*)*, KEK Internal 2003-3,pp.152

[16] Toshiyasu Higo, 高電界試験状況と今後の開発について, 第 5 回高エネ研メカ・ワークショップ報告集, Apr.2004, KEK, Ibaraki, Japan

KEK Proceedings 2004-10 P.31-49

[17] 岡田文明, マイクロ波工学—基礎と応用— 学研社,1993

[18] 山村大樹, リニアコライダー主線形加速器での電子ビーム輸送, Master thesis (2004)

[19] R. Brun and F. Rademakers, *ROOT — An Object Oriented Data Analysis Framework*, Nucl. Instrum. Methods Phys. Res. A 389, 81 (1997)

http://root.cern.ch/

[20] http://www.toyo.co.jp/daq/index.html

[21] Y. Yasu et al., *Development of a pipeline CAMAC controller with PC/104 plus single board computer*, 2003 Conference for Computing in High-Energy and Nuclear Physics (CHEP 03), La Jolla, California, 24-28 Mar 2003

[22] CC/NET UserGuide http://www-online.kek.jp/ yasu/Parallel-CAMAC/UserGuide/UserGuide.pdf

[23] V.Russier, J.P.Badiali, *Calculation of the electronic work function of Cu and Ag from an extended jellium model*, Phys. Rev. B, 39-18(1989)

[24] S.Döbert, *Status of High-Gradient R&D*, ISG-11, KEK (2003)

[25] P.Wilson, *GRADIENT LIMITATION IN ACCELERATOR STRUCTURES IMPOSED BY SURFACE MELTING*, Workshop on High Gradient RF, ANL (2003)

[26] C.Adolphsen, *NORMAL-CONDUCTING RF STRUCTURE TEST FACILITIES AND RESULTS*, Proceedings of the 2003 Particle Accelerator Conference, pp.668

[27] J.Nelson et al., *Use of Acoustic Emission to Diagnose Breakdown in Accelerator RF Structures*, SLAC-PUB-9808(2003)

付 録A　放電検出システム(CAMAC部)ソースコード

現行バージョン:Last modified 2004/11/25
　文字化けしたコメント等を一部整理しました。

A.1　bdmonitor.h

```
// bdmonitor.h -- BreadDown monitor system main header

#include <list>
#include <stdio.h>
#include <time.h>
#include <iostream>

#include "CEventAnalyzer.h"
#include "CSocketReceiver.h"
#include "CCsyCtrl.h"
#include "CMasterCtrl.h"
#include "CCamacCtrl.h"

#define DEBUG 1

// global functions

// Initialization functions
// Create your object (probably at global scope),then call them at your constructor.

// If your class derives from CEventAnalyzer and enable analyze feature,
// call the function below at constructor.
void AddAnalyzer(CEventAnalyzer *pAnalyzer);

// If from CSocketReceiver
void AddReceiver(CSocketReceiver *pReceiver);

// global variables
extern std::list<CEventAnalyzer *> g_liAnalyzer;
extern std::list<CSocketReceiver *> g_liReceiver;

extern CCamacCtrl g_camac;
extern CMasterCtrl g_master;
extern CCsyCtrl g_csy;
extern CSocketCtrl g_socket;

// utility function
inline char * GetTime(bool bNewTime = true,struct tm *tim2 = NULL)
{
    static char s[1024];

    if(bNewTime){
        time_t t;
        time(&t);
        struct tm *tim = localtime(&t);
        strftime(s,1024,"%Y/%m/%d %H:%M:%S",tim);
        if(tim2)memcpy(tim2,tim,sizeof(struct tm));
    }
```

```
    return s;
}
```

A.2 bdmonitor.cpp

```
// ROOT library
#include "TROOT.h"
#include "TFile.h"
#include "TNtuple.h"

#ifndef WIN32
#include <unistd.h>
#include <sys/ioctl.h>
#endif

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
// Standard library
#include <iostream>
#include <assert.h>
#include <vector>
#include <list>

#include "bdmonitor.h"
#include "CCamacCtrl.h"
#include "CSingleEvent.h"
#include "CEventAnalyzer.h"
#include "CSocketReceiver.h"
#include "CCsyCtrl.h"
#include "CMasterCtrl.h"

#include "CAnalyzeXRay.h"
#include "CAnalyzeFADC.h"

using namespace std;

TROOT MyTinyApp("GLCTA-X-Ntuple", "GLCTA X-ray detection system Ntuple generator");

list<CEventAnalyzer *> g_liAnalyzer;
list<CSocketReceiver *> g_liReceiver;

CCamacCtrl g_camac;
CMasterCtrl g_master;
CCsyCtrl g_csy;
CSocketCtrl g_socket;

// global functions
void AddAnalyzer(CEventAnalyzer *pAnalyzer){g_liAnalyzer.push_back(pAnalyzer);}
void AddReceiver(CSocketReceiver *pReceiver){g_liReceiver.push_back(pReceiver);}

int main(int argc,char *argv[]) {

    try{
        // XRay analyzer
        CAnalyzeXRay xray;
        // RF analyzer
        CAnalyzeFADC rf;

        cout << "GLCTA breakdown record system compiled at " << __TIME__ << endl;
        g_master.Initialize();

        cout << "MasterCtrl Initialized. Go to main loop." << endl;

        g_master.MainLoop();

        cout << "Main loop returned. exit." << endl;
    }
    catch(const char *pStr)
    {
        cout << pStr << endl;
        cout << "Scan Aborted." << endl;
    }

    return 0;
}
```

A.3 Camac-slot.h

```
// Camac-slot.h
// defines slot no. for each module

// Common Modules
#define CSY            23
#define SCALER         22
#define OUTREG         21
#define CSY2          13

// 500MHz FADCs
#define FADC500_1     16
#define FADC500_2     18

#define FADC500_DATASIZE 1000
```

```
// 100MHz FADCs
#define FADC100_1    11
#define FADC100_2    12
#define FADC100_3    13
#define FADC100_4    14

// Xray ADCs
#define XRAYADC_1    7
#define XRAYADC_2    8

// Xray TDCs
#define XRAYTDC_1    4
#define XRAYTDC_2    5
```

A.4 CMasterCtrl.h

```
// Master Control Class : Controls Start,Stop,Status,etc.
#ifndef CMASTERCTRL_H_INC
#define CMASTERCTRL_H_INC

#include "CSocketReceiver.h"
#include "CCsyCtrl.h"
#include "CSocketCtrl.h"
#include "streamers.h"
#include "TTree.h"
#include "TString.h"

class TFile;
class CMasterCtrl : public CSocketReceiver
{
public:
    CMasterCtrl(void);
    virtual ~CMasterCtrl(void);

    // Main loop
    void MainLoop(void);

    // Socket Callback
    virtual bool Receive(std::istrstream &in,std::ostrstream &out);

    // Commands
    void Start(void);
    void Stop(bool bBreakDown);
    bool Status(void) const {return m_bStatus;}
    void Clear(void);

    void SetBreakdownMessage(const char *pStrBreakdownMessage);

    // Commands : To be implemented
    /*
    void BreakDownDetection(int nDetectionMode);
    void CallbackEnable(bool bBreakDownCallback,bool bDataFullCallback);
    */

private:
    CMasterParam *m_pParam;
    CPulseSummary *m_pSummary;
    TTree * m_pTreeSummary;
    bool m_bStatus;
    TFile *m_pFile;
    const char *m_pStrFilenameBase;
    char *m_pStrFilename;
    char *m_pStrLastFilename;

    bool m_bOverWrite;
    TString m_strMessage;
//    char *m_pStrBreakdownMessage;
protected:
    void SaveBin(const char * pStrFilename);
    void SaveBinPeriodic(const char *pStrFilename);
    void Save(const char *pStrFilename);
public:
    // Fill summary : called from CCsyCtrl
    void FillSummary(){m_pTreeSummary->Fill();}
    void Initialize(void);
};

#endif
```

A.5 CMasterCtrl.cpp

```
#include "bdmonitor.h"
#include "CMasterCtrl.h"

// ROOT include
#include "TFile.h"

#include <time.h>
#include <iostream>
#include <iomanip>
using namespace std;

#include "CCamacCtrl.h"
```

```
#include "Camac-slot.h"

CMasterCtrl::CMasterCtrl(void)
{
    m_pFile = NULL;
    m_pStrFilename = NULL;
    // Initialize Valiables
    m_bStatus = false;    // Not Collecting data.

    // Set Category
    SetTargetString("Main");
    // Add to Socket Receiver
    g_liReceiver.push_back(this);
    // Add to CsyCtrl
//    g_liCsyCtrl.push_back(this);

    m_bOverWrite = false;    // Modified 20040311

    m_pStrFilenameBase = "/mnt/nas/data/rawdata";
    m_pStrLastFilename = NULL;

}

CMasterCtrl::~CMasterCtrl(void)
{
}

void CMasterCtrl::Initialize(void)
{
    // Set Buffer file
    time_t t;
    time(&t);
    struct tm *tim = localtime(&t);
    char s[1024],s2[1024],sRecent[1024];

    // File "Recent"
    sprintf(sRecent,"%s/recent.root",m_pStrFilenameBase);
    // Read only
    TFile f(sRecent);

    // File to save
    sprintf(s2,"%s/%%Y/%%m/%%Y%%m%%d.bdmon.root",m_pStrFilenameBase);
    tim = localtime(&t);
    strftime(s,1024,s2,tim);
    m_pStrFilename = new char[strlen(s)+1];
    strcpy(m_pStrFilename,s);
    m_pFile = new TFile(m_pStrFilename,"UPDATE");
    if(!m_pFile)
    {
        printf("Data file cannot opened. exit.\n");
        exit(1);
    }
    // current dir is savefile

    // Initialize MasterParam
    m_pParam = GetParamFromFile<CMasterParam>("MasterParam",&f);

    // Initialize socket
    g_socket.Initialize();

    // CAMAC Initialize
    g_camac.Init(0);

    // Summary Initialize
    m_pTreeSummary = new TTree("PulseSummary","summary data of every pulse");
    m_pTreeSummary->SetAutoSave((int)1e+9);    // Autosave=1GB(practically no autosave)
    m_pSummary = new CPulseSummary;
    m_pTreeSummary->Branch("Summary","CPulseSummary",&m_pSummary);

    // Analyzer Initialize
    list<CEventAnalyzer *>::iterator itAna;
    for(itAna = g_liAnalyzer.begin();itAna != g_liAnalyzer.end();itAna ++)
        (*itAna)->Initialize(gDirectory,&f,m_pSummary);
    for(itAna = g_liAnalyzer.begin();itAna != g_liAnalyzer.end();itAna ++)
        (*itAna)->Clear();

    // Csy initialize
    g_csy.Initialize(&f);

    // Save Setting
    m_pFile->Write();

    // re-attach file "recent"
    // Changed to HARD link : 20041005
    sprintf(s,"rm %s\n",sRecent);
    system(s);
    sprintf(s,"ln %s %s\n",m_pStrFilename,sRecent);    // hard link
    system(s);

    // file "recent" close automatically
}

// Main Loop;
void CMasterCtrl::MainLoop(void)
{
    // Infinite loop
    while(1)
    {
        // Check Socket
        try{
            g_socket.CheckSocket();
        }catch(const char *p)
        {
            cout << "Socket error." << endl;
```

```
            cout << p << endl;
        }

        // Status
        if(!Status())
        {
#ifndef WIN32
            sleep(1);
#endif
            continue;
        }

//        cout << "|" << flush;
        // Event Reading
        g_camac.ReadEvent();

        std::list<CEventAnalyzer *>::iterator itAna;
        for(itAna = g_liAnalyzer.begin();itAna != g_liAnalyzer.end();itAna ++)
            (*itAna)->CallAnalyzer();
    }
}


bool CMasterCtrl::Receive(std::istrstream &in,std::ostrstream &out)
{
    char s[256];

    in >> std::setw(255) >> s;

    cout << "Main : Command " << s << " accepted." << endl;

    // Parse message string
    if(!strcmp(s,"Start"))
    {
        if(Status())return false;

        Start();
        return true;
    }
    else if(!strcmp(s,"Stop"))
    {
        int bBD;

        in >> bBD;

        if(!Status())return false;

        if(bBD)
            g_csy.BreakDown();
//            g_csy.Output();
        else
            Stop(false);
            return true;
    }
    else if(!strcmp(s,"AutoSave"))
    {
        int b;

        in >> b;

        m_pParam->bAutoSave = b;
        return true;
    }
    else if(!strcmp(s,"AutoStart"))
    {
        int b;

        in >> b;

        m_pParam->bAutoStart = b;
        return true;
    }
    else if(!strcmp(s,"Status"))
    {
        return Status();
    }
/*    else if(!strcmp(s,"Save"))
    {
        in >> std::setw(255) >> s;

        GetTime();
        Save(s);
        return true;
    }
    else if(!strcmp(s,"SaveBin"))
    {
        in >> std::setw(255) >> s;

        GetTime();
        SaveBin(s);
        return true;
    }
*/   else if(!strcmp(s,"Clear"))
    {
        Clear();
        return true;
    }
    else if(!strcmp(s,"GetLastFilename"))
    {
        if(m_pStrLastFilename == NULL)
        {
            out << endl;
            return false;
```

```
        }
        else
        {
            out << m_pStrLastFilename << endl;
            return true;
        }
    }
    else if(!strcmp(s,"SetInterlock"))
    {
        int nInterlock;
        in >> nInterlock;

        m_pParam->bInterlock = nInterlock;

        return !in.fail();
    }
    else if(!strcmp(s,"GetInterlock"))
    {
        out << m_pParam->bInterlock;

        return true;
    }
    else
        throw("Message invalid.");

    // not reach here;
    return false;
}


// Commands implemented

void CMasterCtrl::Start(void)
{
    // start camac first to avoid missing trigger
    g_camac.Start();

    // Analyzer Start
    std::list<CEventAnalyzer *>::iterator it;
    for(it = g_liAnalyzer.begin();it != g_liAnalyzer.end();it++){
        (*it)->Start();
    }

    // activate main loop
    m_bStatus = true;

    cout << "Start command successful." << endl;
}

void CMasterCtrl::Stop(bool bBreakDown)
{
    // Retrieve Filename for Autosave
```

```
struct tm tim;
GetTime(true,&tim);
char s[1024],s2[1024],s3[1024];
sprintf(s2,"%s/%%Y/%%m/%c%%Y%%m%%d%%H%%M%%S",m_pStrFilenameBase,bBreakDown?'b':'n');
strftime(s,1024,s2,&tim);

if(bBreakDown)
{
    // Notice breakdown to console
    if(m_pParam->bAutoSave)
    {
        sprintf(s3,"Breakdown\n%s",s);
        g_socket.Callback(s3);
    }
    else
        g_socket.Callback("Breakdown\nNoAutoSave");

    // Stop RF pulse
    if(m_pParam->bInterlock)
    {
        int nData = 1 << 15;    // 15ch
        g_camac.Naf(OUTREG,0,17,nData);
    }
}

// Stop sequence

// Status Change
m_bStatus = false;

// Stop camac TODO: maybe unnecessory
g_camac.Stop();

cout << "Camac stopped." << endl;

// Stop Analyzers
std::list<CEventAnalyzer *>::iterator it;
for(it = g_liAnalyzer.begin();it != g_liAnalyzer.end();it++){
    (*it)->Stop();
}

// Autosave
if(m_pParam->bAutoSave)
{
    // AutoSave
    sprintf(s3,"%s.bdmon-camac.bin",s);
    SaveBin(s3);

    // AutoSave
    sprintf(s3,"%s.bdmon-camac.periodic.bin",s);
    SaveBinPeriodic(s3);
```

```
        cout << "AutoSave finished." << endl;

        // AutoClear
        Clear();

        if(m_pParam->bAutoStart && bBreakDown)
        {
            // AutoStart
            Start();
            cout << "AutoStart finished." << endl;
        }
    }
}

void CMasterCtrl::Save(const char *pStrFilename)
{
    // Title of Pulse summary
    TString s("Pulse summary at ");
    s += GetTime(false);
    m_pTreeSummary->SetTitle(s);

    // Save Analyzers
    std::list<CEventAnalyzer *>::iterator it;
    for(it = g_liAnalyzer.begin();it != g_liAnalyzer.end();it++){
        (*it)->Save(pStrFilename);
    }

    // Write buffer file.
    gDirectory->Write("",m_bOverWrite ? TObject::kOverwrite : 0);

    if(pStrFilename){
        // copy buffer to specified file
        TString s;
        s += "cp ";
        s += m_pStrFilename;
        s += " ";
        s += pStrFilename;

        // copy !
        system(s);
    }
}

// Save Raw Data
void CMasterCtrl::SaveBin(const char * pStrFilename)
{
    // file saved to current file
    Save(NULL);

    // Save Analyzers : separated file
    std::list<CEventAnalyzer *>::iterator it;
    for(it = g_liAnalyzer.begin();it != g_liAnalyzer.end();it++){
        (*it)->SaveBin(pStrFilename);
    }

    std::ofstream out;
    out.open(pStrFilename);

    if(!out.is_open())throw("Failed to open binary file.");

    // Save Message
    out << "<MESSAGE>" << endl;
    out << (const char *)m_strMessage;
    out << endl;
    out << "</MESSAGE>" << endl;

    // Save Analyzers
    for(it = g_liAnalyzer.begin();it != g_liAnalyzer.end();it++){
        (*it)->SaveBin(&out);
    }

    delete[] m_pStrLastFilename;

    m_pStrLastFilename = new char[strlen(pStrFilename)+1];
    strcpy(m_pStrLastFilename,pStrFilename);
    cout << "Binary file saved to " << m_pStrLastFilename << endl;
}

void CMasterCtrl::SaveBinPeriodic(const char * pStrFilename)
{
    std::ofstream out;
    out.open(pStrFilename);

    if(!out.is_open())throw("Failed to open periodic binary file.");

    // Save Analyzers
    std::list<CEventAnalyzer *>::iterator it;
    for(it = g_liAnalyzer.begin();it != g_liAnalyzer.end();it++){
        (*it)->SaveBinPeriodic(&out);
    }

    cout << "Periodic binary file saved to " << pStrFilename << endl;
}


void CMasterCtrl::Clear(void)
{
    // Clear Sequence
    std::list<CEventAnalyzer *>::iterator it;
    for(it = g_liAnalyzer.begin();it != g_liAnalyzer.end();it++){
        (*it)->Clear();
    }

    m_pTreeSummary->Reset();
```

```
    m_strMessage = "";
}

void CMasterCtrl::SetBreakdownMessage(const char *pStrBreakdownMessage)
{
    if(m_strMessage != "")
        m_strMessage += ", ";
    m_strMessage += pStrBreakdownMessage;
}
```

A.6 CCamacCtrl.h

```
// CCamacCtrl.h - include cc7x00 library and read events from /dev/dc

#ifndef CCamacCtrl_INCLUDED
#define CCamacCtrl_INCLUDED

#ifndef WIN32
#include <unistd.h>
#include <sys/ioctl.h>
#else
#include <io.h>
#endif

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>

extern "C"{
#include "pcc2.h"
}

#include <iostream>
using namespace std;

#include "CSingleEvent.h"

class CCamacCtrl
{
public:
    class CCamacFrame{
        friend class CCamacCtrl;
    public:
        CCamacFrame(int nBufSize = 32770)
        {
            m_nBufSize = (nBufSize-2)/2;
            m_pBufSend = new int[nBufSize];
            m_pBufReceive = new int[nBufSize];

            Clear();
        }

        ~CCamacFrame(){delete[] m_pBufSend;delete[] m_pBufReceive;}

        void Clear()
        {
            cam_gen_init(m_nBufSize,m_pBufSend);
            cam_gen_init(m_nBufSize,m_pBufReceive);
        }

        int Put(int n,int a,int f,int data = 0)
        {
            int nRet;
            if((nRet = cam_gen_cc(m_pBufSend,n,a,f,data)) != 0){
                throw("CamacFrame creation failed at cam_gen_cc");
            }

            return m_pBufSend[1] - 1;
        }

        void Get(int n,int *data,int *q = 0, int *x = 0)
        {
            if(n >= m_pBufReceive[1])throw("Camac Receive data failed at cam_extract_cc");

            int stat = (m_pBufReceive[n*2+2] >> 24) & 0xff;
            *data = m_pBufReceive[n*2+2] & 0xffffff;
            if(q)
                *q = stat & 1;
            if(x)
                *x = stat & 2;

//            cout << stat << ',' << flush;
        }

        int GetSize()const
        {
            return m_pBufSend[1];
        }

    private:
        int m_nBufSize;
        int *m_pBufSend;
        int *m_pBufReceive;
    };

    CCamacCtrl(){m_fd = 0;m_binton = false;m_binit = false;}
    ~CCamacCtrl(){if(m_fd)close(m_fd);}
```

```
    // Camac ctrl commands
    void Init(int ncrate);
    void Start();
    void Stop();

    // Message Ctrl
    void Send(CCamacFrame *pFrame);
    void SendSingleCommand(int n,int a,int f,int data = 0,int *pResult = 0,int *q = 0,int *x = 0);
    // cc7x00 compatible
    int Naf(int n, int a, int f, int & data)
    {int nResult,q,x;SendSingleCommand(n,a,f,data,&nResult,&q,&x);data = nResult;return !q||!x;}
    int Naf(int n,int a,int f){int data = 0; return Naf(n,a,f,data);}

    // LAM Ctrl
    bool WaitTrig(void);

    void GetEventID(int &nEventID,int &nAcqID)const {nEventID = m_nEventID;nAcqID = m_nAcqID;}

    // CC7x00 compatible
    // 1 event read to CSingleEvent
    void ReadEvent();
    // from Analyzers
    const CSingleEvent & GetEvent(int nSlot){return m_arrEvents[nSlot];}

private:
    int m_fd;    // file descriptor for /dev/dc
    int m_crate;
    bool m_binit;
    bool m_binton;
    int m_nEventID,m_nAcqID;

    CCamacFrame m_frame[4];    // frame keep : {SCALER_XRAY,FADCCH13,FADCCH24,CLEAR}

    CSingleEvent m_arrEvents[25];    // for each slot

    void EnableNextAcq();        // Module clear
    void InitFrame();
public:
};

// the Unique instance
extern CCamacCtrl camac;

#endif
```

