# 修士学位論文

# ATLAS実験ミューオントリガー用<br>大型Thin Gap Chamber検査システムの開発

神戸大学大学院自然科学研究科物理学専攻

林　健一

1999年2月

**概 要**

世界最大の重心系衝突エネルギーを持つ大型陽子・陽子衝突型加速器LHC (Large Hadron Collider)が、2005年の実験開始を目標として、欧州素粒子物理学研究所 (CERN)で計画されている。我々ATLAS Japan ミューオングループは、LHCに設置されるATLAS測定器のミューオントリガー検出器 Thin Gap Chamber(TGC)の設計・製作を担当しており、2000年春からの大量生産を予定している。LHC実験では、これまでの実験では経験したことのない膨大なバックグラウンドが予想される。そういった厳しい環境下に於いてミューオントリガー検出器は、優れた時間分解能や検出効率などを高い信頼性で要求される。これまでの研究によりTGCの基本的な設計及び製作方法は確立しており、TGCはその要求に応えうることが確認されている。しかし、製作される各々のTGCについて動作・性能を保障するために、その検査は不可欠である。TGCは、測定部面積1 $m^2$ 以上の大型のものを約500台という規模で量産するため、その検査は専用の大規模な設備を必要とし、そのデータ収集 (DAQ)システムもまた、高い処理能力を持った専用のものを構築しなくてはならない。そこで、このTGC検査設備のためのDAQシステムの設計を行った。設計は高い拡張性を得ることを目的とし、ネットワークに分散したPCを利用したシステムを採用した。また、予備実験にこのDAQシステムを実装し、その性能を評価した。

# 目 次

# 1 序論

## 1.1 LHC 計画

LHCは大型陽子陽子衝突型加速器 (Large Hadron Collider) の略で、スイス、ジュネーブにある欧州素粒子物理学研究所 CERN(Conseil Européen pour la Recherche Nucléaire) において、TeV エネルギー領域での素粒子物理研究を目指し、2005 年の稼働を予定して建設が進められている。

LHCでは未発見粒子の Higgs 粒子の探索が、LEP-II*での探索上限である約 90 GeV から約 1 TeV までの全ての領域をカバーすることができるほか、超対称性粒子や現在知られていない全く新しい相互作用などの発見も期待される。

LHC 加速器の主要パラメータを表 1 に示す [4]。このうち、TeV エネルギー領域での素粒子反応を捉えるために特に重要になるのは、重心系衝突エネルギーとビーム・ルミノシティである。また、ルミノシティを稼ぐためにビーム衝突間隔は非常に短い時間になっている。

表 1: LHC 加速器の主要パラメータ

| | |
|---|---|
| 主リング周長 | 26,658.87 m |
| 陽子ビームエネルギー | 7.0 TeV |
| ルミノシティ | $10^{34}\,\mathrm{cm}^{-2}\mathrm{s}^{-1}$ |
| 入射エネルギー | 450 GeV |
| バンチ間隔 | 25 ns |
| バンチ当りの陽子数 | $1 \times 10^{11}$ |
| バンチ数 | 2,835 |
| ビーム/ルミノシティ寿命 | 22/10 時間 |
| 二口径双極電磁石 | 1,232 台 |
| 二口径四極電磁石 | 368 台 |
| 双極電磁石磁場 | 8.36 Tesla |
| バンチ長さ | 75 mm |
| 衝突点でのビーム半径 | 16 mm |
| ビーム衝突角度 | 200 mrad |
| バンチ衝突当りの陽子衝突数 | 19 |

加速器は現在 LEP が使っている周長 27 km のトンネル内部に設置される。加速器は 8 対称で、8 アーク部および実験に使う 4 衝突点と、入射やビームダンプの

*CERN で稼働中の電子・陽電子衝突型加速器 LEP(Large Electron Positron collider) で、現在重心系衝突エネルギーを 189 GeV で運転している

ための 4 交差点からなる。その衝突点の 1 か所に日本の実験グループが参加する ATLAS 測定器が設置される。

LHC の最大ルミノシティーにおいては、これまで高エネルギー物理実験が経験したことのない膨大な放射線の中で希少なイベントを選別しなくてはならない。LHC 実験の成否は各測定器の性能にかかっているということになる。

## 1.2 LHC の物理

LHC がめざす物理には Higgs 粒子の探索、超対称性粒子の探索、第 3 世代のクォークである t クォーク、b クォークの物理等がある。ここではその中でも最も重要な Higgs 粒子について述べる [1, 2, 4, 5]。

Higgs 粒子の生成で最大の断面積を持つプロセスは、衝突する陽子から放射された 2 つのグルーオンが t クォークのループを介し融合し、Higgs 粒子を作るプロセス (図 1(a)) である。次に断面積の大きいプロセスは、衝突する陽子からのクォーク、反クォークがそれぞれ W、Z 粒子を放射し、これらが融合して Higgs 粒子を作る場合 (図 1(b)) である。Higgs 粒子の質量が比較的軽い場合、重要になってくるのが次の 2 つのプロセスで、1 つは t クォーク対を伴うプロセス (図 1(c)) で、もう 1 つは W、Z 粒子を伴うプロセス (図 1(d)) である。

Higgs 粒子の探索では、Higgs 粒子の質量によって断面積、崩壊過程および崩壊幅が違うので、探索する崩壊モードも異なってくる。質量別に表 2 にまとめる。

表 2: Higgz 粒子を探索する主な崩壊モード

| | Higgs 粒子の質量範囲 (GeV) | 探索する主な崩壊モード |
|---|---|---|
| a | $80 < m_H < 120$ | $H^0 \to \gamma\gamma$ |
| b | $120 < m_H < 2m_Z$ | $H^0 \to Z^0Z^{0*} \to \ell^+\ell^-\ell^+\ell^-$ |
| c | $m_H > 2m_Z$ | $H^0 \to Z^0Z^0 \to \ell^+\ell^-\ell^+\ell^-$ |
| d | $m_H > 800$ | $H^0 \to Z^0Z^0 \to \ell^+\ell^-\nu\nu$<br>$H^0 \to W^+W^- \to \ell\nu jj$ |

a. $80\,\mathrm{GeV} < m_H < 120\,\mathrm{GeV}$
$H^0 \to \gamma\gamma$ のモードは分岐比が小さいので、バックグラウンドをいかに落せるかが重要になってくる。連続したバックグラウンドの上に狭い質量ピークを観測するために、$\gamma$ についてエネルギーと角度の高い分解能が要求される。また、LHC が低ルミノイティーで走っている時ならば、図 1(c)、(d) で生成された Higgs 粒子の b クォーク対への崩壊も探索可能である。この場合一緒に生成された W 粒子や t クォークが指標とできバックグラウンドを減らすことができる。

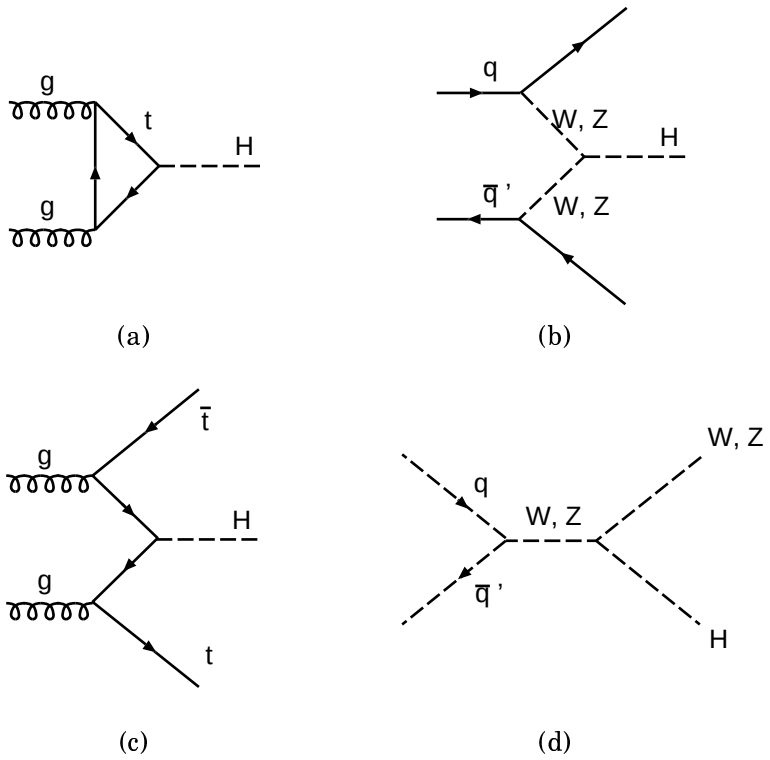


図 1: Higgs 粒子の生成。(a) グルーオン-グルーオン融合、(b)W-W,Z-Z 融合、(c)$t\bar{t}$ クォーク随伴生成、(d)W,Z 粒子随伴生成。

b. $120\,\mathrm{GeV} < m_H < 2m_Z$
この質量領域で最も信頼性のあるモードは $H^0 \to Z^0Z^{0*} \to \ell^+\ell^-\ell^+\ell^-$ ($Z^{0*}$ は off-shell) である。バックグラウンドとしては $q\bar{q} \to Z^0Z^{0*}/Z^0\gamma^* \to \ell^+\ell^-\ell^+\ell^-$ があるが、それほど多くない。

c. $m_H > 2m_Z$
$H^0 \to Z^0Z^0 \to \ell^+\ell^-\ell^+\ell^-$ モードを用いる。このモードはバックグラウンドが少なく、LHC の最も得意とするモードである。主なバックグラウンドとしては、$q\bar{q} \to Z^0Z^0 \to \ell^+\ell^-\ell^+\ell^-$ があるが、それほど多くはない。

d. $m_H > 800\,\mathrm{GeV}$
Higgs 粒子の質量が大きくなると、その崩壊幅は急激に広がるので、信号とバックグラウンドの区別が難しくなってくる。さらに、Higgs 粒子の生成断面積も、質量が大きくなるにつれて減少していく。したがって、この領域での Higgs 粒子探索には分岐比の高いプロセス、$H^0 \to Z^0Z^0 \to \ell^+\ell^-\nu\nu$ や $H^0 \to W^+W^- \to \ell\nu jj$ を用いて、できるだけイベント数を稼ぐことが必要がある。これらの分岐比は $H^0 \to Z^0Z^0 \to \ell^+\ell^-\ell^+\ell^-$ に比べて前者は約 6 倍、後者は約 150 倍ある。さらにこの領域では、W、Z 粒子が融合して Higgs 粒子が作られる過程 (図 1(b)) に注目して、終状態の 2 つのクォーク・ジェットを捕らえること

により、バックグラウンドを減らすことができる。

## 1.3 ATLAS 測定器

ATLAS 測定器 (図 2) は LHC の高ルミノシティ運転時の高い粒子頻度のもとでも、e、$\gamma$、$\mu$、jet、missing$E_T$、b-tagging など多くのシグナルをバランス良く取り出し、精度の良い測定を行なうことを目的とした汎用測定器である。その規模は、半径約 12 m、長さ約 44 m の円筒形で、全重量は約 7000 トンにも及ぶ。ATLAS 測定器は主に次のものから構成されている [1]。

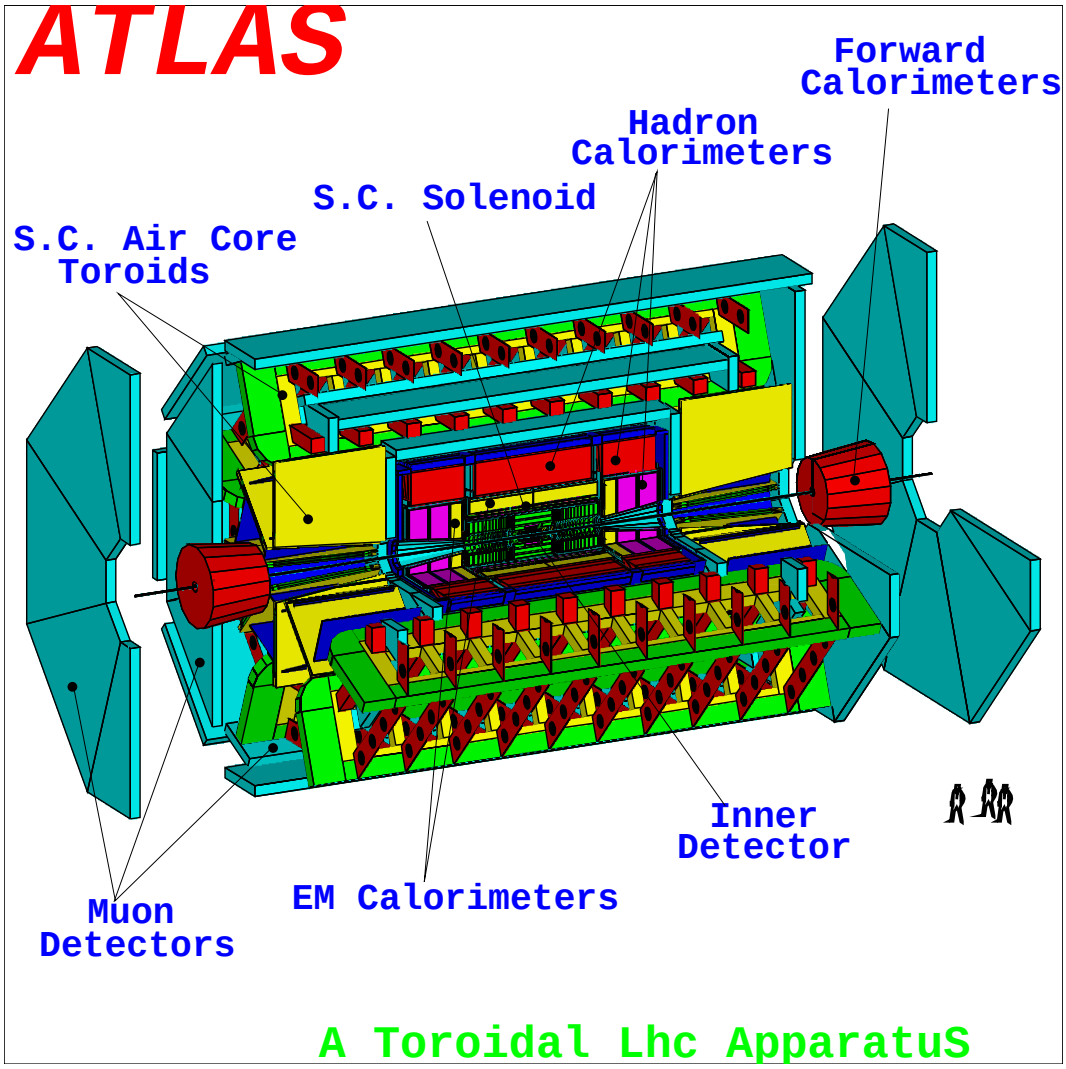


図 2: ATLAS 測定器 [3]

1. **内部飛跡検出器**
   この検出器の主な機能は、荷電粒子の飛跡認識、運動量測定、反応点測定および電子の識別である。内側にはピクセル検出器、シリコン・ストリップ検出器からなる半導体検出器が置かれ、外側には連続飛跡検出器としてストロー検出器が多層に積層されている。電子の識別を行なうため、それぞれのストロー検出器の間には遷移放射を引き起こす物質が挿入されている。これらの飛跡検出器は中心磁場 2 テスラの超伝導ソレノイドの内部に置かれる。運動量分解能は横運動量が 500 GeV の粒子に対して 30 % である。
2. **カロリーメータ**
   カロリーメータの役割は、電子や $\gamma$ 線のエネルギーと位置測定、ジェットのエネルギー測定などである。従来の測定器のカロリーメータに比べて非常に高い粒子頻度と放射線量の環境で稼働しなければならない。内側に電磁カロリーメータ、外側にハドロン・カロリーメータがある。電磁カロリーメータは耐放射性に優れ、安定性の良い液体アルゴン・カロリーメータである。ハドロン・カロリーメータは領域によって 2 種のタイプに分かれる。バレル部 ($|\eta^\dagger| < 1.6$) は鉄の吸収体とタイル状シンチレータおよび波長変換ファイバーからなる。エンドキャップ部 ($1.5 < |\eta| < 3.2$) の領域は放射線量が高いので、銅の吸収体と液体アルゴンを組み合わせたカロリーメータとなっている。
3. **ミューオン検出器**
   LHC 実験においてミューオンの検出が果たす役割は大きい。目的とする物理的事象の崩壊モードの多くが荷電レプトンを含んでいる。その中でもミューオンは他の粒子と違い測定器の中で反応することなく外にまで突き抜ける。したがって、ミューオン検出器は最外部に設置されており、ミューオンを高い S/N 比で検出することができる。さらに、ミューオンの運動量測定を精度良く行なうため超伝導空芯トロイド磁石が採用されている。このトロイド磁石は、粒子の多重散乱を最小に抑え、ミューオンの測定可能な $\eta$ 領域を広く取ることができる。バレル部のトロイド磁石は 8 個の超伝導コイルから成り外径 20 m 長さ 26 m の巨大な構造物である。エンドキャップ部も前後方それぞれ 8 組の超伝導コイルから成っている。これらのトロイド磁石は、偏向力がバレル部で 2.5 $\sim$ 4 Tm、エンドキャップ部では 7 Tm になる。運動量の精密測定を行なう検出器は MDT (Monitored Drift Tube) と CSC (Cathode Strip Chamber) からなる。また、トリガー専用のミューオン検出器として、バレル部 ($|\eta| < 1.05$) には RPC (Resistive Plate Chamber)、放射線量の高いエンドキャップ部 ($1.05 < |\eta| < 2.4$) には本論文で述べる TGC (Thin Gap Chamber) が置かれる。

以上のように ATLAS 測定器は、ミューオンの検出・測定に優れており標準 Higgs 粒子の発見を第一の目的としている。図 3 に ATLAS 測定器を低ルミノシティで 3

---

$^\dagger\eta$ はラピディティと呼ばれ、定義は $\eta = \frac{1}{2}\ln(\frac{E+P_L}{E-P_L}) \simeq -\ln(\tan\frac{\theta}{2})$ である。

年、高ルミノシティで 1 年 (約 $10^5\,\mathrm{pb}^{-1}$) 動かした時の標準 Higgs 粒子の発見ポテンシャルを示す。

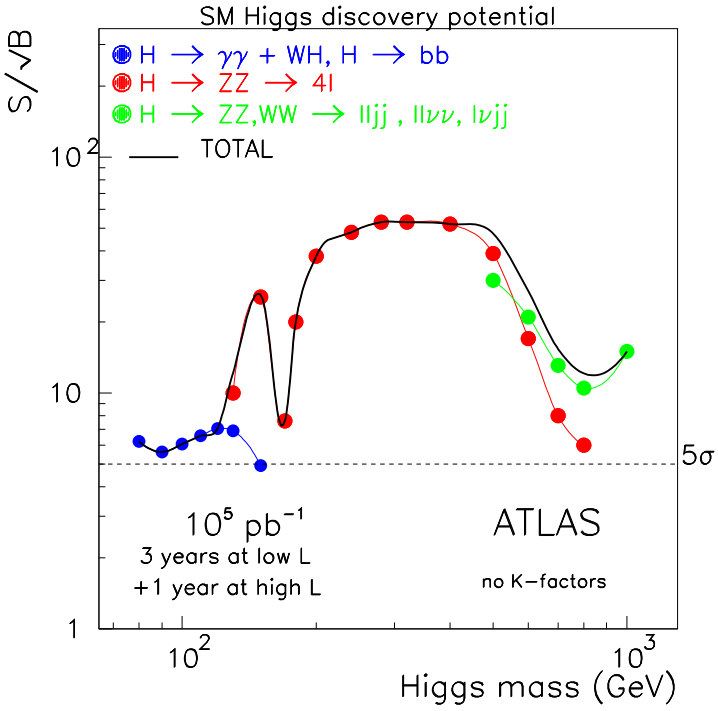


図 3: ATLAS 測定器を低ルミノシティで 3 年、高ルミノシティで 1 年動かした時の標準 Higgs 粒子発見ポテンシャル。[3]

