Tutorial for Running UNIDAQ on DECstation with VME-CAMAC Interface

(in Japanese)

Masaru Tairadate

Department of Accelerator Science
School of Mathematical and Physical Science
The Graduate University for Advanced Studies

Abstract

The tutorial is an introduction to UNIDAQ(Portable Data Acquisition System) for DECstation. The tutorial is based on the UNIDAQ Documentation Set. The tutorial describes the essential terms extracted from various functions of the UNIDAQ and the useful terms which will be supported on the future version of UNIDAQ. If you read the tutorial, you will be able to install the UNIDAQ, collect data of PSEUDO/CAMAC events, record data to file, and analyze data with PAW(CERN library). The tutorial is intended for the person who has no experience with using data acquisition system, but has basic knowledge of CAMAC, UNIX, FORTRAN, C, HBOOK and PAW.

目 次

DECstation で UNIDAQ を走らせるための tutorial

平舘　優[1]
総合研究大学院大学 数物科学研究科 加速器科学専攻[2]

Dec. 16, 1994

---

[1] E-mail: taira@kek.jp
[2] 高エネルギー物理学研究所 データ処理センター

Abstract

UNIDAQ はSSC プロジェクトの実験グループ (SDC) によって開発された汎用データ収集システムである。この tutorial は DECstation で UNIDAQ ver2.3 を走らせるためのものであり、UNIDAQ Documentation Set ver2.2(Installation Guide, User's Guide, Technical Guide) に基づいて作成した。UNIDAQ の全機能の中から必要な最低限の機能を抜き出し、次のバージョンへ実装されるであろう機能を加え、「これだけあればとりあえずデータ収集可能な」UNIDAQ システムを述べる。この tutorial によって、CAMAC の読みだしを行なわないで疑似データを生成する Pseudo event 及び CAMAC からデータを読みだす User event を使用しデータを作成して、データの記録／解析／表示を行なう UNIDAQ データ収集システムの実装／使用法を知ることができる。この tutorial だけで一通りのことが出来るよう配慮したつもりだが、もっと詳しい説明がほしい、あるいは更に応用的な使い方をしたいという人のために、Documentation Set の参照が望ましいと思われる部分には随時その旨を記した。

なお、この tutorial はデータ収集システムの経験はなくとも CAMAC、UNIX、FORTRAN、C、HBOOK/PAW(CERN-lib) については基本的な知識を持っているという人を対象とする。

Chapter 1

Concept of UNIDAQ

最近のパーソナルコンピュータ (PC) やワークステーション (WS) の性能向上には目を見張るものがある。CPU 性能のみならずディスクやメモリの価格低下も著しい。またネットワークによる異機種間接続ももはや当り前のこととなった。大学の研究室などでもすでに 3 ~ 4 台の WS を導入し、PC と LAN によって接続されているところが多いことであろう。外部との接続も、通信コストの問題は依然としてあるものの、一般的になってきている。とくに WS は CPU ・OS ・バスともに主に 32bit で設計されており、ネットワーク機能を標準装備していることもあり汎用機からの移行が急速かつスムースに行われた。

このような WS の急速な普及は、RISC-CPU の搭載によるマシンの小型 ・低価格化というハードウエアの側面もさることながら、OS に UNIX を採用したというソフトウエアの側面によるところが決して少なくない。UNIX は小型で柔軟性があるうえ仕様もソースコードも公開された数少ない OS であったため、WS ベンダ各社は競って自社のマシンに移植した。メーカーが異なっても同じ OS を搭載している点がユーザに受け、価格の魅力と相まって市場は一気に拡大した。

高エネルギー物理の分野でよく使われるデータ解析ソフトウエア、CERN library も多くの WS の上で動くようになっており、WS は高エネルギー物理における検出器の設計 ・開発 ・性能評価や実験データの解析をする上で既に欠かせないものとなっている。

一方データ収集 (DAQ) の分野においては、UNIX-WS はこれまでほとんど活躍の場を与えられていなかった。その理由は

UNIX はリアルタイム OS ではない

という点につきる。データ解析のときと違い、コンピュータの不感時間 (dead time) はデータ収集に重大な影響を及ぼす。不感時間に制限がない、すなわちリアルタイム性の保証されていない OS は DAQ には向かないというのがこれまでの一般的な見方であった。しかし、100MIPS を超える高性能 CPU の出現やデバイスドライバの改良、また UNIX 自身のチューン ・アップなどによって UNIX も DAQ に使えるレベルに到達している。実際、最近の機種を用いたベンチマークテストによると VMS と同等あるいはそれ以上のパフォーマンスを実現している。すなわち高エネルギー実験の DAQ が要求する程度のリアルタイム性を事実上確保したといえる。

パフォーマンスが同等なら移植性の高い UNIX のほうが有利であり、ハードウエアのコストも安くてすむ WS と組み合わせることでポータブルな DAQ システムを構築できる。さらに、CPU が直接 CAMAC/FASTBUS/TKO 等の回路システムを制御する従来の形態を用いず、ボードビジネスの世界で標準バスとして活躍している VMEbus を仲介させる。これにより、CPU(WS) が変わっても容易にシステムを再構築できる[1] 。このように、VME を外部バスとした UNIX-WS 上に作られた DAQ システムが UNIDAQ

---

[1] CAMAC や FASTBUS は非常に限られた分野でしか使われていないので、計算機バスとのインタフェースボードを開発してい

である。以下、UNIDAQ の特徴を述べる。

分散処理　UNIDAQ の大きな特徴として分散処理が上げられる。まず、データ収集に必要なタスクを段階的に区分けし、それぞれのタスクに対しプロセスを与えた。例えば CAMAC からデータを取り込み buffer に入れる部分を collector プロセス、buffer からデータを読みファイルに格納する部分を recorder プロセス、buffer からデータを読みヒストグラムプログラムに入れる部分を analyzer プロセスというように割り当てた。そしてこれらのプロセスを管理する buffer manager(NOVA [2]) を設け、各々のプロセスがいまどのような状態にあるのか、また buffer 内のデータの様子を監視できるようにした。このような形態にすることで、ユーザが必要としているタスクのみを実行できるほか、あるタスクの走っているプロセスがダウンしても他のタスクに影響を及ぼさない。

Portability　UNIX の高い移植性を損なわないよう UNIDAQ は作られている。UNIDAQ はプロセスが統一的な構造を持っているため目的に応じたプロセスを容易に作成することができる。

既存のアプリケーションの利用　CAMAC ライブラリは国内外で標準的に使われだした KEK 版を、ヒストグラム/グラフ描画ライブラリとしては高エネルギー物理学でよく使われる HBOOK/PAW(CERN library) を採用した。この 2 つのアプリケーションは様々なプラットフォームに対応しており、UNIDAQ の中からこれらのライブラリを利用する場合も、その作法や操作性には何等変わりがない。もちろん出力ファイルの互換性も保たれている。したがってこれらのアプリケーションを既に利用したことのあるユーザが新たに学習すべきことは特にない。

Networking　UNIDAQ は複数のコンピュータシステムを 1 つのデータ収集システムとしてサービスする機能を有している。そのため、1 つのコンピュータシステムからなる小さな実験用 UNIDAQ システムから高速で大きな実験用に構築された異機種コンピュータの複合体からなる UNIDAQ システムまでをカバーすることができる。もちろん、これら異機種コンピュータ上での UNIDAQ の外部仕様は統一されており、コンピュータの違いによる使い勝手の差ができるだけないように設計されている。現在までに UNIDAQ がサポートしているコンピュータシステムを以下に挙げておく。

	機種 (CPU)	OS	Bus	VME interface
Sun	SPARCstation (SPARC)	SunOS, Solaris	SBus	SFVME-100 [3], PT-SBS915 [4]
DEC	DECstation (MIPS R3000)	ULTRIX	TURBOchannel	DEC PMABV/T6000
HP	HP 9000 (PA-RISC)	HP-UX	EISA	BIT3
HP	HP 9000 (PA-RISC)	HP-RT	VMEbus	(VME ボードコンピュータ)
SGI	IRIS (MIPS R3000/4000)	IRIX	VMEbus	——

また UNIX ではないが 68K の VxWorks [5] システムの上でも動く。なお、現在 DEC Alpha AXP(DEC OSF/1) への移植作業が行われている。

るベンダは少ない。これに対し計算機バス-VMEbus インタフェースについては多くのボードメーカーがが供給を行っている。

[2] **N**ew **O**n-line **V**ME based data **A**cquisition system.

[3] Solflower Computer, Inc.

[4] Performance Technologies, Inc.

[5] WindRiver Inc.

このように UNIDAQ は様々な機能を有しているがゆえ、とくに第 2 版以降規模が大きくなってきた。初めてデータ収集システムをやろうという人にとってこの全ての機能を理解することは困難であり、またその必要もない。この tutorial はそのような DAQ の初心者、あるいは UNIDAQ に初めて接する人のために、UNIDAQ 全体の見通しをよくするためのものである。これから述べる UNIDAQ システムは、DECstation5000 シリーズ計算機に DEC 製 VME アダプタを接続し VME-CAMAC インターフェイスに Kinetic 社のモデル 2917 を使用するハードウエアに、OS として ULTRIX(UNIX) を搭載した、シングルコンピュータシステムの上に実装されることを前提とする。

Chapter 2

Installation

本章ではまず、UNIDAQをインストールするにあたって必要な環境について述べ、配布パッケージの入手と展開方法、そして共用ディレクトリへのインストール、個人ディレクトリへのインストールと順次述べていくことにする。

2.1 Requirements

UNIDAQはそのインストール時及び実行時にFORTRAN、Cの各compiler及びlibraryを必要とするばかりでなく、CERNライブラリがインストールされていることが必要である。Window systemとしてX11、ヒストグラムプログラムとしてHBOOK、データ解析プログラムとしてPAWを利用する。ユーザはUNIDAQをインストールしようとしているマシンにあらかじめこれらの実行環境をつくっておく必要がある（Installation Guide 2.1参照）。また、UNIDAQ配布パッケージにはCAMACやVMEのデバイスドライバは含まれていない。UNIDAQのインストール時にはそれらのデバイスドライバがインストールされていることが前提とはならないものの、実際にCAMACやVMEを使用するプログラムを動かすときは必ずインストールされていなければならない。従って、このtutorialでは必要である。CAMAC/VMEのデバイスドライバおよびライブラリは、onlnews.kek.jpに

/pub/kek/camac/camacv1.0_decs.tar.Z
/pub/kek/camac/camlib-docs-1.0.tar.Z
/pub/kek/vme/gendev1.01_decs.tar.Z