A.7 CCamacCtrl.cpp

```
// CCamacCtrl.cpp - include cc7x00 library and read events from /dev/dc

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#ifndef WIN32
#include <unistd.h>
#include <sys/ioctl.h>
#else
#include <io.h>
#endif

#include <stdlib.h>
// Standard library
#include <iostream>
#include <assert.h>
#include <vector>

// cc-net library
extern "C" {
#include "pcc2.h"
}

#include "Camac-slot.h"
#include <iostream>
using namespace std;
#include "CCamacCtrl.h"

#define MEASURE_SPEED2

void CCamacCtrl::Init(int ncrate)
{
    assert(!m_binit);

    m_crate = ncrate;

    // Device open
    m_fd = cam_open();
    if(m_fd == -1)throw("Failed to open CC/NET device at cam_open");
    cam_clear_fifo(m_fd);

    // Initialize Commands
    CCamacFrame fr;
    fr.Put(25,0,17);    // Z
    fr.Put(25,0,26);    // Inhibit on
    fr.Put(25,1,24);    // Interrupt disable
    fr.Put(25,2,24);    // Fast cycle disable
    fr.Put(25,1,16,0);    // LAM disable
```

```
    Send(&fr);

    InitFrame();

    m_binit = true;
}

void CCamacCtrl::Start()
{
    assert(m_binit);
    assert(!m_binton);

    // Inhibit off
    CCamacFrame fr;
    fr.Put(25,0,24);
    fr.Put(SCALER,0,9);    // Scaler clear
    fr.Put(SCALER,1,9);

    Send(&fr);

    // DAQ count clear
    int nData,nDaqStat;
    cam_single_daq(m_fd,DAQEXE_CTRL_CLRCNT,&nData,&nDaqStat);
    cam_single_daq(m_fd,DAQEXE_CTRL_CLRBSY,&nData,&nDaqStat);

    m_nAcqID = m_nEventID = 0;

    m_binton = true;

    EnableNextAcq();
}

void CCamacCtrl::Stop()
{
    assert(m_binit);
    assert(m_binton);

    // Inhibit on
    SendSingleCommand(25,0,26);

    m_binton = false;
}

void CCamacCtrl::Send(CCamacCtrl::CCamacFrame *pFrame)
{
    int nRet;
    if((nRet = cam_exec(m_fd,pFrame->m_pBufSend,pFrame->m_pBufReceive)) != 0){
        throw("Camac execution failed at cam_exec");
    }
}

void CCamacCtrl::SendSingleCommand(int n,int a,int f,int data,int *pResult,int *q,int *x)
{
    int qq=0,xx=0,nRet,data2=0;
    data2 = data;

    if((nRet = cam_single_cc(m_fd,n,a,f,&data2,&qq,&xx)) != 0)
    {
        cout << "nRet = " << nRet << endl;
        throw("Camac execution failed.");
    }
    if(pResult)
        *pResult = data2;
    if(q)
        *q = qq;
    if(x)
        *x = xx;
}

bool CCamacCtrl::WaitTrig(void)
{
#ifdef MEASURE_SPEED2
    SendSingleCommand(OUTREG,0,16,0);
#endif
    bool b = !cam_wait_trig(m_fd,&m_nEventID,0);
    m_nAcqID ++;

    return b;
}

// 1 event read
void CCamacCtrl::ReadEvent()
{
    // GLCTA bdmonitor Specific !!
    int i;
    int nn;

    assert(m_binit);
    assert(m_binton);

    // Wait Next Trigger
    if(!WaitTrig()){
        cerr << "Trigger wait failed." << endl;
        return;
    }

#ifdef MEASURE_SPEED2
    SendSingleCommand(OUTREG,0,17,1<<12);
#endif

    int nEventID,nAcqID,nResult,q,x;

    // Scaler & X-ray frame
    CCamacFrame &fr1 = m_frame[0];
```

```
    Send(&fr1);

    int n = 0;

    // Scaler
    fr1.Get(n++,&nEventID,&q,&x);
    if(!q || !x)cout << "Failed to get event ID!" << endl;
//    cout << "EventID = " << nEventID << endl;
    fr1.Get(n++,&nAcqID,&q,&x);
    if(!q || !x)cout << "Failed to get acq. ID!" << endl;
//    cout << "AcqID = " << nAcqID << endl;

    // X-ray ADC
    {
        CSingleEvent &ev = m_arrEvents[XRAYADC_1];
        ev.m_nEventID = nEventID;
        ev.m_nAcqID = nAcqID;
        ev.m_nBufSize = 12;

        nn = 0;
        for(i=0;i<12;i++){
            fr1.Get(n++,&nResult,&q,&x);
            if(!x || !q)nn++;
            ev.m_pBuf[i] = (short)nResult;
        }
        if(nn)cout << "X-ray ADC Ch1 No-q or No-x count : " << nn << endl;
    }
    {
        CSingleEvent &ev = m_arrEvents[XRAYADC_2];
        ev.m_nEventID = nEventID;
        ev.m_nAcqID = nAcqID;
        ev.m_nBufSize = 12;

        nn = 0;
        for(i=0;i<12;i++){
            fr1.Get(n++,&nResult,&q,&x);
            if(!x || !q)nn++;
            ev.m_pBuf[i] = (short)nResult;
        }
        if(nn)cout << "X-ray ADC Ch2 No-q or No-x count : " << nn << endl;
    }
    {
        // X-ray TDC
        CSingleEvent &ev = m_arrEvents[XRAYTDC_1];
        ev.m_nEventID = nEventID;
        ev.m_nAcqID = nAcqID;
        ev.m_nBufSize = 8;

        nn=0;
        for(i=0;i<8;i++){
            fr1.Get(n++,&nResult,&q,&x);
            if(!x || !q)nn++;
            ev.m_pBuf[i] = (short)nResult;
        }
        if(nn)cout << "X-ray TDC Ch1 No-q or No-x count : " << nn << endl;
    }
    {
        CSingleEvent &ev = m_arrEvents[XRAYTDC_2];
        ev.m_nEventID = nEventID;
        ev.m_nAcqID = nAcqID;
        ev.m_nBufSize = 8;

        nn = 0;
        for(i=0;i<8;i++){
            fr1.Get(n++,&nResult,&q,&x);
            if(!x || !q)nn++;
            ev.m_pBuf[i] = (short)nResult;
        }
        if(nn)cout << "X-ray TDC Ch2 No-q or No-x count : " << nn << endl;
    }

    // RF FADC ////////////////////////////////////////////////
    int nReadSize = FADC500_DATASIZE;
    int nBufSize = nReadSize*2+4;

    // Wait Lam
    q = 0;
    n = 0;
    while(!q || !x){
        SendSingleCommand(FADC500_1,0,8,0,&nResult,&q,&x);
        n++;
    }
//    cout << n << endl;

    CCamacFrame &fr2 = m_frame[1];
    n = 0;
    Send(&fr2);

    CSingleEvent &ev = m_arrEvents[FADC500_1];
    ev.m_nEventID = nEventID;
    ev.m_nAcqID = nAcqID;
    ev.m_nBufSize = nBufSize;
    ev.m_pBuf[0] = 0;
    ev.m_pBuf[1] = nReadSize;

    CSingleEvent &ev2 = m_arrEvents[FADC500_2];
    ev2.m_nEventID = nEventID;
    ev2.m_nAcqID = nAcqID;
    ev2.m_nBufSize = nBufSize;
    ev2.m_pBuf[0] = 2;
    ev2.m_pBuf[1] = nReadSize;

    nn = 0;
    for(i=0;i<nReadSize;i++){
```

```
        fr2.Get(n++,&nResult,&q,&x);
        if(!x || !q)nn++;
        ev.m_pBuf[i+2] = (short)nResult;
    }
    if(nn)cout << "RF FADC Ch1 No-q or No-x count : " << nn << endl;
    nn = 0;
    for(i=0;i<nReadSize;i++){
        fr2.Get(n++,&nResult,&q,&x);
        if(!x || !q)nn++;
        ev2.m_pBuf[i+2] = (short)nResult;
    }
    if(nn)cout << "RF FADC Ch2 No-q or No-x count : " << nn << endl;

    // Fire output register to read rest channel
    SendSingleCommand(OUTREG,0,17,0xc3);    // port 1,2,7,8

    // Wait "q"

    q = 0;
    n = 0;
    while(!q || !x){
        SendSingleCommand(FADC500_1,1,8,0,&nResult,&q,&x);
        n++;
    }
//    cout << n << endl;

    // read rest channel
    n = 0;
    CCamacFrame &fr4 = m_frame[2];
    Send(&fr4);

    ev.m_pBuf[2+nReadSize] = 1;
    ev.m_pBuf[3+nReadSize] = nReadSize;
    ev2.m_pBuf[2+nReadSize] = 3;
    ev2.m_pBuf[3+nReadSize] = nReadSize;

    nn = 0;
    for(i=0;i<nReadSize;i++){
        fr4.Get(n++,&nResult,&q,&x);
        if(!x || !q)nn++;
        ev.m_pBuf[4+nReadSize+i] = (short)nResult;
    }
    if(nn)cout << "RF FADC Ch1 No-q or No-x count : " << nn << endl;
    nn=0;
    for(i=0;i<nReadSize;i++){
        fr4.Get(n++,&nResult,&q,&x);
        if(!x || !q)nn++;
        ev2.m_pBuf[4+nReadSize+i] = (short)nResult;
    }
    if(nn)cout << "RF FADC Ch2 No-q or No-x count : " << nn << endl;

#ifdef MEASURE_SPEED2
    SendSingleCommand(OUTREG,0,16,1<<12);
#endif

    // enable next acq
    EnableNextAcq();
}

void CCamacCtrl::EnableNextAcq()
{
    Send(&m_frame[3]);

    int nData,nDaqStat;
    cam_single_daq(m_fd,DAQEXE_CTRL_CLRBSY,&nData,&nDaqStat);
    cam_enable_trig(m_fd);
}

void CCamacCtrl::InitFrame()
{
    int i;

    // Scaler & X-ray frame ////////////////////////////
    m_frame[0].Clear();
    // SCALER
    m_frame[0].Put(SCALER,0,0);
    m_frame[0].Put(SCALER,1,0);

    // X-ray ADC
    for(i=0;i<12;i++)m_frame[0].Put(XRAYADC_1,i,0);
    for(i=0;i<12;i++)m_frame[0].Put(XRAYADC_2,i,0);

    // X-ray TDC
    for(i=0;i<8;i++)m_frame[0].Put(XRAYTDC_1,i,0);
    for(i=0;i<8;i++)m_frame[0].Put(XRAYTDC_2,i,0);

    // FADC frame Ch 1 & 3 ////////////////////////////////
    m_frame[1].Clear();

    for(i=0;i<FADC500_DATASIZE;i++)m_frame[1].Put(FADC500_1,0,0);
    for(i=0;i<FADC500_DATASIZE;i++)m_frame[1].Put(FADC500_2,0,0);

    // FADC frame Ch 2 & 4 ////////////////////////////////
    m_frame[2].Clear();

    for(i=0;i<FADC500_DATASIZE;i++)m_frame[2].Put(FADC500_1,1,0);
    for(i=0;i<FADC500_DATASIZE;i++)m_frame[2].Put(FADC500_2,1,0);

    // Clear frame ////////////////////////////////////////
    m_frame[3].Clear();

    // ADCs & TDCs for X-ray
    m_frame[3].Put(XRAYADC_1,0,9);
    m_frame[3].Put(XRAYADC_2,0,9);
```

```
    m_frame[3].Put(XRAYTDC_1,0,9);
    m_frame[3].Put(XRAYTDC_2,0,9);

    // FADCs for RF
    m_frame[3].Put(FADC500_1,0,9);
    m_frame[3].Put(FADC500_2,0,9);
    m_frame[3].Put(FADC500_1,0,25);
    m_frame[3].Put(FADC500_2,0,25);
}
```

A.8 pcc2.h

```
// pcc2.h - including prototypes required for C++

#ifndef PCC2_H_INCLUDE
#define PCC2_H_INCLUDE

#include "ccnet/pcc.h"

// Prototypes
void dump_pccreg(struct pccreg *pccreg);
int cam_naf( int n, int a, int f );
int gen_cam_command(int *data1, int *data2, int n, int a, int f, int data, int flag );
int gen_daq_command(int *data1, int *data2, int func, int flag );
int cam_gen_init( int length, int* buf );
int cam_gen_cc( int *buf, int n, int a, int f, int data );
int cam_gen_daq( int* buf, int cmd, int data );
void cam_decode_cc_frame( int data1, int data2, int* n, int* a, int* f, int* data, int* status );
int cam_extract_cc_data( int* rplybuf, int len, int* actuallen, int* data );
int cam_extract_cc_status( int* rplybuf, int len, int* actuallen, int* status );
int cam_extract_cc_qx( int status, int* q, int* x );
int cam_extract_daq_data( int* rplybuf, int len, int* actuallen, int* data );
int cam_open( void );
int cam_close( int fd );
int cam_single_cc( int fd, int n, int a, int f, int *data, int *q, int *x );
int cam_single_daq( int fd, int func, int *data, int *daq_status );
int cam_reset( int fd );
int cam_clear_fifo( int fd );
int cam_dump_pccreg( int fd, struct pccreg *pccreg );
int cam_get( int fd, int* data, int* rply );
int cam_getint( int fd, int* data, int* rply );
int cam_put(int fd, int data, int cmd );
int cam_exec_pio( int fd, int* cmdbuf, int* rplybuf );
int cam_exec_dma( int fd, int* cmdbuf, int* rplybuf );
int cam_exec_dma_seq( int fd, int* cmdbuf, int* rplybuf );
int cam_exec( int fd, int* cmdbuf, int* rplybuf );
int cam_enable_lam( int fd, int enable_pattern );
int cam_wait_lam( int fd, int* lam_pattern, int timeout );
int cam_disable_lam( int fd );
int cam_enable_trig( int fd );
int cam_wait_trig( int fd, int* event_count, int timeout );
int cam_disable_trig( int fd );

#endif
```

A.9 CSingleEvent.h

```
// CSingleEvent.h
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_CSINGLEEVENT_H__1F92C521_9D72_4952_AE04_558B0A5797EA__INCLUDED_)
#define AFX_CSINGLEEVENT_H__1F92C521_9D72_4952_AE04_558B0A5797EA__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <memory.h>

class CSingleEvent
{
public:
    CSingleEvent();
    virtual ~CSingleEvent();
private:
    CSingleEvent(const CSingleEvent &ref){*this = ref;}
    CSingleEvent & operator =(const CSingleEvent &ref)
      {
        m_nEventID = ref.m_nEventID;
        m_nAcqID = ref.m_nAcqID;
        m_nBufSize = ref.m_nBufSize;
        memcpy(m_pBuf,ref.m_pBuf,m_nBufSize * sizeof(short));

        return *this;
      }
public:
    // Event ID (Count whether event data were acquisited or not)
    int m_nEventID;
    // Acquisition ID (Count only when event data were acquisited)
    int m_nAcqID;

    // Contains Event Data : Acsess from branch class
    short m_pBuf[32768];
    int m_nBufSize;
```

```
};

#endif // !defined(AFX_CSINGLEEVENT_H__1F92C521_9D72_4952_AE04_558B0A5797EA__INCLUDED_)
```

A.10 CSingleEvent.cpp

```
// CSingleEvent.cpp
//////////////////////////////////////////////////////////////////////

#include "CSingleEvent.h"
#include "stdlib.h"

CSingleEvent::CSingleEvent()
{
    m_nBufSize = 0;
    m_nAcqID = 0;
    m_nEventID = 0;
}

CSingleEvent::~CSingleEvent()
{
}
```


A.11 CCsyCtrl.h

```
// CCsyCtrl.h
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_CCSYCTRL_H__4EC858D3_C5BC_4993_873A_5A8F9F29AF05__INCLUDED_)
#define AFX_CCSYCTRL_H__4EC858D3_C5BC_4993_873A_5A8F9F29AF05__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "CEventAnalyzer.h"
#include "CSocketReceiver.h"
#include "streamers.h"

using namespace std;

class TFile;
class CCsyCtrl : public CEventAnalyzer , public CSocketReceiver
{
public:
    CCsyCtrl();
    virtual ~CCsyCtrl();

private:
    bool m_bBreakDown;
    bool m_bEnterBD;
    CCsyParam *m_pParam;
public:
    virtual void Start(void);
    virtual void Stop(void);
    virtual void Analyze(void);
    virtual bool Receive(istrstream &in,ostrstream &out);
    virtual void Initialize(TFile *pRecent);
    void BreakDown(void);
    bool InBreakDownSequence(void){return m_bBreakDown;}
    virtual void Save(const char * pStrFilename){m_pParam->Write();}

    void Output();
    int m_nAfterRunRemain[4];

    char *m_pStrBreakdownMessage;
};

#endif // !defined(AFX_CCSYCTRL_H__4EC858D3_C5BC_4993_873A_5A8F9F29AF05__INCLUDED_)
```

A.12 CCsyCtrl.cpp

```
// CCsyCtrl.cpp 椿
//////////////////////////////////////////////////////////////////////

#include "bdmonitor.h"
#include "CCamacCtrl.h"
#include "CCsyCtrl.h"
#include "Camac-slot.h"
#include "TFile.h"
#include <iostream>

CCsyCtrl::CCsyCtrl()
{
    // Add to Receiver
    AddReceiver(this);
    SetTargetString("Trigger");
```

```
    SetAnalyzeEnabled(true);

    m_bBreakDown = false;
    m_bEnterBD = false;
}

CCsyCtrl::~CCsyCtrl()
{
}

void CCsyCtrl::Initialize(TFile *pRecent)
{
    // Add to Analyzer
    AddAnalyzer(this);

    // Retrieve parameter
    m_pParam = GetParamFromFile<CCsyParam>("CsyParam",pRecent);

    if(GetAnalyzeEnabled())
    {
        g_camac.Naf(CSY,0,26);
        g_camac.Naf(CSY2,0,26);
    }
    else
    {
        g_camac.Naf(CSY,0,24);
        g_camac.Naf(CSY2,0,24);
    }
}

void CCsyCtrl::Start(void)
{
    if(GetAnalyzeEnabled())
    {
        int data = 15;
        // CSY2 : through mode
        g_camac.Naf(CSY2,0,24);

        // First Pulse Enable
        for(int i = 0;i<4;i++)
            g_camac.Naf(CSY,i,16,data);

    }
    SetIdFinished(1);
}


void CCsyCtrl::Stop(void)
{
    // CSY2 Control Mode
    g_camac.Naf(CSY2,0,26);
}

void CCsyCtrl::Analyze(void)
{
    int nID = GetIdFinished();

    // Check if analyze finished
    std::list<CEventAnalyzer *>::iterator it;
    for(it = g_liAnalyzer.begin();it != g_liAnalyzer.end();it++){
        if((*it)->GetAnalyzeEnabled() && ((*it)->GetIdFinished() < nID))
        {
            return;
        }
    }

    // Analyze finished

    // Fill Pulse Summary
    g_master.FillSummary();

    if(m_bEnterBD)
    {
        cout << "Entering breakdown mode: AfterRun = " << m_pParam->nAfterRun[0] << endl;

        int i;
        // AfterRun Set
        for(i=0;i<4;i++)
            m_nAfterRunRemain[i] = m_pParam->nAfterRun[i];

        m_bBreakDown = true;
        m_bEnterBD = false;
    }
    if(m_bBreakDown)
    {
        cout << "Breakdown sequence:" << flush;

        // Breakdown mode : check remain
        for(int i=0;i<4;i++){
            int data = 15;
            if(m_nAfterRunRemain[i] > 0)
                g_camac.Naf(CSY,i,16,data);
            m_nAfterRunRemain[i] --;
        }
        cout << "Remain = " << m_nAfterRunRemain[0] << endl;

        // AfterRun Finish Check
        if(m_nAfterRunRemain[0] < 0)
        {
            // AfterRun Finished

            // exit breakdown mode
```

```
            m_bBreakDown = false;

            g_master.Stop(true);    // Normal stop

            cout << "Breakdown sequence finished." << endl;

            return;
        }
        else
            // not necessary , for security
            g_camac.Naf(CSY2,0,24);
    }
    else
    {
        // Enable next pulse
        int data = 15;
        for(int i = 0;i<4;i++){
            g_camac.Naf(CSY,i,16,data);
        }
        // not necessary , for security
        g_camac.Naf(CSY2,0,24);

        data = 1 << 14;
        g_camac.Naf(OUTREG,0,17,data);
    }
    SetIdFinished(nID + 1);

    cout << "." << flush;
//    printf("Event #%d Analyze finished.\n",nID);
}


bool CCsyCtrl::Receive(istrstream &in,ostrstream &out)
{
    char s[256];

    in >> std::setw(255) >> s;

    cout << "Trigger : Command " << s << " accepted." << endl;

    // Parse message string
    if(!strcmp(s,"Enable"))
    {
        bool bEnable;
        in >> bEnable;

        if(GetAnalyzeEnabled() == bEnable)
            return false;    // meaningless

        g_camac.Naf(CSY,0,bEnable ? 24 : 26);

        SetAnalyzeEnabled(bEnable);

        m_pParam->bBDdetection = bEnable;

        return true;
    }
    else if(!strcmp(s,"GetEnable"))
    {
        return GetAnalyzeEnabled();
    }
    else if(!strcmp(s,"SetAfterRun"))
    {
        for(int i=0;i<4;i++){
            in >> m_pParam->nAfterRun[i];
            cout << "SetAfterRun: nAfterRun[" << i << "] = "
                 << m_pParam->nAfterRun[i] << endl;
        }

        return !in.fail();
    }
    else if(!strcmp(s,"GetAfterRun"))
    {
        for(int i=0;i<4;i++){
            out << m_pParam->nAfterRun[i] << endl;
            cout << "GetAfterRun: nAfterRun[" << i << "] = "
                << m_pParam->nAfterRun[i] << endl;
        }

        return true;
    }
    else
        throw("Message invalid.");

    // not reach here;
    return false;
}

void CCsyCtrl::Output()
{
    int i;
    for(i=0;i<4;i++)
        g_camac.Naf(CSY,i,0);

    return;
}

void CCsyCtrl::BreakDown(void)
{
    if(!m_pParam->bBDdetection){
        cout << "Breakdown occurred,but BDdetection flag not set." << endl;
        return;    // No-op.
    }
    if(m_bBreakDown)
        return;    // Already in BreakDown mode.
```

```
    // BreakDown Mode on.
    m_bEnterBD = true;
}
```

