PC-Linux と ROOT を用いた CAMAC データ収集システム開発

筑波大学一学群自然学類物理学専攻

平成 12 年度卒業論文

970330　黒木善昭

指導教官　三明康郎、江角晋一

Abstract

高エネルギー原子核実験においては、イベントは非同期に大量に発生しており、さらに高速にとり扱う必要がある。そのため、検出器らのデータ読み出しにはエレクトロニクスやコンピュータは必要不可欠な道具となっている。またこれらを組み合わせたデータ収集システムについても同様のことが言える。ここで、現在までの開発研究によって基本的なデータ収集性能は我々が小規模実験用などに要求する性能を満たしている。しかし、その一方でコンピュータや解析用プログラムは著しい開発をかさね、以前のそれよりもさらに便利で強力なものに変化している。本研究ではそのような背景のもと、依然より開発されていたシステムの作業能率面を向上させるために、解析ソフトウエア“ROOT”と Linux-OS を用いたデータ収集システムの開発を行った。また、基本的な測定性能が失われていないか確認の意味もこめ、性能評価を行った。結果は、CAMAC に対する Single Action (Read)を行う場合は、

データ収集性能 ＝ 96.1～346.0 ±0.7～3.0 [kB/sec]

という結果が得られ、**小規模実験用などに要求する性能を満たしている事が分かった。**

目次

第1章　序論

1.1 データ収集システムに求められる性能

高エネルギー原子核実験においてのデータ収集システムに求められる性能としては様々な性能が考えられるが、本研究では主に“データ収集性能”と“作業能率”という二つの性能を考慮し開発を行った。

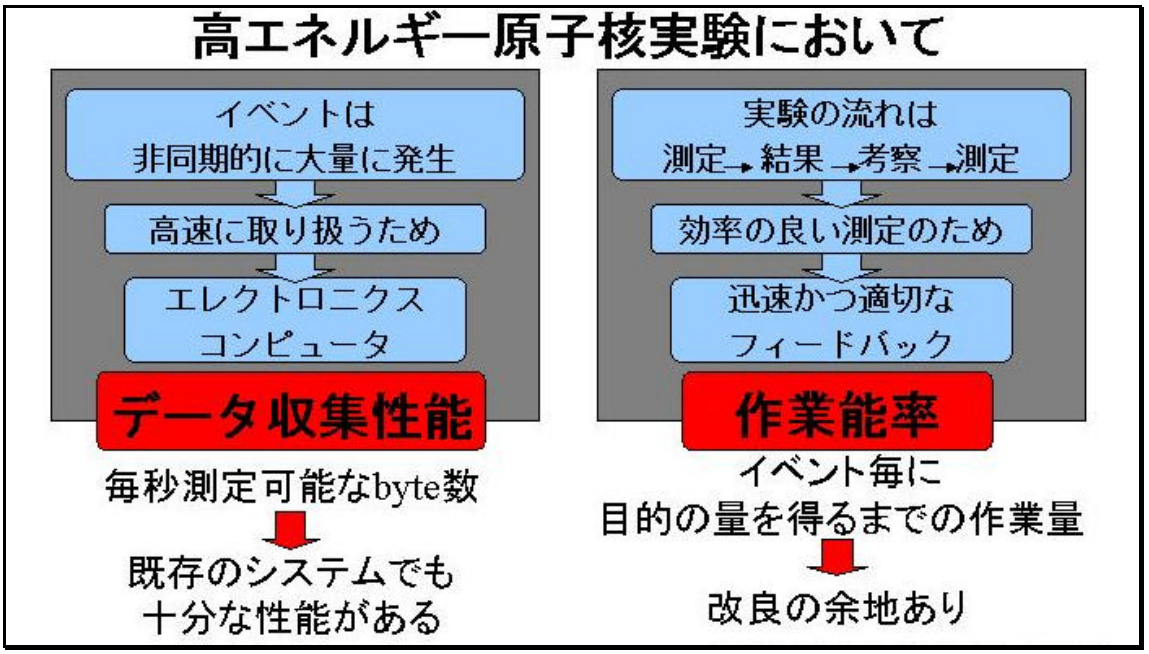


図 1：データ収集システムに求められる性能

1.2 データ収集性能

高エネルギー原子核実験においては、イベントは非同期に大量に発生しており、さらに高速にとり扱う必要がある。そのため、検出器らのデータ読み出しにはエレクトロニクスやコンピュータなどが使用されており、測定で得られるデータの統計や精度はそれらエレクトロニクスやコンピュータの性能に大きな影響を受けてしまう。そのため、データ収集システムに求められる性能の一つとしては、1 秒当たりに測定可能なデータの byte 数によって表現することのできる“ データ収集性能 ”が上げられる。これはデータ収集システムにおいては基本的性能であり、十分な研究開発がなされている。そのため既存の物でも十分な性能があり、大規模実験のような環境下での使用や、新しいデバイス・ソフトウエアに対応させる等の場合を除けば、改めて開発を行う必要はない。

1.3 作業能率

高エネルギー原子核実験に限らず、全ての実験においておおまかな流れは、測定を行い結果を得て、それをもとに考察し、さらにその考察を元として次の測定の方針と方法を決定するというものである。ここで、即座に考察を行えるようにし、測定者の負担をできる限り軽くするためにまた実験自体の効率を上げるために、迅速かつ適切なフィードバックが必要である。このようなことは“ データ収集性能 ”と比べ数値での表現が難しく、システムの補助的性能であるがため、その開発や研究は遅れをとっていた。しかし、今日のコンピュータの普及と性能の進歩に伴い、コンピュータ・ユーザーインターフェースの重要性が問われるようになり、急速な向上を示している。その良い例としては GUI（Graphic User Interface）を用いた技術を上げることができる。

本研究では、データ収集システムに求められるユーザーインターフェース的性能として、得られて実験データから目的の量を得るまでの人間が行う作業量を比較のための量とした、“ 作業能率 ”と呼ぶことにする。

第2章　開発目的

2.1 現状

我々のグループでは“ KODAQ ”と呼ばれるデータ収集システムを現在まで使用してきた。これは開発当時東京大学原子核研究所・小俣和夫氏を中心とするグループにより開発されたプログラムであり、平成 7 年度の筑波大学物理学専攻卒業論文の一環として、三浦大輔氏が我々高エネルギー原子核実験グループに導入し、解析環境との連携を向上する開発を行ったデータ収集システムである。
このデータ収集システムは測定収集したデータを CERN Program Library を用いて解析を行うことを前提としてある。ここで CERN Program Library とは CERN（欧州原子核研究所）が開発・サポートする数値計算、データ解析用プログラムライブラリの総称の事を指している。CERN Program Library は高エネルギー原子核実験におけるデータ解析に必要な機能が備わり、さらに高エネルギー原子核実験においての標準的な数値計算、データ解析用のプログラムライブラリとなっていたため“ KODAQ ”は強力なデータ収集システムであるといえる。
しかし、現在、“ KODAQ ”の基本的なデータ収集性能は依然有効であるが、三浦氏が導入を行った当時からの問題点、またその後のコンピュータ環境の変化に伴い発生してきた問題点等により作業能率の面から考えた性能は十分なものとはいえなくなっている。

MS-DOS：シングルタスク

三浦氏が導入を行った当時からの問題点として“ KODAQ ”の動作環境、特に OS（オペレーション・システム）についての問題が上げられる。“ KODAQ ”の動作 OS は MS-DOS であり、Microsoft WINDOWS 等のマルチタスク機能を持ち合わせた OS でないため、解析と共に他の作業を行うのは非常に困難なこととなっている。また、WINDOWS９８上での DOS-WINDOW による“ KODAQ ”の使用は可能であるが動作自体は MS-DOS 上で行わなければならず、他の作業は一切実行不可能となるため、結果としてシングルタスク OS 上で動いているのとおなじことであった。

作業能率の低下

三浦氏が開発した当時の解析はWINDOW上で起動するCERN Program Library を使用していたが、研究室のコンピュータ環境の変化に伴い、主な解析は UNIX 系 OS 上で行うようになってきた。このため、測定系と解析系の統一性が崩れ、作業能率は低下した。さらにインターネット等の普及と共に現在コンピュータネットワークにおけるセキュリティー問題は深刻なものとなっている。それを受け我々高エネルギー原子核実験グループにおいても UNIX のセキュリティーを強化したため、WINDOWS 等からの直接データ転送（FTP）を行うことが不可能となった。このため、各自その場に応じたデータ転送方を考え解析を行っていたが、これは非常に作業能率の悪いものだったといえる。

2.2 本研究の目的

そこで、本研究の目的として、**我々筑波大学高エネルギー原子核実験グループにおけるテスト実験の**典型的な環境として上げられる、100Hz の Event Rate において 18words のデータを測定可能な程度の**データ収集性能と**オンライン測定とオフライン解析が数コマンドで可能な作業能率を持ち合わせたデータ収集システムの開発とした。

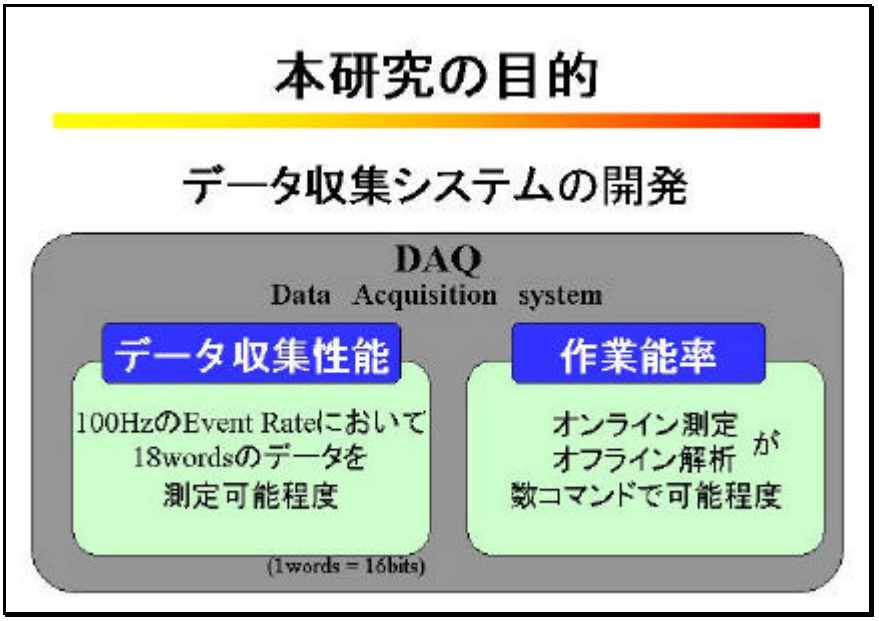


図 2：本研究の目的

第3章　作業能率

3.1 作業能率の向上

前述のように“作業能率”は、数値での表現が難しく、さらにその向上のためにはハードウエア的にもソフトウエア的にも高度な技術を必要とする。そのため、近年ようやく重要視されはじめ、現在ユーザーインターフェースを重視した多くのハードウエア・ソフトウエアが開発され標準的なものとなっている。この事は高エネルギー原子核実験においても同様で様々なハードウエア・ソフトウエアが開発されている。そして、その多くはただ使いやすいといったことだけでなく、より強力な基本性能も持ち合わせたものとなっている。
本研究ではこのような現状を踏まえた上で以下のようなシステム構成のデータ収集システムを開発することにした。

3.2 システム構成

図 3は本研究で採用した測定系の一例の模式図になる。測定対象である放射線が検出器、シンチレーションカウンタを通過することによりカウンタを発光させる。この光は光電子増倍管によりアナログの電気信号へとおきかえられ、NIM（Nuclear Instrument Module）規格などのエレクトロニクス回路を通った後、コンピュータとエレクトロニクスの情報交換方法についてのハード的・ソフト的規格である CAMAC 規格に渡される。最後にコンピュータへと送られ測定データとして適切な形式をもって保存されていく。
ここで、検出器やエレクトロニクスの部分の仕様や規格への汎用性を持たせるために CAMAC 規格を採用した。また、**パーソナルコンピュータの普及が急速に進んでおり、またその性能も一昔前のワークステーションに匹敵する程に向上してきている状況をふまえ、新しいシステムのホストコンピュータとしてパーソナルコンピュータ・**PC（AT 互換機）**を採用し、システムの構築を行った**。OS としては Linux を、主な解析用ソフトウエアとしては ROOT を採用した。

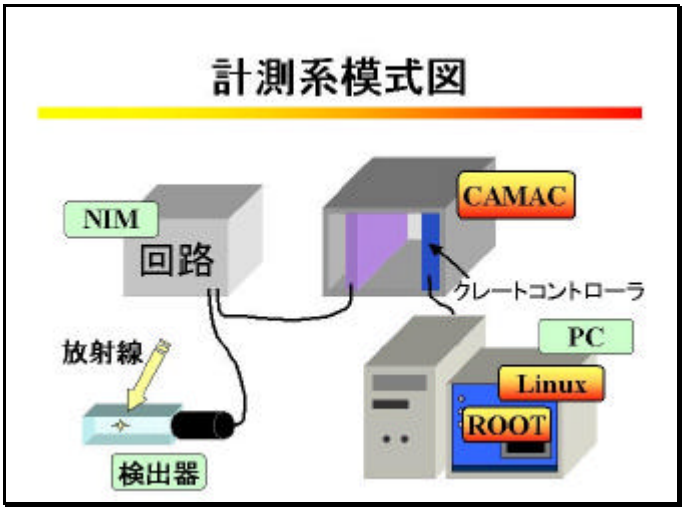


図 3：測定系模式図

以下、CAMAC・Linux・ROOT について説明とその利点について述べる。

3.2.1 CAMAC

CAMAC とはコンピュータと制御・測定系との間の情報（測定データ及び制御情報）の交換方法について、ハード及びソフトの両面における規格のことをさし、“Computer Automated measurement And Control”の略称である。