として置いてあるのでanonymous FTPで取り寄せることができる[1]。実装の手順については各ドキュメントを参照のこと。

なお、インストールに際しおよそ10MBのディスク空き容量を確保しておかなければならない。本tutorialは次の環境下で作成された。

Workstation	: DECstation 5000/125 32MB-memory 2.2GB-disk
OS	: Ultrix ver 4.2a (lev. 47)
Window system	: X11 Release 4 (UWS 4.2a)
TURBOchannel-VME adaptor	: DEC PMABV/T6000
VME-CAMAC Interface	: Kinetic model 2917

---

[1] これらのデバイスドライバの実装時に新しいUNIXのカーネルを作る必要がある。UNIDAQをインストールしようとするマシンのシステム管理者に、あらかじめドライバ及びそのライブラリの実装の有無を（実装されている場合はそのバージョンも）確認すること。

Compiler : DEC FORTRAN v3.2-164,
MIPS C 2.10
CERN library : PAW ver 2.04/15, HBOOK ver 4.14 (CERN-lib 94a)
CAMAC library : CAMAC library for UNIX ver 1.0 (by KEK)

2.2 Getting and Expanding Distributed Package

UNIDAQパッケージの入手は誰でも可能だが、ここではその展開・共有ディレクトリへのインストールはマシンのスーパーユーザが行うものとする。以降、# はスーパーユーザプロンプトを意味する。また、ユーザがコンソール入力すべき部分は`typewriter`体で示した。

UNIDAQのtop directoryを例えば/usr/localにつくりそこにカレントを移す。

```
# mkdir /usr/local/UNIDAQ
# cd /usr/local/UNIDAQ
```

ftpによりonlnews.kek.jpからパッケージを持ってくる。

```
# ftp onlnews.kek.jp
Name (onlnews:root): anonymous
Password: [your mail address]
ftp> cd pub/Unidaq/Unidaq2.3
ftp> binary
ftp> get Unidaq_v2.3.tar.Z
ftp> bye
```

パッケージを解凍・展開する。

```
# zcat Unidaq_2.3.tar.Z | tar xovf -
```

Documentation Setが手元にない場合は、この段階で /usr/local/UNIDAQ/v2.3/doc にDocumentation SetのPostScriptファイルがある[2]ので、これを出力しておくとよいだろう(100ページ程度)。

2.3 Installation to Common Directory

インストール作業は2段階に分かれる。まず、マシンの共用ディレクトリにフルインストールを行い、次に各ユーザのホームディレクトリにパーソナルインストールを行う。この作業によりユーザは自分の好きなようにプロセスのカスタマイズが行えるようになる。

WSはOSとしていずれもUNIXを採用しているとはいえ、各メーカーあるいは機種によってその環境は多種多様である。また、同一機種であってもマシンごとに使用環境は異なる。UNIDAQは、個々のマシンの使用環境に合わせて作業が出来るようにいくつかの環境変数を用意している。ここではその設定の仕方について記す。

パッケージを展開した時点でdirectory UNIDAQの下にv2.3というdirectoryができる。そこにカレントを移す。

```
# cd v2.3
```

シェルスクリプトinstall.cshを実行する。各設定値に対し、defaultでよい場合はリターンキーを、そうで

---

[2] 1994年12月現在のバージョンは2.2である。したがってパッケージの内容と異なる部分がいくつかある。とくにインストレーションに関してはこのtutorialに沿って行うこと。

ない場合は適切な値を入力する。

```
# ./install.csh
Setting UNIDAQ root directory.
TOPUNIDAQ = [/usr/local/UNIDAQ/v2.3]
Is this installation for:
      ULTRIX = 1
      VXWORKS = 2
            Make your choice now [1]
Setting CERN library root.
CERN_ROOT = [/usr/local/cern/v92a] /kek/cern/94a
Setting camac/vme library directory. Please chooose from:
      UNIDAQ distribution (v 1.0)  = 1
      Your own local directory    = 2
      camvme library with DEC turbo = 3
          Make your choice now [1]
    Enter the top camac directory [/usr/local/vme/camac]
Setting the MURMUR root (optional; if not used, just take the defaults).
MURMUR_DIR = [/usr/murmur/v1_1]
Setting the IP address where the murmur server will run.
As with the MURMUR dir, if you don't use murmur, just take the default.
MURMUR_SERVER_ADD = [134.3.30.43]
Setting the location of the log book file.
Force log.bookYYMMDD files in $TOPUNIDAQ/log for system install.
```

ここで環境変数設定ファイル setup.csh がサブディレクトリ conf に作成される。このファイルには上記の環境変数の他、X11 library、C compiler、F77 compiler、UNIDAQ bin、man directory 等の設定も含まれる。次に setup.csh を実行する（環境変数を有効にする）。

```
# source setup.csh
```

これで実装作業の行える環境が整った。Make を実行する。

```
# make clean
# make
  _____[3]

All Done.
```

この make によって lib, NOVA, src, examples, murmur_client, bin サブディレクトリツリー内のすべての make が実行され、UNIDAQ の実装作業が完了した。

2.4 Installation to Personal Directory

ユーザ名 daquser が UNIDAQ を使用するには以下の作業を行っておく必要がある。まず、UNIDAQ の環境変数を有効にする。

```
> source /usr/local/UNIDAQ/v2.3/setup.csh
```

---

[3] この間 5 ~ 10 分。ここで murmur に対するエラーメッセッセージが 3 つほど出るが無視してよい。

ユーザにとって必要なUNIDAQのディレクトリ構造をホームディレクトリにつくる。

```
> user_tree
Please specify the destination directory for the user UNIDAQ tree
  [/usr/local/UNIDAQ/v2.3/user_UNIDAQ] /people/daquser/user_UNIDAQ
The user UNIDAQ tree user_UNIDAQ
        is now installed in directory /people/daquser
```

カレントをディレクトリuser_UNIDAQへ移す。

```
        > cd user_UNIDAQ
```

シェルスクリプトuser_install.cshを実行する。各設定値に対し、defaultでよい場合にはリターンキーを、そうでない場合は適切な値を入力する。

```
> ./user_install.csh

  Setting UNIDAQ user directory.
USERUNIDAQ = [/people/daquser/user_UNIDAQ]
Setting CERN library root.
CERN_ROOT = [/kek/cern/94a]

  If TCL/TK is installed on this machine, and you wish to use it
      enter the topdirectory of the TCL/TK installation here, e.g.
      /usr/local/tcltk Otherwise, just hit return
top TCL/TK directory [ ]

  Setting the location of the log book file.
"Locate log.bookYYMMDD files in [$USERUNIDAQ/log]? "


  Installing Dynamic offline analyzer main program
GLobal memory and Non-Dynamic analyzers will be built
```

作成されたsetup.cshを実行する。(ユーザ用の環境変数を有効にする。)

```
> source setup.csh
```

makeを実行する。

```
> make
  ——
All Done.
```

最後に、環境設定をlogin毎に有効にするため各自の.cshrc等の初期設定ファイルに

```
source $HOME/user_UNIDAQ/setup.csh
```

と記述しておく。これでdaquserは他のユーザのソース変更を気にすることなく、自分のUNIDAQとして使うことができる。

以上でUNIDAQを使用する環境が整った。次章では簡単な例題として、疑似データによるシステムの動作確認を行う。

Chapter 3

Testing the System by Pseudo Data

UNIDAQ には User's Guide の Figure 1 にあるように様々なプロセスがある。このなかでユーザにとって重要なプロセスが collector と analyzer であり、この間のデータの受け渡しをしているのが buffer manager(NOVA) である。Collector は CAMAC からの生データ（あるいは疑似データ）を buffer に取り込む、いわばデータの入口に相当する。一方、analyzer は buffer からデータを取り出しヒストグラミングなどの処理を実行した後、それを PAW に渡してグラフ描画を行わせるという、最終的なデータの出口に相当する。プロセスの性格を反映して、collector は C 言語で、analyzer は C 言語及び FORTRAN 言語で記述されている。

最初の例として CAMAC からデータを読みださず疑似データを生成する pseudo event を用いて、データ (event) の発生からグラフ表示までを行なう「自分の UNIDAQ」の作成 / 実行の手順を追ってみる。

3.1 NOVA Buffer Manager and Data Format

Buffer manager(NOVA) はデータを扱う様々なプロセス (collector, analyzer, etc...) の間に立ち、データ管理を行う重要なプロセスである。この NOVA が、どのようなフォーマットでデータの受け渡しを行っているのかを理解しておくことがまず先決である。レコードの形式は、Technical Guide の 8 章 Figure 7 に示されているように 6 words の header 部と 1 word 以上の event data/record related data 部からなっている[1]。

Header には buffer length、record type、run number、event number などが記録される。Trigger がかかったとき（event が発生したとき）と、pause、resume などのコマンドが発せられたときとでは 7 word 目以降の内容が異なる。前者の場合 event data がそのまま入るのに対し、後者の場合タイムスタンプおよびコメントが記録される。

このように、collector や analyzer など他のプロセスとのデータ入出力処理はすべて word を単位とした record format で行われる。

3.2 Collector Process

Collector はいくつかのモジュールによって構成されているが、ほとんどのユーザにとって当面理解しておけばよいのは pseudo_event.c というファイルである。実際に CAMAC を用いる場合は user_event.c とい

---

[1] この tutorial では 32bit マシンで話を進めているので 1 word といった場合 4bytes を指す。

うファイルがこれに対応し、ここで各実験の用途・測定系に合わせた記述をする。CAMAC ライブラリの call はこのファイルの中で行う。

まず、source file を実際にみてみることにしよう。以下の source は

$USERUNIDAQ/collector/pseudo_event_1.c

である。

```
/*************************************
 This file is modified for collector1
      (original: pseudo_event.c)
           on 6 Dec.1994
           by M.Tairadate.
**************************************/

  extern int b_flush_request;
  pseudo_init(max_lengthP)
  int *max_lengthP;                  /* max length in 4 byte words for event */
  {
      *max_lengthP=10000;
      return 0;
  }

  pseudo_event(max_length, event_length, buf)
  int *max_length;
  int *event_length;
  int *buf;
  {
      int gauss();

      *buf=gauss();                       /* 乱数を buffer に格納 */
      sleep(1);                           /* ウエイト */
      *event_length=1;                    /* イベント長 （word 単位） */
      b_flush_request=1;                  /* バッファのフラッシュ */
      return;
  }

  int gauss()                           /* 正規乱数発生の関数 */
  {
      int j, z;
      float x=0.0;
      for(j=1; j<=12; j++){
          x+=(float)rand()/2147483648.0;
      }
```

```
    z=10.0*(x-6.0)+50.0;                    /* 平均50、分散10の正規乱数 */
    return(z);
}

pseudo_begin()
{return;
}
pseudo_pause()
{return;
}
pseudo_resume()
{return;
}
pseudo_end()
{return;
}
```

pseudo_event.c

中心極限定理によって平均50、分散10の正規乱数を発生させ、それをbufferに格納する、という作業を1秒間隔で行っている。ここでは1次元のデータしか扱っていないのでevent_lengthは1としている。この作業はuser_level commandであるpause、end、exitのいずれかが発せられるまで続けられる。Bufferに格納されたデータはすべてbuffer managerによって管理され、他のプロセス（例えばanalyzer）に渡される。