## 1.4 ATLAS 測定器におけるミューオン

LHC 実験では、多くの事象の中から物理的に重要な事象だけを選別することが重要である。そのなかで、ミューオンを用いた散乱事象の取捨選択はトリガーの基本的な技術であり、実験の要となる。

ミューオントリガーチェンバー (図 4) は、通過するミューオンのトロイド磁場による曲がり具合いを測定することにより横向き運動量 $p_t$ を調べ、事象の取捨選択を行なうものである。図 5 にミューオントリガーの仕組みを示す。図 5 の TGC1 は 3 層が 1 組になった 3 重層 (Triplet) になっており、TGC2、TGC3 はそれぞれ 2 層ずつが 1 組になった 2 重層 (Doublet) になっている。RPC1、RPC2、RPC3 はそれぞれ 2 層ずつからなっている。

low-$p_t$(>6 GeV) のトリガーを出す条件としては、バレル部では RPC1 と RPC2、エンドキャップ部では TGC2 と TGC3 の 2 つの 2 重層の間でコインシデンスがとられ、ある幅の window 内に粒子が通過することが要求される。ここである幅の

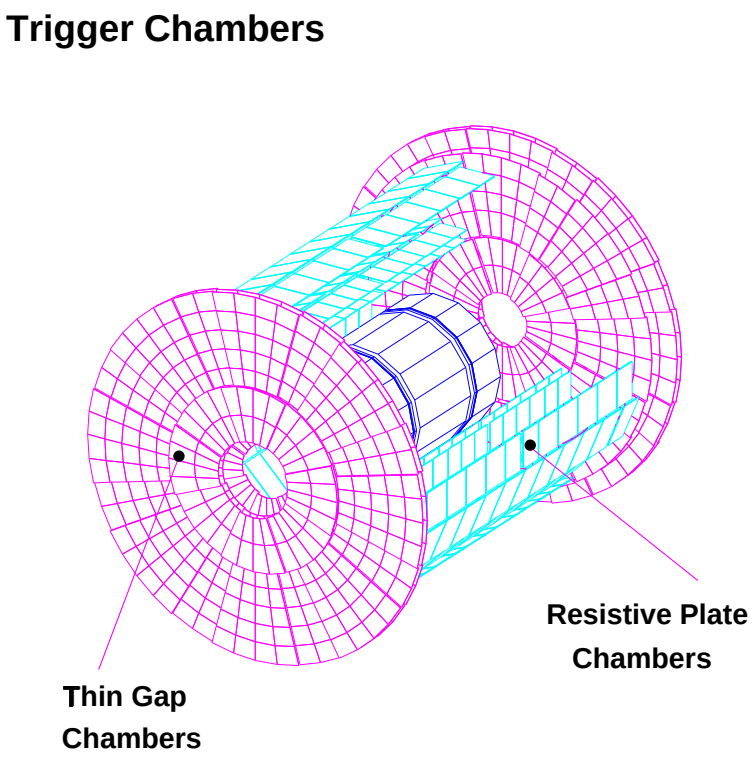


図 4: ATLAS 測定器のミューオントリガーチェンバー [3]

window とは $p_t \approx 6\,\mathrm{GeV}$ のミューオンを 90 % の効率で捕らえることの出来る幅のことを言う。4 層の間のコインシデンスの条件としては 4 層のうち少なくとも 3 層がヒットすることが要求され、この条件が独立に Z、$\phi$ または R、$\phi$ の両方向で満たされる必要がある。

high-$p_t$($>$20 GeV) のトリガーを出す条件としては、low-$p_t$ のトリガー条件に更に、エンドキャップ部では TGC 1 の 3 層のうち少なくとも 2 層が、バレル部では RPC 3 の 2 層うち少なくとも 1 層がヒットすることが満たされなくてはいけない。この場合の window の幅は、$p_t \approx 20\,\mathrm{GeV}$ のミューオンが 90 % の効率で捕らえることの出来る幅である。

TGC の置かれるエンドキャップ部には、陽子・陽子素過程によるジェットやビームとビームパイプ等との相互作用等によって生じるハドロン衝突型加速器特有の高頻度の粒子入射が予想されている。モンテカルロシミュレーションにより見積もられたミュー粒子検出器のエンドキャップ部におけるバックグラウンドレートを表 3 に示すが、シールドの欠損などによってレートが 1 桁ほども上昇することも考えられる [6]。

以上のことなどにより、ミューオントリガー検出器には以下のような要請がある。

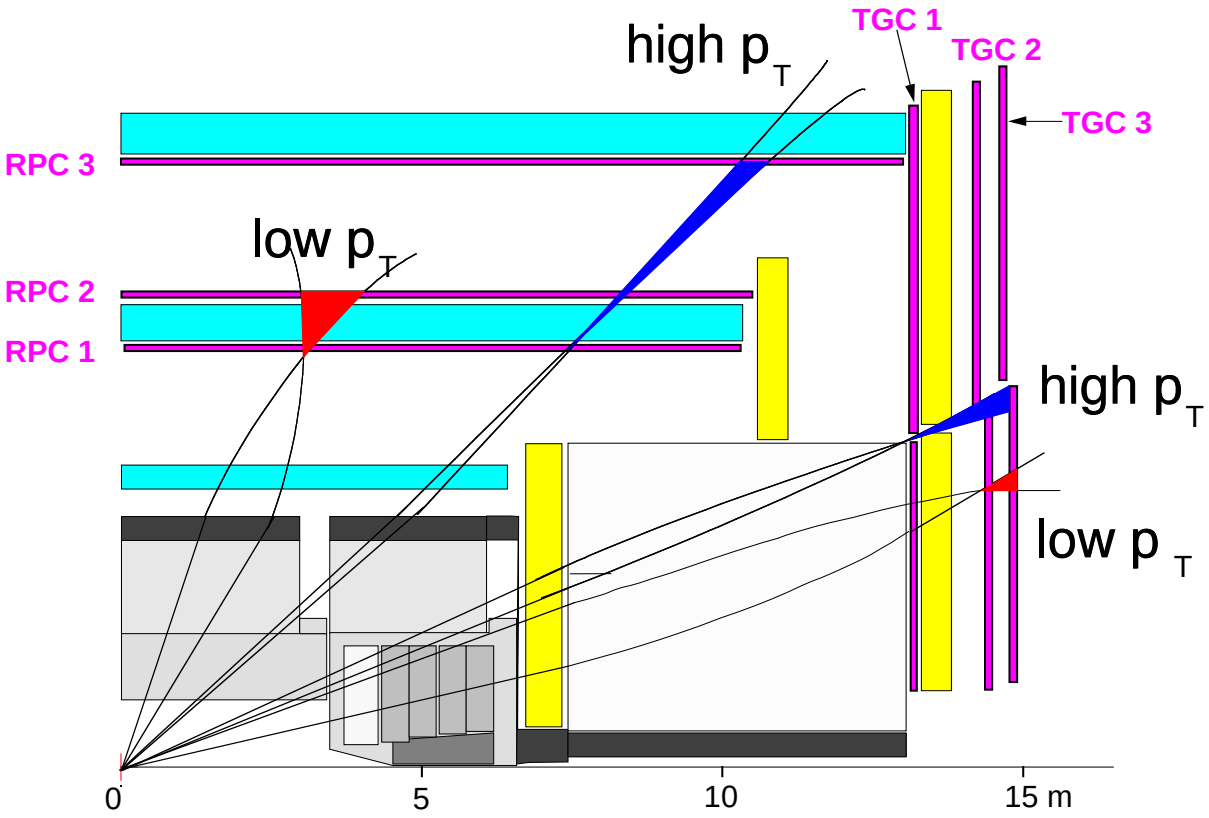


図 5: ミューオントリガーの仕組み。low-$p_t$ に対するトリガーは 2 つの 2 層のトリガーチェンバーのコインシデンスがとられる。high-$p_t$ に対しては、さらにバレル部では 2 層、エンドキャップ部では 3 層のコインシデンスがとられる。影の入った所はカロリーメータとアブソーバーである。正負それぞれのミューオンする飛跡を示している。[3]

- 1 kHz/cm$^2$ 程度の高入射粒子頻度において 10 年程度十分安定に動作すること。
- バンチ識別を行なうため、バンチ間隔の 25 ns よりも小さい Time Jitter [‡]。
- 98 % 以上の検出効率。
- $p_t$(横方向運動量) をトリガー条件に組み込むための、1 cm 程度の位置分解能。

## 1.5 Thin Gap Chamber

まず、ATLAS で使用する TGC の断面図を図 6 に示す。ここでは、基本的な TGC の構造・動作原理と、ATLAS 用 TGC の製作について述べる。

[‡]ここでは、信号到着時間分布のエントリー中 95 %が含まれる最小の時間幅である

表 3: ミュー粒子検出器に対するバックグラウンドレート ($1.44 < |\eta| < 2.3$)

| | n | $\gamma$ | $\mu$ | hadrons$^{\pm}$ | e$^{\pm}$ |
|---|---|---|---|---|---|
| rate(kHz/cm$^2$ $\pm$ %) | $4.1 \pm 3$ | $2.7 \pm 3$ | $2.1 \times 10^{-3} \pm 9$ | $2.8 \times 10^{-3} \pm 7$ | $1.2 \times 10^{-2} \pm 7$ |

### 1.5.1 構造・動作原理

TGC の構造は図 6 で示すように基本的には多線形比例計数箱 (MWPC:Multi Wire Proportioal Chamber) である。TGC は、その名 (Thin Gap Chamber) の通りアノードワイヤー間隔に対して、アノードワイヤーとカソード面との間隔 (gap) が通常の MWPC に比べて非常に狭い (thin) のが特徴である。

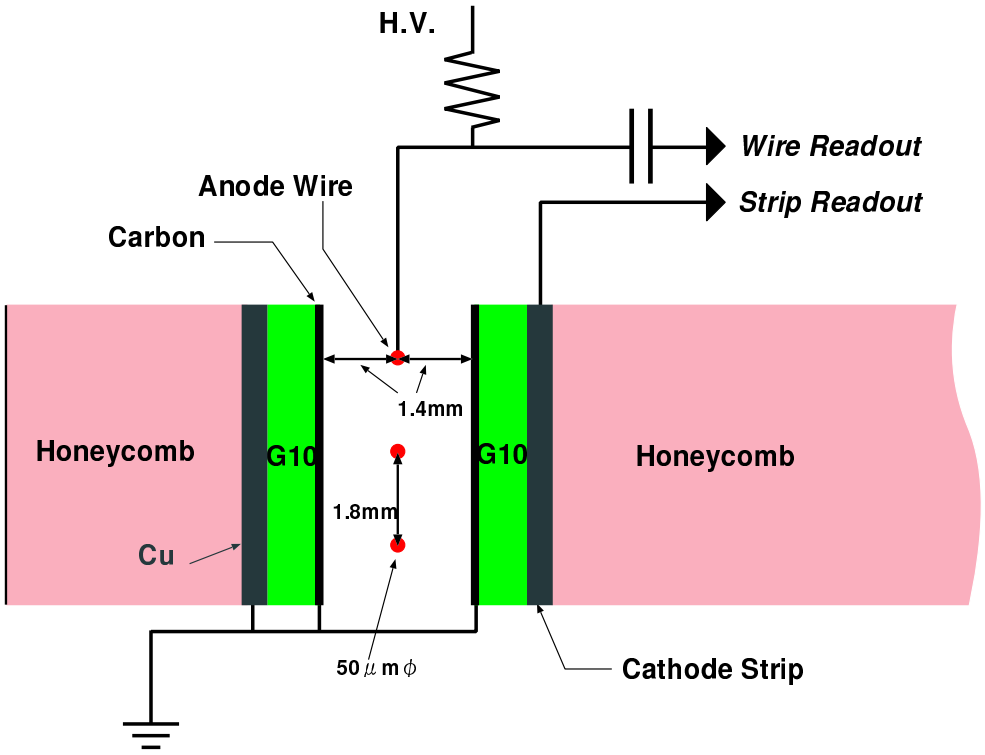


図 6: ATLAS 用 TGC の断面図

TGC の信号が発生する動作原理を述べる [11, 12, 13]。チェンバー内を荷電粒子が通過すると、電離作用によって充填ガスの分子がその飛跡に沿ってイオン化する (1 次電離)。アノードワイヤーには高電圧が印加されており、この 1 次電離により生じた電子とイオンはチェンバー内の電場により、それぞれアノード、カソードに向かってドリフトしていく。このドリフトの間に多数のガス分子との衝突が起きる。イオンは移動度が小さいが、電子は電場によって容易に加速され、大き

な運動エネルギーを持ちうる。電子がワイヤー近傍の $10^4 \sim 10^5\,\mathrm{V/cm}$ を越える強い電場の中に入ると、電子が衝突の間に得るエネルギーがガス分子の電離エネルギーより大きくなり、さらにイオン対が生成され2次電離が起こる。この2次電離過程で作られた電子も電場によって加速され、ガス分子と衝突してさらに電離を起こす。このような反応が連鎖的に続き、タウンゼントなだれという形に発展する。一方、電子と対となって生成された陽イオンは、電子がワイヤーに到達する時間よりもはるかに長い時間をかけ、カソードプレートに向かってドリフトして行く。このなだれ生成の推移を図7に示す。

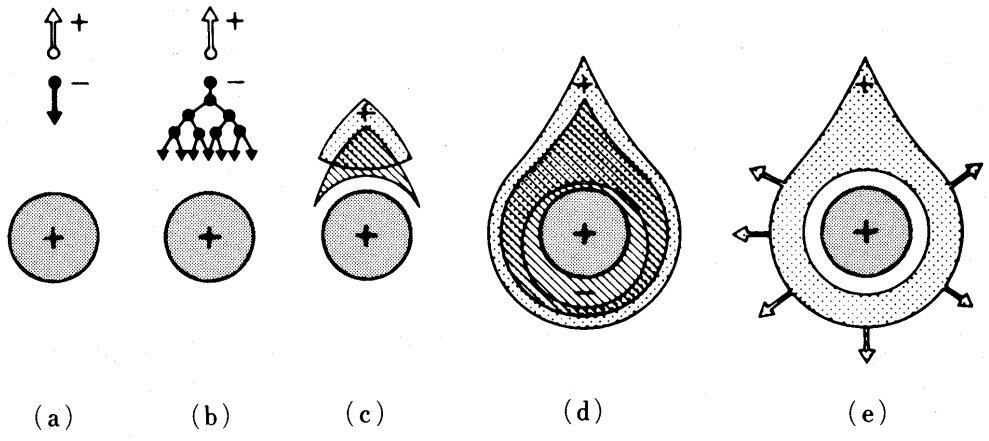


図 7: アノードワイヤー付近でのタウンゼントなだれの推移。(a) 1次電離電子が陽極に向かって移動する。(b) 電場により電子は運動エネルギーを得てさらにガス分子を電離する。ガス増幅が始まる。(c) 電子とイオン雲がお互いに離れる。(d) と (e) 電子雲はアノードワイヤーに向かってドリフトしそれを取り囲む。イオン雲はアノードワイヤーの半径方向に離れていく。[11]

以上の過程で起きた電子と陽イオンの移動により、アノードワイヤーとカソードストリップにそれぞれ電荷が誘起され、これがパルス電流として検出される。

### 1.5.2 ATLAS 用 TGC の製作

TGC は 1996 年に高エネルギー加速器研究機構 (以下 KEK) の東カウンターホール・$\pi$2 エリアで行なわれたビームテスト、K4 エリアで行なわれたエイジングテスト等でミューオントリガー検出器に要請される条件を満たすことが可能であると確認された [7, 8, 9, 10]。

TGC のパラメーターは図6に示した通りで、アノードワイヤー間隔 1.8 mm、gap 1.4 mm である。アノードワイヤーは通常の MWPC で使われるものよりも太い直径 50 $\mu$m の金メッキしたタングステンワイヤーを使い、張力 350 g 重で張られている。カソード面には $\sim 1\,\mathrm{M\Omega/\square}$ の高抵抗のカーボンが塗布されている。カソー

ド面の外側には銅のストリップがワイヤーと直行して並べられている。G10[§]の板を挟んで外側には静電シールドとして銅メッキがされている。充填ガスとしては、$CO_2$+$n$-pentane (50:50) が用いられる。

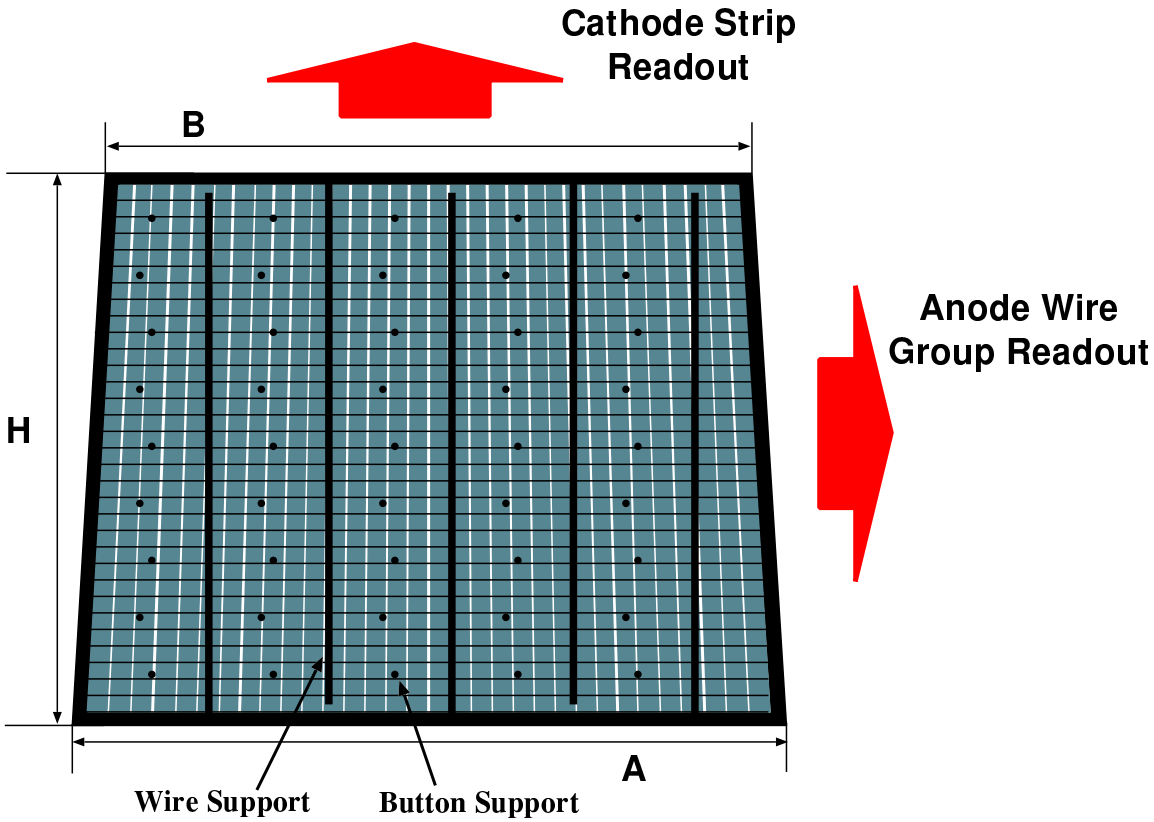


図 8: TGC のチェンバー内の概観図。ワイヤーは横方向に張られておりワイヤーグループの境界に線を引いてある。、ストリップは縦方向に 32 枚に分割されている。黒く示したサポート部は不感領域となる。