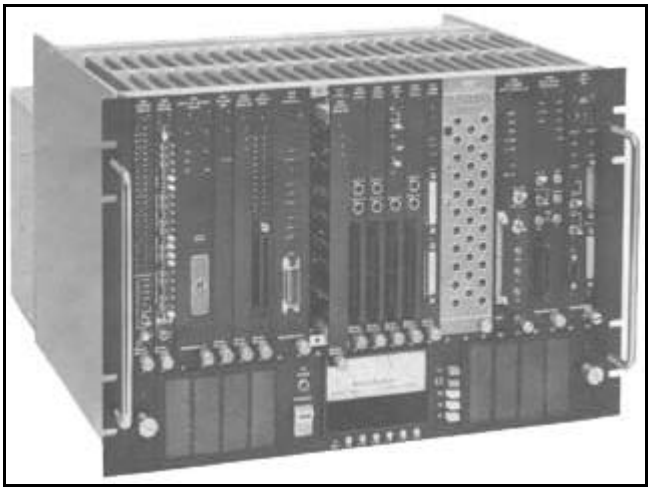


図 4：CAMAC 外観

世界標準規格

1960 年代後半、エレクトロニクスの進歩に伴い、原子核物理実験においても、小型コンピュータを用いたオンライン計測が普及し、大量のデジタル情報処理を取り扱うようになってきた。このような動向の中から、それら大量のデジタル情報を取り扱うオンライン測定系に対する新たな規格を要求する声がおき、CERN （欧州共同原子核物理学研究所）を中心として提唱され、ヨーロッパ共同体下部組織である ESONE 委員会がこれを受ける形となって、ヨーロッパ統一規格として、1969 年 CAMAC 規格を正式に制定した。その後、ACE 、IEEE で承認され国際標準規格（ANSI/IEEE publication SH-08482 ）となった

そのため独自に開発・製作を行っている測定系と比較し、その優位性は顕著となっている。ハードウエア及びソフトウエア (コンピュータとのプロトコル) が世界的に標準化されているため、系の更新・拡張も CAMAC 基本ファンクションモジュール (I／O モジュール) をベースとして行なえ、コストをおさえながら、系の老化を徹底的に減じることが可能となっている。

CAMAC 自体、現在では既に古く良いものであるとはいえないが、これまでに蓄えられたハード・ソフトの資産が多く受け継がれてきており、また様々な用途に合わせた多くのモジュールを商業的にも比較的容易に入手することができるなどの理由で、制定されて 20 年以上経つ現在でも最もよく用いられている。

コンピュータ独立性

高度な互換性、保守性、拡張性、柔軟性を保持しつつ、コンピュータの構造に依存しない Online 制御／測定系を構成することを目的として定められた CAMAC では、コンピュータの I/O bus にインターフェースする System Driver “ Crate Controller ”とそれに付随した System Bus“ Data way ”にインターフェースする分割された制御／測定ユニット“ Module ”とを“ Crate ”と呼ばれるケースに実装するという方式を取る。各 Module 及び“ Crate Controller ”は、自由に抜き差しできるような、“ 1Module Instrumentation System ”になっている。CAMAC クレートの後部は 43 ×2 のボードエッジコネクタが取り付けられ、バス線（CAMAC データウェイ）に接続されている。各ノーマルステーションには、24bit の READ 、WRITE それぞれの専用バス、9bit のコントロールバス、モジュールのイニシャライズ、クリア、インヒビット、のための Z 、C 、I 、モジュールのステーションナンバー、コマンドアクセプト、レスポンスを示す N 、X 、Q 、割り込み線 L （LAM: Look-At-Me ）、タイミング信号線 S1 、S2 、バスの使用を示す B 、の各 1bit のバス線につながる。コントロールステーションは、READ 、WRITE のバス線に代わり、各ステーションからの L と N の線がつながる。コントロールバスは、F （5bit ）、A （4bit ）

で構成され、F はファンクション、A はサブアドレスを指定するものである。これらと N を組み合わせて、各ステーションに配置される CAMAC モジュールをコントロールする。コントロールに用いる N 、A 、F の信号は、ハンドシェイク行わないで 1 サイクルで動作が完全に終了し、その動作でデータ転送も完了する。バスのサイクル時間は、1μsec と規定されており最大３ MB/sec の転送能力を持つ。CAMAC クレートとコンピュータの接続は、クレートコントローラとコンピュータインターフェイスを用いて行う。クレートコントローラは CAMAC クレートの右端に配置する。

CAMAC システム構成

- CAMAC Crate
  Module の収納が容易でもあると同時に、背面に Card-Edge Connector のついたマザーボードを配しており、Plug-in された Module が Dataway と呼ばれる System Bus と結合されるようになっている。また、Dataway には電源ラインもあり、Module は Dataway から電源を供給されるようになっている。
- Crate Controller
  コンピュータとインターフェイスしており、コンピュータの指令によって、Dataway Operation を行う。これにより Module コンピュータ間の情報交換が行われる。また、Module からの割り込みもコンピュータへ仲介する。その構造はコンピュータに依存しており、使用しているコンピュータにより Crate Controller も異なったものが使用される。
- CAMAC Module
  CAMAC 規格によって、形状、寸法が規定されている。全部のパネルには通常いくつかのコネクターがついていて、それらを通じて外部機器と接続され、制御、測定を行うようになっている。また後部は Crate の Card-Edge Connector に Plug-in できるような構造になっており、Plug-in することにより Module と Dataway が接続され、コンピュータとの情報交換が可能になる。
- Computer
  パーソナルコンピュータ、ミニコンピュータ、大型コンピュータ、マイクロプロセッサ等。

3.2.2 Linux

“Linux”はフリーな UNIX 互換オペレーションシステム(OS)のひとつである。本来、“Linux”とは OS の核となる kernel のみを指すが、一般的には Linux kernel ベースのシステム全体を示す。POSIX 規格（IEEE standard 1003.1 ）にしたがった実装がされ、フィンランド Helsinki 大学の Linux Torvalds 氏によって作成されその後インターネット上で大きく発展してきた OS であり、その特徴のひとつとして「再配布」が可能なことがよくあげらる。そのため、エンドユーザのプラットフォームとして高価な UNIX システムに取って代わりつつある。カーネルの機能は System V 互換になっているが、ネットワークデーモンなどのユーティリティ機能は BSD から移植されており、System V と BSD の拡張機能の両方を兼ね備える。また、基本的なコマンドは GNU のものが使われており、さらに C コンパイラ等のプログラム開発環境をはじめとするほとんどの GNU ツールも移植されている。このように Linux は UNIX システムにおける機能のほとんどを実現しており、ソースレベルで互換性を持っている。以下に、Linux の持つ主な機能をあげる。

マルチタスク

測定したデータを Online で表示または解析する上で OS に要求される性能のひとつに“マルチタスク”があげられる。“マルチタスク”とは 1 台のコンピュータで同時に複数の処理を並行して行う OS のアプリケーション管理機能を指す。CPU の処理時間を非常に短い単位に分割し、複数のアプリケーションソフトに順番に割り当てることによって、複数の処理を同時に行っているようにみせている。MS-DOS 等のように同時に 1 つのアプリケーションソフトしか起動できないシングルタスク OS と比較して、複数のアプリケーションを同時に使用することができるためアプリケーションの拡張性、汎用性に優れているといえる。対して、OS のオーバーヘッドが少ないという意味では、MS-DOS 等の単純なシステムの方が有利とされているが、これは今日の驚異的な CPU 速度向上によって大きな問題とはいえなくなっている。

UNIX 互換 OS

測定された Data の最終的な解析は UNIX 系 OS 上で行う事が主である、そのため測定系も UNIX 互換 OS 上で行う事のメリットは単に Data の受け渡しが容易であるということだけにとどまらず、Stand Alone での online 解析でも同等の解析を行える可能性を持たせ、さらにそこで使用した解析マクロを後の解析に

そのまま使用できると言う利点もある。
また、UNIX 互換 OS であること自体、様々なメリットを有している。例えば、TCP/IP 、IPX プロトコル、Aplle Talk ネットワークといったネットワークプロトコルをサポートしており、これらを利用したネットワークサービス機能を備え、また深い信頼性を有していることや、マルチユーザであるために Data 管理が容易であることなどがあげられる。また、単に OS 自体が安定であると言うメリットもある。

コストパフォーマンス

Linux は x86, Motorola 68k, Digital Alpha, SPARC, Mips, Motorola PowerPC 等の各アーキテクチャで動かすことができる。現在特に使用されているのは intel 系のものと Alpha 系のものが上げられる。さらにその二つの中でも信頼性と動作しているプログラムや周辺機器の多さといった面から見ると intel 系 の機種を選択するのが適当だとされている。intel 系機種の代表的なものとして IBM の AT 互換機が存在するが、これはここで述べるまでもなく現在広く普及している。そのため、単価も安く。OS の Linux にいたっては Free であるので、高い OS ライセンス料や workstation のように高いハードウエアの必要な他の UNIX マシンと比較してはるかに低価格で同等の性能を手にすることができるようになっている。

3.2.3 ROOT

“ROOT”とは C++を基礎開発言語としたオブジェクト指向型データ解析用アプリケーションのカプセル化したライブラリーのことを指す。これは CERN（欧州原子核研究所）等で開発が進められており、同じく同研究所で開発されている CERN Program Library と同等以上の機能を兼ね備えている。

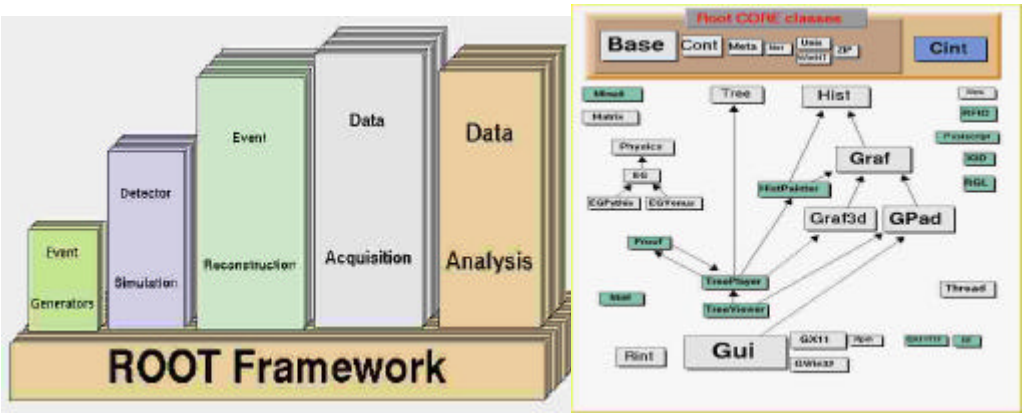


図 5：ROOT 概念図

C++

ROOT はその基礎言語に C++採用している。このことがそれまでの CERN Program Library と大きく異なる点といってよいだろう。確かに CERN Program Library は数多くの強力な解析 TOOL を含んでおり、それらが現在の高エネルギー物理学実験において重要な役割を担ってきていることは確かなことだ。しかし CERN Program Library の基礎言語である FORTRAN にはいくつもの制約や C++等で重要視されている機能が備わっていないため、より複雑で高度なアプリケーションを開発するためには限界がある。そのため、それまでの CERN Program Library の概念、機能を元に C++による新たなフレームワークを作り出すプロジェクトが発足された。

OS 親和性

基礎言語に C++を採用している利点として OS（オペレーションシステム）への親和性が非常に高いことが上げられる。これは一般的なオペレーションシステムがよくC++の基礎となっている C 言語によって記述されているためといえる。例えば新たにアプリケーションプログラムの開発を行う場合、オペレーションシステムの持つ機能を使用したいとき C 言語で書いておけばスムーズな使用が可能となる。また、もともと C 言語はプログラム実行速度が高速でかつ、文字データの扱いが容易に行える特性上多くのアプリケーションにおいてシステム記述言語として採用されている。

オブジェクト指向

C++の持つオブジェクト指向性はより複雑で高度な処理を容易に実行可能とする。この良い例として GUI が上げられる。例えばウインドを一つ開くことですら、実際にそれを1から定義し実行させるには大量のプログラムが必要となってしまう。しかし、通常ウインドは同様のものを使用しても何ら問題なく、そのため既存のウインド用プログラム（フレーム）の部品をライブラリから持ってくるような形式を取りプログラムは書かれている。このように一人がライブラリを作成すれば済むような構造をもつため、非常に汎用性の高くまた拡張性・柔軟性・保守性を有する。そのため、C++は**高エネルギー原子核実験における解析プログラムとして要求される**複雑で高度な処理を容易に実行可能とし、現在**高エネルギー原子核実験では以前の一般的な言語であった** Fortran **からの移行がすすんでいる。**

3.2.4 システム全体としてのメリット

CAMAC・Linux・ROOT **のそれぞれについては現在高エネルギー原子核実験において要求されるほとんどの性能を持つことがいえる。さらに、これらの複合システムであるデータ収集システムにおいても、十分な性能を得られること容易に想像できることだ。ここで現在、**CAMAC **と** Linux **を用いたデータ収集システムは**

多数開発されており、その性能も非常に高いもいものといえる。一方でこれらとROOT を結びつけたデータ収集システムは開発されていなかった。しかし、先にあげたとおり現在高エネルギー原子核実験における ROOT の重要性は高いものがあり、今後さらにそれは増してゆく事も想像できることを考慮に入れると、ROOTをシステムの一部、データ処理・解析プロセスとして持つデータ収集システムの開発は強く望むところであった。そこで、本研究においてはこの ROOT をシステムの一部として用いてデータ収集システムの開発をおこなった。

3.3 開発結果

3.3.1 開発環境

本研究におけるデータ収集系システムの開発には以下のようなハードウェアの環境のもとで行った。

CAMAC Controller

(株)東洋テクニカ製 CC／7700

CAMAC 規格タイプ A-2 準拠のクレートコントローラ

Interface Board

(株)東洋テクニカ製 CC／PCI

ホスト・コンピュータのインターフェース、PCI 規格

Computer

自作 PC（AT 互換機）を使用した。

マザーボード：Aopen AX63PRO

CPU：PentiumⅢ 550MHz

メモリー：128MB(PC-100)

ハードディスク：Seagate ST320430 20.4G IDE7200RPM

ビデオカード：Ati RageⅡC

イーサネットカード：crega Fast Ether Ⅱ PCI-Tx

OS：RedHat Linux6.2J （kernel 2.2 系で動作が可能です。）

3.3.2 データストリーム

高速な割込みを実現するため、データ収集プロセスはデバイスドライバーの形でカーネルに組み込まれている。また、解析等の比較的実行速度の遅いプロセスが Data を受け取っても、読み落としが発生しないように共有メモリーの使用を採用している。