ソースの内容を理解したところで、これをmakeにかける。

```
> make collector1 [2]
Compiling pseudo_event_1.c ...
Building /people/daquser/user_UNIDAQ/bin/collector1
```

これでcollectorの実行モジュールcollector1が作成された。

3.3 Analyzer Process

AnalyzerにはSUN、SGI、HPバージョンとDECstationバージョンがある。各プラットフォームの特徴に合わせて、前者は”dynamic link”によって処理をPAWに渡すのに対し、後者は”shared memory”にストアされたヒストグラムをPAWがアクセスするという手法をとっている。このtutorialは”DECstarionのための”という題名だが、実は機種依存性の強い部分はここだけで、他の部分は他の機種にもほとんどそのまま通用する。ここではDECstationバージョンについて説明する。（Technical Guide 11.2に基づく。）

作業は$USERUNIDAQ/analyzer/analyzer.glディレクトリにて行う。(glはglobal shared memoryの意。)Analyzerもいくつかのモジュールにより構成されているが、ユーザが適当なファイルを（例えばanal_pseudo.fという名前で）作成しMakefileに登録する。このユーザ作成ファイル(FORTRAN file)において、各自がHBOOKのヒストグラムライブラリをcallする。以下のソースは

---

[2] 通常は変更を加えたソースのみをコンパイルする。インストール直後及び強制的に全ソースをリコンパイルしたい場合はmake cleanを一度実行した後makeする。なお、ターゲットモジュールを指定しない場合はオリジナルのcollectorが作成される。

$USERUNIDAQ/analyzer/analyzer.gl/anal_pseudo.f

である。

```
C***************************************
C This file is modified for anal_pseudo
C        (original: anal_tmc.f)
C            on 6 Dec.1994
C            by M.Tairadate.
C***************************************

   subroutine ana_init()                          ! DATAのbooking
   common /pawc/hmemory(10000)                    ! 作業領域の確保

   call hlimap(10000,'GAUSS')                     ! Shared mamoryの大きさ及び名前の宣言
   call hbook1(10,'GAUSS distribution',25,0.,100.,0.) ! 格納場所の確保

   return
   end

   subroutine ana_exec(idata,iheader)             ! DATAのfilling
   integer idata(*)                               ! DATA
   integer iheader(*)                             ! ヘッダ
   common /pawc/hmemory(10000)                    ! 作業領域の確保

   call hf1(10,float(idata(1)),1.)                ! DATAの格納

   return
   end
```

anal_pseudo.f

最初のsubroutine ana_init()ではbooking（データ格納場所の確保）を行っている。hlimapではopenするshared memoryの大きさを10000（通常はPAWの作業領域の値に合わせる）、名前を'GAUSS'と宣言している。Shared memory内のデータに対しては複数のプロセスがアクセスすることができるので、同一データについて全く異なった処理を同時に実行できる[3]。hbook1では、x軸のレンジを[0,100]、分割を25、ヒストグラムIDが10のものを表示する、としている。一方、次のsubroutine ana_exec()ではfilling（データの格納）を行っている。AnalyzerはNOVA bufferにデータが供給される度にana_exec()をcallしidataにデータを入れる。この際、FORTRANの配列の先頭は1番地であるがCのそれは0番地であるので、idata(n)はcollectorのbuf[n-1]に相当することに注意。また、hf1でデータ(idata)をヒストグラムID 10に割り当てているが、hf1で受け入れることのできるデータ型はrealまたはreal*4であることに注意する必要がある。

---

[3] たとえば、analyzerを2つ作っておきこれを同時に走らせることも可能である（ただし異なった名前でなければならない）。

Analyzer ではmake を実行する際にMakefile を変更する必要がある。Makefile の例を以下に示す[4]。ユーザ作成ファイルの名前を変更しない場合（例えばanal_pseudo.f だけを使う場合）は手を加えることはないが、名前を変更したらいまanal_pseudo となっているところ（下のソース内で ^^^^^^^^^^^ をつけた部分）を全てその名前に変える。また、ユーザ作成ファイルは複数あっても問題はなく、その場合はそれぞれについて同様の記述をMakefile の中にしておく。

```
> cd $USERUNIDAQ/analyzer/analyzer.gl
> make clean
> emacs Makefile
```

```
#******************************************************
# This file is modified for anal_pseudo and anal_camac
#                     on 6 Dec.1994
#                     by M.Tairadate.
#******************************************************

MAINDIR  = $(TOPUNIDAQ)
USERDIR  = $(USERUNIDAQ)
BINDIR   = $(USERDIR)/bin
MDIR     = ./memmap/obj

TLIB     =  -L$(MAINDIR)/lib -ltmplt
PLIB     = -L$(CERN_ROOT)/lib -lpawlib -lpacklib -lkernlib -lgenlib \
                              -lgraflib -lgrafX11 -lpacklib -lkernlib -lX11
MLIB     = ./dummies/libdummy.a
MOBJ     =  $(MDIR)/hcreatei.o $(MDIR)/hcreatem.o
CLIB     = -L$(CAMDIR)/lib
NLIB     = -L$(MAINDIR)/NOVA/lib -lnova -L$(MAINDIR)/NOVA/netlib -lnetnova

CFLAGS   = -I$(MAINDIR)/include  -I$(MAINDIR)/NOVA/include

DFLAGS   = -D$(MACHINE)

CCN      = $(CC) $(CFLAGS) $(DFLAGS)

all             :
                @make all.$(MACHINE)

all.DEC         : sublib2 ana_gl
                @echo "  ana_gl (global memory analyzer) Finished"

.c.o    :
```

---

[4] ここでは例題に関係ない部分は省略している。

```
        $(CCN) -c $(CFLAGS) $(DFLAGS) $<

.f.o    :
        $(FF) -c $<

sublib  :
        cd memmap/src; make

sublib2 :
        cd dummies; make

anal_pseudo : $(BINDIR)/anal_pseudo
^^^^^^^^^^^             ^^^^^^^^^^^
anal_camac  : $(BINDIR)/anal_camac

$(BINDIR)/anal_pseudo : analyzer.o anal_pseudo.o
          ^^^^^^^^^^^              ^^^^^^^^^^^
                $(FF) analyzer.o anal_pseudo.o -o $(BINDIR)/anal_pseudo \
                                 ^^^^^^^^^^^                ^^^^^^^^^^^
                $(MLIB) $(TLIB) $(PLIB) $(NLIB)

$(BINDIR)/anal_camac : analyzer.o anal_camac.o
                $(FF) analyzer.o anal_camac.o -o $(BINDIR)/anal_camac \
                $(MLIB) $(TLIB) $(PLIB) $(NLIB)

clean           :
                -rm -f $(BINDIR)/anal_pseudo
                                 ^^^^^^^^^^^
                -rm -f $(BINDIR)/anal_camac
                -rm -f *.o *~ core
                cd dummies; make clean

manpage         :
                $(TOPUNIDAQ)/bin/ManMaker analyzer.c 2a ../man/man2/
                rm ../man/man2/junk
```

Makefile for analyzer

Make を実行する。

> make[5]
.....
> make anal_pseudo

---

[5] この make は make clean が行われたときにはライブラリ作成のため一度行っておく。

```
f77 -c anal_pseudo.f
f77 analyzer.o anal_pseudo.o -o /people/daquser/user_UNIDAQ/bin/anal_pseudo
./dummies/libdummy.a -L/usr/local/UNIDAQ/v2.3/lib -ltmplt -L/kek/cern/94a/lib
-lpawlib -lpacklib -lkernlib -lgenlib -lgraflib -lgrafX11 -lpacklib -lkernlib
-lX11 -L/usr/local/UNIDAQ/v2.3/NOVA/lib -lnova
-L/usr/local/UNIDAQ/v2.3/NOVA/netlib -lnetnova
```

これで analyzer の実行モジュール anal_pseudo が作成された。

以上で collector 及び analyzer の各プロセスを実行する準備が整った。Collector をより積極的に modify したいという人は、Technical Guide 10 を参照されたい。

3.4 Making PAW Macro File

Collector、analyzer 実行の前に PAW のマクロファイル ∗.kumac を作っておくと便利である。anal_pseudo 用のマクロ$USERUNIDAQ/bin/pseudo.kumac を示す。

```
**************************************
* This file is edited for anal_pseudo
*          on 6 Dec.1994
*          by M.Tairadate.
**************************************

  macro pseudo                              | マクロ名の宣言
  global GAUSS                              | Shared memory name の宣言
  graphics/option statistic                 | 統計の表示
  histgram/plot 10 k                        | Histgram ID=10 のプロット
  loop:
        histgram/plot 10 u                  | アップデート
        wait ' ' 1                          | ウエイト
        goto loop
  return
```

pseudo.kumac

pseudo_event_1.c の記述に合わせてヒストグラムを 1 秒毎にアップデートさせている。コマンドの詳細については PAW reference mannual を参照のこと。

3.5 Running Collector and Analyzer

ここでの作業は User's Guide 5.3 に基づいている。まず、カレントを bin directory に移す。

```
> cd $USERUNIDAQ/bin
```

NOVA デーモンを実行する。

```
> novad /tmp/nova
```

次に collector 及び analyzer を走らせる。

```
> collector1 &
> anal_pseudo &
```

これらのプロセスが動いているか確認するために次のコマンドが用意されている。

```
> status
```
[6]

```
   process_name           node             pid       msgqid      status
 ----------------  ------------------  ---------  ----------  ----------
     collector1                           997          2         ALIVE
    anal_pseudo                          1006          3         ALIVE
```

Collector が CAMAC からの実際のデータを読み出すか疑似データを生成するかの選択はつぎのコマンドで行なう。実際のデータを読み出す場合、

```
> ask collector1 set eventsource=CAMAC
```