チェンバーの平面図を図 8 に示す。チェンバー内にはアノードワイヤーのたるみを補正するワイヤーサポート、カソード面のゆがみを補正するボタン型サポートが取り付けられ、サポート部は不感領域となる。

信号の読み出しは、横方向に張られたアノードワイヤーは数十本をまとめたグループを 1 チャンネルとして信号を読み出す。カソードストリップは縦方向に 32 枚に分割されており、それぞれを 1 チャンネルとして読み出す。読み出し部には、エレクトロニクスとして ASD ボードを搭載する。ASD(Amplifier Shaper Discriminator) は、信号増幅 (Amplifiler) を行い、信号のテール部分をカットし (Shaper)、デジタル信号化 (Discriminator) の 3 つの回路を備えたチップである。

[§]成分:60 % ガラス ($SiO_2$)+40 % エポキシ樹脂

表 4: 日本で製作される ATLAS TGC のパラメーター

| | ユニット数 | | 寸法 | | |
|---|---|---|---|---|---|
| Type | Doublet | Triplet | H (mm) | A (mm) | B (mm) |
| T4 | 96 | 0 | 1957 | 915 | 658 |
| T5 | 96 | 0 | 1769 | 852 | 620 |
| T7 | 192 | 96 | 1250 | 1379 | 1215 |

ATLAS で使用する TGC は 2 層のチェンバーを重ねた構造の Doublet と 3 層構造の Triplet の 2 種類が製作される。表 4 に日本で製作される TGC の寸法と台数を示す。表の H、A、B はそれぞれ図 8 で示される長さである。現在計画されている TGC 製作は、2000 年 4 月から 2002 年末までに Doublet384 台、Triplet96 台の合計 480 台 (1056 Chamber) であり、一週間に 5 台のペースを目標としている。計画では、製作は高エネルギー加速器研究機構 (KEK) で行われ、二ヵ月毎に 40 台が、神戸大学に設置する専用の検査設備へと搬送される。検査に合格した TGC は CERN へ輸送される。

## 1.6 TGC 検査システム

製作された個々の TGC が ATLAS 実験で要請される性能・動作を持つものであることを検査することは不可欠である。初期不良のある TGC は実験に用いることはできない。日本で製作される TGC の検査は主に神戸大学において行われ、その検査項目は以下の通りである。

**エージングテスト** 先ず、高電圧の印加の可否を検査する。これにより、内部のワイヤー切断やカレントパスの有無をみる。充填ガスについては、運用の便をはかり $CO_2$ を用いる。ガスのリークの有無も同時に検査する。

**宇宙線長期テスト** ここでの検査項目は主に TGC の検出効率である。前述のように TGC には高い検出効率が要求されており、検出部全面に渡ってそれを満たしていなくてはならず、この検査は不可欠である。これは連続して約 10 日間行い、長期安定性も併せて検査する。

宇宙線長期テストでは検出効率を求めるために、宇宙線を用いる。宇宙線通過経路の測定のための検出器 Drift Tube を用意し、これにより経路を決定した宇宙線通過を TGC が検出できたかどうかで、宇宙線通過位置での TGC の検出効率を測定する。これには検出器のみならず、専用のトリガーシステムやデータ収集システムを構築する必要がある。また、製作ペースと検査期間から、同時に検査しなくてはならない TGC の台数は 10 台となり、設備の規模も大きくなる。

このため、一連の検査の中で最も大規模な設備となるのが、宇宙線長期テストとなる。

## 1.7 データ収集 (DAQ) システム

DAQ(Data Acquisition) は一般にデータ収集を意味しているが、物理学実験の分野に於いては、単にデータを収集するだけではなく、それと同時にデータの保存や収集したデータの解析など様々な機能を備えたものでなくてはならない。

実験においてデータ収集を行うためには、イベント (事象) の発生 (トリガー) に即座に応答し、必要な処理を次のトリガーが発生するまでの間にリアルタイムに処理を行う必要がある。

検出器の規模が大きく 1 イベント当たりのデータ量が多い程、また、イベントレートが高い程、DAQ に求められる処理能力は大きくなる。最近では、実験に使用する検出器が複雑化する傾向にあり、大量のデータを高速に収集することが要求されるようになってきた。

このように、データ収集システムは非常に大規模なソフトウエアシステムとなり、その性能が実験の精度や信頼性を決定する重要な実験装置の一部と位置付けることができる。

DAQ は基本的に以下のような設備から構成され、データは 1 から 3 へ読み出される。

1. **検出器** 検出信号がアナログ信号として発生する。
2. **エレクトロニクス** 検出器からの信号が、フロントエンドのエレクトロニクスによってアナログ情報からデジタル情報へと変換される。
3. **コンピュータ** コンピュータがデジタル化されたデータを集め、記録や解析が行われる。コンピュータとしては、ワークステーション、ボードコンピュータ、PC などが挙げられる。

本稿で述べる宇宙線長期検査は、物理実験としても規模が大きく、そのデータ収集システムもまた大規模なものが必要であると言える。この検査でデータ収集システムに要求される性能として挙げられるのは、

- データ量は可変長で、約 120 word/event
- 宇宙線の検出頻度は、約 100 Hz
- オンライン解析を行い、検出効率、Time Jitter 等の検査項目についてグラフの表示等を行う。
- 3 年間安定して動作し続け、また検査期間である 10 日間程度は、ほぼ自動で制御可能であること。これは TGC の検査が長期に渡るためである。

- 収集したデータは一日当たり非圧縮で 2 GByte 近くになるが、これを外部の記憶装置に書き込む。

といったものであり、これらの要求を満たすよう DAQ システムを開発する必要がある。特にデータ量と検出頻度に対応できる処理性能とリアルタイム性が重要となる。

また、必要となる TGC のデータは、ヒット情報と時間情報である。TGC の検出効率を求めるための宇宙線通過時のヒット情報が必要となる。TGC 全面に渡って検出効率の一様性も検査するため、全てのチャンネルのデータが必要である。時間情報は Time Jitter を求めるのに必要となる。この値はチェンバー内の局所的なばらつきが少ないことが分かっているため、一部のチャンネルのデータで十分である。

本論文では、この宇宙線長期テストのデータ収集システムの設計および、それに基づいて実装したシステムでの処理性能の評価について述べる。このデータ収集システムには高い拡張性を持たせることを目的とした設計を行い、その結果、上記の要求を満たす拡張性の高いシステムを構築できることが確認できた。

2 章で、宇宙線長期テストの設備を構成する検出器、フロントエンド・エレクトロニクスについて説明し、DAQ 構築の条件を示す。3 章で、示された条件に基づいた DAQ システムの設計を行い、4 章でその DAQ の実装を、テスト用システムに対して行い、性能を評価した。考察と結論を 5 章で行った。

## 2　設備構成

ここでは宇宙線長期テスト設備を構成する検出器、フロントエンド・エレクトロニクスについて述べる。使用する検出器は、検査する TGC の他には、トリガー用にシンチレーションカウンターを、宇宙線のトラッキング (経路決定) 用に Drift Tube を用いる。エレクトロニクスに関しては、シグナル読み出し用に VME モジュールである TMC、SWINE を、トリガー用ロジック回路には NIM モジュール、VME・Interrupt & I/O Register モジュールを使用する (図 9 参照)。

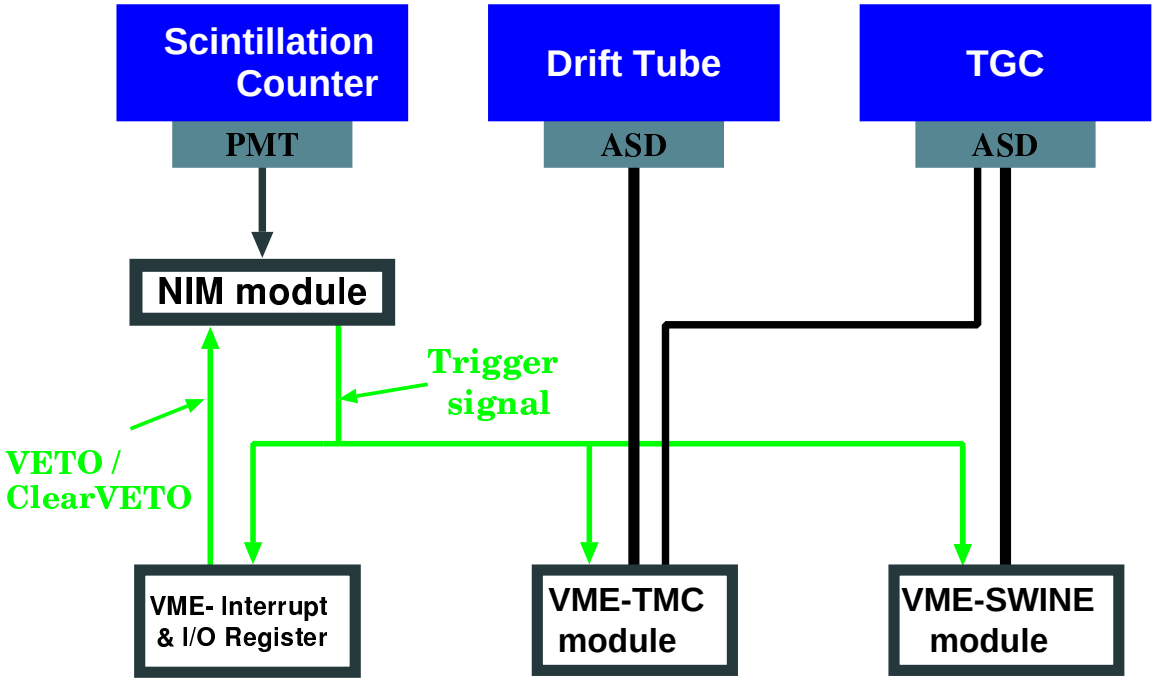


図 9: 検出器、エレクトロニクス構成

シンチレーションカウンターが宇宙線通過を検知し、NIM モジュールのロジック回路を通じて TMC、SWINE モジュールにイベント発生を通知する。TMC、SWINE は、Drift Tube、TGC からの信号を ASD ボードを通じて読み込む。TMC は時間情報へ、SWINE はヒット情報へとデータを変換する。データ変換の間、トリガーシグナルをブロックする VETO とその解除には、Interrupt & I/O Register のインタラプト・レジスター機能を利用する。

これらのデータから、Drift Tube の時間データを用いてトラッキングを行い、TGC のヒット情報から、その検出効率を算出し、TGC の時間情報から Time Jitter を算出する。

以下で、検出器、フロントエンド・エレクトロニクスについて述べる。

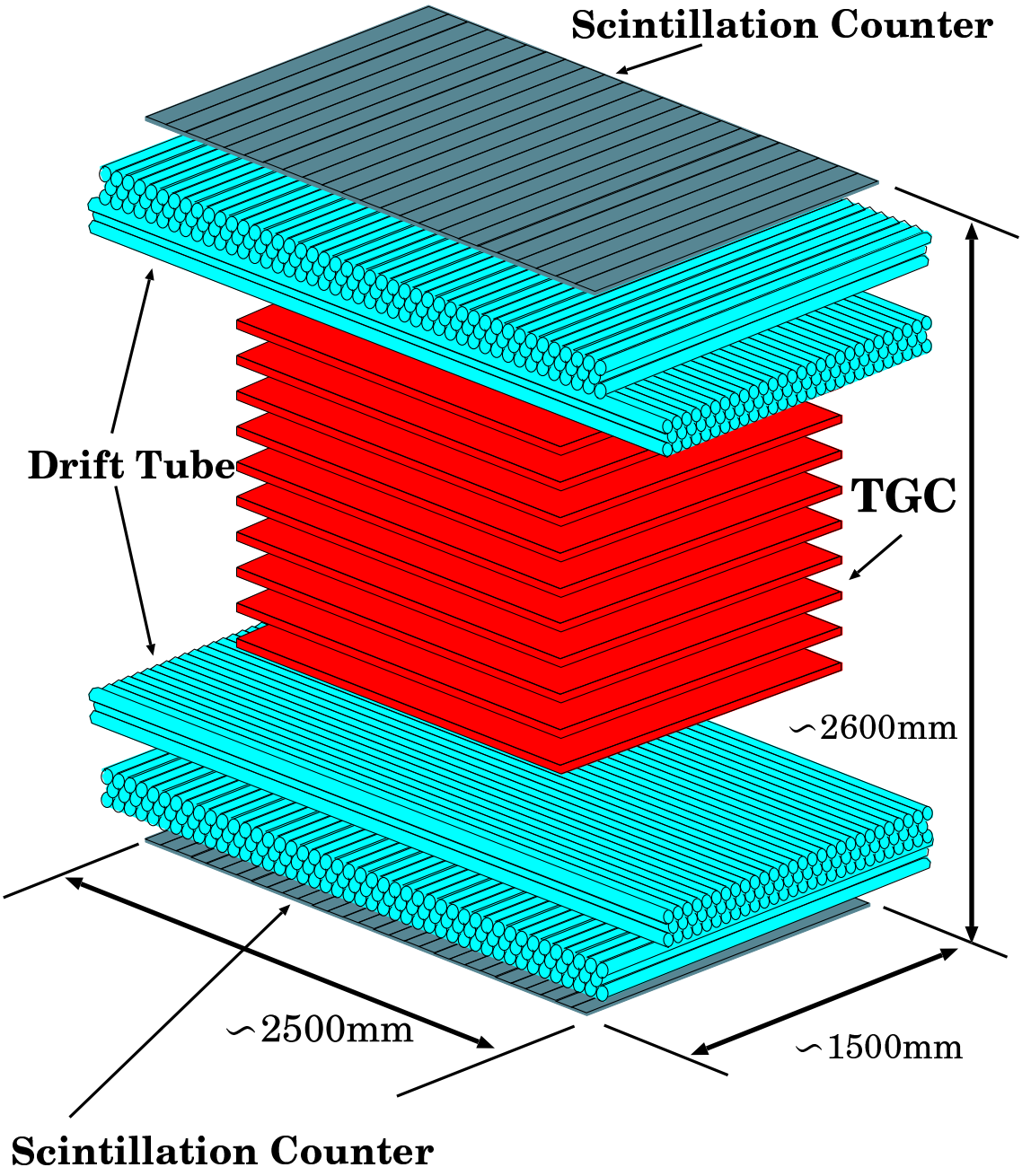


図 10: 各検出器の配置。ここでは架台、エレクトロニクス等は省略している。

## 2.1 検出器

検出器の構成と配置を、図 10 に示す。

検査設備は、TGC Doublet を一度に 10 台検査できる規模で製作される。シンチレーションカウンター及び Drift Tube のトラッキングによりカバーできる中央 TGC 部分の有効測定領域は、約 $1300 \times 2000\,\mathrm{mm}^2$ である。また、この配置による宇宙線の検出頻度は約 100Hz と見積もることができる。

### 2.1.1 Drift Tube

Drift Tube は円筒形の比例計数管であり、その動作原理は構造の違いを除いては TGC と同じである。

Drift Tube を使用する目的はトラッキングを行うことであるが、その仕組みについて述べる。Drift Tube の有感部は円筒電場となるため、粒子の通過によって生成した電子はまっすぐに anode wire へと向かう。この結果、粒子通過からシグナル発生までの経過時間から、anode wire とトラック (粒子通過経路) までの距離が一意に決まる (図 11(a) 参照)。更に、複数の層を重ね合わせる事により、トラックの、Drift Tube に垂直な成分が求められる (図 11(b) 参照)。また、Drift Tube を互いに垂直な方向に配置する事により、トラックが確定する。

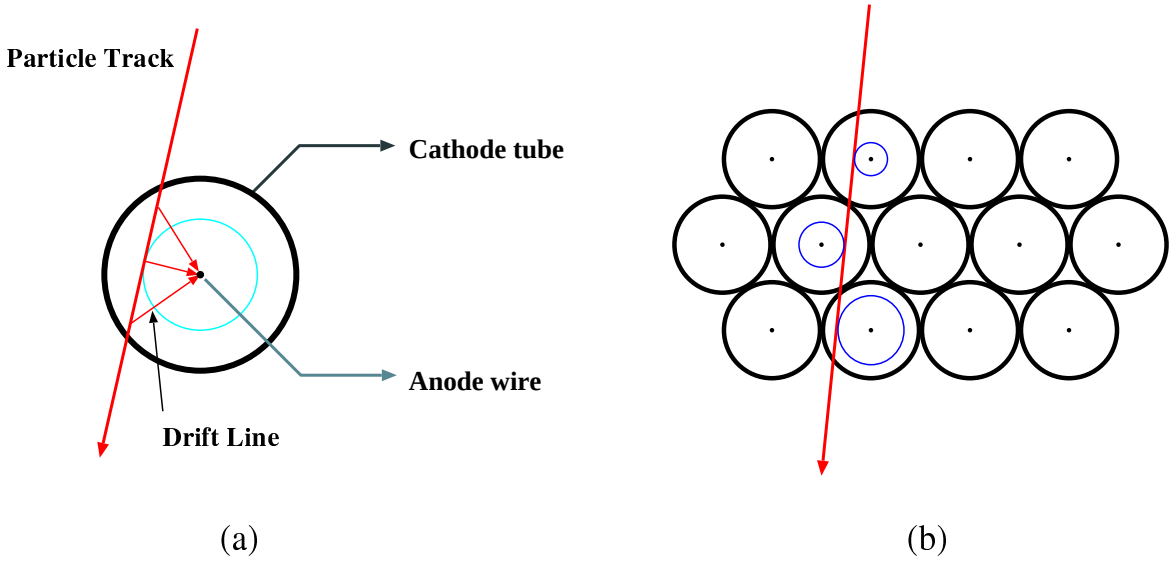


図 11: Drift Tube によるトラッキング

本設備に用意する Drift Tube の構造は、Cathode tube がアルミニウム製 (直径 5cm、厚さ 2mm)、Anode wire は直径 50 $\mu$m である。円筒両端には同じ くアルミニウム製のキャップを取り付ける。充填ガスには Ar+ethane(50:50) を用いる。上下

それぞれに対して、X 軸に 2500 mm の長さのものを 3 層で 83 本、Y 軸に 1500 mm のものを 131 本設置する。

シグナルの読み出し線はエンドキャップ部分から、TGC と同様に ASD ボードへと接続される。

### 2.1.2 シンチレーションカウンター

上面に $70 \times 1300 \times 10\,\mathrm{mm}^3$ の大きさのものを 29 枚、下面に $90 \times 1300 \times 2.6\,\mathrm{mm}^3$ のものを 23 枚並べて配置する。カウンターの両端に接続した Photomultiplier からの検出シグナルは NIM モジュールのロジック回路へと送られる。NIM モジュールでは、カウンターの両側からのシグナルのコインシデンスを取ってそのカウンターのシグナルとする。上部・下部それぞれからのシグナルのコインシデンスを取ることにより、宇宙線通過のトリガー信号が発生する。