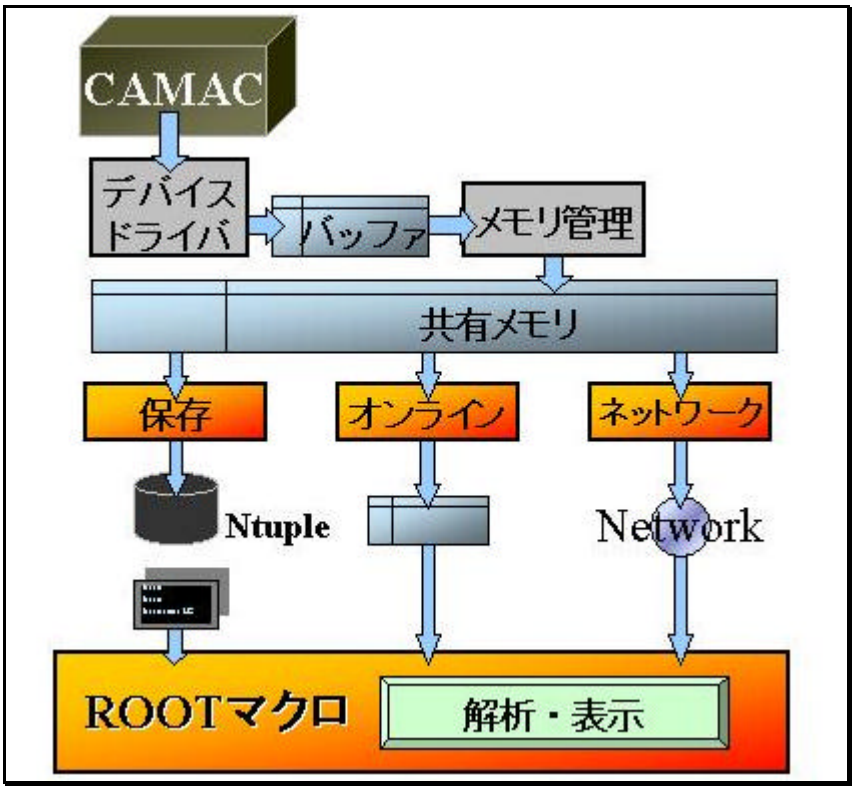


図 6：データストリーム

CAMAC からのデータはカーネルに組み込まれたドライバが 64kwords のバッファに書き込まれる。これを“ blocker ”と呼ばれるブロック化プロセスが 8 連の 16kBit に区切った共有メモリーに落としていく。このとき、デバイスドライバーの 64 キロワードのバッファーが一杯になった場合には、ブロック化プロセスから読まれるまでデータ収集を停止するようになっている。その後、データは 3 つのプロセスによって読み出される。主となるものは“ analyzer ”と呼ばれているもので、ASCII、Ntuple 等の形式でデータをディスクに保存する。また、“ daqcl ”と呼ばれるプロセスは読み出したデータをヒストグラムにまとめ、ネットワークを通して他のマシーン（Linux, Unix, Win 等）や、自らにデータを送りオンラインでデータ表示を可能にしている。また、スタンドアローンオンラインモニターとして、“ mapon ”と呼ばれるプロセスが存在している。どちらのモニターを使用するかは容易に選択できる。また、測定によって得られたデータファイルは NFS（Netowork File System）等によって効率よく他のマシーンへ転送できるようになっている。

3.3.3 主なプロセス

デバイスドライバ

一般的な UNIX の I/O 制御と同様に、CAMAC 制御もデバイスドライバを通して行われる。バスの初期化等の速度を要求しないオペレーションは、このドライバに“ ioctl ”を発行することによって行っている。一般的な原子核実験では非同期に発生するイベントに対して遅延なくデータの読み出しをする必要があり、デバイスドライバを介して処理していては間に合わない。そこで、割り込み発生時の処理をデバイスドライバ内に記述し、データを適当なバッファーに貯めておいて他のプログラムがデバイスドライバから読み出す方法を取っている。つまり、実験装置や読み出しデータに応じて、デバイスドライバを書き換える必要があることとなる。Linux では kernel を止めずに組み込み、削除可能なデバイスドライバであるローダブルモジュールの仕組みが採用されている。このため、ユーザーは測定条件に応じて一つのルーチンファイルを書き換えるのみでドライバの再構築をすることが可能である。

動作

ここで測定開始時、終了時、またイベント測定時に kernel に対して行われている処理の説明をおこなう。

- **測定開始時**

  測定開始時の処理としては図７のような処理が行われており、この処理が実行されると、先ず測定 flag を 1 にする。次にデバイスドライバへの割り込み許可を出す。

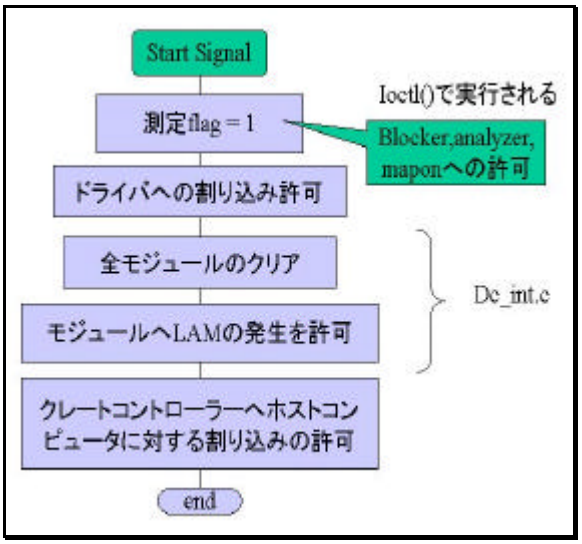


図 7：Start Signal

これによってデバイスドライバは linux 上のプロセスから割り込みを受

けるようになる。次に、CAMAC のモジュールに対してクリア命令を発行し、Interrupt Register 等の特定のモジュールに対して LAM の発生を許可する命令を発行する。最後に Crate Controller に対してホストコンピュータへの割り込み要求を出す許可を行う。以上のような処理によって Interrupt Register 等の特定のモジュールにシグナルが入りモジュールが LAM を発生すると、Crate Controller は LAM の発生をホストコンピュータへ知らせるようになる。これによって、測定が可能となる。

- **測定終了時**

  測定終了時の処理は開始時の逆の処理といっていい。ここでは、先ず測定 flag を 0 に落し。次にデバイスドライバへの割り込み許可を取り消す。最後に、Crate Controller に対してホストコンピュータへの割り込み要求を出す許可を取り消す。以上のような処理によってデバイスドライバへの割り込み要求は途絶え、データは収集されなくなる。

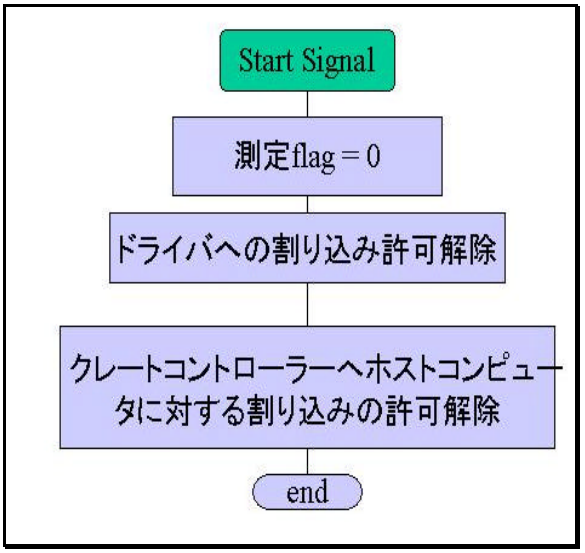


図 8:Stop Siganl

- **イベント測定時**

  測定開始の処理をおこなったあと、Interrupt Register 等の特定のモジュールにシグナルが入りモジュールが LAM を発生すると、Crate Controller は LAM の発生をホストコンピュータへ知らせるために kernel 内のデバイスドライバに割り込み要求を行う。これを受けたデバイスドライバは図 9 のような処理をおこなう。

  初めに、デバイスドライバに対して起こる割り込み要求が全て CAMAC・Crate Controller からのものとは限らないので、その割り込み要求の発生

元を確認する。次にデータを格納するバッファの残量があるかを確認する。ここでもし残量がない場合には読み出しプロセスである“blocker”がデータを読み出して行くまで、データ収集処理を行わないよう待機処理を行う。割り込みが CAMAC からのものでありバッファも十分に残っている場合、デバイスドライバは CAMAC の各モジュールに対して指定された NAF コマンドを発行しデータを読み出してくる。その全てのコマンドが終了するとモジュールのクリアを行い、割り込みの全ての処理を終了する。このようなサイクルをもってデバイスドライバはデータ収集を行う。

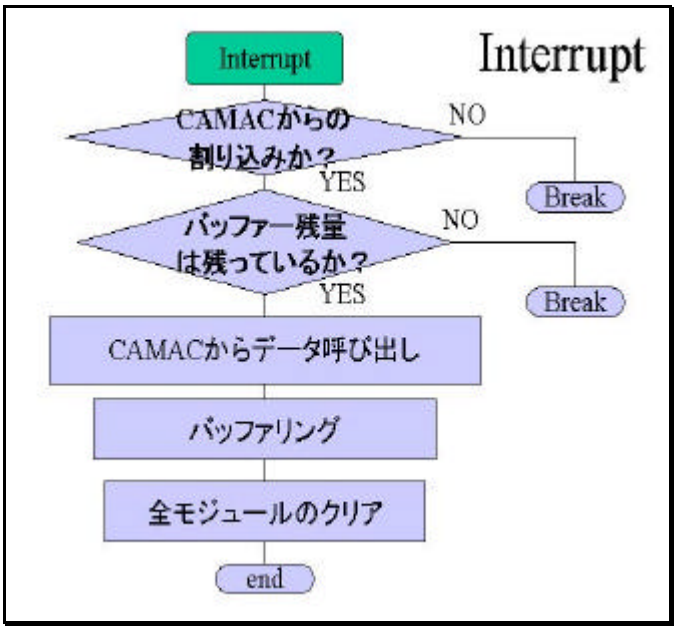


図 9:Interruput 処理

また、このような割り込み処理のルーチンは dc_int.c に記述されている。また、この設定ファイルはデバイスドライバとして make の際に自動的にインクルードされ Linux kernel 内に組み込まれる、そのため幾つかの制限をうけることになる。

- Kernel が持っているオブジェクトのみが呼び出し可能となる。通常での C のライブラリである sdtio.h（標準入出力）ですら使用は不可能となる。
- 無限ループを作成しない。
- メモリ保護例外、零割りを作成しない。
- 変数は static 型で記述する。

  auto 型等で巨大な配列ができると危険となる。

以上のような制限と注意を守らないと OS のハングアップにつながる恐れがあるので注意をしておかなくてはならない。また、その他にも異常

動作を起こすとカーネルパニックの原因となりうるので注意が必要となる。

以下にここで使用される主な関数をあげる。

- void camac_CNAF(int C, int N, int A, int F)
  クレート番号 C[0-7],ステーション番号 N[1-23],サブアドレス A[0-15],ファンクション F[0-31]を送り、CAMAC サイクルを起動させる。
- void camac_NAF(int N, int A, int F)
  CNAF と同様、ただし C=0 と固定されている。
- void camac_write16(short data)
  次の CAMAC コマンドのために 16Bit のデータを送る。
- void camac_write24(long data)
  次の CAMAC コマンドのために 24Bit のデータを送る。
- short camac_read16( )
  前の CAMAC コマンドで得られた 16Bit のデータを返す。
- long camac_read24( )
  前の CAMAC コマンドで得られた 24Bit のデータを返す。
- printk(const char *format ,...)
  stdio 内の関数 printf()と似たもので、使用法は同様。出力は/var/log/messages となるが、頻繁に使用するとシステムの負荷が増大する。

“blocker”

このプロセスは共有メモリーの管理を行っている。図 10 はこれのフローチャートになる。測定フラッグが立っていること確認し、バッファーから読み出したデータを共有メモリーへと書き出している。この時、共有メモリーの構造は“ camac.h ”というファイルによって定義されている。

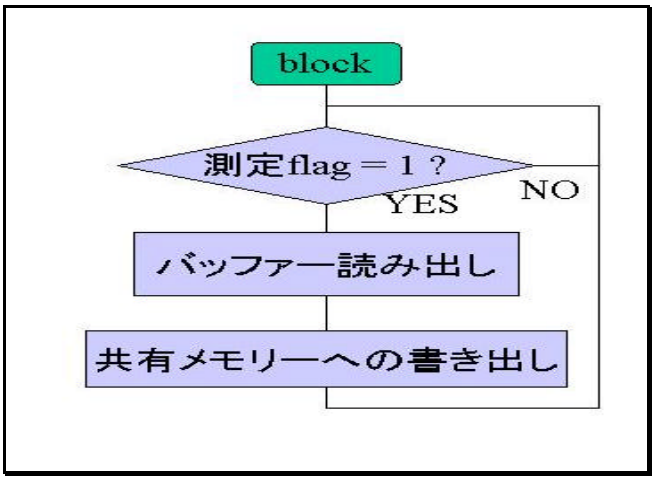


図 10：blocker フォローチャート

“共有メモリー”

共有メモリは 16kbyte ごとに区切った 8 連の構造を持ち、共有メモリからデータを読み出す各プロセスの処理が遅くデータ収集に追いつかない場合には各共有メモリの区切ごとにイベント単位で間引きをおこなっている。共有メモリのデータ構造、キー等の情報は設定ファイル“ camac.h ”に記述されており、また間引きを行うときの制御にはセマフォを使用している。また、共有メモリの上にあるデータの読み出しは、複数のプロセスから行える。

“analyzer”

共有メモリから読み出されたデータを ASCII、Ntuple 等の形式でディスクに保存する。このシステムのメインストリームとなる。Ntuple 形式は ROOT のクラス“ TNuple ”を使用しオブジェクトの生成を行い、クラス“ TFile ”によってそのオブジェクトをファイルに落としてゆく。
図 11 はこのプロセスのフローチャートになる。

このプロセスは起動されるとはじめにデバイスドライバや共有メモリーが正常に動いているかを確認する。その後、データを保存するためのファイルや ROOT のオブジェクトを開き、それらが正しく開ければ、共有メモリからデータを読み出してくる。ここで、測定されたデータにはイベント番号が付けられ、これが測定開始時にユーザによって設定された値に達したとき測定の終了プロセスが実行される。