A.13 CEventAnalyzer.h

```
// CEventAnalyzer.h
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_CEVENTANALYZER_H__3E91FE6B_D4AA_4DC8_9F60_CEB6F6B978C3__INCLUDED_)
#define AFX_CEVENTANALYZER_H__3E91FE6B_D4AA_4DC8_9F60_CEB6F6B978C3__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <stdio.h>
#include <fstream>

#include "TDirectory.h"
#include "streamers.h"

class CEventAnalyzer
{
public:
    virtual void Initialize(TDirectory *pFile,TDirectory *pRecent,CPulseSummary *pSummary){}
    virtual void Clear(){}
    virtual void Start(){ClearIdFinished();printf("ID cleared.\n");}
    virtual void Stop(){}
    virtual void Save(const char *pStrFilename){}
    virtual void SaveBin(const char *pStrFilename){}
    virtual void SaveBin(std::ostream *pout){}
    virtual void SaveBinPeriodic(std::ostream *pout){}

    CEventAnalyzer(){m_bEnabled = true;}
    virtual ~CEventAnalyzer(){}

    void SetAnalyzeEnabled(bool b){m_bEnabled = b;}
    bool GetAnalyzeEnabled()const{return m_bEnabled;}

    void CallAnalyzer(){if(m_bEnabled)Analyze();}

    void ClearIdFinished(){m_nIdFinished = 0;}
    int GetIdFinished()const{return m_nIdFinished;}
protected:
    virtual void Analyze() = 0;
    void SetIdFinished(int nID){m_nIdFinished = nID;}

private:
    bool m_bEnabled;
    int m_nIdFinished;
};

#endif // !defined(AFX_CEVENTANALYZER_H__3E91FE6B_D4AA_4DC8_9F60_CEB6F6B978C3__INCLUDED_)
```

A.14 CSocketCtrl.h

```
#ifndef CSOCKETCTRL_H_INC
#define CSOCKETCTRL_H_INC

#include <list>

class CSocketCtrl
{
public:
    CSocketCtrl(void);
    virtual ~CSocketCtrl(void);
    void Initialize(void);

private:
    int m_nSock;    // Acceptor socket descriptor
    int m_nSockCallback;
    std::list<int> m_liSocAccepted;
    std::list<int> m_liSocCallback;

    int m_nIDCount;    // atf_socket_buffer ID
public:
    void CheckSocket(void);
    void Callback(const char *pStr);
};

#endif
```

A.15 CSocketCtrl.cpp

```
// CSocketCtrl.cpp
```

```
#define SOCKET_PORT_NO 5555
#define SOCKET_PORT_NO_CALLBACK 5556

#include "CSocketCtrl.h"

extern "C"
{
#include "atf_socket.h"
}

#include "bdmonitor.h"
#include "CSocketReceiver.h"
#include <sstream>
#include <iostream>
#include <iomanip>

using namespace std;

CSocketCtrl::CSocketCtrl(void)
{
}

CSocketCtrl::~CSocketCtrl(void)
{
}

void CSocketCtrl::Initialize(void)
{
    // open acceptor soccket
    if(atf_socket_server_open(&m_nSock,SOCKET_PORT_NO) != IATF_SOCKET_NORMAL)
        throw("Acceptor socket cannot opened.");

    if(atf_socket_server_open(&m_nSockCallback,SOCKET_PORT_NO_CALLBACK) != IATF_SOCKET_NORMAL)
        throw("Acceptor socket cannot opened.");

    m_nIDCount = 0;
}

void CSocketCtrl::CheckSocket(void)
{
    const int nTimeout = 1;

    int nNewSock;
    // accept new connection
    if(atf_socket_server_accept(m_nSock,&nNewSock) == IATF_SOCKET_NORMAL)
    {
        // new connection arrived.
        m_liSocAccepted.push_back(nNewSock);

        cout << "Socket attached. Socket # = " << nNewSock << endl;
    }
    if(atf_socket_server_accept(m_nSockCallback,&nNewSock) == IATF_SOCKET_NORMAL)
    {
        // new connection arrived.
        m_liSocCallback.push_back(nNewSock);

        cout << "Callback socket attached. Socket # = " << nNewSock << endl;
    }

    // Check each sockets connected.
    std::list<int>::iterator itSoc;
    bool bInc = true;
    for(itSoc = m_liSocAccepted.begin();itSoc != m_liSocAccepted.end();bInc ? itSoc++ : 0)
    {
        int nSock = *itSoc;
        bInc = true;

        SOCKET_BUFFER buf_read,buf_write;
        int nErr;
        int nRet;

        nRet = atf_socket_read(nSock,&buf_read,0,&nErr);
        // atf_socket_confirm necessory?

        if(nRet == IATF_SOCKET_TIMEOUT)
        {
            continue;
        }
        else if(nRet == IATF_SOCKET_DISCONNECTED)
        {
            // socket detached.
            itSoc = m_liSocAccepted.erase(itSoc);
            bInc = false;

            cout << "Socket detached." << endl;

            continue;
        }
        else if(nRet != IATF_SOCKET_NORMAL)throw("Socket read error.");

        cout << "Message arrived. Size = " << buf_read.header.nbyte << " bytes." << endl;

        // data arrived.
        // stream for input
        std::istrstream input(buf_read.data.c,buf_read.header.nbyte - MX_HEADER_BYTE);
        // stream for output
        std::ostrstream output(buf_write.data.c,MX_C_ARRAY_SIZE);

        cout << "Arrived data: " << (buf_read.data.c) << endl;
        // check header
        unsigned short sHeader;
        input >> sHeader;
        if(sHeader != 0xbd00)throw("Invalid message header arrived.");
```

```
// get message category
short nCategory;
input >> nCategory;
if(nCategory > 3 || nCategory < 1)throw("Invalid message category arrived.");

// get message ID
int nID;
input >> nID;

// category name must be < 254 bytes
char s[256];

switch(nCategory)
{
case 1:{    // Request
    input >> std::setw(255) >> s;    // Limit size

    cout << "Message Category = " << s << endl;

    // Create header for response message
    output << (unsigned short)0xbd00 << endl;
    output << (short)2 << endl;    // response
    output << (int)nID << endl;    // Message ID

    // search matched category
    std::list<CSocketReceiver *>::iterator it;
    for(it = g_liReceiver.begin();it != g_liReceiver.end();it++)
    {
        if((*it)->IsTarget(s))
        {
            bool b;
            try{
                // category matched.
                b = (*it)->Receive(input,output);
            }
            catch(const char *pStr)
            {
                printf(pStr);
                b = false;
            }

            // message footer
            output << (unsigned short)(b ? 0xbdff : 0xbdfe) << endl;

            // loop exit
            break;
        }
    }

    if(it == g_liReceiver.end()){
        // message not found.
        // error response back.
        output << (unsigned short)0xbdfe << endl;
    }

    // send response
    buf_write.header.code = HEADER_CODE;
    buf_write.header.id = (m_nIDCount ++) % MX_BUFFER_ID;
    buf_write.header.nbyte = output.pcount() + MX_HEADER_BYTE;
    buf_write.header.type = DATA_TYPE_STRING;

    nRet = atf_socket_write(nSock,&buf_write,nTimeout,&nErr);
    if(nRet != IATF_SOCKET_NORMAL)throw("Socket write error.");

    // atf_socket_confirm necessory?

    break;
}
case 2:    // response
case 3: // callback
    throw("Message category 2,3 is currently not accepted.");
}

// Receive footer
unsigned short sFooter;
input >> sFooter;
if(sFooter != 0xbdff)throw("Invalid message footer arrived.");
}

}

void CSocketCtrl::Callback(const char *pStr)
{
    SOCKET_BUFFER buf_read,buf_write;
    int nErr;
    int nRet;
    const int nTimeout = 100;
    static int nID = 0;

    // stream for output
    std::ostrstream output(buf_write.data.c,MX_C_ARRAY_SIZE);

    // Create header for callback message
    output << (unsigned short)0xbd00 << endl;
    output << (short)3 << endl;    // Callback
    output << (int)++nID << endl;    // Message ID

    output << pStr << endl;    // Callback Message;
    output << (unsigned short)0xbdff << endl;    // message footer

    buf_write.header.code = HEADER_CODE;
    buf_write.header.id = (m_nIDCount ++) % MX_BUFFER_ID;
    buf_write.header.nbyte = output.pcount() + MX_HEADER_BYTE;
```

```
buf_write.header.type = DATA_TYPE_STRING;


std::list<int>::iterator itSoc;
bool bInc = true;
for(itSoc = m_liSocCallback.begin();itSoc != m_liSocCallback.end();bInc ? itSoc++ : 0)
{
    int nSock = *itSoc;
    bInc = true;

    // write to the socket
    nRet = atf_socket_write(nSock,&buf_write,nTimeout,&nErr);

    if(nRet != IATF_SOCKET_NORMAL)
    {
        // socket detached.
        itSoc = m_liSocCallback.erase(itSoc);
        bInc = false;

        if(nRet == IATF_SOCKET_DISCONNECTED)
            cout << "Callback Socket detached." << endl;
        else
            cout << "Callback Socket error. Maybe detached." << endl;

        continue;
    }

    cout << "Callback " << pStr << " sent." << endl;

    // Wait for reply
    nRet = atf_socket_read(nSock,&buf_read,nTimeout,&nErr);
    if(nRet == IATF_SOCKET_TIMEOUT)
    {
        cout << "No callback reply." << endl;
        continue;
    }
    else if(nRet == IATF_SOCKET_DISCONNECTED)
    {
        // socket detached.
        itSoc = m_liSocCallback.erase(itSoc);
        bInc = false;

        cout << "Callback Socket detached." << endl;

        continue;
    }
    else if(nRet != IATF_SOCKET_NORMAL){
        cerr << "Socket read error.nRet= " << nRet << " , nErr= " << nErr << endl;
        throw("Socket error. breakdown sequence stopped.");
    }
    cout << "Arrived data: " << (buf_read.data.c) << endl;

        // stream for input
        std::istrstream input(buf_read.data.c,buf_read.header.nbyte - MX_HEADER_BYTE);
        // check header
        unsigned short sHeader;
        input >> sHeader;
        if(sHeader != 0xbd00)throw("Invalid message header arrived.");

        // get message category
        short nCategory;
        input >> nCategory;
        if(nCategory != 4)throw("Invalid message category arrived.");

        // get message ID
        int nIDRecv;
        input >> nIDRecv;
        if(nIDRecv != nID) throw("Invalid message ID.");

        // Receive footer
        unsigned short sFooter;
        input >> sFooter;
        if(sFooter == 0xbdfe) throw("Reply message says NG.");
        if(sFooter != 0xbdff) throw("Invalid message footer.");

        cout << "Callback reply received." << endl;
    }
}
```

A.16 atf_socket.h - modified from original

```
/* atf_socket.h */

/*
!============================================================================
    The parameter currently used by socket function
!============================================================================
*/

#define DATA_TYPE_STRING        0
#define DATA_TYPE_INT           1
#define DATA_TYPE_FLOAT         2
#define DATA_TYPE_SHORT         3

/*
```

```
!================================================================================
    socket data structure.
!================================================================================
*/

#define HEADER_CODE             123456
#define MX_BUFFER_ID            10000

#define MX_BUFFER_BYTE          65535
#define MX_HEADER_BYTE          128
#define MX_DATA_BYTE            MX_BUFFER_BYTE - MX_HEADER_BYTE

#define MX_I_ARRAY_SIZE         16351   // MX_DATA_BYTE / sizeof(int)
#define MX_F_ARRAY_SIZE         16351   // MX_DATA_BYTE / sizeof(float)
#define MX_S_ARRAY_SIZE         32703   // MX_DATA_BYTE / sizeof(short)
#define MX_C_ARRAY_SIZE         65407   // MX_DATA_BYTE / sizeof(char)

typedef struct {

    struct {
    int        code;
    unsigned int    id;
    int        receipt_state;
    int        type;
    int        nbyte;
    char        reserve[108];
    } header;

    union {
        int             i[MX_I_ARRAY_SIZE];
        float           f[MX_F_ARRAY_SIZE];
        short           s[MX_S_ARRAY_SIZE];
        char            c[MX_C_ARRAY_SIZE];
    } data;

} SOCKET_BUFFER;

/*
!================================================================================
    socket function
!================================================================================
*/

// ANSI-Compatible prototype declaration added by Taikan SUEHARA 2003/10/21
// since the old-type declaration is not compatible with C++.

int atf_socket_server_open(int *sock, int port);
int atf_socket_server_accept(int sock, int *nsock);

int atf_socket_client_connection();

int atf_socket_ready();
int atf_socket_empty();
int atf_socket_confirm();
int atf_socket_close();

int atf_socket_read(int sock,SOCKET_BUFFER *buffer,int timeout,int *err);
int atf_socket_read_int();
int atf_socket_read_float();
int atf_socket_read_short();
int atf_socket_read_string();

int atf_socket_write(int sock,SOCKET_BUFFER *write_data,int timeout, int *err );
int atf_socket_write_int();
int atf_socket_write_float();
int atf_socket_write_short();
int atf_socket_write_string();

/*
!================================================================================
    socket function return code.
!================================================================================
*/

#define     IATF_SOCKET_NORMAL          0x00000001

#define     IATF_SOCKET_ERROR           0x00000000
#define     IATF_SOCKET_TIMEOUT         0x00000002
#define     IATF_SOCKET_DISCONNECTED    0x00000004
#define     IATF_SOCKET_INVALID_DATA    0x00000008
#define     IATF_SOCKET_INVALID_TYPE    0x00000010
#define     IATF_SOCKET_ECONNREFUSED    0X00000020

#define     IATF_SOCKET_INVALID_ARG1    0x00000040
#define     IATF_SOCKET_INVALID_ARG2    0x00000080
#define     IATF_SOCKET_INVALID_ARG3    0x00000100
#define     IATF_SOCKET_INVALID_ARG4    0x00000200
#define     IATF_SOCKET_INVALID_ARG5    0x00000400
```

A.17 CSocketReceiver.h

```
// CSocketReceiver.h
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_CSOCKETRECEIVER_H__5EA9A07A_DE15_4F1A_992B_A758B03B43D2__INCLUDED_)
#define AFX_CSOCKETRECEIVER_H__5EA9A07A_DE15_4F1A_992B_A758B03B43D2__INCLUDED_

#if _MSC_VER > 1000
#pragma once
```

```
#endif // _MSC_VER > 1000

#include <strstream>
#include <iomanip>
using namespace std;

class CSocketReceiver
{
public:
    CSocketReceiver();
    virtual ~CSocketReceiver();
    bool IsTarget(const char *pStr);

    // Callback function : you need to override here.
    // called when socket with your target arrived.
    virtual bool Receive(istrstream &in,ostrstream &out) = 0;
protected:
    void SetTargetString(const char *pStrTarget);
private:
    char * m_pStrTarget;
};

#endif // !defined(AFX_CSOCKETRECEIVER_H__5EA9A07A_DE15_4F1A_992B_A758B03B43D2__INCLUDED_)
```


A.18 CSocketReceiver.cpp

```
// CSocketReceiver.cpp
//////////////////////////////////////////////////////////////////////

#include <string.h>
#include "CSocketReceiver.h"

CSocketReceiver::CSocketReceiver()
{
    m_pStrTarget = NULL;
}

CSocketReceiver::~CSocketReceiver()
{
    delete[] m_pStrTarget;
}

bool CSocketReceiver::IsTarget(const char *pStr)
{
    if(!m_pStrTarget)
        return false;
    else
        return !strcmp(pStr,m_pStrTarget);
}

void CSocketReceiver::SetTargetString(const char *pStrTarget)
{
    delete[] m_pStrTarget;

    m_pStrTarget = new char[strlen(pStrTarget)+1];
    strcpy(m_pStrTarget,pStrTarget);
}
```

A.19 streamers.h

```
// streamers.h structures of streamers
#ifndef STREAMERS_H_INC
#define STREAMERS_H_INC

#include "TObject.h"
#include "TArrayI.h"
#include <limits.h>

class peakRelative_t : public TObject
{
public:
    int use;
    int rangeMin;
    int rangeMax;
    int start;
    int limitMin;
    int limitMax;
    int useBaseline;

    peakRelative_t(){
        use = 0;rangeMin = 0;rangeMax = 500;start = 0;limitMin = 0;limitMax = 10000;
        useBaseline = 0;n1old = 0;n2old = 255;isstart = 0;
    }
    void Start(){isstart = 0;n1old=0;n2old=0;}

    // temporal variable
    // if detection already started or not (already = 1 , not yet = 0)
    int isstart;    //!
    // privious peak value
    int n1old;    //!
    int n2old;    //!

    ClassDef(peakRelative_t,10);
};
```

```
class peak_t : public TObject
{
public:
    int use;
    int rangeMin;
    int rangeMax;
    int start;
    int limitMin;
    int limitMax;
    int useBaseline;
    peak_t(){use = 0;rangeMin = 0;rangeMax = 500;
            start = 0;limitMin = 0;limitMax = 255;isstart = 0;useBaseline = 0;}
    void Start(){isstart = 0;}

    // temporal variable
    // if detection already started or not (already = 1 , not yet = 0)
    int isstart;    //!

    ClassDef(peak_t,10);
};

class integral_t : public TObject
{
public:
    int use;
    int rangeMin;
    int rangeMax;
    int start;
    int limitMin;
    int limitMax;
    int useBaseline;

    integral_t(){use = 0;rangeMin = 0;rangeMax = 500;
            start = 0;limitMin = 0;limitMax = 255*500;isstart = 0;useBaseline = 0;}
    void Start(){isstart = 0;}

    // temporal variable
    // if detection already started or not (already = 1 , not yet = 0)
    int isstart;    //!

    ClassDef(integral_t,10);
};

class integralRelative_t : public TObject
{
public:
    int use;
    int rangeMin;
    int rangeMax;
    int start;
    int limitMin;
    int limitMax;
    int useBaseline;

    integralRelative_t(){
        use = 0;rangeMin = 0;rangeMax = 500;start = 0;limitMin = 0;limitMax = 10000;
        useBaseline = 0;isstart = 0;
    }

    void Start(){isstart = 0;old = 0;}

    // temporal variable
    // if detection already started or not (already = 1 , not yet = 0)
    int isstart;    //!
    // privious peak value
    int old;    //!

    ClassDef(integralRelative_t,10);
};

class CRFBaseline : public TNamed{
public:
    int useBaseline;
    int baseline;
    int autoBaselineRangeMin;
    int autoBaselineRangeMax;

    CRFBaseline(){ useBaseline = 0;baseline = 0;
            autoBaselineRangeMin = 0;autoBaselineRangeMax =500;}
    int CalcBaseline(const short s[2000]);

    ClassDef(CRFBaseline,10);
};

inline int CRFBaseline::CalcBaseline(const short data[2000])
{
    int nUse = useBaseline;
    int nAbsolute = baseline;
    int nAutoMin = autoBaselineRangeMin;
    int nAutoMax = autoBaselineRangeMax;

    // calc baseline
    int nBaseline = 0;

    // ignore minimum & maximum point
    int nMin = 256;
    int nMax = -1;

    switch(nUse){
    case 0: nBaseline = 255;break;
    case 1: nBaseline = 255 - nAbsolute;break;
    case 2:
        if(nAutoMax - nAutoMin < 3){
```

```
            nBaseline = 255;
            break;
        }
        for( int i = nAutoMin; i < nAutoMax; i++ )
        {
            if(nMin > data[i]){
                if(nMin < 256)nBaseline += nMin;
                nMin = data[i];
            }else if(nMax < data[i]){
                if(nMax >= 0)nBaseline += nMax;
                nMax = data[i];
            }else
                nBaseline += data[i];
        }
        nBaseline /= (nAutoMax - nAutoMin - 2);
        break;
    }

    return nBaseline;
}

class CRFBDParam : public TNamed{
public:
    int nID;

    peak_t peak;
    integral_t integral;
    peakRelative_t peakRelative;
    integralRelative_t integralRelative;

    CRFBaseline baseline;

    CRFBDParam()
    {
        nIntervalPeriodic = 3000;    // record every 3000 pulse
    }
    ~CRFBDParam(){}

    Int_t nIntervalPeriodic;

    ClassDef(CRFBDParam,12);
};

class CXRayBDParam : public TNamed{
public:
    int nThreshold;
    int nChannels;
    bool bEnable;

    CXRayBDParam(){nThreshold = 100;nChannels = 2;bEnable = true;}

    ClassDef(CXRayBDParam,11);
};


// Pulse Summaries ////////////////////////////////////////////
class CPulseSummaryRFChannel : public TNamed
{
public:
    int nIntegral;
    int nPeakLow;
    int nPeakHigh;
    float fIntegralRelative;
    float fPeakRelativeLow;
    float fPeakRelativeHigh;
    int nBaseline;

    ClassDef(CPulseSummaryRFChannel,2);
};

class CPulseSummaryRF : public TNamed
{
private:
    CPulseSummaryRFChannel ch1;
    CPulseSummaryRFChannel ch2;
    CPulseSummaryRFChannel ch3;
    CPulseSummaryRFChannel ch4;

public:
    CPulseSummaryRFChannel *ch[4];        //!

    CPulseSummaryRF(){Init();}
    void Init(){
        ch[0] = &ch1; ch[1] = &ch2; ch[2] = &ch3; ch[3] = &ch4;
    }

    ClassDef(CPulseSummaryRF,3);
};

class CPulseSummaryXRay : public TNamed
{
public:
    Int_t nADC[12];
    Int_t nTDC[12];

    ClassDef(CPulseSummaryXRay,2);
};

class CPulseSummary : public TNamed
{
public:
    CPulseSummaryRF rf;
    CPulseSummaryXRay xray;
```

```
    int nEventID;
    int nAcqID;

    ClassDef(CPulseSummary,4);
};

// Pulse Summaries end //////////////////////////////////////////

// RF Waveform //////////////////////////////////////////////////
class CRFWaveform : public TNamed
{
public:
    TArrayI arrData[4];

    Int_t nEventID;
    Int_t nAcqID;

    TString strTime;

    CRFWaveform(){nEventID = 0;nAcqID = 0;}

    ClassDef(CRFWaveform,1);
};

class CMasterParam : public TNamed{
public:
    CMasterParam(){bAutoSave = true;bAutoStart = false;bInterlock = true;}
    bool bAutoSave;
    bool bAutoStart;
    bool bInterlock;

    ClassDef(CMasterParam,2);
};

class CCsyParam : public TNamed{
public:
    CCsyParam(){bBDdetection = true;for(int i=0;i<4;i++)nAfterRun[i]=0;}

    bool bBDdetection;
    Int_t nAfterRun[4];

    ClassDef(CCsyParam,3);
};

template<class T>
T* GetParamFromFile(const char *pName,TDirectory *pRecent)
{
    // Object to return;
    T* pRet = NULL;

    pRet = (T*)(gDirectory->Get(pName));
    if(!pRet){
        // not found. try to get from recent
        T* pTmp = (T*)(pRecent->Get(pName));
        if(pTmp){
            cout << pName << " retrieve from recent." << endl;
            pRet = (T*)(pTmp->Clone());
            pRet->Write();
        }
        else{
            // Finally create;
            cout << pName << " not found." << endl;
            pRet = new T;
            pRet->SetName(pName);
            pRet->Write();
        }
    }
    return pRet;
}

#endif
```