## 2.2 VME モジュール

高エネルギー実験で使用されるデータ収集用のバス規格としては、CAMAC、FAST-BUS、NIM、TKO、VME などが挙げられるが、産業界では VME の規格が多く、高エネルギー実験でも VME の規格の採用が増えている。今回の検査設備に必要となる TDC 等のモジュールに関しても、高エネルギー実験の分野で目的に敵ったものが VME 規格により開発されていることから、本検査設備のデータ収集用バスは VME を採用した。

VME バスのモジュールは、モジュールボードを差し込むことのできるスロットを最大 21 備えた、バックプレーンを装備したクレートを用いて運用する。

TGC 及び Drift Tube からのシグナル読み出し用の VME モジュールには、それぞれ TMC-VME TDC モジュール、SWINE NWPC Readout システムを使用する。ここで、それぞれのモジュールについて述べ、検査システムで 1 イベント当たりの読み出しデータ量を見積もる。TMC、SWINE 共にデータは 16 bit 幅であるため、データ量は 16 bit を 1 word として計算した。

### 2.2.1 TMC

TMC は Time Memory Cell の略で、時間ディジタル変換回路 (TDC、Time to Digital Converter) の一方式である。TMC は入力信号の状態を記録するフリップフロップが並列に並べられている。書き込み信号が入ると、可変遅延ゲートによって、各フリップフロップの書き込みタイミングが 1 ステップずつ遅れて入力される。従って、入力データを可変遅延ゲートの遅延時間の間隔でフリップフロップに記録することができる。[14]

ここで使用するのは、TMC LSIを組み込んだ32チャンネルの6U VMEバス規格の汎用TDCモジュールである。このモジュールの基本仕様を表5に示す。

表 5: 32 ch TMC-VME モジュールの基本仕様 [14]

| | |
|---|---|
| TMC チップ | TMC-TEG3 チップ (4 チャンネル/チップ)×8 |
| DSP | モトローラ DSP56002(∼ 40 MHz) |
| DSP 用 I/O | フロントパネルからのシリアル I/O<br>VME バスからパラレル I/O(8 bit) |
| DSP 用 RAM | MCM56824(8 k × 24 bit) × 2 |
| ROM | 27512(64 k × 8 bit) × 3 |
| DPM | IDT7025(8 k × 16 bit) |
| データ入力 | 32 チャンネル ECL 差動電圧入力<br>34 ピンフラットケーブルコネクタ ×2 |
| スタート・ストップ入力 | NIM 入力 LEMO 型コネクタ |
| スタート同期出力 | NIM 入力 LEMO 型コネクタ |
| 量子化ステップ | 0.781 ns[40 MHz クロック時] |
| 時間分解能 | $\sigma = 370$ ps[40 MHz クロック時] |
| フルスケール | 3.2 $\mu$s[40 MHz クロック時] |
| VME バス I/F | アドレス幅 24、データ幅 16 bit、<br>スレーブブロック転送機能、割り込み機能 |
| ボードサイズ | VME6U × 1600 mm |
| 消費電力 | 8 W 以下 (-1.2 V 0.16 A: +5 V 1.4 A) |

TMC LSIから読み出したデータはヒットのなかったものは削除され、コンピュータへの転送用に用意されるため、データは可変長となる。ここで、このモジュールのパラメータを幾つか示す。

**dready** データの変換や読み出しのためのフラグ。VME 側から 0 をセットする (書き込む) ことにより、TMC はデータ変換の準備を行う。データ変換が終われば、dready は 1 にセットされるので、値が 1 となるのを待つ。

**ntotal** 変換されたデータの数 (by 16 bit word) がセットされる。

**drun** 0 をセットする (書き込む) ことによって以下のパラメータをセット可能にする。

**dcstsp** Common start(=1) と common stop(=0) の mode を選択。

**dcount** Time range を選択。25 ns × dcount(Max.126)

**dmodule** データに付加されるモジュール ID。

データは 1 つのヒットにつき、2 word からなる。1 word はモジュール ID とチャンネル、もう 1 word は TDC データである。

検査設備で用いる Drift Tube は全部合わせて 428 本であり、1 本ずつ 1 チャンネルとして読み出すため、32 ch の TMC モジュールが 14 台必要となる。ここで、TMC モジュールから読み出すデータ量を見積もる。上から下へ Drift Tube を宇宙線が突き抜けたとすると、Drift Tube の層は全部で 12 あるため、ヒット数は 12 チャンネルであると期待できる。多めに見積もり、宇宙線が 2 本通過したとして全部で 24 チャンネルと考えると、データ量は 1 イベント当たり 48 word となる。

データ変換速度は、下の式により見積もることができる。[14]

$$\mathrm{t}(\mu\mathrm{sec}) = 0.4 \times (\text{チャンネル数}) \times (\mathrm{dcount}) + 4 \times (\text{ヒット数})$$

Drift Tube からの読み出しには dcount $= 50(1.25\,\mu$sec) を使用し、32 チャンネルのデータ入力である。1 モジュールで 2 つのヒットがあった時のデータ変換速度は、おおよそ 648 $\mu$sec と見積もることができる。

### 2.2.2 SWINE *

SWINE(Super WIre Net Encoder) は MWPC 等からの入力のヒット情報を読み出すシステムで、TGC の検出効率を測定するのに最適なモジュールであるといえる [15]。

SWINE は、PORQ(Pulse On ReQuest) と HOG(Hold On Go) の 2 種類の VME モジュールからなり、さらに独自の J2 バックプレーンを用い、制御シグナルなどは、このバスを通して分配される。PORQ はクレートに 1 台必要でクロック、トリガー等の制御シグナルを HOG モジュールに分配する。スレーブボードである HOG は、ASD からの入力をうけ、ヒットの有無を決定する。1 クレートに最大 15 枚が搭載可能である。HOG は一定のゲート幅を設定し、その間の ASD 入力からの信号の有無を読み出す。ゲート幅は、20 ns ステップで 320 ns までの値を設定できる。

HOG モジュールのチャンネル数は 64 で、データは 4 word からなる。1 ch 当たり 1 bit が割り当てられ、ヒットの有無を読み出すことができる。ワイヤーグループ、ストリップ共に 32 チャンネルである Doublet を 10 台検査する場合、TGC の読み出しは 1280 チャンネルである。このとき、必要となる HOG モジュールは 20 台であり、1 イベント当たりのデータ量は 80 word となる。

HOG モジュールがデータを内部のバッファへ書き込むのに要する時間は、ゲート幅 $+ \sim 500$ ns である。

*SWINE は現在、高エネルギー加速器研究機構において開発中である。

# 3　DAQシステムの設計

DAQシステムは2章で述べた検出器、フロントエンド・エレクトロニクスに加えて、データ収集を司るホストコンピュータ、それらの上で走るソフトウエアから成る。前章までに述べたDAQに対する要求、及び検出器・エレクトロニクスの構成に基づいて、DAQシステムを設計した。

以下、それぞれの構成要素について設計の根拠と妥当性について論じる。

## 3.1　コンピュータ環境

読み出し用モジュールを載せるVMEクレートを含めた、コンピュータ構成を図12のように設計した。構成要素は、PCと、PCとVMEのインターフェイス、およびFast Ethernetによるネットワーク環境である。

ハードウエアから見たデータ収集の流れは、

1. 各VMEモジュールが検出器からのデータを変換する。
2. 各VMEクレート毎にPCが、PCI-VME Bus Adaptorを通じて、変換されたデータを読み出す。
3. 読み出されたデータは、Fast Ethernetを介して解析用のPCへ転送される。

といったものになる。

### 3.1.1　PC-UNIX

IBM PC-AT互換機 (PC) の普及は急速に進んでおり、その性能は今やワークステーションと互角またはそれ以上といえる。さらに価格を考慮すればそのコストパフォーマンスの高さは他のどんなコンピュータよりも優れているといえる。また、周辺機器も豊富に存在し、他のシステムとの親和性も高い。この様に現在のPCは、高い性能と汎用性を兼ね備えたものといえる。DAQシステムのコンピュータとしてはPCを選択した。OSは、PC用のUNIX OSとして広く普及しているLinuxを採用した。

### 3.1.2　PC-VME Bus インターフェイス

PCを採用する場合、PCとVME busとのインターフェイスが必要になる。これは、Bit3社製model 616 PCI-VME bus Adaptorを使用することができる。これは、PCに接続するPCIカードとVME側のモジュール、その間のケーブルから構成される。このアダプターの機能は、互いのメモリ空間を共有するものであり、互いのメモリを自らのメモリのように扱うことができるため[16]、PCがVMEモジュールを直接コントロールすることが可能になる。

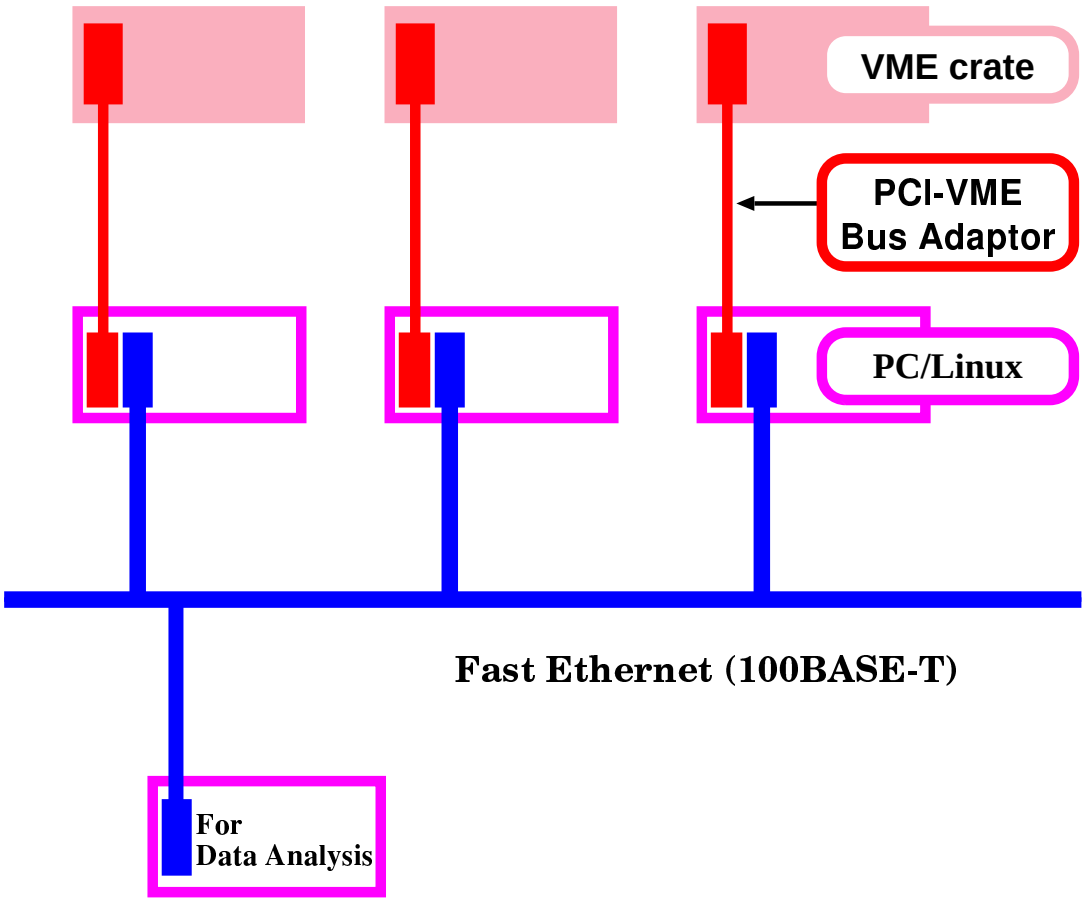


図 12: コンピュータ環境

Bus Adaptor の PCI カードに対して Linux OS 用デバイスドライバが必要となるが、これに関しては、vmehb[17] というドライバが開発されている。これは WWW で公開されており自由に使用可能であるため、これを Linux Kernel へ組み込むこととした。また、このドライバを利用した、C 言語ライブラリも公開されている (vmelib[18]) ので、これを用いて開発を行う。

### 3.1.3 ネットワーク分散システム

ここで、DAQ への要求に対応しさらに処理性能を上げるため、PC、Bus Adaptor、VME crate のセットを複数用意し、それぞれの PC が分担して読み出しを行うことが可能な構成を考える。この構成をとると、Adaptor が増えるため VME bus からのデータの転送速度はセットの数に比例し、読み出し処理性能も PC の数が増えるため同様に向上する。またセット数の設定により、設備の規模や DAQ 性能への要求の変化にも対応できる拡張性の高いシステムが構築できる。

これを実現するためには複数の独立した PC を制御する仕組みを要する。まず

必要となるのは前述したデータ収集においてそれぞれのPC間で同期を取ることであり、それができなければ、それぞれのPCから独立に収集したデータを1つのイベントに再構築することは不可能である。

同期を取る方法としては、VME側に専用のハードウエアを用意する方法と、もう一つはネットワークを利用し、PC上の読み出しプロセスが互いにネットワーク通信を行う方法とがある。前者の方が速度の面では有利であるが、専用のハードウエアを用意する必要がある。後者は、速度面では多少劣るが、これはソフトウエアでの実現となるのでどのような実装も可能である。さらに、読み込んだデータや周辺機器の共有、それぞれのPC上でのプロセスの実行などの必要性から、ネットワーク接続は現在の実験環境では不可欠であり、ハードウエアとして新たに必要となるものはない。また、UNIXでは標準でネットワーク通信を行う環境が整っており、開発もスムーズに進めることが可能である。

以上の理由により、データ収集用のPCはネットワーク上に分散して配置する。通信速度が遅くてはネットワークに分散させる利点が失われるため、ネットワークには、転送速度100 Mbpsの Fast Ethernet(100 BASE-T)を導入する。現在では、PCに標準搭載されるネットワーク・インターフェース・カードの多くが対応しているなど普及が進んでいて導入も容易である。

以上のようなネットワークに分散したシステムの利点をさらに活用し、データの解析のみを行う専用のコンピュータを、読み出し用PCとは別に配置する。読み出し処理と解析処理を分担させ全体の性能を向上する。

## 3.2 ソフトウエア設計

図12で示したシステムの上で走るDAQのソフトウエアを設計を行った。その構成を図13に示す。

各PC上のプロセスは、必要に応じてネットワーク上の他のプロセスとTCP/IPプロトコルを用いた通信を行うことによって、DAQを実行する。

VMEに接続している各PCに1つ、データの読み出しを担当するプロセスが存在する。1つのイベントを読み出す時、データはそれぞれのプロセスが独立に読み出す。これらのデータが1つのイベントであることを通知するために、データ読み出しのプロセスは全て、TCPを利用したソケット通信を行う。ただし、TCPは一対一で接続を確立する必要があるため、読み出しプロセスは、サーバの役割を担当するマスタープロセスが1つと、そのクライアントとして振舞う幾つかのスレーブプロセスを設定し、それぞれが接続を行うようにした。

TMCのデータを読み出すプロセスは、Drift Tubeのデータを用いてイベント・フィルタを行い、イベントを選別するこの結果は同じくソケット通信により、他のプロセスへ通知し、”良い”イベントのデータのみが解析プロセスへ送られる。通知するイベント・フィルタの結果は、全てのプロセスへ伝える必要があるため、イベント・フィルタを行うのはマスタープロセスでなくてはならない。加えて、フィ

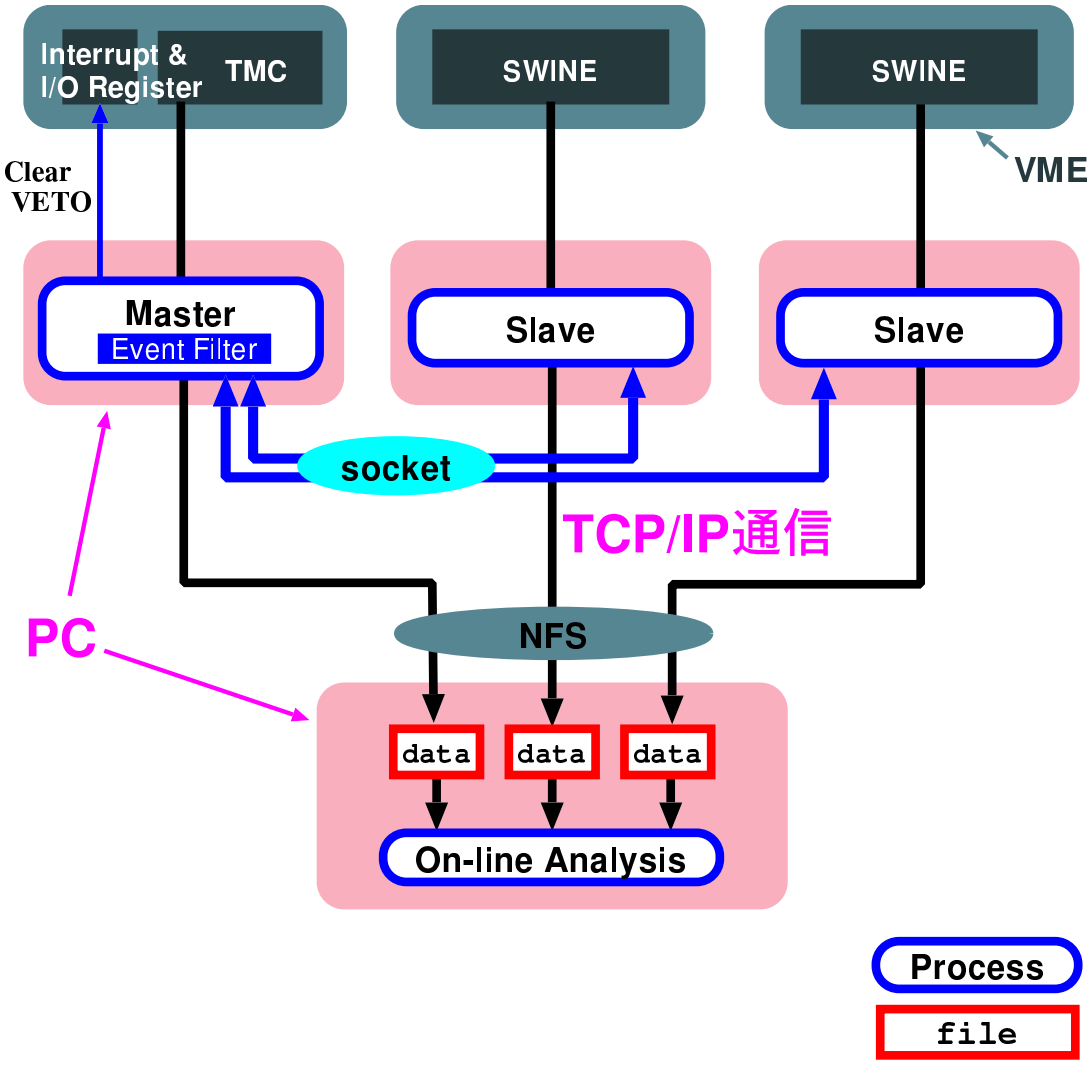


図 13: ソフトウエア設計

ルタには Drift Tube のヒットパターンを用いるため、マスタープロセスが Drift Tube のデータを読み出すことになる。

それぞれのプロセスが読み出したデータを、解析プロセスへ受け渡すため、ネットワークを通じデータを転送する必要がある。これには、TCP/IP を用いた NFS(Network File System) を利用して、データをファイルに書き込み、転送を行う。解析プロセスは指定されたファイルを読み込み、解析を行う。