疑似データを生成する場合、

```
> ask collector1 set eventsource=PSEUDO
```

既定値は CAMAC である。Collector の data processing を次のコマンドで行う。

```
> ask collector1 begin
```

Begin コマンドを実行した段階では pause 状態になっているのでこれを resume する。

```
> ask collector1 resume
```

これで collector は疑似データの生成を開始する。PAW を走らせる。

```
> pawX11
```

マクロを実行する。

```
paw> exec pseudo
```

これで HIGZ window にヒストグラムが表示される。Figure 3.1にその出力を示す。

3.6 Stop UNIDAQ

まず、PAW を終了する。pseudo.kumac では無限ループに入っているので<CTRL-C>によって中断する。

```
paw> exit
```

Collector を終了する。

```
> ask collector1 end
> ask collector1 exit
```

Analyzer を終了する。

```
> ask anal_pseudo end
```

---

[6] 詳しくは User's Guide 7.2 参照。

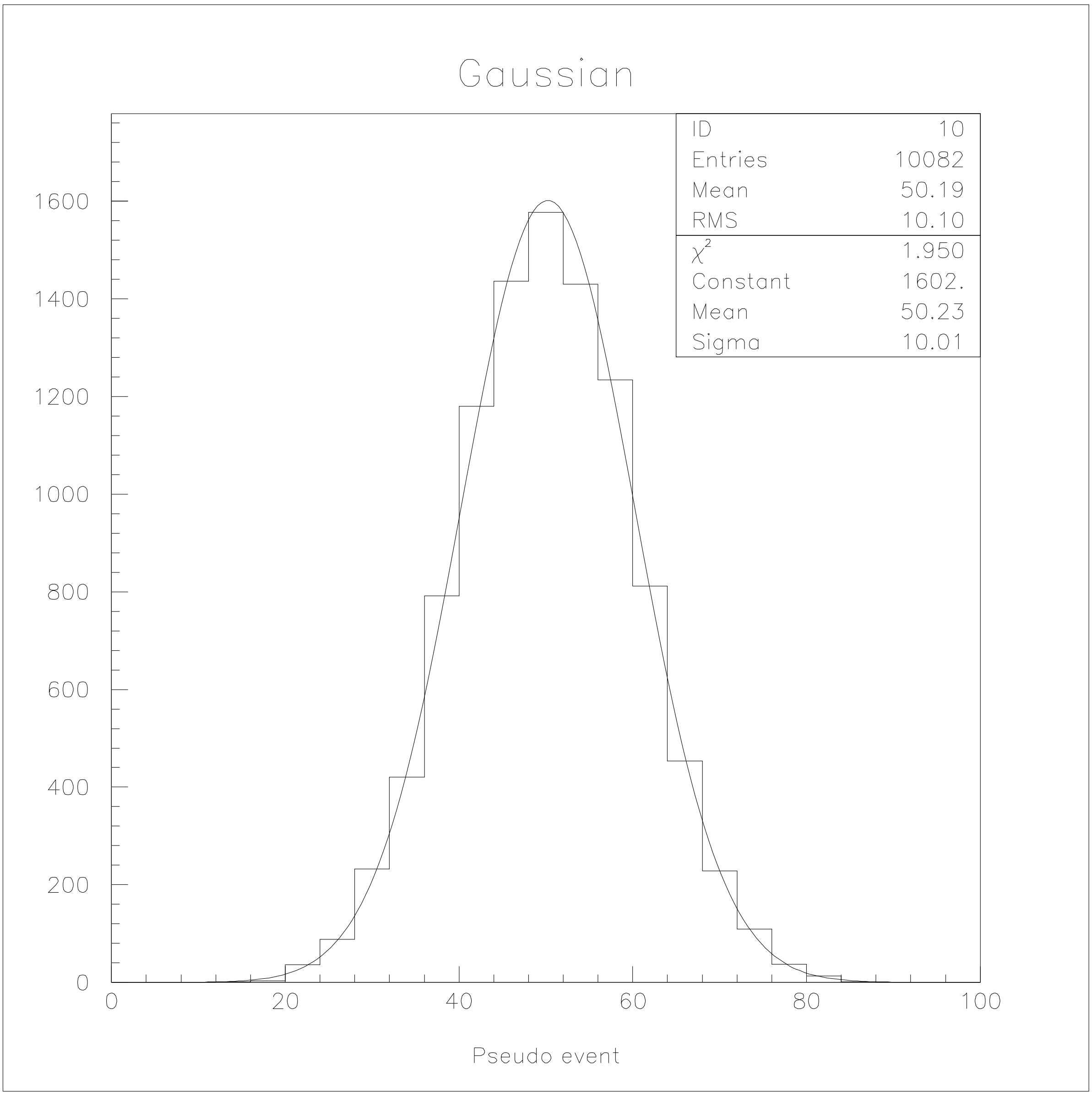

Figure 3.1: PSEUDO event(generated random number)

```
> ask anal_pseudo exit
```

Reset コマンド[7] で UNIDAQ のプロセスを kill する。

```
> Reset_local
Are you sure you want to kill ALL local unidaq processes?
Type YES to continue >> yes
try nice reset
kill whichever process is left
Remove the shared memory
All done
```

これで UNIDAQ に関するすべてのプロセスが kill され、shared memory が解除されて終了した。

以上、UNIDAQ の基本的な操作法について説明した。上記の作業を支障なく行うことができれば、UNIDAQ が一応正常にインストールされていると考えてよい。次章からは X-window 環境を積極的に利用した CAMAC data processing について述べる。

---

[7] collector や analyzer を個別に終了しなくても Reset によって自動的に全てのプロセスは kill される。使用に当たっては注意すること。

Chapter 4

CAMAC Data Processing

この節ではCAMACからデータを取り出し、それをリアルタイムで解析・グラフ表示する方法について述べる。CAMACの場合もユーザが行うべき作業はpseudo eventのときと基本的に変わらない。異なる部分は、collectorの中でCAMAC libraryをcallするという点である。また、data strageに関連するrecorderプロセスとUNIdumpプロセスの説明も行う。

なお、繰り返し言うが、CAMACデバイスドライバ及びライブラリが第2章で示したように正しくインストールされていることを確認する必要がある。バージョンの違い、インストール手順の違いはシステムのクラッシュを引き起こす要因となる。

4.1 Hardware Configuration

Camac eventの例題としてscintillation counterを用いて宇宙線の速度を求める実験を行う。Figure 4.1にそのsetupを示す。最初にPMT1~3を隣接させて設置する。全てのPMTに同時に信号が来たときTDCをスタートさせるとともに、割り込み要求を出す(LAM request)。PMT3にdelayを設けこの信号のタイミングでTDCをストップさせる。このときTDC0~3にそれぞれPMT1andPMT2、PMT1、PMT2、PMT3の信号を入れておく。Dataの読みだしが終わり次第latchを解除し、次のeventに備える。同じロジックを用いて今度はPMT3を160cm程離して置く。こうしてTDC3の値の分布をPMT3の移動の前後で比較し、どのくらいの時間変化があるかをみる。

4.2 Collector Process

CAMAC eventを扱うときに必要となるファイルuser_event.cはいくつかの関数から成っている。関数user_initではeventの最大長max_lengthをword単位で指定する。その他の関数はコマンドを発行することによりcallされる。

まずbeginコマンドによりuser_beginが呼ばれ、CAMACのオープン及び初期化、LAMのdisableを行う。次にresumeコマンドによりuser_resumeが呼ばれ、interrupt registerの初期化とCAMACコントローラのLAM enableを行う。

次に処理はuser_eventに移る。最初イベント長を0にしておく。一旦TDC及びinterrupt registerをclearし、ついでinterrupt registerをLAM enableする。同時にoutput registerのCH1に書き込みを行う。これによってNIM gate generatorのstopに最初の信号が入り、latchが解除されtriggerの待機状態

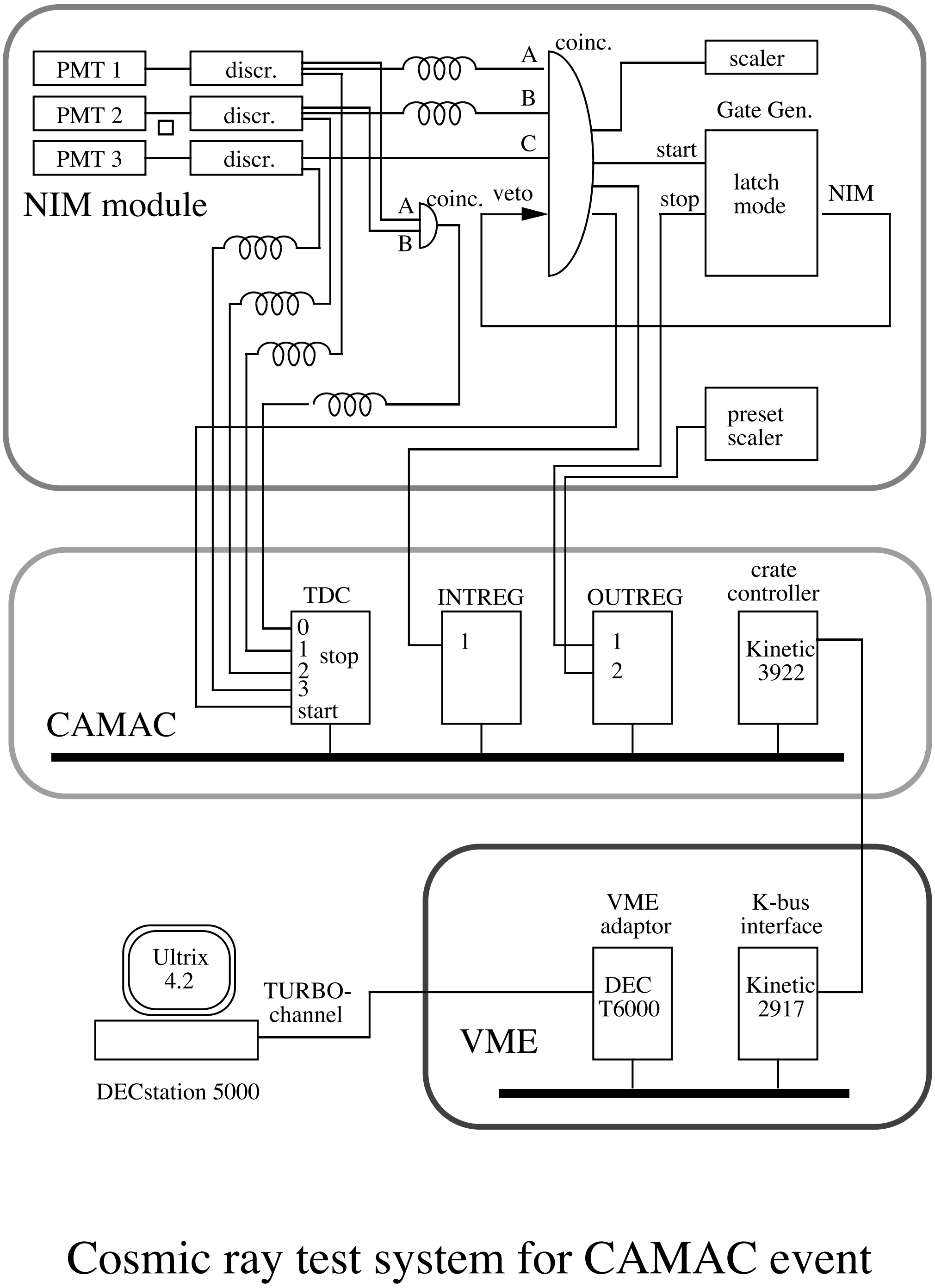

Figure 4.1: Setup for CAMAC event

になる。ここで timeout を設定しておく。そして Interrupt register に LAM がかかった[1] とき、TDC0 ~ 3 の data を buffer[0] ~ buffer[3] に格納する。4 チャネル分のデータを扱うのでイベント長 4 を指定する。