A.20 LinkDef.h

```
#ifdef __CINT__

#pragma link off all globals;
#pragma link off all classes;
#pragma link off all functions;

#pragma link C++ class CCsyParam+;
#pragma link C++ class CMasterParam+;
#pragma link C++ class CRFBDParam+;
#pragma link C++ class CXRayBDParam+;
#pragma link C++ class peak_t+;
#pragma link C++ class integral_t+;
#pragma link C++ class peakRelative_t+;
#pragma link C++ class integralRelative_t+;
#pragma link C++ class CRFBaseline+;
#pragma link C++ class CPulseSummary+;
#pragma link C++ class CPulseSummaryRF+;
#pragma link C++ class CPulseSummaryRFChannel+;
#pragma link C++ class CPulseSummaryXRay+;
#pragma link C++ class CRFWaveform+;
#endif
```

A.21 CAnalyzeFADC.h

```
// CAnalyzeFADC.h
////////////////////////////////////////////////////////////////////

#if !defined(AFX_CANALYZEFADC_H__E3558EF3_8368_4B67_A2DF_4709FBAAD6C4__INCLUDED_)
#define AFX_CANALYZEFADC_H__E3558EF3_8368_4B67_A2DF_4709FBAAD6C4__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "CEventAnalyzer.h"
#include "CSocketReceiver.h"
#include <vector>
#include "streamers.h"

class CAnalyzeFADC : public CEventAnalyzer , public CSocketReceiver
{
public:

    enum{BDMODE_PEAK = 0,BDMODE_INTEGRAL,BDMODE_PEAK_RELATIVE,BDMODE_INTEGRAL_RELATIVE};

    virtual void Save(const char * pStrFilename);
    virtual void SaveBin(std::ostream *pout);
    virtual void SaveBinPeriodic(std::ostream *pout);
    void Analyze();
    virtual void Initialize(TDirectory *pFile,TDirectory *pRecent,CPulseSummary *pSummary);
    virtual void Clear();
    virtual bool Receive(istrstream &in,ostrstream &out);
    CAnalyzeFADC();
    virtual ~CAnalyzeFADC();
    void WriteParams(bool bBD = false,bool bNewTime = true);

    int BDCheckPeak( int channel, short data[2][4][2000] );
    int BDCheckIntegral( int channel, short data[2][4][2000] );
    int BDCheckPeakRelative( int channel, short data[2][4][2000] );
    int BDCheckIntegralRelative( int channel, short data[2][4][2000] );


private:
    TTree *m_pTreeBD;        // RF data : n pulse before BD
    TTree *m_pTreePeriodic;        // RF data : periodic recording

    CPulseSummary *m_pSummary;    // Every pulse recording
    int m_nBufSize;
    int m_nDataSize;        // FADC datasize
    int m_nCount;            // entries written

    bool m_bVerbose;
    bool m_bBeforeDetection;
    CRFBDParam *m_pParam[4];    // [nCh]

//    std::vector<std::vector<float> > m_vctvctRingBuf;
//    std::vector<std::vector<float> >::iterator m_itRingBuf;
//    std::vector<std::vector<float> >::iterator m_itCurrent;    // Head of last event

    std::vector<CRFWaveform> m_vctRingBuf;
    std::vector<CRFWaveform>::iterator m_itRingBuf;

    CRFWaveform *m_pRF;        // Current waveform
    CRFWaveform *m_pRFBD;        // used for save, just pointer (don't attach new object)

    int CalcBaseline(short data[2000],int nUse,int nAbsolute,int nAutoMin,int nAutoMax);

public:
    void WriteBinary(std::ostream *pout,bool bSingle);
    virtual void Start();
};

#endif // !defined(AFX_CANALYZEFADC_H__E3558EF3_8368_4B67_A2DF_4709FBAAD6C4__INCLUDED_
```