### 3.2.1 TCP/IP

ネットワーク上のコンピュータが通信を行うためには、厳密に定義されたプロトコルを用いる。プロトコルとは、通信に参加するもの同士の規則や慣例の集ま

りである。TCP/IPは、正式名称を”DARPA Internetプロトコル群”という[19]プロトコルの集まりである。TCP/IPプロトコルの特徴は、

- 特定のベンダー独自のプロトコルではない。
- パーソナルコンピュータからスーパーコンピュータにいたる、あらゆる種類のコンピュータ上で実現されている。

といったものなどがあげられ、インターネットや、UNIXのネットワーク通信における標準的な通信プロトコルとなっている。プログラミングに対して提供されるものとしては、TCP/IPプロトコル群のうち、TCPとUDPの2つのプロトコルが用意されている。

DAQプロセスがネットワーク通信を行うのに用いるソケットとNFSについて述べる。

**ソケット・システムコール**　ソケットは、UNIXシステムにおいてネットワーク通信を担う基本的な機能であり、プログラミング環境としてはシステムコールであるC言語の関数群として、実現されている。ソケットは、通信プロトコルを特定してはいないが、実際はTCP/IP以外のプロトコルは用意されていない場合が多い。

データ読み出しプロセスに関しては、ソケットのタイプとしては、TCPを用いたストリームソケットを使用する。TCPプロトコルでは、通信データはエラー無しで届くことが保障され、送信した順序も保たれるが、一対一で接続を確立する必要がある。一方、ソケットのもう一つのタイプ、データグラムソケットはUDPを用いているが、UDPはエラー無しの到着と送信の順序は保障されないので、データグラムソケットを使用する場合はチェックを独自に行わなくはならない。このため、ストリームソケットを採用する。

**NFS**　NFSはネットワークを介して透過的にファイル・アクセスを行うことを可能にする機能で、標準的なUNIX OSで利用できる基本的な機能の一つである。これにより通常のファイルを扱うのと同様にネットワーク上の他のコンピュータのファイルを扱うことができる。NFSはUDPプロトコルを使用している。

データ収集システムにおいてデータを転送するための方法としては、このNFSを利用する方法と、転送のためのプロセスを用意する方法があげられる。

NFSを利用する場合は、転送のためのプロセスを新たに開発する必要はなくなる。しかし、NFSではいずれかのコンピュータ上のハードディスクにデータを書き込み、解析プロセスはもう一度読み出さないといけないという手間がある。問題になるのは、データを書き込むだけの容量をハードディスクに確保する必要があるということと、読み書きの処理速度であると考えられる。容量に関しては、最近のPCに搭載されているハードディスクは、最低でも数GByteの大容量のものであり、問題ないと考えられる。ただし、処理速度に関しては考慮する必要がある。

また NFS を利用した場合は、データを渡すデータ収集プロセス、データを読み込む解析プロセスは、ファイルの読み書きの動作だけでデータの受渡しが可能であり、データ収集プロセスと解析プロセスの間でデータの受渡し時に同期を考える必要はなくなる。

以上の理由により、データ転送には NFS を利用する。

### 3.2.2 イベント・フィルタ

解析の段階では Drift Tube のデータを用いて宇宙線のトラッキング (経路決定) を行うわけであるが、トラッキングが不可能なイベントのデータは不要である。トラッキングの可否は、ヒットパターン、つまり信号のあったチャンネルの配置だけで大まかに判別できるため、解析に入る前の段階でイベントをふるい落とすことが可能である。

最近の CPU の高い性能を活用することにより、また、読み出しプロセスを分散させて処理速度を向上させたため、読み出し処理の段階でこうしたフィルタをかけることが可能となる。これにより解析プロセスへ転送するデータの量を減らすことができ、ネックとなるネットワークを通じたデータ転送や、それ以降の処理の負担を減少することができる。

### 3.2.3 開発言語

UNIX OS は、C 言語で開発されており、ハードウエアへアクセスするような低レベルな処理の機能や、TCP/IP 通信の機能は、まず C 言語のために用意されている。この章で述べた、VME へのアクセスを行う vmelib や UNIX システムコールも C 言語のライブラリ関数であり、C 言語で DAQ の開発を行うことが容易となる。

一方、検査設備が要求する DAQ システムの規模は大きなもので、扱うデータの種類と量も多く、その処理は複雑なものとなるため、C 言語を拡張した C++言語を用いた。

C++言語の機能を利用すれば、プログラムを関連する幾つかのサブグループに分解して考えることができるようになる。各サブグループは自己包含型のオブジェクトになっており、該当するオブジェクトに関連する命令とデータで構成される。これにより、複雑さは減少し、大規模なプログラムを構築できるようになる。

以上のことから、C++言語を用いて DAQ システムの開発を行った。

# 4 DAQシステムの実装とその評価

この章では 3 章で述べた設計に基づいたデータ収集システムの実装を行い、プロセスの処理時間を測定して評価を行う。データ収集システムは、リアルタイム性が求められ、検査設備では宇宙線検出頻度の約 100 Hz に対応できる処理時間が要求される。

## 4.1 読み出しプロセスのシーケンス

マスタープロセスと 2 つのスレーブプロセスからなる、読み出しプロセスのシーケンスを図 14 に示す。

このシーケンスにより、読み出しプロセスは同期を取りつつ 1 イベントずつ読み出しを行う。このシーケンスに基づいて C++言語で記述したソースコードの一部を Appendix A に掲載する。

図の各項目での動作は、

- VME へアクセスを行う部分
- ソケット通信を行う部分
- NFS を利用したデータの転送を行う部分
- イベント・フィルタの部分

の 4 種類に分けられる。それぞれの項目について説明を行う。

### 4.1.1 VME へのアクセス

VME へのアクセスする手段としては vmelib を使用する。vmelib のライブラリ関数のうち、model 616 で利用できるのは、vme_mapopen() と vme_mapclose() の 2 つの関数である。vme_mapopen() 関数は、VME bus 上の任意のアドレス空間を PC 上の仮想メモリ空間へと割り当てるもので、PC 上のプロセスはメモリを扱うのと同じ様に、VME 上のモジュールとデータのやりとりを行うことが可能になる。vme_mapclose() は、この割当を閉じる関数である。

**Initialize *** まず vme_mapopen() 関数により、TMC、Interrupt & I/O Register、SWINE の各モジュールのアドレスをメモリに割り当てる。また TMC のパラメータの設定を行う。その後、イベント毎のデータを読み出すループへ入る。

**Ready *** TMC では、dready = 0 と書き込み、TMC にデータ読み出しの完了を伝え、次のデータ変換を開始させる。

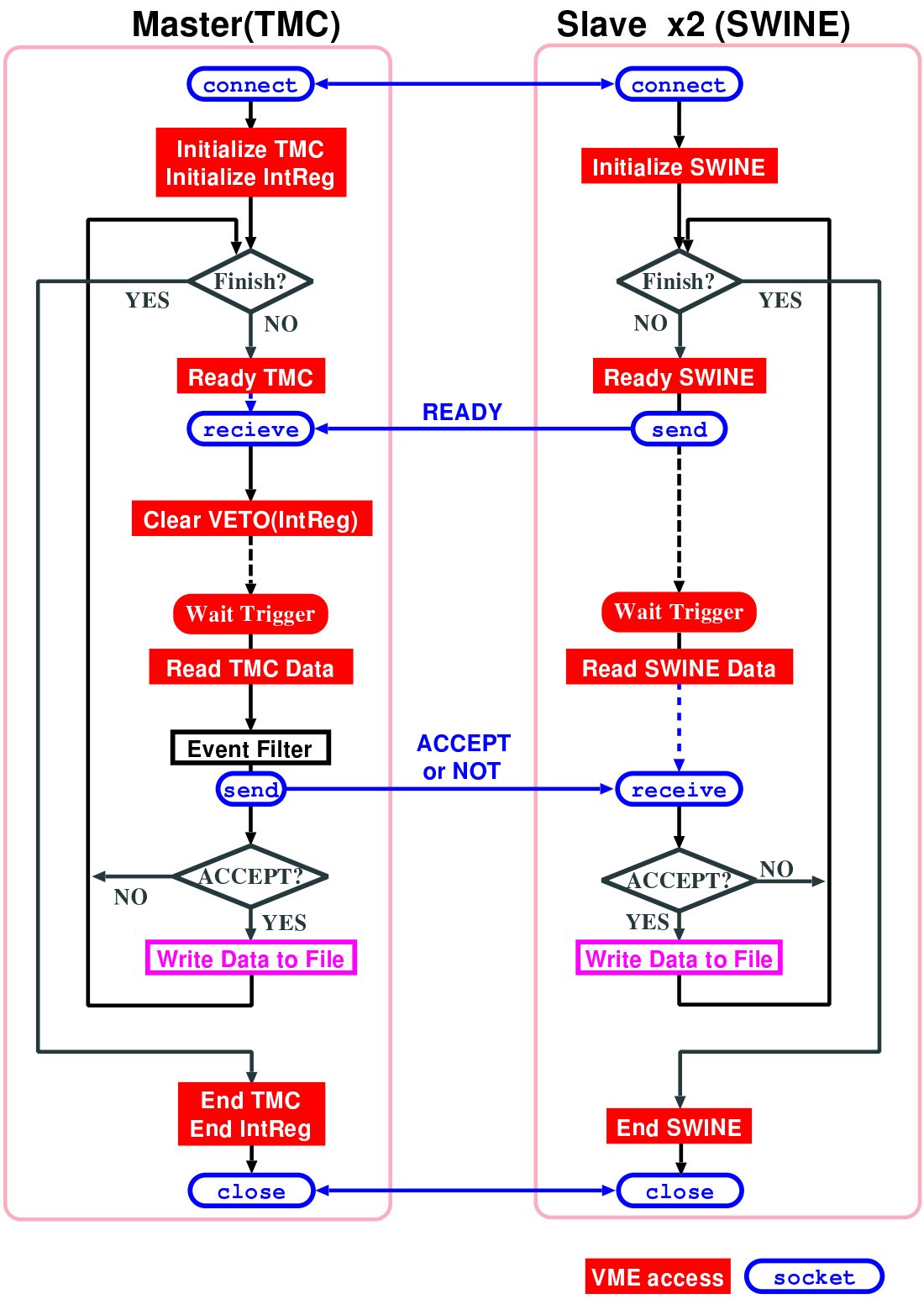


図 14: 読み出しシーケンス (DT: Drift Tube, IntReg: Intterupt & I/O Register)

**Clear VETO** VETOを解除する。Interrupt & I/O Registerの状態を読み出した(1 word)後、CLEARのフラグを立てて、書き込み(1 word)を行う。

**Wait Trigger** トリガー信号が来て、データ変換が完了するのを待つ。すなわちTMCでdreadyの値をモニタし1となるまで、繰り返し読み出しを行う。

**Read * Data** TMCでは、ntotalにセットされた値の数だけデータを読み出す(hit数 ×2 word)。

**End *** vme_mapclose()関数で、アクセスを終了する。

### 4.1.2 ソケット通信

ソケット通信は、システムコールとして用意されている、各ライブラリ関数を用いて行う。例えば送信、受信に関してはsend()、recv()関数が用意される。

**connect** マスタープロセスとそれぞれのスレーブプロセスが互いに接続する。

**READY** 次の読み出し準備が完了すると、それをスレーブプロセスはマスタープロセスに伝える。マスターは全てのプロセスで準備が完了するのを待つ。

**ACCEPT or not** マスターはEvent Filterの結果を各スレーブへと伝える。各プロセスでは”良い”データのみがファイルへと書き込む。

**close** 互いの接続を切って終了する。

### 4.1.3 NFSを利用したデータ転送

NFSを利用してデータを転送する部分に関しては、通常のファイルを扱うのと何ら変わりがなく、NFSによりネットワークを介しているファイルを指定するだけで良い。

**Write Data to File** ネットワークを介したファイルに、データを書き込む。

### 4.1.4 イベント・フィルタ

**Event Filter** Drift Tubeのデータからそのヒットパターンを調べ、トラッキングの不可能なイベントをふるい落とす計算を行う。

## 4.2 VMEへのアクセス時間

PC上のプロセスから、Bus Adaptorを通してVMEモジュールの読み出し (READ)、書き込み (WRITE) を行い、その処理にかかる時間を測定した。測定したモジュールはTMC、Interrupt & I/O Registerの2つであり、いずれのモジュールもREAD、WRITEは16 bit幅 (1 word) のデータサイズである。

測定方法は、READもしくはWRITEの動作を1000回forループ文で繰り返し、その前後のシステムタイムを取得し、その差分を求める。算出した値を1000で割れば、動作1回の処理時間を割り出すことができる。システムタイムの取得は、UNIXシステムコールgettimeofday()関数を用いた。この関数を用いて$\mu$sec単位で測定することができる。forループ自体の処理は、ほとんど結果に影響しないことが測定により分かっている。

図15にTMC，図16にInterrupt & I/O Registerの測定した処理時間を示す。

また、表6にその平均処理時間をまとめた。

表 6: VMEモジュールのREAD/WRITE処理時間 (単位: $\mu$sec/word)

| | TMC | Interrupt & I/O Register |
|---|---|---|
| Read | 2.1 | 2.3 |
| Write | 1.8 | 2.0 |

この結果を用い、検査設備用データ収集プロセスで1イベント読み出すのに必要な、VMEアクセスの処理時間を算出する。ここで、dready = 1を待つ動作は読み出し1回と数える。Interrupt & I/O RegisterはREAD、WRITEを各1 wordずつ行い、TMCのWRITEは1 wordである。TMCのREAD動作は、2 wordに加えヒット数毎に2 wordのデータが読み出される。図15、16での平均値から、TMC READ = 2.1 $\mu$sec、TMC WRITE = 1.8 $\mu$sec、Interrupt & I/O Register READ = 2.3 $\mu$sec、Interrupt & I/O Register WRITE = 2.0 $\mu$secの値を用い、

$$
\begin{aligned}
\mathrm{t}(\mu\mathrm{sec}) &= 2.3 + 2.0 + (1.8 + 2.1 \times 2) \times (\text{モジュール数}) + 2.1 \times 2 \times (\text{ヒット数}) \\
&= 4.3 + 6.0 \times (\text{モジュール数}) + 4.2 \times (\text{ヒット数})
\end{aligned}
$$

ヒット数は、全てのTMCモジュールのデータ数の合計である。このシステムではTMCは4台使用していて、ヒット数が全てで20であったとするとVMEへのREAD/WRITEの処理時間は約112 $\mu$secとなる。

SWINEの読み出しの処理時間を2 $\mu$secと仮定すると、2台のPCから読み出す構成を考えているので、それぞれ約100 $\mu$secの処理時間が必要になると考えられる、分散して読み出すことで処理が速くなることが分かる。