また pause が発せられると LAM が disable され、end が発せられると camac がクローズされる。pause は resume によって、end は begin によってそれぞれ実行を再開できる。以下のソースは

$USERUNIDAQ/collector/user_event_1.c

である。

```
/*************************************
 This file is modified for collector1
      (original: user_event.c)
           on 6 Dec.1994
           by M.Tairadate.
*************************************/

/*
 * this is the code that a user would typically modify
 *   for their own purposes.
 *
 * The following routines must be provided.
 *
 * user_event()  : executed for each event
 *     arguments :
 *         int *buffer;        pointer to buffer area
 *         int *length;        available buffer length
 *         int *event_length;  user_event must set the event length
 *         int *event_ID;      user_event must set the "event ID" word
 *         int *max_length;    max length (words) an event buffer may occupy
 *         int *mode;          mode of event taking, for those using it
 *     return    :
 *         ERANGE              buffer length is not enough for the event.
 *         ETIMEDOUT           timed out on waiting an event
 *         0                   normal return
 *
 * user_init()   : executed once at the beginning
 *     arguments :
 *         int *max_length;    maximum event length in words.
 * user_begin()  : executed on "begin" command
 *     arguments :
 *         int *runtype;       runtype variable, for those using it.
 *         int *max_length;    maximum event length in words.
```

---

[1] Interrupt register には NIM からの LAM 信号以外の何らかのシグナルが混入することがある。これを除去するため、interrupt register の data が 0 のときは fake data とみなし TDC を読みに行かないようにしている。

```
 * user_resume() : executed on "resume" command
 *     arguments :
 *         int *mode;          mode of event taking, for those using it
 * user_pause()  : executed on "pause" command
 *     arguments :
 *         int *mode;          mode of event taking, for those using it
 * user_end()    : executed on "end" command
 *     arguments :
 *         int *runtype;       runtype variable, for those using it.
 * user_exit()   : executed once at the end
 *
 */

#include "template.h"
extern int debug_level;

/*Local headers */
#include "camlib.h"

/* define CAMAC station Number */
#define  TDC     20                   /* TDC */
#define  INTREG   5                   /* Interrupt Register */
#define  OUTREG  15                   /* Output Register */

/* Macros */
#define  ENABLE_INTREG (1<<(INTREG-1)) /* Mask value for LAM */
#define  TIMEOUT 60                   /* Timeout(sec.) for Wait LAM */
#define  MAX_EVENT_SIZE 100           /* Maximum event size */

/* Handles for event source */
extern char eventsource[20];
int eventSrcType;
#define PSEUDO 1
#define OTHER  0

/* Functions */
error_exit(status)
int status; {
    perror(strerror(status));
    exit(status);
}

user_init( max_length ) int *max_length; {
  if (strcmp(eventsource,"PSEUDO") == 0) {
```

```
    eventSrcType = PSEUDO;
    *max_length = MAX_EVENT_SIZE;
  }
  else  {
    *max_length = MAX_EVENT_SIZE;
    eventSrcType = OTHER;
  }
  return;
}

user_event(buffer,length,event_length,event_ID,max_length,mode)
     int *buffer;       /* pointer to buffer                           */
     int *length;       /* buffer length                               */
     int *event_length; /* user_event must set the event length        */
     int *event_ID;     /* user_event must set the event ID word       */
     int *max_length;   /* max words an event should occupy            */
     int *mode;         /* mode of event taking, for those who use it  */
{
   int status, a, i, q, x;
   short out=3;                                       /* For OUTREG */

  *event_length=0;

  if (eventSrcType == PSEUDO)  {                      /* PSEUDO event */
    pseudo_event ( max_length, event_length, buffer );
    *event_ID = EVENT_RECORD;
    return 0;
  }
  else   {                                            /* OTHER event */
    while(*event_length == 0){
        status = CAMAC(NAF(TDC,   0, 9), buffer, &q, &x ); /* Clear TDC */
        status = CAMAC(NAF(INTREG,0, 9),  &i, &q, &x ); /* Clear LAM request */
        /* Enable Interrupt */
        status = CAMAC(NAF(INTREG,0,26),  &i, &q, &x );
        /* Trigger Start */
        status = CAMAC(NAF(OUTREG,0,17), &out, &q, &x ); /* Write to OUTREG */
        /* Wait Interrupt */
        status = CAM_WaitLAM(TIMEOUT);                /* Timeout */
        if( status != 0 ){
            printf("Timeout!   status = %d\n", status);
        }
        /* Read Interrupt register */
        status = CAMAC(NAF(INTREG,0,0), &i, &q, &x );
        /* Read Data from TDC*/
```

```
        if (i != 0) {
            for(a=0; a<=3; a++){
                status = CAMAC(NAF(TDC,a,0), &(buffer[a]), &q, &x );
                if( status ) error_exit(status);
            }
            printf(" buf= %d  %d  %d  %d\n",
                     buffer[0], buffer[1], buffer[2], buffer[3]);
            /* UNIdump, dataview があれば、あえて表示しなくともよい。*/

            *event_length=4;                     /* Number of words */
        }
    }

  /*
   * flush_buffer("","") force to forward the nova buffer to the
   * processes on the down-stream.
   * event_length and event_ID must be ready on calling "flush_buffer()"
   */
    if (*mode == 0)  {
      *event_ID = EVENT_RECORD;
      *event_length=10;
    }
    else  {
      *event_ID = ENVIRONMENT_RECORD;
      *event_length=100;
      flush_buffer("","");
    }

  /*
   * end of one event
   */

    return 0;
  }                                /*  End of OTHER event source switch  */
}

user_begin(run_type,max_length) int *run_type,*max_length; {
  int status;
  if (strcmp(eventsource,"PSEUDO") == 0)  eventSrcType = PSEUDO;
  else                                    eventSrcType = OTHER;

  if (eventSrcType == PSEUDO)  {
    *max_length = 150;
    pseudo_begin();
```

```
    return 0;
  }
  else {
   /*
    * Open and Initialize CAMAC
    */
    status = CAM_Open();
    if( status ) error_exit(status);
    printf("CAM_Open : OK\n");
    status = CSETBR(0);
    if( status ) error_exit(status);
    printf("CSETBR   : OK\n");
    status = CSETCR(0);
    if( status ) error_exit(status);
    printf("CSETCR   : OK\n");
    status = CGENZ();
    if( status ) error_exit(status);
    printf("CGENZ    : OK\n");
    status = CGENC();
    if( status ) error_exit(status);
    printf("CGENC    : OK\n");
    status = CSETI();
    if( status ) error_exit(status);
    printf("CSETI    : OK\n");
    status = CREMI();
    if( status ) error_exit(status);
    printf("CREMI    : OK\n");
    status = CAM_DisableLAM();
    if( status ) error_exit(status);
    printf("CAM_DisableLAM : OK\n");

    return;
  }
}

user_pause( mode ) int *mode; {
  int status;

  if (eventSrcType == PSEUDO)  {
    pseudo_pause();
    return 0;
  }
  else {
    status = CAM_DisableLAM();
```

```
    if( status ) error_exit(status);
    printf("CAM_DisableLAM : OK\n");

    return;
  }
}

user_resume( mode ) int *mode; {
  int status, i, q, x;
  short out=3;                        /* For OUTREG */

  if (eventSrcType == PSEUDO)  {
    pseudo_resume();
    return 0;
  }
  else {

   /*
    * Initialize CAMAC
    */
    status = CSETBR(0);
    if( status ) error_exit(status);
    printf("CSETBR   : OK\n");
    status = CSETCR(0);
    if( status ) error_exit(status);
    printf("CSETCR   : OK\n");
    status = CGENZ();
    if( status ) error_exit(status);
    printf("CGENZ    : OK\n");
    status = CGENC();
    if( status ) error_exit(status);
    printf("CGENC    : OK\n");
    status = CREMI();
    if( status ) error_exit(status);
    printf("CREMI    : OK\n");

    /* Initialize Interrupt Register */
    i = 0;
    status = CAMAC(NAF(INTREG,0, 9), &i, &q, &x );   /* Clear LAM */
    status = CAMAC(NAF(INTREG,0,24), &i, &q, &x );   /* Disable LAM */
    status = CAMAC(NAF(INTREG,0,11), &i, &q, &x );   /* Clear LAM Mask */

    /* Enable LAM */
    status = CAM_EnableLAM(ENABLE_INTREG);
```

```
    if( status ) error_exit(status);
    printf("CAM_EnableLAM  : OK\n");

    return;
  }
}

user_end( run_type ) int *run_type; {
  int status;

  if (eventSrcType == PSEUDO)  {
    pseudo_end();
    return 0;
  }
  else {
    /*
     * Close CAMAC
     */
    status = CAM_DisableLAM();
    if( status ) error_exit(status);
    printf("CAM_DisableLAM : OK\n");
    status = CAM_Close();
    if( status ) error_exit(status);
    printf("CAM_Close: OK\n");

    return;
  }
}

user_exit() {
   return;
}
```

user_event_1.c

前章で述べた pseudo_event.c は user_event.c の中から呼ばれていることがわかる。なお、user_event.c 中の各関数は collector.c の中から呼ばれている。