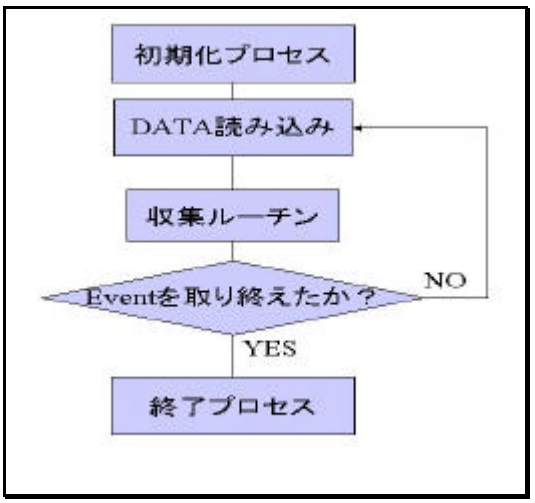


図 11：analyzer フローチャート

“ mapon ”

このプロセスはスタンドアローンでオンラインモニタリングを行う場合に用いるプロセスである。図 12 を見られると、共有メモリからデータを受け取る方法、内部処理の構造は“ analyzer ”とほぼ同じものということが分かる。ここで“ analyzer ”との違いはその内部で実際呼び出されている ROOT のクラスの違いであり、“ Thist ”を使用してヒストグラム・オブジェクトを生成しそこにデータを取り込んだ後、それをクラス“ TMapFile ”を使用して共有メモリへと格納している。

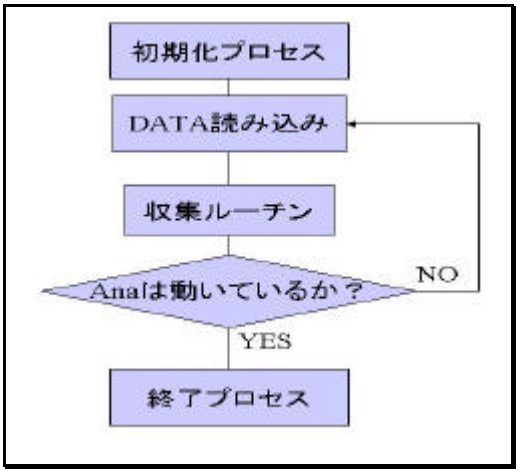


図 12：mapon フローチャート

“ daqcl ”

このプロセスは“ mapon ”と違い、ネットワーク経由でオンラインモニタリングを行う場合に用いるプロセスである。これも“ analyzer ”と似た構造を持ち、“ mapon ”と同様にヒストグラムにデータを収納した後、ROOT のネットワークスクラスを使用して他のコンピュータへ生成したヒストグラム・オブジェクトを転送する。

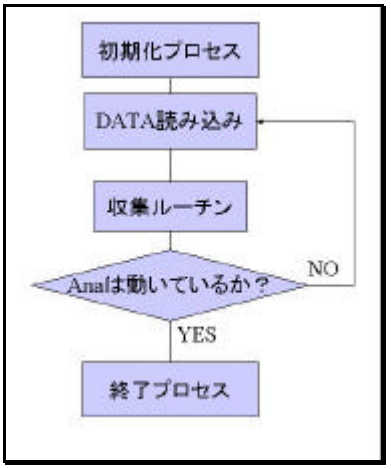


図 13：daqcl フローチャート

3.3.4 ユーザインターフェイス

図 14：コマンドパネル

本研究で開発を行ったデータ収集システムでは測定開始等のコマンドはシェルスクリプト形式でまとめられ、コマンドラインからの起動が主な方法となるのだが、図 14 は Linux・RedHat6.2J 上のウインドウマネージャ“GNOME”を使用した例である。この例では GNOME の“パネル”と呼ばれる部分にデータ収集システムの基本的な実行コマンドを“アイコン”として組み込んだ。
これらの中でも主に必要になるコマンドは図 14 の 4 つのコマンドとなる。

- **初期設定**
  PC 再起動時、デバイスドライバ設定変更時等にこのボタンをクリックする。このボタンによってシェルスクリプトとして書かれた“ setup ”が実行される。先ず、起動中のメモリ管理プロセスである“ blocker ”を停止させ、次に kernel に組み込まれている古いデバイスドライバを削除する。次に新たに作成したデバイスドライバの組み込みを行い、同様に新規の“ blocker ”を起動させる。さらにバッファ、共有メモリのクリアーを行う。最後にシステムステータス表示用のプロセスを起動させる。

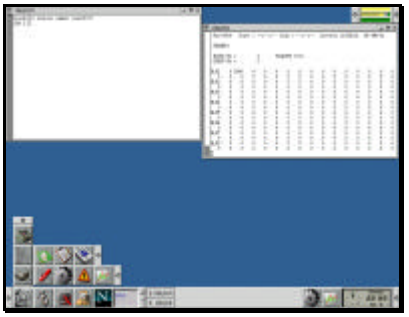


図 15：初期設定画面

- **測定開始**

  測定開始時にこのボタンをクリックすると収集システムは起動しデータを測定・収集しはじめる。このボタンによってシェルスクリプトとして書かれた“ start ”が実行される。このシェルスクリプトは図 17 のようなプロセスを実行する。具体的に見てゆくと、データ収集システムが測定中になっていないかということ事を確認するために、測定中の状態を示す“ flag ”が 1 か 0 か判断する。ここで、1 の場合は測定中であり、0 の場合は測定中ではない。そして測定が行われていないことを確認した後、“ analyzer ”、“ mapon ”、“ daqcl ”といった 3 つのプロセスが起動していないか確認をし、もし起動しているようであればそれぞれを停止するようになっている。次にバッファの初期化を行った後、ランナンバーと測定イベント数の入力を求めるウインドウを表示するプロセスを起動させ、そこで測定者がその二つを入力すると“ analyzer ”、“ mapon ”、“ daqcl ”の三つのプロセスを再起動させる。

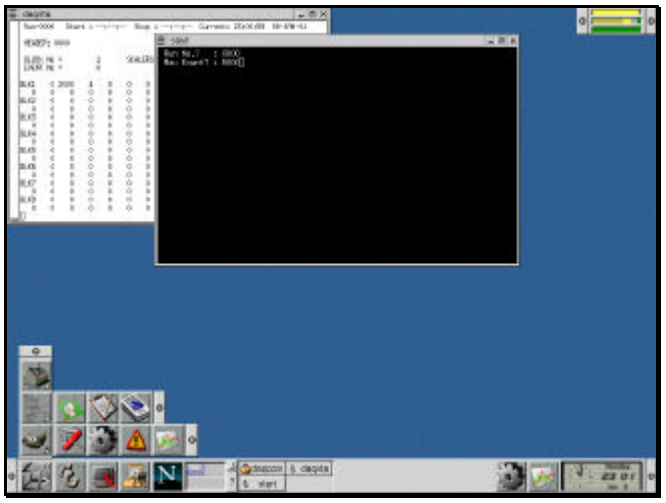


図 16：測定開始画面

そして最後に測定開始の合図を kernel に対して出すプロセスに処理を渡し、終了する。測定は入力したイベント数を“ analyzer ”が処理すると自動的に終了のシグナルを kernel に対して発行するようになっている。画面には図 16 のように TERM が一つ開きランナンバーをイベント数を聞いてくる処理しか目にすることはできない。

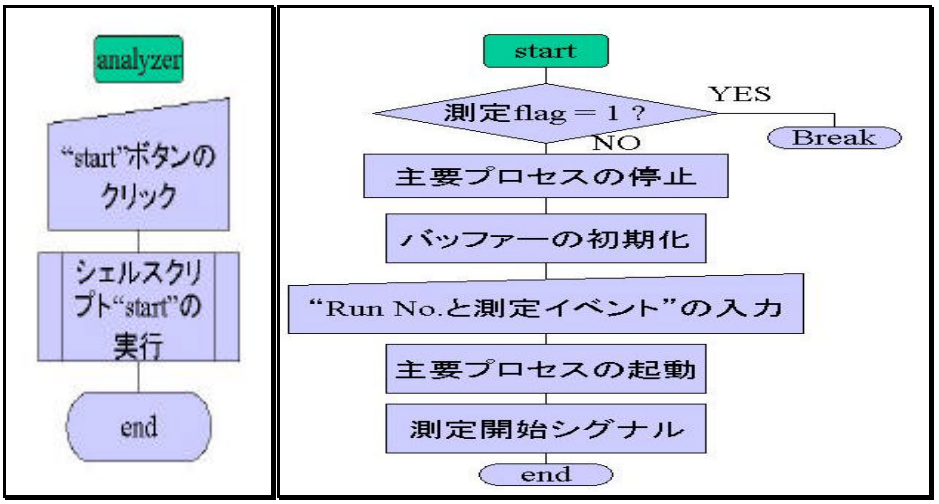


図 17：GUI・Start 処理

- **測定終了**

  測定中に何かの都合で測定を中止したい場合がある。このようなときに収集システムを終了させる時にこのボタンをクリックする。

  このボタンによってシェルスクリプトとして書かれた“ stop ”が実行される。このシェルスクリプトは図 18 のようなプロセスを実行し、測定開始の場合と同様にデータ収集システムが測定中かどうか確認をする。この時、測定中は flag=1 なので、その場合には測定開始の合図を kernel に対して出すプロセスに処理を渡し、その処理が終了したら“ analyzer ”、“ mapon ”、“ daqcl ”といった 3 つのプロセスをそれぞれ停止するようになっている。

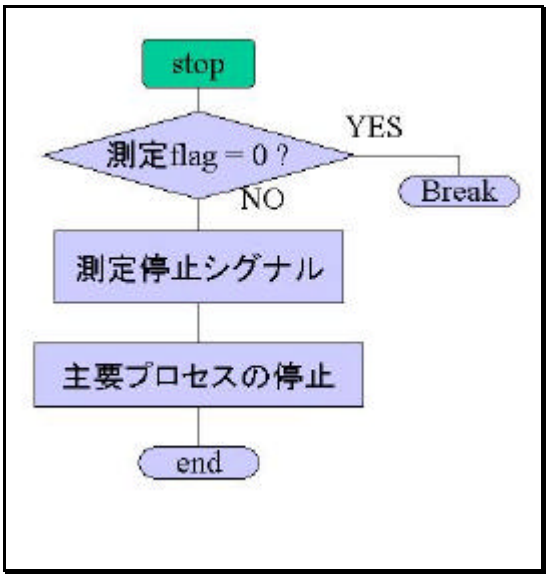


図 18：stop 処理

- **オンライン表示**

  このボタンは測定中、もしくは測定終了時に得られたデータを設定しておいた形式で表示するためのものである。ここで、データは“ mapon ”によって共有メモリに格納されたデータを読みこんでくることになる。そのため、“ mapon ”中で使用している ROOT のクラス“ TMapFile ”を同様に使用しなければならない。このため、数行の固有のプログラムはあるもののそれ以外の部分はオフラインで用いる ROOT 用のマクロと何ら変るものではなく、ROOT で行える解析は、統計解析・fitting 等どのようなものでも可能である。しかし、特に fitting のような機能は非常に CPU 占有率が高く、また処理スピードも低速であるので PC・コンピュータの性能によって使用する機能に制限がかかってしまうことが考えられる。

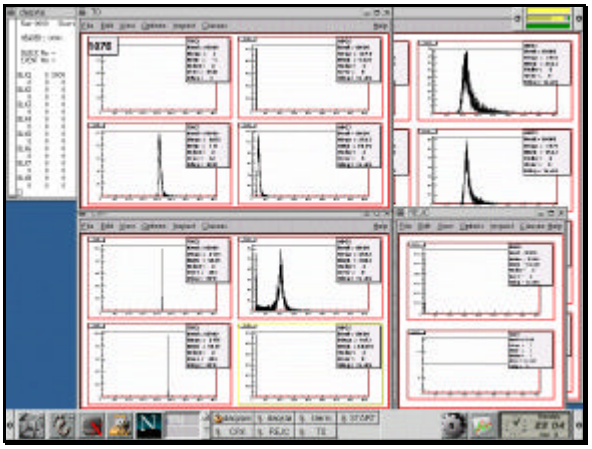


図 19：オンライン表示画面

3.3.5 解析用マクロ

このデータ収集システムにおいては解析用プログラムである ROOT をスタンドアローンで持つため、“ analyzer ”によって保存された Ntuple 正形式・または ASCII 形式のファイルは即座に解析を行うことが可能である。実際にオフラインで解析を行うときに用いる ROOT 用のマクロファイルさえあればそれを実行するのみで可能となる。これは、実はあたりまえのことを言っているに過ぎなく、この場合の解析は通常、解析で ROOT 使用している事と何ら変りはないからである。
このように実験の際に解析用マクロファイルの容易ができていれば、その実験における作業能率は飛躍的に向上するといえる。

3.4 作業能率

以上のようにデータ収集システムの開発が行えたが、“作業能率”という言葉を定義にもどりその評価をおこなう。ここで、“作業能率”とはイベント毎に測定できたデータから目的の量を得るまでの人間が行う作業量と定義付けされている。そのため、ここで測定者が行う作業量についてまとめる。

本研究で開発したデータ収集システムにおいてはその測定時に必要とされる操作は全て GUI を用いてよういされ、

- 初期設定
- 測定開始
- オンライン表示
- オフライン解析用マクロの実行

の四つでまかなえている。さらにそれらの間でキーボードからの入力はランナンバーと測定イベント数、オフライン解析用マクロの実行の 3 コマンドとなる。
ここでオフラインマクロの内容にもよるが非常に強力な解析を数コマンドで実行することが可能となっている。

以上まとめると、スタンドアローンでもネットワークでもオンライン測定ができ、さらにオフライン解析も数コマンドで実行可能なデータ収集システムの開発が行えたといえ、このデータ収集システムの有する作業能率は非常に高いものであるといえる。

第 4 章データ収集性能

4.1 性能評価

4.1.1 システムの構造

高速な割込みを実現するため、データ収集プロセスはデバイスドライバーの形でカーネルに組み込まれている。また、解析等の比較的実行速度の遅いプロセスが Data を受け取っても、読み落としが発生しないように共有メモリーの使用を採用している。このため各プロセスの処理時間を評価するためにはプロセスごとに単独に行うか間接的に評価する必要がある。ここでは間接的な評価法を用いて評価を行った。

4.1.2 評価法

データ収集システムの基本性能であるデータ収集性能が実際にこのシステムに備わっているか評価を行った。評価法はシステムをバッファの部分で二つの部分に区切り、それぞれの性能を評価し、それを用いて全体の性能を評価する方法を取った。区切った二つの部分の前半はデバイスドライバが CAMAC から受け取ったデータをバッファに毎秒何バイト書き書き込むことができるかというものであり、“ドライバの割り込み処理性能”と呼ぶこととする。対してそれ以後の処理、バッファに書き込まれたデータをディスクへ毎秒何バイト保存できるかというものを“ディスクへのデータ書き込み性能”と呼ぶこととする。これら 2 つの性能によって全体の収集性能が決定されることになるが、このとき割り込み性能方が悪ければ割り込み性能からの影響を強く受け、逆に書き込み性能方が悪ければ書き込み性能からの影響を強く受ける。