A.22 CAnalyzeFADC.cpp

```
// CAnalyzeFADC.cpp
////////////////////////////////////////////////////////////////////

//#define CHNUM 100

#include "Camac-slot.h"

#include <assert.h>
// ROOT library
#include "TNtuple.h"

#include "CSingleEvent.h"
#include "CCamacCtrl.h"

#include "CAnalyzeFADC.h"
#include "bdmonitor.h"

#include "TROOT.h"
#include "CCsyCtrl.h"

////////////////////////////////////////////////////////////////////
// \z/V?///////////////////////////////////////////////////////////////////////

CAnalyzeFADC::CAnalyzeFADC()
{
```

```
    m_pTreeBD = NULL;
    m_pTreePeriodic = NULL;

    // Add to Analyzer
    AddAnalyzer(this);

    // Add to Receiver
    AddReceiver(this);
    SetTargetString("RF");

    // Default: enable
    SetAnalyzeEnabled(true);    // Modified 20040225

    m_nDataSize = FADC500_DATASIZE;
    m_nBufSize = 1024;        // store 1024 events

    m_bVerbose = false;

    m_pSummary = NULL;
}

CAnalyzeFADC::~CAnalyzeFADC()
{

}

void CAnalyzeFADC::Initialize(TDirectory *pFile,TDirectory *pRecent,CPulseSummary *pSummary)
{
    m_pTreeBD = new TTree("rf","glcta-bdm rf-waveform");
    m_pTreePeriodic = new TTree("rf_periodic","periodic rf-waveform");
    m_vctRingBuf.resize(m_nBufSize);

    m_pRF = new CRFWaveform;

    m_pTreePeriodic->Branch("rf","CRFWaveform",&m_pRF);
    m_pTreeBD->Branch("rf","CRFWaveform",&m_pRFBD);

    int i;
    for(i=0;i<4;i++)
        m_pRF->arrData[i].Set(m_nDataSize);


    m_pSummary = pSummary;
    m_pSummary->rf.Init();

    Clear();

//    int i;
    for(i=0;i<4;i++){
        // make param name
        TString str = "RFBDParam.";
        str += i;
        str += "ch";

        // try to open from file
        m_pParam[i] = (CRFBDParam *)(pFile->Get(str));
        if(!m_pParam[i]){
            // not found. try to get from recent
            CRFBDParam *pParam = (CRFBDParam *)(pRecent->Get(str));
            if(pParam){
                cout << "Param" << i << "retrieve from recent." << endl;
                m_pParam[i] = (CRFBDParam *)pParam->Clone();
                m_pParam[i]->Write();
            }
            else{
                // Finally create;
                cout << "Param" << i << " not found." << endl;
                m_pParam[i] = new CRFBDParam;
                m_pParam[i]->SetName(str);
                m_pParam[i]->Write();
            }
        }
    }
}

int CAnalyzeFADC::BDCheckPeak(int channel, short rawdata[2][4][2000] )
{
    int i;

    peak_t &param = m_pParam[channel]->peak;
    short *pdata = rawdata[param.useBaseline][channel];
    CPulseSummaryRFChannel &summary = *m_pSummary->rf.ch[channel];

    // Calc peak
    int n1=255,n11=255,n2=0,n21=0;
    // Check peak
    for( i=param.rangeMin; i<param.rangeMax; i++ )
    {
        int n = pdata[i];
        // current is minimum : formar minimum is second minimum , current is first minimum;
        if(n < n1){n11=n1;n1=n;}
        // current is second minimum : simply replace second minimum to current value;
        else if(n < n11){n11=n;}

        if(n > n2){n21=n2;n2=n;}
    }

    // output to summary
    summary.nPeakLow = n11;
    summary.nPeakHigh = n21;

    if(!param.use)return(0);

    if(!param.isstart){
```

```
        if(n21 >= param.start)
        {
            cout << "Peak[" << channel+1 << "] start. " << n21 << "/" << param.start << endl;
            param.isstart = 1;
            return(0);
        }

    }
    else {
        int lcl = param.limitMin;
        int ucl = param.limitMax;

        if(n11 < lcl || n21 > ucl)
        {
            cout << "Peak[" << channel+1 << "] : BreakDown Found!! nMin:"<< n11 << "/" << param.limitMin
                 << ",nMax:" << n21 << "/" << param.limitMax << endl;
            param.isstart = 0;

            TString str;
            str = "Peak : ch=";
            str += channel+1;

            g_master.SetBreakdownMessage((const char *)str);
            return(1);
        }

    }

    return(0);
}

int CAnalyzeFADC::BDCheckIntegral( int channel, short rawdata[2][4][2000] )
{
    integral_t &param = m_pParam[channel]->integral;
    short *pdata = rawdata[param.useBaseline][channel];
    CPulseSummaryRFChannel &summary = *m_pSummary->rf.ch[channel];
    int i;

    int nIntegral = 0;
    for( i=param.rangeMin; i<param.rangeMax; i++ )
    {
        nIntegral += pdata[i];
    }
    summary.nIntegral = nIntegral;

    if(!param.use){
        return(0);
    }

    if(!param.isstart){
        if( nIntegral > param.start )
        {
            cout << "Integral[" << channel+1 << "] start. " << nIntegral << "/" << param.start << endl;
            param.isstart = 1;
        }
    }
    // nCh,stop,integral
    else{
        // detection
        if( nIntegral > param.limitMax || nIntegral < param.limitMin)
        {
            cout << "Integral[" << channel+1 << "] : BreakDown Found!! "
                 << nIntegral << "/" << param.limitMin << "/" << param.limitMax << endl;
            param.isstart = 0;

            TString str;
            str = "Integral : ch=";
            str += channel+1;

            g_master.SetBreakdownMessage((const char *)str);
            return(1);
        }
    }

    return(0);
}

int CAnalyzeFADC::BDCheckPeakRelative( int channel, short rawdata[2][4][2000] )
{
    peakRelative_t &param = m_pParam[channel]->peakRelative;
    short *pdata = rawdata[param.useBaseline][channel];

    CPulseSummaryRFChannel &summary = *m_pSummary->rf.ch[channel];

    int i;
    // Calc peak
    int n1=255,n11=255,n2=0,n21=0;
    // Check peak
    for( i=param.rangeMin; i<param.rangeMax; i++ )
    {
        int n = pdata[i];

        // current is minimum : formar minimum is second minimum , current is first minimum;
        if(n < n1){n11=n1;n1=n;}
        // current is second minimum : simply replace second minimum to current value;
        else if(n < n11){n11=n;}

        if(n > n2){n21=n2;n2=n;}
        else if(n > n21){n21=n;}
    }

    int n1old = param.n1old;
    int n2old = param.n2old;
```

```
    summary.fPeakRelativeLow  = (float)n11 * 100.0 / (param.n1old ? (float)param.n1old : 0.01);
    summary.fPeakRelativeHigh = (float)n21 * 100.0 / (param.n2old ? (float)param.n2old : 0.01);
    param.n1old = n11;
    param.n2old = n21;

    if( param.use != 1 ){
        return(0);
    }

    if(!param.isstart){
        // Calc integral
        int nIntegral = 0;
        for( i=param.rangeMin; i<param.rangeMax; i++ )
        {
            nIntegral += pdata[i];
        }

        // detection
        if( nIntegral > param.start )
        {
            cout << "PeakRelative[" << channel+1 << "] start. "
             << nIntegral << "/" << param.start << endl;

            param.isstart = 1;
        }
    }
    // nCh,stop,peak
    else{
        int lcl = n1old * param.limitMin / 100;
        int ucl = n2old * param.limitMax / 100;

        // detection
        if((param.limitMin && n11 < lcl) || (param.limitMax && n21 > ucl))
        {
            cout << "PeakRelative[" << channel+1 << "] : BreakDown Found!! nMin:"
                << n11 << "/" << lcl << ",nMax:" << n21 << "/" << ucl << endl;

            param.isstart = 0;

            TString str;
            str = "PeakRelative : ch=";
            str += channel+1;

            g_master.SetBreakdownMessage(str);
            return(1);
        }
    }

    return(0);
}

int CAnalyzeFADC::BDCheckIntegralRelative( int channel, short rawdata[2][4][2000] )
{
    integralRelative_t &param = m_pParam[channel]->integralRelative;
    short *pdata = rawdata[param.useBaseline][channel];
    CPulseSummaryRFChannel &summary = *m_pSummary->rf.ch[channel];

    int i;
    // Calc integral
    int nIntegral = 0;
    int nIntegralOld = param.old;
    for( i=param.rangeMin; i<param.rangeMax; i++ )
    {
        nIntegral += pdata[i];
    }
    summary.fIntegralRelative  = (float)nIntegral * 100 / (param.old ? (float)param.old : 1);
    param.old = nIntegral;

    if(!param.use)return(0);

    if(!param.isstart){
        // detection
        if( nIntegral > param.start )
        {
            cout << "IntegralRelative[" << channel+1 << "] start. " << nIntegral << "/" << param.start << endl;
            param.isstart = 1;
        }
    }
    // nCh,stop,integral
    else{
        int lcl = nIntegralOld * param.limitMin / 100;
        int ucl = nIntegralOld * param.limitMax / 100;

        // detection
        if((param.limitMin != 0 && nIntegral < lcl) ||(param.limitMax !=0 && nIntegral > ucl))
        {
            cout << "IntegralRelative[" << channel+1 << "] : BreakDown Found!! "
                 << nIntegral << "/" << lcl << "/" << ucl << endl;
            param.isstart = 0;

            TString str;
            str = "IntegralRelative : ch=";
            str += channel+1;

            g_master.SetBreakdownMessage(str);
            return(1);
        }
    }

    return(0);
}

void CAnalyzeFADC::Analyze( )
{
```

```
    const CSingleEvent &ev = g_camac.GetEvent(FADC500_1);
    const CSingleEvent &ev2= g_camac.GetEvent(FADC500_2);
    int i;
    int nDataSize;
    int nID = GetIdFinished() + 1;
    int nCh;
    const short *ps = NULL;
    nDataSize = ev.m_pBuf[1];

//    int data[4][2000];
    short data[2][4][2000];

    // ------------------------------------------------------------------
    //Read
    // ------------------------------------------------------------------
    if(ev.m_nAcqID > nID || ev2.m_nAcqID > nID)
    {
        cout << "Unexpected big IDs detected.Ignore them." << endl;
        cout << "Expected: " << nID << ", detected: " << ev.m_nAcqID
          << "," << ev2.m_nAcqID << endl;
        return;
    }
    else if(ev.m_nAcqID < nID-1 || ev2.m_nAcqID < nID-1)
    {
        cout << "Unexpected small IDs detected.Ignore them." << endl;
        cout << "Expected: " << nID-1 << ", detected: " << ev.m_nAcqID
          << "," << ev2.m_nAcqID << endl;
        return;
    }
    else if(ev.m_nAcqID < nID || ev2.m_nAcqID < nID)
    {
        return;    // Not all data gathered
    }

    m_pSummary->nEventID = ev.m_nEventID;
    m_pSummary->nAcqID = ev.m_nAcqID;

    // ------------------------------------------------------------------------------------------
    //Fill to ring buffer & detection buffer
    // ------------------------------------------------------------------------------------------

    assert(nDataSize == ev2.m_pBuf[1]);

    m_pRF->nEventID = ev.m_nEventID;
    m_pRF->nAcqID = ev.m_nAcqID;
    m_pRF->strTime = GetTime();

    for( nCh=0; nCh<4; nCh++ ){
        switch(nCh)
        {
            case 0:ps = &ev.m_pBuf[2];break;
            case 1:ps = &ev.m_pBuf[nDataSize+4];break;
            case 2:ps = &ev2.m_pBuf[2];break;
            case 3:ps = &ev2.m_pBuf[nDataSize+4];break;
        }

        int nBaseline = m_pParam[nCh]->baseline.CalcBaseline(ps);

        for( i=0; i<nDataSize ; i++ ){
            m_pRF->arrData[nCh][i] = ps[i];
            data[0][nCh][i] = 255 - ps[i];
            data[1][nCh][i] = nBaseline - ps[i];
        }
    }

    // Periodic record
    if(m_pRF->nAcqID % m_pParam[0]->nIntervalPeriodic == 0)
        m_pTreePeriodic->Fill();

    // Write to ring buffer
    *m_itRingBuf = *m_pRF;
    m_itRingBuf ++;
    if(m_itRingBuf == m_vctRingBuf.end())m_itRingBuf = m_vctRingBuf.begin();
    if(m_nCount < m_nBufSize) m_nCount ++;

    // Data'd been written : inclement AnalyzeingID
    SetIdFinished(nID);

    // ------------------------------------------------------------------------
    // BDCheckStart
    // ------------------------------------------------------------------------
    // Already in BreakDown sequence : no need to detect BD
    if(g_csy.InBreakDownSequence())return;

    bool bBD = false;
    // BreakDown detection
    for(nCh=0;nCh<4;nCh++)
    {
        // detection criterion
        if( BDCheckPeak( nCh, data ))bBD = true;
        if( BDCheckIntegral( nCh, data ))bBD = true;
        if( BDCheckPeakRelative( nCh, data ))bBD = true;
        if( BDCheckIntegralRelative( nCh, data ))bBD = true;

        // status message
        if(m_bVerbose){
            CPulseSummaryRFChannel &s = *m_pSummary->rf.ch[nCh];
            cout << "[" << nCh+1 << "]" << s.nPeakLow << "/" << s.nPeakHigh << ","
            << s.nIntegral << "," << s.fPeakRelativeLow << "/"
            << s.fPeakRelativeHigh << "," << s.fIntegralRelative
                << endl;
        }
    }
    // BDCheck
```

```
    if(bBD){
        g_csy.BreakDown();
    }

}

bool CAnalyzeFADC::Receive(istrstream &in,ostrstream &out)
{
    char s[256];

    in >> std::setw(255) >> s;

    cout << "AnalyzeRF : Command " << s << " received." << endl;

    // TODO: state check

    // Parse message string
    if(!strcmp(s,"Enable"))
    {
        int bEnable;

        in >> bEnable;
        SetAnalyzeEnabled(bEnable);

        return !in.fail();
    }
    if(!strcmp(s,"GetEnable"))
    {
        return GetAnalyzeEnabled();
    }
    else if(!strcmp(s,"BDParam"))
    {
        int nCh;
        int nMode;

        in >> nCh;
        in >> nMode;

        int nUseBaseline;
        switch(nMode)
        {
        case BDMODE_PEAK:
            in >> m_pParam[nCh-1]->peak.use;
            in >> m_pParam[nCh-1]->peak.rangeMin;
            in >> m_pParam[nCh-1]->peak.rangeMax;
            in >> m_pParam[nCh-1]->peak.start;
            in >> m_pParam[nCh-1]->peak.limitMin;
            in >> m_pParam[nCh-1]->peak.limitMax;
            break;
        case BDMODE_INTEGRAL:
            in >> m_pParam[nCh-1]->integral.use;
            in >> m_pParam[nCh-1]->integral.rangeMin;
            in >> m_pParam[nCh-1]->integral.rangeMax;
            in >> m_pParam[nCh-1]->integral.start;
            in >> m_pParam[nCh-1]->integral.limitMin;
            in >> m_pParam[nCh-1]->integral.limitMax;
            break;
        case BDMODE_PEAK_RELATIVE:
            in >> m_pParam[nCh-1]->peakRelative.use;
            in >> m_pParam[nCh-1]->peakRelative.rangeMin;
            in >> m_pParam[nCh-1]->peakRelative.rangeMax;
            in >> m_pParam[nCh-1]->peakRelative.start;
            in >> m_pParam[nCh-1]->peakRelative.limitMin;
            in >> m_pParam[nCh-1]->peakRelative.limitMax;
            in >> nUseBaseline;
            m_pParam[nCh-1]->peakRelative.useBaseline = (nUseBaseline ? 1 : 0);
            m_pParam[nCh-1]->baseline.useBaseline = nUseBaseline;
            in >> m_pParam[nCh-1]->baseline.baseline;
            in >> m_pParam[nCh-1]->baseline.autoBaselineRangeMin;
            in >> m_pParam[nCh-1]->baseline.autoBaselineRangeMax;
            break;
        case BDMODE_INTEGRAL_RELATIVE:
            in >> m_pParam[nCh-1]->integralRelative.use;
            in >> m_pParam[nCh-1]->integralRelative.rangeMin;
            in >> m_pParam[nCh-1]->integralRelative.rangeMax;
            in >> m_pParam[nCh-1]->integralRelative.start;
            in >> m_pParam[nCh-1]->integralRelative.limitMin;
            in >> m_pParam[nCh-1]->integralRelative.limitMax;
            in >> nUseBaseline;
            m_pParam[nCh-1]->integralRelative.useBaseline = (nUseBaseline ? 1 : 0);
            m_pParam[nCh-1]->baseline.useBaseline = nUseBaseline;
            in >> m_pParam[nCh-1]->baseline.baseline;
            in >> m_pParam[nCh-1]->baseline.autoBaselineRangeMin;
            in >> m_pParam[nCh-1]->baseline.autoBaselineRangeMax;
            break;
        }

        WriteParams();

        return !in.fail();
    }
    else if(!strcmp(s,"GetBDParam"))
    {
        int i;
        for(i=0;i<4;i++)
        {
            out << m_pParam[i]->peak.use << endl;
            out << m_pParam[i]->peak.rangeMin << endl;
            out << m_pParam[i]->peak.rangeMax << endl;
            out << m_pParam[i]->peak.start << endl;
            out << m_pParam[i]->peak.limitMin << endl;
            out << m_pParam[i]->peak.limitMax << endl;

            out << m_pParam[i]->integral.use << endl;
```

```
        out << m_pParam[i]->integral.rangeMin << endl;
        out << m_pParam[i]->integral.rangeMax << endl;
        out << m_pParam[i]->integral.start << endl;
        out << m_pParam[i]->integral.limitMin << endl;
        out << m_pParam[i]->integral.limitMax << endl;

        out << m_pParam[i]->peakRelative.use << endl;
        out << m_pParam[i]->peakRelative.rangeMin << endl;
        out << m_pParam[i]->peakRelative.rangeMax << endl;
        out << m_pParam[i]->peakRelative.start << endl;
        out << m_pParam[i]->peakRelative.limitMin << endl;
        out << m_pParam[i]->peakRelative.limitMax << endl;
        out << (m_pParam[i]->peakRelative.useBaseline ?
        m_pParam[i]->baseline.useBaseline : 0) << endl;
        out << m_pParam[i]->baseline.baseline << endl;
        out << m_pParam[i]->baseline.autoBaselineRangeMin << endl;
        out << m_pParam[i]->baseline.autoBaselineRangeMax << endl;

        out << m_pParam[i]->integralRelative.use << endl;
        out << m_pParam[i]->integralRelative.rangeMin << endl;
        out << m_pParam[i]->integralRelative.rangeMax << endl;
        out << m_pParam[i]->integralRelative.start << endl;
        out << m_pParam[i]->integralRelative.limitMin << endl;
        out << m_pParam[i]->integralRelative.limitMax << endl;
        out << (m_pParam[i]->integralRelative.useBaseline ?
        m_pParam[i]->baseline.useBaseline : 0) << endl;
        out << m_pParam[i]->baseline.baseline << endl;
        out << m_pParam[i]->baseline.autoBaselineRangeMin << endl;
        out << m_pParam[i]->baseline.autoBaselineRangeMax << endl;
    }
    return true;
}
else if(!strcmp(s,"BDParamNew"))
{
    int nCh;
    int nVersion;
    int nMode;

    in >> nVersion;
    if(nVersion !=1){
        cout << "Invalid command version." << endl;
        return false;
    }
    in >> nCh;
    in >> nMode;

    switch(nMode)
    {
    case BDMODE_PEAK:
        in >> m_pParam[nCh-1]->peak.use;
        in >> m_pParam[nCh-1]->peak.rangeMin;
        in >> m_pParam[nCh-1]->peak.rangeMax;
        in >> m_pParam[nCh-1]->peak.start;
        in >> m_pParam[nCh-1]->peak.limitMin;
        in >> m_pParam[nCh-1]->peak.limitMax;
        in >> m_pParam[nCh-1]->peak.useBaseline;
        break;
    case BDMODE_INTEGRAL:
        in >> m_pParam[nCh-1]->integral.use;
        in >> m_pParam[nCh-1]->integral.rangeMin;
        in >> m_pParam[nCh-1]->integral.rangeMax;
        in >> m_pParam[nCh-1]->integral.start;
        in >> m_pParam[nCh-1]->integral.limitMin;
        in >> m_pParam[nCh-1]->integral.limitMax;
        in >> m_pParam[nCh-1]->integral.useBaseline;
        break;
    case BDMODE_PEAK_RELATIVE:
        in >> m_pParam[nCh-1]->peakRelative.use;
        in >> m_pParam[nCh-1]->peakRelative.rangeMin;
        in >> m_pParam[nCh-1]->peakRelative.rangeMax;
        in >> m_pParam[nCh-1]->peakRelative.start;
        in >> m_pParam[nCh-1]->peakRelative.limitMin;
        in >> m_pParam[nCh-1]->peakRelative.limitMax;
        in >> m_pParam[nCh-1]->peakRelative.useBaseline;
        break;
    case BDMODE_INTEGRAL_RELATIVE:
        in >> m_pParam[nCh-1]->integralRelative.use;
        in >> m_pParam[nCh-1]->integralRelative.rangeMin;
        in >> m_pParam[nCh-1]->integralRelative.rangeMax;
        in >> m_pParam[nCh-1]->integralRelative.start;
        in >> m_pParam[nCh-1]->integralRelative.limitMin;
        in >> m_pParam[nCh-1]->integralRelative.limitMax;
        in >> m_pParam[nCh-1]->integralRelative.useBaseline;
        break;
    }

    WriteParams();

    return !in.fail();
}
else if(!strcmp(s,"GetBDParamNew"))
{
    int nVersion;
    in >> nVersion;
    if(nVersion !=1){
        cout << "Invalid command version." << endl;
        return false;
    }
    int i;
    for(i=0;i<4;i++)
    {
        out << m_pParam[i]->peak.use << endl;
        out << m_pParam[i]->peak.rangeMin << endl;
        out << m_pParam[i]->peak.rangeMax << endl;
```

```
        out << m_pParam[i]->peak.start << endl;
        out << m_pParam[i]->peak.limitMin << endl;
        out << m_pParam[i]->peak.limitMax << endl;
        out << m_pParam[i]->peak.useBaseline << endl;

        out << m_pParam[i]->integral.use << endl;
        out << m_pParam[i]->integral.rangeMin << endl;
        out << m_pParam[i]->integral.rangeMax << endl;
        out << m_pParam[i]->integral.start << endl;
        out << m_pParam[i]->integral.limitMin << endl;
        out << m_pParam[i]->integral.limitMax << endl;
        out << m_pParam[i]->integral.useBaseline << endl;

        out << m_pParam[i]->peakRelative.use << endl;
        out << m_pParam[i]->peakRelative.rangeMin << endl;
        out << m_pParam[i]->peakRelative.rangeMax << endl;
        out << m_pParam[i]->peakRelative.start << endl;
        out << m_pParam[i]->peakRelative.limitMin << endl;
        out << m_pParam[i]->peakRelative.limitMax << endl;
        out << m_pParam[i]->peakRelative.useBaseline << endl;

        out << m_pParam[i]->integralRelative.use << endl;
        out << m_pParam[i]->integralRelative.rangeMin << endl;
        out << m_pParam[i]->integralRelative.rangeMax << endl;
        out << m_pParam[i]->integralRelative.start << endl;
        out << m_pParam[i]->integralRelative.limitMin << endl;
        out << m_pParam[i]->integralRelative.limitMax << endl;
        out << m_pParam[i]->integralRelative.useBaseline << endl;
    }
    return true;
}
else if(!strcmp(s,"Baseline"))
{
    int nVersion;
    int nCh;

    in >> nVersion;
    if(nVersion !=1){
        cout << "Invalid command version." << endl;
        return false;
    }
    in >> nCh;

    in >> m_pParam[nCh-1]->baseline.useBaseline;
    in >> m_pParam[nCh-1]->baseline.baseline;
    in >> m_pParam[nCh-1]->baseline.autoBaselineRangeMin;
    in >> m_pParam[nCh-1]->baseline.autoBaselineRangeMax;

    WriteParams();

    return !in.fail();
}
else if(!strcmp(s,"GetBaseline"))
{
    int nVersion;
    int nCh;

    in >> nVersion;
    if(nVersion !=1){
        cout << "Invalid command version." << endl;
        return false;
    }
    in >> nCh;

    out << m_pParam[nCh-1]->baseline.useBaseline << endl;
    out << m_pParam[nCh-1]->baseline.baseline << endl;
    out << m_pParam[nCh-1]->baseline.autoBaselineRangeMin << endl;
    out << m_pParam[nCh-1]->baseline.autoBaselineRangeMax << endl;

    return !in.fail();
}
else if(!strcmp(s,"SetBufferSize"))
{
    int nSize;
    in >> nSize;

    if(in.fail())return false;

    m_nBufSize = nSize;

    // clear data
    Clear();

    return true;
}
else if(!strcmp(s,"GetBufferSize"))
{
    out << m_nBufSize << " ";
    return true;
}
else if(!strcmp(s,"GetCurrentData"))
{
    if(!m_nCount)return false;

    WriteBinary(&out,true);
    return true;
}
else if(!strcmp(s,"GetVerbose")){
    out << m_bVerbose << endl;
    return true;
}
else if(!strcmp(s,"SetVerbose")){
    int nVerbose;
    in >> nVerbose;
```

```
        m_bVerbose = nVerbose;
        return true;
    }
    else if(!strcmp(s,"GetInterval")){
        out << m_pParam[0]->nIntervalPeriodic << endl;
        return true;
    }
    else if(!strcmp(s,"SetInterval")){
        int nInterval;
        in >> nInterval;
        int i;
        for(i=0;i<4;i++)
            m_pParam[i]->nIntervalPeriodic = nInterval;
        return true;
    }
    else
        throw("Message invalid.");

    // not reach here;
    return false;
}

void CAnalyzeFADC::Save(const char * pStrFilename)
{
    // Set Ntuple title
    TString s("RF signal at ");
    s += GetTime(false);        // time used to get filename
    m_pTreeBD->SetTitle(s);
    m_pTreePeriodic->SetTitle(s);

    // copy ntuple from RingBuf
    std::vector<CRFWaveform>::iterator it = m_itRingBuf;
    if(m_nCount < m_nBufSize) it = m_vctRingBuf.begin();
    do
    {
        // read 1 pulse
        m_pRFBD = &(*it);
        m_pTreeBD->Fill();

        it++;
        if(it == m_vctRingBuf.end())it = m_vctRingBuf.begin();
    }while(it != m_itRingBuf);

    // write params
    WriteParams(true,false);
}

void CAnalyzeFADC::SaveBin(std::ostream *pout)
{
    WriteBinary(pout,false);
}
```

```
void CAnalyzeFADC::SaveBinPeriodic(std::ostream *pout)
{
    int n = (int)m_pTreePeriodic->GetEntries();

    // Category RF-FADC
    std::ostream &out = *pout;
    out << "<RF-FADC-PERIODIC>" << endl;

    // Time
    out << "Time: " << GetTime(false) << endl;

    // Title
    out << "Title: Time,Pf,Prs,Pra,Ptr" << endl;

    // Data Size
    out << "DataSize: " << m_nDataSize << endl;

    // Data Count
    out << "DataCount: " << n << endl;;

    // Loop Start

    struct{
        unsigned short time;
        unsigned char pf;
        unsigned char prs;
        unsigned char pra;
        unsigned char ptr;
    }s;

    int j;
    CRFWaveform *pRF = new CRFWaveform;
    TBranch *pBranch = m_pTreePeriodic->GetBranch("rf");
    pBranch->SetAddress(&pRF);

    for(j=0;j<n;j++)
    {
        m_pTreePeriodic->GetEntry(j);

        // Data Section Start
        int nEventID = pRF->nEventID;
        int nAcqID = pRF->nAcqID;

        out << "Data " << nEventID << " " << nAcqID << " " << pRF->strTime << " Start" << endl;

        int i;
        for(i=0;i<m_nDataSize;i++){
            s.time = (unsigned short)i;
            s.pf   = (unsigned char)pRF->arrData[0][i];
            s.prs  = (unsigned char)pRF->arrData[1][i];
            s.pra  = (unsigned char)pRF->arrData[2][i];
            s.ptr  = (unsigned char)pRF->arrData[3][i];
```

```
            out.write((const char *)&s,sizeof(s));
        }

        // Data Section End
        out << endl << "Data " << nEventID << " " << nAcqID << " End" << endl;
    }
    pBranch->SetAddress(&m_pRF);
    //delete pRF;

    // Category RF-FADC End
    out << "</RF-FADC-PERIODIC>" << endl;
}

void CAnalyzeFADC::Clear()
{
    if(m_pTreeBD)m_pTreeBD->Reset();
    if(m_pTreePeriodic)m_pTreePeriodic->Reset();

    m_itRingBuf = m_vctRingBuf.begin();
    m_nCount = 0;

    CEventAnalyzer::Clear();
}

void CAnalyzeFADC::WriteBinary(std::ostream *pout,bool bSingle)
{
    // Category RF-FADC
    std::ostream &out = *pout;
    out << "<RF-FADC>" << endl;

    // Time
    out << "Time: " << GetTime(false) << endl;

    // Title
    out << "Title: Time,Pf,Prs,Pra,Ptr" << endl;

    // Data Size
    out << "DataSize: " << m_nDataSize << endl;

    // Data Count
    out << "DataCount: " << int(bSingle ? 1 : m_nCount) << endl;;

    // Loop Start
    std::vector<CRFWaveform>::iterator it = m_itRingBuf;
    if(bSingle)
    {
        if(it == m_vctRingBuf.begin())it = m_vctRingBuf.end();
        it--;
    }

    if(!bSingle && (m_nCount < m_nBufSize)) it = m_vctRingBuf.begin();

    struct{
        unsigned short time;
        unsigned char pf;
        unsigned char prs;
        unsigned char pra;
        unsigned char ptr;
    }s;

    do
    {
        // Data Section Start
        int nEventID = it->nEventID;
        int nAcqID = it->nAcqID;

        out << "Data " << nEventID << " " << nAcqID << " Start" << endl;

//        cout << "Array Size = " << it->arrData[0].GetSize() << endl;

        int i;
        for(i=0;i<m_nDataSize;i++){
            s.time = (unsigned short)i;
            s.pf   = (unsigned char)it->arrData[0][i];
            s.prs  = (unsigned char)it->arrData[1][i];
            s.pra  = (unsigned char)it->arrData[2][i];
            s.ptr  = (unsigned char)it->arrData[3][i];

            out.write((const char *)&s,sizeof(s));
        }

        // Data Section End
        out << endl << "Data " << nEventID << " " << nAcqID << " End" << endl;

        it ++;
        if(it == m_vctRingBuf.end())it = m_vctRingBuf.begin();
    }while(it != m_itRingBuf);

    // Category RF-FADC End
    out << "</RF-FADC>" << endl;
}

void CAnalyzeFADC::Start()
{
    cout << "FADCAnalyzer start." << endl;
    m_bBeforeDetection = true;

    int i;
    for(i=0;i<4;i++)
    {
        m_pParam[i]->peak.Start();
        m_pParam[i]->integral.Start();
        m_pParam[i]->peakRelative.Start();
```

```
        m_pParam[i]->integralRelative.Start();

        cout << "Parameter " << i << " initialized." << endl;
    }

    CEventAnalyzer::Start();
}

void CAnalyzeFADC::WriteParams(bool bBD,bool bNewTime)
{
    int i;
    char s[1024];
    for(i=0;i<4;i++){
        sprintf(s,"RF param %s at %s ID %d"
            ,bBD ? "saved  " : "changed",GetTime(bNewTime),GetIdFinished());
        m_pParam[i]->SetTitle(s);
        m_pParam[i]->Write();
    }
}
```

A.23 CAnalyzeXRay.h

```
// CAnalyzeXRay.h
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_CANALYZEXRAY_H__DB2C370F_DA09_4606_AC8F_916B20C8F039__INCLUDED_)
#define AFX_CANALYZEXRAY_H__DB2C370F_DA09_4606_AC8F_916B20C8F039__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "CEventAnalyzer.h"
#include "CSocketReceiver.h"
#include "streamers.h"

using namespace std;

class CAnalyzeXRay : public CEventAnalyzer , public CSocketReceiver
{
public:
    void Initialize(TDirectory *pFile,TDirectory *pRecent,CPulseSummary *pSummary);
    virtual void Analyze();
    virtual void Clear(){if(m_pNtuple)m_pNtuple->Reset();CEventAnalyzer::Clear();
    memset(fPrevious,0,sizeof(fPrevious));}
    virtual bool Receive(istrstream &in,ostrstream &out);
    virtual void SaveBin(const char *pStrFilename);
    CAnalyzeXRay();
    virtual ~CAnalyzeXRay();

private:
    TNtuple *m_pNtuple;
    TNtuple *m_pNtuple2;
    int *m_pnOffset;
    int m_nChTDCOrigin;
    int m_nCh;

    CPulseSummaryXRay *m_pSummary;
    CXRayBDParam *m_pParam;
    float fPrevious[12];
    bool m_bSaveBin;
};

#endif // !defined(AFX_CANALYZEXRAY_H__DB2C370F_DA09_4606_AC8F_916B20C8F039__INCLUDED_)
```