4.3 Analyzer Process

Analyzer に関しては pseudo event のときとほとんど同じ記述である。異なる点は shared memory name を'TDC' としたこと、4 つのグラフを表示するようにしたことである。Subroutine ana_init() は analyzer を実行したとき最初に 1 回だけ call されるのに対し、subroutine ana_exec() は event が発生する度に毎回 call される。以下のソースは

$USERUNIDAQ/analyzer/analyzer.gl/anal_camac.f

である。

```
C***************************************
C This file is modified for anal_camac
C       (original: anal_tmc.f)
C           on 6 Dec.1994
C           by M.Tairadate.
C***************************************

      subroutine ana_init()
      common /pawc/hmemory(10000)

      call hlimap(10000,'TDC')                   ! Memory の大きさ及び名前の宣言
      call htitle('TDC output(cosmic ray)')      ! Global title の設定
      call hbook1(10,'TDC-0(PMT A and B)$',50,200.,800.,0.)
      call hbook1(20,'TDC-1(PMT A)$',50,200.,800.,0.)
      call hbook1(30,'TDC-2(PMT B)$',50,200.,800.,0.)
      call hbook1(40,'TDC-3(PMT C)$',50,1000.,1600.,0.)

      return
      end

      subroutine ana_exec(idata,iheader)
      integer idata(*)
      integer iheader(*)

      print *,'idata= ',(idata(i),i=1,4)         ! DATA の表示
      call hf1(10,float(idata(1)),1.)
      call hf1(20,float(idata(2)),1.)
      call hf1(30,float(idata(3)),1.)
      call hf1(40,float(idata(4)),1.)

      return
      end
```

anal_camac.f

4.4 Recorder Process and UNIdump

Pseudo event の説明では、collector から受け取ったデータを即解析してグラフ化する方法について述べたが、実際には生データをファイルに残しておくのが一般的であろう。そのためのプロセスが recorder である。Collector や analyzer と違い、recorder にはユーザが直接ソースに手を加えるべき部分はない。設定値は全て command で入力可能である。例えば

```
> ask recorder set dest_dir = $USERUNIDAQ/dat
```

とするとデータファイルの出力先がディレクトリ$USERUNIDAQ/dat に指定される。データは binary 形式で保管されるので、ユーザがこれを参照するときは od などの UNIX コマンドによって表示させる。また、UNIdump というプロセスによって NOVA buffer の様子を確認できる。UNIdump はヘッダ情報とともに event data を各 word 別に 4 byte 整数、2 byte 整数、ASCII、16 進表示させるものである。例えば

```
> ask UNIdump on 0 3
```

とすると 0~3word の event buffer の内容が表示される。

なお、recorder 及び UNIdump のソースはそれぞれ

$USERUNIDAQ/recorder、$TOPUNIDAQ/src/etc/tools

の各ディレクトリにある。

4.5 Executing Script File

Pseudo event のときはコマンドラインから逐一コマンドを入力していたが、実行させたいプロセスの数が多くなってくるとこれでは大変である。そこで、ここでは shell script によって NOVA、collector、analyzer、recorder、UNIdump の各プロセスを連続（並列）実行させることにする。以下のソースは

$TOPUNIDAQ/examples/example_script_a

をもとにして bin ディレクトリに作った

$USERUNIDAQ/bin/example_script_camac

である。

```
#******************************************************
# This file is modified for collector1 and anal_camac
#   (original: $TOPUNIDAQ/examples/example_script_a)
#                     on 6 Dec.1994
#                     by M.Tairadate.
#******************************************************
  echo
  echo ------------------------------------------------------------------
  echo Unidaq Example Run with NOVA, collector, analyzer, recorder,
  echo        dump and PAW
  echo ------------------------------------------------------------------
  echo
  echo ------------------------------------------------------------------
  echo Starting NOVA
  echo ------------------------------------------------------------------
```

```
echo
$TOPUNIDAQ/NOVA/src/novad /tmp/nova
echo
echo -------------------------------------------------------------------
echo Starting collector
echo -------------------------------------------------------------------
echo
xterm -title "Unidaq COLLECTOR" -fg red -bd red -sb -geometry 62x15+610 \
-e collector1 &
sleep 1
echo
echo -------------------------------------------------------------------
echo Starting analyzer
echo -------------------------------------------------------------------
echo
xterm -title "Unidaq ANALYZER" -fg purple -bd purple -sb -geometry 62x15\
+610+240 -e anal_camac &
sleep 1
echo
echo -------------------------------------------------------------------
echo Starting recorder
echo -------------------------------------------------------------------
echo
xterm -title "Unidaq RECORDER" -fg orange -bd orange -sb -geometry 62x15\
+610+480 -e recorder &
sleep 1
echo
echo -------------------------------------------------------------------
echo Starting dump
echo -------------------------------------------------------------------
echo
xterm -title "Unidaq DUMP" -fg blue -bd blue -sb -geometry 65x19+610+720 \
-e UNIdump &
sleep 1
echo
echo -------------------------------------------------------------------
echo Executing PAW
echo -------------------------------------------------------------------
echo
xterm -title "PAW session" -sb -geometry 65x19+300+730 -e pawX11 &
echo
echo -------------------------------------------------------------------
echo ask UNIdump on 0 15  (Display header and through the 15th word)
echo -------------------------------------------------------------------
```

```
echo
ask UNIdump on 0 15
echo
echo -------------------------------------------------------------------
echo set collector eventsource = CAMAC
echo set recorder dest_dir = $USERUNIDAQ/dat
echo -------------------------------------------------------------------
echo
ask collector set eventsource = CAMAC
ask recorder set dest_dir = $USERUNIDAQ/dat
echo
echo -------------------------------------------------------------------
echo Status  (Display current status of UNIDAQ processes)
echo -------------------------------------------------------------------
echo
status
sleep 1
echo
echo -------------------------------------------------------------------
echo ask collector1
echo -------------------------------------------------------------------
echo
ask collector1
echo
echo -------------------------------------------------------------------
echo END
echo -------------------------------------------------------------------
echo
ask anal_camac end
sleep 1
ask recorder end
sleep 1
ask UNIdump off
echo
echo -------------------------------------------------------------------
echo EXIT
echo -------------------------------------------------------------------
echo
ask anal_camac exit
sleep 1
ask recorder exit
sleep 1
ask UNIdump exit
sleep 1
```

```
echo
echo -------------------------------------------------------------------
echo Reset_local
echo -------------------------------------------------------------------
echo
Reset_local yes
echo
echo -------------------------------------------------------------------
echo Finished the script......
echo -------------------------------------------------------------------
echo
```

example_script_camac

各プロセスを独立した window 上に走らせることにより、それぞれのプロセスの状態がリアルタイムでモニタできる。以下、script の実行手順を述べる。

まず、データファイル用のディレクトリを作る。

```
> mkdir $USERUNIDAQ/dat
```

カレントを bin directory に移す。

```
> cd $USERUNIDAQ/bin
```

Script を実行する。

```
> ./example_script_camac
```

すると 5 つの window が現れ、PAW の window で workstation type の入力待ちになる。適切な入力を行い HIGZ window を出しておく。次に最初の window で

```
collector1> begin 908
```

とすることにより run がスタートする。908 はユーザが決める run number で既定値は 0 である。そして

```
collector1> resume
```

によりデータの取り込みが開始される。3.5 節同様 PAW の window で

```
paw> exec camac
```

とすると HIGZ window にグラフが表示される。また、recorder によって$USERUNIDAQ/dat/r0000908.dat に event buffer の内容が出力される。Collector プロンプトの状態では user_level command である begin、resume、pause、end、exit の実行が可能である[2]。pause の解除は resume で行い、run のストップは end による。終了は exit によって行う（先に PAW を終了させておく）。

```
collector1> end
collector1> exit
```

引き続き collector 以外の他のプロセスも順次 end、exit され最後に reset_local によって NOVA が kill される。以下にグラフ描画用 PAW マクロファイル

$USERUNIDAQ/bin/camac.kumac

を示す。

---

[2] もちろん set <variable> = <value> の形式で user_level 変数（例えば eventsource）の設定も可能である。

```
*************************************
* This file is edited for anal_camac
*           on 6 Dec.1994
*           by M.Tairadate.
*************************************

  macro camac
  global TDC
  zone 2 2                                    | グラフ配置の指定
  graphics/optiton statistics
  histogram/plot 10 k
  histogram/plot 20 k
  histogram/plot 30 k
  histogram/plot 30 k
  loop:
        histogram/plot 10 u
        histogram/plot 20 u
        histogram/plot 30 u
        histogram/plot 40 u
        wait ' ' 30                           | ウエイト (30 秒間隔)
        goto loop
  return
```

camac.kumac

4.6 Data Analysis with PAW

この節は UNIDAQ とは直接関係ないが、参考のため recorder によってつくられたデータファイル r0000908.dat を通常のテキストファイル camac.dat に変換したものとして、PAW によって解析する方法について簡単に述べることにする。

Alphanumeric file である camac.dat には TDC0-3 の出力が 4 元データとして記録されている。これをまず 4 次元 ntuple に読み込む。次に 4 つの 1 次元 histogram を生成しておきそこへ ntuple からデータを投影する。そしてデータを plot and fitting する。

```
> cd $USERUNIDAQ/dat
> cp ../bin/paw_camac.kumac .
> pawX11
paw> exec paw_camac
```