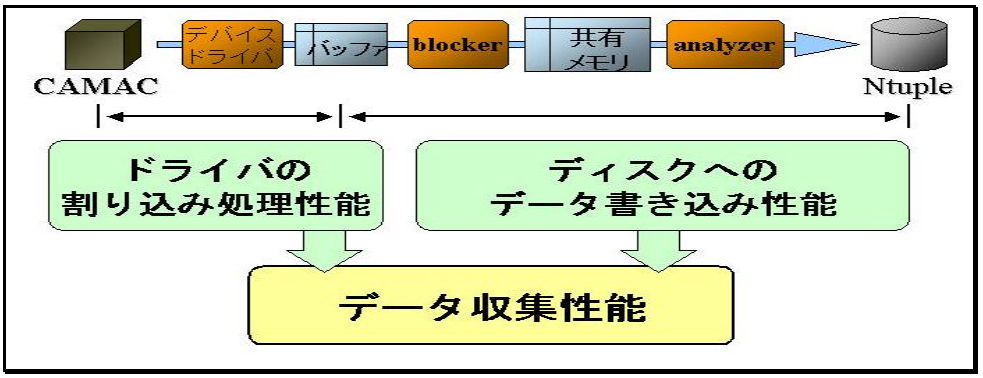


図 20：評価方法

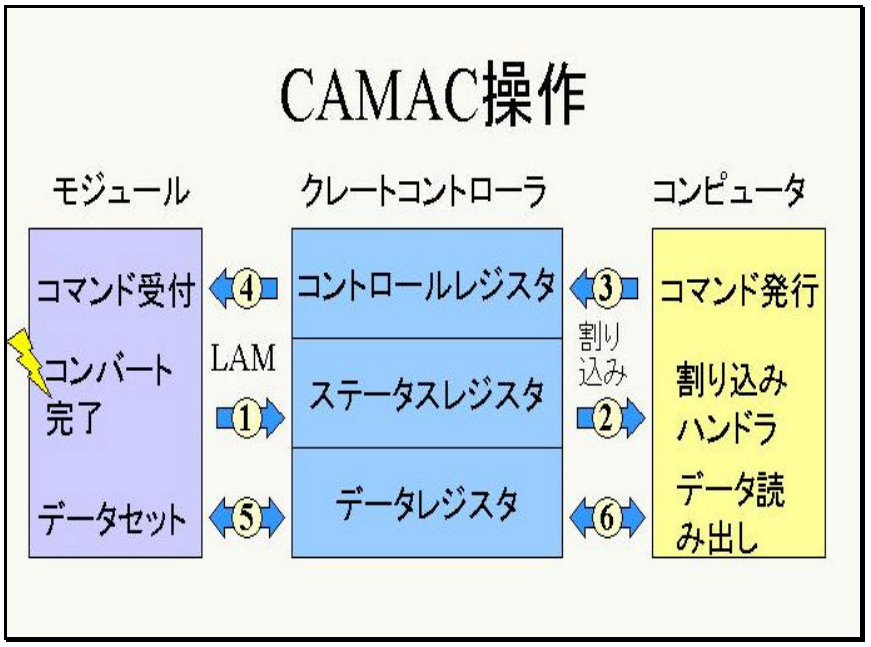


図 21：CAMAC 操作

割り込み処理性能

割り込み処理性能を考える上で、ドライバが行うCAMAC 操作を考えてみると。NIM等のシグナルがモジュールに入るとモジュールはデータをコンバートし、コンバートの完了と共にLAM(Look At Me)を発生する。これを受けたクレートコントローラはドライバに対して割り込み要求を出す。ドライバは NAF コマンドの発行をおこない、コマンドはクレイトコントローラを通ってモジュールに渡さる。

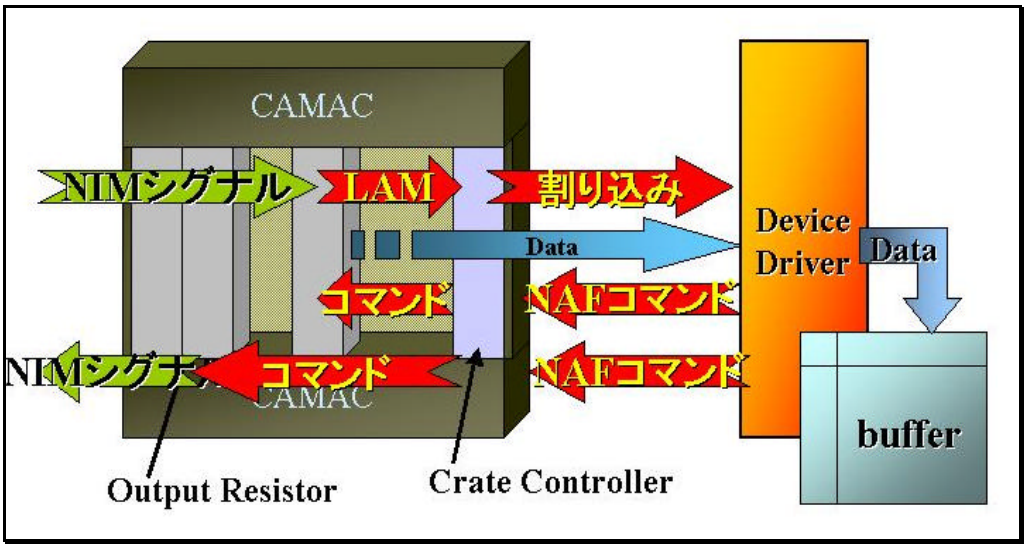


図 22：割り込み処理

モジュールはそのコマンドによってデータを DataWay にセットし、クレートコントローラへ渡される。これをデバイスドライバは読みに行き、得られたデータをバッファに書き込んでゆく。このようなサイクルで指定した全てのモジュールのアドレスからデータを読み出してゆく。そして、それが終了するとドライバは Output Register に対して終了 NIM シグナルを出す NAF コマンドを発行し、これを受けた Output Register は指定の NIM シグナルを出力する。以上のように入力シグナルが入り、終了のシグナルか出力されるまでの所要時間をデバイスドライバの割り込み処理時間とし、これを用いてい以下のような式によって割り込み処理性能を定義した。

$$\text{割り込み処理性能 [byte/sec]} = \frac{2 \times \text{Words}}{\text{割り込み処理時間}}$$

式 1

また、実際の測定においては割り込み処理性能は CAMAC に入れるトリガーシグナルと Output Register からの終了シグナルをオシロスコープで測定した。

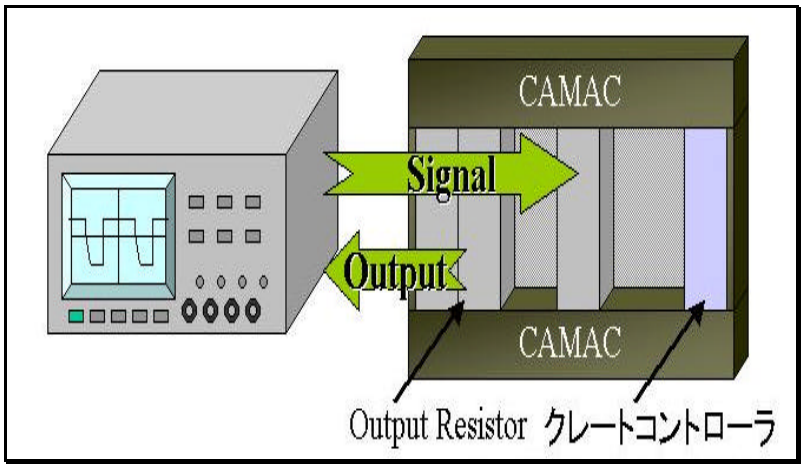


図 23：割り込み処理時間測定模式図

書き込み処理性能

次に“データ書き込み性能”について説明を行う。デバイスドライバと“analyzer”(disk I/O 処理プロセス)にそれぞれ Event を一つ処理するごとに 1 増加する変数を持たせ、これらの増減を比較し、この式のようにその割合を取ったものをデータ書き込み効率と

定義した。これはデバイスドライバが測定したイベントを disk に保存した割合を示した量といえる。

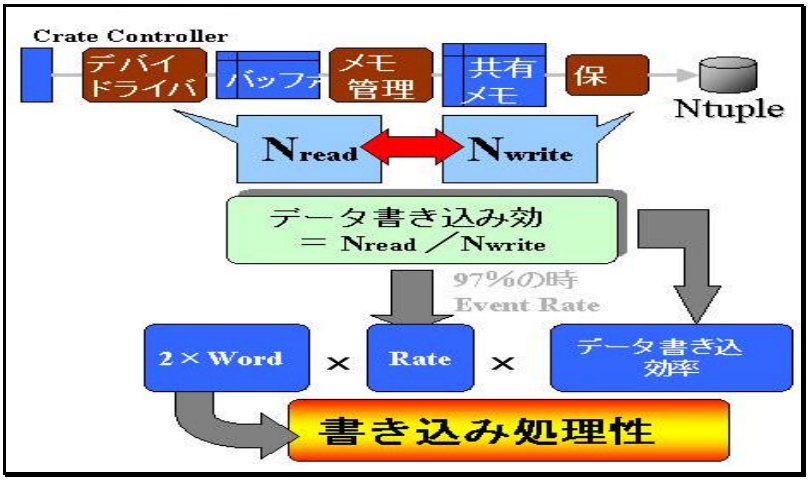


図 24：書き込み処理性能

ここで、この量が 97%を下回る時の word 数と Event Rate を用いて先ほどの ‘割り込み処理性能 ”と似たこのような式にあてはめて “データ書き込み性能 ”とした。

書き込み処理性能[B/sec]= 2×Words×Rate×データ書き込み性能 式 2

4.2 測定セットアップ

以下は性能評価のために行った測定のセットアップとなる。

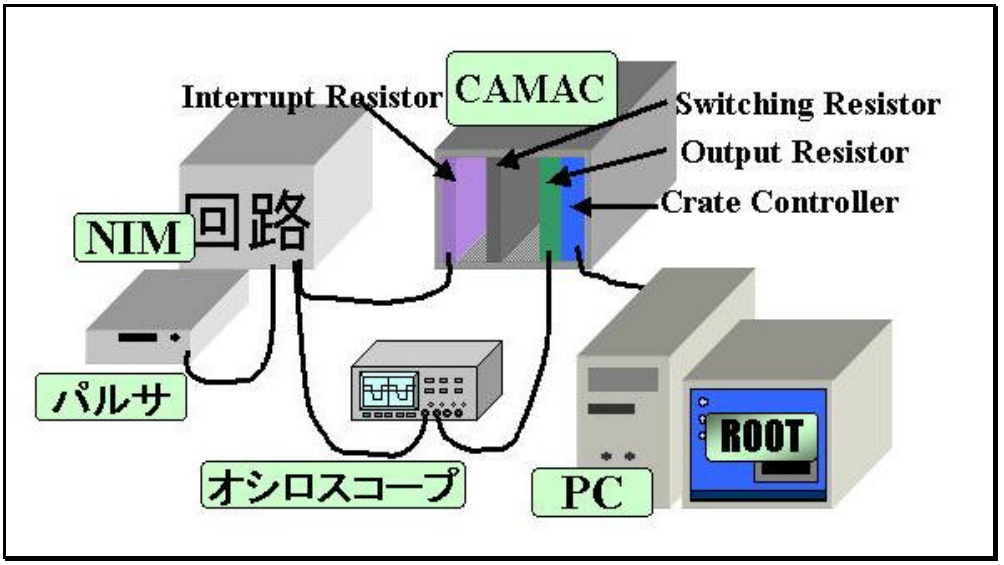


図 25：setup

ここで割り込み処理性能、書き込み処理性能を求めるためにそれぞれ割り込み処

理時間とデータ書き込み効率を測定するが、これらは同時に行えるので以下のようなセットアップになった。また、CAMAC モジュールのデータ変換時間を考慮しなくても良いように今回はSwitching Register を使用した。また、トリガーシグナルはパルサを用い、Event Rate を自由に変更し測定が行えるようにした。**ここで割り込み処理時間**は CAMAC に入れるトリガーシグナルとOutput Register からのシグナルをオシロスコープを用いて測定し、書き込み性能は PC に残るデータを元に測定した。測定はword 数、Event Rate を変えながら１点につき10 回行い、結果としての値はそれらの平均を用いた。