A.24 CAnalyzeXRay.cpp

```
// CAnalyzeXRay.cpp
//////////////////////////////////////////////////////////////////////
#include "Camac-slot.h"

#include <assert.h>
#include <iostream>
#include <iomanip>
// ROOT library
#include "TNtuple.h"

#include "bdmonitor.h"

#include "CAnalyzeXRay.h"

#include "CSingleEvent.h"
#include "CCamacCtrl.h"


CAnalyzeXRay::CAnalyzeXRay()
{
    m_pNtuple = NULL;

    // Add to Analyzer
    AddAnalyzer(this);

    // Add to Receiver
    AddReceiver(this);
```

```
    SetTargetString("XRay");

    // By now, ch # is fixed.
    m_nCh = 12;
    m_pnOffset = new int[12];
    // offset is 0 by default
    memset(m_pnOffset,0,sizeof(int)*12);

    // Default: enable
    SetAnalyzeEnabled(true);

    // Default: 10ch is RF
    m_nChTDCOrigin = 10;

}

CAnalyzeXRay::~CAnalyzeXRay()
{
    delete[] m_pnOffset;
}

void CAnalyzeXRay::Initialize(TDirectory *pFile,TDirectory *pRecent,CPulseSummary *pSummary)
{
    // TODO: delete
    m_pNtuple = new TNtuple("acc-gamma","Photons by PMT around Acc.structure",
    "eventid:acqid:adc1:tdc1:adc2:tdc2:adc3:tdc3:adc4:tdc4:adc5:tdc5:adc6:tdc6:"
    "adc7:tdc7:adc8:tdc8:adc9:tdc9:adc10:tdc10:adc11:tdc11:adc12:tdc12");

    m_pNtuple2 = new TNtuple("acc-gamma2","Photons by PMT around Acc.structure,ch# order",
    "eventid:acqid:ch:adc:tdc");

    m_pNtuple->SetAutoSave((int)1e+9);    // Autosave=1GB(practically no autosave)
    m_pNtuple2->SetAutoSave((int)1e+9);    // Autosave=1GB(practically no autosave)

    m_pSummary = &pSummary->xray;

    // Retrieve parameter
    m_pParam = GetParamFromFile<CXRayBDParam>("XRayBDParam",pRecent);
}
void CAnalyzeXRay::Analyze()
{

    const CSingleEvent &evAdc1 = g_camac.GetEvent(XRAYADC_1);
    const CSingleEvent &evAdc2 = g_camac.GetEvent(XRAYADC_2);
    const CSingleEvent &evTdc1 = g_camac.GetEvent(XRAYTDC_1);
    const CSingleEvent &evTdc2 = g_camac.GetEvent(XRAYTDC_2);

    int nTDCOrigin;
    int nID = GetIdFinished() + 1;

    if(evAdc1.m_nAcqID > nID || evAdc2.m_nAcqID > nID || evTdc1.m_nAcqID > nID || evTdc2.m_nAcqID > nID)
    {
        cout << "Unexpected big IDs detected.Ignore them." << endl;
        return;
    }
    if(evAdc1.m_nAcqID < nID || evAdc2.m_nAcqID < nID || evTdc1.m_nAcqID < nID || evTdc2.m_nAcqID < nID)
    {
//        printf("Not all data gathered.%d,%d,%d,%d\n",evAdc1.m_nAcqID,evAdc2.m_nAcqID,evTdc1.m_nAcqID,evTdc2.m_nAcq
        return;    // Not all data gathered
    }

    assert(evTdc1.m_nBufSize >= 6);
    assert(evTdc2.m_nBufSize >= 6);

    if(m_nChTDCOrigin < 0)
        nTDCOrigin = 0;
    else if(m_nChTDCOrigin < 6)
        nTDCOrigin = evTdc1.m_pBuf[m_nChTDCOrigin];
    else
        nTDCOrigin = evTdc2.m_pBuf[m_nChTDCOrigin - 6];


    float f[26];    // 12 x 2 + 2

    f[0] = (float)evAdc1.m_nEventID;
    f[1] = (float)evAdc1.m_nAcqID;

    // Data write to ntuple & ntuple2 & pulse summary
    int i;
    for(i=0;i<6;i++){
        f[i*2+2] = (float)(evAdc1.m_pBuf[i] - m_pnOffset[i]);
        f[i*2+3] = (float)(evTdc1.m_pBuf[i] - (i == m_nChTDCOrigin ? 0 : nTDCOrigin));

        m_pNtuple2->Fill(f[0],f[1],i,f[i*2+2],f[i*2+3]);

        m_pSummary->nADC[i] = evAdc1.m_pBuf[i] - m_pnOffset[i];
        m_pSummary->nTDC[i] = evTdc1.m_pBuf[i] - (i == m_nChTDCOrigin ? 0 : nTDCOrigin);
    }
    for(;i<12;i++){
        f[i*2+2] = (float)(evAdc2.m_pBuf[i-6] - m_pnOffset[i]);
        f[i*2+3] = (float)(evTdc2.m_pBuf[i-6] - (i == m_nChTDCOrigin ? 0 : nTDCOrigin));
        m_pNtuple2->Fill(f[0],f[1],i,f[i*2+2],f[i*2+3]);

        m_pSummary->nADC[i] = evAdc2.m_pBuf[i-6] - m_pnOffset[i];
        m_pSummary->nTDC[i] = evTdc2.m_pBuf[i-6] - (i == m_nChTDCOrigin ? 0 : nTDCOrigin);
    }
    m_pNtuple->Fill(f);

    // Data'd been written : inclement AnalyzeingID
    SetIdFinished(nID);
//    printf("Event # %d recorded.\n",GetIdFinished());

    if(!m_pParam->bEnable)return;
```

```
    // Already in BreakDown sequence : no need to detect BD
    if(g_csy.InBreakDownSequence())return;
    // Breakdown detection
    int nBDcount = 0;
    for(i=0;i<12;i++)
    {
        if(fPrevious[i] > 0 && fPrevious[i] + m_pParam->nThreshold < f[i*2+2])
        {
            nBDcount ++;
        }
        // save previous values
        fPrevious[i] = f[i*2+2];
    }
    if(nBDcount >= m_pParam->nChannels){
        g_master.SetBreakdownMessage("XRay");
        g_csy.BreakDown();
    }

}

bool CAnalyzeXRay::Receive(istrstream &in,ostrstream &out)
{
    char s[256];

    in >> std::setw(255) >> s;

    cout << "AnalyzeXRay : Command " << s << " received." << endl;

    // Parse message string
    if(!strcmp(s,"Enable"))
    {
        int bEnable;

        in >> bEnable;
        SetAnalyzeEnabled(bEnable);

        return !in.fail();
    }
    if(!strcmp(s,"GetEnable"))
    {
        return GetAnalyzeEnabled();
    }
    else if(!strcmp(s,"GetBDEnable"))
    {
        return m_pParam->bEnable;
    }
    else if(!strcmp(s,"SetBDEnable"))
    {
        int bEnable;

        in >> bEnable;
        m_pParam->bEnable = bEnable;

        return !in.fail();
    }
    else if(!strcmp(s,"SetADCOffset"))
    {
        int i;
        for(i=0;i<m_nCh;i++)
        {
            in >> m_pnOffset[i];
        }

        return !in.fail();
    }
    else if(!strcmp(s,"SetChTDCOrigin"))
    {
        in >> m_nChTDCOrigin;

        return !in.fail();
    }
    else if(!strcmp(s,"SetBDParam"))
    {
        int nVersion;
        in >> nVersion;
        if(nVersion != 1)
        {
            cout << "Invalid command version." << endl;
            return false;
        }

        in >> m_pParam->nThreshold;
        in >> m_pParam->nChannels;

        return !in.fail();
    }
    else if(!strcmp(s,"GetBDParam"))
    {
        int nVersion;
        in >> nVersion;
        if(nVersion != 1)
        {
            cout << "Invalid command version." << endl;
            return false;
        }

        out << m_pParam->nThreshold << endl;
        out << m_pParam->nChannels << endl;

        return !in.fail();
    }
    else
        throw("Message invalid.");
```

```
    // not reach here;
    return false;
}

using namespace std;
void CAnalyzeXRay::SaveBin(const char *pStrFilename)
{
    TString s = pStrFilename;
    s += ".xray";

    ofstream out;
    out.open(s);

    out << "<X-ray>" << endl;

    // Time
    out << "Time: " << GetTime(false) << endl;

    // Title
    out << "Title: ch,adc,tdc" << endl;

    // Data Size
    out << "DataSize: " << 12 << endl;

    // Data Count
    int nCount = m_pNtuple->GetEntries();
    int nWrite = nCount;
    if(nWrite > 512)nWrite = 512;

    out << "DataCount: " << nWrite << endl;;

    int i;
    for(i=nCount-nWrite;i<nCount;i++)
    {
        m_pNtuple->GetEntry(i);
        float *p = m_pNtuple->GetArgs();

        out << "Data " << (int)p[0] << " " << (int)p[1] << " Start" << endl;

        int j;
        for(j=0;j<12;j++){
            unsigned char c[3];
            c[0] = j;
            c[1] = (unsigned char)p[2+j*2];
            c[2] = (unsigned char)p[3+j*2];
            out.write((const char *)c,3);
        }

        // Data Section End
        out << endl << "Data " << (int)p[0] << " " << (int)p[1] << " End" << endl;
    }
}
```


A.25 Makefile

```
# Makefile ( n source file 1 execute file version )
# 2002/12/ 4

PACKAGE    = bdmonitor
SRCS    = bdmonitor.cpp CAnalyzeXRay.cpp CAnalyzeFADC.cpp CCamacCtrl.cpp \
CSingleEvent.cpp CSocketReceiver.cpp CCsyCtrl.cpp CMasterCtrl.cpp \
CSocketCtrl.cpp rootdict.cpp
HEADS    = CAnalyzeXRay.h CAnalyzeFADC.h CCamacCtrl.h CEventAnalyzer.h \
CSingleEvent.h CSocketReceiver.h CCsyCtrl.h bdmonitor.h CMasterCtrl.h \
CSocketCtrl.h rootdict.h Camac-slot.h
OBJS    = $(SRCS:.cpp=.o)

FILES    = README Makefile $(HEADS) $(SRCS)
VER    = `date +%Y%m%d`

# common (*.o)
LD    = gcc
LDFLAGS    = -g $(DEBUG)
LDLIBS    = camac.o -lm $(shell root-config --libs) -L/mnt/nas/data/lib/atf_socket/archive -latf_socket_functions

# C (*.c)
CC    = gcc
CFLAGS    = -g -O2 -Wall $(DEBUG) -Wno-deprecated
CPPFLAGS= -I.

# C++ (*.cc,*.cpp)
CXX    = g++
CXXFLAGS= -g -O3 -Wall $(DEBUG) -Wno-deprecated
INCLUDE=/cern/root/include

# etc
SHELL    = /bin/sh
RM    = rm -f

### rules ###

.SUFFIXES:
.SUFFIXES: .o .c .cc .cpp .f .p

all: $(PACKAGE)

$(PACKAGE): $(OBJS)
    $(LD) $(LDFLAGS) $(OBJS) -o $@ $(LDLIBS)

$(OBJS): $(HEADS) Makefile
```

```
.c.o:
    $(CC) $(CFLAGS) $(CPPFLAGS) -c $< -o $@
.cc.o:
    $(CXX) $(CXXFLAGS) $(CPPFLAGS) -c $< -o $@
.cpp.o:
    $(CXX) $(CXXFLAGS) $(CPPFLAGS) -I$(INCLUDE) -c $< -o $@

dict:
    rootcint -f rootdict.cpp -c streamers.h LinkDef.h

clean:
    $(RM) $(PACKAGE) $(OBJS)
    $(RM) core gmon.out *~ #*#
```

付 録B　放電検出システム(VME部)ソースコード

現行バージョン:Last modified 2004/11/02
　文字化けしたコメント等を一部整理しました。

B.1　bdmonitor.h

```
// bdmonitor.h -- BreadDown monitor system main header

#include <list>
#include <stdio.h>
#include <time.h>
#include <iostream>

#include "CEventAnalyzer.h"
#include "CSocketReceiver.h"
#include "CMasterCtrl.h"
#include "CVmeCtrl.h"

#include "myvme.h"

// global functions

// Initialization functions
// Create your object (probably at global scope),then call them at your constructor.

// If your class derives from CEventAnalyzer and enable analyze feature,
// call the function below at constructor.
void AddAnalyzer(CEventAnalyzer *pAnalyzer);
// If from CSocketReceiver
void AddReceiver(CSocketReceiver *pReceiver);

// global variables
extern std::list<CEventAnalyzer *> g_liAnalyzer;
extern std::list<CSocketReceiver *> g_liReceiver;

extern CVmeCtrl g_vme;
extern CMasterCtrl g_master;
extern CSocketCtrl g_socket;
```

B.2　bdmonitor.cpp

```
// bdmonitor.cpp - main source

// ROOT library
#include "TROOT.h"
#include "TFile.h"
#include "TNtuple.h"

#ifndef WIN32
#include <unistd.h>
#include <sys/ioctl.h>
#endif

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
```

```
// Standard library
#include <iostream>
#include <assert.h>
#include <vector>
#include <list>

#include "bdmonitor.h"
#include "CVmeCtrl.h"
#include "CEventAnalyzer.h"
#include "CSocketReceiver.h"
#include "CMasterCtrl.h"

#include "CAnalyzeAcoustic.h"

using namespace std;

TROOT MyTinyApp("GLCTA-X-Ntuple", "GLCTA X-ray detection system Ntuple generator");

list<CEventAnalyzer *> g_liAnalyzer;
list<CSocketReceiver *> g_liReceiver;

CVmeCtrl g_vme;
CMasterCtrl g_master;
CSocketCtrl g_socket;

// global functions
void AddAnalyzer(CEventAnalyzer *pAnalyzer){g_liAnalyzer.push_back(pAnalyzer);}
void AddReceiver(CSocketReceiver *pReceiver){g_liReceiver.push_back(pReceiver);}

int main(int argc,char *argv[]) {

    try{
        // RF analyzer
        CAnalyzeAcoustic ac;

        cout << "GLCTA breakdown record system compiled at " << __TIME__ << endl;
        g_master.Initialize();

        cout << "MasterCtrl Initialized. Go to main loop." << endl;

        g_master.MainLoop();

        cout << "Main loop returned. exit." << endl;
    }
    catch(const char *pStr)
    {
        cout << pStr << endl;
        cout << "Scan Aborted." << endl;
    }

    return 0;
}
```


B.3 CMasterCtrl.h

```
// Master Control Class : Controls Start,Stop,Status,etc.
#ifndef CMASTERCTRL_H_INC
#define CMASTERCTRL_H_INC

#include "CSocketReceiver.h"
#include "CSocketCtrl.h"

class TFile;
class CMasterCtrl : public CSocketReceiver
{
public:
    CMasterCtrl(void);
    virtual ~CMasterCtrl(void);

    // Main loop
    void MainLoop(void);

    // Socket Callback
    virtual bool Receive(std::istrstream &in,std::ostrstream &out);

    // Commands
    void Start(void);
    void Stop(void);
    void Save(const char *pStrFilename);
    void SaveBin(const char *pStrFilename);
    bool Status(void) const {return m_bStatus;}
    void Clear(void);

private:
    bool m_bStatus;
    TFile *m_pFile;

    bool m_bOverWrite;

public:
    void Initialize(void);
};

#endif
```

B.4 CMasterCtrl.cpp

```
// CMasterCtrl.cpp
#include "bdmonitor.h"
#include "CMasterCtrl.h"

// ROOT include
#include "TFile.h"

#include <unistd.h>
#include <time.h>
#include <iostream>
#include <iomanip>
using namespace std;

CMasterCtrl::CMasterCtrl(void)
{
    m_pFile = NULL;
    // Initialize Valiables
    m_bStatus = false;    // Not Collecting data.

    // Set Category
    SetTargetString("Main");
    // Add to Socket Receiver
    g_liReceiver.push_back(this);

    m_bOverWrite = false;    // Modified 20040311
}

CMasterCtrl::~CMasterCtrl(void)
{
}

void CMasterCtrl::Initialize(void)
{
        const char *pStrFilenameBase = "/mnt/nas/data/rawdata";

    // Set Buffer file
    time_t t;
    time(&t);
    struct tm *tim = localtime(&t);
    char s[1024],s2[1024],sRecent[1024];

    // File "Recent"
    sprintf(sRecent,"%s/recent-vme.root",pStrFilenameBase);
    // Read only
    TFile f(sRecent);

    // File to save
    sprintf(s,"%s/%%Y/%%m/%%Y%%m%%d.vme.root",pStrFilenameBase);
    tim = localtime(&t);
    strftime(s2,1024,s,tim);

    m_pFile = new TFile(s2,"UPDATE");
    if(!m_pFile)
    {
        printf("Data file cannot opened. exit.\n");
        exit(1);
    }
    // current dir is savefile
    m_pFile->SetCompressionLevel(1);

    // Initialize socket
    g_socket.Initialize();

    // VME Initialize
    g_vme.Init();

    // Analyzer Initialize
    list<CEventAnalyzer *>::iterator itAna;
    for(itAna = g_liAnalyzer.begin();itAna != g_liAnalyzer.end();itAna ++)
        (*itAna)->Initialize(gDirectory,&f);
    for(itAna = g_liAnalyzer.begin();itAna != g_liAnalyzer.end();itAna ++)
        (*itAna)->Clear();

    // Save Setting
    m_pFile->Write();

    // re-attach file "recent"
    sprintf(s,"rm %s\n",sRecent);
    system(s);
    sprintf(s,"ln %s %s\n",s2,sRecent);    // Changed to HARD link 20041005
    system(s);

    // file "recent" close automatically
}

// Main Loop;
void CMasterCtrl::MainLoop(void)
{
    // Infinite loop
    while(1)
    {
        // Check Socket
        try{
            g_socket.CheckSocket();
        }catch(const char *p)
        {
            cout << "Socket error." << endl;
            cout << p << endl;
        }

        // Status
        if(!Status())
        {
```

```
#ifndef WIN32
            usleep(10000);    // 10msec
#endif
            continue;
        }

        std::list<CEventAnalyzer *>::iterator itAna;
        for(itAna = g_liAnalyzer.begin();itAna != g_liAnalyzer.end();itAna ++)
            (*itAna)->CallAnalyzer();
    }
}


bool CMasterCtrl::Receive(std::istrstream &in,std::ostrstream &out)
{
    char s[256];

    in >> std::setw(255) >> s;

    cout << "Main : Command " << s << " accepted." << endl;

    // Parse message string
    if(!strcmp(s,"Start"))
    {
        if(Status())return false;

        Start();
        return true;
    }
    else if(!strcmp(s,"Stop"))
    {
        if(!Status())return false;

        Stop();
        return true;
    }
    else if(!strcmp(s,"Status"))
    {
        return Status();
    }
    else if(!strcmp(s,"Save"))
    {
        in >> std::setw(255) >> s;

        Save(s);
        return true;
    }
    else
        throw("Message invalid.");

    // not reach here;
    return false;
}


// Commands implemented

void CMasterCtrl::Start(void)
{
    // Analyzer Clear
    Clear();

    // Analyzer Start
    std::list<CEventAnalyzer *>::iterator it;
    for(it = g_liAnalyzer.begin();it != g_liAnalyzer.end();it++){
        (*it)->Start();
    }

    // activate main loop
    m_bStatus = true;

    printf("Start command successful.\n");
}

void CMasterCtrl::Stop()
{
    // Stop sequence

    // Status Change
    m_bStatus = false;

    // Stop Analyzers
    std::list<CEventAnalyzer *>::iterator it;
    for(it = g_liAnalyzer.begin();it != g_liAnalyzer.end();it++){
        (*it)->Stop();
    }
}

void CMasterCtrl::Save(const char *pStrFilename)
{
        cout << "Save "<< pStrFilename << endl;

    // Stop at first
    if(Status())
        Stop();
    cout << "Stop successful." << endl;

    // Save Analyzers
    std::list<CEventAnalyzer *>::iterator it;
    for(it = g_liAnalyzer.begin();it != g_liAnalyzer.end();it++){
        (*it)->Save(pStrFilename);
    }

    cout << "Analyzer save successful." << endl;
```

```
    // Write buffer file.
    gDirectory->Write("",m_bOverWrite ? TObject::kOverwrite : 0);

    cout << "File write successful." << endl;

    // to SaveBin
    SaveBin(pStrFilename);


    // Autostart
    Start();
}

// Save Raw Data
void CMasterCtrl::SaveBin(const char * pStrFilename)
{
    // file saved to current file
  //    Save(NULL);

    FILE *fp = fopen(pStrFilename,"wb");
    if(!fp)throw("Failed to open binary file.");

    // Save Analyzers
    std::list<CEventAnalyzer *>::iterator it;
    for(it = g_liAnalyzer.begin();it != g_liAnalyzer.end();it++){
        (*it)->SaveBin(fp);
    }

    fclose(fp);
}


void CMasterCtrl::Clear(void)
{
    // Clear Sequence
    std::list<CEventAnalyzer *>::iterator it;
    for(it = g_liAnalyzer.begin();it != g_liAnalyzer.end();it++){
        (*it)->Clear();
    }
}
```


B.5 CVmeCtrl.h

```
// CVmeCtrl.h - include cc7x00 library and read events from /dev/dc
#ifndef CVmeCtrl_INCLUDED
#define CVmeCtrl_INCLUDED

#include "myvme.h"

class CVmeCtrl
{
public:
    CVmeCtrl(){
      m_binit = false;
      bus_handle = 0;
      int i;
      for(i=0;i<N_VME_ACOUSTIC_MODULE;i++)
        window_handle[i] = 0;
    }

    ~CVmeCtrl()
    {
      int i;
      for(i=0;i<N_VME_ACOUSTIC_MODULE;i++)
        if(window_handle[i]){
          vme_master_window_unmap(bus_handle,window_handle[i]);
          vme_master_window_release(bus_handle, window_handle[i]);
        }

      if(bus_handle)
        vme_term(bus_handle);
    }

    // Camac ctrl commands
    void Init();
    void Start();
    void Stop();

    unsigned int * GetPtrToModule(int nCh);

private:
    bool m_binit;
    vme_bus_handle_t bus_handle;
    vme_master_handle_t window_handle[N_VME_ACOUSTIC_MODULE];
};

#endif
```

B.6 CVmeCtrl.cpp

```
// CVmeCtrl.cpp
#include <sys/types.h>
#include <sys/stat.h>
```

```
#include <fcntl.h>

#ifndef WIN32
#include <unistd.h>
#include <sys/ioctl.h>
#else
#include <io.h>
#endif

#include <stdlib.h>
// Standard library
#include <iostream>
#include <assert.h>
#include <vector>
using namespace std;

#include "vme/vme.h"
#include "vme/vme_api.h"
#include "CVmeCtrl.h"

// VME Initialization
void CVmeCtrl::Init()
{
    assert(!m_binit);

    // VME Initialize
    if(vme_init(&bus_handle))throw("Error initializing the VMEbus");

    // Create window (ie. attachment between register of VME module and memory on PC)
    int i;
    for(i=0;i<N_VME_ACOUSTIC_MODULE;i++){
      if (vme_master_window_create(bus_handle, &window_handle[i],
                        s_nAddr[i], ADDRESS_MODIFIER, NBYTES,
                        VME_CTL_PWEN, NULL))throw("Error creating the window");
    }

    m_binit = true;
}

void CVmeCtrl::Start()
{
    assert(m_binit);

    // no-op.
}

void CVmeCtrl::Stop()
{
    assert(m_binit);

    // no-op.
}

unsigned int * CVmeCtrl::GetPtrToModule(int nCh)
{
  return (unsigned int *)vme_master_window_map(bus_handle, window_handle[nCh], 0);
}
```

B.7 common_include.h

```
#define MAX_BOARD_READ 200

#define MAX_VME_BOARDS    6

// by Suehara 20040326
#define N_VME_ACOUSTIC_MODULE 4

#define N_ACC_CHANNELS 16
#define N_CHANNELS_64 4
#define N_CHANNELS_32 8
#define MAX_FIFO (3*1024)

#define MAX_N_RECORDS 1024
#define MAX_TIMING 60

#define NCHANNELS_PER_PLOT 4
#define NCHANNELS_PER_CABLE 4

/* disused */
#define OBSOLETE_SPARK_LEVEL 40
#define OBSOLETE_SPARK_FACTOR 25
#define OBSOLETE_BREAKDOWN_VALUE 3000
#define OBSOLETE_BREAKDOWN_AMPLITUDE 2000
#define OBSOLETE_EARLY_BREAKDOWN_AMPLITUDE 2000

#define N_BREAKDOWN_ALGO 9
#define N_FIRST_SET_ALGO 3

#define LINY 0
#define LOGY 1


#define SENSORS_MIDDLE_VALUE 2048
#define FIRST_RECORDS 200

//Channel Mask
#define CHANNEL_MASK 0x0FFF

/* Verbose codes */
```

```
#define VERBOSE 1
#define INFORM 5
#define SILENT 9


/* function return code */
#define OK 1
#define FAILED 99


/* Return code */
#define CORRECT_EXIT 0

#define ERROR_BOARD_READ_FAILED 10
#define ERROR_BOARD_ACQUIRE_FAILED 11
#define ERROR_LOOPBACK_FAILED 12

#define ERROR_BAD_PARAMETER_COUNT 50
#define ERROR_BAD_PARAMETER_VALUE 51

#define ERROR_TOO_MANY_FIFO 60

#define ERROR_NON_ZERO_NULL_VALUE 101


#define ERROR_DUMP_FILE_NOT_OPENED 191
#define ERROR_OPENING_BREAKDOWN_FILE 192
#define ERROR_DUMP_FILE_NOT_COMPLETE 193

#define ERROR_WRONG_FILE_FORMAT 195
#define ERROR_WRONG_BOARD_SEPARATOR 196

#define DEFAULT_FILE_NAME "moduleDump.bin"


#define FILE_PREFIXE "VME Accoustic Data"
#define BOARD_SEPARATOR 123123

#define CHANNEL_FILE_NAME "channels_names."
#define CHANNEL_FILE_EXT  ".channels"
```