とすることでグラフが表示される。以下に camac event 解析マクロ

$USERUNIDAQ/bin/paw_camac.kumac

を示す。

```
**********************************************
* This file is edited for user_event_1(CAMAC)
*           on 6 Dec.1994
*           by M.Tairadate.
**********************************************
  ntuple/create 1 'camac' 4 ' ' 1000 ' ' ' ' ' ' ' ' | 4変数ntupleの生成
  ntuple/read 1 camac.dat                            | データの読み込み
  ntuple/print 1                                     | ID=1のsummary表示
  zone 2 2                                           | グラフ配置の指定
  graphics/set ygti 1.0                              | 上マージンの設定
  graphics/set gsiz 0.5                              | Global titleの大きさ
  graphics/option statistics                         | 統計の表示
  graphics/option date                               | 日時の表示
  title_global 'TDC output (cosmic ray)'             | Global title
  histogram/create/1dhisto 10 'TDC-0(PMT 1 and 2)' 30 200. 800.    | 1次元
  histogram/create/1dhisto 20 'TDC-1(PMT 1)' 30 200. 800.          | histogram
  histogram/create/1dhisto 30 'TDC-2(PMT 2)' 30 200. 800.          | のbooking
  histogram/create/1dhisto 40 'TDC-3(PMT 3, close)' 30 1000. 1600. |
  ntuple/project 10 1.1                              |
  ntuple/project 20 1.2                              | ntuple -> histogram への
  ntuple/project 30 1.3                              | データの投影
  ntuple/project 40 1.4                              |
  graphics/option fit                                | fittingに関する統計の表示
  histogram/plot 10                                  | データのplot
  atitle '35/ns' ' '                                 | 軸タイトルの定義
  histogram/plot 20
  atitle '35/ns' ' '
  histogram/plot 30
  atitle '35/ns' ' '
  histogram/fit 40 g                                 | ガウス分布によるfitting
  atitle '35/ns' ' '
```

paw_camac.kumac

次のマクロを実行するとヒストグラムがプリントアウトされる。

```
paw> shell cp ../analyzer/analyzer.gl/tlpr.kumac .
paw> exec tlpr
```

このPostScriptプリンタ出力マクロ

$USERUNIDAQ/analyzer/analyzer.gl/tlpr.kumac

を以下に示す。プリンタ名、ヒストグラムIDを適当な値に設定すること。

```
macro tlpr 1='ps'                                 | プリンタの指定
message 'printing to ' [1]
*
*  to use exec psprint
*  prints the current picture
*
  option zfl
  fortran/file 66 tmp.ps                          | 出力ファイルの指定
  graphics/metafile 66 -111                       | format の指定 (portrait-PostScript)
  picture/plot 10                                 |
  picture/plot 20                                 | 現在の図の描画
  picture/plot 30                                 |
  picture/plot 40                                 |
  fortran/close 66                                | 出力ファイルの close
  shell lpr -p[1] -r tmp.ps                       | プリンタへの出力とファイルの削除
return
```

tlpr.kumac

このようにして、PMT1/2 と PMT3 を隣接して置いた場合と離して置いた場合の 2 回の測定の結果がそれぞれ Figure 4.2および Figure 4.3である。TDC3 の出力を比較すると宇宙線の飛行時間が約 5.3nsec. であることがわかる。よって宇宙線速度はおよそ 3×10^8m/s となる。

以上、実際に CAMAC からデータを読みだし PAW で解析するまでを説明した。これで基本的な DAQ システムを組むことができるはずである。

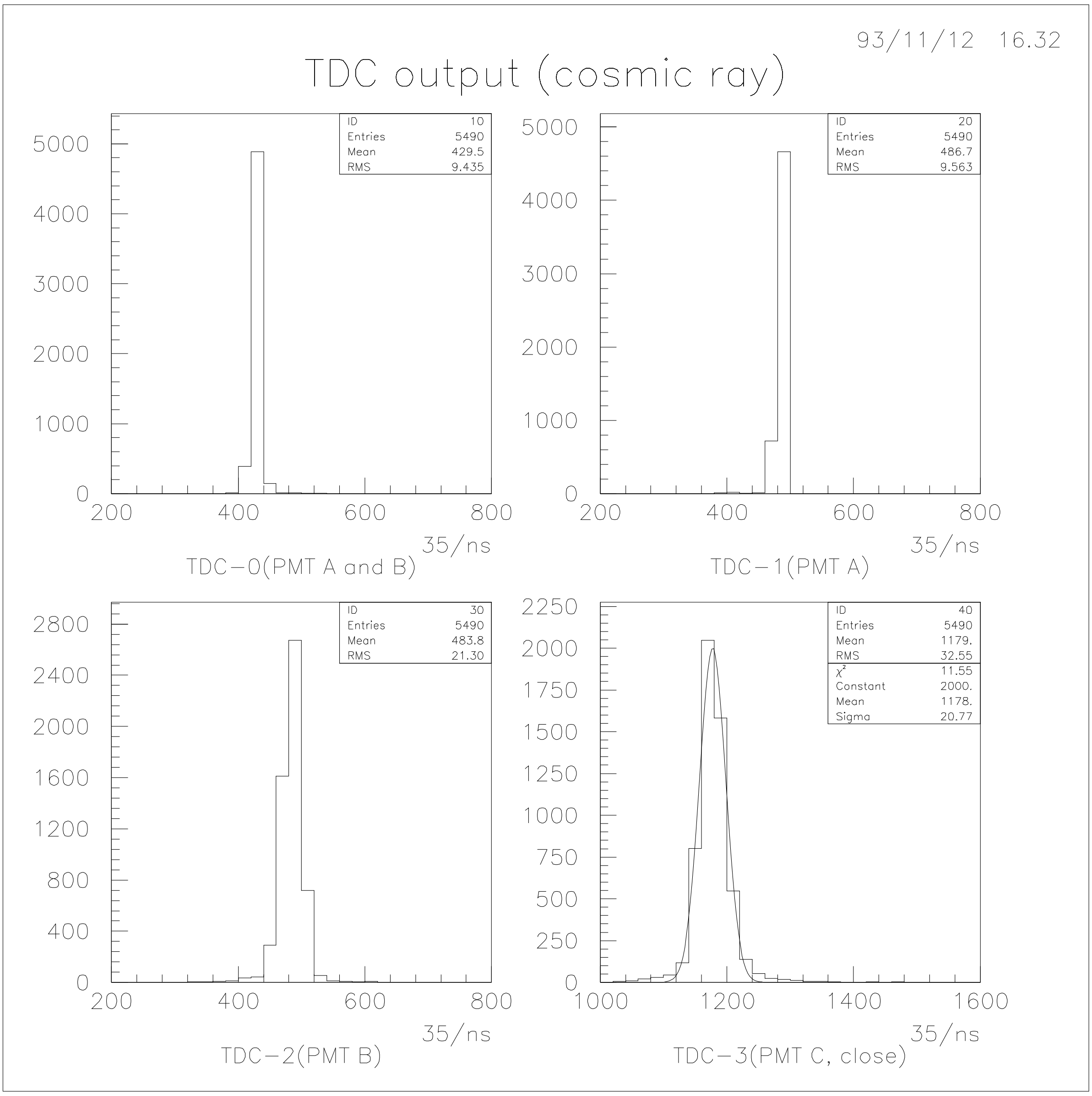

Figure 4.2: CAMAC event(set PMT3 close to PMT1/2)

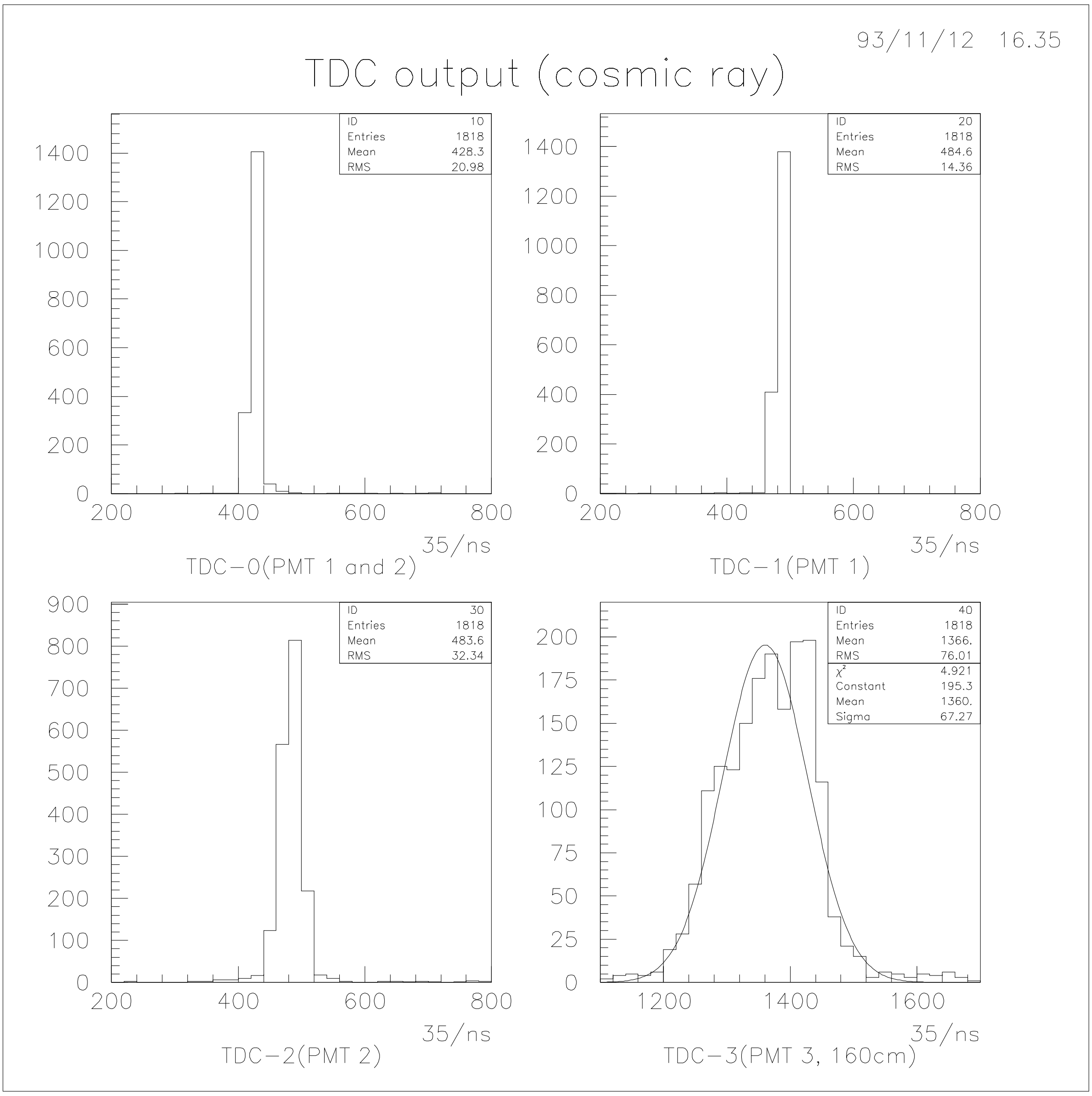

Figure 4.3: CAMAC event(set PMT3 apart from PMT1/2)

Chapter 5

Additional Tools

前章までに UNIDAQ の基本となる概念及びその操作法について述べた。本章ではいくつかの便利な tool について記述する。ただし、これらは必須となる事項ではないので、ここでは簡単な紹介にとどめ、詳しくは Documentation Set に譲ることにする。