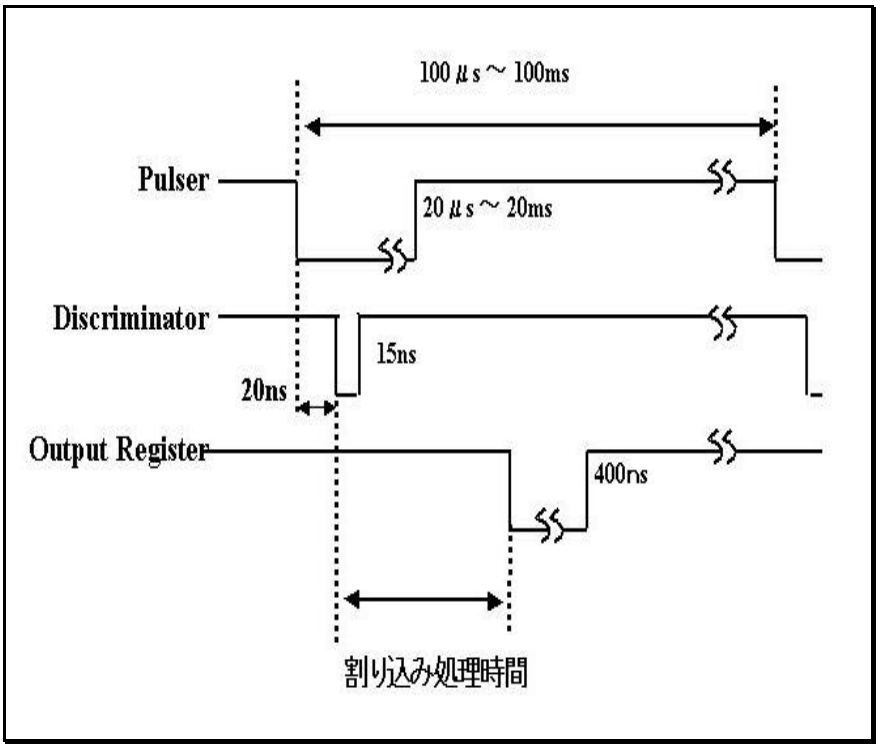


図 26：入出力信号

4.3 測定結果

4.3.1 割り込み処理性能

結果として、初めにデバイスドライバの割り込み処理時間を示す。

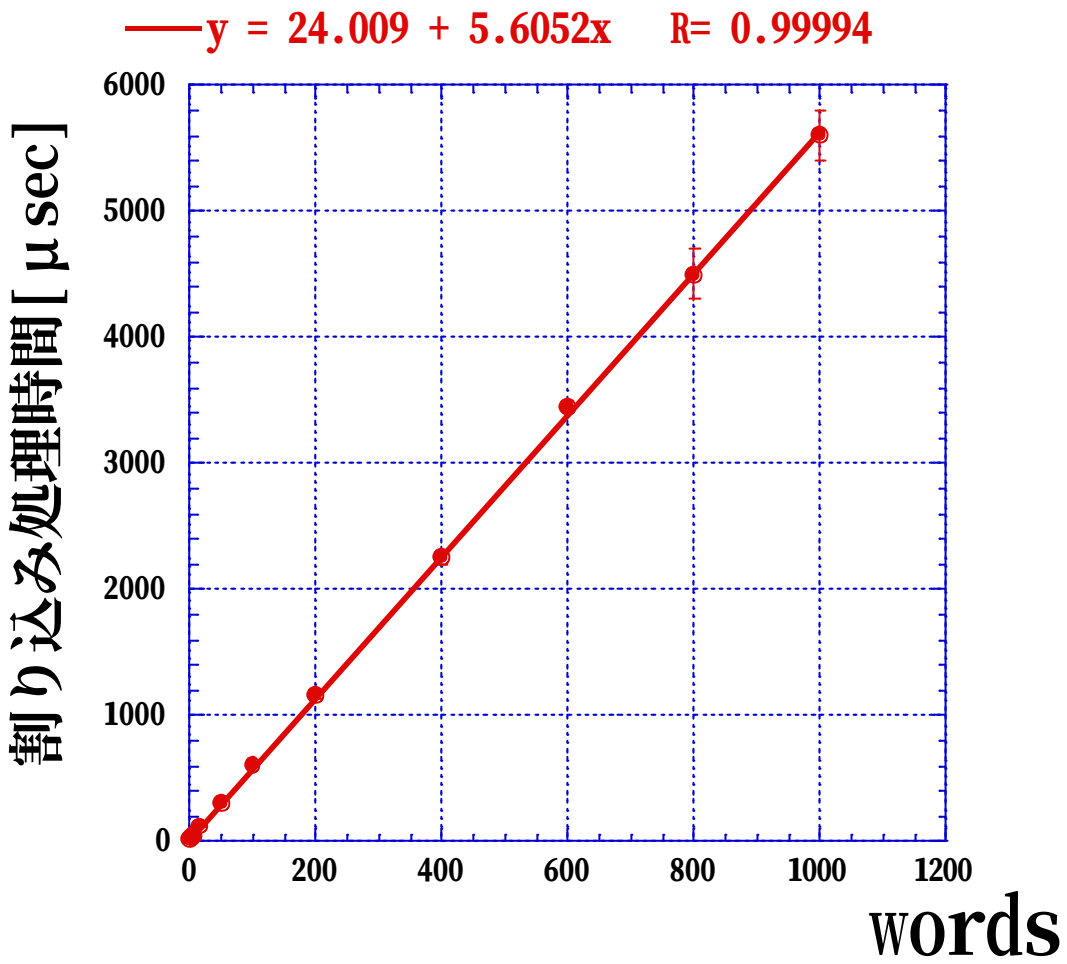


図 27：割り込み処理時間

横軸に word 数、縦軸に測定した割り込み処理時間をプロットし、直線で近似した。その結果が右の式のようになり、定数項の部分は OS・Linux の割り込み応答時間に相当する。また、5.6×words の部分は 1word 読み込みにかかる CAMAC access の Single Action の処理時間に相当する。

この各点の割り込み処理時間を用いて式１によって求めた割り込み処理性能は図 28 のようになった。ここで横軸は word 数、縦軸は割り込み処理性能となっている。

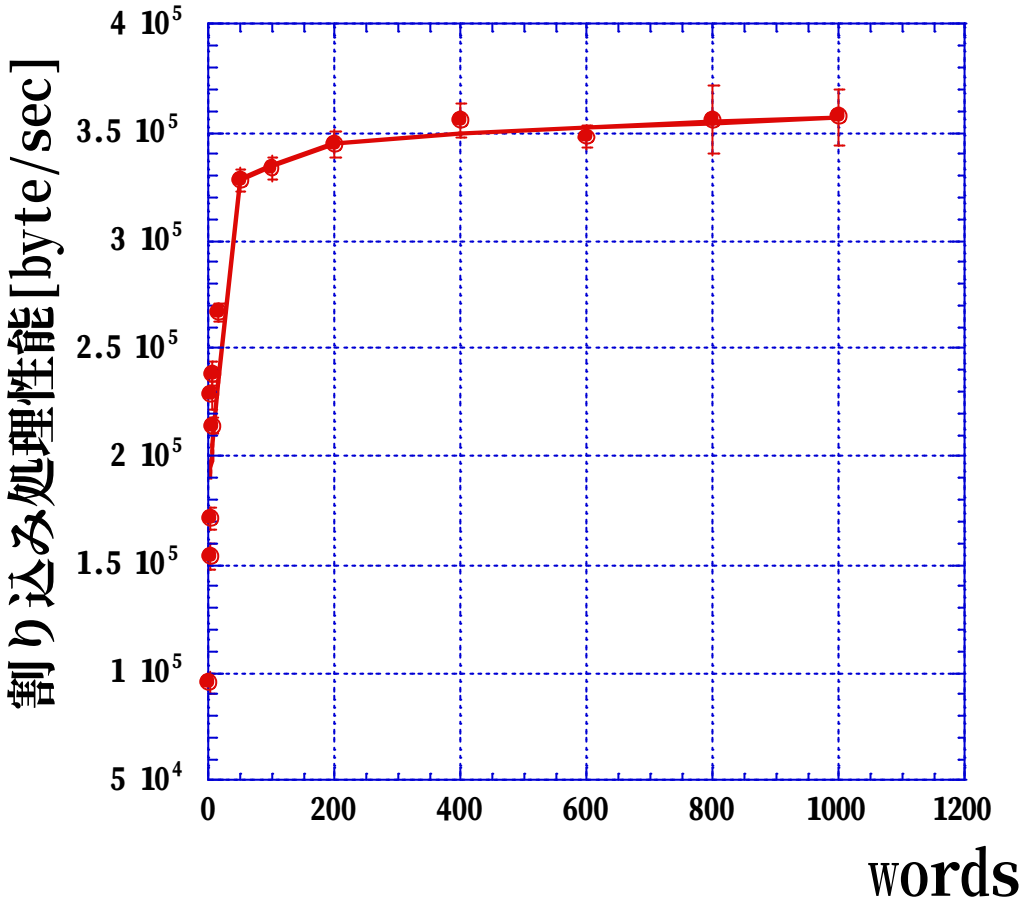


図 28：割り込み処理性能

これより50words から1000words までの各点の平均を取ることにより、割り込み処理性能として、

346.0±3.0[kB/sec]

を言う結果を得ることができた。

4.3.2 書き込み処理性能

次にデータ書き込み効率を示す。

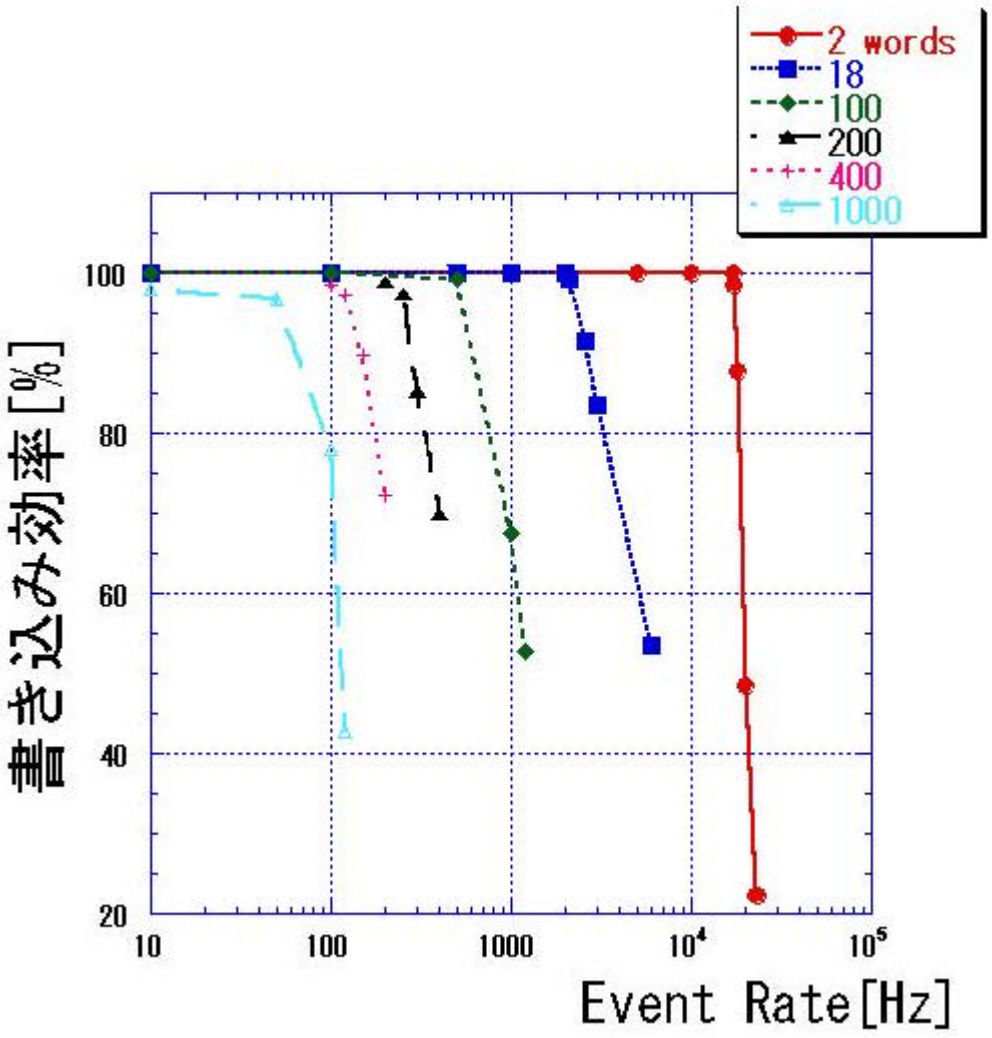


図 29：書き込み処理効率 これは横軸 Event Rate [Hz]、縦軸データ書き込み効率[%]となって、この図 29 より各 word で、いくらの Rate まで 100%の測定が可能かを理解することができる。ここで、本研究の目的の一つである測定環境 18words、100Hz の場合においては十分な効率を持つことが分かった。

さらに、この効率が 97%となる Event Rate を出し、式２を用いてデータ収集性能を求めると図 30 のようになる。

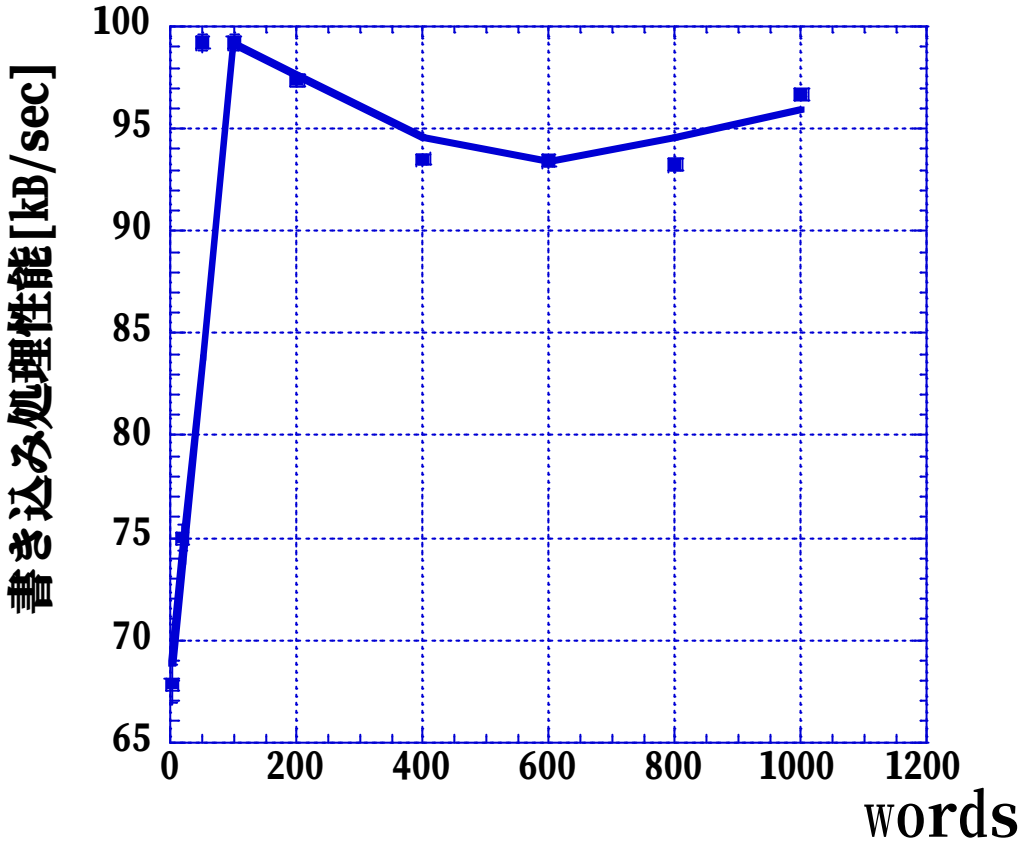


図 30：書き込み処理性能

ここで横軸は word 数、縦軸は割り込み処理性能となっている。
これより50words から1000words までの各点の平均を取ることにより、書き込み処理性能として、

96.1±0.7[kB/sec]