B.8 myvme.h

```
#ifndef MYVME_H
#define MYVME_H

#include "common_include.h"
#include "vme/vme.h"
#include "vme/vme_api.h"

#include <stdio.h>

#define PORT_NO 61000
#define TIMEOUT 15 //sec

#ifdef DEBUG /* Variable de debugage; si la variable DEBUG est definit */
#define HERE printf("[%s:%d]\n",__FILE__,__LINE__);
   /* lors de la compilation, alors toutes les instructions AFF*/
#define AFF(x) printf("[%s:%d]",__FILE__,__LINE__); printf x ;printf("\n");
    /* lors de la compilation, alors toutes les instructions AFF*/
#define AFFS(x) printf x ;printf("\n");
    /* lors de la compilation, alors toutes les instructions AFF*/
#else /*  deviendront equivalentes a printf sinon elle seront ignorees*/
#define HERE
#define AFF(x)
#define AFFS(x)
#endif

#define VME_ADDRESS        0x222200
#define ADDRESS_MODIFIER   VME_A24SD
#define NBYTES             0x0020
static int s_nAddr[N_VME_ACOUSTIC_MODULE] = {
  VME_ADDRESS,
  VME_ADDRESS + 0x100,
  VME_ADDRESS + 0x200,
  VME_ADDRESS + 0x300
};

#define N_RECORDS 1024
#define N_SAMPLES_PER_MODULE 3
//File version 3 = multiboards support
#define FILE_VERSION 7


/* Board parameters */
#define BOARD_FIFO_VALUE_LOCATION 0x00
#define BOARD_MODE_LOCATION    0x20
#define BOARD_CLOCK_CONTROL_LOCATION 0x24
#define BOARD_GAIN_CONTROL_LOCATION 0x28
#define BOARD_LOOPBACK_LOCATION 0x2C
#define BOARD_VERSION_LOCATION 0x30
#define BOARD_TRIGGER_FREQ_LOCATION 0x34
#define BOARD_CLOCK_FREQ_LOCATION 0x38
#define BOARD_FIFO_COUNT_LOCATION 0x3C


//#define
```

```
#define MODE_READ 1
#define MODE_ACQUIRE 2
#define MODE_READ_SEND_VALUE 0x00
#define MODE_ACQUIRE_SEND_VALUE 0x01

#define MODE_FIFO_FULL_RECV_VALUE 0x00000001
#define MODE_READ_RECV_VALUE 0x00000002

/* Texts for command line parameters */
#define TXT_MODE_PARAMETER  "mode"
#define TXT_CLOCK_PARAMETER "clock"
#define TXT_GAIN_PARAMETER  "gain"


//Max length of char string
#define MAXLENGTH 100

#define DUMPFILE "dump.txt"
#define SENSORS_FILE "sensors.txt"

#define BOARD_ZERO 0

/**
 *
 * Reading/writing the VME
 *
 */

uint32_t readValue(uint32_t *ptrToBoard,unsigned int address,int verbose) ;
uint64_t readValue64b(uint32_t *ptrToBoard,unsigned int address);
uint32_t readValue32b(uint32_t *ptrToBoard,unsigned int address);

void writeValue(uint32_t *ptrToBoard,unsigned int address,uint32_t value);

int getBoardMode(uint32_t *ptrToBoard,int verbose);
int setBoardMode(uint32_t *ptrToBoard,int mode,int verbose);

int loopBackTest(uint32_t *ptrToBoard,int verbose);

//uint32_t boardVersion(uint32_t *ptrToBoard,int verbose);

void boardParametersHelp();


int check_socket( char *fileName );

#endif
```


B.9 myvme.cpp

```
#include "atf_socket.h"
#include "myvme.h"
#include "vme/vme.h"
#include "vme/vme_api.h"

#include "stdlib.h"
#include "time.h"

uint32_t readValue(uint32_t *ptrToBoard,unsigned int address,int verbose)
{
  uint32_t *ptrToValue;

  ptrToValue=(uint32_t*)((uint32_t)ptrToBoard+address);
  if (verbose==VERBOSE) {
    printf("Address readValue %.2x: ",address);
    printf("%.2x \n", *ptrToValue);
  }
  /*
  //Debug
  if ((address==BOARD_FIFO_COUNT_LOCATION)&&((*ptrToValue)!=0)) {
    printf("Debug Address %.2x: ",address);
    printf("%.2x \n", *ptrToValue);
  }
  //end debug
  */
  return (uint32_t)(*ptrToValue);
} //read value


// Reads 32 bits from the board; for speed reasons this can not be verbose
uint32_t readValue32b(uint32_t *ptrToBoard,unsigned int address)
{
  uint32_t *ptrToValue;

  ptrToValue=(uint32_t*)((uint32_t)ptrToBoard+address);
  /*
  if (verbose==VERBOSE) {
    printf("Address %.2x: ",address);
    printf("%.2x \n", *ptrToValue);
    }
  */
  return (uint32_t)*ptrToValue;
} //read value

// Reads 64 bits from the board; for speed reasons this can not be verbose
uint64_t readValue64b(uint32_t *ptrToBoard,unsigned int address)
{
  uint64_t *ptrToValue;

  ptrToValue=(uint64_t*)((uint32_t)ptrToBoard+address);
```

```
  /*
  if (verbose==VERBOSE) {
    printf("Address %.2x: ",address);
    printf("%.2x \n", *ptrToValue);
    }
  */
  return (uint64_t)*ptrToValue;
} //read value

void writeValue(uint32_t *ptrToBoard,unsigned int address,uint32_t value)
{
  uint32_t *ptrToValue;

  ptrToValue=(uint32_t*)((uint32_t)ptrToBoard+address);
  (*ptrToValue)=value;

} //read value

int getBoardMode(uint32_t *ptrToBoard,int verbose)
{
  uint32_t mode=readValue(ptrToBoard,BOARD_MODE_LOCATION,verbose);
  int modeValue;
  /*
  if (verbose==VERBOSE) {
    printf("Board raw mode: %.2x %.2x\n",mode,(mode&&0x00000002));
  }
  */
  if ((mode&MODE_READ_RECV_VALUE)==MODE_READ_RECV_VALUE) {
    modeValue=MODE_READ;
  } else {
    modeValue=MODE_ACQUIRE;
  }
  if ((verbose==VERBOSE)||(verbose==INFORM)) {
    if (modeValue==MODE_READ) {
      printf("Read ");
    } else {
      printf("Acquire ");
    }
    if ((mode&MODE_FIFO_FULL_RECV_VALUE)==MODE_FIFO_FULL_RECV_VALUE) {
      printf("[FIFO full]");
    } else {
      printf("[FIFO not full]");
    }
  }  // verbose
  return modeValue;
}

int setBoardMode(uint32_t *ptrToBoard,int mode,int verbose)
{
  if (mode==MODE_READ) {
    writeValue(ptrToBoard,BOARD_MODE_LOCATION,MODE_READ_SEND_VALUE);
  } else {
    writeValue(ptrToBoard,BOARD_MODE_LOCATION,MODE_ACQUIRE_SEND_VALUE);
  }
  int iloop=0;
  while ((getBoardMode(ptrToBoard,verbose)!=mode)&&(iloop<10000)) {
    /*
    if (verbose==VERBOSE) {
      printf("Board mode not yet set: %d %d\n",getBoardMode(ptrToBoard,VERBOSE),mode);
    }
    */
    //    sleep(1);
    iloop++;
  } // while board mode is not set
  if (getBoardMode(ptrToBoard,verbose)!=mode) {
    if (verbose==VERBOSE) {
      printf("Board mode set failed!!!\n");
    }
    return FAILED;
  } else {
    return OK;
  }
} // setBoardMode


int loopBackTest(uint32_t *ptrToBoard,int verbose)
{
  static int randomInitialized=0;

  if (randomInitialized!=1) {
    //Initialize random numbers generator
    srandom(time(NULL));
    if (verbose==VERBOSE) {
      printf("Random number generator initialized.\n");
    }
    randomInitialized=1;
  } else {
    if (verbose==VERBOSE) {
      printf("Random number generator already initialized.\n");
    }
  }

  //get a random number
  long int randomValue=random();

  //writes it
  writeValue(ptrToBoard,BOARD_LOOPBACK_LOCATION,randomValue);

  long int returnedValue=readValue(ptrToBoard,BOARD_LOOPBACK_LOCATION,verbose);



  if (randomValue==returnedValue) {
    if (verbose==VERBOSE) {
```

```
      printf("Loopback OK ( %d / %d )\n",randomValue,returnedValue);
    }
    return OK;
  } else {
    if (verbose==VERBOSE) {
      printf("Loopback failed ( %d / %d )!!!\n",randomValue,returnedValue);
    }
    return FAILED;
  } //check returned value
}//loopBackTest

void boardParametersHelp(char *progName)
{
  printf("Usage: %s [parameter value] [parameter value] [parameter value]\n",progName);
  printf("       \t parameter is the name of a parameter:\n");
  printf("       \t \t %s sets the board mode (acceptable values are 0 (read) or 3 (acquire)\n",TXT_MODE_PARAMETER);
  printf("       \t \t %s sets the clock mode (acceptable values are between 0 and 7\n",TXT_CLOCK_PARAMETER);
  printf("       \t \t %s sets the gain mode (acceptable values are between 0 and fff\n",TXT_GAIN_PARAMETER);
  printf("       \t All parameters must be associated to a value.\n");
  printf("       \t Values must be in hexadecimal format.\n");
  return;
} // board version
```


B.10 CEventAnalyzer.h

```
// CEventAnalyzer.h
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_CEVENTANALYZER_H__3E91FE6B_D4AA_4DC8_9F60_CEB6F6B978C3__INCLUDED_)
#define AFX_CEVENTANALYZER_H__3E91FE6B_D4AA_4DC8_9F60_CEB6F6B978C3__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <stdio.h>
#include <fstream>

#include "TDirectory.h"

class CEventAnalyzer
{
public:
    virtual void Initialize(TDirectory *pFile,TDirectory *pRecent){}
    virtual void Clear(){}
    virtual void Start(){ClearIdFinished();printf("ID cleared.\n");}
    virtual void Stop(){}
    virtual void Save(const char *pStrFilename){}
    virtual void SaveBin(FILE *fp){}

    CEventAnalyzer(){m_bEnabled = true;}
    virtual ~CEventAnalyzer(){}

    void SetAnalyzeEnabled(bool b){m_bEnabled = b;}
    bool GetAnalyzeEnabled()const{return m_bEnabled;}

    void CallAnalyzer(){if(m_bEnabled)Analyze();}

    void ClearIdFinished(){m_nIdFinished = 0;}
    int GetIdFinished()const{return m_nIdFinished;}
protected:
    virtual void Analyze() = 0;
    void SetIdFinished(int nID){m_nIdFinished = nID;}
private:
    bool m_bEnabled;
    int m_nIdFinished;    //!
public:
};

#endif // !defined(AFX_CEVENTANALYZER_H__3E91FE6B_D4AA_4DC8_9F60_CEB6F6B978C3__INCLUDED_)
```

B.11 CSocketCtrl.h

CAMAC の同名ファイルと同じ。

B.12 CSocketCtrl.cpp

CAMAC の同名ファイルと同じ。

B.13 atf_socket.h

CAMAC の同名ファイルと同じ。

B.14 streamers.h

```
// streamers.h structures of streamers
#ifndef STREAMERS_H_INC
#define STREAMERS_H_INC

#include "TFile.h"
#include "TArrayI.h"
#include "TObjArray.h"
#include "common_include.h"
#include <vector>

using namespace std;

class CAcousticModule : public TObject
{
public:
    Int_t nModule;
    Int_t nGain;
    Int_t nClock;

    // to store raw data
//    vector<Int_t> arrData[N_ACC_CHANNELS];
    TArrayI arrData[16];

    ClassDef(CAcousticModule,3);
};

class CAcousticData : public TNamed
{
public:

    CAcousticData(){nEventID = nAcqID = -1;}

    Int_t nEventID;
    Int_t nAcqID;

//    CAcousticModule module[N_VME_ACOUSTIC_MODULE];
    CAcousticModule module[4];

    ClassDef(CAcousticData,3);
};

class CAcousticArray : public TNamed
{
public:
    CAcousticArray(){}
    virtual ~CAcousticArray(){}

    Int_t nCount;
    Int_t nCurrent;

    CAcousticData *arr;

    ClassDef(CAcousticArray,3);
};

class CAcousticParam : public TNamed
{
public:
    int m_nBufSize;             // buffer size
    int m_bEveryRead;        // Read every pulse or 3 pulse around BD

    CAcousticParam(){m_nBufSize = 3;m_bEveryRead = false;}
    ClassDef(CAcousticParam,1);
};

template<class T>
T* GetParamFromFile(const char *pName,TDirectory *pRecent)
{
    // Object to return;
    T* pRet = NULL;

    pRet = (T*)(gDirectory->Get(pName));
    if(!pRet){
        // not found. try to get from recent
        T* pTmp = (T*)(pRecent->Get(pName));
        if(pTmp){
            cout << pName << " retrieve from recent." << endl;
            pRet = (T*)(pTmp->Clone());
            pRet->Write();
        }
        else{
            // Finally create;
            cout << pName << " not found." << endl;
            pRet = new T;
            pRet->SetName(pName);
            pRet->Write();
        }
    }
    return pRet;
}

#endif
```

B.15 LinkDef.h

```
#ifdef __CINT__
```

```
#pragma link off all globals;
#pragma link off all classes;
#pragma link off all functions;

#pragma link C++ class CAcousticModule+;
#pragma link C++ class CAcousticData+;
#pragma link C++ class CAcousticArray+;
#pragma link C++ class CAcousticParam+;

#endif
```

B.16 CAnalyzeAcoustic.h

```
// CAnalyzeAcoustic.h
//////////////////////////////////////////////////////////////////////

#ifndef CANALYZEFADC_H_INC
#define CANALYZEFADC_H_INC

#include "CEventAnalyzer.h"
#include "CSocketReceiver.h"
#include "common_include.h"
#include "streamers.h"

class CAnalyzeAcoustic : public CEventAnalyzer , public CSocketReceiver
{
public:

    virtual void Save(const char * pStrFilename);
    virtual void SaveBin(FILE *fp);
    void Analyze();
    virtual void Initialize(TDirectory *pFile,TDirectory *pRecent);
    virtual void Clear();
    virtual bool Receive(istrstream &in,ostrstream &out);
    CAnalyzeAcoustic();
    virtual ~CAnalyzeAcoustic();

private:
    int m_nDataSize;        // Acoustic datasize (must be 1024)
    int m_nCount;            //! entries written
    unsigned int *m_piVmeBuffer[N_VME_ACOUSTIC_MODULE];//!
    // Buffers mapped to the VME registers of acoustic modules
    bool m_bBD;            // Analyze by force : BD mode

    CAcousticArray *m_pArrData;
    TTree *m_pTreeData;
    CAcousticParam *m_pParam;
    int m_nCurrent;            //! current index of Array

public:
    void WriteBinary(FILE *fp);
    virtual void Start();

};

#endif // !defined(AFX_CANALYZEFADC_H__E3558EF3_8368_4B67_A2DF_4709FBAAD6C4__INCLUDED_
```