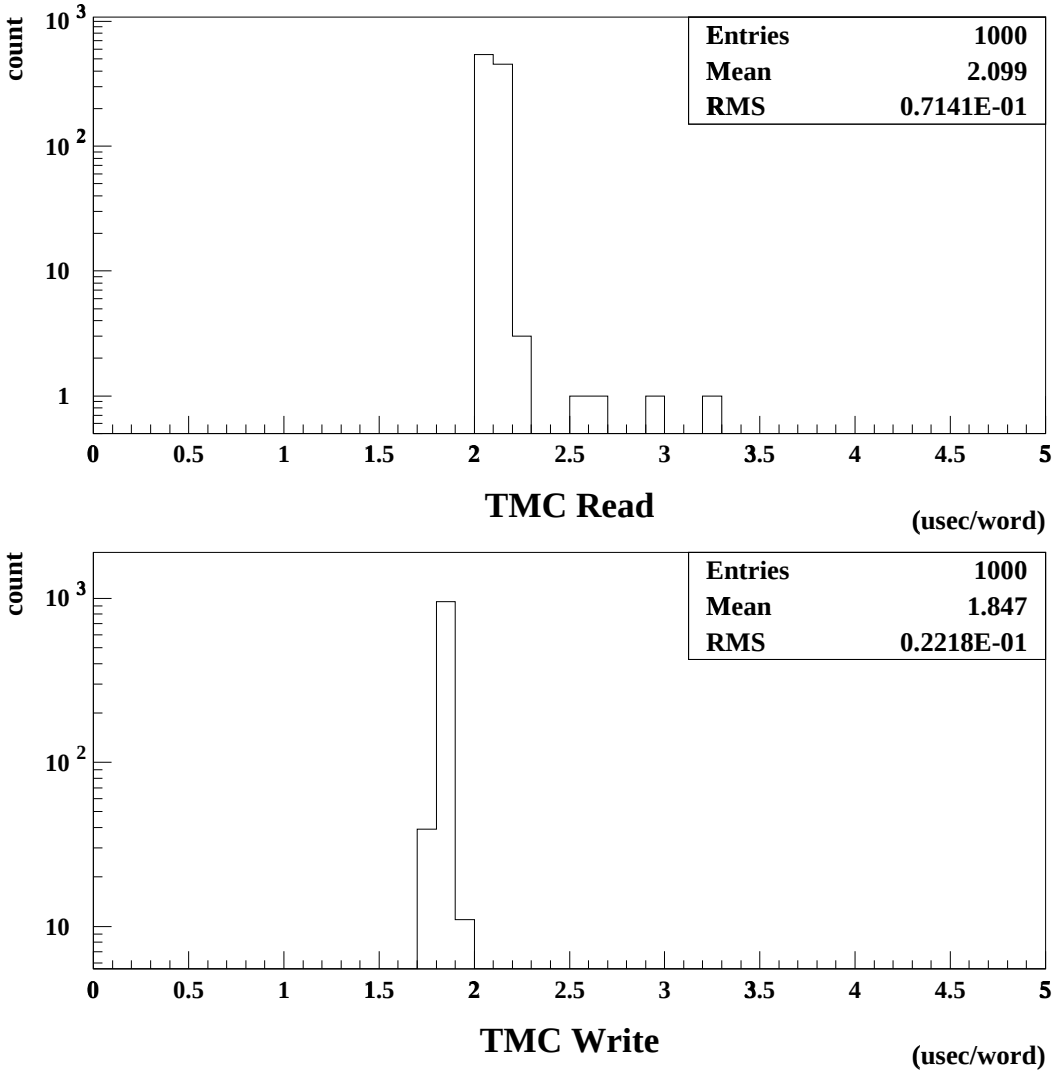


図 15: TMC の READ/WRITE 時間。動作は、ntotal の読み出しと dready への書き込みである。

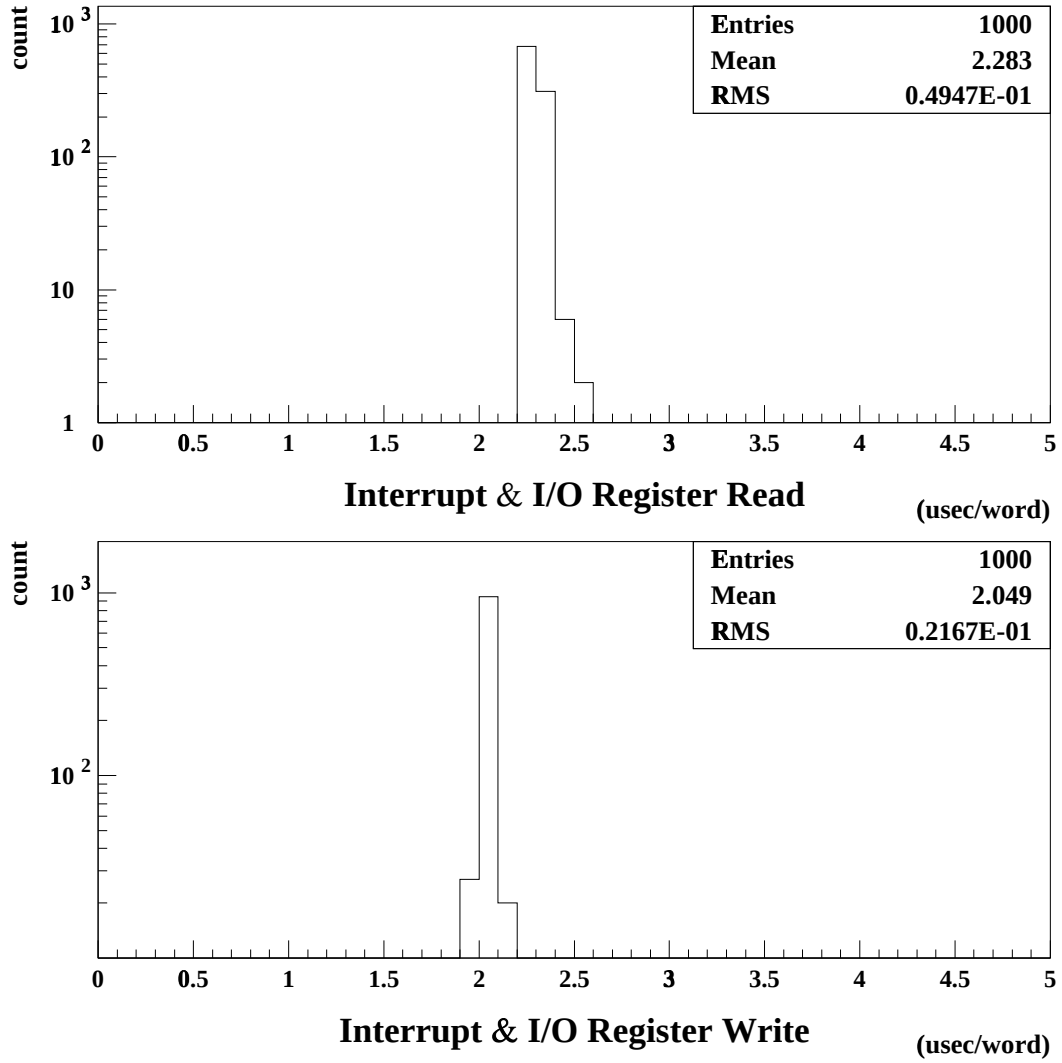


図 16: Interrupt & I/O Register の READ/WRITE 時間

## 4.3 ソケット通信の処理時間

次に、実際にネットワークに分散したシステム上でデータ収集を行うプロセスを評価するために、読み出しプロセスのうち、ソケット通信部分以外をダミーに置き換えたプログラムを用いる。

ここでは、プロセスの中でネットワーク通信に関係した部分の処理時間を測定する。測定に用いた PC は 2 台で 1 台の HUB に接続し、それ以外には接続しない。測定ではネットワークは Ethernet(10BASE-T)、FastEthernet(100BASE-T) を用いた (図 17(a) 参照)。また、スレーブの 2 つのプロセスは同じ PC 上で実行した (図 17(b) 参照)。

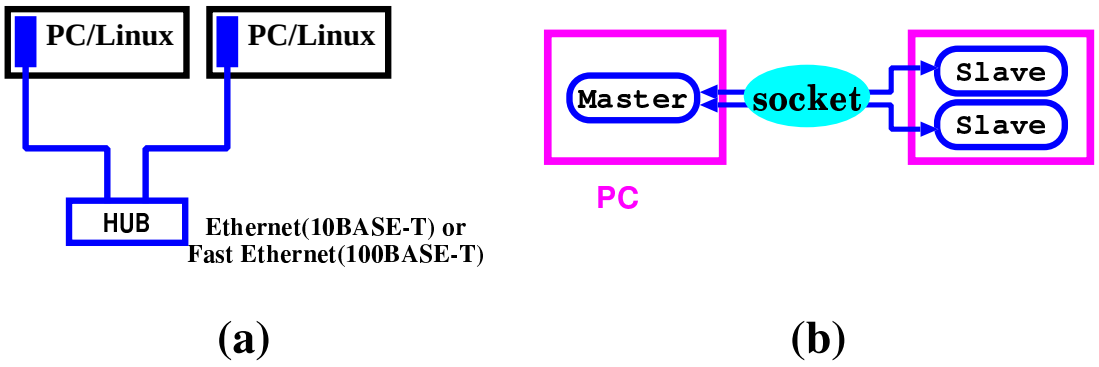


図 17: ソケット通信プロセス用セットアップ

方法は 4.2 と同様に行い、1 イベント読み出しのループを単位にして、3 つのプロセスでそれぞれ同時に時間を測定する。測定時は、互いの通信以外の動作は行っていない。

図 18 に Ehernet 使用時の結果を、図 19 に Fast Ethernet の結果を示す。

3 つのプロセスは同期を取っているため、処理時間はほとんど同じであるといえる。これにより、読み出しプロセスが、1 イベント読み出す間にかかるネットワーク通信の処理時間は、Ethernet を用いた場合は、約 374 $\mu$sec、Fast Ethernet の場合は、約 238 $\mu$sec と見積もることができる。Ethernet よりも Fast Ethernet の方が処理は速いが、せいぜい 1.5 倍程度である。これは、ネットワーク以外での処理時間が影響しているためと考えられる。特に、ここで用いているソケットによる通信で、1 回に転送するデータは TCP/IP が付加するヘッダ (58 Byte) を含めて、約 70 Byte でありそれほど多くはない。

システムコールは、他のライブラリ関数とは違い、ユーザプロセスが UNIX の Kernel に対して要求を行い、Kernel が処理を行う関数である。そのため、ここで用いているソケットの処理には、ユーザプロセス・Kernel 間の処理の移動 (コンテキスト・スイッチ) や、送受信したデータのコピー (コンテキスト間コピー) などの余分な処理が含まれる。また、一般に UNIX の Kernel ではこういったユーザプロ

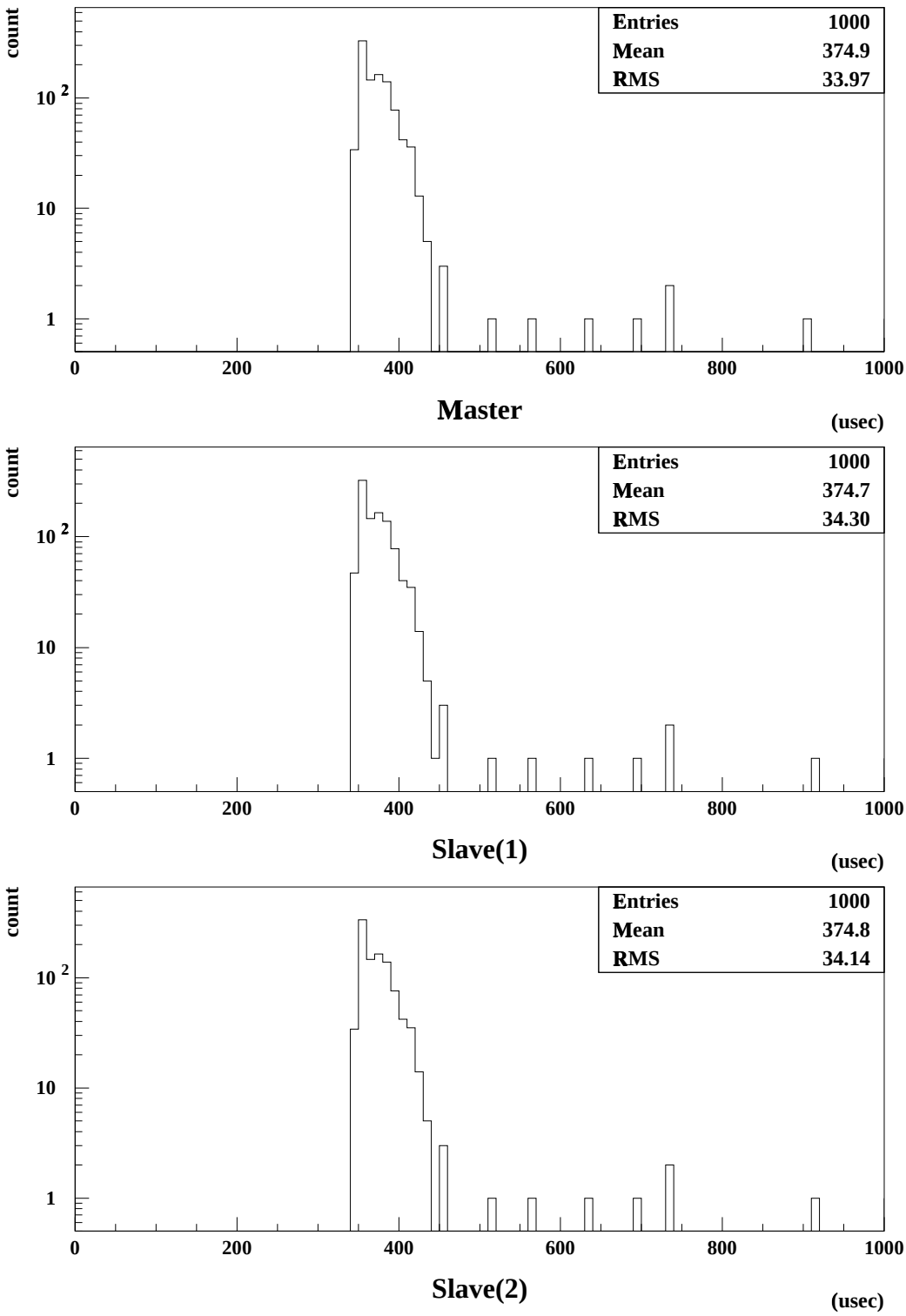


図 18: 読み出しプロセスのソケット通信の処理時間: Ethernet(10BASE-T)

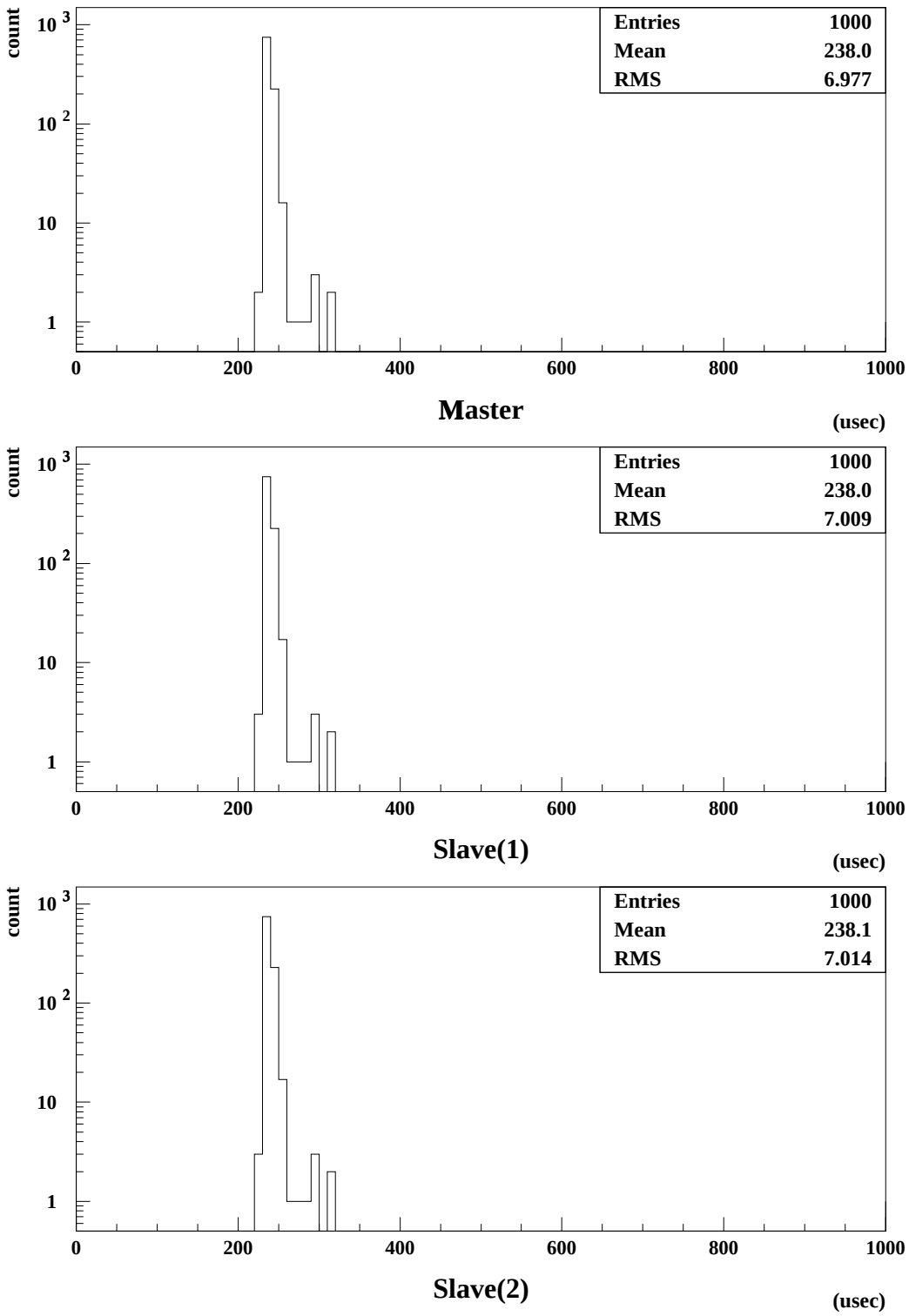


図 19: 読み出しプロセスのソケット通信の処理時間: Fast Ethernet(100BASE-T)

セスからの要求を優先的に扱うことはないため、さらに処理が遅延されることも起こりうる。

つまり、システムコールの呼出に時間をとられるため、今回のように通信によって転送するデータが少ない場合は、ネットワークの転送速度の向上が全体の処理速度の向上に寄与しないことが考えられる。

## 4.4 NFS を利用したデータの転送速度

読み出しプロセスは NFS の機能を利用し、ネットワーク上のファイルへデータを書き込むため、ファイル書き込み、つまりデータ転送の速度を測定する。また、合わせて読み出しの速度も測定する。

PC、ネットワーク等の設定は 4.3 と同じである。プログラムの読み出しは、istream クラスのメンバ関数 read() を、書き込みは、ostream クラスのメンバ関数 write() を使用し、ファイルを開いて固定長の文字列の読み出し、書き込みを行った。つまり、この文字列の長さが、1 度に転送するデータのサイズになる。

時間測定には 4.2、4.3 と同様の方法を取り、read()/write() 関数を 1000 回繰り返した時間を測定した。

結果を図 20 に示す。横軸が 1 度に転送したデータの長さで、その時に要した時間が縦軸に示される。

この結果から、データの転送速度が表 7 のように求めることができる。

表 7: NFS 利用時のデータの転送速度

| | Ethernet (10 BASE-T) | Fast Ethernet (100 BASE-T) |
|---|---|---|
| Read | 8.10 Mbps | 34.0 Mbps |
| Write | 8.06 Mbps | 36.5 Mbps |

Ethernet での結果は、その最大データ転送速度である 10 Mbps に近い値が得られており、この値はネットワークの転送速度を示していると考えることができる。一方 Fast Ethernet での転送速度は、100 Mbps の最大転送速度をかなり下回る結果となっている。これは、ソケットの問題と同様にネットワーク以外の処理速度が影響しているものと考えられる。

NFS においてその要素として考えられるのは、ハードディスクへの読み書きの速度である。このため、ネットワークを介さないローカルな PC で、ハードディスクへのファイルの読み書きの処理時間を測定した。条件は上と同じで、ローカルなハードディスク上のファイルへの読み書きを行った。また、今回の測定で用いたハードディスクの規格は全て同じである (拡張 IDE)。結果としては、Read = 67.3 Mbps、Write = 75.5 Mbps であり、Fast Ethernet の転送速度を下回っている。

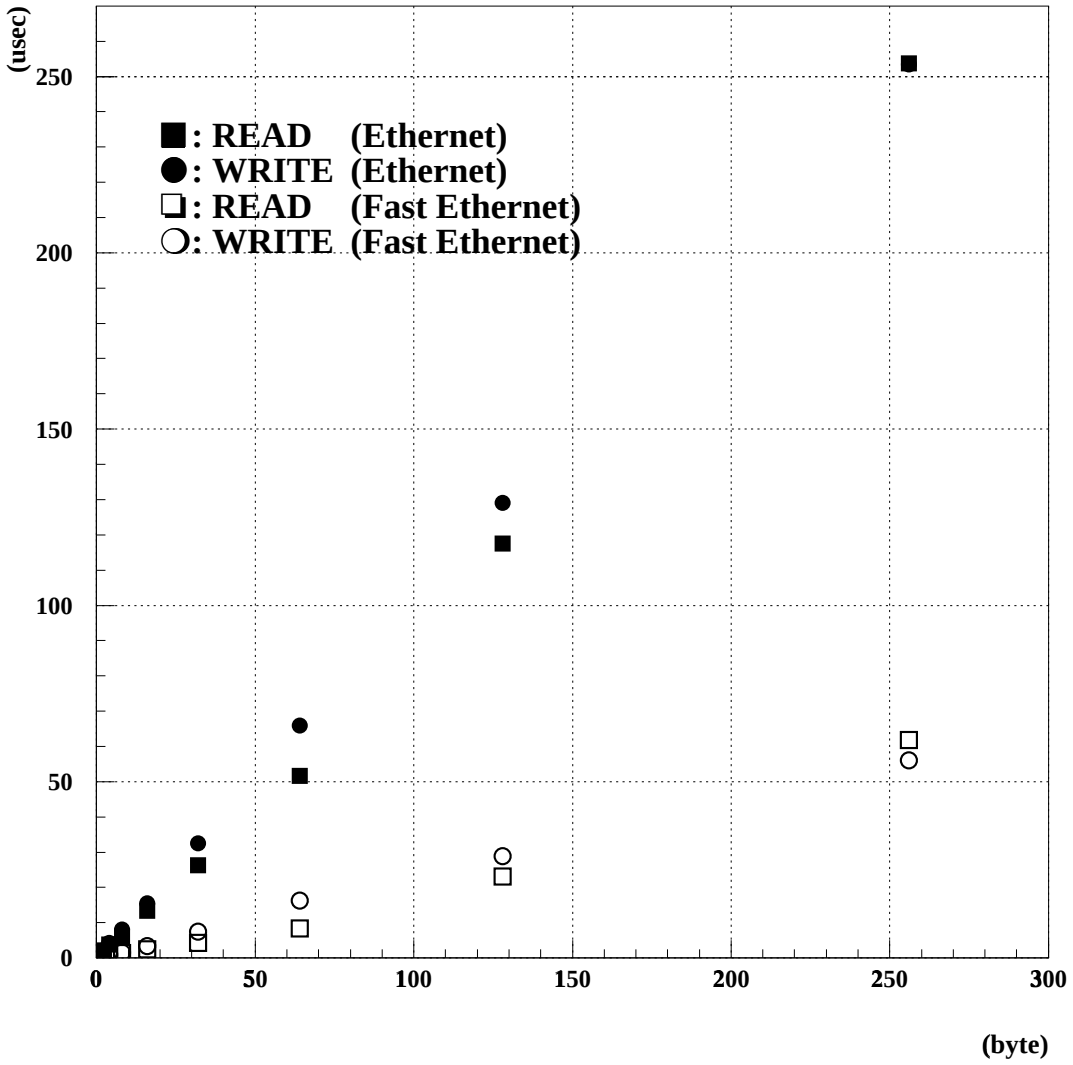


図 20: NFS を利用したデータの転送時間。横軸は 1 回当たり転送するデータサイズ。

このことから、Fast Ethernet 使用時の転送速度は、ハードディスクへのアクセスの処理時間が影響を与えているものと考えられ、期待する転送速度が達成できないが、今回使用したハードディスクの規格は現在普及しているものの中では、比較的処理速度が遅い。このため、さらに速度の速い規格のものを採用すれば、Fast Ethernet の最大転送速度である 100 Mbps に近い値が期待できる。