を言う結果を得ることができた。ただし、この値は常にこのレートでデータが送られた場合であり、データが発生しない期間があった場合、共有メモリにデータは保存されその後に処理をされるので、実際に取れるレートはこの値よりもさらに良くなる。
例えばシンクロトロンのように加速に数秒、事象発生（衝突・スピル）に数秒といったスピル構造をもつ場合など考え、ここで加速に3 秒、スピルに1 秒とした場合は書き込み性能は 3 倍になったかのように見える。（実際には性能は向上はしてはなく、それに相当する性能があるというだけである。）

4.4 データ収集性能

測定の結果として以下のような結果になった。

割り込み処理性能 = 346.0 ±3.0 [kB/sec]

書き込み処理性能 = 96.1 ±0.7 [kB/sec]

これをもとに考え、性能の低いものが全体としての性能を決定してしまうので本研究で開発を行ったデータ収集システムのデータ収集性能としては以下のようになる。

データ収集性能 ＝ 96.1～346.0 ±0.7～3.0 [kB/sec]

こらは CAMAC に対する Single Action での性能であり、スタンドアローンで使用されてある事も考えると十分な性能であるといえる。

第 5 章 運用

KEK 実験(T479)における運用法

最後に実際にこのデータ収集システムが使用された実験について簡単に述べる。実験は昨年（2000 年）の 12 月 KEK・PS・T1 カウンタにおいて行われた。実験名は T479 である。PS のスピル構造は 3 秒間に 1 秒の spill があり、この時のトリガーRate は 1spill に 100 回程度のものであった。また、1 トリガーに対し18words のデータを呼び出していたので、システムに求められる性能としては 1.2kB/sec となり、これは十分測定可能な値である。

実験は 2 週間行われ、その期間中に2,424,000 カウントの Event が測定されましたが、これのデータ書き込み効率は 100%であった。また、オンラインモニタについても98.2±0.7%の効率を得ることができた。

第 6 章 今後の発展

本研究での開発は CAMAC・Linux・ROOT を用いた事により大きな成果が上げれたといえる。ここで、今後のこのデータ収集システムの発展を考えると、以下の５つが上げられる。

- デバイスドライバはその CAMAC Crate によって変更する必要がまだ残っている。これをより汎用性をもたせるためにモジュール化を行う必要がある。
- Word 数などの変更時にユーザーは環境ファイルを書き換える必要があるがこれを自動化することにより、さらなる作業能率の向上を図ることができる。
- ROOT のヒストグラムクラス・ファイル保存クラス等を使用しているが、逆にこのデータ収集システムを ROOT の一つのクラスとしてしまえばさらに作業能率や汎用性の向上が期待される。
- 本研究では OS に Linux を用いているがそのリアルタイム性を求める場合、RT-Linux というものが存在し OS 割り込み要求時間が 5μsec 程削減でき、割り込み処理時間は短縮される。
- データ収集性能を決めているコンピュータのハードウエア的要因にはハードディスクのI/O 速度がかかわっていることが予想され、これの調査と改良を行うことによりデータ収集性能の向上が期待される。

第7章 まとめ

開発及び測定の結果、CAMAC に対する Single Action (Read)を行う場合は

データ収集性能 ＝ 96.1～346.0 ±0.7～3.0 [kB/sec]

という結果が得られ、**本研究で開発したデータ収集システムには以下のような性能があり、本研究の目的を達成できたことが確認できた**。

- **高エネルギー原子核実験グループにおけるテスト実験の**典型的な環境として上げられる、100Hz の Event Rate において 18words のデータを測定可能な**データ収集性能を有する**。
- オンライン測定・解析・オフライン解析において解析ソフト“ROOT”を用いた fitting・統計処理等の有利な解析が数コマンドで可能な作業能率を有する。

謝辞

本研究を行うにあたって様々な面で貴重なご指導、助言をいただいた三明康郎教授、江角晋一講師に深く感謝の意を示します。
また、データ収集システムのドライバと基礎となるシステムを開発された東北大学助教授岩佐 直仁氏と本研究における貴重なアドバイスをいただいた韓国 Yonsei 大学 HongJoo Kim 氏に感謝の意を示します。
また、本研究やその他様々な面において助言や協力をいただいた佐藤進助手、加藤純雄技官、中條達也氏、清道明男氏、鈴木美和子氏、相澤美智子氏、圷雄太氏、箱崎大祐氏、小野雅也氏、鶴岡裕士氏に簡単ですが感謝の意を示します。

参考文献

[1]ROOT　　An Object Oriented Data Analysis Framework
Rene Brun & Fons Rademakers/

[2]Technics for Nuclear and Particle Physics Experiments　Wiiliam R. Leo/

[3]Data-acquisition system for a target multifragmentation experiment
with large solid angle detectors,　Y.Tanaka et.al,/
Nuclear Instruments and Methods in Physics Research A 425 (1999) 323-331

[4]Performance of CAMAC data acquisition system under Linux,
Y.Tanaka, M.Haseno, Y.Nagasaka, M.Nomachi et.al/
Presented at Xth IEEE Real Time Conference(RT97)
Proc. 10th IEEE Real Time Conf., 1997, p535

[5]**物理学辞典　/倍風館**

[6]Linux **デバイスドライバ　山崎康広/ライリー・ジャパン**　1998

[7]CNU C Compiler　**遠藤俊徳/秀和システム**　1999

[8]**標準** Red Had Linux **リファレンス**　David Pitts **他/インプレス**　2000

Program List

以下に本論文中で名前が上がった主なプログラムを載せる。

[1]blocker.c	**共有メモリーの管理プログラム**
[2]dc_int.c	**割り込み処理記述プログラム**
[3]analyzer.c	ASCII・Ntuple **ファイル保存プログラム**
[4]mapon.c	**スタンドアローンオンライン表示プログラム**
[5]daqcl.c	**ネットワークオンライン表示プログラム**

KUROQ

Version 1.0.3

- Quick Guide -

Version 0.2

Editor: Y. Kuroki

January 4, 2001

Contents

ご感想、ご意見お待ちしております。

kuroki@utkhii.px.tsukuba.ac.jp

1. INTRODUCTION

1.1 FEATURE

マルチタスク･マルチユーザーOS の Linux 上で動作。

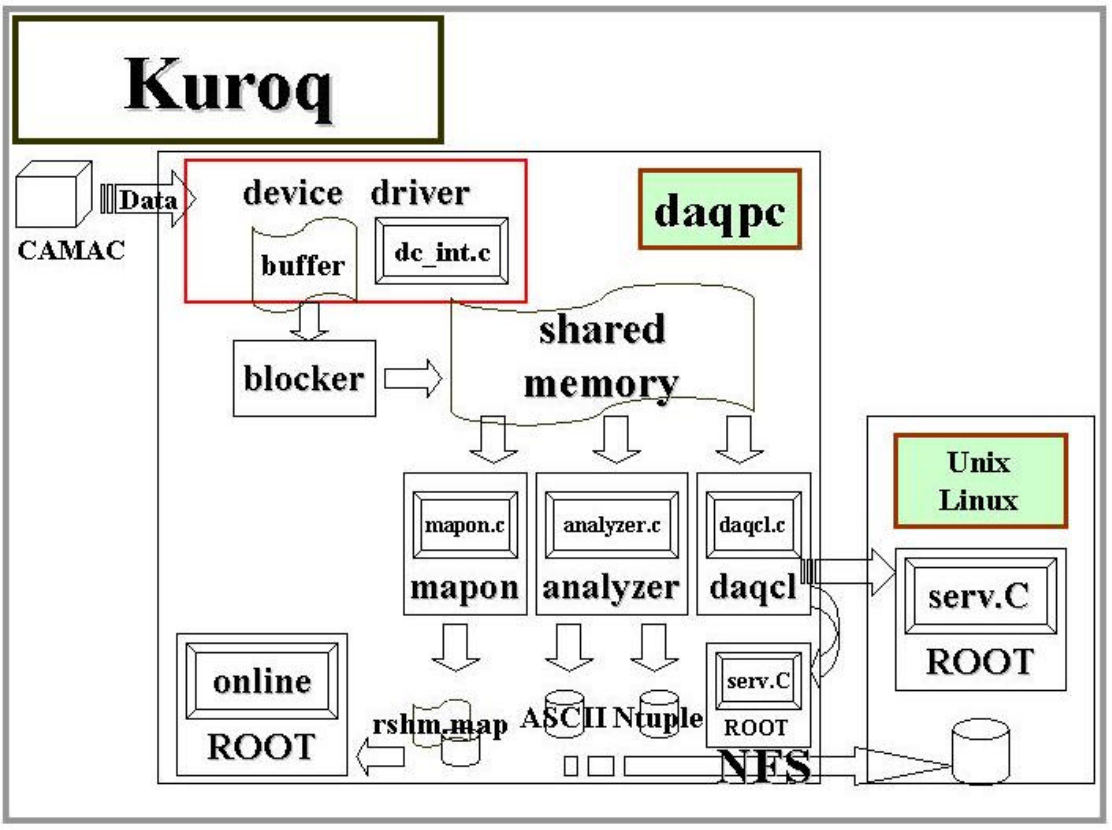
ROOT によるオンラインモニタリング、解析が可能。　（要マシーンパワー）

NETWORK 上にあるマシーン（Linux,Unix,Win 等）でのオンライン表示が可能

DAQ 本体の性能：24.0+5.6×works [　μsec]　(1 work = 16bit)

DiskI/O を含む性能：　96.1~346.0±0.7~3.0［kB/sec］

1.2 OVER VIEW



高速な割込みを実現するため、データ収集プロセスはデバイスドライバーの形でカーネルに組み込まれています。 CAMAC からの DATA はカーネルに組み込まれたドライバーが 64kwords の buffer に書き込まれます。これを blocker と呼ばれるブロック化プロセスが 8 連の 16kBit に区切った共有メモリーに落としていきます。このとき、デバイスドライバーの 64 キロワードのバッファーが一杯になった場合には、ブロック化プロセスから読まれる

までデータ収集を停止するようになっています。その後、データは 3 つのプロセスによって読み出されます。主となるものは“ analyzer ”と呼ばれているもので、ASCII、Ntuple 等の形式でデータをディスクに保存できます。また、“ daqcl ”と呼ばれるプロセスは読み出したデータをヒストグラムにまとめ、ネットワークを通して他のマシーン（linux,Unix,Win 等）や、自らにデータを送りオンラインでデータ表示を可能にしています。また、他の方法によるオンラインモニターとして、“ mapon ”と呼ばれるプロセスが存在しています。どちらのモニターを使用するかは容易に選択できます。また、測定によって得られたデータファイルは NFS 等によって効率よく他のマシーンへ転送できるようになっています。

2. CHANGES

KEK テスト実験用に調整を行ったものをもとに VersionUp します。（近日中）
“kuroq”正式リリースは version-1.0.2 からとなります。
以前のプロトタイプ DAQ 等からの VersionUp 情報は
　DAQPC：/daq/src/version
をご覧下さい。

3. QUICK START

3.1 Default

この DAQ のデフォルトは以下のようになっています。
LAM：N=1　A=0
ADC：N=3　A=0　　TDC：N=4　A=0
の計 2ch、2word (1word = 16 bit)のデータを測定する設定になっています。

4.2 Make

デフォルト設定のままでよいのなら quick-start が行えます。”
src/”において”make”とコマンドしてください。これで実行 File が生成されました。
これで DAQ の使用は可能となります。
コマンドについては「6.COMMANDS」を参照してください。

4. SETTING UP

DAQ の使用環境は以下のとおりになります。

3.1 Hardware

- CRATE CONTROLLER ＆ INTERFACE BOARD
  (株)東洋テクニカ製 CC／7700 ， CC／PCI 型
- PC
  PC には AT 互換機を用いてください。
  IRQ 等の関係で PCI ボードの差し替えが必要になるときがあります。
  “ /proc/pci ”を参照して確認をしてください。
  （注：pci には書かれていなくても実際には使用されている IRQ もありますので注意しましょう。最も良いのは Windows の設定で確認することです。）
- OS
  OS はRedHat Linux6.2J(kernel 2.2系)をインストールしておいてください。
  また、SMP(Symmetric MultiProcessor,*1)には対応していません。
  Dual CPU のPC を使用するときは非SMP-Kernel をBoot時に選んでくだい。
  例えば RH6.2 ではデフォルトで Lilo に“ linux ”,“ linux-up ”があります。
  ここで、“ linux ”は SMP、“ linux-up ”は非 SMP となります。

3.2 Install ROOT

この DAQ は ROOT を使用した解析を前提としたものです。(ROOT を導入していない環境では動きません。）ここでは ROOT の INSTALL 方法については RH6.2J においての例を上げますが、その他の OS、version 等での install は ROOT のホームページ(http://root.cern.ch)を参照してください。

例）RH6.2J　kernel2.2.14-5.0　gcc version 2.91.66 (egcs-1.1.2 release)

ホームページから gcc の version をあをせて以下の File を落としてきます。

ftp://root.cern.ch/root/root_v2.25.03.Linux.2.2.14.tar.gz

“/cern”のようなディレクトリを作り、そこに”tar”等のコマンドで解凍します。
後は自分のホームにある”.bshrc”等に以下のような設定を加えるだけです。

```
# ROOT
export ROOTSYS=/cern/root
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ROOTSYS/lib
export PATH=$PATH:$ROOTSYS/bin
```

ためしに”%root”と打ってみてください。ロゴが出てきましたか？

3.3 DAQ-Source

DAQ のソースを入手してきます。（注：現段階ではこの DAQ は NET 配布をしていません。ソース等を希望する方は直接この文章の著者に問い合わせて下さい。）
以下は著者のグループでの例です。

DAQ 専用 PC において・・・
“/daq/src”からソース(new.tgz)を自分の workspace にコピーしてきます。
“cp /daq/src/new.tgz . ”
次にそれの解凍をします。
“tar zxvf new.tgz”
すると”kuroq-1.0.2/”が作成されます。
この下に作られる File については後の「FILES」で説明しています。

3.4 Make Dvices

これは、kuroq を install しようとしている PC に初めて install するときに行うコマンドです。（すでに”/dev/PCccI7700”が存在していれば必要はありません。）PCI ボードのデバイスを”/dev”に作らないといけませんので以下のようなコマンドを”src/”で行ってください。ただし root(管理者)の権限が必要となります。

```
#make devices
```

これで”/dev/PCIcc7700”という File が作られます。

3.5 Chmod +s

通常、ドライバーのインストールは管理者(root)の権限が必要となってきます。しかし、以下の 3 つのコマンドにスパーユーザービットを立てれば平ユーザーでのシステム構築が可能となります。（注：セキュリティー的に問題がある場合は各自の判断で行ってください。）