B.17 CAnalyzeAcoustic.cpp

```
// CAnalyzeAcoustic.cpp
//////////////////////////////////////////////////////////////////////

#include <assert.h>
// ROOT library
#include "TNtuple.h"

#include "myvme.h"
#include "CAnalyzeAcoustic.h"
#include "bdmonitor.h"

#include "TROOT.h"
#include "streamers.h"
#include "CVmeCtrl.h"

//////////////////////////////////////////////////////////////////////
// \z/V?//////////////////////////////////////////////////////////////////////

CAnalyzeAcoustic::CAnalyzeAcoustic()
{
    m_pArrData = NULL;

    // Add to Analyzer
    AddAnalyzer(this);

    // Currently no setting
    AddReceiver(this);
    SetTargetString("Acoustic");

    // Default: enable
    SetAnalyzeEnabled(true);    // Modified 20040225

    m_nDataSize = 1024;
    m_bBD = false;
```

```
    m_nCurrent = 0;
}

CAnalyzeAcoustic::~CAnalyzeAcoustic()
{

}

void CAnalyzeAcoustic::Initialize(TDirectory *pFile,TDirectory *pRecent)
{
    m_pArrData = new CAcousticArray;
    m_pArrData->SetName("acoustic");
    m_pArrData->nCount = 0;
    m_pArrData->arr = NULL;

    m_pTreeData = new TTree("acoustic","acoustic signal");
    m_pTreeData->SetAutoSave((int)1e+9);     // Autosave=1GB(practically no autosave)

    // Retrieve parameter
    m_pParam = GetParamFromFile<CAcousticParam>("AcousticParam",pRecent);

    Clear();

    // attach buffer
    int i;
    for(i=0;i<N_VME_ACOUSTIC_MODULE;i++)
      m_piVmeBuffer[i] = g_vme.GetPtrToModule(i);

}

void CAnalyzeAcoustic::Analyze( )
{
    if(!m_pParam->m_bEveryRead && !m_bBD)
        return;

    int nID = GetIdFinished();
    int i;
    unsigned int maxfifo = 0;
    unsigned int minfifo = 3072;

    // check if all data arrived
    for(i=0;i<N_VME_ACOUSTIC_MODULE;i++)
      {
        unsigned int nfifo = m_piVmeBuffer[i][0x3c/4];

        if(maxfifo < nfifo)maxfifo = nfifo;
        if(minfifo > nfifo)minfifo = nfifo;

        if(nfifo<1024) // data not arrived
          return;
      }
    // all data arrived
    // switch to read mode
    for(i=0;i<N_VME_ACOUSTIC_MODULE;i++)
      {
        unsigned int &nBufMode = m_piVmeBuffer[i][0x20/4];
        nBufMode = MODE_READ_SEND_VALUE;
      }

    cout << (maxfifo >= 3072 ? "!" : maxfifo > 1024 ? "o" : ".") << flush;

    // read data
    while(minfifo>= 1024){
        CAcousticData &data = m_pArrData->arr[m_nCurrent];
        data.nEventID = data.nAcqID = nID; // temporarily
        for(i=0;i<N_VME_ACOUSTIC_MODULE;i++)
          {
            CAcousticModule &mod = data.module[i];
            mod.nModule = i;
            mod.nClock = m_piVmeBuffer[i][0x24/4];
            mod.nGain = m_piVmeBuffer[i][0x28/4];

            int j,k;
            for(j=0;j<m_nDataSize;j++)
              for(k=0;k<N_ACC_CHANNELS/2;k++){
              // 8 registers / module x 2 ch / register = 16 ch / module
            unsigned int buf = m_piVmeBuffer[i][k];
            mod.arrData[2*k][j]   = buf / 65536; // upper bits
            mod.arrData[2*k+1][j] = buf % 65536; // lower bits
              }
          }
        m_nCurrent ++;
        if(m_nCurrent >= m_pParam->m_nBufSize)
            m_nCurrent = 0;

        m_nCount ++;

        // data store complete
        SetIdFinished(++nID);

        minfifo -= 1024;
    }

    // switch to acquire mode
    for(i=0;i<N_VME_ACOUSTIC_MODULE;i++)
      {
        unsigned int &nBufMode = m_piVmeBuffer[i][0x20/4];
        nBufMode = MODE_ACQUIRE_SEND_VALUE;
      }

}
```

```
bool CAnalyzeAcoustic::Receive(istrstream &in,ostrstream &out)
{
    char s[256];

    in >> std::setw(255) >> s;

    cout << "AnalyzeAcoustic : Command " << s << " received." << endl;

    // Parse message string
    if(!strcmp(s,"Enable"))
    {
        int bEnable;

        in >> bEnable;
        SetAnalyzeEnabled(bEnable);

        return !in.fail();
    }
    if(!strcmp(s,"GetEnable"))
    {
        return GetAnalyzeEnabled();
    }
    else if(!strcmp(s,"SetBufferSize"))
    {
        int nSize;
        in >> nSize;

        if(in.fail())return false;

        m_pParam->m_nBufSize = nSize;
        // clear data
        Clear();

        return true;
    }
    else if(!strcmp(s,"GetBufferSize"))
    {
        out << m_pParam->m_nBufSize << " ";
        return true;
    }
    else if(!strcmp(s,"SetReadEvery"))
    {
        int bEvery;
        in >> bEvery;

        m_pParam->m_bEveryRead = bEvery;

        return true;
    }
    else if(!strcmp(s,"GetReadEvery"))
    {
        out << m_pParam->m_bEveryRead << " ";
        return true;
    }
    else if(!strcmp(s,"SetGain"))
    {
        unsigned int nGain;
        unsigned int nCh;

        in >> nCh;

        // Ch number validation
        if(nCh > N_VME_ACOUSTIC_MODULE || nCh == 0)
        {
            cout << "Invalid ch #." << endl;
            return false;
        }

        in >> nGain;

        m_piVmeBuffer[nCh-1][0x28/4] = nGain;

        return true;
    }
    else if(!strcmp(s,"GetGain"))
    {
        int nCh;
        in >> nCh;
        // Ch number validation
        if(nCh > N_VME_ACOUSTIC_MODULE || nCh == 0)
        {
            out << 0;
            cout << "Invalid ch #." << endl;
            return false;
        }
        out << m_piVmeBuffer[nCh-1][0x28/4] << endl;
        return true;
    }
    else if(!strcmp(s,"SetClock"))
    {
        int nCh;
        in >> nCh;
        // Ch number validation
        if(nCh > N_VME_ACOUSTIC_MODULE || nCh == 0)
        {
            cout << "Invalid ch #." << endl;
            return false;
        }

        int nClock;
        in >> nClock;
        m_piVmeBuffer[nCh-1][0x24/4] = nClock;

        return true;
```

```
    }
    else if(!strcmp(s,"GetClock"))
    {
        int nCh;
        in >> nCh;
        // Ch number validation
        if(nCh > N_VME_ACOUSTIC_MODULE || nCh == 0)
        {
            out << 0;
            cout << "Invalid ch #." << endl;
            return false;
        }

        out << m_piVmeBuffer[nCh-1][0x24/4] << " ";
        return true;
    }
    else
        throw("Message invalid.");

    // not reach here;
    return false;
}

void CAnalyzeAcoustic::Save(const char * pStrFilename)
{
    // Analyze last data
    m_bBD = true;
    Analyze();
    m_bBD = false;

        cout << "Save to root file..." << endl;

    char s[1024];
    sprintf(s,"Acoustic signal at %s",pStrFilename);

/*
    m_pArrData->SetTitle(s);
    m_pArrData->nCurrent = m_nCurrent;
    m_pArrData->Write();
*/
    m_pTreeData->SetTitle(s);
    CAcousticData *p = new CAcousticData;
    m_pTreeData->Branch("acoustic","CAcousticData",&p);

    int i = m_pArrData->nCurrent;

    do{
        p = &m_pArrData->arr[i];
        m_pTreeData->Fill();
        i++;
        i%= m_pArrData->nCount;
    }while(i!=m_pArrData->nCurrent);

    cout << "Save finished." << endl;
}

void CAnalyzeAcoustic::SaveBin(FILE *fp)
{
    WriteBinary(fp);
}

void CAnalyzeAcoustic::Clear()
{
    int i,j,k;
    // Initialize data buffer
    if(m_pParam->m_nBufSize != m_pArrData->nCount)
    {
        cout << "Remaking buffer. BufSize = " << m_pParam->m_nBufSize << endl;
        delete[] m_pArrData->arr;
        m_pArrData->arr = new CAcousticData[m_pParam->m_nBufSize];
//        m_pArrData->arr.resize(m_pParam->m_nBufSize);
        m_pArrData->nCount = m_pParam->m_nBufSize;

        for(i=0;i<m_pParam->m_nBufSize;i++)
        {
            for(j=0;j<N_VME_ACOUSTIC_MODULE;j++)
                for(k=0;k<N_ACC_CHANNELS;k++)
                    m_pArrData->arr[i].module[j].arrData[k].Set(m_nDataSize);
        }
    }

     m_nCount = 0;
    m_nCurrent = 0;

    m_pTreeData->Reset();

    CEventAnalyzer::Clear();
}

void CAnalyzeAcoustic::WriteBinary(FILE *file)
{
    /* File format data */
    fwrite(FILE_PREFIXE,sizeof(char),strlen(FILE_PREFIXE),file);
    int toWrite=FILE_VERSION;
    fwrite(&toWrite,sizeof(int),1,file);

    //Number of records written
    toWrite=N_RECORDS;
    fwrite(&toWrite,sizeof(int),1,file);

    //ADC clock (register 24H)
    fwrite(&m_piVmeBuffer[0][0x24/4],sizeof(uint32_t),1,file);

    //ADC gain (register 28H)
```

```
fwrite(&m_piVmeBuffer[0][0x28/4],sizeof(uint32_t),1,file);

//Number of boards
int nBoards = N_VME_ACOUSTIC_MODULE;
fwrite(&nBoards,sizeof(uint32_t),1,file);

//Read the sensors settings
FILE *sensorsSettings=fopen(SENSORS_FILE,"r"); /* opening the dumpinfo file */

int sensorsSettingsVersion=-1;
if (sensorsSettings!=NULL) {
  char* sensorsSettingsStr=(char *)malloc(MAXLENGTH*sizeof(char));
  fread(sensorsSettingsStr,sizeof(char),MAXLENGTH,sensorsSettings);
  char* c=sensorsSettingsStr;
  sensorsSettingsVersion=0;
  while (((*c)>='0')&&((*c)<='9')) {
    sensorsSettingsVersion*=10;
    sensorsSettingsVersion+=(int)((*c)-'0');
    c++;
  }
  //      printf("Sensor settings %s / %d \n",sensorsSettingsStr,sensorsSettingsVersion);
  fclose(sensorsSettings);
}

//Write the sensors settings in the data file
fwrite(&sensorsSettingsVersion,sizeof(uint32_t),1,file);

int boardLoop;
unsigned int *ptrToBoard[4];

int i;
for(i=0;i<4;i++)ptrToBoard[i] = m_piVmeBuffer[i];

for (boardLoop=0;boardLoop<N_VME_ACOUSTIC_MODULE;boardLoop++) {

  if (ptrToBoard[boardLoop]!=0) {
    //Board separator
    toWrite=BOARD_SEPARATOR;
    fwrite(&toWrite,sizeof(int),1,file);

    //ADC clock (register 24H)
    uint32_t clock = readValue(ptrToBoard[boardLoop],BOARD_CLOCK_CONTROL_LOCATION,SILENT);
    fwrite(&clock,sizeof(int),1,file);

    //ADC gain (register 28H)
    uint32_t gain = readValue(ptrToBoard[boardLoop],BOARD_GAIN_CONTROL_LOCATION,SILENT);
    fwrite(&gain,sizeof(int),1,file);

    //trig freq (register 38H)
    uint32_t trigFreq=readValue(ptrToBoard[boardLoop],BOARD_TRIGGER_FREQ_LOCATION,SILENT);
    fwrite(&trigFreq,sizeof(uint32_t),1,file);

    //clock freq (register 38H)
    uint32_t clockFreq=readValue(ptrToBoard[boardLoop],BOARD_CLOCK_FREQ_LOCATION,SILENT);
    fwrite(&clockFreq,sizeof(uint32_t),1,file);

    //board address
    uint32_t addrBoard=s_nAddr[boardLoop];
    fwrite(&addrBoard,sizeof(uint32_t),1,file);

    toWrite = (m_nCount > m_pParam->m_nBufSize ? m_pParam->m_nBufSize : m_nCount);
    fwrite(&toWrite,sizeof(int),1,file);

    toWrite=0;
    fwrite(&toWrite,sizeof(uint32_t),1,file);

    // put raw data: format may be data[Time][Channel]
    int nTime,nCh;
    int i = m_nCurrent;
    if(m_nCount < m_pParam->m_nBufSize)i = 0;

    do
    {
        CAcousticModule &data = m_pArrData->arr[i].module[boardLoop];
        for(nTime = 0;nTime<MAX_N_RECORDS;nTime++){
            for(nCh = 0;nCh < 8;nCh ++){
                unsigned short s[2];
                s[0] = data.arrData[nCh*2][nTime];
                s[1] = data.arrData[nCh*2+1][nTime];
                fwrite(&s,2,2,file);
            }
        }
        i++;
        if(i >= m_pParam->m_nBufSize)i = 0;

    }while(i != m_nCurrent);
  } // board valid
} // for each board
}

void CAnalyzeAcoustic::Start()
{
       int i;

  // switch to acquire mode
  for(i=0;i<N_VME_ACOUSTIC_MODULE;i++)
    {
      unsigned int &nBufMode = m_piVmeBuffer[i][0x20/4];
      nBufMode = MODE_ACQUIRE_SEND_VALUE;
    }

  cout << "AcousticAnalyzer start." << endl;
  CEventAnalyzer::Start();
}
```

B.18 Makefile

```
# Makefile ( n source file 1 execute file version )
# 2002/12/ 4

PACKAGE    = bdmonitor
SRCS    = bdmonitor.cpp CAnalyzeAcoustic.cpp CVmeCtrl.cpp CSocketReceiver.cpp \
CMasterCtrl.cpp CSocketCtrl.cpp myvme.cpp rootdict.cpp
HEADS    = CAnalyzeAcoustic.h CVmeCtrl.h CEventAnalyzer.h CSocketReceiver.h \
bdmonitor.h CMasterCtrl.h CSocketCtrl.h myvme.h rootdict.h
OBJS    = $(SRCS:.cpp=.o)

FILES    = README Makefile $(HEADS) $(SRCS)
VER    = `date +%Y%m%d`

# common (*.o)
LD    = gcc
LDFLAGS    = -g $(DEBUG)
LDLIBS    = -lm $(shell root-config --libs) -L/mnt/nas/data/lib/atf_socket/archive \
-latf_socket_functions -L/mnt/nas/data/lib/vmisft-7433-3.2/vme_universe/lib -lvme

# C (*.c)
CC    = gcc
CFLAGS    = -g -O2 -Wall $(DEBUG) -Wno-deprecated
CPPFLAGS= -I.

# C++ (*.cc,*.cpp)
CXX    = g++
CXXFLAGS= -g -O2 -Wall $(DEBUG) -Wno-deprecated
INCLUDE=/cern/root/include

### rules ###

.SUFFIXES:
.SUFFIXES: .o .c .cc .cpp .f .p

all: $(PACKAGE)

$(PACKAGE): $(OBJS)
    $(LD) $(LDFLAGS) $(OBJS) -o $@ $(LDLIBS)

$(OBJS): $(HEADS) Makefile

.c.o:
    $(CC) $(CFLAGS) $(CPPFLAGS) -c $< -o $@
.cc.o:
    $(CXX) $(CXXFLAGS) $(CPPFLAGS) -c $< -o $@
.cpp.o:
    $(CXX) $(CXXFLAGS) $(CPPFLAGS) -I$(INCLUDE) -c $< -o $@

dict:
    rootcint -f rootdict.cpp -c streamers.h LinkDef.h

clean:
    $(RM) $(PACKAGE) $(OBJS)
    $(RM) core gmon.out *~ #*#
```

付 録C　BDMS-CAMAC 通信メッセージ一覧

1.Category "Main"
測定開始/終了/保存など、測定全体に関わるコマンド群です。

メッセージ:Start
パラメータ:なし
応答:OK/NG
説明:
測定を開始します。すでに測定中の場合は NG を返します。

メッセージ:Stop
パラメータ:int bBreakDown (0/normal,1/breakdown)
応答:OK/NG
説明:
測定を停止します。測定中でない場合は NG を返します。
bBreakDown を 0 にした場合は即時停止します。
bBreakDown を 1 にした場合は BreakDown のシーケンス終了後に停止します。
AutoSave が 1 になっていた場合、停止後自動で Save します。
さらに AutoStart が 1 になっていた場合、Save 後自動で Start します。
（AutoStart は実装されていますが使わないことになっています）

メッセージ:Status
パラメータ:なし
応答:OK/NG (OK/測定中,NG/測定停止)
説明:
現在の動作状態を取得します。

メッセージ:AutoSave
パラメータ:int bAutoSave (0/no,1/yes)
応答:OK のみ
説明:
AutoSave を On/Off します。

メッセージ:AutoStart
パラメータ:int bAuboStart (0/no,1/yes)
応答:OK のみ
説明:
AutoStart を On/Off します。

メッセージ:Status
パラメータ:なし
応答:OK/NG
説明:
現在測定中かどうか調べます。OK なら測定中、NG なら停止中です。

メッセージ:Clear
パラメータ:なし
応答:OK のみ
説明:
測定データをクリアします。保存した後クリアが必要です。

メッセージ:GetLastFilename
パラメータ:なし
応答:string strFilename (ファイル名)

OK/NG (NG/保存されていない)
説明:
最後に保存したファイル名を返します。
まだ保存していないときは空白行と NG が返ります。

メッセージ:SetInterlock
パラメータ:int nInterlock (1/Interlock 出力 ON,0/OFF)
応答:OK/NG (NG/通信エラーのみ)
説明:
放電検出時に Fast RF Interlock モジュールへ出力するための
Output Register のパルス出力を ON/OFF します。

メッセージ:GetInterlock
パラメータ:int nInterlock (1/Interlock 出力 ON,0/OFF)
応答:int nInterlock (1/出力 ON,0/OFF)
OK のみ
説明:
Interlock 用パルス出力の ON/OFF を取得します。

2.Category "Trigger"
トリガーコントロールのためのメッセージ群です。

メッセージ:Enable
パラメータ:int bEnable (0/disable,1/enable)
応答:OK/NG
説明:
Trigger Control の On/Off を行います。
Off の場合、Trigger Control は行いません (垂れ流しになります)。
この場合、BreakDown 関係の測定は行いません。
(BreakDown が起きても何も起こりません)
On の場合は、Trigger Control および BreakDown 検知を行います。

メッセージ:GetEnable
パラメータ:なし
応答:OK/NG (OK/enable,NG/disable)
説明:
Trigger Control 状態を取得します。

メッセージ:SetAfterRun
パラメータ:int nAfterRun[4] (放電検出後通過パルス数)
応答:OK/NG (NG/通信エラーのみ)
説明:
放電検出後何パルス通過させるかを CSY のチャンネルごとに指定します。
0 なら放電検出したパルスで止めます。

メッセージ:GetAfterRun
パラメータ:
応答:int nAfterRun[4]
OK のみ
説明:
上記 SetAfterRun で設定したパルス数を取得します。

3.Category "RF"
RF 測定に関するメッセージ群です。

メッセージ:Enable
パラメータ:int bEnable (0/disable,1/enable)
応答:OK/NG
説明:
Disable にした場合、RF の測定はいっさい行いません。
データの保存もありません。

メッセージ:GetEnable
パラメータ:なし
応答:OK/NG
説明:
RF 測定 enable/disable 状態を取得します。

メッセージ:BDParamNew
パラメータ:int nVersion (1 固定)
int nCh (1-4)
int nMode (0/Peak,1/Integral,2/PeakRelative,3/IntegralRelative)
int use (0/detection しない,1/する)
int rangeMin
int rangeMax
int start
int limitMin
int limitMax

```
          int useBaseline(0/Baseline を引かない,1/引く)
応答:OK/NG (NG:通信エラー)
説明:

メッセージ:GetBDParamNew
パラメータ:int nVersion (1 固定)
応答:Ch1Peak-use,Ch1Peak-rangeMin,....,Ch1Peak-useBaseline,Ch1Integral-use,.OK,
     Ch1PeakRelative-use,...,Ch1IntegralRelative-use,...,Ch2Peak-use,.
     という順に、7x4x4=108 個の int が返る。
     OK のみ
説明:

メッセージ:BDParam
メッセージ:GetBDParam
旧バージョンのため省略。

メッセージ:Baseline
パラメータ:int nVersion (1 固定)
           int nCh (1-4)
           int use (0/Baseline 引かない,1/固定 Baseline,2/Auto Baseline)
           int baseline (use==1 の場合の固定 Baseline)
           int autoBaselineRangeMin (use==2 の場合の Baseline 範囲下限)
           int autoBaselineRangeMax (use==2 の場合の Baseline 範囲上限)
応答:OK/NG (NG:通信エラー)
説明:

メッセージ:GetBaseline
パラメータ:int nVersion (1 固定)
           int nCh (1-4)
応答:int use (0/Baseline 引かない,1/固定 Baseline,2/Auto Baseline)
     int baseline (use==1 の場合の固定 Baseline)
     int autoBaselineRangeMin (use==2 の場合の Baseline 範囲下限)
     int autoBaselineRangeMax (use==2 の場合の Baseline 範囲上限)
説明:

メッセージ:SetBufferSize
パラメータ:int size
応答:OK/NG (NG/通信エラー)
説明:
RF FADC リングバッファのサイズを設定します。
現在のデータはクリアされます。

メッセージ:GetBufferSize
パラメータ:
応答:int size
説明:
現在のリングバッファのサイズを返します。

メッセージ:SetVerbose
パラメータ:int bVerbose (1/詳細,0/簡易)
応答:OK
説明:
コンソールへのメッセージ表示を設定します。
bVerbose=1 で詳細なメッセージを表示します。

メッセージ:GetVerbose
パラメータ:
応答:int bVerbose
     OK
説明:
現在のメッセージ表示モードを取得します。

メッセージ:SetInterval
パラメータ:int nInterval (周期保存パルス数)
応答:OK
説明:
RF 波形周期保存の頻度を設定します。

メッセージ:GetInterval
パラメータ:
応答:int nInterval
     OK
説明:
RF 波形周期保存の頻度を取得します。

メッセージ:GetCurrentData
パラメータ:
```

応答:いろいろ
　　OK/NG(NG/波形がない)
説明:
最新の RF 波形を返し ます。
フォーマットはバイナリで、ファイルの形式と同じです。
ただし<MESSAGE>-</MESSAGE>間はありません。
波形は最新の 1 イベントのみです。

4.Category "XRay"
X 線測定に関するメッセージ群です。
メッセージ:Enable
パラメータ:int bEnable (0/disable,1/enable)
応答:OK/NG
説明:
Disable にした場合、X 線の測定はいっさい行いません。
データの保存もありません。

メッセージ:GetEnable
パラメータ:なし
応答:OK/NG
説明:
X 線測定 enable/disable 状態を取得し ます。

メッセージ:BDEnable
パラメータ:int bEnable (0/disable,1/enable)
応答:OK/NG
説明:
X 線放電検出の有効/無効を設定し ます。

メッセージ:GetBDEnable
パラメータ:なし
応答:OK/NG
説明:
X 線放電検出有効/無効を取得し ます。

メッセージ:SetADCOffset
パラメータ:int nOffset * 12
応答:OK/NG
説明:
12 個のパラメータをとり、各 Adc のチャンネルにオフセットをつけます。

メッセージ:SetChTDCOrigin
パラメータ:int nCh
応答:OK/NG
説明:
TDC の Origin をとるためのチャンネル#(RF を入れている Ch) を入れます。

メッセージ:SetBDParam
パラメータ:int version (1 固定)
　　int threshold
　　int nChannels
応答:OK/NG(NG/通信エラー)
説明:
X 線放電検出設定を行います。
前パルスとの比較で threshold を越えたチャンネルが nChannels 以上
あった場合放電とみなし ます。

メッセージ:SetBDParam
パラメータ:int version (1 固定)
応答:int threshold
　　int nChannels
　　OK/NG(NG/通信エラー)
説明:
X 線放電検出設定を取得し ます。

付 録D　BDMS-VME 通信メッセージ一覧

1.Category "Main"
測定開始/終了/保存など、測定全体に関わるコマンド群です。

メッセージ:Start
パラメータ:なし
応答:OK/NG
説明:
測定を開始します。すでに測定中の場合は NG を返します。

メッセージ:Stop
パラメータ:なし
応答:OK/NG
説明:
測定を停止します。測定中でない場合は NG を返します。

メッセージ:Status
パラメータ:なし
応答:OK/NG (OK/測定中,NG/測定停止)
説明:
現在の動作状態を取得します。

メッセージ:Save
パラメータ:string strFilename(ファイル名)
応答:OK のみ
説明:
現在のデータを保存します。
保存は root, バイナリ両方の形式に行います。

2.Category "Acoustic"
Acoustic 測定に関するメッセージ群です。

メッセージ:Enable
パラメータ:int bEnable (0/disable,1/enable)
応答:OK/NG
説明:
Disable にした場合、Acoustic データの収集は行いません。

メッセージ:GetEnable
パラメータ:なし
応答:OK/NG
説明:
Acoustic 測定 enable/disable 状態を取得します。

メッセージ:SetReadEvery
パラメータ:int bEvery (0/Module 蓄積モード,1/毎パルス読み込みモード)
戻り値　　:OK/NG

読込モードを切り替えます。
Module 蓄積モードは 3pulse しか蓄えられませんが取りこぼしはありません。
毎パルス読込モードは毎パルス PC へ取り込むのでパルス数制限がありません
が取りこぼします (12Hz 以下推奨)

メッセージ:GetReadEvery

戻り値　　:int bEvery (0/Module 蓄積モード ,1/毎パルス読み込みモード)
　　OK

上記読込モード の取得です。

メッセージ:SetBufferSize
パラメータ:int nSize
戻り値　　:OK/NG

取得するパルス数を指定します。
Module 蓄積モードでは、3 に設定してください。
毎パルス読込モードでは、任意の値が設定できます。
メモリを取得/解放するため、やや時間がかかります。

メッセージ:GetBufferSize
戻り値　　:int nSize
　　OK

上記取得パルス数を取得します。

メッセージ:SetGain
パラメータ:int module (1-4)
　　int gain
戻り値　　:OK/NG

module ごとの Gain を設定します。
設定する値は前のメールの通りです。

メッセージ:GetGain
パラメータ:int module (1-4)
戻り値　　:int gain
　　OK/NG

module ごとの Gain を取得します。

メッセージ:SetClock
パラメータ:int module (1-4)
　　int clock
戻り値　　:OK/NG

module ごとの Clock を設定します。
設定する値は前のメールの通りです。

メッセージ:GetClock
パラメータ:int module (1-4)
戻り値　　:int clock
　　OK/NG

module ごとの Clock を取得します。