この結果を用いて、実際に 1 イベント当たりのデータの転送時間を見積もる。2.2 で求めた TMC と SWINE のデータ量から、1 イベントのデータ量をおおよそ 130 word と考えると、Ethernet で約 235 $\mu$sec、Fast Ethernet で約 65 $\mu$sec と見積もることができる。また、Fast Ethernet の限界に近い 80 Mbps の転送速度が達成できた場合は、約 26 $\mu$sec という短い時間で転送が可能となると考えられる。

## 4.5 イベント・フィルタ

どういった条件でイベントを選別するかというのは、実際に検査設備の Drift Tube のデータを見て検討すべきことであるが、ここでは、

- Drift Tube の各層でのヒット数の合計を調べ、一定以上のヒットがあった場合はトラッキング不可能として、落す。
- X-Y、上下のそれぞれの Drift Tube について、3 層のうち 2 層以上でヒットがある場合のみデータを受け入れる。

という項目で選別を行う処理時間を調べた。

TMC のヒットチャンネル数 24 を適当に設定して、上の処理を行う計算をしてみたところ、同様の測定方法で約 30 $\mu$sec という結果が得られた。

今まで述べてきた他の処理に要する時間に比べて十分に速い処理であり、PC の性能が十分に活かせているといえる。

## 4.6 解析プロセス

この DAQ システムでの、データ収集プロセスからオンライン解析へのデータの受渡しは、ファイルを通じて行われるため、オンライン解析もまた複雑な手続きを必要としない。

本検査システムでの解析の主な動作は、

- VME のモジュールやチャンネルの番号で構成されたデータを、検出器の種類やチャンネルで構成し直す Event Building。
- 再構成された Drift Tube のデータを用いた Tracking。

である。これにより TGC のチャンネル毎の Time Jitter は、すぐに求めることができ、トラッキングのデータを用いて TGC のヒットのデータから各部分での検出効率を求めることができる。

ここでは、検査設備用に行われた予備実験に、設計に基づいた DAQ を実装しデータの収集を行い、その解析の結果を示す。

### 4.6.1 予備実験用読み出しシステム

検出器とセットアップを図 21、22 に示す。実際の検査設備と違う点を挙げる。

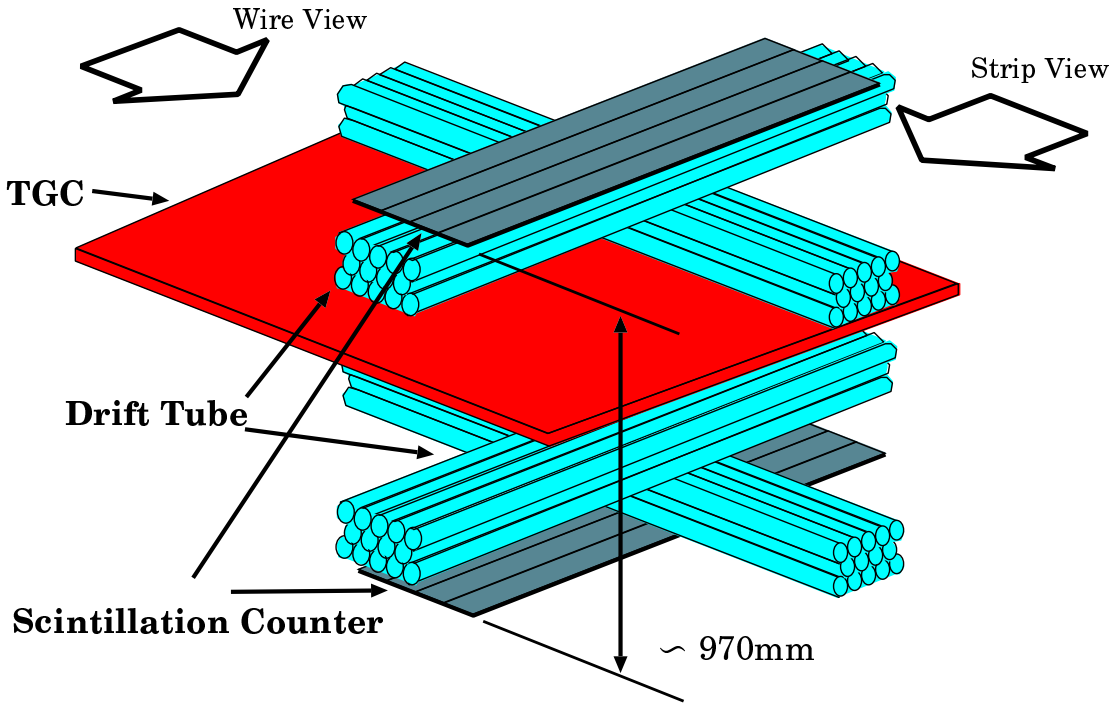


図 21: 予備実験の検出器構成

まず 1 つめは、検出器の数が少ないことである。TGC の大量生産はまだ始まっておらず、試験製作機 (1 台) を用いた。セットアップ時には Drift Tube も全ては使用することができず、1.5 m のもののみを 56 本使用している。また、幅 7 cm のシンチレーション・カウンターを上下 4 枚ずつ並べた。この構成での Drift Tube による TGC 部の有効測定領域は、約 $20 \times 20\,\mathrm{cm}^2$ となる。また、宇宙線の検出頻度は約 10 Hz である。

チャンネル数は、TGC はワイヤーグループ、ストリップ各 16 チャンネルのみを読み出す。Drift Tube は 56 チャンネルで、今回はシンチレーション・カウンターからの信号も TMC により読み出す。次に、現在開発中である WINE システムは用いないことである。このため、TGC の読み出しは TMC で行う。合計で TMC

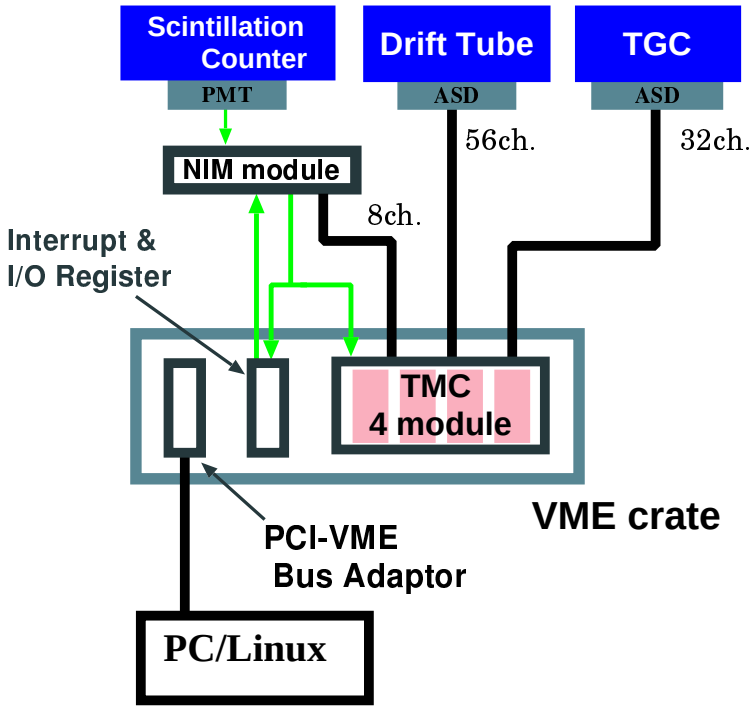


図 22: 予備実験のセットアップ

モジュールは 4 台用いる。そして規模が小さいためネットワークを使用せず、読み出しに使用する PC は 1 台のみである。

使用した PC の主なパラメータを表 8 に示す。

表 8: PC の主なパラメータ

| | |
|---|---|
| CPU | 266 MHz MMX PentiumⅡ |
| Memory | EDO 128 MByte |
| Hard Disk | IDE 4.3 GByte |
| Network Interface Card | 3Com905B-J-TX (10/100BASE-TX) |
| OS | Linux2.0.35 |
| C/C++ compiler | gcc-2.8.1 |

次に、予備実験用に変更した、TMC モジュールから各イベントのデータを読み込むプロセスのシーケンスを図 23 に示す。ネットワークに分散していないため、通信を行わず、単一のプロセスで読み出しを行う。

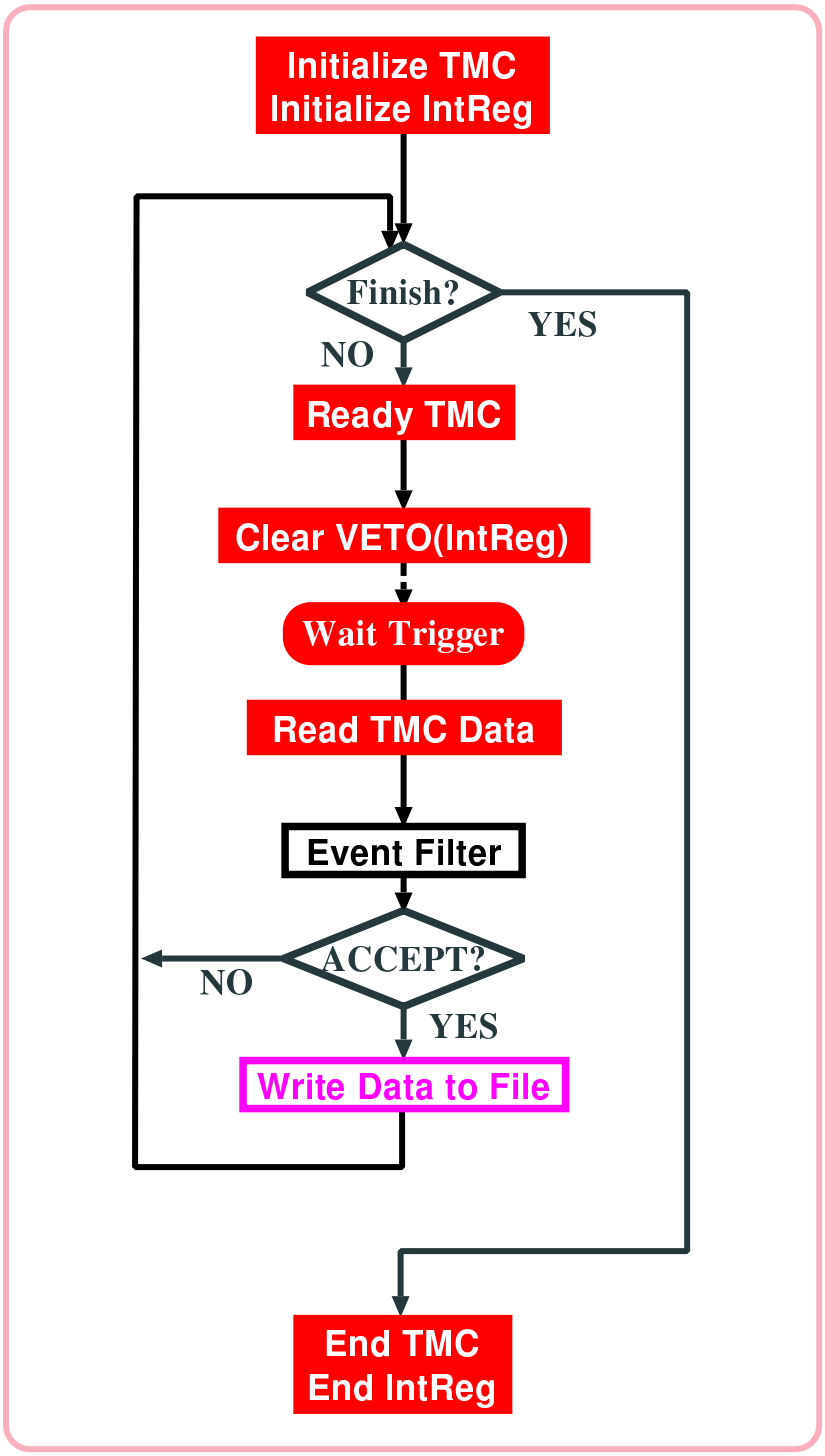


図 23: 予備実験用読み出しシーケンス (DT: Drift Tube, IntReg: Intterupt & I/O Register)

### 4.6.2 解析結果

図 24 は、ヒットの有無とトラッキングの結果を 1 イベントずつ図 21 で示した検出器の側部 2 方向 (Strip View, Wire View) から表示する解析プロセスの画面を示したものである。GUI(Graphical User Interface) を用いて簡単に、個別のイベントを詳細に確認することができる。

図 25 は、オンラインで解析し結果を表示したものである。4 つとも TGC を上から見た (図 8 の) 視点で、Drift Tube により求めた宇宙線通過の TGC 上での位置をプロットしたものである。通過した位置の TGC のワイヤーグループからの読み出しがヒットであった場合、つまり信号が検出された場合は左上 (Wire hit event) にプロットを行い、ヒットでなかった場合、つまり信号が検出できなかった場合は右上 (Wire non-hit event) にプロットを行っている。ストリップからの読み出しに関しても同様に、左下 (Strip hit event) がヒットであった場合、右下 (Strip non-hit channel) がヒットでなかった場合のプロットを行っている。

つまりこのプロットで大まかな Efficiency を見ることができる。図 25 のプロットでは図 8 で示した不感部分が確認できる。

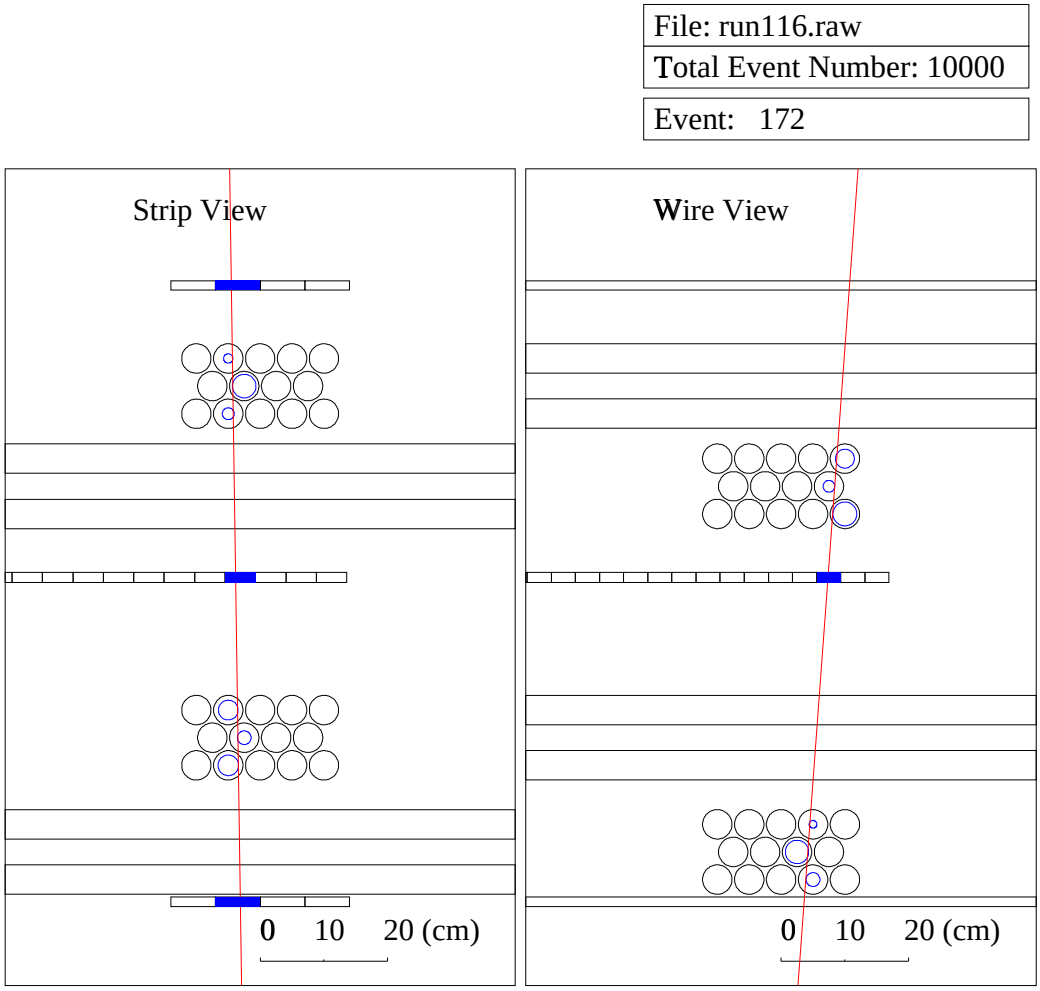


図 24: イベント・ディスプレイの画面。Strip View、Wire View の視点は図 21 で示したもの。シンチレーション・カウンター、TGC に関してはヒットしたチャンネルを示している。Drift Tube は、時間情報から求めたワイヤーからの距離と、それらから導出したトラッキングを描いている。