- /sbin/insmod
- /sbin/lsmod
- /sbin/rmmod

スパーユーザービットの立て方は、以下のコマンドで行います。

chmod +s file

ここで file は各コマンドとなります。

5. EDITTING FILES

主な設定は“dc_int.c”と“analyzer.c”,“mapon.c”,“daqcl.c”の4つで行います。目的に応じて変更してください。

また、2word(デフォルト),16word用のFileは”src/2ch”、”src/16ch”以下にありますので参考にしてください。

5.1 dc_int.c

このFileには割り込み処理ルーチンが記述されています。

測定開始や終了、CAMACモジュールのクリアー、またLAM（LookAtMe）が指定したモジュールで発生した時にどのような処理を行うかを設定しています。ですので、この部分を変更することによりある程度の処理は可能です。

しかし、このFileはドライバの一部となりカーネルに組み込まれますので幾つか制限をうけています。

- Kernelが持っているオブジェクトのみが呼び出し可能です。（sdtio等は不可）
- 無限ループを作らない。
- 変数はstatic型。（auto型で巨大な配列ができると危険）

以上のような制限と注意を守らないとOSのハングアップにつながる恐れがありますので注意してください。また、その他にも異常動作を起こすとカーネルパニックの引き金になります。/var/log/messages等を見てデバックを行ってください。

データを読み出したいModuleのステイションNo.（N），チャンネルのアドレス（A）等を変更するだけの場合は、頭の”#define ADC 3”等の数字を変更しさえすれば大丈夫です。（注：整数でなくてはいけません。）

例：
```
#define ADC      3     ( N )
#define ADC_A    0     ( A )
#define ADC_RF   0     ( Read Function )
#define ADC_CF   9     ( Clear Function )
#define ADC_CA   0     ( Clear Address )
```

また、割り込み処理は” static int dc_int(---) { ”以下に書かれています。読み出すデータの数を増やしたい時などはここを変更してください。

例：ADC,１wordを読み込む場合は…

```
buffer[0]=1;
camac_NAF(ADC,ADC_A,ADC_RF);
```

```
buffer[1]=camac_read16();
```

“buffer[x]”に各データは収められます。
ただし、”buffer[1]”にはここで設定する buffer の数(最大のｘ)を指定しなければなりません。
次に“Camac_NAF(---)”は CAMAC の NAF コマンドを発行するものです。
最後に”camac_read16( )”によってデータを読み込んでいます。

以下はここで使用される主な関数になります。

- **void camac_CNAF(int C, int N, int A, int F)**
  クレート番号 C[0-7],ステーション番号 N[1-23],サブアドレス A[0-15],ファンクション F[0-31]を送り、CAMAC サイクルを起動させる。
- **void camac_NAF(int N, int A, int F)**
  CNAF と同様、ただし C=0 と固定されている。
- **void camac_write16(short data)**
  次の CAMAC コマンドのために 16Bit のデータを送る。
- **void camac_write24(long data)**
  次の CAMAC コマンドのために 24Bit のデータを送る。
- **short camac_read16( )**
  前の CAMAC コマンドで得られた 16Bit のデータを返す。
- **long camac_read24( )**
  前の CAMAC コマンドで得られた 24Bit のデータを返す。
- **printk(const char *format ,...)**
  stdio 内の関数 printf()と似たもので、使用法は同様。
  Kernel 内に有るので使用が可能となっています。
  出力は/var/log/messages です。ただし、頻繁に使用しするとシステムの負荷が増大するので注意してください。

“ dc_int.c”と違いこの File と以下の 3 つは kernel に組み込まれることは有りません。そのため、通常の関数が使用できます。この DAQ では主に ROOT のクラスを使用するように作られていますので、ROOT で行えることは基本的には実行可能です。

5.2 analyzer.c

この DAQ におけるデータの主な流れは ASCII、Ntuple 形式 File へ保存されています。ここではその形式等が記述されています。
以下は word 数をかえる場合に変更が必要となる部分です。

```
/* Creat Ntuple */
TNtuple *ntp1 = new TNtuple("ntp1","DAQ1","n:adc1:tdc1");
  //  Ntuple の宣言をしています。最後の"n:adc1:tdc1"を変更をしてください。
  //  n はイベント番号。adc1、tdc1 はそれぞれ Ntuple での変数名となります。

int a1,t1;                          //  変数宣言です。" a "はADC ," t "はTDC。
a1 = shmp->buffer[analys_block%MAX_BLOCK].data[pos+2];
                          // "pos+2"は " buffer[ 1 ] " のデータを受け取ります。
t1 = shmp->buffer[analys_block%MAX_BLOCK].data[pos+3];
                          // "pos+3"は " buffer[ 2 ] " のデータを受け取ります。
fprintf(fp,"%d\t%d\n",a1,t1);       //  ASCIIの出力をしています。
                  //" %d " は十進数表示、" ¥t " は「Tab」を意味しています。
ntp1->Fill(event,a1,t1);
        //  Ntuple の Fill です。
```

5.3 daqcl.c

Network 経由で Data を送る形式等を記述してあります。
再表示スピードは

```
    int updata_rate = 100;
```

のように設定さえています。これは、100 event に一回転送することを示します。この設定の数が大きくなればデータの読み落としは少なくなりますが、再表示に時間がかかってしまいます。

以下は転送先の変更時に関わる部分です。

```
/*  Open connection to server */
TSocket *sock1 = new TSocket("130.158.105.104", 9090);
                  //  ここでIP等によって転送先を指定しています。
                  //  localhostとすると自分にもDataが返ってきます。
```

また、word 数を変えるときは基本的に”analyzer.c”と同じですが、ROOT のクラス“ TH1 ”を使用している事に注意してください。Ntuple の使用はサポートしていません。

5.4 mapon.c

スタンドアローンでオンラインモニターを行う設定の記述をしています。
これも word 数を変えるときは基本的に”analyzer.c”と同じですが、”daqcl.c”同様 ROOT のヒストグラムクラス“ TH1 ”を使用しています。

6. Make

先ほど編集した File を実行可能な File にしなくてはいけないので“ Make ”を行います。
エラーが出なければ結構です。（作者の力不足のため警告は出てしないます。）
次にドライバーのインストールがうまく行くかテストを行います。

```
% make install
```

正常に組み込まれれば以下のような表示があります。

```
/sbin/insmod PCIcc7700
cat /proc/devices
character devices:
1    mem
   -   省略   -
70   PCIcc7700
```

デフォルトでうまく動かないときは以下の二点を確認してください。

1. MAJOR
   “ /proc/devices ”を参照して指定している No.が使用されていないかを確認。
2. IOPROT
   “ /proc/ioport ”を参照して使用されていないかを確認

もし、使用されていたら空いているところを使用してください。

7. COMMANDS

“kurokq”、“local_daq”、“basic_daq”の 3 つのコマンドパッケージを用意しました。

- “kurokq”はネットワーク経由のオンラインモニター
  （スタンドアローンでの使用も可能です。）
- “local_daq”はローカルオンラインモニター
- “basic_daq”は非オンラインモニター

7.1 kuroq

〔使用手順〕

I. 先ず一つ目の term で“setup”を実行。

II. 別の term を開き、オンラインモニターの受け取り側の“serc.C”を ROOT で起動し"Server: Waiting DAQ start"を表示させておきます。

III. はじめの term で”start”を実行して測定を開始します。
＜ 測定が終了したら自動的に測定はとまります。＞

IV. 測定を続ける場合は、オンラインモニターの受け取り側の質問に答え II.と同様に"Server: Waiting DAQ start"を表示させておきます。

V. はじめの term で”start”を実行して測定を開始します。
＜ 以下は同様の手順で行います。＞

- **Setup**
  DAQ のセットアップコマンドです。
  ドライバーのインストール、blocker の起動、後述の clear を行います。
- **Start**
  測定開始コマンドです。
  注：これは後の”9.Macro”でも述べていますが、start をする前に受け取り側”serv.C”を起動させ、"Server: Waiting DAQ start"という表示を出しておいてください。
  先ず全体の clear を行ったあと、RunNo.と測定するイベント数を聞いてきますので入力してください。イベント数入力後に測定開始となります。また、デフォルトはイベント数は 10000 に設定してあります。デフォルトで使用したい場合は何も入力せずにリターンを押してください。設定したイベントをとり終えると自動的に測定が終了します。

- **Stop**
  測定中止コマンドです。
  通常は”analyzer”がイベントをとり終えると自動的に測定が終了しますが、その途中で終了したい場合はこのコマンドを使用します。
  ただしこのコマンドで終了した場合、正しく保存されるデータは ASCII 形式の物のみとなります。
- **Clear**
  バッファー、共有メモリー、ステータス等を初期化します。

7.2 local_daq

〔使用手順〕

I. **先ず一つ目の term で“ setup ”を実行。**
II. **ROOT の term とモニターが開きますので、ROOT の term で”.! ./start”を実行して測定を開始します。**
III. **測定中オンラインモニターを使用したいときは**
”.x [MACRO]”の実行をします。
＜　測定が終了したら自動的に測定はとまります。＞
IV. **測定を続けたいときも ROOT で”start”を実行して測定を開始します。**
＜　以下は同様の手順で行います。＞

- **Setup**
  DAQ のセットアップコマンドです。
  ドライバーのインストール、blocker の起動、clear を行い ROOT を起動させます。Online 用の Window も開きますので使いやすいように並べ替えてください。
- **Start**
  測定を開始します。このコマンドは setup で起動した ROOT 上で行ってください。（RCINT を使用します。形式は例のとおりです。）

```
root[1] .! ./start
```

  また、他の機能は Kuroq の start と同じです。
- **Stop**
  測定中止コマンドです。Kuroq の stop と同等。
- **clear**
  Kuroq の clear と同等。
- **start_net**
  start と同等ですが、daqcl（ネット転送）も同時に起動します。

7.3 basic_daq

〔使用手順〕

I. 先ず一つ目の term で“ basic_start ”を実行。
測定状況を表示する term が開き、測定が開始されます。

II. 測定を続けたいときは”basic_restart”を実行。
＜ 以下は同様の手順で行います。＞

- basic_start
  DAQ のセットアップならびに測定を開始するコマンドです。
  基本機能は他の setup+start と同等です。
- basic_restart
  測定の再スタートを行うコマンドです。
  基本機能は他の start と同等です。
- clear
  基本機能は他の clear と同等です。

8. DATA & LOG

[log]

測定開始時刻、各プログラム（ana,net,map）の測定カウント数、測定終了時刻が”log/run.log”にログとして記録されます。これは 3 つのコマンドパッケージ（kuroq、local_daq,basic_daq）に共通のものです。また、測定途中でコマンド”stop”によって測定を中止した場合は測定終了時刻は記録されません。

[DATA]

保存されるデータ形式はすべて共通で、ASCII、Ntuple（ROOT）形式になります。この二つの保存の書式は“ analyzer.c ”で記述されています。

また、オンラインモニター用のデータの流れは以下のような違いがあります。

[kuroq]

共有メモリーから ROOT の TH 1（ヒストグラム）に Fill されたデータは ROOT のクラス TSocket によって外部（localhost も可）に転送されます。

[local_daq]

共有メモリーから ROOT の TH 1（ヒストグラム）に Fill されたデータは ROOT のクラス TmapFile によって ROOT（Cint）の方にデータを転送しています。この際、共有メモリー、ハードディスクスワップ等を使用しています。

9. MACRO

保存された ASCII はもちろん Ntuple、NETWORK 経由によって転送されたデータ、オンラインモニターで転送されたデータ、すべてにおいて ROOT での解析が可能です。以下のような Macro-File を必要に応じて編集してください。これらは中身を見ていただければ分かるかと思いますが、ROOT の基本的な知識があれば容易に変更は可能です。詳しくは ROOT のホームページをご覧下さい。

ASCII → hist.C
Ntuple → ntuple.C
Net → serv.C
Map → online

10. NETWORK

Linux 上で実行されているため、便利な Linux(UNIX)の技術を使用できます。

10.1 SSH

ssh、Secure Shell（http://www.ssh.com/about/company/index.html）を使用して遠隔地からの DAQ の操作、設定の変更、解析の全てが容易に行えます。

注：ssh によってオンラインモニターを使用する場合、ssh でログインした後以下のようなコマンドを打って、環境変数を変更しておいてください。

%export DISPLAY=(自分の IP):0.0

10.2 NFS

NFS によって DAQ の DATA 領域を共有とした場合、FTP、SFTP 等の面倒な作業が必要なくなります。これによって他のマシーンで解析を行う場合でも、その作業能率は格段に向上されます。

11. TOOLS

開発の途中でテストを兼ねて作ったプログラムを幾つか用意しています。

11.1 TDC-TEST

市販の TDC-TESTER を使用して TDC の時間分解能を測定､テストするものです。実験の前などにお使いください。

11.2 PORING DAQ

“ Poring 方式 ”による DAQ です。

動作確認に使用してください。これは g++でコンパイルしてください。

11.3 SCAN-MODULES

各 Module の動作確認のためのプラグラムです。“ scan-sta.c ”は CRATE CONTROLLER のステータスを読み込みます。“ scan-x-a0.c ”は各 Module の A=0 に備わっている Function をわりだします。（X の返答により認識しています。）

11.4 Org-test

Iwasa さんが作られたテストプログラムです。特に“test0.c”は基本的な動作確認をするときには最適です。gcc でコンパイルしてください。

11.5 stdcom

“COM”と出ている window と数字がたくさん表示されている window が出てきます。“COM”からはドライバーにいくつかのコマンドを送れます。数字が出ているほうはオンラインモニターになっているようです。著者自身この 2 つについては完璧に理解していないので今回はここまでにしておきます。

11.6 C10_2

10 進数から 2 進数に変えるための物です。

12. REFERENCE

***1：SMP**

“Symmetric MultiProcessor**”の略で, 対称型マルチプロセッシングともいう。複数の** CPU **が基本的に同等なものとして振る舞うことができるマルチプロセッシングの方法。仕事を区別することなく，各** CPU **に与えられるため，アプリケーションソフトウェアの設計は容易といえる。**

ご感想、ご意見お待ちしております。

kuroki@utkhii.px.tsukuba.ac.jp

Macro File List

KEK 実験で使用したマクロファイルの一部を例として載せる。

[1]raw3.C	時間分解能表示
[2]slew-dpro.C	slewing 補正
[3]slew-d3.C	slewing 補正後の時間分解能