5.1 Playback of A Data File

UNIDAQ は、buffer manager(NOVA) が管理する data buffer の内容を標準出力に渡したり (to_pipe)、逆に標準入力から NOVA に取り込む (from_pipe) 機能を備えている。これにより複数の WS で data buffer を共有したり、データを加工することができる。たとえば data buffer の内容を 8 進表示させたい場合、

```
to_pipe | od -h | more
```

とすればよい。これを各 word ごとに見やすくしたのが UNIdump である。また recorder によって出力されたデータファイル (r*.dat) は、to_pipe による出力と同じものである。よってこのデータファイルを利用し、イベントを再現することができる。たとえば

```
cat r0000908.dat | from_pipe
```

とすれば、collector の代わりに NOVA に対しデータを提供することになり、analyzer ・PAW にグラフの再描画をさせることができる。To_pipe、from_pipe については User's Guide 5.5、6.27、7.12 および 7.13 に記述がある。

5.2 Dataview Process

前章では、データの受け渡しがどのようになされているかを理解するために、collector、analyzer でそれぞれデータ (buf, idata) の画面表示をさせ、さらに UNIdump を用いてヘッダを含めて event buffer の内容を表示させた。これから述べる dataview というプロセスはデータの画面表示専用であり、独立した window を開きそこにユーザが指定したフォーマットでカレントデータを表示するものである。ユーザは次の 2 つのファイルを用途に合わせて変更する必要がある。

$USERUNIDAQ/dataview/datawin.h [1]

$USERUNIDAQ/dataview/dataconv.c

以下のソースは第 3 章で用いた pseudo event 用に修正したものである。修正箇所やその方法については、User's Guide 3.4、5.3 および Technical Guide 12 を参照されたい。

---

[1] ファイル名を変更した場合 datawin.c 中の include 部を変更する。

```
/************************************
 This file is modified for dataview1
       (original: datawin.h)
           on 6 Dec.1994
           by M.Tairadate.
************************************/

/*******************************************************
 FILE: datawin.h defines the variables to be displayed
 Author: C. Timmermans, University of Michigan
 *******************************************************/

struct dw_def {
  char txt[20];
  int (*conversion)();
};

/*****************************************************
  Please enter the action routines here. These routines descrbe
  the action taken when clicking a button. At this point
  they should be added as:
           extern int routine();
  These routines are to be coded in file dataconv.c, which
  contains a description of how to do that
 *****************************************************/
extern int gen_num();

struct dw_def dview[] = {
/*****************************************************
 For each variable to be displayed two fields needs to be filled:
 char txt[20];         The text to be displayed in front of the value
  int (*conversion)(); A routine converting the current data to a
                       string. This routine needs to be added here as
                       (int(*)())routine
 ******************************************************/
  "Generated data",  (int (*)())gen_num,
};

/*****************************************************
 The MAX_N_SL_DATA parameter needs to be at least as large
 as the number of displayable variables defined above
```

```
 *******************************************************/
#define MAX_N_SL_DATA 10

int n_sl_data = sizeof(dview)/sizeof(struct dw_def);
```

datawin1.h

```
/*************************************
 This file is modified for dataview1
        (original: dataconv.c)
           on 6 Dec.1994
           by M.Tairadate.
**************************************/

/********************************************************************
 File dataconv.c: contains the conversion routines from data to
                  string
 author: C. Timmermans.
 ********************************************************************/

#include "template.h"

/*****************************************************************
 The conversion routines have the following form:

  routine(header,data,txt)
  int *header,*data;
  char *txt;
  {
  *************************************************************
  Using the header and the data-buffer, one has to fill the txt
  character string. As an example I will fill txt with the
  hexadecimal rep. of the first dataword, but only if this
  data corresponds to an actual event
  **************************************************************
  if(header[1] == EVENT_RECORD) sprintf(txt,"%x",data[0]);
  else strcpy(txt,"No Dataword");
  }
******************************************************************/
gen_num(header,data,txt)
int *header,*data;
```

```
char *txt;
{
  if(header[1] == BEGINRUN_RECORD) strcpy(txt,"Begin");
  else if(header[1] == PAUSERUN_RECORD) strcpy(txt,"Pause");
  else if(header[1] == ENDRUN_RECORD) strcpy(txt,"End");
  else if(header[1] == RESUMERUN_RECORD) strcpy(txt,"Resume");
  else if(header[1] == EVENT_RECORD) sprintf(txt,"%d",data[0]);
  else strcpy(txt,"Unknown runstate");
}
```

dataconv1.c

5.3 Control Processes

第３章ではUNIXのコマンドラインからUNIDAQのプロセスを実行し、コマンドを発行した。第４章ではそれをshell scriptで連続実行させるようにした。これにより、プロセスの役割やデータの流れを理解することが出来たと思う。実際の本格的な実験では、より対話的で見やすいユーザ・インタフェースが望まれる局面もあると思われる。このような要望に対応するため、UNIDAQはグラフィカル・ユーザ・インタフェース(GUI)とコマンド・インタプリタを核とするcontrol process群を用意している。これらのプロセスを用いることで、実験中の各プロセス・データの状態監視がしやすくなり、よりきめ細かな設定も行えるようになる。以下、個別に簡単な説明を行う。(各項目末尾にDocumentation Setの参照部分を示した。)

Run Control Process　各UNIDAQプロセスに対するコマンド・インタプリタの役割を果たす。コマンドのリストをファイルで与えることができ、デフォルトでは

$USERUNIDAQ/conf/command.list

が読まれるようになっている。[User's Guide 3.6、Technical Guide 13]

Operator Process　Collectorなどのプロセスに対しGUI上でコマンドやパラメータを与える。コマンドの発行はRun Controlにより行われるのでRun Controlが動作していることが前提となる。このプロセスを起動すると４つのwindowが表示され、マウスを使ってのオペレーションが可能となる。[User's Guide 3.7、Technical Guide 14]

XPC Process　XPC(eXperimental Process Control)はすべてのUNIDAQプロセスの生死を監視し、終了や再起動を行うことができる。もしプロセスが異常終了した場合、それがXPCによって起動されたものであるならば、そのプロセスは自動的に再起動される。XPCによるプロセスの起動は

```
> ask xpc start yes [process_name]
```

のようにして行う。[User's Guide 3.8、Technical Guide 16]

XPC Checker　プロセスの生死を監視するXPC自身が異常終了してしまう場合に備え、XPCを監視するプロセス。XPCが死んでいるのを発見した場合、新たにXPCを起動する。XPC Checkerは他のプロセス同様XPCによって監視される。[User's Guide 3.9、Technical Guide 17]

Logbook Process　その名の通りロギングを行うプロセス。UNIDAQの各プロセスのスタート・ストップ・エラーなどの情報を$USERUNIDAQ/log/log.bookYYMMDDに記録する。logviewと組み合わせて使うことで、現在実行中のプロセスの状態を監視することができ、またエラーコードから動作不具合時の原因究明をすることができる。一連のプロセスのなかで最初に実行し最後に終了する。[User's Guide 3.10、Technical Guide 15]

Logview　Logbookによって作られたファイルlog.bookYYMMDDを表示するためのツール。画面にwindowを開き15秒毎にログファイルを監視する。ファイルの更新があったときその内容を表示する。当然ながら、Logviewを使うときはlogbookが動いていなければならない。Logviewは(UNIDAQの)プロセスではなくツールなので、statusコマンドでは表示されない。よって、killするときはwindow内で<CTRL-C>を実行する。[User's Guide 7.6]

以上のプロセス・ツールに加えdataviewを用いて、第3章で使ったpseudo event用にshell scriptを作成した。

$USERUNIDAQ/bin/xpc_script_pseudo

である。以下にそのソースを示す。

```
#******************************************************
# This file is modified for collector1 and anal_pseudo
#   (original: $TOPUNIDAQ/examples/example_script_a)
#                     on 6 Dec.1994
#                     by M.Tairadate.
#******************************************************
echo
echo -------------------------------------------------------------------
echo Unidaq Example Run with NOVA and Control Processes
echo -------------------------------------------------------------------
echo
echo -------------------------------------------------------------------
echo Starting NOVA
echo -------------------------------------------------------------------
echo
$TOPUNIDAQ/NOVA/src/novad /tmp/nova
echo
echo -------------------------------------------------------------------
echo Starting xpc
echo -------------------------------------------------------------------
echo
xpc &
echo
echo -------------------------------------------------------------------
echo Starting logbook, collector, analyzer, UNIdump, recorder, dataview,
```

```
echo  runco, operator, xpc_checker, logview
echo -------------------------------------------------------------------
echo
ask xpc start yes logbook
ask xpc start yes xterm -fg red -bd red -sb -geometry 62x15+910+10 \
-e collector1
ask xpc start yes xterm -fg purple -bd purple -sb -geometry 62x15+910+250 \
-e anal_pseudo
ask xpc start yes xterm -fg blue -bd blue -sb -geometry 65x19+910+730 \
-e UNIdump
ask xpc start yes recorder
ask xpc start yes dataview1
ask xpc start yes runco
ask xpc start yes operator
ask xpc start yes xpc_checker
ask xpc start yes xterm -fg orange -bd orange -sb -geometry 62x15+910+490 \
-e logview
echo
sleep 15
status
ask xpc
ask xpc stop collector1
ask xpc stop anal_pseudo
ask xpc stop UNIdump
ask xpc stop recorder
ask xpc stop dataview1
ask xpc stop runco
ask xpc stop xpc_checker
ask xpc stop logbook
status
Reset_local yes
echo
echo -------------------------------------------------------------------
echo Finished the script......
echo -------------------------------------------------------------------
echo
```

xpc_script_pseudo

このscriptをもとにユーザが自分の実験に即した記述を行っていけばよい。

以上、いくつかの便利なツールを含めUNIDAQを支障なく運用できる程度までの説明を行った。前にも述べたように、UNIDAQの大きな特徴はユーザの要求に柔軟に対応できる点である。collector.cの改造や全く新しいプロセスの開発もテンプレートに則って行えば問題ない。ぜひ試みてもらいたい。

Acknowledgements

このdocumentを書くにあたり、高エネルギー物理学研究所・データ処理センター 渡瀬 芳行教授、同・物理オンライングループ 安 芳次氏、能町 正治助手のご指導を受けました。また、東京工業大学・理学部 田島 靖久さん、筑波大学・理学部 池田 拓史君には草稿版より不明な点の指摘をしてもらいました。方々のご協力に感謝いたします。