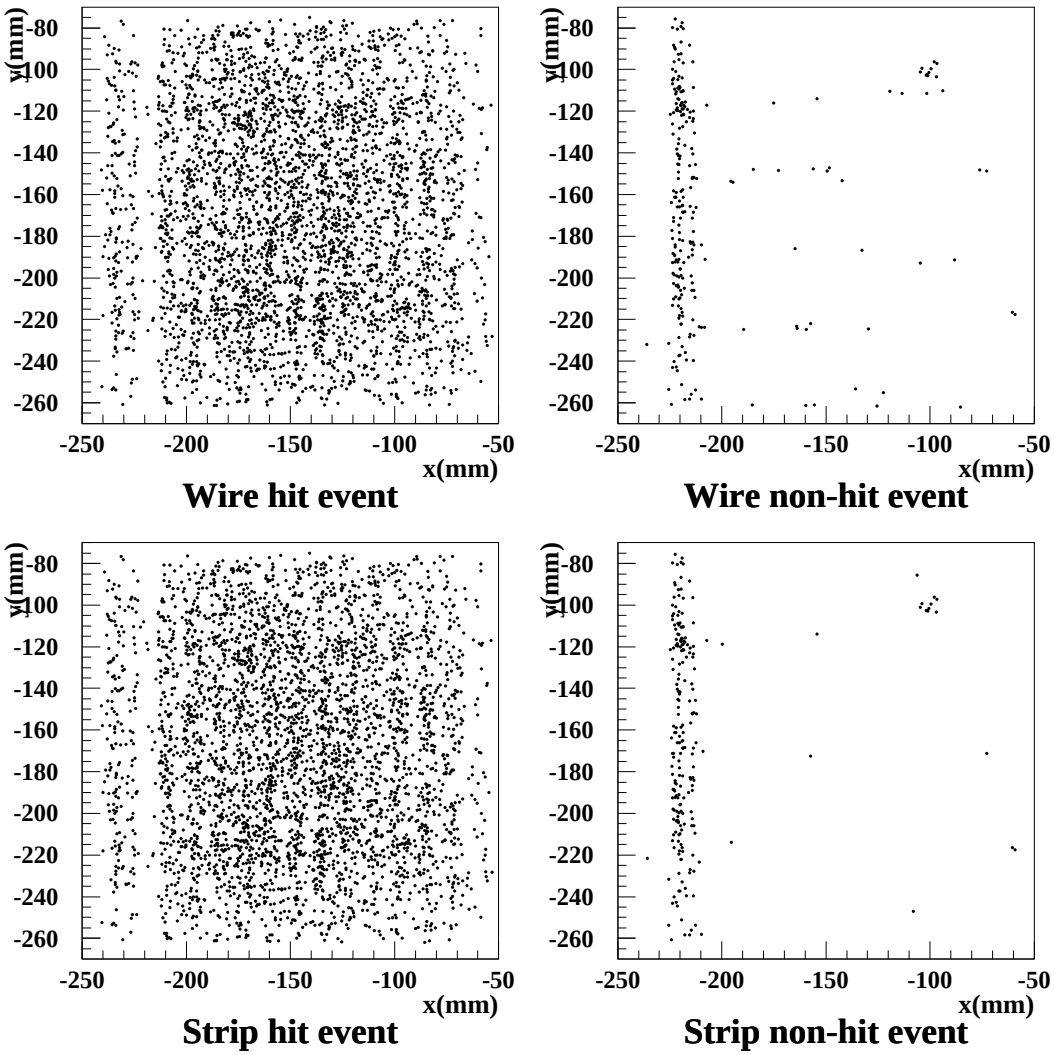


図 25: Drift Tube hit map

# 5 まとめ

## 5.1 考察

読みだしプロセスの処理速度についての考察を行う。

4 章で図 14 のシーケンスで示したそれぞれの処理に要する時間について測定を行い、おおよその値を見積もった。これらの処理時間から、全体の処理速度を表 9 の様に求めた。

表 9: 1 イベントの読み出しに要する処理時間 (単位: $\mu$sec)

| | Ethernet (10 BASE-T) | Fast Ethernet (100 BASE-T) |
|---|---|---|
| VME へのアクセス | 189 | |
| ソケット通信 | 375 | 238 |
| NFS を利用したデータ書き込み | 235 | 65 |
| イベント・フィルタ | $\sim 30$ | |
| 合計 | $\sim 830$ | $\sim 520$ |

VME へのアクセスの時間は TMC を読み出すプロセス (マスタープロセス) での値を示した。

NFS を利用したデータ書き込みでは、全てのモジュールからのデータを書き込む場合の時間となっている。ネットワーク上では、並行してデータを転送することはできないので、全て足し合わせた処理時間を要する。

以上のことから、1 イベント読み出すのにデータ収集システムが必要な処理時間は、Fast Ethernet を採用した場合は、約 520 $\mu$sec であると見積もることができる。この時間は、イベントが発生して VME モジュールがデータ変換を終了してから、トリガーの VETO が解除されるまでの平均時間を表している。

一方、VME モジュールのデータ変換時間は、SWINE よりも TMC が長く、約 650 $\mu$sec であり、結果として、検査システム全体でのデッドタイムはおよそ 1.2 ms といえる。

これは、検査設備での宇宙線検出頻度の約 100 Hz に十分対応できる速度であると判断することができる。

改善が期待される箇所としては、200 $\mu sec$ を越える処理時間を要している、ソケット通信があげられる。これは、プロセスに優先的な処理を設定できないという、UNIX OS の問題であると考えられる。また、NFS によるデータの転送能力については、ハードディスクへの書き込みが問題であるが、これは解析プロセスやデータ保存にも関わることであり、I/O 処理の速いハードディスクを選択することによって、さらに速度の向上を計ることが望ましいと言える。

## 5.2 結論

ATLAS 用 TGC 検査設備に必要なデータ収集 (DAQ) システムの設計及び実装を行った。

データ収集システムは、コンピュータ環境として PC を採用し、Fast Ethernet を用いネットワークに分散するシステムとした。読み出しプロセスは VME モジュールにアクセスしてデータを読み出し、ネットワーク上に分散した PC 上のプロセスは互いに TCP/IP 通信を行うことにより、DAQ を進めていく。

開発したシステムは要求される性能と処理速度を達成できることが確認できた。その特徴としては以下のものが挙げられる。

### 5.2.1 PC 資産の活用

近年の高性能な PC を活かし、ソフトウエアで処理可能な部分は、でき得る限りソフトウエアに担当させるよう設計を行った。その主な要素を以下に示す。

**複数の PC からのデータ読み出しプロセスの同期** 3 台の PC 上で行うデータ読み出しのプロセスは同期を取る必要があるが、各プロセス間で通信を行うことによりこれを実現した。この結果として専用のハードウエアの導入が不要となり、システムの変更に対しても柔軟に対応できる、拡張性の高いシステムが構築できた。

**イベント・フィルタ** Drift Tube のヒットパターンを判別し、フィルタをかけるような処理を、データ読み出しプロセスが行うようにした。このような効率的な読み出しを行うことにより無駄なデータが減らされ、ネットワークを通じたデータ転送や、それ以降のデータ処理にかかる負担の軽減につながる。

### 5.2.2 ネットワークに分散したシステム

本 DAQ システムはネットワーク上にデータ収集を行うプロセスを分散して配置し、さらにオンライン解析のプロセスも分けたことにより、高い処理性能を達成することができることが確認できた。加えて簡単にシステムを拡張することも可能となる。また、TCP/IP 通信を行うことにより、種類の異なる計算機の間でも自由に通信できる、ハードウエアに依存しない DAQ システムを構築した。

**ネットワーク通信を用いてのプロセスの同期** データ読み出しプロセスが取る同期は、ネットワークを介するため、TCP/IP プロトコルを利用した通信により行う。これは UNIX システムコールであるソケットで実装した。

**NFS を利用したデータ転送** UNIX 系 OS に標準装備されている NFS を利用することで、安定で信頼性の高いデータ転送を簡潔に行うことができ、データ収

集プロセスからオンライン解析プロセスへのデータの受渡しは、ファイルを介したものであり、互いに同期を取る必要はない。

ネットワークに Fast Ethernet を採用することにより、以上の処理に於いて十分な転送速度を達成できることが、確認できた。

# A DAQソースコード

ここでは、図 14 のシーケンスに基づいて C++言語を用いて記述した読み出し処理を行うソースコードの一部を掲載する。TMC のデータを読み出すプログラムが TMCReader、SWINE のデータを読み出すプログラムが SWINEReader である。

## A.1 TMCReader

TMCReader は、主なものとして main 関数の 29-31 行で宣言している以下のクラスを用いている。

| | |
|---|---|
| SockMaster | スレーブである SWINEReader とソケット通信を行う。 |
| TMCinterface | TMC にアクセスしデータを読み出す。また、イベント・フィルタも行う。 |
| IntReg | Interrupt & I/O Register にアクセスし、VETO を解除する。 |

また、main 関数 42-59 行が 1 イベントを読み出すループとなっている。

```
 1 /*
 2 * tmc.c
 3 *      Creation Date : Dec    1998
 4 *      Author        : K. Hayashi
 5 */
 6
 7 #define MAIN
 8
 9 #include <fstream.h>
10 #include <strstream.h>
11 #include <stdlib.h>
12 #include <signal.h>
13 #include "libdaq.h"
14 #include "vme/tmc_if.h"
15 #include "vme/intreg.h"
16 #include "socket/socket.h"
17 #include "param.h"
18
19 const int       slave_num       = 2;
20
21 /*=======================================================
22         main
23  =======================================================*/
24 int main( int argc, char **argv )
25 {
26   option        opt( argc, argv );
27   if( !opt )    return -1;
28
```

```
29   SockMaster    sock( slave_num );
30   TMCinterface  tmc( opt.db, opt.level );
31   IntReg        intreg;
32
33   ofstream      fout( opt.file );
34   if( !fout )   cerr << opt.file << err_exit;
35
36   sig_term = 0;
37   signal( SIGTERM, terminate );
38
39   int filter;
40   EventCounter counter( "TMCReader" );
41   fout << run_flg;
42   while( counter < opt.total && sig_term != 1 ){
43
44         tmc.ready();
45         sock.recv_ready();
46
47         intreg.clear_veto();
48
49         tmc.read();
50
51         if( tmc.filter() == 0 ) filter = Accept;
52         else                    filter = Reject;
53
54         sock.send( filter );
55         if( filter == Accept ){
56                 fout << tmc;
57                 counter++;
58         }
59   }
60   fout << end_flg;
61
62   fout.close();
63   return 0;
64 }
65 /*---------------End of File-----------------------*/
```

```
 1 /*
 2  * socket.h
 3  *   Creation date:  Jan  8 1999
 4  *   Author:         K. Hayashi
 5  */
 6
 7 #ifndef __SOCKET_H
 8 #define __SOCKET_H
 9
10   #include <unistd.h>
11
12 #define         ReaderPort      "daq_reader"
13 #define         Ready_message   "Ready!"
14 #define         Accept_message  "Accept"
15 #define         Reject_message  "Reject"
16 const int       Ready           = 1;
```

```
17 const int       Accept          = 2;
18 const int       Reject          = 3;
19 const int       sock_error      = -1;
20 const int       sock_close      = 0;
21 const int       buff_size       = 32;
22
23 /*---------------------------------------
24      ReaderSocket
25   ---------------------------------------*/
26 class ReaderSocket {
27   protected:
28      int     ls;
29      int     s_port;
30      ReaderSocket( void );
31      ~ReaderSocket(){ close( ls ); }
32      int     recv( int );
33      int     send( int, const int );
34 };
35
36 /*---------------------------------------
37      SockMaster      : ReaderSocket
38   ---------------------------------------*/
39 class SockMaster : public ReaderSocket {
40      int     slave_num;
41      int*    sk;
42   public:
43      SockMaster( const int );
44      ~SockMaster();
45      void    recv_ready( void );
46      void    send( const int );
47 };
48
49 /*----------------------------------------
50      SockSlave       : ReaderSocket
51   ----------------------------------------*/
52 class SockSlave : public ReaderSocket {
53   public:
54      SockSlave( char *master );
55      void    send_ready( void ){ if( send( ls, Ready ) != 0 ) exit(-1); }
56      int     recv_trig( void );
57 };
58
59 #endif
60 /*-----------End of File-------------------*/
```

```
 1 /*
 2  * tmc_if.h
 3  *      Creation Date   : 1998
 4  *      Author          : K. Hayashi
 5  */
 6
 7 #ifndef __TMC_IF_H
 8 #define __TMC_IF_H
 9
```

```
10   #include "vmelib.hh"
11   #include "vme_param.h"
12   #include "tmc_data.h"
13
14 const int       TMC_DATA_SIZE   = 4022;
15 const int       TMC_SIZE        = 16208;
16
17 typedef struct {
18   unsigned short        header;
19   unsigned short        content;
20 } EVENT;
21
22 typedef volatile struct {
23   unsigned short        dready;
24   unsigned short        reserved[15];
25   unsigned short        ntotal;
26   unsigned short        status;
27   unsigned short        t0;
28   unsigned short        lastwp;
29   EVENT                 data[ TMC_DATA_SIZE ];
30   unsigned short        Drun;
31   unsigned short        Dcstsp;
32   unsigned short        Dcount;
33   unsigned short        Ddisp;
34   unsigned short        Dsuboff;
35   unsigned short        Dedge;
36   unsigned short        Dmodule;
37   unsigned short        reserved2;
38   unsigned short        offset[ TMC_CH ];
39 } TMC_FORM;
40
41 //////////TMCinterface  : TMCData////////////////////
42 class TMCinterface : public TMCData {
43         TMC_FORM**      mm;
44         int             level;
45   public:
46         TMCinterface( const char*, const int );
47         ~TMCinterface();
48         void            ready( void );
49         void            read( void );
50         int             filter( void );
51 };
52
53 #endif
54 /*--------End of File--------------------------*/
```

```
 1 /*
 2  * intreg.h
 3  *      Creation Date:  Dec    1998
 4  *      Author:         K. Hayashi
 5  */
 6
 7 #ifndef __INTREG_H
 8 #define __INTREG_H
```

```
 9
10   #include "vmelib.hh"
11   #include "vme_param.h"
12
13 const int       INTREG_SIZE     = 0x10;
14
15 typedef volatile struct{
16   unsigned short latch[2];
17   unsigned short flipflop;
18   unsigned short in;
19   unsigned short pulse;
20   unsigned short level;
21   unsigned short csr[2];
22 } INTREG;
23
24 class IntReg{
25         INTREG  *mm;
26   public:
27         IntReg( void );
28         ~IntReg();
29         void    clear_veto( void );
30 };
31
32 /*-------------------------------------------------------
33   CSR SET
34
35         data-bit        write           read
36         D07             --              busy 3
37         D06             enable 3        enable 3
38         D05             --              busy 1/2
39         D04             enable 1/2      envable 1/2
40         D03             mask 1/2        mask 1/2
41         D02             --              --
42         D01             clr 1/2         --
43         D00             clr 3           --
44 -------------------------------------------------------*/
45
46 #endif
47 /*------------------End of File-----------------------------------------*/
```

## A.2 SWINEReader

SWINEReaderが用いるのは、主に 26,27 行で宣言している以下のクラスである。

SockSlave　　マスターである TMCReader とソケット通信を行う。

SWINEinterface　　SWINE にアクセスしデータを読み出す。

1 イベントを読み出すループは、37-47 行で記述されている。

```
 1 /*
 2  * swine.c
 3  *      Creation Date : Dec    1998
 4  *      Author        : K. Hayashi
 5  */
 6
 7 #define MAIN
 8
 9 #include <fstream.h>
10 #include <strstream.h>
11 #include <stdlib.h>
12 #include <signal.h>
13 #include "libdaq.h"
14 #include "vme/swine_if.h"
15 #include "socket/socket.h"
16 #include "param.h"
17
18 /*=========================================================
19         main
20  =========================================================*/
21 int main( int argc, char **argv )
22 {
23   option        opt( argc, argv );
24   if( !opt )    return -1;
25
26   SWINEinterface swine( opt.id, opt.db );
27   SockSlave     sock( opt.master );
28
29   ofstream      fout( opt.file );
30   if( !fout )   cerr << opt.file << err_exit;
31
32   sig_term = 0;
33   signal( SIGTERM, terminate );
34
35   EventCounter counter( "SWINEReader", opt.id );
36   fout << run_flg;
37   while( counter < opt.total && sig_term != 1 ){
38
39         swine.ready();
40         sock.send_ready();
41
42         swine.read();
43         if( sock.recv_trig() == Accept ){
44                 fout << swine;
45                 counter++;
46         }
47   }
48   fout << end_flg;
49
50   fout.close();
51   return 0;
52 }
53 /*----------End of File--------------------------------*/
```

# 図 目 次



# 表 目 次

# 参考文献


[1] ATLAS Technical Proposal,CERN/LHCC/94-43(1994)

[2] ATLAS Muon Spectrometer Tecnical Design Report,CERN/LHCC/97-22 (1997)

[3] ATLAS homepage, *http://atlasinfo.cern.ch/Atlas/Welcome.html*

[4] 尼子勝哉「大型陽子・陽子衝突型加速器 (LHC) 計画とその物理」: 日本物理学会誌 Vol.52,No.7(1997)508-516

[5] 高エネルギー加速器研究機構 素粒子原子核研究所 山内一夫「標準 Higgs 粒子の探索」ATLAS-Japan workshop NARUTO(1997)

[6] A.Ferrari and P.R.Sala,*Background rates in the muon system:recent results and the effect of the tungsten plug*,ATLAS Internal Note Muon-No-90(1995)

[7] 東京大学　吉田光宏、修士学位論文 「ミューオントリガーチェンバーの信号特性の研究とフロントエンドモノリシック IC の開発」

[8] 信州大学　細田隆志、修士学位論文「アトラス・ミュー粒子検出器用トリガーチェンバーの研究」

[9] H.Fukui et al.,*Results of An Ageing Test of Thin Gap Chamber for ATLAS Endcap Muon Trigger*,(1997) submitted to ATLAS Muon Note

[10] H.Fukui et al.,*Studies on ageing effect and rate dependence of Thin Gap Chamber*,(1998)

[11] K. クラインクネヒト 著 培風館、「粒子線検出器–放射線計測の基礎と応用–」

[12] Glenn F.Knoll 著 日刊工業新聞社、「放射線計測ハンドブック」

[13] F.Sauli,*PRINCIPLES OF OPERATION OF MULTIWIRE PROPORTIONAL AND DRIFT CHAMBERS*,CERN Report 77-09(1997)

[14] 東京農工大学 白須英貴、修士学位論文「TMC LSI を用いた高エネルギー物理学実験用電子回路技術の開発」

[15] 東京大学 松浦聡、「マルチワイヤーチェンバーの読み出しシステムの開発」日本物理学会　秋の分科会 (1998)

[16] Model 616 Adaptor Hardware Manual, Bit 3 Computer Corporation(1996)

[17] Natalia Kruszynska, *VME Host Bridge driver for Linux on PC (http://www.nikhef.nl/user/natalia/projects/vmehb.html)*

[18] 高エネルギー加速器研究機構 オンライングループ 仲吉一男、「PC/Linux 上での VMEbus ベースデータ収集システムの構築」日本物理学会　秋の分科会 (1997)

[19] W. リチャード・スティーブンス著 トッパン、「UNIX ネットワークプログラミング」

# 謝辞

本研究を進めるにあたって、適切な御指導並びに助言を与えて下さいました武田廣教授、野崎光昭教授、川越清以助教授、藏重久弥助教授、神戸大学工学部本間康浩助教授に深く感謝します。

高エネルギー加速器研究機構での研究活動にあたり、様々な助言、御指導を下さいました近藤敬比古教授、大須賀鬨雄助教授、岩崎博行助教授、佐々木修助手、田中秀治助手、山内一夫氏に感謝します。また、研究活動全般にわたり適切な御指導をしていただいた素粒子物理学国際研究センター小林富雄教授、信州大学竹下徹助教授に感謝します。VME 全般に関して御指導して下さいました京都大学坂本宏助教授に深く感謝します。

研究活動のみならず研究生活全体においても惜しみない御支援、御協力をしていただいた福井秀人氏に心から感謝します。ビームテスト時にいろいろとお世話していただいた本間謙輔氏、陣内修氏に感謝します。そして、研究生活を共にし様々な支援をしてくれました津野総司氏、松浦聡氏、佐藤構二氏、加賀志穂佳氏、中村友昭氏、水谷奈津子氏、鈴木修氏、塚原知宏氏、そして東京大学、信州大学、京都大学、KEK、神戸大学の方々に心から感謝します。